Technical Reference Manual

J721S2/TDA4VE/TDA4AL/TDA4VL/AM68 Processors Silicon Revision 1.0 Texas Instruments Families of Products

Table of Contents

Read This First....2
About This Manual....2
Related Documentation From Texas Instruments....2
Glossary....2
Support Resources....2
Export Control Notice....2
Release History....3
1 Introduction....3
1.1 Device Overview....3
1.2 Module Descriptions....10
1.3 Device Identification....26
2 Memory Maps....26
2.1 MAIN Memory Map....27
2.2 MCU Memory Map....81
2.3 WKUP Memory Map....86
2.4 Processors View Memory Map....87
2.5 Region-based Address Translation....89
3 System Interconnect....89
3.1 System Interconnect Overview....89
3.2 System Interconnect Functional Description....90
4 Initialization....165
4.1 Initialization Overview....166
4.2 Boot Process....169
4.3 Boot Mode Pins....174
4.4 Boot Parameter Tables....201
4.5 Boot Image Format....210
4.6 Boot Modes....217
4.7 Boot Memory Maps....222
5 Device Configuration....223
5.1 Control Module (CTRL_MMR)....223
5.2 Power....231
5.3 Reset....256
5.4 Clocking....275
5.5 Module Integration....355
6 Processors and Accelerators....394
6.1 Compute Cluster....395
6.2 Dual-A72 MPU Subsystem....397
6.3 Dual-R5F MCU Subsystem....408
6.4 C71x DSP Subsystem....479
6.5 Graphics Accelerator (GPU)....488
6.6 Video Accelerator....489
6.7 Vision Pre-processing Accelerator (VPAC)....496
6.8 Depth and Motion Perception Accelerator (DMPAC)....677
7 Interprocessor Communication....678
7.1 Mailbox....679
7.2 Spinlock....691
8 Memory Controllers....695

Copyright © 2024 Texas Instruments Incorporated

8.1 Multicore Shared Memory Controller (MSMC)....696
8.2 DDR Subsystem (DDRSS)....718
8.3 Peripheral Virtualization Unit (PVU)....763
8.4 Region-based Address Translation (RAT) Module....768
9 Interrupts....769
9.1 Interrupt Architecture....769
9.2 Interrupt Controllers....770
9.3 Interrupt Routers....780
9.4 Interrupt Sources....780
10 Data Movement Architecture (DMA)....780
10.1 DMA Architecture....781
10.2 Navigator Subsystem (NAVSS)....843
10.3 Peripheral DMA (PDMA)....903
10.4 Data Routing Unit (DRU)....967
11 Time Sync....980
11.1 Time Sync Module (CPTS)....981
11.2 Timer Manager....987
11.3 Time Sync and Compare Events....992
12 Peripherals....993
12.1 General Connectivity Peripherals....994
12.2 High-speed Serial Interfaces....1134
12.3 Memory Interfaces....1185
12.4 Industrial and Control Interfaces....1398
12.5 Audio Interfaces....1551
12.6 Display Subsystem (DSS) and Peripherals....1605
12.7 Camera Subsystem....1802
12.8 Shared MIPI D-PHY Transmitter (DPHY_TX)....1850
12.9 Timer Modules....1853
12.10 Internal Diagnostics Modules....1883
13 On-Chip Debug....1941
14 Revision History....1941

Read This First

About This Manual

This Technical Reference Manual (TRM) details the integration, the environment, the functional description, and the programming models for each peripheral and subsystem in the device.

Related Documentation From Texas Instruments

For a complete listing of related documentation and development-support tools for the device, visit the Texas Instruments website at www.ti.com.

Glossary

TI Glossary	This glossary lists and explains terms, acronyms, and definitions.

Support Resources

TI E2E™ support forums are an engineer's go-to source for fast, verified answers and design help — straight from the experts. Search existing answers or ask your own question to get the quick design help you need.

Linked content is provided "AS IS" by the respective contributors. They do not constitute TI specifications and do not necessarily reflect TI's views; see TI's Terms of Use.

Export Control Notice

Recipient agrees to not knowingly export or re-export, directly or indirectly, any product or technical data (as defined by the U.S., EU, and other Export Administration Regulations) including software, or any controlled product restricted by other applicable national regulations, received from disclosing party under nondisclosure obligations (if any), or any direct product of such technology, to any destination to which such export or re-export is restricted or prohibited by U.S. or other applicable laws, without obtaining prior authorization from U.S. Department of Commerce and other competent Government authorities to the extent required by those laws.

Copyright © 2024 Texas Instruments Incorporated

Release History

The following table summarizes the J721S2 TDA4VE TDA4AL TDA4VL Silicon Revision 1.0 Technical Reference Manual versions.

Version	Literature Number	Date	Notes
*	SPRUJ08	March 2022	See (1)

Trademarks

TI E2E™ is a trademark of Texas Instruments.
eMMC™ is a trademark of MultiMediaCard Association.
Neon™, CoreSight™, and CoreLink™ are trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere.
DisplayPort™ is a trademark of VESA.
HD Radio™ is a trademark of iBiquity Digital Corporation.
HyperBus™ is a trademark of Mobiveil Inc.
Arm®, Cortex®, and Thumb® are registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere.
PCI-Express® and PCIe® are registered trademarks of PCI-SIG.
Secure Digital® is a registered trademark of SD Card Association.
VESA® is a registered trademark of VESA.
Cypress® is a registered trademark of Cypress Semiconductor Corporation.
All trademarks are the property of their respective owners.

1 Introduction

This chapter introduces the features, subsystems, and architecture of the J721S2 Processor Platform high-performance System-on-Chip (SoC).

Note

This document describes the Superset architecture, processors and peripherals of the J721S2 Family of SoCs, which are part of the K3 Multicore SoC architecture platform. Not all features are available on each family of devices. The superset J721S2 devices are available for preproduction software development. Software should constrain the features used to match the intended production device. For more information on the specific features, processors and peripherals available on a particular device, refer to the Device Comparison table in the corresponding device-specific Data sheet.

The J721S2 Processor Platforms are hereinafter commonly referred to as J721S2 *platform*, *device*, or *SoC*.

TI limits support for this family of SoCs to features that are supported via Software Development Kits (SDK). The SDK “build sheet” is available for download as part of each SDK and should be referenced to understand the subset of SoC hardware functionality that is available in software:

https://www.ti.com/tool/PROCESSOR-SDK-J721S2

https://www.ti.com/tool/PROCESSOR-SDK-AM68A

This document describes the Superset features of the Modules integrated into this Device. See the Module Integration section of this document for a list of module features not supported by the integration on this Device.

1.1 Device Overview

1.1.1 Device Overview Feature List

The DRA829 and TDA4VM SoCs are part of the K3 Multicore SoC architecture platform. The SoCs are targeted for automotive applications and aim to meet the complex processing needs of modern embedded products. They are designed as a low power, high performance and highly integrated device architecture, providing significant

Copyright © 2024 Texas Instruments Incorporated

levels of processing power, graphics capability, video and imaging processing, virtualization, and coherent memory support. In addition, these SoCs support state of the art security and functional safety features.

Key distinguishing device features:

- 64-bit architecture with virtualization and coherent memory support, which leverages full processing capability of 64-bit Arm® Cortex® -A72
- Fully programmable industrial communication subsystems to enable future-proof designs for customers that need to adopt the new Gigabit Time-sensitive Networks (TSN) standards, but still need full support on legacy protocols and continuous system optimization over the product deployment
- Integration of vision hardware processing accelerators to facilitate extensive processing requirements in low power budget for automotive ADAS and machine vision applications
- Integration of a general-purpose microcontroller unit (MCU) with a dual Arm® Cortex®-R5F MCU subsystem, available for general purpose use as two cores or in lockstep, intended to help customers achieve functional safety goals for their end products
- Integration of a next-generation fixed and floating-point C71x Digital Signal Processor (DSP) that significantly boosts power over a broad range of general signal processing tasks for both general applications and automotive functions which also incorporates advanced techniques to improve control code efficiency and ease of programming such as branch prediction, protected pipeline, precise exception and virtual memory management
- Tightly coupled Matrix Multiplication Accelerator (MMA) that extends the C71x DSP architecture's scalar and vector facilities enabling deep learning and enhance vision, analytics and wide range of general applications. The achieved total TOPS (Tera Operations Per Second) performance significantly differentiates the device for single board computer in machine vision and deep learning applications
- Key display features including flexibility to interface with different panel types (eDP, DSI, DPI) with multi-layer hardware composition
- Integration of hardware features that help applications to achieve functional safety mechanisms
- Robust security architecture with sandboxed Security Controller managing all secure configurations with high performance client-server messaging scheme between secure Security Controller and all cores
- Simplified solution for power supply management, enabling lower cost system solution (on-die bias LDOs and power good comparators for minimal power sequencing requirements consistent with low cost supply design)

Processor cores:

- Two C7x floating point, vector DSP, up to 1.0 GHz, 160 GFLOPS, 512 GOPS
- Deep-learning matrix multiply accelerator (MMA), up to 8 TOPS (8b) at 1.0 GHz
- Vision Processing Accelerators (VPAC) with Image Signal Processor (ISP) and multiple vision assist accelerators
- Depth and Motion Processing Accelerators (DMPAC)
- Dual 64-bit Arm® Cortex®-A72 microprocessor subsystem at up to 2 GHz
  - 1MB shared L2 cache per dual-core Cortex®-A72 cluster
  - 32KB L1 DCache and 48KB L1 ICache per Cortex®-A72 core
- Up to six Arm® Cortex®-R5F MCUs at up to 1.0 GHz
  - 16K I-Cache, 16K D-Cache, 64K L2 TCM
  - Two Arm® Cortex®-R5F MCUs in isolated MCU subsystem
  - Four (TDA4VE) or Two (TDA4AL/TDA4VL) Arm® Cortex®-R5F MCUs in general compute partition
- GPU IMG BXS-64-4, 256kB Cache, up to 800 MHz, 50 GFLOPS, 4 GTexels/s (TDA4VE and TDA4VL)
- Custom-designed interconnect fabric supporting near max processing entitlement

Memory subsystem:

- Up to 4MB of on-chip L3 RAM with ECC and coherency
  - ECC error protection
  - Shared coherent cache
  - Supports internal DMA engine
- Up to Two External Memory Interface (EMIF) modules with ECC
  - Supports LPDDR4 memory types
  - Supports speeds up to 4266 MT/s
  - Two (TDA4VE) or One (TDA4AL/TDA4VL) 32-bit data bus with inline ECC up to 17 GB/s per EMIF

Copyright © 2024 Texas Instruments Incorporated

- General-Purpose Memory Controller (GPMC)
- One (TDA4AL/TDA4VL) or Two (TDA4VE) 12KB on-chip SRAM in MAIN domain, protected by ECC

Functional Safety:

- Functional Safety-Compliant targeted (on select part numbers)
- Developed for functional safety applications
- Documentation available to aid ISO 26262 functional safety system design up to ASIL-D/SIL-3 targeted
- Systematic capability up to ASIL-D/SIL-3 targeted
- Hardware integrity up to ASIL-D/SIL-3 targeted for MCU Domain
- Hardware integrity up to ASIL-B/SIL-2 targeted for Main Domain
- Safety-related certification
  - ISO 26262 planned

Device security (on select part numbers):

- Secure boot with secure runtime support
- Customer programmable root key, up to RSA-4K or ECC-512
- Embedded hardware security module
- Crypto hardware accelerators – PKA with ECC, AES, SHA, RNG, DES and 3DES

High speed serial interfaces:

- One PCI-Express® (PCIe) Gen3 controllers
  - Up to four lanes per controller
  - Gen1 (2.5GT/s), Gen2 (5.0GT/s), and Gen3 (8.0GT/s) operation with auto-negotiation
- One USB 3.0 dual-role device (DRD) subsystem
  - Enhanced SuperSpeed Gen1 Port
  - Supports Type-C switching
  - Independently configurable as USB host, USB peripheral, or USB DRD
- Two CSI2.0 4L RX plus Two CSI2.04L TX

Copyright © 2024 Texas Instruments Incorporated

Ethernet

- Two RGMII/RMII interfaces

Automotive interfaces:

- Twenty Modular Controller Area Network (MCAN) modules with full CAN-FD support

Display subsystem:

- One (TDA4AL/TDA4VL) or Two (TDA4VE) DSI 4L TX (up to 2.5K)
- One eDP 4L (TDA4VE/TDA4VL)
- One DPI

Audio interfaces:

- Five Multichannel Audio Serial Port (MCASP) modules

Copyright © 2024 Texas Instruments Incorporated

Video acceleration:

- TDA4VE: H.264/H.265 Encode/Decode (up to 480 MP/s)
- TDA4AL: H.264/H.265 Encode *only* (up to 480 MP/s)
- TDA4VL: H.264/H.265 Encode/Decode (up to 240 MP/s)

Flash memory interfaces:

- Embedded MultiMediaCard Interface (eMMC™ 5.1)
- One Secure Digital® 3.0/Secure Digital Input Output 3.0 interfaces (SD3.0/SDIO3.0)
- Two simultaneous flash interfaces configured as
  - One OSPI or HyperBus™ or QSPI, and
  - One QSPI

Copyright © 2024 Texas Instruments Incorporated

System-on-Chip (SoC) architecture:

- 16-nm FinFET technology
- 23 mm x 23 mm, 0.8-mm pitch, 770-pin FCBGA (ALZ)

Companion Power Management ICs (PMIC):

- Functional Safety-Compliant support up to ASIL-D / SIL-3 targeted
- Flexible mapping to support different use cases

1.1.2 Device Block Diagram

Figure 1-1 shows the SoCs top-level block diagram with domains partitions.

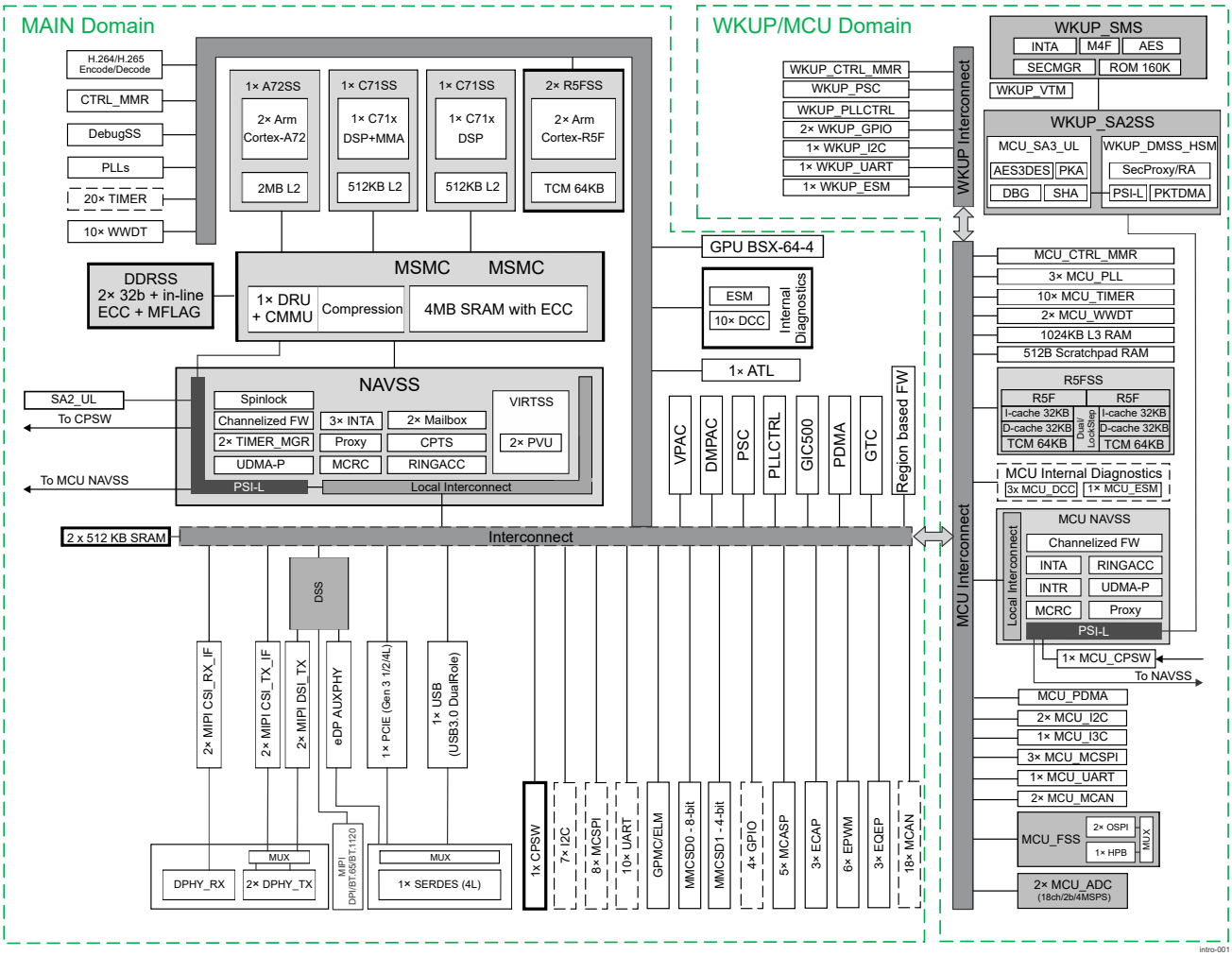


Figure 1-1. Device Top-level Block Diagram

1.1.3 Modules Allocation and Instances within Device Domains

Table 1-1 shows device IPs allocation within device domains.

Table 1-1. Modules Allocation and Instances within Device Domains

Module Full Name	Module Abbreviation	Domain		
		WKUP	MCU	MAIN
Dual-core Arm Cortex-A72 MPU	A72SS	-	-	1
Dual-core Arm Cortex-R5F Subsystem	R5FSS	-	1	2

Copyright © 2024 Texas Instruments Incorporated

Table 1-1. Modules Allocation and Instances within Device Domains (continued)

Module Full Name	Module Abbreviation	Domain		
		WKUP	MCU	MAIN
C71x Digital Signal Processor Subsystem with Matrix Multiplication Accelerator	C71SS (w/ MMA)	-	-	1
C71x Digital Signal Processor Subsystem without Matrix Multiplication Accelerator	C71SS	-	-	1
Arm Cortex-M4F based Security Management Subsystem	SMS	1	-	-
Graphics Processing Unit (BXS-64-4)	GPU	-	-	1
Video Accelerator	CODEC	-	-	1
Vision Pre-processing Accelerator	VPAC	-	-	1
Depth and Motion Perception Accelerator	DMPAC	-	-	1
Mailbox	MAILBOX	-	-	2
Spinlock	SPINLOCK	-	-	1
Multicore Shared Memory Controller	MSMC	-	-	1
DDR Subsystem	DDRSS	-	-	2
Peripheral Virtualization Unit	PVU	-	-	2
Navigator Subsystem	NAVSS	-	1	1
Unified DMA Controller	UDMA	-	1	1
Ring Accelerator	RINGACC	-	1	1
Proxy	PROXY	-	1	1
Secure Proxy	SEC_PROXY	-	1	1
Interrupt Aggregator	INTR_AGGR	-	1	3
Data Routing Unit	DRU	-	-	1
Common Platform Time Sync Module	CPTS	-	1	6
Timer Manager	TIMER_MGR	-	-	2
Analog-to-Digital Converter	ADC	-	2	-
General-Purpose Input/Output	GPIO	2	-	4
Inter-Integrated Circuit	I2C	1	2	7
Improved Inter-Integrated Circuit	I3C	-	1	-
Multichannel Serial Peripheral Interface	MCSPI	-	3	8
Universal Asynchronous Receiver/Transmitter	UART	1	1	10
Gigabit Ethernet Switch	CPSW2G	-	1	1
Peripheral Component Interconnect Express	PCIe	-	-	1
Universal Serial Bus Subsystem	USBSS	-	-	1
Serializer/Deserializer	SERDES	-	-	1
Flash Memory Subsystem	FSS	-	1	-
Octal Serial Peripheral Interface	OSPI	-	2	-
HyperBus Interface	HPB	-	1	-
General-Purpose Memory Controller	GPMC	-	-	1
Error Location Module	ELM	-	-	1
Multimedia Card/Secure Digital Interface	MMCSD	-	-	2
Enhanced Capture Module	ECAP	-	-	3
Enhanced Pulse Width Modulation Module	EPWM	-	-	6
Enhanced Quadrature Encoder Pulse Module	EQEP	-	-	3
Controller Area Network Interface	MCAN	-	2	18
Audio Tracking Logic	ATL	-	-	1
Multichannel Audio Serial Port	MCASP	-	-	5
Display Subsystem	DSS	-	-	1
MIPI Display Serial Interface	DSI	-	-	2

Copyright © 2024 Texas Instruments Incorporated

Table 1-1. Modules Allocation and Instances within Device Domains (continued)

Module Full Name	Module Abbreviation	Domain		
		WKUP	MCU	MAIN
Embedded DisplayPort Transmitter	eDP	-	-	1
Camera Streaming Receiver Interface	CSI_RX_IF	-	-	2
Camera Streaming Transmitter Interface	CSI_TX_IF	-	-	2
Shared D-PHY Transmitter	DPHY_TX	-	-	2
Global Timer Counter	GTC	-	-	1
Real Time Interrupt Windowed Watchdog Module	RTI	-	2	9
Timers	TIMER	-	10	20
Dual Clock Comparator	DCC	-	3	10
Error Signaling Module	ESM	1	1	1
Memory Cyclic Redundancy Check	MCRC	-	1	1

Note

The supported set of features and peripherals is device part number dependent. For more information, see the device-specific Datasheet.

1.2 Module Descriptions

1.2.1 Arm Cortex-A72 Subsystem

Each core of the dual-core Arm Cortex-A72 Microprocessor Units (MPU) has the following main features:

- Full Armv8-A architecture compliancy
- Advanced Single Instruction Multiple Data (SIMD) and floating point extension (Arm Neon™)
- Armv8 cryptography extensions
- Superscalar, variable length, out-of-order pipeline
- 48KB program and 32KB data Level 1 (L1) Cache
- 1MB shared Level 2 (L2) Cache
- ECC protection for L1 data cache and L2 Cache
- Parity protection for L1 Instruction Cache
- Dynamic branch prediction with Branch Target Buffer (BTB) and Global History Buffer (GHB) RAMs, return stack, and indirect predictor
- Arm General Interrupt Controller (GICv3) architecture
- Support timers for each Cortex-A72 core
- 512-bit wide, synchronous or asynchronous VBUSM.C master interface
- Arm CoreSight™ Debug and Trace Architecture
- Advanced power management for low power optimization
- SoC level dedicated RTI windowed watchdog timer per core

1.2.2 Arm Cortex-R5F Processor

Each instance of the dual-core Arm Cortex-R5F processor supports the following main features:

- Armv7-R architecture
- Two modes of operation, boot-time configurable:
  - Split mode: two independently operating cores (asymmetric multi processing, no coherence)
  - Lock (lockstep) mode: one main operating core with the other operating in lockstep
  - Boot-time configurable to be in split or lock mode
- 32KB instruction and 32KB data SECDED ECC protected L1 cache per core
- 64KB of Tightly Coupled Memory (TCM) per core in a split mode
- 128KB of TCM for CPU0 in lock mode
- Full-Precision Floating Point (VFPv3)
- 8 breakpoints, 8 watch points

Copyright © 2024 Texas Instruments Incorporated

- 16-region Memory Protection Unit (MPU)
- CoreSight Debug Access Port (DAP)
- CoreSight ETM-R5 interface
- Performance Monitoring Unit (PMU)
- 32-bit to 48-bit Region-based Address Translation (RAT) on memory access masters
- Integrated Vectored Interrupt Manager (VIM)
- Interfaces:
  - 64-bit VBUSM master pair (1 read, 1 write) for L3 memory accesses (per core)
  - 64-bit VBUSM slave for TCM access (per core)
  - 32-bit VBUSM master pair (1 read, 1 write) for peripheral access
  - 32-bit VBUSP master for peripheral access (per core)
  - 32-bit VBUSP slave configuration port (per core)
  - 32-bit VBUSP slave debug port

1.2.3 C71x DSP Subsystem

Each instance of the C71x DSP supports the following main features:

- True 64-bit C7120 CPU core with:
  - Instruction fetch unit
  - Instruction dispatch unit
  - Instruction decode unit
  - CPU dual data path with one 64-bit scalar side (side A) and one 512-bit vector side (side B)
  - CPU control logic
  - Test, debug, and interrupt logic
  - Enhanced Instruction Set Architecture (ISA)
  - OpenCL features
- Matrix Multiply Accelerator (MMA) as a special functional unit in C71x CorePac CPU (C71SS0 only)
- L1 Program Memory Controller (PMC) with 32KB L1P memory, all cache
- L1 Data Memory Controller (DMC) with 48KB L1D memory, configurable as cache and/or SRAM
- L2 Unified Memory Controller (UMC) with 512KB L2 memory, configurable as cache and/or SRAM
- Multi-dimensional Streaming Engine (SE) - flexible, high bandwidth mechanism for reading large quantities of data into C71x DSP
- CorePac Memory Management Unit (CMMU)
- Power-down controller
- Debug capabilities

1.2.4 Graphics Processing Unit

The Graphics Processing Unit (GPU) which accelerates 2-dimensional (2D) and 3-dimensional (3D) graphics and compute applications. It supports the following main features:

- Architecture based on IMG BXS-64-4 with 256KB cache
- OpenGL 3.x and Vulkan API support
- Up to 50 GFLOPS
- Support for GPU virtualization
- Support for HyperLane Technology, with 8 HyperLanes available

1.2.5 Video Accelerator

The Video Accelerator (CODEC) has the following main features, among others:

- Encode/Decode:
- – H.265/HEVC – Main and Main Still Picture Profile @L5.1 High-tier
  - H.264/AVC – Baseline/Constrained Baseline/Main/High Profiles Level @L5.2
  - Max resolution: 8192x8192
  - YUV420 video format
  - YUV422 video format (only encoder)
  - 8-bit depth
- Encode, Decode or combination up to 4k60 total rate

Copyright © 2024 Texas Instruments Incorporated

- Up to 8x separate concurrent video encode, decode or combination (8x1080p30)

1.2.6 Vision Pre-processing Accelerator

The Vision Pre-processing Accelerator (VPAC) provides a set of common vision primitive functions, performing various pixel data processing tasks, such as: color processing and enhancement, noise filtering, wide dynamic range (WDR) processing, lens distortion correction, pixel remap for de-warping, on-the-fly scale generation, on-the-fly pyramid generation, and offloads these common tasks from the main SoC processors (ARM, DSP, etc.). The VPAC includes the following processing and infrastructure sub-modules:

- Vision Imaging Sub-System (VISS) which provides raw data image processing such as:
  - Wide Dynamic Range (WDR) merge
  - Defect Pixel Correction (DPC)
  - Lens Shading Correction (LSC)
  - Global/Local Brightness and Contrast Enhancement (GLBCE)
  - Advanced Spatial Noise Filter (NSF4V)
  - Edge Enhancement (EE)
  - Demosaicing
  - Color conversion
- Lens Distortion Correction (LDC) block, which provides data reading from memory (DDR or on-chip) and applies perspective transformation as well as correction of lens distortion (including fisheye lenses).
- Multi-Scalar (MSC) block reads data from memory (DDR or on-chip) to internal shared level 2 (SL2) memory and generates up to 10 scaled outputs from one or two inputs, with various scaling ratios.
- Noise Filter (NF) block reads data from memory (DDR or on-chip) to internal SL2 memory and does Bilateral filtering to remove noise.
- Hardware Thread Scheduler (HTS) provides inter-processor communication among various VPAC sub-modules (VISS, LDC, MSC and NF) as well as with the local DMA Engine (UTC).
- Internal Shared Level 2 (SL2) memory for data exchange across VPAC sub-modules (VISS, LDC, MSC and NF) and from DDR/MSMC, using the K3 Data Movement Architecture (DMA) mechanism
- Load/Store Engine (LSE) which performs data load and store tasks on a the SL2 memory for the hardware accelerator algorithm cores
- VISS to LDC direct OTF (On-the-Fly) for multi-camera
- YUV422 10 and 12 bit format support on all units except NF (e.g. LDC/MSC)
- Simultaneous visual (HV) and analytics (MV) output to system memory including L3 flex connect, saving need for additional read from system memory for HV+MV processing.
- Chromatic Aberration Correction (CAC) to support lower cost lens
- RGBIR support

1.2.7 Depth and Motion Perception Accelerator

The Depth and Motion Perception Accelerator (DMPAC) computes dense stereo depth maps (*depth*) and dense optical flow vectors (*motion*) from camera inputs. The stereo and optical flow processing is partitioned into two top level sub-blocks: the Dense Optical Flow (DOF) engine and the Stereo Disparity Engine (SDE). The DOF and SDE blocks share a common local memory, DMA, external messaging and control infrastructure. The DMPAC provides the following main features, among others:

- Image resolution up to 2MPix (maximum horizontal resolution up to 2048 pixels; maximum vertical resolution up to 1024 pixels)
- Maximum throughput up to 220MPix/sec for DOF, and up to 84-100MPix/sec for SDE
- Simultaneous operation of Stereo (1MPix@30fps) and Optical flow (1MP@30fps)
- 12-bit fully packed luminance data input pixel data format (other formats supported through conversion)
- Packed 16-bit (disparity) or 32-bit (flow vector) output data formats
- Shared Level 2 (SL2) memory sub-system, which serves the data transfer of the DOF and SDE blocks
- Unified Transfer Controller (UTC), which serves as a DMA engine
- Hardware Thread Scheduler (HTS) block for messaging and control mechanism
- Counter, Timer and System Event Trace (CTSET) module, which provides event tracing capability for the Stereo and Optical Flow hardware threads

Copyright © 2024 Texas Instruments Incorporated

1.2.8 Navigator Subsystem

The Navigator subsystem (NAVSS) can be used for efficient transfer of data support between software, firmware and hardware in all combinations. It consists of the following main modules:

- Unified DMA Controller with the following main modes and features:
  - K3 DMA Architecture compliant Tx/Rx port implementation
  - K3 DMA Architecture compliant Packet-Oriented DMA Functionality (UDMA-P)
  - K3 DMA Architecture compliant Third Party Channel Controller
  - K3 DMA Architecture compliant Unified Transfer Controller
  - K3 DMA Architecture compliant Unified DMA channels which all share the execution hardware using time division multiplexing
- Ring Accelerator provides hardware acceleration to enable straightforward passing of work between a producer and a consumer and has the following main features:
  - Supports 1024 independent memory-mapped ring structures
  - Supports various modes for each ring based on usage and compatibility
  - Provides 2-words deep shared incoming Transfer Response FIFO
  - Provides bit-wide source VBUSM read/write slave interface for accesses from DMA controller entities
- Proxy module with the following main features:
  - Provides proxy buffers to store large data bursts that a host can only access in smaller amounts
  - Keeps the large data coherent until the complete data has been accessed
  - Allows interleaved access between multiple hosts using multiple proxies
  - Supports a pre-configured number of target resources to proxy with pre-configured number of channels, size of each channel, and address offset per each target
- Secure proxy module is a modified version of the proxy module and in addition has the following main features:
  - Supports a number of threads, where each has their own independent proxy function
  - Supports a programmable fixed queue for each proxy thread
  - Supports multiple producers all writing to the same queue
  - Supports programmable thresholds for when to generate events
  - Supports a max message count for outbound proxy threads limiting the number of messages a thread can produce
- Interrupt Aggregator modules provide a centralized machine which handles the termination of system events to that they can be coherently processed by the host(s) in the system. Main features are as follows:
  - 64-bit VBUSP slave using 64-bit registers
  - Provide a set of TI Interrupt Architecture compliant interrupt status and mask registers which are used to pass specific event status to one or more host blocks.
  - Provide a set of Global Event Input (GEVI) counters which can count events delivered via an ingress Event Transport Lane (ETL)
  - Provides a set of Local Event Input (LEVI) to Global event registers which can be used to convert pulsed discrete interrupt inputs or clock synchronous rising edge events into Global events on an egress ETL
  - Provides a set of GEVI 'Multicast' registers which can take a Global event from an ingress ETL and generate two egress Global events on two egress ETL interfaces
- Peripheral Virtualization Unit (PVU) module which provides TLBs (Translation Look-aside Buffers) for static virtual address translation on a CBA VBUSM bus with the following main features:
  - Implements a channelized TLB for virtual address translation
  - Supports a pre-configured number of entries per TLB channel
  - Supports TLB chaining to extend the search but at a latency penalty
  - Supports a 48-bit address size
  - Supports page sizes of 4KB, 16KB, 64KB, 2MB, 32MB, 512MB, 1GB, and 16GB
  - Support TLBs in software mode, where they are maintained by software only
  - Produces a fault interrupt when the TLB misses or upon a permission error
- Mailbox module to facilitate the communication between the various on-chip processors of the device by providing a queued mailbox-interrupt mechanism with the following main features:
  - 12 clusters

Copyright © 2024 Texas Instruments Incorporated

  - 32-bit message width
  - Message reception and queue-not-full notification using interrupts
  - Non-intrusive emulation
- Spinlock module (256 hardware semaphores) for synchronizing the processes running on multiple processors in the device.
- Timer Manager modules to support timing operations for the processes running on multiple processors, with the following main features:
  - 1024 × 32-bit RAM-based independent timers (2048 in total)
  - Event interface to an interrupt aggregation module in the NAVSS subsystem with events triggering when a timer expires or when an expired timer is reset or deactivated
  - Host access to determine which timer(s) expired
  - 32 registers with individual timeout status (one bit per timer)
  - Groups of 16 timers separated into pages of 4-K address space
  - Timer bits within each page to read expiration status for each timer when software only has access to that page
  - 10 µs time to cycle through all of the timers
  - Host access to reset individual timers
  - Periodic hardware timers – a timer may be set to automatically reprogram itself upon expiration without software intervention.
- Time Sync modules to facilitate host control of time sync operations, each with the following main features:
  - Supports a selection of multiple external clock sources
  - Software control of time sync events via interrupt or polling
  - Supports 8 hardware timestamp push inputs
  - Supports timestamp counter compare output
  - Supports timestamp counter bit output
  - Supports 6 timestamp generator function outputs
  - 32-bit and 64-bit timestamp modes
- Memory Cyclic Redundancy Check module used to perform CRC to verify the integrity of a memory system with the following main features:
  - Four channels to perform background signature verification on any memory subsystem
  - Data compression on 8-, 16-, 32-, and 64-bit data size
  - Dedicated CRC value register per channel which contains the pre-determined CRC value
  - Timed base event trigger from timer to initiate DMA data transfer
  - Programmable 20-bit pattern counter per channel to count the number of data patterns for compression
  - Three modes of operation: Auto, Semi-CPU, and Full-CPU
  - Timeout interrupt generation if CRC is not performed within the time limit
  - Per channel DMA request generation to initiate CRC value transfer

1.2.9 Region-based Address Translation Module

The Region-based Address Translation (RAT) module performs a region based address translation of a 32-bit input address into a 48-bit output address. The RAT provides the following main features:

- 16 regions with dedicated registers for attributes configuration:
  - Region base address
  - Region size
  - Translated base address
- Address translation for only enabled regions
- Region boundary crossing transactions error generation

1.2.10 Data Routing Unit

The Data Routing Unit (DRU) is a high bandwidth, flexible routing engine with programmable DMA transfer requests which enables performing of high speed data transfers between memory mapped slave endpoints, processor caches and shared caches. It behaves like a DMA transfer controller, moving data at MPU frequency and has the following main features:

- Programmable configuration registers for direct transfer request submission

Copyright © 2024 Texas Instruments Incorporated

- Read and write command queues
- Programmable priority for each queue
- Two dedicated ports (one read and one write) to generate independent read and write commands
- Support for region based and channelized firewall
- Independent 48-bit address fields for source and destinations
- Up to four dimensional data transfers
- Error detection and Correction
- CMMU support for IO virtualization
- Compression for DDR bandwidth reduction in CNN application and video stream data from VPAC

1.2.11 Display Subsystem

The Display Subsystem (DSS) is a flexible composition-enabled display subsystem that supports multiple high resolution display outputs. It consists of the following main modules:

- Display Controller (DISPC), with the following main features:
  - Support of multi-layer blending and transparency for each of display outputs
  - Supports write-back pipeline with scaling to enable memory-to-memory composition and/or to capture a display output for Ethernet video encoding
  - Supports gamma correction and programmable color control in both source and destination pipelines
  - Embedded DMA Controller with the following main features:
    - Support for 1D-only DMA transfers
    - Support for 48b addressable memory space
    - Support for memory fragmentation through external PAT at SoC level
    - Integrated shared buffer management for pipelines within the same DMA controller group
    - Programmable DMA requests management
    - Support for source image flip along X and Y-axis
    - Support for secure access to firewall protected frame buffer in DDR memory
  - Two input display processing Video Pipelines, each supporting:
    - Wide range of input RGB source pixel formats
    - Wide range of input YUV source pixel formats
    - Programmable poly-phase filter (scaler)
    - Programmable color space conversion
    - Programmable Brightness/Contrast/Hue/Saturation
    - Programmable Gamma Correction LUT
    - Luma Key generation
    - 10-bit processing pipeline
  - Two input display processing Video Lite Pipelines, each supporting:
    - Wide range of input RGB source pixel formats
    - Wide range of input YUV source pixel formats
    - YUV420 to YUV422 chroma up-sampling using an average filter
    - YUV422 to YUV444 chroma up-sampling using a 4-tap filter based on Catmull-Rom algorithm
    - Programmable color space conversion
    - Programmable Brightness/Contrast/Hue/Saturation
    - Programmable Gamma Correction LUT
    - Luma Key generation
    - 10-bit processing pipeline
  - One Write-back (WB) pipeline, supporting:
    - Wide range of destination RGB pixel formats
    - Wide range of destination YUV pixel formats
    - Programmable poly-phase filter (scaler)
    - Output capture and Memory-to-memory (M2M) operation modes
  - Four Overlay Managers (OVR), each supporting:
    - Input pixel format: ARGB48-12121212
    - Output pixel format: ARGB48-12121212

Copyright © 2024 Texas Instruments Incorporated

    - Overlay of the input pipelines
    - Up to 5 input layers blending
    - Transparency color key
    - Alpha blending support: Embedded pixel alpha (ARGB and RGBA), global pixel, and combination of global pixel and pixel alpha
    - Z-order programmable (full flexibility)
    - Color bar test pattern insertion
    - Any overlay output can be selected to drive the Write-back pipeline
  - Four Video Port (VP) display outputs, each supporting:
    - 36-bit per pixel on the RGB output interface
    - Independent programmable timing generator, supporting up to 600 MHz pixel clock video formats
    - Independent programmable 10-bit gamma correction
    - Independent programmable multiple cycles output format on 8/9/12/16-bit interface
    - Selection between RGB and YUV422 output pixel
    - Configurable VP output mode
  - Internal diagnostic features:
    - Supports up to 4 programmable (position/size) check regions on the DISPC video port display outputs
    - Support for 1 check region on each input video pipeline output
    - MISR (Multiple Input Signature Register) used on each check region to perform data correctness check and/or freeze frame detection
  - Local power features:
    - Low power saving modes
    - On-the-fly Dynamic Frequency Scaling (DFS) support
    - Capability to associate all buffers a single pipeline for a display self-refresh
  - System interconnect ports:
    - Two 128-bit VBUSM master interfaces for data read/write
    - One 32-bit VBUSP slave interface for configuration
- Fram Buffer Decompression Core (FBDC), that performs a decompression on lossless compressed images on a tile-by-tile basis.
- MIPI Display Serial Interface (DSI) transmitter host controller, with the following main features:
  - Compliance with MIPI DSI 1.3.1 and previous protocol specifications
  - Compliance with Stereoscopic Display Format (SDF) specification
  - Video and command operational modes
  - Both burst and non-burst modes for video mode data transmission
  - Up to 4 virtual channels via command mode
  - Bi-directional communication and escape mode
  - Pixel clock rate range: 25-330 MHz
  - Programmable display resolutions
  - 16/18/24/30/36-bit RGB input data formats for video mode
  - RGB16, RGB18 packed, and RGB24 input data formats for command mode
  - All generic data types defined by MIPI
  - Display Command Set (DCS) transparent to the protocol engine
  - ECC on the APB interface
  - Data splitter for 2-,3-, or 4-data lane configuration
  - Connection to a single MIPI D-PHY complex I/O through an 8-bit Protocol Peripheral Interface (PPI)
  - Tearing effect (TE) input signals for command mode display
  - Bus contention recovery
  - Video mode pattern generator: color bar pattern image and D-PHY BET testing pattern
  - APB slave interface with 32-bit data and address for configuration
- The MIPI DSI Physical Layer (D-PHY) module with the following main features:
  - Compliance with MIPI D-PHY 1.2 physical layer interface specification and features
  - 1, 2 or 4 data lanes, in addition to clock signaling
  - Maximum data rate up to 2.5 Gbps per data lane
  - Protocol Peripheral Interface (PPI)

Copyright © 2024 Texas Instruments Incorporated

  - HS continuous and burst mode
  - Low-Power (LP), Ultra-Lower Power Mode (ULPM), and Shutdown modes
  - Forward direction and reverse direction escape modes
  - Automatic termination control in both high-speed and low-power modes
  - Single 32-bit VBUSP slave interface
- Embedded DisplayPort (eDP) transmitter host controller with the following main features:
  - Compliance with VESA® DisplayPort™ (DP) 1.3 (with 1.4 DSC/FEC support) specification
  - Compliance with VESA Embedded DisplayPort (eDP) 1.4 specification
  - Static configuration of either DP or eDP mode
  - Link rates up to High Bit Rate 3 (HBR3)
  - Pixel clock rate range: 25-600 MHz
  - 8, 10, and 12 bpc (bits per component), in RGB/YCbCr444 colorimetry formats (CEA-861 compliant) and YCbCr422 (via simple decimation)
  - Data splitter for 1-, 2-, or 4-data lane configuration
  - Single Stream Transport (SST)
  - Multiple Stream Transport (MST)
  - High-bandwidth Digital Content Protection (HDCP) data encryption via an embedded HDCP core
  - Display Stream Compression (DSC) encoded stream data transport via an embedded DSC core
  - Forward Error Correction (FEC) encoder with/without DSC enabled in DP mode
  - Single Stream Transport (SST)
  - Audio transport features
  - Metadata transport via Main Stream Attribute (MSA) packet or via SDP
  - APB slave ports for TX/PHY controller configuration
  - SAPB (secure) slave port for secure connection
  - Video source muxing options
  - One 32-bit VBUSP slave interface used for configuration
  - ECC on the critical memories
  - Parity check on the configuration interface
  - Encoder self-check diagnostics support in the DSC core
  - Injection of ECC and parity errors
- eDP (Physical Layer) SERDES and Aux PHY modules with the following main features:
  - DP1.3, HBR3 and eDP1.4a HBR3 throughput
  - 1, 2, or 4 lanes at 1.62Gbps, 2.7Gbps, 5.4Gbps, and 8.1Gbps per lane
  - Additional link rates (2.16, 2.43, 3.24, 4.32Gbps) per lane in eDP mode
  - Reduced differential voltage swing (0.2/0.25/0.30/0.35/0.40/0.45) in eDP mode
  - Hot Plug Detect (HPD) for connection detection and interrupt from sink
  - Integrated Low Jitter, Fixed Bandwidth PLL
  - DisplayPort physical layer functionalities:
    - Scrambler
    - 8/10-bit encoder (within the eDP transmitter)
    - Inter Lane Skew Insertion
    - Training Pattern Generation – TPS1,2,3,4 PRBS7 and 80-bit custom training pattern generation (bypassing the scrambler and encoder)
- 1 Mbps AUX PHY for link training, DPCD register access, HDCP authentication and EDID access

1.2.12 Camera Subsystem

Camera Subsystem unites three camera streaming interfaces – receiver and transmitter, allowing the device to stream video inputs from multiple cameras to the video processing accelerator (VPAC) or to internal memory and to output CSI-2 protocol image data to any device that supports MIPI CSI-2 protocol. Main modules are as follows:

- Camera Streaming Interface Receiver (CSI_RX_IF) with the following main features:
  - Compliant to MIPI CSI-2 v1.3+ and MIPI CSI-2 v2.0
  - Supports up to 16 virtual channels per input
  - Supports one 4MP camera or eight 2MP camera streams

Copyright © 2024 Texas Instruments Incorporated

  - Supports data rate up to 2.5 Gbps per lane (wire rate)
  - Supports 1, 2, 3, or 4 Data Lane connections to MIPI D-PHY Receiver (DPHY_RX)
  - Over 25 different programmable formats including YUV420, YUV422, RGB, Raw, and User Defined
  - Supports four independent (simultaneous) output streams:
    - Two VP 32-bit streams to VISS inputs of VPAC image processing accelerator
    - One (up to 4 channels) PPI 16-bit pixel retransmission interface to Camera Streaming Interface Transmitter (CSI_TX_IF)
    - One (up to 32 Channels) DMA interface through a 128-bit Packet Streaming Interface Link (PSI_L) connection to NAVSS for transfers to memory:
- Functional and data path error interrupts
- ECC support
- MIPI D-PHY Receiver (DPHY_RX) with the following main features:
  - Allows the device to input video streams from external sensor cameras and other CSI2 compliant sources
  - Compliant to MIPI D-PHY standard v1.2
  - Supports up to 4 data and 1 clock lanes
  - Supports up to 2.5 Gbps (with deskew) and 1.5 Gbps (without deskew) per data lane
  - Clock lane Control / Interface logic type: CIL-SCNN for HS and low power receiving
  - Data lane Control / Interface logic type: CIL-SFAN for HS and low power receiving
  - Data lanes can be independently operated in HS or ULP mode
  - Swapping of DP/DN signals within each clock/data pair
- Camera Streaming Interface Transmitter (CSI_TX_IF) with the following main features:
  - Compliant to MIPI CSI-2 v1.3+, MIPI CSI-2 v2.0, and MIPI D-PHY v1.2
  - Data rate up to 2.5 Gbps per lane (wire rate)
  - Supports 1, 2, 3, or 4 Data Lane connections to MIPI D-PHY Transmitter (DPHY_TX)
  - Over 25 different programmable formats including YUV420, YUV422, RGB, Raw, and User Defined
  - Support of 16 virtual channels
  - Support of four configurable input streams

1.2.13 Shared D-PHY Transmitter

The DPHY_TX module provides an option for video output interfacing by implementing a four lane, MIPI D-PHY Transmitter. DPHY_TX module supports the following main features:

- Compliancy to MIPI D-PHY Standard version 1.2
- Supports up to 4 data lanes
- Supports up to 2.5 Gbps (with deskew) and 1.5 Gbps (without deskew) per data lane
- Supports Escape mode
- Data Lanes can be independently operated in HS or ULP mode
- Includes a CMN block with reference generators / resistor calibration and an integrated PLL
- Fault detection

1.2.14 Multicore Shared Memory Controller

Multicore Shared Memory Controller (MSMC) provides high-bandwidth resource access both to and from all of the connected processing elements and the rest of the system and supports the following main features:

- 4MB (4 banks x 1MB) SRAM with ECC
- 512-bit processor port bus and 40-bit physical address bus
- Coherent unified bi-directional interfaces to connect to processors or device masters
- One infrastructure master interface
- Single external memory master interface
- Support of distributed virtual system
- Bandwidth management with starvation bound
- Two-level Quality-of-Service (QoS) support for real-time/non-real-time split
- Security firewall for SRAM/cache and external memory
- ECC error protection
- Trace and debug features
- Support of dynamic clock gating on all logic units

Copyright © 2024 Texas Instruments Incorporated

1.2.15 DDR Subsystem

The DDR Subsystem (DDRSS) is used as an interface to external SDRAM devices which can be utilized for storing program or data. DDRSS provides the following main features:

- Support of LPDDR4 memory type
- 32-bit memory bus interface with in-line ECC
- Up to 8 GB per DDRSS across 2 ranks (4 GB per rank)
- System bus interface: little endian only with 256-bit data width
- Configuration bus Interface: little endian only with 32-bit data width
- Support of dual rank configuration
- Support of automatic idle power saving mode when no or low activity is detected
- Class of Service (CoS) - three latency classes supported
- Prioritized refresh scheduling
- Statistical counters for performance management

1.2.16 General Purpose Input/Output Interface

The General Purpose Input/Output (GPIO) modules provide dedicated general-purpose pins that can be configured as either inputs or outputs. Modules main features are:

- Support of 9 banks x 16 GPIO pins
- Support of up to 9 banks of interrupt capable GPIOs
- Interrupts can be triggered by rising and/or falling edge, specified for each interrupt capable GPIO pin
- Set/clear functionality per individual GPIO pin

1.2.17 Inter-Integrated Circuit Interface

The multi-master Inter-Integrated Circuit (I2C) interfaces support the following main features:

- Compliancy to the Philips I2C-bus specification version 2.1
- Support of standard mode (up to 100 Kbps) and fast mode (up to 400 Kbps)
- Support of 7-bit and 10-bit device addressing modes
- Support of multi-master transmitter/slave receiver and receiver/slave transmitter modes
- Built-in FIFOs with programmable size of 8 to 64 bytes for buffered read or write
- 8-bit-wide data access
- Support of Auto Idle, Idle Request/Idle Acknowledge handshake, and Asynchronous Wakeup mechanisms
- Low power consumption

1.2.18 Improved Inter-Integrated Circuit Interface

The multi-master Improved Inter-Integrated Circuit (I3C) module supports the following main features:

- Supports Single Data Rate (SDR) and High Data Rate – Dual Data Rate (HD-DDR) communication modes
- Support of Common Command Codes (CCC)
- Hot-Join capability
- In Band Interrupts (IBI)
- Support of Dynamic Address Assignment (DAA)
- Support of Static Addressing (SA)
- FIFO Buffers
- Registers to store the parameters for the response to an IBI interrupt from a number of slaves

1.2.19 Multi-channel Serial Peripheral Interface

The Multi-channel Serial Peripheral Interface (MCSPI) module supports the following main features:

- Serial clock with programmable frequency, polarity, and phase for each channel
- Wide selection of MCSPI word lengths, ranging from 4 to 32 bits
- Up to four master channels, or single channel in slave mode
- Support of different master multichannel modes
- Single interrupt line for multiple interrupt source events
- Support of start-bit write command
- Support of start-bit pause and break sequence
- Built-in FIFO available for a single channel

Copyright © 2024 Texas Instruments Incorporated

1.2.20 Universal Asynchronous Receiver/Transmitter

The configurable Universal Asynchronous Receiver/Transmitter (UART) interface supports the following main features:

- 16C750-compatible
- Support of RS-485 external transceiver auto flow control
- Dual 64-byte FIFOs – one per each received and transmitted data paths
- Programmable and selectable transmit and receive FIFO trigger levels for DMA and interrupt generation
- Programmable sleep mode
- Baud rates up to 3.6 Mbps with 48 MHz functional clock
- Auto-baud between 1200 bits/s and 115.2 Kbits/s (only when 48 MHz function clock is used)
- Support of IrDA 1.4 Slow Infrared (SIR), Medium Infrared (MIR), and Fast Infrared (FIR) communications
- Support of Consumer Infrared Remote control mode (CIR) with programmable data encoding

1.2.21 Peripheral Component Interconnect Express Subsystem

The Peripheral Component Interconnect express (PCIe) subsystem with shared SerDes lines provides the following main features:

- Compliant to PCI-Express® Base Specification, Revision 4.0 (Version 0.7)
- 4-lane configuration with up to 8.0 Gbps/lane (Gen3). Can be used in 1x4, 1x2, 1x1 modes.
- Gen3 (8 Gbps 128/130-bit encoding), Gen2 (5 Gbps 8/10-bit encoding), and Gen1 (2.5 Gbps 8/10-bit encoding) with auto-negotiation
- Dual mode: Root Port (RP) or End Point (EP) operation modes, selectable via bootstrap pins
- Dynamic PIPE width change when switching between Gen1/2/3 modes
- Constant 32-bit PIPE width for Gen1/2/3 modes
- Maximum payload size of 256 bytes
- Maximum remote read request size of 4KB
- Single-root I/O Virtualization (SR-IOV) with Physical Functions (PF) and Virtual Functions (VF) in End Point mode
- Maximum number of non-posted outstanding transactions: 32
- Resizable Base Address Registers (BAR) capability
- Separate Reference Clock with Independent Spread (SRIS)
- Legacy, MSI and MSI-X Interrupt Support
- 32 outbound address translation regions
- Precision time measurement (PTM)

1.2.22 Universal Serial Bus (USB) Subsystem

The Universal Serial Bus (USB) subsystem with integrated PHY has the following main features:

- Dual-Role Device (DRD) capability
- Compliance with USB 3.1 Gen1 Specification
- Support of Peripheral (aka Device) mode at Super Speed (SS at 5 Gbps), High Speed (HS at 480 Mbps), and Full Speed (FS at 12 Mbps)
- Support of Host mode at SS (5 Gbps), HS (480 Mbps), FS (12 Mbps), and Low Speed (LS at 1.5 Mbps)
- Support of Host Negotiation Protocol (HNP)
- Support of USB3 low power protocol states (U0, U1, U2, and U3)
- USB instance contains a single xHCI compliant with xHCI 1.0 specification with internal DMA controller
- ECC on internal RAMs
- Embedded USB 2.0 PHY

1.2.23 SerDes

The Serializer/Deserializer (SERDES) Multi-protocol Multi-link modules support the following main blocks:

- Quad lane PHY with common module for peripheral and Tx clocking handling
- Physical coding sub-block for data translation from/to the parallel interface, as well as data encoding/decoding and ymbol alignment
- MUX module for device interfaces multiplexing into a single SERDES lane (Tx and Rx)
- A wrapper for sending control and reporting status signals from the SerDes and muxes

Copyright © 2024 Texas Instruments Incorporated

1.2.24 General Purpose Memory Controller with Error Location Module

The General-Purpose Memory Controller (GPMC) with Error Location Module (ELM) is dedicated for interfacing with external memory devices and has the following main features:

- Support of 8- or 16-bit-wide data path to external memory devices
- Supports up to 4 independent chip-select regions of programmable size and programmable base addresses on 16MB, 32MB, 64MB, or 128MB boundary in a total address space of 1GB
- Support of the following wide range of external memories/devices:
  - Asynchronous or synchronous 8-bit wide memory or device (non-burst device)
  - Asynchronous or synchronous 16-bit wide memory or device
  - 16-bit non-multiplexed NOR flash device
  - 16-bit address and data multiplexed NOR flash device
  - 8-bit and 16-bit NAND flash device
  - 16-bit pseudo-SRAM (pSRAM) device
- Supports various interface protocols when communicating with external memory or external devices:
  - Asynchronous read/write access
  - Asynchronous read page access (4, 8, and 16 Word16)
  - Synchronous read/write access
  - Synchronous read burst access without wrap capability (4, 8, and 16 Word16)
  - Synchronous read burst access with wrap capability (4, 8, and 16 Word16)
- Supports on-the-fly error code detection using the Bose-Chaudhuri-Hocquenghem (BCH) or Hamming code to improve the reliability of NAND with a minimum effect on software (NAND flash with 512-byte page size or greater)
- ELM module which used in a conjunction with the GPMC and provides ECC calculation (up to 16-bit) for NAND support and ability to work in both page-based and continuous modes, has the following main features:
  - 4, 8, and 16 bits per 512-byte block error-location, based on BCH algorithms
  - Eight simultaneous processing contexts
  - Page-based and continuous modes
  - Interrupt generation on error-location process completion

1.2.25 Multimedia Card/Secure Digital Interface

The Multimedia Card/Secure Digital (MMCSD) controller supports the following main features:

- 8-bit or 4-bit wide data bus, depending on instance
- Support of eMMC5.1 Host Specification (JESD84-B51)
- Support of SD Host Controller Standard Specification - SDIO 3.00
- Integrated DMA controller supporting SD Advanced DMA - ADMA2 and ADMA3
- eMMC Electrical Standard 5.1 (JESD84-B51)
- Multimedia card features:
  - Backward compatible with earlier eMMC standards
  - Legacy MMC SDR: 1.8 V, 8/4/1-bit bus width, 0-25 MHz, 25/12.5/3.125 MB/s
  - High Speed SDR: 1.8 V, 8/4/1-bit bus width, 0-50 MHz, 50/25/6.25 MB/s
  - High Speed DDR: 1.8 V, 8/4-bit bus width, 0-50 MHz, 100/50 MB/s
  - HS200 SDR: 1.8 V, 0-200 MHz, 8/4-bit bus width, 200/100 MB/s
- SD card support: SDIO, SDR12, SDR25, SDR50, DDR50
- System bus interface: CBA 4.0 VBUSM master port with 64-bit data width and 64-bit address, little endian only
- Configuration bus interface: CBA 4.0 VBUSM with 32-bit data width, 32-bit aligned accesses only, linear incrementing addressing mode, little endian only

1.2.26 Enhanced Capture Module

The Enhanced Capture (ECAP) module provides accurate timing for different events. When not being used for event capture, its resources can be used to generate a single channel of asymmetrical PWM waveforms (configurable as either one capture input, or as one auxiliary PWM output). The ECAP module supports the following main features:

Copyright © 2024 Texas Instruments Incorporated

- 32-bit time base counter
- 4 x 32 bits event time-stamp capture registers
- 4 stage sequencer (Mod4 counter), synchronized to external events
- Independent edge polarity selection for up to four sequenced time-stamp capture events
- Input capture signal pre-scaling (from 1 to 16)
- Interrupt capabilities on any of the four capture events
- Support of different capture modes (single shot capture, continuous mode capture, absolute timestamp capture or delta mode time-stamp capture)

1.2.27 Enhanced Pulse-Width Modulation Module

The Enhanced Pulse-Width Modulation (EPWM) module supports the following main features:

- Dedicated 16-bit time-base counter with period and frequency control
- Two independent PWM outputs that can be used in different configurations (with single-edge operation, with dual-edge symmetric operation or one independent PWM output with dual-edge asymmetric operation)
- Asynchronous override control of PWM signals through software
- Programmable phase-control support for lag or lead operation relative to other EPWM modules
- Dead-band generation with independent rising and falling edge delay control
- Programmable trip zone allocation of both cycle-by-cycle trip and one-shot trip on fault conditions
- Events enabling to trigger both CPU interrupts and start of ADC conversions

1.2.28 Enhanced Quadrature Encoder Pulse Module

The 32-bit Enhanced Quadrature Encoder Pulse (EQEP) module for position, speed, and frequency measurements supports the following main features:

- Input synchronization
- Three stage/six stage digital noise filter
- Quadrature decoder unit
- Position counter and control unit for position measurement
- Quadrature edge capture unit for low speed measurement
- Unit time base for speed/frequency measurement
- Watchdog timer for detecting stalls

1.2.29 Controller Area Network

The Controller Area Network (MCAN) interface supports both classic CAN and CAN FD (CAN with Flexible Data-Rate) specifications and has the following main features:

- Conforms with CAN Protocol version 2.0 part A, B and ISO 11898-1:2015
- Full CAN FD (up to 64 data bytes) support
- SAE J1939 support
- AUTOSAR support
- Up to 32 dedicated transmit buffers and 64 dedicated receive buffers
- Two configurable receive FIFOs, up to 64 elements each
- Configurable transmit FIFO, up to 32 elements
- Configurable transmit queue, up to 32 elements
- Configurable transmit event FIFO, up to 32 elements
- Up to 128 filter elements
- Maskable interrupts, two interrupt lines
- Timestamp Counter

1.2.30 Audio Tracking Logic

The Audio Tracking Logic (ATL) module, which is used by HD Radio™ applications to synchronize the digital audio output to the baseband clock, supports the following main features:

- Contains four ATL instances, for HD Radio support and asynchronous sample rate conversion assistance
- Each instance tracks the time error between two syncs (local Audio Word Select [AWS] and Baseband Word Select [BWS])
- Each instance selects between 8 mux choices for BWS and 8 mux choices for AWS

Copyright © 2024 Texas Instruments Incorporated

- Each instance generates modulated ATCLK clock signals with software-initiated pulse stealing
- Selection between interface or functional clock to run error counting timers and to derive modulated clock outputs
- Clock and reset management: receives clock and reset signals from the device PSC module
- Hardware reset
- Local software reset

1.2.31 Multi-channel Audio Serial Port

The Multi-channel Audio Serial Ports (MCASP) is a general purpose audio serial port, useful for Time-Division Multiplexed (TDM) stream, Inter-IC Sound (I2S) protocols reception and transmission as well as for an inter-component Digital audio Interface Transmission (DIT). The MCASP module has the following main features:

- Connection to audio Analog-to-Digital Converters (ADC), Digital-to-Analog Converters (DAC), codec, digital audio interface receiver (DIR), and Sony/Philips Digital Interface (S/PDIF) transmit physical layer components
- Support of Time Division Multiplexed (TDM) interface, Inter-IC Sound (I2S) standard, and similar bit stream formats
- Integrated Digital Audio Interface Transmitter (DIT) with enhanced channel status/user data RAM and support of S/PDIF, IEC60958-1, and AES-3 formats
- Independent serializer for each AXRx channel of each MCASP module
- A single 32-bit buffer per serializer for transmit and receive operations
- Support for two DMA requests (one per direction)
- One transmit and one receive interrupt requests common for all serializers
- Two independent clock generator modules for transmit and receive allowing the MCASP to receive and transmit at different rates
- Support of up to 16 serial data pins

1.2.32 Timers

There are three different types of timer modules:

- The Global Time Counter (GTC) module can be used for time synchronization and debug trace time stamping with the following main features:
  - 64-bit up counter
  - No rollover during the lifetime of the device
  - Compatible with Armv8 system counter requirements
  - Outputs reflected binary (Gray) encoded timer value for system timer bus distribution to other modules
  - Selectable counter bit output as a push event that can be used by CPTS modules, timers or interface protocols
- The Windowed Watchdog Timer (WWDT), implemented by using the Digital Windowed Watchdog (DWWD) function of the Real Time Interrupt (RTI) module providing timer functionality for operation systems and benchmarking code with the following main features:
  - Two independent 64 bit counter blocks
  - Four configurable compare registers for generating operating system ticks
  - Free running counter 0 can be incremented by either the internal prescale counter or by an external event
  - Selectable RTI clock input (derived from any of the available clock sources)
  - Fast enabling/disabling of events
- The Timer module with support of the following main features:
  - Free running 32-bit upward counter
  - Generates a 1-ms tick with a 32.768 kHz functional clock
  - Interrupts generated on overflow, compare and capture
  - Supported modes of operation: compare and capture, auto-reload and start-stop
  - Programmable divider clock source (2^n, where n = [0-8])
  - Dedicated input trigger for capture mode and dedicated output trigger/PWM signal
  - On-the-fly read/write register (while counting) for systems operation and benchmarking code

1.2.33 Internal Diagnostics Modules

Internal diagnostics modules provide monitoring and diagnostic functions required to achieve certain safety compliance levels:

Copyright © 2024 Texas Instruments Incorporated

- Dual Clock Comparator (DCC) modules, used to determine the accuracy of a clock signal during the time execution of an application, each having the following main features:
  - Two independent counter blocks count clock pulses from each clock source
  - Each counter block is programmable, however, for proper operation the counters must be programmed with seed values that respect the ratio of the two clock frequencies
  - Configurable time base for error signal
  - Error signal generation when one of the clocks is out of specification
  - Clock frequency measurement
- Error Signaling Module (ESM) for safety-related events and/or errors aggregation from throughout the device into one location supports the following main features:
  - Up to 1024 level or pulse error event inputs
  - Selectable low and high priority interrupt error pin prioritization of each error event
  - Error pin to signal severe device failure
  - Configurable time base for error signal
  - Error forcing capability
  - Internal redundant flops on safety critical fields
- ECC aggregator modules supporting ECC mechanism for providing increased system reliability via reduction of memory software errors by allowing single bit errors to be detected and corrected (SEC) and double bit errors to be detected (DED). Applied to different memories in many of the subsystems, each of the ECC aggregators has the following main features:
  - Reduces memory software errors via single error correction (SEC) and double error detection (DED)
  - Provides a mechanism to control and monitor the ECC RAMs in a module or subsystem
  - Supports software readable status of ECC errors (single and double-bit) and associated info such as RAM address and data bit or bits that are in error
  - Aggregates level pending status from the ECC RAMs in two interrupts to the device CPU – interrupt for correctable error (SEC) and interrupt for uncorrectable error (DED)
  - Supports up to 256 ECC endpoints (either ECC RAM or interconnect ECC component)
  - Single bit error detection via parity checking results in a non-correctable error interrupt
- Memory Cyclic Redundancy Check (MCRC) module used to perform CRC to verify the integrity of a memory system – part of the NAVSS

1.2.34 Analog-to-Digital Converter

The Analog-to-Digital Converter (ADC) module contains a single 12-bit ADC which can be multiplexed to any 1 of 8 analog inputs (channels) and supports the following main features:

- 4 MSPS rate with a 60 MHz sample clock
- Single-ended or differential input options
- ADC module can be configured and transformed into digital test inputs
- Programmable 16 steps Finite State Machine (FSM) sequencer

1.2.35 Gigabit Ethernet Switch

The 2-port Gigabit Ethernet Switch (CPSW) subsystem provides Ethernet packet communication for the device and has the following main features:

- One Ethernet port (port 1) with selectable RGMII and RMII interfaces and an internal Communications Port Programming Interface (CPPI) port (port 0)
- Synchronous 10/100/1000 Mbit operation
- Flexible logical FIFO-based packet buffer structure
- Eight priority level Quality Of Service (QOS) support (802.1p)
- Support for Audio/Video Bridging (P802.1Qav/D6.0)
- Support for IEEE 1588 Clock Synchronization (2008 Annex D, Annex E and Annex F)
- DSCP Priority Mapping (IPv4 and IPv6)
- Energy Efficient Ethernet (EEE) support (802.3az)
- Flow Control Support (802.3x)
- Wire rate switching (802.1d)
- Non-Blocking switch fabric

Copyright © 2024 Texas Instruments Incorporated

- Time Sensitive Network Support
- Address Lookup Engine (ALE)
- EtherStats and 802.3Stats Remote Network Monitoring (RMON) statistics gathering (per port statistics)
- Ethernet Mac transmit to Ethernet Mac receive Loopback mode (digital loopback) supported
- CPSGMII Loopback Modes (transmit to receive)
- Maximum frame size of 2024 bytes
- Management Data Input/Output (MDIO) module for PHY Management with Clause 45 support
- Programmable interrupt control with selected interrupt pacing
- Host port CPPI Streaming Packet Interface (CPPI_GCLK)
- Digital loopback and FIFO loopback modes supported
- Emulation support
- Full duplex mode supported in 10/100/1000 Mbps. Half-duplex mode supported only in 10/100 Mbps modes only
- RAM Error Detection and Correction (SECDED)

1.2.36 Octal Serial Peripheral Interface and HyperBus Memory Controller as a Flash Subsystem

The Flash Subsystem (FSS) provides access to external flash devices via Octal Serial Peripheral Interface (OSPI) and HyperBus interface along with encryption/decryption, authentication, and in-line ECC protection. FSS supports the following main features:

- Provides two OSPIs or one OSPI and one HyperBus flash interfaces
- OSPI0/HyperBus interface supports:
  - Execute in place (XIP) operation
  - 32-byte Block Copy (BC) operation
  - ECC and/or authentication with four configurable authentication regions and authentication on 32-byte blocks
- OSPI supports up to 4 devices
- OSPI supports single, dual, quad, or octal SPI devices
- HyperBus interface supports up to 2 devices
- The OSPI and HyperBus interface have independent power management for low power operations

1.2.37 Security Management Subsystem

The Security Management Subsystem (SMS) which provides control over the device boot sequencing, device management, and security. With the factory-sealed firmware, SMS main functions include:

- Device management (security only)
- Device boot configuration and sequence
- Secure boot setup
- Decryption routines
- Firewall control for isolation and Security
- Runtime Security Management and resource allocation

Arm Cortex-M4F based SMS acts as a system security master and protects critical security assets during run-time. As part of booting a High Security (HS) device, SMS uses on-chip keys to establish root-of-trust and authenticate images to reinforce trust. SMS acts also as main boot processor and as such is the very first subsystem that is brought out of reset after device power-on-reset.

Main components of the SMS are:

- Two independent M4F processor cores with floating point extension (primary and secondary)
- M4F primary core features include:
  - RTI/WDT (only digital watchdog (non-windowed) feature is supported in SMS primary core context)
  - 128KB IMEM and 48KB DMEM, accessible from M4F primary core and system masters via firewall
  - Messaging between M4F core and host processors using Secure Proxy and RA located in MCU_NAVSS and MAIN NAVSS.
  - 160KB ROM for boot of M4F core
- M4F secondary core features include:
  - Dedicated RTI/WWDT (with windowed watchdog mode, disabled by default)

Copyright © 2024 Texas Instruments Incorporated

  - 192KB IMEM and 64KB DMEM, accessible from M4F secondary core and system masters via firewall
  - Messaging between M4F secondary core and up to five host cores (including primary core and 4 other device level cores) using Secure Proxy and RA
- Common features of both M4F primary and secondary cores:
  - Following resources can be accessed from either primary or secondary core with permissions via firewall:
    - Four 32-bit Timers - same as SOC level timers
    - AES engine with 128, 192 and 256-bit support
    - Security Manager
    - SMS control module - contains various control, configuration and status MMRs including firewall management for the full device
  - Other core features:
    - Ability to execute code from unified memory or external memories
    - Up to 240 input interrupts, level or pulse interrupts, capable of waking up the SMS cores from low power mode
    - Two interrupt outputs (per M4F core) to host SOC; support of both level and pulse interrupts
    - One fault detected interrupt output (per M4F core); support of both level and pulse interrupts
    - DAP based debug interface to the M4F core
    - ITM trace to chip level trace framework
    - Support of double detection and single error correction
    - Support of Little Endian mode only
    - In addition to local SMS RAM, the SMS M4F cores may utilize MSMC memory space as secure RAM via firewall

1.3 Device Identification

The manufacturer identity, the boundary scan part number, and the silicon revision of the device can be read in the WKUP_CTRL_MMR0_JTAGID register. See Table 1-2 and Table 1-3, for more information.

Table 1-2. Device JTAG ID

WKUP_CTRL_MMR0_JTAGID Register Field	Value	Comment
[31-28] VARIANT	See Table 1-3	Silicon Revision (SR) identifier.
[27-12] PARTNO	See Table 1-3	Part number for boundary scan.
[11-1] MFG	0x17	Manufacturer identity (TI).
[0] LSB	0x1	Always reads 1.

Table 1-3. Device JTAG ID Values

Silicon Type	VARIANT	PARTNO	WKUP_CTRL_MMR0_JTAGID Register Value
TDA4VE SR1.0 TDA4VL SR1.0 TDA4AL SR1.0	0x0	0xBB75	0x0BB7502F

2 Memory Maps

2.1 MAIN Memory Map.....27
2.2 MCU Memory Map.....81
2.3 WKUP Memory Map.....86
2.4 Processors View Memory Map.....87
2.5 Region-based Address Translation.....89

Copyright © 2024 Texas Instruments Incorporated

2.1 MAIN Memory Map

Table 2-1. MAIN Memory Map

Region Name	Start Address	End Address	Size
PSRAM2KECC0_RAM	0x0000000000	0x0000000800	2 KB
CTRL_MMR0_CFG0	0x0000100000	0x0000120000	128 KB
PSRAMECC0_RAM	0x0000200000	0x0000200400	1 KB
PSC0	0x0000400000	0x0000401000	4 KB
PLLCTRL0	0x0000410000	0x0000410200	512 B
GPIO0	0x0000600000	0x0000600100	256 B
GPIO2	0x0000610000	0x0000610100	256 B
GPIO4	0x0000620000	0x0000620100	256 B
GPIO6	0x0000630000	0x0000630100	256 B
PLL0_CFG	0x0000680000	0x00006A0000	128 KB
ESM0_CFG	0x0000700000	0x0000701000	4 KB
AM_MAIN_INFRA_TO_MAIN_INFRA_STOG0_CFG	0x0000780000	0x0000780400	1 KB
DCC0	0x0000800000	0x0000800040	64 B
DCC1	0x0000804000	0x0000804040	64 B
DCC2	0x0000808000	0x0000808040	64 B
DCC3	0x000080C000	0x000080C040	64 B
DCC4	0x0000810000	0x0000810040	64 B
DCC5	0x0000814000	0x0000814040	64 B
DCC6	0x0000818000	0x0000818040	64 B
DCC7	0x000081C000	0x000081C040	64 B
DCC8	0x0000820000	0x0000820040	64 B
DCC9	0x0000824000	0x0000824040	64 B
GPIOMUX_INTRTR0_CFG	0x0000A00000	0x0000A00800	2 KB
CMPEVENT_INTRTR0_CFG	0x0000A30000	0x0000A30200	512 B
TIMESYNC_INTRTR0_INTR_ROUTER_CFG	0x0000A40000	0x0000A40800	2 KB
GTC0_GTC_CFG0	0x0000A80000	0x0000A80400	1 KB
GTC0_GTC_CFG1	0x0000A90000	0x0000A94000	16 KB
GTC0_GTC_CFG2	0x0000AA0000	0x0000AA4000	16 KB
GTC0_GTC_CFG3	0x0000AB0000	0x0000AB4000	16 KB
MAIN_CBASS0_ERR	0x0000B00000	0x0000B00400	1 KB
CBASS_INFRA_NON_SAFE0_ERR	0x0000B04000	0x0000B04400	1 KB
CBASS_FW0_ERR	0x0000B08000	0x0000B08400	1 KB
PSRAMECC0_ECC_AGGR	0x0000C00000	0x0000C00400	1 KB
PSRAM2KECC0_ECC_AGGR	0x0000C01000	0x0000C01400	1 KB
ECC_AGGR0_ECC_AGGR	0x0000C02000	0x0000C02400	1 KB
PBIST0	0x0000D00000	0x0000D00400	1 KB
DFTSS0	0x0000D10000	0x0000D10400	1 KB
PBIST1	0x0000D20000	0x0000D20400	1 KB
PBIST4	0x0000D30000	0x0000D30400	1 KB
COMPUTE_CLUSTER0_GIC_TRANSLATER	0x0001000000	0x0001400000	4 MB
COMPUTE_CLUSTER0_GIC_DISTRIBUTOR	0x0001800000	0x0001810000	64 KB
COMPUTE_CLUSTER0_GIC_MESSAGE_BASED_SPIS	0x0001810000	0x0001820000	64 KB
COMPUTE_CLUSTER0_GIC_ITS	0x0001820000	0x0001830000	64 KB
COMPUTE_CLUSTER0_GIC_REDISTRIBUTOR_CONTROL_LPI_0	0x0001900000	0x0001910000	64 KB
COMPUTE_CLUSTER0_GIC_REDISTRIBUTOR_SGI_PPI_0	0x0001910000	0x0001920000	64 KB

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
COMPUTE_CLUSTER0_GIC_REDISTRIBUTOR_CONTROL_LPI_1	0x0001920000	0x0001930000	64 KB
COMPUTE_CLUSTER0_GIC_REDISTRIBUTOR_SGI_PPI_1	0x0001930000	0x0001940000	64 KB
COMPUTE_CLUSTER0_GIC_REDISTRIBUTOR_CONTROL_LPI_2	0x0001940000	0x0001950000	64 KB
COMPUTE_CLUSTER0_GIC_REDISTRIBUTOR_SGI_PPI_2	0x0001950000	0x0001960000	64 KB
COMPUTE_CLUSTER0_GIC_REDISTRIBUTOR_CONTROL_LPI_3	0x0001960000	0x0001970000	64 KB
COMPUTE_CLUSTER0_GIC_REDISTRIBUTOR_SGI_PPI_3	0x0001970000	0x0001980000	64 KB
COMPUTE_CLUSTER0_GIC_REDISTRIBUTOR_CONTROL_LPI_4	0x0001980000	0x0001990000	64 KB
COMPUTE_CLUSTER0_GIC_REDISTRIBUTOR_SGI_PPI_4	0x0001990000	0x00019A0000	64 KB
COMPUTE_CLUSTER0_GIC_REDISTRIBUTOR_CONTROL_LPI_5	0x00019A0000	0x00019B0000	64 KB
COMPUTE_CLUSTER0_GIC_REDISTRIBUTOR_SGI_PPI_5	0x00019B0000	0x00019C0000	64 KB
COMPUTE_CLUSTER0_GIC_REDISTRIBUTOR_CONTROL_LPI_6	0x00019C0000	0x00019D0000	64 KB
COMPUTE_CLUSTER0_GIC_REDISTRIBUTOR_SGI_PPI_6	0x00019D0000	0x00019E0000	64 KB
COMPUTE_CLUSTER0_GIC_REDISTRIBUTOR_CONTROL_LPI_7	0x00019E0000	0x00019F0000	64 KB
COMPUTE_CLUSTER0_GIC_REDISTRIBUTOR_SGI_PPI_7	0x00019F0000	0x0001A00000	64 KB
I2C0_CFG	0x0002000000	0x0002000100	256 B
I2C1_CFG	0x0002010000	0x0002010100	256 B
I2C2_CFG	0x0002020000	0x0002020100	256 B
I2C3_CFG	0x0002030000	0x0002030100	256 B
I2C4_CFG	0x0002040000	0x0002040100	256 B
I2C5_CFG	0x0002050000	0x0002050100	256 B
I2C6_CFG	0x0002060000	0x0002060100	256 B
MCSPI0_CFG	0x0002100000	0x0002100400	1 KB
MCSPI1_CFG	0x0002110000	0x0002110400	1 KB
MCSPI2_CFG	0x0002120000	0x0002120400	1 KB
MCSPI3_CFG	0x0002130000	0x0002130400	1 KB
MCSPI4_CFG	0x0002140000	0x0002140400	1 KB
MCSPI5_CFG	0x0002150000	0x0002150400	1 KB
MCSPI6_CFG	0x0002160000	0x0002160400	1 KB
MCSPI7_CFG	0x0002170000	0x0002170400	1 KB
RTI0_CFG	0x0002200000	0x0002200100	256 B
RTI1_CFG	0x0002210000	0x0002210100	256 B
RTI15_CFG	0x00022F0000	0x00022F0100	256 B
RTI16_CFG	0x0002300000	0x0002300100	256 B
RTI17_CFG	0x0002310000	0x0002310100	256 B
RTI28_CFG	0x00023C0000	0x00023C0100	256 B
RTI29_CFG	0x00023D0000	0x00023D0100	256 B
RTI30_CFG	0x00023E0000	0x00023E0100	256 B
RTI31_CFG	0x00023F0000	0x00023F0100	256 B
TIMER0_CFG	0x0002400000	0x0002400400	1 KB
TIMER1_CFG	0x0002410000	0x0002410400	1 KB
TIMER2_CFG	0x0002420000	0x0002420400	1 KB
TIMER3_CFG	0x0002430000	0x0002430400	1 KB
TIMER4_CFG	0x0002440000	0x0002440400	1 KB
TIMER5_CFG	0x0002450000	0x0002450400	1 KB
TIMER6_CFG	0x0002460000	0x0002460400	1 KB
TIMER7_CFG	0x0002470000	0x0002470400	1 KB
TIMER8_CFG	0x0002480000	0x0002480400	1 KB

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
TIMER9_CFG	0x0002490000	0x0002490400	1 KB
TIMER10_CFG	0x00024A0000	0x00024A0400	1 KB
TIMER11_CFG	0x00024B0000	0x00024B0400	1 KB
TIMER12_CFG	0x00024C0000	0x00024C0400	1 KB
TIMER13_CFG	0x00024D0000	0x00024D0400	1 KB
TIMER14_CFG	0x00024E0000	0x00024E0400	1 KB
TIMER15_CFG	0x00024F0000	0x00024F0400	1 KB
TIMER16_CFG	0x0002500000	0x0002500400	1 KB
TIMER17_CFG	0x0002510000	0x0002510400	1 KB
TIMER18_CFG	0x0002520000	0x0002520400	1 KB
TIMER19_CFG	0x0002530000	0x0002530400	1 KB
AM_HC2_TO_HC_CFG_STOG5_CFG	0x0002604000	0x0002604400	1 KB
AM_RC_TO_HC2_STOG7_CFG	0x0002606000	0x0002606400	1 KB
AM_RC_TO_RC_CFG_STOG3_CFG	0x0002608000	0x0002608400	1 KB
AM_IPPHY_TO_IPPHY_STOG1_CFG	0x000260A000	0x000260A400	1 KB
AM_RC_TO_HC2_STOG6_CFG	0x000260C000	0x000260C400	1 KB
AM_NAVSS_TO_AC_NON_SAFE_STOG4_CFG	0x0002610000	0x0002610400	1 KB
AM_AC_CFG_TO_AC_CFG_NON_SAFE_STOG2_CFG	0x0002612000	0x0002612400	1 KB
AM_AC_CFG_TO_AC_CFG_NON_SAFE_STOG9_CFG	0x0002614000	0x0002614400	1 KB
AM_IPPHY_TO_RTI_GPU_STOG8_CFG	0x0002616000	0x0002616400	1 KB
MCAN14_SS	0x0002680000	0x0002680100	256 B
MCAN14_CFG	0x0002681000	0x0002681200	512 B
MCAN14_MSGMEM_RAM	0x0002688000	0x0002690000	32 KB
MCAN15_SS	0x0002690000	0x0002690100	256 B
MCAN15_CFG	0x0002691000	0x0002691200	512 B
MCAN15_MSGMEM_RAM	0x0002698000	0x00026A0000	32 KB
MCAN16_SS	0x00026A0000	0x00026A0100	256 B
MCAN16_CFG	0x00026A1000	0x00026A1200	512 B
MCAN16_MSGMEM_RAM	0x00026A8000	0x00026B0000	32 KB
MCAN17_SS	0x00026B0000	0x00026B0100	256 B
MCAN17_CFG	0x00026B1000	0x00026B1200	512 B
MCAN17_MSGMEM_RAM	0x00026B8000	0x00026C0000	32 KB
MCAN0_SS	0x0002700000	0x0002700100	256 B
MCAN0_CFG	0x0002701000	0x0002701200	512 B
MCAN0_MSGMEM_RAM	0x0002708000	0x0002710000	32 KB
MCAN1_SS	0x0002710000	0x0002710100	256 B
MCAN1_CFG	0x0002711000	0x0002711200	512 B
MCAN1_MSGMEM_RAM	0x0002718000	0x0002720000	32 KB
MCAN2_SS	0x0002720000	0x0002720100	256 B
MCAN2_CFG	0x0002721000	0x0002721200	512 B
MCAN2_MSGMEM_RAM	0x0002728000	0x0002730000	32 KB
MCAN3_SS	0x0002730000	0x0002730100	256 B
MCAN3_CFG	0x0002731000	0x0002731200	512 B
MCAN3_MSGMEM_RAM	0x0002738000	0x0002740000	32 KB
MCAN4_SS	0x0002740000	0x0002740100	256 B
MCAN4_CFG	0x0002741000	0x0002741200	512 B
MCAN4_MSGMEM_RAM	0x0002748000	0x0002750000	32 KB

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
MCAN5_SS	0x0002750000	0x0002750100	256 B
MCAN5_CFG	0x0002751000	0x0002751200	512 B
MCAN5_MSGMEM_RAM	0x0002758000	0x0002760000	32 KB
MCAN6_SS	0x0002760000	0x0002760100	256 B
MCAN6_CFG	0x0002761000	0x0002761200	512 B
MCAN6_MSGMEM_RAM	0x0002768000	0x0002770000	32 KB
MCAN7_SS	0x0002770000	0x0002770100	256 B
MCAN7_CFG	0x0002771000	0x0002771200	512 B
MCAN7_MSGMEM_RAM	0x0002778000	0x0002780000	32 KB
MCAN8_SS	0x0002780000	0x0002780100	256 B
MCAN8_CFG	0x0002781000	0x0002781200	512 B
MCAN8_MSGMEM_RAM	0x0002788000	0x0002790000	32 KB
MCAN9_SS	0x0002790000	0x0002790100	256 B
MCAN9_CFG	0x0002791000	0x0002791200	512 B
MCAN9_MSGMEM_RAM	0x0002798000	0x00027A0000	32 KB
MCAN10_SS	0x00027A0000	0x00027A0100	256 B
MCAN10_CFG	0x00027A1000	0x00027A1200	512 B
MCAN10_MSGMEM_RAM	0x00027A8000	0x00027B0000	32 KB
MCAN11_SS	0x00027B0000	0x00027B0100	256 B
MCAN11_CFG	0x00027B1000	0x00027B1200	512 B
MCAN11_MSGMEM_RAM	0x00027B8000	0x00027C0000	32 KB
MCAN12_SS	0x00027C0000	0x00027C0100	256 B
MCAN12_CFG	0x00027C1000	0x00027C1200	512 B
MCAN12_MSGMEM_RAM	0x00027C8000	0x00027D0000	32 KB
MCAN13_SS	0x00027D0000	0x00027D0100	256 B
MCAN13_CFG	0x00027D1000	0x00027D1200	512 B
MCAN13_MSGMEM_RAM	0x00027D8000	0x00027E0000	32 KB
PDMA5_REGS	0x00027E0000	0x00027E0400	1 KB
PDMA6_REGS	0x00027E1000	0x00027E1400	1 KB
PDMA7_REGS	0x00027E2000	0x00027E2400	1 KB
UART0	0x0002800000	0x0002800200	512 B
UART1	0x0002810000	0x0002810200	512 B
UART2	0x0002820000	0x0002820200	512 B
UART3	0x0002830000	0x0002830200	512 B
UART4	0x0002840000	0x0002840200	512 B
UART5	0x0002850000	0x0002850200	512 B
UART6	0x0002860000	0x0002860200	512 B
UART7	0x0002870000	0x0002870200	512 B
UART8	0x0002880000	0x0002880200	512 B
UART9	0x0002890000	0x0002890200	512 B
PCIE1_CORE_PCIE_INTD_CFG_INTD_CFG	0x0002910000	0x0002911000	4 KB
PCIE1_CORE_VMAP_MMRS	0x0002914000	0x0002915000	4 KB
PCIE1_CORE_ECC_AGGR0	0x0002915000	0x0002915400	1 KB
PCIE1_CORE_CPTS_CFG_CPTS_VBUSP	0x0002916000	0x0002916400	1 KB
PCIE1_CORE_USER_CFG_USER_CFG	0x0002917000	0x0002917400	1 KB
VUSR_DUAL0_VUSR	0x0002960000	0x0002962000	8 KB
COMPUTE_CLUSTER0_DDR0_0_SS_CFG	0x0002980000	0x0002980200	512 B

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
COMPUTE_CLUSTER0_DDR0_0_CTL_CFG	0x0002990000	0x0002998000	32 KB
COMPUTE_CLUSTER0_DDR1_1_SS_CFG	0x00029A0000	0x00029A0200	512 B
COMPUTE_CLUSTER0_DDR1_1_CTL_CFG	0x00029B0000	0x00029B8000	32 KB
PCIE1_CORE_ECC_AGGR1	0x0002A02000	0x0002A02400	1 KB
USB0_RAMS_INJ_CFG	0x0002A10000	0x0002A10400	1 KB
USB0_ECC_AGGR	0x0002A13000	0x0002A13400	1 KB
CPSW1_ECC	0x0002A22000	0x0002A22400	1 KB
SA2_UL0_ECC_AGGR	0x0002A23000	0x0002A23400	1 KB
MMCSD0_ECC_AGGR_RXMEM	0x0002A24000	0x0002A24400	1 KB
MMCSD0_ECC_AGGR_TXMEM	0x0002A25000	0x0002A25400	1 KB
MMCSD1_ECC_AGGR_RXMEM	0x0002A26000	0x0002A26400	1 KB
MMCSD1_ECC_AGGR_TXMEM	0x0002A27000	0x0002A27400	1 KB
R5FSS0_EVNT_BUS_VBUSP_MMRS	0x0002A2D000	0x0002A2D100	256 B
R5FSS1_EVNT_BUS_VBUSP_MMRS	0x0002A2E000	0x0002A2E100	256 B
MSRAM_512K0_ECC_AGGR_REGS	0x0002A2F000	0x0002A2F400	1 KB
CSI_RX_IF0_ECC_AGGR_CFG	0x0002A30000	0x0002A30400	1 KB
CSI_RX_IF1_ECC_AGGR_CFG	0x0002A31000	0x0002A31400	1 KB
CSI_TX_IF_V2_0_ECC_AGGR_CFG	0x0002A38000	0x0002A38400	1 KB
CSI_TX_IF_V2_0_ECC_AGGR_BYTE_CFG	0x0002A38400	0x0002A38800	1 KB
CSI_TX_IF_V2_1_ECC_AGGR_CFG	0x0002A39000	0x0002A39400	1 KB
CSI_TX_IF_V2_1_ECC_AGGR_BYTE_CFG	0x0002A39400	0x0002A39800	1 KB
MCAN8_ECC_AGGR	0x0002A40000	0x0002A40400	1 KB
MCAN9_ECC_AGGR	0x0002A41000	0x0002A41400	1 KB
MCAN10_ECC_AGGR	0x0002A42000	0x0002A42400	1 KB
MCAN11_ECC_AGGR	0x0002A43000	0x0002A43400	1 KB
MCAN12_ECC_AGGR	0x0002A44000	0x0002A44400	1 KB
MCAN13_ECC_AGGR	0x0002A45000	0x0002A45400	1 KB
MCAN14_ECC_AGGR	0x0002A46000	0x0002A46400	1 KB
MCAN15_ECC_AGGR	0x0002A47000	0x0002A47400	1 KB
MCAN16_ECC_AGGR	0x0002A48000	0x0002A48400	1 KB
MCAN17_ECC_AGGR	0x0002A49000	0x0002A49400	1 KB
VPAC0_ECC_AGGR	0x0002A60000	0x0002A60400	1 KB
VPAC0_VISS_ECC_AGGR	0x0002A61000	0x0002A61400	1 KB
VPAC0_LDC_ECC_AGGR	0x0002A63000	0x0002A63400	1 KB
R5FSS0_CORE0_ECC_AGGR	0x0002A68000	0x0002A68400	1 KB
R5FSS1_CORE0_ECC_AGGR	0x0002A69000	0x0002A69400	1 KB
DMPAC0_ECC_AGGR	0x0002A6A000	0x0002A6A400	1 KB
MCAN0_ECC_AGGR	0x0002A78000	0x0002A78400	1 KB
MCAN1_ECC_AGGR	0x0002A79000	0x0002A79400	1 KB
MCAN2_ECC_AGGR	0x0002A7A000	0x0002A7A400	1 KB
MCAN3_ECC_AGGR	0x0002A7B000	0x0002A7B400	1 KB
MCAN4_ECC_AGGR	0x0002A7C000	0x0002A7C400	1 KB
MCAN5_ECC_AGGR	0x0002A7D000	0x0002A7D400	1 KB
MCAN6_ECC_AGGR	0x0002A7E000	0x0002A7E400	1 KB
MCAN7_ECC_AGGR	0x0002A7F000	0x0002A7F400	1 KB
CBASS_DEBUG0_ERR	0x0002A80000	0x0002A80400	1 KB
AEP_HC2_CBASS0_ERR	0x0002A83000	0x0002A83400	1 KB

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
CBASS_AC_CFG0_ERR	0x0002A84000	0x0002A84400	1 KB
CBASS_AC_NONSAFE0_ERR	0x0002A85000	0x0002A85400	1 KB
CBASS_DATADEBUG0_ERR	0x0002A86000	0x0002A86400	1 KB
CBASS_CSI0_ERR	0x0002A88000	0x0002A88400	1 KB
AEP_HC_CFG_CBASS0_ERR	0x0002A89000	0x0002A89400	1 KB
CBASS_RC0_ERR	0x0002A8C000	0x0002A8C400	1 KB
CBASS_RC_CFG0_ERR	0x0002A8D000	0x0002A8D400	1 KB
CBASS_IPPHY0_ERR	0x0002A8F000	0x0002A8F400	1 KB
AM_PULSAR0_MEM_CBASS0_ERR	0x0002A90000	0x0002A90400	1 KB
AM_PULSAR0_SLV_CBASS0_ERR	0x0002A91000	0x0002A91400	1 KB
AM_PULSAR1_MEM_CBASS0_ERR	0x0002A92000	0x0002A92400	1 KB
CBASS_IPPHY_SAFE0_ERR	0x0002A94000	0x0002A94400	1 KB
AM_PULSAR1_PERIPH_SWITCH_CBASS0_ERR	0x0002A95000	0x0002A95400	1 KB
CBASS_AC_CFG_NONSAFE0_ERR	0x0002A97000	0x0002A97400	1 KB
AM_AC_MERGER_CBASS0_ERR	0x0002A98000	0x0002A98400	1 KB
DSS_EDP0_MHDPTX_WRAPPER_ECC_AGGR_CORE_CFG	0x0002AC0000	0x0002AC0400	1 KB
DSS_EDP0_MHDPTX_WRAPPER_ECC_AGGR_PHY_CFG	0x0002AC1000	0x0002AC1400	1 KB
DSS_EDP0_MHDPTX_WRAPPER_ECC_AGGR_DSC_CFG	0x0002AC2000	0x0002AC2400	1 KB
IVC_DOM0_ECC_AGGR20_REGS	0x0002AE0000	0x0002AE0400	1 KB
IVC_DOM1_ECC_AGGR21_REGS	0x0002AE1000	0x0002AE1400	1 KB
IVC_DOM0_ECC_AGGR16_REGS	0x0002AF0000	0x0002AF0400	1 KB
IVC_DOM1_ECC_AGGR17_REGS	0x0002AF1000	0x0002AF1400	1 KB
IVC_DOM0_ECC_AGGR18_REGS	0x0002AF2000	0x0002AF2400	1 KB
IVC_DOM1_ECC_AGGR19_REGS	0x0002AF3000	0x0002AF3400	1 KB
ECC_AGGR4_ECC_AGGR	0x0002AF4000	0x0002AF4400	1 KB
ECC_AGGR5_ECC_AGGR	0x0002AF5000	0x0002AF5400	1 KB
MAIN_IP_ECC_AGGR0_ECC_AGGR	0x0002AF6000	0x0002AF6400	1 KB
ECC_AGGR6_ECC_AGGR	0x0002AF7000	0x0002AF7400	1 KB
ECC_AGGR9_ECC_AGGR	0x0002AF9000	0x0002AF9400	1 KB
ECC_AGGR10_ECC_AGGR	0x0002AFA000	0x0002AFA400	1 KB
ECC_AGGR11_ECC_AGGR	0x0002AFB000	0x0002AFB400	1 KB
MSRAM_512K1_ECC_AGGR_REGS	0x0002AFC000	0x0002AFC400	1 KB
VUSR_DUAL0_REGS	0x0002AFD000	0x0002AFD400	1 KB
MCASP0_CFG	0x0002B00000	0x0002B02000	8 KB
MCASP0_DMA	0x0002B08000	0x0002B08400	1 KB
MCASP1_CFG	0x0002B10000	0x0002B12000	8 KB
MCASP1_DMA	0x0002B18000	0x0002B18400	1 KB
MCASP2_CFG	0x0002B20000	0x0002B22000	8 KB
MCASP2_DMA	0x0002B28000	0x0002B28400	1 KB
MCASP3_CFG	0x0002B30000	0x0002B32000	8 KB
MCASP3_DMA	0x0002B38000	0x0002B38400	1 KB
MCASP4_CFG	0x0002B40000	0x0002B42000	8 KB
MCASP4_DMA	0x0002B48000	0x0002B48400	1 KB
EPWM0_EPWM	0x0003000000	0x0003000100	256 B
EPWM1_EPWM	0x0003010000	0x0003010100	256 B
EPWM2_EPWM	0x0003020000	0x0003020100	256 B
EPWM3_EPWM	0x0003030000	0x0003030100	256 B

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
EPWM4_EPWM	0x0003040000	0x0003040100	256 B
EPWM5_EPWM	0x0003050000	0x0003050100	256 B
ECAP0_CTL_STS	0x0003100000	0x0003100100	256 B
ECAP1_CTL_STS	0x0003110000	0x0003110100	256 B
ECAP2_CTL_STS	0x0003120000	0x0003120100	256 B
ATL0_REG	0x00031F0000	0x00031F0400	1 KB
EQEP0_REG	0x0003200000	0x0003200100	256 B
EQEP1_REG	0x0003210000	0x0003210100	256 B
EQEP2_REG	0x0003220000	0x0003220100	256 B
PBIST7	0x0003300000	0x0003300400	1 KB
PBIST8	0x0003310000	0x0003310400	1 KB
PBIST5	0x0003340000	0x0003340400	1 KB
PBIST11	0x0003350000	0x0003350400	1 KB
PBIST3	0x0003370000	0x0003370400	1 KB
PBIST2	0x0003380000	0x0003380400	1 KB
PBIST10	0x0003390000	0x0003390400	1 KB
AEP_GPU_BXS464_WRAP0_MEM	0x00033A0000	0x00033A0400	1 KB
MAIN_USART_PSILSS0_MMRS	0x0003400000	0x0003401000	4 KB
CPSW_PSILSS0_MMRS	0x0003404000	0x0003405000	4 KB
DEBUG_PSILSS0_MMRS	0x0003408000	0x0003409000	4 KB
CSI_PSILSS0_MMRS	0x0003410000	0x0003411000	4 KB
SA2_CPSW_PSILSS0_MMRS	0x0003414000	0x0003415000	4 KB
DMPAC_VPAC_PSILSS0_MMRS	0x000341C000	0x000341D000	4 KB
NAVSS0_NBSS_CFG_REGS0_MMRS	0x0003700000	0x0003700100	256 B
NAVSS0_NBSS_CFG_ECCAGGR0_REGS	0x0003701000	0x0003701400	1 KB
NAVSS0_NBSS_NB0_CFG_MMRS	0x0003702000	0x0003702100	256 B
NAVSS0_NBSS_NB1_CFG_MMRS	0x0003703000	0x0003703100	256 B
NAVSS0_NBSS_CFG_MSMC0_SLV_VIRTID_CFG_MMRS	0x0003710000	0x0003710100	256 B
VPAC0_VPAC_REGS_VPAC_REGS_CFG_IP_MMRS	0x0003800000	0x0003800400	1 KB
VPAC0_CTSET2_WRAP_CFG_CTSET2_CFG	0x0003802000	0x0003804000	8 KB
VPAC0_CP_INTD_CFG_INTD_CFG	0x0003804000	0x0003805000	4 KB
VPAC0_HTS_S_VBUSP	0x0003810000	0x0003820000	64 KB
VPAC0_PAR_VPAC_LDC0_S_VBUSP_MMR_VBUSP	0x0003820000	0x0003820400	1 KB
VPAC0_PAR_VPAC_LDC0_S_VBUSP_VPAC_LDC_LSE_CFG_VP	0x0003820400	0x0003820600	512 B
VPAC0_PAR_VPAC_LDC0_S_VBUSP_PIXWRINTF_DUALY_LUTCFG_DUALY_LUT	0x0003820800	0x0003821000	2 KB
VPAC0_PAR_VPAC_LDC0_S_VBUSP_PIXWRINTF_DUALC_LUTCFG_DUALC_LUT	0x0003821000	0x0003821800	2 KB
VPAC0_PAR_VPAC_LDC0_S_VBUSP_MEMCFG_LOOP_MESH_VBUSPI_MESH_MEM	0x0003822000	0x0003824000	8 KB
VPAC0_PAR_VPAC_LDC0_S_VBUSP_MEMCFG_LOOP_Y_VBUSPI_Y_MEM	0x0003828000	0x0003830000	32 KB
VPAC0_PAR_VPAC_LDC0_S_VBUSP_MEMCFG_LOOP_CBCR_VBUSPI_CBCR_MEM	0x0003830000	0x0003838000	32 KB
VPAC0_PAR_VPAC_MSC_CFG_VP_CFG_VP	0x00038C0000	0x00038C0800	2 KB
VPAC0_PAR_VPAC_MSC_CFG_VP_LSE_CFG_VP	0x00038C0800	0x00038C0A00	512 B
VPAC0_PAR_VPAC_NF_S_VBUSP_MMR_VBUSP_NF_CFG	0x00038C2000	0x00038C3000	4 KB
VPAC0_PAR_VPAC_NF_S_VBUSP_VPAC_NF_LSE_CFG_VP	0x00038C3000	0x00038C3200	512 B

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
VPAC0_PAR_VPAC_VISS0_S_VBUSP_MMR_CFG_VISS_TOP	0x0003900000	0x0003900200	512 B
VPAC0_PAR_VPAC_VISS0_S_VBUSP_VPAC_VISS_LSE_CFG_VP	0x0003900400	0x0003900600	512 B
VPAC0_PAR_VPAC_VISS0_S_VBUSP_K3_GLBCE_TOP_CFG_GLBCE	0x0003903800	0x0003904000	2 KB
VPAC0_PAR_VPAC_VISS0_S_VBUSP_K3_GLBCE_TOP_STATMEM_CFG_GLBCE_STATMEM	0x0003904000	0x0003908000	16 KB
VPAC0_PAR_VPAC_VISS0_S_VBUSP_VISS_FCP_CFA_VBUSP_FLEXCFA	0x0003908000	0x0003910000	32 KB
VPAC0_PAR_VPAC_VISS0_S_VBUSP_VISS_FCP_FCC_VBUSP_FLEXCC	0x0003910000	0x0003910800	2 KB
VPAC0_PAR_VPAC_VISS0_S_VBUSP_VISS_FCP_FCC_VBUSP_FLEXCC_CONTRASTC1	0x0003910800	0x0003911000	2 KB
VPAC0_PAR_VPAC_VISS0_S_VBUSP_VISS_FCP_FCC_VBUSP_FLEXCC_CONTRASTC2	0x0003911000	0x0003911800	2 KB
VPAC0_PAR_VPAC_VISS0_S_VBUSP_VISS_FCP_FCC_VBUSP_FLEXCC_CONTRASTC3	0x0003911800	0x0003912000	2 KB
VPAC0_PAR_VPAC_VISS0_S_VBUSP_VISS_FCP_FCC_VBUSP_FLEXCC_Y8R8	0x0003912000	0x0003912800	2 KB
VPAC0_PAR_VPAC_VISS0_S_VBUSP_VISS_FCP_FCC_VBUSP_FLEXCC_C8G8	0x0003912800	0x0003913000	2 KB
VPAC0_PAR_VPAC_VISS0_S_VBUSP_VISS_FCP_FCC_VBUSP_FLEXCC_S8B8	0x0003913000	0x0003913800	2 KB
VPAC0_PAR_VPAC_VISS0_S_VBUSP_VISS_FCP_FCC_VBUSP_FLEXCC_HIST	0x0003913800	0x0003914000	2 KB
VPAC0_PAR_VPAC_VISS0_S_VBUSP_VISS_FCP_FCC_VBUSP_FLEXCC_LINE	0x0003918000	0x0003918800	2 KB
VPAC0_PAR_VPAC_VISS0_S_VBUSP_VISS_RAWFE_CFG_MMR_S_VBUSP_RAWFE_CFG	0x0003920000	0x0003920400	1 KB
VPAC0_PAR_VPAC_VISS0_S_VBUSP_VISS_RAWFE_CFG_H3A_WRAP_CFG_RAWFE_H3A_CFG	0x0003920400	0x0003920500	256 B
VPAC0_PAR_VPAC_VISS0_S_VBUSP_VISS_RAWFE_CFG_LUT3_RAM_RAWFE_PWL_LUT3_RAM	0x0003920800	0x0003921000	2 KB
VPAC0_PAR_VPAC_VISS0_S_VBUSP_VISS_RAWFE_CFG_LUT2_RAM_RAWFE_PWL_LUT2_RAM	0x0003921000	0x0003921800	2 KB
VPAC0_PAR_VPAC_VISS0_S_VBUSP_VISS_RAWFE_CFG_LUT1_RAM_RAWFE_PWL_LUT1_RAM	0x0003921800	0x0003922000	2 KB
VPAC0_PAR_VPAC_VISS0_S_VBUSP_VISS_RAWFE_CFG_WDR_LUT_RAM_RAWFE_WDR_LUT_RAM	0x0003922000	0x0003922800	2 KB
VPAC0_PAR_VPAC_VISS0_S_VBUSP_VISS_RAWFE_CFG_H3A_LUT_RAM_RAWFE_H3A_LUT_RAM	0x0003922800	0x0003923000	2 KB
VPAC0_PAR_VPAC_VISS0_S_VBUSP_VISS_RAWFE_CFG_DPC_RAM_RAWFE_DPC_LUT_RAM	0x0003923000	0x0003923400	1 KB
VPAC0_PAR_VPAC_VISS0_S_VBUSP_VISS_RAWFE_CFG_DPC_LRAM_RAWFE_DPC_LRAM	0x0003924000	0x0003926000	8 KB
VPAC0_PAR_VPAC_VISS0_S_VBUSP_VISS_RAWFE_CFG_LSC_RAM_RAWFE_LSC_LUT_RAM	0x0003928000	0x0003930000	32 KB
VPAC0_PAR_VPAC_VISS0_S_VBUSP_VISS_RAWFE_CFG_H3A_WRAP_ARAM_RAWFE_H3A_ARAM	0x0003930000	0x0003932000	8 KB
VPAC0_PAR_VPAC_VISS0_S_VBUSP_VISS_RAWFE_CFG_H3A_WRAP_LRAM_RAWFE_H3A_LRAM	0x0003932000	0x0003934000	8 KB
VPAC0_PAR_VPAC_VISS0_S_VBUSP_VISS_NSF4V_CFG_MMR_VBUSP_NSF4VCORE	0x0003940000	0x0003940800	2 KB
VPAC0_PAR_VPAC_VISS0_S_VBUSP_VISS_NSF4V_CFG_RAWHIST_HISTDATA_VBUSP_RAWHIST	0x0003940800	0x0003940A00	512 B
VPAC0_PAR_VPAC_VISS0_S_VBUSP_VISS_NSF4V_CFG_RAWHIST_HISTLUT_VBUSP_RAWHIST_LUT	0x0003941000	0x0003941800	2 KB

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
VPAC0_PAR_VPAC_VISS0_S_VBUSP_VISS_NSF4V_CFG_MEM_MMRRAM_VBUSP_MMR_RAM	0x0003944000	0x0003948000	16 KB
VPAC0_PAR_VPAC_VISS0_S_VBUSP_VISS_FCP_EE_VBUSP_FLEXEE	0x0003950000	0x0003958000	32 KB
VPAC0_PAR_VPAC_VISS0_S_VBUSP_VISS_FCP_CFA_VBUSP_FLEXCFA_DLUTS	0x0003958000	0x000395C000	16 KB
VPAC0_PAR_VPAC_VISS0_S_VBUSP_VISS_FCP2_CFA_VBUSP_FLEXCFA_DLUTS	0x000395C000	0x0003960000	16 KB
VPAC0_PAR_VPAC_VISS0_S_VBUSP_VISS_FCP2_EE_VBUSP_FLEXEE	0x0003960000	0x0003968000	32 KB
VPAC0_PAR_VPAC_VISS0_S_VBUSP_VISS_FCP2_CFA_VBUSP_FLEXCFA	0x0003968000	0x0003970000	32 KB
VPAC0_PAR_VPAC_VISS0_S_VBUSP_VISS_FCP2_FCC_VBUSP_FLEXCC	0x0003970000	0x0003970800	2 KB
VPAC0_PAR_VPAC_VISS0_S_VBUSP_VISS_FCP2_FCC_VBUSP_FLEXCC_CONTRASTC1	0x0003970800	0x0003971000	2 KB
VPAC0_PAR_VPAC_VISS0_S_VBUSP_VISS_FCP2_FCC_VBUSP_FLEXCC_CONTRASTC2	0x0003971000	0x0003971800	2 KB
VPAC0_PAR_VPAC_VISS0_S_VBUSP_VISS_FCP2_FCC_VBUSP_FLEXCC_CONTRASTC3	0x0003971800	0x0003972000	2 KB
VPAC0_PAR_VPAC_VISS0_S_VBUSP_VISS_FCP2_FCC_VBUSP_FLEXCC_Y8R8	0x0003972000	0x0003972800	2 KB
VPAC0_PAR_VPAC_VISS0_S_VBUSP_VISS_FCP2_FCC_VBUSP_FLEXCC_C8G8	0x0003972800	0x0003973000	2 KB
VPAC0_PAR_VPAC_VISS0_S_VBUSP_VISS_FCP2_FCC_VBUSP_FLEXCC_S8B8	0x0003973000	0x0003973800	2 KB
VPAC0_PAR_VPAC_VISS0_S_VBUSP_VISS_FCP2_FCC_VBUSP_FLEXCC_HIST	0x0003973800	0x0003974000	2 KB
VPAC0_PAR_VPAC_VISS0_S_VBUSP_VISS_FCP2_FCC_VBUSP_FLEXCC_LINE	0x0003978000	0x0003978800	2 KB
VPAC0_PAR_VPAC_VISS0_S_VBUSP_VISS_CAC_S_VBUSP_MMRCFG_CAC	0x0003980000	0x0003980400	1 KB
VPAC0_PAR_VPAC_VISS0_S_VBUSP_VISS_CAC_S_VBUSP_CORE_LUT_CFG_LUT_MEM	0x0003982000	0x0003984000	8 KB
VPAC0_PAR_VPAC_VISS0_S_VBUSP_VISS_CAC_S_VBUSP_LINEMEM_CFG_LINE_MEM	0x0003984000	0x0003988000	16 KB
VPAC0_DRU_UTC_VPAC0_DRU_MMR_CFG_DRU_DRU	0x0003A00000	0x0003A04000	16 KB
VPAC0_DRU_UTC_VPAC0_DRU_MMR_CFG_DRU_DRU_SET	0x0003A04000	0x0003A08000	16 KB
VPAC0_DRU_UTC_VPAC0_DRU_MMR_CFG_DRU_DRU_QUEUE	0x0003A08000	0x0003A10000	32 KB
VPAC0_DRU_UTC_VPAC0_DRU_MMR_CFG_DRU_DRU_CHNRT	0x0003A40000	0x0003A60000	128 KB
VPAC0_DRU_UTC_VPAC0_DRU_MMR_CFG_DRU_DRU_CHRT	0x0003A60000	0x0003A80000	128 KB
VPAC0_DRU_UTC_VPAC0_DRU_MMR_CFG_DRU_DRU_CHATOMIC_DEBUG	0x0003A80000	0x0003AA0000	128 KB
VPAC0_DRU_UTC_VPAC0_DRU_MMR_CFG_DRU_DRU_CAUSE	0x0003AE0000	0x0003B00000	128 KB
VPAC0_DRU_UTC_VPAC1_DRU_MMR_CFG_DRU_DRU	0x0003B00000	0x0003B04000	16 KB
VPAC0_DRU_UTC_VPAC1_DRU_MMR_CFG_DRU_DRU_SET	0x0003B04000	0x0003B08000	16 KB
VPAC0_DRU_UTC_VPAC1_DRU_MMR_CFG_DRU_DRU_QUEUE	0x0003B08000	0x0003B10000	32 KB
VPAC0_DRU_UTC_VPAC1_DRU_MMR_CFG_DRU_DRU_CHNRT	0x0003B40000	0x0003B60000	128 KB
VPAC0_DRU_UTC_VPAC1_DRU_MMR_CFG_DRU_DRU_CHRT	0x0003B60000	0x0003B80000	128 KB
VPAC0_DRU_UTC_VPAC1_DRU_MMR_CFG_DRU_DRU_CHATOMIC_DEBUG	0x0003B80000	0x0003BA0000	128 KB
VPAC0_DRU_UTC_VPAC1_DRU_MMR_CFG_DRU_DRU_CAUSE	0x0003BE0000	0x0003C00000	128 KB
USB0_MMR_MMRVBP_USBSS_CMN	0x0004104000	0x0004104100	256 B
USB0_PHY2	0x0004108000	0x0004108400	1 KB

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
CODEC0_VPU	0x0004210000	0x0004220000	64 KB
CSI_TX_IF_V2_0_TX_SHIM_VBUSP_MMR_CSI2TXIF_V2	0x0004400000	0x0004401000	4 KB
CSI_TX_IF_V2_0_VBUS2APB_WRAP_VBUSP_APB_CSI2TX_V2	0x0004404000	0x0004405000	4 KB
CSI_TX_IF_V2_0_CP_INTD_CFG_INTD_CFG	0x0004408000	0x0004409000	4 KB
CSI_TX_IF_V2_1_TX_SHIM_VBUSP_MMR_CSI2TXIF_V2	0x0004410000	0x0004411000	4 KB
CSI_TX_IF_V2_1_VBUS2APB_WRAP_VBUSP_APB_CSI2TX_V2	0x0004414000	0x0004415000	4 KB
CSI_TX_IF_V2_1_CP_INTD_CFG_INTD_CFG	0x0004418000	0x0004419000	4 KB
DPHY_TX0	0x0004480000	0x0004481000	4 KB
DPHY_TX1	0x0004481000	0x0004482000	4 KB
CSI_RX_IF0_RX_SHIM_VBUSP_MMR_CSI2RXIF	0x0004500000	0x0004501000	4 KB
CSI_RX_IF0_VBUS2APB_WRAP_VBUSP_APB_CSI2RX	0x0004504000	0x0004505000	4 KB
CSI_RX_IF0_CP_INTD_CFG_INTD_CFG	0x0004508000	0x0004509000	4 KB
CSI_RX_IF1_RX_SHIM_VBUSP_MMR_CSI2RXIF	0x0004510000	0x0004511000	4 KB
CSI_RX_IF1_VBUS2APB_WRAP_VBUSP_APB_CSI2RX	0x0004514000	0x0004515000	4 KB
CSI_RX_IF1_CP_INTD_CFG_INTD_CFG	0x0004518000	0x0004519000	4 KB
DPHY_RX0_VBUS2APB_WRAP_VBUSP_K3_DPHY_RX	0x0004580000	0x0004581000	4 KB
DPHY_RX0_MMR_SLV_K3_DPHY_WRAP	0x0004581000	0x0004581100	256 B
DPHY_RX1_VBUS2APB_WRAP_VBUSP_K3_DPHY_RX	0x0004590000	0x0004591000	4 KB
DPHY_RX1_MMR_SLV_K3_DPHY_WRAP	0x0004591000	0x0004591100	256 B
DSS_DSI0_DSI_TOP_ECC_AGGR_SYS_CFG	0x0004700000	0x0004700400	1 KB
DSS_DSI1_DSI_TOP_ECC_AGGR_SYS_CFG	0x0004701000	0x0004701400	1 KB
DSS_DSI0_DSI_WRAP_MMR_VBUSP_CFG_DSI_WRAP	0x0004710000	0x0004710100	256 B
DSS_DSI1_DSI_WRAP_MMR_VBUSP_CFG_DSI_WRAP	0x0004720000	0x0004720100	256 B
DSS_DSI0_DSI_TOP_VBUSP_CFG_DSI_0_DSI	0x0004800000	0x0004900000	1 MB
DSS_DSI1_DSI_TOP_VBUSP_CFG_DSI_0_DSI	0x0004900000	0x0004A00000	1 MB
DSS0_DISPC_0_COMMON_M	0x0004A00000	0x0004A10000	64 KB
DSS0_DISPC_0_COMMON_S0	0x0004A10000	0x0004A20000	64 KB
DSS0_VIDL1	0x0004A20000	0x0004A30000	64 KB
DSS0_VIDL2	0x0004A30000	0x0004A40000	64 KB
DSS0_VID1	0x0004A50000	0x0004A60000	64 KB
DSS0_VID2	0x0004A60000	0x0004A70000	64 KB
DSS0_OVR1	0x0004A70000	0x0004A80000	64 KB
DSS0_VP1	0x0004A80000	0x0004A90000	64 KB
DSS0_OVR2	0x0004A90000	0x0004AA0000	64 KB
DSS0_VP2	0x0004AA0000	0x0004AB0000	64 KB
DSS0_OVR3	0x0004AB0000	0x0004AC0000	64 KB
DSS0_VP3	0x0004AC0000	0x0004AD0000	64 KB
DSS0_OVR4	0x0004AD0000	0x0004AE0000	64 KB
DSS0_VP4	0x0004AE0000	0x0004AF0000	64 KB
DSS0_WB	0x0004AF0000	0x0004B00000	64 KB
DSS0_DISPC_0_COMMON_S1	0x0004B00000	0x0004B10000	64 KB
DSS0_DISPC_0_COMMON_S2	0x0004B10000	0x0004B20000	64 KB
SA2_UL0	0x0004E00000	0x0004E01000	4 KB
SA2_UL0_MMRA	0x0004E01000	0x0004E01200	512 B
SA2_UL0_EIP_76	0x0004E10000	0x0004E10080	128 B
SA2_UL0_EIP_29T2	0x0004E20000	0x0004E30000	64 KB
DSS_EDP0_INTG_CFG_VP	0x0004F40000	0x0004F40100	256 B

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
DSS_EDP0_V2A_S_CORE_VP_REGS_SAPB	0x0004F48000	0x0004F48100	256 B
MMCSD0_CTL_CFG	0x0004F80000	0x0004F81000	4 KB
MMCSD0_SS_CFG	0x0004F88000	0x0004F88400	1 KB
MMCSD1_CTL_CFG	0x0004FB0000	0x0004FB1000	4 KB
MMCSD1_SS_CFG	0x0004FB8000	0x0004FB8400	1 KB
SERDES_10G0	0x0005060000	0x0005070000	64 KB
ELM0	0x0005380000	0x0005381000	4 KB
GPMC0_CFG	0x0005390000	0x0005390400	1 KB
EFUSE0	0x00053F0000	0x00053F0100	256 B
R5FSS0_COMPARE_CFG	0x0005B00000	0x0005B00100	256 B
R5FSS0_CORE1_ECC_AGGR	0x0005B10000	0x0005B10400	1 KB
R5FSS1_COMPARE_CFG	0x0005B20000	0x0005B20100	256 B
R5FSS1_CORE1_ECC_AGGR	0x0005B30000	0x0005B30400	1 KB
R5FSS0_CORE0_ATCM	0x0005C00000	0x0005C10000	64 KB
R5FSS0_CORE0_BTCM	0x0005C10000	0x0005C20000	64 KB
R5FSS0_CORE1_ATCM	0x0005D00000	0x0005D08000	32 KB
R5FSS0_CORE1_BTCM	0x0005D10000	0x0005D18000	32 KB
R5FSS1_CORE0_ATCM	0x0005E00000	0x0005E10000	64 KB
R5FSS1_CORE0_BTCM	0x0005E10000	0x0005E20000	64 KB
R5FSS1_CORE1_ATCM	0x0005F00000	0x0005F08000	32 KB
R5FSS1_CORE1_BTCM	0x0005F10000	0x0005F18000	32 KB
USB0_VBP2APB_WRAP_CONTROLLER_VBP_CORE_ADDR_MAP	0x0006000000	0x0006400000	4 MB
DEBUGSS0_SYS	0x0008000000	0x0008001000	4 KB
CCDEBUGSS1_SYS	0x0008004000	0x0008005000	4 KB
CCDEBUGSS0_SYS	0x0008008000	0x0008009000	4 KB
STM0_STIMULUS	0x0009000000	0x000A000000	16 MB
DSS_EDP0_V2A_CORE_VP_REGS_APB	0x000A000000	0x000A040000	256 KB
CPSW1_NUSS	0x000C200000	0x000C400000	2 MB
PCIE1_CORE_DBN_CFG_PCIE_CORE	0x000D800000	0x000E000000	8 MB
DMPAC0_DMPAC_REGS_DMPAC_REGS_CFG_IP_MMRS	0x000F400000	0x000F400400	1 KB
DMPAC0_CP_INTD_CFG_INTD_CFG	0x000F401000	0x000F402000	4 KB
DMPAC0_HTS_S_VBUSP	0x000F408000	0x000F410000	32 KB
DMPAC0_CTSET2_WRAP_CFG_CTSET2_CFG	0x000F420000	0x000F422000	8 KB
DMPAC0_DMPAC_FOCO_0_CFG_SLV_DMPAC_FOCO_CORE_FOCO_REGS_CFG_IP_MMRS	0x000F424000	0x000F424040	64 B
DMPAC0_DMPAC_FOCO_0_CFG_SLV_VPAC_FOCO_LSE_CFG_VP	0x000F424200	0x000F424400	512 B
DMPAC0_DMPAC_FOCO_1_CFG_SLV_DMPAC_FOCO_CORE_FOCO_REGS_CFG_IP_MMRS	0x000F428000	0x000F428040	64 B
DMPAC0_DMPAC_FOCO_1_CFG_SLV_VPAC_FOCO_LSE_CFG_VP	0x000F428200	0x000F428400	512 B
DMPAC0_PAR_DOF_CFG_VP_MMR_VBUSP_DOFCORE	0x000F480000	0x000F481000	4 KB
DMPAC0_PAR_DOF_CFG_VP_MEM_MMRRAM_VBUSP_MMR_RAM	0x000F4C0000	0x000F500000	256 KB
DMPAC0_PAR_PAR_SDE_S_VBUSP_MMR_VBUSP_MMR	0x000F500000	0x000F501000	4 KB
DMPAC0_PAR_PAR_SDE_S_VBUSP_MEM_MMRRAM_VBUSP_MMR_RAM	0x000F540000	0x000F580000	256 KB
DMPAC0_DRU_UTC_DMPAC0_DRU_MMR_CFG_DRU_DRU	0x000F600000	0x000F604000	16 KB
DMPAC0_DRU_UTC_DMPAC0_DRU_MMR_CFG_DRU_DRU_SET	0x000F604000	0x000F608000	16 KB
DMPAC0_DRU_UTC_DMPAC0_DRU_MMR_CFG_DRU_DRU_QUEUE	0x000F608000	0x000F610000	32 KB
DMPAC0_DRU_UTC_DMPAC0_DRU_MMR_CFG_DRU_DRU_CHNRT	0x000F640000	0x000F660000	128 KB

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
DMPAC0_DRU_UTC_DMPAC0_DRU_MMR_CFG_DRU_DRU_CHRT	0x000F660000	0x000F680000	128 KB
DMPAC0_DRU_UTC_DMPAC0_DRU_MMR_CFG_DRU_DRU_CHATOMIC_DEBUG	0x000F680000	0x000F6A0000	128 KB
DMPAC0_DRU_UTC_DMPAC0_DRU_MMR_CFG_DRU_DRU_CAUSE	0x000F6E0000	0x000F700000	128 KB
PCIE1_DAT0	0x0018000000	0x0020000000	128 MB
GPMC0_DATA	0x0020000000	0x0028000000	128 MB
NAVSS0_MSRAM0_SLV_RAM	0x0030000000	0x0030010000	64 KB
NAVSS0_MODSS_INTA0_CFG	0x0030800000	0x0030800020	32 B
NAVSS0_MODSS_INTA1_CFG	0x0030801000	0x0030801020	32 B
NAVSS0_UDMASS_INTA0_CFG	0x0030802000	0x0030802020	32 B
NAVSS0_UDMASS_INTA0_CFG_UNMAP	0x0030880000	0x0030890000	64 KB
NAVSS0_MODSS_INTA0_CFG_IMAP	0x0030900000	0x0030902000	8 KB
NAVSS0_MODSS_INTA1_CFG_IMAP	0x0030908000	0x003090A000	8 KB
NAVSS0_UDMASS_INTA0_IMAP	0x0030940000	0x0030950000	64 KB
NAVSS0_NAV_DDR0_VIRTID_CFG_MMRS	0x0030A02000	0x0030A02100	256 B
NAVSS0_NAV_DDR1_VIRTID_CFG_MMRS	0x0030A03000	0x0030A03100	256 B
NAVSS0_UDMASS_UDMAP0_CFG_TCHAN	0x0030B00000	0x0030B20000	128 KB
NAVSS0_UDMASS_UDMAP0_CFG_RCHAN	0x0030C00000	0x0030C08000	32 KB
NAVSS0_UDMASS_UDMAP0_CFG_RFLOW	0x0030D00000	0x0030D04000	16 KB
NAVSS0_SPINLOCK	0x0030E00000	0x0030E08000	32 KB
NAVSS0_TIMERMGR0_CFG_CONFIG	0x0030E80000	0x0030E80200	512 B
NAVSS0_TIMERMGR1_CFG_CONFIG	0x0030E81000	0x0030E81200	512 B
NAVSS0_TIMERMGR0_CFG_OES	0x0030F00000	0x0030F01000	4 KB
NAVSS0_TIMERMGR1_CFG_OES	0x0030F01000	0x0030F02000	4 KB
NAVSS0_IO_PVU0_CFG_MMRS	0x0030F80000	0x0030F81000	4 KB
NAVSS0_IO_PVU1_CFG_MMRS	0x0030F81000	0x0030F82000	4 KB
NAVSS0_PVU0_SRC_TOG_CFG	0x0030F90000	0x0030F90400	1 KB
NAVSS0_PVU0_CFG_TOG_CFG	0x0030F91000	0x0030F91400	1 KB
NAVSS0_ECCAGGR0_REGS	0x0031000000	0x0031000400	1 KB
NAVSS0_UDMASS_ECCAGGR0_CFG_REGS	0x0031001000	0x0031001400	1 KB
NAVSS0_VIRTSS_ECCAGGR_CFG	0x0031002000	0x0031002400	1 KB
NAVSS0_UDMASS_INTA0_CFG_GCNTCFG	0x0031040000	0x0031044000	16 KB
NAVSS0_UDMASS_RINGACC0_CFG	0x0031080000	0x00310C0000	256 KB
NAVSS0_CFG	0x00310C0000	0x00310C0100	256 B
NAVSS0_CPTS	0x00310D0000	0x00310D0400	1 KB
NAVSS0_INTR0_INTR_ROUTER_CFG	0x00310E0000	0x00310E4000	16 KB
NAVSS0_UDMASS_INTA0_CFG_L2G	0x0031100000	0x0031102000	8 KB
NAVSS0_UDMASS_INTA0_CFG_MCAST	0x0031110000	0x0031114000	16 KB
NAVSS0_PROXY0_CFG_BUF_CFG	0x0031120000	0x0031120100	256 B
NAVSS0_PROXY_BUF	0x0031130000	0x0031134000	16 KB
NAVSS0_SEC_PROXY0_CFG_MMRS	0x0031140000	0x0031140100	256 B
NAVSS0_UDMASS_UDMAP0_CFG	0x0031150000	0x0031150100	256 B
NAVSS0_UDMASS_RINGACC0_GCFG	0x0031160000	0x0031160400	1 KB
NAVSS0_PSILSS0_CFG_MMRS	0x0031170000	0x0031171000	4 KB
NAVSS0_BCDMA0_CFG_GCFG	0x00311A0000	0x00311A0100	256 B
NAVSS0_MCRC	0x0031F70000	0x0031F71000	4 KB
NAVSS0_UDMASS_PSILCFG0_CFG_PROXY	0x0031F78000	0x0031F78200	512 B

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
NAVSS0_MAILBOX_REGS0	0x0031F80000	0x0031F80200	512 B
NAVSS0_MAILBOX_REGS1	0x0031F81000	0x0031F81200	512 B
NAVSS0_MAILBOX_REGS2	0x0031F82000	0x0031F82200	512 B
NAVSS0_MAILBOX_REGS3	0x0031F83000	0x0031F83200	512 B
NAVSS0_MAILBOX_REGS4	0x0031F84000	0x0031F84200	512 B
NAVSS0_MAILBOX_REGS5	0x0031F85000	0x0031F85200	512 B
NAVSS0_MAILBOX_REGS6	0x0031F86000	0x0031F86200	512 B
NAVSS0_MAILBOX_REGS7	0x0031F87000	0x0031F87200	512 B
NAVSS0_MAILBOX_REGS8	0x0031F88000	0x0031F88200	512 B
NAVSS0_MAILBOX_REGS9	0x0031F89000	0x0031F89200	512 B
NAVSS0_MAILBOX_REGS10	0x0031F8A000	0x0031F8A200	512 B
NAVSS0_MAILBOX_REGS11	0x0031F8B000	0x0031F8B200	512 B
NAVSS0_MAILBOX1_REGS0	0x0031F90000	0x0031F90200	512 B
NAVSS0_MAILBOX1_REGS1	0x0031F91000	0x0031F91200	512 B
NAVSS0_MAILBOX1_REGS2	0x0031F92000	0x0031F92200	512 B
NAVSS0_MAILBOX1_REGS3	0x0031F93000	0x0031F93200	512 B
NAVSS0_MAILBOX1_REGS4	0x0031F94000	0x0031F94200	512 B
NAVSS0_MAILBOX1_REGS5	0x0031F95000	0x0031F95200	512 B
NAVSS0_MAILBOX1_REGS6	0x0031F96000	0x0031F96200	512 B
NAVSS0_MAILBOX1_REGS7	0x0031F97000	0x0031F97200	512 B
NAVSS0_MAILBOX1_REGS8	0x0031F98000	0x0031F98200	512 B
NAVSS0_MAILBOX1_REGS9	0x0031F99000	0x0031F99200	512 B
NAVSS0_MAILBOX1_REGS10	0x0031F9A000	0x0031F9A200	512 B
NAVSS0_MAILBOX1_REGS11	0x0031F9B000	0x0031F9B200	512 B
NAVSS0_UDMASS_RINGACC0_CFG_MON	0x0032000000	0x0032020000	128 KB
NAVSS0_TIMERMGR0_CFG_TIMERS	0x0032200000	0x0032240000	256 KB
NAVSS0_TIMERMGR1_CFG_TIMERS	0x0032240000	0x0032280000	256 KB
NAVSS0_SEC_PROXY0_CFG_RT	0x0032400000	0x0032600000	2 MB
NAVSS0_SEC_PROXY0_CFG_SCFG	0x0032800000	0x0032A00000	2 MB
NAVSS0_SEC_PROXY0_SRC_TARGET_DATA	0x0032C00000	0x0032E00000	2 MB
NAVSS0_PROXY_TARGET0_DATA	0x0033000000	0x0033040000	256 KB
NAVSS0_PROXY0_BUF_CFG	0x0033400000	0x0033440000	256 KB
NAVSS0_UDMASS_INTA0_CFG_GCNTRTI	0x0033800000	0x0033A00000	2 MB
NAVSS0_MODSS_INTA0_CFG_INTR	0x0033C00000	0x0033C40000	256 KB
NAVSS0_MODSS_INTA1_CFG_INTR	0x0033C40000	0x0033C80000	256 KB
NAVSS0_UDMASS_INTA0_CFG_INTR	0x0033D00000	0x0033E00000	1 MB
NAVSS0_UDMASS_UDMAP0_CFG_RCHANRT	0x0034000000	0x0034080000	512 KB
NAVSS0_UDMASS_UDMAP0_CFG_TCHANRT	0x0035000000	0x0035200000	2 MB
NAVSS0_BCDMA0_CFG_TCHAN	0x0035840000	0x0035841000	4 KB
NAVSS0_BCDMA0_CFG_RCHAN	0x0035880000	0x0035882000	8 KB
NAVSS0_BCDMA0_CFG_RING	0x0035900000	0x0035904000	16 KB
NAVSS0_BCDMA0_CFG_TCHANRT	0x0035C00000	0x0035C10000	64 KB
NAVSS0_BCDMA0_CFG_RCHANRT	0x0035D00000	0x0035D20000	128 KB
NAVSS0_BCDMA0_CFG_RINGRT	0x0035E00000	0x0035E80000	512 KB
NAVSS0_IO_PVU0_CFG_TLBIF_TLB	0x0036000000	0x0036040000	256 KB
NAVSS0_IO_PVU1_CFG_TLBIF_TLB	0x0036040000	0x0036080000	256 KB
NAVSS0_UDMASS_RINGACC0_SRC_FIFOS	0x0038000000	0x0038400000	4 MB

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
NAVSS0_UDMASS_RINGACC0_CFG_RT	0x003C000000	0x003C400000	4 MB
MCU_MCAN0_ECC_AGGR	0x0040700000	0x0040700400	1 KB
MCU_MCAN1_ECC_AGGR	0x0040701000	0x0040701400	1 KB
MCU_ADC12FCC0_ECC	0x0040707000	0x0040707400	1 KB
MCU_ADC12FCC1_ECC	0x0040708000	0x0040708400	1 KB
MCU_MSRAM_1MB0_ECC_AGGR_REGS	0x004070B000	0x004070B400	1 KB
MCU_I3C0_P_ECC_AGGR_CFG	0x0040720000	0x0040720400	1 KB
MCU_I3C0_S_ECC_AGGR_CFG	0x0040721000	0x0040721400	1 KB
MCU_I3C1_P_ECC_AGGR_CFG	0x0040722000	0x0040722400	1 KB
MCU_I3C1_S_ECC_AGGR_CFG	0x0040723000	0x0040723400	1 KB
MCU_VDC_INFRA_VBUSP_32B_SRC_SAFEG0_CFG	0x0040731000	0x0040731400	1 KB
MCU_VDC_SOC_FW_VBUSP_32B_SRC_SAFEG1_CFG	0x0040732000	0x0040732400	1 KB
WKUP_ECC_AGGR0_REGS	0x0042410000	0x0042410400	1 KB
WKUP_VTM0_ECCAGGR_CFG	0x0042810000	0x0042810400	1 KB
WKUP_VDC_INFRA_VBUSP_32B_SRC_SAFEG0_CFG	0x0042900000	0x0042900400	1 KB
MAIN_CBASS0_FW	0x0045000000	0x0045008000	32 KB
CBASS_INFRA_NON_SAFE0_FW	0x0045010000	0x0045020000	64 KB
COMPUTE_CLUSTER0_MPU0_COREPAC_FW	0x0045040400	0x0045040800	1 KB
COMPUTE_CLUSTER0_DSP0_L2_FW	0x0045042000	0x0045042400	1 KB
COMPUTE_CLUSTER0_DSP0_MDMA_FW	0x0045042400	0x0045042800	1 KB
COMPUTE_CLUSTER0_DSP1_L2_FW	0x0045042800	0x0045042C00	1 KB
COMPUTE_CLUSTER0_DSP1_MDMA_FW	0x0045042C00	0x0045043000	1 KB
COMPUTE_CLUSTER0_DRU0_FW	0x0045047000	0x0045047400	1 KB
COMPUTE_CLUSTER0_DRU0_MMR_FW	0x0045048000	0x0045050000	32 KB
CBASS_IPPHY_SAFE0_FW	0x0045200000	0x0045220000	128 KB
CBASS_IPPHY0_FW	0x0045220000	0x0045240000	128 KB
CBASS_RC0_FW	0x0045240000	0x0045244000	16 KB
CBASS_RC_CFG0_FW	0x0045258000	0x0045258800	2 KB
CBASS_CSI0_FW	0x0045260000	0x0045268000	32 KB
CBASS_DATADEBUG0_FW	0x0045268000	0x0045270000	32 KB
AEP_HC2_CBASS0_FW	0x0045278000	0x0045280000	32 KB
AEP_HC_CFG_CBASS0_FW	0x0045280000	0x0045288000	32 KB
CBASS_AC_NONSAFE0_FW	0x0045290000	0x0045290800	2 KB
CBASS_AC_CFG_NONSAFE0_FW	0x00452A0000	0x00452B0000	64 KB
AM_PULSAR0_SLV_CBASS0_FW	0x00452B0000	0x00452C0000	64 KB
CBASS_AC_CFG0_FW	0x00452C0000	0x00452C8000	32 KB
AM_PULSAR0_MEM_CBASS0_FW	0x00452D0000	0x00452D0400	1 KB
AM_PULSAR1_MEM_CBASS0_FW	0x00452E0000	0x00452E2000	8 KB
NAVSS0_UDMASS_DMSC_FW	0x0045400000	0x0045480000	512 KB
NAVSS0_MODSS_DMSC_FW	0x0045480000	0x00454C0000	256 KB
NAVSS0_VIRTSS_DMSC_FW	0x0045500000	0x0045508000	32 KB
DMPAC0_CFG_DMSC_FW	0x00455D8000	0x00455DC000	16 KB
DMPAC0_SL2_DMSC_FW	0x00455E0000	0x00455E1000	4 KB
VPAC0_DMSC_VPAC_SCRPFW	0x00455E8000	0x00455EC000	16 KB
VPAC0_DMSC_VPAC_SCRMFW	0x00455F0000	0x00455F1000	4 KB
COMPUTE_CLUSTER0_DMSC_PRIVID	0x0045830000	0x0045834000	16 KB
NAVSS0_MODSS_DMSC_ISC	0x0045840000	0x0045840800	2 KB

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
DMPAC0_SL2_DMSC_ISC	0x0045858000	0x004585C000	16 KB
VPAC0_DMSC_VPAC_SCRMISC	0x0045860000	0x0045862000	8 KB
NAVSS0_UDMASS_RINGACC0_ISC	0x0045870000	0x0045878000	32 KB
CBASS_IPPHY_SAFE0_ISC	0x0045878000	0x0045879000	4 KB
CBASS_RC0_ISC	0x0045880000	0x0045890000	64 KB
AEP_HC2_CBASS0_ISC	0x0045898000	0x004589C000	16 KB
CBASS_DATADEBUG0_ISC	0x00458A0000	0x00458A0800	2 KB
CBASS_AC_NONSAFE0_ISC	0x00458C0000	0x00458C8000	32 KB
AM_PULSAR0_MEM_CBASS0_ISC	0x00458C8000	0x00458C9000	4 KB
AM_PULSAR1_PERIPH_SWITCH_CBASS0_ISC	0x00458CA000	0x00458CC000	8 KB
NAVSS0_UDMASS_DMSC_ISC	0x00458D0000	0x00458D0400	1 KB
AM_PULSAR1_MEM_CBASS0_ISC	0x00458D8000	0x00458DA000	8 KB
NAVSS0_CRED	0x00458E8000	0x00458E8400	1 KB
COMPUTE_CLUSTER0_DMSC_EMULATION	0x0045900000	0x0045904000	16 KB
SEC_MMR0_DBG_CTRL	0x0045944000	0x0045948000	16 KB
COMPUTE_CLUSTER0_DMSC_BOOT	0x0045A00000	0x0045A10000	64 KB
SEC_MMR0_BOOT_CTRL	0x0045A40000	0x0045A44000	16 KB
CBASS_AC_NONSAFE0_GLB	0x0045B08000	0x0045B08400	1 KB
NAVSS0_MODSS_DMSC_GLB	0x0045B0A000	0x0045B0A400	1 KB
NAVSS0_UDMASS_DMSC_GLB	0x0045B0B000	0x0045B0B400	1 KB
NAVSS0_VIRTSS_DMSC_GLB	0x0045B0B800	0x0045B0BC00	1 KB
MAIN_CBASS0_GLB	0x0045B0C000	0x0045B0C400	1 KB
DMPAC0_CFG_DMSC_GLB	0x0045B0D000	0x0045B0D400	1 KB
DMPAC0_SL2_DMSC_GLB	0x0045B0D400	0x0045B0D800	1 KB
VPAC0_DMSC_VPAC_SCRMGLB	0x0045B0D800	0x0045B0DC00	1 KB
VPAC0_DMSC_VPAC_SCRPGLB	0x0045B0DC00	0x0045B0E000	1 KB
CBASS_CSI0_GLB	0x0045B20400	0x0045B20800	1 KB
CBASS_DATADEBUG0_GLB	0x0045B20800	0x0045B20C00	1 KB
AEP_HC_CFG_CBASS0_GLB	0x0045B21000	0x0045B21400	1 KB
CBASS_IPPHY0_GLB	0x0045B21400	0x0045B21800	1 KB
CBASS_RC0_GLB	0x0045B22000	0x0045B22400	1 KB
CBASS_RC_CFG0_GLB	0x0045B22400	0x0045B22800	1 KB
AEP_HC2_CBASS0_GLB	0x0045B22800	0x0045B22C00	1 KB
CBASS_AC_CFG0_GLB	0x0045B22C00	0x0045B23000	1 KB
AM_PULSAR0_MEM_CBASS0_GLB	0x0045B23000	0x0045B23400	1 KB
AM_PULSAR0_SLV_CBASS0_GLB	0x0045B23400	0x0045B23800	1 KB
AM_PULSAR1_MEM_CBASS0_GLB	0x0045B23800	0x0045B23C00	1 KB
AM_PULSAR1_PERIPH_SWITCH_CBASS0_GLB	0x0045B23C00	0x0045B24000	1 KB
CBASS_INFRA_NON_SAFE0_GLB	0x0045B24000	0x0045B24400	1 KB
CBASS_IPPHY_SAFE0_GLB	0x0045B24400	0x0045B24800	1 KB
CBASS_AC_CFG_NONSAFE0_GLB	0x0045B25000	0x0045B25400	1 KB
COMPUTE_CLUSTER0_MPU0_COREPAC_FW_GLB	0x0045BC0400	0x0045BC0800	1 KB
COMPUTE_CLUSTER0_DSP0_L2_FW_GLB	0x0045BC2000	0x0045BC2400	1 KB
COMPUTE_CLUSTER0_DSP0_MDMA_FW_GLB	0x0045BC2400	0x0045BC2800	1 KB
COMPUTE_CLUSTER0_DSP1_L2_FW_GLB	0x0045BC2800	0x0045BC2C00	1 KB
COMPUTE_CLUSTER0_DSP1_MDMA_FW_GLB	0x0045BC2C00	0x0045BC3000	1 KB
COMPUTE_CLUSTER0_DRU0_FW_GLB	0x0045BC7000	0x0045BC7400	1 KB

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
COMPUTE_CLUSTER0_DRU0_MMR_FW_GLB	0x0045BC8000	0x0045BC8400	1 KB
NAVSS0_MODSS_DMSC_QOS	0x0045D40000	0x0045D40800	2 KB
CBASS_IPPHY_SAFE0_QOS	0x0045D78000	0x0045D79000	4 KB
CBASS_RC0_QOS	0x0045D80000	0x0045D88000	32 KB
AEP_HC2_CBASS0_QOS	0x0045D98000	0x0045DA0000	32 KB
CBASS_DATADEBUG0_QOS	0x0045DA0000	0x0045DA0800	2 KB
AM_PULSAR1_MEM_CBASS0_QOS	0x0045DA8000	0x0045DAA000	8 KB
CBASS_AC_NONSAFE0_QOS	0x0045DC0000	0x0045DC8000	32 KB
AM_PULSAR0_MEM_CBASS0_QOS	0x0045DC8000	0x0045DC9000	4 KB
AM_PULSAR1_PERIPH_SWITCH_CBASS0_QOS	0x0045DCA000	0x0045DCC000	8 KB
MCUM_MCU_ECC_AGGR0_REGS	0x0047200000	0x0047200400	1 KB
IAM_CNM_WAVE521CL_MAIN_0_PRI_M_VBUSM_R_BW_LIMITER0_REGS	0x0048000000	0x0048001000	4 KB
IAM_CNM_WAVE521CL_MAIN_0_PRI_M_VBUSM_W_BW_LIMITER0_REGS	0x0048001000	0x0048002000	4 KB
IGPU_MAIN_0_M0_VBUSM_R_ASYNC_BW_LIMITER0_REGS	0x0048006000	0x0048007000	4 KB
IGPU_MAIN_0_M0_VBUSM_W_ASYNC_BW_LIMITER0_REGS	0x0048007000	0x0048008000	4 KB
COMPUTE_CLUSTER0_CPU0	0x0060000000	0x0061000000	16 MB
NAVSS0_SRAM0	0x0060000000	0x0080000000	512 MB
NAVSS0_SRAM1	0x0060000000	0x0080000000	512 MB
COMPUTE_CLUSTER0_CPU1	0x0061000000	0x0062000000	16 MB
COMPUTE_CLUSTER0_CPU2	0x0062000000	0x0063000000	16 MB
COMPUTE_CLUSTER0_CPU3	0x0063000000	0x0064000000	16 MB
COMPUTE_CLUSTER0_CPU4	0x0064000000	0x0065000000	16 MB
COMPUTE_CLUSTER0_CPU5	0x0065000000	0x0066000000	16 MB
COMPUTE_CLUSTER0_CPU6	0x0066000000	0x0067000000	16 MB
COMPUTE_CLUSTER0_CPU7	0x0067000000	0x0068000000	16 MB
COMPUTE_CLUSTER0_CPU8	0x0068000000	0x0069000000	16 MB
COMPUTE_CLUSTER0_CPU9	0x0069000000	0x006A000000	16 MB
COMPUTE_CLUSTER0_CPU10	0x006A000000	0x006B000000	16 MB
COMPUTE_CLUSTER0_CPU11	0x006B000000	0x006C000000	16 MB
COMPUTE_CLUSTER0_CPU12	0x006C000000	0x006D000000	16 MB
COMPUTE_CLUSTER0_MMR_DRU0_MMR_CFG_DRU	0x006D000000	0x006D004000	16 KB
COMPUTE_CLUSTER0_MMR_DRU0_MMR_CFG_DRU_SET	0x006D004000	0x006D008000	16 KB
COMPUTE_CLUSTER0_MMR_DRU0_MMR_CFG_DRU_QUEUE	0x006D008000	0x006D00A000	8 KB
COMPUTE_CLUSTER0_MMR_DRU0_MMR_CFG_DRU_MMU	0x006D00A000	0x006D00C000	8 KB
COMPUTE_CLUSTER0_MMR_DRU0_MMR_CFG_DRU_UTLB	0x006D00C000	0x006D010000	16 KB
COMPUTE_CLUSTER0_MMR_DRU0_MMR_CFG_DRU_MEM_ATT0	0x006D010000	0x006D020000	64 KB
COMPUTE_CLUSTER0_MMR_DRU0_MMR_CFG_DRU_MEM_ATT1	0x006D020000	0x006D030000	64 KB
COMPUTE_CLUSTER0_MMR_DRU0_MMR_CFG_DRU_MEM_ATT2	0x006D030000	0x006D040000	64 KB
COMPUTE_CLUSTER0_MMR_DRU0_MMR_CFG_DRU_CHNRT	0x006D040000	0x006D060000	128 KB
COMPUTE_CLUSTER0_MMR_DRU0_MMR_CFG_DRU_CHRT	0x006D060000	0x006D080000	128 KB
COMPUTE_CLUSTER0_MMR_DRU0_MMR_CFG_DRU_CHATOMIC_DEBUG	0x006D080000	0x006D0A0000	128 KB
COMPUTE_CLUSTER0_MMR_DRU0_MMR_CFG_DRU_CHCORE	0x006D0A0000	0x006D0C0000	128 KB
COMPUTE_CLUSTER0_MMR_DRU0_MMR_CFG_DRU_CAUSE	0x006D0E0000	0x006D100000	128 KB
COMPUTE_CLUSTER0_MSMC_CFGS0	0x006E000000	0x006F000000	16 MB
COMPUTE_CLUSTER0_UNALLOCATED0	0x006F000000	0x0070000000	16 MB

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
COMPUTE_CLUSTER0_MSMC_SRAM	0x0070000000	0x0074000000	64 MB
COMPUTE_CLUSTER0_MSMC_ATOMIC_COUNTERS	0x0074000000	0x0078000000	64 MB
COMPUTE_CLUSTER0_CLEC	0x0078000000	0x0080000000	128 MB
NAVSS0_DDR0_MEM	0x0080000000	0x0100000000	2 GB
NAVSS0_DDR1_MEM	0x0080000000	0x0100000000	2 GB
NAVSS0_DDR0_MEM1	0x0800000000	0x1000000000	32 GB
NAVSS0_DDR1_MEM1	0x0800000000	0x1000000000	32 GB
PCIE1_DAT1	0x4100000000	0x4200000000	4 GB
NAVSS0_ALIAS64K_MCRC0_S_CFG_MCRC64	0x4A1F700000	0x4A1F710000	64 KB
NAVSS0_ALIAS64K_PSILCFG0_CFG_PROXY	0x4A1F780000	0x4A1F782000	8 KB
NAVSS0_ALIAS64K_MAILBOX0_CFG_REGS0	0x4A1F800000	0x4A1F802000	8 KB
NAVSS0_ALIAS64K_MAILBOX0_CFG_REGS1	0x4A1F810000	0x4A1F812000	8 KB
NAVSS0_ALIAS64K_MAILBOX0_CFG_REGS2	0x4A1F820000	0x4A1F822000	8 KB
NAVSS0_ALIAS64K_MAILBOX0_CFG_REGS3	0x4A1F830000	0x4A1F832000	8 KB
NAVSS0_ALIAS64K_MAILBOX0_CFG_REGS4	0x4A1F840000	0x4A1F842000	8 KB
NAVSS0_ALIAS64K_MAILBOX0_CFG_REGS5	0x4A1F850000	0x4A1F852000	8 KB
NAVSS0_ALIAS64K_MAILBOX0_CFG_REGS6	0x4A1F860000	0x4A1F862000	8 KB
NAVSS0_ALIAS64K_MAILBOX0_CFG_REGS7	0x4A1F870000	0x4A1F872000	8 KB
NAVSS0_ALIAS64K_MAILBOX0_CFG_REGS8	0x4A1F880000	0x4A1F882000	8 KB
NAVSS0_ALIAS64K_MAILBOX0_CFG_REGS9	0x4A1F890000	0x4A1F892000	8 KB
NAVSS0_ALIAS64K_MAILBOX0_CFG_REGS10	0x4A1F8A0000	0x4A1F8A2000	8 KB
NAVSS0_ALIAS64K_MAILBOX0_CFG_REGS11	0x4A1F8B0000	0x4A1F8B2000	8 KB
NAVSS0_ALIAS64K_MAILBOX1_CFG_REGS0	0x4A1F900000	0x4A1F902000	8 KB
NAVSS0_ALIAS64K_MAILBOX1_CFG_REGS1	0x4A1F910000	0x4A1F912000	8 KB
NAVSS0_ALIAS64K_MAILBOX1_CFG_REGS2	0x4A1F920000	0x4A1F922000	8 KB
NAVSS0_ALIAS64K_MAILBOX1_CFG_REGS3	0x4A1F930000	0x4A1F932000	8 KB
NAVSS0_ALIAS64K_MAILBOX1_CFG_REGS4	0x4A1F940000	0x4A1F942000	8 KB
NAVSS0_ALIAS64K_MAILBOX1_CFG_REGS5	0x4A1F950000	0x4A1F952000	8 KB
NAVSS0_ALIAS64K_MAILBOX1_CFG_REGS6	0x4A1F960000	0x4A1F962000	8 KB
NAVSS0_ALIAS64K_MAILBOX1_CFG_REGS7	0x4A1F970000	0x4A1F972000	8 KB
NAVSS0_ALIAS64K_MAILBOX1_CFG_REGS8	0x4A1F980000	0x4A1F982000	8 KB
NAVSS0_ALIAS64K_MAILBOX1_CFG_REGS9	0x4A1F990000	0x4A1F992000	8 KB
NAVSS0_ALIAS64K_MAILBOX1_CFG_REGS10	0x4A1F9A0000	0x4A1F9A2000	8 KB
NAVSS0_ALIAS64K_MAILBOX1_CFG_REGS11	0x4A1F9B0000	0x4A1F9B2000	8 KB
NAVSS0_ALIAS64K_RINGACC0_CFG_MON	0x4A20000000	0x4A20200000	2 MB
NAVSS0_ALIAS64K_TIMERMGR0_CFG_TIMERS	0x4A22000000	0x4A22400000	4 MB
NAVSS0_ALIAS64K_TIMERMGR1_CFG_TIMERS	0x4A22400000	0x4A22800000	4 MB
NAVSS0_ALIAS64K_SEC_PROXY0_CFG_RT	0x4A24000000	0x4A26000000	32 MB
NAVSS0_ALIAS64K_SEC_PROXY0_CFG_SCFG	0x4A28000000	0x4A2A000000	32 MB
NAVSS0_ALIAS64K_SEC_PROXY0_SRC_TARGET_DATA	0x4A2C000000	0x4A2E000000	32 MB
NAVSS0_ALIAS64K_PROXY0_SRC_TARGET0_DATA	0x4A30000000	0x4A30400000	4 MB
NAVSS0_ALIAS64K_PROXY0_CFG_BUF_CFG	0x4A34000000	0x4A34400000	4 MB
NAVSS0_ALIAS64K_UDMASS_INTA0_CFG_GCNTRTI	0x4A38000000	0x4A3A000000	32 MB
NAVSS0_ALIAS64K_MODSS_INTA0_CFG_INTR	0x4A3C000000	0x4A3C400000	4 MB
NAVSS0_ALIAS64K_MODSS_INTA1_CFG_INTR	0x4A3C400000	0x4A3C800000	4 MB
NAVSS0_ALIAS64K_UDMASS_INTA0_CFG_INTR	0x4A3D000000	0x4A3E000000	16 MB
NAVSS0_ALIAS64K_UDMAP0_CFG_RCHANRT	0x4A40000000	0x4A40800000	8 MB

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
NAVSS0_ALIAS64K_UDMAP0_CFG_TCHANRT	0x4A50000000	0x4A52000000	32 MB
NAVSS0_ALIAS64K_BCDMA0_CFG_TCHAN	0x4A58400000	0x4A58410000	64 KB
NAVSS0_ALIAS64K_BCDMA0_CFG_RCHAN	0x4A58800000	0x4A58820000	128 KB
NAVSS0_ALIAS64K_BCDMA0_CFG_RING	0x4A59000000	0x4A59040000	256 KB
NAVSS0_ALIAS64K_BCDMA0_CFG_TCHANRT	0x4A5C000000	0x4A5C100000	1 MB
NAVSS0_ALIAS64K_BCDMA0_CFG_RCHANRT	0x4A5D000000	0x4A5D200000	2 MB
NAVSS0_ALIAS64K_BCDMA0_CFG_RINGRT	0x4A5E000000	0x4A5E800000	8 MB
NAVSS0_ALIAS64K_IO_PVU0_CFG_TLBIF_TLB	0x4A60000000	0x4A60400000	4 MB
NAVSS0_ALIAS64K_IO_PVU1_CFG_TLBIF_TLB	0x4A60400000	0x4A60800000	4 MB
NAVSS0_ALIAS64K_RINGACC0_SRC_FIFOS	0x4A80000000	0x4A84000000	64 MB
NAVSS0_ALIAS64K_RINGACC0_CFG_RT	0x4AC0000000	0x4AC4000000	64 MB
NAVSS0_VIRT_ALIAS_0_MODSS_INTA0_CFG	0x4B00800000	0x4B00800020	32 B
NAVSS0_VIRT_ALIAS_0_MODSS_INTA1_CFG	0x4B00801000	0x4B00801020	32 B
NAVSS0_VIRT_ALIAS_0_UDMASS_INTA0_CFG	0x4B00802000	0x4B00802020	32 B
NAVSS0_VIRT_ALIAS_0_UDMASS_INTA0_CFG_UNMAP	0x4B00880000	0x4B00890000	64 KB
NAVSS0_VIRT_ALIAS_0_MODSS_INTA0_CFG_IMAP	0x4B00900000	0x4B00902000	8 KB
NAVSS0_VIRT_ALIAS_0_MODSS_INTA1_CFG_IMAP	0x4B00908000	0x4B0090A000	8 KB
NAVSS0_VIRT_ALIAS_0_UDMASS_INTA0_CFG_IMAP	0x4B00940000	0x4B00950000	64 KB
NAVSS0_VIRT_ALIAS_0_NAV_DDR0_VIRTID_CFG_MMRS	0x4B00A02000	0x4B00A02100	256 B
NAVSS0_VIRT_ALIAS_0_NAV_DDR1_VIRTID_CFG_MMRS	0x4B00A03000	0x4B00A03100	256 B
NAVSS0_VIRT_ALIAS_0_UDMAP0_CFG_TCHAN	0x4B00B00000	0x4B00B20000	128 KB
NAVSS0_VIRT_ALIAS_0_UDMAP0_CFG_RCHAN	0x4B00C00000	0x4B00C08000	32 KB
NAVSS0_VIRT_ALIAS_0_UDMAP0_CFG_RFLOW	0x4B00D00000	0x4B00D04000	16 KB
NAVSS0_VIRT_ALIAS_0_SPINLOCK0_CFG	0x4B00E00000	0x4B00E08000	32 KB
NAVSS0_VIRT_ALIAS_0_TIMERMGR0_CFG_CONFIG	0x4B00E80000	0x4B00E80200	512 B
NAVSS0_VIRT_ALIAS_0_TIMERMGR1_CFG_CONFIG	0x4B00E81000	0x4B00E81200	512 B
NAVSS0_VIRT_ALIAS_0_TIMERMGR0_CFG_OES	0x4B00F00000	0x4B00F01000	4 KB
NAVSS0_VIRT_ALIAS_0_TIMERMGR1_CFG_OES	0x4B00F01000	0x4B00F02000	4 KB
NAVSS0_VIRT_ALIAS_0_IO_PVU0_CFG_MMRS	0x4B00F80000	0x4B00F81000	4 KB
NAVSS0_VIRT_ALIAS_0_IO_PVU1_CFG_MMRS	0x4B00F81000	0x4B00F82000	4 KB
NAVSS0_VIRT_ALIAS_0_PVU0_SRC_TOG_CFG	0x4B00F90000	0x4B00F90400	1 KB
NAVSS0_VIRT_ALIAS_0_PVU0_CFG_TOG_CFG	0x4B00F91000	0x4B00F91400	1 KB
NAVSS0_VIRT_ALIAS_0_ECCAGGR0	0x4B01000000	0x4B01000400	1 KB
NAVSS0_VIRT_ALIAS_0_UDMASS_ECCAGGR_CFG	0x4B01001000	0x4B01001400	1 KB
NAVSS0_VIRT_ALIAS_0_VIRTSS_ECCAGGR_CFG	0x4B01002000	0x4B01002400	1 KB
NAVSS0_VIRT_ALIAS_0_UDMASS_INTA0_CFG_GCNTCFG	0x4B01040000	0x4B01044000	16 KB
NAVSS0_VIRT_ALIAS_0_RINGACC0_CFG	0x4B01080000	0x4B010C0000	256 KB
NAVSS0_VIRT_ALIAS_0_REGS0_CFG_MMRS	0x4B010C0000	0x4B010C0100	256 B
NAVSS0_VIRT_ALIAS_0_CPTS0_S_VBUSP_CPTS_VBUSP	0x4B010D0000	0x4B010D0400	1 KB
NAVSS0_VIRT_ALIAS_0_INTR0_CFG_INTR_ROUTER_CFG	0x4B010E0000	0x4B010E4000	16 KB
NAVSS0_VIRT_ALIAS_0_UDMASS_INTA0_CFG_L2G	0x4B01100000	0x4B01102000	8 KB
NAVSS0_VIRT_ALIAS_0_UDMASS_INTA0_CFG_MCAST	0x4B01110000	0x4B01114000	16 KB
NAVSS0_VIRT_ALIAS_0_PROXY0_CFG_BUF_CFG_GCFG	0x4B01120000	0x4B01120100	256 B
NAVSS0_VIRT_ALIAS_0_PROXY0_CFG_BUFRAM_SLV_RAM	0x4B01130000	0x4B01134000	16 KB
NAVSS0_VIRT_ALIAS_0_SEC_PROXY0_CFG_MMRS	0x4B01140000	0x4B01140100	256 B
NAVSS0_VIRT_ALIAS_0_UDMAP0_CFG_GCFG	0x4B01150000	0x4B01150100	256 B
NAVSS0_VIRT_ALIAS_0_RINGACC0_CFG_GCFG	0x4B01160000	0x4B01160400	1 KB

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
NAVSS0_VIRT_ALIAS_0_PSILSS0_CFG_MMRS	0x4B01170000	0x4B01171000	4 KB
NAVSS0_VIRT_ALIAS_0_BCDMA0_CFG_GCFG	0x4B011A0000	0x4B011A0100	256 B
NAVSS0_VIRT_ALIAS_0_MCRC0_S_CFG_MCRC64	0x4B01F70000	0x4B01F71000	4 KB
NAVSS0_VIRT_ALIAS_0_PSILCFG0_CFG_PROXY	0x4B01F78000	0x4B01F78200	512 B
NAVSS0_VIRT_ALIAS_0_MAILBOX0_CFG_REGS0	0x4B01F80000	0x4B01F80200	512 B
NAVSS0_VIRT_ALIAS_0_MAILBOX0_CFG_REGS1	0x4B01F81000	0x4B01F81200	512 B
NAVSS0_VIRT_ALIAS_0_MAILBOX0_CFG_REGS2	0x4B01F82000	0x4B01F82200	512 B
NAVSS0_VIRT_ALIAS_0_MAILBOX0_CFG_REGS3	0x4B01F83000	0x4B01F83200	512 B
NAVSS0_VIRT_ALIAS_0_MAILBOX0_CFG_REGS4	0x4B01F84000	0x4B01F84200	512 B
NAVSS0_VIRT_ALIAS_0_MAILBOX0_CFG_REGS5	0x4B01F85000	0x4B01F85200	512 B
NAVSS0_VIRT_ALIAS_0_MAILBOX0_CFG_REGS6	0x4B01F86000	0x4B01F86200	512 B
NAVSS0_VIRT_ALIAS_0_MAILBOX0_CFG_REGS7	0x4B01F87000	0x4B01F87200	512 B
NAVSS0_VIRT_ALIAS_0_MAILBOX0_CFG_REGS8	0x4B01F88000	0x4B01F88200	512 B
NAVSS0_VIRT_ALIAS_0_MAILBOX0_CFG_REGS9	0x4B01F89000	0x4B01F89200	512 B
NAVSS0_VIRT_ALIAS_0_MAILBOX0_CFG_REGS10	0x4B01F8A000	0x4B01F8A200	512 B
NAVSS0_VIRT_ALIAS_0_MAILBOX0_CFG_REGS11	0x4B01F8B000	0x4B01F8B200	512 B
NAVSS0_VIRT_ALIAS_0_MAILBOX1_CFG_REGS0	0x4B01F90000	0x4B01F90200	512 B
NAVSS0_VIRT_ALIAS_0_MAILBOX1_CFG_REGS1	0x4B01F91000	0x4B01F91200	512 B
NAVSS0_VIRT_ALIAS_0_MAILBOX1_CFG_REGS2	0x4B01F92000	0x4B01F92200	512 B
NAVSS0_VIRT_ALIAS_0_MAILBOX1_CFG_REGS3	0x4B01F93000	0x4B01F93200	512 B
NAVSS0_VIRT_ALIAS_0_MAILBOX1_CFG_REGS4	0x4B01F94000	0x4B01F94200	512 B
NAVSS0_VIRT_ALIAS_0_MAILBOX1_CFG_REGS5	0x4B01F95000	0x4B01F95200	512 B
NAVSS0_VIRT_ALIAS_0_MAILBOX1_CFG_REGS6	0x4B01F96000	0x4B01F96200	512 B
NAVSS0_VIRT_ALIAS_0_MAILBOX1_CFG_REGS7	0x4B01F97000	0x4B01F97200	512 B
NAVSS0_VIRT_ALIAS_0_MAILBOX1_CFG_REGS8	0x4B01F98000	0x4B01F98200	512 B
NAVSS0_VIRT_ALIAS_0_MAILBOX1_CFG_REGS9	0x4B01F99000	0x4B01F99200	512 B
NAVSS0_VIRT_ALIAS_0_MAILBOX1_CFG_REGS10	0x4B01F9A000	0x4B01F9A200	512 B
NAVSS0_VIRT_ALIAS_0_MAILBOX1_CFG_REGS11	0x4B01F9B000	0x4B01F9B200	512 B
NAVSS0_VIRT_ALIAS_0_RINGACC0_CFG_MON	0x4B02000000	0x4B02020000	128 KB
NAVSS0_VIRT_ALIAS_0_TIMERMGR0_CFG_TIMERS	0x4B02200000	0x4B02240000	256 KB
NAVSS0_VIRT_ALIAS_0_TIMERMGR1_CFG_TIMERS	0x4B02240000	0x4B02280000	256 KB
NAVSS0_VIRT_ALIAS_0_SEC_PROXY0_CFG_RT	0x4B02400000	0x4B02600000	2 MB
NAVSS0_VIRT_ALIAS_0_SEC_PROXY0_CFG_SCFG	0x4B02800000	0x4B02A00000	2 MB
NAVSS0_VIRT_ALIAS_0_SEC_PROXY0_SRC_TARGET_DATA	0x4B02C00000	0x4B02E00000	2 MB
NAVSS0_VIRT_ALIAS_0_PROXY0_SRC_TARGET0_DATA	0x4B03000000	0x4B03040000	256 KB
NAVSS0_VIRT_ALIAS_0_PROXY0_CFG_BUF_CFG	0x4B03400000	0x4B03440000	256 KB
NAVSS0_VIRT_ALIAS_0_UDMASS_INTA0_CFG_GCNTRTI	0x4B03800000	0x4B03A00000	2 MB
NAVSS0_VIRT_ALIAS_0_MODSS_INTA0_CFG_INTR	0x4B03C00000	0x4B03C40000	256 KB
NAVSS0_VIRT_ALIAS_0_MODSS_INTA1_CFG_INTR	0x4B03C40000	0x4B03C80000	256 KB
NAVSS0_VIRT_ALIAS_0_UDMASS_INTA0_CFG_INTR	0x4B03D00000	0x4B03E00000	1 MB
NAVSS0_VIRT_ALIAS_0_UDMAP0_CFG_RCHANRT	0x4B04000000	0x4B04080000	512 KB
NAVSS0_VIRT_ALIAS_0_UDMAP0_CFG_TCHANRT	0x4B05000000	0x4B05200000	2 MB
NAVSS0_VIRT_ALIAS_0_BCDMA0_CFG_TCHAN	0x4B05840000	0x4B05841000	4 KB
NAVSS0_VIRT_ALIAS_0_BCDMA0_CFG_RCHAN	0x4B05880000	0x4B05882000	8 KB
NAVSS0_VIRT_ALIAS_0_BCDMA0_CFG_RING	0x4B05900000	0x4B05904000	16 KB
NAVSS0_VIRT_ALIAS_0_BCDMA0_CFG_TCHANRT	0x4B05C00000	0x4B05C10000	64 KB
NAVSS0_VIRT_ALIAS_0_BCDMA0_CFG_RCHANRT	0x4B05D00000	0x4B05D20000	128 KB

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
NAVSS0_VIRT_ALIAS_0_BCDMA0_CFG_RINGRT	0x4B05E00000	0x4B05E80000	512 KB
NAVSS0_VIRT_ALIAS_0_IO_PVU0_CFG_TLBIF_TLB	0x4B06000000	0x4B06040000	256 KB
NAVSS0_VIRT_ALIAS_0_IO_PVU1_CFG_TLBIF_TLB	0x4B06040000	0x4B06080000	256 KB
NAVSS0_VIRT_ALIAS_0_RINGACC0_SRC_FIFOS	0x4B08000000	0x4B08400000	4 MB
NAVSS0_VIRT_ALIAS_0_RINGACC0_CFG_RT	0x4B0C000000	0x4B0C400000	4 MB
NAVSS0_VIRT_ALIAS_1_MODSS_INTA0_CFG	0x4B10800000	0x4B10800020	32 B
NAVSS0_VIRT_ALIAS_1_MODSS_INTA1_CFG	0x4B10801000	0x4B10801020	32 B
NAVSS0_VIRT_ALIAS_1_UDMASS_INTA0_CFG	0x4B10802000	0x4B10802020	32 B
NAVSS0_VIRT_ALIAS_1_UDMASS_INTA0_CFG_UNMAP	0x4B10880000	0x4B10890000	64 KB
NAVSS0_VIRT_ALIAS_1_MODSS_INTA0_CFG_IMAP	0x4B10900000	0x4B10902000	8 KB
NAVSS0_VIRT_ALIAS_1_MODSS_INTA1_CFG_IMAP	0x4B10908000	0x4B1090A000	8 KB
NAVSS0_VIRT_ALIAS_1_UDMASS_INTA0_CFG_IMAP	0x4B10940000	0x4B10950000	64 KB
NAVSS0_VIRT_ALIAS_1_NAV_DDR0_VIRTID_CFG_MMRS	0x4B10A02000	0x4B10A02100	256 B
NAVSS0_VIRT_ALIAS_1_NAV_DDR1_VIRTID_CFG_MMRS	0x4B10A03000	0x4B10A03100	256 B
NAVSS0_VIRT_ALIAS_1_UDMAP0_CFG_TCHAN	0x4B10B00000	0x4B10B20000	128 KB
NAVSS0_VIRT_ALIAS_1_UDMAP0_CFG_RCHAN	0x4B10C00000	0x4B10C08000	32 KB
NAVSS0_VIRT_ALIAS_1_UDMAP0_CFG_RFLOW	0x4B10D00000	0x4B10D04000	16 KB
NAVSS0_VIRT_ALIAS_1_SPINLOCK0_CFG	0x4B10E00000	0x4B10E08000	32 KB
NAVSS0_VIRT_ALIAS_1_TIMERMGR0_CFG_CONFIG	0x4B10E80000	0x4B10E80200	512 B
NAVSS0_VIRT_ALIAS_1_TIMERMGR1_CFG_CONFIG	0x4B10E81000	0x4B10E81200	512 B
NAVSS0_VIRT_ALIAS_1_TIMERMGR0_CFG_OES	0x4B10F00000	0x4B10F01000	4 KB
NAVSS0_VIRT_ALIAS_1_TIMERMGR1_CFG_OES	0x4B10F01000	0x4B10F02000	4 KB
NAVSS0_VIRT_ALIAS_1_IO_PVU0_CFG_MMRS	0x4B10F80000	0x4B10F81000	4 KB
NAVSS0_VIRT_ALIAS_1_IO_PVU1_CFG_MMRS	0x4B10F81000	0x4B10F82000	4 KB
NAVSS0_VIRT_ALIAS_1_PVU0_SRC_TOG_CFG	0x4B10F90000	0x4B10F90400	1 KB
NAVSS0_VIRT_ALIAS_1_PVU0_CFG_TOG_CFG	0x4B10F91000	0x4B10F91400	1 KB
NAVSS0_VIRT_ALIAS_1_ECCAGGR0	0x4B11000000	0x4B11000400	1 KB
NAVSS0_VIRT_ALIAS_1_UDMASS_ECCAGGR_CFG	0x4B11001000	0x4B11001400	1 KB
NAVSS0_VIRT_ALIAS_1_VIRTSS_ECCAGGR_CFG	0x4B11002000	0x4B11002400	1 KB
NAVSS0_VIRT_ALIAS_1_UDMASS_INTA0_CFG_GCNTCFG	0x4B11040000	0x4B11044000	16 KB
NAVSS0_VIRT_ALIAS_1_RINGACC0_CFG	0x4B11080000	0x4B110C0000	256 KB
NAVSS0_VIRT_ALIAS_1_REGS0_CFG_MMRS	0x4B110C0000	0x4B110C0100	256 B
NAVSS0_VIRT_ALIAS_1_CPTS0_S_VBUSP_CPTS_VBUSP	0x4B110D0000	0x4B110D0400	1 KB
NAVSS0_VIRT_ALIAS_1_INTR0_CFG_INTR_ROUTER_CFG	0x4B110E0000	0x4B110E4000	16 KB
NAVSS0_VIRT_ALIAS_1_UDMASS_INTA0_CFG_L2G	0x4B11100000	0x4B11102000	8 KB
NAVSS0_VIRT_ALIAS_1_UDMASS_INTA0_CFG_MCAST	0x4B11110000	0x4B11114000	16 KB
NAVSS0_VIRT_ALIAS_1_PROXY0_CFG_BUF_CFG_GCFG	0x4B11120000	0x4B11120100	256 B
NAVSS0_VIRT_ALIAS_1_PROXY0_CFG_BUFRAM_SLV_RAM	0x4B11130000	0x4B11134000	16 KB
NAVSS0_VIRT_ALIAS_1_SEC_PROXY0_CFG_MMRS	0x4B11140000	0x4B11140100	256 B
NAVSS0_VIRT_ALIAS_1_UDMAP0_CFG_GCFG	0x4B11150000	0x4B11150100	256 B
NAVSS0_VIRT_ALIAS_1_RINGACC0_CFG_GCFG	0x4B11160000	0x4B11160400	1 KB
NAVSS0_VIRT_ALIAS_1_PSILSS0_CFG_MMRS	0x4B11170000	0x4B11171000	4 KB
NAVSS0_VIRT_ALIAS_1_BCDMA0_CFG_GCFG	0x4B111A0000	0x4B111A0100	256 B
NAVSS0_VIRT_ALIAS_1_MCRC0_S_CFG_MCRC64	0x4B11F70000	0x4B11F71000	4 KB
NAVSS0_VIRT_ALIAS_1_PSILCFG0_CFG_PROXY	0x4B11F78000	0x4B11F78200	512 B
NAVSS0_VIRT_ALIAS_1_MAILBOX0_CFG_REGS0	0x4B11F80000	0x4B11F80200	512 B
NAVSS0_VIRT_ALIAS_1_MAILBOX0_CFG_REGS1	0x4B11F81000	0x4B11F81200	512 B

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
NAVSS0_VIRT_ALIAS_1_MAILBOX0_CFG_REGS2	0x4B11F82000	0x4B11F82200	512 B
NAVSS0_VIRT_ALIAS_1_MAILBOX0_CFG_REGS3	0x4B11F83000	0x4B11F83200	512 B
NAVSS0_VIRT_ALIAS_1_MAILBOX0_CFG_REGS4	0x4B11F84000	0x4B11F84200	512 B
NAVSS0_VIRT_ALIAS_1_MAILBOX0_CFG_REGS5	0x4B11F85000	0x4B11F85200	512 B
NAVSS0_VIRT_ALIAS_1_MAILBOX0_CFG_REGS6	0x4B11F86000	0x4B11F86200	512 B
NAVSS0_VIRT_ALIAS_1_MAILBOX0_CFG_REGS7	0x4B11F87000	0x4B11F87200	512 B
NAVSS0_VIRT_ALIAS_1_MAILBOX0_CFG_REGS8	0x4B11F88000	0x4B11F88200	512 B
NAVSS0_VIRT_ALIAS_1_MAILBOX0_CFG_REGS9	0x4B11F89000	0x4B11F89200	512 B
NAVSS0_VIRT_ALIAS_1_MAILBOX0_CFG_REGS10	0x4B11F8A000	0x4B11F8A200	512 B
NAVSS0_VIRT_ALIAS_1_MAILBOX0_CFG_REGS11	0x4B11F8B000	0x4B11F8B200	512 B
NAVSS0_VIRT_ALIAS_1_MAILBOX1_CFG_REGS0	0x4B11F90000	0x4B11F90200	512 B
NAVSS0_VIRT_ALIAS_1_MAILBOX1_CFG_REGS1	0x4B11F91000	0x4B11F91200	512 B
NAVSS0_VIRT_ALIAS_1_MAILBOX1_CFG_REGS2	0x4B11F92000	0x4B11F92200	512 B
NAVSS0_VIRT_ALIAS_1_MAILBOX1_CFG_REGS3	0x4B11F93000	0x4B11F93200	512 B
NAVSS0_VIRT_ALIAS_1_MAILBOX1_CFG_REGS4	0x4B11F94000	0x4B11F94200	512 B
NAVSS0_VIRT_ALIAS_1_MAILBOX1_CFG_REGS5	0x4B11F95000	0x4B11F95200	512 B
NAVSS0_VIRT_ALIAS_1_MAILBOX1_CFG_REGS6	0x4B11F96000	0x4B11F96200	512 B
NAVSS0_VIRT_ALIAS_1_MAILBOX1_CFG_REGS7	0x4B11F97000	0x4B11F97200	512 B
NAVSS0_VIRT_ALIAS_1_MAILBOX1_CFG_REGS8	0x4B11F98000	0x4B11F98200	512 B
NAVSS0_VIRT_ALIAS_1_MAILBOX1_CFG_REGS9	0x4B11F99000	0x4B11F99200	512 B
NAVSS0_VIRT_ALIAS_1_MAILBOX1_CFG_REGS10	0x4B11F9A000	0x4B11F9A200	512 B
NAVSS0_VIRT_ALIAS_1_MAILBOX1_CFG_REGS11	0x4B11F9B000	0x4B11F9B200	512 B
NAVSS0_VIRT_ALIAS_1_RINGACC0_CFG_MON	0x4B12000000	0x4B12020000	128 KB
NAVSS0_VIRT_ALIAS_1_TIMERMGR0_CFG_TIMERS	0x4B12200000	0x4B12240000	256 KB
NAVSS0_VIRT_ALIAS_1_TIMERMGR1_CFG_TIMERS	0x4B12240000	0x4B12280000	256 KB
NAVSS0_VIRT_ALIAS_1_SEC_PROXY0_CFG_RT	0x4B12400000	0x4B12600000	2 MB
NAVSS0_VIRT_ALIAS_1_SEC_PROXY0_CFG_SCFG	0x4B12800000	0x4B12A00000	2 MB
NAVSS0_VIRT_ALIAS_1_SEC_PROXY0_SRC_TARGET_DATA	0x4B12C00000	0x4B12E00000	2 MB
NAVSS0_VIRT_ALIAS_1_PROXY0_SRC_TARGET0_DATA	0x4B13000000	0x4B13040000	256 KB
NAVSS0_VIRT_ALIAS_1_PROXY0_CFG_BUF_CFG	0x4B13400000	0x4B13440000	256 KB
NAVSS0_VIRT_ALIAS_1_UDMASS_INTA0_CFG_GCNTRTI	0x4B13800000	0x4B13A00000	2 MB
NAVSS0_VIRT_ALIAS_1_MODSS_INTA0_CFG_INTR	0x4B13C00000	0x4B13C40000	256 KB
NAVSS0_VIRT_ALIAS_1_MODSS_INTA1_CFG_INTR	0x4B13C40000	0x4B13C80000	256 KB
NAVSS0_VIRT_ALIAS_1_UDMASS_INTA0_CFG_INTR	0x4B13D00000	0x4B13E00000	1 MB
NAVSS0_VIRT_ALIAS_1_UDMAP0_CFG_RCHANRT	0x4B14000000	0x4B14080000	512 KB
NAVSS0_VIRT_ALIAS_1_UDMAP0_CFG_TCHANRT	0x4B15000000	0x4B15200000	2 MB
NAVSS0_VIRT_ALIAS_1_BCDMA0_CFG_TCHAN	0x4B15840000	0x4B15841000	4 KB
NAVSS0_VIRT_ALIAS_1_BCDMA0_CFG_RCHAN	0x4B15880000	0x4B15882000	8 KB
NAVSS0_VIRT_ALIAS_1_BCDMA0_CFG_RING	0x4B15900000	0x4B15904000	16 KB
NAVSS0_VIRT_ALIAS_1_BCDMA0_CFG_TCHANRT	0x4B15C00000	0x4B15C10000	64 KB
NAVSS0_VIRT_ALIAS_1_BCDMA0_CFG_RCHANRT	0x4B15D00000	0x4B15D20000	128 KB
NAVSS0_VIRT_ALIAS_1_BCDMA0_CFG_RINGRT	0x4B15E00000	0x4B15E80000	512 KB
NAVSS0_VIRT_ALIAS_1_IO_PVU0_CFG_TLBIF_TLB	0x4B16000000	0x4B16040000	256 KB
NAVSS0_VIRT_ALIAS_1_IO_PVU1_CFG_TLBIF_TLB	0x4B16040000	0x4B16080000	256 KB
NAVSS0_VIRT_ALIAS_1_RINGACC0_SRC_FIFOS	0x4B18000000	0x4B18400000	4 MB
NAVSS0_VIRT_ALIAS_1_RINGACC0_CFG_RT	0x4B1C000000	0x4B1C400000	4 MB
NAVSS0_VIRT_ALIAS_2_MODSS_INTA0_CFG	0x4B20800000	0x4B20800020	32 B

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
NAVSS0_VIRT_ALIAS_2_MODSS_INTA1_CFG	0x4B20801000	0x4B20801020	32 B
NAVSS0_VIRT_ALIAS_2_UDMASS_INTA0_CFG	0x4B20802000	0x4B20802020	32 B
NAVSS0_VIRT_ALIAS_2_UDMASS_INTA0_CFG_UNMAP	0x4B20880000	0x4B20890000	64 KB
NAVSS0_VIRT_ALIAS_2_MODSS_INTA0_CFG_IMAP	0x4B20900000	0x4B20902000	8 KB
NAVSS0_VIRT_ALIAS_2_MODSS_INTA1_CFG_IMAP	0x4B20908000	0x4B2090A000	8 KB
NAVSS0_VIRT_ALIAS_2_UDMASS_INTA0_CFG_IMAP	0x4B20940000	0x4B20950000	64 KB
NAVSS0_VIRT_ALIAS_2_NAV_DDR0_VIRTID_CFG_MMRS	0x4B20A02000	0x4B20A02100	256 B
NAVSS0_VIRT_ALIAS_2_NAV_DDR1_VIRTID_CFG_MMRS	0x4B20A03000	0x4B20A03100	256 B
NAVSS0_VIRT_ALIAS_2_UDMAP0_CFG_TCHAN	0x4B20B00000	0x4B20B20000	128 KB
NAVSS0_VIRT_ALIAS_2_UDMAP0_CFG_RCHAN	0x4B20C00000	0x4B20C08000	32 KB
NAVSS0_VIRT_ALIAS_2_UDMAP0_CFG_RFLOW	0x4B20D00000	0x4B20D04000	16 KB
NAVSS0_VIRT_ALIAS_2_SPINLOCK0_CFG	0x4B20E00000	0x4B20E08000	32 KB
NAVSS0_VIRT_ALIAS_2_TIMERMGR0_CFG_CONFIG	0x4B20E80000	0x4B20E80200	512 B
NAVSS0_VIRT_ALIAS_2_TIMERMGR1_CFG_CONFIG	0x4B20E81000	0x4B20E81200	512 B
NAVSS0_VIRT_ALIAS_2_TIMERMGR0_CFG_OES	0x4B20F00000	0x4B20F01000	4 KB
NAVSS0_VIRT_ALIAS_2_TIMERMGR1_CFG_OES	0x4B20F01000	0x4B20F02000	4 KB
NAVSS0_VIRT_ALIAS_2_IO_PVU0_CFG_MMRS	0x4B20F80000	0x4B20F81000	4 KB
NAVSS0_VIRT_ALIAS_2_IO_PVU1_CFG_MMRS	0x4B20F81000	0x4B20F82000	4 KB
NAVSS0_VIRT_ALIAS_2_PVU0_SRC_TOG_CFG	0x4B20F90000	0x4B20F90400	1 KB
NAVSS0_VIRT_ALIAS_2_PVU0_CFG_TOG_CFG	0x4B20F91000	0x4B20F91400	1 KB
NAVSS0_VIRT_ALIAS_2_ECCAGGR0	0x4B21000000	0x4B21000400	1 KB
NAVSS0_VIRT_ALIAS_2_UDMASS_ECCAGGR_CFG	0x4B21001000	0x4B21001400	1 KB
NAVSS0_VIRT_ALIAS_2_VIRTSS_ECCAGGR_CFG	0x4B21002000	0x4B21002400	1 KB
NAVSS0_VIRT_ALIAS_2_UDMASS_INTA0_CFG_GCNTCFG	0x4B21040000	0x4B21044000	16 KB
NAVSS0_VIRT_ALIAS_2_RINGACC0_CFG	0x4B21080000	0x4B210C0000	256 KB
NAVSS0_VIRT_ALIAS_2_REGS0_CFG_MMRS	0x4B210C0000	0x4B210C0100	256 B
NAVSS0_VIRT_ALIAS_2_CPTS0_S_VBUSP_CPTS_VBUSP	0x4B210D0000	0x4B210D0400	1 KB
NAVSS0_VIRT_ALIAS_2_INTR0_CFG_INTR_ROUTER_CFG	0x4B210E0000	0x4B210E4000	16 KB
NAVSS0_VIRT_ALIAS_2_UDMASS_INTA0_CFG_L2G	0x4B21100000	0x4B21102000	8 KB
NAVSS0_VIRT_ALIAS_2_UDMASS_INTA0_CFG_MCAST	0x4B21110000	0x4B21114000	16 KB
NAVSS0_VIRT_ALIAS_2_PROXY0_CFG_BUF_CFG_GCFG	0x4B21120000	0x4B21120100	256 B
NAVSS0_VIRT_ALIAS_2_PROXY0_CFG_BUFRAM_SLV_RAM	0x4B21130000	0x4B21134000	16 KB
NAVSS0_VIRT_ALIAS_2_SEC_PROXY0_CFG_MMRS	0x4B21140000	0x4B21140100	256 B
NAVSS0_VIRT_ALIAS_2_UDMAP0_CFG_GCFG	0x4B21150000	0x4B21150100	256 B
NAVSS0_VIRT_ALIAS_2_RINGACC0_CFG_GCFG	0x4B21160000	0x4B21160400	1 KB
NAVSS0_VIRT_ALIAS_2_PSILSS0_CFG_MMRS	0x4B21170000	0x4B21171000	4 KB
NAVSS0_VIRT_ALIAS_2_BCDMA0_CFG_GCFG	0x4B211A0000	0x4B211A0100	256 B
NAVSS0_VIRT_ALIAS_2_MCRC0_S_CFG_MCRC64	0x4B21F70000	0x4B21F71000	4 KB
NAVSS0_VIRT_ALIAS_2_PSILCFG0_CFG_PROXY	0x4B21F78000	0x4B21F78200	512 B
NAVSS0_VIRT_ALIAS_2_MAILBOX0_CFG_REGS0	0x4B21F80000	0x4B21F80200	512 B
NAVSS0_VIRT_ALIAS_2_MAILBOX0_CFG_REGS1	0x4B21F81000	0x4B21F81200	512 B
NAVSS0_VIRT_ALIAS_2_MAILBOX0_CFG_REGS2	0x4B21F82000	0x4B21F82200	512 B
NAVSS0_VIRT_ALIAS_2_MAILBOX0_CFG_REGS3	0x4B21F83000	0x4B21F83200	512 B
NAVSS0_VIRT_ALIAS_2_MAILBOX0_CFG_REGS4	0x4B21F84000	0x4B21F84200	512 B
NAVSS0_VIRT_ALIAS_2_MAILBOX0_CFG_REGS5	0x4B21F85000	0x4B21F85200	512 B
NAVSS0_VIRT_ALIAS_2_MAILBOX0_CFG_REGS6	0x4B21F86000	0x4B21F86200	512 B
NAVSS0_VIRT_ALIAS_2_MAILBOX0_CFG_REGS7	0x4B21F87000	0x4B21F87200	512 B

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
NAVSS0_VIRT_ALIAS_2_MAILBOX0_CFG_REGS8	0x4B21F88000	0x4B21F88200	512 B
NAVSS0_VIRT_ALIAS_2_MAILBOX0_CFG_REGS9	0x4B21F89000	0x4B21F89200	512 B
NAVSS0_VIRT_ALIAS_2_MAILBOX0_CFG_REGS10	0x4B21F8A000	0x4B21F8A200	512 B
NAVSS0_VIRT_ALIAS_2_MAILBOX0_CFG_REGS11	0x4B21F8B000	0x4B21F8B200	512 B
NAVSS0_VIRT_ALIAS_2_MAILBOX1_CFG_REGS0	0x4B21F90000	0x4B21F90200	512 B
NAVSS0_VIRT_ALIAS_2_MAILBOX1_CFG_REGS1	0x4B21F91000	0x4B21F91200	512 B
NAVSS0_VIRT_ALIAS_2_MAILBOX1_CFG_REGS2	0x4B21F92000	0x4B21F92200	512 B
NAVSS0_VIRT_ALIAS_2_MAILBOX1_CFG_REGS3	0x4B21F93000	0x4B21F93200	512 B
NAVSS0_VIRT_ALIAS_2_MAILBOX1_CFG_REGS4	0x4B21F94000	0x4B21F94200	512 B
NAVSS0_VIRT_ALIAS_2_MAILBOX1_CFG_REGS5	0x4B21F95000	0x4B21F95200	512 B
NAVSS0_VIRT_ALIAS_2_MAILBOX1_CFG_REGS6	0x4B21F96000	0x4B21F96200	512 B
NAVSS0_VIRT_ALIAS_2_MAILBOX1_CFG_REGS7	0x4B21F97000	0x4B21F97200	512 B
NAVSS0_VIRT_ALIAS_2_MAILBOX1_CFG_REGS8	0x4B21F98000	0x4B21F98200	512 B
NAVSS0_VIRT_ALIAS_2_MAILBOX1_CFG_REGS9	0x4B21F99000	0x4B21F99200	512 B
NAVSS0_VIRT_ALIAS_2_MAILBOX1_CFG_REGS10	0x4B21F9A000	0x4B21F9A200	512 B
NAVSS0_VIRT_ALIAS_2_MAILBOX1_CFG_REGS11	0x4B21F9B000	0x4B21F9B200	512 B
NAVSS0_VIRT_ALIAS_2_RINGACC0_CFG_MON	0x4B22000000	0x4B22020000	128 KB
NAVSS0_VIRT_ALIAS_2_TIMERMGR0_CFG_TIMERS	0x4B22200000	0x4B22240000	256 KB
NAVSS0_VIRT_ALIAS_2_TIMERMGR1_CFG_TIMERS	0x4B22240000	0x4B22280000	256 KB
NAVSS0_VIRT_ALIAS_2_SEC_PROXY0_CFG_RT	0x4B22400000	0x4B22600000	2 MB
NAVSS0_VIRT_ALIAS_2_SEC_PROXY0_CFG_SCFG	0x4B22800000	0x4B22A00000	2 MB
NAVSS0_VIRT_ALIAS_2_SEC_PROXY0_SRC_TARGET_DATA	0x4B22C00000	0x4B22E00000	2 MB
NAVSS0_VIRT_ALIAS_2_PROXY0_SRC_TARGET0_DATA	0x4B23000000	0x4B23040000	256 KB
NAVSS0_VIRT_ALIAS_2_PROXY0_CFG_BUF_CFG	0x4B23400000	0x4B23440000	256 KB
NAVSS0_VIRT_ALIAS_2_UDMASS_INTA0_CFG_GCNTRTI	0x4B23800000	0x4B23A00000	2 MB
NAVSS0_VIRT_ALIAS_2_MODSS_INTA0_CFG_INTR	0x4B23C00000	0x4B23C40000	256 KB
NAVSS0_VIRT_ALIAS_2_MODSS_INTA1_CFG_INTR	0x4B23C40000	0x4B23C80000	256 KB
NAVSS0_VIRT_ALIAS_2_UDMASS_INTA0_CFG_INTR	0x4B23D00000	0x4B23E00000	1 MB
NAVSS0_VIRT_ALIAS_2_UDMAP0_CFG_RCHANRT	0x4B24000000	0x4B24080000	512 KB
NAVSS0_VIRT_ALIAS_2_UDMAP0_CFG_TCHANRT	0x4B25000000	0x4B25200000	2 MB
NAVSS0_VIRT_ALIAS_2_BCDMA0_CFG_TCHAN	0x4B25840000	0x4B25841000	4 KB
NAVSS0_VIRT_ALIAS_2_BCDMA0_CFG_RCHAN	0x4B25880000	0x4B25882000	8 KB
NAVSS0_VIRT_ALIAS_2_BCDMA0_CFG_RING	0x4B25900000	0x4B25904000	16 KB
NAVSS0_VIRT_ALIAS_2_BCDMA0_CFG_TCHANRT	0x4B25C00000	0x4B25C10000	64 KB
NAVSS0_VIRT_ALIAS_2_BCDMA0_CFG_RCHANRT	0x4B25D00000	0x4B25D20000	128 KB
NAVSS0_VIRT_ALIAS_2_BCDMA0_CFG_RINGRT	0x4B25E00000	0x4B25E80000	512 KB
NAVSS0_VIRT_ALIAS_2_IO_PVU0_CFG_TLBIF_TLB	0x4B26000000	0x4B26040000	256 KB
NAVSS0_VIRT_ALIAS_2_IO_PVU1_CFG_TLBIF_TLB	0x4B26040000	0x4B26080000	256 KB
NAVSS0_VIRT_ALIAS_2_RINGACC0_SRC_FIFOS	0x4B28000000	0x4B28400000	4 MB
NAVSS0_VIRT_ALIAS_2_RINGACC0_CFG_RT	0x4B2C000000	0x4B2C400000	4 MB
NAVSS0_VIRT_ALIAS_3_MODSS_INTA0_CFG	0x4B30800000	0x4B30800020	32 B
NAVSS0_VIRT_ALIAS_3_MODSS_INTA1_CFG	0x4B30801000	0x4B30801020	32 B
NAVSS0_VIRT_ALIAS_3_UDMASS_INTA0_CFG	0x4B30802000	0x4B30802020	32 B
NAVSS0_VIRT_ALIAS_3_UDMASS_INTA0_CFG_UNMAP	0x4B30880000	0x4B30890000	64 KB
NAVSS0_VIRT_ALIAS_3_MODSS_INTA0_CFG_IMAP	0x4B30900000	0x4B30902000	8 KB
NAVSS0_VIRT_ALIAS_3_MODSS_INTA1_CFG_IMAP	0x4B30908000	0x4B3090A000	8 KB
NAVSS0_VIRT_ALIAS_3_UDMASS_INTA0_CFG_IMAP	0x4B30940000	0x4B30950000	64 KB

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
NAVSS0_VIRT_ALIAS_3_NAV_DDR0_VIRTID_CFG_MMRS	0x4B30A02000	0x4B30A02100	256 B
NAVSS0_VIRT_ALIAS_3_NAV_DDR1_VIRTID_CFG_MMRS	0x4B30A03000	0x4B30A03100	256 B
NAVSS0_VIRT_ALIAS_3_UDMAP0_CFG_TCHAN	0x4B30B00000	0x4B30B20000	128 KB
NAVSS0_VIRT_ALIAS_3_UDMAP0_CFG_RCHAN	0x4B30C00000	0x4B30C08000	32 KB
NAVSS0_VIRT_ALIAS_3_UDMAP0_CFG_RFLOW	0x4B30D00000	0x4B30D04000	16 KB
NAVSS0_VIRT_ALIAS_3_SPINLOCK0_CFG	0x4B30E00000	0x4B30E08000	32 KB
NAVSS0_VIRT_ALIAS_3_TIMERMGR0_CFG_CONFIG	0x4B30E80000	0x4B30E80200	512 B
NAVSS0_VIRT_ALIAS_3_TIMERMGR1_CFG_CONFIG	0x4B30E81000	0x4B30E81200	512 B
NAVSS0_VIRT_ALIAS_3_TIMERMGR0_CFG_OES	0x4B30F00000	0x4B30F01000	4 KB
NAVSS0_VIRT_ALIAS_3_TIMERMGR1_CFG_OES	0x4B30F01000	0x4B30F02000	4 KB
NAVSS0_VIRT_ALIAS_3_IO_PVU0_CFG_MMRS	0x4B30F80000	0x4B30F81000	4 KB
NAVSS0_VIRT_ALIAS_3_IO_PVU1_CFG_MMRS	0x4B30F81000	0x4B30F82000	4 KB
NAVSS0_VIRT_ALIAS_3_PVU0_SRC_TOG_CFG	0x4B30F90000	0x4B30F90400	1 KB
NAVSS0_VIRT_ALIAS_3_PVU0_CFG_TOG_CFG	0x4B30F91000	0x4B30F91400	1 KB
NAVSS0_VIRT_ALIAS_3_ECCAGGR0	0x4B31000000	0x4B31000400	1 KB
NAVSS0_VIRT_ALIAS_3_UDMASS_ECCAGGR_CFG	0x4B31001000	0x4B31001400	1 KB
NAVSS0_VIRT_ALIAS_3_VIRTSS_ECCAGGR_CFG	0x4B31002000	0x4B31002400	1 KB
NAVSS0_VIRT_ALIAS_3_UDMASS_INTA0_CFG_GCNTCFG	0x4B31040000	0x4B31044000	16 KB
NAVSS0_VIRT_ALIAS_3_RINGACC0_CFG	0x4B31080000	0x4B310C0000	256 KB
NAVSS0_VIRT_ALIAS_3_REGS0_CFG_MMRS	0x4B310C0000	0x4B310C0100	256 B
NAVSS0_VIRT_ALIAS_3_CPTS0_S_VBUSP_CPTS_VBUSP	0x4B310D0000	0x4B310D0400	1 KB
NAVSS0_VIRT_ALIAS_3_INTR0_CFG_INTR_ROUTER_CFG	0x4B310E0000	0x4B310E4000	16 KB
NAVSS0_VIRT_ALIAS_3_UDMASS_INTA0_CFG_L2G	0x4B31100000	0x4B31102000	8 KB
NAVSS0_VIRT_ALIAS_3_UDMASS_INTA0_CFG_MCAST	0x4B31110000	0x4B31114000	16 KB
NAVSS0_VIRT_ALIAS_3_PROXY0_CFG_BUF_CFG_GCFG	0x4B31120000	0x4B31120100	256 B
NAVSS0_VIRT_ALIAS_3_PROXY0_CFG_BUFRAM_SLV_RAM	0x4B31130000	0x4B31134000	16 KB
NAVSS0_VIRT_ALIAS_3_SEC_PROXY0_CFG_MMRS	0x4B31140000	0x4B31140100	256 B
NAVSS0_VIRT_ALIAS_3_UDMAP0_CFG_GCFG	0x4B31150000	0x4B31150100	256 B
NAVSS0_VIRT_ALIAS_3_RINGACC0_CFG_GCFG	0x4B31160000	0x4B31160400	1 KB
NAVSS0_VIRT_ALIAS_3_PSILSS0_CFG_MMRS	0x4B31170000	0x4B31171000	4 KB
NAVSS0_VIRT_ALIAS_3_BCDMA0_CFG_GCFG	0x4B311A0000	0x4B311A0100	256 B
NAVSS0_VIRT_ALIAS_3_MCRC0_S_CFG_MCRC64	0x4B31F70000	0x4B31F71000	4 KB
NAVSS0_VIRT_ALIAS_3_PSILCFG0_CFG_PROXY	0x4B31F78000	0x4B31F78200	512 B
NAVSS0_VIRT_ALIAS_3_MAILBOX0_CFG_REGS0	0x4B31F80000	0x4B31F80200	512 B
NAVSS0_VIRT_ALIAS_3_MAILBOX0_CFG_REGS1	0x4B31F81000	0x4B31F81200	512 B
NAVSS0_VIRT_ALIAS_3_MAILBOX0_CFG_REGS2	0x4B31F82000	0x4B31F82200	512 B
NAVSS0_VIRT_ALIAS_3_MAILBOX0_CFG_REGS3	0x4B31F83000	0x4B31F83200	512 B
NAVSS0_VIRT_ALIAS_3_MAILBOX0_CFG_REGS4	0x4B31F84000	0x4B31F84200	512 B
NAVSS0_VIRT_ALIAS_3_MAILBOX0_CFG_REGS5	0x4B31F85000	0x4B31F85200	512 B
NAVSS0_VIRT_ALIAS_3_MAILBOX0_CFG_REGS6	0x4B31F86000	0x4B31F86200	512 B
NAVSS0_VIRT_ALIAS_3_MAILBOX0_CFG_REGS7	0x4B31F87000	0x4B31F87200	512 B
NAVSS0_VIRT_ALIAS_3_MAILBOX0_CFG_REGS8	0x4B31F88000	0x4B31F88200	512 B
NAVSS0_VIRT_ALIAS_3_MAILBOX0_CFG_REGS9	0x4B31F89000	0x4B31F89200	512 B
NAVSS0_VIRT_ALIAS_3_MAILBOX0_CFG_REGS10	0x4B31F8A000	0x4B31F8A200	512 B
NAVSS0_VIRT_ALIAS_3_MAILBOX0_CFG_REGS11	0x4B31F8B000	0x4B31F8B200	512 B
NAVSS0_VIRT_ALIAS_3_MAILBOX1_CFG_REGS0	0x4B31F90000	0x4B31F90200	512 B
NAVSS0_VIRT_ALIAS_3_MAILBOX1_CFG_REGS1	0x4B31F91000	0x4B31F91200	512 B

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
NAVSS0_VIRT_ALIAS_3_MAILBOX1_CFG_REGS2	0x4B31F92000	0x4B31F92200	512 B
NAVSS0_VIRT_ALIAS_3_MAILBOX1_CFG_REGS3	0x4B31F93000	0x4B31F93200	512 B
NAVSS0_VIRT_ALIAS_3_MAILBOX1_CFG_REGS4	0x4B31F94000	0x4B31F94200	512 B
NAVSS0_VIRT_ALIAS_3_MAILBOX1_CFG_REGS5	0x4B31F95000	0x4B31F95200	512 B
NAVSS0_VIRT_ALIAS_3_MAILBOX1_CFG_REGS6	0x4B31F96000	0x4B31F96200	512 B
NAVSS0_VIRT_ALIAS_3_MAILBOX1_CFG_REGS7	0x4B31F97000	0x4B31F97200	512 B
NAVSS0_VIRT_ALIAS_3_MAILBOX1_CFG_REGS8	0x4B31F98000	0x4B31F98200	512 B
NAVSS0_VIRT_ALIAS_3_MAILBOX1_CFG_REGS9	0x4B31F99000	0x4B31F99200	512 B
NAVSS0_VIRT_ALIAS_3_MAILBOX1_CFG_REGS10	0x4B31F9A000	0x4B31F9A200	512 B
NAVSS0_VIRT_ALIAS_3_MAILBOX1_CFG_REGS11	0x4B31F9B000	0x4B31F9B200	512 B
NAVSS0_VIRT_ALIAS_3_RINGACC0_CFG_MON	0x4B32000000	0x4B32020000	128 KB
NAVSS0_VIRT_ALIAS_3_TIMERMGR0_CFG_TIMERS	0x4B32200000	0x4B32240000	256 KB
NAVSS0_VIRT_ALIAS_3_TIMERMGR1_CFG_TIMERS	0x4B32240000	0x4B32280000	256 KB
NAVSS0_VIRT_ALIAS_3_SEC_PROXY0_CFG_RT	0x4B32400000	0x4B32600000	2 MB
NAVSS0_VIRT_ALIAS_3_SEC_PROXY0_CFG_SCFG	0x4B32800000	0x4B32A00000	2 MB
NAVSS0_VIRT_ALIAS_3_SEC_PROXY0_SRC_TARGET_DATA	0x4B32C00000	0x4B32E00000	2 MB
NAVSS0_VIRT_ALIAS_3_PROXY0_SRC_TARGET0_DATA	0x4B33000000	0x4B33040000	256 KB
NAVSS0_VIRT_ALIAS_3_PROXY0_CFG_BUF_CFG	0x4B33400000	0x4B33440000	256 KB
NAVSS0_VIRT_ALIAS_3_UDMASS_INTA0_CFG_GCNTRTI	0x4B33800000	0x4B33A00000	2 MB
NAVSS0_VIRT_ALIAS_3_MODSS_INTA0_CFG_INTR	0x4B33C00000	0x4B33C40000	256 KB
NAVSS0_VIRT_ALIAS_3_MODSS_INTA1_CFG_INTR	0x4B33C40000	0x4B33C80000	256 KB
NAVSS0_VIRT_ALIAS_3_UDMASS_INTA0_CFG_INTR	0x4B33D00000	0x4B33E00000	1 MB
NAVSS0_VIRT_ALIAS_3_UDMAP0_CFG_RCHANRT	0x4B34000000	0x4B34080000	512 KB
NAVSS0_VIRT_ALIAS_3_UDMAP0_CFG_TCHANRT	0x4B35000000	0x4B35200000	2 MB
NAVSS0_VIRT_ALIAS_3_BCDMA0_CFG_TCHAN	0x4B35840000	0x4B35841000	4 KB
NAVSS0_VIRT_ALIAS_3_BCDMA0_CFG_RCHAN	0x4B35880000	0x4B35882000	8 KB
NAVSS0_VIRT_ALIAS_3_BCDMA0_CFG_RING	0x4B35900000	0x4B35904000	16 KB
NAVSS0_VIRT_ALIAS_3_BCDMA0_CFG_TCHANRT	0x4B35C00000	0x4B35C10000	64 KB
NAVSS0_VIRT_ALIAS_3_BCDMA0_CFG_RCHANRT	0x4B35D00000	0x4B35D20000	128 KB
NAVSS0_VIRT_ALIAS_3_BCDMA0_CFG_RINGRT	0x4B35E00000	0x4B35E80000	512 KB
NAVSS0_VIRT_ALIAS_3_IO_PVU0_CFG_TLBIF_TLB	0x4B36000000	0x4B36040000	256 KB
NAVSS0_VIRT_ALIAS_3_IO_PVU1_CFG_TLBIF_TLB	0x4B36040000	0x4B36080000	256 KB
NAVSS0_VIRT_ALIAS_3_RINGACC0_SRC_FIFOS	0x4B38000000	0x4B38400000	4 MB
NAVSS0_VIRT_ALIAS_3_RINGACC0_CFG_RT	0x4B3C000000	0x4B3C400000	4 MB
NAVSS0_VIRT_ALIAS_4_MODSS_INTA0_CFG	0x4B40800000	0x4B40800020	32 B
NAVSS0_VIRT_ALIAS_4_MODSS_INTA1_CFG	0x4B40801000	0x4B40801020	32 B
NAVSS0_VIRT_ALIAS_4_UDMASS_INTA0_CFG	0x4B40802000	0x4B40802020	32 B
NAVSS0_VIRT_ALIAS_4_UDMASS_INTA0_CFG_UNMAP	0x4B40880000	0x4B40890000	64 KB
NAVSS0_VIRT_ALIAS_4_MODSS_INTA0_CFG_IMAP	0x4B40900000	0x4B40902000	8 KB
NAVSS0_VIRT_ALIAS_4_MODSS_INTA1_CFG_IMAP	0x4B40908000	0x4B4090A000	8 KB
NAVSS0_VIRT_ALIAS_4_UDMASS_INTA0_CFG_IMAP	0x4B40940000	0x4B40950000	64 KB
NAVSS0_VIRT_ALIAS_4_NAV_DDR0_VIRTID_CFG_MMRS	0x4B40A02000	0x4B40A02100	256 B
NAVSS0_VIRT_ALIAS_4_NAV_DDR1_VIRTID_CFG_MMRS	0x4B40A03000	0x4B40A03100	256 B
NAVSS0_VIRT_ALIAS_4_UDMAP0_CFG_TCHAN	0x4B40B00000	0x4B40B20000	128 KB
NAVSS0_VIRT_ALIAS_4_UDMAP0_CFG_RCHAN	0x4B40C00000	0x4B40C08000	32 KB
NAVSS0_VIRT_ALIAS_4_UDMAP0_CFG_RFLOW	0x4B40D00000	0x4B40D04000	16 KB
NAVSS0_VIRT_ALIAS_4_SPINLOCK0_CFG	0x4B40E00000	0x4B40E08000	32 KB

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
NAVSS0_VIRT_ALIAS_4_TIMERMGR0_CFG_CONFIG	0x4B40E80000	0x4B40E80200	512 B
NAVSS0_VIRT_ALIAS_4_TIMERMGR1_CFG_CONFIG	0x4B40E81000	0x4B40E81200	512 B
NAVSS0_VIRT_ALIAS_4_TIMERMGR0_CFG_OES	0x4B40F00000	0x4B40F01000	4 KB
NAVSS0_VIRT_ALIAS_4_TIMERMGR1_CFG_OES	0x4B40F01000	0x4B40F02000	4 KB
NAVSS0_VIRT_ALIAS_4_IO_PVU0_CFG_MMRS	0x4B40F80000	0x4B40F81000	4 KB
NAVSS0_VIRT_ALIAS_4_IO_PVU1_CFG_MMRS	0x4B40F81000	0x4B40F82000	4 KB
NAVSS0_VIRT_ALIAS_4_PVU0_SRC_TOG_CFG	0x4B40F90000	0x4B40F90400	1 KB
NAVSS0_VIRT_ALIAS_4_PVU0_CFG_TOG_CFG	0x4B40F91000	0x4B40F91400	1 KB
NAVSS0_VIRT_ALIAS_4_ECCAGGR0	0x4B41000000	0x4B41000400	1 KB
NAVSS0_VIRT_ALIAS_4_UDMASS_ECCAGGR_CFG	0x4B41001000	0x4B41001400	1 KB
NAVSS0_VIRT_ALIAS_4_VIRTSS_ECCAGGR_CFG	0x4B41002000	0x4B41002400	1 KB
NAVSS0_VIRT_ALIAS_4_UDMASS_INTA0_CFG_GCNTCFG	0x4B41040000	0x4B41044000	16 KB
NAVSS0_VIRT_ALIAS_4_RINGACC0_CFG	0x4B41080000	0x4B410C0000	256 KB
NAVSS0_VIRT_ALIAS_4_REGS0_CFG_MMRS	0x4B410C0000	0x4B410C0100	256 B
NAVSS0_VIRT_ALIAS_4_CPTS0_S_VBUSP_CPTS_VBUSP	0x4B410D0000	0x4B410D0400	1 KB
NAVSS0_VIRT_ALIAS_4_INTR0_CFG_INTR_ROUTER_CFG	0x4B410E0000	0x4B410E4000	16 KB
NAVSS0_VIRT_ALIAS_4_UDMASS_INTA0_CFG_L2G	0x4B41100000	0x4B41102000	8 KB
NAVSS0_VIRT_ALIAS_4_UDMASS_INTA0_CFG_MCAST	0x4B41110000	0x4B41114000	16 KB
NAVSS0_VIRT_ALIAS_4_PROXY0_CFG_BUF_CFG_GCFG	0x4B41120000	0x4B41120100	256 B
NAVSS0_VIRT_ALIAS_4_PROXY0_CFG_BUFRAM_SLV_RAM	0x4B41130000	0x4B41134000	16 KB
NAVSS0_VIRT_ALIAS_4_SEC_PROXY0_CFG_MMRS	0x4B41140000	0x4B41140100	256 B
NAVSS0_VIRT_ALIAS_4_UDMAP0_CFG_GCFG	0x4B41150000	0x4B41150100	256 B
NAVSS0_VIRT_ALIAS_4_RINGACC0_CFG_GCFG	0x4B41160000	0x4B41160400	1 KB
NAVSS0_VIRT_ALIAS_4_PSILSS0_CFG_MMRS	0x4B41170000	0x4B41171000	4 KB
NAVSS0_VIRT_ALIAS_4_BCDMA0_CFG_GCFG	0x4B411A0000	0x4B411A0100	256 B
NAVSS0_VIRT_ALIAS_4_MCRC0_S_CFG_MCRC64	0x4B41F70000	0x4B41F71000	4 KB
NAVSS0_VIRT_ALIAS_4_PSILCFG0_CFG_PROXY	0x4B41F78000	0x4B41F78200	512 B
NAVSS0_VIRT_ALIAS_4_MAILBOX0_CFG_REGS0	0x4B41F80000	0x4B41F80200	512 B
NAVSS0_VIRT_ALIAS_4_MAILBOX0_CFG_REGS1	0x4B41F81000	0x4B41F81200	512 B
NAVSS0_VIRT_ALIAS_4_MAILBOX0_CFG_REGS2	0x4B41F82000	0x4B41F82200	512 B
NAVSS0_VIRT_ALIAS_4_MAILBOX0_CFG_REGS3	0x4B41F83000	0x4B41F83200	512 B
NAVSS0_VIRT_ALIAS_4_MAILBOX0_CFG_REGS4	0x4B41F84000	0x4B41F84200	512 B
NAVSS0_VIRT_ALIAS_4_MAILBOX0_CFG_REGS5	0x4B41F85000	0x4B41F85200	512 B
NAVSS0_VIRT_ALIAS_4_MAILBOX0_CFG_REGS6	0x4B41F86000	0x4B41F86200	512 B
NAVSS0_VIRT_ALIAS_4_MAILBOX0_CFG_REGS7	0x4B41F87000	0x4B41F87200	512 B
NAVSS0_VIRT_ALIAS_4_MAILBOX0_CFG_REGS8	0x4B41F88000	0x4B41F88200	512 B
NAVSS0_VIRT_ALIAS_4_MAILBOX0_CFG_REGS9	0x4B41F89000	0x4B41F89200	512 B
NAVSS0_VIRT_ALIAS_4_MAILBOX0_CFG_REGS10	0x4B41F8A000	0x4B41F8A200	512 B
NAVSS0_VIRT_ALIAS_4_MAILBOX0_CFG_REGS11	0x4B41F8B000	0x4B41F8B200	512 B
NAVSS0_VIRT_ALIAS_4_MAILBOX1_CFG_REGS0	0x4B41F90000	0x4B41F90200	512 B
NAVSS0_VIRT_ALIAS_4_MAILBOX1_CFG_REGS1	0x4B41F91000	0x4B41F91200	512 B
NAVSS0_VIRT_ALIAS_4_MAILBOX1_CFG_REGS2	0x4B41F92000	0x4B41F92200	512 B
NAVSS0_VIRT_ALIAS_4_MAILBOX1_CFG_REGS3	0x4B41F93000	0x4B41F93200	512 B
NAVSS0_VIRT_ALIAS_4_MAILBOX1_CFG_REGS4	0x4B41F94000	0x4B41F94200	512 B
NAVSS0_VIRT_ALIAS_4_MAILBOX1_CFG_REGS5	0x4B41F95000	0x4B41F95200	512 B
NAVSS0_VIRT_ALIAS_4_MAILBOX1_CFG_REGS6	0x4B41F96000	0x4B41F96200	512 B
NAVSS0_VIRT_ALIAS_4_MAILBOX1_CFG_REGS7	0x4B41F97000	0x4B41F97200	512 B

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
NAVSS0_VIRT_ALIAS_4_MAILBOX1_CFG_REGS8	0x4B41F98000	0x4B41F98200	512 B
NAVSS0_VIRT_ALIAS_4_MAILBOX1_CFG_REGS9	0x4B41F99000	0x4B41F99200	512 B
NAVSS0_VIRT_ALIAS_4_MAILBOX1_CFG_REGS10	0x4B41F9A000	0x4B41F9A200	512 B
NAVSS0_VIRT_ALIAS_4_MAILBOX1_CFG_REGS11	0x4B41F9B000	0x4B41F9B200	512 B
NAVSS0_VIRT_ALIAS_4_RINGACC0_CFG_MON	0x4B42000000	0x4B42020000	128 KB
NAVSS0_VIRT_ALIAS_4_TIMERMGR0_CFG_TIMERS	0x4B42200000	0x4B42240000	256 KB
NAVSS0_VIRT_ALIAS_4_TIMERMGR1_CFG_TIMERS	0x4B42240000	0x4B42280000	256 KB
NAVSS0_VIRT_ALIAS_4_SEC_PROXY0_CFG_RT	0x4B42400000	0x4B42600000	2 MB
NAVSS0_VIRT_ALIAS_4_SEC_PROXY0_CFG_SCFG	0x4B42800000	0x4B42A00000	2 MB
NAVSS0_VIRT_ALIAS_4_SEC_PROXY0_SRC_TARGET_DATA	0x4B42C00000	0x4B42E00000	2 MB
NAVSS0_VIRT_ALIAS_4_PROXY0_SRC_TARGET0_DATA	0x4B43000000	0x4B43040000	256 KB
NAVSS0_VIRT_ALIAS_4_PROXY0_CFG_BUF_CFG	0x4B43400000	0x4B43440000	256 KB
NAVSS0_VIRT_ALIAS_4_UDMASS_INTA0_CFG_GCNTRTI	0x4B43800000	0x4B43A00000	2 MB
NAVSS0_VIRT_ALIAS_4_MODSS_INTA0_CFG_INTR	0x4B43C00000	0x4B43C40000	256 KB
NAVSS0_VIRT_ALIAS_4_MODSS_INTA1_CFG_INTR	0x4B43C40000	0x4B43C80000	256 KB
NAVSS0_VIRT_ALIAS_4_UDMASS_INTA0_CFG_INTR	0x4B43D00000	0x4B43E00000	1 MB
NAVSS0_VIRT_ALIAS_4_UDMAP0_CFG_RCHANRT	0x4B44000000	0x4B44080000	512 KB
NAVSS0_VIRT_ALIAS_4_UDMAP0_CFG_TCHANRT	0x4B45000000	0x4B45200000	2 MB
NAVSS0_VIRT_ALIAS_4_BCDMA0_CFG_TCHAN	0x4B45840000	0x4B45841000	4 KB
NAVSS0_VIRT_ALIAS_4_BCDMA0_CFG_RCHAN	0x4B45880000	0x4B45882000	8 KB
NAVSS0_VIRT_ALIAS_4_BCDMA0_CFG_RING	0x4B45900000	0x4B45904000	16 KB
NAVSS0_VIRT_ALIAS_4_BCDMA0_CFG_TCHANRT	0x4B45C00000	0x4B45C10000	64 KB
NAVSS0_VIRT_ALIAS_4_BCDMA0_CFG_RCHANRT	0x4B45D00000	0x4B45D20000	128 KB
NAVSS0_VIRT_ALIAS_4_BCDMA0_CFG_RINGRT	0x4B45E00000	0x4B45E80000	512 KB
NAVSS0_VIRT_ALIAS_4_IO_PVU0_CFG_TLBIF_TLB	0x4B46000000	0x4B46040000	256 KB
NAVSS0_VIRT_ALIAS_4_IO_PVU1_CFG_TLBIF_TLB	0x4B46040000	0x4B46080000	256 KB
NAVSS0_VIRT_ALIAS_4_RINGACC0_SRC_FIFOS	0x4B48000000	0x4B48400000	4 MB
NAVSS0_VIRT_ALIAS_4_RINGACC0_CFG_RT	0x4B4C000000	0x4B4C400000	4 MB
NAVSS0_VIRT_ALIAS_5_MODSS_INTA0_CFG	0x4B50800000	0x4B50800020	32 B
NAVSS0_VIRT_ALIAS_5_MODSS_INTA1_CFG	0x4B50801000	0x4B50801020	32 B
NAVSS0_VIRT_ALIAS_5_UDMASS_INTA0_CFG	0x4B50802000	0x4B50802020	32 B
NAVSS0_VIRT_ALIAS_5_UDMASS_INTA0_CFG_UNMAP	0x4B50880000	0x4B50890000	64 KB
NAVSS0_VIRT_ALIAS_5_MODSS_INTA0_CFG_IMAP	0x4B50900000	0x4B50902000	8 KB
NAVSS0_VIRT_ALIAS_5_MODSS_INTA1_CFG_IMAP	0x4B50908000	0x4B5090A000	8 KB
NAVSS0_VIRT_ALIAS_5_UDMASS_INTA0_CFG_IMAP	0x4B50940000	0x4B50950000	64 KB
NAVSS0_VIRT_ALIAS_5_NAV_DDR0_VIRTID_CFG_MMRS	0x4B50A02000	0x4B50A02100	256 B
NAVSS0_VIRT_ALIAS_5_NAV_DDR1_VIRTID_CFG_MMRS	0x4B50A03000	0x4B50A03100	256 B
NAVSS0_VIRT_ALIAS_5_UDMAP0_CFG_TCHAN	0x4B50B00000	0x4B50B20000	128 KB
NAVSS0_VIRT_ALIAS_5_UDMAP0_CFG_RCHAN	0x4B50C00000	0x4B50C08000	32 KB
NAVSS0_VIRT_ALIAS_5_UDMAP0_CFG_RFLOW	0x4B50D00000	0x4B50D04000	16 KB
NAVSS0_VIRT_ALIAS_5_SPINLOCK0_CFG	0x4B50E00000	0x4B50E08000	32 KB
NAVSS0_VIRT_ALIAS_5_TIMERMGR0_CFG_CONFIG	0x4B50E80000	0x4B50E80200	512 B
NAVSS0_VIRT_ALIAS_5_TIMERMGR1_CFG_CONFIG	0x4B50E81000	0x4B50E81200	512 B
NAVSS0_VIRT_ALIAS_5_TIMERMGR0_CFG_OES	0x4B50F00000	0x4B50F01000	4 KB
NAVSS0_VIRT_ALIAS_5_TIMERMGR1_CFG_OES	0x4B50F01000	0x4B50F02000	4 KB
NAVSS0_VIRT_ALIAS_5_IO_PVU0_CFG_MMRS	0x4B50F80000	0x4B50F81000	4 KB
NAVSS0_VIRT_ALIAS_5_IO_PVU1_CFG_MMRS	0x4B50F81000	0x4B50F82000	4 KB

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
NAVSS0_VIRT_ALIAS_5_PVU0_SRC_TOG_CFG	0x4B50F90000	0x4B50F90400	1 KB
NAVSS0_VIRT_ALIAS_5_PVU0_CFG_TOG_CFG	0x4B50F91000	0x4B50F91400	1 KB
NAVSS0_VIRT_ALIAS_5_ECCAGGR0	0x4B51000000	0x4B51000400	1 KB
NAVSS0_VIRT_ALIAS_5_UDMASS_ECCAGGR_CFG	0x4B51001000	0x4B51001400	1 KB
NAVSS0_VIRT_ALIAS_5_VIRTSS_ECCAGGR_CFG	0x4B51002000	0x4B51002400	1 KB
NAVSS0_VIRT_ALIAS_5_UDMASS_INTA0_CFG_GCNTCFG	0x4B51040000	0x4B51044000	16 KB
NAVSS0_VIRT_ALIAS_5_RINGACC0_CFG	0x4B51080000	0x4B510C0000	256 KB
NAVSS0_VIRT_ALIAS_5_REGS0_CFG_MMRS	0x4B510C0000	0x4B510C0100	256 B
NAVSS0_VIRT_ALIAS_5_CPTS0_S_VBUSP_CPTS_VBUSP	0x4B510D0000	0x4B510D0400	1 KB
NAVSS0_VIRT_ALIAS_5_INTR0_CFG_INTR_ROUTER_CFG	0x4B510E0000	0x4B510E4000	16 KB
NAVSS0_VIRT_ALIAS_5_UDMASS_INTA0_CFG_L2G	0x4B51100000	0x4B51102000	8 KB
NAVSS0_VIRT_ALIAS_5_UDMASS_INTA0_CFG_MCAST	0x4B51110000	0x4B51114000	16 KB
NAVSS0_VIRT_ALIAS_5_PROXY0_CFG_BUF_CFG_GCFG	0x4B51120000	0x4B51120100	256 B
NAVSS0_VIRT_ALIAS_5_PROXY0_CFG_BUFRAM_SLV_RAM	0x4B51130000	0x4B51134000	16 KB
NAVSS0_VIRT_ALIAS_5_SEC_PROXY0_CFG_MMRS	0x4B51140000	0x4B51140100	256 B
NAVSS0_VIRT_ALIAS_5_UDMAP0_CFG_GCFG	0x4B51150000	0x4B51150100	256 B
NAVSS0_VIRT_ALIAS_5_RINGACC0_CFG_GCFG	0x4B51160000	0x4B51160400	1 KB
NAVSS0_VIRT_ALIAS_5_PSILSS0_CFG_MMRS	0x4B51170000	0x4B51171000	4 KB
NAVSS0_VIRT_ALIAS_5_BCDMA0_CFG_GCFG	0x4B511A0000	0x4B511A0100	256 B
NAVSS0_VIRT_ALIAS_5_MCRC0_S_CFG_MCRC64	0x4B51F70000	0x4B51F71000	4 KB
NAVSS0_VIRT_ALIAS_5_PSILCFG0_CFG_PROXY	0x4B51F78000	0x4B51F78200	512 B
NAVSS0_VIRT_ALIAS_5_MAILBOX0_CFG_REGS0	0x4B51F80000	0x4B51F80200	512 B
NAVSS0_VIRT_ALIAS_5_MAILBOX0_CFG_REGS1	0x4B51F81000	0x4B51F81200	512 B
NAVSS0_VIRT_ALIAS_5_MAILBOX0_CFG_REGS2	0x4B51F82000	0x4B51F82200	512 B
NAVSS0_VIRT_ALIAS_5_MAILBOX0_CFG_REGS3	0x4B51F83000	0x4B51F83200	512 B
NAVSS0_VIRT_ALIAS_5_MAILBOX0_CFG_REGS4	0x4B51F84000	0x4B51F84200	512 B
NAVSS0_VIRT_ALIAS_5_MAILBOX0_CFG_REGS5	0x4B51F85000	0x4B51F85200	512 B
NAVSS0_VIRT_ALIAS_5_MAILBOX0_CFG_REGS6	0x4B51F86000	0x4B51F86200	512 B
NAVSS0_VIRT_ALIAS_5_MAILBOX0_CFG_REGS7	0x4B51F87000	0x4B51F87200	512 B
NAVSS0_VIRT_ALIAS_5_MAILBOX0_CFG_REGS8	0x4B51F88000	0x4B51F88200	512 B
NAVSS0_VIRT_ALIAS_5_MAILBOX0_CFG_REGS9	0x4B51F89000	0x4B51F89200	512 B
NAVSS0_VIRT_ALIAS_5_MAILBOX0_CFG_REGS10	0x4B51F8A000	0x4B51F8A200	512 B
NAVSS0_VIRT_ALIAS_5_MAILBOX0_CFG_REGS11	0x4B51F8B000	0x4B51F8B200	512 B
NAVSS0_VIRT_ALIAS_5_MAILBOX1_CFG_REGS0	0x4B51F90000	0x4B51F90200	512 B
NAVSS0_VIRT_ALIAS_5_MAILBOX1_CFG_REGS1	0x4B51F91000	0x4B51F91200	512 B
NAVSS0_VIRT_ALIAS_5_MAILBOX1_CFG_REGS2	0x4B51F92000	0x4B51F92200	512 B
NAVSS0_VIRT_ALIAS_5_MAILBOX1_CFG_REGS3	0x4B51F93000	0x4B51F93200	512 B
NAVSS0_VIRT_ALIAS_5_MAILBOX1_CFG_REGS4	0x4B51F94000	0x4B51F94200	512 B
NAVSS0_VIRT_ALIAS_5_MAILBOX1_CFG_REGS5	0x4B51F95000	0x4B51F95200	512 B
NAVSS0_VIRT_ALIAS_5_MAILBOX1_CFG_REGS6	0x4B51F96000	0x4B51F96200	512 B
NAVSS0_VIRT_ALIAS_5_MAILBOX1_CFG_REGS7	0x4B51F97000	0x4B51F97200	512 B
NAVSS0_VIRT_ALIAS_5_MAILBOX1_CFG_REGS8	0x4B51F98000	0x4B51F98200	512 B
NAVSS0_VIRT_ALIAS_5_MAILBOX1_CFG_REGS9	0x4B51F99000	0x4B51F99200	512 B
NAVSS0_VIRT_ALIAS_5_MAILBOX1_CFG_REGS10	0x4B51F9A000	0x4B51F9A200	512 B
NAVSS0_VIRT_ALIAS_5_MAILBOX1_CFG_REGS11	0x4B51F9B000	0x4B51F9B200	512 B
NAVSS0_VIRT_ALIAS_5_RINGACC0_CFG_MON	0x4B52000000	0x4B52020000	128 KB
NAVSS0_VIRT_ALIAS_5_TIMERMGR0_CFG_TIMERS	0x4B52200000	0x4B52240000	256 KB

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
NAVSS0_VIRT_ALIAS_5_TIMERMGR1_CFG_TIMERS	0x4B52240000	0x4B52280000	256 KB
NAVSS0_VIRT_ALIAS_5_SEC_PROXY0_CFG_RT	0x4B52400000	0x4B52600000	2 MB
NAVSS0_VIRT_ALIAS_5_SEC_PROXY0_CFG_SCFG	0x4B52800000	0x4B52A00000	2 MB
NAVSS0_VIRT_ALIAS_5_SEC_PROXY0_SRC_TARGET_DATA	0x4B52C00000	0x4B52E00000	2 MB
NAVSS0_VIRT_ALIAS_5_PROXY0_SRC_TARGET0_DATA	0x4B53000000	0x4B53040000	256 KB
NAVSS0_VIRT_ALIAS_5_PROXY0_CFG_BUF_CFG	0x4B53400000	0x4B53440000	256 KB
NAVSS0_VIRT_ALIAS_5_UDMASS_INTA0_CFG_GCNTRTI	0x4B53800000	0x4B53A00000	2 MB
NAVSS0_VIRT_ALIAS_5_MODSS_INTA0_CFG_INTR	0x4B53C00000	0x4B53C40000	256 KB
NAVSS0_VIRT_ALIAS_5_MODSS_INTA1_CFG_INTR	0x4B53C40000	0x4B53C80000	256 KB
NAVSS0_VIRT_ALIAS_5_UDMASS_INTA0_CFG_INTR	0x4B53D00000	0x4B53E00000	1 MB
NAVSS0_VIRT_ALIAS_5_UDMAP0_CFG_RCHANRT	0x4B54000000	0x4B54080000	512 KB
NAVSS0_VIRT_ALIAS_5_UDMAP0_CFG_TCHANRT	0x4B55000000	0x4B55200000	2 MB
NAVSS0_VIRT_ALIAS_5_BCDMA0_CFG_TCHAN	0x4B55840000	0x4B55841000	4 KB
NAVSS0_VIRT_ALIAS_5_BCDMA0_CFG_RCHAN	0x4B55880000	0x4B55882000	8 KB
NAVSS0_VIRT_ALIAS_5_BCDMA0_CFG_RING	0x4B55900000	0x4B55904000	16 KB
NAVSS0_VIRT_ALIAS_5_BCDMA0_CFG_TCHANRT	0x4B55C00000	0x4B55C10000	64 KB
NAVSS0_VIRT_ALIAS_5_BCDMA0_CFG_RCHANRT	0x4B55D00000	0x4B55D20000	128 KB
NAVSS0_VIRT_ALIAS_5_BCDMA0_CFG_RINGRT	0x4B55E00000	0x4B55E80000	512 KB
NAVSS0_VIRT_ALIAS_5_IO_PVU0_CFG_TLBIF_TLB	0x4B56000000	0x4B56040000	256 KB
NAVSS0_VIRT_ALIAS_5_IO_PVU1_CFG_TLBIF_TLB	0x4B56040000	0x4B56080000	256 KB
NAVSS0_VIRT_ALIAS_5_RINGACC0_SRC_FIFOS	0x4B58000000	0x4B58400000	4 MB
NAVSS0_VIRT_ALIAS_5_RINGACC0_CFG_RT	0x4B5C000000	0x4B5C400000	4 MB
NAVSS0_VIRT_ALIAS_6_MODSS_INTA0_CFG	0x4B60800000	0x4B60800020	32 B
NAVSS0_VIRT_ALIAS_6_MODSS_INTA1_CFG	0x4B60801000	0x4B60801020	32 B
NAVSS0_VIRT_ALIAS_6_UDMASS_INTA0_CFG	0x4B60802000	0x4B60802020	32 B
NAVSS0_VIRT_ALIAS_6_UDMASS_INTA0_CFG_UNMAP	0x4B60880000	0x4B60890000	64 KB
NAVSS0_VIRT_ALIAS_6_MODSS_INTA0_CFG_IMAP	0x4B60900000	0x4B60902000	8 KB
NAVSS0_VIRT_ALIAS_6_MODSS_INTA1_CFG_IMAP	0x4B60908000	0x4B6090A000	8 KB
NAVSS0_VIRT_ALIAS_6_UDMASS_INTA0_CFG_IMAP	0x4B60940000	0x4B60950000	64 KB
NAVSS0_VIRT_ALIAS_6_NAV_DDR0_VIRTID_CFG_MMRS	0x4B60A02000	0x4B60A02100	256 B
NAVSS0_VIRT_ALIAS_6_NAV_DDR1_VIRTID_CFG_MMRS	0x4B60A03000	0x4B60A03100	256 B
NAVSS0_VIRT_ALIAS_6_UDMAP0_CFG_TCHAN	0x4B60B00000	0x4B60B20000	128 KB
NAVSS0_VIRT_ALIAS_6_UDMAP0_CFG_RCHAN	0x4B60C00000	0x4B60C08000	32 KB
NAVSS0_VIRT_ALIAS_6_UDMAP0_CFG_RFLOW	0x4B60D00000	0x4B60D04000	16 KB
NAVSS0_VIRT_ALIAS_6_SPINLOCK0_CFG	0x4B60E00000	0x4B60E08000	32 KB
NAVSS0_VIRT_ALIAS_6_TIMERMGR0_CFG_CONFIG	0x4B60E80000	0x4B60E80200	512 B
NAVSS0_VIRT_ALIAS_6_TIMERMGR1_CFG_CONFIG	0x4B60E81000	0x4B60E81200	512 B
NAVSS0_VIRT_ALIAS_6_TIMERMGR0_CFG_OES	0x4B60F00000	0x4B60F01000	4 KB
NAVSS0_VIRT_ALIAS_6_TIMERMGR1_CFG_OES	0x4B60F01000	0x4B60F02000	4 KB
NAVSS0_VIRT_ALIAS_6_IO_PVU0_CFG_MMRS	0x4B60F80000	0x4B60F81000	4 KB
NAVSS0_VIRT_ALIAS_6_IO_PVU1_CFG_MMRS	0x4B60F81000	0x4B60F82000	4 KB
NAVSS0_VIRT_ALIAS_6_PVU0_SRC_TOG_CFG	0x4B60F90000	0x4B60F90400	1 KB
NAVSS0_VIRT_ALIAS_6_PVU0_CFG_TOG_CFG	0x4B60F91000	0x4B60F91400	1 KB
NAVSS0_VIRT_ALIAS_6_ECCAGGR0	0x4B61000000	0x4B61000400	1 KB
NAVSS0_VIRT_ALIAS_6_UDMASS_ECCAGGR_CFG	0x4B61001000	0x4B61001400	1 KB
NAVSS0_VIRT_ALIAS_6_VIRTSS_ECCAGGR_CFG	0x4B61002000	0x4B61002400	1 KB
NAVSS0_VIRT_ALIAS_6_UDMASS_INTA0_CFG_GCNTCFG	0x4B61040000	0x4B61044000	16 KB

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
NAVSS0_VIRT_ALIAS_6_RINGACC0_CFG	0x4B61080000	0x4B610C0000	256 KB
NAVSS0_VIRT_ALIAS_6_REGS0_CFG_MMRS	0x4B610C0000	0x4B610C0100	256 B
NAVSS0_VIRT_ALIAS_6_CPTS0_S_VBUSP_CPTS_VBUSP	0x4B610D0000	0x4B610D0400	1 KB
NAVSS0_VIRT_ALIAS_6_INTR0_CFG_INTR_ROUTER_CFG	0x4B610E0000	0x4B610E4000	16 KB
NAVSS0_VIRT_ALIAS_6_UDMASS_INTA0_CFG_L2G	0x4B61100000	0x4B61102000	8 KB
NAVSS0_VIRT_ALIAS_6_UDMASS_INTA0_CFG_MCAST	0x4B61110000	0x4B61114000	16 KB
NAVSS0_VIRT_ALIAS_6_PROXY0_CFG_BUF_CFG_GCFG	0x4B61120000	0x4B61120100	256 B
NAVSS0_VIRT_ALIAS_6_PROXY0_CFG_BUFRAM_SLV_RAM	0x4B61130000	0x4B61134000	16 KB
NAVSS0_VIRT_ALIAS_6_SEC_PROXY0_CFG_MMRS	0x4B61140000	0x4B61140100	256 B
NAVSS0_VIRT_ALIAS_6_UDMAP0_CFG_GCFG	0x4B61150000	0x4B61150100	256 B
NAVSS0_VIRT_ALIAS_6_RINGACC0_CFG_GCFG	0x4B61160000	0x4B61160400	1 KB
NAVSS0_VIRT_ALIAS_6_PSILSS0_CFG_MMRS	0x4B61170000	0x4B61171000	4 KB
NAVSS0_VIRT_ALIAS_6_BCDMA0_CFG_GCFG	0x4B611A0000	0x4B611A0100	256 B
NAVSS0_VIRT_ALIAS_6_MCRC0_S_CFG_MCRC64	0x4B61F70000	0x4B61F71000	4 KB
NAVSS0_VIRT_ALIAS_6_PSILCFG0_CFG_PROXY	0x4B61F78000	0x4B61F78200	512 B
NAVSS0_VIRT_ALIAS_6_MAILBOX0_CFG_REGS0	0x4B61F80000	0x4B61F80200	512 B
NAVSS0_VIRT_ALIAS_6_MAILBOX0_CFG_REGS1	0x4B61F81000	0x4B61F81200	512 B
NAVSS0_VIRT_ALIAS_6_MAILBOX0_CFG_REGS2	0x4B61F82000	0x4B61F82200	512 B
NAVSS0_VIRT_ALIAS_6_MAILBOX0_CFG_REGS3	0x4B61F83000	0x4B61F83200	512 B
NAVSS0_VIRT_ALIAS_6_MAILBOX0_CFG_REGS4	0x4B61F84000	0x4B61F84200	512 B
NAVSS0_VIRT_ALIAS_6_MAILBOX0_CFG_REGS5	0x4B61F85000	0x4B61F85200	512 B
NAVSS0_VIRT_ALIAS_6_MAILBOX0_CFG_REGS6	0x4B61F86000	0x4B61F86200	512 B
NAVSS0_VIRT_ALIAS_6_MAILBOX0_CFG_REGS7	0x4B61F87000	0x4B61F87200	512 B
NAVSS0_VIRT_ALIAS_6_MAILBOX0_CFG_REGS8	0x4B61F88000	0x4B61F88200	512 B
NAVSS0_VIRT_ALIAS_6_MAILBOX0_CFG_REGS9	0x4B61F89000	0x4B61F89200	512 B
NAVSS0_VIRT_ALIAS_6_MAILBOX0_CFG_REGS10	0x4B61F8A000	0x4B61F8A200	512 B
NAVSS0_VIRT_ALIAS_6_MAILBOX0_CFG_REGS11	0x4B61F8B000	0x4B61F8B200	512 B
NAVSS0_VIRT_ALIAS_6_MAILBOX1_CFG_REGS0	0x4B61F90000	0x4B61F90200	512 B
NAVSS0_VIRT_ALIAS_6_MAILBOX1_CFG_REGS1	0x4B61F91000	0x4B61F91200	512 B
NAVSS0_VIRT_ALIAS_6_MAILBOX1_CFG_REGS2	0x4B61F92000	0x4B61F92200	512 B
NAVSS0_VIRT_ALIAS_6_MAILBOX1_CFG_REGS3	0x4B61F93000	0x4B61F93200	512 B
NAVSS0_VIRT_ALIAS_6_MAILBOX1_CFG_REGS4	0x4B61F94000	0x4B61F94200	512 B
NAVSS0_VIRT_ALIAS_6_MAILBOX1_CFG_REGS5	0x4B61F95000	0x4B61F95200	512 B
NAVSS0_VIRT_ALIAS_6_MAILBOX1_CFG_REGS6	0x4B61F96000	0x4B61F96200	512 B
NAVSS0_VIRT_ALIAS_6_MAILBOX1_CFG_REGS7	0x4B61F97000	0x4B61F97200	512 B
NAVSS0_VIRT_ALIAS_6_MAILBOX1_CFG_REGS8	0x4B61F98000	0x4B61F98200	512 B
NAVSS0_VIRT_ALIAS_6_MAILBOX1_CFG_REGS9	0x4B61F99000	0x4B61F99200	512 B
NAVSS0_VIRT_ALIAS_6_MAILBOX1_CFG_REGS10	0x4B61F9A000	0x4B61F9A200	512 B
NAVSS0_VIRT_ALIAS_6_MAILBOX1_CFG_REGS11	0x4B61F9B000	0x4B61F9B200	512 B
NAVSS0_VIRT_ALIAS_6_RINGACC0_CFG_MON	0x4B62000000	0x4B62020000	128 KB
NAVSS0_VIRT_ALIAS_6_TIMERMGR0_CFG_TIMERS	0x4B62200000	0x4B62240000	256 KB
NAVSS0_VIRT_ALIAS_6_TIMERMGR1_CFG_TIMERS	0x4B62240000	0x4B62280000	256 KB
NAVSS0_VIRT_ALIAS_6_SEC_PROXY0_CFG_RT	0x4B62400000	0x4B62600000	2 MB
NAVSS0_VIRT_ALIAS_6_SEC_PROXY0_CFG_SCFG	0x4B62800000	0x4B62A00000	2 MB
NAVSS0_VIRT_ALIAS_6_SEC_PROXY0_SRC_TARGET_DATA	0x4B62C00000	0x4B62E00000	2 MB
NAVSS0_VIRT_ALIAS_6_PROXY0_SRC_TARGET0_DATA	0x4B63000000	0x4B63040000	256 KB
NAVSS0_VIRT_ALIAS_6_PROXY0_CFG_BUF_CFG	0x4B63400000	0x4B63440000	256 KB

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
NAVSS0_VIRT_ALIAS_6_UDMASS_INTA0_CFG_GCNTRTI	0x4B63800000	0x4B63A00000	2 MB
NAVSS0_VIRT_ALIAS_6_MODSS_INTA0_CFG_INTR	0x4B63C00000	0x4B63C40000	256 KB
NAVSS0_VIRT_ALIAS_6_MODSS_INTA1_CFG_INTR	0x4B63C40000	0x4B63C80000	256 KB
NAVSS0_VIRT_ALIAS_6_UDMASS_INTA0_CFG_INTR	0x4B63D00000	0x4B63E00000	1 MB
NAVSS0_VIRT_ALIAS_6_UDMAP0_CFG_RCHANRT	0x4B64000000	0x4B64080000	512 KB
NAVSS0_VIRT_ALIAS_6_UDMAP0_CFG_TCHANRT	0x4B65000000	0x4B65200000	2 MB
NAVSS0_VIRT_ALIAS_6_BCDMA0_CFG_TCHAN	0x4B65840000	0x4B65841000	4 KB
NAVSS0_VIRT_ALIAS_6_BCDMA0_CFG_RCHAN	0x4B65880000	0x4B65882000	8 KB
NAVSS0_VIRT_ALIAS_6_BCDMA0_CFG_RING	0x4B65900000	0x4B65904000	16 KB
NAVSS0_VIRT_ALIAS_6_BCDMA0_CFG_TCHANRT	0x4B65C00000	0x4B65C10000	64 KB
NAVSS0_VIRT_ALIAS_6_BCDMA0_CFG_RCHANRT	0x4B65D00000	0x4B65D20000	128 KB
NAVSS0_VIRT_ALIAS_6_BCDMA0_CFG_RINGRT	0x4B65E00000	0x4B65E80000	512 KB
NAVSS0_VIRT_ALIAS_6_IO_PVU0_CFG_TLBIF_TLB	0x4B66000000	0x4B66040000	256 KB
NAVSS0_VIRT_ALIAS_6_IO_PVU1_CFG_TLBIF_TLB	0x4B66040000	0x4B66080000	256 KB
NAVSS0_VIRT_ALIAS_6_RINGACC0_SRC_FIFOS	0x4B68000000	0x4B68400000	4 MB
NAVSS0_VIRT_ALIAS_6_RINGACC0_CFG_RT	0x4B6C000000	0x4B6C400000	4 MB
NAVSS0_VIRT_ALIAS_7_MODSS_INTA0_CFG	0x4B70800000	0x4B70800020	32 B
NAVSS0_VIRT_ALIAS_7_MODSS_INTA1_CFG	0x4B70801000	0x4B70801020	32 B
NAVSS0_VIRT_ALIAS_7_UDMASS_INTA0_CFG	0x4B70802000	0x4B70802020	32 B
NAVSS0_VIRT_ALIAS_7_UDMASS_INTA0_CFG_UNMAP	0x4B70880000	0x4B70890000	64 KB
NAVSS0_VIRT_ALIAS_7_MODSS_INTA0_CFG_IMAP	0x4B70900000	0x4B70902000	8 KB
NAVSS0_VIRT_ALIAS_7_MODSS_INTA1_CFG_IMAP	0x4B70908000	0x4B7090A000	8 KB
NAVSS0_VIRT_ALIAS_7_UDMASS_INTA0_CFG_IMAP	0x4B70940000	0x4B70950000	64 KB
NAVSS0_VIRT_ALIAS_7_NAV_DDR0_VIRTID_CFG_MMRS	0x4B70A02000	0x4B70A02100	256 B
NAVSS0_VIRT_ALIAS_7_NAV_DDR1_VIRTID_CFG_MMRS	0x4B70A03000	0x4B70A03100	256 B
NAVSS0_VIRT_ALIAS_7_UDMAP0_CFG_TCHAN	0x4B70B00000	0x4B70B20000	128 KB
NAVSS0_VIRT_ALIAS_7_UDMAP0_CFG_RCHAN	0x4B70C00000	0x4B70C08000	32 KB
NAVSS0_VIRT_ALIAS_7_UDMAP0_CFG_RFLOW	0x4B70D00000	0x4B70D04000	16 KB
NAVSS0_VIRT_ALIAS_7_SPINLOCK0_CFG	0x4B70E00000	0x4B70E08000	32 KB
NAVSS0_VIRT_ALIAS_7_TIMERMGR0_CFG_CONFIG	0x4B70E80000	0x4B70E80200	512 B
NAVSS0_VIRT_ALIAS_7_TIMERMGR1_CFG_CONFIG	0x4B70E81000	0x4B70E81200	512 B
NAVSS0_VIRT_ALIAS_7_TIMERMGR0_CFG_OES	0x4B70F00000	0x4B70F01000	4 KB
NAVSS0_VIRT_ALIAS_7_TIMERMGR1_CFG_OES	0x4B70F01000	0x4B70F02000	4 KB
NAVSS0_VIRT_ALIAS_7_IO_PVU0_CFG_MMRS	0x4B70F80000	0x4B70F81000	4 KB
NAVSS0_VIRT_ALIAS_7_IO_PVU1_CFG_MMRS	0x4B70F81000	0x4B70F82000	4 KB
NAVSS0_VIRT_ALIAS_7_PVU0_SRC_TOG_CFG	0x4B70F90000	0x4B70F90400	1 KB
NAVSS0_VIRT_ALIAS_7_PVU0_CFG_TOG_CFG	0x4B70F91000	0x4B70F91400	1 KB
NAVSS0_VIRT_ALIAS_7_ECCAGGR0	0x4B71000000	0x4B71000400	1 KB
NAVSS0_VIRT_ALIAS_7_UDMASS_ECCAGGR_CFG	0x4B71001000	0x4B71001400	1 KB
NAVSS0_VIRT_ALIAS_7_VIRTSS_ECCAGGR_CFG	0x4B71002000	0x4B71002400	1 KB
NAVSS0_VIRT_ALIAS_7_UDMASS_INTA0_CFG_GCNTCFG	0x4B71040000	0x4B71044000	16 KB
NAVSS0_VIRT_ALIAS_7_RINGACC0_CFG	0x4B71080000	0x4B710C0000	256 KB
NAVSS0_VIRT_ALIAS_7_REGS0_CFG_MMRS	0x4B710C0000	0x4B710C0100	256 B
NAVSS0_VIRT_ALIAS_7_CPTS0_S_VBUSP_CPTS_VBUSP	0x4B710D0000	0x4B710D0400	1 KB
NAVSS0_VIRT_ALIAS_7_INTR0_CFG_INTR_ROUTER_CFG	0x4B710E0000	0x4B710E4000	16 KB
NAVSS0_VIRT_ALIAS_7_UDMASS_INTA0_CFG_L2G	0x4B71100000	0x4B71102000	8 KB
NAVSS0_VIRT_ALIAS_7_UDMASS_INTA0_CFG_MCAST	0x4B71110000	0x4B71114000	16 KB

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
NAVSS0_VIRT_ALIAS_7_PROXY0_CFG_BUF_CFG_GCFG	0x4B71120000	0x4B71120100	256 B
NAVSS0_VIRT_ALIAS_7_PROXY0_CFG_BUFRAM_SLV_RAM	0x4B71130000	0x4B71134000	16 KB
NAVSS0_VIRT_ALIAS_7_SEC_PROXY0_CFG_MMRS	0x4B71140000	0x4B71140100	256 B
NAVSS0_VIRT_ALIAS_7_UDMAP0_CFG_GCFG	0x4B71150000	0x4B71150100	256 B
NAVSS0_VIRT_ALIAS_7_RINGACC0_CFG_GCFG	0x4B71160000	0x4B71160400	1 KB
NAVSS0_VIRT_ALIAS_7_PSILSS0_CFG_MMRS	0x4B71170000	0x4B71171000	4 KB
NAVSS0_VIRT_ALIAS_7_BCDMA0_CFG_GCFG	0x4B711A0000	0x4B711A0100	256 B
NAVSS0_VIRT_ALIAS_7_MCRC0_S_CFG_MCRC64	0x4B71F70000	0x4B71F71000	4 KB
NAVSS0_VIRT_ALIAS_7_PSILCFG0_CFG_PROXY	0x4B71F78000	0x4B71F78200	512 B
NAVSS0_VIRT_ALIAS_7_MAILBOX0_CFG_REGS0	0x4B71F80000	0x4B71F80200	512 B
NAVSS0_VIRT_ALIAS_7_MAILBOX0_CFG_REGS1	0x4B71F81000	0x4B71F81200	512 B
NAVSS0_VIRT_ALIAS_7_MAILBOX0_CFG_REGS2	0x4B71F82000	0x4B71F82200	512 B
NAVSS0_VIRT_ALIAS_7_MAILBOX0_CFG_REGS3	0x4B71F83000	0x4B71F83200	512 B
NAVSS0_VIRT_ALIAS_7_MAILBOX0_CFG_REGS4	0x4B71F84000	0x4B71F84200	512 B
NAVSS0_VIRT_ALIAS_7_MAILBOX0_CFG_REGS5	0x4B71F85000	0x4B71F85200	512 B
NAVSS0_VIRT_ALIAS_7_MAILBOX0_CFG_REGS6	0x4B71F86000	0x4B71F86200	512 B
NAVSS0_VIRT_ALIAS_7_MAILBOX0_CFG_REGS7	0x4B71F87000	0x4B71F87200	512 B
NAVSS0_VIRT_ALIAS_7_MAILBOX0_CFG_REGS8	0x4B71F88000	0x4B71F88200	512 B
NAVSS0_VIRT_ALIAS_7_MAILBOX0_CFG_REGS9	0x4B71F89000	0x4B71F89200	512 B
NAVSS0_VIRT_ALIAS_7_MAILBOX0_CFG_REGS10	0x4B71F8A000	0x4B71F8A200	512 B
NAVSS0_VIRT_ALIAS_7_MAILBOX0_CFG_REGS11	0x4B71F8B000	0x4B71F8B200	512 B
NAVSS0_VIRT_ALIAS_7_MAILBOX1_CFG_REGS0	0x4B71F90000	0x4B71F90200	512 B
NAVSS0_VIRT_ALIAS_7_MAILBOX1_CFG_REGS1	0x4B71F91000	0x4B71F91200	512 B
NAVSS0_VIRT_ALIAS_7_MAILBOX1_CFG_REGS2	0x4B71F92000	0x4B71F92200	512 B
NAVSS0_VIRT_ALIAS_7_MAILBOX1_CFG_REGS3	0x4B71F93000	0x4B71F93200	512 B
NAVSS0_VIRT_ALIAS_7_MAILBOX1_CFG_REGS4	0x4B71F94000	0x4B71F94200	512 B
NAVSS0_VIRT_ALIAS_7_MAILBOX1_CFG_REGS5	0x4B71F95000	0x4B71F95200	512 B
NAVSS0_VIRT_ALIAS_7_MAILBOX1_CFG_REGS6	0x4B71F96000	0x4B71F96200	512 B
NAVSS0_VIRT_ALIAS_7_MAILBOX1_CFG_REGS7	0x4B71F97000	0x4B71F97200	512 B
NAVSS0_VIRT_ALIAS_7_MAILBOX1_CFG_REGS8	0x4B71F98000	0x4B71F98200	512 B
NAVSS0_VIRT_ALIAS_7_MAILBOX1_CFG_REGS9	0x4B71F99000	0x4B71F99200	512 B
NAVSS0_VIRT_ALIAS_7_MAILBOX1_CFG_REGS10	0x4B71F9A000	0x4B71F9A200	512 B
NAVSS0_VIRT_ALIAS_7_MAILBOX1_CFG_REGS11	0x4B71F9B000	0x4B71F9B200	512 B
NAVSS0_VIRT_ALIAS_7_RINGACC0_CFG_MON	0x4B72000000	0x4B72020000	128 KB
NAVSS0_VIRT_ALIAS_7_TIMERMGR0_CFG_TIMERS	0x4B72200000	0x4B72240000	256 KB
NAVSS0_VIRT_ALIAS_7_TIMERMGR1_CFG_TIMERS	0x4B72240000	0x4B72280000	256 KB
NAVSS0_VIRT_ALIAS_7_SEC_PROXY0_CFG_RT	0x4B72400000	0x4B72600000	2 MB
NAVSS0_VIRT_ALIAS_7_SEC_PROXY0_CFG_SCFG	0x4B72800000	0x4B72A00000	2 MB
NAVSS0_VIRT_ALIAS_7_SEC_PROXY0_SRC_TARGET_DATA	0x4B72C00000	0x4B72E00000	2 MB
NAVSS0_VIRT_ALIAS_7_PROXY0_SRC_TARGET0_DATA	0x4B73000000	0x4B73040000	256 KB
NAVSS0_VIRT_ALIAS_7_PROXY0_CFG_BUF_CFG	0x4B73400000	0x4B73440000	256 KB
NAVSS0_VIRT_ALIAS_7_UDMASS_INTA0_CFG_GCNTRTI	0x4B73800000	0x4B73A00000	2 MB
NAVSS0_VIRT_ALIAS_7_MODSS_INTA0_CFG_INTR	0x4B73C00000	0x4B73C40000	256 KB
NAVSS0_VIRT_ALIAS_7_MODSS_INTA1_CFG_INTR	0x4B73C40000	0x4B73C80000	256 KB
NAVSS0_VIRT_ALIAS_7_UDMASS_INTA0_CFG_INTR	0x4B73D00000	0x4B73E00000	1 MB
NAVSS0_VIRT_ALIAS_7_UDMAP0_CFG_RCHANRT	0x4B74000000	0x4B74080000	512 KB
NAVSS0_VIRT_ALIAS_7_UDMAP0_CFG_TCHANRT	0x4B75000000	0x4B75200000	2 MB

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
NAVSS0_VIRT_ALIAS_7_BCDMA0_CFG_TCHAN	0x4B75840000	0x4B75841000	4 KB
NAVSS0_VIRT_ALIAS_7_BCDMA0_CFG_RCHAN	0x4B75880000	0x4B75882000	8 KB
NAVSS0_VIRT_ALIAS_7_BCDMA0_CFG_RING	0x4B75900000	0x4B75904000	16 KB
NAVSS0_VIRT_ALIAS_7_BCDMA0_CFG_TCHANRT	0x4B75C00000	0x4B75C10000	64 KB
NAVSS0_VIRT_ALIAS_7_BCDMA0_CFG_RCHANRT	0x4B75D00000	0x4B75D20000	128 KB
NAVSS0_VIRT_ALIAS_7_BCDMA0_CFG_RINGRT	0x4B75E00000	0x4B75E80000	512 KB
NAVSS0_VIRT_ALIAS_7_IO_PVU0_CFG_TLBIF_TLB	0x4B76000000	0x4B76040000	256 KB
NAVSS0_VIRT_ALIAS_7_IO_PVU1_CFG_TLBIF_TLB	0x4B76040000	0x4B76080000	256 KB
NAVSS0_VIRT_ALIAS_7_RINGACC0_SRC_FIFOS	0x4B78000000	0x4B78400000	4 MB
NAVSS0_VIRT_ALIAS_7_RINGACC0_CFG_RT	0x4B7C000000	0x4B7C400000	4 MB
NAVSS0_VIRT_ALIAS_8_MODSS_INTA0_CFG	0x4B80800000	0x4B80800020	32 B
NAVSS0_VIRT_ALIAS_8_MODSS_INTA1_CFG	0x4B80801000	0x4B80801020	32 B
NAVSS0_VIRT_ALIAS_8_UDMASS_INTA0_CFG	0x4B80802000	0x4B80802020	32 B
NAVSS0_VIRT_ALIAS_8_UDMASS_INTA0_CFG_UNMAP	0x4B80880000	0x4B80890000	64 KB
NAVSS0_VIRT_ALIAS_8_MODSS_INTA0_CFG_IMAP	0x4B80900000	0x4B80902000	8 KB
NAVSS0_VIRT_ALIAS_8_MODSS_INTA1_CFG_IMAP	0x4B80908000	0x4B8090A000	8 KB
NAVSS0_VIRT_ALIAS_8_UDMASS_INTA0_CFG_IMAP	0x4B80940000	0x4B80950000	64 KB
NAVSS0_VIRT_ALIAS_8_NAV_DDR0_VIRTID_CFG_MMRS	0x4B80A02000	0x4B80A02100	256 B
NAVSS0_VIRT_ALIAS_8_NAV_DDR1_VIRTID_CFG_MMRS	0x4B80A03000	0x4B80A03100	256 B
NAVSS0_VIRT_ALIAS_8_UDMAP0_CFG_TCHAN	0x4B80B00000	0x4B80B20000	128 KB
NAVSS0_VIRT_ALIAS_8_UDMAP0_CFG_RCHAN	0x4B80C00000	0x4B80C08000	32 KB
NAVSS0_VIRT_ALIAS_8_UDMAP0_CFG_RFLOW	0x4B80D00000	0x4B80D04000	16 KB
NAVSS0_VIRT_ALIAS_8_SPINLOCK0_CFG	0x4B80E00000	0x4B80E08000	32 KB
NAVSS0_VIRT_ALIAS_8_TIMERMGR0_CFG_CONFIG	0x4B80E80000	0x4B80E80200	512 B
NAVSS0_VIRT_ALIAS_8_TIMERMGR1_CFG_CONFIG	0x4B80E81000	0x4B80E81200	512 B
NAVSS0_VIRT_ALIAS_8_TIMERMGR0_CFG_OES	0x4B80F00000	0x4B80F01000	4 KB
NAVSS0_VIRT_ALIAS_8_TIMERMGR1_CFG_OES	0x4B80F01000	0x4B80F02000	4 KB
NAVSS0_VIRT_ALIAS_8_IO_PVU0_CFG_MMRS	0x4B80F80000	0x4B80F81000	4 KB
NAVSS0_VIRT_ALIAS_8_IO_PVU1_CFG_MMRS	0x4B80F81000	0x4B80F82000	4 KB
NAVSS0_VIRT_ALIAS_8_PVU0_SRC_TOG_CFG	0x4B80F90000	0x4B80F90400	1 KB
NAVSS0_VIRT_ALIAS_8_PVU0_CFG_TOG_CFG	0x4B80F91000	0x4B80F91400	1 KB
NAVSS0_VIRT_ALIAS_8_ECCAGGR0	0x4B81000000	0x4B81000400	1 KB
NAVSS0_VIRT_ALIAS_8_UDMASS_ECCAGGR_CFG	0x4B81001000	0x4B81001400	1 KB
NAVSS0_VIRT_ALIAS_8_VIRTSS_ECCAGGR_CFG	0x4B81002000	0x4B81002400	1 KB
NAVSS0_VIRT_ALIAS_8_UDMASS_INTA0_CFG_GCNTCFG	0x4B81040000	0x4B81044000	16 KB
NAVSS0_VIRT_ALIAS_8_RINGACC0_CFG	0x4B81080000	0x4B810C0000	256 KB
NAVSS0_VIRT_ALIAS_8_REGS0_CFG_MMRS	0x4B810C0000	0x4B810C0100	256 B
NAVSS0_VIRT_ALIAS_8_CPTS0_S_VBUSP_CPTS_VBUSP	0x4B810D0000	0x4B810D0400	1 KB
NAVSS0_VIRT_ALIAS_8_INTR0_CFG_INTR_ROUTER_CFG	0x4B810E0000	0x4B810E4000	16 KB
NAVSS0_VIRT_ALIAS_8_UDMASS_INTA0_CFG_L2G	0x4B81100000	0x4B81102000	8 KB
NAVSS0_VIRT_ALIAS_8_UDMASS_INTA0_CFG_MCAST	0x4B81110000	0x4B81114000	16 KB
NAVSS0_VIRT_ALIAS_8_PROXY0_CFG_BUF_CFG_GCFG	0x4B81120000	0x4B81120100	256 B
NAVSS0_VIRT_ALIAS_8_PROXY0_CFG_BUFRAM_SLV_RAM	0x4B81130000	0x4B81134000	16 KB
NAVSS0_VIRT_ALIAS_8_SEC_PROXY0_CFG_MMRS	0x4B81140000	0x4B81140100	256 B
NAVSS0_VIRT_ALIAS_8_UDMAP0_CFG_GCFG	0x4B81150000	0x4B81150100	256 B
NAVSS0_VIRT_ALIAS_8_RINGACC0_CFG_GCFG	0x4B81160000	0x4B81160400	1 KB
NAVSS0_VIRT_ALIAS_8_PSILSS0_CFG_MMRS	0x4B81170000	0x4B81171000	4 KB

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
NAVSS0_VIRT_ALIAS_8_BCDMA0_CFG_GCFG	0x4B811A0000	0x4B811A0100	256 B
NAVSS0_VIRT_ALIAS_8_MCRC0_S_CFG_MCRC64	0x4B81F70000	0x4B81F71000	4 KB
NAVSS0_VIRT_ALIAS_8_PSILCFG0_CFG_PROXY	0x4B81F78000	0x4B81F78200	512 B
NAVSS0_VIRT_ALIAS_8_MAILBOX0_CFG_REGS0	0x4B81F80000	0x4B81F80200	512 B
NAVSS0_VIRT_ALIAS_8_MAILBOX0_CFG_REGS1	0x4B81F81000	0x4B81F81200	512 B
NAVSS0_VIRT_ALIAS_8_MAILBOX0_CFG_REGS2	0x4B81F82000	0x4B81F82200	512 B
NAVSS0_VIRT_ALIAS_8_MAILBOX0_CFG_REGS3	0x4B81F83000	0x4B81F83200	512 B
NAVSS0_VIRT_ALIAS_8_MAILBOX0_CFG_REGS4	0x4B81F84000	0x4B81F84200	512 B
NAVSS0_VIRT_ALIAS_8_MAILBOX0_CFG_REGS5	0x4B81F85000	0x4B81F85200	512 B
NAVSS0_VIRT_ALIAS_8_MAILBOX0_CFG_REGS6	0x4B81F86000	0x4B81F86200	512 B
NAVSS0_VIRT_ALIAS_8_MAILBOX0_CFG_REGS7	0x4B81F87000	0x4B81F87200	512 B
NAVSS0_VIRT_ALIAS_8_MAILBOX0_CFG_REGS8	0x4B81F88000	0x4B81F88200	512 B
NAVSS0_VIRT_ALIAS_8_MAILBOX0_CFG_REGS9	0x4B81F89000	0x4B81F89200	512 B
NAVSS0_VIRT_ALIAS_8_MAILBOX0_CFG_REGS10	0x4B81F8A000	0x4B81F8A200	512 B
NAVSS0_VIRT_ALIAS_8_MAILBOX0_CFG_REGS11	0x4B81F8B000	0x4B81F8B200	512 B
NAVSS0_VIRT_ALIAS_8_MAILBOX1_CFG_REGS0	0x4B81F90000	0x4B81F90200	512 B
NAVSS0_VIRT_ALIAS_8_MAILBOX1_CFG_REGS1	0x4B81F91000	0x4B81F91200	512 B
NAVSS0_VIRT_ALIAS_8_MAILBOX1_CFG_REGS2	0x4B81F92000	0x4B81F92200	512 B
NAVSS0_VIRT_ALIAS_8_MAILBOX1_CFG_REGS3	0x4B81F93000	0x4B81F93200	512 B
NAVSS0_VIRT_ALIAS_8_MAILBOX1_CFG_REGS4	0x4B81F94000	0x4B81F94200	512 B
NAVSS0_VIRT_ALIAS_8_MAILBOX1_CFG_REGS5	0x4B81F95000	0x4B81F95200	512 B
NAVSS0_VIRT_ALIAS_8_MAILBOX1_CFG_REGS6	0x4B81F96000	0x4B81F96200	512 B
NAVSS0_VIRT_ALIAS_8_MAILBOX1_CFG_REGS7	0x4B81F97000	0x4B81F97200	512 B
NAVSS0_VIRT_ALIAS_8_MAILBOX1_CFG_REGS8	0x4B81F98000	0x4B81F98200	512 B
NAVSS0_VIRT_ALIAS_8_MAILBOX1_CFG_REGS9	0x4B81F99000	0x4B81F99200	512 B
NAVSS0_VIRT_ALIAS_8_MAILBOX1_CFG_REGS10	0x4B81F9A000	0x4B81F9A200	512 B
NAVSS0_VIRT_ALIAS_8_MAILBOX1_CFG_REGS11	0x4B81F9B000	0x4B81F9B200	512 B
NAVSS0_VIRT_ALIAS_8_RINGACC0_CFG_MON	0x4B82000000	0x4B82020000	128 KB
NAVSS0_VIRT_ALIAS_8_TIMERMGR0_CFG_TIMERS	0x4B82200000	0x4B82240000	256 KB
NAVSS0_VIRT_ALIAS_8_TIMERMGR1_CFG_TIMERS	0x4B82240000	0x4B82280000	256 KB
NAVSS0_VIRT_ALIAS_8_SEC_PROXY0_CFG_RT	0x4B82400000	0x4B82600000	2 MB
NAVSS0_VIRT_ALIAS_8_SEC_PROXY0_CFG_SCFG	0x4B82800000	0x4B82A00000	2 MB
NAVSS0_VIRT_ALIAS_8_SEC_PROXY0_SRC_TARGET_DATA	0x4B82C00000	0x4B82E00000	2 MB
NAVSS0_VIRT_ALIAS_8_PROXY0_SRC_TARGET0_DATA	0x4B83000000	0x4B83040000	256 KB
NAVSS0_VIRT_ALIAS_8_PROXY0_CFG_BUF_CFG	0x4B83400000	0x4B83440000	256 KB
NAVSS0_VIRT_ALIAS_8_UDMASS_INTA0_CFG_GCNTRTI	0x4B83800000	0x4B83A00000	2 MB
NAVSS0_VIRT_ALIAS_8_MODSS_INTA0_CFG_INTR	0x4B83C00000	0x4B83C40000	256 KB
NAVSS0_VIRT_ALIAS_8_MODSS_INTA1_CFG_INTR	0x4B83C40000	0x4B83C80000	256 KB
NAVSS0_VIRT_ALIAS_8_UDMASS_INTA0_CFG_INTR	0x4B83D00000	0x4B83E00000	1 MB
NAVSS0_VIRT_ALIAS_8_UDMAP0_CFG_RCHANRT	0x4B84000000	0x4B84080000	512 KB
NAVSS0_VIRT_ALIAS_8_UDMAP0_CFG_TCHANRT	0x4B85000000	0x4B85200000	2 MB
NAVSS0_VIRT_ALIAS_8_BCDMA0_CFG_TCHAN	0x4B85840000	0x4B85841000	4 KB
NAVSS0_VIRT_ALIAS_8_BCDMA0_CFG_RCHAN	0x4B85880000	0x4B85882000	8 KB
NAVSS0_VIRT_ALIAS_8_BCDMA0_CFG_RING	0x4B85900000	0x4B85904000	16 KB
NAVSS0_VIRT_ALIAS_8_BCDMA0_CFG_TCHANRT	0x4B85C00000	0x4B85C10000	64 KB
NAVSS0_VIRT_ALIAS_8_BCDMA0_CFG_RCHANRT	0x4B85D00000	0x4B85D20000	128 KB
NAVSS0_VIRT_ALIAS_8_BCDMA0_CFG_RINGRT	0x4B85E00000	0x4B85E80000	512 KB

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
NAVSS0_VIRT_ALIAS_8_IO_PVU0_CFG_TLBIF_TLB	0x4B86000000	0x4B86040000	256 KB
NAVSS0_VIRT_ALIAS_8_IO_PVU1_CFG_TLBIF_TLB	0x4B86040000	0x4B86080000	256 KB
NAVSS0_VIRT_ALIAS_8_RINGACC0_SRC_FIFOS	0x4B88000000	0x4B88400000	4 MB
NAVSS0_VIRT_ALIAS_8_RINGACC0_CFG_RT	0x4B8C000000	0x4B8C400000	4 MB
NAVSS0_VIRT_ALIAS_9_MODSS_INTA0_CFG	0x4B90800000	0x4B90800020	32 B
NAVSS0_VIRT_ALIAS_9_MODSS_INTA1_CFG	0x4B90801000	0x4B90801020	32 B
NAVSS0_VIRT_ALIAS_9_UDMASS_INTA0_CFG	0x4B90802000	0x4B90802020	32 B
NAVSS0_VIRT_ALIAS_9_UDMASS_INTA0_CFG_UNMAP	0x4B90880000	0x4B90890000	64 KB
NAVSS0_VIRT_ALIAS_9_MODSS_INTA0_CFG_IMAP	0x4B90900000	0x4B90902000	8 KB
NAVSS0_VIRT_ALIAS_9_MODSS_INTA1_CFG_IMAP	0x4B90908000	0x4B9090A000	8 KB
NAVSS0_VIRT_ALIAS_9_UDMASS_INTA0_CFG_IMAP	0x4B90940000	0x4B90950000	64 KB
NAVSS0_VIRT_ALIAS_9_NAV_DDR0_VIRTID_CFG_MMRS	0x4B90A02000	0x4B90A02100	256 B
NAVSS0_VIRT_ALIAS_9_NAV_DDR1_VIRTID_CFG_MMRS	0x4B90A03000	0x4B90A03100	256 B
NAVSS0_VIRT_ALIAS_9_UDMAP0_CFG_TCHAN	0x4B90B00000	0x4B90B20000	128 KB
NAVSS0_VIRT_ALIAS_9_UDMAP0_CFG_RCHAN	0x4B90C00000	0x4B90C08000	32 KB
NAVSS0_VIRT_ALIAS_9_UDMAP0_CFG_RFLOW	0x4B90D00000	0x4B90D04000	16 KB
NAVSS0_VIRT_ALIAS_9_SPINLOCK0_CFG	0x4B90E00000	0x4B90E08000	32 KB
NAVSS0_VIRT_ALIAS_9_TIMERMGR0_CFG_CONFIG	0x4B90E80000	0x4B90E80200	512 B
NAVSS0_VIRT_ALIAS_9_TIMERMGR1_CFG_CONFIG	0x4B90E81000	0x4B90E81200	512 B
NAVSS0_VIRT_ALIAS_9_TIMERMGR0_CFG_OES	0x4B90F00000	0x4B90F01000	4 KB
NAVSS0_VIRT_ALIAS_9_TIMERMGR1_CFG_OES	0x4B90F01000	0x4B90F02000	4 KB
NAVSS0_VIRT_ALIAS_9_IO_PVU0_CFG_MMRS	0x4B90F80000	0x4B90F81000	4 KB
NAVSS0_VIRT_ALIAS_9_IO_PVU1_CFG_MMRS	0x4B90F81000	0x4B90F82000	4 KB
NAVSS0_VIRT_ALIAS_9_PVU0_SRC_TOG_CFG	0x4B90F90000	0x4B90F90400	1 KB
NAVSS0_VIRT_ALIAS_9_PVU0_CFG_TOG_CFG	0x4B90F91000	0x4B90F91400	1 KB
NAVSS0_VIRT_ALIAS_9_ECCAGGR0	0x4B91000000	0x4B91000400	1 KB
NAVSS0_VIRT_ALIAS_9_UDMASS_ECCAGGR_CFG	0x4B91001000	0x4B91001400	1 KB
NAVSS0_VIRT_ALIAS_9_VIRTSS_ECCAGGR_CFG	0x4B91002000	0x4B91002400	1 KB
NAVSS0_VIRT_ALIAS_9_UDMASS_INTA0_CFG_GCNTCFG	0x4B91040000	0x4B91044000	16 KB
NAVSS0_VIRT_ALIAS_9_RINGACC0_CFG	0x4B91080000	0x4B910C0000	256 KB
NAVSS0_VIRT_ALIAS_9_REGS0_CFG_MMRS	0x4B910C0000	0x4B910C0100	256 B
NAVSS0_VIRT_ALIAS_9_CPTS0_S_VBUSP_CPTS_VBUSP	0x4B910D0000	0x4B910D0400	1 KB
NAVSS0_VIRT_ALIAS_9_INTR0_CFG_INTR_ROUTER_CFG	0x4B910E0000	0x4B910E4000	16 KB
NAVSS0_VIRT_ALIAS_9_UDMASS_INTA0_CFG_L2G	0x4B91100000	0x4B91102000	8 KB
NAVSS0_VIRT_ALIAS_9_UDMASS_INTA0_CFG_MCAST	0x4B91110000	0x4B91114000	16 KB
NAVSS0_VIRT_ALIAS_9_PROXY0_CFG_BUF_CFG_GCFG	0x4B91120000	0x4B91120100	256 B
NAVSS0_VIRT_ALIAS_9_PROXY0_CFG_BUFRAM_SLV_RAM	0x4B91130000	0x4B91134000	16 KB
NAVSS0_VIRT_ALIAS_9_SEC_PROXY0_CFG_MMRS	0x4B91140000	0x4B91140100	256 B
NAVSS0_VIRT_ALIAS_9_UDMAP0_CFG_GCFG	0x4B91150000	0x4B91150100	256 B
NAVSS0_VIRT_ALIAS_9_RINGACC0_CFG_GCFG	0x4B91160000	0x4B91160400	1 KB
NAVSS0_VIRT_ALIAS_9_PSILSS0_CFG_MMRS	0x4B91170000	0x4B91171000	4 KB
NAVSS0_VIRT_ALIAS_9_BCDMA0_CFG_GCFG	0x4B911A0000	0x4B911A0100	256 B
NAVSS0_VIRT_ALIAS_9_MCRC0_S_CFG_MCRC64	0x4B91F70000	0x4B91F71000	4 KB
NAVSS0_VIRT_ALIAS_9_PSILCFG0_CFG_PROXY	0x4B91F78000	0x4B91F78200	512 B
NAVSS0_VIRT_ALIAS_9_MAILBOX0_CFG_REGS0	0x4B91F80000	0x4B91F80200	512 B
NAVSS0_VIRT_ALIAS_9_MAILBOX0_CFG_REGS1	0x4B91F81000	0x4B91F81200	512 B
NAVSS0_VIRT_ALIAS_9_MAILBOX0_CFG_REGS2	0x4B91F82000	0x4B91F82200	512 B

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
NAVSS0_VIRT_ALIAS_9_MAILBOX0_CFG_REGS3	0x4B91F83000	0x4B91F83200	512 B
NAVSS0_VIRT_ALIAS_9_MAILBOX0_CFG_REGS4	0x4B91F84000	0x4B91F84200	512 B
NAVSS0_VIRT_ALIAS_9_MAILBOX0_CFG_REGS5	0x4B91F85000	0x4B91F85200	512 B
NAVSS0_VIRT_ALIAS_9_MAILBOX0_CFG_REGS6	0x4B91F86000	0x4B91F86200	512 B
NAVSS0_VIRT_ALIAS_9_MAILBOX0_CFG_REGS7	0x4B91F87000	0x4B91F87200	512 B
NAVSS0_VIRT_ALIAS_9_MAILBOX0_CFG_REGS8	0x4B91F88000	0x4B91F88200	512 B
NAVSS0_VIRT_ALIAS_9_MAILBOX0_CFG_REGS9	0x4B91F89000	0x4B91F89200	512 B
NAVSS0_VIRT_ALIAS_9_MAILBOX0_CFG_REGS10	0x4B91F8A000	0x4B91F8A200	512 B
NAVSS0_VIRT_ALIAS_9_MAILBOX0_CFG_REGS11	0x4B91F8B000	0x4B91F8B200	512 B
NAVSS0_VIRT_ALIAS_9_MAILBOX1_CFG_REGS0	0x4B91F90000	0x4B91F90200	512 B
NAVSS0_VIRT_ALIAS_9_MAILBOX1_CFG_REGS1	0x4B91F91000	0x4B91F91200	512 B
NAVSS0_VIRT_ALIAS_9_MAILBOX1_CFG_REGS2	0x4B91F92000	0x4B91F92200	512 B
NAVSS0_VIRT_ALIAS_9_MAILBOX1_CFG_REGS3	0x4B91F93000	0x4B91F93200	512 B
NAVSS0_VIRT_ALIAS_9_MAILBOX1_CFG_REGS4	0x4B91F94000	0x4B91F94200	512 B
NAVSS0_VIRT_ALIAS_9_MAILBOX1_CFG_REGS5	0x4B91F95000	0x4B91F95200	512 B
NAVSS0_VIRT_ALIAS_9_MAILBOX1_CFG_REGS6	0x4B91F96000	0x4B91F96200	512 B
NAVSS0_VIRT_ALIAS_9_MAILBOX1_CFG_REGS7	0x4B91F97000	0x4B91F97200	512 B
NAVSS0_VIRT_ALIAS_9_MAILBOX1_CFG_REGS8	0x4B91F98000	0x4B91F98200	512 B
NAVSS0_VIRT_ALIAS_9_MAILBOX1_CFG_REGS9	0x4B91F99000	0x4B91F99200	512 B
NAVSS0_VIRT_ALIAS_9_MAILBOX1_CFG_REGS10	0x4B91F9A000	0x4B91F9A200	512 B
NAVSS0_VIRT_ALIAS_9_MAILBOX1_CFG_REGS11	0x4B91F9B000	0x4B91F9B200	512 B
NAVSS0_VIRT_ALIAS_9_RINGACC0_CFG_MON	0x4B92000000	0x4B92020000	128 KB
NAVSS0_VIRT_ALIAS_9_TIMERMGR0_CFG_TIMERS	0x4B92200000	0x4B92240000	256 KB
NAVSS0_VIRT_ALIAS_9_TIMERMGR1_CFG_TIMERS	0x4B92240000	0x4B92280000	256 KB
NAVSS0_VIRT_ALIAS_9_SEC_PROXY0_CFG_RT	0x4B92400000	0x4B92600000	2 MB
NAVSS0_VIRT_ALIAS_9_SEC_PROXY0_CFG_SCFG	0x4B92800000	0x4B92A00000	2 MB
NAVSS0_VIRT_ALIAS_9_SEC_PROXY0_SRC_TARGET_DATA	0x4B92C00000	0x4B92E00000	2 MB
NAVSS0_VIRT_ALIAS_9_PROXY0_SRC_TARGET0_DATA	0x4B93000000	0x4B93040000	256 KB
NAVSS0_VIRT_ALIAS_9_PROXY0_CFG_BUF_CFG	0x4B93400000	0x4B93440000	256 KB
NAVSS0_VIRT_ALIAS_9_UDMASS_INTA0_CFG_GCNTRTI	0x4B93800000	0x4B93A00000	2 MB
NAVSS0_VIRT_ALIAS_9_MODSS_INTA0_CFG_INTR	0x4B93C00000	0x4B93C40000	256 KB
NAVSS0_VIRT_ALIAS_9_MODSS_INTA1_CFG_INTR	0x4B93C40000	0x4B93C80000	256 KB
NAVSS0_VIRT_ALIAS_9_UDMASS_INTA0_CFG_INTR	0x4B93D00000	0x4B93E00000	1 MB
NAVSS0_VIRT_ALIAS_9_UDMAP0_CFG_RCHANRT	0x4B94000000	0x4B94080000	512 KB
NAVSS0_VIRT_ALIAS_9_UDMAP0_CFG_TCHANRT	0x4B95000000	0x4B95200000	2 MB
NAVSS0_VIRT_ALIAS_9_BCDMA0_CFG_TCHAN	0x4B95840000	0x4B95841000	4 KB
NAVSS0_VIRT_ALIAS_9_BCDMA0_CFG_RCHAN	0x4B95880000	0x4B95882000	8 KB
NAVSS0_VIRT_ALIAS_9_BCDMA0_CFG_RING	0x4B95900000	0x4B95904000	16 KB
NAVSS0_VIRT_ALIAS_9_BCDMA0_CFG_TCHANRT	0x4B95C00000	0x4B95C10000	64 KB
NAVSS0_VIRT_ALIAS_9_BCDMA0_CFG_RCHANRT	0x4B95D00000	0x4B95D20000	128 KB
NAVSS0_VIRT_ALIAS_9_BCDMA0_CFG_RINGRT	0x4B95E00000	0x4B95E80000	512 KB
NAVSS0_VIRT_ALIAS_9_IO_PVU0_CFG_TLBIF_TLB	0x4B96000000	0x4B96040000	256 KB
NAVSS0_VIRT_ALIAS_9_IO_PVU1_CFG_TLBIF_TLB	0x4B96040000	0x4B96080000	256 KB
NAVSS0_VIRT_ALIAS_9_RINGACC0_SRC_FIFOS	0x4B98000000	0x4B98400000	4 MB
NAVSS0_VIRT_ALIAS_9_RINGACC0_CFG_RT	0x4B9C000000	0x4B9C400000	4 MB
NAVSS0_VIRT_ALIAS_10_MODSS_INTA0_CFG	0x4BA0800000	0x4BA0800020	32 B
NAVSS0_VIRT_ALIAS_10_MODSS_INTA1_CFG	0x4BA0801000	0x4BA0801020	32 B

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
NAVSS0_VIRT_ALIAS_10_UDMASS_INTA0_CFG	0x4BA0802000	0x4BA0802020	32 B
NAVSS0_VIRT_ALIAS_10_UDMASS_INTA0_CFG_UNMAP	0x4BA0880000	0x4BA0890000	64 KB
NAVSS0_VIRT_ALIAS_10_MODSS_INTA0_CFG_IMAP	0x4BA0900000	0x4BA0902000	8 KB
NAVSS0_VIRT_ALIAS_10_MODSS_INTA1_CFG_IMAP	0x4BA0908000	0x4BA090A000	8 KB
NAVSS0_VIRT_ALIAS_10_UDMASS_INTA0_CFG_IMAP	0x4BA0940000	0x4BA0950000	64 KB
NAVSS0_VIRT_ALIAS_10_NAV_DDR0_VIRTID_CFG_MMRS	0x4BA0A02000	0x4BA0A02100	256 B
NAVSS0_VIRT_ALIAS_10_NAV_DDR1_VIRTID_CFG_MMRS	0x4BA0A03000	0x4BA0A03100	256 B
NAVSS0_VIRT_ALIAS_10_UDMAP0_CFG_TCHAN	0x4BA0B00000	0x4BA0B20000	128 KB
NAVSS0_VIRT_ALIAS_10_UDMAP0_CFG_RCHAN	0x4BA0C00000	0x4BA0C08000	32 KB
NAVSS0_VIRT_ALIAS_10_UDMAP0_CFG_RFLOW	0x4BA0D00000	0x4BA0D04000	16 KB
NAVSS0_VIRT_ALIAS_10_SPINLOCK0_CFG	0x4BA0E00000	0x4BA0E08000	32 KB
NAVSS0_VIRT_ALIAS_10_TIMERMGR0_CFG_CONFIG	0x4BA0E80000	0x4BA0E80200	512 B
NAVSS0_VIRT_ALIAS_10_TIMERMGR1_CFG_CONFIG	0x4BA0E81000	0x4BA0E81200	512 B
NAVSS0_VIRT_ALIAS_10_TIMERMGR0_CFG_OES	0x4BA0F00000	0x4BA0F01000	4 KB
NAVSS0_VIRT_ALIAS_10_TIMERMGR1_CFG_OES	0x4BA0F01000	0x4BA0F02000	4 KB
NAVSS0_VIRT_ALIAS_10_IO_PVU0_CFG_MMRS	0x4BA0F80000	0x4BA0F81000	4 KB
NAVSS0_VIRT_ALIAS_10_IO_PVU1_CFG_MMRS	0x4BA0F81000	0x4BA0F82000	4 KB
NAVSS0_VIRT_ALIAS_10_PVU0_SRC_TOG_CFG	0x4BA0F90000	0x4BA0F90400	1 KB
NAVSS0_VIRT_ALIAS_10_PVU0_CFG_TOG_CFG	0x4BA0F91000	0x4BA0F91400	1 KB
NAVSS0_VIRT_ALIAS_10_ECCAGGR0	0x4BA1000000	0x4BA1000400	1 KB
NAVSS0_VIRT_ALIAS_10_UDMASS_ECCAGGR_CFG	0x4BA1001000	0x4BA1001400	1 KB
NAVSS0_VIRT_ALIAS_10_VIRTSS_ECCAGGR_CFG	0x4BA1002000	0x4BA1002400	1 KB
NAVSS0_VIRT_ALIAS_10_UDMASS_INTA0_CFG_GCNTCFG	0x4BA1040000	0x4BA1044000	16 KB
NAVSS0_VIRT_ALIAS_10_RINGACC0_CFG	0x4BA1080000	0x4BA10C0000	256 KB
NAVSS0_VIRT_ALIAS_10_REGS0_CFG_MMRS	0x4BA10C0000	0x4BA10C0100	256 B
NAVSS0_VIRT_ALIAS_10_CPTS0_S_VBUSP_CPTS_VBUSP	0x4BA10D0000	0x4BA10D0400	1 KB
NAVSS0_VIRT_ALIAS_10_INTR0_CFG_INTR_ROUTER_CFG	0x4BA10E0000	0x4BA10E4000	16 KB
NAVSS0_VIRT_ALIAS_10_UDMASS_INTA0_CFG_L2G	0x4BA1100000	0x4BA1102000	8 KB
NAVSS0_VIRT_ALIAS_10_UDMASS_INTA0_CFG_MCAST	0x4BA1110000	0x4BA1114000	16 KB
NAVSS0_VIRT_ALIAS_10_PROXY0_CFG_BUF_CFG_GCFG	0x4BA1120000	0x4BA1120100	256 B
NAVSS0_VIRT_ALIAS_10_PROXY0_CFG_BUFRAM_SLV_RAM	0x4BA1130000	0x4BA1134000	16 KB
NAVSS0_VIRT_ALIAS_10_SEC_PROXY0_CFG_MMRS	0x4BA1140000	0x4BA1140100	256 B
NAVSS0_VIRT_ALIAS_10_UDMAP0_CFG_GCFG	0x4BA1150000	0x4BA1150100	256 B
NAVSS0_VIRT_ALIAS_10_RINGACC0_CFG_GCFG	0x4BA1160000	0x4BA1160400	1 KB
NAVSS0_VIRT_ALIAS_10_PSILSS0_CFG_MMRS	0x4BA1170000	0x4BA1171000	4 KB
NAVSS0_VIRT_ALIAS_10_BCDMA0_CFG_GCFG	0x4BA11A0000	0x4BA11A0100	256 B
NAVSS0_VIRT_ALIAS_10_MCRC0_S_CFG_MCRC64	0x4BA1F70000	0x4BA1F71000	4 KB
NAVSS0_VIRT_ALIAS_10_PSILCFG0_CFG_PROXY	0x4BA1F78000	0x4BA1F78200	512 B
NAVSS0_VIRT_ALIAS_10_MAILBOX0_CFG_REGS0	0x4BA1F80000	0x4BA1F80200	512 B
NAVSS0_VIRT_ALIAS_10_MAILBOX0_CFG_REGS1	0x4BA1F81000	0x4BA1F81200	512 B
NAVSS0_VIRT_ALIAS_10_MAILBOX0_CFG_REGS2	0x4BA1F82000	0x4BA1F82200	512 B
NAVSS0_VIRT_ALIAS_10_MAILBOX0_CFG_REGS3	0x4BA1F83000	0x4BA1F83200	512 B
NAVSS0_VIRT_ALIAS_10_MAILBOX0_CFG_REGS4	0x4BA1F84000	0x4BA1F84200	512 B
NAVSS0_VIRT_ALIAS_10_MAILBOX0_CFG_REGS5	0x4BA1F85000	0x4BA1F85200	512 B
NAVSS0_VIRT_ALIAS_10_MAILBOX0_CFG_REGS6	0x4BA1F86000	0x4BA1F86200	512 B
NAVSS0_VIRT_ALIAS_10_MAILBOX0_CFG_REGS7	0x4BA1F87000	0x4BA1F87200	512 B
NAVSS0_VIRT_ALIAS_10_MAILBOX0_CFG_REGS8	0x4BA1F88000	0x4BA1F88200	512 B

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
NAVSS0_VIRT_ALIAS_10_MAILBOX0_CFG_REGS9	0x4BA1F89000	0x4BA1F89200	512 B
NAVSS0_VIRT_ALIAS_10_MAILBOX0_CFG_REGS10	0x4BA1F8A000	0x4BA1F8A200	512 B
NAVSS0_VIRT_ALIAS_10_MAILBOX0_CFG_REGS11	0x4BA1F8B000	0x4BA1F8B200	512 B
NAVSS0_VIRT_ALIAS_10_MAILBOX1_CFG_REGS0	0x4BA1F90000	0x4BA1F90200	512 B
NAVSS0_VIRT_ALIAS_10_MAILBOX1_CFG_REGS1	0x4BA1F91000	0x4BA1F91200	512 B
NAVSS0_VIRT_ALIAS_10_MAILBOX1_CFG_REGS2	0x4BA1F92000	0x4BA1F92200	512 B
NAVSS0_VIRT_ALIAS_10_MAILBOX1_CFG_REGS3	0x4BA1F93000	0x4BA1F93200	512 B
NAVSS0_VIRT_ALIAS_10_MAILBOX1_CFG_REGS4	0x4BA1F94000	0x4BA1F94200	512 B
NAVSS0_VIRT_ALIAS_10_MAILBOX1_CFG_REGS5	0x4BA1F95000	0x4BA1F95200	512 B
NAVSS0_VIRT_ALIAS_10_MAILBOX1_CFG_REGS6	0x4BA1F96000	0x4BA1F96200	512 B
NAVSS0_VIRT_ALIAS_10_MAILBOX1_CFG_REGS7	0x4BA1F97000	0x4BA1F97200	512 B
NAVSS0_VIRT_ALIAS_10_MAILBOX1_CFG_REGS8	0x4BA1F98000	0x4BA1F98200	512 B
NAVSS0_VIRT_ALIAS_10_MAILBOX1_CFG_REGS9	0x4BA1F99000	0x4BA1F99200	512 B
NAVSS0_VIRT_ALIAS_10_MAILBOX1_CFG_REGS10	0x4BA1F9A000	0x4BA1F9A200	512 B
NAVSS0_VIRT_ALIAS_10_MAILBOX1_CFG_REGS11	0x4BA1F9B000	0x4BA1F9B200	512 B
NAVSS0_VIRT_ALIAS_10_RINGACC0_CFG_MON	0x4BA2000000	0x4BA2020000	128 KB
NAVSS0_VIRT_ALIAS_10_TIMERMGR0_CFG_TIMERS	0x4BA2200000	0x4BA2240000	256 KB
NAVSS0_VIRT_ALIAS_10_TIMERMGR1_CFG_TIMERS	0x4BA2240000	0x4BA2280000	256 KB
NAVSS0_VIRT_ALIAS_10_SEC_PROXY0_CFG_RT	0x4BA2400000	0x4BA2600000	2 MB
NAVSS0_VIRT_ALIAS_10_SEC_PROXY0_CFG_SCFG	0x4BA2800000	0x4BA2A00000	2 MB
NAVSS0_VIRT_ALIAS_10_SEC_PROXY0_SRC_TARGET_DATA	0x4BA2C00000	0x4BA2E00000	2 MB
NAVSS0_VIRT_ALIAS_10_PROXY0_SRC_TARGET0_DATA	0x4BA3000000	0x4BA3040000	256 KB
NAVSS0_VIRT_ALIAS_10_PROXY0_CFG_BUF_CFG	0x4BA3400000	0x4BA3440000	256 KB
NAVSS0_VIRT_ALIAS_10_UDMASS_INTA0_CFG_GCNTRTI	0x4BA3800000	0x4BA3A00000	2 MB
NAVSS0_VIRT_ALIAS_10_MODSS_INTA0_CFG_INTR	0x4BA3C00000	0x4BA3C40000	256 KB
NAVSS0_VIRT_ALIAS_10_MODSS_INTA1_CFG_INTR	0x4BA3C40000	0x4BA3C80000	256 KB
NAVSS0_VIRT_ALIAS_10_UDMASS_INTA0_CFG_INTR	0x4BA3D00000	0x4BA3E00000	1 MB
NAVSS0_VIRT_ALIAS_10_UDMAP0_CFG_RCHANRT	0x4BA4000000	0x4BA4080000	512 KB
NAVSS0_VIRT_ALIAS_10_UDMAP0_CFG_TCHANRT	0x4BA5000000	0x4BA5200000	2 MB
NAVSS0_VIRT_ALIAS_10_BCDMA0_CFG_TCHAN	0x4BA5840000	0x4BA5841000	4 KB
NAVSS0_VIRT_ALIAS_10_BCDMA0_CFG_RCHAN	0x4BA5880000	0x4BA5882000	8 KB
NAVSS0_VIRT_ALIAS_10_BCDMA0_CFG_RING	0x4BA5900000	0x4BA5904000	16 KB
NAVSS0_VIRT_ALIAS_10_BCDMA0_CFG_TCHANRT	0x4BA5C00000	0x4BA5C10000	64 KB
NAVSS0_VIRT_ALIAS_10_BCDMA0_CFG_RCHANRT	0x4BA5D00000	0x4BA5D20000	128 KB
NAVSS0_VIRT_ALIAS_10_BCDMA0_CFG_RINGRT	0x4BA5E00000	0x4BA5E80000	512 KB
NAVSS0_VIRT_ALIAS_10_IO_PVU0_CFG_TLBIF_TLB	0x4BA6000000	0x4BA6040000	256 KB
NAVSS0_VIRT_ALIAS_10_IO_PVU1_CFG_TLBIF_TLB	0x4BA6040000	0x4BA6080000	256 KB
NAVSS0_VIRT_ALIAS_10_RINGACC0_SRC_FIFOS	0x4BA8000000	0x4BA8400000	4 MB
NAVSS0_VIRT_ALIAS_10_RINGACC0_CFG_RT	0x4BAC000000	0x4BAC400000	4 MB
NAVSS0_VIRT_ALIAS_11_MODSS_INTA0_CFG	0x4BB0800000	0x4BB0800020	32 B
NAVSS0_VIRT_ALIAS_11_MODSS_INTA1_CFG	0x4BB0801000	0x4BB0801020	32 B
NAVSS0_VIRT_ALIAS_11_UDMASS_INTA0_CFG	0x4BB0802000	0x4BB0802020	32 B
NAVSS0_VIRT_ALIAS_11_UDMASS_INTA0_CFG_UNMAP	0x4BB0880000	0x4BB0890000	64 KB
NAVSS0_VIRT_ALIAS_11_MODSS_INTA0_CFG_IMAP	0x4BB0900000	0x4BB0902000	8 KB
NAVSS0_VIRT_ALIAS_11_MODSS_INTA1_CFG_IMAP	0x4BB0908000	0x4BB090A000	8 KB
NAVSS0_VIRT_ALIAS_11_UDMASS_INTA0_CFG_IMAP	0x4BB0940000	0x4BB0950000	64 KB
NAVSS0_VIRT_ALIAS_11_NAV_DDR0_VIRTID_CFG_MMRS	0x4BB0A02000	0x4BB0A02100	256 B

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
NAVSS0_VIRT_ALIAS_11_NAV_DDR1_VIRTID_CFG_MMRS	0x4BB0A03000	0x4BB0A03100	256 B
NAVSS0_VIRT_ALIAS_11_UDMAP0_CFG_TCHAN	0x4BB0B00000	0x4BB0B20000	128 KB
NAVSS0_VIRT_ALIAS_11_UDMAP0_CFG_RCHAN	0x4BB0C00000	0x4BB0C08000	32 KB
NAVSS0_VIRT_ALIAS_11_UDMAP0_CFG_RFLOW	0x4BB0D00000	0x4BB0D04000	16 KB
NAVSS0_VIRT_ALIAS_11_SPINLOCK0_CFG	0x4BB0E00000	0x4BB0E08000	32 KB
NAVSS0_VIRT_ALIAS_11_TIMERMGR0_CFG_CONFIG	0x4BB0E80000	0x4BB0E80200	512 B
NAVSS0_VIRT_ALIAS_11_TIMERMGR1_CFG_CONFIG	0x4BB0E81000	0x4BB0E81200	512 B
NAVSS0_VIRT_ALIAS_11_TIMERMGR0_CFG_OES	0x4BB0F00000	0x4BB0F01000	4 KB
NAVSS0_VIRT_ALIAS_11_TIMERMGR1_CFG_OES	0x4BB0F01000	0x4BB0F02000	4 KB
NAVSS0_VIRT_ALIAS_11_IO_PVU0_CFG_MMRS	0x4BB0F80000	0x4BB0F81000	4 KB
NAVSS0_VIRT_ALIAS_11_IO_PVU1_CFG_MMRS	0x4BB0F81000	0x4BB0F82000	4 KB
NAVSS0_VIRT_ALIAS_11_PVU0_SRC_TOG_CFG	0x4BB0F90000	0x4BB0F90400	1 KB
NAVSS0_VIRT_ALIAS_11_PVU0_CFG_TOG_CFG	0x4BB0F91000	0x4BB0F91400	1 KB
NAVSS0_VIRT_ALIAS_11_ECCAGGR0	0x4BB1000000	0x4BB1000400	1 KB
NAVSS0_VIRT_ALIAS_11_UDMASS_ECCAGGR_CFG	0x4BB1001000	0x4BB1001400	1 KB
NAVSS0_VIRT_ALIAS_11_VIRTSS_ECCAGGR_CFG	0x4BB1002000	0x4BB1002400	1 KB
NAVSS0_VIRT_ALIAS_11_UDMASS_INTA0_CFG_GCNTCFG	0x4BB1040000	0x4BB1044000	16 KB
NAVSS0_VIRT_ALIAS_11_RINGACC0_CFG	0x4BB1080000	0x4BB10C0000	256 KB
NAVSS0_VIRT_ALIAS_11_REGS0_CFG_MMRS	0x4BB10C0000	0x4BB10C0100	256 B
NAVSS0_VIRT_ALIAS_11_CPTS0_S_VBUSP_CPTS_VBUSP	0x4BB10D0000	0x4BB10D0400	1 KB
NAVSS0_VIRT_ALIAS_11_INTR0_CFG_INTR_ROUTER_CFG	0x4BB10E0000	0x4BB10E4000	16 KB
NAVSS0_VIRT_ALIAS_11_UDMASS_INTA0_CFG_L2G	0x4BB1100000	0x4BB1102000	8 KB
NAVSS0_VIRT_ALIAS_11_UDMASS_INTA0_CFG_MCAST	0x4BB1110000	0x4BB1114000	16 KB
NAVSS0_VIRT_ALIAS_11_PROXY0_CFG_BUF_CFG_GCFG	0x4BB1120000	0x4BB1120100	256 B
NAVSS0_VIRT_ALIAS_11_PROXY0_CFG_BUFRAM_SLV_RAM	0x4BB1130000	0x4BB1134000	16 KB
NAVSS0_VIRT_ALIAS_11_SEC_PROXY0_CFG_MMRS	0x4BB1140000	0x4BB1140100	256 B
NAVSS0_VIRT_ALIAS_11_UDMAP0_CFG_GCFG	0x4BB1150000	0x4BB1150100	256 B
NAVSS0_VIRT_ALIAS_11_RINGACC0_CFG_GCFG	0x4BB1160000	0x4BB1160400	1 KB
NAVSS0_VIRT_ALIAS_11_PSILSS0_CFG_MMRS	0x4BB1170000	0x4BB1171000	4 KB
NAVSS0_VIRT_ALIAS_11_BCDMA0_CFG_GCFG	0x4BB11A0000	0x4BB11A0100	256 B
NAVSS0_VIRT_ALIAS_11_MCRC0_S_CFG_MCRC64	0x4BB1F70000	0x4BB1F71000	4 KB
NAVSS0_VIRT_ALIAS_11_PSILCFG0_CFG_PROXY	0x4BB1F78000	0x4BB1F78200	512 B
NAVSS0_VIRT_ALIAS_11_MAILBOX0_CFG_REGS0	0x4BB1F80000	0x4BB1F80200	512 B
NAVSS0_VIRT_ALIAS_11_MAILBOX0_CFG_REGS1	0x4BB1F81000	0x4BB1F81200	512 B
NAVSS0_VIRT_ALIAS_11_MAILBOX0_CFG_REGS2	0x4BB1F82000	0x4BB1F82200	512 B
NAVSS0_VIRT_ALIAS_11_MAILBOX0_CFG_REGS3	0x4BB1F83000	0x4BB1F83200	512 B
NAVSS0_VIRT_ALIAS_11_MAILBOX0_CFG_REGS4	0x4BB1F84000	0x4BB1F84200	512 B
NAVSS0_VIRT_ALIAS_11_MAILBOX0_CFG_REGS5	0x4BB1F85000	0x4BB1F85200	512 B
NAVSS0_VIRT_ALIAS_11_MAILBOX0_CFG_REGS6	0x4BB1F86000	0x4BB1F86200	512 B
NAVSS0_VIRT_ALIAS_11_MAILBOX0_CFG_REGS7	0x4BB1F87000	0x4BB1F87200	512 B
NAVSS0_VIRT_ALIAS_11_MAILBOX0_CFG_REGS8	0x4BB1F88000	0x4BB1F88200	512 B
NAVSS0_VIRT_ALIAS_11_MAILBOX0_CFG_REGS9	0x4BB1F89000	0x4BB1F89200	512 B
NAVSS0_VIRT_ALIAS_11_MAILBOX0_CFG_REGS10	0x4BB1F8A000	0x4BB1F8A200	512 B
NAVSS0_VIRT_ALIAS_11_MAILBOX0_CFG_REGS11	0x4BB1F8B000	0x4BB1F8B200	512 B
NAVSS0_VIRT_ALIAS_11_MAILBOX1_CFG_REGS0	0x4BB1F90000	0x4BB1F90200	512 B
NAVSS0_VIRT_ALIAS_11_MAILBOX1_CFG_REGS1	0x4BB1F91000	0x4BB1F91200	512 B
NAVSS0_VIRT_ALIAS_11_MAILBOX1_CFG_REGS2	0x4BB1F92000	0x4BB1F92200	512 B

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
NAVSS0_VIRT_ALIAS_11_MAILBOX1_CFG_REGS3	0x4BB1F93000	0x4BB1F93200	512 B
NAVSS0_VIRT_ALIAS_11_MAILBOX1_CFG_REGS4	0x4BB1F94000	0x4BB1F94200	512 B
NAVSS0_VIRT_ALIAS_11_MAILBOX1_CFG_REGS5	0x4BB1F95000	0x4BB1F95200	512 B
NAVSS0_VIRT_ALIAS_11_MAILBOX1_CFG_REGS6	0x4BB1F96000	0x4BB1F96200	512 B
NAVSS0_VIRT_ALIAS_11_MAILBOX1_CFG_REGS7	0x4BB1F97000	0x4BB1F97200	512 B
NAVSS0_VIRT_ALIAS_11_MAILBOX1_CFG_REGS8	0x4BB1F98000	0x4BB1F98200	512 B
NAVSS0_VIRT_ALIAS_11_MAILBOX1_CFG_REGS9	0x4BB1F99000	0x4BB1F99200	512 B
NAVSS0_VIRT_ALIAS_11_MAILBOX1_CFG_REGS10	0x4BB1F9A000	0x4BB1F9A200	512 B
NAVSS0_VIRT_ALIAS_11_MAILBOX1_CFG_REGS11	0x4BB1F9B000	0x4BB1F9B200	512 B
NAVSS0_VIRT_ALIAS_11_RINGACC0_CFG_MON	0x4BB2000000	0x4BB2020000	128 KB
NAVSS0_VIRT_ALIAS_11_TIMERMGR0_CFG_TIMERS	0x4BB2200000	0x4BB2240000	256 KB
NAVSS0_VIRT_ALIAS_11_TIMERMGR1_CFG_TIMERS	0x4BB2240000	0x4BB2280000	256 KB
NAVSS0_VIRT_ALIAS_11_SEC_PROXY0_CFG_RT	0x4BB2400000	0x4BB2600000	2 MB
NAVSS0_VIRT_ALIAS_11_SEC_PROXY0_CFG_SCFG	0x4BB2800000	0x4BB2A00000	2 MB
NAVSS0_VIRT_ALIAS_11_SEC_PROXY0_SRC_TARGET_DATA	0x4BB2C00000	0x4BB2E00000	2 MB
NAVSS0_VIRT_ALIAS_11_PROXY0_SRC_TARGET0_DATA	0x4BB3000000	0x4BB3040000	256 KB
NAVSS0_VIRT_ALIAS_11_PROXY0_CFG_BUF_CFG	0x4BB3400000	0x4BB3440000	256 KB
NAVSS0_VIRT_ALIAS_11_UDMASS_INTA0_CFG_GCNTRTI	0x4BB3800000	0x4BB3A00000	2 MB
NAVSS0_VIRT_ALIAS_11_MODSS_INTA0_CFG_INTR	0x4BB3C00000	0x4BB3C40000	256 KB
NAVSS0_VIRT_ALIAS_11_MODSS_INTA1_CFG_INTR	0x4BB3C40000	0x4BB3C80000	256 KB
NAVSS0_VIRT_ALIAS_11_UDMASS_INTA0_CFG_INTR	0x4BB3D00000	0x4BB3E00000	1 MB
NAVSS0_VIRT_ALIAS_11_UDMAP0_CFG_RCHANRT	0x4BB4000000	0x4BB4080000	512 KB
NAVSS0_VIRT_ALIAS_11_UDMAP0_CFG_TCHANRT	0x4BB5000000	0x4BB5200000	2 MB
NAVSS0_VIRT_ALIAS_11_BCDMA0_CFG_TCHAN	0x4BB5840000	0x4BB5841000	4 KB
NAVSS0_VIRT_ALIAS_11_BCDMA0_CFG_RCHAN	0x4BB5880000	0x4BB5882000	8 KB
NAVSS0_VIRT_ALIAS_11_BCDMA0_CFG_RING	0x4BB5900000	0x4BB5904000	16 KB
NAVSS0_VIRT_ALIAS_11_BCDMA0_CFG_TCHANRT	0x4BB5C00000	0x4BB5C10000	64 KB
NAVSS0_VIRT_ALIAS_11_BCDMA0_CFG_RCHANRT	0x4BB5D00000	0x4BB5D20000	128 KB
NAVSS0_VIRT_ALIAS_11_BCDMA0_CFG_RINGRT	0x4BB5E00000	0x4BB5E80000	512 KB
NAVSS0_VIRT_ALIAS_11_IO_PVU0_CFG_TLBIF_TLB	0x4BB6000000	0x4BB6040000	256 KB
NAVSS0_VIRT_ALIAS_11_IO_PVU1_CFG_TLBIF_TLB	0x4BB6040000	0x4BB6080000	256 KB
NAVSS0_VIRT_ALIAS_11_RINGACC0_SRC_FIFOS	0x4BB8000000	0x4BB8400000	4 MB
NAVSS0_VIRT_ALIAS_11_RINGACC0_CFG_RT	0x4BBC000000	0x4BBC400000	4 MB
NAVSS0_VIRT_ALIAS_12_MODSS_INTA0_CFG	0x4BC0800000	0x4BC0800020	32 B
NAVSS0_VIRT_ALIAS_12_MODSS_INTA1_CFG	0x4BC0801000	0x4BC0801020	32 B
NAVSS0_VIRT_ALIAS_12_UDMASS_INTA0_CFG	0x4BC0802000	0x4BC0802020	32 B
NAVSS0_VIRT_ALIAS_12_UDMASS_INTA0_CFG_UNMAP	0x4BC0880000	0x4BC0890000	64 KB
NAVSS0_VIRT_ALIAS_12_MODSS_INTA0_CFG_IMAP	0x4BC0900000	0x4BC0902000	8 KB
NAVSS0_VIRT_ALIAS_12_MODSS_INTA1_CFG_IMAP	0x4BC0908000	0x4BC090A000	8 KB
NAVSS0_VIRT_ALIAS_12_UDMASS_INTA0_CFG_IMAP	0x4BC0940000	0x4BC0950000	64 KB
NAVSS0_VIRT_ALIAS_12_NAV_DDR0_VIRTID_CFG_MMRS	0x4BC0A02000	0x4BC0A02100	256 B
NAVSS0_VIRT_ALIAS_12_NAV_DDR1_VIRTID_CFG_MMRS	0x4BC0A03000	0x4BC0A03100	256 B
NAVSS0_VIRT_ALIAS_12_UDMAP0_CFG_TCHAN	0x4BC0B00000	0x4BC0B20000	128 KB
NAVSS0_VIRT_ALIAS_12_UDMAP0_CFG_RCHAN	0x4BC0C00000	0x4BC0C08000	32 KB
NAVSS0_VIRT_ALIAS_12_UDMAP0_CFG_RFLOW	0x4BC0D00000	0x4BC0D04000	16 KB
NAVSS0_VIRT_ALIAS_12_SPINLOCK0_CFG	0x4BC0E00000	0x4BC0E08000	32 KB
NAVSS0_VIRT_ALIAS_12_TIMERMGR0_CFG_CONFIG	0x4BC0E80000	0x4BC0E80200	512 B

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
NAVSS0_VIRT_ALIAS_12_TIMERMGR1_CFG_CONFIG	0x4BC0E81000	0x4BC0E81200	512 B
NAVSS0_VIRT_ALIAS_12_TIMERMGR0_CFG_OES	0x4BC0F00000	0x4BC0F01000	4 KB
NAVSS0_VIRT_ALIAS_12_TIMERMGR1_CFG_OES	0x4BC0F01000	0x4BC0F02000	4 KB
NAVSS0_VIRT_ALIAS_12_IO_PVU0_CFG_MMRS	0x4BC0F80000	0x4BC0F81000	4 KB
NAVSS0_VIRT_ALIAS_12_IO_PVU1_CFG_MMRS	0x4BC0F81000	0x4BC0F82000	4 KB
NAVSS0_VIRT_ALIAS_12_PVU0_SRC_TOG_CFG	0x4BC0F90000	0x4BC0F90400	1 KB
NAVSS0_VIRT_ALIAS_12_PVU0_CFG_TOG_CFG	0x4BC0F91000	0x4BC0F91400	1 KB
NAVSS0_VIRT_ALIAS_12_ECCAGGR0	0x4BC1000000	0x4BC1000400	1 KB
NAVSS0_VIRT_ALIAS_12_UDMASS_ECCAGGR_CFG	0x4BC1001000	0x4BC1001400	1 KB
NAVSS0_VIRT_ALIAS_12_VIRTSS_ECCAGGR_CFG	0x4BC1002000	0x4BC1002400	1 KB
NAVSS0_VIRT_ALIAS_12_UDMASS_INTA0_CFG_GCNTCFG	0x4BC1040000	0x4BC1044000	16 KB
NAVSS0_VIRT_ALIAS_12_RINGACC0_CFG	0x4BC1080000	0x4BC10C0000	256 KB
NAVSS0_VIRT_ALIAS_12_REGS0_CFG_MMRS	0x4BC10C0000	0x4BC10C0100	256 B
NAVSS0_VIRT_ALIAS_12_CPTS0_S_VBUSP_CPTS_VBUSP	0x4BC10D0000	0x4BC10D0400	1 KB
NAVSS0_VIRT_ALIAS_12_INTR0_CFG_INTR_ROUTER_CFG	0x4BC10E0000	0x4BC10E4000	16 KB
NAVSS0_VIRT_ALIAS_12_UDMASS_INTA0_CFG_L2G	0x4BC1100000	0x4BC1102000	8 KB
NAVSS0_VIRT_ALIAS_12_UDMASS_INTA0_CFG_MCAST	0x4BC1110000	0x4BC1114000	16 KB
NAVSS0_VIRT_ALIAS_12_PROXY0_CFG_BUF_CFG_GCFG	0x4BC1120000	0x4BC1120100	256 B
NAVSS0_VIRT_ALIAS_12_PROXY0_CFG_BUFRAM_SLV_RAM	0x4BC1130000	0x4BC1134000	16 KB
NAVSS0_VIRT_ALIAS_12_SEC_PROXY0_CFG_MMRS	0x4BC1140000	0x4BC1140100	256 B
NAVSS0_VIRT_ALIAS_12_UDMAP0_CFG_GCFG	0x4BC1150000	0x4BC1150100	256 B
NAVSS0_VIRT_ALIAS_12_RINGACC0_CFG_GCFG	0x4BC1160000	0x4BC1160400	1 KB
NAVSS0_VIRT_ALIAS_12_PSILSS0_CFG_MMRS	0x4BC1170000	0x4BC1171000	4 KB
NAVSS0_VIRT_ALIAS_12_BCDMA0_CFG_GCFG	0x4BC11A0000	0x4BC11A0100	256 B
NAVSS0_VIRT_ALIAS_12_MCRC0_S_CFG_MCRC64	0x4BC1F70000	0x4BC1F71000	4 KB
NAVSS0_VIRT_ALIAS_12_PSILCFG0_CFG_PROXY	0x4BC1F78000	0x4BC1F78200	512 B
NAVSS0_VIRT_ALIAS_12_MAILBOX0_CFG_REGS0	0x4BC1F80000	0x4BC1F80200	512 B
NAVSS0_VIRT_ALIAS_12_MAILBOX0_CFG_REGS1	0x4BC1F81000	0x4BC1F81200	512 B
NAVSS0_VIRT_ALIAS_12_MAILBOX0_CFG_REGS2	0x4BC1F82000	0x4BC1F82200	512 B
NAVSS0_VIRT_ALIAS_12_MAILBOX0_CFG_REGS3	0x4BC1F83000	0x4BC1F83200	512 B
NAVSS0_VIRT_ALIAS_12_MAILBOX0_CFG_REGS4	0x4BC1F84000	0x4BC1F84200	512 B
NAVSS0_VIRT_ALIAS_12_MAILBOX0_CFG_REGS5	0x4BC1F85000	0x4BC1F85200	512 B
NAVSS0_VIRT_ALIAS_12_MAILBOX0_CFG_REGS6	0x4BC1F86000	0x4BC1F86200	512 B
NAVSS0_VIRT_ALIAS_12_MAILBOX0_CFG_REGS7	0x4BC1F87000	0x4BC1F87200	512 B
NAVSS0_VIRT_ALIAS_12_MAILBOX0_CFG_REGS8	0x4BC1F88000	0x4BC1F88200	512 B
NAVSS0_VIRT_ALIAS_12_MAILBOX0_CFG_REGS9	0x4BC1F89000	0x4BC1F89200	512 B
NAVSS0_VIRT_ALIAS_12_MAILBOX0_CFG_REGS10	0x4BC1F8A000	0x4BC1F8A200	512 B
NAVSS0_VIRT_ALIAS_12_MAILBOX0_CFG_REGS11	0x4BC1F8B000	0x4BC1F8B200	512 B
NAVSS0_VIRT_ALIAS_12_MAILBOX1_CFG_REGS0	0x4BC1F90000	0x4BC1F90200	512 B
NAVSS0_VIRT_ALIAS_12_MAILBOX1_CFG_REGS1	0x4BC1F91000	0x4BC1F91200	512 B
NAVSS0_VIRT_ALIAS_12_MAILBOX1_CFG_REGS2	0x4BC1F92000	0x4BC1F92200	512 B
NAVSS0_VIRT_ALIAS_12_MAILBOX1_CFG_REGS3	0x4BC1F93000	0x4BC1F93200	512 B
NAVSS0_VIRT_ALIAS_12_MAILBOX1_CFG_REGS4	0x4BC1F94000	0x4BC1F94200	512 B
NAVSS0_VIRT_ALIAS_12_MAILBOX1_CFG_REGS5	0x4BC1F95000	0x4BC1F95200	512 B
NAVSS0_VIRT_ALIAS_12_MAILBOX1_CFG_REGS6	0x4BC1F96000	0x4BC1F96200	512 B
NAVSS0_VIRT_ALIAS_12_MAILBOX1_CFG_REGS7	0x4BC1F97000	0x4BC1F97200	512 B
NAVSS0_VIRT_ALIAS_12_MAILBOX1_CFG_REGS8	0x4BC1F98000	0x4BC1F98200	512 B

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
NAVSS0_VIRT_ALIAS_12_MAILBOX1_CFG_REGS9	0x4BC1F99000	0x4BC1F99200	512 B
NAVSS0_VIRT_ALIAS_12_MAILBOX1_CFG_REGS10	0x4BC1F9A000	0x4BC1F9A200	512 B
NAVSS0_VIRT_ALIAS_12_MAILBOX1_CFG_REGS11	0x4BC1F9B000	0x4BC1F9B200	512 B
NAVSS0_VIRT_ALIAS_12_RINGACC0_CFG_MON	0x4BC2000000	0x4BC2020000	128 KB
NAVSS0_VIRT_ALIAS_12_TIMERMGR0_CFG_TIMERS	0x4BC2200000	0x4BC2240000	256 KB
NAVSS0_VIRT_ALIAS_12_TIMERMGR1_CFG_TIMERS	0x4BC2240000	0x4BC2280000	256 KB
NAVSS0_VIRT_ALIAS_12_SEC_PROXY0_CFG_RT	0x4BC2400000	0x4BC2600000	2 MB
NAVSS0_VIRT_ALIAS_12_SEC_PROXY0_CFG_SCFG	0x4BC2800000	0x4BC2A00000	2 MB
NAVSS0_VIRT_ALIAS_12_SEC_PROXY0_SRC_TARGET_DATA	0x4BC2C00000	0x4BC2E00000	2 MB
NAVSS0_VIRT_ALIAS_12_PROXY0_SRC_TARGET0_DATA	0x4BC3000000	0x4BC3040000	256 KB
NAVSS0_VIRT_ALIAS_12_PROXY0_CFG_BUF_CFG	0x4BC3400000	0x4BC3440000	256 KB
NAVSS0_VIRT_ALIAS_12_UDMASS_INTA0_CFG_GCNTRTI	0x4BC3800000	0x4BC3A00000	2 MB
NAVSS0_VIRT_ALIAS_12_MODSS_INTA0_CFG_INTR	0x4BC3C00000	0x4BC3C40000	256 KB
NAVSS0_VIRT_ALIAS_12_MODSS_INTA1_CFG_INTR	0x4BC3C40000	0x4BC3C80000	256 KB
NAVSS0_VIRT_ALIAS_12_UDMASS_INTA0_CFG_INTR	0x4BC3D00000	0x4BC3E00000	1 MB
NAVSS0_VIRT_ALIAS_12_UDMAP0_CFG_RCHANRT	0x4BC4000000	0x4BC4080000	512 KB
NAVSS0_VIRT_ALIAS_12_UDMAP0_CFG_TCHANRT	0x4BC5000000	0x4BC5200000	2 MB
NAVSS0_VIRT_ALIAS_12_BCDMA0_CFG_TCHAN	0x4BC5840000	0x4BC5841000	4 KB
NAVSS0_VIRT_ALIAS_12_BCDMA0_CFG_RCHAN	0x4BC5880000	0x4BC5882000	8 KB
NAVSS0_VIRT_ALIAS_12_BCDMA0_CFG_RING	0x4BC5900000	0x4BC5904000	16 KB
NAVSS0_VIRT_ALIAS_12_BCDMA0_CFG_TCHANRT	0x4BC5C00000	0x4BC5C10000	64 KB
NAVSS0_VIRT_ALIAS_12_BCDMA0_CFG_RCHANRT	0x4BC5D00000	0x4BC5D20000	128 KB
NAVSS0_VIRT_ALIAS_12_BCDMA0_CFG_RINGRT	0x4BC5E00000	0x4BC5E80000	512 KB
NAVSS0_VIRT_ALIAS_12_IO_PVU0_CFG_TLBIF_TLB	0x4BC6000000	0x4BC6040000	256 KB
NAVSS0_VIRT_ALIAS_12_IO_PVU1_CFG_TLBIF_TLB	0x4BC6040000	0x4BC6080000	256 KB
NAVSS0_VIRT_ALIAS_12_RINGACC0_SRC_FIFOS	0x4BC8000000	0x4BC8400000	4 MB
NAVSS0_VIRT_ALIAS_12_RINGACC0_CFG_RT	0x4BCC000000	0x4BCC400000	4 MB
NAVSS0_VIRT_ALIAS_13_MODSS_INTA0_CFG	0x4BD0800000	0x4BD0800020	32 B
NAVSS0_VIRT_ALIAS_13_MODSS_INTA1_CFG	0x4BD0801000	0x4BD0801020	32 B
NAVSS0_VIRT_ALIAS_13_UDMASS_INTA0_CFG	0x4BD0802000	0x4BD0802020	32 B
NAVSS0_VIRT_ALIAS_13_UDMASS_INTA0_CFG_UNMAP	0x4BD0880000	0x4BD0890000	64 KB
NAVSS0_VIRT_ALIAS_13_MODSS_INTA0_CFG_IMAP	0x4BD0900000	0x4BD0902000	8 KB
NAVSS0_VIRT_ALIAS_13_MODSS_INTA1_CFG_IMAP	0x4BD0908000	0x4BD090A000	8 KB
NAVSS0_VIRT_ALIAS_13_UDMASS_INTA0_CFG_IMAP	0x4BD0940000	0x4BD0950000	64 KB
NAVSS0_VIRT_ALIAS_13_NAV_DDR0_VIRTID_CFG_MMRS	0x4BD0A02000	0x4BD0A02100	256 B
NAVSS0_VIRT_ALIAS_13_NAV_DDR1_VIRTID_CFG_MMRS	0x4BD0A03000	0x4BD0A03100	256 B
NAVSS0_VIRT_ALIAS_13_UDMAP0_CFG_TCHAN	0x4BD0B00000	0x4BD0B20000	128 KB
NAVSS0_VIRT_ALIAS_13_UDMAP0_CFG_RCHAN	0x4BD0C00000	0x4BD0C08000	32 KB
NAVSS0_VIRT_ALIAS_13_UDMAP0_CFG_RFLOW	0x4BD0D00000	0x4BD0D04000	16 KB
NAVSS0_VIRT_ALIAS_13_SPINLOCK0_CFG	0x4BD0E00000	0x4BD0E08000	32 KB
NAVSS0_VIRT_ALIAS_13_TIMERMGR0_CFG_CONFIG	0x4BD0E80000	0x4BD0E80200	512 B
NAVSS0_VIRT_ALIAS_13_TIMERMGR1_CFG_CONFIG	0x4BD0E81000	0x4BD0E81200	512 B
NAVSS0_VIRT_ALIAS_13_TIMERMGR0_CFG_OES	0x4BD0F00000	0x4BD0F01000	4 KB
NAVSS0_VIRT_ALIAS_13_TIMERMGR1_CFG_OES	0x4BD0F01000	0x4BD0F02000	4 KB
NAVSS0_VIRT_ALIAS_13_IO_PVU0_CFG_MMRS	0x4BD0F80000	0x4BD0F81000	4 KB
NAVSS0_VIRT_ALIAS_13_IO_PVU1_CFG_MMRS	0x4BD0F81000	0x4BD0F82000	4 KB
NAVSS0_VIRT_ALIAS_13_PVU0_SRC_TOG_CFG	0x4BD0F90000	0x4BD0F90400	1 KB

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
NAVSS0_VIRT_ALIAS_13_PVU0_CFG_TOG_CFG	0x4BD0F91000	0x4BD0F91400	1 KB
NAVSS0_VIRT_ALIAS_13_ECCAGGR0	0x4BD1000000	0x4BD1000400	1 KB
NAVSS0_VIRT_ALIAS_13_UDMASS_ECCAGGR_CFG	0x4BD1001000	0x4BD1001400	1 KB
NAVSS0_VIRT_ALIAS_13_VIRTSS_ECCAGGR_CFG	0x4BD1002000	0x4BD1002400	1 KB
NAVSS0_VIRT_ALIAS_13_UDMASS_INTA0_CFG_GCNTCFG	0x4BD1040000	0x4BD1044000	16 KB
NAVSS0_VIRT_ALIAS_13_RINGACC0_CFG	0x4BD1080000	0x4BD10C0000	256 KB
NAVSS0_VIRT_ALIAS_13_REGS0_CFG_MMRS	0x4BD10C0000	0x4BD10C0100	256 B
NAVSS0_VIRT_ALIAS_13_CPTS0_S_VBUSP_CPTS_VBUSP	0x4BD10D0000	0x4BD10D0400	1 KB
NAVSS0_VIRT_ALIAS_13_INTR0_CFG_INTR_ROUTER_CFG	0x4BD10E0000	0x4BD10E4000	16 KB
NAVSS0_VIRT_ALIAS_13_UDMASS_INTA0_CFG_L2G	0x4BD1100000	0x4BD1102000	8 KB
NAVSS0_VIRT_ALIAS_13_UDMASS_INTA0_CFG_MCAST	0x4BD1110000	0x4BD1114000	16 KB
NAVSS0_VIRT_ALIAS_13_PROXY0_CFG_BUF_CFG_GCFG	0x4BD1120000	0x4BD1120100	256 B
NAVSS0_VIRT_ALIAS_13_PROXY0_CFG_BUFRAM_SLV_RAM	0x4BD1130000	0x4BD1134000	16 KB
NAVSS0_VIRT_ALIAS_13_SEC_PROXY0_CFG_MMRS	0x4BD1140000	0x4BD1140100	256 B
NAVSS0_VIRT_ALIAS_13_UDMAP0_CFG_GCFG	0x4BD1150000	0x4BD1150100	256 B
NAVSS0_VIRT_ALIAS_13_RINGACC0_CFG_GCFG	0x4BD1160000	0x4BD1160400	1 KB
NAVSS0_VIRT_ALIAS_13_PSILSS0_CFG_MMRS	0x4BD1170000	0x4BD1171000	4 KB
NAVSS0_VIRT_ALIAS_13_BCDMA0_CFG_GCFG	0x4BD11A0000	0x4BD11A0100	256 B
NAVSS0_VIRT_ALIAS_13_MCRC0_S_CFG_MCRC64	0x4BD1F70000	0x4BD1F71000	4 KB
NAVSS0_VIRT_ALIAS_13_PSILCFG0_CFG_PROXY	0x4BD1F78000	0x4BD1F78200	512 B
NAVSS0_VIRT_ALIAS_13_MAILBOX0_CFG_REGS0	0x4BD1F80000	0x4BD1F80200	512 B
NAVSS0_VIRT_ALIAS_13_MAILBOX0_CFG_REGS1	0x4BD1F81000	0x4BD1F81200	512 B
NAVSS0_VIRT_ALIAS_13_MAILBOX0_CFG_REGS2	0x4BD1F82000	0x4BD1F82200	512 B
NAVSS0_VIRT_ALIAS_13_MAILBOX0_CFG_REGS3	0x4BD1F83000	0x4BD1F83200	512 B
NAVSS0_VIRT_ALIAS_13_MAILBOX0_CFG_REGS4	0x4BD1F84000	0x4BD1F84200	512 B
NAVSS0_VIRT_ALIAS_13_MAILBOX0_CFG_REGS5	0x4BD1F85000	0x4BD1F85200	512 B
NAVSS0_VIRT_ALIAS_13_MAILBOX0_CFG_REGS6	0x4BD1F86000	0x4BD1F86200	512 B
NAVSS0_VIRT_ALIAS_13_MAILBOX0_CFG_REGS7	0x4BD1F87000	0x4BD1F87200	512 B
NAVSS0_VIRT_ALIAS_13_MAILBOX0_CFG_REGS8	0x4BD1F88000	0x4BD1F88200	512 B
NAVSS0_VIRT_ALIAS_13_MAILBOX0_CFG_REGS9	0x4BD1F89000	0x4BD1F89200	512 B
NAVSS0_VIRT_ALIAS_13_MAILBOX0_CFG_REGS10	0x4BD1F8A000	0x4BD1F8A200	512 B
NAVSS0_VIRT_ALIAS_13_MAILBOX0_CFG_REGS11	0x4BD1F8B000	0x4BD1F8B200	512 B
NAVSS0_VIRT_ALIAS_13_MAILBOX1_CFG_REGS0	0x4BD1F90000	0x4BD1F90200	512 B
NAVSS0_VIRT_ALIAS_13_MAILBOX1_CFG_REGS1	0x4BD1F91000	0x4BD1F91200	512 B
NAVSS0_VIRT_ALIAS_13_MAILBOX1_CFG_REGS2	0x4BD1F92000	0x4BD1F92200	512 B
NAVSS0_VIRT_ALIAS_13_MAILBOX1_CFG_REGS3	0x4BD1F93000	0x4BD1F93200	512 B
NAVSS0_VIRT_ALIAS_13_MAILBOX1_CFG_REGS4	0x4BD1F94000	0x4BD1F94200	512 B
NAVSS0_VIRT_ALIAS_13_MAILBOX1_CFG_REGS5	0x4BD1F95000	0x4BD1F95200	512 B
NAVSS0_VIRT_ALIAS_13_MAILBOX1_CFG_REGS6	0x4BD1F96000	0x4BD1F96200	512 B
NAVSS0_VIRT_ALIAS_13_MAILBOX1_CFG_REGS7	0x4BD1F97000	0x4BD1F97200	512 B
NAVSS0_VIRT_ALIAS_13_MAILBOX1_CFG_REGS8	0x4BD1F98000	0x4BD1F98200	512 B
NAVSS0_VIRT_ALIAS_13_MAILBOX1_CFG_REGS9	0x4BD1F99000	0x4BD1F99200	512 B
NAVSS0_VIRT_ALIAS_13_MAILBOX1_CFG_REGS10	0x4BD1F9A000	0x4BD1F9A200	512 B
NAVSS0_VIRT_ALIAS_13_MAILBOX1_CFG_REGS11	0x4BD1F9B000	0x4BD1F9B200	512 B
NAVSS0_VIRT_ALIAS_13_RINGACC0_CFG_MON	0x4BD2000000	0x4BD2020000	128 KB
NAVSS0_VIRT_ALIAS_13_TIMERMGR0_CFG_TIMERS	0x4BD2200000	0x4BD2240000	256 KB
NAVSS0_VIRT_ALIAS_13_TIMERMGR1_CFG_TIMERS	0x4BD2240000	0x4BD2280000	256 KB

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
NAVSS0_VIRT_ALIAS_13_SEC_PROXY0_CFG_RT	0x4BD2400000	0x4BD2600000	2 MB
NAVSS0_VIRT_ALIAS_13_SEC_PROXY0_CFG_SCFG	0x4BD2800000	0x4BD2A00000	2 MB
NAVSS0_VIRT_ALIAS_13_SEC_PROXY0_SRC_TARGET_DATA	0x4BD2C00000	0x4BD2E00000	2 MB
NAVSS0_VIRT_ALIAS_13_PROXY0_SRC_TARGET0_DATA	0x4BD3000000	0x4BD3040000	256 KB
NAVSS0_VIRT_ALIAS_13_PROXY0_CFG_BUF_CFG	0x4BD3400000	0x4BD3440000	256 KB
NAVSS0_VIRT_ALIAS_13_UDMASS_INTA0_CFG_GCNTRTI	0x4BD3800000	0x4BD3A00000	2 MB
NAVSS0_VIRT_ALIAS_13_MODSS_INTA0_CFG_INTR	0x4BD3C00000	0x4BD3C40000	256 KB
NAVSS0_VIRT_ALIAS_13_MODSS_INTA1_CFG_INTR	0x4BD3C40000	0x4BD3C80000	256 KB
NAVSS0_VIRT_ALIAS_13_UDMASS_INTA0_CFG_INTR	0x4BD3D00000	0x4BD3E00000	1 MB
NAVSS0_VIRT_ALIAS_13_UDMAP0_CFG_RCHANRT	0x4BD4000000	0x4BD4080000	512 KB
NAVSS0_VIRT_ALIAS_13_UDMAP0_CFG_TCHANRT	0x4BD5000000	0x4BD5200000	2 MB
NAVSS0_VIRT_ALIAS_13_BCDMA0_CFG_TCHAN	0x4BD5840000	0x4BD5841000	4 KB
NAVSS0_VIRT_ALIAS_13_BCDMA0_CFG_RCHAN	0x4BD5880000	0x4BD5882000	8 KB
NAVSS0_VIRT_ALIAS_13_BCDMA0_CFG_RING	0x4BD5900000	0x4BD5904000	16 KB
NAVSS0_VIRT_ALIAS_13_BCDMA0_CFG_TCHANRT	0x4BD5C00000	0x4BD5C10000	64 KB
NAVSS0_VIRT_ALIAS_13_BCDMA0_CFG_RCHANRT	0x4BD5D00000	0x4BD5D20000	128 KB
NAVSS0_VIRT_ALIAS_13_BCDMA0_CFG_RINGRT	0x4BD5E00000	0x4BD5E80000	512 KB
NAVSS0_VIRT_ALIAS_13_IO_PVU0_CFG_TLBIF_TLB	0x4BD6000000	0x4BD6040000	256 KB
NAVSS0_VIRT_ALIAS_13_IO_PVU1_CFG_TLBIF_TLB	0x4BD6040000	0x4BD6080000	256 KB
NAVSS0_VIRT_ALIAS_13_RINGACC0_SRC_FIFOS	0x4BD8000000	0x4BD8400000	4 MB
NAVSS0_VIRT_ALIAS_13_RINGACC0_CFG_RT	0x4BDC000000	0x4BDC400000	4 MB
NAVSS0_VIRT_ALIAS_14_MODSS_INTA0_CFG	0x4BE0800000	0x4BE0800020	32 B
NAVSS0_VIRT_ALIAS_14_MODSS_INTA1_CFG	0x4BE0801000	0x4BE0801020	32 B
NAVSS0_VIRT_ALIAS_14_UDMASS_INTA0_CFG	0x4BE0802000	0x4BE0802020	32 B
NAVSS0_VIRT_ALIAS_14_UDMASS_INTA0_CFG_UNMAP	0x4BE0880000	0x4BE0890000	64 KB
NAVSS0_VIRT_ALIAS_14_MODSS_INTA0_CFG_IMAP	0x4BE0900000	0x4BE0902000	8 KB
NAVSS0_VIRT_ALIAS_14_MODSS_INTA1_CFG_IMAP	0x4BE0908000	0x4BE090A000	8 KB
NAVSS0_VIRT_ALIAS_14_UDMASS_INTA0_CFG_IMAP	0x4BE0940000	0x4BE0950000	64 KB
NAVSS0_VIRT_ALIAS_14_NAV_DDR0_VIRTID_CFG_MMRS	0x4BE0A02000	0x4BE0A02100	256 B
NAVSS0_VIRT_ALIAS_14_NAV_DDR1_VIRTID_CFG_MMRS	0x4BE0A03000	0x4BE0A03100	256 B
NAVSS0_VIRT_ALIAS_14_UDMAP0_CFG_TCHAN	0x4BE0B00000	0x4BE0B20000	128 KB
NAVSS0_VIRT_ALIAS_14_UDMAP0_CFG_RCHAN	0x4BE0C00000	0x4BE0C08000	32 KB
NAVSS0_VIRT_ALIAS_14_UDMAP0_CFG_RFLOW	0x4BE0D00000	0x4BE0D04000	16 KB
NAVSS0_VIRT_ALIAS_14_SPINLOCK0_CFG	0x4BE0E00000	0x4BE0E08000	32 KB
NAVSS0_VIRT_ALIAS_14_TIMERMGR0_CFG_CONFIG	0x4BE0E80000	0x4BE0E80200	512 B
NAVSS0_VIRT_ALIAS_14_TIMERMGR1_CFG_CONFIG	0x4BE0E81000	0x4BE0E81200	512 B
NAVSS0_VIRT_ALIAS_14_TIMERMGR0_CFG_OES	0x4BE0F00000	0x4BE0F01000	4 KB
NAVSS0_VIRT_ALIAS_14_TIMERMGR1_CFG_OES	0x4BE0F01000	0x4BE0F02000	4 KB
NAVSS0_VIRT_ALIAS_14_IO_PVU0_CFG_MMRS	0x4BE0F80000	0x4BE0F81000	4 KB
NAVSS0_VIRT_ALIAS_14_IO_PVU1_CFG_MMRS	0x4BE0F81000	0x4BE0F82000	4 KB
NAVSS0_VIRT_ALIAS_14_PVU0_SRC_TOG_CFG	0x4BE0F90000	0x4BE0F90400	1 KB
NAVSS0_VIRT_ALIAS_14_PVU0_CFG_TOG_CFG	0x4BE0F91000	0x4BE0F91400	1 KB
NAVSS0_VIRT_ALIAS_14_ECCAGGR0	0x4BE1000000	0x4BE1000400	1 KB
NAVSS0_VIRT_ALIAS_14_UDMASS_ECCAGGR_CFG	0x4BE1001000	0x4BE1001400	1 KB
NAVSS0_VIRT_ALIAS_14_VIRTSS_ECCAGGR_CFG	0x4BE1002000	0x4BE1002400	1 KB
NAVSS0_VIRT_ALIAS_14_UDMASS_INTA0_CFG_GCNTCFG	0x4BE1040000	0x4BE1044000	16 KB
NAVSS0_VIRT_ALIAS_14_RINGACC0_CFG	0x4BE1080000	0x4BE10C0000	256 KB

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
NAVSS0_VIRT_ALIAS_14_REGS0_CFG_MMRS	0x4BE10C0000	0x4BE10C0100	256 B
NAVSS0_VIRT_ALIAS_14_CPTS0_S_VBUSP_CPTS_VBUSP	0x4BE10D0000	0x4BE10D0400	1 KB
NAVSS0_VIRT_ALIAS_14_INTR0_CFG_INTR_ROUTER_CFG	0x4BE10E0000	0x4BE10E4000	16 KB
NAVSS0_VIRT_ALIAS_14_UDMASS_INTA0_CFG_L2G	0x4BE1100000	0x4BE1102000	8 KB
NAVSS0_VIRT_ALIAS_14_UDMASS_INTA0_CFG_MCAST	0x4BE1110000	0x4BE1114000	16 KB
NAVSS0_VIRT_ALIAS_14_PROXY0_CFG_BUF_CFG_GCFG	0x4BE1120000	0x4BE1120100	256 B
NAVSS0_VIRT_ALIAS_14_PROXY0_CFG_BUFRAM_SLV_RAM	0x4BE1130000	0x4BE1134000	16 KB
NAVSS0_VIRT_ALIAS_14_SEC_PROXY0_CFG_MMRS	0x4BE1140000	0x4BE1140100	256 B
NAVSS0_VIRT_ALIAS_14_UDMAP0_CFG_GCFG	0x4BE1150000	0x4BE1150100	256 B
NAVSS0_VIRT_ALIAS_14_RINGACC0_CFG_GCFG	0x4BE1160000	0x4BE1160400	1 KB
NAVSS0_VIRT_ALIAS_14_PSILSS0_CFG_MMRS	0x4BE1170000	0x4BE1171000	4 KB
NAVSS0_VIRT_ALIAS_14_BCDMA0_CFG_GCFG	0x4BE11A0000	0x4BE11A0100	256 B
NAVSS0_VIRT_ALIAS_14_MCRC0_S_CFG_MCRC64	0x4BE1F70000	0x4BE1F71000	4 KB
NAVSS0_VIRT_ALIAS_14_PSILCFG0_CFG_PROXY	0x4BE1F78000	0x4BE1F78200	512 B
NAVSS0_VIRT_ALIAS_14_MAILBOX0_CFG_REGS0	0x4BE1F80000	0x4BE1F80200	512 B
NAVSS0_VIRT_ALIAS_14_MAILBOX0_CFG_REGS1	0x4BE1F81000	0x4BE1F81200	512 B
NAVSS0_VIRT_ALIAS_14_MAILBOX0_CFG_REGS2	0x4BE1F82000	0x4BE1F82200	512 B
NAVSS0_VIRT_ALIAS_14_MAILBOX0_CFG_REGS3	0x4BE1F83000	0x4BE1F83200	512 B
NAVSS0_VIRT_ALIAS_14_MAILBOX0_CFG_REGS4	0x4BE1F84000	0x4BE1F84200	512 B
NAVSS0_VIRT_ALIAS_14_MAILBOX0_CFG_REGS5	0x4BE1F85000	0x4BE1F85200	512 B
NAVSS0_VIRT_ALIAS_14_MAILBOX0_CFG_REGS6	0x4BE1F86000	0x4BE1F86200	512 B
NAVSS0_VIRT_ALIAS_14_MAILBOX0_CFG_REGS7	0x4BE1F87000	0x4BE1F87200	512 B
NAVSS0_VIRT_ALIAS_14_MAILBOX0_CFG_REGS8	0x4BE1F88000	0x4BE1F88200	512 B
NAVSS0_VIRT_ALIAS_14_MAILBOX0_CFG_REGS9	0x4BE1F89000	0x4BE1F89200	512 B
NAVSS0_VIRT_ALIAS_14_MAILBOX0_CFG_REGS10	0x4BE1F8A000	0x4BE1F8A200	512 B
NAVSS0_VIRT_ALIAS_14_MAILBOX0_CFG_REGS11	0x4BE1F8B000	0x4BE1F8B200	512 B
NAVSS0_VIRT_ALIAS_14_MAILBOX1_CFG_REGS0	0x4BE1F90000	0x4BE1F90200	512 B
NAVSS0_VIRT_ALIAS_14_MAILBOX1_CFG_REGS1	0x4BE1F91000	0x4BE1F91200	512 B
NAVSS0_VIRT_ALIAS_14_MAILBOX1_CFG_REGS2	0x4BE1F92000	0x4BE1F92200	512 B
NAVSS0_VIRT_ALIAS_14_MAILBOX1_CFG_REGS3	0x4BE1F93000	0x4BE1F93200	512 B
NAVSS0_VIRT_ALIAS_14_MAILBOX1_CFG_REGS4	0x4BE1F94000	0x4BE1F94200	512 B
NAVSS0_VIRT_ALIAS_14_MAILBOX1_CFG_REGS5	0x4BE1F95000	0x4BE1F95200	512 B
NAVSS0_VIRT_ALIAS_14_MAILBOX1_CFG_REGS6	0x4BE1F96000	0x4BE1F96200	512 B
NAVSS0_VIRT_ALIAS_14_MAILBOX1_CFG_REGS7	0x4BE1F97000	0x4BE1F97200	512 B
NAVSS0_VIRT_ALIAS_14_MAILBOX1_CFG_REGS8	0x4BE1F98000	0x4BE1F98200	512 B
NAVSS0_VIRT_ALIAS_14_MAILBOX1_CFG_REGS9	0x4BE1F99000	0x4BE1F99200	512 B
NAVSS0_VIRT_ALIAS_14_MAILBOX1_CFG_REGS10	0x4BE1F9A000	0x4BE1F9A200	512 B
NAVSS0_VIRT_ALIAS_14_MAILBOX1_CFG_REGS11	0x4BE1F9B000	0x4BE1F9B200	512 B
NAVSS0_VIRT_ALIAS_14_RINGACC0_CFG_MON	0x4BE2000000	0x4BE2020000	128 KB
NAVSS0_VIRT_ALIAS_14_TIMERMGR0_CFG_TIMERS	0x4BE2200000	0x4BE2240000	256 KB
NAVSS0_VIRT_ALIAS_14_TIMERMGR1_CFG_TIMERS	0x4BE2240000	0x4BE2280000	256 KB
NAVSS0_VIRT_ALIAS_14_SEC_PROXY0_CFG_RT	0x4BE2400000	0x4BE2600000	2 MB
NAVSS0_VIRT_ALIAS_14_SEC_PROXY0_CFG_SCFG	0x4BE2800000	0x4BE2A00000	2 MB
NAVSS0_VIRT_ALIAS_14_SEC_PROXY0_SRC_TARGET_DATA	0x4BE2C00000	0x4BE2E00000	2 MB
NAVSS0_VIRT_ALIAS_14_PROXY0_SRC_TARGET0_DATA	0x4BE3000000	0x4BE3040000	256 KB
NAVSS0_VIRT_ALIAS_14_PROXY0_CFG_BUF_CFG	0x4BE3400000	0x4BE3440000	256 KB
NAVSS0_VIRT_ALIAS_14_UDMASS_INTA0_CFG_GCNTRTI	0x4BE3800000	0x4BE3A00000	2 MB

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
NAVSS0_VIRT_ALIAS_14_MODSS_INTA0_CFG_INTR	0x4BE3C00000	0x4BE3C40000	256 KB
NAVSS0_VIRT_ALIAS_14_MODSS_INTA1_CFG_INTR	0x4BE3C40000	0x4BE3C80000	256 KB
NAVSS0_VIRT_ALIAS_14_UDMASS_INTA0_CFG_INTR	0x4BE3D00000	0x4BE3E00000	1 MB
NAVSS0_VIRT_ALIAS_14_UDMAP0_CFG_RCHANRT	0x4BE4000000	0x4BE4080000	512 KB
NAVSS0_VIRT_ALIAS_14_UDMAP0_CFG_TCHANRT	0x4BE5000000	0x4BE5200000	2 MB
NAVSS0_VIRT_ALIAS_14_BCDMA0_CFG_TCHAN	0x4BE5840000	0x4BE5841000	4 KB
NAVSS0_VIRT_ALIAS_14_BCDMA0_CFG_RCHAN	0x4BE5880000	0x4BE5882000	8 KB
NAVSS0_VIRT_ALIAS_14_BCDMA0_CFG_RING	0x4BE5900000	0x4BE5904000	16 KB
NAVSS0_VIRT_ALIAS_14_BCDMA0_CFG_TCHANRT	0x4BE5C00000	0x4BE5C10000	64 KB
NAVSS0_VIRT_ALIAS_14_BCDMA0_CFG_RCHANRT	0x4BE5D00000	0x4BE5D20000	128 KB
NAVSS0_VIRT_ALIAS_14_BCDMA0_CFG_RINGRT	0x4BE5E00000	0x4BE5E80000	512 KB
NAVSS0_VIRT_ALIAS_14_IO_PVU0_CFG_TLBIF_TLB	0x4BE6000000	0x4BE6040000	256 KB
NAVSS0_VIRT_ALIAS_14_IO_PVU1_CFG_TLBIF_TLB	0x4BE6040000	0x4BE6080000	256 KB
NAVSS0_VIRT_ALIAS_14_RINGACC0_SRC_FIFOS	0x4BE8000000	0x4BE8400000	4 MB
NAVSS0_VIRT_ALIAS_14_RINGACC0_CFG_RT	0x4BEC000000	0x4BEC400000	4 MB
NAVSS0_VIRT_ALIAS_15_MODSS_INTA0_CFG	0x4BF0800000	0x4BF0800020	32 B
NAVSS0_VIRT_ALIAS_15_MODSS_INTA1_CFG	0x4BF0801000	0x4BF0801020	32 B
NAVSS0_VIRT_ALIAS_15_UDMASS_INTA0_CFG	0x4BF0802000	0x4BF0802020	32 B
NAVSS0_VIRT_ALIAS_15_UDMASS_INTA0_CFG_UNMAP	0x4BF0880000	0x4BF0890000	64 KB
NAVSS0_VIRT_ALIAS_15_MODSS_INTA0_CFG_IMAP	0x4BF0900000	0x4BF0902000	8 KB
NAVSS0_VIRT_ALIAS_15_MODSS_INTA1_CFG_IMAP	0x4BF0908000	0x4BF090A000	8 KB
NAVSS0_VIRT_ALIAS_15_UDMASS_INTA0_CFG_IMAP	0x4BF0940000	0x4BF0950000	64 KB
NAVSS0_VIRT_ALIAS_15_NAV_DDR0_VIRTID_CFG_MMRS	0x4BF0A02000	0x4BF0A02100	256 B
NAVSS0_VIRT_ALIAS_15_NAV_DDR1_VIRTID_CFG_MMRS	0x4BF0A03000	0x4BF0A03100	256 B
NAVSS0_VIRT_ALIAS_15_UDMAP0_CFG_TCHAN	0x4BF0B00000	0x4BF0B20000	128 KB
NAVSS0_VIRT_ALIAS_15_UDMAP0_CFG_RCHAN	0x4BF0C00000	0x4BF0C08000	32 KB
NAVSS0_VIRT_ALIAS_15_UDMAP0_CFG_RFLOW	0x4BF0D00000	0x4BF0D04000	16 KB
NAVSS0_VIRT_ALIAS_15_SPINLOCK0_CFG	0x4BF0E00000	0x4BF0E08000	32 KB
NAVSS0_VIRT_ALIAS_15_TIMERMGR0_CFG_CONFIG	0x4BF0E80000	0x4BF0E80200	512 B
NAVSS0_VIRT_ALIAS_15_TIMERMGR1_CFG_CONFIG	0x4BF0E81000	0x4BF0E81200	512 B
NAVSS0_VIRT_ALIAS_15_TIMERMGR0_CFG_OES	0x4BF0F00000	0x4BF0F01000	4 KB
NAVSS0_VIRT_ALIAS_15_TIMERMGR1_CFG_OES	0x4BF0F01000	0x4BF0F02000	4 KB
NAVSS0_VIRT_ALIAS_15_IO_PVU0_CFG_MMRS	0x4BF0F80000	0x4BF0F81000	4 KB
NAVSS0_VIRT_ALIAS_15_IO_PVU1_CFG_MMRS	0x4BF0F81000	0x4BF0F82000	4 KB
NAVSS0_VIRT_ALIAS_15_PVU0_SRC_TOG_CFG	0x4BF0F90000	0x4BF0F90400	1 KB
NAVSS0_VIRT_ALIAS_15_PVU0_CFG_TOG_CFG	0x4BF0F91000	0x4BF0F91400	1 KB
NAVSS0_VIRT_ALIAS_15_ECCAGGR0	0x4BF1000000	0x4BF1000400	1 KB
NAVSS0_VIRT_ALIAS_15_UDMASS_ECCAGGR_CFG	0x4BF1001000	0x4BF1001400	1 KB
NAVSS0_VIRT_ALIAS_15_VIRTSS_ECCAGGR_CFG	0x4BF1002000	0x4BF1002400	1 KB
NAVSS0_VIRT_ALIAS_15_UDMASS_INTA0_CFG_GCNTCFG	0x4BF1040000	0x4BF1044000	16 KB
NAVSS0_VIRT_ALIAS_15_RINGACC0_CFG	0x4BF1080000	0x4BF10C0000	256 KB
NAVSS0_VIRT_ALIAS_15_REGS0_CFG_MMRS	0x4BF10C0000	0x4BF10C0100	256 B
NAVSS0_VIRT_ALIAS_15_CPTS0_S_VBUSP_CPTS_VBUSP	0x4BF10D0000	0x4BF10D0400	1 KB
NAVSS0_VIRT_ALIAS_15_INTR0_CFG_INTR_ROUTER_CFG	0x4BF10E0000	0x4BF10E4000	16 KB
NAVSS0_VIRT_ALIAS_15_UDMASS_INTA0_CFG_L2G	0x4BF1100000	0x4BF1102000	8 KB
NAVSS0_VIRT_ALIAS_15_UDMASS_INTA0_CFG_MCAST	0x4BF1110000	0x4BF1114000	16 KB
NAVSS0_VIRT_ALIAS_15_PROXY0_CFG_BUF_CFG_GCFG	0x4BF1120000	0x4BF1120100	256 B

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
NAVSS0_VIRT_ALIAS_15_PROXY0_CFG_BUFRAM_SLV_RAM	0x4BF1130000	0x4BF1134000	16 KB
NAVSS0_VIRT_ALIAS_15_SEC_PROXY0_CFG_MMRS	0x4BF1140000	0x4BF1140100	256 B
NAVSS0_VIRT_ALIAS_15_UDMAP0_CFG_GCFG	0x4BF1150000	0x4BF1150100	256 B
NAVSS0_VIRT_ALIAS_15_RINGACC0_CFG_GCFG	0x4BF1160000	0x4BF1160400	1 KB
NAVSS0_VIRT_ALIAS_15_PSILSS0_CFG_MMRS	0x4BF1170000	0x4BF1171000	4 KB
NAVSS0_VIRT_ALIAS_15_BCDMA0_CFG_GCFG	0x4BF11A0000	0x4BF11A0100	256 B
NAVSS0_VIRT_ALIAS_15_MCRC0_S_CFG_MCRC64	0x4BF1F70000	0x4BF1F71000	4 KB
NAVSS0_VIRT_ALIAS_15_PSILCFG0_CFG_PROXY	0x4BF1F78000	0x4BF1F78200	512 B
NAVSS0_VIRT_ALIAS_15_MAILBOX0_CFG_REGS0	0x4BF1F80000	0x4BF1F80200	512 B
NAVSS0_VIRT_ALIAS_15_MAILBOX0_CFG_REGS1	0x4BF1F81000	0x4BF1F81200	512 B
NAVSS0_VIRT_ALIAS_15_MAILBOX0_CFG_REGS2	0x4BF1F82000	0x4BF1F82200	512 B
NAVSS0_VIRT_ALIAS_15_MAILBOX0_CFG_REGS3	0x4BF1F83000	0x4BF1F83200	512 B
NAVSS0_VIRT_ALIAS_15_MAILBOX0_CFG_REGS4	0x4BF1F84000	0x4BF1F84200	512 B
NAVSS0_VIRT_ALIAS_15_MAILBOX0_CFG_REGS5	0x4BF1F85000	0x4BF1F85200	512 B
NAVSS0_VIRT_ALIAS_15_MAILBOX0_CFG_REGS6	0x4BF1F86000	0x4BF1F86200	512 B
NAVSS0_VIRT_ALIAS_15_MAILBOX0_CFG_REGS7	0x4BF1F87000	0x4BF1F87200	512 B
NAVSS0_VIRT_ALIAS_15_MAILBOX0_CFG_REGS8	0x4BF1F88000	0x4BF1F88200	512 B
NAVSS0_VIRT_ALIAS_15_MAILBOX0_CFG_REGS9	0x4BF1F89000	0x4BF1F89200	512 B
NAVSS0_VIRT_ALIAS_15_MAILBOX0_CFG_REGS10	0x4BF1F8A000	0x4BF1F8A200	512 B
NAVSS0_VIRT_ALIAS_15_MAILBOX0_CFG_REGS11	0x4BF1F8B000	0x4BF1F8B200	512 B
NAVSS0_VIRT_ALIAS_15_MAILBOX1_CFG_REGS0	0x4BF1F90000	0x4BF1F90200	512 B
NAVSS0_VIRT_ALIAS_15_MAILBOX1_CFG_REGS1	0x4BF1F91000	0x4BF1F91200	512 B
NAVSS0_VIRT_ALIAS_15_MAILBOX1_CFG_REGS2	0x4BF1F92000	0x4BF1F92200	512 B
NAVSS0_VIRT_ALIAS_15_MAILBOX1_CFG_REGS3	0x4BF1F93000	0x4BF1F93200	512 B
NAVSS0_VIRT_ALIAS_15_MAILBOX1_CFG_REGS4	0x4BF1F94000	0x4BF1F94200	512 B
NAVSS0_VIRT_ALIAS_15_MAILBOX1_CFG_REGS5	0x4BF1F95000	0x4BF1F95200	512 B
NAVSS0_VIRT_ALIAS_15_MAILBOX1_CFG_REGS6	0x4BF1F96000	0x4BF1F96200	512 B
NAVSS0_VIRT_ALIAS_15_MAILBOX1_CFG_REGS7	0x4BF1F97000	0x4BF1F97200	512 B
NAVSS0_VIRT_ALIAS_15_MAILBOX1_CFG_REGS8	0x4BF1F98000	0x4BF1F98200	512 B
NAVSS0_VIRT_ALIAS_15_MAILBOX1_CFG_REGS9	0x4BF1F99000	0x4BF1F99200	512 B
NAVSS0_VIRT_ALIAS_15_MAILBOX1_CFG_REGS10	0x4BF1F9A000	0x4BF1F9A200	512 B
NAVSS0_VIRT_ALIAS_15_MAILBOX1_CFG_REGS11	0x4BF1F9B000	0x4BF1F9B200	512 B
NAVSS0_VIRT_ALIAS_15_RINGACC0_CFG_MON	0x4BF2000000	0x4BF2020000	128 KB
NAVSS0_VIRT_ALIAS_15_TIMERMGR0_CFG_TIMERS	0x4BF2200000	0x4BF2240000	256 KB
NAVSS0_VIRT_ALIAS_15_TIMERMGR1_CFG_TIMERS	0x4BF2240000	0x4BF2280000	256 KB
NAVSS0_VIRT_ALIAS_15_SEC_PROXY0_CFG_RT	0x4BF2400000	0x4BF2600000	2 MB
NAVSS0_VIRT_ALIAS_15_SEC_PROXY0_CFG_SCFG	0x4BF2800000	0x4BF2A00000	2 MB
NAVSS0_VIRT_ALIAS_15_SEC_PROXY0_SRC_TARGET_DATA	0x4BF2C00000	0x4BF2E00000	2 MB
NAVSS0_VIRT_ALIAS_15_PROXY0_SRC_TARGET0_DATA	0x4BF3000000	0x4BF3040000	256 KB
NAVSS0_VIRT_ALIAS_15_PROXY0_CFG_BUF_CFG	0x4BF3400000	0x4BF3440000	256 KB
NAVSS0_VIRT_ALIAS_15_UDMASS_INTA0_CFG_GCNTRTI	0x4BF3800000	0x4BF3A00000	2 MB
NAVSS0_VIRT_ALIAS_15_MODSS_INTA0_CFG_INTR	0x4BF3C00000	0x4BF3C40000	256 KB
NAVSS0_VIRT_ALIAS_15_MODSS_INTA1_CFG_INTR	0x4BF3C40000	0x4BF3C80000	256 KB
NAVSS0_VIRT_ALIAS_15_UDMASS_INTA0_CFG_INTR	0x4BF3D00000	0x4BF3E00000	1 MB
NAVSS0_VIRT_ALIAS_15_UDMAP0_CFG_RCHANRT	0x4BF4000000	0x4BF4080000	512 KB
NAVSS0_VIRT_ALIAS_15_UDMAP0_CFG_TCHANRT	0x4BF5000000	0x4BF5200000	2 MB
NAVSS0_VIRT_ALIAS_15_BCDMA0_CFG_TCHAN	0x4BF5840000	0x4BF5841000	4 KB

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
NAVSS0_VIRT_ALIAS_15_BCDMA0_CFG_RCHAN	0x4BF5880000	0x4BF5882000	8 KB
NAVSS0_VIRT_ALIAS_15_BCDMA0_CFG_RING	0x4BF5900000	0x4BF5904000	16 KB
NAVSS0_VIRT_ALIAS_15_BCDMA0_CFG_TCHANRT	0x4BF5C00000	0x4BF5C10000	64 KB
NAVSS0_VIRT_ALIAS_15_BCDMA0_CFG_RCHANRT	0x4BF5D00000	0x4BF5D20000	128 KB
NAVSS0_VIRT_ALIAS_15_BCDMA0_CFG_RINGRT	0x4BF5E00000	0x4BF5E80000	512 KB
NAVSS0_VIRT_ALIAS_15_IO_PVU0_CFG_TLBIF_TLB	0x4BF6000000	0x4BF6040000	256 KB
NAVSS0_VIRT_ALIAS_15_IO_PVU1_CFG_TLBIF_TLB	0x4BF6040000	0x4BF6080000	256 KB
NAVSS0_VIRT_ALIAS_15_RINGACC0_SRC_FIFOS	0x4BF8000000	0x4BF8400000	4 MB
NAVSS0_VIRT_ALIAS_15_RINGACC0_CFG_RT	0x4BFC000000	0x4BFC400000	4 MB
DEBUGSS_WRAP0_ROM_TABLE_0_0	0x4C00000000	0x4C00001000	4 KB
DEBUGSS_WRAP0_RESV0_0	0x4C00001000	0x4C00002000	4 KB
DEBUGSS_WRAP0_CFGAP0	0x4C00002000	0x4C00002100	256 B
DEBUGSS_WRAP0_APBAP0	0x4C00002100	0x4C00002200	256 B
DEBUGSS_WRAP0_AXIAP0	0x4C00002200	0x4C00002300	256 B
DEBUGSS_WRAP0_PWRAP0	0x4C00002300	0x4C00002400	256 B
DEBUGSS_WRAP0_PVIEW0	0x4C00002400	0x4C00002500	256 B
DEBUGSS_WRAP0_JTAGAP0	0x4C00002500	0x4C00002600	256 B
DEBUGSS_WRAP0_SECAP0	0x4C00002600	0x4C00002700	256 B
DEBUGSS_WRAP0_CORTEX0_CFG0	0x4C00002700	0x4C00002800	256 B
DEBUGSS_WRAP0_CORTEX1_CFG0	0x4C00002800	0x4C00002900	256 B
DEBUGSS_WRAP0_CORTEX2_CFG0	0x4C00002900	0x4C00002A00	256 B
DEBUGSS_WRAP0_CORTEX3_CFG0	0x4C00002A00	0x4C00002B00	256 B
DEBUGSS_WRAP0_CORTEX4_CFG0	0x4C00002B00	0x4C00002C00	256 B
DEBUGSS_WRAP0_CORTEX5_CFG0	0x4C00002C00	0x4C00002D00	256 B
DEBUGSS_WRAP0_CORTEX6_CFG0	0x4C00002D00	0x4C00002E00	256 B
DEBUGSS_WRAP0_CORTEX7_CFG0	0x4C00002E00	0x4C00002F00	256 B
DEBUGSS_WRAP0_CORTEX8_CFG0	0x4C00002F00	0x4C00003000	256 B
DEBUGSS_WRAP0_RESV1_0	0x4C00003000	0x4C00004000	4 KB
DEBUGSS_WRAP0_RESV2_0	0x4C00004000	0x4C02004000	32 MB
DEBUGSS_WRAP0_ROM_TABLE_1_0	0x4C20000000	0x4C20001000	4 KB
DEBUGSS_WRAP0_CSCTI0	0x4C20001000	0x4C20002000	4 KB
DEBUGSS_WRAP0_DRM0	0x4C20002000	0x4C20003000	4 KB
DEBUGSS_WRAP0_RESV3_0	0x4C20003000	0x4C20004000	4 KB
DEBUGSS_WRAP0_CSTPIU0	0x4C20004000	0x4C20005000	4 KB
DEBUGSS_WRAP0_CTF0	0x4C20005000	0x4C20006000	4 KB
DEBUGSS_WRAP0_RESV4_0	0x4C20006000	0x4C21006000	16 MB
COMPUTE_CLUSTER0_CCROM	0x4C30000000	0x4C30001000	4 KB
DEBUGSS_WRAP0_EXT_APB0	0x4C30000000	0x4C40000000	256 MB
COMPUTE_CLUSTER0_CTSET	0x4C30100000	0x4C30102000	8 KB
COMPUTE_CLUSTER0_CTI0	0x4C30102000	0x4C30103000	4 KB
COMPUTE_CLUSTER0_CTI1	0x4C30103000	0x4C30104000	4 KB
COMPUTE_CLUSTER0_CTI2	0x4C30104000	0x4C30105000	4 KB
COMPUTE_CLUSTER0_CTI3	0x4C30105000	0x4C30106000	4 KB
COMPUTE_CLUSTER0_CTI4	0x4C30106000	0x4C30107000	4 KB
COMPUTE_CLUSTER0_CTI5	0x4C30107000	0x4C30108000	4 KB
COMPUTE_CLUSTER0_CTI6	0x4C30108000	0x4C30109000	4 KB
COMPUTE_CLUSTER0_CTI7	0x4C30109000	0x4C3010A000	4 KB

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
COMPUTE_CLUSTER0_AGGR0	0x4C30140000	0x4C30180000	256 KB
COMPUTE_CLUSTER0_AGGR1	0x4C30180000	0x4C301C0000	256 KB
COMPUTE_CLUSTER0_CPAC0	0x4C30400000	0x4C30800000	4 MB
COMPUTE_CLUSTER0_CPAC4	0x4C31400000	0x4C31800000	4 MB
COMPUTE_CLUSTER0_CPAC5	0x4C31800000	0x4C31C00000	4 MB
CCDEBUGSS0_ROM	0x4C3C000000	0x4C3C001000	4 KB
CCDEBUGSS0_ATB_REPLICATOR_CFG_CXATBREPLICATOR_CFG	0x4C3C004000	0x4C3C005000	4 KB
CCDEBUGSS0_TBR_VBUSP_WRAP_TBR_CFG_TBR_CFG	0x4C3C005000	0x4C3C006000	4 KB
CCDEBUGSS0_ARM_CTI_0_CFG_CSCTI_CFG	0x4C3C006000	0x4C3C007000	4 KB
CCDEBUGSS0_ARM_CTI_1_CFG_CSCTI_CFG	0x4C3C008000	0x4C3C009000	4 KB
CCDEBUGSS0_ARM_CTI_2_CFG_CSCTI_CFG	0x4C3C009000	0x4C3C00A000	4 KB
CCDEBUGSS0_ARM_CTI_3_CFG_CSCTI_CFG	0x4C3C00A000	0x4C3C00B000	4 KB
CCDEBUGSS0_ARM_CTI_4_CFG_CSCTI_CFG	0x4C3C00B000	0x4C3C00C000	4 KB
CCDEBUGSS0_ARM_CTI_5_CFG_CSCTI_CFG	0x4C3C00C000	0x4C3C00D000	4 KB
CCDEBUGSS0_ARM_CTI_6_CFG_CSCTI_CFG	0x4C3C00D000	0x4C3C00E000	4 KB
CCDEBUGSS0_ARM_CTI_7_CFG_CSCTI_CFG	0x4C3C00E000	0x4C3C00F000	4 KB
CCDEBUGSS0_ARM_CTI_8_CFG_CSCTI_CFG	0x4C3C00F000	0x4C3C010000	4 KB
CCDEBUGSS1_ROM	0x4C3C010000	0x4C3C011000	4 KB
CCDEBUGSS1_ATB_REPLICATOR_CFG_CXATBREPLICATOR_CFG	0x4C3C014000	0x4C3C015000	4 KB
CCDEBUGSS1_TBR_VBUSP_WRAP_TBR_CFG_TBR_CFG	0x4C3C015000	0x4C3C016000	4 KB
CCDEBUGSS1_ARM_CTI_0_CFG_CSCTI_CFG	0x4C3C016000	0x4C3C017000	4 KB
CCDEBUGSS1_ARM_CTI_1_CFG_CSCTI_CFG	0x4C3C018000	0x4C3C019000	4 KB
CCDEBUGSS1_ARM_CTI_2_CFG_CSCTI_CFG	0x4C3C019000	0x4C3C01A000	4 KB
CCDEBUGSS1_ARM_CTI_3_CFG_CSCTI_CFG	0x4C3C01A000	0x4C3C01B000	4 KB
CCDEBUGSS1_ARM_CTI_4_CFG_CSCTI_CFG	0x4C3C01B000	0x4C3C01C000	4 KB
CCDEBUGSS1_ARM_CTI_5_CFG_CSCTI_CFG	0x4C3C01C000	0x4C3C01D000	4 KB
CCDEBUGSS1_ARM_CTI_6_CFG_CSCTI_CFG	0x4C3C01D000	0x4C3C01E000	4 KB
CCDEBUGSS1_ARM_CTI_7_CFG_CSCTI_CFG	0x4C3C01E000	0x4C3C01F000	4 KB
CCDEBUGSS1_ARM_CTI_8_CFG_CSCTI_CFG	0x4C3C01F000	0x4C3C020000	4 KB
DEBUGSS0_ROM	0x4C3C020000	0x4C3C021000	4 KB
DEBUGSS0_CTSET2_WRAP_CFG_CTSET2_CFG	0x4C3C022000	0x4C3C024000	8 KB
DEBUGSS0_ATB_REPLICATOR_CFG_CXATBREPLICATOR_CFG	0x4C3C024000	0x4C3C025000	4 KB
DEBUGSS0_TBR_VBUSP_WRAP_TBR_CFG_TBR_CFG	0x4C3C025000	0x4C3C026000	4 KB
DEBUGSS0_ARM_CTI_0_CFG_CSCTI_CFG	0x4C3C026000	0x4C3C027000	4 KB
DEBUGSS0_ARM_CTI_1_CFG_CSCTI_CFG	0x4C3C028000	0x4C3C029000	4 KB
DEBUGSS0_ARM_CTI_2_CFG_CSCTI_CFG	0x4C3C029000	0x4C3C02A000	4 KB
DEBUGSS0_ARM_CTI_3_CFG_CSCTI_CFG	0x4C3C02A000	0x4C3C02B000	4 KB
DEBUGSS0_ARM_CTI_4_CFG_CSCTI_CFG	0x4C3C02B000	0x4C3C02C000	4 KB
DEBUGSS0_ARM_CTI_5_CFG_CSCTI_CFG	0x4C3C02C000	0x4C3C02D000	4 KB
DEBUGSS0_ARM_CTI_6_CFG_CSCTI_CFG	0x4C3C02D000	0x4C3C02E000	4 KB
DEBUGSS0_ARM_CTI_7_CFG_CSCTI_CFG	0x4C3C02E000	0x4C3C02F000	4 KB
DEBUGSS0_ARM_CTI_8_CFG_CSCTI_CFG	0x4C3C02F000	0x4C3C030000	4 KB
STM0_CXSTM	0x4C3D200000	0x4C3D201000	4 KB
STM0_CTI_CSCTI	0x4C3D201000	0x4C3D202000	4 KB
DEBUGSUSPENDRTR0_INTR_ROUTER_CFG	0x4C3D300000	0x4C3D302000	8 KB
CPT2_AGGR0_MMR	0x4C3E100000	0x4C3E100100	256 B
CPT2_AGGR0_STP2ATB_CFG	0x4C3E100100	0x4C3E100200	256 B

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
CPT2_AGGR0_MEM0	0x4C3E120000	0x4C3E121000	4 KB
CPT2_AGGR0_MEM1	0x4C3E121000	0x4C3E122000	4 KB
CPT2_AGGR0_MEM2	0x4C3E122000	0x4C3E123000	4 KB
CPT2_AGGR0_MEM3	0x4C3E123000	0x4C3E124000	4 KB
CPT2_AGGR0_MEM4	0x4C3E124000	0x4C3E125000	4 KB
CPT2_AGGR0_MEM5	0x4C3E125000	0x4C3E126000	4 KB
CPT2_AGGR0_MEM6	0x4C3E126000	0x4C3E127000	4 KB
CPT2_AGGR0_MEM7	0x4C3E127000	0x4C3E128000	4 KB
CPT2_AGGR0_MEM8	0x4C3E128000	0x4C3E129000	4 KB
CPT2_AGGR0_MEM9	0x4C3E129000	0x4C3E12A000	4 KB
CPT2_AGGR0_MEM10	0x4C3E12A000	0x4C3E12B000	4 KB
CPT2_AGGR0_MEM11	0x4C3E12B000	0x4C3E12C000	4 KB
CPT2_AGGR0_MEM12	0x4C3E12C000	0x4C3E12D000	4 KB
CPT2_AGGR0_MEM13	0x4C3E12D000	0x4C3E12E000	4 KB
CPT2_AGGR0_MEM14	0x4C3E12E000	0x4C3E12F000	4 KB
CPT2_AGGR0_MEM15	0x4C3E12F000	0x4C3E130000	4 KB
CPT2_AGGR0_MEM16	0x4C3E130000	0x4C3E131000	4 KB
CPT2_AGGR0_MEM17	0x4C3E131000	0x4C3E132000	4 KB
CPT2_AGGR0_MEM18	0x4C3E132000	0x4C3E133000	4 KB
CPT2_AGGR0_MEM19	0x4C3E133000	0x4C3E134000	4 KB
CPT2_AGGR0_MEM20	0x4C3E134000	0x4C3E135000	4 KB
CPT2_AGGR0_MEM21	0x4C3E135000	0x4C3E136000	4 KB
CPT2_AGGR0_MEM22	0x4C3E136000	0x4C3E137000	4 KB
CPT2_AGGR0_MEM23	0x4C3E137000	0x4C3E138000	4 KB
CPT2_AGGR0_MEM24	0x4C3E138000	0x4C3E139000	4 KB
CPT2_AGGR0_MEM25	0x4C3E139000	0x4C3E13A000	4 KB
CPT2_AGGR0_MEM26	0x4C3E13A000	0x4C3E13B000	4 KB
CPT2_AGGR0_MEM27	0x4C3E13B000	0x4C3E13C000	4 KB
CPT2_AGGR0_MEM28	0x4C3E13C000	0x4C3E13D000	4 KB
CPT2_AGGR0_MEM29	0x4C3E13D000	0x4C3E13E000	4 KB
CPT2_AGGR0_MEM30	0x4C3E13E000	0x4C3E13F000	4 KB
CPT2_AGGR0_MEM31	0x4C3E13F000	0x4C3E140000	4 KB
CPT2_AGGR4_MMR	0x4C3E140000	0x4C3E140100	256 B
CPT2_AGGR4_STP2ATB_CFG	0x4C3E140100	0x4C3E140200	256 B
CPT2_AGGR4_MEM0	0x4C3E160000	0x4C3E161000	4 KB
CPT2_AGGR4_MEM1	0x4C3E161000	0x4C3E162000	4 KB
CPT2_AGGR4_MEM2	0x4C3E162000	0x4C3E163000	4 KB
CPT2_AGGR4_MEM3	0x4C3E163000	0x4C3E164000	4 KB
CPT2_AGGR4_MEM4	0x4C3E164000	0x4C3E165000	4 KB
CPT2_AGGR4_MEM5	0x4C3E165000	0x4C3E166000	4 KB
CPT2_AGGR4_MEM6	0x4C3E166000	0x4C3E167000	4 KB
CPT2_AGGR4_MEM7	0x4C3E167000	0x4C3E168000	4 KB
CPT2_AGGR4_MEM8	0x4C3E168000	0x4C3E169000	4 KB
CPT2_AGGR4_MEM9	0x4C3E169000	0x4C3E16A000	4 KB
CPT2_AGGR4_MEM10	0x4C3E16A000	0x4C3E16B000	4 KB
CPT2_AGGR4_MEM11	0x4C3E16B000	0x4C3E16C000	4 KB
CPT2_AGGR4_MEM12	0x4C3E16C000	0x4C3E16D000	4 KB

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
CPT2_AGGR4_MEM13	0x4C3E16D000	0x4C3E16E000	4 KB
CPT2_AGGR4_MEM14	0x4C3E16E000	0x4C3E16F000	4 KB
CPT2_AGGR4_MEM15	0x4C3E16F000	0x4C3E170000	4 KB
CPT2_AGGR4_MEM16	0x4C3E170000	0x4C3E171000	4 KB
CPT2_AGGR4_MEM17	0x4C3E171000	0x4C3E172000	4 KB
CPT2_AGGR4_MEM18	0x4C3E172000	0x4C3E173000	4 KB
CPT2_AGGR4_MEM19	0x4C3E173000	0x4C3E174000	4 KB
CPT2_AGGR4_MEM20	0x4C3E174000	0x4C3E175000	4 KB
CPT2_AGGR4_MEM21	0x4C3E175000	0x4C3E176000	4 KB
CPT2_AGGR4_MEM22	0x4C3E176000	0x4C3E177000	4 KB
CPT2_AGGR4_MEM23	0x4C3E177000	0x4C3E178000	4 KB
CPT2_AGGR4_MEM24	0x4C3E178000	0x4C3E179000	4 KB
CPT2_AGGR4_MEM25	0x4C3E179000	0x4C3E17A000	4 KB
CPT2_AGGR4_MEM26	0x4C3E17A000	0x4C3E17B000	4 KB
CPT2_AGGR4_MEM27	0x4C3E17B000	0x4C3E17C000	4 KB
CPT2_AGGR4_MEM28	0x4C3E17C000	0x4C3E17D000	4 KB
CPT2_AGGR4_MEM29	0x4C3E17D000	0x4C3E17E000	4 KB
CPT2_AGGR4_MEM30	0x4C3E17E000	0x4C3E17F000	4 KB
CPT2_AGGR4_MEM31	0x4C3E17F000	0x4C3E180000	4 KB
CPT2_AGGR1_MMR	0x4C3E180000	0x4C3E180100	256 B
CPT2_AGGR1_STP2ATB_CFG	0x4C3E180100	0x4C3E180200	256 B
CPT2_AGGR1_MEM0	0x4C3E1A0000	0x4C3E1A1000	4 KB
CPT2_AGGR1_MEM1	0x4C3E1A1000	0x4C3E1A2000	4 KB
CPT2_AGGR1_MEM2	0x4C3E1A2000	0x4C3E1A3000	4 KB
CPT2_AGGR1_MEM3	0x4C3E1A3000	0x4C3E1A4000	4 KB
CPT2_AGGR1_MEM4	0x4C3E1A4000	0x4C3E1A5000	4 KB
CPT2_AGGR1_MEM5	0x4C3E1A5000	0x4C3E1A6000	4 KB
CPT2_AGGR1_MEM6	0x4C3E1A6000	0x4C3E1A7000	4 KB
CPT2_AGGR1_MEM7	0x4C3E1A7000	0x4C3E1A8000	4 KB
CPT2_AGGR1_MEM8	0x4C3E1A8000	0x4C3E1A9000	4 KB
CPT2_AGGR1_MEM9	0x4C3E1A9000	0x4C3E1AA000	4 KB
CPT2_AGGR1_MEM10	0x4C3E1AA000	0x4C3E1AB000	4 KB
CPT2_AGGR1_MEM11	0x4C3E1AB000	0x4C3E1AC000	4 KB
CPT2_AGGR1_MEM12	0x4C3E1AC000	0x4C3E1AD000	4 KB
CPT2_AGGR1_MEM13	0x4C3E1AD000	0x4C3E1AE000	4 KB
CPT2_AGGR1_MEM14	0x4C3E1AE000	0x4C3E1AF000	4 KB
CPT2_AGGR1_MEM15	0x4C3E1AF000	0x4C3E1B0000	4 KB
CPT2_AGGR1_MEM16	0x4C3E1B0000	0x4C3E1B1000	4 KB
CPT2_AGGR1_MEM17	0x4C3E1B1000	0x4C3E1B2000	4 KB
CPT2_AGGR1_MEM18	0x4C3E1B2000	0x4C3E1B3000	4 KB
CPT2_AGGR1_MEM19	0x4C3E1B3000	0x4C3E1B4000	4 KB
CPT2_AGGR1_MEM20	0x4C3E1B4000	0x4C3E1B5000	4 KB
CPT2_AGGR1_MEM21	0x4C3E1B5000	0x4C3E1B6000	4 KB
CPT2_AGGR1_MEM22	0x4C3E1B6000	0x4C3E1B7000	4 KB
CPT2_AGGR1_MEM23	0x4C3E1B7000	0x4C3E1B8000	4 KB
CPT2_AGGR1_MEM24	0x4C3E1B8000	0x4C3E1B9000	4 KB
CPT2_AGGR1_MEM25	0x4C3E1B9000	0x4C3E1BA000	4 KB

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
CPT2_AGGR1_MEM26	0x4C3E1BA000	0x4C3E1BB000	4 KB
CPT2_AGGR1_MEM27	0x4C3E1BB000	0x4C3E1BC000	4 KB
CPT2_AGGR1_MEM28	0x4C3E1BC000	0x4C3E1BD000	4 KB
CPT2_AGGR1_MEM29	0x4C3E1BD000	0x4C3E1BE000	4 KB
CPT2_AGGR1_MEM30	0x4C3E1BE000	0x4C3E1BF000	4 KB
CPT2_AGGR1_MEM31	0x4C3E1BF000	0x4C3E1C0000	4 KB
CPT2_AGGR3_MMR	0x4C3E1C0000	0x4C3E1C0100	256 B
CPT2_AGGR3_STP2ATB_CFG	0x4C3E1C0100	0x4C3E1C0200	256 B
CPT2_AGGR3_MEM0	0x4C3E1E0000	0x4C3E1E1000	4 KB
CPT2_AGGR3_MEM1	0x4C3E1E1000	0x4C3E1E2000	4 KB
CPT2_AGGR3_MEM2	0x4C3E1E2000	0x4C3E1E3000	4 KB
CPT2_AGGR3_MEM3	0x4C3E1E3000	0x4C3E1E4000	4 KB
CPT2_AGGR3_MEM4	0x4C3E1E4000	0x4C3E1E5000	4 KB
CPT2_AGGR3_MEM5	0x4C3E1E5000	0x4C3E1E6000	4 KB
CPT2_AGGR3_MEM6	0x4C3E1E6000	0x4C3E1E7000	4 KB
CPT2_AGGR3_MEM7	0x4C3E1E7000	0x4C3E1E8000	4 KB
CPT2_AGGR3_MEM8	0x4C3E1E8000	0x4C3E1E9000	4 KB
CPT2_AGGR3_MEM9	0x4C3E1E9000	0x4C3E1EA000	4 KB
CPT2_AGGR3_MEM10	0x4C3E1EA000	0x4C3E1EB000	4 KB
CPT2_AGGR3_MEM11	0x4C3E1EB000	0x4C3E1EC000	4 KB
CPT2_AGGR3_MEM12	0x4C3E1EC000	0x4C3E1ED000	4 KB
CPT2_AGGR3_MEM13	0x4C3E1ED000	0x4C3E1EE000	4 KB
CPT2_AGGR3_MEM14	0x4C3E1EE000	0x4C3E1EF000	4 KB
CPT2_AGGR3_MEM15	0x4C3E1EF000	0x4C3E1F0000	4 KB
CPT2_AGGR3_MEM16	0x4C3E1F0000	0x4C3E1F1000	4 KB
CPT2_AGGR3_MEM17	0x4C3E1F1000	0x4C3E1F2000	4 KB
CPT2_AGGR3_MEM18	0x4C3E1F2000	0x4C3E1F3000	4 KB
CPT2_AGGR3_MEM19	0x4C3E1F3000	0x4C3E1F4000	4 KB
CPT2_AGGR3_MEM20	0x4C3E1F4000	0x4C3E1F5000	4 KB
CPT2_AGGR3_MEM21	0x4C3E1F5000	0x4C3E1F6000	4 KB
CPT2_AGGR3_MEM22	0x4C3E1F6000	0x4C3E1F7000	4 KB
CPT2_AGGR3_MEM23	0x4C3E1F7000	0x4C3E1F8000	4 KB
CPT2_AGGR3_MEM24	0x4C3E1F8000	0x4C3E1F9000	4 KB
CPT2_AGGR3_MEM25	0x4C3E1F9000	0x4C3E1FA000	4 KB
CPT2_AGGR3_MEM26	0x4C3E1FA000	0x4C3E1FB000	4 KB
CPT2_AGGR3_MEM27	0x4C3E1FB000	0x4C3E1FC000	4 KB
CPT2_AGGR3_MEM28	0x4C3E1FC000	0x4C3E1FD000	4 KB
CPT2_AGGR3_MEM29	0x4C3E1FD000	0x4C3E1FE000	4 KB
CPT2_AGGR3_MEM30	0x4C3E1FE000	0x4C3E1FF000	4 KB
CPT2_AGGR3_MEM31	0x4C3E1FF000	0x4C3E200000	4 KB
CPT2_AGGR2_MMR	0x4C3E200000	0x4C3E200100	256 B
CPT2_AGGR2_STP2ATB_CFG	0x4C3E200100	0x4C3E200200	256 B
CPT2_AGGR2_MEM0	0x4C3E220000	0x4C3E221000	4 KB
CPT2_AGGR2_MEM1	0x4C3E221000	0x4C3E222000	4 KB
CPT2_AGGR2_MEM2	0x4C3E222000	0x4C3E223000	4 KB
CPT2_AGGR2_MEM3	0x4C3E223000	0x4C3E224000	4 KB
CPT2_AGGR2_MEM4	0x4C3E224000	0x4C3E225000	4 KB

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
CPT2_AGGR2_MEM5	0x4C3E225000	0x4C3E226000	4 KB
CPT2_AGGR2_MEM6	0x4C3E226000	0x4C3E227000	4 KB
CPT2_AGGR2_MEM7	0x4C3E227000	0x4C3E228000	4 KB
CPT2_AGGR2_MEM8	0x4C3E228000	0x4C3E229000	4 KB
CPT2_AGGR2_MEM9	0x4C3E229000	0x4C3E22A000	4 KB
CPT2_AGGR2_MEM10	0x4C3E22A000	0x4C3E22B000	4 KB
CPT2_AGGR2_MEM11	0x4C3E22B000	0x4C3E22C000	4 KB
CPT2_AGGR2_MEM12	0x4C3E22C000	0x4C3E22D000	4 KB
CPT2_AGGR2_MEM13	0x4C3E22D000	0x4C3E22E000	4 KB
CPT2_AGGR2_MEM14	0x4C3E22E000	0x4C3E22F000	4 KB
CPT2_AGGR2_MEM15	0x4C3E22F000	0x4C3E230000	4 KB
CPT2_AGGR2_MEM16	0x4C3E230000	0x4C3E231000	4 KB
CPT2_AGGR2_MEM17	0x4C3E231000	0x4C3E232000	4 KB
CPT2_AGGR2_MEM18	0x4C3E232000	0x4C3E233000	4 KB
CPT2_AGGR2_MEM19	0x4C3E233000	0x4C3E234000	4 KB
CPT2_AGGR2_MEM20	0x4C3E234000	0x4C3E235000	4 KB
CPT2_AGGR2_MEM21	0x4C3E235000	0x4C3E236000	4 KB
CPT2_AGGR2_MEM22	0x4C3E236000	0x4C3E237000	4 KB
CPT2_AGGR2_MEM23	0x4C3E237000	0x4C3E238000	4 KB
CPT2_AGGR2_MEM24	0x4C3E238000	0x4C3E239000	4 KB
CPT2_AGGR2_MEM25	0x4C3E239000	0x4C3E23A000	4 KB
CPT2_AGGR2_MEM26	0x4C3E23A000	0x4C3E23B000	4 KB
CPT2_AGGR2_MEM27	0x4C3E23B000	0x4C3E23C000	4 KB
CPT2_AGGR2_MEM28	0x4C3E23C000	0x4C3E23D000	4 KB
CPT2_AGGR2_MEM29	0x4C3E23D000	0x4C3E23E000	4 KB
CPT2_AGGR2_MEM30	0x4C3E23E000	0x4C3E23F000	4 KB
CPT2_AGGR2_MEM31	0x4C3E23F000	0x4C3E240000	4 KB
CPT2_AGGR5_MMR	0x4C3E240000	0x4C3E240100	256 B
CPT2_AGGR5_STP2ATB_CFG	0x4C3E240100	0x4C3E240200	256 B
CPT2_AGGR5_MEM0	0x4C3E260000	0x4C3E261000	4 KB
CPT2_AGGR5_MEM1	0x4C3E261000	0x4C3E262000	4 KB
CPT2_AGGR5_MEM2	0x4C3E262000	0x4C3E263000	4 KB
CPT2_AGGR5_MEM3	0x4C3E263000	0x4C3E264000	4 KB
CPT2_AGGR5_MEM4	0x4C3E264000	0x4C3E265000	4 KB
CPT2_AGGR5_MEM5	0x4C3E265000	0x4C3E266000	4 KB
CPT2_AGGR5_MEM6	0x4C3E266000	0x4C3E267000	4 KB
CPT2_AGGR5_MEM7	0x4C3E267000	0x4C3E268000	4 KB
CPT2_AGGR5_MEM8	0x4C3E268000	0x4C3E269000	4 KB
CPT2_AGGR5_MEM9	0x4C3E269000	0x4C3E26A000	4 KB
CPT2_AGGR5_MEM10	0x4C3E26A000	0x4C3E26B000	4 KB
CPT2_AGGR5_MEM11	0x4C3E26B000	0x4C3E26C000	4 KB
CPT2_AGGR5_MEM12	0x4C3E26C000	0x4C3E26D000	4 KB
CPT2_AGGR5_MEM13	0x4C3E26D000	0x4C3E26E000	4 KB
CPT2_AGGR5_MEM14	0x4C3E26E000	0x4C3E26F000	4 KB
CPT2_AGGR5_MEM15	0x4C3E26F000	0x4C3E270000	4 KB
CPT2_AGGR5_MEM16	0x4C3E270000	0x4C3E271000	4 KB
CPT2_AGGR5_MEM17	0x4C3E271000	0x4C3E272000	4 KB

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
CPT2_AGGR5_MEM18	0x4C3E272000	0x4C3E273000	4 KB
CPT2_AGGR5_MEM19	0x4C3E273000	0x4C3E274000	4 KB
CPT2_AGGR5_MEM20	0x4C3E274000	0x4C3E275000	4 KB
CPT2_AGGR5_MEM21	0x4C3E275000	0x4C3E276000	4 KB
CPT2_AGGR5_MEM22	0x4C3E276000	0x4C3E277000	4 KB
CPT2_AGGR5_MEM23	0x4C3E277000	0x4C3E278000	4 KB
CPT2_AGGR5_MEM24	0x4C3E278000	0x4C3E279000	4 KB
CPT2_AGGR5_MEM25	0x4C3E279000	0x4C3E27A000	4 KB
CPT2_AGGR5_MEM26	0x4C3E27A000	0x4C3E27B000	4 KB
CPT2_AGGR5_MEM27	0x4C3E27B000	0x4C3E27C000	4 KB
CPT2_AGGR5_MEM28	0x4C3E27C000	0x4C3E27D000	4 KB
CPT2_AGGR5_MEM29	0x4C3E27D000	0x4C3E27E000	4 KB
CPT2_AGGR5_MEM30	0x4C3E27E000	0x4C3E27F000	4 KB
CPT2_AGGR5_MEM31	0x4C3E27F000	0x4C3E280000	4 KB
DEBUGSS_WRAP0_ROM_TABLE_0_1	0x4C40000000	0x4C40001000	4 KB
DEBUGSS_WRAP0_RESV0_1	0x4C40001000	0x4C40002000	4 KB
DEBUGSS_WRAP0_CFGAP1	0x4C40002000	0x4C40002100	256 B
DEBUGSS_WRAP0_APBAP1	0x4C40002100	0x4C40002200	256 B
DEBUGSS_WRAP0_AXIAP1	0x4C40002200	0x4C40002300	256 B
DEBUGSS_WRAP0_PWRAP1	0x4C40002300	0x4C40002400	256 B
DEBUGSS_WRAP0_PVIEW1	0x4C40002400	0x4C40002500	256 B
DEBUGSS_WRAP0_JTAGAP1	0x4C40002500	0x4C40002600	256 B
DEBUGSS_WRAP0_SECAP1	0x4C40002600	0x4C40002700	256 B
DEBUGSS_WRAP0_CORTEX0_CFG1	0x4C40002700	0x4C40002800	256 B
DEBUGSS_WRAP0_CORTEX1_CFG1	0x4C40002800	0x4C40002900	256 B
DEBUGSS_WRAP0_CORTEX2_CFG1	0x4C40002900	0x4C40002A00	256 B
DEBUGSS_WRAP0_CORTEX3_CFG1	0x4C40002A00	0x4C40002B00	256 B
DEBUGSS_WRAP0_CORTEX4_CFG1	0x4C40002B00	0x4C40002C00	256 B
DEBUGSS_WRAP0_CORTEX5_CFG1	0x4C40002C00	0x4C40002D00	256 B
DEBUGSS_WRAP0_CORTEX6_CFG1	0x4C40002D00	0x4C40002E00	256 B
DEBUGSS_WRAP0_CORTEX7_CFG1	0x4C40002E00	0x4C40002F00	256 B
DEBUGSS_WRAP0_CORTEX8_CFG1	0x4C40002F00	0x4C40003000	256 B
DEBUGSS_WRAP0_RESV1_1	0x4C40003000	0x4C40004000	4 KB
DEBUGSS_WRAP0_RESV2_1	0x4C40004000	0x4C42004000	32 MB
DEBUGSS_WRAP0_ROM_TABLE_1_1	0x4C60000000	0x4C60001000	4 KB
DEBUGSS_WRAP0_CSCTI1	0x4C60001000	0x4C60002000	4 KB
DEBUGSS_WRAP0_DRM1	0x4C60002000	0x4C60003000	4 KB
DEBUGSS_WRAP0_RESV3_1	0x4C60003000	0x4C60004000	4 KB
DEBUGSS_WRAP0_CSTPIU1	0x4C60004000	0x4C60005000	4 KB
DEBUGSS_WRAP0_CTF1	0x4C60005000	0x4C60006000	4 KB
DEBUGSS_WRAP0_RESV4_1	0x4C60006000	0x4C61006000	16 MB
DEBUGSS_WRAP0_EXT_APB1	0x4C70000000	0x4C80000000	256 MB
COMPUTE_CLUSTER0_MSMC_PBIST0	0x4D10000000	0x4D10010000	64 KB
COMPUTE_CLUSTER0_MPU_PBIST0	0x4D10010000	0x4D10020000	64 KB
COMPUTE_CLUSTER0_DSP0_PBIST	0x4D10050000	0x4D10050400	1 KB
COMPUTE_CLUSTER0_DSP1_PBIST	0x4D10060000	0x4D10060400	1 KB
COMPUTE_CLUSTER0_MSMC_ECC_AGGR0	0x4D20000000	0x4D20000400	1 KB

Copyright © 2024 Texas Instruments Incorporated

Table 2-1. MAIN Memory Map (continued)

Region Name	Start Address	End Address	Size
COMPUTE_CLUSTER0_MSMC_ECC_AGGR1	0x4D20000400	0x4D20000800	1 KB
COMPUTE_CLUSTER0_MSMC_DDR_0_ECC_AGGR2	0x4D20000800	0x4D20000C00	1 KB
COMPUTE_CLUSTER0_MSMC_DDR_1_ECC_AGGR3	0x4D20000C00	0x4D20001000	1 KB
COMPUTE_CLUSTER0_MPU0_COREPAC_ECC_AGGR	0x4D20010000	0x4D20010400	1 KB
COMPUTE_CLUSTER0_MPU0_CORE0_ECC_AGGR	0x4D20010400	0x4D20010800	1 KB
COMPUTE_CLUSTER0_MPU0_CORE1_ECC_AGGR	0x4D20010800	0x4D20010C00	1 KB
COMPUTE_CLUSTER0_DSP0_ECC_AGGR	0x4D20050000	0x4D20050400	1 KB
COMPUTE_CLUSTER0_DSP1_ECCAGGR	0x4D20060000	0x4D20060400	1 KB
COMPUTE_CLUSTER0_DDR0_0_ECC_AGGR_CTL	0x4D200B0000	0x4D200B0400	1 KB
COMPUTE_CLUSTER0_DDR0_0_ECC_AGGR_VBUS	0x4D200B0400	0x4D200B0800	1 KB
COMPUTE_CLUSTER0_DDR0_0_ECC_AGGR_CFG	0x4D200B0800	0x4D200B0C00	1 KB
COMPUTE_CLUSTER0_DDR1_1_ECC_AGGR_CTL	0x4D200B0C00	0x4D200B1000	1 KB
COMPUTE_CLUSTER0_DDR1_1_ECC_AGGR_VBUS	0x4D200B1000	0x4D200B1400	1 KB
COMPUTE_CLUSTER0_DDR1_1_ECC_AGGR_CFG	0x4D200B1400	0x4D200B1800	1 KB
COMPUTE_CLUSTER0_ECC_AGGR	0x4D200C0000	0x4D200C0400	1 KB
COMPUTE_CLUSTER0_GICSS_VBUSM_GASKET_CFG_GICSS_VBUSM_GASKET_CFG	0x4D200C0400	0x4D200C0800	1 KB
COMPUTE_CLUSTER0_CC	0x4D21000000	0x4D21010000	64 KB
R5FSS0_CORE0_ICACHE	0x4E00000000	0x4E00800000	8 MB
R5FSS0_CORE0_DCACHE	0x4E00800000	0x4E01000000	8 MB
R5FSS0_CORE1_ICACHE	0x4E01000000	0x4E01800000	8 MB
R5FSS0_CORE1_DCACHE	0x4E01800000	0x4E02000000	8 MB
R5FSS1_CORE0_ICACHE	0x4E10000000	0x4E10800000	8 MB
R5FSS1_CORE0_DCACHE	0x4E10800000	0x4E11000000	8 MB
R5FSS1_CORE1_ICACHE	0x4E11000000	0x4E11800000	8 MB
R5FSS1_CORE1_DCACHE	0x4E11800000	0x4E12000000	8 MB
AEP_GPU_BXS464_WRAP0_CORE_MMRS	0x4E20000000	0x4E20080000	512 KB
VPAC0_VPAC_TOP_PAC_BASE_MEM_SLV_CBASS_STRIPE_MSRAM_SLV	0x4F00000000	0x4F00080000	512 KB
DMPAC0_DMPAC_TOP_DOF_INFRA_DMPAC_BASE_MEM_SLV_CBASS_STRIPE_MSRAM_SLV	0x4F01000000	0x4F01080000	512 KB
MSRAM_512K0_RAM	0x4F02000000	0x4F02080000	512 KB
MSRAM_512K1_RAM	0x4F02080000	0x4F02100000	512 KB
VUSR_DUAL0_VUSR_PORTAL	0x5800000000	0x5900000000	4 GB

2.2 MCU Memory Map

Table 2-2. MCU Memory Map

Region Name	Start Address	End Address	Size
MAIN2MCU_LVL_INTRTR0_CFG	0x0000A10000	0x0000A10800	2 KB
MAIN2MCU_PLS_INTRTR0_CFG	0x0000A20000	0x0000A20800	2 KB
MCU_NAVSS0_ECCAGGR0	0x0028380000	0x0028380400	1 KB
MCU_NAVSS0_UDMASS_ECCAGGR0	0x0028381000	0x0028381400	1 KB
MCU_NAVSS0_UDMASS_INTA0_CFG	0x00283C0000	0x00283C0020	32 B
MCU_NAVSS0_UDMASS_UDMAP0_CFG_RFLOW	0x0028400000	0x0028402000	8 KB
MCU_NAVSS0_UDMASS_RINGACC0_CFG	0x0028440000	0x0028480000	256 KB
MCU_NAVSS0_UDMASS_INTA0_GCNT	0x0028480000	0x0028482000	8 KB
MCU_NAVSS0_UDMASS_UDMAP0_TCHAN	0x00284A0000	0x00284A4000	16 KB

Copyright © 2024 Texas Instruments Incorporated

Table 2-2. MCU Memory Map (continued)

Region Name	Start Address	End Address	Size
MCU_NAVSS0_UDMASS_UDMAP0_RCHAN	0x00284C0000	0x00284C4000	16 KB
MCU_NAVSS0_CFG	0x0028520000	0x0028520100	256 B
MCU_NAVSS0_INTR0_CFG	0x0028540000	0x0028540800	2 KB
MCU_NAVSS0_UDMASS_INTA0_IMAP	0x0028560000	0x0028564000	16 KB
MCU_NAVSS0_UDMASS_INTA0_I2G	0x0028570000	0x0028570200	512 B
MCU_NAVSS0_UDMASS_INTA0_MCAST	0x0028580000	0x0028581000	4 KB
MCU_NAVSS0_PROXY_CFG_GCFG	0x0028590000	0x0028590100	256 B
MCU_NAVSS0_PROXY_CFG_BUF	0x00285A0000	0x00285A4000	16 KB
MCU_NAVSS0_SEC_PROXY0_CFG	0x00285B0000	0x00285B0100	256 B
MCU_NAVSS0_UDMASS_UDMAP0_CFG_GCFG	0x00285C0000	0x00285C0100	256 B
MCU_NAVSS0_UDMASS_RINGACC0_CFG_GCFG	0x00285D0000	0x00285D0400	1 KB
MCU_NAVSS0_UDMASS_PSILSS0_CFG_MMRS	0x00285E0000	0x00285E1000	4 KB
MCU_NAVSS0_MCRC	0x002A264000	0x002A265000	4 KB
MCU_NAVSS0_UDMASS_PSILSS_CFG0_PROXY	0x002A268000	0x002A268200	512 B
MCU_NAVSS0_UDMASS_RINGACC0_CFG_MON	0x002A280000	0x002A2A0000	128 KB
MCU_NAVSS0_SEC_PROXY0_CFG_RT	0x002A380000	0x002A400000	512 KB
MCU_NAVSS0_SEC_PROXY0_CFG_SCFG	0x002A400000	0x002A480000	512 KB
MCU_NAVSS0_SEC_PROXY0_TARGET_DATA	0x002A480000	0x002A500000	512 KB
MCU_NAVSS0_PROXY0_TARGET0_DATA	0x002A500000	0x002A540000	256 KB
MCU_NAVSS0_PROXY0_BUF_CFG	0x002A580000	0x002A5C0000	256 KB
MCU_NAVSS0_UDMASS_INTA0_GCNTRTI	0x002A600000	0x002A700000	1 MB
MCU_NAVSS0_UDMASS_INTA0_INTR	0x002A700000	0x002A800000	1 MB
MCU_NAVSS0_UDMASS_UDMAP_RCHANRT	0x002A800000	0x002A840000	256 KB
MCU_NAVSS0_UDMASS_UDMAP_TCHANRT	0x002AA00000	0x002AA40000	256 KB
MCU_NAVSS0_UDMASS_RINGACC0_FIFOS	0x002B000000	0x002B400000	4 MB
MCU_NAVSS0_UDMASS_RINGACC0_CFG_RT	0x002B800000	0x002BC00000	4 MB
MCU_R5FSS0_CORE0_ECC_AGGR	0x0040080000	0x0040080400	1 KB
MCU_R5FSS0_CORE1_ECC_AGGR	0x00400C0000	0x00400C0400	1 KB
MCU_R5FSS0_COMPARE_CFG	0x00400F0000	0x00400F0100	256 B
MCU_DCC0	0x0040100000	0x0040100040	64 B
MCU_DCC1	0x0040110000	0x0040110040	64 B
MCU_DCC2	0x0040120000	0x0040120040	64 B
MCU_ADC12FCC0_ADC	0x0040200000	0x0040200400	1 KB
MCU_ADC12FCC0_ADC12_FIFO_DMA	0x0040208000	0x0040208400	1 KB
MCU_ADC12FCC1_ADC	0x0040210000	0x0040210400	1 KB
MCU_ADC12FCC1_ADC12_FIFO_DMA	0x0040218000	0x0040218400	1 KB
MCU_PSRAM0_RAM	0x0040280000	0x0040280200	512 B
MCU_MCSPI0_CFG	0x0040300000	0x0040300400	1 KB
MCU_MCSPI1_CFG	0x0040310000	0x0040310400	1 KB
MCU_MCSPI2_CFG	0x0040320000	0x0040320400	1 KB
MCU_TIMER0_CFG	0x0040400000	0x0040400400	1 KB
MCU_TIMER1_CFG	0x0040410000	0x0040410400	1 KB
MCU_TIMER2_CFG	0x0040420000	0x0040420400	1 KB
MCU_TIMER3_CFG	0x0040430000	0x0040430400	1 KB
MCU_TIMER4_CFG	0x0040440000	0x0040440400	1 KB
MCU_TIMER5_CFG	0x0040450000	0x0040450400	1 KB
MCU_TIMER6_CFG	0x0040460000	0x0040460400	1 KB

Copyright © 2024 Texas Instruments Incorporated

Table 2-2. MCU Memory Map (continued)

Region Name	Start Address	End Address	Size
MCU_TIMER7_CFG	0x0040470000	0x0040470400	1 KB
MCU_TIMER8_CFG	0x0040480000	0x0040480400	1 KB
MCU_TIMER9_CFG	0x0040490000	0x0040490400	1 KB
MCU_MCAN0_MSGMEM_RAM	0x0040500000	0x0040508000	32 KB
MCU_MCAN0_SS	0x0040520000	0x0040520100	256 B
MCU_MCAN0_CFG	0x0040528000	0x0040528200	512 B
MCU_MCAN1_MSGMEM_RAM	0x0040540000	0x0040548000	32 KB
MCU_MCAN1_SS	0x0040560000	0x0040560100	256 B
MCU_MCAN1_CFG	0x0040568000	0x0040568200	512 B
MCU_RTI0_CFG	0x0040600000	0x0040600100	256 B
MCU_RTI1_CFG	0x0040610000	0x0040610100	256 B
MCU_CPSW0_ECC	0x0040709000	0x0040709400	1 KB
MCU_SA3_SS0_ECC_AGGR	0x004070C000	0x004070C400	1 KB
MCU_R5FSS0_EVNT_BUS_VBUSP_MMRS	0x004072F000	0x004072F100	256 B
MCU_TIMEOUT_64B2_CFG	0x0040730000	0x0040730400	1 KB
MCU_TIMEOUT_64B3_CFG	0x0040736000	0x0040736400	1 KB
MCU_TIMEOUT_64B4_CFG	0x0040737000	0x0040737400	1 KB
MCU_ESM0_CFG	0x0040800000	0x0040801000	4 KB
MCU_SA3_SS0_REGS	0x0040900000	0x0040901000	4 KB
MCU_SA3_SS0_MMRA	0x0040901000	0x0040901200	512 B
MCU_SA3_SS0_EIP_76	0x0040910000	0x0040910080	128 B
MCU_SA3_SS0_EIP_29T2	0x0040920000	0x0040930000	64 KB
MCU_UART0	0x0040A00000	0x0040A00200	512 B
MCU_I2C0_CFG	0x0040B00000	0x0040B00100	256 B
MCU_I2C1_CFG	0x0040B10000	0x0040B10100	256 B
MCU_I3C0_MMR_MMRVBP	0x0040B80000	0x0040B80200	512 B
MCU_I3C0_VBP2APB_WRAP_CORE_VBP_MIPI_I3C_MST	0x0040B88000	0x0040B88400	1 KB
MCU_I3C1_MMR_MMRVBP	0x0040B90000	0x0040B90200	512 B
MCU_I3C1_VBP2APB_WRAP_CORE_VBP_MIPI_I3C_MST	0x0040B98000	0x0040B98400	1 KB
MCU_EFUSE0	0x0040C00000	0x0040C00100	256 B
MCU_PLL0_CFG	0x0040D00000	0x0040D04000	16 KB
MCU_PBIST0	0x0040E00000	0x0040E00400	1 KB
MCU_PBIST2	0x0040E10000	0x0040E10400	1 KB
MCU_PBIST1	0x0040E20000	0x0040E20400	1 KB
MCU_CTRL_MMR0_CFG0	0x0040F00000	0x0040F20000	128 KB
MCU_R5FSS0_CORE0_ATCM	0x0041000000	0x0041010000	64 KB
MCU_R5FSS0_CORE0_BTCM	0x0041010000	0x0041020000	64 KB
MCU_R5FSS0_CORE1_ATCM	0x0041400000	0x0041408000	32 KB
MCU_R5FSS0_CORE1_BTCM	0x0041410000	0x0041418000	32 KB
MCU_ROM0	0x0041800000	0x0041840000	256 KB
MCU_MSRAM_1MB0_RAM	0x0041C00000	0x0041D00000	1 MB
MCU_SA3_SS0_SEC_PROXY_SRC_TARGET_DATA	0x0043600000	0x0043610000	64 KB
MCU_SA3_SS0_ECCAGGR_CFG	0x0043702000	0x0043702400	1 KB
MCU_SA3_SS0_PSILCFG_CFG_PROXY	0x0044801000	0x0044801200	512 B
MCU_SA3_SS0_PSILSS_CFG_MMRS	0x0044802000	0x0044803000	4 KB
MCU_SA3_SS0_IPCSS_SEC_PROXY_CFG_MMRS	0x0044804000	0x0044804100	256 B
MCU_SA3_SS0_IPCSS_RINGACC_CFG_GCFG	0x0044805000	0x0044805400	1 KB

Copyright © 2024 Texas Instruments Incorporated

Table 2-2. MCU Memory Map (continued)

Region Name	Start Address	End Address	Size
MCU_SA3_SS0_INTAGGR_CFG	0x0044808000	0x0044808020	32 B
MCU_SA3_SS0_INTAGGR_CFG_IMAP	0x0044809000	0x0044809400	1 KB
MCU_SA3_SS0_INTAGGR_CFG_MCAST	0x004480A000	0x004480A400	1 KB
MCU_SA3_SS0_INTAGGR_CFG_GCNTCFG	0x004480B000	0x004480B400	1 KB
MCU_SA3_SS0_INTAGGR_CFG_INTR	0x0044810000	0x0044818000	32 KB
MCU_SA3_SS0_INTAGGR_CFG_GCNTRTI	0x0044820000	0x0044840000	128 KB
MCU_SA3_SS0_INTAGGR_CFG_UNMAP	0x0044840000	0x0044850000	64 KB
MCU_SA3_SS0_IPCSS_SEC_PROXY_CFG_SCFG	0x0044860000	0x0044880000	128 KB
MCU_SA3_SS0_IPCSS_SEC_PROXY_CFG_RT	0x0044880000	0x00448A0000	128 KB
MCU_SA3_SS0_IPCSS_RINGACC_CFG	0x00448C0000	0x0044900000	256 KB
MCU_SA3_SS0_PKTDMA_CFG_GCFG	0x0044910000	0x0044910100	256 B
MCU_SA3_SS0_PKTDMA_CFG_RFLOW	0x0044911000	0x0044911400	1 KB
MCU_SA3_SS0_PKTDMA_CFG_RCHAN	0x0044912000	0x0044912400	1 KB
MCU_SA3_SS0_PKTDMA_CFG_TCHAN	0x0044913000	0x0044913200	512 B
MCU_SA3_SS0_PKTDMA_CFG_RCHANRT	0x0044914000	0x0044918000	16 KB
MCU_SA3_SS0_PKTDMA_CFG_TCHANRT	0x0044918000	0x004491A000	8 KB
MCU_SA3_SS0_PKTDMA_CFG_RING	0x004491A000	0x004491C000	8 KB
MCU_SA3_SS0_PKTDMA_CFG_RINGRT	0x0044940000	0x0044980000	256 KB
MCU_SA3_SS0_IPCSS_RINGACC_CFG_RT	0x0044C00000	0x0045000000	4 MB
MCU_CBASS0_FW	0x0045100000	0x0045180000	512 KB
MCU_NAVSS0_MODSS_DMSC_FW	0x0045600000	0x0045640000	256 KB
MCU_SA3_SS0_CRED	0x004580B000	0x004580B400	1 KB
MCU_CBASS0_ISC	0x0045810000	0x0045814000	16 KB
MCU_NAVSS0_MODSS_DMSC_ISC	0x0045818000	0x0045819000	4 KB
MCU_NAVSS0_UDMASS_RINGACC0_ISC	0x0045820000	0x0045824000	16 KB
MCU_SEC_MMR0_DBG_CTRL	0x0045950000	0x0045950400	1 KB
MCU_SEC_MMR0_CFG0	0x0045A50000	0x0045A50400	1 KB
MCU_NAVSS0_MODSS_DMSC_GLB	0x0045B04000	0x0045B04400	1 KB
MCU_CBASS0_GLB	0x0045B06000	0x0045B06400	1 KB
MCU_CBASS0_QOS	0x0045D10000	0x0045D14000	16 KB
MCU_NAVSS0_MODSS_DMSC_QOS	0x0045D20000	0x0045D21000	4 KB
MCU_CPSW0_NUSS	0x0046000000	0x0046200000	2 MB
MCU_FSS0_CFG	0x0047000000	0x0047000100	256 B
MCU_FSS0_FSAS_CFG	0x0047010000	0x0047010100	256 B
MCU_FSS0_OTFA_CFG	0x0047020000	0x0047021000	4 KB
MCU_FSS0_HPB_SS_CFG	0x0047030000	0x0047030100	256 B
MCU_FSS0_HPB_CTRL	0x0047034000	0x0047034100	256 B
MCU_FSS0_OSPI0_CTRL	0x0047040000	0x0047040100	256 B
MCU_FSS0_OSPI0_SS_CFG	0x0047044000	0x0047044200	512 B
MCU_FSS0_OSPI1_CTRL	0x0047050000	0x0047050100	256 B
MCU_FSS0_OSPI1_SS_CFG	0x0047054000	0x0047054200	512 B
MCU_FSS0_HPB_ECC_AGGR	0x0047060000	0x0047060400	1 KB
MCU_FSS0_OSPI1_ECC_AGGR	0x0047064000	0x0047064400	1 KB
MCU_FSS0_OSPI0_ECC_AGGR	0x0047068000	0x0047068400	1 KB
MCU_CBASS0_ERR	0x0047100000	0x0047100400	1 KB
MCU_CBASS_DEBUG0_ERR	0x0047104000	0x0047104400	1 KB
MCU_CBASS_FW0_ERR	0x0047108000	0x0047108400	1 KB

Copyright © 2024 Texas Instruments Incorporated

Table 2-2. MCU Memory Map (continued)

Region Name	Start Address	End Address	Size
MCU_FSS0_DAT_REG1	0x0050000000	0x0058000000	128 MB
MCU_FSS0_OSPI1_R1	0x0058000000	0x0060000000	128 MB
MCU_FSS0_DAT_REG0	0x0400000000	0x0500000000	4 GB
MCU_FSS0_DAT_REG3	0x0500000000	0x0600000000	4 GB
MCU_FSS0_OSPI1_R0	0x0600000000	0x0700000000	4 GB
MCU_FSS0_OSPI1_R3	0x0700000000	0x0800000000	4 GB
MCU_CPT2_AGGR0_MMR	0x4C3E000000	0x4C3E000100	256 B
MCU_CPT2_AGGR0_STP2ATB_CFG	0x4C3E000100	0x4C3E000200	256 B
MCU_CPT2_AGGR0_MEM0	0x4C3E020000	0x4C3E021000	4 KB
MCU_CPT2_AGGR0_MEM1	0x4C3E021000	0x4C3E022000	4 KB
MCU_CPT2_AGGR0_MEM2	0x4C3E022000	0x4C3E023000	4 KB
MCU_CPT2_AGGR0_MEM3	0x4C3E023000	0x4C3E024000	4 KB
MCU_CPT2_AGGR0_MEM4	0x4C3E024000	0x4C3E025000	4 KB
MCU_CPT2_AGGR0_MEM5	0x4C3E025000	0x4C3E026000	4 KB
MCU_CPT2_AGGR0_MEM6	0x4C3E026000	0x4C3E027000	4 KB
MCU_CPT2_AGGR0_MEM7	0x4C3E027000	0x4C3E028000	4 KB
MCU_CPT2_AGGR0_MEM8	0x4C3E028000	0x4C3E029000	4 KB
MCU_CPT2_AGGR0_MEM9	0x4C3E029000	0x4C3E02A000	4 KB
MCU_CPT2_AGGR0_MEM10	0x4C3E02A000	0x4C3E02B000	4 KB
MCU_CPT2_AGGR0_MEM11	0x4C3E02B000	0x4C3E02C000	4 KB
MCU_CPT2_AGGR0_MEM12	0x4C3E02C000	0x4C3E02D000	4 KB
MCU_CPT2_AGGR0_MEM13	0x4C3E02D000	0x4C3E02E000	4 KB
MCU_CPT2_AGGR0_MEM14	0x4C3E02E000	0x4C3E02F000	4 KB
MCU_CPT2_AGGR0_MEM15	0x4C3E02F000	0x4C3E030000	4 KB
MCU_CPT2_AGGR0_MEM16	0x4C3E030000	0x4C3E031000	4 KB
MCU_CPT2_AGGR0_MEM17	0x4C3E031000	0x4C3E032000	4 KB
MCU_CPT2_AGGR0_MEM18	0x4C3E032000	0x4C3E033000	4 KB
MCU_CPT2_AGGR0_MEM19	0x4C3E033000	0x4C3E034000	4 KB
MCU_CPT2_AGGR0_MEM20	0x4C3E034000	0x4C3E035000	4 KB
MCU_CPT2_AGGR0_MEM21	0x4C3E035000	0x4C3E036000	4 KB
MCU_CPT2_AGGR0_MEM22	0x4C3E036000	0x4C3E037000	4 KB
MCU_CPT2_AGGR0_MEM23	0x4C3E037000	0x4C3E038000	4 KB
MCU_CPT2_AGGR0_MEM24	0x4C3E038000	0x4C3E039000	4 KB
MCU_CPT2_AGGR0_MEM25	0x4C3E039000	0x4C3E03A000	4 KB
MCU_CPT2_AGGR0_MEM26	0x4C3E03A000	0x4C3E03B000	4 KB
MCU_CPT2_AGGR0_MEM27	0x4C3E03B000	0x4C3E03C000	4 KB
MCU_CPT2_AGGR0_MEM28	0x4C3E03C000	0x4C3E03D000	4 KB
MCU_CPT2_AGGR0_MEM29	0x4C3E03D000	0x4C3E03E000	4 KB
MCU_CPT2_AGGR0_MEM30	0x4C3E03E000	0x4C3E03F000	4 KB
MCU_CPT2_AGGR0_MEM31	0x4C3E03F000	0x4C3E040000	4 KB
MCU_R5FSS0_CORE0_ICACHE	0x5400000000	0x5400800000	8 MB
MCU_R5FSS0_CORE0_DCACHE	0x5400800000	0x5401000000	8 MB
MCU_R5FSS0_CORE1_ICACHE	0x5401000000	0x5401800000	8 MB
MCU_R5FSS0_CORE1_DCACHE	0x5401800000	0x5402000000	8 MB

Copyright © 2024 Texas Instruments Incorporated

2.3 WKUP Memory Map

Table 2-3. WKUP Memory Map

Region Name	Start Address	End Address	Size
WKUP_PSC0	0x0042000000	0x0042001000	4 KB
WKUP_PLLCTRL0	0x0042010000	0x0042010200	512 B
WKUP_VTM0_MMR_VBUSP_CFG1	0x0042040000	0x0042040400	1 KB
WKUP_VTM0_MMR_VBUSP_CFG2	0x0042050000	0x0042050400	1 KB
WKUP_DDPA0	0x0042060000	0x0042060400	1 KB
WKUP_ESM0_CFG	0x0042080000	0x0042081000	4 KB
WKUP_GPIO1	0x0042100000	0x0042100100	256 B
WKUP_GPIO0	0x0042110000	0x0042110100	256 B
WKUP_I2C0_CFG	0x0042120000	0x0042120100	256 B
WKUP_GPIOMUX_INTRTR0_CFG	0x0042200000	0x0042200400	1 KB
WKUP_UART0	0x0042300000	0x0042300200	512 B
WKUP_CBASS0_ERR	0x0042400000	0x0042400400	1 KB
WKUP_FW_CBASS0_ERR	0x0042404000	0x0042404400	1 KB
WKUP_CTRL_MMR0_CFG0	0x0043000000	0x0043020000	128 KB
WKUP_SMS0_ECC_AGGR	0x0043700000	0x0043700400	1 KB
WKUP_SMS0_HSM_ECC	0x0043701000	0x0043701400	1 KB
WKUP_SMS0_TIFS_DMSS_HSM_ECC	0x0043702000	0x0043702400	1 KB
WKUP_SMS0_HSM_WDT_RTI	0x0043935000	0x0043935100	256 B
WKUP_SMS0_HSM_CTRL_MMR	0x0043936000	0x0043937000	4 KB
WKUP_SMS0_HSM_RAT_MMRS	0x0043A00000	0x0043A01000	4 KB
WKUP_SMS0_HSM_SRAM0_0	0x0043C00000	0x0043C20000	128 KB
WKUP_SMS0_HSM_SRAM0_1	0x0043C20000	0x0043C30000	64 KB
WKUP_SMS0_HSM_SRAM1	0x0043C30000	0x0043C40000	64 KB
WKUP_SMS0_TIFS_SRAM0	0x0044040000	0x0044060000	128 KB
WKUP_SMS0_TIFS_SRAM1_0	0x0044060000	0x0044068000	32 KB
WKUP_SMS0_TIFS_SRAM1_1	0x0044068000	0x004406C000	16 KB
WKUP_SMS0_PWR	0x0044130000	0x0044130800	2 KB
WKUP_SMS0_DMTIMER0	0x0044133000	0x0044133400	1 KB
WKUP_SMS0_DMTIMER1	0x0044134000	0x0044134400	1 KB
WKUP_SMS0_WDT_RTI	0x0044135000	0x0044135100	256 B
WKUP_SMS0_RTI	0x0044135100	0x0044135200	256 B
WKUP_SMS0_RAT	0x0044200000	0x0044201000	4 KB
WKUP_SMS0_SEC	0x0044230000	0x0044231000	4 KB
WKUP_SMS0_SECMGR	0x0044234000	0x0044238000	16 KB
WKUP_SMS0_DMTIMER2	0x0044238000	0x0044238400	1 KB
WKUP_SMS0_DMTIMER3	0x0044239000	0x0044239400	1 KB
WKUP_SMS0_AES	0x004423C000	0x004423E000	8 KB
WKUP_SMS0_TIFS_DMSS_HSM	0x0044800000	0x0045000000	8 MB
WKUP_SMS0_FW	0x0045000000	0x0046000000	16 MB
WKUP_CBASS0_FW	0x0045020000	0x0045030000	64 KB
WKUP_SMS0_CBASS_FW	0x0045080000	0x00450A0000	128 KB
WKUP_SMS0_HSM_CBASS_FW	0x00450A0000	0x00450B0000	64 KB
WKUP_SMS0_CBASS_ISC	0x0045808000	0x0045809000	4 KB
WKUP_SMS0_HSM_CBASS_ISC	0x004580A000	0x004580B000	4 KB
WKUP_SMS0_DMSS_HSM_FWMGR_CFG	0x004580B000	0x004580B400	1 KB

Copyright © 2024 Texas Instruments Incorporated

Table 2-3. WKUP Memory Map (continued)

Region Name	Start Address	End Address	Size
WKUP_SMS0_CBASS_GLB	0x0045B00000	0x0045B00400	1 KB
WKUP_SMS0_HSM_CBASS_GLB	0x0045B00800	0x0045B00C00	1 KB
WKUP_CBASS0_GLB	0x0045B02000	0x0045B02400	1 KB
WKUP_CBASS0_QOS	0x0045D00000	0x0045D00800	2 KB

2.4 Processors View Memory Map

Section 2.1 through Section 2.3 show the processors view memory maps.

Note

The memory locations not shown in Section 2.1 through Section 2.3 are either unallocated or reserved and not used. Accesses to these locations are not recommended and should be avoided.

2.4.1 COMPUTE_CLUSTER0 Memory Map

Table 2-4. COMPUTE_CLUSTER0 Memory Map

Region Name	Start Address	End Address	Size
DDR0_1_ECC_AGGR_VBUS	0x0000000000	0x0000000003	4 B

2.4.2 DMPAC0 Memory Map

Table 2-5. DMPAC0 Memory Map

Region Name	Start Address	End Address	Size
KSBUS_MSRAM0_SLV_RAM	0x0000000000	0x0000000003	4 B
KSBUS_MSRAM1_SLV_RAM	0x0000000000	0x0000000003	4 B
KSBUS_MSRAM2_SLV_RAM	0x0000000000	0x0000000003	4 B
KSBUS_MSRAM3_SLV_RAM	0x0000000000	0x0000000003	4 B

2.4.3 R5FSS0 Memory Map

Table 2-6. R5FSS0 Memory Map

Region Name	Start Address	End Address	Size
ATCM	0x0000000000	0x0000007FFF	32 KB
RAT_REGION0	0x0000008000	0x0001FFFFFF	32 MB
NON_RAT_SOC_REGION0	0x0002000000	0x0003FFFFFF	32 MB
RAT_REGION1	0x0004000000	0x000EFFFFFF	176 MB
NON_RAT_SOC_REGION1	0x000F000000	0x000FF7FFFF	16 MB
VIC_CFG	0x000FF80000	0x000FF83FFF	16 KB
NON_RAT_SOC_REGION2	0x000FF84000	0x000FF8FFFF	48 KB
RAT_CFG	0x000FF90000	0x000FF90FFF	4 KB
NON_RAT_SOC_REGION3	0x000FF91000	0x000FFFFFFF	444 KB
RAT_REGION2	0x0010000000	0x004100FFFF	784 MB
BTCM	0x0041010000	0x0041017FFF	32 KB
RAT_REGION3	0x0041018000	0x007FFFFFFF	1008 MB
RAT_REGION4	0x0080000000	0x00FFFFFFFF	2 GB

Copyright © 2024 Texas Instruments Incorporated

2.4.4 R5FSS1 Memory Map

Table 2-7. R5FSS1 Memory Map

Region Name	Start Address	End Address	Size
ATCM	0x0000000000	0x0000007FFF	32 KB
RAT_REGION0	0x0000008000	0x0001FFFFFF	32 MB
NON_RAT_SOC_REGION0	0x0002000000	0x0003FFFFFF	32 MB
RAT_REGION1	0x0004000000	0x000EFFFFFF	176 MB
NON_RAT_SOC_REGION1	0x000F000000	0x000FF7FFFF	16 MB
VIC_CFG	0x000FF80000	0x000FF83FFF	16 KB
NON_RAT_SOC_REGION2	0x000FF84000	0x000FF8FFFF	48 KB
RAT_CFG	0x000FF90000	0x000FF90FFF	4 KB
NON_RAT_SOC_REGION3	0x000FF91000	0x000FFFFFFF	444 KB
RAT_REGION2	0x0010000000	0x004100FFFF	784 MB
BTCM	0x0041010000	0x0041017FFF	32 KB
RAT_REGION3	0x0041018000	0x007FFFFFFF	1008 MB
RAT_REGION4	0x0080000000	0x00FFFFFFFF	2 GB

2.4.5 MCU_NAVSS0 Memory Map

Table 2-8. MCU_NAVSS0 Memory Map

Region Name	Start Address	End Address	Size
MSRAM1_SLV_RAM	0x0000000000	0x0000000003	4 B

2.4.6 MCU_R5FSS0 Memory Map

Table 2-9. MCU_R5FSS0 Memory Map

Region Name	Start Address	End Address	Size
ATCM	0x0000000000	0x0000007FFF	32 KB
RAT_REGION0	0x0000008000	0x0040007FFF	1 GB
NON_RAT_SOC_REGION0	0x0040008000	0x0040F87FFF	16 MB
VIC_CFG	0x0040F80000	0x0040F83FFF	16 KB
NON_RAT_SOC_REGION1	0x0040F84000	0x0040F8FFFF	48 KB
RAT_CFG	0x0040F90000	0x0040F90FFF	4 KB
NON_RAT_SOC_REGION2	0x0040F91000	0x0040FFFFFF	444 KB
RAT_REGION1	0x0041000000	0x004100FFFF	64 KB
BTCM	0x0041010000	0x0041017FFF	32 KB
RAT_REGION3	0x0041018000	0x007FFFFFFF	1008 MB
RAT_REGION4	0x0080000000	0x00FFFFFFFF	2 GB

2.4.7 MCU_SA3_SS0 Memory Map

Table 2-10. MCU_SA3_SS0 Memory Map

Region Name	Start Address	End Address	Size
RINGACC__SRC__FIFOS	0x0000400000	0x0000405FFF	24 KB

Copyright © 2024 Texas Instruments Incorporated

2.4.8 WKUP_SMS0 Memory Map

Table 2-11. WKUP_SMS0 Memory Map

Region Name	Start Address	End Address	Size
HSM_WDT_RTI	0x0043935000	0x00439350FF	256 B

2.5 Region-based Address Translation

The SoC memory map view shown in this section are used by the processors in COMPUTE_CLUSTER0 but there are processors which support only 32-bit addressing and cannot access higher address ranges. For these processors, RAT modules are integrated to remap the 32-bit addresses to 48-bit addresses allowing higher address range accesses. The "RAT_REGION" regions shown in *RAT Registers* are dedicated to RAT remapping.

3 System Interconnect

The following sections describe the device system interconnect.

3.1 System Interconnect Overview

All modules and subsystems in the device communicate with each other through the system interconnect for any memory map accesses. It is partitioned into the following sections:

- MAIN_CBASS Group - located in VD_CORE and connect most of the device modules in the device MAIN domain.
- INFRA_CBASS0 and INFRA_NS_CBASS0 - located in VD_CORE and contains almost all infrastructure and internal diagnostic components. It also contains peripherals not power managed in the MAIN domain.
- MCU_CBASS0 - located in VD_MCU and connects modules from the MCU domain.
- WKUP_CBASS0 - located in VD_WKUP and connects modules in the WKUP domain.

These interconnects are used for data transfers and configuration. They are composed by switch fabrics enabling fast internal data movement. They also provide low-latency and concurrent data transfers between initiator and target peripherals.

The MAIN_CBASS GROUP is composed of the following interconnects:

- CBASS_IPPHY_S0
- CBASS_IPPHY
- CBASS_RC0
- CBASS_RC_CFG_B0
- CBASS_CSI0
- CBASS_DEBUG0
- CBASS_HC2_0
- CBASS_HC_CFG_B0
- CBASS_AC0
- CBASS_AC_CFG0
- CBASS_PULSAR_SLV0
- CBASS_AC_CFG0
- CBASS_PULSAR0_MEM0
- CBASS_PULSAR1_MEM0

Figure 3-1 shows the device system interconnect. All modules and subsystems can be classified into two categories: initiators and targets. The initiators are capable of initiating read and write transfers in the system.

Copyright © 2024 Texas Instruments Incorporated

The targets on the other hand depend on the initiators to perform transfers to and from them. They cannot generate read/write requests but can respond to these requests generating interrupts or DMA requests.

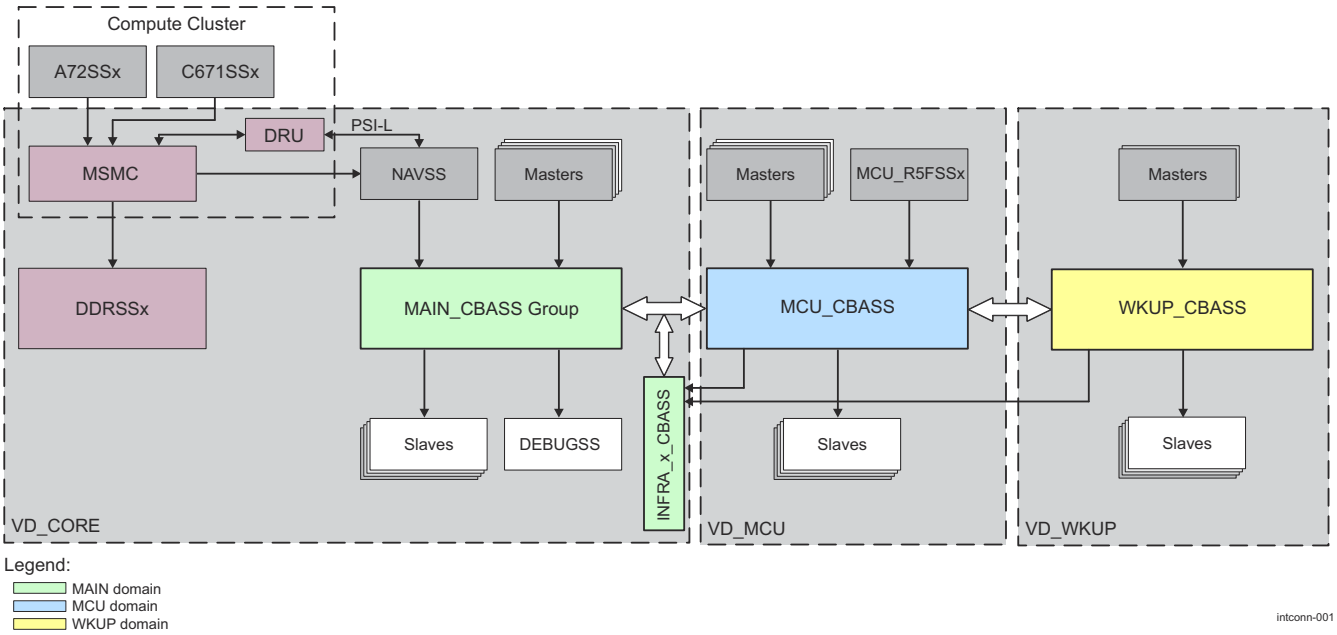


Figure 3-1. Device System Interconnect Overview

3.2 System Interconnect Functional Description

3.2.1 Quality of Service (QoS)

The device interconnect provides two mechanisms to achieve quality of service (QoS): parallel routing based on order ID and arbitration based on priority. The order ID has two main usages: to order execution of transactions and parallel paths selection. The parallel paths are defined as multiple paths between master and slave. If two or more slaves share same physical memory map assignment, it is considered parallel paths.

The CBASS_MAP_i[7-4] ORDERID bit field is for transactions for each master and each channel of the master. This allows different mapping not only for each master, but also for each channel of the master, if multiple channels are supported. The CBASS_MAP_i[7-4] ORDERID bit field specifies the initial order ID value. The ORDERIDx bit fields in CBASS_GRP_MAP1_j and CBASS_GRP_MAP2_j registers take the initial order ID value and result in a final order ID value, which is nothing more than a 4-bit identifier used for path routing. The final order ID is used to differentiate the paths to a slave and thus allowing load balancing of the traffic over each path.

The interconnect performs traffic routing and switching from a number of masters to a number of slaves. The masters send transactions. The interconnect decodes which slave is being accessed based on special signals and memory map and routes that transaction to the slave. The interconnect also arbitrates for a slave when there are multiple transactions being requested simultaneously. The arbitration is based on transaction priority. Once a transaction is selected, the interconnect sends the transaction to the slave. The transaction priority is controlled per master and per channel of the master through the CBASS_MAP_i[14-12] EPRIORITY bit field. The priority setting for each master can be done by any processor. Increasing the priority level for certain master can help the transaction from that master to have advantage on winning arbitration, therefore improving the latency and bandwidth for the transactions from that master.

Since the order ID is controlled from configuration registers, the user can re-partition the traffic among parallel paths to achieve better load balance adjusted for specific use cases. As order ID has routing implication for both command and return data, its configuration must not be changed during run time. Otherwise, the return data may be routed to wrong path and cause system hang. To avoid this, the system must be put into idle state with no traffic on the interconnect before reprogramming procedure.

Copyright © 2024 Texas Instruments Incorporated

Note

Route ID is not related to QoS. For route ID description, see Section 3.2.2.

Note

There are no master modules on the INFRA_CBASS0 and INFRA_NS_CBASS0 interconnects.

The interconnect inside NAVSS0 uses order ID to provide multiple (at least two) parallel paths to DDR and a separate set of multiple (at least two) parallel paths to SRAM. NAVSS0 also provides multiple (at least two) parallel paths for the DMA traffic to SoC level, which can provide isolated DMA traffic paths. For more information, see *Navigator Subsystem (NAVSS)*.

The MSMC does not use order ID for routing purpose to provide separate physical paths for transaction. Instead, it provides two threads to isolate two classes of transactions: thread 0 and thread 2. The arbitration between these two threads is based on credits, and thread 2 has priority over thread 0 when both threads have credits available for transfer. In addition, there is bandwidth management scheme based on transaction priority and bandwidth starvation prevention mechanism. For more information, see *Multicore Shared Memory Controller (MSMC)*.

By default all masters send transactions with order ID = 0. All transactions with the same order ID execute in order if they are sent to the same slave or going through a common bridge. On the SoC level interconnect, the order ID is used to partition the transactions to MSMC and DDR data space into parallel routing paths. Further, the north bridge inside NAVSS0 provides multiple parallel paths to the compute cluster. Each path is separated by order ID value. All read commands towards MSMC and DDR sharing the same order ID and sharing the same master path from SoC side (including NAVSS0) provide read return data in order back to the master port. The write response can be returned out of order for the same order ID value. Therefore, programming order ID has implications on overall system performance as well as achieving QoS for certain class of traffic. Multiple configurations are needed to make sure that the QoS goal is met.

For any write to address range 0x4500_0000 to 0x45FF_FFFF, it is recommended to read back the value after the write to make sure the write landed. The registers in this regions are mainly for firewall configuration, ISC/DMA credential configuration, QoS MMR configuration.

On QoS MMR configuration, in order to configure certain transaction to be the highest priority, the priority field needs to be set to zero for that transaction. In order to make sure that priority field is indeed to set to zero, the following sequence should be followed:

1. Read the priority field, if it is non-zero value, follow a write to overwrite priority field to be 0x0.
2. If read back priority field is zero, write to QoS MMR register to set priority field to be 0x7. Read back the same priority field. If the read back value is 0x7, write to Qos MMR register to set priority field to be 0x0. If the read back value is 0x0, it means this register is not implemented and priority setting is not working.

Note

See Appendix Spreadsheet for CBASS0 QoS MMR details.

3.2.2 Route ID

The route ID is used by the interconnect to route return status and data back to the transaction initiator. Each master has one or multiple unique route IDs assigned to it. The route ID has no other usage. There are no software controls for the route ID.

Note

See Appendix Spreadsheet for Route ID details.

Copyright © 2024 Texas Instruments Incorporated

3.2.3 Initiator-Side Security Controls (ISC)

Initiator-Side Security Control (ISC) modules are hosted at the initiator side where the transactions are sourced. ISC controls the security sideband attributes that are applied to outgoing transactions and have ability to override security values under exclusive control of DMSC.

ISC Capabilities:

- Attach Priv-ID to each bus transaction.
- Priv-ID is a source ID attached to identify the source of a transaction in the system.
- By default, each initiator in the system has a unique Priv-ID.
- Multiple modules can be assigned the same Priv-ID to form logical groups.
- Assert, De-assert or pass-through secure bit.
- Assert, De-assert or pass through Priv-bits.

ISCs are connected to the dedicated security VBUSP interconnect and can be exclusively configured by DMSC. ISC optionally can support region and channel number based security control, where, based on incoming channel or address, the associated security controls, are applied.

Copyright © 2024 Texas Instruments Incorporated

Figure 3-2 presents ISC config registers per region.

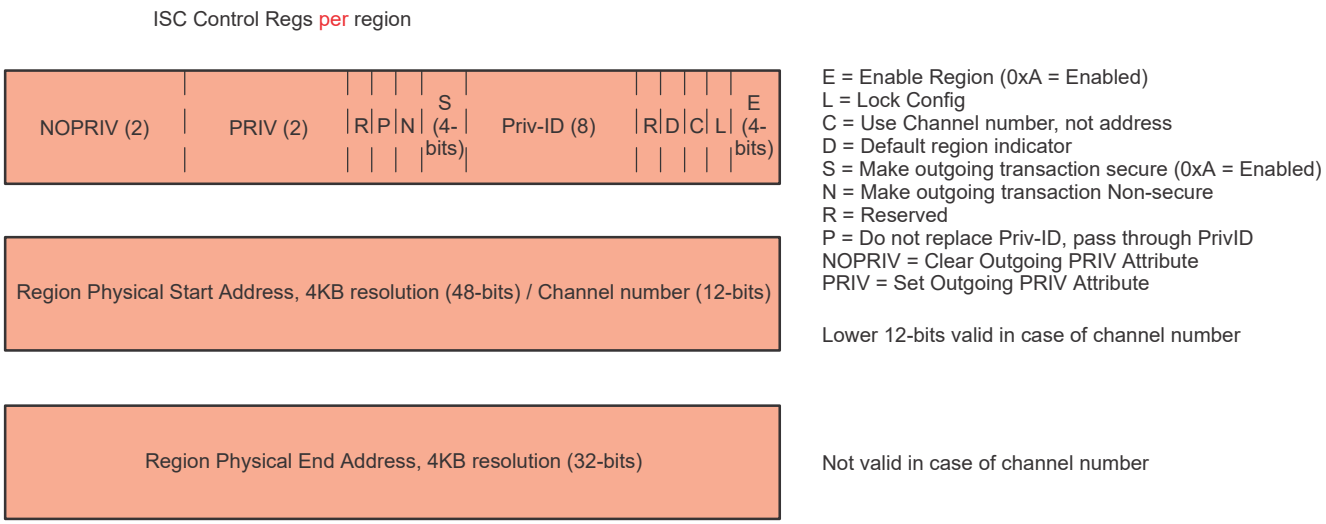


Figure 3-2. ISC Config Registers per Region

Priv-ID is used to identify the logical source of transaction and is attached by ISC. The Priv-IDs are allocated based on subsystem as part of platform definition.

Note

See Appendix Spreadsheet for ISC and Priv-ID details.

3.2.3.1 Special System Level Priv-ID

This section describes Special Priv-IDs that must not be assigned to any ISC. These Priv-ID when encountered by various blocks like firewall have special meaning.

Table 3-1 presents Special Priv-IDs.

Table 3-1. Special Priv-IDs

Special Priv-IDs	Hex Value	Decimal Value
System Reserved Priv-ID	0xC1, 0xC2, 0xC4, 0xC6, 0xC7	193, 194, 196, 198, 199
Wild card Priv ID	0xC3	195
Block Priv ID	0xC5	197
DMA Reserved Priv-ID	0xC0	192

3.2.4 Firewalls (FW)

Firewalls play an important role in implementing overall SoC security by providing a means to allow or restrict access to device resources to any given master entity or Secure/Non-Secure/Priv/User world. Firewalls are placed at various data path points throughout the SoC to control access to protected asset (Peripheral, memory and so forth).

Firewalls ensure that assets can be protected and are only accessible by allowed master and in selected operation mode (Secure, Non-Secure, Priv, User, write, read and so forth). In case of firewall violations, the transaction is dropped and the appropriate violation code is registered with associated parameters.

There are three types of firewalls, Peripheral firewalls, Memory firewalls (referred as one group **Region Based Firewall** in the device-specific TRM) and Channelized firewalls.

Each Firewall module in the SoC has a unique Firewall ID that allows software to decode the source of each violation. The Firewall IDs are allocated based on subsystem as part of platform definition. The full SoC view of Firewall IDs for various Firewall modules are captured in each device spec.

Copyright © 2024 Texas Instruments Incorporated

3.2.4.1 Peripheral Firewalls (FW)

Peripheral firewall modules are placed in front of peripherals like UART, SPI and so forth. They protect access to the peripheral registers/memory and typically have firewall regions as number of regions supported by module that is being protected.

All Peripheral firewalls support 3 Priv-ID slots that allow multiple masters/initiators to access protected peripherals.

The peripheral firewall is configured using dedicated VBUSP port to CBASS that connects to DMSC private VBUSP interconnect.

Figure 3-3 presents Peripheral Firewall.

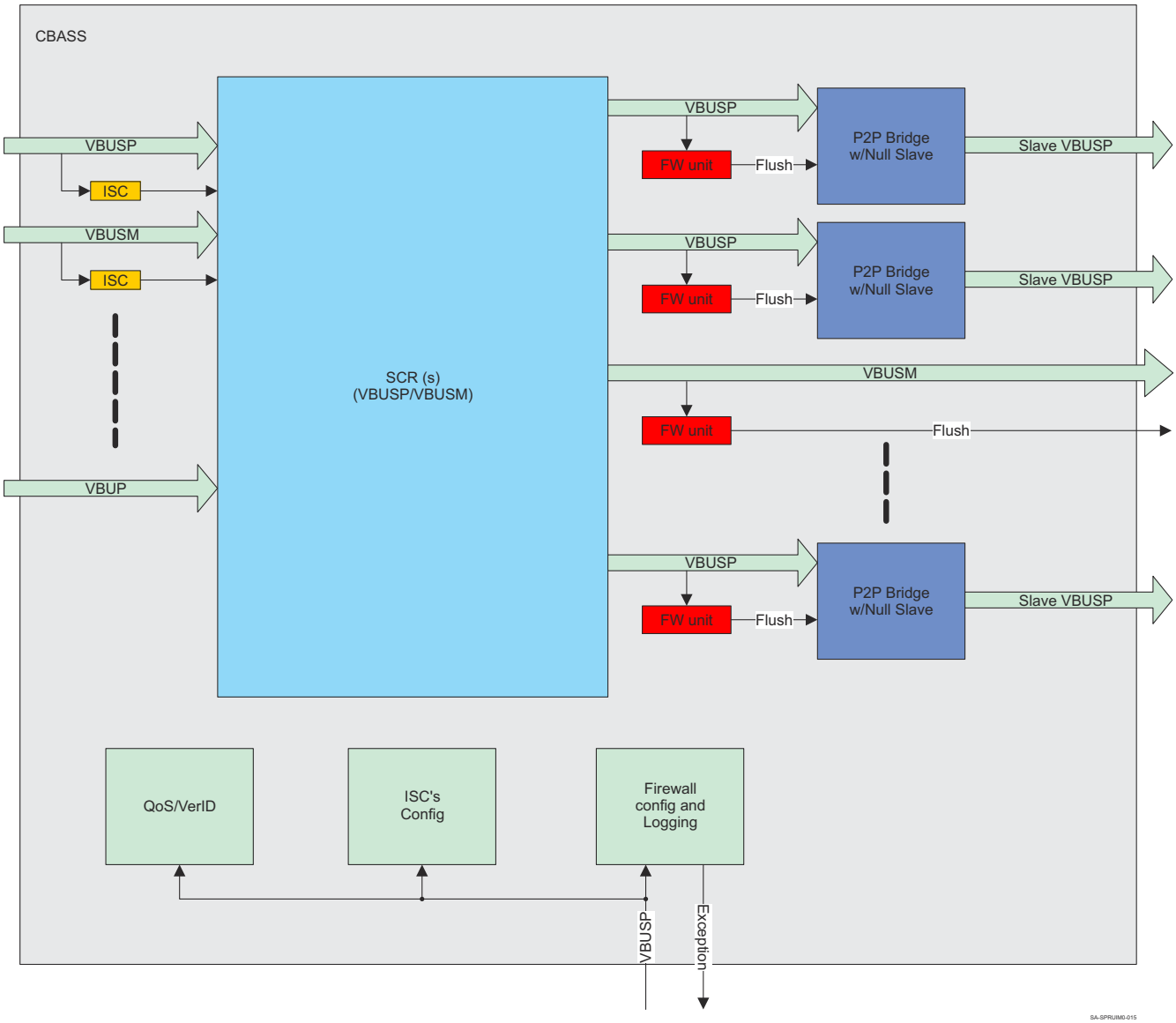


Figure 3-3. Peripheral Firewall

Each region is defined by start and end physical address and associated permission/control register as shown in Figure 3-4.

Copyright © 2024 Texas Instruments Incorporated

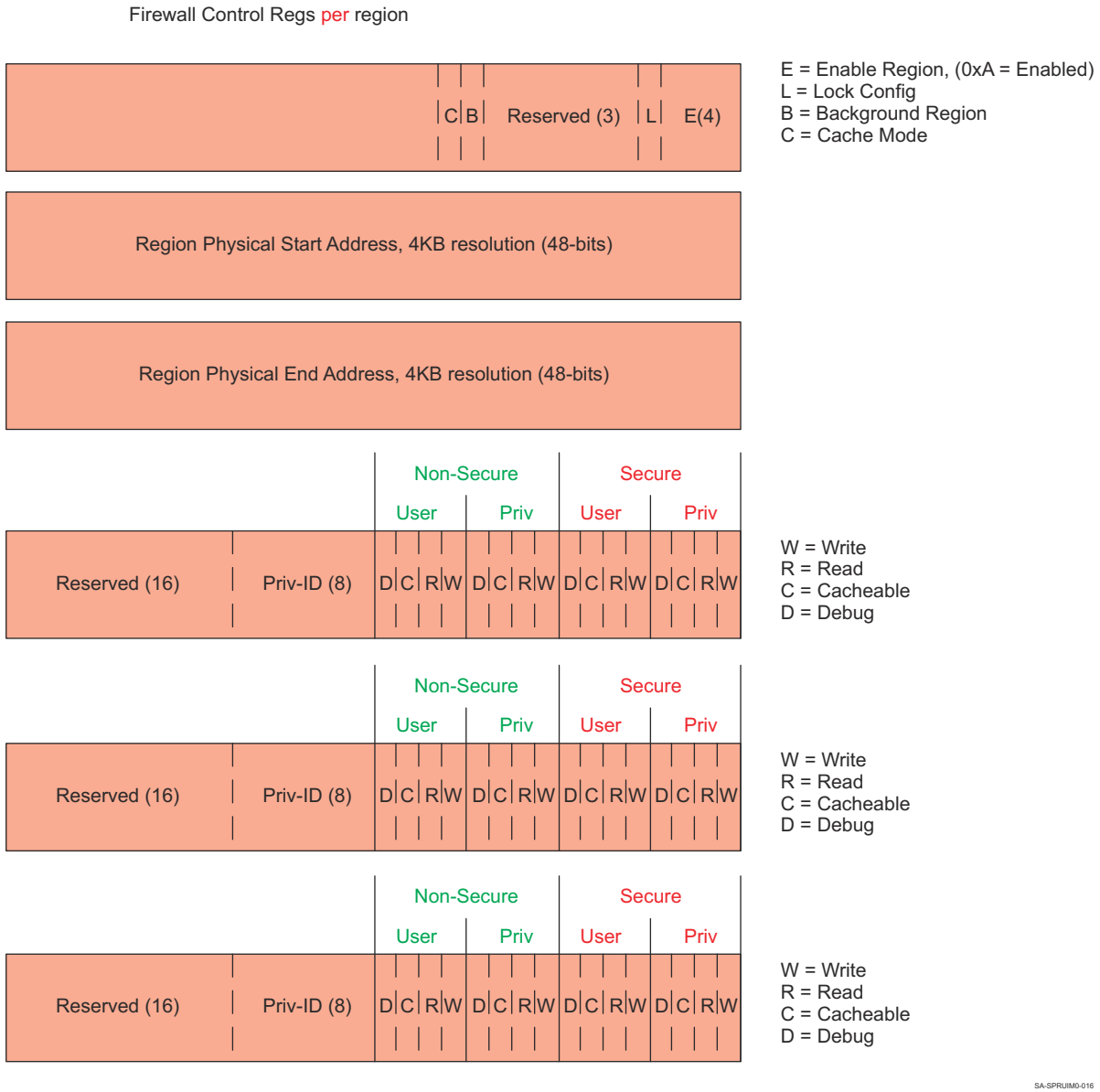


Figure 3-4. Peripheral Firewall Config Registers per Regions

A Peripheral's Firewall module can have one or more regions. In case there is more than 1 region, the registers are duplicated for each region.

If multiple regions are supported, a firewall configuration can be defined as either a Background or Foreground region depending on the setting of the Background (B) bit. Foreground regions can overlap background regions and their Firewall settings take precedence over the Background settings when there is overlap. However, background regions cannot overlap each other and foreground regions cannot overlap each other. A firewall error will result in case a memory transaction is made to the improper overlap regions.

Figure 3-5 presents peripherals firewall regions.

Copyright © 2024 Texas Instruments Incorporated

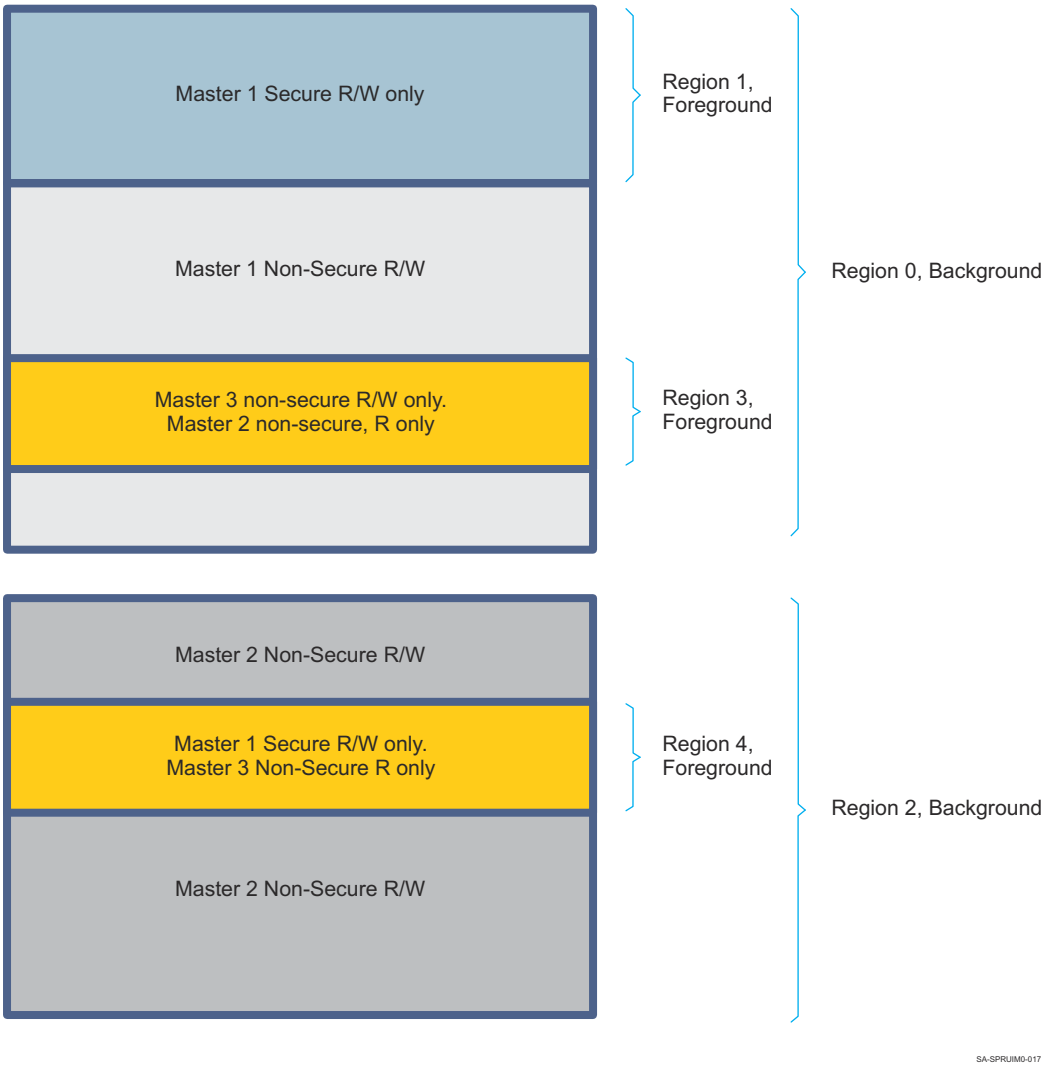


Figure 3-5. Peripherals Firewall Regions

In the above case, Region 0 and Region 2 are background regions (background bit set in control register). Region 1, Region 3 and Region 4 are **not** background (foreground) regions. In case the incoming transaction hits in address in Region 1, the permissions of Region 1 are applied to filter incoming transaction, thereby completely ignoring the permission of the background Region 0.

The following tables show the initiator and target firewalls, firewall IDs, physical base address of the corresponding firewall, number of firewall regions, and areas covered by the firewall. The firewall ID uniquely identifies each firewall. In case of firewall violation this ID is logged and can be read through the CBASS_EXCEPTION_LOGGING_HEADER0[23-8] SRC_ID field.

Note

See Appendix Spreadsheet for Target Firewalls details.

3.2.4.2 Memory or Region-based Firewalls

Memory/Data firewalls are designed to protect Memory (SRAM/DDR and so forth) and data regions (GPMC and so forth). Memory/Data firewalls have a defined firewall region count so that the memory can be partiontioned into multiple firewall regions.

The architecture of Memory/Data firewalls is similar to Peripheral Firewalls, however, the Memory/Data firewall can have either 1 or 3 Priv-ID slots per region, with associated permissions. The number of Priv-ID slots is

Copyright © 2024 Texas Instruments Incorporated

determined based on placement of firewall block. Each region in this type of firewall is defined by a physical start and end address.

Figure 3-6 presents Memory/Data firewall config registers with 3 Priv-ID slots per region.

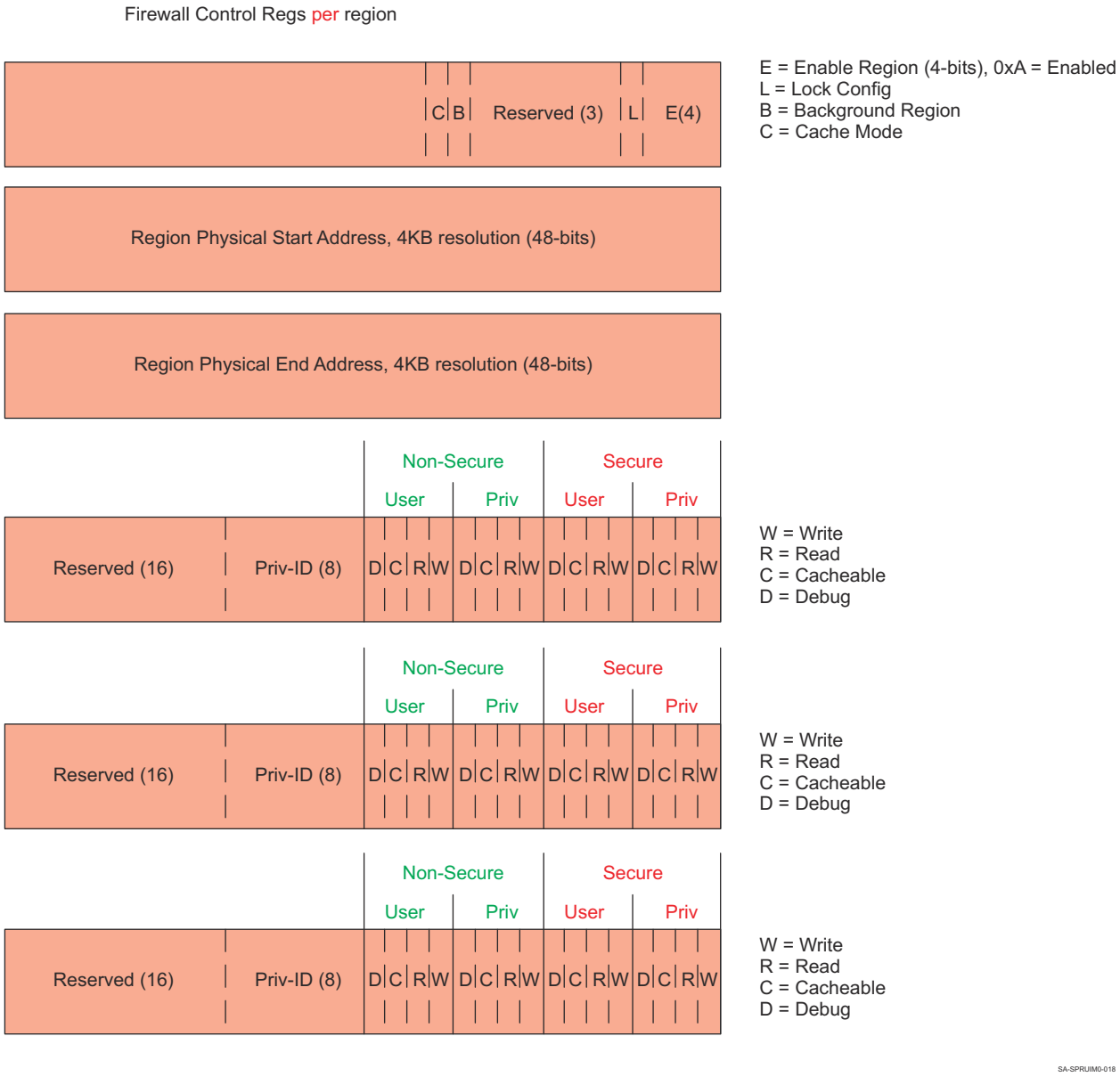


Figure 3-6. Memory/Data Firewall Config Registers with 3 Priv-ID slot per Region

Copyright © 2024 Texas Instruments Incorporated

Figure 3-7 presents Memory/Data firewall config registers with 1 Priv-ID slot per region.

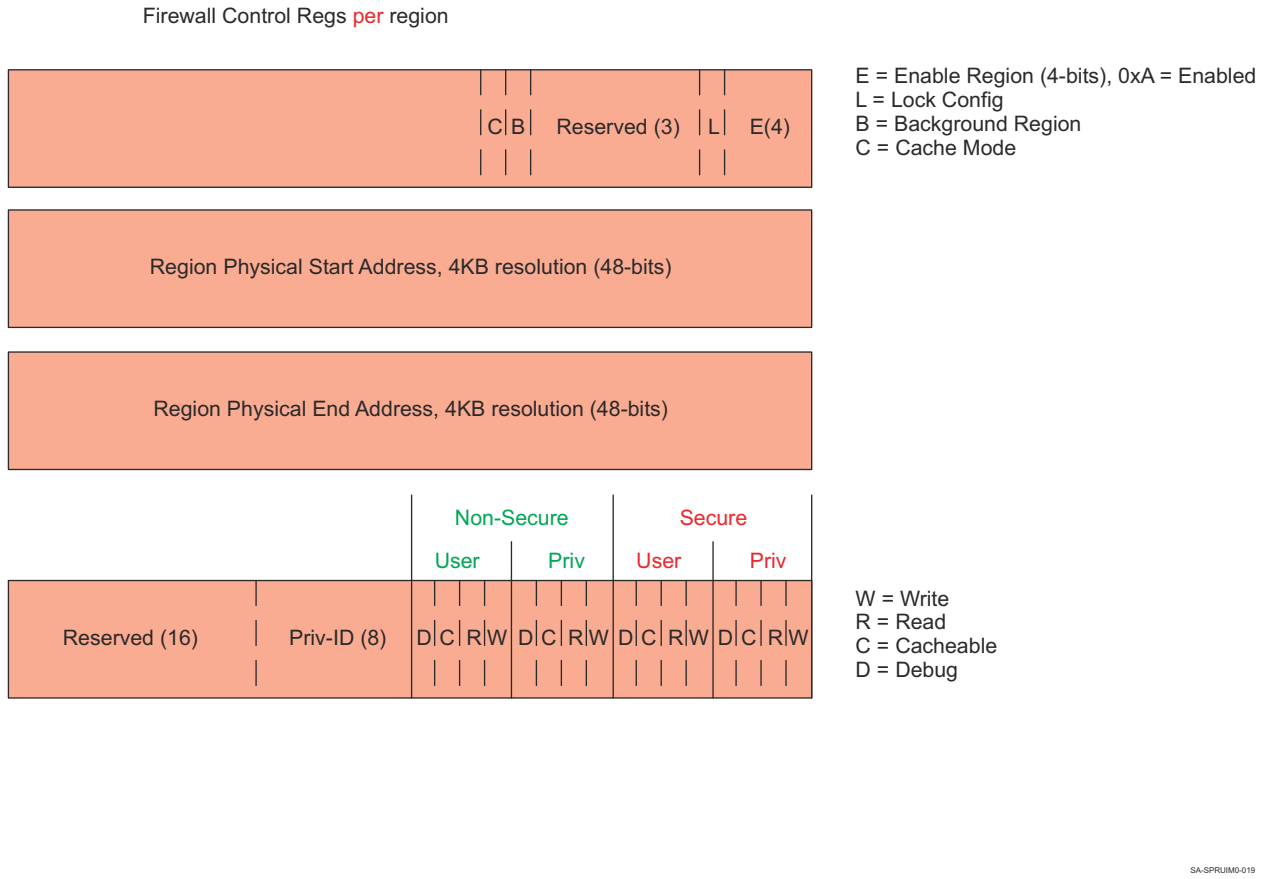


Figure 3-7. Memory/Data Firewall Config Registers with 1 Priv-ID slot per Region

The Memory/Data firewall is configured using dedicated VBUSP port to CBASS/AXI to VBUSM.C Bridge/DRU that connects to DMSC private VBUSP interconnect.

Memory/Data Firewalls also support background regions as in peripheral firewalls.

3.2.4.2.1 Region Based Firewall Functional Description

The region based firewall provides a memory region based protection and isolation mechanism. For each defined memory region, it checks the transaction attributes such as secure vs non-secure, debug vs non-debug, supervisor vs user mode, read vs write and so on. The transaction is dropped, if it does not match the configuration and appropriate violation code is registered with offending parameters.

Host modules (for example, AXI2VBUSMC bridge or CBASSes) provide transaction attributes that are checked by the firewalls, which then determine if the transaction should be blocked or passed. The firewall implements filtering algorithm that compares these host incoming transaction parameters against the region policy to give block indication to the host. Firewall region registers configure the filtering mechanism which includes setting region address range, region permission registers and region control register. There are also firewall exception registers used for violation reporting and logging.

The region based firewall supports multiple regions. Each one is defined by address range and associated access permission. The minimum memory region size is 4KB. The firewall concurrently checks the incoming transaction against all enabled regions looking for violations.

Each firewall is associated with the following registers:

- CBASS_FW_REGION_i_CONTROL
- CBASS_FW_REGION_i_PERMISSION_0 to CBASS_FW_REGION_i_PERMISSION_2
- CBASS_FW_REGION_i_START_ADDRESS_L and CBASS_FW_REGION_i_START_ADDRESS_H

Copyright © 2024 Texas Instruments Incorporated

- CBASS_FW_REGION_i_END_ADDRESS_L and CBASS_FW_REGION_i_END_ADDRESS_H

Each region is defined by start and end physical address and associated permission and control registers. A firewall can have 1-24 regions. In case there is more than 1 region, then these registers are duplicated for all regions. Setting the CBASS_FW_REGION_i_CONTROL[3-0] ENABLE field to 0xA enables the region and makes firewall check active for this region. Setting to 0x1 the CBASS_FW_REGION_i_CONTROL[8] BACKGROUND bit indicates to the firewall that the region is background region. Setting to 0x1 the CBASS_FW_REGION_i_CONTROL[9] CACHE_MODE bit ignores cacheable check, so that it cannot fail. In this case the access check is performed based on the READ and WRITE bits in CBASS_FW_REGION_i_PERMISSION_x. Clearing CACHE_MODE enables the cacheable check, so that cacheable transactions are allowed only when the corresponding cacheable permission bit (CBASS_FW_REGION_i_PERMISSION_x [y_CACHEABLE]) is set. If the CBASS_FW_REGION_i_CONTROL[4] LOCK bit is set to 0x1, then the region configuration cannot be changed at all. This is one-time change. This bit is typically used for primary master to consume and lock its resources and then pass firewall control to secondary master.

In case two regions overlap, the CBASS_FW_REGION_i_CONTROL[8] BACKGROUND bit is used to select the appropriate permission to be used. The region whose BACKGROUND bit is 0x0 (foreground) takes precedence and its permissions are taken into effect. The background region is ignored.

Note

There can be only one background region per firewall. Foreground regions can have overlapping addresses only with the background region.

It is a software mistake to have overlapping regions with the BACKGROUND bit set to the same value (either background or foreground). Software must be careful to not configure two overlapping regions with the BACKGROUND bit set to the same value.

The firewall also checks if the transaction crosses a 4KB boundary. If so, the transaction is blocked regardless of any matching regions.

When a region is set to be debugable through the corresponding CBASS_FW_REGION_i_PERMISSION_x [y_DEBUG] bit, the firewall ignores the read and write checks for any debug transactions, so that it cannot fail. This allows debug breakpoints to be written out to code even in read-only regions.

When a region is set to be cacheable through the corresponding CBASS_FW_REGION_i_PERMISSION_x [y_CACHEABLE] bit and CACHE_MODE = 0x0, the firewall ignores the read and write checks for any transaction (cacheable or not), so that it cannot fail due to these checks. This allows a write allocated cache to read a cache line even in write-only regions. Due to caches not protecting user and supervisor data from each other, the firewall allows cacheable access to the region when either the user cacheable permission or the supervisor cacheable permission is set.

The firewall notifies the host that the transaction is blocked in case of the following violation conditions:

- All regions are disabled.
- Incoming address does not hit any region.
- Non-secure incoming read transaction attempting to access configured secure-only write/read region.
- Non-secure incoming write transaction attempting to access configured secure-only write/read region.
- Secure incoming read transaction attempting to access configured non-secure-only write/read region.
- Secure incoming write transaction attempting to access configured non-secure region.
- Incoming secure write transaction attempting to access configured secure read region. This check is ignored if the region is configured as cacheable.
- Incoming secure read transaction attempting to access configured secure write region. This check is ignored if the region is configured as cacheable.
- Incoming non-secure write transaction attempting to access configured non-secure read region. This check is ignored if the region is configured as cacheable.
- Incoming non-secure read transaction attempting to access configured non-secure write region. This check is ignored if the region is configured as cacheable.

Copyright © 2024 Texas Instruments Incorporated

- Incoming non-secure debug access to non-debugable secure region.
- Incoming non-secure debug access to non-debugable non-secure region.
- Incoming secure debug access to non-debugable secure region.
- Incoming secure debug access to non-debugable non-secure region.
- Incoming cacheable transaction attempting to access non-cacheable configured region.

All violation parameters caused the exception are logged in the firewall exception registers described in *Firewall Exception Registers*. Table 3-2 shows the mapping between the register fields and violation parameters. If the CBASS_EXCEPTION_LOGGING_CONTROL[0] DISABLE_F bit is set to 0x1, logging is disabled.

If violation occurs, the corresponding firewall notifies WKUP_DMSC0 driving high a dedicated signal. The notification (that is, the signal) can be masked by setting to 0x1 the CBASS_EXCEPTION_LOGGING_CONTROL[1] DISABLE_PEND bit. The firewall exception notification signal gets automatically cleared when the CBASS_EXCEPTION_LOGGING_DATA3 register is read. That signal can be manually set via the CBASS_EXCEPTION_PEND_SET[0] PEND_SET bit and cleared via the CBASS_EXCEPTION_PEND_CLEAR[0] PEND_CLR bit. Reading one of these two bits returns the status of the signal (that is, violation occurred or not).

Table 3-2. Firewall Violation Parameters

Field	Value	Description
CBASS_EXCEPTION_LOGGING_HEADER0[31-24] TYPE_F	0x1	Exception type for firewall violation. This is fixed for the SoC firewalls and is used by software to detect that this corresponding violation comes from a firewall.
CBASS_EXCEPTION_LOGGING_HEADER0[23-8] SRC_ID	0x-	Firewall ID. Unique for each firewall. This is used to identify the exact firewall that issued violation so that software detects the precise source.
CBASS_EXCEPTION_LOGGING_HEADER0[7-0] DEST_ID	0x-	Destination ID of node where the firewall violation has be to routed.
CBASS_EXCEPTION_LOGGING_HEADER1[31-24] GROUP	0x0	Exception group. This is used to group exceptions to category. All firewall exceptions are in one group.
CBASS_EXCEPTION_LOGGING_HEADER1[23-16] CODE	Exception code:	
	0x0	Reserved.
	0x1	No region enabled.
	0x2	Incoming address does not hit any active region and transaction is dropped.
	0x3	Reserved.
	0x4	Cacheable error. A cacheable transaction attempting to read/write non-cached marked region.
	0x5	Debug error. A debug transaction attempting to read/ write a non-allowed debug region.
	0x6	Read error. A Read transaction attempting to read from non-allowed read region.
	0x7	Write error. A Write transaction attempting to write from non-allowed write region.
	0x8	4KB crossing error. A transaction attempting to cross a 4KB boundary.
CBASS_EXCEPTION_LOGGING_DATA0[31-0] ADDR_L	0x-	Lower 32 address bits (31:0) of the incoming transaction
CBASS_EXCEPTION_LOGGING_DATA1[15-0] ADDR_H	0x-	Upper 16 address bits (47:32) of the incoming transaction

Copyright © 2024 Texas Instruments Incorporated

Table 3-2. Firewall Violation Parameters (continued)

Field	Value	Description
CBASS_EXCEPTION_LOGGING_DATA2[7-0] PRIV_ID	Not used	Incoming transaction parameters
CBASS_EXCEPTION_LOGGING_DATA2[8] SECURE	0x-	
CBASS_EXCEPTION_LOGGING_DATA2[9] PRIV	0x-	
CBASS_EXCEPTION_LOGGING_DATA2[10] CACHEABLE	0x-	
CBASS_EXCEPTION_LOGGING_DATA2[11] DEBUG	0x-	
CBASS_EXCEPTION_LOGGING_DATA2[12] READ	0x-	
CBASS_EXCEPTION_LOGGING_DATA2[13] WRITE	0x-	
CBASS_EXCEPTION_LOGGING_DATA2[27-16] ROUTEID	0x-	
CBASS_EXCEPTION_LOGGING_DATA3[9-0] BYTECNT	0x-	Byte count of the incoming transaction

3.2.4.2.2 Channelized Firewalls

A Channelized firewall is designed to protect modules that have logical channels (Ring Accelerator and so forth, for example). These firewalls operate at the resolution of a channel and have no association with physical addresses. Each channel is defined as a logical control entity.

In case of channelized firewalls, the number of regions are less than or equal to number of logical channels and each typical channel/region is very small memory space (typically in Bytes, like 16 bytes). A channel is owned by one processing entity, hence only 1 Priv-ID slot is provided for each Channelized firewall along with associated permissions.

Copyright © 2024 Texas Instruments Incorporated

Figure 3-8 presents Channelized firewall config registers with 1 Priv-ID slot per region.

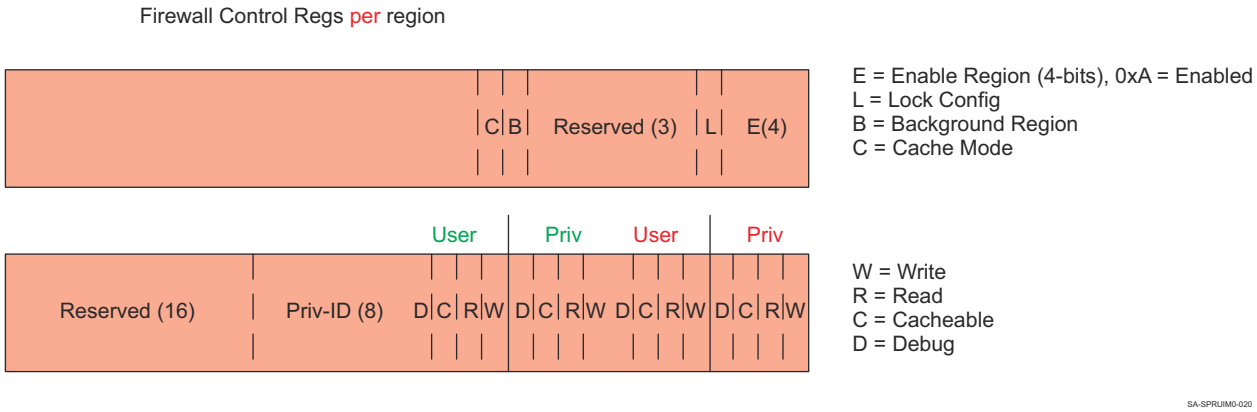


Figure 3-8. Channelized Firewall Config Registers with 1 Priv-ID slot per Region

The Channelized firewall is configured using dedicated VBUSP port to CBASS/Bridges that connects to DMSC private VBUSP interconnect.

Note

For more information about the interconnect firewalls, see *Interconnect Firewalls*.

3.2.4.2.2.1 Channelized Firewall Functional Description

The channelized firewall protects an address space that consists of multiple channels where each channel needs its own permissions. This is in contrast to the region based firewall which implements a number of regions that can protect a programmed address range. The channelized firewall extends this to a larger number of ranges that can be protected, but each range is no longer programmable and is instead fixed to a particular channel address range. This allows each channel to be owned and protected individually. The channels within same memory region have same size. The system can allocate each channel to a particular owner or owner groups with certain permissions and guarantee that others cannot access that channel. This protection is useful for either accessing data resources, or control for each resource. The channelized firewall provides an efficient way to implement access protection and isolation requiring finer granularity than the region based firewall.

Same as the region based firewall, the channelized firewall may also support multiple regions. Each region contains a number of channels that need to be protected individually and has size in bytes equal to the channel's data to protect. This allows multiple regions so that the firewall can protect a module containing multiple sets of registers that are each channelized and need separate protections, such as data access as well as control setup.

The transaction addresses are compared against region addresses to identify which region is being accessed. Then the offset within the region is used to identify the particular channel being accessed. If a valid channelized region is not decoded during the firewall check, then the transaction is passed through unmodified because the module may have other regions which are not channelized, but accesses to them are already protected by a region based firewall so the channelized one should not block them.

The permission check is performed just like for a region based firewall. The channelized firewall checks for user or supervisor priviledge levels. It also checks for transactions that cross important boundaries and gives errors if violated. The first check is for a transaction crossing a 4KB address boundary. This is illegal on the bus, so the firewall checks for compliance. The second check is if the transaction crosses a channel boundary. The channelized firewall blocks a transaction accessing multiple channels in the same burst, as it cannot check the permission for the entire range of channels. The channelized firewall also supports error logging when a transaction fails the checks. It sends information about the type of error and the transaction that caused it.

The channelized firewall is associated with the following registers:

Copyright © 2024 Texas Instruments Incorporated

- *_CH_i_CONTROL, where "i" can be from 0 to 63 and denotes the channel. This register is equivalent to the CBASS_FW_REGION_i_CONTROL register of the region based firewall.
- *_ CH_i_PERMISSION_x, where "i" can be from 0 to 63 and denotes the channel and "x" can be 0 to 2. This register is equivalent to the CBASS_FW_REGION_i_PERMISSION_x register of the region based firewall.

3.2.5 Null Error Reporting

The interconnect supports capturing null slave errors for reporting. A null slave error occurs when a transaction does not address any slaves based on the address map, resulting in the transaction returning error status. The null errors are captured into registers and an interrupt is generated. There is only one set of reporting registers, so only one error can be captured until cleared by software. The report captures some details of the transaction, such as the Route ID to identify the master involved and the address accessed. See *Null Error Reporting Registers* for details about the null error reporting registers.

Copyright © 2024 Texas Instruments Incorporated

3.2.6 Initiator-Target Connections

Table 3-4 and Table 3-5 list the initiator and target end point connections on the CBASS0. A cell may contain one of the following:

- Y – There is a connection between this initiator and that target
- N – There is no connection between this initiator and that target.

Table 3-3. Compute Cluster Connectivity Matrix

Target	Initiator				
	A72 Corepac	C7x-A	C7x-B	DRU	System Initiator
C7x-A L2 SRAM	N	Y	N	Y	Y
C7x-B L2 SRAM	N	N	Y	Y	Y
MSMC L3 4MB	Y	Y	Y	Y	Y
MSMC L3 MMR	Y	Y	Y	Y	Y
DRU MMR	N	Y	Y	N	Y
CLEC MMR	N	Y	Y	N	Y
EMIF/DDR	Y	Y	Y	Y	Y
System Endpoint	Y	Y	Y	Y	N

Table 3-4. Connectivity Matrix (Part 1)

Targets	Initiators																						
	COMPUTE_CLUSTER0 (GIC)	DEBUGSS0_DBGMST	DEBUGSS0	DMPAC0_DATA	DSS0_DMA	DSS0_FBDC	EMMC8SS0	EMMCSD4SS0	ENCODER0	GPU0	LED	MCU_PDMA_ADC0_MEMR0	MCU_PDMA_MISC_G00	MCU_PDMA_MISC_G10	MCU_PDMA_MISC_G20	MCU_R5FSS0_CORE0	MCU_R5FSS0_CORE1	NAVSS MAIN	NAVSS MCU	PCIE0	PCIE1	PCIE2	PCIE3
ATL0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
BWLIMITER_ AC_SPARE1	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
BWLIMITER_ AC_SPARE2	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
BWLIMITER_ AC_SPARE3	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
BWLIMITER_ AC_SPARE4	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
BWLIMITER_ AC_SPARE5	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
BWLIMITER_ GPU0_M0_WR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
BWLIMITER_ GPU0_M1_RD	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
BWLIMITER_ GPU0_M1_WR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-4. Connectivity Matrix (Part 1) (continued)

Targets	Initiators																						
	COMPUTE_CLUSTER0 (GIC)	DEBUGSS0_DBGMST	DEBUGSS0	DMPAC0_DATA	DSS0_DMA	DSS0_FBDC	EMMC8SS0	EMMCSD4SS0	ENCODER0	GPU0	LED	MCU_PDMA_ADC0_MEMR0	MCU_PDMA_MISC_G00	MCU_PDMA_MISC_G10	MCU_PDMA_MISC_G20	MCU_R5FSS0_CORE0	MCU_R5FSS0_CORE1	NAVSS MAIN	NAVSS MCU	PCIE0	PCIE1	PCIE2	PCIE3
BWLIMITER_ HC_SPARE1	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
BWLIMITER_ HC_SPARE2	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
BWLIMITER_ HC_SPARE3	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
BWLIMITER_ WAVE521CL0/1_WR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
BWLIMITER_AC_SPARE0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
BWLIMITER_ENCODER0_RD	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
BWLIMITER_ENCODER0_WR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
BWLIMITER_GPU0_M0_RD	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
BWLIMITER_HC_SPARE0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
BWLIMITER_WAVE521CL0/1_RD	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_ SPARE1_GLB	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_ SPARE1_QOS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_ SPARE3_GLB	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_AC_ NONSAFE0_ERR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_AC_CFG_NONSAFE0_ERR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_AC_CFG_NONSAFE0_FW	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_AC_CFG_NONSAFE0_GLB	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_AC_CFG0_ERR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_AC_CFG0_FW	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_AC_CFG0_GLB	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_AC_MERGER0_ERR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_AC_NONSAFE0_FW	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_AC_NONSAFE0_GLB	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_AC_NONSAFE0_ISC	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_AC_NONSAFE0_QOS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_CSI0_ERR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-4. Connectivity Matrix (Part 1) (continued)

Targets	Initiators																						
	COMPUTE_CLUSTER0 (GIC)	DEBUGSS0_DBGMST	DEBUGSS0	DMPAC0_DATA	DSS0_DMA	DSS0_FBDC	EMMC8SS0	EMMCSD4SS0	ENCODER0	GPU0	LED	MCU_PDMA_ADC0_MEMR0	MCU_PDMA_MISC_G00	MCU_PDMA_MISC_G10	MCU_PDMA_MISC_G20	MCU_R5FSS0_CORE0	MCU_R5FSS0_CORE1	NAVSS MAIN	NAVSS MCU	PCIE0	PCIE1	PCIE2	PCIE3
CBASS_CSI0_FW	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_CSI0_GLB	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_DATADEBUG0_ERR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_DATADEBUG0_FW	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_DATADEBUG0_GLB	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_DATADEBUG0_ISC	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_DATADEBUG0_QOS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_DEBUG0_ERR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_FW0_ERR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_HC_CFG_B0_ERR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_HC_CFG_B0_FW	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_HC_CFG_B0_GLB	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_HC20_ERR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_HC20_FW	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_HC20_GLB	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_HC20_ISC	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_HC20_QOS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_INFRA_NON_SAFE0_ERR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_INFRA0_ERR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_INFRA0_FW	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_INFRA0_GLB	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_IPPHY_SAFE0_ERR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_IPPHY_SAFE0_FW	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_IPPHY_SAFE0_GLB	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_IPPHY0_ERR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_IPPHY0_FW	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-4. Connectivity Matrix (Part 1) (continued)

Targets	Initiators																						
	COMPUTE_CLUSTER0 (GIC)	DEBUGSS0_DBGMST	DEBUGSS0	DMPAC0_DATA	DSS0_DMA	DSS0_FBDC	EMMC8SS0	EMMCSD4SS0	ENCODER0	GPU0	LED	MCU_PDMA_ADC0_MEMR0	MCU_PDMA_MISC_G00	MCU_PDMA_MISC_G10	MCU_PDMA_MISC_G20	MCU_R5FSS0_CORE0	MCU_R5FSS0_CORE1	NAVSS MAIN	NAVSS MCU	PCIE0	PCIE1	PCIE2	PCIE3
CBASS_IPPHY0_GLB	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_IPPHY0_ISC	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_IPPHY0_QOS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_MAIN_NON_INFRA_FW	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_MAIN_NON_INFRA_GLB	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_R5FSS0_MEM_ERR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_R5FSS0_MEM_FW	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_R5FSS0_MEM_GLB	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_R5FSS0_MEM_ISC	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_R5FSS0_MEM_QOS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_R5FSS0_SLV_ERR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_R5FSS0_SLV_FW	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_R5FSS0_SLV_GLB	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_R5FSS1_MEM_ERR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_R5FSS1_MEM_FW	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_R5FSS1_MEM_GLB	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_R5FSS1_MEM_ISC	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_R5FSS1_MEM_QOS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_R5FSS1_PERIPH_ERR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_R5FSS1_PERIPH_GLB	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_R5FSS1_PERIPH_ISC	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_R5FSS1_PERIPH_QOS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_RC_CFG_B0_ERR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_RC_CFG_B0_FW	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_RC_CFG_B0_GLB	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_RC0_ERR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-4. Connectivity Matrix (Part 1) (continued)

Targets	Initiators																						
	COMPUTE_CLUSTER0 (GIC)	DEBUGSS0_DBGMST	DEBUGSS0	DMPAC0_DATA	DSS0_DMA	DSS0_FBDC	EMMC8SS0	EMMCSD4SS0	ENCODER0	GPU0	LED	MCU_PDMA_ADC0_MEMR0	MCU_PDMA_MISC_G00	MCU_PDMA_MISC_G10	MCU_PDMA_MISC_G20	MCU_R5FSS0_CORE0	MCU_R5FSS0_CORE1	NAVSS MAIN	NAVSS MCU	PCIE0	PCIE1	PCIE2	PCIE3
CBASS_RC0_FW	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_RC0_GLB	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_RC0_ISC	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_RC0_QOS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_SPARE_FW	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_SPARE0_ERR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_SPARE0_GLB	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_SPARE0_ISC	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_SPARE0_QOS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_SPARE1_ERR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_SPARE1_ISC	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_SPARE2_ERR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_SPARE2_GLB	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CBASS_SPARE3_ERR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CCDEBUGSS0_ROM	N	Y	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CCDEBUGSS0_SYS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CCDEBUGSS1_ROM	N	Y	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CCDEBUGSS1_SYS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CCDEBUGSS2_ROM	N	Y	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CCDEBUGSS2_SYS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CMPEVENT_INTRTR0_INTR_ROUTER_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
COMPUTE_CLUSTER0_CCROM	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
COMPUTE_CLUSTER0_DDRCTL0_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
COMPUTE_CLUSTER0_DDRCTL1_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
COMPUTE_CLUSTER0_DDRCTL2_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
COMPUTE_CLUSTER0_DDRSS0_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-4. Connectivity Matrix (Part 1) (continued)

Targets	Initiators																						
	COMPUTE_CLUSTER0 (GIC)	DEBUGSS0_DBGMST	DEBUGSS0	DMPAC0_DATA	DSS0_DMA	DSS0_FBDC	EMMC8SS0	EMMCSD4SS0	ENCODER0	GPU0	LED	MCU_PDMA_ADC0_MEMR0	MCU_PDMA_MISC_G00	MCU_PDMA_MISC_G10	MCU_PDMA_MISC_G20	MCU_R5FSS0_CORE0	MCU_R5FSS0_CORE1	NAVSS MAIN	NAVSS MCU	PCIE0	PCIE1	PCIE2	PCIE3
COMPUTE_CLUSTER0_DDRSS1_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
COMPUTE_CLUSTER0_DDRSS2_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
COMPUTE_CLUSTER0_DDRSS3_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
COMPUTE_CLUSTER0_SMS_BOOT	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
COMPUTE_CLUSTER0_SMS_EMULATION	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
COMPUTE_CLUSTER0_SMS_FW	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
COMPUTE_CLUSTER0_SMS_GLB	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
COMPUTE_CLUSTER0_SMS_PRIVID	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
COMPUTE_CLUSTER0_GIC_DISTRIBUTOR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
COMPUTE_CLUSTER0_GIC_TRANSLATER	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
COMPUTE_CLUSTER0_MSMC0_CC	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
COMPUTE_CLUSTER0_MSMC0_ECC_AGGR0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
COMPUTE_CLUSTER0_MSMC0_PBIST	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
COMPUTE_CLUSTER0_MSMC0_SRAM	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
COMPUTE_CLUSTER1_CCROM	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CPSW0_ECC	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CPSW0_NUSS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CPSW1_ECC	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CPSW1_NUSS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CPT2_AGGR0_MMR	N	Y	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CPT2_AGGR1_MMR	N	Y	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CPT2_AGGR2_MMR	N	Y	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CPT2_AGGR3_MMR	N	Y	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CPT2_AGGR4_MMR	N	Y	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CPT2_AGGR5_MMR	N	Y	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CPT2_AGGR6_MMR	N	Y	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-4. Connectivity Matrix (Part 1) (continued)

Targets	Initiators																						
	COMPUTE_CLUSTER0 (GIC)	DEBUGSS0_DBGMST	DEBUGSS0	DMPAC0_DATA	DSS0_DMA	DSS0_FBDC	EMMC8SS0	EMMCSD4SS0	ENCODER0	GPU0	LED	MCU_PDMA_ADC0_MEMR0	MCU_PDMA_MISC_G00	MCU_PDMA_MISC_G10	MCU_PDMA_MISC_G20	MCU_R5FSS0_CORE0	MCU_R5FSS0_CORE1	NAVSS MAIN	NAVSS MCU	PCIE0	PCIE1	PCIE2	PCIE3
CPT2_AGGR7_MMR	N	Y	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CSI_PSILSS0_MMRS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CSI_RX_IF0_CP_INTD_CFG_INTD_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CSI_RX_IF0_ECC_AGGR_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CSI_RX_IF0_RX_SHIM_VBUSP_MMR_CSI2RXIF	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CSI_RX_IF0_VBUS2APB_WRAP_VBUSP_APB_CSI2RX	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CSI_RX_IF1_CP_INTD_CFG_INTD_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CSI_RX_IF1_ECC_AGGR_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CSI_RX_IF1_RX_SHIM_VBUSP_MMR_CSI2RXIF	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CSI_RX_IF1_VBUS2APB_WRAP_VBUSP_APB_CSI2RX	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CSI_RX_IF2_CP_INTD_CFG_INTD_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CSI_RX_IF2_ECC_AGGR_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CSI_RX_IF2_RX_SHIM_VBUSP_MMR_CSI2RXIF	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CSI_RX_IF2_VBUS2APB_WRAP_VBUSP_APB_CSI2RX	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CSI_TX_IF0_CP_INTD_CFG_INTD_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CSI_TX_IF0_ECC_AGGR_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CSI_TX_IF0_TX_SHIM_VBUSP_MMR_CSI2TXIF	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CSI_TX_IF0_VBUS2APB_WRAP_VBUSP_APB_CSI2TX	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CSI_TX_IF1_CP_INTD_CFG_INTD_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CSI_TX_IF1_ECC_AGGR_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CSI_TX_IF1_TX_SHIM_VBUSP_MMR_CSI2TXIF	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CSI_TX_IF1_VBUS2APB_WRAP_VBUSP_APB_CSI2TX	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
CTRL_MMR0_CFG0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
DCC0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
DCC1	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
DCC2	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-4. Connectivity Matrix (Part 1) (continued)

Targets	Initiators																						
	COMPUTE_CLUSTER0 (GIC)	DEBUGSS0_DBGMST	DEBUGSS0	DMPAC0_DATA	DSS0_DMA	DSS0_FBDC	EMMC8SS0	EMMCSD4SS0	ENCODER0	GPU0	LED	MCU_PDMA_ADC0_MEMR0	MCU_PDMA_MISC_G00	MCU_PDMA_MISC_G10	MCU_PDMA_MISC_G20	MCU_R5FSS0_CORE0	MCU_R5FSS0_CORE1	NAVSS MAIN	NAVSS MCU	PCIE0	PCIE1	PCIE2	PCIE3
DCC3	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
DCC4	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
DCC5	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
DCC6	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
DCC7	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
DCC8	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
DCC9	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
DEBUG_PSILSS_MMRS	N	Y	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
DEBUGSS_WRAP0	N	Y	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
DEBUGSS1_ROM	N	Y	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
DEBUGSS1_SYS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
DEBUGSUSPENDRTR0_INTR_ROUTER_CFG	N	Y	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
DFTSS0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
DMPAC_VPAC_PSILSS_MMRS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
DMPAC0_CFG	N	N	Y	N	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
DMPAC0_CFG_SMS_FW	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
DMPAC0_CFG_SMS_GLB	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
DMPAC0_DMPAC_TOP_DOF_INFRA_DMPAC_BASE_MEM_SLV_CBASS_STRIPE_MSRAM_SLV	N	N	Y	N	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
DMPAC0_KSDW_ECC_AGGR_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
DMPAC0_SL2_SMS_FW	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
DMPAC0_SL2_SMS_GLB	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
DMPAC0_SL2_SMS_ISC	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
DMSS_HSM_CFG1	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
DMSS_HSM_SECPROXY	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
DPHY_RX0_VBUS2APB_WRAP_VBUSP_K3_DPHY_RX	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-4. Connectivity Matrix (Part 1) (continued)

Targets	Initiators																						
	COMPUTE_CLUSTER0 (GIC)	DEBUGSS0_DBGMST	DEBUGSS0	DMPAC0_DATA	DSS0_DMA	DSS0_FBDC	EMMC8SS0	EMMCSD4SS0	ENCODER0	GPU0	LED	MCU_PDMA_ADC0_MEMR0	MCU_PDMA_MISC_G00	MCU_PDMA_MISC_G10	MCU_PDMA_MISC_G20	MCU_R5FSS0_CORE0	MCU_R5FSS0_CORE1	NAVSS MAIN	NAVSS MCU	PCIE0	PCIE1	PCIE2	PCIE3
DPHY_RX1_VBUS2APB_WRAP_VBUSP_K3_DPHY_RX	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
DPHY_RX2_VBUS2APB_WRAP_VBUSP_K3_DPHY_RX	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
DPHY_TX0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
DPHY_TX1	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
DSS_DSI0_DSI_TOP_ECC_AGGR_SYS_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
DSS_DSI0_DSI_WRAP_MMR_VBUSP_CFG_DSI_WRAP	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
DSS_DSI1_DSI_TOP_ECC_AGGR_SYS_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
DSS_DSI1_DSI_WRAP_MMR_VBUSP_CFG_DSI_WRAP	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
DSS_EDP0_INTG_CFG_VP	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
DSS_EDP0_MHDPTX_WRAPPER_ECC_AGGR_CORE_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
DSS_EDP0_MHDPTX_WRAPPER_ECC_AGGR_DSC_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
DSS_EDP0_MHDPTX_WRAPPER_ECC_AGGR_PHY_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
DSS_EDP0_V2A_CORE_VP_REGS_APB	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
DSS_EDP0_V2A_S_CORE_VP_REGS_SAPB	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
DSS0_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
DSSUL_DSI0_DSI_TOP_VBUSP_CFG_DSI_0_DSI	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
DSSUL_DSI1_DSI_TOP_VBUSP_CFG_DSI_0_DSI	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
DSSUL0_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
ECAP0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
ECAP1	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
ECAP2	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
ECC_AGGR_IPPHY_REGS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
ECC_AGGR0_ECC_AGGR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
ECC_AGGR10_REGS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
ECC_AGGR11_REGS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
ECC_AGGR16_REGS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-4. Connectivity Matrix (Part 1) (continued)

Targets	Initiators																						
	COMPUTE_CLUSTER0 (GIC)	DEBUGSS0_DBGMST	DEBUGSS0	DMPAC0_DATA	DSS0_DMA	DSS0_FBDC	EMMC8SS0	EMMCSD4SS0	ENCODER0	GPU0	LED	MCU_PDMA_ADC0_MEMR0	MCU_PDMA_MISC_G00	MCU_PDMA_MISC_G10	MCU_PDMA_MISC_G20	MCU_R5FSS0_CORE0	MCU_R5FSS0_CORE1	NAVSS MAIN	NAVSS MCU	PCIE0	PCIE1	PCIE2	PCIE3
ECC_AGGR17_REGS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
ECC_AGGR18_REGS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
ECC_AGGR19_REGS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
ECC_AGGR20_REGS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
ECC_AGGR21_REGS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
ECC_AGGR4_REGS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
ECC_AGGR5_REGS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
ECC_AGGR6_REGS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
ECC_AGGR6_REGS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
EFUSE0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
ELM0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
ENCODER0_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
EPWM0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
EPWM1	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
EPWM2	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
EPWM3	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
EPWM4	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
EPWM5	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
EQEP0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
EQEP1	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
EQEP2	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
ESM0_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
GPIO0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
GPIO2	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
GPIO4	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
GPIO6	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-4. Connectivity Matrix (Part 1) (continued)

Targets	Initiators																						
	COMPUTE_CLUSTER0 (GIC)	DEBUGSS0_DBGMST	DEBUGSS0	DMPAC0_DATA	DSS0_DMA	DSS0_FBDC	EMMC8SS0	EMMCSD4SS0	ENCODER0	GPU0	LED	MCU_PDMA_ADC0_MEMR0	MCU_PDMA_MISC_G00	MCU_PDMA_MISC_G10	MCU_PDMA_MISC_G20	MCU_R5FSS0_CORE0	MCU_R5FSS0_CORE1	NAVSS MAIN	NAVSS MCU	PCIE0	PCIE1	PCIE2	PCIE3
GPIOMUX_INTRTR0_INTR_ROUTER_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
GPMC0_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
GPMC0_DATA	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
GPU0_CORE_MMRS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
GPU0_PBIST_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
GTC0_GTC_CFG0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
GTC0_GTC_CFG1	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
GTC0_GTC_CFG2	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
GTC0_GTC_CFG3	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
I2C0_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
I2C1_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
I2C2_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
I2C3_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
I2C4_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
I2C5_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
I2C6_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MAIN_NAVSS Aliased spaces	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MAIN_NAVSS0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MAIN_SEC_MMR0_BOOT_CTRL	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MAIN_SEC_MMR0_DBG_CTRL	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MAIN_TO_MCU	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MAIN2MCU_LVL_INTRTR0_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MAIN2MCU_PLS_INTRTR0_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN0_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN0_ECC_AGGR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN0_MSGMEM_RAM	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-4. Connectivity Matrix (Part 1) (continued)

Targets	Initiators																						
	COMPUTE_CLUSTER0 (GIC)	DEBUGSS0_DBGMST	DEBUGSS0	DMPAC0_DATA	DSS0_DMA	DSS0_FBDC	EMMC8SS0	EMMCSD4SS0	ENCODER0	GPU0	LED	MCU_PDMA_ADC0_MEMR0	MCU_PDMA_MISC_G00	MCU_PDMA_MISC_G10	MCU_PDMA_MISC_G20	MCU_R5FSS0_CORE0	MCU_R5FSS0_CORE1	NAVSS MAIN	NAVSS MCU	PCIE0	PCIE1	PCIE2	PCIE3
MCAN0_SS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN1_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN1_ECC_AGGR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN1_MSGMEM_RAM	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN1_SS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN10_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN10_ECC_AGGR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN10_MSGMEM_RAM	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN10_SS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN11_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN11_ECC_AGGR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN11_MSGMEM_RAM	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN11_SS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN12_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN12_ECC_AGGR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN12_MSGMEM_RAM	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN12_SS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN13_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN13_ECC_AGGR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN13_MSGMEM_RAM	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN13_SS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN14_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN14_ECC_AGGR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN14_MSGMEM_RAM	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN14_SS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN15_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-4. Connectivity Matrix (Part 1) (continued)

Targets	Initiators																						
	COMPUTE_CLUSTER0 (GIC)	DEBUGSS0_DBGMST	DEBUGSS0	DMPAC0_DATA	DSS0_DMA	DSS0_FBDC	EMMC8SS0	EMMCSD4SS0	ENCODER0	GPU0	LED	MCU_PDMA_ADC0_MEMR0	MCU_PDMA_MISC_G00	MCU_PDMA_MISC_G10	MCU_PDMA_MISC_G20	MCU_R5FSS0_CORE0	MCU_R5FSS0_CORE1	NAVSS MAIN	NAVSS MCU	PCIE0	PCIE1	PCIE2	PCIE3
MCAN15_ECC_AGGR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN15_MSGMEM_RAM	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN15_SS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN16_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN16_ECC_AGGR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN16_MSGMEM_RAM	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN16_SS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN17_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN17_ECC_AGGR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN17_MSGMEM_RAM	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN17_SS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN2_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN2_ECC_AGGR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN2_MSGMEM_RAM	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN2_SS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN3_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN3_ECC_AGGR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN3_MSGMEM_RAM	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN3_SS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN4_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN4_ECC_AGGR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN4_MSGMEM_RAM	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN4_SS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN5_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN5_ECC_AGGR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN5_MSGMEM_RAM	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-4. Connectivity Matrix (Part 1) (continued)

Targets	Initiators																						
	COMPUTE_CLUSTER0 (GIC)	DEBUGSS0_DBGMST	DEBUGSS0	DMPAC0_DATA	DSS0_DMA	DSS0_FBDC	EMMC8SS0	EMMCSD4SS0	ENCODER0	GPU0	LED	MCU_PDMA_ADC0_MEMR0	MCU_PDMA_MISC_G00	MCU_PDMA_MISC_G10	MCU_PDMA_MISC_G20	MCU_R5FSS0_CORE0	MCU_R5FSS0_CORE1	NAVSS MAIN	NAVSS MCU	PCIE0	PCIE1	PCIE2	PCIE3
MCAN5_SS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN6_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN6_ECC_AGGR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN6_MSGMEM_RAM	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN6_SS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN7_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN7_ECC_AGGR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN7_MSGMEM_RAM	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN7_SS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN8_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN8_ECC_AGGR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN8_MSGMEM_RAM	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN8_SS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN9_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN9_ECC_AGGR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN9_MSGMEM_RAM	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCAN9_SS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCASP0_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCASP0_DMA	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCASP1_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCASP1_DMA	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCASP2_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCASP2_DMA	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCASP3_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCASP3_DMA	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCASP4_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-4. Connectivity Matrix (Part 1) (continued)

Targets	Initiators																						
	COMPUTE_CLUSTER0 (GIC)	DEBUGSS0_DBGMST	DEBUGSS0	DMPAC0_DATA	DSS0_DMA	DSS0_FBDC	EMMC8SS0	EMMCSD4SS0	ENCODER0	GPU0	LED	MCU_PDMA_ADC0_MEMR0	MCU_PDMA_MISC_G00	MCU_PDMA_MISC_G10	MCU_PDMA_MISC_G20	MCU_R5FSS0_CORE0	MCU_R5FSS0_CORE1	NAVSS MAIN	NAVSS MCU	PCIE0	PCIE1	PCIE2	PCIE3
MCASP4_DMA	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCSPI0_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCSPI1_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCSPI2_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCSPI3_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCSPI4_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCSPI5_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCSPI6_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCSPI7_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_ADC0	N	N	Y	Y	N	N	N	N	N	N	Y	Y	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_ADC0_ECC_REGS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_ADC0_FIFO	N	N	Y	Y	N	N	N	N	N	N	Y	Y	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_ADC1	N	N	Y	Y	N	N	N	N	N	N	Y	Y	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_ADC1_ECC_REGS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_ADC1_FIFO	N	N	Y	Y	N	N	N	N	N	N	Y	Y	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_CBASS_DEBUG0_ERR	N	Y	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_CBASS_FW0_ERR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_CBASS0_ERR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_CBASS0_FW	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_CBASS0_GLB	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_CBASS0_ISC	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_CBASS0_QOS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_CPSW0_ECC	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_CPSW0_NUSS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_CPT2_AGGR0_MMR	N	Y	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_CTRL_MMR0_CFG0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-4. Connectivity Matrix (Part 1) (continued)

Targets	Initiators																						
	COMPUTE_CLUSTER0 (GIC)	DEBUGSS0_DBGMST	DEBUGSS0	DMPAC0_DATA	DSS0_DMA	DSS0_FBDC	EMMC8SS0	EMMCSD4SS0	ENCODER0	GPU0	LED	MCU_PDMA_ADC0_MEMR0	MCU_PDMA_MISC_G00	MCU_PDMA_MISC_G10	MCU_PDMA_MISC_G20	MCU_R5FSS0_CORE0	MCU_R5FSS0_CORE1	NAVSS MAIN	NAVSS MCU	PCIE0	PCIE1	PCIE2	PCIE3
MCU_DCC0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_DCC1	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_DCC2	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_ECC_AGGR0_ECC_AGGR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_EFUSE0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_ESM0_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_FSS0_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_FSS0_DAT_REG0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_FSS0_DAT_REG1	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_FSS0_DAT_REG3	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_FSS0_FSAS_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_FSS0_HPB_CTRL	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_FSS0_HPB_ECC_AGGR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_FSS0_HPB_SS_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_FSS0_OSPI0_CTRL	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_FSS0_OSPI0_ECC_AGGR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_FSS0_OSPI0_SS_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_FSS0_OSPI1_CTRL	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_FSS0_OSPI1_ECC_AGGR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_FSS0_OSPI1_R0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_FSS0_OSPI1_R1	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_FSS0_OSPI1_R3	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_FSS0_OSPI1_SS_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_FSS0_OTFA_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_I2C0_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_I2C1_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-4. Connectivity Matrix (Part 1) (continued)

Targets	Initiators																						
	COMPUTE_CLUSTER0 (GIC)	DEBUGSS0_DBGMST	DEBUGSS0	DMPAC0_DATA	DSS0_DMA	DSS0_FBDC	EMMC8SS0	EMMCSD4SS0	ENCODER0	GPU0	LED	MCU_PDMA_ADC0_MEMR0	MCU_PDMA_MISC_G00	MCU_PDMA_MISC_G10	MCU_PDMA_MISC_G20	MCU_R5FSS0_CORE0	MCU_R5FSS0_CORE1	NAVSS MAIN	NAVSS MCU	PCIE0	PCIE1	PCIE2	PCIE3
MCU_I3C0_MMR_MMRVBP	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_I3C0_P_ECC_AGGR_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_I3C0_S_ECC_AGGR_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_I3C0_VBP2APB_WRAP_CORE_VBP_MIPI_I3C_MST	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_MCAN0_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	M	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_MCAN0_ECC_AGGR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_MCAN0_MSGMEM_RAM	N	N	Y	Y	N	N	N	N	N	N	Y	N	Y	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_MCAN0_SS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_MCAN1_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_MCAN1_ECC_AGGR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_MCAN1_MSGMEM_RAM	N	N	Y	Y	N	N	N	N	N	N	Y	N	Y	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_MCAN1_SS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_MCSPI0_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	Y	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_MCSPI1_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	Y	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_MCSPI2_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
MCU_MSRAM_1MB0_ECC_AGGR_REGS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_MSRAM_1MB0_RAM	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_NAVSS0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_NAVSS0_MODSS_SMS_FW	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_NAVSS0_MODSS_SMS_GLB	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_NAVSS0_MODSS_SMS_ISC	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_NAVSS0_MODSS_SMS_QOS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_NAVSS0_UDMASS_RINGACC0_ISC	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_PBIST0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_PBIST1	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_PLL0_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-4. Connectivity Matrix (Part 1) (continued)

Targets	Initiators																						
	COMPUTE_CLUSTER0 (GIC)	DEBUGSS0_DBGMST	DEBUGSS0	DMPAC0_DATA	DSS0_DMA	DSS0_FBDC	EMMC8SS0	EMMCSD4SS0	ENCODER0	GPU0	LED	MCU_PDMA_ADC0_MEMR0	MCU_PDMA_MISC_G00	MCU_PDMA_MISC_G10	MCU_PDMA_MISC_G20	MCU_R5FSS0_CORE0	MCU_R5FSS0_CORE1	NAVSS MAIN	NAVSS MCU	PCIE0	PCIE1	PCIE2	PCIE3
MCU_PSRAM0_RAM	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_R5FSS0_COMPARE_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_R5FSS0_CORE0_ATCM	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	N	Y	Y	Y	Y	Y	Y	Y
MCU_R5FSS0_CORE0_BTCM	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	N	Y	Y	Y	Y	Y	Y	Y
MCU_R5FSS0_CORE0_DCACHE	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_R5FSS0_CORE0_ECC_AGGR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_R5FSS0_CORE0_ICACHE	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_R5FSS0_CORE1_ATCM	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	N	Y	Y	Y	Y	Y	Y
MCU_R5FSS0_CORE1_BTCM	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	N	Y	Y	Y	Y	Y	Y
MCU_R5FSS0_CORE1_DCACHE	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_R5FSS0_CORE1_ICACHE	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_R5FSS0_ECC_AGGR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_R5FSS0_SOC_REGION	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_ROM0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_RTI0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_RTI1	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_SA3_UL0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_SA3_UL0_ECC_AGGR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_SEC_MMR0_CFG0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_SEC_MMR0_DBG_CTRL	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_TIMER0_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_TIMER1_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_TIMER2_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_TIMER3_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_TIMER4_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_TIMER5_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-4. Connectivity Matrix (Part 1) (continued)

Targets	Initiators																						
	COMPUTE_CLUSTER0 (GIC)	DEBUGSS0_DBGMST	DEBUGSS0	DMPAC0_DATA	DSS0_DMA	DSS0_FBDC	EMMC8SS0	EMMCSD4SS0	ENCODER0	GPU0	LED	MCU_PDMA_ADC0_MEMR0	MCU_PDMA_MISC_G00	MCU_PDMA_MISC_G10	MCU_PDMA_MISC_G20	MCU_R5FSS0_CORE0	MCU_R5FSS0_CORE1	NAVSS MAIN	NAVSS MCU	PCIE0	PCIE1	PCIE2	PCIE3
MCU_TIMER6_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_TIMER7_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_TIMER8_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_TIMER9_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_TO_MAIN	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MCU_UART0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y
MMCSD0_CTL_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MMCSD0_ECC_AGGR_RXMEM	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MMCSD0_ECC_AGGR_TXMEM	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MMCSD0_SS_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MMCSD1_CTL_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MMCSD1_ECC_AGGR_RXMEM	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MMCSD1_ECC_AGGR_TXMEM	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MMCSD1_SS_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MSRAM16KX256E0_ECC_AGGR_REGS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MSRAM16KX256E0_RAM	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MSRAM16KX256E1_ECC_AGGR_REGS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MSRAM16KX256E1_RAM	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MSRAM16KX256E2_ECC_AGGR_REGS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
MSRAM16KX256E2_RAM	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
NAVSS0_DDR0_MEM, NAVSS0_DDR1_MEM	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
NAVSS0_DDR0_MEM1,NAVSS0_DDR1_MEM1	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
NAVSS0_MODSS_SMS_FW	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
NAVSS0_MODSS_SMS_GLB	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
NAVSS0_MODSS_SMS_ISC	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
NAVSS0_MODSS_SMS_QOS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-4. Connectivity Matrix (Part 1) (continued)

Targets	Initiators																						
	COMPUTE_CLUSTER0 (GIC)	DEBUGSS0_DBGMST	DEBUGSS0	DMPAC0_DATA	DSS0_DMA	DSS0_FBDC	EMMC8SS0	EMMCSD4SS0	ENCODER0	GPU0	LED	MCU_PDMA_ADC0_MEMR0	MCU_PDMA_MISC_G00	MCU_PDMA_MISC_G10	MCU_PDMA_MISC_G20	MCU_R5FSS0_CORE0	MCU_R5FSS0_CORE1	NAVSS MAIN	NAVSS MCU	PCIE0	PCIE1	PCIE2	PCIE3
NAVSS0_NBSS_CFG_REGS0_MMRS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
NAVSS0_SRAM0, NAVSS0_SRAM1	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
NAVSS0_UDMASS_SMS_FW	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
NAVSS0_UDMASS_SMS_GLB	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
NAVSS0_UDMASS_SMS_ISC	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
NAVSS0_UDMASS_RINGACC0_ISC	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
NAVSS0_VIRTSS_SMS_FW	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
NAVSS0_VIRTSS_SMS_GLB	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
NAVSS0_VIRTSS_SMS_ISC	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
PBIST_MAIN_INFRA0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
PBIST_MAIN_INFRA1	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
PBIST_MAIN_INFRA2	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
PBIST0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
PBIST1	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
PBIST10	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
PBIST11	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
PBIST2	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
PBIST3	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
PBIST4	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
PBIST5	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
PBIST6	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
PBIST7	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
PBIST8	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
PBIST9	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
PCIE0_CORE_CPTS_CFG_CPTS_VBUSP	N	Y	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
PCIE0/4_CORE_DBN_CFG_PCIE_CORE	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-4. Connectivity Matrix (Part 1) (continued)

Targets	Initiators																						
	COMPUTE_CLUSTER0 (GIC)	DEBUGSS0_DBGMST	DEBUGSS0	DMPAC0_DATA	DSS0_DMA	DSS0_FBDC	EMMC8SS0	EMMCSD4SS0	ENCODER0	GPU0	LED	MCU_PDMA_ADC0_MEMR0	MCU_PDMA_MISC_G00	MCU_PDMA_MISC_G10	MCU_PDMA_MISC_G20	MCU_R5FSS0_CORE0	MCU_R5FSS0_CORE1	NAVSS MAIN	NAVSS MCU	PCIE0	PCIE1	PCIE2	PCIE3
PCIE0_CORE_ECC_AGGR0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
PCIE0_CORE_ECC_AGGR1	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
PCIE0_CORE_PCIE_INTD_CFG_INTD_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
PCIE0_CORE_USER_CFG_USER_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
PCIE0_CORE_VMAP*_MMRS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
PCIE0_DAT0	N	N	Y	Y	N	N	Y	Y	Y	Y	Y	N	N	N	N	Y	Y	Y	Y	N	Y	Y	Y
PCIE0_DAT1	N	N	Y	Y	N	N	Y	Y	Y	Y	Y	N	N	N	N	Y	Y	Y	Y	N	Y	Y	Y
PCIE1_CORE_CPTS_CFG_CPTS_VBUSP	N	Y	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
PCIE1_CORE_DBN_CFG_PCIE_CORE	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
PCIE1_CORE_ECC_AGGR0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
PCIE1_CORE_ECC_AGGR1	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
PCIE1_CORE_PCIE_INTD_CFG_INTD_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
PCIE1_CORE_USER_CFG_USER_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
PCIE1_CORE_VMAP*_MMRS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
PCIE1_DAT0	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	N	N	N	Y	Y	Y	Y	Y	N	Y	Y
PCIE1_DAT1	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	N	N	N	Y	Y	Y	Y	Y	N	Y	Y
PCIE2_CORE_CPTS_CFG_CPTS_VBUSP	N	Y	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
PCIE2_CORE_DBN_CFG_PCIE_CORE	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
PCIE2_CORE_ECC_AGGR0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
PCIE2_CORE_ECC_AGGR1	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
PCIE2_CORE_PCIE_INTD_CFG_INTD_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
PCIE2_CORE_USER_CFG_USER_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
PCIE2_CORE_VMAP_MMRS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
PCIE2_DAT0	N	N	Y	Y	N	N	Y	Y	Y	Y	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y
PCIE2_DAT1	N	N	Y	Y	N	N	Y	Y	Y	Y	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y
PCIE3_CORE_CPTS_CFG_CPTS_VBUSP	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-4. Connectivity Matrix (Part 1) (continued)

Targets	Initiators																						
	COMPUTE_CLUSTER0 (GIC)	DEBUGSS0_DBGMST	DEBUGSS0	DMPAC0_DATA	DSS0_DMA	DSS0_FBDC	EMMC8SS0	EMMCSD4SS0	ENCODER0	GPU0	LED	MCU_PDMA_ADC0_MEMR0	MCU_PDMA_MISC_G00	MCU_PDMA_MISC_G10	MCU_PDMA_MISC_G20	MCU_R5FSS0_CORE0	MCU_R5FSS0_CORE1	NAVSS MAIN	NAVSS MCU	PCIE0	PCIE1	PCIE2	PCIE3
PCIE3_CORE_DBN_CFG_PCIE_CORE	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
PCIE3_CORE_ECC_AGGR0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
PCIE3_CORE_ECC_AGGR1	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
PCIE3_CORE_PCIE_INTD_CFG_INTD_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
PCIE3_CORE_USER_CFG_USER_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
PCIE3_CORE_VMAP_MMRS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
PCIE3_DAT0	N	N	Y	Y	N	N	Y	Y	Y	Y	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	N
PCIE3_DAT1	N	N	Y	Y	N	N	Y	Y	Y	Y	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	N
PDMA_MCAN_ECC_AGGR_REGS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
PDMA_SPI_G0_ECC_AGGR_REGS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
PDMA_SPI_G1_ECC_AGGR_REGS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
PLL0_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
PLLCTRL0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
PSC0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
PSRAM2KECC0_ECC_AGGR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
PSRAM2KECC0_RAM	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
PSRAMECC0_ECC_AGGR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
PSRAMECC0_RAM	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS0_COMPARE_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS0_CORE0_ATCM	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS0_CORE0_BTCM	N	N	Y	Y	N	N	Y	Y	Y	Y	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS0_CORE0_DCACHE	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS0_CORE0_ECC_AGGR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS0_CORE0_ICACHE	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS0_CORE1_ATCM	N	N	Y	Y	N	N	Y	Y	Y	Y	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS0_CORE1_BTCM	N	N	Y	Y	N	N	Y	Y	Y	Y	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-4. Connectivity Matrix (Part 1) (continued)

Targets	Initiators																						
	COMPUTE_CLUSTER0 (GIC)	DEBUGSS0_DBGMST	DEBUGSS0	DMPAC0_DATA	DSS0_DMA	DSS0_FBDC	EMMC8SS0	EMMCSD4SS0	ENCODER0	GPU0	LED	MCU_PDMA_ADC0_MEMR0	MCU_PDMA_MISC_G00	MCU_PDMA_MISC_G10	MCU_PDMA_MISC_G20	MCU_R5FSS0_CORE0	MCU_R5FSS0_CORE1	NAVSS MAIN	NAVSS MCU	PCIE0	PCIE1	PCIE2	PCIE3
R5FSS0_CORE1_DCACHE	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS0_CORE1_ICACHE	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS0_ECC_AGGR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS0_EVENT_BUS_REGS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS0_SOC_REGION (Debug APB)	N	Y	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS1_COMPARE_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS1_CORE0_ATCM	N	N	Y	Y	N	N	Y	Y	Y	Y	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS1_CORE0_BTCM	N	N	Y	Y	N	N	Y	Y	Y	Y	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS1_CORE0_DCACHE	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS1_CORE0_ECC_AGGR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS1_CORE0_ICACHE	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS1_CORE1_ATCM	N	N	Y	Y	N	N	Y	Y	Y	Y	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS1_CORE1_BTCM	N	N	Y	Y	N	N	Y	Y	Y	Y	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS1_CORE1_DCACHE	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS1_CORE1_ICACHE	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS1_ECC_AGGR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS1_EVENT_BUS_REGS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS1_SOC_REGION (Debug APB)	N	Y	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS2_COMPARE_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS2_CORE0_ATCM	N	N	Y	Y	N	N	Y	Y	Y	Y	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS2_CORE0_BTCM	N	N	Y	Y	N	N	Y	Y	Y	Y	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS2_CORE0_DCACHE	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS2_CORE0_ECC_AGGR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS2_CORE0_ICACHE	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS2_CORE1_ATCM	N	N	Y	Y	N	N	Y	Y	Y	Y	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS2_CORE1_BTCM	N	N	Y	Y	N	N	Y	Y	Y	Y	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-4. Connectivity Matrix (Part 1) (continued)

Targets	Initiators																						
	COMPUTE_CLUSTER0 (GIC)	DEBUGSS0_DBGMST	DEBUGSS0	DMPAC0_DATA	DSS0_DMA	DSS0_FBDC	EMMC8SS0	EMMCSD4SS0	ENCODER0	GPU0	LED	MCU_PDMA_ADC0_MEMR0	MCU_PDMA_MISC_G00	MCU_PDMA_MISC_G10	MCU_PDMA_MISC_G20	MCU_R5FSS0_CORE0	MCU_R5FSS0_CORE1	NAVSS MAIN	NAVSS MCU	PCIE0	PCIE1	PCIE2	PCIE3
R5FSS2_CORE1_DCACHE	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS2_CORE1_ICACHE	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS2_ECC_AGGR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS2_EVENT_BUS_REGS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS2_SOC_REGION (Debug APB)	N	Y	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS3_SOC_REGION (Debug APB)	N	Y	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS4_SOC_REGION (Debug APB)	N	Y	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS5_SOC_REGION (Debug APB)	N	Y	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
RTI0_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
RTI1_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
RTI10_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
RTI11_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
RTI12_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
RTI13_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
RTI14_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
RTI15_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
RTI17_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
RTI18_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
RTI2_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
RTI3_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
RTI4_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
RTI5_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
RTI6_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
RTI7_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
RTI8_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
RTI9_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-4. Connectivity Matrix (Part 1) (continued)

Targets	Initiators																						
	COMPUTE_CLUSTER0 (GIC)	DEBUGSS0_DBGMST	DEBUGSS0	DMPAC0_DATA	DSS0_DMA	DSS0_FBDC	EMMC8SS0	EMMCSD4SS0	ENCODER0	GPU0	LED	MCU_PDMA_ADC0_MEMR0	MCU_PDMA_MISC_G00	MCU_PDMA_MISC_G10	MCU_PDMA_MISC_G20	MCU_R5FSS0_CORE0	MCU_R5FSS0_CORE1	NAVSS MAIN	NAVSS MCU	PCIE0	PCIE1	PCIE2	PCIE3
SA2_CPSW_PSILSS_MMRS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
SA2_UL0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
SA2_UL0_ECC_AGGR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
SERDES0_10G0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
SERDES0_16G0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
SERDES1_10G0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
SERDES1_16G0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
SERDES2_16G0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
SERDES4_10G0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
SPARE_AC_REGION1	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
SPARE_HC_REGION0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
SPARE_RAM_REGION0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
SPI_CPSW_PSILSS_MMRS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
STM0_CXSTM	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
STM0_STIMULUS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
STOG0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
STOG1	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
STOG2	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
STOG3	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
STOG4	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
STOG5	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
STOG6	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
STOG7	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
STOG8	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
STOG9	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
STOG10	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-4. Connectivity Matrix (Part 1) (continued)

Targets	Initiators																						
	COMPUTE_CLUSTER0 (GIC)	DEBUGSS0_DBGMST	DEBUGSS0	DMPAC0_DATA	DSS0_DMA	DSS0_FBDC	EMMC8SS0	EMMCSD4SS0	ENCODER0	GPU0	LED	MCU_PDMA_ADC0_MEMR0	MCU_PDMA_MISC_G00	MCU_PDMA_MISC_G10	MCU_PDMA_MISC_G20	MCU_R5FSS0_CORE0	MCU_R5FSS0_CORE1	NAVSS MAIN	NAVSS MCU	PCIE0	PCIE1	PCIE2	PCIE3
STOG11	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
STOG12	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
STOG13	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
STOG14	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
STOG15	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
TIMER0_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
TIMER1_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
TIMER2_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
TIMER3_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
TIMER4_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
TIMER5_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
TIMER6_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
TIMER7_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
TIMER8_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
TIMER9_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
TIMER10_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
TIMER11_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
TIMER12_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
TIMER13_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
TIMER14_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
TIMER15_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
TIMER16_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
TIMER17_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
TIMER18_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
TIMER19_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
TIMESYNC_INTRTR0_INTR_ROUTER_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-4. Connectivity Matrix (Part 1) (continued)

Targets	Initiators																						
	COMPUTE_CLUSTER0 (GIC)	DEBUGSS0_DBGMST	DEBUGSS0	DMPAC0_DATA	DSS0_DMA	DSS0_FBDC	EMMC8SS0	EMMCSD4SS0	ENCODER0	GPU0	LED	MCU_PDMA_ADC0_MEMR0	MCU_PDMA_MISC_G00	MCU_PDMA_MISC_G10	MCU_PDMA_MISC_G20	MCU_R5FSS0_CORE0	MCU_R5FSS0_CORE1	NAVSS MAIN	NAVSS MCU	PCIE0	PCIE1	PCIE2	PCIE3
UART_G00_PSILSS_MMRS	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
UART0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
UART1	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
UART2	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
UART3	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
UART4	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
UART5	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
UART6	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
UART7	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
UART8	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
UART9	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
UFS0_HCLK_ECC_AGGR_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
UFS0_IPS_TCLK_ERR_INJ_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
UFS0_P2A_WRAP_CFG_VBP_UFSHCI	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
UFS0_SYSCFG_SS_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
USB0_ECC_AGGR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
USB0_MMR_MMRVBP_USBSS_CMN	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
USB0_PHY2	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
USB0_RAMS_INJ_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
USB0_VBP2APB_WRAP_CONTROLLER_VBP_CORE_ADDR_MAP	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
VPAC0_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
VPAC0_SMS_VPAC_SCRMFW	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
VPAC0_SMS_VPAC_SCRMGLB	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
VPAC0_SMS_VPAC_SCRMISC	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
VPAC0_SMS_VPAC_SCRPFW	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-4. Connectivity Matrix (Part 1) (continued)

Targets	Initiators																						
	COMPUTE_CLUSTER0 (GIC)	DEBUGSS0_DBGMST	DEBUGSS0	DMPAC0_DATA	DSS0_DMA	DSS0_FBDC	EMMC8SS0	EMMCSD4SS0	ENCODER0	GPU0	LED	MCU_PDMA_ADC0_MEMR0	MCU_PDMA_MISC_G00	MCU_PDMA_MISC_G10	MCU_PDMA_MISC_G20	MCU_R5FSS0_CORE0	MCU_R5FSS0_CORE1	NAVSS MAIN	NAVSS MCU	PCIE0	PCIE1	PCIE2	PCIE3
VPAC0_SMS_VPAC_SCRPGLB	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
VPAC0_KSDW_ECC_AGGR_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
VPAC0_VPAC_TOP_PAC_BASE_MEM_SLV_CBASS_STRIPE_MSRAM_SLV	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
VPAC1_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
VPAC1_SMS_VPAC_SCRMFW	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
VPAC1_SMS_VPAC_SCRMGLB	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
VPAC1_SMS_VPAC_SCRMISC	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
VPAC1_SMS_VPAC_SCRPFW	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
VPAC1_SMS_VPAC_SCRPGLB	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
VPAC1_KSDW_ECC_AGGR_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
VPAC1_VPAC_TOP_PAC_BASE_MEM_SLV_CBASS_STRIPE_MSRAM_SLV	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
VUSR0_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
VUSR0_CTL_ECC	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
VUSR0_PORTAL	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
VUSR1_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
VUSR1_CTL_ECC	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
VUSR1_PORTAL	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
WAVE521CL0/1_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
WKUP_CBASS_FW0_ERR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
WKUP_CBASS0_ERR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
WKUP_CBASS0_FW	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
WKUP_CBASS0_GLB	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
WKUP_CTRL_MMR0_CFG0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
WKUP_DDPA0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-4. Connectivity Matrix (Part 1) (continued)

Targets	Initiators																						
	COMPUTE_CLUSTER0 (GIC)	DEBUGSS0_DBGMST	DEBUGSS0	DMPAC0_DATA	DSS0_DMA	DSS0_FBDC	EMMC8SS0	EMMCSD4SS0	ENCODER0	GPU0	LED	MCU_PDMA_ADC0_MEMR0	MCU_PDMA_MISC_G00	MCU_PDMA_MISC_G10	MCU_PDMA_MISC_G20	MCU_R5FSS0_CORE0	MCU_R5FSS0_CORE1	NAVSS MAIN	NAVSS MCU	PCIE0	PCIE1	PCIE2	PCIE3
WKUP_SMS0_ECC_AGGR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
WKUP_SMS0_IRAM	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
WKUP_ECC_AGGR0_ECC_AGGR	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
WKUP_ESM0_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
WKUP_GPIO0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
WKUP_GPIO1	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
WKUP_GPIOMUX_INTRTR0_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
WKUP_I2C0_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
WKUP_PLLCTRL0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
WKUP_PSC0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
WKUP_UART0	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
WKUP_VTM0_ECCAGGR_CFG	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
WKUP_VTM0_MMR_VBUSP_CFG1	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
WKUP_VTM0_MMR_VBUSP_CFG2	N	N	Y	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-5. Connectivity Matrix (Part 2)

Targets	Initiators																						
	PDMA_DEBUG_CCSOC0_MEMR0	PDMA_DEBUG_CCSOC1_MEMR0	PDMA_MCAN0	PDMA_MCASP0	PDMA_SPI_G00	PDMA_SPI_G10	PDMA_UART_G00	PDMA_UART_G10	PDMA_UART_G20	R5FSS0_CORE0	R5FSS1_CORE0	R5FSS2_CORE0	R5FSS0_CORE1	R5FSS1_CORE1	R5FSS2_CORE1	SA2_UL	SPARE_MST_RSVD	UFS0	USB0	VPAC0_DATA_MST0	VPAC0_DATA_MST1	VPAC0_LDC0	VPAC1_DATA_MST0
ATL0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
BWLIMITER_ AC_SPARE1	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
BWLIMITER_ AC_SPARE2	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
BWLIMITER_ AC_SPARE3	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
BWLIMITER_ AC_SPARE4	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
BWLIMITER_ AC_SPARE5	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
BWLIMITER_ GPU0_M0_WR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
BWLIMITER_ GPU0_M1_RD	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
BWLIMITER_ GPU0_M1_WR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
BWLIMITER_ HC_SPARE1	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
BWLIMITER_ HC_SPARE2	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
BWLIMITER_ HC_SPARE3	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
BWLIMITER_ WAVE521CL0/1_WR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
BWLIMITER_AC_SPARE0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
BWLIMITER_ENCODER0_RD	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
BWLIMITER_ENCODER0_WR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
BWLIMITER_GPU0_M0_RD	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
BWLIMITER_HC_SPARE0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
BWLIMITER_WAVE521CL0/1_RD	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_ SPARE1_GLB	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_ SPARE1_QOS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_ SPARE3_GLB	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_AC_ NONSAFE0_ERR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_AC_CFG_NONSAFE0_ERR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-5. Connectivity Matrix (Part 2) (continued)

Targets	Initiators																						
	PDMA_DEBUG_CCSOC0_MEMR0	PDMA_DEBUG_CCSOC1_MEMR0	PDMA_MCAN0	PDMA_MCASP0	PDMA_SPI_G00	PDMA_SPI_G10	PDMA_UART_G00	PDMA_UART_G10	PDMA_UART_G20	R5FSS0_CORE0	R5FSS1_CORE0	R5FSS2_CORE0	R5FSS0_CORE1	R5FSS1_CORE1	R5FSS2_CORE1	SA2_UL	SPARE_MST_RSVD	UFS0	USB0	VPAC0_DATA_MST0	VPAC0_DATA_MST1	VPAC0_LDC0	VPAC1_DATA_MST0
CBASS_AC_CFG_NONSAFE0_FW	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_AC_CFG_NONSAFE0_GLB	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_AC_CFG0_ERR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_AC_CFG0_FW	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_AC_CFG0_GLB	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_AC_MERGER0_ERR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_AC_NONSAFE0_FW	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_AC_NONSAFE0_GLB	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_AC_NONSAFE0_ISC	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_AC_NONSAFE0_QOS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_CSI0_ERR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_CSI0_FW	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_CSI0_GLB	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_DATADEBUG0_ERR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_DATADEBUG0_FW	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_DATADEBUG0_GLB	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_DATADEBUG0_ISC	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_DATADEBUG0_QOS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_DEBUG0_ERR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_FW0_ERR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_HC_CFG_B0_ERR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_HC_CFG_B0_FW	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_HC_CFG_B0_GLB	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_HC20_ERR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-5. Connectivity Matrix (Part 2) (continued)

Targets	Initiators																						
	PDMA_DEBUG_CCSOC0_MEMR0	PDMA_DEBUG_CCSOC1_MEMR0	PDMA_MCAN0	PDMA_MCASP0	PDMA_SPI_G00	PDMA_SPI_G10	PDMA_UART_G00	PDMA_UART_G10	PDMA_UART_G20	R5FSS0_CORE0	R5FSS1_CORE0	R5FSS2_CORE0	R5FSS0_CORE1	R5FSS1_CORE1	R5FSS2_CORE1	SA2_UL	SPARE_MST_RSVD	UFS0	USB0	VPAC0_DATA_MST0	VPAC0_DATA_MST1	VPAC0_LDC0	VPAC1_DATA_MST0
CBASS_HC20_FW	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_HC20_GLB	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_HC20_ISC	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_HC20_QOS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_INFRA_NON_SAFE0_ERR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_INFRA0_ERR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_INFRA0_FW	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_INFRA0_GLB	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_IPPHY_SAFE0_ERR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_IPPHY_SAFE0_FW	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_IPPHY_SAFE0_GLB	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_IPPHY0_ERR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_IPPHY0_FW	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_IPPHY0_GLB	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_IPPHY0_ISC	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_IPPHY0_QOS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_MAIN_NON_INFRA_FW	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_MAIN_NON_INFRA_GLB	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_R5FSS0_MEM_ERR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_R5FSS0_MEM_FW	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_R5FSS0_MEM_GLB	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_R5FSS0_MEM_ISC	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_R5FSS0_MEM_QOS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_R5FSS0_SLV_ERR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-5. Connectivity Matrix (Part 2) (continued)

Targets	Initiators																						
	PDMA_DEBUG_CCSOC0_MEMR0	PDMA_DEBUG_CCSOC1_MEMR0	PDMA_MCAN0	PDMA_MCASP0	PDMA_SPI_G00	PDMA_SPI_G10	PDMA_UART_G00	PDMA_UART_G10	PDMA_UART_G20	R5FSS0_CORE0	R5FSS1_CORE0	R5FSS2_CORE0	R5FSS0_CORE1	R5FSS1_CORE1	R5FSS2_CORE1	SA2_UL	SPARE_MST_RSVD	UFS0	USB0	VPAC0_DATA_MST0	VPAC0_DATA_MST1	VPAC0_LDC0	VPAC1_DATA_MST0
CBASS_R5FSS0_SLV_FW	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_R5FSS0_SLV_GLB	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_R5FSS1_MEM_ERR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_R5FSS1_MEM_FW	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_R5FSS1_MEM_GLB	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_R5FSS1_MEM_ISC	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_R5FSS1_MEM_QOS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_R5FSS1_PERIPH_ERR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_R5FSS1_PERIPH_GLB	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_R5FSS1_PERIPH_ISC	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_R5FSS1_PERIPH_QOS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_RC_CFG_B0_ERR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_RC_CFG_B0_FW	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_RC_CFG_B0_GLB	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_RC0_ERR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_RC0_FW	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_RC0_GLB	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_RC0_ISC	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_RC0_QOS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_SPARE_FW	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_SPARE0_ERR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_SPARE0_GLB	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_SPARE0_ISC	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_SPARE0_QOS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-5. Connectivity Matrix (Part 2) (continued)

Targets	Initiators																						
	PDMA_DEBUG_CCSOC0_MEMR0	PDMA_DEBUG_CCSOC1_MEMR0	PDMA_MCAN0	PDMA_MCASP0	PDMA_SPI_G00	PDMA_SPI_G10	PDMA_UART_G00	PDMA_UART_G10	PDMA_UART_G20	R5FSS0_CORE0	R5FSS1_CORE0	R5FSS2_CORE0	R5FSS0_CORE1	R5FSS1_CORE1	R5FSS2_CORE1	SA2_UL	SPARE_MST_RSVD	UFS0	USB0	VPAC0_DATA_MST0	VPAC0_DATA_MST1	VPAC0_LDC0	VPAC1_DATA_MST0
CBASS_SPARE1_ERR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_SPARE1_ISC	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_SPARE2_ERR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_SPARE2_GLB	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CBASS_SPARE3_ERR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CCDEBUGSS0_ROM	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CCDEBUGSS0_SYS	N	Y	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CCDEBUGSS1_ROM	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CCDEBUGSS1_SYS	Y	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CCDEBUGSS2_ROM	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CCDEBUGSS2_SYS	N	Y	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CMPEVENT_INTRTR0_INTR_ROUTER_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
COMPUTE_CLUSTER0_CCROM	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
COMPUTE_CLUSTER0_DDRCTL0_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
COMPUTE_CLUSTER0_DDRCTL1_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
COMPUTE_CLUSTER0_DDRCTL2_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
COMPUTE_CLUSTER0_DDRSS0_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
COMPUTE_CLUSTER0_DDRSS1_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
COMPUTE_CLUSTER0_DDRSS2_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
COMPUTE_CLUSTER0_DDRSS3_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
COMPUTE_CLUSTER0_SMS_BOOT	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
COMPUTE_CLUSTER0_SMS_EMULATION	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
COMPUTE_CLUSTER0_SMS_FW	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
COMPUTE_CLUSTER0_SMS_GLB	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-5. Connectivity Matrix (Part 2) (continued)

Targets	Initiators																						
	PDMA_DEBUG_CCSOC0_MEMR0	PDMA_DEBUG_CCSOC1_MEMR0	PDMA_MCAN0	PDMA_MCASP0	PDMA_SPI_G00	PDMA_SPI_G10	PDMA_UART_G00	PDMA_UART_G10	PDMA_UART_G20	R5FSS0_CORE0	R5FSS1_CORE0	R5FSS2_CORE0	R5FSS0_CORE1	R5FSS1_CORE1	R5FSS2_CORE1	SA2_UL	SPARE_MST_RSVD	UFS0	USB0	VPAC0_DATA_MST0	VPAC0_DATA_MST1	VPAC0_LDC0	VPAC1_DATA_MST0
COMPUTE_CLUSTER0_SMS_PRIVID	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
COMPUTE_CLUSTER0_GIC_DISTRIBUTOR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
COMPUTE_CLUSTER0_GIC_TRANSLATER	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
COMPUTE_CLUSTER0_MSMC0_CC	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
COMPUTE_CLUSTER0_MSMC0_ECC_AGGR0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
COMPUTE_CLUSTER0_MSMC0_PBIST	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
COMPUTE_CLUSTER0_MSMC0_SRAM	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
COMPUTE_CLUSTER1_CCROM	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CPSW0_ECC	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CPSW0_NUSS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CPSW1_ECC	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CPSW1_NUSS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CPT2_AGGR0_MMR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CPT2_AGGR1_MMR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CPT2_AGGR2_MMR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CPT2_AGGR3_MMR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CPT2_AGGR4_MMR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CPT2_AGGR5_MMR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CPT2_AGGR6_MMR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CPT2_AGGR7_MMR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CSI_PSILSS0_MMRS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CSI_RX_IF0_CP_INTD_CFG_INTD_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CSI_RX_IF0_ECC_AGGR_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CSI_RX_IF0_RX_SHIM_VBUSP_MMR_CSI2RXIF	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-5. Connectivity Matrix (Part 2) (continued)

Targets	Initiators																						
	PDMA_DEBUG_CCSOC0_MEMR0	PDMA_DEBUG_CCSOC1_MEMR0	PDMA_MCAN0	PDMA_MCASP0	PDMA_SPI_G00	PDMA_SPI_G10	PDMA_UART_G00	PDMA_UART_G10	PDMA_UART_G20	R5FSS0_CORE0	R5FSS1_CORE0	R5FSS2_CORE0	R5FSS0_CORE1	R5FSS1_CORE1	R5FSS2_CORE1	SA2_UL	SPARE_MST_RSVD	UFS0	USB0	VPAC0_DATA_MST0	VPAC0_DATA_MST1	VPAC0_LDC0	VPAC1_DATA_MST0
CSI_RX_IF0_VBUS2APB_WRAP_VBUSP_APB_CSI2RX	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CSI_RX_IF1_CP_INTD_CFG_INTD_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CSI_RX_IF1_ECC_AGGR_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CSI_RX_IF1_RX_SHIM_VBUSP_MMR_CSI2RXIF	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CSI_RX_IF1_VBUS2APB_WRAP_VBUSP_APB_CSI2RX	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CSI_RX_IF2_CP_INTD_CFG_INTD_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CSI_RX_IF2_ECC_AGGR_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CSI_RX_IF2_RX_SHIM_VBUSP_MMR_CSI2RXIF	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CSI_RX_IF2_VBUS2APB_WRAP_VBUSP_APB_CSI2RX	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CSI_TX_IF0_CP_INTD_CFG_INTD_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CSI_TX_IF0_ECC_AGGR_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CSI_TX_IF0_TX_SHIM_VBUSP_MMR_CSI2TXIF	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CSI_TX_IF0_VBUS2APB_WRAP_VBUSP_APB_CSI2TX	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CSI_TX_IF1_CP_INTD_CFG_INTD_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CSI_TX_IF1_ECC_AGGR_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CSI_TX_IF1_TX_SHIM_VBUSP_MMR_CSI2TXIF	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CSI_TX_IF1_VBUS2APB_WRAP_VBUSP_APB_CSI2TX	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
CTRL_MMR0_CFG0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
DCC0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
DCC1	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
DCC2	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
DCC3	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
DCC4	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
DCC5	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-5. Connectivity Matrix (Part 2) (continued)

Targets	Initiators																						
	PDMA_DEBUG_CCSOC0_MEMR0	PDMA_DEBUG_CCSOC1_MEMR0	PDMA_MCAN0	PDMA_MCASP0	PDMA_SPI_G00	PDMA_SPI_G10	PDMA_UART_G00	PDMA_UART_G10	PDMA_UART_G20	R5FSS0_CORE0	R5FSS1_CORE0	R5FSS2_CORE0	R5FSS0_CORE1	R5FSS1_CORE1	R5FSS2_CORE1	SA2_UL	SPARE_MST_RSVD	UFS0	USB0	VPAC0_DATA_MST0	VPAC0_DATA_MST1	VPAC0_LDC0	VPAC1_DATA_MST0
DCC6	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
DCC7	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
DCC8	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
DCC9	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
DEBUG_PSILSS_MMRS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
DEBUGSS_WRAP0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
DEBUGSS1_ROM	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
DEBUGSS1_SYS	Y	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
DEBUGSUSPENDRTR0_INTR_ROUTER_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
DFTSS0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
DMPAC_VPAC_PSILSS_MMRS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
DMPAC0_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
DMPAC0_CFG_SMS_FW	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
DMPAC0_CFG_SMS_GLB	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
DMPAC0_DMPAC_TOP_DOF_INFRA_DMPAC_BASE_MEM_SLV_CBASS_STRIPE_MSRAM_SLV	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
DMPAC0_KSDW_ECC_AGGR_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
DMPAC0_SL2_SMS_FW	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
DMPAC0_SL2_SMS_GLB	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
DMPAC0_SL2_SMS_ISC	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
DMSS_HSM_CFG1	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
DMSS_HSM_SECPROXY	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
DPHY_RX0_VBUS2APB_WRAP_VBUSP_K3_DPHY_RX	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
DPHY_RX1_VBUS2APB_WRAP_VBUSP_K3_DPHY_RX	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-5. Connectivity Matrix (Part 2) (continued)

Targets	Initiators																						
	PDMA_DEBUG_CCSOC0_MEMR0	PDMA_DEBUG_CCSOC1_MEMR0	PDMA_MCAN0	PDMA_MCASP0	PDMA_SPI_G00	PDMA_SPI_G10	PDMA_UART_G00	PDMA_UART_G10	PDMA_UART_G20	R5FSS0_CORE0	R5FSS1_CORE0	R5FSS2_CORE0	R5FSS0_CORE1	R5FSS1_CORE1	R5FSS2_CORE1	SA2_UL	SPARE_MST_RSVD	UFS0	USB0	VPAC0_DATA_MST0	VPAC0_DATA_MST1	VPAC0_LDC0	VPAC1_DATA_MST0
DPHY_RX2_VBUS2APB_WRAP_VBUSP_K3_DPHY_RX	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
DPHY_TX0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
DPHY_TX1	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
DSS_DSI0_DSI_TOP_ECC_AGGR_SYS_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
DSS_DSI0_DSI_WRAP_MMR_VBUSP_CFG_DSI_WRAP	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
DSS_DSI1_DSI_TOP_ECC_AGGR_SYS_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
DSS_DSI1_DSI_WRAP_MMR_VBUSP_CFG_DSI_WRAP	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
DSS_EDP0_INTG_CFG_VP	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
DSS_EDP0_MHDPTX_WRAPPER_ECC_AGGR_CORE_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
DSS_EDP0_MHDPTX_WRAPPER_ECC_AGGR_DSC_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
DSS_EDP0_MHDPTX_WRAPPER_ECC_AGGR_PHY_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
DSS_EDP0_V2A_CORE_VP_REGS_APB	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
DSS_EDP0_V2A_S_CORE_VP_REGS_SAPB	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
DSS0_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
DSSUL_DSI0_DSI_TOP_VBUSP_CFG_DSI_0_DSI	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
DSSUL_DSI1_DSI_TOP_VBUSP_CFG_DSI_0_DSI	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
DSSUL0_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
ECAP0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
ECAP1	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
ECAP2	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
ECC_AGGR_IPPHY_REGS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
ECC_AGGR0_ECC_AGGR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
ECC_AGGR10_REGS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
ECC_AGGR11_REGS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-5. Connectivity Matrix (Part 2) (continued)

Targets	Initiators																						
	PDMA_DEBUG_CCSOC0_MEMR0	PDMA_DEBUG_CCSOC1_MEMR0	PDMA_MCAN0	PDMA_MCASP0	PDMA_SPI_G00	PDMA_SPI_G10	PDMA_UART_G00	PDMA_UART_G10	PDMA_UART_G20	R5FSS0_CORE0	R5FSS1_CORE0	R5FSS2_CORE0	R5FSS0_CORE1	R5FSS1_CORE1	R5FSS2_CORE1	SA2_UL	SPARE_MST_RSVD	UFS0	USB0	VPAC0_DATA_MST0	VPAC0_DATA_MST1	VPAC0_LDC0	VPAC1_DATA_MST0
ECC_AGGR16_REGS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
ECC_AGGR17_REGS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
ECC_AGGR18_REGS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
ECC_AGGR19_REGS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
ECC_AGGR20_REGS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
ECC_AGGR21_REGS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
ECC_AGGR4_REGS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
ECC_AGGR5_REGS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
ECC_AGGR6_REGS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
ECC_AGGR6_REGS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
EFUSE0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
ELM0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
ENCODER0_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
EPWM0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
EPWM1	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
EPWM2	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
EPWM3	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
EPWM4	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
EPWM5	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
EQEP0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
EQEP1	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
EQEP2	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
ESM0_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
GPIO0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-5. Connectivity Matrix (Part 2) (continued)

Targets	Initiators																						
	PDMA_DEBUG_CCSOC0_MEMR0	PDMA_DEBUG_CCSOC1_MEMR0	PDMA_MCAN0	PDMA_MCASP0	PDMA_SPI_G00	PDMA_SPI_G10	PDMA_UART_G00	PDMA_UART_G10	PDMA_UART_G20	R5FSS0_CORE0	R5FSS1_CORE0	R5FSS2_CORE0	R5FSS0_CORE1	R5FSS1_CORE1	R5FSS2_CORE1	SA2_UL	SPARE_MST_RSVD	UFS0	USB0	VPAC0_DATA_MST0	VPAC0_DATA_MST1	VPAC0_LDC0	VPAC1_DATA_MST0
GPIO2	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
GPIO4	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
GPIO6	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
GPIOMUX_INTRTR0_INTR_ROUTER_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
GPMC0_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
GPMC0_DATA	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
GPU0_CORE_MMRS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
GPU0_PBIST_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
GTC0_GTC_CFG0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
GTC0_GTC_CFG1	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
GTC0_GTC_CFG2	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
GTC0_GTC_CFG3	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
I2C0_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
I2C1_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
I2C2_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
I2C3_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
I2C4_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
I2C5_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
I2C6_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MAIN_NAVSS Aliased spaces	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MAIN_NAVSS0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MAIN_SEC_MMR0_BOOT_CTRL	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MAIN_SEC_MMR0_DBG_CTRL	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MAIN_TO_MCU	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-5. Connectivity Matrix (Part 2) (continued)

Targets	Initiators																						
	PDMA_DEBUG_CCSOC0_MEMR0	PDMA_DEBUG_CCSOC1_MEMR0	PDMA_MCAN0	PDMA_MCASP0	PDMA_SPI_G00	PDMA_SPI_G10	PDMA_UART_G00	PDMA_UART_G10	PDMA_UART_G20	R5FSS0_CORE0	R5FSS1_CORE0	R5FSS2_CORE0	R5FSS0_CORE1	R5FSS1_CORE1	R5FSS2_CORE1	SA2_UL	SPARE_MST_RSVD	UFS0	USB0	VPAC0_DATA_MST0	VPAC0_DATA_MST1	VPAC0_LDC0	VPAC1_DATA_MST0
MAIN2MCU_LVL_INTRTR0_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MAIN2MCU_PLS_INTRTR0_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN0_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN0_ECC_AGGR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN0_MSGMEM_RAM	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN0_SS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN1_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN1_ECC_AGGR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN1_MSGMEM_RAM	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN1_SS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN10_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN10_ECC_AGGR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN10_MSGMEM_RAM	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN10_SS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN11_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN11_ECC_AGGR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN11_MSGMEM_RAM	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN11_SS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN12_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN12_ECC_AGGR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN12_MSGMEM_RAM	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN12_SS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN13_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN13_ECC_AGGR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-5. Connectivity Matrix (Part 2) (continued)

Targets	Initiators																						
	PDMA_DEBUG_CCSOC0_MEMR0	PDMA_DEBUG_CCSOC1_MEMR0	PDMA_MCAN0	PDMA_MCASP0	PDMA_SPI_G00	PDMA_SPI_G10	PDMA_UART_G00	PDMA_UART_G10	PDMA_UART_G20	R5FSS0_CORE0	R5FSS1_CORE0	R5FSS2_CORE0	R5FSS0_CORE1	R5FSS1_CORE1	R5FSS2_CORE1	SA2_UL	SPARE_MST_RSVD	UFS0	USB0	VPAC0_DATA_MST0	VPAC0_DATA_MST1	VPAC0_LDC0	VPAC1_DATA_MST0
MCAN13_MSGMEM_RAM	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN13_SS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN14_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN14_ECC_AGGR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN14_MSGMEM_RAM	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN14_SS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN15_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN15_ECC_AGGR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN15_MSGMEM_RAM	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN15_SS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN16_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN16_ECC_AGGR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN16_MSGMEM_RAM	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN16_SS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN17_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN17_ECC_AGGR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN17_MSGMEM_RAM	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN17_SS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN2_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN2_ECC_AGGR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN2_MSGMEM_RAM	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN2_SS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN3_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN3_ECC_AGGR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-5. Connectivity Matrix (Part 2) (continued)

Targets	Initiators																						
	PDMA_DEBUG_CCSOC0_MEMR0	PDMA_DEBUG_CCSOC1_MEMR0	PDMA_MCAN0	PDMA_MCASP0	PDMA_SPI_G00	PDMA_SPI_G10	PDMA_UART_G00	PDMA_UART_G10	PDMA_UART_G20	R5FSS0_CORE0	R5FSS1_CORE0	R5FSS2_CORE0	R5FSS0_CORE1	R5FSS1_CORE1	R5FSS2_CORE1	SA2_UL	SPARE_MST_RSVD	UFS0	USB0	VPAC0_DATA_MST0	VPAC0_DATA_MST1	VPAC0_LDC0	VPAC1_DATA_MST0
MCAN3_MSGMEM_RAM	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN3_SS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN4_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN4_ECC_AGGR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN4_MSGMEM_RAM	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN4_SS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN5_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN5_ECC_AGGR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN5_MSGMEM_RAM	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN5_SS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN6_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN6_ECC_AGGR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN6_MSGMEM_RAM	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN6_SS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN7_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN7_ECC_AGGR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN7_MSGMEM_RAM	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN7_SS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN8_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN8_ECC_AGGR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN8_MSGMEM_RAM	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN8_SS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN9_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN9_ECC_AGGR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-5. Connectivity Matrix (Part 2) (continued)

Targets	Initiators																						
	PDMA_DEBUG_CCSOC0_MEMR0	PDMA_DEBUG_CCSOC1_MEMR0	PDMA_MCAN0	PDMA_MCASP0	PDMA_SPI_G00	PDMA_SPI_G10	PDMA_UART_G00	PDMA_UART_G10	PDMA_UART_G20	R5FSS0_CORE0	R5FSS1_CORE0	R5FSS2_CORE0	R5FSS0_CORE1	R5FSS1_CORE1	R5FSS2_CORE1	SA2_UL	SPARE_MST_RSVD	UFS0	USB0	VPAC0_DATA_MST0	VPAC0_DATA_MST1	VPAC0_LDC0	VPAC1_DATA_MST0
MCAN9_MSGMEM_RAM	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCAN9_SS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCASP0_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCASP0_DMA	N	N	N	Y	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCASP1_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCASP1_DMA	N	N	N	Y	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCASP2_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCASP2_DMA	N	N	N	Y	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCASP3_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCASP3_DMA	N	N	N	Y	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCASP4_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCASP4_DMA	N	N	N	Y	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCSPI0_CFG	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCSPI1_CFG	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCSPI2_CFG	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCSPI3_CFG	N	N	N	N	Y	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCSPI4_CFG	N	N	N	N	N	Y	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCSPI5_CFG	N	N	N	N	N	Y	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCSPI6_CFG	N	N	N	N	N	Y	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCSPI7_CFG	N	N	N	N	N	Y	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_ADC0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_ADC0_ECC_REGS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_ADC0_FIFO	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_ADC1	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-5. Connectivity Matrix (Part 2) (continued)

Targets	Initiators																						
	PDMA_DEBUG_CCSOC0_MEMR0	PDMA_DEBUG_CCSOC1_MEMR0	PDMA_MCAN0	PDMA_MCASP0	PDMA_SPI_G00	PDMA_SPI_G10	PDMA_UART_G00	PDMA_UART_G10	PDMA_UART_G20	R5FSS0_CORE0	R5FSS1_CORE0	R5FSS2_CORE0	R5FSS0_CORE1	R5FSS1_CORE1	R5FSS2_CORE1	SA2_UL	SPARE_MST_RSVD	UFS0	USB0	VPAC0_DATA_MST0	VPAC0_DATA_MST1	VPAC0_LDC0	VPAC1_DATA_MST0
MCU_ADC1_ECC_REGS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_ADC1_FIFO	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_CBASS_DEBUG0_ERR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_CBASS_FW0_ERR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_CBASS0_ERR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_CBASS0_FW	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_CBASS0_GLB	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_CBASS0_ISC	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_CBASS0_QOS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_CPSW0_ECC	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_CPSW0_NUSS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_CPT2_AGGR0_MMR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_CTRL_MMR0_CFG0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_DCC0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_DCC1	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_DCC2	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_ECC_AGGR0_ECC_AGGR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_EFUSE0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_ESM0_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_FSS0_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_FSS0_DAT_REG0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_FSS0_DAT_REG1	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_FSS0_DAT_REG3	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_FSS0_FSAS_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-5. Connectivity Matrix (Part 2) (continued)

Targets	Initiators																						
	PDMA_DEBUG_CCSOC0_MEMR0	PDMA_DEBUG_CCSOC1_MEMR0	PDMA_MCAN0	PDMA_MCASP0	PDMA_SPI_G00	PDMA_SPI_G10	PDMA_UART_G00	PDMA_UART_G10	PDMA_UART_G20	R5FSS0_CORE0	R5FSS1_CORE0	R5FSS2_CORE0	R5FSS0_CORE1	R5FSS1_CORE1	R5FSS2_CORE1	SA2_UL	SPARE_MST_RSVD	UFS0	USB0	VPAC0_DATA_MST0	VPAC0_DATA_MST1	VPAC0_LDC0	VPAC1_DATA_MST0
MCU_FSS0_HPB_CTRL	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_FSS0_HPB_ECC_AGGR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_FSS0_HPB_SS_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_FSS0_OSPI0_CTRL	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_FSS0_OSPI0_ECC_AGGR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_FSS0_OSPI0_SS_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_FSS0_OSPI1_CTRL	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_FSS0_OSPI1_ECC_AGGR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_FSS0_OSPI1_R0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_FSS0_OSPI1_R1	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_FSS0_OSPI1_R3	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_FSS0_OSPI1_SS_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_FSS0_OTFA_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_I2C0_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_I2C1_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_I3C0_MMR_MMRVBP	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_I3C0_P_ECC_AGGR_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_I3C0_S_ECC_AGGR_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_I3C0_VBP2APB_WRAP_CORE_VBP_MIPI_I3C_MST	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_MCAN0_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_MCAN0_ECC_AGGR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_MCAN0_MSGMEM_RAM	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_MCAN0_SS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_MCAN1_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-5. Connectivity Matrix (Part 2) (continued)

Targets	Initiators																						
	PDMA_DEBUG_CCSOC0_MEMR0	PDMA_DEBUG_CCSOC1_MEMR0	PDMA_MCAN0	PDMA_MCASP0	PDMA_SPI_G00	PDMA_SPI_G10	PDMA_UART_G00	PDMA_UART_G10	PDMA_UART_G20	R5FSS0_CORE0	R5FSS1_CORE0	R5FSS2_CORE0	R5FSS0_CORE1	R5FSS1_CORE1	R5FSS2_CORE1	SA2_UL	SPARE_MST_RSVD	UFS0	USB0	VPAC0_DATA_MST0	VPAC0_DATA_MST1	VPAC0_LDC0	VPAC1_DATA_MST0
MCU_MCAN1_ECC_AGGR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_MCAN1_MSGMEM_RAM	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_MCAN1_SS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_MCSPI0_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_MCSPI1_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_MCSPI2_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_MSRAM_1MB0_ECC_AGGR_REGS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_MSRAM_1MB0_RAM	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_NAVSS0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_NAVSS0_MODSS_SMS_FW	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_NAVSS0_MODSS_SMS_GLB	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_NAVSS0_MODSS_SMS_ISC	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_NAVSS0_MODSS_SMS_QOS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_NAVSS0_UDMASS_RINGACC0_ISC	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_PBIST0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_PBIST1	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_PLL0_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_PSRAM0_RAM	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_R5FSS0_COMPARE_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_R5FSS0_CORE0_ATCM	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_R5FSS0_CORE0_BTCM	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_R5FSS0_CORE0_DCACHE	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_R5FSS0_CORE0_ECC_AGGR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_R5FSS0_CORE0_ICACHE	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-5. Connectivity Matrix (Part 2) (continued)

Targets	Initiators																						
	PDMA_DEBUG_CCSOC0_MEMR0	PDMA_DEBUG_CCSOC1_MEMR0	PDMA_MCAN0	PDMA_MCASP0	PDMA_SPI_G00	PDMA_SPI_G10	PDMA_UART_G00	PDMA_UART_G10	PDMA_UART_G20	R5FSS0_CORE0	R5FSS1_CORE0	R5FSS2_CORE0	R5FSS0_CORE1	R5FSS1_CORE1	R5FSS2_CORE1	SA2_UL	SPARE_MST_RSVD	UFS0	USB0	VPAC0_DATA_MST0	VPAC0_DATA_MST1	VPAC0_LDC0	VPAC1_DATA_MST0
MCU_R5FSS0_CORE1_ATCM	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_R5FSS0_CORE1_BTCM	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_R5FSS0_CORE1_DCACHE	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_R5FSS0_CORE1_ICACHE	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_R5FSS0_ECC_AGGR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_R5FSS0_SOC_REGION	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_ROM0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_RTI0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_RTI1	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_SA3_UL0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_SA3_UL0_ECC_AGGR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_SEC_MMR0_CFG0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_SEC_MMR0_DBG_CTRL	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_TIMER0_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_TIMER1_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_TIMER2_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_TIMER3_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_TIMER4_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_TIMER5_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_TIMER6_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_TIMER7_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_TIMER8_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_TIMER9_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MCU_TO_MAIN	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-5. Connectivity Matrix (Part 2) (continued)

Targets	Initiators																						
	PDMA_DEBUG_CCSOC0_MEMR0	PDMA_DEBUG_CCSOC1_MEMR0	PDMA_MCAN0	PDMA_MCASP0	PDMA_SPI_G00	PDMA_SPI_G10	PDMA_UART_G00	PDMA_UART_G10	PDMA_UART_G20	R5FSS0_CORE0	R5FSS1_CORE0	R5FSS2_CORE0	R5FSS0_CORE1	R5FSS1_CORE1	R5FSS2_CORE1	SA2_UL	SPARE_MST_RSVD	UFS0	USB0	VPAC0_DATA_MST0	VPAC0_DATA_MST1	VPAC0_LDC0	VPAC1_DATA_MST0
MCU_UART0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MMCSD0_CTL_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MMCSD0_ECC_AGGR_RXMEM	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MMCSD0_ECC_AGGR_TXMEM	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MMCSD0_SS_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MMCSD1_CTL_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MMCSD1_ECC_AGGR_RXMEM	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MMCSD1_ECC_AGGR_TXMEM	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MMCSD1_SS_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MSRAM16KX256E0_ECC_AGGR_REGS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MSRAM16KX256E0_RAM	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
MSRAM16KX256E1_ECC_AGGR_REGS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MSRAM16KX256E1_RAM	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
MSRAM16KX256E2_ECC_AGGR_REGS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
MSRAM16KX256E2_RAM	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
NAVSS0_DDR0_MEM, NAVSS0_DDR1_MEM	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
NAVSS0_DDR0_MEM1,NAVSS0_DDR1_MEM1	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
NAVSS0_MODSS_SMS_FW	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
NAVSS0_MODSS_SMS_GLB	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
NAVSS0_MODSS_SMS_ISC	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
NAVSS0_MODSS_SMS_QOS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
NAVSS0_NBSS_CFG_REGS0_MMRS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
NAVSS0_SRAM0, NAVSS0_SRAM1	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
NAVSS0_UDMASS_SMS_FW	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-5. Connectivity Matrix (Part 2) (continued)

Targets	Initiators																						
	PDMA_DEBUG_CCSOC0_MEMR0	PDMA_DEBUG_CCSOC1_MEMR0	PDMA_MCAN0	PDMA_MCASP0	PDMA_SPI_G00	PDMA_SPI_G10	PDMA_UART_G00	PDMA_UART_G10	PDMA_UART_G20	R5FSS0_CORE0	R5FSS1_CORE0	R5FSS2_CORE0	R5FSS0_CORE1	R5FSS1_CORE1	R5FSS2_CORE1	SA2_UL	SPARE_MST_RSVD	UFS0	USB0	VPAC0_DATA_MST0	VPAC0_DATA_MST1	VPAC0_LDC0	VPAC1_DATA_MST0
NAVSS0_UDMASS_SMS_GLB	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
NAVSS0_UDMASS_SMS_ISC	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
NAVSS0_UDMASS_RINGACC0_ISC	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
NAVSS0_VIRTSS_SMS_FW	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
NAVSS0_VIRTSS_SMS_GLB	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
NAVSS0_VIRTSS_SMS_ISC	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PBIST_MAIN_INFRA0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PBIST_MAIN_INFRA1	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PBIST_MAIN_INFRA2	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PBIST0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PBIST1	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PBIST10	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PBIST11	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PBIST2	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PBIST3	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PBIST4	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PBIST5	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PBIST6	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PBIST7	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PBIST8	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PBIST9	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PCIE0_CORE_CPTS_CFG_CPTS_VBUSP	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PCIE0_CORE_DBN_CFG_PCIE_CORE	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PCIE0_CORE_ECC_AGGR0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-5. Connectivity Matrix (Part 2) (continued)

Targets	Initiators																						
	PDMA_DEBUG_CCSOC0_MEMR0	PDMA_DEBUG_CCSOC1_MEMR0	PDMA_MCAN0	PDMA_MCASP0	PDMA_SPI_G00	PDMA_SPI_G10	PDMA_UART_G00	PDMA_UART_G10	PDMA_UART_G20	R5FSS0_CORE0	R5FSS1_CORE0	R5FSS2_CORE0	R5FSS0_CORE1	R5FSS1_CORE1	R5FSS2_CORE1	SA2_UL	SPARE_MST_RSVD	UFS0	USB0	VPAC0_DATA_MST0	VPAC0_DATA_MST1	VPAC0_LDC0	VPAC1_DATA_MST0
PCIE0_CORE_ECC_AGGR1	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PCIE0_CORE_PCIE_INTD_CFG_INTD_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PCIE0_CORE_USER_CFG_USER_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PCIE0_CORE_VMAP_MMRS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PCIE0_DAT0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
PCIE0_DAT1	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
PCIE1_CORE_CPTS_CFG_CPTS_VBUSP	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PCIE1_CORE_DBN_CFG_PCIE_CORE	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PCIE1_CORE_ECC_AGGR0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PCIE1_CORE_ECC_AGGR1	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PCIE1_CORE_PCIE_INTD_CFG_INTD_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PCIE1_CORE_USER_CFG_USER_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PCIE1_CORE_VMAP_MMRS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PCIE1_DAT0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
PCIE1_DAT1	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
PCIE2_CORE_CPTS_CFG_CPTS_VBUSP	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PCIE2_CORE_DBN_CFG_PCIE_CORE	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PCIE2_CORE_ECC_AGGR0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PCIE2_CORE_ECC_AGGR1	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PCIE2_CORE_PCIE_INTD_CFG_INTD_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PCIE2_CORE_USER_CFG_USER_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PCIE2_CORE_VMAP_MMRS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PCIE2_DAT0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
PCIE2_DAT1	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-5. Connectivity Matrix (Part 2) (continued)

Targets	Initiators																						
	PDMA_DEBUG_CCSOC0_MEMR0	PDMA_DEBUG_CCSOC1_MEMR0	PDMA_MCAN0	PDMA_MCASP0	PDMA_SPI_G00	PDMA_SPI_G10	PDMA_UART_G00	PDMA_UART_G10	PDMA_UART_G20	R5FSS0_CORE0	R5FSS1_CORE0	R5FSS2_CORE0	R5FSS0_CORE1	R5FSS1_CORE1	R5FSS2_CORE1	SA2_UL	SPARE_MST_RSVD	UFS0	USB0	VPAC0_DATA_MST0	VPAC0_DATA_MST1	VPAC0_LDC0	VPAC1_DATA_MST0
PCIE3_CORE_CPTS_CFG_CPTS_VBUSP	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PCIE3_CORE_DBN_CFG_PCIE_CORE	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PCIE3_CORE_ECC_AGGR0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PCIE3_CORE_ECC_AGGR1	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PCIE3_CORE_PCIE_INTD_CFG_INTD_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PCIE3_CORE_USER_CFG_USER_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PCIE3_CORE_VMAP_MMRS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PCIE3_DAT0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
PCIE3_DAT1	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
PDMA_MCAN_ECC_AGGR_REGS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PDMA_SPI_G0_ECC_AGGR_REGS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PDMA_SPI_G1_ECC_AGGR_REGS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PLL0_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PLLCTRL0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PSC0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PSRAM2KECC0_ECC_AGGR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PSRAM2KECC0_RAM	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PSRAMECC0_ECC_AGGR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
PSRAMECC0_RAM	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
R5FSS0_COMPARE_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
R5FSS0_CORE0_ATCM	N	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS0_CORE0_BTCM	N	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS0_CORE0_DCACHE	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
R5FSS0_CORE0_ECC_AGGR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-5. Connectivity Matrix (Part 2) (continued)

Targets	Initiators																						
	PDMA_DEBUG_CCSOC0_MEMR0	PDMA_DEBUG_CCSOC1_MEMR0	PDMA_MCAN0	PDMA_MCASP0	PDMA_SPI_G00	PDMA_SPI_G10	PDMA_UART_G00	PDMA_UART_G10	PDMA_UART_G20	R5FSS0_CORE0	R5FSS1_CORE0	R5FSS2_CORE0	R5FSS0_CORE1	R5FSS1_CORE1	R5FSS2_CORE1	SA2_UL	SPARE_MST_RSVD	UFS0	USB0	VPAC0_DATA_MST0	VPAC0_DATA_MST1	VPAC0_LDC0	VPAC1_DATA_MST0
R5FSS0_CORE0_ICACHE	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
R5FSS0_CORE1_ATCM	N	N	N	N	N	N	N	N	N	Y	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS0_CORE1_BTCM	N	N	N	N	N	N	N	N	N	Y	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS0_CORE1_DCACHE	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
R5FSS0_CORE1_ICACHE	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
R5FSS0_ECC_AGGR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
R5FSS0_EVENT_BUS_REGS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
R5FSS0_SOC_REGION (Debug APB)	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
R5FSS1_COMPARE_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
R5FSS1_CORE0_ATCM	N	N	N	N	N	N	N	N	N	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS1_CORE0_BTCM	N	N	N	N	N	N	N	N	N	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS1_CORE0_DCACHE	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
R5FSS1_CORE0_ECC_AGGR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
R5FSS1_CORE0_ICACHE	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
R5FSS1_CORE1_ATCM	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS1_CORE1_BTCM	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS1_CORE1_DCACHE	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
R5FSS1_CORE1_ICACHE	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
R5FSS1_ECC_AGGR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
R5FSS1_EVENT_BUS_REGS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
R5FSS1_SOC_REGION (Debug APB)	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
R5FSS2_COMPARE_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
R5FSS2_CORE0_ATCM	N	N	N	N	N	N	N	N	N	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS2_CORE0_BTCM	N	N	N	N	N	N	N	N	N	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-5. Connectivity Matrix (Part 2) (continued)

Targets	Initiators																						
	PDMA_DEBUG_CCSOC0_MEMR0	PDMA_DEBUG_CCSOC1_MEMR0	PDMA_MCAN0	PDMA_MCASP0	PDMA_SPI_G00	PDMA_SPI_G10	PDMA_UART_G00	PDMA_UART_G10	PDMA_UART_G20	R5FSS0_CORE0	R5FSS1_CORE0	R5FSS2_CORE0	R5FSS0_CORE1	R5FSS1_CORE1	R5FSS2_CORE1	SA2_UL	SPARE_MST_RSVD	UFS0	USB0	VPAC0_DATA_MST0	VPAC0_DATA_MST1	VPAC0_LDC0	VPAC1_DATA_MST0
R5FSS2_CORE0_DCACHE	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
R5FSS2_CORE0_ECC_AGGR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
R5FSS2_CORE0_ICACHE	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
R5FSS2_CORE1_ATCM	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS2_CORE1_BTCM	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS2_CORE1_DCACHE	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
R5FSS2_CORE1_ICACHE	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
R5FSS2_ECC_AGGR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
R5FSS2_EVENT_BUS_REGS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
R5FSS2_SOC_REGION (Debug APB)	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
R5FSS3_SOC_REGION (Debug APB)	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
R5FSS4_SOC_REGION (Debug APB)	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
R5FSS5_SOC_REGION (Debug APB)	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
RTI0_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
RTI1_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
RTI10_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
RTI11_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
RTI12_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
RTI13_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
RTI14_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
RTI15_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
RTI17_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
RTI18_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
RTI2_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-5. Connectivity Matrix (Part 2) (continued)

Targets	Initiators																						
	PDMA_DEBUG_CCSOC0_MEMR0	PDMA_DEBUG_CCSOC1_MEMR0	PDMA_MCAN0	PDMA_MCASP0	PDMA_SPI_G00	PDMA_SPI_G10	PDMA_UART_G00	PDMA_UART_G10	PDMA_UART_G20	R5FSS0_CORE0	R5FSS1_CORE0	R5FSS2_CORE0	R5FSS0_CORE1	R5FSS1_CORE1	R5FSS2_CORE1	SA2_UL	SPARE_MST_RSVD	UFS0	USB0	VPAC0_DATA_MST0	VPAC0_DATA_MST1	VPAC0_LDC0	VPAC1_DATA_MST0
RTI3_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
RTI4_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
RTI5_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
RTI6_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
RTI7_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
RTI8_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
RTI9_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
SA2_CPSW_PSILSS_MMRS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
SA2_UL0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
SA2_UL0_ECC_AGGR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
SERDES0_10G0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
SERDES0_16G0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
SERDES1_10G0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
SERDES1_16G0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
SERDES2_16G0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
SERDES3_16G0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
SERDES4_10G0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
SPARE_AC_REGION1	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
SPARE_HC_REGION0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
SPARE_RAM_REGION0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
SPI_CPSW_PSILSS_MMRS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
STM0_CXSTM	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
STM0_STIMULUS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
STOG0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-5. Connectivity Matrix (Part 2) (continued)

Targets	Initiators																						
	PDMA_DEBUG_CCSOC0_MEMR0	PDMA_DEBUG_CCSOC1_MEMR0	PDMA_MCAN0	PDMA_MCASP0	PDMA_SPI_G00	PDMA_SPI_G10	PDMA_UART_G00	PDMA_UART_G10	PDMA_UART_G20	R5FSS0_CORE0	R5FSS1_CORE0	R5FSS2_CORE0	R5FSS0_CORE1	R5FSS1_CORE1	R5FSS2_CORE1	SA2_UL	SPARE_MST_RSVD	UFS0	USB0	VPAC0_DATA_MST0	VPAC0_DATA_MST1	VPAC0_LDC0	VPAC1_DATA_MST0
STOG1	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
STOG2	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
STOG3	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
STOG4	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
STOG5	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
STOG6	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
STOG7	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
STOG8	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
STOG9	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
STOG10	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
STOG11	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
STOG12	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
STOG13	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
STOG14	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
STOG15	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
TIMER0_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
TIMER1_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
TIMER2_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
TIMER3_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
TIMER4_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
TIMER5_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
TIMER6_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
TIMER7_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
TIMER8_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-5. Connectivity Matrix (Part 2) (continued)

Targets	Initiators																						
	PDMA_DEBUG_CCSOC0_MEMR0	PDMA_DEBUG_CCSOC1_MEMR0	PDMA_MCAN0	PDMA_MCASP0	PDMA_SPI_G00	PDMA_SPI_G10	PDMA_UART_G00	PDMA_UART_G10	PDMA_UART_G20	R5FSS0_CORE0	R5FSS1_CORE0	R5FSS2_CORE0	R5FSS0_CORE1	R5FSS1_CORE1	R5FSS2_CORE1	SA2_UL	SPARE_MST_RSVD	UFS0	USB0	VPAC0_DATA_MST0	VPAC0_DATA_MST1	VPAC0_LDC0	VPAC1_DATA_MST0
TIMER9_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
TIMER10_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
TIMER11_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
TIMER12_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
TIMER13_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
TIMER14_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
TIMER15_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
TIMER16_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
TIMER17_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
TIMER18_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
TIMER19_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
TIMESYNC_INTRTR0_INTR_ROUTER_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
UART_G00_PSILSS_MMRS	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
UART0	N	N	N	N	N	N	Y	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
UART1	N	N	N	N	N	N	Y	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
UART2	N	N	N	N	N	N	N	Y	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
UART3	N	N	N	N	N	N	N	Y	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
UART4	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
UART5	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
UART6	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
UART7	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
UART8	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
UART9	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
UFS0_HCLK_ECC_AGGR_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-5. Connectivity Matrix (Part 2) (continued)

Targets	Initiators																						
	PDMA_DEBUG_CCSOC0_MEMR0	PDMA_DEBUG_CCSOC1_MEMR0	PDMA_MCAN0	PDMA_MCASP0	PDMA_SPI_G00	PDMA_SPI_G10	PDMA_UART_G00	PDMA_UART_G10	PDMA_UART_G20	R5FSS0_CORE0	R5FSS1_CORE0	R5FSS2_CORE0	R5FSS0_CORE1	R5FSS1_CORE1	R5FSS2_CORE1	SA2_UL	SPARE_MST_RSVD	UFS0	USB0	VPAC0_DATA_MST0	VPAC0_DATA_MST1	VPAC0_LDC0	VPAC1_DATA_MST0
UFS0_IPS_TCLK_ERR_INJ_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
UFS0_P2A_WRAP_CFG_VBP_UFSHCI	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
UFS0_SYSCFG_SS_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
USB0_ECC_AGGR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
USB0_MMR_MMRVBP_USBSS_CMN	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
USB0_PHY2	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
USB0_RAMS_INJ_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
USB0_VBP2APB_WRAP_CONTROLLER_VBP_CORE_ADDR_MAP	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
VPAC0_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	N	N	N	Y
VPAC0_SMS_VPAC_SCRMFW	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
VPAC0_SMS_VPAC_SCRMGLB	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
VPAC0_SMS_VPAC_SCRMISC	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
VPAC0_SMS_VPAC_SCRPFW	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
VPAC0_SMS_VPAC_SCRPGLB	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
VPAC0_KSDW_ECC_AGGR_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
VPAC0_VPAC_TOP_PAC_BASE_MEM_SLV_CBASS_STRIPE_MSRAM_SLV	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	N	N	N	Y
VPAC1_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	N
VPAC1_SMS_VPAC_SCRMFW	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
VPAC1_SMS_VPAC_SCRMGLB	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
VPAC1_SMS_VPAC_SCRMISC	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
VPAC1_SMS_VPAC_SCRPFW	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
VPAC1_SMS_VPAC_SCRPGLB	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
VPAC1_KSDW_ECC_AGGR_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-5. Connectivity Matrix (Part 2) (continued)

Targets	Initiators																						
	PDMA_DEBUG_CCSOC0_MEMR0	PDMA_DEBUG_CCSOC1_MEMR0	PDMA_MCAN0	PDMA_MCASP0	PDMA_SPI_G00	PDMA_SPI_G10	PDMA_UART_G00	PDMA_UART_G10	PDMA_UART_G20	R5FSS0_CORE0	R5FSS1_CORE0	R5FSS2_CORE0	R5FSS0_CORE1	R5FSS1_CORE1	R5FSS2_CORE1	SA2_UL	SPARE_MST_RSVD	UFS0	USB0	VPAC0_DATA_MST0	VPAC0_DATA_MST1	VPAC0_LDC0	VPAC1_DATA_MST0
VPAC1_VPAC_TOP_PAC_BASE_MEM_SLV_CBASS_STRIPE_MSRAM_SLV	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	N
VUSR0_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
VUSR0_CTL_ECC	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
VUSR0_PORTAL	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
VUSR1_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
VUSR1_CTL_ECC	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
VUSR1_PORTAL	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
WAVE521CL0/1_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
WKUP_CBASS_FW0_ERR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
WKUP_CBASS0_ERR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
WKUP_CBASS0_FW	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
WKUP_CBASS0_GLB	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
WKUP_CTRL_MMR0_CFG0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
WKUP_DDPA0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
WKUP_SMS0_ECC_AGGR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
WKUP_SMS0_IRAM	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
WKUP_ECC_AGGR0_ECC_AGGR	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
WKUP_ESM0_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
WKUP_GPIO0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
WKUP_GPIO1	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
WKUP_GPIOMUX_INTRTR0_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
WKUP_I2C0_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
WKUP_PLLCTRL0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y

Copyright © 2024 Texas Instruments Incorporated

Table 3-5. Connectivity Matrix (Part 2) (continued)

Targets	Initiators																						
	PDMA_DEBUG_CCSOC0_MEMR0	PDMA_DEBUG_CCSOC1_MEMR0	PDMA_MCAN0	PDMA_MCASP0	PDMA_SPI_G00	PDMA_SPI_G10	PDMA_UART_G00	PDMA_UART_G10	PDMA_UART_G20	R5FSS0_CORE0	R5FSS1_CORE0	R5FSS2_CORE0	R5FSS0_CORE1	R5FSS1_CORE1	R5FSS2_CORE1	SA2_UL	SPARE_MST_RSVD	UFS0	USB0	VPAC0_DATA_MST0	VPAC0_DATA_MST1	VPAC0_LDC0	VPAC1_DATA_MST0
WKUP_PSC0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
WKUP_UART0	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
WKUP_VTM0_ECCAGGR_CFG	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
WKUP_VTM0_MMR_VBUSP_CFG1	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y
WKUP_VTM0_MMR_VBUSP_CFG2	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	Y	Y	Y

Table 3-6. Connectivity Matrix (Part 3)

	VPAC1_DATA_MST1	VPAC1_LDC0	VUSR0	VUSR1	WAVE521CL	WKUP_SMS_FWMGR	WKUP_SMS0	GPU0_XPU
ATL0	Y	Y	Y	Y	N	N	Y	N
BWLIMITER_AC_SPARE1	Y	Y	Y	Y	N	N	Y	N
BWLIMITER_AC_SPARE2	Y	Y	Y	Y	N	N	Y	N
BWLIMITER_AC_SPARE3	Y	Y	Y	Y	N	N	Y	N
BWLIMITER_AC_SPARE4	Y	Y	Y	Y	N	N	Y	N
BWLIMITER_AC_SPARE5	Y	Y	Y	Y	N	N	Y	N
BWLIMITER_GPU0_M0_WR	Y	Y	Y	Y	N	N	Y	N
BWLIMITER_GPU0_M1_RD	Y	Y	Y	Y	N	N	Y	N
BWLIMITER_GPU0_M1_WR	Y	Y	Y	Y	N	N	Y	N
BWLIMITER_HC_SPARE1	Y	Y	Y	Y	N	N	Y	N
BWLIMITER_HC_SPARE2	Y	Y	Y	Y	N	N	Y	N
BWLIMITER_HC_SPARE3	Y	Y	Y	Y	N	N	Y	N
BWLIMITER_ WAVE521CL0/1_WR	Y	Y	Y	Y	N	N	Y	N
BWLIMITER_AC_SPARE0	Y	Y	Y	Y	N	N	Y	N
BWLIMITER_ENCODER0_RD	Y	Y	Y	Y	N	N	Y	N
BWLIMITER_ENCODER0_WR	Y	Y	Y	Y	N	N	Y	N
BWLIMITER_GPU0_M0_RD	Y	Y	Y	Y	N	N	Y	N

Copyright © 2024 Texas Instruments Incorporated

Table 3-6. Connectivity Matrix (Part 3) (continued)

	VPAC1_DATA_MST1	VPAC1_LDC0	VUSR0	VUSR1	WAVE521CL	WKUP_SMS_FWMGR	WKUP_SMS0	GPU0_XPU
BWLIMITER_HC_SPARE0	Y	Y	Y	Y	N	N	Y	N
BWLIMITER_WAVE521CL0/1_RD	Y	Y	Y	Y	N	N	Y	N
CBASS_SPARE1_GLB	Y	Y	Y	Y	N	Y	Y	N
CBASS_SPARE1_QOS	Y	Y	Y	Y	N	Y	Y	N
CBASS_SPARE3_GLB	Y	Y	Y	Y	N	Y	Y	N
CBASS_AC_NONSAFE0_ERR	Y	Y	Y	Y	N	Y	Y	N
CBASS_AC_CFG_NONSAFE0_ERR	Y	Y	Y	Y	N	Y	Y	N
CBASS_AC_CFG_NONSAFE0_FW	Y	Y	Y	Y	N	Y	Y	N
CBASS_AC_CFG_NONSAFE0_GLB	Y	Y	Y	Y	N	Y	Y	N
CBASS_AC_CFG0_ERR	Y	Y	Y	Y	N	Y	Y	N
CBASS_AC_CFG0_FW	Y	Y	Y	Y	N	Y	Y	N
CBASS_AC_CFG0_GLB	Y	Y	Y	Y	N	Y	Y	N

Copyright © 2024 Texas Instruments Incorporated

4 Initialization

This chapter describes the steps for non-secure device initialization.

4.1 Initialization Overview.....166
4.2 Boot Process.....169
4.3 Boot Mode Pins.....174
4.4 Boot Parameter Tables.....201
4.5 Boot Image Format.....210
4.6 Boot Modes.....217
4.7 Boot Memory Maps.....222

Copyright © 2024 Texas Instruments Incorporated

4.1 Initialization Overview

Figure 4-1 is an overview of the initialization process and its steps:

- **Preinitialization:** Power, clock, and control connections must be present, and the boot configuration pins must be held at the desired logical levels.
- **Power, clock, reset ramp sequence:** Specific sequence that is applied by the power-management chip(s)
- **ROM code:** Responsible for finding, for downloading, and for executing the initial software (SBL)
- **Initial software:** Software that loads, prepares, and passes control to application software or to the high-level operating system (HLOS)
- **High-Level Operating System** or bare-metal application which runs on main processor(s)

Figure 4-1. Initialization Process

The first two steps in the initialization process are hardware-oriented; however, they require an understanding of the process of configuring these system interface pins (balls on the device), which have software-configurable functionality. This configuration is an essential part of the chip configuration and is application-dependent. This chapter discusses these system-interface pins, the associated configuration registers, and memory structures that are vital to the correct initialization of the device.

4.1.1 ROM Code Overview

ROM bootloader (or ROM Code) is a software that resides in a on-chip read-only memory (ROM) to assist the customer in transferring and executing their application code. The device has two ROM codes operating in tandem – the MCU ROM code, and the SMS ROM code.

In order to accommodate various system scenarios, the ROM Code supports several boot modes. These boot modes can be broadly classified as:

- Host boot modes
- Memory boot modes.

During a host boot, the device is configured to receive code from a host via the selected interface. Either the host writes the application code directly into internal memory or the ROM Code receives the application code on the selected interface and stores it in internal memory.

During a memory boot, the device transfers code from non-volatile memory to internal memory for execution.

In all boot modes, the entire boot operation can be partitioned into two sections:

1. Hardware initialization phase
2. Boot process.

During initialization, the ROM Code configures the device resources (PLLs, peripherals, pins) as needed to support the boot process. The resources used depend on the boot mode requirements.

During the boot process the boot image can be loaded into device memory and executed, or executed in place, depending on the boot peripheral. SMS will perform code verification and allow, or forbid, the image execution.

Main configuration source for boot after power-up are the BOOTMODE pins sampled automatically after reset release and stored in device status registers. At ROM Code startup, these pin values are read from the registers to create the boot peripheral list and the boot configuration tables used later to initialize and startup the PLLs and boot peripherals.

The ROM Code also provides a multi-stage boot mechanism.

4.1.2 Bootloader Modes

Table 4-1 shows the boot modes supported by ROM code.

Copyright © 2024 Texas Instruments Incorporated

Table 4-1. ROM Code Boot Modes

Boot Mode	Boot Media/Host	SoC Peripheral	Peripheral's Domain[1]	Can be a Backup Mode?[2]	Notes
No-boot/Dev-boot	No media or host	None	N/A	N	No boot or development boot – debug modes
Serial NAND	Serial NAND flash	MCU_FSS0_OSPI0	MCU	N	On OSPI port
OSPI	OSPI flash	MCU_FSS0_OSPI0	MCU	N	On OSPI port
QSPI	QSPI flash/EEPROM	MCU_FSS0_OSPI0	MCU	N	On OSPI port
SPI	SPI EEPROM	MCU_FSS0_OSPI0	MCU	Y[3]	On OSPI port
Hyperflash	Hyperflash NOR flash	MCU_FSS0_HPB0	MCU	N	-
Ethernet	External host	MCU_CPSW0	MCU	Y[3]	In BOOTP mode. RGMII or RMII PHY
I2C	I^2C EEPROM	MCU_I2C0	MCU	Y	I2C slave boot is not supported
UART	External host	MCU_UART0 or WKUP_UART0	MCU	Y	XMODEM protocol
MMCSD	MMC/SD card	MMCSD0 or MMCSD1	MAIN	Y[3]	Boot from User Data Area (UDA) or file system
eMMC	eMMC flash	MMCSD0 or MMCSD1	MAIN	N	Boot from boot partition with auto-fall back to file system
USB	External host	USB0	MAIN	Y[3]	USB device mode boot using DFU (device firmware upgrade). Boot from USB flash drive is not supported. Boot is running on USB2.0 speeds.
PCIe	External host	PCIE1	MAIN	N	-
xSPI	xSPI flash	MCU_FSS0_OSPI0	MCU	N	On OSPI port

(1) Peripherals in the MCU domain are available for boot even when main domain is not powered.
(2) The peripheral can be selected also as a backup boot mode. A backup mode is tryed if primary boot mode fails.
(3) When in normal boot flow (MCU Only = 0)

Note

Because different devices support different sets of peripherals, see the *device-specific Datasheet* to obtain the list of peripherals supported in your device.

4.1.3 Terminology

- **Boot Mode Pins:** Boot mode pins provide vital information to ROM code for boot. These pins must be properly set up before power ramp.
- **Bootstrap:** Initial software launched by the ROM code during the memory booting phase.
- **Boot Header:** Optional structure that precedes the initial software and allows the redefinition of the ROM code default settings.
- **Downloaded software:** Initial software downloaded into the internal static RAM (SRAM) by the ROM code during the peripheral booting phase.
- **eFuse:** A one-time programmable memory location usually set at the factory.
- **Flash loader:** Downloaded software launched by the ROM code during the preflashing stage. It also programs an image in external memories.
- **GP device:** General-purpose device (SoC) or a non-secure device.
- **Initial software:** Software executed by any of the ROM code mechanisms (memory booting or peripheral booting). Initial software is a generic term for bootstrap and downloaded software. This can be the SBL (secondary bootloader) responsible for loading an OS.
- **Memory booting:** ROM code mechanism that consists of executing initial software from external memory.
- **Master CPU:** The Arm® Cortex® CPU for which CPU-ID is 0. It configures the multicore platform and starts the ROM code to ensure device booting from a mass storage memory (memory booting) or a peripheral interface (peripheral booting).

Copyright © 2024 Texas Instruments Incorporated

- **Peripheral booting:** ROM code mechanism that consists of polling selected interfaces, downloading, and executing initial software (in this case, downloaded software) in the internal RAM.
- **Preflashing:** A specific case of peripheral booting where the ROM code mechanism is used to program the external flash memory.
- **ROM Code:** or ROM bootloader (RBL), the on-chip software in device ROM that executes first and implements booting.
- **ROM Code-controlled Boot Phase:** This phase covers the sequence operations from the time the platform releases the reset to the time first user- or customer-owned software starts execution. This phase is fully controlled by the device ROM code.
- **OCMC RAM memory:** On-chip RAM memory used by ROM code during boot and also for loading the booting image. ROM code is using MCU_MSRAM0
- **Booting Parameter Table:** A logical structure stored in the on-chip RAM memory and contains information for the boot, such as the boot file name or an address to boot from.

Copyright © 2024 Texas Instruments Incorporated

4.2 Boot Process

4.2.1 MCU ROM Code Architecture

The MCU ROM code has the following components (see also Figure 4-2):

- Main
- Buffer manager
- X.509
- Log and Trace
- System
- Protocol
- Driver

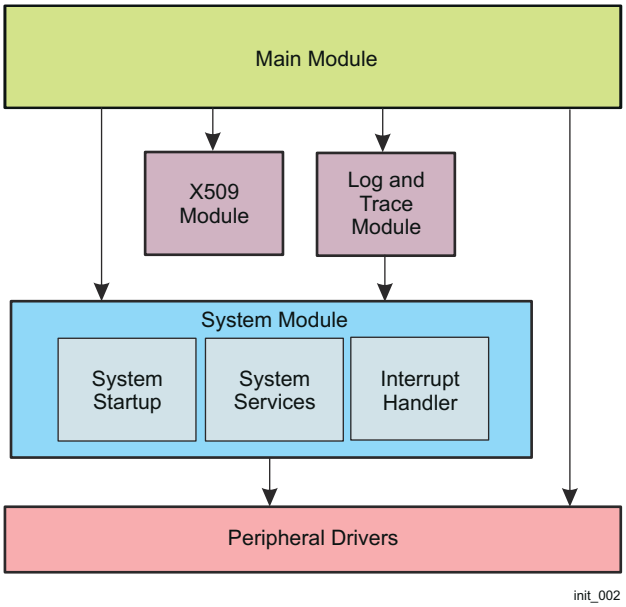


Figure 4-2. MCU ROM Code Architecture

4.2.1.1 Main Module

The Main module contains the top level execution loop. This loop repeats until a boot image has been received or directed to sleep by the SMS. The main loop has three different execution sub-paths based on the boot peripheral.

- **Image Path** This path is used by the USB-DFU, PCIe, OSPI, Hyperflash, and QSPI boot modes. In these cases the image data can be directly read by both, MCU and SMS, in place.
- **Block Path** This path is used by the SPI, I2C, UART, eMMC, Ethernet, Serial NAND, and MMC/SD cards in raw mode. In this mode data is received from the peripheral in blocks. Blocks are accumulated in the boot buffer until a full X.509 certificate header has been received, at which point this full certificate and any subsequent blocks are passed to the SMS as they arrive.
- **Filesystem Path** This path is used by the MMC/SD cards in filesystem mode. This mode executes exactly like in the block path, except that the boot image location is defined by a filesystem.

The main level is able to detect if a received boot image is in the correct format and reject non-conforming images.

4.2.1.2 X509 Module

The X509 module parses the boot header. The boot header is an X.509 certificate as defined in RFC5280. Extensions specific to boot are described in Section 4.5.2, *X.509 Certificate*.

This module also includes an OID decoder as well as defines OID values as C #define constants for any values that can be used during boot.

Copyright © 2024 Texas Instruments Incorporated

4.2.1.3 Buffer Manager Module

The buffer manager module is used to allocate buffers to hold boot data. The MCU code allocates a buffer when it is ready to read a block of data from the boot peripheral. Once the data is read the buffer ownership is passed to the SMS. When the SMS has completed processing the data in the buffer ownership is returned to the MCU, which then frees the buffer.

4.2.1.4 Log and Trace Module

The Log and Trace modules are not integral to the boot process. Instead, they provide ability to record operation of the ROM code and track unexpected occurrences.

Log and trace function is currently TI internal.

4.2.1.5 System Module

The system module provides services to other modules. These services are not directly related to boot drivers. The system module is the only module that operates in the supervisor mode of the MCU (a few services will run in user). The system module supports the following functions:

- Interrupt enable, disable, and service
- IPC
- Power
- Pinmux
- Clock
- Task switch

The main level is able to detect if a received boot image is in the correct format and reject non-conforming images.

4.2.1.6 Protocol Module

The protocol modules provide implementation of high-level data transfer protocols. The BOOTP/TFTP and XMODEM protocols reside in this layer. These modules provide services as defined in well-known standards.

4.2.1.7 Driver Module

The driver module implements the low level peripheral drivers.

4.2.2 SMS ROM Description

In a general-purpose (GP) device, SMS ROM performs the following functions:

- Device management
- Configures the boot vectors (in BOOT_CFG) and controls reset release of MCU core. That is, SMS is the boot master of MCU core.
- IPC configuration via MCU_NAVSS rings and Secure Proxy
- PLL configuration (MCU subsystem and SA2UL)
- X509 certificate parsing
- SA2UL configuration to SHA512 for image integrity checks
- Test flow support – Wait-In-Reset commands
- SMS firmware loading

4.2.3 Boot Process Flow

The MCU boot process flow is shown in Figure 4-3.

Copyright © 2024 Texas Instruments Incorporated

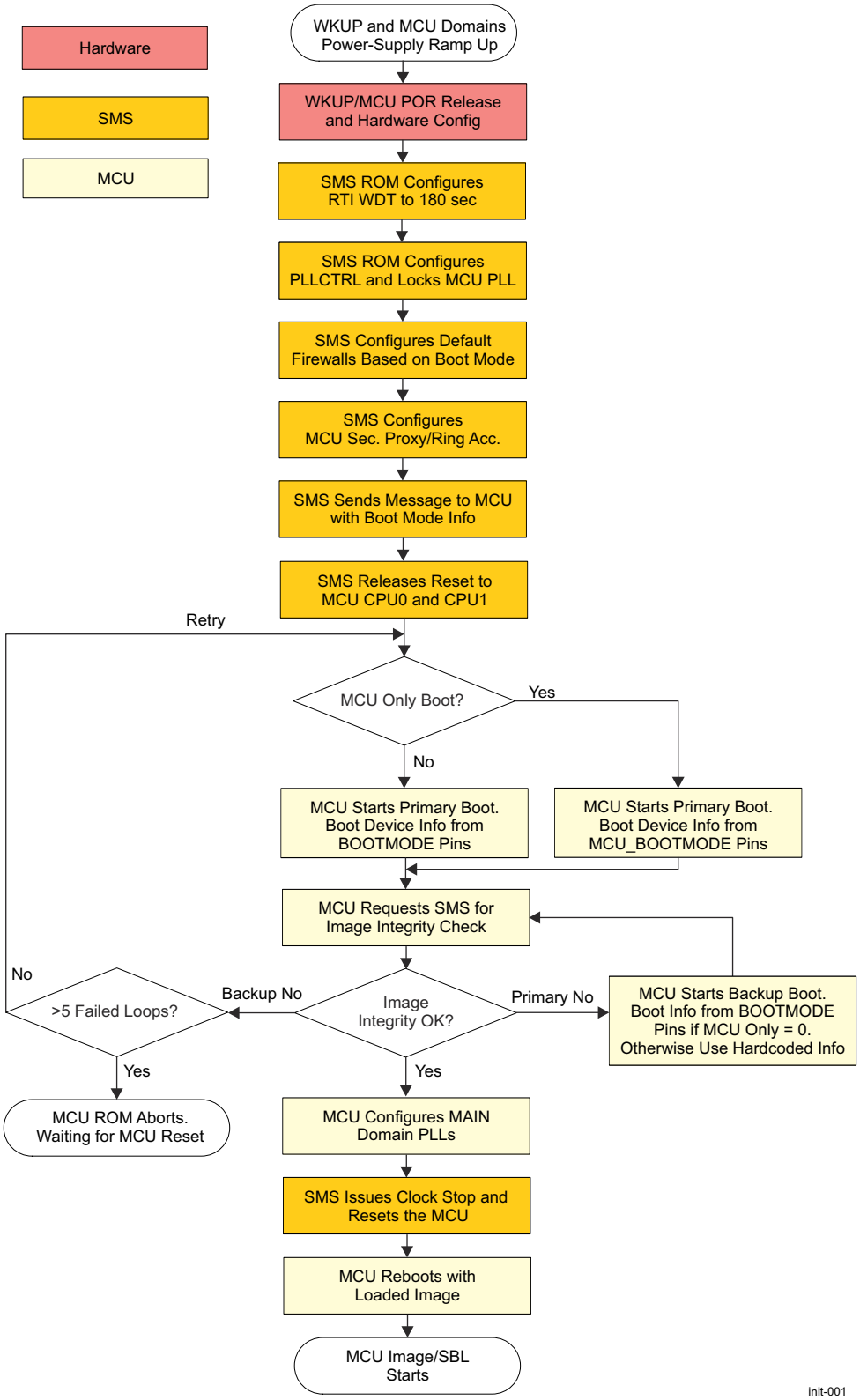


Figure 4-3. Boot Process

The values of (MCU_)BOOTMODE pins are latched into the Device Status register (CTRLMMR_WKUP_DEVSTAT) by hardware as the device comes out of global cold reset. When MCU Only boot was selected, only MCU_BOOTMODE pins are taken into account and only MCUSS set of peripherals are

Copyright © 2024 Texas Instruments Incorporated

available for boot. For more information how to set BOOTMODE and MCU_BOOTMODE pins, see Section 4.3, *Boot Mode Pins*.

The SMS is the boot master of MCU. SMS performs the necessary configurations and releases MCU's reset for CPU0 and CPU1 simultaneously even when in split mode.

Note

SMS releases MCU's reset for CPU0 and CPU1 simultaneously regardless if MCU CPUs are in lockstep or split mode.

The MCU checks the boot mode pins and then configures the apropriate peripheral interface to get access to a boot image. A cursory check of the image is made, and the image is passed to SMS. SMS ROM then will perform code verification and route the boot image to the on-chip RAM. Once the image has been received, MCU enters a clean state and idles. SMS ROM code will assert reset to the MCU, redirect the boot vector to the newly loaded image, and release the reset. This restarts the MCU with the MCU ROM code fully disconnected.

The MCU ROM code executes only on initial power up (POR). On subsequent (warm) resets, the reset vector base address will point to run-time loaded code (the SBL), and not ROM.

Note

SMS ROM sets up a 3-minute watchdog timer (MCU_RTI0) timeout. During this time, the MCU boot needs to get completed, otherwise a WDT reset will occur. Once the MCU image is loaded (SBL/SPL), SMS ROM will restart the watchdog timer for additional 3 minutes upon entering the MCU SBL. The customer-provided MCU image needs to load and install the TI-provided SYSFW image into the SMS, which will manage the watchdog timer during run time.

Note

The following system conditions must be met at POR to perform device boot:

- USB cable plug must be inserted
- Ethernet PHY is powered up and out of reset
- The SD card cage must be powered before entering the SD card boot mode. A SD/MMC card with pre-loaded image must be inserted
- Memory devices must be up and ready (power ramped up and reset completed) at device startup:
  - eMMC
  - OSPI/QSPI/xSPI flash
  - SPI or I^2C EEPROM
  - Hyperflash
  - Serial NAND

Failing to meet these requirements may result in boot fail and performing a backup boot (if available for that mode).

Figure 4-4 describes the external bootloader (SBL) typical tasks.

Copyright © 2024 Texas Instruments Incorporated

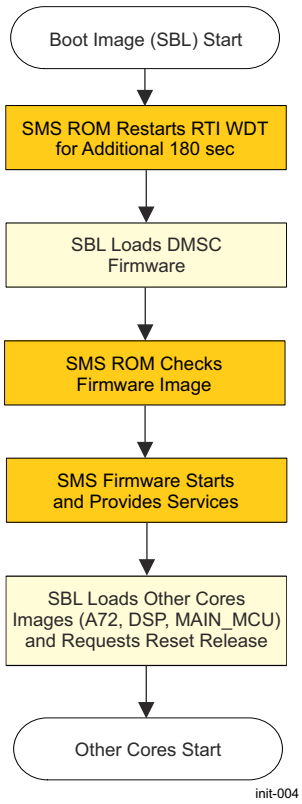


Figure 4-4. External Bootloader Tasks

Upon MCU reset and SBL execution start, SMS ROM restarts the RTI watchdog timer for additional 180 seconds of timeout. During that time, SBL must load the SMS firmware provided by TI otherwise a MCU reset will occur as a preventive measure against software misbehavior.

One of the SBL's main tasks is to load the SMS firmware. Only after this task is performed, SBL can load the other processor (A72s, R5s) image and request a reset release from SMS firmware for those cores.

4.2.4 MCU Only vs Normal Boot

Users can select a special (MCU only) boot flow, which can boot the MCU, even if the main domain is unpowered. The MCU-only boot differences versus the normal boot are described below:

Normal Boot — Requires minimum of:

- VDDS*_MCU
- VDD_MCU
- and VDD_CORE, VDD_CPU

voltage rails ON. Other voltage rails are optional.

Power domains for WKUP, MCUSS and Core domain are ON. Power domains required for the specified boot mode may be turned-on by the boot ROM code, if they are OFF by default.

MCU-Only Boot — Requires minimum of:

- VDDS*_MCU
- VDD_MCU

voltage rails ON. Other voltage rails are optional. Power domains for WKUP and MCUSS are ON (only R5 is switchable). Core domain is unknown (may be powered or unpowered). Only limited set of boot modes involving MCUSS peripherals are supported. Power domains required for the specified boot mode may be turned-on by the boot ROM code, if they are OFF by default

Copyright © 2024 Texas Instruments Incorporated

4.3 Boot Mode Pins

Boot Mode pins provide means to select the boot mode and options before the device is powered up. After every POR, they are the main source to populate the Boot Parameter Tables. See Section 4.4, *Boot Parameter Tables* for table list and description.

Boot mode pins can be divided into the following categories:

- **MCU_BOOTMODE[02:00]** – Denote system clock frequency (WKUP_HFOSC0) to ROM code for PLL configuration.
- **MCU_BOOTMODE[05:03]** – Select the requested boot (primary) mode after POR, that is, the peripheral/memory to boot from. These are the only pins which direct boot on MCU Only boot mode
- **MCU_BOOTMODE06** – MCU Only boot mode. In MCU Only boot, ROM code skips BOOTMODE pins and attempts boot only from MCU peripherals
- **MCU_BOOTMODE[09:08]** – These pins can select tests to be performed on power-up (POST). POST runs in hardware, before the ROM code starts
- **BOOTMODE0** – This pin allows for additional primary boot modes to be selected when in non-MCU Only (normal) mode
- **BOOTMODE[3:1]** – Select the backup boot mode, that is, the peripheral/memory to boot from, if primary boot device failed.
- **BOOTMODE[6:4]** – These pins provide optional configurations for primary boot and are used in conjuction with the boot mode selected. See Section 4.3.2.3 and the corresponding boot mode section.
- **BOOTMODE7** – This pin provides optional configurations for the backup boot devices. See Section 4.3.2.4 and the corresponding boot mode section.

Note

It is user's responsibilty to set the boot mode pins (via pullups or pulldowns, and jumpers/switches) depending on the desired boot scenario.

System board must provide means to switch bootmode pin settings easily. In a field design, some pins may be hardcoded to 0 or 1 only after a careful evaluation (for example, if using MCU Only pin = 1).

Note

CTRLMMR_WKUP_DEVSTAT[23:16] register reflects the BOOTMODE pin values sampled after MCU_POR release.

CTRLMMR_WKUP_DEVSTAT[9:0] register reflects the MCU_BOOTMODE pin values sampled after MCU_POR release.

Copyright © 2024 Texas Instruments Incorporated

4.3.1 MCU_BOOTMODE Pin Mapping

The MCU_BOOTMODE pins are sampled during both, MCU-Only and normal boot. The MCU_BOOTMODE pins are the only pins sampled during MCU-only boot mode, that is, BOOTMODE pins are not considered. MCU_BOOTMODE pin mapping is shown in Table 4-2.

Table 4-2. MCU_BOOTMODE Pin Mapping

MCU 9	MCU 8	MCU 7	MCU 6	MCU 5	MCU 4	MCU 3	MCU 2	MCU 1	MCU 0
POST Config[1]		OVRD[2]	MCU Only	Primary Boot Mode A			PLL Config		

(1) POST function is not handled by ROM Code. It is executed in hardware before ROM Code starts.
(2) Pull MCU 7 to 0 on GP devices.

Table 4-3 describes the MCU_BOOTMODE pins that need to be set according to the system clock provided to the device.

The ROM Code will configure any PLLs required during the boot process. The ROM Code does not have the ability to select HFOSC1 (in main domain) during initial boot, however the selection can be done through the boot certificate (see Section 4.5, *Boot Image Format*).

Table 4-3. PLL Reference Clock Selection

PLL Config Pins			Ref Clock (MHz)
MCU 2	MCU 1	MCU 0	
0	0	0	19.2
0	0	1	20
0	1	0	24
0	1	1	25
1	0	0	26
1	0	1	27
1	1	0	Reserved
1	1	1	No PLL setup[1]

(1) Only no-boot, Hyperflash, and OSPI modes are expected to function with PLL config = 7 (No PLL setup)

If MCU Only mode (boot from MCU domain only) needs to be selected, the MCU Only pin must be set to 1 (see Table 4-4).

Table 4-4. MCU Only Selection

MCU Only Pin	MCU Only Selection
MCU 6	
0	Normal boot mode. Both MCU_BOOTMODE and BOOTMODE pins are read by ROM code
1	MCU Only boot mode. Boot can proceed only from MCU peripherals

Power-On Self-Test sequence is executed as directed by pins shown in Table 4-5. Dedicated eFuse bits may override the POST pins. See register CTRLMMR_WKUP_POST_OPT in section *WKUP_CTRL_MMR0 Registers* and POST section in the Safety Manual for details.

Table 4-5. POST Selection

POST Config Pins		POST Sequence
MCU 9	MCU 8	
0	0	DMSC LBIST followed by MCU LBIST followed by PBIST[2] [3]
0	1	DMSC LBIST and MCU LBIST in parallel followed by PBIST[2]
1	0	DMSC LBISTReserved[2]
1	1	POST bypass[1]

(1) eFuse may has been programmed so that these pins are overriden and a POST sequence is always executed (no bypass possible).
(2) This sequence may has been replaced by another via eFuse settings. See register CTRLMMR_WKUP_POST_OPT in section *WKUP_CTRL_MMR0 Registers* and POST section in Safety Manual, for details.

Copyright © 2024 Texas Instruments Incorporated

(3) To determine if device under test supports LBIST POST, Register CTRLMMR_WKUP_POST_CFG0 (0x4300_C2C4); bit 31, which defines whether the PLL for FAST POST is enabled, can be read:
- [31] = 0, LBIST POST Timeout is expected on the device under test
- [31] = 1, LBIST POST is expected to pass

Table 4-6 lists the boot modes (both primary and backup), available when MCU Only boot was selected. For normal boot modes, refer to Table 4-8 and Table 4-9

Table 4-6. Boot Mode Selection When MCU Only (MCU 6) = 1

Primary Boot Mode A Pins			Primary Boot Mode Selected	Backup Boot Mode Assigned[1]
MCU 5	MCU 4	MCU 3		
0	0	0	Hyperflash	UART
0	0	1	OSPI	UART
0	1	0	QSPI	UART
0	1	1	xSPI	UART
1	0	0	Ethernet RGMII	I2C
1	0	1	Serial NAND	I2C
1	1	0	I2C	UART
1	1	1	UART	I2C

(1) When MCU Only = 1, backup mode is assigned per primary mode. In general, non-flash primary boot modes are backed up with flash modes, and flash primary modes use non-flash for backup.

Copyright © 2024 Texas Instruments Incorporated

4.3.2 BOOTMODE Pin Mapping

In normal boot operation (MCU Only = 0), the ROM execution is directed also through the main boot mode pins. This provides more flexibility and more booting peripherals to boot from. The Main domain must be powered and functional when MCU Only = 0.

Main boot mode pins are shown in Table 4-7.

Table 4-7. BOOTMODE Pin Mapping

7	6	5	4	3	2	1	0
Backup Boot Mode Config	Primary Boot Mode Config			Backup Boot Mode			Primary Boot Mode B

4.3.2.1 Primary Boot Mode Selection

The primary boot mode is the first mode attempted after reset. Table 4-8 lists all possible primary boot modes. Boot modes with Boot Mode B = 1 are available during normal boot only.

Table 4-8. Primary Boot Mode Selection When MCU Only (MCU 6) = 0

Primary Boot Mode B Pin	Primary Boot Mode A Pins			Boot Mode Selected
0	MCU 5	MCU 4	MCU 3	
0	0	0	0	Hyperflash
0	0	0	1	OSPI
0	0	1	0	QSPI
0	0	1	1	xSPI
0	1	0	0	Ethernet RGMII
0	1	0	1	Serial NAND
0	1	1	0	I2C
0	1	1	1	UART
1	0	0	0	MMC/SD card
1	0	0	1	eMMC
1	0	1	0	USB
1	1	0	0	Ethernet RMII
1	1	0	1	PCIe
1	1	1	0	SPI
1	1	1	1	No-boot/Dev boot

4.3.2.2 Backup Boot Mode Selection When MCU Only = 0

With MCU Only = 0, the backup boot mode is selected via pins within the main BOOTMODE map. Table 4-9 lists all possible backup boot modes when MCU Only = 0. For backup boot modes when MCU Only = 1, please refer to Table 4-6.

Table 4-9. Backup Mode Selection When MCU Only (MCU 6) = 0

Backup Boot Mode Pins			Backup Boot Mode Selected
3	2	1	
0	0	0	None (no backup boot will be attempted)
0	0	1	USB – Device (DFU)
0	1	0	Reserved
0	1	1	UART
1	0	0	Ethernet
1	0	1	MMC/SD card
1	1	0	SPI
1	1	1	I2C

Copyright © 2024 Texas Instruments Incorporated

4.3.2.3 Primary Boot Mode Configuration

When in normal boot flow (MCU Only = 0), it is possible to modify certain peripheral settings, such as chip-select, bus speed, and others, depending on the peripheral selected.

The mapping of configuration pins per boot mode is shown in Table 4-10. See the respective sections for details.

Table 4-10. Primary Boot Mode Configuration (MCU 6 = 0)

Primary Boot Mode Config Pins[(1)]			Primary Boot Mode B Pin	Primary Boot Mode A Pins			Primary Boot Mode
6	5	4	0	MCU 5	MCU 4	MCU 3	
Speed			0	0	0	0	Hyperflash
Speed		Csel	0	0	0	1	OSPI
Port		Csel	0	0	1	0	QSPI
SFDP	Read Cmd	Mode	0	0	1	1	xSPI
	Delay	Link stat	0	1	0	0	Ethernet RGMII
Reserved	Read Mode 2	Read Mode 1	0	1	0	1	Serial NAND
Bus reset	Reserved	Addr	0	1	1	0	I2C
		Port	0	1	1	1	UART
Port	Bus width	Fs/raw	1	0	0	0	MMC/SD card
Port	Bus width	Voltage	1	0	0	1	eMMC
Reserved	Mode	Lane Swap	1	0	1	0	USB
Clkout	Clk src	Reserved	1	1	0	0	Ethernet RMII
Reserved	Ssc	Clocking	1	1	0	1	PCIe
Port	Mode	Csel	1	1	1	0	SPI
Split	Arm/Thumb	No/Dev	1	1	1	1	No-boot/Dev boot

(1) Shaded cells are reserved pins for that boot mode.

4.3.2.4 Backup Boot Mode Configuration

When in normal boot flow (MCU Only = 0), it is possible to modify certain peripheral settings, such as chip-select, bus speed, and others depending on the peripheral selected.

The mapping of the backup configuration pin per boot mode selected is shown in Table 4-11. See the respective sections for details.

Table 4-11. Backup Boot Mode Configuration (MCU 6 = 0)

Backup Boot Mode Config Pin[(1)]	Backup Boot Mode Pins			Backup Boot Mode
7	3	2	1	
	0	0	0	None (no backup boot will be attempted)
Reserved	0	0	1	USB – Device (DFU)
Port	0	1	0	Reserved
Port	0	1	1	UART
I/F	1	0	0	Ethernet
Port	1	0	1	MMC/SD card
Reserved	1	1	0	SPI
Reserved	1	1	1	I2C

(1) Shaded cells are don't cares and pins can take any value.

Copyright © 2024 Texas Instruments Incorporated

4.3.3 No-boot/Dev-boot Configuration

Table 4-13 shows configuration pins assignment to functions when No-boot/Dev-boot was selected. In this mode, ROM code is bypassed and user is allowed to run his own code for development or debug.

MCU-only mode is not compatible with no-boot/dev-boot since the DebugSS is in the main domain.

Table 4-12. No-boot/Dev-boot Configuration Fields

BOOTMODE Pins	Field	Value	Description	MCU Only=1 Value
6	Split	0	Split/Lockstep configuration of MCU R5 cores derived from efuse	N/A
		1	MCU R5 cores forced to split mode	
5	Arm/Thumb	0	ARM mode reset vectors	N/A
		1	Thumb mode reset vectors	
4	No/Dev	0	Development Boot	N/A
		1	No boot	

During the *Development boot (BOOTMODE[4] = 0)*, the SMS ROM code will act as if boot of the primary image has completed, and the SMS ROM then will be waiting for a firmware load message from MCU R5. Thus user you can load a standard u-boot/SPL image to the R5 RAM. U-boot/SPL will then load the SMS firmware and complete the full boot.

In *No-boot (BOOTMODE[4] = 1)*, both the SMS and MCU R5 ROMs are bypassed and both CPUs are held in a dummy branch-to-self loop. No-boot is the most minimal device touch state by the ROM - only minimal hardware configurations are done and none of the PLLs is locked/configured. No-boot is suitable if user wants to load his own PLL, Pad config, and other basic settings.

Copyright © 2024 Texas Instruments Incorporated

4.3.4 Hyperflash Boot Device Configuration

The following boot mode pin configuration and corresponding pin usage and mux configuration are shown below. This is the hyperflash single chip select configuration. It can be selected with MCU only active. If MCU only is not active, the main pins must be configured as shown. This is the only configuration that is supported on Hyperflash boot mode.

Table 4-13 shows configuration pins assignment to functions when boot mode is the Hyperflash mode.

Table 4-13. Hyperflash Configuration Fields

BOOTMODE Pins	Field	Value	Description	MCU Only=1 Value
6	Speed	0	83 MHz DDR speed	0
		1	166 MHz DDR speed	
5	-	-	Don't care pin	-
4	-	-	Don't care pin	-

Table 4-14 summarizes the HyperBus pin configuration done by ROM code.

Table 4-14. HyperBus Pin Usage

Device Pin	Module Signal	Pull Enable	Pull Direction	Driver Index	Rx En/Dis	Pinmux Sel
MCU_OSPI0_CLK0	MCU_HYPERBUS0_CK	No	Up	0	Disable	1
MCU_OSPI0_LBCLK0	MCU_HYPERBUS0_CKn	No	Up	0	Disable	1
MCU_OSPI0_DQS	MCU_HYPERBUS0_RWDS	Yes	Up	0	Enable	1
MCU_OSPI0_D0	MCU_HYPERBUS0_DQ0	Yes	Up	0	Enable	1
MCU_OSPI0_D1	MCU_HYPERBUS0_DQ1	Yes	Up	0	Enable	1
MCU_OSPI0_D2	MCU_HYPERBUS0_DQ2	Yes	Up	0	Enable	1
MCU_OSPI0_D3	MCU_HYPERBUS0_DQ3	Yes	Up	0	Enable	1
MCU_OSPI0_D4	MCU_HYPERBUS0_DQ4	Yes	Up	0	Enable	1
MCU_OSPI0_D5	MCU_HYPERBUS0_DQ5	Yes	Up	0	Enable	1
MCU_OSPI0_D6	MCU_HYPERBUS0_DQ6	Yes	Up	0	Enable	1
MCU_OSPI0_D7	MCU_HYPERBUS0_DQ7	Yes	Up	0	Enable	1
MCU_OSPI0_CSn0	MCU_HYPERBUS0_CSn0	Yes	Up	0	Disable	1
MCU_OSPI0_CSn1	MCU_HYPERBUS_RESETn	Yes	Up	0	Disable	1
MCU_OSPI0_CSn2	MCU_HYPERBUS0_RESET On	Yes	Up	0	Enable	2
MCU_OSPI0_CSn3	MCU_HYPERBUS_INTn	Yes	Up	0	Enable	2

Copyright © 2024 Texas Instruments Incorporated

4.3.5 OSPI Boot Device Configuration

Octal SPI flash memories support various protocols, and the OSPI boot mode of the device only supports a specific protocol defined below. Additionally, if the flash memory is complaint with JEDEC xSPI standards JESD251 and JESD216D, refer to Section 4.3.8. The OSPI protocol is described according to bit-width (1 or 8) and data rate (S or D for *S*ingle Data rate or *D*ouble Data rate) for the Command/Address/Data segments of the protocol. The OSPI boot mode supports 1S-1S-8S mode. The Command and Address issued are 8 bits and 24 bits, respectively. The Read Command issued for OSPI mode is 0x8b, followed by zero for address and 8 dummy cycles. The frequency of operation is 33 MHz.

The following boot mode pin configuration and corresponding pin usage and mux configuration are shown below. This is the OSPI boot mode.

Primary boot mode B must be set to 0 if MCU only is set to 0.

Table 4-15 shows configuration pins assignment to functions when boot mode is the Octal SPI.

Table 4-15. OSPI Configuration Fields

BOOTMODE Pins	Field	Value	Description	MCU Only=1 Value
6	Speed	0	33 MHz using manual tap selection	0
		1	Reserved	
5	Iclk	0	Iclock source external	0
		1	Iclock source internal	
4	Csel	0	Boot Flash is on CS 0	0
		1	Boot Flash is on CS 1	

Table 4-16 summarizes the OSPI pin configuration done by ROM code for OSPI boot device.

Table 4-16. OSPI Pin Usage

Device Pin	Module Signal	Pull Enable	Pull Direction	Driver Index	Rx En/Dis	Pinmux Sel
MCU_OSPI0_CLK0	MCU_OSPI0_CLK	Disable	Up	0	Disable	0
MCU_OSPI0_LBCLK0	MCU_OSPI0_LBCLK0	Disable	Up	0	Enable	0
MCU_OSPI0_DQS	MCU_OSPI0_DQS	Disable	Up	0	Enable	0
MCU_OSPI0_D0	MCU_OSPI0_D0	Enable	Up	0	Enable	0
MCU_OSPI0_D1	MCU_OSPI0_D1	Enable	Up	0	Enable	0
MCU_OSPI0_D2	MCU_OSPI0_D2	Enable	Up	0	Enable	0
MCU_OSPI0_D3	MCU_OSPI0_D3	Enable	Up	0	Enable	0
MCU_OSPI0_D4	MCU_OSPI0_D4	Enable	Up	0	Enable	0
MCU_OSPI0_D5	MCU_OSPI0_D5	Enable	Up	0	Enable	0
MCU_OSPI0_D6	MCU_OSPI0_D6	Enable	Up	0	Enable	0
MCU_OSPI0_D7	MCU_OSPI0_D7	Enable	Up	0	Enable	0
MCU_OSPI0_CSn0	MCU_OSPI0_CSn0	Enable	Up	0	Disable	0
MCU_OSPI0_CSn1	MCU_OSPI0_CSn1	Enable	Up	0	Disable	0

Copyright © 2024 Texas Instruments Incorporated

4.3.6 QSPI Boot Device Configuration

The QSPI boot mode supports the 1S-1S-4S mode only (Bit-width =1 Or 4, Single Data Rate). The Command and Address issued are 8 bits and 24 bits, respectively. The Read Command issued for QSPI is 0x6b, followed by zero for address and 8 dummy cycles. The frequency of operation supported is 33 MHz.

The following boot mode pin configuration and corresponding pin usage and mux configuration are shown below. This is the QSPI Port 0 boot mode.

Primary boot mode B and Primary Boot Mode Config 6 must be set to 0 if MCU only is set to 0.

Table 4-17 shows configuration pins assignment to functions when boot mode is the QSPI on OSPI mode.

Table 4-17. QSPI Boot Configuration Fields

BOOTMODE Pins	Field	Value	Description	MCU Only=1 Value
6	Port	0	Port 1	0
		1	Port 0	
5	Iclk	0	Iclock source external	0
		1	Iclock source internal	
4	Csel	0	Boot Flash is on CS 0	0
		1	Boot Flash is on CS 1	

Table 4-18 summarizes the OSPI pin configuration done by ROM code for QSPI boot device on port 0.

Table 4-18. QSPI Port 0 Pin Usage

Device Pin	Module Signal	Pull Enable	Pull Direction	Driver Index	Rx En/Dis	Pinmux Sel
MCU_OSPI0_CLK0	MCU_OSPI0_CLK	Disable	Up	0	Disable	0
MCU_OSPI0_LBCLKO	MCU_OSPI0_LBCLKO	Disable	Up	0	Enable	0
MCU_OSPI0_DQS	MCU_OSPI0_DQS	Disable	Up	0	Enable	0
MCU_OSPI0_D0	MCU_OSPI0_D0	Enable	Up	0	Enable	0
MCU_OSPI0_D1	MCU_OSPI0_D1	Enable	Up	0	Enable	0
MCU_OSPI0_D2	MCU_OSPI0_D2	Enable	Up	0	Enable	0
MCU_OSPI0_D3	MCU_OSPI0_D3	Enable	Up	0	Enable	0
MCU_OSPI0_CSn0	MCU_OSPI0_CSn0	Enable	Up	0	Disable	0
MCU_OSPI0_CSn1	MCU_OSPI0_CSn1	Enable	Up	0	Disable	0

Table 4-19. QSPI Port 1 Pin Usage

Device Pin	Module Signal	Pull Enable	Pull Direction	Driver Index	Rx En/Dis	Pinmux Sel
MCU_OSPI1_CLK	MCU_OSPI1_CLK	Disable	Up	0	Disable	0
MCU_OSPI1_LBCLKO	MCU_OSPI1_LBCLKO	Disable	Up	0	Enable	0
MCU_OSPI1_DQS	MCU_OSPI1_DQS	Disable	Up	0	Enable	0
MCU_OSPI1_D0	MCU_OSPI1_D0	Enable	Up	0	Enable	0
MCU_OSPI1_D1	MCU_OSPI1_D1	Enable	Up	0	Enable	0
MCU_OSPI1_D2	MCU_OSPI1_D2	Enable	Up	0	Enable	0
MCU_OSPI1_D3	MCU_OSPI1_D3	Enable	Up	0	Enable	0
MCU_OSPI1_CSn0	MCU_OSPI1_CSn0	Enable	Up	0	Disable	0
MCU_OSPI1_CSn1	MCU_OSPI1_CSn1	Enable	Up	0	Disable	0

Copyright © 2024 Texas Instruments Incorporated

4.3.7 SPI Boot Device Configuration

The SPI boot mode supports the 1S-1S-1S mode only (Bit-width =1, Single Data Rate). The Command and Address issued are 8 bits and 24 bits, respectively. The Read Command issued for SPI is 0x03, followed by zero for address and 0 dummy cycles.The frequency of operation supported is 4.156 MHz.

The following boot mode pin configuration and corresponding pin usage and mux configuration are shown below. This is the SPI Port 0 boot mode.

Table 4-20. MCU_BOOTMODE Pin Map SPI Port 0

9	8	7	6	5	4	3	2	1	0
Rsvd (not for boot use)		OVRD	MCU Only	Primary Boot Mode A			PLL Config		
X	X	X	0	1	1	0	X	X	X

Table 4-21. BOOTMODE Pin Map SPI Port 0

7	6	5	4	3	2	1	0
Backup Boot Mode Config	Primary Boot Mode Config			Backup Boot Mode			Primary Boot B
X	0/1	0/1	0/1	X	X	X	1

Primary boot mode B and Primary Boot Mode Config 6 must be set to 0 if MCU only is set to 0.

Table 4-22 shows configuration pins assignment to functions when boot mode is the SPI on OSPI port mode.

The SPI bus will be run at 4.156 MHz.

Table 4-22. SPI Boot Configuration Fields

BOOTMODE Pins	Field	Value	Description	MCU Only=1 Value
6 (7)[(1)]	Port	0	Port 0	N/A
		1	Port 1	
5	Mode	0	SPI Mode 0	N/A
		1	SPI Mode 3	
4	Csel	0	Boot Flash is on CS 0	N/A
		1	Boot Flash is on CS 1	

(1) When SPI was chosen as a backup mode and MCU Only = 0

Table 4-23 summarizes the OSPI pin configuration done by ROM code for SPI boot device on port 0.

Table 4-23. SPI Port 0 Pin Usage

Device Pin	Module Signal	Pull Enable	Pull Direction	Driver Index	Rx En/Dis	Pinmux Sel
MCU_OSPI0_CLK0	MCU_OSPI0_CLK	Disable	Up	0	Disable	0
MCU_OSPI0_LBCLKO	MCU_OSPI0_LBCLKO	Disable	Up	0	Enable	0
MCU_OSPI0_DQS	MCU_OSPI0_DQS	Disable	Up	0	Enable	0
MCU_OSPI0_D0	MCU_OSPI0_D0	Enable	Up	0	Enable	0
MCU_OSPI0_D1	MCU_OSPI0_D1	Enable	Up	0	Enable	0
MCU_OSPI0_CSn0	MCU_OSPI0_CSn0	Enable	Up	0	Disable	0
MCU_OSPI0_CSn1	MCU_OSPI0_CSn1	Enable	Up	0	Disable	0

Table 4-24. SPI Port 1 Pin Usage

Device Pin	Module Signal	Pull Enable	Pull Direction	Driver Index	Rx En/Dis	Pinmux Sel
MCU_OSPI1_CLK	MCU_OSPI1_CLK	Disable	Up	0	Disable	0
MCU_OSPI1_LBCLKO	MCU_OSPI1_LBCLKO	Disable	Up	0	Enable	0
MCU_OSPI1_DQS	MCU_OSPI1_DQS	Disable	Up	0	Enable	0
MCU_OSPI1_D0	MCU_OSPI1_D0	Enable	Up	0	Enable	0

Copyright © 2024 Texas Instruments Incorporated

Table 4-24. SPI Port 1 Pin Usage (continued)

Device Pin	Module Signal	Pull Enable	Pull Direction	Driver Index	Rx En/Dis	Pinmux Sel
MCU_OSPI1_D1	MCU_OSPI1_D1	Enable	Up	0	Enable	0
MCU_OSPI1_CSn0	MCU_OSPI1_CSn0	Enable	Up	0	Disable	0
MCU_OSPI1_CSn1	MCU_OSPI1_CSn1	Enable	Up	0	Disable	0

Copyright © 2024 Texas Instruments Incorporated

4.3.8 xSPI Boot Device Configuration

The xSPI protocol defines 1S-1S-1S mode for general backwards compatibility, and 8D-8D-8D for maximum throughput (where bit-width (1 or 8) and data rate (S or D for *S*ingle Data rate or *D*ouble Data rate).

For 1S-1S-1S mode of operation (Bit-width =1, Single Data Rate): The Command and Address issued are 8 bits and 24 bits, respectively. The Read Command issued is 0x0b followed by zero for address and 8 dummy cycles. The frequency of operation supported is 50 MHz.

For 8D-8D-8D mode of operation (Bit-width =8, Double Data Rate): The Command and Address issued are 8 bits and 32 bits, respectively. The Read Command issued is 0x0b or 0xee, followed by zero for address, 16 or 20 dummy cycles. The frequency of operation supported is 25 MHz. Additionally, the flash is expected to be configured in 8D mode out of POR through nonvolatile configuration register.

For SFDP mode, ROM starts operation in 1S-1S-1S mode, reads SFDP header from flash memory to get 8D-8D-8D switching sequence, Read Command, CMD Extension, and Byte Order. SFDP parsing of ROM is described below; on successful parsing, ROM issues an 8D-8D-8D command switching sequence and then reads the boot image in 8D-8D-8D mode with the read command specified in the SFDP header.

The following boot mode pin configuration and corresponding pin usage and mux configuration are shown below. This is the xSPI boot mode.

Table 4-25. MCU_BOOTMODE Pin Map xSPI

9	8	7	6	5	4	3	2	1	0
Rsvd (not for boot use)		OVRD	MCU Only	Primary Boot Mode A			PLL Config		
X	X	X	X	0	1	1	X	X	X

Table 4-26. BOOTMODE Pin Map xSPI

7	6	5	4	3	2	1	0
Backup Boot Mode Config	Primary Boot Mode Config			Backup Boot Mode			Primary Boot B
X	X	X	X	X	X	X	0

Table 4-27 shows configuration pins assignment to functions when boot mode is the xSPI on OSPI port mode.

Table 4-27. xSPI Boot Configuration Fields

BOOTMODE Pins	Field	Value	Description	MCU Only=1 Value
6	SFDP	0	SFDP (Serial Flash Discovery Parameter) Disabled	1
		1	SFDP Enabled	
5	Pin Cmd	0	0x0B Read Command	0
		1	0xEE Read Command	
4	Mode	0	SPI-STR (1S-1S-1S) at 50 MHz	0
		1	OCTAL-DTR (8D-8D-8D) at 25 MHz	

Table 4-28 summarizes the OSPI pin configuration done by ROM code for xSPI boot device on port 0.

Table 4-28. xSPI Pin Usage

Device Pin	Module Signal	Pull Enable	Pull Direction	Driver Index	Rx En/Dis	Pinmux Sel
MCU_OSPI0_CLK0	MCU_OSPI0_CLK	Disable	Up	0	Disable	0
MCU_OSPI0_LBCLKO	MCU_OSPI0_LBCLKO	Disable	Up	0	Enable	0
MCU_OSPI0_DQS	MCU_OSPI0_DQS	Disable	Up	0	Enable	0
MCU_OSPI0_D0	MCU_OSPI0_D0	Enable	Up	0	Enable	0
MCU_OSPI0_D1	MCU_OSPI0_D1	Enable	Up	0	Enable	0
MCU_OSPI0_D2	MCU_OSPI0_D2	Enable	Up	0	Enable	0
MCU_OSPI0_D3	MCU_OSPI0_D3	Enable	Up	0	Enable	0

Copyright © 2024 Texas Instruments Incorporated

Table 4-28. xSPI Pin Usage (continued)

Device Pin	Module Signal	Pull Enable	Pull Direction	Driver Index	Rx En/Dis	Pinmux Sel
MCU_OSPI0_D4	MCU_OSPI0_D4	Enable	Up	0	Enable	0
MCU_OSPI0_D5	MCU_OSPI0_D5	Enable	Up	0	Enable	0
MCU_OSPI0_D6	MCU_OSPI0_D6	Enable	Up	0	Enable	0
MCU_OSPI0_D7	MCU_OSPI0_D7	Enable	Up	0	Enable	0
MCU_OSPI0_CSn0	MCU_OSPI0_CSn0	Enable	Up	0	Disable	0
MCU_OSPI0_CSn1	MCU_OSPI0_CSn1	Enable	Up	0	Disable	0

Copyright © 2024 Texas Instruments Incorporated

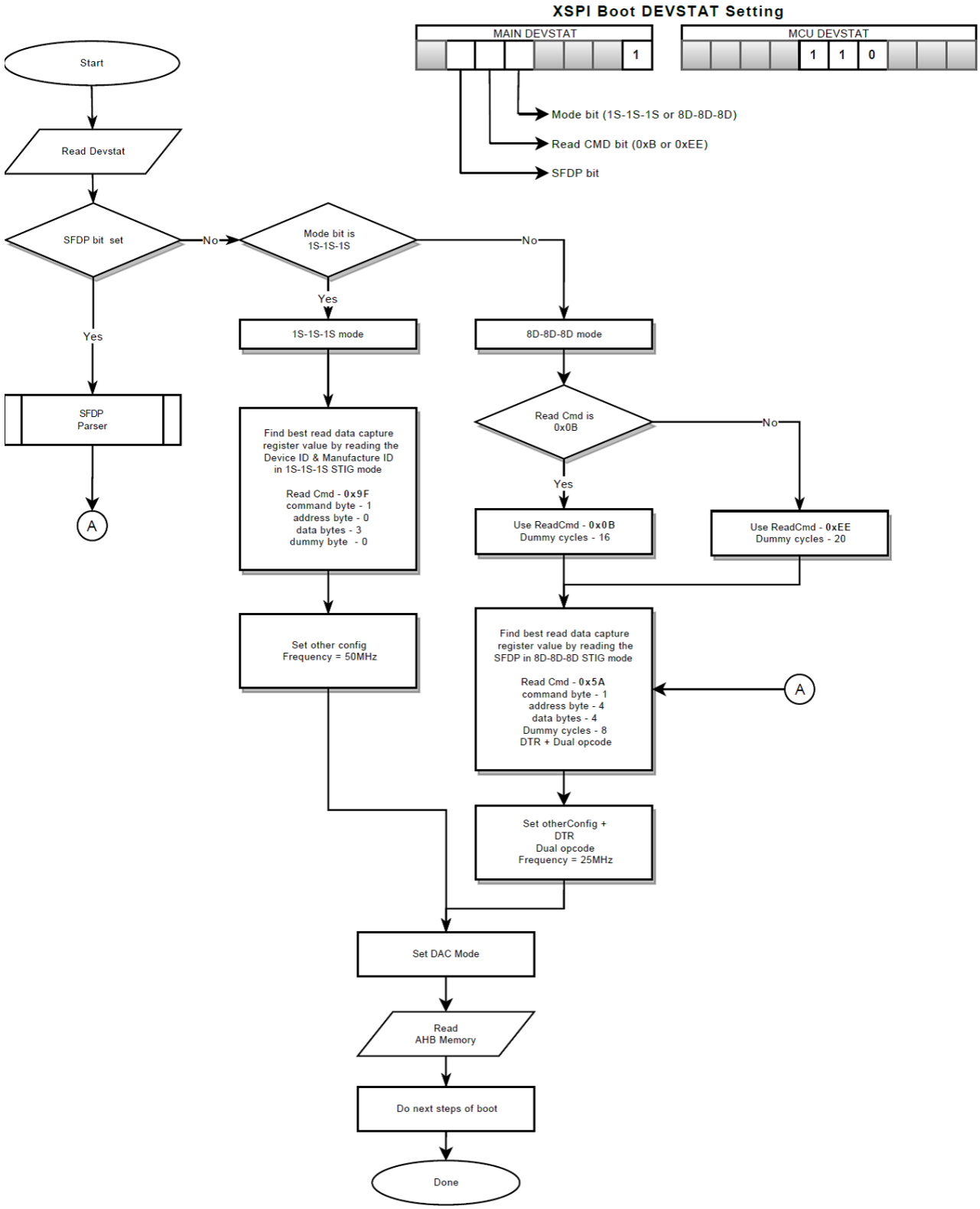


Figure 4-5. Title TBD

Copyright © 2024 Texas Instruments Incorporated

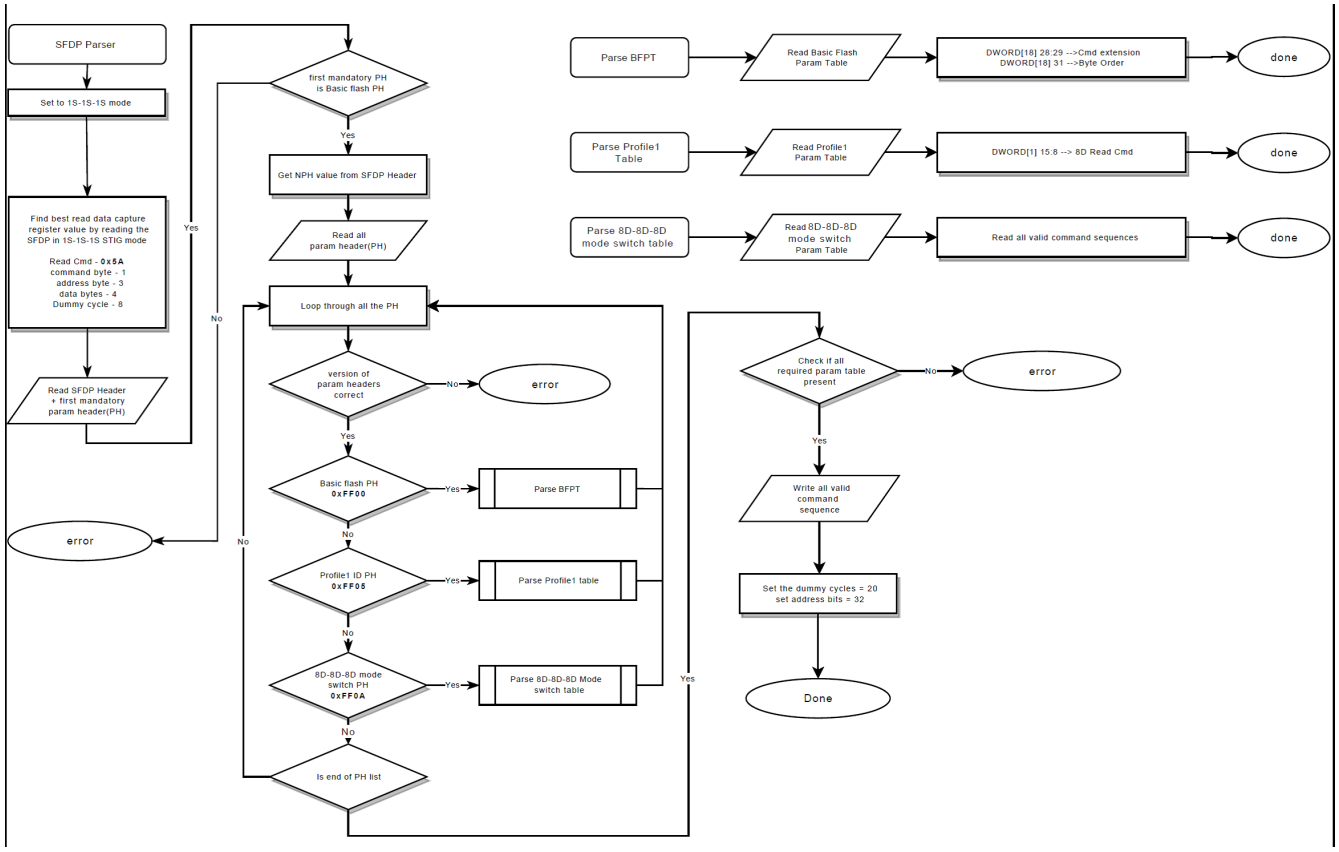


Figure 4-6. ROM SFDP Parser Flow

Copyright © 2024 Texas Instruments Incorporated

4.3.9 I2C Boot Device Configuration

Table 4-29 shows configuration pins assignment to functions when boot mode is the I2C mode.

Table 4-29. I2C Boot Configuration Fields

BOOTMODE Pins	Field	Value	Description	MCU Only=1 Value
6	Bus reset	0	Hung bus reset attempt after 1 ms	0
		1	No hung bus reset attempted	
4	Address	0	EEPROM's address is 0x50	0
		1	EEPROM's address is 0x51	

The I^2C bus is considered inactive if the data line is low and clock remains high for the specified timeout time. Recovery consists of driving the clock a stop condition is detected. A stop condition is a transition on the data line from 0 to 1 while the clock line is high. If the clock line is stuck low there is no way to take control of the bus.

Table 4-30 summarizes the I2C pin configuration done by ROM code for I^2C boot device.

Table 4-30. I2C Pin Usage

Device Pin	Module Signal	Pull Enable	Pull Direction	Driver Index	Rx En/Dis	Pinmux Sel
MCU_I2C0_SCL	MCU_I2C0_SCL	Enable	Up	0	Enable	0
MCU_I2C0_SDA	MCU_I2C0_SDA	Enable	Up	0	Enable	0

Copyright © 2024 Texas Instruments Incorporated

4.3.10 MMC/SD Card Boot Device Configuration

Table 4-31 shows configuration pins assignment to functions when boot mode is the MMC/SD card mode.

Table 4-31. MMC/SD Card Boot Configuration Fields

BOOTMODE Pins	Field	Value	Description	MCU Only=1 Value
6 (7)[1]	Port	0	MMC Port 0	N/A
		1	MMC Port 1	
5	Bus Width	0	8-bit bus width for MMC/SD 0 4-bit bus width for MMC/SD 1	N/A
		1	1-bit bus width	
4	FS/Raw	0	Filesystem mode	N/A
		1	Raw Mode	

(1) When MMCSD was chosen as a backup mode and MCU Only = 0

Table 4-32 summarizes the MMC pin configuration done by ROM code for MMC/SD Card boot device on port 1.

Note

Note that MMC Port 0 has no pin mux.

Table 4-32. MMC/SD Card Port 1 Pin Usage

Device Pin	Module Signal	Pull Enable	Pull Direction	Driver Index	Rx En/Dis	Pinmux Sel
MMC1_DAT3[1]	MMC1_DAT3	Enable	Up	0	Enable	0
MMC1_DAT2[1]	MMC1_DAT2	Enable	Up	0	Enable	0
MMC1_DAT1[1]	MMC1_DAT1	Enable	Up	0	Enable	0
MMC1_DAT0	MMC1_DAT0	Enable	Up	0	Enable	0
MMC1_CLKB	MMC1_CLK	Enable	Up	0	Enable	0
MMC1_CLK	MMC1_CLK	Enable	Up	0	Enable	0
MMC1_CMD	MMC1_CMD	Enable	Up	0	Enable	0
MMC1_SDCD	MMC1_SDCD	Enable	Up	0	Enable	0

(1) In 1-bit mode, only MMC1_DAT0 is configured.

Copyright © 2024 Texas Instruments Incorporated

4.3.11 eMMC Boot Device Configuration

Table 4-33 shows configuration pins assignment to functions when boot mode is the eMMC mode.

Table 4-33. eMMC Boot Configuration Fields

BOOTMODE Pins	Field	Value	Description	MCU Only=1 Value
6	Port	0	Port 0	N/A
		1	Reserved	
5	Bus Width	0	Max width by port (8-bit on port 0)	N/A
		1	1-bit only	
4	Voltage	0	1.8V	N/A
		1	3.3V	

Note

Note that MMC Port 0 has no pin mux options.

4.3.11.1 eMMC Flash

To support eMMC boot mode across a device warm reset, eMMC flash has the following requirements:

1. The reset line must be connected to the eMMC flash input reset pin.
2. The eMMC flash ext_csd[162] RST_n_ENABLE must be set to the expected configuration for the warm-reset signal to propagate to the flash device for the boot process to succeed. Possible configurations are shown in Table 4-34.

Note

For warm_reset eMMC boot to function correctly, 0x1 must be written to RST_n_ENABLE in the ext_csd[162] Register. This will enable the RST_n signal.

Table 4-34. ext_csd[162] Register

Bit	Field	Type	Description
7:2	Reserved		Reserved
1:0	RST_n_ENABLE	R/W	0x0: RST_n signal is temporarily disabled (default) 0x1: RST_n signal is permanently enabled 0x2 RST_n signal is permanently disabled 0x3: Reserved

Copyright © 2024 Texas Instruments Incorporated

4.3.12 Ethernet Boot Device Configuration

Table 4-35 shows configuration pins assignment to functions when boot mode is the Ethernet RGMII mode.

Table 4-35. Ethernet RGMII Boot Configuration Fields

BOOTMODE Pins	Field	Value	Description	MCU Only=1 Value
6	Clkout	0	25 MHz clock not generated on MCU_CLKOUT0	0
		1	25 MHz clock generated on MCU_CLKOUT0	
5	Delay	0	RGMII with internal Tx delay	0
		1	RGMII with external Tx delay	
4	Link stat	0	MDIO PHY scan used for speed/duplex setup	0
		1	RGMII status register used for speed/duplex setup	

Table 4-36 shows configuration pins assignment to functions when boot mode is the Ethernet RMII mode.

Table 4-36. Ethernet RMII Boot Configuration Fields

BOOTMODE Pins	Field	Value	Description	MCU Only=1 Value
6	Clk out	0	50 MHz clock not generated on MCU_CLKOUT0	0
		1	50 MHz clock generated on MCU_CLKOUT0	
5	Clk src	0	External clock source	0
		1	Internal clock source	
4	Reserved	X	Not used	N/A

Table 4-37 shows configuration pins assignment to functions when boot mode is the Ethernet is the backup mode.

Table 4-37. Ethernet Backup Boot Configuration Field

BOOTMODE Pins	Field	Value	Description	MCU Only=1 Value
7[(1)]	Interface	0	RGMII with internal Tx delay	N/A
		1	RMII with external clock source	

(1) When Ethernet was chosen as a backup mode and MCU Only = 0

Table 4-38 summarizes the RGMII pin configuration done by ROM code for Ethernet boot device on RGMII port.

Table 4-38. RGMII Pin Usage

Device Pin	Module Signal	Pull Enable	Pull Direction	Driver Index	Rx En/Dis	Pinmux Sel
MCU_RGMII1_TX_CTL	MCU_RGMII1_TX_CTL	Disable	Down	0	Disable	0
MCU_RGMII1_RX_CTL	MCU_RGMII1_RX_CTL	Disable	Down	0	Enable	0
MCU_RGMII1_TD3	MCU_RGMII_TD3	Disable	Down	0	Disable	0
MCU_RGMII1_TD2	MCU_RGMII_TD2	Disable	Down	0	Disable	0
MCU_RGMII1_TD1	MCU_RGMII_TD1	Disable	Down	0	Disable	0
MCU_RGMII1_TD0	MCU_RGMII_TD0	Disable	Down	0	Disable	0
MCU_RGMII1_TXC	MCU_RGMII1_TXC	Disable	Down	0	Disable	0
MCU_RGMII1_RXC	MCU_RGMII1_RXC	Disable	Down	0	Enable	0
MCU_RGMII1_RD3	MCU_RGMII1_RD3	Disable	Down	0	Enable	0
MCU_RGMII1_RD2	MCU_RGMII1_RD2	Disable	Down	0	Enable	0
MCU_RGMII1_RD1	MCU_RGMII1_RD1	Disable	Down	0	Enable	0
MCU_RGMII1_RD0	MCU_RGMII1_RD0	Disable	Down	0	Enable	0
MCU_MDIO0_MDIO[(1)]	MCU_MDIO0_MDIO	Enable	Up	0	Enable	0
MCU_MDIO0_MDC[(1)]	MCU_MDIO0_MDC	Enable	Up	0	Disable	0

Copyright © 2024 Texas Instruments Incorporated

Table 4-38. RGMII Pin Usage (continued)

Device Pin	Module Signal	Pull Enable	Pull Direction	Driver Index	Rx En/Dis	Pinmux Sel
WKUP_GPIO_11[(2)]	MCU_CLKOUT0	Disable	Down	0	Disable	6

(1) Note that the MDIO pins are only configured if the link stat is detected via PHY scan.
(2) If MCU only is set to 0 and BOOTMODE6 is set to 1 then the 25 MHz clock out is generated.

Table 4-39 summarizes the RMII pin configuration done by ROM code for Ethernet boot device on RMII port.

Table 4-39. RMII Pin Usage

Device Pin	Module Signal	Pull Enable	Pull Direction	Driver Index	Rx En/Dis	Pinmux Sel
MCU_RGMII1_TX_CTL	MCU_RMII1_CRS_DV	Disable	Down	0	Enable	1
MCU_RGMII1_RX_CTL	MCU_RMII1_RX_ER	Disable	Down	0	Enable	1
MCU_RGMII1_TD1	MCU_RMII1_TXD1	Disable	Down	0	Disable	1
MCU_RGMII1_TD0	MCU_RMII1_TXD0	Disable	Down	0	Disable	1
MCU_RGMII1_TXC	MCU_RMII1_TX_EN	Disable	Down	0	Disable	1
MCU_RGMII1_RXC	MCU_RMII1_REF_CLK	Disable	Down	0	Enable	1
MCU_RGMII1_RD1	MCU_RMII1_RXD1	Disable	Down	0	Enable	1
MCU_RGMII1_RD0	MCU_RMII1_RXD0	Disable	Down	0	Enable	1
MCU_MDIO0_MDIO	MCU_MDIO0_MDIO	Enable	Up	0	Enable	0
MCU_MDIO0_MDC	MCU_MDIO_MDC	Enable	Up	0	Disable	0
WKUP_GPIO_11[(1)]	MCU_CLKOUT0	Disable	Down	0	Disable	6

(1) If BOOTMODE6 is set to 1 then a 50 MHz clock out is generated.

Copyright © 2024 Texas Instruments Incorporated

4.3.13 USB Boot Device Configuration

Table 4-40 shows configuration pins assignment to functions when boot mode is the USB mode.

Table 4-40. USB Boot Configuration Fields

BOOTMODE Pins	Field	Value	Description	MCU Only=1 Value
5	Mode	0	DFU (USB device firmware upgrade)	N/A
		1	Reserved	
4	Lane Swap	0	D+/D- lines are not swapped	N/A
		1	D+/D- lines are swapped	

Table 4-41 summarizes the USB pin configuration done by ROM code for USB boot device on port 0.

Table 4-41. USB Port 0 Pin Usage

Device Pin	Module Signal	Pull Enable	Pull Direction	Driver Index	Rx En/Dis	Tx En/Dis	Pinmux Sel
TIMER_IO1	USB0_DRVVBUS	Enable	Down	0	Disable	Enable	6

Note

Note that other USB pins do not have pin mux options.

Copyright © 2024 Texas Instruments Incorporated

4.3.14 PCIe Boot Device Configuration

Table 4-42 shows configuration pins assignment to functions when boot mode is the PCIe mode.

Table 4-42. PCIe Boot Configuration Fields

BOOTMODE Pins	Field	Value	Description	MCU Only=1 Value
5	SSC	0	SSC Enabled	N/A
		1	SSC Disabled	
4	Clocking	0	PHY clock from external pins	N/A
		1	PHY clock from internal source	

Table 4-43 summarizes the PCIe pin configuration done by ROM code for PCIe boot device on port 1.

Table 4-43. PCIe Port 1 Pin Usage

Device Pin	Module Signal	Pull Enable	Pull Direction	Driver Index	Rx En/Dis	Tx En/Dis	Pinmux Sel
TIMER_IO0	PCIE1_CLKREQn	Enable	Up	0	Enable	Enable	6

Note

Note that PCIe SerDes pins do not have pin mux options.

Copyright © 2024 Texas Instruments Incorporated

4.3.15 UART Boot Device Configuration

ROM Code always configures the UART port to 115200 kbaud, 8-n-1 mode, and the XMODEM protocol is used to transfer the boot data.

Table 4-44. UART Boot Configuration Fields

BOOTMODE Pins	Field	Value	Description	MCU Only=1 Value
6	Reserved	-	Not used	N/A
5	Reserved	-	Not used	N/A
4 (7)[(1)]	Port	0	MCU_UART (port 0)	0
		1	WKUP_UART (port 1)	

(1) When UART was chosen as a backup mode and MCU Only=0

Table 4-45 summarizes the UART pin configuration done by ROM code for UART host on port 0.

Table 4-45. UART Port 0 Pin Usage

Device Pin	Module Signal	Pull Enable	Pull Direction	Driver Index	Rx En/Dis	Pinmux Sel
WKUP_GPIO0_12	MCU_UART_TXD	Enable	Up	0	Disable	0
WKUP_GPIO0_13	MCU_UART_RXD	Enable	Up	0	Enable	0

Table 4-46 summarizes the UART pin configuration done by ROM code for UART host on port 1.

Table 4-46. UART Port 1 Pin Usage

Device Pin	Module Signal	Pull Enable	Pull Direction	Driver Index	Rx En/Dis	Pinmux Sel
WKUP_UART_TXD	WKUP_UART_TXD	Enable	Up	0	Disable	0
WKUP_UART_RXD	WKUP_UART_RXD	Enable	Up	0	Enable	0

4.3.16 Serial NAND Boot Device Configuration

Serial NAND defines 1S-1S-1S mode for general backwards compatibility, and 1S-1S-8S for maximum throughput (S here means *S*ingle Data rate). Serial NAND memory array is organized into pages of size 2KB/4KB. Read is a two-step process where a complete page is first read into flash's internal buffer/cache using Page read command and then the host controller reads from internal buffer in 1- or 4- or 8-bit mode using the read commands. Page read command that is issued is 0x13, followed by 24 address bits. The frequency of operation supported is 50 MHz.

For 1S-1S-1S mode of operation (Bit-width =1, Single Data Rate). The Command and Address issued are 8 bits and 16 bits respectively. The Read Command that is issued is 0x0b, followed by address bits and 8 dummy cycles.

For 1S-1S-8S mode of operation (Bit-width =8, Single Data Rate). The Command and Address issued are 8 bits and 16 bits respectively. The Read Command that is issued is 0x8b, followed by address bits and 8 dummy cycles. Additionally, flash is configured in 8-bit mode after POR through volatile configuration register if the manufacturer is Winbond.

For 1S-1S-4S mode of operation (Bit-width =4, Single Data Rate). The Command and Address issued are 8 bits and 16 bits respectively. The Read Command that is issued is 0x6b, followed by address bits and 8 dummy cycles.

Note that in 8-bit mode pin mux is done for all 8 OSPI data lines and in 4-bit/1-bit mode pin mux is done for 4 OSPI data lines. This is done to disable the HOLD functionality feature in 1-bit mode.

Serial NAND boot expects ECC to be auto-managed by the flash. Most of the flashes have the ECC enabled by default and can do 1-bit correction and 2-bit detection for ECC errors. ROM checks for 2-bit ECC error via status register 3 (address 0xC0) bit 5 after every page load. In case of 2-bit ECC error the boot will fail and ROM will take the fallback option.

Copyright © 2024 Texas Instruments Incorporated

Serial NAND boot also manages bad blocks that can be present in the flash at time of shipment or develop during the lifetime. Bad block marker is a non-FFh data byte stored at Byte 0 of spare area of Page 0 for each bad block and ROM checks for the same while reading the first page of a memory block. ROM will skip the particular block if it is marked as bad and move to the next one.

The following boot mode pin configuration and corresponding pin usage and mux configuration are shown below. This is the Serial NAND boot mode.

Note

Serial Nand driver in ROM does not support devices that have multiple planes, as they require special handling to read even-numbered blocks.

Table 4-47. Serial NAND Configuration Fields

BOOTMODE Pins	Field	Value	Description	MCU Only=1 Value
4	Read Mode 1	0	OSPI/ 1-1-8 Mode (valid only when Read Mode 2 is 0)	0
		1	QSPI/ 1-1-4 mode (valid only when Read Mode 2 is 0)	
5	Read Mode 2	0	Reserved (Read mode is taken from Read Mode 1)	0
		1	SPI/ 1-1-1 mode (Read mode is taken from Read Mode 2 and Read Mode 1 is ignored)	

Table 4-48. Serial NAND Pin Usage

Device Pin	Module Signal	Pull Enable	Pull Direction	Driver Index	Rx En/Dis	Pinmux Sel
MCU_OSPI0_CLK	MCU_OSPI0_CLK	Disable	Up	0	Disable	0
MCU_OSPI0_LBCLKO	MCU_OSPI0_LBCLKO	Disable	Up	0	Enable	0
MCU_OSPI0_DQS	MCU_OSPI0_DQS	Disable	Up	0	Enable	0
MCU_OSPI0_D0	MCU_OSPI0_D0	Enable	Up	0	Enable	0
MCU_OSPI0_D1	MCU_OSPI0_D1	Enable	Up	0	Enable	0
MCU_OSPI0_D2	MCU_OSPI0_D2	Enable	Up	0	Enable	0
MCU_OSPI0_D3	MCU_OSPI0_D3	Enable	Up	0	Enable	0
MCU_OSPI0_D4	MCU_OSPI0_D4	Enable	Up	0	Enable	0
MCU_OSPI0_D5	MCU_OSPI0_D5	Enable	Up	0	Enable	0
MCU_OSPI0_D6	MCU_OSPI0_D6	Enable	Up	0	Enable	0
MCU_OSPI0_D7	MCU_OSPI0_D7	Enable	Up	0	Enable	0
MCU_OSPI0_CSn0	MCU_OSPI0_CSn0	Enable	Up	0	Disable	0
MCU_OSPI0_CSn1	MCU_OSPI0_CSn1	Enable	Up	0	Disable	0

Copyright © 2024 Texas Instruments Incorporated

4.3.17 PLL Configuration

ROM code must be aware of the reference clock provided to PLLs. That is, the speed of the quartz crystal, or the clock supplied by an external clock oscillator. On how to indicate the PLL reference clock, see Table 4-3, *PLL Reference Clock Selection*.

See *Clock Management*, for the PLL reference clock scheme.

ROM code configures only PLLs which are required during boot. Therefore, if a PLL is required for the backup boot mode but not the primary boot mode, and if the backup boot mode never executes, then the PLLs required for backup boot are not enabled. Table 4-49 lists the enabled PLLs according to module or boot peripheral.

Table 4-49. PLL Configuration by Boot Mode

MCU_PLL0	MCU_PLL1	MCU_PLL2	Main PLL0	Main PLL1	Main PLL2	Main PLL3	Boot Mode
✓		✓					Hyperflash
✓	✓						OSPI/QSPI/SPI
✓		✓					Ethernet
✓	✓						I2C
✓	✓						UART
✓			✓				eMMC/MMC/SD
✓			✓	✓			USB
✓			✓		✓		PCIe
✓							Serial NAND

Note

All clock frequencies are in megahertz.

4.3.17.1 MCU_PLL0, MCU_PLL2, Main PLL0, and Main PLL3

Table 4-50 summarizes the ROM code settings used to configure MCU_PLL0, MCU_PLL2, Main PLL0, and Main PLL3 for the supported input clocks.

Table 4-50. PLL Configuration for MCU_PLL0, MCU_PLL2, Main PLL0, and Main PLL3

2000 MHz VCO frequency							
Ref clk	Refdiv	Pfd freq	Fbdiv	Frac	Vco	Delta	Postdiv
19.2	1	19.2	104	2796203	2000	3.97×10^{-8}	0 or 1
20	1	20	100	0	2000	0	0 or 1
24	1	24	83	5592405	2000	1.99×10^{-8}	0 or 1
25	1	25	80	0	2000	0	0 or 1
26	1	26	76	15486661	2000	5.50×10^{-8}	0 or 1
27	1	27	74	1242757	2000	4.42×10^{-8}	0 or 1

4.3.17.2 MCU_PLL1

The MCU_PLL1 is a fractional PLL configured with a VCO frequency of 2400 MHz.

Table 4-51. PLL Configuration for MCU_PLL1

2400 MHz VCO frequency							
Ref clk	Refdiv	Pfd freq	Fbdiv	Frac	Vco	Delta	Postdiv
19.2	1	19.2	125	0	2400	0	0
20	1	20	120	0	2400	0	0
24	1	24	100	0	2400	0	0
25	1	96	96	0	2400	0	0
26	1	26	92	5162220	2400	$1.8e\times10^{-8}$	0

Copyright © 2024 Texas Instruments Incorporated

Table 4-51. PLL Configuration for MCU_PLL1 (continued)

2400 MHz VCO frequency							
Ref clk	**Refdiv**	**Pfd freq**	**Fbdiv**	**Frac**	**Vco**	**Delta**	**Postdiv**
27	1	27	88	14913081	2400	5.53×10^{-8}	0

4.3.17.3 Main PLL1

The main PLL is a fractional PLL configured with a VCO frequency of 1920 MHz.

Table 4-52. PLL Configuration for Main PLL1

1920 MHz VCO frequency							
Ref clk	**Refdiv**	**Pfd freq**	**Fbdiv**	**Frac**	**Vco**	**Delta**	**Postdiv**
19.2	1	19.2	100	0	1920	0	0
20	1	20	96	0	1920	0	0
24	1	24	80	0	1920	0	0
25	1	25	76	13421773	1920	4.77×10^{-8}	0
26	1	26	73	14196106	1920	5.04×10^{-8}	0
27	1	27	71	1864135	1920	6.62×10^{-8}	0

4.3.17.4 Main PLL2

Main PLL2 is an fractional PLL configured with a VCO frequency of 1800 MHz.

Table 4-53. PLL Configuration for Main PLL 2

1800 MHz VCO frequency							
Ref clk	**Refdiv**	**Pfd freq**	**Fbdiv**	**Frac**	**Vco**	**Delta**	**Postdiv**
19.2	1	19.2	93	12582912	1800	0	0
20	1	20	90	0	1800	0	0
24	1	24	75	0	1800	0	0
25	1	25	72	0	1800	0	0
26	1	26	69	3871665	1800	1.38×10^{-8}	0
27	1	27	66	11184811	1800	3.97×10^{-8}	0

4.3.17.5 HSDIV Values

MCU PLL0

- HSDIV0 - (2-1)

MCU PLL1

- HSDIV0 - (6-1)
- HSDIV1 - (40-1)
- HSDIV2 - (30-1)
- HSDIV3 - (50-1)
- HSDIV4 - (18-1)

MCU PLL2

- HSDIV0 - (8-1)
- HSDIV1 - (4-1)
- HSDIV2 - (10-1)
- HSDIV3 - (25-1)
- HSDIV4 - (6-1)

MAIN PLL0

- HSDIV0 - (4-1)
- HSDIV1 - (5-1)
- HSDIV2 - (10-1)
- HSDIV3 - (15-1)

Copyright © 2024 Texas Instruments Incorporated

- HSDIV4 - (25-1)
- HSDIV5 - (20-1)
- HSDIV6 - (5-1)
- HSDIV7 - (5-1)
- HSDIV8 - (5 - 1)

MAIN PLL1

- HSDIV0 - (10-1)
- HSDIV1 - (6-1)
- HSDIV2 - (10-1)
- HSDIV3 - (10-1)
- HSDIV4 - unused
- HSDIV5 - (5-1)
- HSDIV6 - (50 - 1)
- HSDIV7 - (40-1)
- HSDIV8 - (48-1)

MAIN PLL2

- HSDIV0 - unused
- HSDIV1 - (3-1)
- HSDIV2 - (9-1)
- HSDIV3 - (6-1)
- HSDIV4 - (18-1)
- HSDIV5 - unused
- HSDIV6 - (4-1)
- HSDIV7 - (9-1)

4.3.17.6

Note

Note that the bringup and configuration of bootmode-specific PLLs by ROM code will result in a bootmode-specific device clocking setup which for example has an impact on which peripheral modules can readily be clocked and used by SBL/SPL prior to loading and bringing up System Firmware (SYSFW).

Copyright © 2024 Texas Instruments Incorporated

4.4 Boot Parameter Tables

The boot parameter tables direct the main module boot process. On cold boot the tables are created based on pin strapped values (see Section 4.3, *Boot Mode Pins*) and built-in data. The ROM Code supports two parameter tables stored as an array in a fixed memory address in the MSRAM, each of size 512 bytes. The ROM will attempt to boot using the primary table. On boot failure, ROM Code will retry using the second table.

Using two tables handles two cases.

- The first is in initial board manufacture where the primary boot table specifies boot from a flash device, and the flash is blank. ROM Code would then switch to the secondary boot mode which would receive the image externally (Ethernet, USB, PCIe, UART) and this image would flash the boot image.
- The second case is failure due to total flash corruption. In all flash parameter tables there exist backup addresses within the primary boot mode. This covers the problem of a flash update failure with a backup image present on the same flash device.

The boot tables reside at a fixed location in memory which is described in Section 4.7.2, *Global Memory Addresses Used by ROM Code*.

4.4.1 Common Header

These boot parameter tables have certain parameters common across all the boot modes, while the rest of the parameters are unique to the boot modes. The common entries in the boot parameter table are shown in Table 4-54.

Table 4-54. Boot Parameter Table Common Header

Byte Offset	Size (bytes)	Name	Description
0	2	Length	The length of the table
2	2	Checksum	Ones complement checksum over length bytes in the table. If 0 the checksum is not validated.
4	2	Peripheral	Identifies the boot peripheral and format of the table after the common header. See Table 4-55
6	2	Reserved	Reserved
8	4	Timeout	Timeout for this boot mode, in milliseconds
12	4	Magic	Magic value 0x01AD0911
16	40	PLL Config 0	PLL Configuration 0. See Table 4-56
56	40	PLL Config 1	PLL Configuration 1
96	40	PLL Config 2	PLL Configuraiton 2
136	40	PLL Config 3	PLL Configuration 3
176	40	PLL Config 4	PLL Configuration 4
216	40	PLL Config 5	PLL Configuration 5

Table 4-55 lists the possible boot modes used in the boot parameter tables.

Table 4-55. Boot Peripheral Selection

Peripheral Field Value	Description
0	Sleep (No boot)
10	Ethernet Reserved
20	Ethernet BOOTP/TFTP (general)
21	Ethernet RGMII specific
22	Ethernet RMII specific
30	PCIe
31	PCIe 1-lane
32	PCIe 2-lane
40	I2C
50	SPI

Copyright © 2024 Texas Instruments Incorporated

Table 4-55. Boot Peripheral Selection (continued)

Peripheral Field Value	Description
60	UART
70	USB DFU
71	USB Reserved
72	USB Reserved
80	QSPI
85	OSPI
90	Hyperflash Reserved
100	MMC/SD Card (general)
101	eMMC (general)
102	MMC 1-bit
103	MMC 4-bit
104	MMC 8-bit
110	GPMC Reserved
120	UFS Reserved
150	Serial NAND

4.4.2 PLL Setup

Table 4-56 through Table 4-60 describe the PLL configuration fields.

Table 4-56. Boot Parameter Table PLL Configuration

Byte Offset	Size (bytes)	Name	Description
0	1	Domain/cfg	See Table 4-57
1	1	Pll number	PLL number indexed from 0. See for PLL numbers.
2	1	Input source	See Table 4-59
3	1	Pll Type	This field must be 1 to indicate an SCPLL
4	4	Input Ref Clock	The PLL input clock, in Q16.16 format
8	4	Feed back divider, integer part	Integer value of feedback divider
12	4	Feed back divider, fractional part	Fractional portion of feedback divider. Total divider is the Integer part + (Fractional part / 2^{24})
16	1	Ref divider	Input clock pre-divider
17	1	Post divider 1	Output post divider 1
18	1	Post divider 2	Output post divider 2
19	1	Reserved	Reserved
20	2	Hsdiv Enable	Bit map. A set bit indicates that the corresponding hsdiv is enabled.
22	2	Reserved	Reserved
24	16	Hsdiv[16]	Array of hs divider values.

Table 4-57. PLL Domain and Enable Configuration

7	6	5	4	3	2	1	0
Reserved		Enable		Reserved			Domain

Table 4-58. PLL Domain and Enable Field Description

Field	Value	Description
Enable	0	PLL not configured
	1	PLL enabled only if currently disabled or in bypass
	2	PLL is unconditionally enabled. If currently enabled with a different configuration the PLL is first disabled
	3	PLL is unconditionally disabled
Domain	0	PLL is in the MCU domain

Copyright © 2024 Texas Instruments Incorporated

Table 4-58. PLL Domain and Enable Field Description (continued)

Field	Value	Description
	1	PLL is in the MAIN domain

Table 4-59. PLL Reference Source Bit Fields

7	6	5	4	3	2	1	0
Source Type			Source Index				

Table 4-60. PLL Reference Source Field Description

Field	Value	Description
Source Type	0	Source is HFOSC
	1	Source is external pin
	2	Reserved
	3-7	Reserved
Source Index	0-31	Source index (HFOSC[0-31] or pin[0-31], depending on Source Type) HFOSC[0] – WKUP_HFOSC0 HFOSC[1] – HFOSC1 (in MAIN domain) PIN[1] – EXT_REFCLK1 pin (not all PLLs, see *Clocking*

4.4.3 PCIe Boot Parameter Table

Table 4-61 shows the boot parameter table for PCIe boot. Must be preceded with the common boot parameters described in Table 4-54.

Table 4-61. PCIe Boot Parameter Table

Byte Offset	Size (bytes)	Name	Default Value	Description
256	4	portNum	From pins	Physical port number
260	4	AddrWid	64	PCIe address width
264	4	LinkRate	8000	Link rate in MHz
268	4	RefClkkHz	100000	Serdes reference clock in kHz
272	4	Nlanes	From pins	Number of PCIe lanes configured (link width)
276	4	Rsvd	4	Reserved
280	4	Rsvd	256	Reserved
284	4	Rsvd	256	Reserved
288	4	Rsvd	0	Reserved
292	4	Rsvd	0	Reserved
296	4	Vendor ID	0x104C	PCIe Vendor ID value (read from control registers)
300	4	Device ID	0xB00D	PCIe Device ID value (read from control registers)
304	4	Class code/revision ID	0x04800001	PCIe class code and revision ID value
308	4	Internal Clock	From Pins	If non-zero, internal (SoC) ref clock is used
312	4	sscEnable	1	Enable spread spectrum clock
316	4	RefSrc	From Pins	Selects serdes reference clock internal source. Refer to *Reference Clock Distribution* for details.

4.4.4 I2C Boot Parameter Table

Table 4-62 shows the boot parameter table for I2C boot. Must be preceded with the common boot parameters described in Table 4-54.

Table 4-62. I2C Boot Parameter Table

Byte Offset	Size (bytes)	Name	Default Value	Description
256	1	Port	0	Physical port number
257	1	Mode	From Pins	0x4E = I2C Master, 0x72 = I2C slave

Copyright © 2024 Texas Instruments Incorporated

Table 4-62. I2C Boot Parameter Table (continued)

Byte Offset	Size (bytes)	Name	Default Value	Description
258	1	Dev Addr	From Pins	I2C address when slave mode (0x10 or 0x11)
259	1	Reserved	0	Reserved
260	4	Mod Clock	0	I2C Module input clock. If 0, it is computed by ROM code.
264	2	Bus Freq	400	I2C Master mode bus frequency, in kHz
266	2	Bus Addr	From Pins	I2C Master mode storage device's address (0x50 or 0x51)
268	2	Read Index	0	Index to the active read offset (0 or 1)
270	2	Read Offset 0	0x0000	I2C Master mode read offset
272	2	Read Offset 1	0x8000	I2C Master mode backup read offset
274	2	Reserved	0	Reserved
276	2	Reserved	0	Reserved
280	2	Busy Timeout	From pins	Number of µs before a bus recovery is attempted. In units of microseconds in Q3 number format. Value of 0 disables bus recovery attempts.

4.4.5 OSPI/QSPI/SPI Boot Parameter Table

Table 4-63 shows the boot parameter table for OSPI, QSPI, or SPI boot. Must be preceded with the common boot parameters described in Table 4-54.

Table 4-63. OSPI/QSPI/SPI Boot Parameter Table

Byte Offset	Size (bytes)	Name	Default Value	Description
256	1	Port	0	Physical port number
257	1	Mode on	From Pins	If non-zero, the mode byte will be sent
258	1	Instruct Width	From Pins	Number of pins used to send instructions (1, 2, 4, 8)
259	1	Address Width	From Pins	Number of pins used to send address (1, 2, 4, 8)
260	1	Data Width	From Pins	Number of pins used to received data (1, 2, 4, 8)
261	1	Address Size	24	16 (SPI only), 24, and 32 bits are the valid address sizes
262	1	Mode	0	OSPI clock polarity and phase mode
263	1	CSEL	From Pins	Chip select number (0–3)
264	1	Read Cmd	From Pins	Command used to read read data
265	1	Mode byte	0	Value used for the mode byte (when active)
266	1	Dummy Cycles	From pins	Number of dummy cycles sent after the read command
267	1	clkRecovery	From pins	Clock recovery
268	1	dqsEnable	0	Enable DQS
269	1	Reserved	0	Reserved
270	2	Module Freq	0	The OSPI module frequency after PLL enable, in kHz. If 0, ROM code uses the value from the module clock tables.
272	4	Bus Frequency	From pins	The OSPI bus frequency, in kHz
276	4	Delay	0x08080808 or 0x01010101	The chip select read delays. Default value is based on the read command
280	4	Tap Delay	0xFFFFFFFF	The read tap selection. If 0xFFFFFFFF, the ROM code will scan the taps to find the best delay. The result will then overwrite the value in this table.
284	4	Internal Clk	From pins	0 = external (dqs) 1 = internal
288	4	notDAC	From pins	When 0, DAC mode is used
292	4	Read Index	0	Index to the active read address (0-1)
296	4	Read Addr 0	0x000000	The initial flash read address
300	4	Read Addr 1	0x400000 (0x4000 SPI)	Backup read address

Copyright © 2024 Texas Instruments Incorporated

Table 4-63. OSPI/QSPI/SPI Boot Parameter Table (continued)

Byte Offset	Size (bytes)	Name	Default Value	Description
304	4	Reserved	0	Reserved
308	4	Reserved	0	Reserved

4.4.6 Ethernet Boot Parameter Table

Table 4-64 is shown segmented into four sections:

1. The first section contains information required to configure the device hardware
2. The second section contains information used by the top level Ethernet module to execute the boot
3. The third section contains information for the Ethernet stack code.
4. The fourth section contains information for creating the BOOTP packet (vendor string, ID string and debug string), and holds information returned in the BOOTP response (Default route and file name)

Table 4-64 shows the boot parameter table for Ethernet boot. Must be preceded with the common boot parameters described in Table 4-54.

Table 4-64. Ethernet Boot Parameter Table

Byte Offset	Size (bytes)	Name	Default Value	Description
Hardware Configuration Options				
256	2	Mod Freq	0	Module clock frequency, kHz. If 0, ROM code computes the value.
258	1	Port Num	0	Physical port number
259	1	Interface	From Pins	0 = RGMII with internal delay 1 = RGMII with external delay 2 = RMII
260	1	Init Level	0	0 = Initialize only not enabled modules 1 = Full ethernet sub-system initialization
261	1	Clock out enable	From pins	0x10 = MCU_CLKOUT enable 0x11 = MCU_CLKOUT disable
262	1	Clk out freq	From pins	0x20 = MCU_CLKOUT 25 MHz (RGMII) 0x21 = MCU_CLKOUT 50 MHz (RMII)
263	1	RMII Clk In	From pins	0x60 = RMII internal (SoC) clock 0x61 = RMII external clock
264	1	Port Enable	0x01	Bit map. A set bit indicates that the corresponding physical port will be enabled
265	1	Phy Query	From pins	0x30 = speed/duplex determined from RGMII status register 0x31 = MDIO used to query PHY 0x32 = Use fixed speed/duplex values from offset 266/267.
266	1	Speed		0x40 = full speed (1Gbit for RGMII, 100Mbit for RMII) 0x41 = slow speed (100Mbit for RGMII, 10Mbit for RMII)
267	1	Duplex		0x50 = full duplex 0x51 = half duplex
Main Level Boot Control				
268	1	Bootp enable	1	0 = Image information already in this structure 1 = Use BOOTP to get boot image information
269	1	Reserved	0	Reserved
270	2	Bootp Timeout	4000	BOOTP timeout in milli-seconds
272	2	TFTP timeout	1000	TFTP timeout in milli-seconds
274	2	Bootp retries	10	Number of BOOTP retries before fail
276	2	TFTP retries	10	Number of TFTP retries before fail
278	2	Reserved	0	Reserved
Network Stack Configuration (plus TFTP server ID)				

Copyright © 2024 Texas Instruments Incorporated

Table 4-64. Ethernet Boot Parameter Table (continued)

Byte Offset	Size (bytes)	Name	Default Value	Description
280	6	MAC Address	From E-fuse	MAC address of the device
286	2	Reserved	0	Reserved
288	4	Device IP	0	IP address of the device. Valid only if BOOTP not enabled
292	4	Net Mask	0	Net mask. Valid only if BOOTP not enabled
296	4	Tftp server IP	0	TFTP server IP. Valid only if BOOTP not enabled
BOOTP send and receive Information				
300	20	Vendor String	"TI K3 Bootp Boot"	BOOTP request vendor string. Valid only if BOOTP not enabled
320	9	Client ID	1-mac-address-0	Client ID. See RFC1700
329	1	ID len	7	Client ID length
330	45	Debug array	SOC ID up to size available	Debug array output
375	1	Debug len	Varies	The number of valid bytes in debug array
376	4	Next hop	0	Next hop IP address. Valid only if bootp not enabled
380	4	Default Route	0	IP default route IP address. Valid only if bootp not enabled
384	128	Boot filename	0	Boot filename. Valid only if bootp not enabled

4.4.7 USB Boot Parameter Table

Table 4-65 shows the boot parameter table for USB boot. Must be preceded with the common boot parameters described in Table 4-54.

Table 4-65. USB Boot Parameter Table

Byte Offset	Size (bytes)	Name	Default Value	Description
256	4	Port	From pins	Physical port number. Always set to 0.
260	4	Reseved	0	Reseved
264	4	Base address 0	From pins (port)	Base address of USB subsystem (CMN)
268	4	Base address 1	From pins (port)	Base address of controller
272	4	phyBaseAddress	From pins (port)	Base address of USB PHY module
276	4	modRefClkkHz	varies	Module reference clock, in kHz
280	4	Vendor ID	0x0451	USB vendor ID. Read from control registers.
284	4	Product ID	0x6163	USB product ID. Read from control registers.
288	4	String Table addr	0x4182BCA8	Pointer to the string table in RAM
292	2	Vendor String offset	0	Offset to vendor string in string table
298	2	Prod string offset	32	Offset to product string in string table
300	2	Serial num string offset	64	Offset to serial number string in string table
302	2	Timeout	5000	USB timeout in milliseconds
304	2	Mode	1	1 = DFU ≠1 = Reserved
305	1	Lane reverse	0	Host mode only. Boot file name in Unicode characters
306	2	Reseved	0	Reseved
308	4	Block size	4096	DFU block size

4.4.8 MMCSD Boot Parameter Table

Table 4-66 shows the boot parameter table for eMMC, MMC or SD card boot. Must be preceded with the common boot parameters described in Table 4-54.

Copyright © 2024 Texas Instruments Incorporated

Table 4-66. MMCSD Boot Parameter Table

Byte Offset	Size (bytes)	Name	Default Value	Description
256	1	Port	From Pins	Physical port number
257	1	eMMC	From Pins	0 = SD/MMC, else eMMC
258	1	bootAck	1 for eMMC 0 for SD/MMC	If 1 and in eMMC mode the controller expects a boot ack from the eMMC
259	1	Media	1 for eMMC 0 for SD/MMC	0 = auto detect card type 1 = MMC 2 = SD
260	1	busWidth	8 for eMMC From Pins for SD/MMC	Number of data pins (1, 4, or 8). If 0, max supported pins of the port are used.
261	1	bootAlt	1 for eMMC 0 for SD/MMC	If 1 and in eMMC mode, the controller uses the alt boot method (CMD1 with arg 0xFFFF_FFFA) to initiate data transfer
262	1	sigVolt	0x18 for port0 0x33 for port1	Sets the signal voltage for the controller. 0x18 = 1.8V 0x33 = 3.3V
263	1	Reserved	0	Reserved
264	4	Max Bus Freq	0	Max bus frequency in kHz. If 0, the value is determined by reading the CSD register from the card (still maxes out at 25 MHz)
268	4	refClkkHz	0	Module reference clock frequency, in kHz. If 0, the ROM code computes the value.
272	4	respTimeout	40000	The timeout period on initial card read, in milli-seconds, Q3 number format
276	128	Filename	"\tiboot3.bin"	For SD/MMC, boot filename in 16-bit Unicode characters (max 64)
404	4	Mode	0x1144D091	0x1144D091 = file system boot 0x1144C180 = raw image boot
408	4	CardIsInit	0	0 = card not yet initialized 1 = card has been initialized
412	4	Rsvd	0	Reserved
416	4	RawIndex	0	Current active read offset (0 or 1)
420	4	Rsvd	0	Reserved
424	4	Raw Offset 0	0x000000	Raw read offset
428	4	Raw Offset 1	0x400000	Backup raw read offset
432	4	Rsvd	0	Reserved
436	4	Rsvd	0	Reserved

4.4.9 UART Boot Parameter Table

Table 4-67 shows the boot parameter table for UART boot. Must be preceded with the common boot parameters described in Table 4-54.

Table 4-67. UART Boot Parameter Table

Byte Offset	Size (bytes)	Name	Default Value	Description
256	1	Magic	0x49	Required Magic value
257	1	Protocol	63 (0x3F)	Specifies the transfer protocol = XMODEM
258	2	Reserved	0	Reserved
260	1	Max error Count	10	Error count resulting in boot abort
261	1	Ack timeout	3	Timeout in seconds on ack
262	1	Char timeout	20	Inter-character timeout in milliseconds
263	1	Reserved	0	Reserved

Copyright © 2024 Texas Instruments Incorporated

Table 4-67. UART Boot Parameter Table (continued)

Byte Offset	Size (bytes)	Name	Default Value	Description
264	4	Port	From Pins	Physical port number
268	4	Mod Ref Clk	48000	Module reference clock, in kHz
272	4	Data Rate	115200	Baud rate (bps)
276	1	Parity	0	0=none, 1=odd, 2=even
277	1	Data bits	8	Only 8 data bit width is supported
278	1	Stop bits	2	Stop bits in Q7.1 format (2 = 1 stop bit)
279	1	Flow Control	0	0=none, 1= RTS/CTS
280	1	Over sample	16	Only 16× and 13× oversample are supported
281	1	Magic 2	0xB7	Required magic value

4.4.10 Hyperflash Boot Parameter Table

The configuration table for Hyperflash is shown in Table 4-68. Must be preceded with the common boot parameters described in Table 4-54.

Table 4-68. Hyperflash Boot Parameter Table

Byte Offset	Size (bytes)	Name	Default Value	Description
256	1	Port	0	Physical port
257	1	Mode	0	0 = DDR, otherwise SDR
258	1	Csel	From pins	Chip select
259	1	nCsel	From pins	Number of valid chip selects
260	4	Rsvd	0	Reserved
264	2	RefFreqkHz	0	Module reference clock in kHz. 0 = ROM code computes the value
266	2	Rsvd	0	Reserved
268	4	busFreqkHz	From pins	Hyperflash bus frequency in kHz (83333 or 166666 kHz)
272	4	Base Address CS0	0	Offset to CS0 memory
276	4	Base Address CS1	0	Offset to CS1 memory
280	4	readIndex	0	Current active read offset (0 or 1)
284	4	Read offset 0	0x000000	Read offset 0
288	4	Read offset 1	0x400000	Read offset 1
292	4	Reserved	0	Reserved
296	4	Reserved	0	Reserved

4.4.11 Serial NAND Boot Parameter Table

The configuration table for Serial NAND is shown in Table 4-69. Must be preceded with the common boot parameters described in Table 4-54.

Table 4-69. Serial NAND Boot Parameter Table

Byte Offset	Size (bytes)	Name	Default Value	Description
256	1	port	0	Physical port number
257	1	modeOn	0	If non-zero the mode byte will be sent after commands
258	1	instructWidth	1	The number of pins used to send instructions (1)
259	1	addressWidth	1	The number of pins used to send data (1)
260	1	dataWidth	Based on Read Mode selection	The number of pins used to read data (1,4,8)
261	1	pageAddrSize	24	24 bit page address size supported.
262	1	addrSize	16	16 bit address sizes supported.

Copyright © 2024 Texas Instruments Incorporated

Table 4-69. Serial NAND Boot Parameter Table (continued)

Byte Offset	Size (bytes)	Name	Default Value	Description
263	1	mode	0	The SERIAL NAND clock polarity and phase value
264	1	csel	0	The chip select value (0-n)
265	1	pageReadCmd	0x13	The page read command
266	1	readCmd	Based on Read Mode selection	The read command
267	1	modeByte	0	The mode byte value
268	1	dummyCycles	8	Number of dummy cycles sent after the read command
269	1	clkRecovery	1	Phy, tap or no clock recovery
270	1	dqsEnable	0	If non-zero DQS is driven into RX DLL, not internal clock
271	1	ddrEnable	0	DDR Enable ment bit: 0 (default) DDR disabled, 1 DDR enabled
273	2	refFreq10kHz	0	The SERIAL NAND module reference frequency in 10kHz units. ROM code computes the value
277	4	busFreqkHz	50000	The SERIAL NAND bus frequency in kHz
281	4	cselReadDelays	0	The chip select read delay (placed directly into the register). Computed based on Bus Frequency
285	4	tapDelay	0xffffffff	The read tap selection. The value DRIVERS_SERIAL_NAND_AUTO_TAP_SELECT will cause the ROM to scan for best tap
286	1	readMode1	From pins	Valid only if readMode2 is 0. Value 0 for readMode1 represents 1S-1S-8S mode and 1 represents 1S-1S-4S mode
287	1	readMode2	From pins	Read mode is taken from readMode1 when readMode2 is 0, means 0 is reserved. Value 1 for readMode2 represents 1S-1S-1S mode
288	1	readMode	From pins	Read mode as derived from read mode 1 and read mode 2
292	4	pageSize	Calculated at run time	Number of bytes per page
296	4	pagesPerBlock	Calculated at run time	Number of pages in a block
300	4	blocksPerUnit	Calculated at run time	Number of blocks in a logical unit
304	4	numLogicalUnits	Calculated at run time	Number of logical units

Copyright © 2024 Texas Instruments Incorporated

4.5 Boot Image Format

4.5.1 Overall Structure

The boot image consists of an X.509 Certificate, followed immediately by a boot image blob.

X.509 Certificate (Variable Size) (Optional)
Boot Image Blob (Variable Size)

4.5.2 X.509 Certificate

The X.509 certificate is described in RFC5280. Section 4.1 of the specification describes the format.

The X.509 fields relevant to the public boot (taken from RFC5280) are shown below.

```
Certificate  ::=  SEQUENCE  {
        tbsCertificate       TBSCertificate,
        signatureAlgorithm   AlgorithmIdentifier,
        signatureValue       BIT STRING  }
   TBSCertificate  ::=  SEQUENCE  {
        version         [0]  EXPLICIT Version DEFAULT v1,
        serialNumber         CertificateSerialNumber,
        signature            AlgorithmIdentifier,
        issuer               Name,
        validity             Validity,
        subject              Name,
        subjectPublicKeyInfo SubjectPublicKeyInfo,
        issuerUniqueID  [1]  IMPLICIT UniqueIdentifier OPTIONAL,
                             -- If present, version MUST be v2 or v3
        subjectUniqueID [2]  IMPLICIT UniqueIdentifier OPTIONAL,
                             -- If present, version MUST be v2 or v3
        extensions      [3]  EXPLICIT Extensions OPTIONAL
                             -- If present, version MUST be v3
        }
  Extensions  ::=  SEQUENCE SIZE (1..MAX) OF Extension
   Extension  ::=  SEQUENCE  {
        extnID      OBJECT IDENTIFIER,
        critical    BOOLEAN DEFAULT FALSE,
        extnValue   OCTET STRING
                    -- contains the DER encoding of an ASN.1 value
                    -- corresponding to the extension type identified
                    -- by extnID
        }
```

In general, an X.509 certificate contains a public key which has been signed by a private key. The public ROM code does not directly use the keys. In non-secure devices, the public key value is in general a don't care condition. The exception is certificates containing a degenerate RSA public key. GP devices with a degenerate RSA key allow for integrity checking of most (but not all) of the certificate.

The public MCU ROM only needs to extract some of information from the X.509 formatted structure:

- The total size of the X.509 Certificate
- The total size of the boot image

The total size of the X.509 Certificate is determined by reading the length of the sequence containing the certificate. The length of the image is determined by parsing the certificate to find the extension field which holds the image length.

The ROM defines several extensions that are used only by TI for boot. These are placed in the extensions field of the TBS certificate.

4.5.3 Organizational Identifier (OID)

OID (organizational identifier) values are represented as a tree structure. TI has the following node registered:

1.3.6.1.4.1.294: iso(1), identified-organization(3), dod(6), internet(1), private(4), enterprise(1), Texas Instruments(294)

Copyright © 2024 Texas Instruments Incorporated

ROM code adds the following branch after *Texas Instruments*: *device-boot(1)*. The OID values shown in this section are leaves off the device boot branch.

4.5.4 X.509 Extensions Specific to Boot

These values are not defined in any standard, but created by TI for boot.

4.5.4.1 Boot Info (OID 1.3.6.1.4.1.294.1.1)

This extension must be present on all boot images. It is from this extension that the image length is extracted.

```
bootInfo ::=  SEQUENCE {
       cert_type: INTEGER,        -- identifies the certificate type
       boot_core:    INTEGER,        -- identifies the boot core
       core_opts:    INTEGER,        -- 32 or 64 bit boot core target
       load_addr:    OCTET STRING,    -- Global address image destination
       image_size:    INTEGER,        -- Image size in bytes
}
```

Table 4-70. Certificate Type Values

Value	Description
0x0000_0001	Primary boot image
0x0000_0002	Firmware image

Table 4-71. Boot Core Values

Value	Description
0x00	Firmware (DMSC) image
0x08	DMSC certificate
0x10	MCU image
0x20	Reserved

Table 4-72. Core Options Bit Fields

31	2	1	0
Reserved		Split	Mode

Table 4-73. Core Options Field Description

Bits	Field	Value	Description
1	Split	0	Dual MCU set to lockstep (two cores in lockstep)
		1	Dual MCU set to split mode (two independent cores)
0	Mode	0	MCU starts execution in Arm® mode
		1	MCU starts execution in Thumb® mode

4.5.4.2 Image Integrity (OID 1.3.6.1.4.1.294.1.2)

```
imageIntegrity ::= SEQUENCE {
       sha_type:    OID,            -- Identifies the SHA type
       hash:        OCTET STRING    -- The SHA of the boot image
}
```

4.5.5 Extended Boot Info Extension

J7AEP ROM supports a combined boot image boot flow. In this flow, a boot binary blob has both Secondary bootloader (SBL) and System Firmware (SYS-FW) embedded in the boot image with a single X509 certificate. This method helps with the following situations:

- Allows ROM to load and run both the bootloader and SYS-FW in parallel without any dependency.
- Optimizes ROM boot time by minimizing different x509 certificate parsing and authentication.

To support this combined boot format, ROM employs a new X509 extension called: ext_boot_info. It supports multiple boot components with a single certificate. It allows up to 5 components as part of this extension:

Copyright © 2024 Texas Instruments Incorporated

- Component1: Mandatory and should point to info about SBL binary
- Component2: Optional and if present should point to SYS-FW binary in all device types
- Component3: Optional (Load section to SBL, new certType)
- Component4: Optional (Load section to SYS-FW, new certType)
- Component5:
  - HS-FS and HS-SE non Prime devices. Mandatory and should point to SYS-FW Inner certificate
  - GP and HS-SE Prime devices. Optional (Load Section to SBL or SYS-FW)

This extended boot info extension replaces boot_seq (boot_info) and image_integrity extensions from the previous sections, and these should be exclusive in any given certificate.

ROM supports other extensions, such as sw_rev and debug_info, in both formats.

ROM selects the combined image flow based on the presence of ext_boot_info extension in the certificate and skips boot_seq (boot_info) and image_integrity extensions boot flow.

Having one component in ext_boot_info is same as legacy flow (that is, flow using boot_seq and image_integrity); for two or more components, ROM starts both SBL and SYS-SW, the third and fourth components are loaded by ROM to the allowed loading memory range of SBL and SYS-FW if there is no overlap in load address with executable binary info.

Each of the components can independently specify hash value of the binary, and ROM validates the hash on HS and GP if RSA degenerate key is used for signing.

Additionally, ROM rejects the full image if hash of any single component mismatches.

4.5.5.1 Impact on HS Device

For HS devices, ROM supports encryption of images optionally when specified with the Encryption extenstion.

Encryption - Encryption of the components must be specified with a new extension called ext_boot_enc. This is an optional extension that must be specified only if any of the components are encrypted with a customer key set. Specify a component number followed by encryption extension details for ROM to decrypt the given components. This extension is used only with combined boot image format (that is, using Extended Boot Info Extension).

Prime vs Non-Prime - The main difference between HS prime and non-prime devices is the presence of the SYS-FW Inner certificate. Extended boot info extension supports both prime and non-prime devices. ROM supports SYS-FW binary as Comp#2 in all device types; for HS-SE non-prime and HS-FS devices, the SYS-FW inner certificate is expected as another optional component in #3, 4, or 5.

4.5.5.2 Extended Boot Info Details

```
1.3.6.1.4.1.294.1.9=ASN1:SEQUENCE:ext_boot_info
[ ext_boot_info ]
 extImgSize = INTEGER:470656
 numComp = INTEGER:4
 sbl=SEQUENCE:comp1
 fw=SEQUENCE:comp2
 bd1=SEQUENCE:comp3
 bd2=SEQUENCE:comp4

[ comp1 ]
 compType = INTEGER:1
 bootCore = INTEGER:16
 compOpts = INTEGER:0
 destAddr = FORMAT:HEX,OCT:41c00000
 compSize = INTEGER:237376
 shaType = OID:2.16.840.1.101.3.4.2.3
 shaValue =
FORMAT:HEX,OCT:6779fdb2b2c27169737c184085b97938bd77bdf698245840f166ca30c7125c29a6675139a25a0a2a3f00a
76d43d082df238c12cb6b293ec0eeb5990bcd603a23

[ comp2 ]
 compType = INTEGER:2
 bootCore = INTEGER:0
 compOpts = INTEGER:0
 destAddr = FORMAT:HEX,OCT:00040000
```


Copyright © 2024 Texas Instruments Incorporated

```
 compSize = INTEGER:196608
 shaType = OID:2.16.840.1.101.3.4.2.3
 shaValue =
FORMAT:HEX,OCT:47f27fb24b81927a928845a4c2b993e6a3d5bdfe5ed01c0ec4f96a7ead991c69bf4ed4b9b958fd36a75f3
3aba04d2c28602a85ca737ee75617d6a9a41f4353a3

[ comp3 ]
 compType = INTEGER:18
 bootCore = INTEGER:0
 compOpts = INTEGER:0
 destAddr = FORMAT:HEX,OCT:00072000
 compSize = INTEGER:16384
 shaType = OID:2.16.840.1.101.3.4.2.3
 shaValue =
FORMAT:HEX,OCT:2d165ec1d8a38acb977d9298e8a6a27491c62d6daa31f921db9135f9a68779b30b384573c6e4e8203de5f
0a47191c3ff9a35b52a911874e07615f10b4b2f5829

[ comp4 ]
 compType = INTEGER:17
 bootCore = INTEGER:16
 compOpts = INTEGER:0
 destAddr = FORMAT:HEX,OCT:41c40000
 compSize = INTEGER:20288
 shaType = OID:2.16.840.1.101.3.4.2.3
 shaValue =
FORMAT:HEX,OCT:12d5be5b2b9774d1b0cad21cbf1dbcdbd7c310657f4334902e3cc6228cf2d8fc844139dae4db6041c38cd
a502d6a900d57039322d360032268a13445021b04a7
```

4.5.5.3 Certificate / Component Types

ROM supports only the following component types; all other component types will be rejected.

- CompType 0x1 for SBL binary - compType = INTEGER:1
- CompType 0x2 for SYS_FW binary - compType = INTEGER:2
- CompType 0x3 for SYS_FW Inner Certificate - compType = INTEGER:3
- CompType 0x11 for SBL Memory load section - compType = INTEGER:17

Primary Image load section, that can be loaded in to SBL allowed memory range:

- CompType 0x12 for SYS_FW Memory load section - compType = INTEGER:18

SYS-FW load section, that can be loaded in to SYS-FW allowed memory range

4.5.5.4 Extended Boot Encryption Info

Extended Boot Encryption Info specifies the imageEncryption extension per component basis. This extension is not applicable for GP or HS-FS devices, and is optional for HS-SE devices if any components in the image are encrypted. In HS-FS and HS-SE non-prime devices, the SYS-FW binary encryption details are part of SYS-FW Inner certificate. This extension is applicable if either SBL, SYS-FW binary in HS-SE prime, or any binary blobs are encrypted in HS-SE devices.

```
1.3.6.1.4.1.294.1.10=ASN1:SEQUENCE:ext_enc_info

[ ext_enc_info ]
 numComp = INTEGER:2
 esbl=SEQUENCE:enc1
 efw=SEQUENCE:enc2

[ enc1 ]
 compNum = INTEGER:1
 iv = FORMAT:HEX,OCT:474bfd801866beecc7ab6d4c61490e1a
 randString = FORMAT:HEX,OCT:772fc5810fa36f516e595ad8adf19260f47a8461f193892746692fbb932727a1
 iterationCnt = INTEGER:0
 salt = FORMAT:HEX,OCT:42ea40851298339c8baa84f29d6b68d0

[ enc2 ]
 compNum = INTEGER:2
 iv = FORMAT:HEX,OCT:aaaaaaaaaaaabeecc7ab6d4c61490e1a
 randString = FORMAT:HEX,OCT:bbbbbbbbbbbbbbbbbe595ad8adf19260f47a8461f193892746692fbb932727a1
 iterationCnt = INTEGER:2
 salt = FORMAT:HEX,OCT:cccccccccccccccccccaa84f29d6b68d0
```


Copyright © 2024 Texas Instruments Incorporated

4.5.5.5 Component Ordering

ROM supports fixed Component ordering for SBL- Binary and SYS-FW Cert and Binary.

For GP and HS-SE Prime devices, Comp#1 should be SBL binary, and Comp#2 should be SYS-FW binary.

For HS-FS and HS-SE non-Prime devices, Comp#1 should be SBL binary, Comp#2 should be SYS-FW Inner Certificate, and Comp#3 should be SYS-FW binary.

4.5.5.6 Memory Load Sections Overlap with Executable Components

Memory load sections for SBL and SYS-FW (CompType 0x11 and 0x12) the destination address and size should not overlap with the corresponding load and execute sections, and should also fall in the allowed memory range by ROM.

4.5.5.7 Device Type and Extended Boot Extension

X509 Extension	Component	GP Device	HS-FS	HS-SE non-prime	HS-SE Prime Device
ext_boot_info	Comp#1	SBL Binary	SBL Binary	SBL Binary	SBL Binary
	Comp#2	SYS-FW binary	SYS-FW binary	SYS-FW binary	SYS-FW binary
	Comp#3	SBL mem load section	SBL mem load section	SBL mem load section	SBL mem load section
	Comp#4	SysFW mem load section	SysFW mem load section	SysFW mem load section	SysFW mem load section
	Comp#5	N/A	SysFW inner certificate	SysFW inner certificate	N/A
ext_enc_info	Comp#1	N/A	N/A[1]	SBL encryption	SBL encryption
	Comp#2			N/A	SYS FW encryption
	Comp#3			SBL memory load Encryption	SBL memory load encryption
	Comp#4			Sys-FW memory load encryption	Sys-FW memory load encryption
	Comp#5			N/A[1]	NA

1. SYS-FW encryption in HS-FS and HS-SE (non Prime) are handled in SYS-FW Inner certificate and they are not part of the extended boot info extensions.

4.5.6 Generating X.509 Certificates

X.509 Certificates are generated using OpenSSL and a configuration script to supply values in the extension fields.

4.5.6.1 Key Generation

The SBL must always be signed with a given OpenSSL key - Secure and GP devices. Secure must have encryption and authentication, GP device must only have authentication. The key used for authentication can be random or specific. If a random key is generated and SBL is signed with this,the ROM will copy the SBL image for authentication using memcopy. With this key, ROM code will be directed to use DMA to load the SBL for authentication which saves boot time.

4.5.6.1.1 Degenerate RSA Keys

Degenerate RSA keys are valid RSA keys with the private exponent set to 1. This results in the signature field being equal to the digest, since in RSA:

$$\text{Signature} = \text{digestprivExp mod nprivExp mod } n^{\text{privExp}} \quad (1)$$

Where n is the key size. Since the hash used is SHA-512 and the signature is an ASN.1 sequence containing the OID defining which has was used as well as the hash value, the degenerate RSA must have a value of n greater than the maximum digest size. Typically 1024-bit is chosen.

The following sequence is used to generate degenerate RSA keys:

Copyright © 2024 Texas Instruments Incorporated

1. Create a random RSA key:

```
openssl genrsa –out key.pem 1024
```

2. Convert to text:

```
openssl rsa –in key.pem –text –noout > key.txt
```

3. Create an asn1 template for the degenerate key called degenerateKey.txt. Simply copy the values for modulus, prime (listed as p in key.txt), prime 2 (listed as q), and coefficient (listed as coeff). Set the public and private key exponents to 1, as well as the values for e1 and e2. See the example below.
4. Convert the template to DER:

```
openssl asn1parse -genconf degenerateKey.txt -out degenerateKey.der
```

5. Sanity check the key:

```
 openssl rsa –in degenerateKey.der –inform der -text -check
```

6. If there are no errors create the degenerate key pem file:

```
openssl rsa –in degenerateKey.der –inform der –outform pem -out degenerateKey.pem
```

An example degenerateKey.txt file is shown.

```
asn1=SEQUENCE:rsa_key
[rsa_key]
version=INTEGER:0
modulus=INTEGER<copied from key.txt>
pubExp=INTEGER:1
privExp=INTEGER:1
p=INTEGER:<copied from key.txt>
q=INTEGER<copied from key.txt>
e1=INTEGER:1
e2=INTEGER:1
coeff=INTEGER<copied from key.txt>
```

Note that when copying the multi-byte fields from key.txt it is necessary to remove the colons, catenate the lines and add a preceding 0x.

4.5.6.2 Configuration Script

An example openssl configuration script is shown below. Not all extensions are required, but all possible are shown.

```
[ req ]
distinguished_name = req_distinguished_name
X509_extensions = v3_ca
prompt = no
dirstring_type = nobmp
[ req_distinguished_name ]
C = GB
ST = HI
L = Boston
O = Texas Instruments., Inc.
OU = DSP
CN = Bob
emailAddress = Bob@hou.ti.com
[ v3_ca ]
basicConstraints = CA:true
1.3.6.1.4.1.294.1.1 = ASN1:SEQUENCE:boot_seq
1.3.6.1.4.1.294.1.2 = ASN1:SEQUENCE:image_integrity
1.3.6.1.4.1.294.1.3 = ASN1:SEQUENCE:swrv
1.3.6.1.4.1.294.1.4 = ASN1:SEQUENCE:encryption
1.3.6.1.4.1.294.1.5 = ASN1:SEQUENCE:key_derivation
1.3.6.1.4.1.294.1.7 = ANSI:SEQUENCE:pllControl
1.3.6.1.4.1.294.1.8 = ANSI:SEQUENCE:debug
[ boot_seq ]
certType =    INTEGER:1
bootCore = INTEGER:16
bootArchWidth = INTEGER:32
destAddr = FORMAT:HEX,OCT:bc934b00
```


Copyright © 2024 Texas Instruments Incorporated

```
imageSize = INTEGER:0x00004860
[ image_integrity ]
shaType = OID:1.3.14.3.2.26
shaValue = FORMAT:HEX,OCT:4cf4d59ef77b5d9ab28d2ceb3c9fe83cb52ae6d2
[ swrv ]
rollback = INTEGER:0x00010001
[ encryption ]
Iv =    FORMAT:HEX,OCT:00112233445566778899aabbccddeeff
Rstring = FORMAT:HEX,OCT:00112233445566778899aabbccddeeff101112131415161718191a1b1c1d1e1f
Icount = INTEGER:1
Salt = FORMAT:HEX,OCT:00112233445566778899aabbccddeeff
[ pllControl ]
pll0_num = INTEGER:0
pll0_cfg = FORMAT:HEX,OCT:00345678900
pll1_num = INTEGER:1
pll0_cfg = FORMAT:HEX,OCT:00345678900
[ debug ]
uid = FORMAT:HEX,OCT:00345678900
type = INTEGER:1
dbgE = INTEGER:0
secDbgEn = INTEGER:0
```

The certificate is then generated using the following openssl command:

```
openssl req -new -x509 -key <private_key_pem_file> -nodes -out <output_X.509_pem_file> -config
<config_file> -sha512
```

If a delegate key is being signed, then add the option -signkey <sign_key_pem_file> to the command above.

4.5.7 Image Data

The image data (blob) is considered simply as a byte stream. On devices that are multiple bytes wide (for example, PCIe) the image must be formatted so that all multi-byte fields match the endianness of the device. The MCU will always run in little endian mode.

Copyright © 2024 Texas Instruments Incorporated

4.6 Boot Modes

4.6.1 I2C Bootloader Operation

4.6.1.1 I2C Initialization Process

In the I2C boot mode, the ROM Code configures the MCU_I2C0 peripheral.

MCU_I2C0 can operate either in master mode or in slave mode. In the master mode, the boot master drives the I^2C slave device where the image is stored. In the slave mode, the data transfer is driven by an external I^2C master device. The master device is usually another SoC device or a FPGA.

A detailed summary of the I^2C boot parameter table and the BOOTMODE pins definitions are listed in Section 4.4.4, *I2C Boot Parameter Table* and Section 4.3.9, *I2C Boot Device Configuration*.

4.6.1.1.1 Block Size

ROM code will read 0x800 bytes before processing the data. The last block must be padded to the next 0x800 byte boundary and it must be padded with zeros.

4.6.1.1.2

The boot code does not support byte address to bus address wrapping. So the maximum image size that can be access is 64 kbytes. What this means is that if the read address is 0xFF00, the read size is 0x200 and the I^2C bus address is 0x50, then 0x100 bytes will be read from 0xFF00 at bus address 0x50, followed by 0x100 bytes from 0x0000 at bus address 0x50. The bus address does not increment when the address rolls over.

4.6.1.2 I2C Loading Process

4.6.1.2.1 Loading a Boot Image From EEPROM

In this mode, the MCU_I2C0 peripheral is configured as I^2C master.

ROM Code will start reading from the I^2C EEPROM at the specified I^2C bus address. This read will be done beginning at the specified base address offset. Data will be read in 2-KB chunks. The data will be stored at the address specified in the boot header. It will continue reading image data until a complete image has been read. When the complete image has been read, the ROM Code will branch to the start address of the image.

4.6.2 SPI Bootloader Operation

4.6.2.1 SPI Initialization Process

In the SPI boot mode, the ROM Code initializes the OSPI peripheral to SPI bus at 4.156 MHz. The image is read from the EEPROM connected to the corresponding OSPI port and chip-select. The the starting offset to be read can be provided by the user through the BOOTMODE pins (Section 4.3.7, *SPI Boot Device Configuration*). A detailed summary of the SPI boot parameter table and the boot configuration definitions are listed in Section 4.4.5, *OSPI/QSPI/SPI Boot Parameter Table*.

4.6.2.2 SPI Loading Process

The SPI boot loading process is similar to the loading process explained in Section 4.6.1.2.1, *Loading a Boot Image From EEPROM*, except that the image is read from a NOR flash. The data formats supported are also the same as in Section 4.6.1.2, *I2C Loading Process*. All the other loading processes that are detailed for I^2C master mode work for the SPI boot mode, too.

4.6.3 QSPI Bootloader Operation

4.6.3.1 QSPI Initialization Process

In the QSPI boot mode, the ROM Code initializes the OSPI peripheral in DAC mode. The image is read from the QSPI flash connected to the selected chip-select. See Section 4.3.6, *QSPI Boot Device Configuration* for OSPI port settings.

A detailed summary of the QSPI boot parameter table and the boot configuration definitions are listed in Section 4.4.5, *QSPI Boot Parameter Table*.

4.6.3.2 QSPI Loading Process

QSPI boot mode is not executable-in-place (XIP). ROM code first copies boot image into on-chip RAM and then executes it.

Copyright © 2024 Texas Instruments Incorporated

4.6.4 OSPI Bootloader Operation

4.6.4.1 OSPI Initialization Process

In the OSPI boot mode, the ROM Code initializes the OSPI peripheral in DAC mode. The image is read from the OSPI flash connected to the selected chip-select. See Section 4.3.5, *OSPI Boot Device Configuration* for OSPI port settings.

A detailed summary of the OSPI boot parameter table and the boot configuration definitions are listed in Section 4.4.5, *OSPI Boot Parameter Table*.

4.6.4.2 OSPI Loading Process

OSPI boot mode is not executable-in-place (XIP). ROM code first copies boot image into on-chip RAM and then executes it.

4.6.5 PCIe Bootloader Operation

4.6.5.1 PCIe Initialization Process

In the PCIe boot mode, the ROM Code configures the PCIe and SerDes from information it obtains from the boot parameter table, see Section 4.4.3, *PCIe Boot Parameter Table*

4.6.5.2 PCIe Loading Process

PCIe boot mode is not executable-in-place (XIP). ROM code first copies boot image into on-chip RAM and then executes it.

PCIe Boot Flow:

Once the PCIe link is up in J7AEP, R5 Rom waits for two specific checks to continue the Boot sequence.

The Host (root complex) after copying the full image has to write the Start address(32bit) of the Image on the address location (0x41CF3FE0).

The Host (root complex) after copying the Full image, it has to write a Magic Word (0xB17CEAD9) on the address location (0x41CF3FE4).

4.6.6 Ethernet Bootloader Operation

4.6.6.1 Ethernet Initialization Process

When the device is set to boot through the Ethernet mode, the ROM Code configures PHY, NAVSS, and the CPSW. These initial configurations, as well as the interface mode (RMII, RGMII), are determined by querying the BOOTMODE pins, see Section 4.3.12, *Ethernet Boot Device Configuration*, and the boot parameter table for the Ethernet boot, see Section 4.4.6, *Ethernet Boot Parameter Table*.

The queue manager is setup to route the packets to queues polled by the ROM Code.

4.6.6.2 Ethernet Loading Process

After device configuration, the bootloader performs a standard BOOTP/TFTP boot. The device sends a BOOTP request with its MAC address to a host TFTP server to be assigned an IP from a pool of addresses. After this connection is established, the device is able to receive image data encapsulated in Ethernet packets, see Section 4.6.6.2.1. Data received is stripped of its network headers and the boot data is stored in internal RAM. When the transfer completes and the image is found in good integrity, the ROM Code will branch to the address defined in the Boot Info field of the boot header.

4.6.6.2.1 Ethernet Boot Data Formats

Ethernet boot uses the BOOTP/TFTP protocol for downloads. Only IPv4 is supported.

4.6.6.2.1.1 Limitations

- Received packets cannot be IP fragmented (should not be a problem in most systems since the BOOTP/TFTP packets have fixed lengths of small size)
- Only DIX Ethernet headers are supported.
- 802.3 with SNAP/LLC not supported
- DIX Ethernet with VLAN not supported

Copyright © 2024 Texas Instruments Incorporated

- 802.3 with VLAN and SNAP/LLC not supported

4.6.6.2.1.2 BOOTP Request

4.6.6.2.1.2.1 MAC Header (DIX)

- Destination MAC = value from parameter table (default is broadcast)
- Source MAC = value from parameter table (default is e-fuse value)
- Type = IPv4 (0x0800)

4.6.6.2.1.2.2 IPv4 Header

- Version = 4
- Header length = 0
- TOS = 0
- Len = computed during operation
- ID = 0x0001
- Flags + Fragment offset = 0
- TTL = 0x10
- Protocol = UDP (17)
- Header checksum = Computed during operation
- SRC IP = 0.0.0.0
- Dest IP = 255.255.255.255

4.6.6.2.1.2.3 UDP Header

- Source port = BOOTP client (68 decimal)
- Destination Port = BOOTP server (67 decimal)
- Length = computed during operation
- Checksum = computed during operation

4.6.6.2.1.2.4 BOOTP Payload

- Opcode = Request (1)
- HW Type = Ethernet (1)
- HW Addr Len = 6
- Hop Count = 0
- Transaction ID = 1
- Number of seconds = 0
- Client IP = 0.0.0.0
- Your IP = 0.0.0.0
- Server IP = 0.0.0.0
- Gateway IP = 0.0.0.0
- Client HW Address = Device MAC address (from parameter table)
- Server hostname = NULL
- Filename = NULL
- Option 60, Vendor ID string, from parameter table
- Option 61, Client ID string, from parameter table

4.6.6.2.1.2.5 TFTP

There are no ROM code specific TFTP configurations.

4.6.6.3 Ethernet Hand Over Process

Once the ROM Code receives the valid packet, it decodes it to get the image sections and loads them in the appropriate memory location. After the image is loaded and validated, the ROM Code starts the boot image execution.

4.6.7 USB Bootloader Operation

The USB boot mode is used to read the boot image from external USB host .

See Section 4.3.13, *USB Boot Device Configuration* and Section 4.4.7, *USB Boot Parameter Table* for the available configuration options.

Copyright © 2024 Texas Instruments Incorporated

More information about USB DFU protocol can be found at http://www.usb.org/developers/docs/devclass_docs/DFU_1.1.pdf.

After successful enumeration, the ROM Code will start reading from the external host as specified by the BOOTMODE pins. It will continue reading data from the memory and storing it in internal RAM until a complete image has been read. When the complete image has been read and found in good integrity, the ROM Code will branch to the address defined in the Boot Info field of the boot header.

4.6.7.1 USB-Specific Attributes

4.6.7.1.1 DFU Device Mode

- Vendor ID = 0x451
- Product ID = 0x6168
- Device ID = 0

4.6.8 MMCSD Bootloader Operation

Only single data rate with backward compatible interface timing is supported.

See Section 4.3.10, *MMCSD Boot Device Configuration* and Section 4.4.8, *MMCSD Boot Parameter Table* for the available configuration options.

If the card type is set to autodetect, then an initial discovery phase is performed:

1. Send CMD 0. (GO IDLE, both for MMC and SD)
2. Send CMD 55. If the card responds then the card type is assumed to be SD and the boot attempt fails (the device do not support SD boot). On timeout the code sends CMD 1. If there is a response then the card type is MMC.

The ROM Code will start reading from the MMCSD memory boot sector, or filesystem, as specified by the BOOTMODE pins. It will continue reading data from the memory and storing it in internal RAM until a complete image has been read. When the complete image has been read and found in good integrity, the ROM Code will branch to the address defined in the Boot Info field of the boot header.

4.6.9 UART Bootloader Operation

4.6.9.1 Initialization Process

In the UART boot mode, the selected UART module (port) is the only peripheral configured. The baud rate, data, parity, and stop bits are configured based on the information in the UART boot parameter table. The boot parameter table definitions and the boot configuration values that can be set are in Section 4.3.15, *UART Boot Device Configuration* and Section 4.4.9, *UART Boot Parameter Table*.

Once the ROM Code configures the UART, it sends the UART pings for few seconds, which can be seen in the host. The pings consist of an ASCII capital *C* character. The UART boot mode supports only the CRC mode of XMODEM and does not support CHECKSUM mode.

4.6.9.2 UART Loading Process

Before the ping from the device stops, load the boot image from the host using the XMODEM protocol.

4.6.9.2.1 UART XMODEM

The XMODEM protocol is used to transfer boot data. Only CRC mode is supported (not checksum), with both 128- and 1024-byte block sizes. The general, format of received frames is shown in Table 4-74 and Table 4-75.

Table 4-74. XMODEM 1024- and 128-byte Data Frames

STX	Block Num	Inv Block Num	1024 data bytes			CRC	CRC
SOH	Block Num	Inv Block Num	128 data bytes	CRC	CRC		

Table 4-75. XMODEM Data Frame Fields

Field	Value	Description
STX	0x02	The start character for 1024-byte CRC data blocks

Copyright © 2024 Texas Instruments Incorporated

Table 4-75. XMODEM Data Frame Fields (continued)

Field	Value	Description
SOH	0x01	The start character for 128-byte CRC data block
Block Num	0x01-0xFF – 0x00	The block number. The first block has value 1, and the block number wraps around 0xFF to 0
Inv Block Num	0xFE-0x00	The inverse block number (bit inverse of the block number)
CRC	Calculated	The 16-bit CRC generated from the polynomial 0x1021

The XMODEM protocol is implemented as a half-duplex protocol as shown in Table 4-76.

Table 4-76. Example of XMODEM Transfer protocol

Transmitter Sends		Receiver Sends
	←	Ping (‘C’)
Frame 1	→	
	←	ACK (or NACK)
Frame 2	→	
	←	ACK (or NACK)
EOT	→	
	←	ACK (or NACK)

4.6.9.3 UART Hand-Over Process

Once the complete image has been read and found in good integrity, the ROM Code will branch to the address defined in the Boot Info field of the boot header.

Copyright © 2024 Texas Instruments Incorporated

4.7 Boot Memory Maps

4.7.1 Memory Layout/MPU

Table 4-77 shows an overview of the MPU configuration. In the R5 MPU, higher numbered regions have priority, therefore in Table 4-77, where two regions overlap, the region on the right defines the memory attributes.

Table 4-77. Memory Layout/MPU

Memory Address	Regions			
0x0000_0000	Region 0 non-executable full access	Region 10		
0x0000_003F				
...				
0x4101_0000		Region 3 TCM User access	Region 4 TCM Priv access	
0x4101_0FFF				
0x4101_1000				
0x4101_7FFF				
...				
0x4180_0000		Region 1 ROM User access	Region 2 ROM Priv access	
0x4180_7FFF				
0x4180_8000				
0x4183_FFFF				
...				
0x41C0_0000		Region 5 RAM All access non-cacheable		
0x41CB_FFFF				
0x41CC_0000			Region 6 cacheable	
0x41CD_FFFF				
0x41CE_0000			Region 7 cacheable	
0x41CE_FFFF				
0x41CF_0000			Region 8 non-cacheable non-executable	
0x41CF_DFFF				
0x41CF_E000				Region 9 cacheable
0x41CF_FFFF				
...				
0xFFFF_FFFF				

4.7.2 Global Memory Addresses Used by ROM Code

The ROM code uses several global memory addresses that are useful for debugging. They are shown in Table 4-78.

Table 4-78. Global Memory Addresses

Group	Address	Size (bytes)	Content
Version	0x4182_FF80	64	ROM code version (stored in ROM)
Free MSRAM	0x41C0_0000	768k	Loadable space for SBL
Warning/Error Logs	0x41CF_C800	512	Warning entries in cold boot
Warning/Error Logs	0x41CF_CA00	512	Severe entries in cold boot
Warning/Error Logs	0x41CF_CC00	256	Critical entries in cold boot
Message Logs	0x41CF_C580	512	Circular message buffer
Message Logs	0x41CF_C780	20	Circular message context log
Warning/Error Logs	0x41CF_C000	512	Warning entries in warm boot
Warning/Error Logs	0x41CF_C200	512	Severe entries in warm boot
Warning/Error Logs	0x41CF_C400	256	Critical entries in warm boot
Warning/Error Logs	0x41CF_CD04	40	Cold boot log context

Copyright © 2024 Texas Instruments Incorporated

Table 4-78. Global Memory Addresses (continued)

Group	Address	Size (bytes)	Content
Warning/Error Logs	0x41CF_CD2C	40	Warm boot log context
Warning/Error Logs	0x41CF_CD00	4	Active context pointer (not index) to one of the cold or warm contexts
Trace	0x41CF_CD80	24	Boot trace context
Trace	0x41CF_CD98	1024	Boot trace entry buffers
Parameter Tables	0x41CF_DBFC	4	Parameter table index (currently active table, 0 or 1)
Parameter Tables	0x41CF_DC00	512	Parameter table 0
Parameter Tables	0x41CF_DE00	512	Parameter table 1

The ROM code version information is a structure shown in Table 4-79

Table 4-79. ROM Code Version

Field	Address	Size (bytes)	Value for PG1
Version Number	0x4182_FF80	4	0x00010001 (1.0.1)
Version Date	0x4182_FF84	8	"30/07/21"
Device Name	0x4182_FF8C	12	j7aep
Commit ID	0x4182_FF98	40	"ff0ea75aac317ea6042724fe775412bca006e697"

4.7.3 Memory Reserved by ROM Code

ROM code reserves all of TCM B (BTCM) and the upper memory address of on-chip RAM (MCU_MSRAM0). The initial section of MCU_MSRAM0 is available for the booted image. This is the only memory available for the boot image since neither MSMC nor TCM A (ATCM) is powered up or available for the boot image.

Table 4-80 shows ROM code memory usage.

Table 4-80. Memory Reserved By ROM Code

Memory	Address	ROM Code Usage
TCM B (BTCM)	0x0000 – 0x7FFF	Reserved by ROM Code
MCU_MSRAM0	0x41C0_0000 – 0x41CB_FFFF	Free for boot image
	0x41CC_0000 – 0x41CF_FFFF	Reserved by ROM Code

5 Device Configuration

5.1 Control Module (CTRL_MMR)

5.1.1 CTRL_MMR Overview

The CTRL_MMR modules on the device control top-level device behavior in various states. They contains registers for configuration, bootstrap signals, I/O pad multiplexing, clock selection, and many others. There are three CTRL_MMR modules in this device - CTRL_MMR0 which resides in MAIN domain, MCU_CTRL_MMR0 in MCU domain, and WKUP_CTRL_MMR0 in the WKUP domain.

There is also a PLL control module that has registers to control the configuration of the device PLLs.

Copyright © 2024 Texas Instruments Incorporated

5.1.2 CTRL_MMR Functional Description

The following sub-sections provide summary and description for most of the registers which are part of the CTRL_MMR0 module memory space. The rest of the registers are described in the corresponding chapters where the functionality associated with particular CTRL_MMR0 register is covered in more detail.

5.1.2.1 Register Partitions

The CTRL_MMR0 memory space is divided into the partitions shown in Table 5-1.

Table 5-1. CTRL_MMR0 Register Partitions

Partition(1)	Description	Proxy0 Offset Range	Proxy1 Offset Range
Partition 0	General configuration and IPC regs	0000h to 1FFFh	2000h to 3FFFh
Partition 1	IP control/status regs	4000h to 5FFFh	6000h to 7FFFh
Partition 2	Clock control/status regs	8000h to 9FFFh	A000h to BFFFh
Partition 3	BIST control/status regs	C000h to DFFFh	E000h to FFFFh
Partition 5	DDR frequency Control	1 4000h to 1 5FFFh	1 6000h to 1 7FFFh
Partition 7	MAIN I/O Pad Configuration regs	1 C000h to 1 DFFFh	1 E000h to 1 FFFFh

(1) Not listed partitions are reserved and not used.

The MCU_CTRL_MMR0 memory space is divided into the partitions shown in Table 5-2.

Table 5-2. MCU_CTRL_MMR0 Register Partitions

Partition(1)	Description	Proxy0 Offset Range	Proxy1 Offset Range
Partition 0	General configuration and IPC regs	0000h to 1FFFh	2000h to 3FFFh
Partition 1	IP control/status regs	4000h to 5FFFh	6000h to 7FFFh
Partition 2	Clock control/status regs	8000h to 9FFFh	A000h to BFFFh
Partition 3	BIST control/status regs	C000h to DFFFh	E000h to FFFFh

The WKUP_CTRL_MMR0 memory space is divided into the partitions shown in Table 5-3.

Table 5-3. WKUP_CTRL_MMR0 Register Partitions

Partition(1)	Description	Proxy0 Offset Range	Proxy1 Offset Range
Partition 0	General configuration and IPC regs	0000h to 1FFFh	2000h to 3FFFh
Partition 1	IP control/status regs	4000h to 5FFFh	6000h to 7FFFh
Partition 2	Clock control/status regs	8000h to 9FFFh	A000h to BFFFh
Partition 3	BIST control/status regs	C000h to DFFFh	E000h to FFFFh
Partition 5	Power Management regs	1 4000h to 1 5FFFh	1 6000h to 1 7FFFh
Partition 6	Power and reset control and status	1 8000h to 1 9000h	1 A000h to 1 B000h
Partition 7	WKUP I/O Pad Configuration regs	1 C000h to 1 DFFFh	1 E000h to 1 FFFFh

5.1.2.2 Pad Configuration Registers

The pad configuration registers are used to configure most of the device pads. Each pad configuration register PADCONFIGn is associated only with one pad and configures features such as:

- Pin multiplexing
- Wakeup
- Debounce and Schmitt Trigger
- Deep sleep
- Driver and Receiver Enables
- IO Isolation

Copyright © 2024 Texas Instruments Incorporated

The selection of functions available on each pad is enumerated in table "*Pin Multiplexing*" of the device-specific Datasheet. The desired function is selected via the MUXMODE field of the associated pad configuration register.

5.1.2.3 Kick Protection Registers

There are Kick and Proxy registers associated with each CTRL_MMR register partition. This section describes the Kick registers. For information about the Proxy registers see Section 5.1.2.4.

The Kick registers provide a protection mechanism which prevents spurious writes from changing the values of its registers. The LOCKi_KICK0 and LOCKi_KICK1 registers are used for this purpose. A write is required first to the LOCKi_KICK0[31-1] KEY field and then to the LOCKi_KICK1[31-0] KEY field with exact data values to unlock the protection mechanism. Once released then all registers within Partition "i" having write permissions can be written to. The read only registers are still read only. An indication for unlocked Partition "i" is when the LOCKi_KICK0[0] UNLOCKED bit is set to 1h. When the protection mechanism is locked (indicated by LOCKi_KICK0[0] UNLOCKED = 0h) none of the registers within Partition "i" can be written to. They can only be read.

The following values must be written to the Kick registers to unlock each partition "i":

- LOCKi_KICK0 = 68EF 3490h
- LOCKi_KICK1 = D172 BC5Ah

Writing any other data value to either of these registers locks the protection mechanism and blocks any writes to the registers that reside in Partition "i".

Note

In order to ensure that all registers from all partitions are write protected, software must always re-lock the protection mechanism after completing the register writes.

The following registers are exceptions and are not write protected by the LOCKi_KICK0 and LOCKi_KICK1 registers:

- The *Inter-processor Communication Registers* described in Section 5.1.2.6.
- The *Kick Protection Registers* described in this section.

The *Inter-processor Communication Registers* registers are not write protected in order to reduce the access latency.

5.1.2.4 Proxy Addressing Registers

There are Kick and Proxy registers associated with each CTRL_MMR partition. This section describes the Proxy registers. For information about the Kick registers see Section 5.1.2.3.

The Proxy registers provide proxy addressing, allowing each of the CTRL_MMR registers to be accessed through two different addresses - Proxy0 and Proxy1. Each partition is 16KB. The first 8KB are for Proxy0 addresses, and the second 8KB are for Proxy1 addresses. The Proxy0 address is intended as the normal read and write access address. The Proxy1 address provides exclusive register write control.

When a bit from the P**x**_CLAIM**y**[31-0] PROXY1_CLAIMED field is set to 1h, a register becomes read-only at its Proxy0 address and may only be written through its Proxy1 address. The Proxy registers themselves may always be read through their Proxy0 address but may only be written through their Proxy1 address.

The **x** (representing the partition) and **y** indices can be calculated through the following equations:

- **x** = INT(HEX2DEC(address_offset)/16384))
- **y** = INT(MOD(HEX2DEC(address_offset),16384)/128)

where:

- INT() rounds down to the nearest integer
- HEX2DEC() converts from hexadecimal to decimal
- MOD() returns the reminder of HEX2DEC(address_offset)/16384

Copyright © 2024 Texas Instruments Incorporated

5.1.2.5 CTRL_MMR Interrupts

Each CTRL_MMR module has an associated ACCESS_ERR_0 interrupt request. The interrupt requests are assocaited with the following registers:

- INTR_RAW_STATUS — interrupt raw status register
- INTR_ENABLED_STATUS_CLEAR — interrupt status register
- INTR_ENABLE — interrupt enable register
- INTR_ENABLE_CLEAR — interrupt disable register

The following applies for the interrupt behavior of each CTRL_MMR module:

- The CTRL_MMR module asserts its interrupt line only if the interrupts are enabled by setting to 1h the corresponding bits in the INTR_ENABLE register. These interrupts can be disabled by setting to 1h the corresponding bits in the INTR_ENABLE_CLEAR register.
- After an interrupt has been serviced, software must clear the corresponding status flag. This is done by setting to 1h the corresponding bit in the INTR_ENABLED_STATUS_CLEAR register which also clears the corresponding bit in the INTR_RAW_STATUS register. The status flags in the INTR_RAW_STATUS register are set even if the corresponding interrupt is disabled unlike those in the INTR_ENABLED_STATUS_CLEAR register, which are set only if the corresponding interrupt is enabled.
- An interrupt is also generated by the CTRL_MMR, if certain bit in the INTR_RAW_STATUS register is set to 1h and the corresponding interrupt is enabled through the INTR_ENABLE register. This feature is useful during user software debugging. In addition, even if interrupts are not enabled the corresponding raw flag in the INTR_RAW_STATUS register is set to 1h when an IRQ condition occurs.
- Even if interrupts are not enabled, a status bit in the INTR_RAW_STATUS register can also be cleared by setting to 1h the corresponding bit in the INTR_ENABLED_STATUS_CLEAR register.

Table 5-4 lists the interrupt events which can assert the (ACCESS_ERR_0) interrupt lines.

Table 5-4. CTRL_MMR Events

Event	Description
PROXY_ERR	Proxy violation interrupt. Occurs when a write is attempted through the Proxy0 address of register whose associated CLAIM bit is set.
LOCK_ERR	Lock violation interrupt. Occurs when writing to a register in a locked CTRL_MMR partition.
ADDR_ERR	Addressing violation interrupt. Occurs when accessing an illegal address inside the CTRL_MMR module.
PROT_ERR	Protection violation interrupt. Occurs when a register is accessed without the required secure/privilege level permissions.

When an error event as described in Table 5-4 occurs, the error associated details are captured in the FAULT_ADDRESS, FAULT_TYPE_STATUS andFAULT_ATTR_STATUS registers. FAULT_ADDRESS contains the address of the first fault access. FAULT_TYPE_STATUS andFAULT_ATTR_STATU contain status attributes associated with the first fault access. To clear the contents of these three registers and allow them to latch the attributes of the next fault the FAULT_CLEAR[0] CLEAR bit must be set to 1h.

5.1.2.6 Inter-processor Communication Registers

The inter-processor communication registers in the CTRL_MMR modules are used for generating interrupts to the device cores. Any master having access to these registers can generate an interrupt by writing 1h to the IPC_SET**x**[0] bits. The interrupt is cleared when 1h is written to the IPC_CLR**x**[0] bits. These registers provide also a *Source ID* facility through the IPC_SET**x**[31-4] IPC_SRC_SET and IPC_CLR**x**[31-4] IPC_SRC_CLR bit fields by which up to 28 different sources of interrupts can be identified. Allocation of the source bits to source processor and meaning is entirely based on software convention. Virtually, anything can be a source for these fields as this is completely controlled by software.

Writing 1h to an IPC_SRC_SET bit sets to 1h both the IPC_SRC_SET and corresponding IPC_SRC_CLR bit. Writing 1h to an IPC_SRC_CLR bit sets to 0h (clears) both the IPC_SRC_CLR and corresponding IPC_SRC_SET bit. The same logic applies also to the IPC_SET**x**[0] IPC_SET and IPC_CLR**x**[0] IPC_CLR bits. Table 5-5 and Table 5-6 shows the mapping between the inter-processor communication (IPC) registers and device cores.

Copyright © 2024 Texas Instruments Incorporated

Table 5-5. CTRL_MMR0 IPC Registers to Device Core Mapping

IPC Register Instance "x"[1] CTRL_MMR0_IPC_SETx CTRL_MMR0_IPC_CLRx	Device Core
0	C7SS0
1	C7SS1
8	A72SS0 core 0
9	A72SS0 core 1
16	R5FSS0 core 0
17	R5FSS0 core 1
18	R5FSS1 core 0
19	R5FSS1 core 1

1. Not listed IPC register instances are reserved and not used.

Table 5-6. MCU_CTRL_MMR0 IPC Registers to Device Core Mapping

IPC Register Instance "x"[1] MCU_CTRL_MMR0_IPC_SETx MCU_CTRL_MMR0_IPC_CLRx	Device Core
0	MCU R5 core0
1	MCU R5 core1
8	DMSC

1. Not listed IPC register instances are reserved and not used.

Note

For latency reasons the IPC registers are not write protected by KICK registers which means that they can be written to without a need for performing unlocking procedure as described in Section 5.1.2.3.

5.1.2.7 Timer IO Muxing Control Registers

Not all timer inputs and outputs are pinned out of the device. Through the timer I/O muxing registers, the inputs and outputs of each TIMER module in a domain can be made available on the TIMER IO device pads in that same domain.

Each TIMER module input in a domain can be configured to be driven by one of the TIMER IO pads in that domain through the corresponding bits in the TIMERx_CTRL registers.

Additionally, each of the TIMER IO pads in a domain can be configured to be driven by any of the TIMER module outputs in that domain through the corresponding bits in the TIMERIOx_CTRL registers.

For more information about each timer, see Section 12.9.3 *Timers*.

5.1.2.8 EHRPWM/EQEP Control and Status Registers

The CTRL_MMR registers which provide control and status information for the EHRPWM/EQEP modules are shown in Table 5-7.

Table 5-7. Summary of the EHRPWM/EQEP Control and Status Registers

Register	Description
EPWM0_CTRLx	Time base clock, source of PWM synchronization input and other controls for the EHRPWM**x** [(1)] module
SOCA_SEL SOCB_SEL	Start of Conversion output source
EQEP_STAT	Provides EQEP**x** phase error status information

(1) The terms "EPWM" and "EHRPWM" are interchangeable in this section.

Copyright © 2024 Texas Instruments Incorporated

5.1.2.9 Clock Muxing and Division Registers

The CLKSEL, OBSCLKx, and CLKOUT registers within the CTRL_MMR modules are dedicated to clock muxing and division for some device modules. Related information can also be found in *Clocking*.

5.1.2.10 Module Control Registers

There are CTRL registers within the CTRL_MMR module that are control other modules within the device. These CTRL registers supplement the registers present inside the module itself, providing additional module functions.

5.1.2.11 DDRSS Dynamic Frequency Change Registers

There are registers within the CTRL_MMR module that are used to dynamically change the DDR clock frequency to support LPDDR4 Frequency Set Point (FSP). These registers are shown in Table 5-8. For more information, see *DDRSS Dynamic Frequency Change Interface* in *Memory Controllers*.

Table 5-8. DDRSS Dynamic Frequency Change Registers

Register Name
CHNG_DDR4_FSP_REQx
CHNG_DDR4_FSP_ACKx
DDR4_FSP_CLKCHNG_REQx
DDR4_FSP_CLKCHNG_ACKx
MULTI_DDR_CFGx
DDR_CFG_LOAD

5.1.2.12 MAC Address Registers

Each device has a unique 48-bit MAC address which is stored in the MAC_ID0/1 registers shown in Table 5-9.

Table 5-9. MAC Address Registers

Bit Field	Description
MAC_ID0[31-0] MACID_LO	Contains the first, second, third and fourth octets of the MCU_CPSW0 MAC address
MAC_ID1[15-0] MACID_HI	Contains the fifth and sixth octets of the MCU_CPSW0 MAC address

5.1.2.13 Feature Registers

The DEVICE_FEATUREx and FEATURE_STATx registers within the CTRL_MMR module provide status information regarding whether specific device modules and features are enabled or disabled.

5.1.2.14 Power and Reset Related Registers

There are several registers within the CTRL_MMR moduels related to power and reset control (CTRL) and status (STAT).

Reset-realted registers:
- RESET_SRC_STAT: Indicates which reset occured
- MAIN_POR_TO_CTRL: Sets the MAIN PORz hardware timeout period
- POR_RST_CTRL: Controls POR behavior
- WARM_RST_CTRL: Controls Warm Reset behavior
- RST_STAT: Indicates the reset status

Power-realted registers:
- PRGx_CTRL/STAT: PRG module control and status of the POK and POR over/undervoltage detectors
- POR_POK*_CTRL/STAT: Control and status of the POR module over/undervoltage detectors
- POK_*_CTRL/STAT: Control and status of the POK module over/undervoltage detectors
- GLDTC_CTRL/STAT: Control and status of the PGD module voltage glitch detectors

5.1.2.15 I/O Debounce Control Registers

Some device pads have debounce logic. The DBOUNCE_CFGx registers are used to configure the debounce period.

Copyright © 2024 Texas Instruments Incorporated

The DBOUNCE_CFG1 register contains the debounce period for pads with PADCONFIGx[13-11] DEBOUNCE_SEL fields set to 1h. The DBOUNCE_CFG2 register contains the debounce period for pads with PADCONFIGx[13-11] DEBOUNCE_SEL fields set to 2h and so on. The DBOUNCE_CFG6 register contains the debounce period for pads with PADCONFIGx[13-11] DEBOUNCE_SEL fields set to 6h.

Note

The debounce logic is not associated with all signals that can be multiplexed on a pad. Only certain signals can use it.

For information about the PADCONFIGx registers, see *Pad Configuration Registers*.

Table 5-10 shows the debounce period values to load in the DBOUNCE_CFGx[5-0] DB_CFG field.

Note

The debounce clock selection should happen before the consumer of the debounced signals is enabled as the clock multiplexers for the debounce logic are NOT glitch-free.

Note

ADC trigger input signals offer debounce logic that can be enabled via the centralized PADCONFIG registers. The PADCONFIG registers corresponding to ADCx_AIN0 are used to control the debounce time for all ADCx_AINn pads.

Copyright © 2024 Texas Instruments Incorporated

Table 5-10. Debounce Period Values

DBOUNCE_CFGx[5-0] DB_CFG Decimal Value	GPIO Case (32.768kHz [(1)])	EQEP Case (20MHz[(1)])	DBOUNCE_CFGx[5-0] DB_CFG Decimal Value	GPIO Case (20MHz[(1)])	EQEP Case (250MHz[(1)])
	Delay[ms]	Delay[µs]		Delay[µs]	Delay[ns]
0	bypassed	bypassed	32	0.05	4
1	1.95	3.2	33	0.1	8
2	2.93	4.8	34	0.15	12
3	3.91	6.4	35	0.2	16
4	4.88	8	36	0.25	20
5	5.86	9.6	37	0.3	24
6	6.84	11.2	38	0.35	28
7	7.81	12.8	39	0.4	32
8	8.79	14.4	40	0.45	36
9	9.77	16	41	0.5	40
10	10.74	17.6	42	0.55	44
11	11.72	19.2	43	0.6	48
12	12.7	20.8	44	0.65	52
13	13.67	22.4	45	0.7	56
14	14.65	24	46	0.75	60
15	15.63	25.6	47	0.8	64
16	16.6	27.2	48	0.85	68
17	17.58	28.8	49	0.9	72
18	18.55	30.4	50	0.95	76
19	19.53	32	51	1	80
20	20.51	33.6	52	1.05	84
21	21.48	35.2	53	1.1	88
22	15.63	25.6	54	25.6	2.048
23	31.25	51.2	55	51.2	4.096
24	46.88	76.8	56	76.8	6.144
25	62.5	102.4	57	102.4	8.192
26	78.13	128	58	128	10.24
27	93.75	153.6	59	153.6	12.288
28	109.38	179.2	60	179.2	14.336
29	125	204.8	61	204.8	16.384
30	140.63	230.4	62	230.4	18.432
31	156.25	256	63	256	20.48

(1) These are the debounce logic clock frequencies.

5.1.3 Control Module Registers

Note

Some single-bit fields in the Control Module (CTRL_MMR) register descriptions do not explicitly state how to interpret values of 0 and 1. For these fields, the interpretation is:

- 0h: Description is FALSE
- 1h: Description is TRUE

Copyright © 2024 Texas Instruments Incorporated

5.2 Power

This chapter describes the power-management architecture implemented in the device.

The Power Management is presented in several general sections - Power Management Subsystems, including both System and Control Modules; Device Power States; Dynamic and Thermal Management.

5.2.1 Power Management Overview

Power management architecture of the device is built over three levels of power controls: clock-, power-, and voltage supplies to the subsystem modules, namely domains. A domain is a group of modules or subsections of the device that share a common entity (for example, common clock source, common voltage source, or common power switch).

A set of Power Domains (PD) are defined for the device, where clock and power supplies to the domain may be controlled by Power Sleep Controllers (PSC). One or multiple power domains may share a single power supply, this level of grouping are referred as Voltage Domains (VD). Some voltage domains may be turned off by externally switching off the power supplies to these domains. The hierarchical relationship is summarized in Section 5.2.2.

Figure 5-1 shows the overview of the common power management of the device.

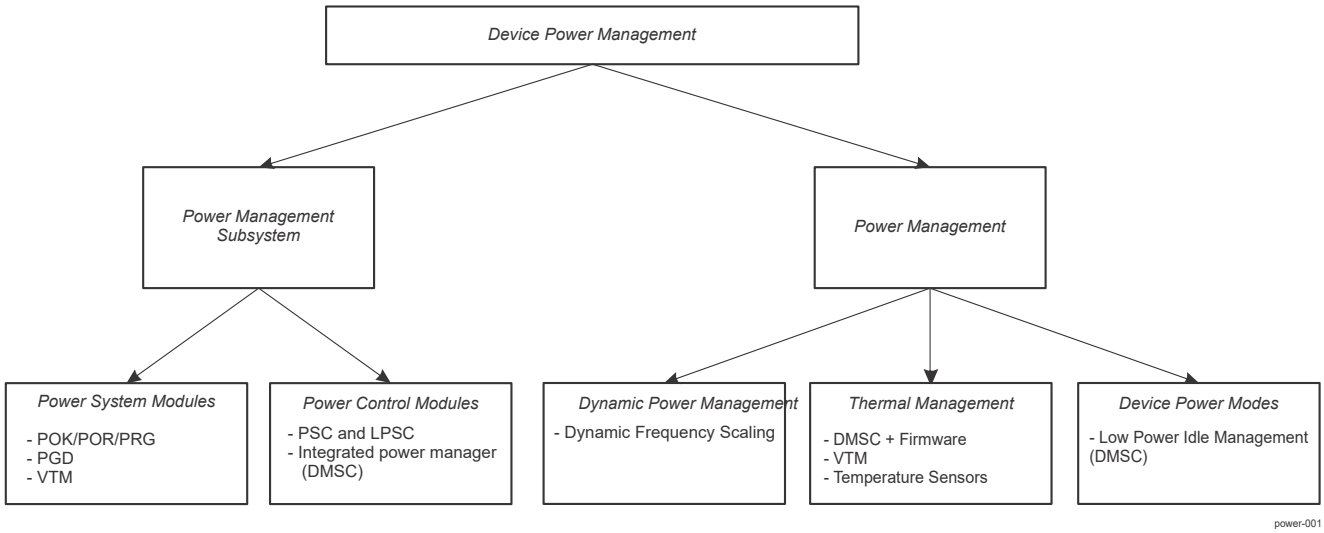


Figure 5-1. Overview of Common Power Management

5.2.2 WKUP_PSC0 Device-Specific Information

Table 5-11 provides the device-level view with module asSoCiations to the clock, power, and voltage domains.

Table 5-11. PSC0 Power Management Device Level Layout

VD Name	PD Name	PD Index	LPSC Name	LPSC Index	PD can be disabled	Components controlled by LPSC
VD_core	GP_Core_CTL	0	LPSC_main_alwayson	0	No	COMPUTE_CLUSTER0_CORE_CORE, CPT2_AGGR1, CPT2_AGGR5, CPT2_AGGR2, CPT2_AGGR4, CPT2_AGGR3, CPT2_AGGR0, DCC0, DCC1, DCC2, DCC3, DCC4, DCC5, DCC6, DCC7, DCC8, DCC9, TIMER4, TIMER5, TIMER6, TIMER7, ESM0, GPIO0, NAVSS0_BCDMA_0, NAVSS0_CPTS_0, NAVSS0_INTR_0, NAVSS0_MAILBOX1_0, NAVSS0_MAILBOX1_1, NAVSS0_MAILBOX1_10, NAVSS0_MAILBOX1_11, NAVSS0_MAILBOX1_2, NAVSS0_MAILBOX1_3, NAVSS0_MAILBOX1_4, NAVSS0_MAILBOX1_5, NAVSS0_MAILBOX1_6, NAVSS0_MAILBOX1_7, NAVSS0_MAILBOX1_8, NAVSS0_MAILBOX1_9, NAVSS0_MAILBOX_0, NAVSS0_MAILBOX_1, NAVSS0_MAILBOX_10, NAVSS0_MAILBOX_11, NAVSS0_MAILBOX_2, NAVSS0_MAILBOX_3, NAVSS0_MAILBOX_4, NAVSS0_MAILBOX_5, NAVSS0_MAILBOX_6, NAVSS0_MAILBOX_7, NAVSS0_MAILBOX_8, NAVSS0_MAILBOX_9, NAVSS0_MCRC_0, NAVSS0_MODSS, NAVSS0_MODSS_INTA_0, NAVSS0_MODSS_INTA_1, NAVSS0_PROXY_0, NAVSS0_PVU_0, NAVSS0_PVU_1, NAVSS0_RINGACC_0, NAVSS0_SPINLOCK_0, NAVSS0_TIMERMGR_0, NAVSS0_TIMERMGR_1, NAVSS0_UDMAP_0, NAVSS0_UDMASS, NAVSS0_UDMASS_INTA_0, NAVSS0_VIRTSS
VD_core	GP_Core_CTL	0	LPSC_main_test	1	No	-
VD_core	GP_Core_CTL	0	LPSC_main_pbist	2	No	PBIST0, PBIST1, PBIST3, PBIST4

Copyright © 2024 Texas Instruments Incorporated

Table 5-11. PSC0 Power Management Device Level Layout (continued)

VD Name	PD Name	PD Index	LPSC Name	LPSC Index	PD can be disabled	Components controlled by LPSC
VD_core	GP_Core_CTL	0	LPSC_per_audio	3	No	MCASP0, MCASP1, MCASP2, MCASP3, MCASP4
VD_core	GP_Core_CTL	0	LPSC_PER_ATL	4	No	ATL0
VD_core	GP_Core_CTL	0	LPSC_PER_MLB	5	No	NAVSS0
VD_core	GP_Core_CTL	0	LPSC_PER_motor	6	No	ECAP0, ECAP1, ECAP2, EQEP0, EQEP1, EQEP2, EPWM0, EPWM1, EPWM2, EPWM3, EPWM4, EPWM5
VD_core	GP_Core_CTL	0	LPSC_PER_miscio	7	No	I2C0
VD_core	GP_Core_CTL	0	LPSC_PER_GPMC	8	No	ELM0, GPMC0
VD_core	GP_Core_CTL	0	LPSC_PER_VPFE	9	No	GPIO2, GPIO4, GPIO6, GTC0, MAIN2MCU_LVL_INTRTR0, MAIN2MCU_PLS_INTRTR0, TIMESYNC_INTRTR0, GPIOMUX_INTRTR0, CMPEVENT_INTRTR0
VD_core	GP_Core_CTL	0	LPSC_PER_VPE	10	No	-
VD_core	GP_Core_CTL	0	LPSC_PER_spare0	11	No	TIMER8, TIMER9, TIMER10, TIMER11, TIMER12, TIMER13, TIMER14, TIMER15, TIMER16, TIMER17, TIMER18, TIMER19
VD_core	GP_Core_CTL	0	LPSC_PER_spare1	12	No	-
VD_core	GP_Core_CTL	0	LPSC_main_debug	13	No	STM0, DEBUGSS_WRAP0, DEBUGSUSPENDRTR0
VD_core	GP_Core_CTL	0	LPSC_EMIF_DATA_0	14	No	EMIF_DATA_0_VD
VD_core	GP_Core_CTL	0	LPSC_EMIF_CFG_0	15	No	DDR0
VD_core	GP_Core_CTL	0	LPSC_EMIF_DATA_1	16	No	EMIF_DATA_1_VD
VD_core	GP_Core_CTL	0	LPSC_EMIF_CFG_1	17	No	DDR1
VD_CC	GP_Core_CTL	0	LPSC_PER_spare2	18	No	-
VD_CC	GP_Core_CTL	0	LPSC_CC_Top_PBIST	19	No	COMPUTE_CLUSTER0_PBIST_WRAP_0
VD_core	GP_Core_CTL	0	LPSC_USB_0	20	No	USB0
VD_core	GP_Core_CTL	0	LPSC_USB_1	21	No	VUSR_DUAL0
VD_core	GP_Core_CTL	0	LPSC_USB_2	22	No	-
VD_core	GP_Core_CTL	0	LPSC_MMC4b_0	23	No	DEV_MMCSD1
VD_core	GP_Core_CTL	0	LPSC_MMC4b_1	24	No	
VD_core	GP_Core_CTL	0	LPSC_MMC8b_0	25	No	MMCSD0
VD_core	GP_Core_CTL	0	LPSC_UFS_0	26	No	-
VD_core	GP_Core_CTL	0	LPSC_UFS_1	27	No	-
VD_core	GP_Core_CTL	0	LPSC_PCIe_0	28	No	-
VD_core	GP_Core_CTL	0	LPSC_PCIe_1	29	No	PCIE1
VD_core	GP_Core_CTL	0	LPSC_PCIe_2	30	No	-
VD_core	GP_Core_CTL	0	LPSC_PCIe_3	31	No	-
VD_core	GP_Core_CTL	0	LPSC_SAUL	32	No	SA2_UL0
VD_core	GP_Core_CTL	0	LPSC_PER_I3C	33	No	-
VD_core	PD_mcanss	1	LPSC_main_mcanss_0	34	No	MCAN0
VD_core	PD_mcanss	1	LPSC_main_mcanss_1	35	No	MCAN1
VD_core	PD_mcanss	1	LPSC_main_mcanss_2	36	No	MCAN2
VD_core	PD_mcanss	1	LPSC_main_mcanss_3	37	No	MCAN3
VD_core	PD_mcanss	1	LPSC_main_mcanss_4	38	No	MCAN4, MCAN5, MCAN6, MCAN7, MCAN8, MCAN9
VD_core	PD_mcanss	1	LPSC_main_mcanss_5	39	No	MCAN10, MCAN11, MCAN12, MCAN13, MCAN14, MCAN15, MCAN16, MCAN17
VD_core	PD_mcanss	1	LPSC_main_mcanss_6	40	No	0
VD_core	PD_mcanss	1	LPSC_main_mcanss_7	41	No	MCSPI0, MCSPI1, MCSPI2, MCSPI3
VD_core	PD_mcanss	1	LPSC_main_mcanss_8	42	No	MCSPI4, MCSPI5, MCSPI6, MCSPI7
VD_core	PD_mcanss	1	LPSC_main_mcanss_9	43	No	UART0, UART1

Copyright © 2024 Texas Instruments Incorporated

Table 5-11. PSC0 Power Management Device Level Layout (continued)

VD Name	PD Name	PD Index	LPSC Name	LPSC Index	PD can be disabled	Components controlled by LPSC
VD_core	PD_mcanss	1	LPSC_main_mcanss_10	44	No	UART2, UART3
VD_core	PD_mcanss	1	LPSC_main_mcanss_11	45	No	UART4, UART5, UART6, UART7, UART8, UART9
VD_core	PD_mcanss	1	LPSC_main_mcanss_12	46	No	I2C1, I2C2, I2C3
VD_core	PD_mcanss	1	LPSC_main_mcanss_13	47	No	I2C4, I2C5, I2C6
VD_core	PD_DSS	2	LPSC_DSS	48	No	DSS0
VD_core	PD_DSS	2	LPSC_DSS_PBIST	49	No	PBIST5
VD_core	PD_DSS	2	LPSC_DSI	50	No	DSI0
VD_core	PD_DSS	2	LPSC_eDP_0	51	No	DSS_EDP0
VD_core	PD_DSS	2	LPSC_eDP_1	52	No	-
VD_core	PD_DSS	2	LPSC_CSIRX_0	53	No	CSI_RX_IF0
VD_core	PD_DSS	2	LPSC_CSIRX_1	54	No	CSI_RX_IF1
VD_core	PD_DSS	2	LPSC_CSIRX_2	55	No	-
VD_core	PD_DSS	2	LPSC_CSITX_0	56	No	CSI_TX_IF_V2_0
VD_core	PD_DSS	2	LPSC_TX_DPHY	57	No	DPHY_TX0
VD_core	PD_DSS	2	LPSC_CSIRX_PHY_0	58	No	DPHY_RX0
VD_core	PD_DSS	2	LPSC_CSIRX_PHY_1	59	No	DPHY_RX1
VD_core	PD_DSS	2	LPSC_CSIRX_PHY_2	60	No	-
VD_core	PD_ICSS	3	LPSC_ICSSG_0	61	No	-
VD_core	PD_ICSS	3	LPSC_ICSSG_1	62	No	-
VD_core	PD_9GSS	4	LPSC_9GSS	63	No	-
VD_core	PD_SERDES_0	5	LPSC_SERDES_0	64	No	SERDES_10G0
VD_core	PD_SERDES_1	6	LPSC_SERDES_1	65	No	-
VD_core	PD_SERDES_2	7	LPSC_SERDES_2	66	No	-
VD_core	PD_SERDES_3	8	LPSC_SERDES_3	67	No	-
VD_core	PD_SERDES_4	9	LPSC_SERDES_4	68	No	-
VD_core	PD_SERDES_5	10	LPSC_SERDES_5	69	No	-
VD_core	PD_timer	11	LPSC_Dmtimer_0	70	No	TIMER0
VD_core	PD_timer	11	LPSC_Dmtimer_1	71	No	TIMER1
VD_core	PD_timer	11	LPSC_Dmtimer_2	72	No	TIMER2
VD_core	PD_timer	11	LPSC_Dmtimer_3	73	No	TIMER3
VD_CC	PD_C71x_0	12	LPSC_C71x_0	74	Yes	COMPUTE_CLUSTER0_C71SS0_0, RTI16
VD_CC	PD_C71x_0	12	LPSC_C71x_0_PBIST	75	Yes	C71X_0_PBIST_VD
VD_CC	PD_C71x_1	13	LPSC_C71x_1	76	Yes	COMPUTE_CLUSTER0_C71SS0_1, RTI17
VD_CC	PD_C71x_1	13	LPSC_C71x_1_PBIST	77	Yes	C71X_1_PBIST_VD
VD_CC	PD_A72_cluster_0	14	LPSC_A72_cluster_0	78	Yes	A72SS0
VD_CC	PD_A72_cluster_0	14	LPSC_A72_cluster_0_PBIST	79	Yes	-
VD_CC	PD_A72_0	15	LPSC_A72_0	80	Yes	A72SS0_CORE0, RTI0
VD_CC	PD_A72_1	16	LPSC_A72_1	81	Yes	A72SS0_CORE1, RTI1
VD_CC	PD_A72_cluster_1	17	LPSC_A72_cluster_1	82	Yes	-
VD_CC	PD_A72_cluster_1	17	LPSC_A72_cluster_1_PBIST	83	Yes	-
VD_CC	PD_A72_2	18	LPSC_A72_2	84	Yes	-
VD_CC	PD_A72_3	19	LPSC_A72_3	85	Yes	-
VD_core	PD_GPUCOM	20	LPSC_gpucom	86	Yes	GPU_BXS464_WRAP0_GPU_SS_0, RTI15
VD_core	PD_GPUCOM	20	LPSC_gpupbist	87	Yes	GPU_BXS464_WRAP0_DFT_EMBED_PBIST_0
VD_core	PD_GPUCORE	21	LPSC_gpucore	88	Yes	-
VD_CC	PD_C66x_0	22	LPSC_c66x_0	89	Yes	-
VD_CC	PD_C66x_0	22	LPSC_c66x_pbist_0	90	Yes	-

Copyright © 2024 Texas Instruments Incorporated

Table 5-11. PSC0 Power Management Device Level Layout (continued)

VD Name	PD Name	PD Index	LPSC Name	LPSC Index	PD can be disabled	Components controlled by LPSC
VD_CC	PD_C66x_1	23	LPSC_c66x_1	91	Yes	-
VD_CC	PD_C66x_1	23	LPSC_c66x_pbist_1	92	Yes	-
VD_core	PD_Pulsar_0	24	LPSC_Pulsar_0_R5_0	93	Yes	R5FSS0_CORE0, RTI28
VD_core	PD_Pulsar_0	24	LPSC_Pulsar_0_R5_1	94	Yes	R5FSS0_CORE1, RTI29
VD_core	PD_Pulsar_0	24	LPSC_Pulsar_pbist_0	95	Yes	PBIST2
VD_core	PD_Pulsar_1	25	LPSC_Pulsar_1_R5_0	96	Yes	R5FSS1_CORE0, RTI30
VD_core	PD_Pulsar_1	25	LPSC_Pulsar_1_R5_1	97	Yes	R5FSS1_CORE1, RTI31
VD_core	PD_Pulsar_1	25	LPSC_Pulsar_pbist_1	98	Yes	PBIST10
VD_core	PD_decode	26	LPSC_decode_0	99	Yes	-
VD_core	PD_decode	26	LPSC_decode_pbist	100	Yes	-
VD_core	PD_encode	27	LPSC_encode_0	101	Yes	-
VD_core	PD_encode	27	LPSC_encode_pbist	102	Yes	-
VD_core	PD_DMPAC	28	LPSC_DMPAC	103	Yes	DMPAC0, DMPAC0_CTSET_0, DMPAC0_INTD_0
VD_core	PD_DMPAC	28	LPSC_SDE	104	Yes	DMPAC0_SDE_0
VD_core	PD_DMPAC	28	LPSC_DMPAC_PBIST	105	Yes	PBIST7
VD_core	PD_VPAC	29	LPSC_VPAC	106	Yes	VPAC0
VD_core	PD_VPAC	29	LPSC_VPAC_PBIST	107	Yes	PBIST8
VD_CC	PD_A72_CL0_2	30	LPSC_A72_Clstr0_Core2	108	Yes	-
VD_CC	PD_A72_CL0_3	31	LPSC_A72_Clstr0_Core3	109	Yes	-
VD_CC	PD_A72_CL1_2	32	LPSC_A72_Clstr1_Core2	110	Yes	-
VD_CC	PD_A72_CL1_3	33	LPSC_A72_Clstr1_Core3	111	Yes	-
VD_core	PD_VPAC_1	34	LPSC_vpac_1	112	Yes	-
VD_core	PD_VPAC_1	34	LPSC_vpac_1_pbist	113	Yes	-
VD_core	PD_ENCODE_1	35	LPSC_encode_1	114	Yes	K3_VPU_WAVE521CL0
VD_core	PD_ENCODE_1	35	LPSC_encode_1_pbist	115	Yes	PBIST11
VD_core	PD_DSI_1	36	LPSC_CSITX_1	116	No	CSI_TX_IF_V2_1, DSS_DSI1
VD_core	PD_DSI_1	36	LPSC_TX_DPHY_1	117	No	DPHY_TX1
VD_core	PD_DSI_1	36	LPSC_dsi_1_pbist	118	No	-
VD_core	PD_CPSW2	37	LPSC_cpsw_2	119	No	CPSW1
VD_core	PD_DDR2	38	LPSC_emif_data_2	120	No	-
VD_core	PD_DDR2	38	LPSC_emif_cfg_2	121	No	-
VD_core	PD_Pulsar_2	39	LPSC_Pulsar_2_R5_0	122	Yes	-
VD_core	PD_Pulsar_2	39	LPSC_Pulsar_2_R5_1	123	Yes	-
VD_core	PD_Pulsar_2	39	LPSC_Pulsar_2_pbist	124	Yes	-
VD_core	PD_SPARE4	40	LPSC_spare_4	125	Yes	-
VD_core	PD_SPARE5	41	LPSC_spare_5	126	Yes	-
VD_core	PD_SPARE6	42	LPSC_spare_6	127	Yes	-

Copyright © 2024 Texas Instruments Incorporated

Table 5-12. WKUP-PSC0 Power Management Device Level Layout

VD Name	PD Name	PD Index	LPSC Name	LPSC Index	PD can be disabled	Components controlled by LPSC
VD_wkup/mcu	GP_CORE_CTL_wkup	0	LPSC_wkup_alwayson	0	No	MCU_CPSW0, MCU_CPT2_AGGR0, MCU_DCC0, MCU_DCC1, MCU_DCC2, MCU_TIMER0, MCU_TIMER1, MCU_TIMER2, MCU_TIMER3, MCU_TIMER4, MCU_TIMER5, MCU_TIMER6, MCU_TIMER7, MCU_TIMER8, MCU_TIMER9, WKUP_ESM0, MCU_ESM0, MCU_FSS0_FSAS_0, MCU_FSS0_HYPERBUS1P0_0, MCU_FSS0_OSPI_0, MCU_FSS0_OSPI_1, WKUP_GPIOMUX_INTRTR0, WKUP_DDPA0, WKUP_VTM0, MCU_I2C0, MCU_I2C1, MCU_NAVSS0_INTR_ROUTER_0, MCU_NAVSS0_MCRC_0, MCU_NAVSS0_MODSS, MCU_NAVSS0_PROXY0, MCU_NAVSS0_RINGACC0, MCU_NAVSS0_UDMAP_0, MCU_NAVSS0_UDMASS, MCU_NAVSS0_UDMASS_INTA_0, MCU_SA3_SS0_DMSS_ECCAGGR_0, MCU_SA3_SS0_INTAGGR_0, MCU_SA3_SS0_PKTDMA_0, MCU_SA3_SS0_RINGACC_0, MCU_SA3_SS0_SA_UL_0, MCU_MCSPI0, MCU_MCSPI1, MCU_MCSPI2, MCU_UART0
VD_wkup/mcu	GP_CORE_CTL_wkup	0	LPSC_dmsc	1	No	
VD_wkup/mcu	GP_CORE_CTL_wkup	0	LPSC_debug2dmsc	2	No	-
VD_wkup/mcu	GP_CORE_CTL_wkup	0	LPSC_wkup_gpio	3	No	WKUP_GPIO0, WKUP_GPIO1, WKUP_I2C0, WKUP_UART0
VD_wkup/mcu	GP_CORE_CTL_wkup	0	LPSC_wkupmcu2main	4	No	WKUPMCU2MAIN_VD
VD_wkup/mcu	GP_CORE_CTL_wkup	0	LPSC_main2wkupmcu	5	No	MAIN2WKUPMCU_VD
VD_wkup/mcu	GP_CORE_CTL_wkup	0	LPSC_mcu_test	6	No	MCU_PBIST0, MCU_PBIST1
VD_wkup/mcu	GP_CORE_CTL_wkup	0	LPSC_mcu_debug	7	No	-
VD_wkup/mcu	GP_CORE_CTL_wkup	0	LPSC_mcu_mcan_0	8	No	MCU_MCAN0
VD_wkup/mcu	GP_CORE_CTL_wkup	0	LPSC_mcu_mcan_1	9	No	MCU_MCAN1
VD_wkup/mcu	GP_CORE_CTL_wkup	0	LPSC_mcu_ospi_0	10	No	MCU_FSS0_OSPI_0
VD_wkup/mcu	GP_CORE_CTL_wkup	0	LPSC_mcu_ospi_1	11	No	MCU_FSS0_OSPI_1
VD_wkup/mcu	GP_CORE_CTL_wkup	0	LPSC_mcu_hyperbus	12	No	MCU_FSS0_HYPERBUS1P0_0
VD_wkup/mcu	GP_CORE_CTL_wkup	0	LPSC_MCU_I3C_0	13	No	MCU_I3C0
VD_wkup/mcu	GP_CORE_CTL_wkup	0	LPSC_MCU_I3C_1	14	No	MCU_I3C1
VD_wkup/mcu	GP_CORE_CTL_wkup	0	LPSC_mcu_adc_0	15	No	MCU_ADC_0
VD_wkup/mcu	GP_CORE_CTL_wkup	0	LPSC_mcu_adc_1	16	No	MCU_ADC_1
VD_wkup/mcu	GP_CORE_CTL_wkup	0	LPSC_wkup_spare0	17	No	
VD_wkup/mcu	GP_CORE_CTL_wkup	0	LPSC_wkup_spare1	18	No	-
VD_wkup/mcu	PD_MCU_Pulsar	1	LPSC_mcu_r5_0	19	Yes	MCU_R5FSS0_CORE0, MCU_RTI0
VD_wkup/mcu	PD_MCU_Pulsar	1	LPSC_mcu_r5_1	20	Yes	MCU_R5FSS0_CORE1, MCU_RTI1
VD_wkup/mcu	PD_MCU_Pulsar	1	LPSC_mcu_pulsar_pbist_0	21	Yes	MCU_PBIST2

Copyright © 2024 Texas Instruments Incorporated

5.2.3 Power Management Subsystems

This section provides an overview of how three Power blocks work together; the three blocks are: Power On Reset module (POR), Power OK module (POK), and the POK Reset Generator – Ping-Pong module (PRG_PP) and a PRG without Ping-Pong (PRG).

5.2.3.1 POK, PRG_PP, and POR Modules

- The POK module is the basic block which is used to monitor a voltage.
- The PRG and PRG_PP module controls a group of POKs. The PRG samples the output of the POK comparator throughout a De-Glitch period and flags the voltage as NOT OK if the voltage violates the limit at every sample.
- One PRG is instantiated within a POR; the POR module is very nearly an invisible wrapper around the PRG (with associated POKs).

The POR / PRG(_PP) / POK are arranged logically in ways defined by Figure 5-2 and a detailed table of how the POKs are integrated is shown in Table 5-15.

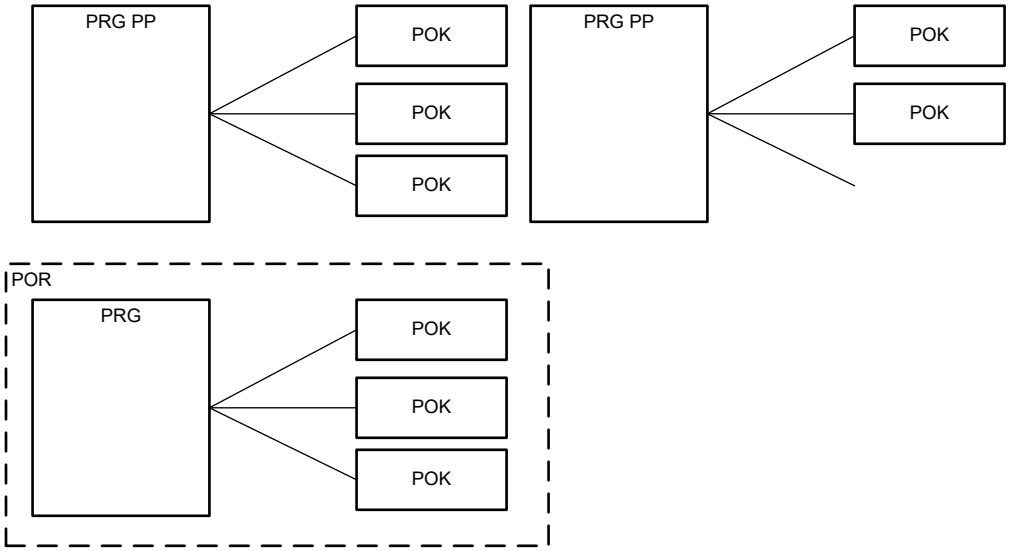


Figure 5-2. PRG / POK /POR Configuration

5.2.3.1.1 Power OK (POK) Modules

5.2.3.1.1.1 POK Overview

POK modules are responsible for accurately detecting average voltage levels.

Two types of POK modules are implemented in this family of devices - POK and POK_SA.

POK module is capable of monitoring a range of supplies and indicating a failure within the programmable upper and lower threshold limits for the supply being monitored. POKs are used to monitor 3.3 V, 1.8 V and core supply levels with programmable threshold levels.

Figure 5-3 shows the concept of the POK. It is a comparator with a fixed 0.45V and the voltage to be monitored. The voltage-to-be-monitored is defined based upon a CTRL register (see *POK Related Registers*) and the silicon hook-up of the POK. The POK has three possible input voltage ports – VDD_MON3P3, VDD_MON1P8, and VDD_MON. The effect of the CTRL register depends upon which of these three ports is selected (defined in Table 5-17); for example, a threshold value of 0x28 in the threshold register would create a different threshold voltage depending upon:

- How the POK was hooked-up (i.e. VDD_MON3P3, VDD_MON1P8, or VDD_MON) see *POK Modules and Monitored Voltages*.
- Whether the POK is selected for undervoltage detection or overvoltage detection as determined by bit 7 in the POK CTRL register (e.g. CTRL_MMR_POR_VDDR_MCU_UV_CTRL)

Copyright © 2024 Texas Instruments Incorporated

The POK output triggers high (i.e. indicating a fail condition) when:

- Under-voltage: the monitored voltage (divided based upon CTRL registers) is less than 0.45V. The CTRL register tables translate the input voltage back to a monitored voltage level (see Table 5-17). If the monitored voltage falls below the trip point, the POK outputs HIGH, (i.e. FAIL).
- Over-voltage: the monitored voltage (divided based upon CTRL registers) is greater than 0.45V. The CTRL register tables translate the input voltage back to a monitored voltage level (see Table 5-17). If the monitored voltage rises above the trip point, the POK outputs HIGH, (i.e. FAIL).

When the monitored voltage threshold is changed, the new threshold settles within 9us.

Figure 5-3 also shows a mux on the output of the comparator. The mux is configured based upon the selection of under-voltage or over-voltage and is used to maintain the proper polarity of the fail signature.

Note

POK_SA (used only on VMON1_ER_VSYS) directly compares the monitored voltage to 0.45V.

Two other features of the POK diagram (i.e. Figure 5-3 and Figure 5-4) need some explanation:

- The comparator can be enabled or disabled individually; note, that once enabled, the comparator output is continuous (though as shown below in is sampled and filtered). The comparator enable settling time is 15us.
- Hysteresis is applied to the comparator input. In general, it is expected that the monitored voltage is far away from the reference and that the hysteresis is moot.

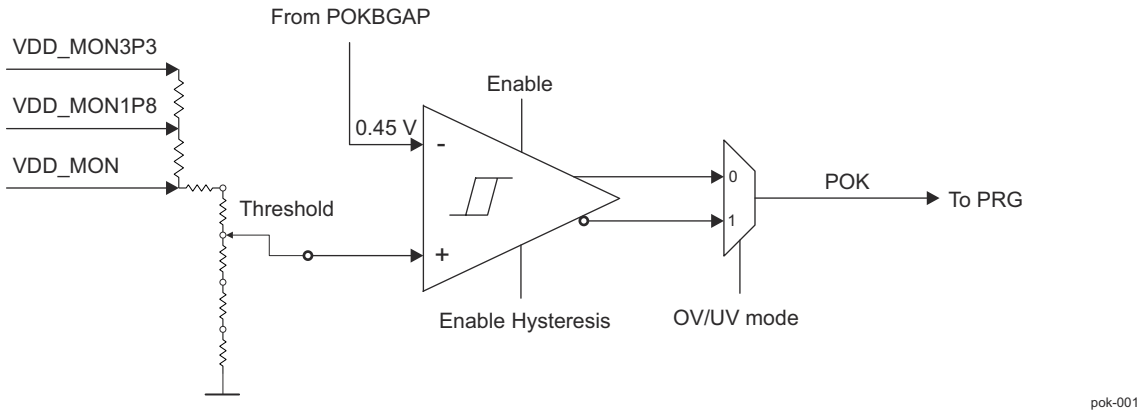


Figure 5-3. POK Block Diagram

Table 5-13. POK Programmable Features

Programmable Feature	Bitfield in the POK Dedicated Register
POK hysteresis enable	[31] HYST_EN
POK over- or under-voltage detection mode	[7] OVER_VOLT_DET
POK trim bits for voltage comparator threshold	[6-0] POK_TRIM

Like the POK, the POK_SA is a comparator with 0.45V reference; however, voltage division must occur in hardware on the PCB before the monitored voltage is applied to VMON1_ER_VSYS.

Copyright © 2024 Texas Instruments Incorporated

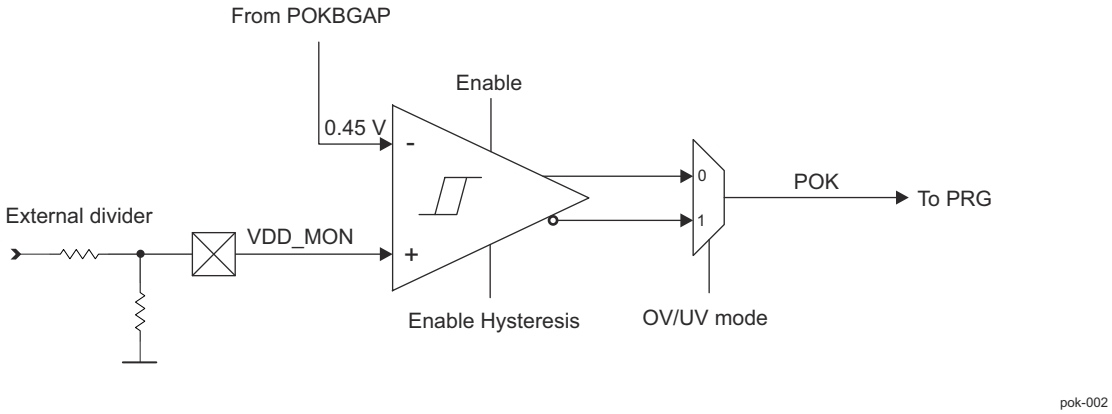


Figure 5-4. POK_SA Block Diagram

Table 5-14. POK_SA Programmable Features

Programmable Feature	Bitfield in the POK Dedicated Register
POK hysteresis enable	[31] HYST_EN
POK over- or under-voltage detection mode	[0] OVER_VOLT_DET

Note

POK configuration is not designed to find and report an instantaneous dip / rise in the voltage.

Table 5-15. POK Modules and Monitored Voltages

Module Instance	Connected to PRG	Type	Voltage Monitored	TAP	OV / UV
POR_POKHV	PRG_POR	POK	VDDA_MCU (1P8V)	VDD_MON1P8	UV
POR_POKLVA	PRG_POR	POK	VDDA_MCU (1P8V)	VDD_MON1P8	OV
POR_POKLVB	PRG_POR	POK	VDD_MCU_UV	VDD_MON	UV
IPOK_VDD_MCU_OV	PRG_POR	POK	VDD_MCU	VDD_MON	OV
IPOK_VDDA_PMIC_IN	PRG_POR	POK_SA	VMON1_ER_VSYS	n/a	UV
IPOK_VDDR_MCU	PRG_PP_MCU	POK	VDDAR_MCU	VDD_MON	OV + UV
IPOK_VDDSHV_WKUP_GEN	PRG_PP_MCU	POK	VDDSHV0_MCU	VDD_MON3P3	OV + UV
IPOK_VMON_CAP_VDDS _MCU_GEN	PRG_PP_MCU	POK	CAP_VDDS0_MCU	VDD_MON1P8	OV + UV
IPOK_VDD_CORE	PRG_PP_MAIN	POK	VDD_CORE	VDD_MON	OV + UV
IPOK_VDDR_CORE	PRG_PP_MAIN	POK	VDDAR_CORE	VDD_MON	OV + UV
IPOK_VDD_CPU	PRG_PP_MAIN	POK	VMON2_IR_VCPU	VDD_MON	OV + UV
IPOK_VMON_EXT	PRG_PP_MAIN	POK	VMON3_IR_VEXT1 P8	VDD_MON1P8	OV + UV
IPOK_VMON_EXT_1P8	PRG_PP_MAIN	POK	VMON4_IR_VEXT1 P8	VDD_MON1P8	OV + UV
IPOK_VMON_EXT_3P3	PRG_PP_MAIN	POK	VMON5_IR_VEXT3 P3	VDD_MON3P3	OV + UV
IPOK_VDD_CPU1	PRG_PP_MAIN	POK	VMON6_IR_VEXT0 P8	VDD_MON	OV + UV

Table 5-16. POK Related Registers

Module Instance	Register
PRG_POR	
POR_POKHV	CTRL_MMR_POR_POKHV_UV_CTRL
POR_POKLVA	CTRL_MMR_POR_POKLVA_OV_CTRL
POR_POKLVB	CTRL_MMR_POR_POKLVB_UV_CTRL

Copyright © 2024 Texas Instruments Incorporated

Table 5-16. POK Related Registers (continued)

Module Instance	Register
IPOK_VDD_MCU_OV	CTRL_MMR_POK_VDD_MCU_OV_CTRL
IPOK_VDDA_PMIC_IN	CTRL_MMR_POK_VDDA_PMIC_IN_CTRL
PRG_PP_MCU	
IPOK_VDDR_MCU	CTRL_MMR_POK_VDDR_MCU_UV_CTRL CTRL_MMR_POK_VDDR_MCU_OV_CTRL
IPOK_VDDSHV_WKUP_GEN	CTRL_MMR_POK_VDDSHV_WKUP_GEN_UV_CTRL CTRL_MMR_POK_VDDSHV_WKUP_GEN_OV_CTRL
IPOK_VMON_CAP_VDDS _MCU_GEN	CTRL_MMR_POK_VMON_CAP_MCU_GEN_UV_CTRL CTRL_MMR_POK_VMON_CAP_MCU_GEN_OV_CTRL
PRG_PP_MAIN	
IPOK_VDD_CORE	CTRL_MMR_POK_VDD_CORE_UV_CTRL CTRL_MMR_POK_VDD_CORE_OV_CTRL
IPOK_VDDR_CORE	CTRL_MMR_POK_VDDR_CORE_UV_CTRL CTRL_MMR_POK_VDDR_CORE_OV_CTRL
IPOK_VDD_CPU	CTRL_MMR_POK_VDD_CPU_UV_CTRL CTRL_MMR_POK_VDD_CPU_OV_CTRL
IPOK_VMON_EXT	CTRL_MMR_POK_VMON_EXT_UV_CTRL CTRL_MMR_POK_VMON_EXT_OV_CTRL
IPOK_VMON_EXT_1P8	CTRL_MMR_POK_VMON_EXT_MAIN1P8_UV_CTRL CTRL_MMR_POK_VMON_EXT_MAIN1P8_OV_CTRL
IPOK_VMON_EXT_3P3	CTRL_MMR_POK_VMON_EXT_MAIN3P3_UV_CTRL CTRL_MMR_POK_VMON_EXT_MAIN3P3_OV_CTRL
IPOK_VDD_CPU1	CTRL_MMR_POK_VDD_CPU1_UV_CTRL CTRL_MMR_POK_VDD_CPU1_OV_CTRL

The possible values of the monitored voltage differ among the POK types, see *POK Modules and Monitored Voltages*.

Table 5-17. POK Threshold Levels

POK_TRIM [6:0]	CORE_POK UV (OVER_VOLT _DET = 0)	CORE_POK OV (OVER_VOLT _DET = 1)	1P8_POK UV (OVER_VOLT _DET = 0)	1P8_POK OV (OVER_VOLT _DET = 1)	3P3_POK UV (OVER_VOLT _DET = 0)	3P3_POK OV (OVER_VOLT _DET = 1)
0x00	0.475 V	0.725 V			1.425 V	2.175 V
0x01	0.4875 V	0x7375 V			1.4625 V	2.2125 V
0x02	0.50 V	0.75 V			1.5 V	2.25 V
…						
0xC				1.432 V		
0xD				1.452 V		
0xE				1.473 V		
…						
0x20			1.432 V			
0x21			1.452 V			
0x22			1.473 V			
…	0.475 V + POK_TRIM[7:0] *0.0125 V	0.725 V + POK_TRIM[7:0] *0.0125 V	0.7775 V + 0.02045* POK_TRIM[7:0]	1.1865 V + 0.02045* POK_TRIM[7:0]	1.425 V + POK_TRIM[7:0] *0.0375 V	2.175 V + POK_TRIM[7:0] *0.0375 V
0x2D						3.8625
0x2E				2.127V		3.9

Copyright © 2024 Texas Instruments Incorporated

Table 5-17. POK Threshold Levels (continued)

POK_TRIM [6:0]	CORE_POK UV (OVER_VOLT _DET = 0)	CORE_POK OV (OVER_VOLT _DET = 1)	1P8_POK UV (OVER_VOLT _DET = 0)	1P8_POK OV (OVER_VOLT _DET = 1)	3P3_POK UV (OVER_VOLT _DET = 0)	3P3_POK OV (OVER_VOLT _DET = 1)
0x2F				2.148 V		3.9375
0x30				2.168 V		
…						
0x42			2.127 V		3.9 V	
0x43			2.148 V		3.9375 V	
0x44	1.325 V		2.168 V		3.975 V	
0x45	1.3375 V					
0x46	1.35 V					
…						
0x48		1.625 V				
0x49		1.6375 V				
0x4A		1.65 V				

Copyright © 2024 Texas Instruments Incorporated

5.2.3.1.2 PoR/Reset Generator (PRG_PP) Modules

5.2.3.1.2.1 PRG / PRG_PP Overview

The POKs are both controlled by and send their output back through the PRG(_PP) module; the PRG(_PP) module, all of which reside in the WKUP domain.

1. Controls the POKs – enable / disable, over-volt / under-volt / ping-pong
2. Samples the continuous output
3. Filters the output

Each POK is controlled, sampled, and filtered by a specific PRG(_PP) (see *POK Modules and Monitored Voltages*) and the control register within that PRG(_PP) is where the POKs are controlled (e.g. CTRL_MMR_PRG_POR_CTRL).

1. Control of the POKs: The POK threshold voltage and hysteresis enable are set in CTRL registers defined in *POK Related Registers*. The PRG(_PP) controls how these POKs are enabled / disabled and how the threshold are applied:
   - Enable / Disable has both;
     a. A global control –POK_EN_SEL (e.g. CTRL_MMR_PRG_PP_MCU_CTRL[15]). POK_EN_SEL disables all controlled POKs when 0 and gives control to the per-POK enable when 1.
     b. A per-POK enable (e.g. CTRL_MMR_PRG_PP_MCU_CTRL[1]. POK_VDDR_MCU_EN)

     To enable a given POK, both levels of enable are required to be set.
   - The options on how thresholds are applied to the POK is the key difference between the PRG_PP and PRG modules. The PRG_PP module will be described in this bullet and then the restriction imposed by the PRG module will be addressed. The PRG_PP can be configured in any of three modes, shown in Table 5-18.
   - Sampling the POKs: The POKs are sampled by a divided-down CLK_12M_RC (WKUP_RC_OSC_12M from the internal RC oscillator). The sampling clock is nominally 12.5MHz (though because it is an RC-generated clock, the frequency is not precise). This input clock is divided by 16 so that sampling records the POK output on the rising edge every ~1.25us.
   - Filtering the output: The output is filtered by means of the DEGLITCH_SEL bitfield (e.g. CTRL_MMR_PRG_PP_MCU_CTRL[17:16]). The bitfield allows the user to select a window of 5us, 10us, 15us or 20us; the POK must be in a FAIL state for ALL samples within the filter window in order for the POK output from the PRG to be seen as FAIL.

Note

DEGLITCH_SEL defines number of samples; the precise duration of the filter is not well controlled.
- For example, the 5us deglitch window samples the POK 4 times and ALL 4 samples must FAIL before the PRG propagates FAIL to the system.
- For example, the 20us deglitch window samples the POK 16 times and ALL 16 samples must FAIL before the PRG propagates FAIL to the system.

Note

PRG in POR does not support ping-pong.

Table 5-18. PRG_PP Modes

Mode:	Example Register / Bitfield
Each POK can be configured to apply the undervoltage threshold continuously. 1. OV_SEL is cleared; the threshold voltage is taken from *UV_CTRL 2. POK_PP_EN is cleared	CTRL_MMR_ POK_VDDR_MCU_UV_CTRL CTRL_MMR_PRG_PP_MCU_CTRL[9]. POK_VDDR_MCU_OV_SEL = 0 CTRL_MMR_PRG_PP_MCU_CTRL[19] POK_PP_EN = 0

Copyright © 2024 Texas Instruments Incorporated

Table 5-18. PRG_PP Modes (continued)

Mode:	Example Register / Bitfield
Each POK can be configured to apply the overvoltage threshold continuously. 1. OV_SEL is set; the threshold voltage is taken from *OV_CTRL 2. POK_PP_EN is cleared	CTRL_MMR_ POK_VDDR_MCU_OV_CTRL CTRL_MMR_PRG_PP_MCU_CTRL[9]. POK_VDDR_MCU_OV_SEL = 1 CTRL_MMR_PRG_PP_MCU_CTRL[19] = 0
All POK can be configured to apply the undervoltage threshold then the overvoltage threshold iteratively (ping-pong) 1. All OV_SEL are cleared 2. POK_PP_EN is set; the threshold voltage is taken from *UV_CTRL in the UV phase; the threshold voltage is taken from *OV_CTRL in the OV phase	CTRL_MMR_ POK_VDDR_MCU_UV_CTRL CTRL_MMR_ POK_VDDR_MCU_OV_CTRL CTRL_MMR_PRG_PP_MCU_CTRL[10:8] = 000b CTRL_MMR_PRG_PP_MCU_CTRL[19] = 1

Copyright © 2024 Texas Instruments Incorporated

5.2.3.1.3 Power on Reset (POR) Module

This section describes the Power on Reset (POR) module.

5.2.3.1.3.1 POR Overview

The POR module should be viewed as a transparent wrapper to the PRG / POKs instantiated within *except* for the following points:

- The bandgap which supplies all POKs is within the POR. The bandgap generates an untrimmed voltage / current after MCU_PORz is released. In order to apply the device-specific trim to the bandgap, the user must set CTRL_MMR_POR_CTRL[7]. TRIM_SEL. The bandgap requires a 100us settling time (during which POK / PRG configuration may occur).
- The device-specific bandgap trim is stored in CTRL_MMR_POR_BANDGAP_CTRL and it is recommended to not modify this trim value.
- The threshold controls for the POKs within the POR are also enabled when TRIM_SEL is set.
- As noted above, the PRG within the POR does not support ping-pong application of thresholds; these POKs monitor a fixed threshold.

5.2.3.1.4 Timing

As noted in above, there are three timings that are critical:

- 100-us bandgap settling
- 15-us POK enable settling can be concurrent to 100-us bandgap settling
- 9-us POK threshold settling which can be concurrent to both the 100-us bandgap settling and the 15us POK settling

Figure 5-5 shows the sampled POK output to the ESM when the POK threshold is ping-ponged.

- A POK settling time of 7 1.25-us clocks is fixed in the design; this provides settling for the new threshold voltage when the POK is ping-ponged
- The deglitch (filter) time can be chosen as {4, 8, 12, 16} 1.25-us clock cycles.
- One 1.25-us clock period provides an update window to the ESM.

In this case, the ESM sees an update of the POK threshold every (7 + {4, 8,12,16} + 1) × 1.25[µs]. Since the ESM sees an update of a threshold every ~15us to every ~30-us, the UV and OV flags are individually being updated every ~30-us to every 60-us.

In the case where the PRG does not ping-pong, a fixed threshold is applied. The deglitch (filter) time can be chosen as {4, 8, 12, 16} 1.25-us clock cycles. The filter is a sliding window that evaluates the filtered output every 1.25-us based upon the last *n* samples (n = {4, 8, 12, 16}). In this case, the ESM sees an update of the POK threshold every 1.25 us.

The POKs are sampled by a divided-down CLK_12M_RC (WKUP_RC_OSC_12M from Section 5.4.4.1.2). The sampling clock is nominally 12.5 MHz (though because it is an RC-generated clock, the frequency is not precise). This input clock is divided by 16 so that sampling records the POK output on the rising edge every ~1.25 us.

Copyright © 2024 Texas Instruments Incorporated

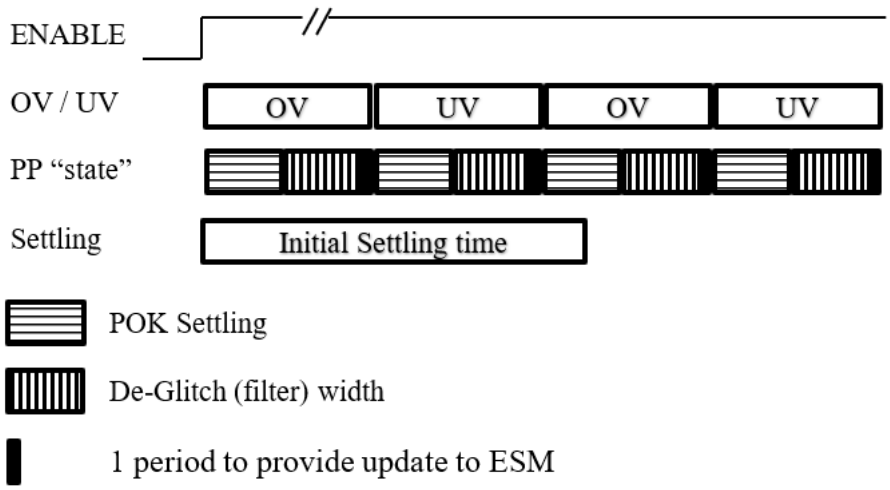


Figure 5-5. Timing for POK with Ping-Pong and Deglitch of 5us

5.2.3.1.5 Restrictions

1. POK monitoring occurs within the initial POK settling (from enable). It is therefore required that the initial state of the POK after enable should be discarded; further, the ESM should be configured for its system response after the initial results during the settling time are discarded.
2. Within a PRG, all POKs must ping-pong or all monitor a static threshold
3. It is not advised to change the threshold from UV to OV while the POK is enabled

Copyright © 2024 Texas Instruments Incorporated

5.2.3.1.6 PRG_PP Programming Model

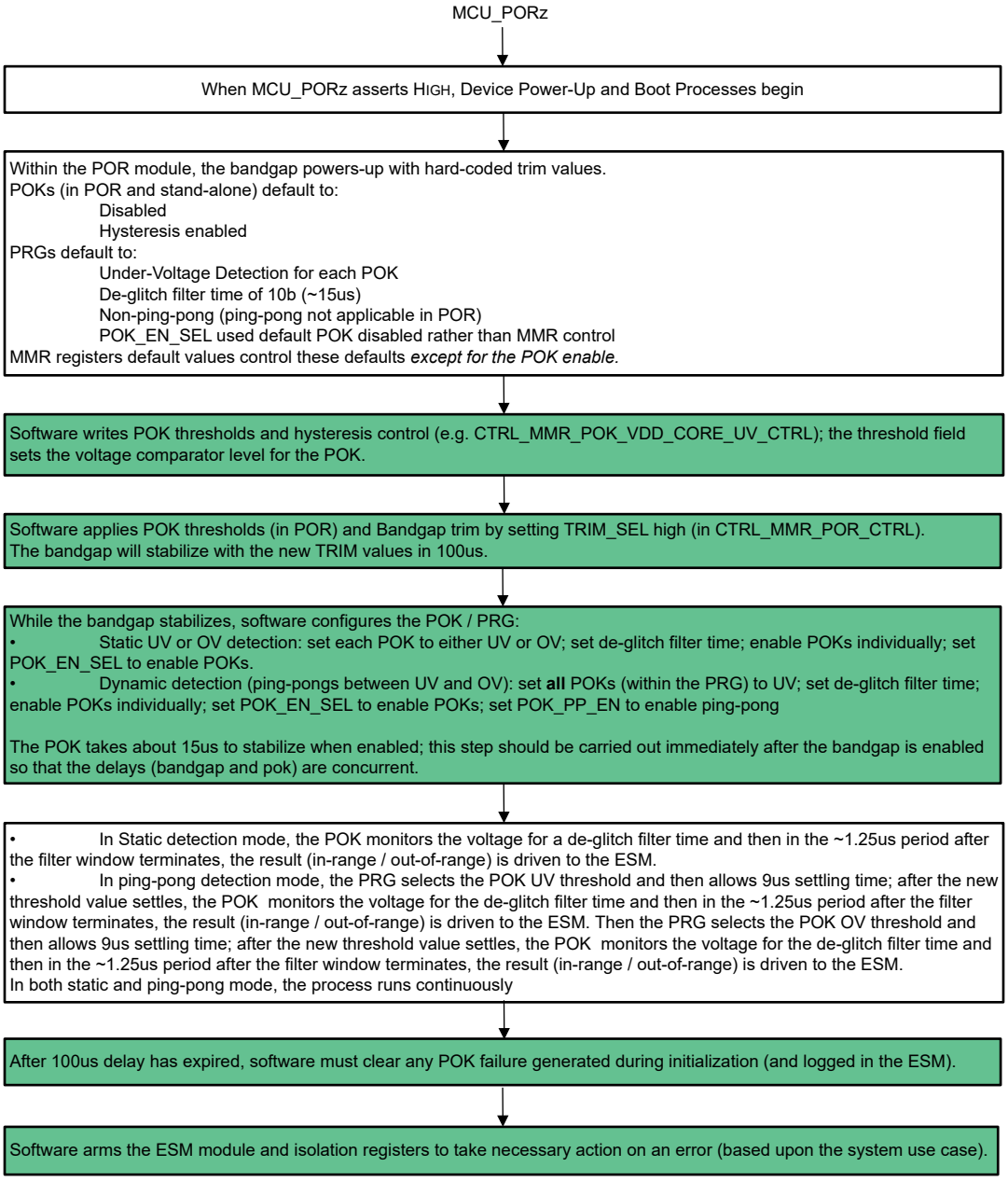


Figure 5-6. PRG_PP Programming Flow

The PRG_PP modules registers used to control and monitor the status of POK modules are:

- CTRL_MMR_PRG_POR_CTRL
- CTRL_MMR_PRG_PP_MCU_CTRL
- CTRL_MMR_PRG_PP_MAIN_CTRL

For more information about control registers, see *Control Module (CTRL_MMR)*.

Copyright © 2024 Texas Instruments Incorporated

5.2.3.2 Power Glitch Detect (PGD) Modules

A Power Glitch Detect (PGD) circuit is used to detect short duration “glitches” on the core and MPU power supplies. The PGD provides a low output when no glitch is detected and a high output when a glitch is detected.

Figure 5-7 shows the PGD block diagram.

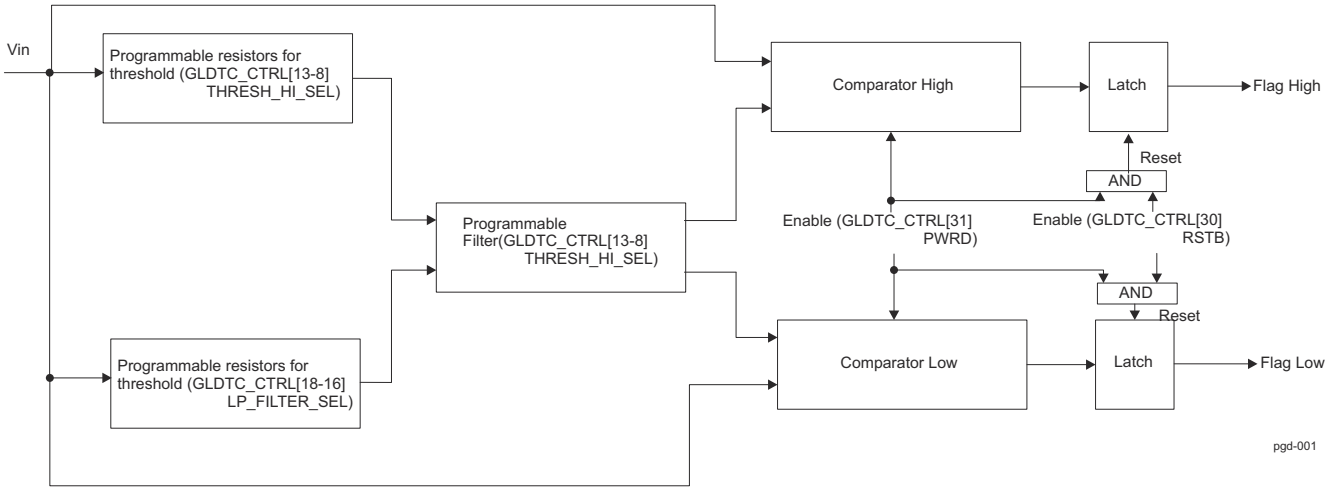


Figure 5-7. PGD Block Diagram

Table 5-19 summarize the PGD integration.

Table 5-19. PGD Integration Summary

Module Instance	Monitored Voltage	PGD Control/Status Registers [(1)]
IPGD_VDD_MCU_CORE	VDD_MCU	CTRL_MMR_VDD_MCU_GLDTC_CTRLCTRL_MMR_VDD_MCU _GLDTC_STAT
IPGD_VDD_MCU_MEM	VDDR_MCU (SRAM)	CTRL_MMR_VDDR_MCU_GLDTC_CTRLCTRL_MMR_VDDR_M CU_GLDTC_STAT
IPGD_VDD_CPU_CORE	VDD_CPU	CTRL_MMR_VDD_CPU_GLDTC_CTRLCTRL_MMR_VDD_CPU_ GLDTC_STAT
IPGD_VDD_MAIN_CORE	VDD_CORE	CTRL_MMR_VDD_CORE_GLDTC_CTRLCTRL_MMR_VDD_CO RE_GLDTC_STAT
IPGD_VDD_CPU_MEM	VDDR_CPU (SRAM)	CTRL_MMR_VDDR_CPU_GLDTC_CTRLCTRL_MMR_VDDR_CP U_GLDTC_STAT
IPGD_VDD_MAIN_MEM	VDDR_CORE (SRAM)	CTRL_MMR_VDDR_CORE_GLDTC_CTRLCTRL_MMR_VDDR_C ORE_GLDTC_STAT

(1) For more information about control registers, see *Control Module (CTRL_MMR)*.

Note

A flag status bit in corresponding STAT register is cleared by clearing the [30] RSTB bit in the corresponding CTRL register, see for STAT and CTRL registers for a certain PGD module.

Copyright © 2024 Texas Instruments Incorporated

5.2.3.3 Voltage and Thermal Manager (VTM)

5.2.3.3.1 VTM Overview

This section describes the Voltage and Thermal Manager (VTM) module in the device.

In the same way that PRGs control and provide output from the POKs, the VTM controls and outputs signals from temperature sensors that are spread around the die.VTM provides the control, status as well as interrupt and event generation related to integrated temperature sensors and thermal events programed by the user. VTM also contains a set of memory mapped registers that store device-specific operating voltages (AVSVNOM) set during device testing. These values may be used for the system AVS software to adjust operating voltages for the device to achieve optimal operating power.

The device supports one VTM module - WKUP_VTM0. Table 5-20 shows VTM module allocation within device domains.

Table 5-20. VTM Module Allocation within Device Domains

Module Instance	Domain		
	WKUP	MCU	MAIN
WKUP_VTM0	✓	-	-

5.2.3.3.1.1 VTM Features

VTM module supports the following features:

- AVS-Class0 only
- Supports up to 8 temperature monitors
- Programming of temperature-crossing thresholds
- Signals when programed thresholds are exceeded - up to 3 alerts:
  - Three full reference 10-bit temperature threshold points, THPT1, THPT2 and THPT0.
- Supports up to 8 temperature monitors
- Allows resolution of 0.5°C for temperature reading and threshold point temperature alert/interrupt generation.
- Supports PMIC/LDO set-up with Class-0 VDD-VID settings
- Support of tentative customization of OPP voltage per device (in support of multiple OPPs)
- AVS-voltage or thermal management for up to 8 voltage domains
- Maximum temperature alert
- Supports one shot sampling mode and continuous monitoring mode for the sensors

Note

The VTM of this family of devices has only 7 temperature sensors and 3 voltage domains.

The ‘j’ notation in the following registers increment from 0 to 6 to map to the 7 temp sensors on the SoC. (Refer to Table 5-25):

- TMPSENS_CTRL_j
- TMPSENS_STAT_j
- TMPSENS_TH_j
- TMPSENS_TH2_j

The ‘j’ notation in the following registers increment from 0 to 2 to map to the 3 voltage domains on the SoC (refer to Table 5-21):

- VD_EVT_SEL_SET_j
- VD_EVT_SEL_CLR_j

5.2.3.3.1.2 VTM Not Supported Features

- No support for AVS-Class0 with temperature compensation (AVS0 + TC)
- No integrated I2C inside VTM
- No direct hardware triggering of I2C transactions by VTM
- No support for voltage and thermal management of IO voltage domains
- No oversampling of the temperature reading

Copyright © 2024 Texas Instruments Incorporated

- No support for eFuse defaults of all register values

Copyright © 2024 Texas Instruments Incorporated

5.2.3.3.2 VTM Functional Description

In the same way that PRGs control and provide output from the POKs, the VTM controls and outputs signals from several temperature sensors that are spread around the die. The VTM provides three interrupts / ESM inputs to the system as well as a warm reset.

The temperature sensor is configured to run off a clock frequency between 1.15MHz and 1.25MHz. The frequency is obtained by dividing-down HFOSC0 or the internal RC clock; the expected usage is that the temperature sensors use HFOSC0 (TSENS_CLK_DIV is the divider). The temperature sensor will sample the temperature on a period defined by VTM_VTM_SAMPLE_CTRL.

When the temperature sensor is enabled, it senses the temperature and reports the temperature as a 10-bit value.

- The raw 10-bit value is reported in VTM_VTM_TMPSENS_STAT_j;
- It is also compared to pre-defined thresholds (each temperature sensor has its own set thresholds set in VTM_VTM_TMPSENS_TH_j and VTM_VTM_TMPSENS_TH2_j) ;
- It is also compared to a pre-defined global threshold set in VTM_VTM_MISC_CTRL2.

Each of these comparisons can be individually enabled (VTM_VTM_TMPSENS_CTRL_j and VTM_VTM_TMPSENS_CTRL_j within mmr_vbusp_cfg2). Additionally, each temperature sensor has status bits for these comparator outputs in VTM_VTM_TMPSENS_STAT_j.

The comparator functionality that drives the interrupts and warm reset work only when the sensor is configured for continuous mode; continuous operation becomes a de facto requirement (and is set in VTM_VTM_TMPSENS_CTRL_j[4].CONT within mmr_vbusp_cfg2 – that is, the mode is set for *each* temperature sensor).

Interrupts / ESM inputs:

The comparator thresholds shall always configured so that TH2 > TH1 > TH0. The concept is:

- TH1 is configured as an early alarm to indicate that the temperature is greater than the threshold defined in VTM_VTM_TMPSENS_TH_j[25-16] TH1_VAL.
- TH2 is configured as a warning to indicate that the temperature is greater than the threshold defined in VTM_VTM_TMPSENS_TH2_j[9-0] TH2_VAL. The concept is that TH1 indicates that the processor is getting hot; TH2 demands immediate attention.
- TH0 is configured to trigger when the sensor detects the temperature less than the threshold defined in VTM_VTM_TMPSENS_TH_j[9-0] TH0_VAL. This interrupt indicates that the thermal mitigations can be relaxed because the temperature has fallen below the original TH1 level.

LT_TH0_INT, if enabled, will get triggered always when the temperature being read is less than TH0, regardless of whether TH1 interrupt and TH2 interrupt are enabled or have ever been triggered. Therefore if TH0 interrupt is generated, then firmware/software is responsible to enable the TH0 interrupt only as part of the interrupt service routine of TH1 and TH2. Otherwise it will keep triggering when is not needed.

Warm Reset input:

Similar to the interrupts / ESM inputs, the warm reset generation is triggered off a threshold defined in VTM_VTM_MISC_CTRL2[9-0] MAXT_OUTRG_ALERT_THR. The reset will be released when the compared temperature drops below VTM_VTM_MISC_CTRL2[25-16] MAXT_OUTRG_ALERT_THR0.

VTM muxing of the temperature comparisons:

The VTM is configured by voltage domains. At this level, all the interrupt / ESM inputs from each temperature sensor can be enabled or disabled; as an example, the comparator outputs from temperature sensor 0 can be ignored in voltage domain 0. The enable is configured with the VTM_VTM_VD_EVT_SEL_SET_j and VTM_VTM_VD_EVT_SEL_CLR_j registers. At the voltage domain-level, the enabled comparator results are combined by interrupt / ESM signal. The combined comparator result can be observed in VTM_VTM_VD_EVT_STAT_j. As is shown in the following table, j increments over voltage domains for these registers (see Table 5-21). The registers TMPSENS_*_j map to the individual temp sensors, as shown in Table 5-25.

Copyright © 2024 Texas Instruments Incorporated

Table 5-21. VTM Voltage Domains

Register Group	Voltage Domain
WKUP_VTM_*_VD_*_0	VDD_MCU
WKUP_VTM_*_VD_*_1	VDD_CORE
WKUP_VTM_*_VD_*_2	VDD_CPU_AVS
others	Unused

After the temperature sensors are combined at a voltage domain level, the outputs from each voltage domain are enabled at the top level of the VTM with VTM_VTM_GT_TH2_INT_EN_SET / VTM_VTM_GT_TH2_INT_EN_CLR, VTM_VTM_GT_TH1_INT_EN_SET / VTM_VTM_GT_TH1_INT_EN_CLR, and VTM_VTM_LT_TH0_INT_EN_SET / VTM_VTM_LT_TH0_INT_EN_CLR. At this top level of the VTM, the status of these signals is checked and cleared with VTM_VTM_GT_TH2_INT_RAW_STAT_SET / VTM_VTM_GT_TH2_INT_EN_STAT_CLR, VTM_VTM_GT_TH1_INT_RAW_STAT_SET / VTM_VTM_GT_TH1_INT_EN_STAT_CLR, and VTM_VTM_LT_TH0_INT_RAW_STAT_SET / VTM_VTM_LT_TH0_INT_EN_STAT_CLR.

VTM control of the global comparison that generates a warm reset:

The global comparison that generates a warm rest is enabled at each temperatures sensor with WKUP_VTM_TMPSENS_CTRL_j[11] within mmr_vbusp_cfg2. Additionally, there is a global enable for this signal in WKUP_VTM_MISC_CTRL[0].

AVS voltage definition:

The VTM also contains the Adaptive Voltage Scaling (AVS) voltage for the VDD_CPU domain. Since the VDD_CPU domain is defined by voltage domain 2, the registers of interest are VTM_VTM_VD_DEVINFO_2 and VTM_VTM_VD_OPPVID_2. If VTM_VTM_VD_DEVINFO_2[12] AVS0_SUP is set, then the device should set the voltage defined in VTM_VTM_VD_OPPVID_2[15-8] OPP_1. The voltage is encoded according to the table below:

Table 5-22. Output Voltage Selection for BUCK Regulators

BUCKn_ VSETn	Output Voltage [V] 20 mV steps	BUCKn_ VSETn	Output Voltage [V] 5 mV steps	BUCKn_ VSETn	Output Voltage [V] 5 mV steps	BUCKn_ VSETn	Output Voltage [V] 10 mV steps	BUCKn_ VSETn	Output Voltage [V] 20 mV steps	BUCKn_ VSETn	Output Voltage [V] 20 mV steps
0x00	0.3	0x0F	0.6	0x41	0.85	0x73	1.1	0xAB	1.66	0xD6	2.52
0x01	0.32	0x10	0.605	0x42	0.855	0x74	1.11	0xAC	1.68	0xD7	2.54
0x02	0.34	0x11	0.61	0x43	0.86	0x75	1.12	0xAD	1.7	0xD8	2.56
0x03	0.36	0x12	0.615	0x44	0.865	0x76	1.13	0xAE	1.72	0xD9	2.58
0x04	0.38	0x13	0.62	0x45	0.87	0x77	1.14	0xAF	1.74	0xDA	2.6
0x05	0.4	0x14	0.625	0x46	0.875	0x78	1.15	0xB0	1.76	0xDB	2.62
0x06	0.42	0x15	0.63	0x47	0.88	0x79	1.16	0xB1	1.78	0xDC	2.64
0x07	0.44	0x16	0.635	0x48	0.885	0x7A	1.17	0xB2	1.8	0xDD	2.66
0x08	0.46	0x17	0.64	0x49	0.89	0x7B	1.18	0xB3	1.82	0xDE	2.68
0x09	0.48	0x18	0.645	0x4A	0.895	0x7C	1.19	0xB4	1.84	0xDF	2.7
0x0A	0.5	0x19	0.65	0x4B	0.9	0x7D	1.2	0xB5	1.86	0xE0	2.72
0x0B	0.52	0x1A	0.655	0x4C	0.905	0x7E	1.21	0xB6	1.88	0xE1	2.74
0x0C	0.54	0x1B	0.66	0x4D	0.91	0x7F	1.22	0xB7	1.9	0xE2	2.76
0x0D	0.56	0x1C	0.665	0x4E	0.915	0x80	1.23	0xB8	1.92	0xE3	2.78
0x0E	0.58	0x1D	0.67	0x4F	0.92	0x81	1.24	0xB9	1.94	0xE4	2.8
		0x1E	0.675	0x50	0.925	0x82	1.25	0xBA	1.96	0xE5	2.82
		0x1F	0.68	0x51	0.93	0x83	1.26	0xBB	1.98	0xE6	2.84
		0x20	0.685	0x52	0.935	0x84	1.27	0xBC	2	0xE7	2.86
		0x21	0.69	0x53	0.94	0x85	1.28	0xBD	2.02	0xE8	2.88

Copyright © 2024 Texas Instruments Incorporated

Table 5-22. Output Voltage Selection for BUCK Regulators (continued)

BUCKn_ VSETn	Output Voltage [V] 20 mV steps	BUCKn_ VSETn	Output Voltage [V] 5 mV steps	BUCKn_ VSETn	Output Voltage [V] 5 mV steps	BUCKn_ VSETn	Output Voltage [V] 10 mV steps	BUCKn_ VSETn	Output Voltage [V] 20 mV steps	BUCKn_ VSETn	Output Voltage [V] 20 mV steps
		0x22	0.695	0x54	0.945	0x86	1.29	0xBE	2.04	0xE9	2.9
		0x23	0.7	0x55	0.95	0x87	1.3	0xBF	2.06	0xEA	2.92
		0x24	0.705	0x56	0.955	0x88	1.31	0xC0	2.08	0xEB	2.94
		0x25	0.71	0x57	0.96	0x89	1.32	0xC1	2.1	0xEC	2.96
		0x26	0.715	0x58	0.965	0x8A	1.33	0xC2	2.12	0xED	2.98
		0x27	0.72	0x59	0.97	0x8B	1.34	0xC3	2.14	0xEE	3.0
		0x28	0.725	0x5A	0.975	0x8C	1.35	0xC4	2.16	0xEF	3.02
		0x29	0.73	0x5B	0.98	0x8D	1.36	0xC5	2.18	0xF0	3.04
		0x2A	0.735	0x5C	0.985	0x8E	1.37	0xC6	2.2	0xF1	3.06
		0x2B	0.74	0x5D	0.99	0x8F	1.38	0xC7	2.22	0xF2	3.08
		0x2C	0.745	0x5E	0.995	0x90	1.39	0xC8	2.24	0xF3	3.1
		0x2D	0.75	0x5F	1.0	0x91	1.4	0xC9	2.26	0xF4	3.12
		0x2E	0.755	0x60	1.005	0x92	1.41	0xCA	2.28	0xF5	3.14
		0x2F	0.76	0x61	1.01	0x93	1.42	0xCB	2.3	0xF6	3.16
		0x30	0.765	0x62	1.015	0x94	1.43	0xCC	2.32	0xF7	3.18
		0x31	0.77	0x63	1.02	0x95	1.44	0xCD	2.34	0xF8	3.2
		0x32	0.775	0x64	1.025	0x96	1.45	0xCE	2.36	0xF9	3.22
		0x33	0.78	0x65	1.03	0x97	1.46	0xCF	2.38	0xFA	3.24
		0x34	0.785	0x66	1.035	0x98	1.47	0xD0	2.4	0xFB	3.26
		0x35	0.79	0x67	1.04	0x99	1.48	0xD1	2.42	0xFC	3.28
		0x36	0.795	0x68	1.045	0x9A	1.49	0xD2	2.44	0xFD	3.3
		0x37	0.8	0x69	1.05	0x9B	1.5	0xD3	2.46	0xFE	3.32
		0x38	0.805	0x6A	1.055	0x9C	1.51	0xD4	2.48	0xFF	3.34
		0x39	0.81	0x6B	1.06	0x9D	1.52	0xD5	2.5		
		0x3A	0.815	0x6C	1.065	0x9E	1.53				
		0x3B	0.82	0x6D	1.07	0x9F	1.54				
		0x3C	0.825	0x6E	1.075	0xA0	1.55				
		0x3D	0.83	0x6F	1.08	0xA1	1.56				
		0x3E	0.835	0x70	1.085	0xA2	1.57				
		0x3F	0.84	0x71	1.09	0xA3	1.58				
		0x40	0.845	0x72	1.095	0xA4	1.59				
						0xA5	1.6				
						0xA6	1.61				
						0xA7	1.62				
						0xA8	1.63				
						0xA9	1.64				
						0xAA	1.65				

5.2.3.3.2.1 VTM Temperature Status and Thermal Management

VTM module provides temperature reading and interrupt/alert information for thermal management.

5.2.3.3.2.1.1 10-bit Temperature Values Versus Temperature

Table 5-23 shows a table lookup method for translating code to temperature from temperature monitors which correspond to the temperature measured read from the VTM0_VTM_TMPSENS_STAT_J [9-0] DATA_OUT

Copyright © 2024 Texas Instruments Incorporated

bitfields. Table 5-23 also provides the values for the temperatures coded in VTM0_VTM_MISC_CTRL2 [25-16] MAXT_OUTRG_ALERT_THR0 and VTM0_VTM_MISC_CTRL2 [9-0] MAXT_OUTRG_ALERT_THR thresholds.

Table 5-23. Temperature Translation Table

Parameter Type	Value								
Temperature [°C] (rounded)	-40	-25	0	25	50	75	100	125	150
Temperature [°C] (unrounded)	-40.03	-24.95	0.01	25.02	49.94	75.03	100.02	124.97	150.00
DATA_OUT Code	28	77	164	260	366	485	620	773	949

Table 5-24 shows a table that gives coefficients of polynomial to calculate code or temperature. The equation to calculate code or temperature is:

$y = a4 \times x^4 + a3 \times x^3 + a2 \times x^2 + a1 \times x + a0$

Table 5-24. Equation Method to Calculate Code or Temperature

Function	Unit	Coefficient				
		a4	a3	a2	a1	a0
Code = f(Temperature)	Decimal value of code	4.9847E-08	1.6972E-05	6.8759E-03	3.6509E+00	1.6407E+02
Temperature = f(Code)	°C	-9.2627E-12	6.0373E-08	-1.7058E-04	3.2512E-01	-4.9002E+01

5.2.3.3.2.2 VTM Temperature Driven Alerts and Interrupts

All 3 alerts from the 7 (5 shown in sketch) temperature monitors are available as inputs of the mask and alert merging logic block of each voltage domain, such that the temperature monitors that are relevant to each voltage domain can be selected as the contributors to the generation of the 3 combined interrupts in each voltage domain (5 shown in sketch; 3 are present in the device). This logic is presented in Figure 5-8. The THERM_MAXTEMP_OUTRANGE_ALERT is not shown in this figure. Notice that the same temperature sensor can contribute to more than one voltage domain and each voltage domain can have multiple sensors contributing to the interrupt generation in that VD, see WKUP_VTM_VD_EVT_SET_j and WKUP_VTM_VD_EVT_CLR_j registers.

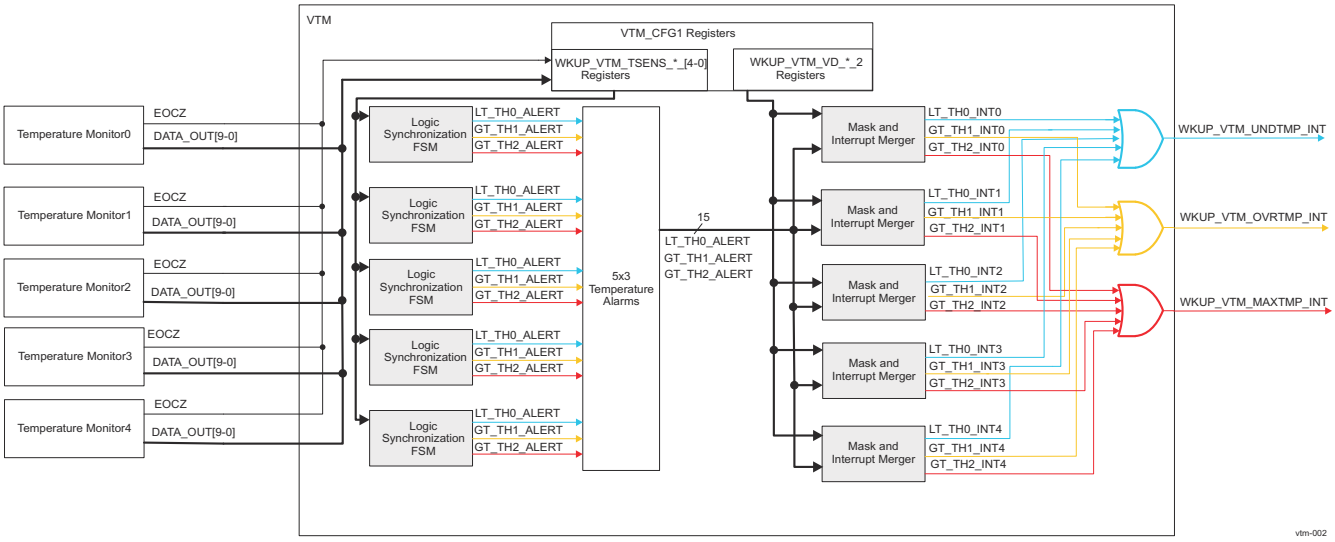


Figure 5-8. VTM Alert and Interrupt Generation

Copyright © 2024 Texas Instruments Incorporated

Note

The interrupts are only active when the sensor is in continuous mode. A one-shot sampling of the sensor will not trigger any interrupts.

Table 5-25 presents the connection of VTM TEMPSENSOR registers groups to voltage domains.

Table 5-25. VTM TEMPSENSOR Register Groups Mapping to Voltage Domains

Register Group	Voltage Domain
WKUP_VTM_TMPSENS_*_0	Near MCU_R5FSS in VDD_MCU
WKUP_VTM_TMPSENS_*_1	On VDD_MCU / VDD_CORE (DDR) boundary
WKUP_VTM_TMPSENS_*_2	Near CODEC in VDD_CORE
WKUP_VTM_TMPSENS_*_3	Near DPHYs in VDD_CORE
WKUP_VTM_TMPSENS_*_4	Near R5FSS on VDD_CPU / VDD_CORE boundary
WKUP_VTM_TMPSENS_*_5	Near A72 and GPU on VDD_CPU / VDD_CORE boundary
WKUP_VTM_TMPSENS_*_6	Near C7x and LPDDR4 on VDD_CPU / VDD_CORE (DDR) boundary

5.2.3.3.2.3 VTM ECC Aggregator

Note

For more information about ECC Aggregator functionality, see *ECC Aggregator*.

5.2.3.3.2.4 VTM Programming Model

5.2.3.3.2.4.1 VTM Maximum Temperature Outrange Alert

THERM_MAXTEMP_OUTRANGE_ALERT is used at the device level in order to generate device warm reset. The alert is asserted when the device overheats and exceeds the maximum temperature limit specified for the device, namely 125°C.

The following is the sequence to assert the THERM_MAXTEMP_OUTRANGE_ALERT:

1. For the output to go active (‘1’), the VTM has to be configured to operate with at least one of its temperature sensors enabled, and WKUP_VTM_TMPSENS_CTRL_j[11] MAXT_OUTRG_EN = 1 for the corresponding enabled sensor.
2. In addition, WKUP_VTM_MISC_CTRL[0] ANYMAXT_OUTRG_ALERT_EN has to be set to ‘1’.
3. With the previous configuration set and at least one of the temperature sensors properly configured, and enabled, if the temperature of the die ever exceeds the programmed value in WKUP_VTM_MISC_CTRL2, the VTM output will be driven ‘1’ once the sensor detects the programmed maximum temperature.
4. At the device level the VTM output, THERM_MAXTEMP_OUTRANGE_ALERT, will drive the PLL controllers into warm reset state with the PLL controllers entering warm reset condition, at the same time that all the PLLs go into clock bypass mode.
5. By the actions described in No.4, the device will considerably reduce its power consumption, which over the passing of a few seconds will cause the device to reduce its temperature.
6. The VTM sensor continues to read the temperature and once the code corresponding to the programmed value in WKUP_VTM_MISC_CTRL2[25-16] MAXT_OUTRG_ALERT_THR0 is detected the VTM drives THERM_MAXTEMP_OUTRANGE_ALERT = 0.
7. After No.6 is fulfilled, the PLL controllers no longer have the THERM_MAXTEMP_OUTRANGE_ALERT driving the warm reset request and thus the PLL controllers bring the SoC out of reset and the boot sequence re-starts. At the same time the PLLs get out of bypass mode and start with their target programmed frequencies for the boot process.

Note

THERM_MAXTEMP_OUTRANGE_ALERT output of VTM is mapped to WKUP_VTM_MISC_CTRL[0] ANY_MAXT_OUTRG_ALERT_EN.

Copyright © 2024 Texas Instruments Incorporated

Notice that exceeding of the programmed value in WKUP_VTM_MISC_CTRL2[9-0] MAXT_OUTRG_ALERT_THR boundary doesn't result in a PORz (cold reset) event, but instead a warm reset event. Otherwise if cold reset is applied to the MCU/WKUP domains of the device, the VTM will be fully reset and the output THERM_MAXTEMP_OUTRANGE_ALERT will be cleared to '0' immediately.

5.2.3.3.2.4.2 Sensors Programming Sequences

Sensors Control Modifications Sequence:

1. Write changes to registers.
2. Wait for 900ns to 1us (long enough for sensor period of slowest 1.15 MHz sensor clock plus some extra).
3. New changes can be written to registers.

Reset Sensors Sequence:

1. Set the WKUP_VTM_TMPSENS_CTRL_j[6] CLRZ bit to enable the sensor (so that it has the ENA input high).
2. Wait for 900ns to 1us for the sensor clock to toggle and register the updated ENA pin.
3. Clear WKUP_VTM_TMPSENS_CTRL_j[6] CLRZ to disable and reset the sensor (as it requires an ENA transition from 1 to 0 to reset initially) .
4. Wait for 900ns to 1us for the sensor clock to toggle and register the updated ENA pin.
5. Now the sensor can be configured and enabled normally.

5.2.3.3.2.5 AVS-Class0

Adaptive Voltage Scaling (AVS) Class 0 is a procedure for lowering the voltage on certain device power rails. AVS Class 0 attempts to normalize the power consumption across all devices. The optimal voltage for each AVS supported rail of each device is determined after analysis in the factory, based on the strength of the device during manufacturing. This value is written in the device eFuse where it can be read through dedicated registers.

AVS can be enabled by system software after primary boot. Refer to *AVS Support* on how to enable AVS based on these registers.

Copyright © 2024 Texas Instruments Incorporated

5.2.4 Dynamic Power Management

5.2.4.1 AVS Support

The device supports only Adaptive Voltage Scaling (AVS) Class 0. This is a procedure for lowering the voltage on certain device power rails. AVS-Class0 attempts to normalize the power consumption across all devices. The optimal voltage for each AVS supported rail of each device is determined after analysis in the factory, based on the strength of the device during manufacturing. This value is written in the device eFuse where it can be read through dedicated registers.

The following supplies support AVS-Class0:

- VDD_CPU_AVS

The AVS voltage for a supply is an E-fuse VID value located in the corresponding WKUP_VTM_VD_OPPVID_j register in the VTM module. *VID Bit-field Values and Their Corresponding Voltage* lists the mapping of the supplies to the VTM control registers. Separate VID values are fused for the different OPPs that the supply supports. For more information about OPPs suported by a supply in the device, see the device-specific DataSheet.

The AVS functionality can be enabled and performed by software after the primary boot of the device is finished. Typically, the voltage adjustment of AVS operation is done as secondary boot software.

Copyright © 2024 Texas Instruments Incorporated

5.3 Reset

This chapter describes the device reset management.

Terminology

- DMSC - Device Management Security Controller
- PSC - Power Sleep Controller
- LPSC - Local Power Sleep Controller
- ESM - Error Signaling Module
- Isolation – the module is exempted from reset

5.3.1 Reset Overview

Resets bring the device or part of the device to a known state after events such as power-up or hardware or software requests.

Figure 5-9 shows an overview of the reset architecture.

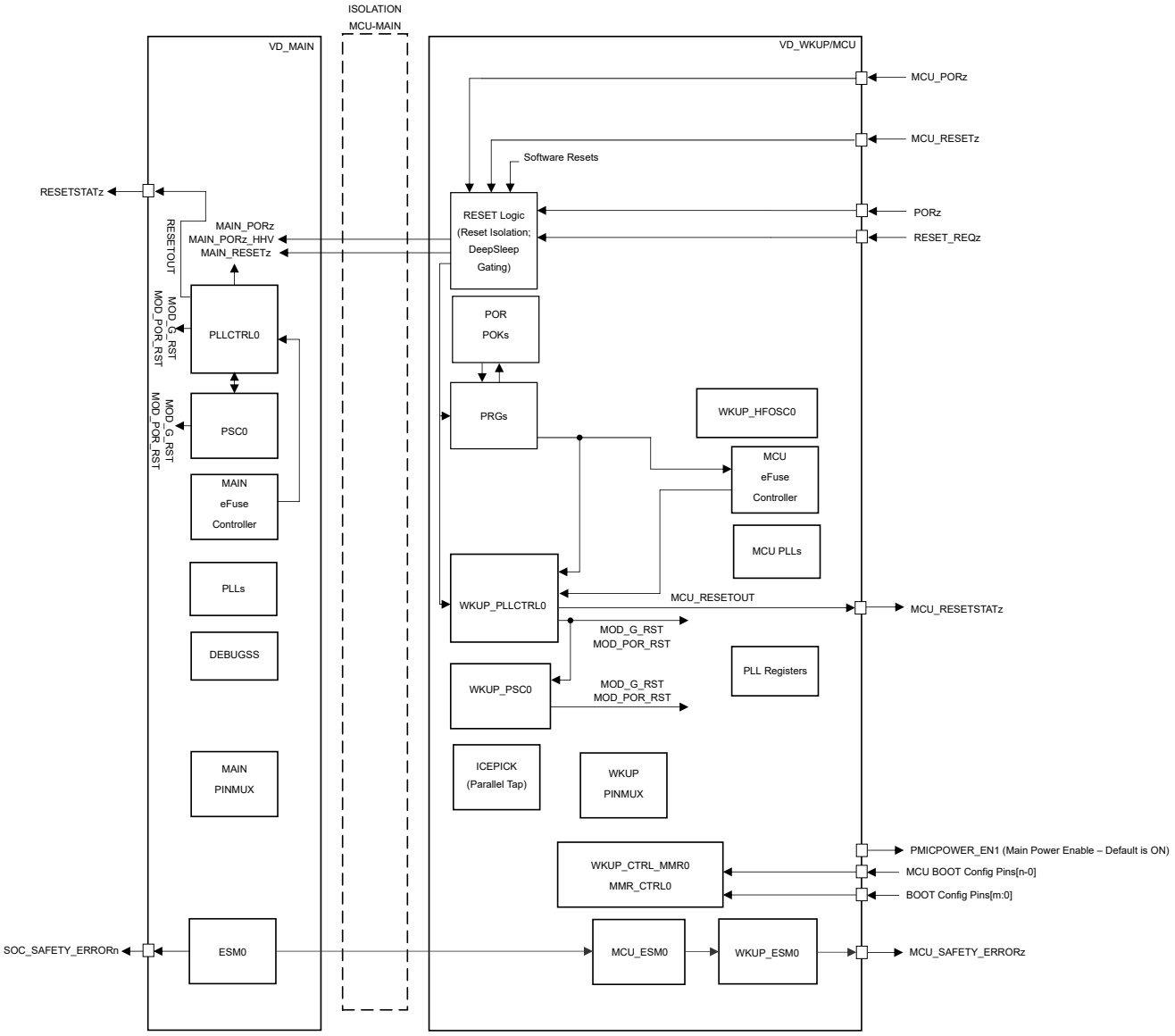


Figure 5-9. Reset Architecture

Copyright © 2024 Texas Instruments Incorporated

Note

For power modules used in the reset see:

- For internal PORz detection, see *Power on Reset (POR) Module*, in *Power.*
- For Power Reset Generator (PRG), see *Power Reset Generator (PRG) Modules*, in *Power.*
- For POK, see *POK Modules*, in *Power.*
- For Power Glitch Detector (PGD), see *Power Glitch Detector (PGD) Modules*, in *Power.*

Note

For information about Boot modes and Boot configuration pins, see *Initialization*.

Note

MOD_POR_RST[n] and MOD_G_RST[n] are connected to certain modules. For more information which is the number of the LPSC of a certain module, see its specific chapter.

5.3.2 Reset Modules

Table 5-26. Reset Modules

Generating Module	Generated Reset	Reset Module	Port in Reset Module	Description
MAIN_DEBUGSS_0	sysr_releasefromwir	WKUP_PLL_CTRL_0	dbg_releasewir_pi	ReleaseWir Input From DebugSS
MAIN_DEBUGSS_0	sysr_waitinreset	WKUP_PLL_CTRL_0	dbg_waitinreset_pi	WaitinReset Input From DebugSS
MAIN_DEBUGSS_0	sysr_releasefromwir	MAIN_PLL_CTRL_0	dbg_releasewir_pi	Releasewir Input From DebugSS
MAIN_DEBUGSS_0	sysr_waitinreset	MAIN_PLL_CTRL_0	dbg_waitinreset_pi	WaitinReset Input From DebugSS
from MCU_PORST pin	Reset_Glue_Review_Reset_uARC	WKUP_SMS_0	sms_custom_por_early_rst_n	Local Reset. Refer Reset uARCH
from MCU_PORST pin	Reset_Glue_Review_Reset_uARC	WKUP_SMS_0	tifs_custom_mod_l_rst_n	Early POR Reset. This reset signal should be gated by autoload done signal from Trim efuse. Refer to Reset uARCH
WKUP_PLL_CTRL_0	rstctl_chip_0_early_rst_n	MAIN_PLL_CTRL_0	vbus_slv_rst_n_chip_rst_n	WKUP PLL CTRL VBUS Reset
WKUP_PLL_CTRL_0	rstctl_por_rst_n	WKUP_CTRL_MMR_0	sys_por_mod_por_rst_n	Wakeup MMR Control SYS POR Reset
WKUP_PLL_CTRL_0	rstctl_por_rst_n	WKUP_ICEMELTER_0	mod_por_rst_n	Icemelter Power-on-Reset (see Reset spec for additional details)
WKUP_PLL_CTRL_0	rstctl_por_boot_cfg_rst_n	WKUP_ICEMELTER_0	por_boot_cfg_rst_n	Icemelter Boot CFG POR RST (see Reset spec for additional details)
WKUP_PLL_CTRL_0	rstctl_por_early_rst_n	WKUP_ICEMELTER_0	por_early_rst_n	Icemelter POR Early Reset (see Reset spec for additional details)
WKUP_PLL_CTRL_0	rstctl_por_rst_n	MCU_CTRL_MMR_0	sys_por_mod_g_rst_n	MCU MMR Control SYS POR Reset
WKUP_PLL_CTRL_0	rstctl_por_rst_n	MCU_EFUSE_0	efuse_ctrl_rst_mod_g_rst_n	WKUP eFuse Module Reset
WKUP_PLL_CTRL_0	rstctl_por_rst_n	MCU_SEC_MMR_0	sys_por_mod_g_rst_n	MCU Security MMR Control SYS POR Reset
MAIN_PLL_CTRL_0	rstctl_chip_0_early_rst_n	MAIN_PLL_CTRL_0	vbus_slv_rst_n_chip_rst_n	MAIN PLL CTRL VBUS Reset
MAIN_PLL_CTRL_0	rstctl_por_boot_cfg_rst_n	MAIN_GTC_0	por_boot_cfg_rst_n	Por that is unstretched and not delayed by any sequential logic. Used for boot config to sample device pins
MAIN_PLL_CTRL_0	rstctl_por_boot_cfg_rst_n	MAIN_CTRL_MMR_0	por_boot_cfg_rst_n	Main Ctrl MMR Boot CFG POR Reset
MAIN_PLL_CTRL_0	rstctl_por_rst_n	MAIN_CTRL_MMR_0	sys_por_mod_por_rst_n	Main Ctrl Module POR Reset
MAIN_PLL_CTRL_0	rstctl_por_boot_cfg_rst_n	MAIN_DEBUGSS_0	dbgssr_por_boot_cfg_rst_n	DebugSS Boot CFG Reset
MAIN_PLL_CTRL_0	rstctl_por_rst_n	MAIN_EFUSE_CTRL_0	efuse_ctrl_rst_mod_g_rst_n	Main eFuse Module Reset
MAIN_PLL_CTRL_0	rstctl_por_rst_n	MAIN_SEC_MMR_0	sys_por_mod_g_rst_n	Main Sec MMR SYS POR Reset
WKUP_PSC_0	psc_mod_wklp_wkup_alwayson_rst_mod_g_rst_n	WKUP_DDPA_0	rstn_mod_g_rst_n	DDPA Module Reset
WKUP_PSC_0	psc_mod_wklp_wkup_alwayson_rst_mod_g_rst_n	WKUP_ESM_0	mod_g_rst_n	
WKUP_PSC_0	psc_mod_wklp_wkup_alwayson_rst_mod_por_rst_n	WKUP_ESM_0	mod_por_rst_n	WKUP ESM0 Module Reset
WKUP_PSC_0	psc_mod_wklp_wkup_gpio_rst_mod_g_rst_n	WKUP_GPIO_0	mod_g_rst_n	Wakeup GPIO0 Module Reset

Copyright © 2024 Texas Instruments Incorporated

Table 5-26. Reset Modules (continued)

Generating Module	Generated Reset	Reset Module	Port in Reset Module	Description
WKUP_PSC_0	psc_mod_wklp_wkup_gpio_rst_mod_g_rst_n	WKUP_GPIO_1	mod_g_rst_n	Wakeup GPIO1 Module Reset
WKUP_PSC_0	psc_mod_wklp_wkup_gpio_rst_mod_g_rst_n	WKUP_I2C_0	mod_g_rst_n	WKUP I2C0 Module Reset
WKUP_PSC_0	psc_mod_wklp_wkup_gpio_rst_mod_g_rst_n	WKUP_UART_0	usart_mod_g_rst_n	WKUP UART0 Module Reset
WKUP_PSC_0	psc_mod_wklp_wkup_alwayson_rst_mod_g_rst_n	WKUP_VTM_0	mod_g_rst_n	VTM Module Reset
WKUP_PSC_0	psc_mod_wklp_wkup_alwayson_rst_mod_por_rst_n	WKUP_VTM_0	mod_por_rst_n	VTM POR Reset
WKUP_PSC_0	psc_mod_wklp_wkup_alwayson_rst_mod_g_rst_n	WKUP_CTRL_MMR_0	mod_g_rst_n	Wakeup MMR Control Module Reset
WKUP_PSC_0	psc_mod_wklp_wkup_alwayson_rst_mod_por_rst_n	WKUP_CTRL_MMR_0	mod_por_rst_n	Wakeup MMR Control POR Reset
WKUP_PSC_0	psc_mod_wklp_wkup_alwayson_rst_mod_g_rst_n	WKUP_ECC_AGGR_0	rst_mod_g_rst_n	WKUP ECC AGGR Module Reset
WKUP_PSC_0	psc_mod_wklp_wkup_alwayson_rst_mod_g_rst_n	WKUP_INTROUTER_GPIOMUX	main_mod_g_rst_n	Wakeup GPIO Mux Interrupt Router Reset
WKUP_PSC_0	psc_mod_wklp_wkup_alwayson_rst_mod_g_rst_n	MCU_1MBYTE_SRAM	rst_mod_g_rst_n	MCU MSRAM Reset
WKUP_PSC_0	psc_mod_wklp_wkup_alwayson_rst_mod_g_rst_n	MCU_512B_SCRATCHPADRAM	rst_mod_g_rst_n	MCU PSRAM Reset
WKUP_PSC_0	psc_mod_wklp_mcu_adc_0_rst_mod_g_rst_n	MCU_ADC_0	rst_mod_g_rst_n	ADC0 Reset
WKUP_PSC_0	psc_mod_wklp_mcu_adc_1_rst_mod_g_rst_n	MCU_ADC_1	rst_mod_g_rst_n	ADC1 Reset
WKUP_PSC_0	psc_mod_wklp_wkup_alwayson_rst_mod_g_rst_n	MCU_CPSW_0	cppi_rst_n_mod_g_rst_n	CPSW2G Module Reset
WKUP_PSC_0	psc_mod_wklp_wkup_alwayson_rst_mod_g_rst_n	MCU_DCC_0	mod_g_rst_n	MCU DCC Module Reset (Instance 0)
WKUP_PSC_0	psc_mod_wklp_wkup_alwayson_rst_mod_g_rst_n	MCU_DCC_1	mod_g_rst_n	MCU DCC Module Reset (Instance 1)
WKUP_PSC_0	psc_mod_wklp_main2wkupmcu_rst_mod_g_rst_n	MCU_DCC_2	mod_g_rst_n	MCU DCC Module Reset (Instance 2)
WKUP_PSC_0	psc_mod_wklp_wkup_alwayson_rst_mod_g_rst_n	MCU_ESM_0	mod_g_rst_n	
WKUP_PSC_0	psc_mod_wklp_wkup_alwayson_rst_mod_por_rst_n	MCU_ESM_0	mod_por_rst_n	MCU ESM0 Module Reset
WKUP_PSC_0	psc_mod_wklp_mcu_hyperbus_rst_mod_g_rst_n	MCU_FSS_0	hpb_rst_mod_g_rst_n	Hyperbus Module Reset
WKUP_PSC_0	psc_mod_wklp_mcu_ospi_0_rst_mod_g_rst_n	MCU_FSS_0	ospi0_rst_mod_g_rst_n	OSPI0 Module Reset
WKUP_PSC_0	psc_mod_wklp_mcu_ospi_1_rst_mod_g_rst_n	MCU_FSS_0	ospi1_rst_mod_g_rst_n	OSPI1 Module Reset
WKUP_PSC_0	psc_mod_wklp_wkup_alwayson_rst_mod_g_rst_n	MCU_FSS_0	rst_mod_g_rst_n	FSS Reset
WKUP_PSC_0	psc_mod_wklp_wkup_alwayson_rst_mod_g_rst_n	MCU_I2C_0	mod_g_rst_n	MCU I2C0 Module Reset
WKUP_PSC_0	psc_mod_wklp_wkup_alwayson_rst_mod_g_rst_n	MCU_I2C_1	mod_g_rst_n	MCU I2C1 Module Reset
WKUP_PSC_0	psc_mod_wklp_mcu_i3c_0_rst_mod_g_rst_n	MCU_I3C_0	i3c_mod_g_rst_n	MCU I3C0 Module Reset
WKUP_PSC_0	psc_mod_wklp_mcu_i3c_1_rst_mod_g_rst_n	MCU_I3C_1	i3c_mod_g_rst_n	Main I3C1 Module Reset
WKUP_PSC_0	psc_mod_wklp_mcu_mcan_0_rst_mod_g_rst_n	MCU_MCANSS_0	mcanss_rst_mod_g_rst_n	MCAN0 Module Reset
WKUP_PSC_0	psc_mod_wklp_mcu_mcan_1_rst_mod_g_rst_n	MCU_MCANSS_1	mcanss_rst_mod_g_rst_n	MCAN1 Module Reset
WKUP_PSC_0	psc_mod_wklp_wkup_alwayson_rst_mod_g_rst_n	MCU_PDMA_ADC_0	rst_mod_g_rst_n	PDMA MCU ADC Module Reset
WKUP_PSC_0	psc_mod_wklp_wkup_alwayson_rst_mod_g_rst_n	MCU_PDMA_G0	rst_mod_g_rst_n	PDMA MCU MISC G0 Module Reset
WKUP_PSC_0	psc_mod_wklp_wkup_alwayson_rst_mod_g_rst_n	MCU_PDMA_G1	rst_mod_g_rst_n	PDMA MCU MISC G1 Module Reset
WKUP_PSC_0	psc_mod_wklp_wkup_alwayson_rst_mod_g_rst_n	MCU_PDMA_G2	rst_mod_g_rst_n	PDMA MCU MISC G2 Module Reset
WKUP_PSC_0	psc_mod_wklp_mcu_r5_0_rst_mod_g_rst_n	MCU_PULSAR_0	cpu0_mod_g_rst_n	Pulsar CPU0 Local Reset
WKUP_PSC_0	psc_mod_wklp_mcu_r5_0_rst_mod_l_rst_n	MCU_PULSAR_0	cpu0_mod_l_rst_n	

Copyright © 2024 Texas Instruments Incorporated

Table 5-26. Reset Modules (continued)

Generating Module	Generated Reset	Reset Module	Port in Reset Module	Description
WKUP_PSC_0	psc_mod_wklp_mcu_r5_1_rst_mod_g_rst_n	MCU_PULSAR_0	cpu1_mod_g_rst_n	Pulsar CPU1 Local Reset
WKUP_PSC_0	psc_mod_wklp_mcu_r5_1_rst_mod_l_rst_n	MCU_PULSAR_0	cpu1_mod_l_rst_n	
WKUP_PSC_0	psc_mod_wklp_mcu_r5_0_rst_mod_g_rst_n	MCU_RTI_0	mod_g_rst_n	MCU RTI0 (MCU Pulsar0 R5 0) Module Reset
WKUP_PSC_0	psc_mod_wklp_mcu_r5_0_rst_mod_por_rst_n	MCU_RTI_0	mod_por_rst_n	MCU RTI0 (MCU Pulsar0 R5 0) Module POR Reset
WKUP_PSC_0	psc_mod_wklp_mcu_r5_1_rst_mod_g_rst_n	MCU_RTI_1	mod_g_rst_n	MCU RTI1 (MCU Pulsar0 R5 1) Module Reset
WKUP_PSC_0	psc_mod_wklp_mcu_r5_1_rst_mod_por_rst_n	MCU_RTI_1	mod_por_rst_n	MCU RTI1 (MCU Pulsar0 R5 1) Module POR Reset
WKUP_PSC_0	psc_mod_wklp_wkup_alwayson_rst_mod_g_rst_n	MCU_SA3SS_0	rst_mod_g_rst_n	MCU SA3_SS Module Reset
WKUP_PSC_0	psc_mod_wklp_wkup_alwayson_rst_mod_g_rst_n	MCU_SPI_0	mod_g_rst_n	MCU SPI0 Module Reset
WKUP_PSC_0	psc_mod_wklp_wkup_alwayson_rst_mod_g_rst_n	MCU_SPI_1	mod_g_rst_n	MCU SPI1 Module Reset
WKUP_PSC_0	psc_mod_wklp_wkup_alwayson_rst_mod_g_rst_n	MCU_SPI_2	mod_g_rst_n	MCU SPI2 Module Reset
WKUP_PSC_0	psc_mod_wklp_wkup_alwayson_rst_mod_g_rst_n	MCU_TIMER_0	timer_mod_g_rst_n	MCU DMTIMER0 Module Reset
WKUP_PSC_0	psc_mod_wklp_wkup_alwayson_rst_mod_g_rst_n	MCU_TIMER_1	timer_mod_g_rst_n	MCU DMTIMER1 Module Reset
WKUP_PSC_0	psc_mod_wklp_mcu_pulsar_pbist_0_rst_mod_g_rst_n	MCU_TIMER_2	timer_mod_g_rst_n	MCU DMTIMER2 Module Reset
WKUP_PSC_0	psc_mod_wklp_wkup_alwayson_rst_mod_g_rst_n	MCU_TIMER_3	timer_mod_g_rst_n	MCU DMTIMER3 Module Reset
WKUP_PSC_0	psc_mod_wklp_wkup_alwayson_rst_mod_g_rst_n	MCU_TIMER_4	timer_mod_g_rst_n	MCU DMTIMER4 Module Reset
WKUP_PSC_0	psc_mod_wklp_wkup_alwayson_rst_mod_g_rst_n	MCU_TIMER_5	timer_mod_g_rst_n	MCU DMTIMER5 Module Reset
WKUP_PSC_0	psc_mod_wklp_wkup_alwayson_rst_mod_g_rst_n	MCU_TIMER_6	timer_mod_g_rst_n	MCU DMTIMER6 Module Reset
WKUP_PSC_0	psc_mod_wklp_wkup_alwayson_rst_mod_g_rst_n	MCU_TIMER_7	timer_mod_g_rst_n	MCU DMTIMER7 Module Reset
WKUP_PSC_0	psc_mod_wklp_wkup_alwayson_rst_mod_g_rst_n	MCU_TIMER_8	timer_mod_g_rst_n	MCU DMTIMER8 Module Reset
WKUP_PSC_0	psc_mod_wklp_wkup_alwayson_rst_mod_g_rst_n	MCU_TIMER_9	timer_mod_g_rst_n	MCU DMTIMER9 Module Reset
WKUP_PSC_0	psc_mod_wklp_wkup_alwayson_rst_mod_g_rst_n	MCU_UART_0	usart_mod_g_rst_n	MCU UART0 Module Reset
WKUP_PSC_0	psc_mod_wklp_wkup_alwayson_rst_mod_por_rst_n	MCU_CPT2_AGGR_0	vrst_mod_por_rst_n	MCU CP Tracer Reset
WKUP_PSC_0	psc_mod_wklp_wkup_alwayson_rst_mod_g_rst_n	MCU_CTRL_MMR_0	mod_g_rst_n	MCU MMR Control Module Reset
WKUP_PSC_0	psc_mod_wklp_wkup_alwayson_rst_mod_por_rst_n	MCU_CTRL_MMR_0	mod_por_rst_n	MCU MMR Control POR Reset
WKUP_PSC_0	psc_mod_wklp_wkup_alwayson_rst_mod_g_rst_n	MCU_ECC_AGGR_0	rst_mod_g_rst_n	MCU ECC Aggr Module Reset
WKUP_PSC_0	psc_mod_wklp_wkup_alwayson_rst_mod_g_rst_n	MCU_NAVSS_0	modss_rst_mod_g_rst_n	MCU NAVSS Reset
WKUP_PSC_0	psc_mod_wklp_wkup_alwayson_rst_mod_g_rst_n	MCU_NAVSS_0	udmass_rst_mod_g_rst_n	MCU NAVSS Reset
WKUP_PSC_0	psc_mod_wklp_mcu_test_rst_mod_g_rst_n	MCU_PBIST_MCU_0	mod_g_rst_n	MCU PBIST Reset
WKUP_PSC_0	psc_mod_wklp_mcu_pulsar_pbist_0_rst_mod_g_rst_n	MCU_PBIST_PULSAR_0	mod_g_rst_n	MCU Pulsar PBIST Module Reset
WKUP_PSC_0	psc_mod_wklp_wkup_alwayson_rst_mod_g_rst_n	MCU_PLL_MMR_0	mod_g_rst_n	MCU PLL MMR CTRL Module Reset
WKUP_PSC_0	psc_mod_wklp_wkup_alwayson_rst_mod_por_rst_n	MCU_PLL_MMR_0	mod_por_rst_n	MCU PLL MMR CTRL PoR Reset
WKUP_PSC_0	psc_mod_wklp_wkup_alwayson_rst_mod_g_rst_n	MCU_PSROM_0	rst_mod_g_rst_n	MCU PSROM Reset
WKUP_PSC_0	psc_mod_wklp_wkup_alwayson_rst_mod_g_rst_n	MCU_SEC_MMR_0	mod_g_rst_n	MCU Security MMR Control Module Reset
WKUP_PSC_0	psc_mod_wklp_wkup_alwayson_rst_mod_por_rst_n	MCU_SEC_MMR_0	mod_por_rst_n	MCU Security MMR Control POR Reset
MCU_TIE_OFF HIGH	TIE-OFF HIGH	MCU_CPSW_0	cppi_iso_rst_n_mod_g_rst_n	CPSW2G Reset Isolation Input

Copyright © 2024 Texas Instruments Incorporated

Table 5-26. Reset Modules (continued)

Generating Module	Generated Reset	Reset Module	Port in Reset Module	Description
MAIN_PSC_0	psc_mod_mnlp_main_debug_rst_mod_por_rst_n	MCU_CPT2_PROBE_0	dbg_mod_por_rst_n	
MAIN_PSC_0	psc_mod_mnlp_main_debug_rst_mod_por_rst_n	MCU_CPT2_PROBE_FSS_0_2	dbg_mod_por_rst_n	
MAIN_PSC_0	psc_mod_mnlp_main_debug_rst_mod_por_rst_n	MCU_CPT2_PROBE_FSS_1_3	dbg_mod_por_rst_n	
MAIN_PSC_0	psc_mod_mnlp_main_debug_rst_mod_por_rst_n	MCU_CPT2_PROBE_SRAM	dbg_mod_por_rst_n	
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_g_rst_n	MAIN_1KBYTE_SCRATCHPADRAM	rst_mod_g_rst_n	Main PSRAM Reset
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_g_rst_n	MAIN_2KBYTE_SCRATCHPADRAM	rst_mod_g_rst_n	Main PSRAM Reset
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_g_rst_n	MAIN_512KBYTE_SRAM_0	rst_mod_g_rst_n	Main MSRAM Reset
MAIN_PSC_0	psc_mod_mnlp_per_atl_rst_mod_g_rst_n	MAIN_ATL_0	mod_g_rst_n	ATL Reset
MAIN_PSC_0	psc_mod_mnlp_c71x_0_pbist_rst_mod_g_rst_n	MAIN_COMPUTE_CLUSTER_0	ac71_4_dft_pbist_rst_mod_g_rst_n	C7x PBIST Reset
MAIN_PSC_0	psc_mod_mnlp_c71x_0_rst_mod_g_rst_n	MAIN_COMPUTE_CLUSTER_0	ac71_4_mma_rst_mod_g_rst_n	MMA Reset
MAIN_PSC_0	psc_mod_mnlp_c71x_0_rst_mod_g_rst_n	MAIN_COMPUTE_CLUSTER_0	ac71_4_rst_mod_g_rst_n	C7x Reset
MAIN_PSC_0	psc_mod_mnlp_c71x_0_rst_mod_l_rst_n	MAIN_COMPUTE_CLUSTER_0	ac71_4_rst_mod_l_rst_n	C7x Local Reset
MAIN_PSC_0	psc_mod_mnlp_c71x_0_rst_mod_por_rst_n	MAIN_COMPUTE_CLUSTER_0	ac71_4_rst_mod_por_rst_n	C7x Por Reset
MAIN_PSC_0	psc_mod_mnlp_c71x_1_pbist_rst_mod_g_rst_n	MAIN_COMPUTE_CLUSTER_0	ac71_5_dft_pbist_rst_mod_g_rst_n	C7x PBIST Reset
MAIN_PSC_0	psc_mod_mnlp_c71x_1_rst_mod_g_rst_n	MAIN_COMPUTE_CLUSTER_0	ac71_5_rst_mod_g_rst_n	C7x Reset
MAIN_PSC_0	psc_mod_mnlp_c71x_1_rst_mod_l_rst_n	MAIN_COMPUTE_CLUSTER_0	ac71_5_rst_mod_l_rst_n	C7x Local Reset
MAIN_PSC_0	psc_mod_mnlp_c71x_1_rst_mod_por_rst_n	MAIN_COMPUTE_CLUSTER_0	ac71_5_rst_mod_por_rst_n	C7x POR Reset
MAIN_PSC_0	psc_mod_mnlp_a72_cluster_0_rst_mod_g_rst_n	MAIN_COMPUTE_CLUSTER_0	arm0_corepac_rst_mod_g_rst_n	MPU0 Corepac Reset
MAIN_PSC_0	psc_mod_mnlp_a72_cluster_0_rst_mod_por_rst_n	MAIN_COMPUTE_CLUSTER_0	arm0_corepac_rst_mod_por_rst_n	MPU0 Corepac POR Reset
MAIN_PSC_0	psc_mod_mnlp_a72_0_rst_mod_l_rst_n	MAIN_COMPUTE_CLUSTER_0	arm0_cpu0_rst_mod_l_rst_n	A72 CPU0 Local Reset
MAIN_PSC_0	psc_mod_mnlp_a72_0_rst_mod_por_rst_n	MAIN_COMPUTE_CLUSTER_0	arm0_cpu0_rst_mod_por_rst_n	A72 CPU0 POR Reset
MAIN_PSC_0	psc_mod_mnlp_a72_1_rst_mod_l_rst_n	MAIN_COMPUTE_CLUSTER_0	arm0_cpu1_rst_mod_l_rst_n	MPU0 CPU1 Reset
MAIN_PSC_0	psc_mod_mnlp_a72_0_rst_mod_por_rst_n	MAIN_COMPUTE_CLUSTER_0	arm0_cpu1_rst_mod_por_rst_n	A72 CPU1 POR Reset
MAIN_PSC_0	psc_mod_mnlp_a72_cluster_0_pbist_rst_mod_g_rst_n	MAIN_COMPUTE_CLUSTER_0	arm0_pbist_rst_mod_g_rst_n	MPU0 PBIST Reset
MAIN_PSC_0	psc_mod_mnlp_mmc4b_1_rst_mod_g_rst_n	MAIN_COMPUTE_CLUSTER_0	dru_msmc_rst_mod_g_rst_n	
MAIN_PSC_0	psc_mod_mnlp_main_mcanss_6_rst_mod_g_rst_n	MAIN_COMPUTE_CLUSTER_0	gic_msmc_rst_mod_g_rst_n	GIC Reset
MAIN_PSC_0	psc_mod_mnlp_cc_top_pbist_rst_mod_g_rst_n	MAIN_COMPUTE_CLUSTER_0	msmc_pbist_rst_mod_g_rst_n	MSMC PBIST Reset
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_g_rst_n	MAIN_COMPUTE_CLUSTER_0	msmc_rst_mod_g_rst_n	MSMC Reset
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_por_rst_n	MAIN_COMPUTE_CLUSTER_0	msmc_rst_mod_por_rst_n	MSMC Reset
MAIN_PSC_0	psc_mod_mnlp_cpsw_2_rst_mod_g_rst_n	MAIN_CPSW_0	cppi_rst_n_mod_g_rst_n	CPSW2G Module Reset
MAIN_PSC_0	psc_mod_mnlp_csirx_0_rst_mod_g_rst_n	MAIN_CSI_RX_0	main_rstn_mod_g_rst_n	CSI RX Instance 0 Module Reset
MAIN_PSC_0	psc_mod_mnlp_csirx_1_rst_mod_g_rst_n	MAIN_CSI_RX_1	main_rstn_mod_g_rst_n	CSI RX Instance 1 Module Reset
MAIN_PSC_0	psc_mod_mnlp_csitx_0_rst_mod_g_rst_n	MAIN_CSI_TX_0	main_rstn_mod_g_rst_n	CSI TX Instance 0 Module Reset
MAIN_PSC_0	psc_mod_mnlp_csitx_1_rst_mod_g_rst_n	MAIN_CSI_TX_1	main_rstn_mod_g_rst_n	CSI TX Instance 1 Module Reset
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_g_rst_n	MAIN_DCC_0	mod_g_rst_n	Main DCC Module Reset (Instance 0)

Copyright © 2024 Texas Instruments Incorporated

Table 5-26. Reset Modules (continued)

Generating Module	Generated Reset	Reset Module	Port in Reset Module	Description
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_g_rst_n	MAIN_DCC_1	mod_g_rst_n	Main DCC Module Reset (Instance 1)
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_g_rst_n	MAIN_DCC_2	mod_g_rst_n	Main DCC Module Reset (Instance 2)
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_g_rst_n	MAIN_DCC_3	mod_g_rst_n	Main DCC Module Reset (Instance 3)
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_g_rst_n	MAIN_DCC_4	mod_g_rst_n	Main DCC Module Reset (Instance 4)
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_g_rst_n	MAIN_DCC_5	mod_g_rst_n	Main DCC Module Reset (Instance 5)
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_g_rst_n	MAIN_DCC_6	mod_g_rst_n	Main DCC Module Reset (Instance 6)
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_g_rst_n	MAIN_DCC_7	mod_g_rst_n	Main DCC Module Reset (Instance 7)
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_g_rst_n	MAIN_DCC_8	mod_g_rst_n	Main DCC Module Reset (Instance 8)
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_g_rst_n	MAIN_DCC_9	mod_g_rst_n	Main DCC Module Reset (Instance 9)
MAIN_PSC_0	psc_mod_mnlp_emif_cfg_0_rst_mod_g_rst_n	MAIN_DDR_EW_WRAP_0	ddrss_cfg_mod_g_rst_n	DDRSS CFG Module Reset
MAIN_PSC_0	psc_mod_mnlp_emif_data_0_rst_mod_g_rst_n	MAIN_DDR_EW_WRAP_0	ddrss_mod_g_rst_n	DDRSS Module Reset
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_g_rst_n	MAIN_DDR_EW_WRAP_0	ddrss_vbus_mod_g_rst_n	DDRSS Module Reset
MAIN_PSC_0	psc_mod_mnlp_emif_cfg_1_rst_mod_g_rst_n	MAIN_DDR_EW_WRAP_1	ddrss_cfg_mod_g_rst_n	
MAIN_PSC_0	psc_mod_mnlp_emif_data_1_rst_mod_g_rst_n	MAIN_DDR_EW_WRAP_1	ddrss_mod_g_rst_n	
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_g_rst_n	MAIN_DDR_EW_WRAP_1	ddrss_vbus_mod_g_rst_n	
MAIN_PSC_0	psc_mod_mnlp_dmpac_rst_mod_g_rst_n	MAIN_DMPAC_0	main_mod_g_rst_n	DMPAC Reset
MAIN_PSC_0	psc_mod_mnlp_dmpac_rst_mod_por_rst_n	MAIN_DMPAC_0	main_mod_por_rst_n	
MAIN_PSC_0	psc_mod_mnlp_dmpac_rst_mod_g_rst_n	MAIN_DMPAC_0	psil_leaf_reset_mod_g_rst_n	
MAIN_PSC_0	psc_mod_mnlp_sde_rst_mod_g_rst_n	MAIN_DMPAC_0	sde_mod_g_rst_n	SDE Reset
MAIN_PSC_0	psc_mod_mnlp_csirx_phy_0_rst_mod_g_rst_n	MAIN_DPHY_RX_0	main_rstn_mod_g_rst_n	CSIRX PHY0 Reset
MAIN_PSC_0	psc_mod_mnlp_csirx_phy_1_rst_mod_g_rst_n	MAIN_DPHY_RX_1	main_rstn_mod_g_rst_n	CSIRX PHY1 Reset
MAIN_PSC_0	psc_mod_mnlp_dss_rst_mod_g_rst_n	MAIN_DSS_0	dss_mod_g_rst_n	DSS Module Reset
MAIN_PSC_0	psc_mod_mnlp_dsi_rst_mod_g_rst_n	MAIN_DSS_DSI_0	dsi_mod_g_rst_n	DSI Module Reset
MAIN_PSC_0	psc_mod_mnlp_csitx_1_rst_mod_g_rst_n	MAIN_DSS_DSI_1	dsi_mod_g_rst_n	
MAIN_PSC_0	psc_mod_mnlp_edp_0_rst_mod_g_rst_n	MAIN_DSS_EDP_0	dptx_mod_g_rst_n	eDP0 Module Reset
MAIN_PSC_0	psc_mod_mnlp_per_motor_rst_mod_g_rst_n	MAIN_ECAP_0	vbus_mod_g_rst_n	Main eCAP0 Module Reset
MAIN_PSC_0	psc_mod_mnlp_per_motor_rst_mod_g_rst_n	MAIN_ECAP_1	vbus_mod_g_rst_n	Main eCAP1 Module Reset
MAIN_PSC_0	psc_mod_mnlp_per_motor_rst_mod_g_rst_n	MAIN_ECAP_2	vbus_mod_g_rst_n	Main eCAP2 Module Reset
MAIN_PSC_0	psc_mod_mnlp_per_gpmc_rst_mod_g_rst_n	MAIN_ELM_0	elm_mod_g_rst_n	Main ELM Module Reset
MAIN_PSC_0	psc_mod_mnlp_mmc4b_0_rst_mod_g_rst_n	MAIN_EMMC4_0	emmcsdss_mod_g_rst_n	Main eMMC/SD 4b Module Reset Instance0
MAIN_PSC_0	psc_mod_mnlp_mmc8b_0_rst_mod_g_rst_n	MAIN_EMMC8_0	emmcss_mod_g_rst_n	Main eMMC/SD 8b Module Reset Instance0
MAIN_PSC_0	psc_mod_mnlp_per_motor_rst_mod_g_rst_n	MAIN_EPWM_0	mod_g_rst_n	Main eHRPWM0 Module Reset
MAIN_PSC_0	psc_mod_mnlp_per_motor_rst_mod_g_rst_n	MAIN_EPWM_1	mod_g_rst_n	Main eHRPWM1 Module Reset
MAIN_PSC_0	psc_mod_mnlp_per_motor_rst_mod_g_rst_n	MAIN_EPWM_2	mod_g_rst_n	Main eHRPWM2 Module Reset
MAIN_PSC_0	psc_mod_mnlp_per_motor_rst_mod_g_rst_n	MAIN_EPWM_3	mod_g_rst_n	Main eHRPWM3 Module Reset

Copyright © 2024 Texas Instruments Incorporated

Table 5-26. Reset Modules (continued)

Generating Module	Generated Reset	Reset Module	Port in Reset Module	Description
MAIN_PSC_0	psc_mod_mnlp_per_motor_rst_mod_g_rst_n	MAIN_EPWM_4	mod_g_rst_n	Main eHRPWM4 Module Reset
MAIN_PSC_0	psc_mod_mnlp_per_motor_rst_mod_g_rst_n	MAIN_EPWM_5	mod_g_rst_n	Main eHRPWM5 Module Reset
MAIN_PSC_0	psc_mod_mnlp_per_motor_rst_mod_g_rst_n	MAIN_EQEP_0	vbus_mod_g_rst_n	Main eQEP0 Module Reset
MAIN_PSC_0	psc_mod_mnlp_per_motor_rst_mod_g_rst_n	MAIN_EQEP_1	vbus_mod_g_rst_n	Main eQEP1 Module Reset
MAIN_PSC_0	psc_mod_mnlp_per_motor_rst_mod_g_rst_n	MAIN_EQEP_2	vbus_mod_g_rst_n	Main eQEP2 Module Reset
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_g_rst_n	MAIN_ESM_0	mod_g_rst_n	
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_por_rst_n	MAIN_ESM_0	mod_por_rst_n	Main ESM0 Module Reset
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_g_rst_n	MAIN_GPIO_0	mod_g_rst_n	Main GPIO0 Module Reset
MAIN_PSC_0	psc_mod_mnlp_per_vpfe_rst_mod_g_rst_n	MAIN_GPIO_2	mod_g_rst_n	Main GPIO2 Module Reset
MAIN_PSC_0	psc_mod_mnlp_per_vpfe_rst_mod_g_rst_n	MAIN_GPIO_4	mod_g_rst_n	Main GPIO4 Module Reset
MAIN_PSC_0	psc_mod_mnlp_per_vpfe_rst_mod_g_rst_n	MAIN_GPIO_6	mod_g_rst_n	Main GPIO6 Module Reset
MAIN_PSC_0	psc_mod_mnlp_per_gpmc_rst_mod_g_rst_n	MAIN_GPMC_0	mod_g_rst_n	GPMC Module Reset
MAIN_PSC_0	psc_mod_mnlp_gpucom_rst_mod_g_rst_n	MAIN_GPU_0	gpu_mod_g_rst_n	
MAIN_PSC_0	psc_mod_mnlp_gpupbist_rst_mod_g_rst_n	MAIN_GPU_0	pbist_rst_mod_g_rst_n	
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_por_rst_n	MAIN_GTC_0	mod_por_rst_n	GTC Module Reset
MAIN_PSC_0	psc_mod_mnlp_per_miscio_rst_mod_g_rst_n	MAIN_I2C_0	mod_g_rst_n	Main I2C0 Module Reset
MAIN_PSC_0	psc_mod_mnlp_per_miscio_rst_mod_g_rst_n	MAIN_I2C_1	mod_g_rst_n	Main I2C1 Module Reset
MAIN_PSC_0	psc_mod_mnlp_per_miscio_rst_mod_g_rst_n	MAIN_I2C_2	mod_g_rst_n	Main I2C2 Module Reset
MAIN_PSC_0	psc_mod_mnlp_per_miscio_rst_mod_g_rst_n	MAIN_I2C_3	mod_g_rst_n	Main I2C3 Module Reset
MAIN_PSC_0	psc_mod_mnlp_per_miscio_rst_mod_g_rst_n	MAIN_I2C_4	mod_g_rst_n	Main I2C4 Module Reset
MAIN_PSC_0	psc_mod_mnlp_per_miscio_rst_mod_g_rst_n	MAIN_I2C_5	mod_g_rst_n	Main I2C5 Module Reset
MAIN_PSC_0	psc_mod_mnlp_per_miscio_rst_mod_g_rst_n	MAIN_I2C_6	mod_g_rst_n	Main I2C6 Module Reset
MAIN_PSC_0	psc_mod_mnlp_main_mcanss_0_rst_mod_g_rst_n	MAIN_MCANSS_0	mcanss_rst_mod_g_rst_n	Main MCANSS Instance0 Module Reset
MAIN_PSC_0	psc_mod_mnlp_main_mcanss_1_rst_mod_g_rst_n	MAIN_MCANSS_1	mcanss_rst_mod_g_rst_n	Main MCANSS Instance1 Module Reset
MAIN_PSC_0	psc_mod_mnlp_main_mcanss_2_rst_mod_g_rst_n	MAIN_MCANSS_2	mcanss_rst_mod_g_rst_n	Main MCANSS Instance2 Module Reset
MAIN_PSC_0	psc_mod_mnlp_main_mcanss_3_rst_mod_g_rst_n	MAIN_MCANSS_3	mcanss_rst_mod_g_rst_n	Main MCANSS Instance3 Module Reset
MAIN_PSC_0	psc_mod_mnlp_per_audio_rst_mod_g_rst_n	MAIN_MCASP_0	mod_g_rst_n	McASP0 Module Reset
MAIN_PSC_0	psc_mod_mnlp_per_audio_rst_mod_g_rst_n	MAIN_MCASP_1	mod_g_rst_n	McASP1 Module Reset
MAIN_PSC_0	psc_mod_mnlp_per_audio_rst_mod_g_rst_n	MAIN_MCASP_2	mod_g_rst_n	McASP2 Module Reset
MAIN_PSC_0	psc_mod_mnlp_per_audio_rst_mod_g_rst_n	MAIN_MCASP_3	mod_g_rst_n	McASP3 Module Reset
MAIN_PSC_0	psc_mod_mnlp_per_audio_rst_mod_g_rst_n	MAIN_MCASP_4	mod_g_rst_n	McASP4 Module Reset
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_g_rst_n	MAIN_NAVSS_0	modss_rst_mod_g_rst_n	Main NAVSS Reset
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_g_rst_n	MAIN_NAVSS_0	nbss_rst_mod_g_rst_n	Main NAVSS Reset
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_g_rst_n	MAIN_NAVSS_0	udmass_rst_mod_g_rst_n	Main NAVSS Reset
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_g_rst_n	MAIN_NAVSS_0	virtss_rst_mod_g_rst_n	Main NAVSS Reset

Copyright © 2024 Texas Instruments Incorporated

Table 5-26. Reset Modules (continued)

Generating Module	Generated Reset	Reset Module	Port in Reset Module	Description
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_g_rst_n	MAIN_NBSS_0	rst_mod_g_rst_n	Main ECC Aggr Module Reset
MAIN_PSC_0	psc_mod_mnlp_pcie_1_rst_mod_por_rst_n	MAIN_PCIE_0	pcie_rst_mod_g_rst_n	PCIe1 Module Reset
MAIN_PSC_0	psc_mod_mnlp_main_debug_rst_mod_g_rst_n	MAIN_PDMA_DEBUG_G0	rst_mod_g_rst_n	PDMA Debug CCMCU Module Reset
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_g_rst_n	MAIN_PDMA_MCAN_0	rst_mod_g_rst_n	PDMA Main MCAN Module Reset
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_g_rst_n	MAIN_PDMA_MCASP_0	rst_mod_g_rst_n	PDMA McASP G0 Module Reset
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_g_rst_n	MAIN_PDMA_UART_G0	rst_mod_g_rst_n	PDMA Main USART G0 Module Reset
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_g_rst_n	MAIN_PDMA_UART_G1	rst_mod_g_rst_n	PDMA Main USART G1 Module Reset
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_g_rst_n	MAIN_PDMA_UART_G2	rst_mod_g_rst_n	PDMA Main USART G2 Module Reset
MAIN_PSC_0	psc_mod_mnlp_pulsar_0_r5_0_rst_mod_g_rst_n	MAIN_PULSAR_0	cpu0_mod_g_rst_n	Pulsar_0 Cpu0 Reset
MAIN_PSC_0	psc_mod_mnlp_pulsar_0_r5_0_rst_mod_l_rst_n	MAIN_PULSAR_0	cpu0_mod_l_rst_n	
MAIN_PSC_0	psc_mod_mnlp_pulsar_0_r5_1_rst_mod_g_rst_n	MAIN_PULSAR_0	cpu1_mod_g_rst_n	Pulsar_0 Cpu1 Reset
MAIN_PSC_0	psc_mod_mnlp_pulsar_0_r5_1_rst_mod_l_rst_n	MAIN_PULSAR_0	cpu1_mod_l_rst_n	
MAIN_PSC_0	psc_mod_mnlp_pulsar_0_r5_0_rst_mod_l_rst_n	MAIN_PULSAR_1	cpu0_mod_l_rst_n	
MAIN_PSC_0	psc_mod_mnlp_pulsar_0_r5_1_rst_mod_l_rst_n	MAIN_PULSAR_1	cpu1_mod_l_rst_n	
MAIN_PSC_0	psc_mod_mnlp_a72_0_rst_mod_g_rst_n	MAIN_RTI_A72_0	mod_g_rst_n	Main RTI0 Module (ARM Clstr0 Core0) Reset
MAIN_PSC_0	psc_mod_mnlp_a72_0_rst_mod_por_rst_n	MAIN_RTI_A72_0	mod_por_rst_n	Main RTI0 Module (ARM Clstr0 Core0) POR Reset
MAIN_PSC_0	psc_mod_mnlp_a72_1_rst_mod_g_rst_n	MAIN_RTI_A72_1	mod_g_rst_n	Main RTI1 Module (ARM Clstr0 Core1) Reset
MAIN_PSC_0	psc_mod_mnlp_a72_1_rst_mod_por_rst_n	MAIN_RTI_A72_1	mod_por_rst_n	Main RTI1 Module (ARM Clstr0 Core1) POR Reset
MAIN_PSC_0	psc_mod_mnlp_tx_dphy_rst_mod_g_rst_n	MAIN_SERDES_0	mod_g_rst_n	DPHY TX Reset
MAIN_PSC_0	psc_mod_mnlp_c71x_0_rst_mod_g_rst_n	MAIN_RTI_C7X_0	mod_g_rst_n	Main RTI16 Module (C7x Instance 0 RTI) Module Reset
MAIN_PSC_0	psc_mod_mnlp_c71x_0_rst_mod_por_rst_n	MAIN_RTI_C7X_0	mod_por_rst_n	Main RTI16 Module (C7x Instance 0 RTI) Module POR Reset
MAIN_PSC_0	psc_mod_mnlp_tx_dphy_1_rst_mod_g_rst_n	MAIN_SERDES_1	mod_g_rst_n	DPHY TX Reset
MAIN_PSC_0	psc_mod_mnlp_c71x_1_rst_mod_g_rst_n	MAIN_RTI_C7X_1	mod_g_rst_n	Main RTI17 Module (C7x Instance 1 RTI) Module Reset
MAIN_PSC_0	psc_mod_mnlp_c71x_1_rst_mod_por_rst_n	MAIN_RTI_C7X_1	mod_por_rst_n	Main RTI17 Module (C7x Instance 1 RTI) Module POR Reset
MAIN_PSC_0	psc_mod_mnlp_main_mcanss_7_rst_mod_g_rst_n	MAIN_SPI_0	mod_g_rst_n	Main SPI0 Module Reset
MAIN_PSC_0	psc_mod_mnlp_gpucom_rst_mod_g_rst_n	MAIN_RTI_GPU	mod_g_rst_n	MAIN RTI15 (GPU RTI) Reset
MAIN_PSC_0	psc_mod_mnlp_gpucom_rst_mod_por_rst_n	MAIN_RTI_GPU	mod_por_rst_n	MAIN RTI15 (GPU RTI) POR Reset
MAIN_PSC_0	psc_mod_mnlp_main_mcanss_7_rst_mod_g_rst_n	MAIN_SPI_1	mod_g_rst_n	Main SPI1 Module Reset
MAIN_PSC_0	psc_mod_mnlp_pulsar_0_r5_0_rst_mod_g_rst_n	MAIN_RTI_PULSAR_R5F_0_0	mod_g_rst_n	MAIN RTI28 (Main Pulsar0 R5_0 RTI) Reset
MAIN_PSC_0	psc_mod_mnlp_pulsar_0_r5_0_rst_mod_por_rst_n	MAIN_RTI_PULSAR_R5F_0_0	mod_por_rst_n	MAIN RTI28 (Main Pulsar0 R5_0 RTI) POR Reset
MAIN_PSC_0	psc_mod_mnlp_main_mcanss_7_rst_mod_g_rst_n	MAIN_SPI_2	mod_g_rst_n	Main SPI2 Module Reset
MAIN_PSC_0	psc_mod_mnlp_pulsar_0_r5_1_rst_mod_g_rst_n	MAIN_RTI_PULSAR_R5F_0_1	mod_g_rst_n	MAIN RTI29 (Main Pulsar0 R5_1 RTI) Reset
MAIN_PSC_0	psc_mod_mnlp_pulsar_0_r5_1_rst_mod_por_rst_n	MAIN_RTI_PULSAR_R5F_0_1	mod_por_rst_n	MAIN RTI29 (Main Pulsar0 R5_1 RTI) POR Reset
MAIN_PSC_0	psc_mod_mnlp_main_mcanss_7_rst_mod_g_rst_n	MAIN_SPI_3	mod_g_rst_n	Main SPI3 Module Reset
MAIN_PSC_0	psc_mod_mnlp_pulsar_1_r5_0_rst_mod_g_rst_n	MAIN_RTI_PULSAR_R5F_1_0	mod_g_rst_n	MAIN RTI30 (Main Pulsar1 R5_0 RTI) Reset

Copyright © 2024 Texas Instruments Incorporated

Table 5-26. Reset Modules (continued)

Generating Module	Generated Reset	Reset Module	Port in Reset Module	Description
MAIN_PSC_0	psc_mod_mnlp_pulsar_1_r5_0_rst_mod_por_rst_n	MAIN_RTI_PULSAR_R5F_1_0	mod_por_rst_n	MAIN RTI30 (Main Pulsar1 R5_0 RTI) POR Reset
MAIN_PSC_0	psc_mod_mnlp_main_mcanss_8_rst_mod_g_rst_n	MAIN_SPI_4	mod_g_rst_n	Main SPI4 Module Reset
MAIN_PSC_0	psc_mod_mnlp_pulsar_1_r5_1_rst_mod_g_rst_n	MAIN_RTI_PULSAR_R5F_1_1	mod_g_rst_n	MAIN RTI31 (Main Pulsar1 R5_1 RTI) Reset
MAIN_PSC_0	psc_mod_mnlp_pulsar_1_r5_1_rst_mod_por_rst_n	MAIN_RTI_PULSAR_R5F_1_1	mod_por_rst_n	MAIN RTI31 (Main Pulsar1 R5_1 RTI) POR Reset
MAIN_PSC_0	psc_mod_mnlp_main_mcanss_8_rst_mod_g_rst_n	MAIN_SPI_5	mod_g_rst_n	Main SPI5 Module Reset
MAIN_PSC_0	psc_mod_mnlp_saul_rst_mod_g_rst_n	MAIN_SA2_UL_0	rst_mod_g_rst_n	Main SA2_UL Module Reset
MAIN_PSC_0	psc_mod_mnlp_main_mcanss_8_rst_mod_g_rst_n	MAIN_SPI_6	mod_g_rst_n	Main SPI6 Module Reset
MAIN_PSC_0	psc_mod_mnlp_main_mcanss_8_rst_mod_g_rst_n	MAIN_SPI_7	mod_g_rst_n	Main SPI7 Module Reset
MAIN_PSC_0	psc_mod_mnlp_dmtimer_0_rst_mod_g_rst_n	MAIN_TIMER_0	timer_mod_g_rst_n	MAIN DMTIMER0 Module Reset
MAIN_PSC_0	psc_mod_mnlp_dmtimer_1_rst_mod_g_rst_n	MAIN_TIMER_1	timer_mod_g_rst_n	MAIN DMTIMER1 Module Reset
MAIN_PSC_0	psc_mod_mnlp_dmtimer_2_rst_mod_g_rst_n	MAIN_TIMER_2	timer_mod_g_rst_n	MAIN DMTIMER2 Module Reset
MAIN_PSC_0	psc_mod_mnlp_dmtimer_3_rst_mod_g_rst_n	MAIN_TIMER_3	timer_mod_g_rst_n	MAIN DMTIMER3 Module Reset
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_g_rst_n	MAIN_TIMER_4	timer_mod_g_rst_n	MAIN DMTIMER4 Module Reset
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_g_rst_n	MAIN_TIMER_5	timer_mod_g_rst_n	MAIN DMTIMER5 Module Reset
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_g_rst_n	MAIN_TIMER_6	timer_mod_g_rst_n	MAIN DMTIMER6 Module Reset
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_g_rst_n	MAIN_TIMER_7	timer_mod_g_rst_n	MAIN DMTIMER7 Module Reset
MAIN_PSC_0	psc_mod_mnlp_per_spare0_rst_mod_g_rst_n	MAIN_TIMER_8	timer_mod_g_rst_n	MAIN DMTIMER8 Module Reset
MAIN_PSC_0	psc_mod_mnlp_per_spare0_rst_mod_g_rst_n	MAIN_TIMER_9	timer_mod_g_rst_n	MAIN DMTIMER9 Module Reset
MAIN_PSC_0	psc_mod_mnlp_per_spare0_rst_mod_g_rst_n	MAIN_TIMER_10	timer_mod_g_rst_n	MAIN DMTIMER10 Module Reset
MAIN_PSC_0	psc_mod_mnlp_per_spare0_rst_mod_g_rst_n	MAIN_TIMER_11	timer_mod_g_rst_n	MAIN DMTIMER11 Module Reset
MAIN_PSC_0	psc_mod_mnlp_per_spare0_rst_mod_g_rst_n	MAIN_TIMER_12	timer_mod_g_rst_n	MAIN DMTIMER12 Module Reset
MAIN_PSC_0	psc_mod_mnlp_per_spare0_rst_mod_g_rst_n	MAIN_TIMER_13	timer_mod_g_rst_n	MAIN DMTIMER13 Module Reset
MAIN_PSC_0	psc_mod_mnlp_per_spare0_rst_mod_g_rst_n	MAIN_TIMER_14	timer_mod_g_rst_n	MAIN DMTIMER14 Module Reset
MAIN_PSC_0	psc_mod_mnlp_per_spare0_rst_mod_g_rst_n	MAIN_TIMER_15	timer_mod_g_rst_n	MAIN DMTIMER15 Module Reset
MAIN_PSC_0	psc_mod_mnlp_per_spare0_rst_mod_g_rst_n	MAIN_TIMER_16	timer_mod_g_rst_n	MAIN DMTIMER16 Module Reset
MAIN_PSC_0	psc_mod_mnlp_per_spare0_rst_mod_g_rst_n	MAIN_TIMER_17	timer_mod_g_rst_n	MAIN DMTIMER17 Module Reset
MAIN_PSC_0	psc_mod_mnlp_per_spare0_rst_mod_g_rst_n	MAIN_TIMER_18	timer_mod_g_rst_n	MAIN DMTIMER18 Module Reset
MAIN_PSC_0	psc_mod_mnlp_per_spare0_rst_mod_g_rst_n	MAIN_TIMER_19	timer_mod_g_rst_n	MAIN DMTIMER19 Module Reset
MAIN_PSC_0	psc_mod_mnlp_main_mcanss_9_rst_mod_g_rst_n	MAIN_UART_0	usart_mod_g_rst_n	Main UART0 Module Reset
MAIN_PSC_0	psc_mod_mnlp_main_mcanss_9_rst_mod_g_rst_n	MAIN_UART_1	usart_mod_g_rst_n	Main UART1 Module Reset
MAIN_PSC_0	psc_mod_mnlp_main_mcanss_10_rst_mod_g_rst_n	MAIN_UART_2	usart_mod_g_rst_n	Main UART2 Module Reset
MAIN_PSC_0	psc_mod_mnlp_main_mcanss_10_rst_mod_g_rst_n	MAIN_UART_3	usart_mod_g_rst_n	Main UART3 Module Reset
MAIN_PSC_0	psc_mod_mnlp_main_mcanss_11_rst_mod_g_rst_n	MAIN_UART_4	usart_mod_g_rst_n	Main UART4 Module Reset
MAIN_PSC_0	psc_mod_mnlp_main_mcanss_11_rst_mod_g_rst_n	MAIN_UART_5	usart_mod_g_rst_n	Main UART5 Module Reset
MAIN_PSC_0	psc_mod_mnlp_main_mcanss_11_rst_mod_g_rst_n	MAIN_UART_6	usart_mod_g_rst_n	Main UART6 Module Reset

Copyright © 2024 Texas Instruments Incorporated

Table 5-26. Reset Modules (continued)

Generating Module	Generated Reset	Reset Module	Port in Reset Module	Description
MAIN_PSC_0	psc_mod_mnlp_main_mcanss_11_rst_mod_g_rst_n	MAIN_UART_7	usart_mod_g_rst_n	Main UART7 Module Reset
MAIN_PSC_0	psc_mod_mnlp_main_mcanss_11_rst_mod_g_rst_n	MAIN_UART_8	usart_mod_g_rst_n	Main UART8 Module Reset
MAIN_PSC_0	psc_mod_mnlp_main_mcanss_11_rst_mod_g_rst_n	MAIN_UART_9	usart_mod_g_rst_n	Main UART9 Module Reset
MAIN_PSC_0	psc_mod_mnlp_usb_0_rst_mod_g_rst_n	MAIN_USB3P0SS_0	rst_mod_g_rst_n	USB0 Reset
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_g_rst_n	MAIN_AC_ECC_AGGR_6	rst_mod_g_rst_n	Main ECC Aggr Module Reset
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_g_rst_n	MAIN_AC_ECC_AGGR_9	rst_mod_g_rst_n	Main ECC Aggr Module Reset
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_por_rst_n	MAIN_CPT2_AGGR_AC_0	vrst_mod_por_rst_n	AC CP Tracer Reset
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_por_rst_n	MAIN_CPT2_AGGR_HC_0	vrst_mod_por_rst_n	HC CP Tracer Reset
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_por_rst_n	MAIN_CPT2_AGGR_RC_0	vrst_mod_por_rst_n	Main CP Tracer Reset
MAIN_PSC_0	psc_mod_mnlp_main_debug_rst_mod_por_rst_n	MAIN_CPT2_PROBE_NAVSS_AC_DDR_SLV_0	dbg_mod_por_rst_n	
MAIN_PSC_0	psc_mod_mnlp_main_debug_rst_mod_por_rst_n	MAIN_CPT2_PROBE_NAVSS_AC_DDR_SLV_1	dbg_mod_por_rst_n	
MAIN_PSC_0	psc_mod_mnlp_main_debug_rst_mod_por_rst_n	MAIN_CPT2_PROBE_NAVSS_AC_SRAM_SLV_0	dbg_mod_por_rst_n	
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_g_rst_n	MAIN_CTRL_MMR_0	mod_g_rst_n	Main Ctrl MMR Module Reset
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_por_rst_n	MAIN_CTRL_MMR_0	mod_por_rst_n	Main Control MMR Reset
MAIN_PSC_0	psc_mod_mnlp_main_debug_rst_mod_por_rst_n	MAIN_DEBUGSS_0	dbgssr_por_rst_n	DebugSS POR Reset
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_g_rst_n	MAIN_DEBUGSS_0	vbus_chip_rst_n	Debugss Vbus Reset
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_g_rst_n	MAIN_ECC_AGGR_R5_0_0	rst_mod_g_rst_n	Main ECC Aggr Module Reset
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_g_rst_n	MAIN_ECC_AGGR_R5_0_1	rst_mod_g_rst_n	Main ECC Aggr Module Reset
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_g_rst_n	MAIN_ECC_AGGR_R5_1_0	rst_mod_g_rst_n	Main ECC Aggr Module Reset
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_g_rst_n	MAIN_ECC_AGGR_R5_1_1	rst_mod_g_rst_n	Main ECC Aggr Module Reset
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_g_rst_n	MAIN_HC_ECC_AGGR_5	rst_mod_g_rst_n	Main ECC Aggr Module Reset
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_g_rst_n	MAIN_INFRA_ECC_AGGR_0	rst_mod_g_rst_n	Main ECC AGGR Module Reset
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_g_rst_n	MAIN_INTROUTER_CMP_EVENT_0	main_mod_g_rst_n	Compare Event Router Module Reset
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_g_rst_n	MAIN_INTROUTER_GPIOMUX_0	main_mod_g_rst_n	Main GPIO Mux Interrupt Router Reset
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_g_rst_n	MAIN_IP_ECC_AGGR_6	rst_mod_g_rst_n	Main ECC Aggr Module Reset
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_g_rst_n	MAIN_LVL_INTROUTER_MAIN2MCU_0	main_mod_g_rst_n	Main to MCU Level Interrupt Router Reset
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_g_rst_n	MAIN_NAVSS_ECC_AGGR_10	rst_mod_g_rst_n	Main ECC Aggr Module Reset
MAIN_PSC_0	psc_mod_mnlp_dmpac_pbist_rst_mod_g_rst_n	MAIN_PBIST_AC_DMPAC	mod_g_rst_n	DMPAC PBIST Module Reset
MAIN_PSC_0	psc_mod_mnlp_dss_pbist_rst_mod_g_rst_n	MAIN_PBIST_AC_EDP_DSI	mod_g_rst_n	DSS PBIST Module Reset
MAIN_PSC_0	psc_mod_mnlp_encode_1_rst_mod_g_rst_n	MAIN_PBIST_AC_ENC_DEC	mod_g_rst_n	WAVE512 0 Reset
MAIN_PSC_0	psc_mod_mnlp_vpac_pbist_rst_mod_g_rst_n	MAIN_PBIST_AC_VPAC	mod_g_rst_n	VPAC PBIST Module Reset
MAIN_PSC_0	psc_mod_mnlp_main_pbist_rst_mod_g_rst_n	MAIN_PBIST_HC	mod_g_rst_n	HC PBIST Module Reset
MAIN_PSC_0	psc_mod_mnlp_main_pbist_rst_mod_l_rst_n	MAIN_PBIST_INFRA_0	mod_g_rst_n	Main PBIST Reset
MAIN_PSC_0	psc_mod_mnlp_pulsar_pbist_0_rst_mod_g_rst_n	MAIN_PBIST_PULSAR_0	mod_g_rst_n	RC Pulsar0 PBIST Module Reset
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_g_rst_n	MAIN_PLL_MMR_0	mod_g_rst_n	Main PLL CTRL Module Reset

Copyright © 2024 Texas Instruments Incorporated

Table 5-26. Reset Modules (continued)

Generating Module	Generated Reset	Reset Module	Port in Reset Module	Description
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_por_rst_n	MAIN_PLL_MMR_0	mod_por_rst_n	Main PLL CTRL Module POR Reset
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_g_rst_n	MAIN_PLS_INTROUTER_MAIN2MCU_0	main_mod_g_rst_n	Main to MCU Pulse Interrupt Router Reset
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_g_rst_n	MAIN_RC_ECC_AGGR_4	rst_mod_g_rst_n	Main ECC Aggr Module Reset
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_g_rst_n	MAIN_SEC_MMR_0	mod_g_rst_n	Main Sec MMR Module Reset
MAIN_PSC_0	psc_mod_mnlp_main_alwayson_rst_mod_por_rst_n	MAIN_SEC_MMR_0	mod_por_rst_n	Main Sec MMR Module POR Reset
MAIN_PSC_0	psc_mod_mnlp_per_vpfe_rst_mod_g_rst_n	MAIN_TIMESYNC_INTROUTER_0	main_mod_g_rst_n	Main TimeSync Event Interrupt Router Reset
MAIN_PSC_0	psc_mod_mnlp_serdes_0_rst_mod_g_rst_n	MAIN_SERDES_0	mod_g_rst_n	Serdes0 Reset
MAIN_PSC_0	psc_mod_mnlp_usb_1_rst_mod_g_rst_n	MAIN_HYPERLINK_0	v0_mod_g_rst_n	
MAIN_PSC_0	psc_mod_mnlp_usb_1_rst_mod_g_rst_n	MAIN_HYPERLINK_0	v1_mod_g_rst_n	
MAIN_PSC_0	psc_mod_mnlp_vpac_rst_mod_g_rst_n	MAIN_VPAC_0	ldc0_rstn_mod_g_rst_n	LDC0 Reset
MAIN_PSC_0	psc_mod_mnlp_vpac_rst_mod_g_rst_n	MAIN_VPAC_0	main_mod_g_rst_n	VPAC Reset
MAIN_PSC_0	psc_mod_mnlp_vpac_rst_mod_por_rst_n	MAIN_VPAC_0	main_mod_por_rst_n	
MAIN_PSC_0	psc_mod_mnlp_vpac_rst_mod_g_rst_n	MAIN_VPAC_0	msc_mod_g_rst_n	MSC Reset
MAIN_PSC_0	psc_mod_mnlp_vpac_rst_mod_g_rst_n	MAIN_VPAC_0	nf_rstn_mod_g_rst_n	NF Reset
MAIN_PSC_0	psc_mod_mnlp_vpac_rst_mod_g_rst_n	MAIN_VPAC_0	viss0_rstn_mod_g_rst_n	VISS Reset
MAIN_TIE_OFF HIGH	TIE-OFF HIGH	MAIN_CPSW_0	cppi_iso_rst_n_mod_g_rst_n	CPSW2G Reset Isolation Input

5.3.3 Reset Sources

Table 5-27 summarizes the resets supported by device.

Table 5-27. Reset Supported in the Device

Reset Source	Type	Sequence (Effect)
MCU_PORz	Input (control) pin Master Power-on-Reset input (active low) to the entire device. Effects all domains – WKUP, MCU, and MAIN No isolation possible.	See Section 5.3.7.2, *MCU_PORz Sequence*.
MCU_RESETz	Input (control) pin Master Warm Reset input (active low) to the entire device. Effects all domains – WKUP, MCU, and MAIN Modules configured for isolation in the MAIN domain are isolated although the WKUP/MCU domain is not isolated from the MAIN domain.	See Section 5.3.7.3, *MCU_RESETz Sequence*.
PORz	Input (control) pin Power-on-Reset input (active low) to the MAIN domain. Effects MAIN domain WKUP/MCU should be isolated from the MAIN domain	See Section 5.3.7.4, *PORz Sequence*.
RESET_REQz	Input (control) pin Warm Reset Request input (active low) to the MAIN domain Effects MAIN domain WKUP/MCU should be isolated from the MAIN domain. Additionally, modules within the MAIN domain may be isolated.	See Section 5.3.7.5, *RESET_REQz Sequence*.
SW_MCU_WARMRST	Software reset: equivalent to MCU_RESETz pin CTRLMMR_WKUP_MCU_WARM_RST_CTRL[(1)]	See Section 5.3.7.3, *MCU_RESETz Sequence*.

Copyright © 2024 Texas Instruments Incorporated

Table 5-27. Reset Supported in the Device (continued)

Reset Source	Type	Sequence (Effect)
SW_MAIN_POR	Software reset: equivalent to PORz pin CTRLMMR_WKUP_POR_RST_CTRL[(1)]	See Section 5.3.7.4, *PORz Sequence*.
SW_MAIN_WARMRST	Software reset: equivalent to RESET_REQz pin CTRLMMR_WKUP_MAIN_WARM_RST_CTRL[(1)]	See Section 5.3.7.5, *RESET_REQz Sequence*.
WKUP_DMSC0 COLD RST	Internal event: equivalent to MCU_RESETz	See Section 5.3.7.3, *MCU_RESETz Sequence*.
WKUP_DMSC0 WARM RST	Internal event: equivalent to MCU_RESETz	See Section 5.3.7.3, *MCU_RESETz Sequence*.
Thermal Over-temperature (VTM)	Internal event: equivalent to MCU_RESETz (except that PLL behavior is different)	See Section 5.3.7.3, *MCU_RESETz Sequence*.
Debug Reset		
Local Resets	Local reset is a reset that is applied to any given module in a module domain or to a CPU and/or its megamodule (any tightly coupled controllers delivered with the CPU) and their associated local watchdog timers.	

(1) These registers are part of the WKUP CTRL_MMR domain, see *Control Module (CTRL_MMR)*.

5.3.4 Reset Status

Table 5-28 summarizes the reset status of the device.

Table 5-28. Reset Status

Reset Status	Type/Description
MCU_RESETSTATz	Output pin Indicates release of internal reset on the WKUP/MCU domain Released synchronous with start of WKUP_DMSC0 ROM code execution.
RESETSTATz	Output pin Indicates release of internal reset on the MAIN domain
THERMAL_RST	Software bit in CTRLMMR_WKUP_RESET_SRC_STAT[(1)]
DBUGSS_RST	
COLD_OUT_RST	
WARM_OUT_RST	
PORZ_PIN	
RESET_REQZ_PIN	
MCU_RSTZ_PIN	
SW_MAIN_POR	
SW_MAIN_WARMRST	
SW_MCU_WARMRST	
MAIN_RST_DONE	Software bit in
MCU_RST_DONE	CTRLMMR_WKUP_RST_STAT[(1)]

(1) These registers are part of the WKUP CTRL_MMR domain, see *Control Module (CTRL_MMR)*.

5.3.5 Reset Control

Table 5-29 lists the reset control registers.

Table 5-29. Reset Control

Reset Control	Control Register[(1)]
WKUP_DMSC0 Timeout control	CTRLMMR_WKUP_MAIN_POR_TO_CTRL (0 – 500 µs in 100 µs increments)
POR_RST_ISO_DONE_Z	CTRLMMR_WKUP_POR_RST_CTRL
SOC_WARMRST_ISO_DONE_Z	CTRLMMR_WKUP_MAIN_WARM_RST_CTRL

(1) These registers are part of the WKUP CTRL_MMR domain, see *Control Module (CTRL_MMR)*.

Copyright © 2024 Texas Instruments Incorporated

5.3.6 BOOTMODE Pins

The MCU_BOOTMODE pins and BOOTMODE direct the MCU_R5FSS boot (first stage boot). MCU_BOOTMODE pins are latched based upon the rising edge of MCU_PORz.

For more details on Bootmode pins, see *Initialization*.

Copyright © 2024 Texas Instruments Incorporated

5.3.7 Reset Sequences

5.3.7.1 MCU_PORz Overview

On MCU_PORz,

- the WKUP_DMSC0 is powered and clocked;
- the MCU_R5FSS0 is powered but not clocked;
- all processors on the MAIN side are powered down (and clock-gated):
  - Dual A72 MPU Sub-systems
  - R5FSS

The general behavior for coming out of an MCU_PORz reset is:

- Hardware initialization

  ROM Boot Loader (or ROM Code): the purpose of the ROM Boot Loader is to load and check integrity of the Secondary Boot Loader (or application image).
- WKUP_DMSC0 Code executes, setting up MCU_R5FSS0 clocks and configuring security. WKUP_DMSC0 code passes Boot info to MCU_R5FSS0 and releases the MCU_R5FSS0 clocks.
- The Bootmode pins define the peripherals involved in the boot and the associated media (see ROM Code boot modes). The media holds the secondary boot loader (SBL) or the entire application image. The WKUP_DMSC0 firmware needs to be incorporated in this code. For MCU_R5FSS ROM functional description, refer to *Initialization*.
- Once the Secondary Boot Loader has been verified, the WKUP_DMSC0 issues a clock stop to the MCU_R5FSS0 and resets the MCU_R5FSS0.

  Secondary Boot Loader: the secondary boot loader configures the device for application.

The previous description gives the high-level behavior for the MCU_PORz reset. Other resets are almost a subset of this procedure. Section 5.3.7.2, Section 5.3.7.3, Section 5.3.7.4, and Section 5.3.7.5 explain this sequence in more detail.

5.3.7.2 MCU_PORz Sequence

MCU_PORz and PORz must be held low until:

- All supplies have ramped to proper levels
- WKUP_HFOSC0 has a stable amplitude that propagates clocks into the core

If the supplies and clock are already stable, MCU_PORz and/or PORz must be held active (low) for a minimum of 1.2 µs.

1. When MCU_PORz is asserted low, WKUP_CTRL_MMR0 and MCU_CTRL_MMR0 registers are reset; a bit within the WKUP_CTRL_MMR0 registers is reset which allows PORz to propagate to the MAIN domain without delay. In the MAIN domain, CTRL_MMR0 registers are reset.
2. The MCU_BOOTMODE pins are latched on the rising edge of MCU_PORz.
3. The release of MCU_PORz begins a read of fuse values to register files; several different chains of e-fuse values are registered in parallel read sequences. Values are consumed by different blocks at various points in this sequence. In parallel and based upon the release of PORz, the MAIN domain e-fuse controller initiates a read of its e-fuse values.
4. LBIST/PBIST checks of the WKUP_DMSC0 and MCU_R5FSS0 cores and memories are run in the WKUP/MCU domain; in the MAIN domain, the BIST engines are controlled by the application software.
5. WKUP_PSC0 is initialized (and PSC0 in the MAIN domain is initialized); concretely, the PSC modules (both WKUP_PSC0 and PSC0) configure default power states and LPSC (clocking) states.
6. After power domain and clock domain initialization is completed, PSC modules release resets to the device (MOD_POR_RST, MOD_G_RST).
7. WKUP_DMSC0 is removed from reset; WKUP_DMSC0 ROM code execution begins. At this time MCU_RESETSTATz is de-asserted.
8. WKUP_DMSC0 executes WKUP_DMSC0 ROM.
   a. Configures MCU_PLL0, MCU_PLL0_HSDIV1, and WKUP_PLLCTRL0 (proper clock frequency for MCU_R5FSS0).
   b. Configures firewalls and message manager.

Copyright © 2024 Texas Instruments Incorporated

   c. Uses the message manager to pass Boot info to MCU_R5FSS0 ROM.
9. MCU_R5FSS0_CORE0 is released from reset and begins to execute ROM
   a. Based upon the value of MCU_BOOTMODE pins , the R5F ROM code configures peripherals and PLLs to enable loading the external code.
   b. Secondary boot-loader is loaded from external memory
   c. MCU ROM requests WKUP_DMSC0 for loaded code authentication
   d. WKUP_DMSC0 stops clocks to MCU_R5FSS0
   e. WKUP_DMSC0 issues an MCU_R5FSS0 reset
   f. MCU_R5FSS0_CORE0 begins executing secondary boot-loader.

At the end of this sequence, MCU_R5FSS0 is executing the secondary bootloader. The control of the device now passes to the customer code.

Note

On warm resets (except those caused by a VTM Thermal over-temperature), the output of various PLLs are either bypassed or left unaffected. For more details, see Section 5.3.8, *PLL Behavior on Reset*.

Note

Reset Isolated domains are isolated during a MCU_RESETz (or any warm reset) event.

5.3.7.3 MCU_RESETz Sequence

MCU_RESETz release requires:

- All supplies have ramped to proper levels
- WKUP_HFOSC0 has a stable amplitude that propagates clocks into the core

MCU_RESETz must be held active (low) for a minimum of 1.2 µs.

When MCU_RESETz is asserted low, it is also immediately passed to the MAIN domain (similar to MCU_PORz asserting PORz).

1. Much of the device is reset. Blocks that are not reset:
   a. Internal 12.5 MHz oscillator
   b. External HF oscillator
   c. Power monitoring circuitry (Bandgap, POR, POK, PGD, PRG)
   d. I/O cells and I/O pad configuration registers
   e. SERDES
   f. efuse Controller/storage
   g. PLLs
   h. Voltage Temperature Module
   i. Global Timebase Counter (GTC)
   j. Boot Mode Latches
   k. Debug Components
   l. PSRAM

   Another group of modules is partially reset:
   a. PSC
   b. WKUP_DMSC0
   c. Debug
   d. STM
   e. MCU_CPSW0 (CPSW2G)
   f. Probe
2. When MCU_RESETz is released, PSC modules release MOD_G_RST to the device.
3. WKUP_DMSC0 is removed from reset; WKUP_DMSC0 ROM code execution begins. At this time MCU_RESETSTATz is de-asserted.
4. WKUP_DMSC0 executes ROM.

Copyright © 2024 Texas Instruments Incorporated

   a. Configures MCU PLL0, MCU PLL0 HSDIV1, and WKUP PLLCTRL0 (proper clock frequency for MCU_R5FSS0).
   b. Configures firewalls and message manager.
   c. Uses the message manager to pass Boot info to MCU
5. MCU_R5FSS0_CORE0 is released from reset and begins to execute ROM
   a. Based upon the value of MCU_BOOTMODE pins ,the MCU ROM code configures peripherals and PLLs to enable loading the external code.
   b. Secondary boot-loader is loaded from external memory
   c. MCU ROM requests WKUP_DMSC0 for loaded code authentication
   d. WKUP_DMSC0 stops clocks to MCU_R5FSS0
   e. WKUP_DMSC0 issues MCU reset
   f. MCU_R5FSS0_CORE0 begins executing secondary boot-loader.

At the end of this sequence, MCU_R5FSS0 is executing the secondary bootloader. The control of the device now passes to the customer code.

5.3.7.4 PORz Sequence

PORz release requires:

- All supplies have ramped to proper levels
- WKUP_HFOSC0 has a stable amplitude that propagates clocks into the core

PORz must be held active (low) for a minimum of 1.2 µs.

If PORz is asserted with MCU_PORz, then PORz propagates to the MAIN domain immediately.

If PORz is asserted independent of MCU_PORz, then the user code must have setup:

- CTRLMMR_WKUP_POR_RST_CTRL[0] POR_RST_ISO_DONE_Z (high) in order to block the PORz from immediate propagation and
- CTRLMMR_WKUP_MAIN_POR_TO_CTRL[2:0] TIMEOUT_PER in order to configure the timeout, after which PORz is unconditionally propagated to MAIN domain.

PORz sequence is:

1. When PORz is asserted low, hardware in the WKUP/MCU domain recognizes the PORz and generates an interrupt to WKUP_DMSC0 and MCU. Reset hardware latches the PORz low level.
2. User software is responsible for isolating WKUP/MCU domain from the MAIN domain.
   The details of the isolation are application specific and are carried out in software. Connections between the WKUP/MCU domain and the MAIN domain include:
   - LPSC_WKUPMCU2MAIN - LPSC4 (WKUP_PSC0)
   - LPSC_MAIN2WKUPMCU - LPSC5 (WKUP_PSC0)
   - Communications channel: MCSPI3 is connected as a master to MCU_MCSPI1; MCSPI4 is directly connected as a slave to MCU_MCSPI2.
   - WKUP_GPIOMUX_INTRTR0
   - MAIN2MCU_LVL_INTRTR0
   - MAIN2MCU_PLS_INTRTR0
   - WKUP_ESM0
   - MCU_ESM0

   When the isolation is complete, MCU_R5FSS0 or WKUP_DMSC0 must write CTRLMMR_WKUP_POR_RST_CTRL[0] POR_RST_ISO_DONE_Z low to allow the PORz signal to propagate.
3. The release of PORz begins a read of eFuse values to register files; several different chains of e-fuse values are registered in parallel read sequences. Values are consumed by different blocks at various points in this sequence.
4. PSC0 is initialized; specifically, PSC0 module configures default power states and LPSC (clocking) states.
5. After power domain / clock domain initialization is completed, PSC modules release resets to the device (MOD_POR_RST, MOD_G_RST).
6. CTRLMMR_WKUP_RST_STAT[0] MAIN_RST_DONE is asserted.

Copyright © 2024 Texas Instruments Incorporated

At the end of this sequence, MCU_R5FSS can read the CTRLMMR_WKUP_RST_STAT[0] MAIN_RST_DONE bit and begin re-configuring the MAIN domain.

5.3.7.5 RESET_REQz Sequence

RESET_REQz release requires:

- All supplies have ramped to proper levels
- WKUP_HFOSC0 has a stable amplitude that propagates clocks into the core

RESET_REQz must be held active (low) for a minimum of 1.2 µs.

Like the PORz sequence, it is necessary to have configured CTRLMMR_WKUP_MAIN_WARM_RST_CTRL[0] SOC_WARMRST_ISO_DONE_Z high in order to block the reset request from immediately propagating. Unlike the PORz sequence, there is no timeout. Also, unlike the PORz sequence, some clock domains within given power domains can be configured to ignore the RESET_REQz signal.

1. When RESET_REQz is asserted low, hardware in the WKUP/MCU domain recognizes the RESET_REQz and generates an interrupt to WKUP_DMSC0 and MCU_R5FSS0. Reset hardware latches the RESET_REQz low level.
2. User software is responsible for isolating WKUP/MCU domain from the MAIN domain.
   The details of the isolation are application specific and are carried out in software. Connections between the WKUP/MCU domain and the MAIN domain include:
   - LPSC_WKUPMCU2MAIN - LPSC4 (WKUP_PSC0)
   - LPSC_MAIN2WKUPMCU - LPSC5 (WKUP_PSC0)
   - Communications channel: MCSPI3 is connected as a master to MCU_MCSPI1; MCSPI4 is directly connected as a slave to MCU_MCSPI2.
   - WKUP_GPIOMUX_INTRTR0
   - MAIN2MCU_LVL_INTRTR0
   - MAIN2MCU_PLS_INTRTR0
   - WKUP_ESM0
   - MCU_ESM0

   When the isolation is complete, MCU_R5FSS0 or WKUP_DMSC0 must write CTRLMMR_WKUP_MAIN_WARM_RST_CTRL[0] SOC_WARMRST_ISO_DONE_Z low to allow the RESET_REQz signal to propagate.
3. Power domains are not reset by RESET_REQz if-and-only-if a LPSC in the power domain is configured to be reset isolated. Otherwise, power domains and clock domains are reset by RESET_REQz.

   Note

   As an example of how to read tables in the PSC: Device Power-Management Layout Section, in *Power*, PSC0 Power Management Device-Level Layout shows that PD_R5FSS_0 is assigned Power Domain Index 24 with LPSCs 93, 94, and 95 controlling clocks/resets to circuitry within this power domain. PSC0 Power Domain Features shows that power domain 24 is OFF by default (if isolation does not block the reset). PSC0 LPSC Features shows that LPSC 93, 94, and 95 are OFF by default *but* LPSC 93 and 94 can be configured for reset isolation.

   If Reset isolation is selected for LPSC 93 and 94, resets and clocks are not affected by the RESET_REQz signal. Power Domain 24 cannot be reset when the LPSCs are active. Note that LPSC 95 **is** reset.

   If Reset isolation is not selected for either LPSC 93 or 94, resets propagate to LPSC 93, 94, and 95; the clocks are stopped to these LPSCs (since their default state is OFF). Power Domain 24 is reset to its OFF state.
4. After power domain / clock domain initialization is completed, PSC modules release resets to the non-isolated parts of the device (MOD_POR_RST, MOD_G_RST).
5. CTRLMMR_WKUP_RST_STAT[0] MAIN_RST_DONE is asserted.

At the end of this sequence, MCU_R5FSS0 can read the CTRLMMR_WKUP_MAIN_WARM_RST_CTRL[0] SOC_WARMRST_ISO_DONE_Z bit and begin re-configuring the MAIN domain.

Copyright © 2024 Texas Instruments Incorporated

Note

On warm resets (except those caused by a VTM Thermal over-temperature), the output of various PLLs are either bypassed or left unaffected. For more details, see Section 5.3.8, *PLL Behavior on Reset*.

5.3.8 PLL Behavior on Reset

Table 5-30 summarizes the PLL behavior on reset.

Table 5-30. PLL Behavior on Reset

Reset Type	PLL Behavior	
	MCU Domain PLLs	MAIN Domain PLLs
MCU_PORz	All PLLs are reset	All PLLs are reset
PORz	All PLLs are unaffected	All PLLs are reset
MCU_RESETz or RESET_REQz (or other warm resets except VTM Thermal over-temperature event)	All PLL outputs are bypassed (to the reference input frequency into the PLL), see Table 5-31.	Some PLL outputs are bypassed (to the reference input frequency into the PLL) and other PLLs are unaffected, see Table 5-31.
Warm reset triggered by a VTM thermal over-temperature event	All PLL outputs are bypassed	All PLL outputs are bypassed

Table 5-31. PLL Behavior on Warm Resets, Except VTM Event

Not-bypassed (unaffected)	Bypassed
PLL1	MCU_PLL0
PLL2	MCU_PLL1
PLL3	MCU_PLL2
PLL16	PLL0
PLL17	PLL4
PLL19	PLL5
	PLL6
	PLL7
	PLL8
	PLL12
	PLL14
	PLL25
	PLL26

The warm reset event bypasses the PLL (that are to be bypassed) while the warm reset is asserted; after the warm reset is released, the PLL switches from bypass to using the PLL again. If the user wants to maintain bypassed PLLs in the bypass mode (for example, to control current slews on the board in a reset event); there are control bits to configure the PLL in this way:

- For MCU_PLL_[2:0]:
  CTRLMMR_WKUP_MCU_PLL_CLKSEL[23] BYP_WARM_RST = 0
  CTRLMMR_WKUP_MCU_PLL_CLKSEL[31] BYPASS_SW_OVRD = 1
- For bypassed PLLs in the MAIN domain:
  CTRLMMR_WKUP_MAIN_PLLn_CLKSEL[23] BYP_WARM_RST = 0
  CTRLMMR_WKUP_MAIN_PLLn_CLKSEL[31] BYPASS_SW_OVRD = 1

The behavior with PLLs configured as defined above is:

- the BYPASS_SW_OVRD bit (= 1) enables the following behavior:
  – the PLL currently is not in bypass mode (for example, the PLL clock propagates) because BYP_WARM_RST is 0

Copyright © 2024 Texas Instruments Incorporated

- when a warm reset occurs, the PLLs configured to bypass automatically set the BYP_WARM_RST to 1. This setting (with BYPASS_SW_OVRD = 1) maintains the bypass state after the warm reset is released.

When software is ready to use the PLL, the BYP_WARM_RST is cleared back to 0.

Copyright © 2024 Texas Instruments Incorporated

5.4 Clocking

This chapter describes the clock architecture of the device.

5.4.1 Clocking Overview

Various external clock inputs/outputs are needed to drive the device. Summary of these input clock signals is as follows:

- High frequency oscillators inputs
  - OSC1_XO/OSC1_XI — external main crystal interface pins connected to internal oscillator which sources reference clock. Provides reference clock to PLLs within MCU domain and MAIN domain. This high-frequency oscillator is used to provide audio clock frequencies to MCASPs.
  - WKUP_OSC0_XO/WKUP_OSC0_XI — external main crystal interface pins connected to internal oscillator which sources reference clock. Provides reference clock to PLLs within WKUP and MAIN domain.
- Low frequency digital input
  - WKUP_LF_CLKIN - Low Frequency 32k digital clock input, optionally sourced from an external PMIC or other clock source. This SoC does not support a LFOSC crystal input.
- General purpose clock inputs
  - MCU_EXT_REFCLK0 - optional external System clock input (MCU domain).
  - EXT_REFCLK1 — optional external System clock input (MAIN domain).
- Peripheral clocks - refer to the Signal Descriptions for peripheral specific clocks

5.4.2 Modules Controlled by PLL

Table 5-32. Modules Controlled by PLL

MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	MCU_DCC_1, WKUP_DDPA_0, WKUP_ESM_0, WKUP_GPIO_0, WKUP_GPIO_1, WKUP_SMS_0, WKUP_UART_0, WKUP_VTM_0, WKUP_CTRL_MMR_0, WKUP_ECC_Aggr_0, WKUP_INTROUTER_GPIOMUX, WKUP_PSC_0, MCU_1MByte_SRAM, MCU_512B_ScratchpadRAM, MCU_ADC_0, MCU_ADC_1, MCU_CPSW_0, MCU_DCC_0, MCU_DCC_1, MCU_DCC_2, MCU_ESM_0, MCU_FSS_0, MCU_I2C_0, MCU_I2C_1, MCU_I3C_0, MCU_I3C_1, MCU_MCANSS_0, MCU_MCANSS_1, MCU_PDMA_ADC_0, MCU_PDMA_G0, MCU_PDMA_G1, MCU_PDMA_G2, MCU_Pulsar_0, MCU_RTI_0, MCU_RTI_1, MCU_SA3SS_0, MCU_SPI_0, MCU_SPI_1, MCU_SPI_2, MCU_Timer_0, MCU_Timer_1, MCU_Timer_2, MCU_Timer_3, MCU_Timer_4, MCU_Timer_5, MCU_Timer_6, MCU_Timer_7, MCU_Timer_8, MCU_Timer_9, MCU_UART_0, MCU_CPT2_Aggr_0, MCU_CPT2_Probe_0, MCU_CPT2_Probe_FSS_0_2, MCU_CPT2_Probe_FSS_1_3, MCU_CPT2_Probe_SRAM, MCU_CTRL_MMR_0, MCU_ECC_Aggr_0, MCU_EFUSE_0, MCU_NAVSS_0, MCU_PBIST_MCU_0, MCU_PBIST_MCU_1, MCU_PBIST_Pulsar_0, MCU_PLL_MMR_0, MCU_PSROM_0, MCU_SEC_MMR_0,
MCU_PLL0.HSDIV1	MCU_ADC_0, MCU_ADC_1, MCU_DCC_1,
MCU_PLL1.HSDIV0	MCU_DCC_0, MCU_SA3SS_0, MCU_PBIST_MCU_1,
MCU_PLL1.HSDIV1	MCU_ADC_0, MCU_ADC_1, MCU_DCC_0,
MCU_PLL1.HSDIV2	MCU_DCC_0, MCU_MCANSS_0, MCU_MCANSS_1,
MCU_PLL1.HSDIV3	WKUP_I2C_0, WKUP_UART_0, MCU_DCC_0, MCU_I2C_0, MCU_I2C_1, MCU_UART_0,
MCU_PLL1.HSDIV4	MCU_DCC_0, MCU_FSS_0,
MCU_PLL2.HSDIV0	MCU_CPSW_0, MCU_DCC_1, MCU_SPI_0, MCU_SPI_1, MCU_SPI_2,
MCU_PLL2.HSDIV1	MCU_CPSW_0, MCU_DCC_1, Main_CPSW_0, Main_GTC_0, Main_NAVSS_0, Main_PCIe_0,
MCU_PLL2.HSDIV2	MCU_DCC_1, MCU_Timer_0, MCU_Timer_1, MCU_Timer_2, MCU_Timer_3, MCU_Timer_4, MCU_Timer_5, MCU_Timer_6, MCU_Timer_7, MCU_Timer_8, MCU_Timer_9,
MCU_PLL2.HSDIV3	MCU_DCC_1, MCU_MCANSS_0, MCU_MCANSS_1,
MCU_PLL2.HSDIV4	MCU_DCC_1, MCU_FSS_0, MCU_PBIST_MCU_1,

Copyright © 2024 Texas Instruments Incorporated

Table 5-32. Modules Controlled by PLL (continued)

MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	Main_1KByte_ScratchPadRAM, Main_2KByte_ScratchPadRAM, Main_512KByte_SRAM_0, Main_512KByte_SRAM_1, Main_ATL_0, Main_Compute_Cluster_0 Clock control, Main_CPSW_0, Main_CSI_Rx_0, Main_CSI_Rx_1, Main_CSI_Tx_0, Main_CSI_Tx_1, Main_DCC_0, Main_DCC_1, Main_DCC_2, Main_DCC_3, Main_DCC_4, Main_DCC_5, Main_DCC_6, Main_DCC_7, Main_DCC_8, Main_DCC_9, Main_DDR_EW_wrap_0, Main_DDR_EW_wrap_1, Main_DPHY_Rx_0, Main_DPHY_Rx_1, Main_DSS_DSI_0, Main_DSS_DSI_1, Main_DSS_EDP_0, Main_ECAP_0, Main_ECAP_1, Main_ECAP_2, Main_ELM_0, Main_EMMC4_0, Main_EMMC8_0, Main_EPWM_0, Main_EPWM_1, Main_EPWM_2, Main_EPWM_3, Main_EPWM_4, Main_EPWM_5, Main_EQEP_0, Main_EQEP_1, Main_EQEP_2, Main_ESM_0, Main_GPIO_0, Main_GPIO_2, Main_GPIO_4, Main_GPIO_6, Main_GPMC_0, Main_GPU_0, Main_GTC_0, Main_I2C_0, Main_I2C_1, Main_I2C_2, Main_I2C_3, Main_I2C_4, Main_I2C_5, Main_I2C_6, Main_MCANSS_0, Main_MCANSS_1, Main_MCANSS_2, Main_MCANSS_3, Main_MCANSS_4, Main_MCANSS_5, Main_MCANSS_6, Main_MCANSS_7, Main_MCANSS_8, Main_MCANSS_9, Main_MCANSS_10, Main_MCANSS_11, Main_MCANSS_12, Main_MCANSS_13, Main_MCANSS_14, Main_MCANSS_15, Main_MCANSS_16, Main_MCANSS_17, Main_MCASP_0, Main_MCASP_1, Main_MCASP_2, Main_MCASP_3, Main_MCASP_4, Main_NAVSS_0, Main_PCIe_0, Main_PDMA_CPSW_0, Main_PDMA_Debug_0, Main_PDMA_Debug_G0, Main_PDMA_Debug_G1, Main_PDMA_MCAN_0, Main_PDMA_McASP_0, Main_PDMA_SPI_G0, Main_PDMA_SPI_G1, Main_PDMA_UART_0, Main_PDMA_UART_G0, Main_PDMA_UART_G1, Main_PDMA_UART_G2, Main_RTI_A72_0, Main_RTI_A72_1, Main_SerDes_0, Main_RTI_C7x_0, Main_SerDes_1, Main_RTI_C7x_1, Main_SPI_0, Main_RTI_GPU, Main_SPI_1, Main_RTI_Pulsar_R5F_0_0, Main_SPI_2, Main_RTI_Pulsar_R5F_0_1, Main_SPI_3, Main_RTI_Pulsar_R5F_1_0, Main_SPI_4, Main_RTI_Pulsar_R5F_1_1, Main_SPI_5, Main_SA2_UL_0, Main_SPI_6, Main_SPI_7, Main_Timer_0, Main_Timer_1, Main_Timer_2, Main_Timer_3, Main_Timer_4, Main_Timer_5, Main_Timer_6, Main_Timer_7, Main_Timer_8, Main_Timer_9, Main_Timer_10, Main_Timer_11, Main_Timer_12, Main_Timer_13, Main_Timer_14, Main_Timer_15, Main_Timer_16, Main_Timer_17, Main_Timer_18, Main_Timer_19, Main_UART_0, Main_UART_1, Main_UART_2, Main_UART_3, Main_UART_4, Main_UART_5, Main_UART_6, Main_UART_7, Main_UART_8, Main_UART_9, Main_USB3p0SS_0, Main_AC_ECC_Aggr_6, Main_AC_ECC_Aggr_9, Main_CPT2_Aggr_AC_0, Main_CPT2_Aggr_ACP_0, Main_CPT2_Aggr_HC_0, Main_CPT2_Aggr_MI_0, Main_CPT2_Aggr_MV_0, Main_CPT2_Aggr_RC_0, Main_CPT2_Probe_ACP_SRAM_SLV_0, Main_CPT2_Probe_ACP_SRAM_SLV_1, Main_CPT2_Probe_DMPAC_AC_SRAM_SLV_0, Main_CPT2_Probe_NAVSS_AC_DDR_SLV_0, Main_CPT2_Probe_NAVSS_AC_DDR_SLV_1, Main_CPT2_Probe_NAVSS_AC_DDR_SLV_2, Main_CPT2_Probe_NAVSS_AC_SRAM_SLV_0, Main_CPT2_Probe_VPAC_AC_SRAM_SLV_0, Main_CTRL_MMR_0, Main_DebugSS_0, Main_ECC_Aggr_R5_0_0, Main_ECC_Aggr_R5_0_1, Main_ECC_Aggr_R5_1_0, Main_ECC_Aggr_R5_1_1, Main_EFUSE_CTRL_0, Main_HC_ECC_Aggr_5, Main_Infra_ECC_Aggr_0, MAIN_INTROUTER_CMP_EVENT_0, MAIN_INTROUTER_GPIOMUX_0, Main_IP_ECC_Aggr_6, Main_LVL_INTROUTER_Main2MCU_0, Main_NAVSS_ECC_Aggr_10, Main_PBIST_AC_EDP_DSI, Main_PBIST_HC, Main_PBIST_Infra_0, Main_PBIST_Infra_1, Main_PBIST_NAVSS, Main_PBIST_Pulsar_0, Main_PBIST_Pulsar_1, Main_PLL_MMR_0, Main_PLS_INTROUTER_Main2MCU_0, Main_PSC_Wrap_0, Main_RC_ECC_Aggr_4, Main_SEC_MMR_0, Main_TimeSync_INTROUTER_0, Main_Hyperlink_0,
MAIN_PLL0.HSDIV1	Main_DCC_4, Main_SA2_UL_0,
MAIN_PLL0.HSDIV2	Main_DCC_0, Main_EMMC4_0, Main_EMMC8_0,
MAIN_PLL0.HSDIV3	Main_DCC_0, Main_GPMC_0,
MAIN_PLL0.HSDIV4	Main_DCC_0, Main_MCANSS_0, Main_MCANSS_1, Main_MCANSS_2, Main_MCANSS_3, Main_MCANSS_4, Main_MCANSS_5, Main_MCANSS_6, Main_MCANSS_7, Main_MCANSS_8, Main_MCANSS_9, Main_MCANSS_10, Main_MCANSS_11, Main_MCANSS_12, Main_MCANSS_13, Main_MCANSS_14, Main_MCANSS_15, Main_MCANSS_16, Main_MCANSS_17,
MAIN_PLL0.HSDIV5	MCU_DCC_2, Main_DCC_1, Main_SPI_0, Main_SPI_1, Main_SPI_2, Main_SPI_3, Main_SPI_4, Main_SPI_5, Main_SPI_6, Main_SPI_7,
MAIN_PLL0.HSDIV6	MCU_CPSW_0, MCU_DCC_1, Main_CPSW_0, Main_DCC_1, Main_GTC_0, Main_NAVSS_0, Main_PCIe_0,
MAIN_PLL0.HSDIV7	Main_ATL_0, Main_DCC_1,

Copyright © 2024 Texas Instruments Incorporated

Table 5-32. Modules Controlled by PLL (continued)

MAIN_PLL0.HSDIV8	Main_DCC_0, Main_Timer_0, Main_Timer_1, Main_Timer_2, Main_Timer_3, Main_Timer_4, Main_Timer_5, Main_Timer_6, Main_Timer_7, Main_Timer_8, Main_Timer_9, Main_Timer_10, Main_Timer_11, Main_Timer_12, Main_Timer_13, Main_Timer_14, Main_Timer_15, Main_Timer_16, Main_Timer_17, Main_Timer_18, Main_Timer_19,
MAIN_PLL1.HSDIV0	Main_DCC_1, Main_I2C_0, Main_I2C_1, Main_I2C_2, Main_I2C_3, Main_I2C_4, Main_I2C_5, Main_I2C_6, Main_UART_0, Main_UART_1, Main_UART_2, Main_UART_3, Main_UART_4, Main_UART_5, Main_UART_6, Main_UART_7, Main_UART_8, Main_UART_9,
MAIN_PLL1.HSDIV1	Main_CPSW_0, Main_DCC_1,
MAIN_PLL1.HSDIV2	Main_DCC_1, Main_EMMC4_0, Main_EMMC8_0,
MAIN_PLL1.HSDIV3	Main_DCC_1, Main_Timer_0, Main_Timer_1, Main_Timer_2, Main_Timer_3, Main_Timer_4, Main_Timer_5, Main_Timer_6, Main_Timer_7, Main_Timer_8, Main_Timer_9, Main_Timer_10, Main_Timer_11, Main_Timer_12, Main_Timer_13, Main_Timer_14, Main_Timer_15, Main_Timer_16, Main_Timer_17, Main_Timer_18, Main_Timer_19,
MAIN_PLL1.HSDIV5	WKUP_UART_0, MCU_UART_0,
MAIN_PLL1.HSDIV6	Main_DCC_1,
MAIN_PLL1.HSDIV7	Main_USB3p0SS_0,
MAIN_PLL1.HSDIV8	Main_CSI_Tx_0, Main_CSI_Tx_1, Main_DCC_2, Main_DSS_DSI_0, Main_DSS_DSI_1, Main_SerDes_0, Main_SerDes_1,
MAIN_PLL2.HSDIV1	Main_DCC_2, Main_DSS_0, Main_GPMC_0,
MAIN_PLL2.HSDIV2	Main_ATL_0, Main_DCC_2, Main_EMMC4_0, Main_EMMC8_0, Main_MCASP_0, Main_MCASP_1, Main_MCASP_2, Main_MCASP_3, Main_MCASP_4,
MAIN_PLL2.HSDIV3	Main_DebugSS_0,
MAIN_PLL2.HSDIV4	Main_DCC_2, Main_SerDes_0, Main_SerDes_1,
MAIN_PLL2.HSDIV6	Main_DCC_2, Main_Timer_0, Main_Timer_1, Main_Timer_2, Main_Timer_3, Main_Timer_4, Main_Timer_5, Main_Timer_6, Main_Timer_7, Main_Timer_8, Main_Timer_9, Main_Timer_10, Main_Timer_11, Main_Timer_12, Main_Timer_13, Main_Timer_14, Main_Timer_15, Main_Timer_16, Main_Timer_17, Main_Timer_18, Main_Timer_19,
MAIN_PLL2.HSDIV7	Main_DCC_2, Main_DCC_7, Main_DSS_DSI_0, Main_DSS_DSI_1, Main_DSS_EDP_0,
MAIN_PLL3.HSDIV0	Main_CPSW_0, Main_DCC_2,
MAIN_PLL3.HSDIV1	MCU_CPSW_0, MCU_DCC_1, Main_CPSW_0, Main_DCC_8, Main_GTC_0, Main_NAVSS_0, Main_PCIe_0,
MAIN_PLL3.HSDIV2	Main_DCC_8, Main_EMMC4_0, Main_EMMC8_0,
MAIN_PLL3.HSDIV3	Main_DCC_9, Main_Timer_0, Main_Timer_1, Main_Timer_2, Main_Timer_3, Main_Timer_4, Main_Timer_5, Main_Timer_6, Main_Timer_7, Main_Timer_8, Main_Timer_9, Main_Timer_10, Main_Timer_11, Main_Timer_12, Main_Timer_13, Main_Timer_14, Main_Timer_15, Main_Timer_16, Main_Timer_17, Main_Timer_18, Main_Timer_19,
MAIN_PLL3.HSDIV4	Main_DCC_9, Main_SerDes_0, Main_SerDes_1,
MAIN_PLL4.HSDIV0	Main_DCC_3, Main_MCASP_0, Main_MCASP_1, Main_MCASP_2, Main_MCASP_3, Main_MCASP_4,
MAIN_PLL4.HSDIV1	Main_ATL_0, Main_DCC_9,
MAIN_PLL4.HSDIV2	Main_DCC_7, Main_DCC_9, Main_Timer_0, Main_Timer_1, Main_Timer_2, Main_Timer_3, Main_Timer_4, Main_Timer_5, Main_Timer_6, Main_Timer_7, Main_Timer_8, Main_Timer_9, Main_Timer_10, Main_Timer_11, Main_Timer_12, Main_Timer_13, Main_Timer_14, Main_Timer_15, Main_Timer_16, Main_Timer_17, Main_Timer_18, Main_Timer_19,
MAIN_PLL5.HSDIV0	Main_DCC_3,
MAIN_PLL5.HSDIV1	Main_DCC_3,
MAIN_PLL6.HSDIV0	Main_DCC_3, Main_GPU_0,
MAIN_PLL7.HSDIV0	C71SS (with MMA), C71SS, MSMC, Main_DCC_3, Main_DDR_EW_wrap_0, Main_DDR_EW_wrap_1, Main_NAVSS_0, Main_NBSS_0,
MAIN_PLL8.HSDIV0	Dual A72SS, Main_DCC_3,
MAIN_PLL9.HSDIV0	Main_DCC_3,
MAIN_PLL12.HSDIV0	Main_DCC_4, Main_DDR_EW_wrap_0,
MAIN_PLL14.HSDIV0	Main_DCC_4, Main_Pulsar_0,

Copyright © 2024 Texas Instruments Incorporated

Table 5-32. Modules Controlled by PLL (continued)

MAIN_PLL14.HSDIV1	Main_DCC_4, Main_Pulsar_1,
MAIN_PLL16.HSDIV0	Main_DCC_4, Main_DSS_0,
MAIN_PLL17.HSDIV0	Main_DCC_5, Main_DSS_0,
MAIN_PLL19.HSDIV0	Main_DCC_5, Main_DSS_0,
MAIN_PLL25.HSDIV0	Main_DCC_5, Main_DMPAC_0,
MAIN_PLL25.HSDIV1	Main_CSI_Rx_0, Main_CSI_Rx_1, Main_DCC_5,
MAIN_PLL26.HSDIV0	Main_DCC_4, Main_DDR_EW_wrap_1,
MAIN_PLL27.HSDIV0	Main_DCC_4,

5.4.3 Clock Mapping

Table 5-33 summarizes the full connectivity of input clocks, module clocks, and PLLs.

Table 5-33. Clock Mapping

Module	Clock Port	Clock	Divider	Control Register	Mux Value	Default Mux Value	Typical Frequency
WKUP_DDPA_0	ddpa_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6				167
WKUP_ESM_0	clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6				167
WKUP_GPIO_0	mmr_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6	WKUP_GPIO_CLKSEL	0		167
WKUP_GPIO_0	mmr_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6	WKUP_GPIO_CLKSEL	1		
WKUP_GPIO_0	mmr_clk	CLK_32K	1	WKUP_GPIO_CLKSEL	2		
WKUP_GPIO_0	mmr_clk	CLK_12M_RC	1	WKUP_GPIO_CLKSEL	3		
WKUP_GPIO_1	mmr_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6				167
WKUP_GPIO_1	mmr_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6	WKUP_GPIO_CLKSEL	0		
WKUP_GPIO_1	mmr_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6	WKUP_GPIO_CLKSEL	1		
WKUP_GPIO_1	mmr_clk	CLK_32K	1	WKUP_GPIO_CLKSEL	2		
WKUP_I2C_0	clk	CLK_12M_RC	1	WKUP_GPIO_CLKSEL	3		167
WKUP_I2C_0	piscl	WKUP_I2C0_SCL	1				
WKUP_I2C_0	pisys_clk	MCU_PLL1.HSDIV3	1	WKUP_PER_CKSEL	0	0	96
WKUP_I2C_0	pisys_clk	HFOSC0	1	WKUP_PER_CKSEL	1	0	[19.2, 20, 24, 25, 26, 27]
WKUP_SMS_0	dap_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	3				333
WKUP_SMS_0	ext_clk	MCU_EXT_REFCLK0	1				100
WKUP_SMS_0	func_32k_rc_clk	CLK_32K	1				0.032000
WKUP_SMS_0	func_32k_rt_clk	LFXOSC	1				0.032768
WKUP_SMS_0	func_mosc_clk	HFOSC0	1				[19.2, 20, 24, 25, 26, 27]
WKUP_SMS_0	rti_timer1_clk	WKUP_TIEOFF LOW	1				
WKUP_SMS_0	rti_timer2_clk	WKUP_TIEOFF LOW	1				
WKUP_SMS_0	rti_timer3_clk	WKUP_TIEOFF LOW	1				
WKUP_SMS_0	sec_efc_fclk	IJ7_EFUSE_CTRL_WRAP_MCU_0.EFC_FCLK	1				
WKUP_SMS_0	vbus_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	3				333
WKUP_UART_0	fclk_clk	MCU_PLL1.HSDIV3	1	WKUP_USART_CLKSEL	0	0	96
WKUP_UART_0	fclk_clk	MAIN_PLL1.HSDIV5	1	WKUP_USART_CLKSEL	1	0	
WKUP_UART_0	fclk_clk	MCU_PLL1.HSDIV3	1	WKUP_PER_CKSEL	0	0	96
WKUP_UART_0	fclk_clk	MAIN_PLL1.HSDIV5	1	WKUP_PER_CKSEL	0	0	
WKUP_UART_0	fclk_clk	HFOSC0	1	WKUP_PER_CKSEL	1	0	[19.2, 20, 24, 25, 26, 27]
WKUP_UART_0	sclki_clk	MCU_TIEOFF LOW	1				

Copyright © 2024 Texas Instruments Incorporated

Table 5-33. Clock Mapping (continued)

Module	Clock Port	Clock	Divider	Control Register	Mux Value	Default Mux Value	Typical Frequency
WKUP_UART_0	vbusp_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6				167
WKUP_VTM_0	fix_ref_clk	HFOSC0	1				[19.2, 20, 24, 25, 26, 27]
WKUP_VTM_0	fix_ref2_clk	CLK_12M_RC	1				13
WKUP_VTM_0	vbusp_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6				167
WKUP_CTRL_MMR_0	vbusp_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6				167
WKUP_ECC_Aggr_0	clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	3				333
WKUP_INTROUTER_GPIOMUX	intr_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6				167
WKUP_PSC_0	clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6				167
WKUP_PSC_0	slow_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	24				42
MCU_1MByte_SRAM	cclk_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6				167
MCU_1MByte_SRAM	vclk_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	3				333
MCU_512B_ScratchpadRAM	clk_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6				167
MCU_ADC_0	adc_clk	HFOSC0	1	MCU_ADC0_CLKSEL	0	0	[19.2, 20, 24, 25, 26, 27]
MCU_ADC_0	adc_clk	MCU_PLL1.HSDIV1	1	MCU_ADC0_CLKSEL	1	0	60
MCU_ADC_0	adc_clk	MCU_PLL0.HSDIV1	1	MCU_ADC0_CLKSEL	2	0	60
MCU_ADC_0	adc_clk	MCU_EXT_REFCLK0	1	MCU_ADC0_CLKSEL	3	0	100
MCU_ADC_0	sys_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	2				500
MCU_ADC_0	vbus_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	3				333
MCU_ADC_1	adc_clk	HFOSC0	1	MCU_ADC1_CLKSEL	0	0	[19.2, 20, 24, 25, 26, 27]
MCU_ADC_1	adc_clk	MCU_PLL1.HSDIV1	1	MCU_ADC1_CLKSEL	1	0	60
MCU_ADC_1	adc_clk	MCU_PLL0.HSDIV1	1	MCU_ADC1_CLKSEL	2	0	60
MCU_ADC_1	adc_clk	MCU_EXT_REFCLK0	1	MCU_ADC1_CLKSEL	3	0	100
MCU_ADC_1	sys_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	2				500
MCU_ADC_1	vbus_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	3				333
MCU_CPSW_0	cppi_clk_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	3				333
MCU_CPSW_0	cpts_rft_clk	MAIN_PLL3.HSDIV1	1	MCU_ENET_CLKSEL	0	0	250
MCU_CPSW_0	cpts_rft_clk	MAIN_PLL0.HSDIV6	1	MCU_ENET_CLKSEL	1	0	250
MCU_CPSW_0	cpts_rft_clk	MCU_CPTS0_RFT_CLK	1	MCU_ENET_CLKSEL	2	0	
MCU_CPSW_0	cpts_rft_clk	CPTS0_RFT_CLK	1	MCU_ENET_CLKSEL	3	0	
MCU_CPSW_0	cpts_rft_clk	MCU_EXT_REFCLK0	1	MCU_ENET_CLKSEL	4	0	100
MCU_CPSW_0	cpts_rft_clk	EXT_REFCLK1	1	MCU_ENET_CLKSEL	5	0	
MCU_CPSW_0	cpts_rft_clk	MAIN_SERDES_0.IP2_LN0_TXMCLK	1	MCU_ENET_CLKSEL	6	0	
MCU_CPSW_0	cpts_rft_clk	MAIN_SERDES_0.IP2_LN1_TXMCLK	1	MCU_ENET_CLKSEL	7	0	
MCU_CPSW_0	cpts_rft_clk	MAIN_SERDES_0.IP2_LN2_TXMCLK	1	MCU_ENET_CLKSEL	8	0	
MCU_CPSW_0	cpts_rft_clk	MAIN_SERDES_0.IP2_LN3_TXMCLK	1	MCU_ENET_CLKSEL	9	0	
MCU_CPSW_0	cpts_rft_clk	MCU_PLL2.HSDIV1	1	MCU_ENET_CLKSEL	14	0	500
MCU_CPSW_0	cpts_rft_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	2	MCU_ENET_CLKSEL	15	0	500
MCU_CPSW_0	gmii_rft_clk	MCU_PLL2.HSDIV0	2				125
MCU_CPSW_0	gmii1_mr_clk	MCU_PLL2.HSDIV0	10				25
MCU_CPSW_0	gmii1_mt_clk	MCU_PLL2.HSDIV0	10				25
MCU_CPSW_0	rgmii_mhz_250_clk	MCU_PLL2.HSDIV0	1				250
MCU_CPSW_0	rgmii_mhz_5_clk	MCU_PLL2.HSDIV0	50				5

Copyright © 2024 Texas Instruments Incorporated

Table 5-33. Clock Mapping (continued)

Module	Clock Port	Clock	Divider	Control Register	Mux Value	Default Mux Value	Typical Frequency
MCU_CPSW_0	rgmii_mhz_50_clk	MCU_PLL2.HSDIV0	5				50
MCU_CPSW_0	rgmii1_rxc_i	MCU_RGMII1_RXC	1				
MCU_CPSW_0	rgmii1_txc_i	MCU_TIEOFF LOW	1				
MCU_CPSW_0	rmii_mhz_50_clk	MCU_RMII1_REF_CLK	1				
MCU_CPSW_0	sgmii1_rxb_clk	MCU_TIEOFF LOW	1				
MCU_CPSW_0	sgmii1_txb_clk	MCU_TIEOFF LOW	1				
MCU_CPSW_0	rgmii_txc_o	MCU_RGMII_TXC	1				
MCU_CPSW_0	mdio_mdclk_o	MCU_MDIO_CLK	1				
MCU_CPSW_0	cpts_genf0	CPTS_GENF0	1				
MCU_CPSW_0	cpts_genf1	CPTS_GENF1	1				
MCU_DCC_0	dcc_input10_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	3	DCC_CLKSRC1	0	0	333
MCU_DCC_0	dcc_clksrc0_clk	MCU_PLL1.HSDIV0	2	DCC_CLKSRC1	1	0	200
MCU_DCC_0	dcc_clksrc1_clk	MCU_PLL1.HSDIV1	1	DCC_CLKSRC1	2	0	60
MCU_DCC_0	dcc_clksrc2_clk	MCU_PLL1.HSDIV2	1	DCC_CLKSRC1	3	0	80
MCU_DCC_0	dcc_clksrc3_clk	MCU_PLL1.HSDIV3	1	DCC_CLKSRC1	4	0	96
MCU_DCC_0	dcc_clksrc4_clk	MCU_PLL1.HSDIV4	1	DCC_CLKSRC1	5	0	400
MCU_DCC_0	dcc_clksrc5_clk	CLK_32K	1	DCC_CLKSRC1	6	0	0.032000
MCU_DCC_0	dcc_clksrc6_clk	LFXOSC	1	DCC_CLKSRC1	7	0	0.032768
MCU_DCC_0	dcc_clksrc7_clk	MCU_EXT_REFCLK0	1	DCC_CLKSRC1	8	0	100
MCU_DCC_0	dcc_input00_clk	HFOSC0	1	DCC_CLKSRC0	0	0	[19.2, 20, 24, 25, 26, 27]
MCU_DCC_0	dcc_input01_clk	CLK_32K	1	DCC_CLKSRC0	1	0	0
MCU_DCC_0	dcc_input02_clk	CLK_12M_RC	1	DCC_CLKSRC0	2	0	13
MCU_DCC_0	vbus_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6				167
MCU_DCC_1	dcc_input10_clk	MCU_PLL2.HSDIV0	1	DCC_CLKSRC1	0	0	250
MCU_DCC_1	dcc_clksrc0_clk	MCU_PLL2.HSDIV1	2	DCC_CLKSRC1	1	0	250
MCU_DCC_1	dcc_clksrc1_clk	MCU_PLL2.HSDIV2	1	DCC_CLKSRC1	2	0	200
MCU_DCC_1	dcc_clksrc2_clk	MCU_PLL2.HSDIV3	1	DCC_CLKSRC1	3	0	80
MCU_DCC_1	dcc_clksrc3_clk	MCU_PLL2.HSDIV4	1	DCC_CLKSRC1	4	0	333
MCU_DCC_1	dcc_clksrc4_clk	MCU_PLL0.HSDIV0	4	DCC_CLKSRC1	5	0	250
MCU_DCC_1	dcc_clksrc5_clk	MCU_PLL0.HSDIV1	1	DCC_CLKSRC1	6	0	60
MCU_DCC_1	dcc_clksrc6_clk	MCU_OSPI0_LBCLKO	1	DCC_CLKSRC1	7	0	
MCU_DCC_1	dcc_clksrc7_clk	MAIN_PLL3.HSDIV1	1	DCC_CLKSRC1 / MCU_ENET_CLKSEL	8 / 0	0 / 0	250
MCU_DCC_1	dcc_clksrc7_clk	MAIN_PLL0.HSDIV6	1	DCC_CLKSRC1 / MCU_ENET_CLKSEL	8 / 1	0 / 0	250
MCU_DCC_1	dcc_clksrc7_clk	MCU_CPTS0_RFT_CLK	1	DCC_CLKSRC1 / MCU_ENET_CLKSEL	8 / 2	0 / 0	
MCU_DCC_1	dcc_clksrc7_clk	CPTS0_RFT_CLK	1	DCC_CLKSRC1 / MCU_ENET_CLKSEL	8 / 3	0 / 0	
MCU_DCC_1	dcc_clksrc7_clk	MCU_EXT_REFCLK0	1	DCC_CLKSRC1 / MCU_ENET_CLKSEL	8 / 4	0 / 0	100
MCU_DCC_1	dcc_clksrc7_clk	EXT_REFCLK1	1	DCC_CLKSRC1 / MCU_ENET_CLKSEL	8 / 5	0 / 0	
MCU_DCC_1	dcc_clksrc7_clk	MAIN_SERDES_0.IP2_LN0_TXMCLK	1	DCC_CLKSRC1 / MCU_ENET_CLKSEL	8 / 6	0 / 0	
MCU_DCC_1	dcc_clksrc7_clk	MAIN_SERDES_0.IP2_LN1_TXMCLK	1	DCC_CLKSRC1 / MCU_ENET_CLKSEL	8 / 7	0 / 0	
MCU_DCC_1	dcc_clksrc7_clk	MAIN_SERDES_0.IP2_LN2_TXMCLK	1	DCC_CLKSRC1 / MCU_ENET_CLKSEL	8 / 8	0 / 0	
MCU_DCC_1	dcc_clksrc7_clk	MAIN_SERDES_0.IP2_LN3_TXMCLK	1	DCC_CLKSRC1 / MCU_ENET_CLKSEL	8 / 9	0 / 0	
MCU_DCC_1	dcc_clksrc7_clk	MCU_PLL2.HSDIV1	1	DCC_CLKSRC1 / MCU_ENET_CLKSEL	8 / 14	0 / 0	500
MCU_DCC_1	dcc_clksrc7_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	2	DCC_CLKSRC1 / MCU_ENET_CLKSEL	8 / 15	0 / 0	500
MCU_DCC_1	dcc_input00_clk	HFOSC0	1				[19.2, 20, 24, 25, 26, 27]
MCU_DCC_1	dcc_input01_clk	LFXOSC	1				0.032768

Copyright © 2024 Texas Instruments Incorporated

Table 5-33. Clock Mapping (continued)

Module	Clock Port	Clock	Divider	Control Register	Mux Value	Default Mux Value	Typical Frequency
MCU_DCC_1	dcc_input02_clk	CLK_12M_RC	1				13
MCU_DCC_1	vbus_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6				167
MCU_DCC_2	dcc_input10_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	3	DCC_CLKSRC1	0	0	333
MCU_DCC_2	dcc_clksrc0_clk	MCU_RMII1_REF_CLK	1	DCC_CLKSRC1	1	0	
MCU_DCC_2	dcc_clksrc1_clk	RGMII1_RXC	1	DCC_CLKSRC1	2	0	
MCU_DCC_2	dcc_clksrc2_clk	HFOSC1	1	DCC_CLKSRC1	3	0	[19.2 - 27]
MCU_DCC_2	dcc_clksrc3_clk	MAIN_PLL0.HSDIV5	1	DCC_CLKSRC1	4	0	50
MCU_DCC_2	dcc_clksrc4_clk	MCU_OSPI1_LBCLKO	1	DCC_CLKSRC1	5	0	
MCU_DCC_2	dcc_clksrc5_clk	MCU_TIEOFF LOW	1	DCC_CLKSRC1	6	0	
MCU_DCC_2	dcc_clksrc6_clk	CLK_12M_RC	1	DCC_CLKSRC1	7	0	13
MCU_DCC_2	dcc_clksrc7_clk	HFOSC0	1	DCC_CLKSRC1	8	0	[19.2, 20, 24, 25, 26, 27]
MCU_DCC_2	dcc_input00_clk	HFOSC0	1	DCC_CLKSRC0	0	0	[19.2, 20, 24, 25, 26, 27]
MCU_DCC_2	dcc_input01_clk	MCU_EXT_REFCLK0	1	DCC_CLKSRC0	1	0	100
MCU_DCC_2	dcc_input02_clk	CLK_12M_RC	1	DCC_CLKSRC0	2	0	13
MCU_DCC_2	vbus_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6				167
MCU_ESM_0	clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6				167
MCU_FSS_0	hpb_clkx1_clk	MCU_PLL2.HSDIV4	2				167
MCU_FSS_0	hpb_clkx1_inv_clk	MCU_PLL2.HSDIV4	2				167
MCU_FSS_0	hpb_clkx2_clk	MCU_PLL2.HSDIV4	1				333
MCU_FSS_0	hpb_clkx2_inv_clk	MCU_PLL2.HSDIV4	1				333
MCU_FSS_0	ospi0_dqs_clk	MCU_OSPI0_DQS	1				
MCU_FSS_0	ospi0_iclk_clk	MCU_OSPI0_DQS	1	MCU_OSPI0_CLKSEL	0	0	
MCU_FSS_0	ospi0_iclk_clk	MCU_FSS_0.OSPI0_OCLK_CLK	1	MCU_OSPI0_CLKSEL	1	0	
MCU_FSS_0	ospi0_rclk_clk	MCU_PLL1.HSDIV4	1	MCU_OSPI0_CLKSEL	0	0	400
MCU_FSS_0	ospi0_rclk_clk	MCU_PLL2.HSDIV4	1	MCU_OSPI0_CLKSEL	1	0	333
MCU_FSS_0	ospi1_dqs_clk	MCU_OSPI1_DQS	1				
MCU_FSS_0	ospi1_iclk_clk	MCU_OSPI1_DQS	1	MCU_OSPI1_CLKSEL	0	0	
MCU_FSS_0	ospi1_iclk_clk	MCU_FSS_0.OSPI1_OCLK_CLK	1	MCU_OSPI1_CLKSEL	1	0	
MCU_FSS_0	ospi1_rclk_clk	MCU_PLL1.HSDIV4	1	MCU_OSPI1_CLKSEL	0	1	400
MCU_FSS_0	ospi1_rclk_clk	MCU_PLL2.HSDIV4	1	MCU_OSPI1_CLKSEL	1	1	333
MCU_FSS_0	vbus_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	3				333
MCU_I2C_0	clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6				167
MCU_I2C_0	piscl	MCU_I2C0_SCL	1				
MCU_I2C_0	pisys_clk	MCU_PLL1.HSDIV3	1				96
MCU_I2C_1	clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6				167
MCU_I2C_1	piscl	MCU_I2C1_SCL	1				
MCU_I2C_1	pisys_clk	MCU_PLL1.HSDIV3	1				96
MCU_I3C_0	i3c_pclk_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6				167
MCU_I3C_0	i3c_scl_di	MCU_I3C0_SCL	1				
MCU_I3C_0	i3c_sclk_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6				167
MCU_I3C_0	i3c_sda_di	MCU_I3C0_SDA	1				
MCU_I3C_1	i3c_pclk_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6				167
MCU_I3C_1	i3c_scl_di	MCU_TIEOFF LOW	1				
MCU_I3C_1	i3c_sclk_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6				167
MCU_I3C_1	i3c_sda_di	MCU_TIEOFF LOW	1				
MCU_MCANSS_0	mcanss_can_rxd	MCU_MCAN0_RX	1				
MCU_MCANSS_0	mcanss_cclk_clk	MCU_PLL2.HSDIV3	1	MCU_MCAN0_CLKSEL	0	0	80

Copyright © 2024 Texas Instruments Incorporated

Table 5-33. Clock Mapping (continued)

Module	Clock Port	Clock	Divider	Control Register	Mux Value	Default Mux Value	Typical Frequency
MCU_MCANSS_0	mcanss_cclk_clk	MCU_EXT_REFCLK0	1	MCU_MCAN0_CLKSEL	1	0	100
MCU_MCANSS_0	mcanss_cclk_clk	MCU_PLL1.HSDIV2	1	MCU_MCAN0_CLKSEL	2	0	80
MCU_MCANSS_0	mcanss_cclk_clk	HFOSC0	1	MCU_MCAN0_CLKSEL	3	0	[19.2, 20, 24, 25, 26, 27]
MCU_MCANSS_0	mcanss_hclk_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6				167
MCU_MCANSS_1	mcanss_can_rxd	MCU_MCAN1_RX	1				
MCU_MCANSS_1	mcanss_cclk_clk	MCU_PLL2.HSDIV3	1	MCU_MCAN1_CLKSEL	0	1	80
MCU_MCANSS_1	mcanss_cclk_clk	MCU_EXT_REFCLK0	1	MCU_MCAN1_CLKSEL	1	1	100
MCU_MCANSS_1	mcanss_cclk_clk	MCU_PLL1.HSDIV2	1	MCU_MCAN1_CLKSEL	2	1	80
MCU_MCANSS_1	mcanss_cclk_clk	HFOSC0	1	MCU_MCAN1_CLKSEL	3	1	[19.2, 20, 24, 25, 26, 27]
MCU_MCANSS_1	mcanss_hclk_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6				167
MCU_PDMA_ADC_0	vclk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	3				333
MCU_PDMA_G0	vclk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6				167
MCU_PDMA_G1	vclk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6				167
MCU_PDMA_G2	vclk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6				167
MCU_Pulsar_0	cpu0_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	1	MCU_R5_CORE0_CLK SEL	0	0	1000
MCU_Pulsar_0	cpu0_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	3	MCU_R5_CORE0_CLK SEL	1	0	333
MCU_Pulsar_0	cpu1_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	1	MCU_R5_CORE0_CLK SEL	0	0	1000
MCU_Pulsar_0	cpu1_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	3	MCU_R5_CORE0_CLK SEL	1	0	333
MCU_Pulsar_0	interface0_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	3				333
MCU_Pulsar_0	interface0_phase	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	3	MCU_R5_CORE0_CLK SEL	0	0	333
MCU_Pulsar_0	interface0_phase	MCU_TIEOFF HIGH	1	MCU_R5_CORE0_CLK SEL	1	0	
MCU_Pulsar_0	interface1_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	3				333
MCU_Pulsar_0	interface1_phase	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	3	MCU_R5_CORE0_CLK SEL	0	0	333
MCU_Pulsar_0	interface1_phase	MCU_TIEOFF HIGH	1	MCU_R5_CORE0_CLK SEL	1	0	
MCU_RTI_0	rti_clk	HFOSC0	1	MCU_RTI0_CLKSEL	0	0	[19.2, 20, 24, 25, 26, 27]
MCU_RTI_0	rti_clk	LFXOSC	1	MCU_RTI0_CLKSEL	1	0	0.032768
MCU_RTI_0	rti_clk	CLK_12M_RC	1	MCU_RTI0_CLKSEL	2	0	13
MCU_RTI_0	rti_clk	CLK_32K	1	MCU_RTI0_CLKSEL	3	0	0.032000
MCU_RTI_0	vbusp_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6				167
MCU_RTI_1	rti_clk	HFOSC0	1	MCU_RTI1_CLKSEL	0	1	[19.2, 20, 24, 25, 26, 27]
MCU_RTI_1	rti_clk	LFXOSC	1	MCU_RTI1_CLKSEL	1	1	0.032768
MCU_RTI_1	rti_clk	CLK_12M_RC	1	MCU_RTI1_CLKSEL	2	1	13
MCU_RTI_1	rti_clk	CLK_32K	1	MCU_RTI1_CLKSEL	3	1	0.032000
MCU_RTI_1	vbusp_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6				167
MCU_SA3SS_0	pka_in_clk	MCU_PLL1.HSDIV0	1				400
MCU_SA3SS_0	x1_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6				167
MCU_SA3SS_0	x2_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	3				333
MCU_SPI_0	clkspiref_clk	MCU_PLL2.HSDIV0	5				50

Copyright © 2024 Texas Instruments Incorporated

Table 5-33. Clock Mapping (continued)

Module	Clock Port	Clock	Divider	Control Register	Mux Value	Default Mux Value	Typical Frequency
MCU_SPI_0	io_clkspii_clk	MCU_SPI0_CLK	1	MCU_SPI0_CLK_LPBK_MUX	0	0	
MCU_SPI_0	io_clkspii_clk	MCU_SPI_0.IO_CLKSPIO_CLK	1	MCU_SPI0_CLK_LPBK_MUX	1	0	
MCU_SPI_0	vbusp_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6				167
MCU_SPI_1	clkspiref_clk	MCU_PLL2.HSDIV0	5				50
MCU_SPI_1	io_clkspii_clk	MCU_SPI1_CLK	1	MCU_SPI1_CLK_LPBK_MUX	0	0	
MCU_SPI_1	io_clkspii_clk	MCU_SPI_1.IO_CLKSPIO_CLK	1	MCU_SPI1_CLK_LPBK_MUX	1	0	
MCU_SPI_1	io_clkspii_clk	MAIN_SPI_3.IO_CLKSPIO_CLK	1	MCU_SPI1_CLK_MUX	0	0	
MCU_SPI_1	io_clkspii_clk	MCU_SPI_1.IO_CLKSPIO_CLK	1	MCU_SPI1_CLK_MUX	1	0	
MCU_SPI_1	io_clkspii_clk	MCU_SPI1_CLK	1	MCU_SPI1_CLK_MUX	1	0	
MCU_SPI_1	vbusp_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6				167
MCU_SPI_2	clkspiref_clk	MCU_PLL2.HSDIV0	5				50
MCU_SPI_2	io_clkspii_clk	MCU_SPI_2.IO_CLKSPIO_CLK	1				
MCU_SPI_2	vbusp_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6				167
MCU_Timer_0	timer_hclk_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6				167
MCU_Timer_0	timer_tclk_clk	HFOSC0	1	MCU_TIMER0_CLKSEL	0	0	[19.2, 20, 24, 25, 26, 27]
MCU_Timer_0	timer_tclk_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	4	MCU_TIMER0_CLKSEL	1	0	250
MCU_Timer_0	timer_tclk_clk	CLK_12M_RC	1	MCU_TIMER0_CLKSEL	2	0	13
MCU_Timer_0	timer_tclk_clk	MCU_PLL2.HSDIV2	1	MCU_TIMER0_CLKSEL	3	0	200
MCU_Timer_0	timer_tclk_clk	MCU_EXT_REFCLK0	1	MCU_TIMER0_CLKSEL	4	0	100
MCU_Timer_0	timer_tclk_clk	LFXOSC	1	MCU_TIMER0_CLKSEL	5	0	0.032768
MCU_Timer_0	timer_tclk_clk	MCU_CPSW_0.CPTS_GENF0	1	MCU_TIMER0_CLKSEL	6	0	
MCU_Timer_0	timer_tclk_clk	CLK_32K	1	MCU_TIMER0_CLKSEL	7	0	0.032000
MCU_Timer_1	timer_hclk_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6				167
MCU_Timer_1	timer_tclk_clk	HFOSC0	1	MCU_TIMER1_CLKSEL	0	1	[19.2, 20, 24, 25, 26, 27]
MCU_Timer_1	timer_tclk_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	1	MCU_TIMER1_CLKSEL	1	1	250
MCU_Timer_1	timer_tclk_clk	CLK_12M_RC	1	MCU_TIMER1_CLKSEL	2	1	13
MCU_Timer_1	timer_tclk_clk	MCU_PLL2.HSDIV2	1	MCU_TIMER1_CLKSEL	3	1	200
MCU_Timer_1	timer_tclk_clk	MCU_EXT_REFCLK0	1	MCU_TIMER1_CLKSEL	4	1	100
MCU_Timer_1	timer_tclk_clk	LFXOSC	1	MCU_TIMER1_CLKSEL	5	1	0.032768
MCU_Timer_1	timer_tclk_clk	MCU_CPSW_0.CPTS_GENF0	1	MCU_TIMER1_CLKSEL	6	1	
MCU_Timer_1	timer_tclk_clk	CLK_32K	1	MCU_TIMER1_CLKSEL	7	1	0.032000
MCU_Timer_1	timer_tclk_clk	HFOSC0	1	MCU_TIMER1_CTRL	0	0	[19.2, 20, 24, 25, 26, 27]
MCU_Timer_1	timer_tclk_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	1	MCU_TIMER1_CTRL	0	0	1000
MCU_Timer_1	timer_tclk_clk	CLK_12M_RC	1	MCU_TIMER1_CTRL	0	0	13
MCU_Timer_1	timer_tclk_clk	MCU_PLL2.HSDIV2	1	MCU_TIMER1_CTRL	0	0	200
MCU_Timer_1	timer_tclk_clk	MCU_EXT_REFCLK0	1	MCU_TIMER1_CTRL	0	0	100
MCU_Timer_1	timer_tclk_clk	LFXOSC	1	MCU_TIMER1_CTRL	0	0	0.032768
MCU_Timer_1	timer_tclk_clk	MCU_CPSW_0.CPTS_GENF0	1	MCU_TIMER1_CTRL	0	0	
MCU_Timer_1	timer_tclk_clk	CLK_32K	1	MCU_TIMER1_CTRL	0	0	0.032000
MCU_Timer_1	timer_tclk_clk	MCU_TIMER_0.TIMER_PWM	1	MCU_TIMER1_CTRL	1	0	
MCU_Timer_2	timer_hclk_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6				167
MCU_Timer_2	timer_tclk_clk	HFOSC0	1	MCU_TIMER2_CLKSEL	0	2	[19.2, 20, 24, 25, 26, 27]

Copyright © 2024 Texas Instruments Incorporated

Table 5-33. Clock Mapping (continued)

Module	Clock Port	Clock	Divider	Control Register	Mux Value	Default Mux Value	Typical Frequency
MCU_Timer_2	timer_tclk_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	1	MCU_TIMER2_CLKSEL	1	2	250
MCU_Timer_2	timer_tclk_clk	CLK_12M_RC	1	MCU_TIMER2_CLKSEL	2	2	13
MCU_Timer_2	timer_tclk_clk	MCU_PLL2.HSDIV2	1	MCU_TIMER2_CLKSEL	3	2	200
MCU_Timer_2	timer_tclk_clk	MCU_EXT_REFCLK0	1	MCU_TIMER2_CLKSEL	4	2	100
MCU_Timer_2	timer_tclk_clk	LFXOSC	1	MCU_TIMER2_CLKSEL	5	2	0.032768
MCU_Timer_2	timer_tclk_clk	MCU_CPSW_0.CPTS_GENF0	1	MCU_TIMER2_CLKSEL	6	2	
MCU_Timer_2	timer_tclk_clk	CLK_32K	1	MCU_TIMER2_CLKSEL	7	2	0.032000
MCU_Timer_3	timer_hclk_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6				167
MCU_Timer_3	timer_tclk_clk	HFOSC0	1	MCU_TIMER3_CLKSEL	0	3	[19.2, 20, 24, 25, 26, 27]
MCU_Timer_3	timer_tclk_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	4	MCU_TIMER3_CLKSEL	1	3	250
MCU_Timer_3	timer_tclk_clk	CLK_12M_RC	1	MCU_TIMER3_CLKSEL	2	3	13
MCU_Timer_3	timer_tclk_clk	MCU_PLL2.HSDIV2	1	MCU_TIMER3_CLKSEL	3	3	200
MCU_Timer_3	timer_tclk_clk	MCU_EXT_REFCLK0	1	MCU_TIMER3_CLKSEL	4	3	100
MCU_Timer_3	timer_tclk_clk	LFXOSC	1	MCU_TIMER3_CLKSEL	5	3	0.032768
MCU_Timer_3	timer_tclk_clk	MCU_CPSW_0.CPTS_GENF0	1	MCU_TIMER3_CLKSEL	6	3	
MCU_Timer_3	timer_tclk_clk	CLK_32K	1	MCU_TIMER3_CLKSEL	7	3	0.032000
MCU_Timer_3	timer_tclk_clk	HFOSC0	1	MCU_TIMER3_CTRL	0	0	[19.2, 20, 24, 25, 26, 27]
MCU_Timer_3	timer_tclk_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	1	MCU_TIMER3_CTRL	0	0	1000
MCU_Timer_3	timer_tclk_clk	CLK_12M_RC	1	MCU_TIMER3_CTRL	0	0	13
MCU_Timer_3	timer_tclk_clk	MCU_PLL2.HSDIV2	1	MCU_TIMER3_CTRL	0	0	200
MCU_Timer_3	timer_tclk_clk	MCU_EXT_REFCLK0	1	MCU_TIMER3_CTRL	0	0	100
MCU_Timer_3	timer_tclk_clk	LFXOSC	1	MCU_TIMER3_CTRL	0	0	0.032768
MCU_Timer_3	timer_tclk_clk	MCU_CPSW_0.CPTS_GENF0	1	MCU_TIMER3_CTRL	0	0	
MCU_Timer_3	timer_tclk_clk	CLK_32K	1	MCU_TIMER3_CTRL	0	0	0.032000
MCU_Timer_3	timer_tclk_clk	MCU_TIMER_2.TIMER_PWM	1	MCU_TIMER3_CTRL	1	0	
MCU_Timer_4	timer_hclk_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6				167
MCU_Timer_4	timer_tclk_clk	HFOSC0	1	MCU_TIMER4_CLKSEL	0	4	[19.2, 20, 24, 25, 26, 27]
MCU_Timer_4	timer_tclk_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	4	MCU_TIMER4_CLKSEL	1	4	250
MCU_Timer_4	timer_tclk_clk	CLK_12M_RC	1	MCU_TIMER4_CLKSEL	2	4	13
MCU_Timer_4	timer_tclk_clk	MCU_PLL2.HSDIV2	1	MCU_TIMER4_CLKSEL	3	4	200
MCU_Timer_4	timer_tclk_clk	MCU_EXT_REFCLK0	1	MCU_TIMER4_CLKSEL	4	4	100
MCU_Timer_4	timer_tclk_clk	LFXOSC	1	MCU_TIMER4_CLKSEL	5	4	0.032768
MCU_Timer_4	timer_tclk_clk	MCU_CPSW_0.CPTS_GENF0	1	MCU_TIMER4_CLKSEL	6	4	
MCU_Timer_4	timer_tclk_clk	CLK_32K	1	MCU_TIMER4_CLKSEL	7	4	0.032000
MCU_Timer_5	timer_hclk_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6				167
MCU_Timer_5	timer_tclk_clk	HFOSC0	1	MCU_TIMER5_CLKSEL	0	5	[19.2, 20, 24, 25, 26, 27]
MCU_Timer_5	timer_tclk_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	4	MCU_TIMER5_CLKSEL	1	5	250
MCU_Timer_5	timer_tclk_clk	CLK_12M_RC	1	MCU_TIMER5_CLKSEL	2	5	13
MCU_Timer_5	timer_tclk_clk	MCU_PLL2.HSDIV2	1	MCU_TIMER5_CLKSEL	3	5	200
MCU_Timer_5	timer_tclk_clk	MCU_EXT_REFCLK0	1	MCU_TIMER5_CLKSEL	4	5	100
MCU_Timer_5	timer_tclk_clk	LFXOSC	1	MCU_TIMER5_CLKSEL	5	5	0.032768
MCU_Timer_5	timer_tclk_clk	MCU_CPSW_0.CPTS_GENF0	1	MCU_TIMER5_CLKSEL	6	5	
MCU_Timer_5	timer_tclk_clk	CLK_32K	1	MCU_TIMER5_CLKSEL	7	5	0.032000
MCU_Timer_5	timer_tclk_clk	HFOSC0	1	MCU_TIMER5_CTRL	0	0	[19.2, 20, 24, 25, 26, 27]

Copyright © 2024 Texas Instruments Incorporated

Table 5-33. Clock Mapping (continued)

Module	Clock Port	Clock	Divider	Control Register	Mux Value	Default Mux Value	Typical Frequency
MCU_Timer_5	timer_tclk_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	1	MCU_TIMER5_CTRL	0	0	1000
MCU_Timer_5	timer_tclk_clk	CLK_12M_RC	1	MCU_TIMER5_CTRL	0	0	13
MCU_Timer_5	timer_tclk_clk	MCU_PLL2.HSDIV2	1	MCU_TIMER5_CTRL	0	0	200
MCU_Timer_5	timer_tclk_clk	MCU_EXT_REFCLK0	1	MCU_TIMER5_CTRL	0	0	100
MCU_Timer_5	timer_tclk_clk	LFXOSC	1	MCU_TIMER5_CTRL	0	0	0.032768
MCU_Timer_5	timer_tclk_clk	MCU_CPSW_0.CPTS_GENF0	1	MCU_TIMER5_CTRL	0	0	
MCU_Timer_5	timer_tclk_clk	CLK_32K	1	MCU_TIMER5_CTRL	0	0	0.032000
MCU_Timer_5	timer_tclk_clk	MCU_TIMER_4.TIMER_PWM	1	MCU_TIMER5_CTRL	1	0	
MCU_Timer_6	timer_hclk_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6				167
MCU_Timer_6	timer_tclk_clk	HFOSC0	1	MCU_TIMER6_CLKSEL	0	6	[19.2, 20, 24, 25, 26, 27]
MCU_Timer_6	timer_tclk_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	4	MCU_TIMER6_CLKSEL	1	6	250
MCU_Timer_6	timer_tclk_clk	CLK_12M_RC	1	MCU_TIMER6_CLKSEL	2	6	13
MCU_Timer_6	timer_tclk_clk	MCU_PLL2.HSDIV2	1	MCU_TIMER6_CLKSEL	3	6	200
MCU_Timer_6	timer_tclk_clk	MCU_EXT_REFCLK0	1	MCU_TIMER6_CLKSEL	4	6	100
MCU_Timer_6	timer_tclk_clk	LFXOSC	1	MCU_TIMER6_CLKSEL	5	6	0.032768
MCU_Timer_6	timer_tclk_clk	MCU_CPSW_0.CPTS_GENF0	1	MCU_TIMER6_CLKSEL	6	6	
MCU_Timer_6	timer_tclk_clk	CLK_32K	1	MCU_TIMER6_CLKSEL	7	6	0.032000
MCU_Timer_7	timer_hclk_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6				167
MCU_Timer_7	timer_tclk_clk	HFOSC0	1	MCU_TIMER7_CLKSEL	0	7	[19.2, 20, 24, 25, 26, 27]
MCU_Timer_7	timer_tclk_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	4	MCU_TIMER7_CLKSEL	1	7	250
MCU_Timer_7	timer_tclk_clk	CLK_12M_RC	1	MCU_TIMER7_CLKSEL	2	7	13
MCU_Timer_7	timer_tclk_clk	MCU_PLL2.HSDIV2	1	MCU_TIMER7_CLKSEL	3	7	200
MCU_Timer_7	timer_tclk_clk	MCU_EXT_REFCLK0	1	MCU_TIMER7_CLKSEL	4	7	100
MCU_Timer_7	timer_tclk_clk	LFXOSC	1	MCU_TIMER7_CLKSEL	5	7	0.032768
MCU_Timer_7	timer_tclk_clk	MCU_CPSW_0.CPTS_GENF0	1	MCU_TIMER7_CLKSEL	6	7	
MCU_Timer_7	timer_tclk_clk	CLK_32K	1	MCU_TIMER7_CLKSEL	7	7	0.032000
MCU_Timer_7	timer_tclk_clk	HFOSC0	1	MCU_TIMER7_CTRL	0	0	[19.2, 20, 24, 25, 26, 27]
MCU_Timer_7	timer_tclk_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	1	MCU_TIMER7_CTRL	0	0	1000
MCU_Timer_7	timer_tclk_clk	CLK_12M_RC	1	MCU_TIMER7_CTRL	0	0	13
MCU_Timer_7	timer_tclk_clk	MCU_PLL2.HSDIV2	1	MCU_TIMER7_CTRL	0	0	200
MCU_Timer_7	timer_tclk_clk	MCU_EXT_REFCLK0	1	MCU_TIMER7_CTRL	0	0	100
MCU_Timer_7	timer_tclk_clk	LFXOSC	1	MCU_TIMER7_CTRL	0	0	0.032768
MCU_Timer_7	timer_tclk_clk	MCU_CPSW_0.CPTS_GENF0	1	MCU_TIMER7_CTRL	0	0	
MCU_Timer_7	timer_tclk_clk	CLK_32K	1	MCU_TIMER7_CTRL	0	0	0.032000
MCU_Timer_7	timer_tclk_clk	MCU_TIMER_6.TIMER_PWM	1	MCU_TIMER7_CTRL	1	0	
MCU_Timer_8	timer_hclk_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6				167
MCU_Timer_8	timer_tclk_clk	HFOSC0	1	MCU_TIMER8_CLKSEL	0	8	[19.2, 20, 24, 25, 26, 27]
MCU_Timer_8	timer_tclk_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	1	MCU_TIMER8_CLKSEL	1	8	1000
MCU_Timer_8	timer_tclk_clk	CLK_12M_RC	1	MCU_TIMER8_CLKSEL	2	8	13
MCU_Timer_8	timer_tclk_clk	MCU_PLL2.HSDIV2	1	MCU_TIMER8_CLKSEL	3	8	200
MCU_Timer_8	timer_tclk_clk	MCU_EXT_REFCLK0	1	MCU_TIMER8_CLKSEL	4	8	100
MCU_Timer_8	timer_tclk_clk	LFXOSC	1	MCU_TIMER8_CLKSEL	5	8	0.032768
MCU_Timer_8	timer_tclk_clk	MCU_CPSW_0.CPTS_GENF0	1	MCU_TIMER8_CLKSEL	6	8	
MCU_Timer_8	timer_tclk_clk	CLK_32K	1	MCU_TIMER8_CLKSEL	7	8	0.032000
MCU_Timer_8	timer_tclk_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	1				1000

Copyright © 2024 Texas Instruments Incorporated

Table 5-33. Clock Mapping (continued)

Module	Clock Port	Clock	Divider	Control Register	Mux Value	Default Mux Value	Typical Frequency
MCU_Timer_9	timer_hclk_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6				167
MCU_Timer_9	timer_tclk_clk	HFOSC0	1	MCU_TIMER9_CLKSEL	0	9	[19.2, 20, 24, 25, 26, 27]
MCU_Timer_9	timer_tclk_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	1	MCU_TIMER9_CLKSEL	1	9	1000
MCU_Timer_9	timer_tclk_clk	CLK_12M_RC	1	MCU_TIMER9_CLKSEL	2	9	13
MCU_Timer_9	timer_tclk_clk	MCU_PLL2.HSDIV2	1	MCU_TIMER9_CLKSEL	3	9	200
MCU_Timer_9	timer_tclk_clk	MCU_EXT_REFCLK0	1	MCU_TIMER9_CLKSEL	4	9	100
MCU_Timer_9	timer_tclk_clk	LFXOSC	1	MCU_TIMER9_CLKSEL	5	9	0.032768
MCU_Timer_9	timer_tclk_clk	MCU_CPSW_0.CPTS_GENF0	1	MCU_TIMER9_CLKSEL	6	9	
MCU_Timer_9	timer_tclk_clk	CLK_32K	1	MCU_TIMER9_CLKSEL	7	9	0.032000
MCU_Timer_9	timer_tclk_clk	HFOSC0	1	MCU_TIMER9_CTRL	0	0	[19.2, 20, 24, 25, 26, 27]
MCU_Timer_9	timer_tclk_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	1	MCU_TIMER9_CTRL	0	0	1000
MCU_Timer_9	timer_tclk_clk	CLK_12M_RC	1	MCU_TIMER9_CTRL	0	0	13
MCU_Timer_9	timer_tclk_clk	MCU_PLL2.HSDIV2	1	MCU_TIMER9_CTRL	0	0	200
MCU_Timer_9	timer_tclk_clk	MCU_EXT_REFCLK0	1	MCU_TIMER9_CTRL	0	0	100
MCU_Timer_9	timer_tclk_clk	LFXOSC	1	MCU_TIMER9_CTRL	0	0	0.032768
MCU_Timer_9	timer_tclk_clk	MCU_CPSW_0.CPTS_GENF0	1	MCU_TIMER9_CTRL	0	0	
MCU_Timer_9	timer_tclk_clk	CLK_32K	1	MCU_TIMER9_CTRL	0	0	0.032000
MCU_Timer_9	timer_tclk_clk	MCU_TIMER_8.TIMER_PWM	1	MCU_TIMER9_CTRL	1	0	
MCU_Timer_9	timer_tclk_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	1				1000
MCU_UART_0	fclk_clk	MCU_PLL1.HSDIV3	1	MCU_USART_CLKSEL	0	0	96
MCU_UART_0	fclk_clk	MAIN_PLL1.HSDIV5	1	MCU_USART_CLKSEL	1	0	
MCU_UART_0	sclki_clk	MCU_TIEOFF LOW	1				
MCU_UART_0	vbusp_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6				167
MCU_CPT2_Aggr_0	vclk_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	3				333
MCU_CPT2_Probe_0	probe_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	3				333
MCU_CPT2_Probe_0	vbus_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	3				333
MCU_CPT2_Probe_FSS_0_2	probe_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	3				333
MCU_CPT2_Probe_FSS_0_2	vbus_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	3				333
MCU_CPT2_Probe_FSS_1_3	probe_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	3				333
MCU_CPT2_Probe_FSS_1_3	vbus_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	3				333
MCU_CPT2_Probe_SRAM	probe_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	3				333
MCU_CPT2_Probe_SRAM	vbus_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	3				333
MCU_CTRL_MMR_0	vbusp_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6				167
MCU_ECC_Aggr_0	clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6				167
MCU_EFUSE_0	pll_ctrl_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	12				83
MCU_EFUSE_0	wkup_osc0_clk	HFOSC0	1	MCU_EFUSE_CLKSEL	0	0	[19.2, 20, 24, 25, 26, 27]
MCU_EFUSE_0	wkup_osc0_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	1	MCU_EFUSE_CLKSEL	1	0	1000
MCU_NAVSS_0	modss_vd2clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	3				333
MCU_NAVSS_0	navss_clk_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	3				333

Copyright © 2024 Texas Instruments Incorporated

Table 5-33. Clock Mapping (continued)

Module	Clock Port	Clock	Divider	Control Register	Mux Value	Default Mux Value	Typical Frequency
MCU_NAVSS_0	pdma_adc_clk_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	3				333
MCU_NAVSS_0	pdma_mcu0_clk_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6				167
MCU_NAVSS_0	pdma_mcu1_clk_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6				167
MCU_NAVSS_0	pdma_mcu2_clk_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6				167
MCU_NAVSS_0	udmass_vd2clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	3				333
MCU_PBIST_MCU_0	clk1_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	2				500
MCU_PBIST_MCU_0	clk2_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	3				333
MCU_PBIST_MCU_0	clk3_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6				167
MCU_PBIST_MCU_0	clk4_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	12				83
MCU_PBIST_MCU_0	clk5_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	12				83
MCU_PBIST_MCU_0	clk6_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	12				83
MCU_PBIST_MCU_0	clk7_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	12				83
MCU_PBIST_MCU_0	clk8_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	12				83
MCU_PBIST_MCU_0	tclk_clk	MAIN_PBIST_CLK	1				
MCU_PBIST_MCU_1	clk1_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	2				500
MCU_PBIST_MCU_1	clk2_clk	MCU_PLL1.HSDIV0	1				400
MCU_PBIST_MCU_1	clk3_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	3				333
MCU_PBIST_MCU_1	clk4_clk	MCU_PLL2.HSDIV4	2				167
MCU_PBIST_MCU_1	clk5_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6				167
MCU_PBIST_MCU_1	clk6_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	12				83
MCU_PBIST_MCU_1	clk7_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	12				83
MCU_PBIST_MCU_1	clk8_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	12				83
MCU_PBIST_MCU_1	tclk_clk	MAIN_PBIST_CLK	1				
MCU_PBIST_Pulsar_0	clk1_clk	MAIN_PBIST_CLK	1				
MCU_PBIST_Pulsar_0	clk2_clk	MAIN_PBIST_CLK	1				
MCU_PBIST_Pulsar_0	clk3_clk	MAIN_PBIST_CLK	1				
MCU_PBIST_Pulsar_0	clk4_clk	MAIN_PBIST_CLK	1				
MCU_PBIST_Pulsar_0	clk5_clk	MAIN_PBIST_CLK	1				
MCU_PBIST_Pulsar_0	clk6_clk	MAIN_PBIST_CLK	1				
MCU_PBIST_Pulsar_0	clk7_clk	MAIN_PBIST_CLK	1				
MCU_PBIST_Pulsar_0	clk8_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	12				83
MCU_PBIST_Pulsar_0	tclk_clk	MAIN_PBIST_CLK	1				
MCU_PLL_MMR_0	vbusp_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6				167
MCU_PSROM_0	clk_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	3				333
MCU_SEC_MMR_0	vbusp_clk	MCU_SYSCLK0 (MCU_PLL0.HSDIV0)	6				167
Main_1KByte_ScratchPadRAM	clk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_2KByte_ScratchPadRAM	clk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_512KByte_SRAM_0	cclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_512KByte_SRAM_0	vclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	1				500

Copyright © 2024 Texas Instruments Incorporated

Table 5-33. Clock Mapping (continued)

Module	Clock Port	Clock	Divider	Control Register	Mux Value	Default Mux Value	Typical Frequency
Main_512KByte_SRAM_1	cclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_512KByte_SRAM_1	vclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	1				500
Main_ATL_0	atl_clk	MAIN_PLL4.HSDIV1	1	ATL_PCLKMUX	0	0	294
Main_ATL_0	atl_clk	MAIN_PLL2.HSDIV2	1	ATL_PCLKMUX	1	0	200
Main_ATL_0	atl_clk	MAIN_PLL0.HSDIV7	1	ATL_PCLKMUX	4	0	200
Main_ATL_0	atl_clk	MCU_EXT_REFCLK0	1	ATL_PCLKMUX	5	0	100
Main_ATL_0	atl_clk	EXT_REFCLK1	1	ATL_PCLKMUX	6	0	
Main_ATL_0	atl_io_port_aws	MAIN_MCASP_0.MCASP_AFSX	1	ATL_AWS_SEL	0	0	
Main_ATL_0	atl_io_port_aws	MAIN_MCASP_0.MCASP_AFSX	1	ATL_AWS_SEL	0	1	
Main_ATL_0	atl_io_port_aws	MAIN_MCASP_0.MCASP_AFSX	1	ATL_AWS_SEL	0	2	
Main_ATL_0	atl_io_port_aws	MAIN_MCASP_0.MCASP_AFSX	1	ATL_AWS_SEL	0	3	
Main_ATL_0	atl_io_port_aws	MAIN_MCASP_1.MCASP_AFSX	1	ATL_AWS_SEL	1	0	
Main_ATL_0	atl_io_port_aws	MAIN_MCASP_1.MCASP_AFSX	1	ATL_AWS_SEL	1	1	
Main_ATL_0	atl_io_port_aws	MAIN_MCASP_1.MCASP_AFSX	1	ATL_AWS_SEL	1	2	
Main_ATL_0	atl_io_port_aws	MAIN_MCASP_1.MCASP_AFSX	1	ATL_AWS_SEL	1	3	
Main_ATL_0	atl_io_port_aws	MAIN_MCASP_2.MCASP_AFSX	1	ATL_AWS_SEL	2	0	
Main_ATL_0	atl_io_port_aws	MAIN_MCASP_2.MCASP_AFSX	1	ATL_AWS_SEL	2	1	
Main_ATL_0	atl_io_port_aws	MAIN_MCASP_2.MCASP_AFSX	1	ATL_AWS_SEL	2	2	
Main_ATL_0	atl_io_port_aws	MAIN_MCASP_2.MCASP_AFSX	1	ATL_AWS_SEL	2	3	
Main_ATL_0	atl_io_port_aws	MAIN_MCASP_3.MCASP_AFSX	1	ATL_AWS_SEL	3	0	
Main_ATL_0	atl_io_port_aws	MAIN_MCASP_3.MCASP_AFSX	1	ATL_AWS_SEL	3	1	
Main_ATL_0	atl_io_port_aws	MAIN_MCASP_3.MCASP_AFSX	1	ATL_AWS_SEL	3	2	
Main_ATL_0	atl_io_port_aws	MAIN_MCASP_3.MCASP_AFSX	1	ATL_AWS_SEL	3	3	
Main_ATL_0	atl_io_port_aws	MAIN_MCASP_4.MCASP_AFSX	1	ATL_AWS_SEL	4	0	
Main_ATL_0	atl_io_port_aws	MAIN_MCASP_4.MCASP_AFSX	1	ATL_AWS_SEL	4	1	
Main_ATL_0	atl_io_port_aws	MAIN_MCASP_4.MCASP_AFSX	1	ATL_AWS_SEL	4	2	
Main_ATL_0	atl_io_port_aws	MAIN_MCASP_4.MCASP_AFSX	1	ATL_AWS_SEL	4	3	
Main_ATL_0	atl_io_port_aws	MAIN_MCASP_0.MCASP_AFSX	1	ATL_AWS_SEL	12	0	
Main_ATL_0	atl_io_port_aws	MAIN_MCASP_0.MCASP_AFSX	1	ATL_AWS_SEL	12	1	
Main_ATL_0	atl_io_port_aws	MAIN_MCASP_0.MCASP_AFSX	1	ATL_AWS_SEL	12	2	
Main_ATL_0	atl_io_port_aws	MAIN_MCASP_0.MCASP_AFSX	1	ATL_AWS_SEL	12	3	
Main_ATL_0	atl_io_port_aws	MAIN_MCASP_1.MCASP_AFSX	1	ATL_AWS_SEL	13	0	
Main_ATL_0	atl_io_port_aws	MAIN_MCASP_1.MCASP_AFSX	1	ATL_AWS_SEL	13	1	
Main_ATL_0	atl_io_port_aws	MAIN_MCASP_1.MCASP_AFSX	1	ATL_AWS_SEL	13	2	
Main_ATL_0	atl_io_port_aws	MAIN_MCASP_1.MCASP_AFSX	1	ATL_AWS_SEL	13	3	
Main_ATL_0	atl_io_port_aws	MAIN_MCASP_2.MCASP_AFSX	1	ATL_AWS_SEL	14	0	
Main_ATL_0	atl_io_port_aws	MAIN_MCASP_2.MCASP_AFSX	1	ATL_AWS_SEL	14	1	
Main_ATL_0	atl_io_port_aws	MAIN_MCASP_2.MCASP_AFSX	1	ATL_AWS_SEL	14	2	
Main_ATL_0	atl_io_port_aws	MAIN_MCASP_2.MCASP_AFSX	1	ATL_AWS_SEL	14	3	
Main_ATL_0	atl_io_port_aws	MAIN_MCASP_3.MCASP_AFSX	1	ATL_AWS_SEL	15	0	
Main_ATL_0	atl_io_port_aws	MAIN_MCASP_3.MCASP_AFSX	1	ATL_AWS_SEL	15	1	
Main_ATL_0	atl_io_port_aws	MAIN_MCASP_3.MCASP_AFSX	1	ATL_AWS_SEL	15	2	
Main_ATL_0	atl_io_port_aws	MAIN_MCASP_3.MCASP_AFSX	1	ATL_AWS_SEL	15	3	
Main_ATL_0	atl_io_port_aws	MAIN_MCASP_4.MCASP_AFSX	1	ATL_AWS_SEL	16	0	
Main_ATL_0	atl_io_port_aws	MAIN_MCASP_4.MCASP_AFSX	1	ATL_AWS_SEL	16	1	
Main_ATL_0	atl_io_port_aws	MAIN_MCASP_4.MCASP_AFSX	1	ATL_AWS_SEL	16	2	
Main_ATL_0	atl_io_port_aws	MAIN_MCASP_4.MCASP_AFSX	1	ATL_AWS_SEL	16	3	
Main_ATL_0	atl_io_port_aws	AUDIO_EXT_REFCLK0	1	ATL_AWS_SEL	24	0	
Main_ATL_0	atl_io_port_aws	AUDIO_EXT_REFCLK0	1	ATL_AWS_SEL	24	1	
Main_ATL_0	atl_io_port_aws	AUDIO_EXT_REFCLK0	1	ATL_AWS_SEL	24	2	
Main_ATL_0	atl_io_port_aws	AUDIO_EXT_REFCLK0	1	ATL_AWS_SEL	24	3	
Main_ATL_0	atl_io_port_aws	AUDIO_EXT_REFCLK1	1	ATL_AWS_SEL	25	0	
Main_ATL_0	atl_io_port_aws	AUDIO_EXT_REFCLK1	1	ATL_AWS_SEL	25	1	
Main_ATL_0	atl_io_port_aws	AUDIO_EXT_REFCLK1	1	ATL_AWS_SEL	25	2	
Main_ATL_0	atl_io_port_aws	AUDIO_EXT_REFCLK1	1	ATL_AWS_SEL	25	3	

Copyright © 2024 Texas Instruments Incorporated

Table 5-33. Clock Mapping (continued)

Module	Clock Port	Clock	Divider	Control Register	Mux Value	Default Mux Value	Typical Frequency
Main_ATL_0	atl_io_port_aws	MAIN_TIEOFF LOW	1				
Main_ATL_0	atl_io_port_bws	MAIN_MCASP_0.MCASP_AFSR	1	ATL_BWS_SEL	0	0	
Main_ATL_0	atl_io_port_bws	MAIN_MCASP_0.MCASP_AFSR	1	ATL_BWS_SEL	0	1	
Main_ATL_0	atl_io_port_bws	MAIN_MCASP_0.MCASP_AFSR	1	ATL_BWS_SEL	0	2	
Main_ATL_0	atl_io_port_bws	MAIN_MCASP_0.MCASP_AFSR	1	ATL_BWS_SEL	0	3	
Main_ATL_0	atl_io_port_bws	MAIN_MCASP_1.MCASP_AFSR	1	ATL_BWS_SEL	1	0	
Main_ATL_0	atl_io_port_bws	MAIN_MCASP_1.MCASP_AFSR	1	ATL_BWS_SEL	1	1	
Main_ATL_0	atl_io_port_bws	MAIN_MCASP_1.MCASP_AFSR	1	ATL_BWS_SEL	1	2	
Main_ATL_0	atl_io_port_bws	MAIN_MCASP_1.MCASP_AFSR	1	ATL_BWS_SEL	1	3	
Main_ATL_0	atl_io_port_bws	MAIN_MCASP_2.MCASP_AFSR	1	ATL_BWS_SEL	2	0	
Main_ATL_0	atl_io_port_bws	MAIN_MCASP_2.MCASP_AFSR	1	ATL_BWS_SEL	2	1	
Main_ATL_0	atl_io_port_bws	MAIN_MCASP_2.MCASP_AFSR	1	ATL_BWS_SEL	2	2	
Main_ATL_0	atl_io_port_bws	MAIN_MCASP_2.MCASP_AFSR	1	ATL_BWS_SEL	2	3	
Main_ATL_0	atl_io_port_bws	MAIN_MCASP_3.MCASP_AFSR	1	ATL_BWS_SEL	3	0	
Main_ATL_0	atl_io_port_bws	MAIN_MCASP_3.MCASP_AFSR	1	ATL_BWS_SEL	3	1	
Main_ATL_0	atl_io_port_bws	MAIN_MCASP_3.MCASP_AFSR	1	ATL_BWS_SEL	3	2	
Main_ATL_0	atl_io_port_bws	MAIN_MCASP_3.MCASP_AFSR	1	ATL_BWS_SEL	3	3	
Main_ATL_0	atl_io_port_bws	MAIN_MCASP_4.MCASP_AFSR	1	ATL_BWS_SEL	4	0	
Main_ATL_0	atl_io_port_bws	MAIN_MCASP_4.MCASP_AFSR	1	ATL_BWS_SEL	4	1	
Main_ATL_0	atl_io_port_bws	MAIN_MCASP_4.MCASP_AFSR	1	ATL_BWS_SEL	4	2	
Main_ATL_0	atl_io_port_bws	MAIN_MCASP_4.MCASP_AFSR	1	ATL_BWS_SEL	4	3	
Main_ATL_0	atl_io_port_bws	MAIN_MCASP_0.MCASP_AFSX	1	ATL_BWS_SEL	12	0	
Main_ATL_0	atl_io_port_bws	MAIN_MCASP_0.MCASP_AFSX	1	ATL_BWS_SEL	12	1	
Main_ATL_0	atl_io_port_bws	MAIN_MCASP_0.MCASP_AFSX	1	ATL_BWS_SEL	12	2	
Main_ATL_0	atl_io_port_bws	MAIN_MCASP_0.MCASP_AFSX	1	ATL_BWS_SEL	12	3	
Main_ATL_0	atl_io_port_bws	MAIN_MCASP_1.MCASP_AFSX	1	ATL_BWS_SEL	13	0	
Main_ATL_0	atl_io_port_bws	MAIN_MCASP_1.MCASP_AFSX	1	ATL_BWS_SEL	13	1	
Main_ATL_0	atl_io_port_bws	MAIN_MCASP_1.MCASP_AFSX	1	ATL_BWS_SEL	13	2	
Main_ATL_0	atl_io_port_bws	MAIN_MCASP_1.MCASP_AFSX	1	ATL_BWS_SEL	13	3	
Main_ATL_0	atl_io_port_bws	MAIN_MCASP_2.MCASP_AFSX	1	ATL_BWS_SEL	14	0	
Main_ATL_0	atl_io_port_bws	MAIN_MCASP_2.MCASP_AFSX	1	ATL_BWS_SEL	14	1	
Main_ATL_0	atl_io_port_bws	MAIN_MCASP_2.MCASP_AFSX	1	ATL_BWS_SEL	14	2	
Main_ATL_0	atl_io_port_bws	MAIN_MCASP_2.MCASP_AFSX	1	ATL_BWS_SEL	14	3	
Main_ATL_0	atl_io_port_bws	MAIN_MCASP_3.MCASP_AFSX	1	ATL_BWS_SEL	15	0	
Main_ATL_0	atl_io_port_bws	MAIN_MCASP_3.MCASP_AFSX	1	ATL_BWS_SEL	15	1	
Main_ATL_0	atl_io_port_bws	MAIN_MCASP_3.MCASP_AFSX	1	ATL_BWS_SEL	15	2	
Main_ATL_0	atl_io_port_bws	MAIN_MCASP_3.MCASP_AFSX	1	ATL_BWS_SEL	15	3	
Main_ATL_0	atl_io_port_bws	MAIN_MCASP_4.MCASP_AFSX	1	ATL_BWS_SEL	16	0	
Main_ATL_0	atl_io_port_bws	MAIN_MCASP_4.MCASP_AFSX	1	ATL_BWS_SEL	16	1	
Main_ATL_0	atl_io_port_bws	MAIN_MCASP_4.MCASP_AFSX	1	ATL_BWS_SEL	16	2	
Main_ATL_0	atl_io_port_bws	MAIN_MCASP_4.MCASP_AFSX	1	ATL_BWS_SEL	16	3	
Main_ATL_0	atl_io_port_bws	AUDIO_EXT_REFCLK0	1	ATL_BWS_SEL	24	0	
Main_ATL_0	atl_io_port_bws	AUDIO_EXT_REFCLK0	1	ATL_BWS_SEL	24	1	
Main_ATL_0	atl_io_port_bws	AUDIO_EXT_REFCLK0	1	ATL_BWS_SEL	24	2	
Main_ATL_0	atl_io_port_bws	AUDIO_EXT_REFCLK0	1	ATL_BWS_SEL	24	3	
Main_ATL_0	atl_io_port_bws	AUDIO_EXT_REFCLK1	1	ATL_BWS_SEL	25	0	
Main_ATL_0	atl_io_port_bws	AUDIO_EXT_REFCLK1	1	ATL_BWS_SEL	25	1	
Main_ATL_0	atl_io_port_bws	AUDIO_EXT_REFCLK1	1	ATL_BWS_SEL	25	2	
Main_ATL_0	atl_io_port_bws	AUDIO_EXT_REFCLK1	1	ATL_BWS_SEL	25	3	
Main_ATL_0	atl_io_port_bws	MAIN_TIEOFF LOW	1				
Main_ATL_0	vbus_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	2				250
Main_Compute_Cluster_0	ac71_4_clk_clk	MAIN_PLL7.HSDIV0	1				1000
Main_Compute_Cluster_0	ac71_5_clk_clk	MAIN_PLL7.HSDIV0	1				1000
Main_Compute_Cluster_0	arm0_clk_clk	MAIN_PLL8.HSDIV0	1				2000
Main_Compute_Cluster_0	arm0_msmc_clk_clk	MAIN_PLL7.HSDIV0	1				1000

Copyright © 2024 Texas Instruments Incorporated

Table 5-33. Clock Mapping (continued)

Module	Clock Port	Clock	Divider	Control Register	Mux Value	Default Mux Value	Typical Frequency
Main_Compute_Cluster_0	arm0_pll_ctrl_clk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	1				500
Main_Compute_Cluster_0	msmc_clk	MAIN_PLL7.HSDIV0	1				1000
Main_Compute_Cluster_0	msmc_pll_ctrl_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	1				500
Main_CPSW_0	cppi_clk_clk	MAIN_PLL1.HSDIV1	1				320
Main_CPSW_0	cpts_rft_clk	MAIN_PLL3.HSDIV1	1	MAIN_CPSW2_CPTS_CLK_MUX	0	0	250
Main_CPSW_0	cpts_rft_clk	MAIN_PLL0.HSDIV6	1	MAIN_CPSW2_CPTS_CLK_MUX	1	0	250
Main_CPSW_0	cpts_rft_clk	MCU_CPTS0_RFT_CLK	1	MAIN_CPSW2_CPTS_CLK_MUX	2	0	
Main_CPSW_0	cpts_rft_clk	CPTS0_RFT_CLK	1	MAIN_CPSW2_CPTS_CLK_MUX	3	0	
Main_CPSW_0	cpts_rft_clk	MCU_EXT_REFCLK0	1	MAIN_CPSW2_CPTS_CLK_MUX	4	0	100
Main_CPSW_0	cpts_rft_clk	EXT_REFCLK1	1	MAIN_CPSW2_CPTS_CLK_MUX	5	0	
Main_CPSW_0	cpts_rft_clk	MAIN_SERDES_0.IP2_LN0_TXMCLK	1	MAIN_CPSW2_CPTS_CLK_MUX	6	0	
Main_CPSW_0	cpts_rft_clk	MAIN_SERDES_0.IP2_LN1_TXMCLK	1	MAIN_CPSW2_CPTS_CLK_MUX	7	0	
Main_CPSW_0	cpts_rft_clk	MAIN_SERDES_0.IP2_LN2_TXMCLK	1	MAIN_CPSW2_CPTS_CLK_MUX	8	0	
Main_CPSW_0	cpts_rft_clk	MAIN_SERDES_0.IP2_LN3_TXMCLK	1	MAIN_CPSW2_CPTS_CLK_MUX	9	0	
Main_CPSW_0	cpts_rft_clk	MCU_PLL2.HSDIV1	1	MAIN_CPSW2_CPTS_CLK_MUX	14	0	500
Main_CPSW_0	cpts_rft_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	1	MAIN_CPSW2_CPTS_CLK_MUX	15	0	500
Main_CPSW_0	gmii_rft_clk	MAIN_PLL3.HSDIV0	2				125
Main_CPSW_0	gmii1_mr_clk	MAIN_PLL3.HSDIV0	10				25
Main_CPSW_0	gmii1_mt_clk	MAIN_PLL3.HSDIV0	10				25
Main_CPSW_0	rgmii_mhz_250_clk	MAIN_PLL3.HSDIV0	1				250
Main_CPSW_0	rgmii_mhz_5_clk	MAIN_PLL3.HSDIV0	50				5
Main_CPSW_0	rgmii_mhz_50_clk	MAIN_PLL3.HSDIV0	5				50
Main_CPSW_0	rgmii1_rxc_i	RGMII1_RXC	1				
Main_CPSW_0	rgmii1_txc_i	MAIN_TIEOFF LOW	1				
Main_CPSW_0	rmii_mhz_50_clk	RMII_REF_CLK	1				
Main_CPSW_0	sgmii1_rxb_clk	MAIN_TIEOFF LOW	1				
Main_CPSW_0	sgmii1_txb_clk	MAIN_TIEOFF LOW	1				
Main_CSI_Rx_0	main_clk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	1				500
Main_CSI_Rx_0	ppi_rx_byte_clk	MAIN_DPHY_RX_0.PPI_RX_BYTE_CLK	1				
Main_CSI_Rx_0	vbus_clk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	2				250
Main_CSI_Rx_0	vp_clk_clk	MAIN_PLL25.HSDIV1	1				720
Main_CSI_Rx_1	main_clk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	1				500
Main_CSI_Rx_1	ppi_rx_byte_clk	MAIN_DPHY_RX_1.PPI_RX_BYTE_CLK	1				
Main_CSI_Rx_1	vbus_clk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	2				250
Main_CSI_Rx_1	vp_clk_clk	MAIN_PLL25.HSDIV1	1				720
Main_CSI_Tx_0	dphy_TxByteClkHS_cl_clk	MAIN_SERDES_0.IP2_PPI_TXBYTECLKHS_CL_CLK	1				
Main_CSI_Tx_0	esc_clk_clk	MAIN_PLL1.HSDIV8	1				20
Main_CSI_Tx_0	main_clk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	1				500
Main_CSI_Tx_0	vbus_clk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	2				250
Main_CSI_Tx_1	dphy_TxByteClkHS_cl_clk	MAIN_SERDES_1.IP2_PPI_TXBYTECLKHS_CL_CLK	1				
Main_CSI_Tx_1	esc_clk_clk	MAIN_PLL1.HSDIV8	1				20

Copyright © 2024 Texas Instruments Incorporated

Table 5-33. Clock Mapping (continued)

Module	Clock Port	Clock	Divider	Control Register	Mux Value	Default Mux Value	Typical Frequency
Main_CSI_Tx_1	main_clk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	1				500
Main_CSI_Tx_1	vbus_clk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	2				250
Main_DCC_0	dcc_input10_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	2	DCC_CLKSRC1	0	0	250
Main_DCC_0	dcc_clksrc0_clk	MAIN_PLL0.HSDIV8	1	DCC_CLKSRC1	1	0	250
Main_DCC_0	dcc_clksrc1_clk	MAIN_PLL0.HSDIV2	1	DCC_CLKSRC1	2	0	200
Main_DCC_0	dcc_clksrc2_clk	MAIN_PLL0.HSDIV3	1	DCC_CLKSRC1	3	0	133
Main_DCC_0	dcc_clksrc3_clk	MAIN_PLL0.HSDIV4	1	DCC_CLKSRC1	4	0	80
Main_DCC_0	dcc_clksrc4_clk	HFOSC0	1	DCC_CLKSRC1	5	0	[19.2, 20, 24, 25, 26, 27]
Main_DCC_0	dcc_clksrc5_clk	HFOSC1	1	DCC_CLKSRC1	6	0	[19.2 - 27]
Main_DCC_0	dcc_clksrc6_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	1	DCC_CLKSRC1	7	0	500
Main_DCC_0	dcc_clksrc7_clk	MAIN_DSS_EDP_0.PHY_LN0_TXCLK	1	DCC_CLKSRC1	8	0	
Main_DCC_0	dcc_input00_clk	HFOSC0	1	DCC_CLKSRC0	0	0	[19.2, 20, 24, 25, 26, 27]
Main_DCC_0	dcc_input01_clk	HFOSC1	1	DCC_CLKSRC0	1	0	[19.2 - 27]
Main_DCC_0	dcc_input02_clk	CLK_12M_RC	1	DCC_CLKSRC0	2	0	13
Main_DCC_0	vbus_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_DCC_1	dcc_input10_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4	DCC_CLKSRC1	0	0	125
Main_DCC_1	dcc_clksrc0_clk	MAIN_PLL0.HSDIV5	1	DCC_CLKSRC1	1	0	50
Main_DCC_1	dcc_clksrc1_clk	MAIN_PLL0.HSDIV6	1	DCC_CLKSRC1	2	0	250
Main_DCC_1	dcc_clksrc2_clk	MAIN_PLL0.HSDIV7	1	DCC_CLKSRC1	3	0	200
Main_DCC_1	dcc_clksrc3_clk	MAIN_PLL1.HSDIV0	1	DCC_CLKSRC1	4	0	192
Main_DCC_1	dcc_clksrc4_clk	MAIN_PLL1.HSDIV1	1	DCC_CLKSRC1	5	0	320
Main_DCC_1	dcc_clksrc5_clk	MAIN_PLL1.HSDIV2	1	DCC_CLKSRC1	6	0	192
Main_DCC_1	dcc_clksrc6_clk	MAIN_PLL1.HSDIV3	1	DCC_CLKSRC1	7	0	192
Main_DCC_1	dcc_clksrc7_clk	MAIN_PLL1.HSDIV6	1	DCC_CLKSRC1	8	0	19
Main_DCC_1	dcc_input00_clk	HFOSC0	1	DCC_CLKSRC0	0	0	[19.2, 20, 24, 25, 26, 27]
Main_DCC_1	dcc_input01_clk	HFOSC1	1	DCC_CLKSRC0	1	0	[19.2 - 27]
Main_DCC_1	dcc_input02_clk	CLK_12M_RC	1	DCC_CLKSRC0	2	0	13
Main_DCC_1	vbus_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_DCC_2	dcc_input10_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4	DCC_CLKSRC1	0	0	125
Main_DCC_2	dcc_clksrc0_clk	MAIN_PLL2.HSDIV7	1	DCC_CLKSRC1	1	0	120
Main_DCC_2	dcc_clksrc1_clk	MAIN_PLL1.HSDIV8	1	DCC_CLKSRC1	2	0	20
Main_DCC_2	dcc_clksrc2_clk	MAIN_TIEOFF LOW	1	DCC_CLKSRC1	3	0	
Main_DCC_2	dcc_clksrc3_clk	MAIN_PLL2.HSDIV4	1	DCC_CLKSRC1	4	0	100
Main_DCC_2	dcc_clksrc4_clk	MAIN_PLL2.HSDIV6	1	DCC_CLKSRC1	5	0	225
Main_DCC_2	dcc_clksrc5_clk	MAIN_PLL2.HSDIV1	2	DCC_CLKSRC1	6	0	300
Main_DCC_2	dcc_clksrc6_clk	MAIN_PLL2.HSDIV2	1	DCC_CLKSRC1	7	0	200
Main_DCC_2	dcc_clksrc7_clk	MAIN_PLL3.HSDIV0	1	DCC_CLKSRC1	8	0	250
Main_DCC_2	dcc_input00_clk	HFOSC0	1	DCC_CLKSRC0	0	0	[19.2, 20, 24, 25, 26, 27]
Main_DCC_2	dcc_input01_clk	HFOSC1	1	DCC_CLKSRC0	1	0	[19.2 - 27]
Main_DCC_2	dcc_input02_clk	CLK_12M_RC	1	DCC_CLKSRC0	2	0	13
Main_DCC_2	vbus_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_DCC_3	dcc_input10_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4	DCC_CLKSRC1	0	0	125
Main_DCC_3	dcc_clksrc0_clk	MAIN_PLL4.HSDIV0	1	DCC_CLKSRC1	1	0	197
Main_DCC_3	dcc_clksrc1_clk	MAIN_PLL5.HSDIV0	2	DCC_CLKSRC1	2	0	344

Copyright © 2024 Texas Instruments Incorporated

Table 5-33. Clock Mapping (continued)

Module	Clock Port	Clock	Divider	Control Register	Mux Value	Default Mux Value	Typical Frequency
Main_DCC_3	dcc_clksrc2_clk	MAIN_PLL5.HSDIV1	2	DCC_CLKSRC1	3	0	275
Main_DCC_3	dcc_clksrc3_clk	MAIN_PLL9.HSDIV0	4	DCC_CLKSRC1	4	0	500
Main_DCC_3	dcc_clksrc4_clk	MAIN_TIEOFF LOW	1	DCC_CLKSRC1	5	0	
Main_DCC_3	dcc_clksrc5_clk	MAIN_PLL6.HSDIV0	4	DCC_CLKSRC1	6	0	200
Main_DCC_3	dcc_clksrc6_clk	MAIN_PLL7.HSDIV0	4	DCC_CLKSRC1	7	0	250
Main_DCC_3	dcc_clksrc7_clk	MAIN_PLL8.HSDIV0	4	DCC_CLKSRC1	8	0	500
Main_DCC_3	dcc_input00_clk	HFOSC0	1	DCC_CLKSRC0	0	0	[19.2, 20, 24, 25, 26, 27]
Main_DCC_3	dcc_input01_clk	HFOSC1	1	DCC_CLKSRC0	1	0	[19.2 - 27]
Main_DCC_3	dcc_input02_clk	CLK_12M_RC	1	DCC_CLKSRC0	2	0	13
Main_DCC_3	vbus_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_DCC_4	dcc_input10_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	2	DCC_CLKSRC1	0	0	250
Main_DCC_4	dcc_clksrc0_clk	MAIN_PLL0.HSDIV1	2	DCC_CLKSRC1	1	0	200
Main_DCC_4	dcc_clksrc1_clk	MAIN_TIEOFF LOW	1	DCC_CLKSRC1	2	0	
Main_DCC_4	dcc_clksrc2_clk	MAIN_PLL12.HSDIV0	4	DCC_CLKSRC1	3	0	267
Main_DCC_4	dcc_clksrc3_clk	MAIN_PLL26.HSDIV0	4	DCC_CLKSRC1	4	0	267
Main_DCC_4	dcc_clksrc4_clk	MAIN_PLL14.HSDIV0	4	DCC_CLKSRC1	5	0	250
Main_DCC_4	dcc_clksrc5_clk	MAIN_PLL14.HSDIV1	4	DCC_CLKSRC1	6	0	250
Main_DCC_4	dcc_clksrc6_clk	MAIN_PLL27.HSDIV0	4	DCC_CLKSRC1	7	0	267
Main_DCC_4	dcc_clksrc7_clk	MAIN_PLL16.HSDIV0	2	DCC_CLKSRC1	8	0	300
Main_DCC_4	dcc_input00_clk	HFOSC0	1	DCC_CLKSRC0	0	0	[19.2, 20, 24, 25, 26, 27]
Main_DCC_4	dcc_input01_clk	HFOSC1	1	DCC_CLKSRC0	1	0	[19.2 - 27]
Main_DCC_4	dcc_input02_clk	CLK_12M_RC	1	DCC_CLKSRC0	2	0	13
Main_DCC_4	vbus_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_DCC_5	dcc_input10_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	1	DCC_CLKSRC1	0	0	500
Main_DCC_5	dcc_clksrc0_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	1	DCC_CLKSRC1	1	0	500
Main_DCC_5	dcc_clksrc1_clk	MAIN_PLL19.HSDIV0	2	DCC_CLKSRC1	2	0	300
Main_DCC_5	dcc_clksrc2_clk	MAIN_PLL17.HSDIV0	2	DCC_CLKSRC1	3	0	300
Main_DCC_5	dcc_clksrc3_clk	MAIN_PLL25.HSDIV0	1	DCC_CLKSRC1	4	0	520
Main_DCC_5	dcc_clksrc4_clk	MAIN_PLL25.HSDIV1	1	DCC_CLKSRC1	5	0	720
Main_DCC_5	dcc_clksrc5_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	1	DCC_CLKSRC1	6	0	500
Main_DCC_5	dcc_clksrc6_clk	GPMC0_CLK	1	DCC_CLKSRC1	7	0	
Main_DCC_5	dcc_clksrc7_clk	MAIN_MCASP_4.MCASP_AHCLKX	1	DCC_CLKSRC1	8	0	
Main_DCC_5	dcc_input00_clk	HFOSC0	1	DCC_CLKSRC0	0	0	[19.2, 20, 24, 25, 26, 27]
Main_DCC_5	dcc_input01_clk	HFOSC1	1	DCC_CLKSRC0	1	0	[19.2 - 27]
Main_DCC_5	dcc_input02_clk	CLK_12M_RC	1	DCC_CLKSRC0	2	0	13
Main_DCC_5	vbus_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_DCC_6	dcc_input10_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4	DCC_CLKSRC1	0	0	125
Main_DCC_6	dcc_clksrc0_clk	MCASP0_ACLKX	1	DCC_CLKSRC1	1	0	
Main_DCC_6	dcc_clksrc1_clk	MCASP0_ACLKR	1	DCC_CLKSRC1	2	0	
Main_DCC_6	dcc_clksrc2_clk	MCASP1_ACLKX	1	DCC_CLKSRC1	3	0	
Main_DCC_6	dcc_clksrc3_clk	MCASP1_ACLKR	1	DCC_CLKSRC1	4	0	
Main_DCC_6	dcc_clksrc4_clk	MCASP2_ACLKX	1	DCC_CLKSRC1	5	0	
Main_DCC_6	dcc_clksrc5_clk	MCASP2_ACLKR	1	DCC_CLKSRC1	6	0	
Main_DCC_6	dcc_clksrc6_clk	MCASP3_ACLKX	1	DCC_CLKSRC1	7	0	
Main_DCC_6	dcc_clksrc7_clk	MCASP3_ACLKR	1	DCC_CLKSRC1	8	0	
Main_DCC_6	dcc_input00_clk	HFOSC0	1	DCC_CLKSRC0	0	0	[19.2, 20, 24, 25, 26, 27]

Copyright © 2024 Texas Instruments Incorporated

Table 5-33. Clock Mapping (continued)

Module	Clock Port	Clock	Divider	Control Register	Mux Value	Default Mux Value	Typical Frequency
Main_DCC_6	dcc_input01_clk	HFOSC1	1	DCC_CLKSRC0	1	0	[19.2 - 27]
Main_DCC_6	dcc_input02_clk	CLK_12M_RC	1	DCC_CLKSRC0	2	0	13
Main_DCC_6	vbus_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_DCC_7	dcc_input10_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	2	DCC_CLKSRC1	0	0	250
Main_DCC_7	dcc_clksrc0_clk	AUDIO_EXT_REFCLK0	1	DCC_CLKSRC1	1	0	
Main_DCC_7	dcc_clksrc1_clk	AUDIO_EXT_REFCLK1	1	DCC_CLKSRC1	2	0	
Main_DCC_7	dcc_clksrc2_clk	MAIN_PLL4.HSDIV2	4	DCC_CLKSRC1	3	0	49
Main_DCC_7	dcc_clksrc3_clk	MAIN_TIEOFF LOW	1	DCC_CLKSRC1	4	0	
Main_DCC_7	dcc_clksrc4_clk	MAIN_TIEOFF LOW	1	DCC_CLKSRC1	5	0	
Main_DCC_7	dcc_clksrc5_clk	VOUT0_EXTPCLKIN	1	DCC_CLKSRC1	6	0	
Main_DCC_7	dcc_clksrc6_clk	MAIN_PLL2.HSDIV7	1	DCC_CLKSRC1	7	0	120
Main_DCC_7	dcc_clksrc7_clk	CPTS0_RFT_CLK	1	DCC_CLKSRC1	8	0	
Main_DCC_7	dcc_input00_clk	HFOSC0	1	DCC_CLKSRC0	0	0	[19.2, 20, 24, 25, 26, 27]
Main_DCC_7	dcc_input01_clk	HFOSC1	1	DCC_CLKSRC0	1	0	[19.2 - 27]
Main_DCC_7	dcc_input02_clk	CLK_12M_RC	1	DCC_CLKSRC0	2	0	13
Main_DCC_7	vbus_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_DCC_8	dcc_input10_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	2	DCC_CLKSRC1	0	0	250
Main_DCC_8	dcc_clksrc0_clk	CLK_32K	1	DCC_CLKSRC1	1	0	0.032000
Main_DCC_8	dcc_clksrc1_clk	LFXOSC	1	DCC_CLKSRC1	2	0	0.032768
Main_DCC_8	dcc_clksrc2_clk	MCU_EXT_REFCLK0	1	DCC_CLKSRC1	3	0	100
Main_DCC_8	dcc_clksrc3_clk	MAIN_DPHY_RX_0.PPI_RX_BYTE_CLK	1	DCC_CLKSRC1	4	0	
Main_DCC_8	dcc_clksrc4_clk	MAIN_DPHY_RX_1.PPI_RX_BYTE_CLK	1	DCC_CLKSRC1	5	0	
Main_DCC_8	dcc_clksrc5_clk	MAIN_TIEOFF LOW	1	DCC_CLKSRC1	6	0	
Main_DCC_8	dcc_clksrc6_clk	MAIN_PLL3.HSDIV1	1	DCC_CLKSRC1	7	0	250
Main_DCC_8	dcc_clksrc7_clk	MAIN_PLL3.HSDIV2	1	DCC_CLKSRC1	8	0	200
Main_DCC_8	dcc_input00_clk	HFOSC0	1	DCC_CLKSRC0	0	0	[19.2, 20, 24, 25, 26, 27]
Main_DCC_8	dcc_input01_clk	HFOSC1	1	DCC_CLKSRC0	1	0	[19.2 - 27]
Main_DCC_8	dcc_input02_clk	CLK_12M_RC	1	DCC_CLKSRC0	2	0	13
Main_DCC_8	vbus_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_DCC_9	dcc_input10_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	2	DCC_CLKSRC1	0	0	250
Main_DCC_9	dcc_clksrc0_clk	MAIN_PLL3.HSDIV3	1	DCC_CLKSRC1	1	0	250
Main_DCC_9	dcc_clksrc1_clk	MAIN_PLL3.HSDIV4	1	DCC_CLKSRC1	2	0	156
Main_DCC_9	dcc_clksrc2_clk	RMII_REF_CLK	1	DCC_CLKSRC1	3	0	
Main_DCC_9	dcc_clksrc3_clk	MAIN_MCASP_4.MCASP_AHCLKR	1	DCC_CLKSRC1	4	0	
Main_DCC_9	dcc_clksrc4_clk	RGMII1_RXC	1	DCC_CLKSRC1	5	0	
Main_DCC_9	dcc_clksrc5_clk	MAIN_PLL4.HSDIV1	1	DCC_CLKSRC1	6	0	294
Main_DCC_9	dcc_clksrc6_clk	MAIN_PLL4.HSDIV2	1	DCC_CLKSRC1	7	0	197
Main_DCC_9	dcc_clksrc7_clk	MAIN_TIEOFF LOW	1	DCC_CLKSRC1	8	0	
Main_DCC_9	dcc_input00_clk	HFOSC0	1	DCC_CLKSRC0	0	0	[19.2, 20, 24, 25, 26, 27]
Main_DCC_9	dcc_input01_clk	HFOSC1	1	DCC_CLKSRC0	1	0	[19.2 - 27]
Main_DCC_9	dcc_input02_clk	CLK_12M_RC	1	DCC_CLKSRC0	2	0	13
Main_DCC_9	vbus_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_DDR_EW_wrap_0	ddrss_cfg_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_DDR_EW_wrap_0	ddrss_ddr_pll_clk	MAIN_PLL12.HSDIV0	1				1067
Main_DDR_EW_wrap_0	ddrss_io_ck	DDR0_CKP	1				
Main_DDR_EW_wrap_0	ddrss_io_ck_n	DDR0_CKN	1				

Copyright © 2024 Texas Instruments Incorporated

Table 5-33. Clock Mapping (continued)

Module	Clock Port	Clock	Divider	Control Register	Mux Value	Default Mux Value	Typical Frequency
Main_DDR_EW_wrap_0	ddrss_vbus_clk	MAIN_PLL7.HSDIV0	1				1000
Main_DDR_EW_wrap_0	pll_ctrl_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	1				500
Main_DDR_EW_wrap_1	ddrss_cfg_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_DDR_EW_wrap_1	ddrss_ddr_pll_clk	MAIN_PLL26.HSDIV0	1				1067
Main_DDR_EW_wrap_1	ddrss_io_ck	DDR1_CKP	1				
Main_DDR_EW_wrap_1	ddrss_io_ck_n	DDR1_CKN	1				
Main_DDR_EW_wrap_1	ddrss_vbus_clk	MAIN_PLL7.HSDIV0	1				1000
Main_DDR_EW_wrap_1	pll_ctrl_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	1				500
Main_DMPAC_0	main_clk	MAIN_PLL25.HSDIV0	1				520
Main_DMPAC_0	psil_leaf_clk	MAIN_PLL25.HSDIV0	1				520
Main_DMPAC_0	sde_clk	MAIN_PLL25.HSDIV0	1				520
Main_DPHY_Rx_0	io_rx_cl_l_m	CSI0_RXCLKN	1				
Main_DPHY_Rx_0	io_rx_cl_l_p	CSI0_RXCLKP	1				
Main_DPHY_Rx_0	jtag_tck	MAIN_TAP_BS_JTAG__CLK	1				
Main_DPHY_Rx_0	main_clk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_DPHY_Rx_1	io_rx_cl_l_m	CSI1_RXCLKN	1				
Main_DPHY_Rx_1	io_rx_cl_l_p	CSI1_RXCLKP	1				
Main_DPHY_Rx_1	jtag_tck	MAIN_TAP_BS_JTAG__CLK	1				
Main_DPHY_Rx_1	main_clk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_DSS_0	dss_func_clk	MAIN_PLL2.HSDIV1	1				600
Main_DSS_0	dss_inst0_dpi_0_in_2x_clk	MAIN_PLL16.HSDIV0	1	DPI_0_PCLK_SEL	0	0	600
Main_DSS_0	dss_inst0_dpi_0_in_2x_clk	MAIN_PLL16.HSDIV0	1	DPI_0_PCLK_SEL	1	0	600
Main_DSS_0	dss_inst0_dpi_0_in_2x_clk	MAIN_PLL17.HSDIV0	1	DPI_0_PCLK_SEL	1	0	600
Main_DSS_0	dss_inst0_dpi_0_in_2x_clk	MAIN_PLL19.HSDIV0	1	DPI_0_PCLK_SEL	1	0	600
Main_DSS_0	dss_inst0_dpi_0_in_2x_clk	VOUT0_EXTPCLKIN	1	DPI_0_PCLK_SEL	1	0	
Main_DSS_0	dss_inst0_dpi_0_in_2x_clk	MAIN_PLL17.HSDIV0	1	DPI_1_PCLK_SEL	0	0	600
Main_DSS_0	dss_inst0_dpi_0_in_2x_clk	MAIN_PLL19.HSDIV0	1	DPI_1_PCLK_SEL	1	0	600
Main_DSS_0	dss_inst0_dpi_0_in_2x_clk	VOUT0_EXTPCLKIN	1	DPI_1_PCLK_SEL	1	0	
Main_DSS_0	dss_inst0_dpi_0_in_2x_clk	MAIN_PLL19.HSDIV0	1	DPI_1_PCLK_SEL	2	0	600
Main_DSS_0	dss_inst0_dpi_0_in_2x_clk	VOUT0_EXTPCLKIN	1	DPI_1_PCLK_SEL	2	0	
Main_DSS_0	dss_inst0_dpi_0_in_2x_clk	MAIN_PLL16.HSDIV0	1	DPI_1_PCLK_SEL	3	0	600
Main_DSS_0	dss_inst0_dpi_0_in_2x_clk	MAIN_PLL19.HSDIV0	1	DPI0_EXT_CLKSEL	0	0	600
Main_DSS_0	dss_inst0_dpi_0_in_2x_clk	VOUT0_EXTPCLKIN	1	DPI0_EXT_CLKSEL	1	0	
Main_DSS_0	dss_inst0_dpi_0_in_2x_clk	DPI_0_PCLK	1				600
Main_DSS_0	dss_inst0_dpi_0_in_clk	MAIN_PLL16.HSDIV0	2	DPI_0_PCLK_SEL	0	0	300
Main_DSS_0	dss_inst0_dpi_0_in_clk	MAIN_PLL16.HSDIV0	2	DPI_0_PCLK_SEL	1	0	300
Main_DSS_0	dss_inst0_dpi_0_in_clk	MAIN_PLL17.HSDIV0	2	DPI_0_PCLK_SEL	1	0	300
Main_DSS_0	dss_inst0_dpi_0_in_clk	MAIN_PLL19.HSDIV0	2	DPI_0_PCLK_SEL	1	0	300
Main_DSS_0	dss_inst0_dpi_0_in_clk	VOUT0_EXTPCLKIN	2	DPI_0_PCLK_SEL	1	0	
Main_DSS_0	dss_inst0_dpi_0_in_clk	MAIN_PLL17.HSDIV0	2	DPI_1_PCLK_SEL	0	0	300
Main_DSS_0	dss_inst0_dpi_0_in_clk	MAIN_PLL19.HSDIV0	2	DPI_1_PCLK_SEL	1	0	300
Main_DSS_0	dss_inst0_dpi_0_in_clk	VOUT0_EXTPCLKIN	2	DPI_1_PCLK_SEL	1	0	
Main_DSS_0	dss_inst0_dpi_0_in_clk	MAIN_PLL19.HSDIV0	2	DPI_1_PCLK_SEL	2	0	300
Main_DSS_0	dss_inst0_dpi_0_in_clk	VOUT0_EXTPCLKIN	2	DPI_1_PCLK_SEL	2	0	

Copyright © 2024 Texas Instruments Incorporated

Table 5-33. Clock Mapping (continued)

Module	Clock Port	Clock	Divider	Control Register	Mux Value	Default Mux Value	Typical Frequency
Main_DSS_0	dss_inst0_dpi_0_in_clk	MAIN_PLL16.HSDIV0	2	DPI_1_PCLK_SEL	3	0	300
Main_DSS_0	dss_inst0_dpi_0_in_clk	MAIN_PLL19.HSDIV0	2	DPI0_EXT_CLKSEL	0	0	300
Main_DSS_0	dss_inst0_dpi_0_in_clk	VOUT0_EXTPCLKIN	2	DPI0_EXT_CLKSEL	1	0	
Main_DSS_0	dss_inst0_dpi_0_in_clk	DPI_0_PCLK	2				300
Main_DSS_0	dss_inst0_dpi_1_in_2x_clk	MAIN_PLL17.HSDIV0	1	DPI_1_PCLK_SEL	0	0	600
Main_DSS_0	dss_inst0_dpi_1_in_2x_clk	MAIN_PLL19.HSDIV0	1	DPI_1_PCLK_SEL	1	0	600
Main_DSS_0	dss_inst0_dpi_1_in_2x_clk	VOUT0_EXTPCLKIN	1	DPI_1_PCLK_SEL	1	0	
Main_DSS_0	dss_inst0_dpi_1_in_2x_clk	MAIN_PLL19.HSDIV0	1	DPI_1_PCLK_SEL	2	0	600
Main_DSS_0	dss_inst0_dpi_1_in_2x_clk	VOUT0_EXTPCLKIN	1	DPI_1_PCLK_SEL	2	0	
Main_DSS_0	dss_inst0_dpi_1_in_2x_clk	MAIN_PLL16.HSDIV0	1	DPI_1_PCLK_SEL	3	0	600
Main_DSS_0	dss_inst0_dpi_1_in_2x_clk	MAIN_PLL19.HSDIV0	1	DPI0_EXT_CLKSEL	0	0	600
Main_DSS_0	dss_inst0_dpi_1_in_2x_clk	VOUT0_EXTPCLKIN	1	DPI0_EXT_CLKSEL	1	0	
Main_DSS_0	dss_inst0_dpi_1_in_2x_clk	DPI_1_PCLK	1				600
Main_DSS_0	dss_inst0_dpi_1_in_clk	MAIN_TIEOFF LOW	1				
Main_DSS_0	dss_inst0_dpi_2_in_2x_clk	MAIN_PLL16.HSDIV0	1	DPI_2_PCLK_SEL	0	0	600
Main_DSS_0	dss_inst0_dpi_2_in_2x_clk	MAIN_PLL17.HSDIV0	1	DPI_2_PCLK_SEL	1	0	600
Main_DSS_0	dss_inst0_dpi_2_in_2x_clk	MAIN_PLL16.HSDIV0	1	DPI_2_PCLK_SEL1	0	0	600
Main_DSS_0	dss_inst0_dpi_2_in_2x_clk	MAIN_PLL17.HSDIV0	1	DPI_2_PCLK_SEL1	0	0	600
Main_DSS_0	dss_inst0_dpi_2_in_2x_clk	MAIN_PLL19.HSDIV0	1	DPI_2_PCLK_SEL1	1	0	600
Main_DSS_0	dss_inst0_dpi_2_in_2x_clk	VOUT0_EXTPCLKIN	1	DPI_2_PCLK_SEL1	1	0	
Main_DSS_0	dss_inst0_dpi_2_in_2x_clk	MAIN_PLL19.HSDIV0	1	DPI0_EXT_CLKSEL	0	0	600
Main_DSS_0	dss_inst0_dpi_2_in_2x_clk	VOUT0_EXTPCLKIN	1	DPI0_EXT_CLKSEL	1	0	
Main_DSS_0	dss_inst0_dpi_2_in_2x_clk	DPI_2_PCLK	1				600
Main_DSS_0	dss_inst0_dpi_2_in_clk	MAIN_PLL16.HSDIV0	2	DPI_2_PCLK_SEL	0	0	300
Main_DSS_0	dss_inst0_dpi_2_in_clk	MAIN_PLL17.HSDIV0	2	DPI_2_PCLK_SEL	1	0	300
Main_DSS_0	dss_inst0_dpi_2_in_clk	MAIN_PLL16.HSDIV0	2	DPI_2_PCLK_SEL1	0	0	300
Main_DSS_0	dss_inst0_dpi_2_in_clk	MAIN_PLL17.HSDIV0	2	DPI_2_PCLK_SEL1	0	0	300
Main_DSS_0	dss_inst0_dpi_2_in_clk	MAIN_PLL19.HSDIV0	2	DPI_2_PCLK_SEL1	1	0	300
Main_DSS_0	dss_inst0_dpi_2_in_clk	VOUT0_EXTPCLKIN	2	DPI_2_PCLK_SEL1	1	0	
Main_DSS_0	dss_inst0_dpi_2_in_clk	MAIN_PLL19.HSDIV0	2	DPI0_EXT_CLKSEL	0	0	300
Main_DSS_0	dss_inst0_dpi_2_in_clk	VOUT0_EXTPCLKIN	2	DPI0_EXT_CLKSEL	1	0	
Main_DSS_0	dss_inst0_dpi_2_in_clk	DPI_2_PCLK	2				300
Main_DSS_0	dss_inst0_dpi_3_in_2x_clk	MAIN_PLL19.HSDIV0	1	DPI_3_PCLK_SEL	3	0	600
Main_DSS_0	dss_inst0_dpi_3_in_2x_clk	VOUT0_EXTPCLKIN	1	DPI_3_PCLK_SEL	3	0	
Main_DSS_0	dss_inst0_dpi_3_in_2x_clk	MAIN_PLL19.HSDIV0	1	DPI_3_PCLK_SEL	4	0	600
Main_DSS_0	dss_inst0_dpi_3_in_2x_clk	VOUT0_EXTPCLKIN	1	DPI_3_PCLK_SEL	4	0	
Main_DSS_0	dss_inst0_dpi_3_in_2x_clk	MAIN_PLL19.HSDIV0	1	DPI_3_PCLK_SEL	5	0	600
Main_DSS_0	dss_inst0_dpi_3_in_2x_clk	VOUT0_EXTPCLKIN	1	DPI_3_PCLK_SEL	5	0	
Main_DSS_0	dss_inst0_dpi_3_in_2x_clk	MAIN_PLL19.HSDIV0	1	DPI_3_PCLK_SEL	6	0	600
Main_DSS_0	dss_inst0_dpi_3_in_2x_clk	VOUT0_EXTPCLKIN	1	DPI_3_PCLK_SEL	6	0	

Copyright © 2024 Texas Instruments Incorporated

Table 5-33. Clock Mapping (continued)

Module	Clock Port	Clock	Divider	Control Register	Mux Value	Default Mux Value	Typical Frequency
Main_DSS_0	dss_inst0_dpi_3_in_2x_clk	MAIN_PLL19.HSDIV0	1	DPI_3_PCLK_SEL	7	0	600
Main_DSS_0	dss_inst0_dpi_3_in_2x_clk	VOUT0_EXTPCLKIN	1	DPI_3_PCLK_SEL	7	0	
Main_DSS_0	dss_inst0_dpi_3_in_2x_clk	MAIN_PLL19.HSDIV0	1	DPI0_EXT_CLKSEL	0	0	600
Main_DSS_0	dss_inst0_dpi_3_in_2x_clk	VOUT0_EXTPCLKIN	1	DPI0_EXT_CLKSEL	1	0	
Main_DSS_0	dss_inst0_dpi_3_in_2x_clk	DPI_3_PCLK	1				600
Main_DSS_0	dss_inst0_dpi_3_in_clk	MAIN_TIEOFF LOW	1				
Main_DSS_DSI_0	dphy_0_rx_esc_clk	MAIN_SERDES_0.IP1_PPI_M_RXCLKESC_CLK	1				
Main_DSS_DSI_0	dphy_0_tx_esc_clk	MAIN_PLL1.HSDIV8	1				20
Main_DSS_DSI_0	dpi_0_clk	MAIN_DSS_0.DSS_INST0_DPI_2_OUT_CLK	1				
Main_DSS_DSI_0	pll_ctrl_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	1				500
Main_DSS_DSI_0	ppi_0_TxByteClkHS_cl_clk	MAIN_SERDES_0.IP1_PPI_TXBYTECLKHS_CL_CLK	1				
Main_DSS_DSI_0	sys_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	2	DSI_DSS_CLK_SEL	0	0	250
Main_DSS_DSI_0	sys_clk	MAIN_PLL2.HSDIV7	1	DSI_DSS_CLK_SEL	1	0	120
Main_DSS_DSI_1	dphy_0_rx_esc_clk	MAIN_SERDES_1.IP1_PPI_M_RXCLKESC_CLK	1				
Main_DSS_DSI_1	dphy_0_tx_esc_clk	MAIN_PLL1.HSDIV8	1				20
Main_DSS_DSI_1	dpi_0_clk	MAIN_DSS_0.DSS_INST0_DPI_1_OUT_CLK	1				
Main_DSS_DSI_1	pll_ctrl_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	1				500
Main_DSS_DSI_1	ppi_0_TxByteClkHS_cl_clk	MAIN_SERDES_1.IP1_PPI_TXBYTECLKHS_CL_CLK	1				
Main_DSS_DSI_1	sys_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	2	DSI_DSS_CLK_SEL	0	0	250
Main_DSS_DSI_1	sys_clk	MAIN_PLL2.HSDIV7	1	DSI_DSS_CLK_SEL	1	0	120
Main_DSS_EDP_0	aif_i2s_clk	MAIN_MCASP_4.MCASP_ACLKX	1				
Main_DSS_EDP_0	dpi_0_2x_clk	MAIN_TIEOFF LOW	1				
Main_DSS_EDP_0	dpi_0_clk	MAIN_TIEOFF LOW	1				
Main_DSS_EDP_0	dpi_1_clk	MAIN_TIEOFF LOW	1				
Main_DSS_EDP_0	dpi_2_2x_clk	MAIN_DSS_0.DSS_INST0_DPI_2_OUT_2X_CLK	1				
Main_DSS_EDP_0	dpi_2_clk	MAIN_DSS_0.DSS_INST0_DPI_0_OUT_CLK	1				
Main_DSS_EDP_0	dpi_3_clk	MAIN_DSS_0.DSS_INST0_DPI_1_OUT_CLK	1				
Main_DSS_EDP_0	dpi_4_clk	MAIN_DSS_0.DSS_INST0_DPI_2_OUT_CLK	1				
Main_DSS_EDP_0	dpi_5_clk	MAIN_DSS_0.DSS_INST0_DPI_3_OUT_CLK	1				
Main_DSS_EDP_0	dptx_mod_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4	eDP_DSS_CLK_SEL	0	0	125
Main_DSS_EDP_0	dptx_mod_clk	MAIN_PLL2.HSDIV7	1	eDP_DSS_CLK_SEL	1	0	120
Main_DSS_EDP_0	phy_ln0_refclk	MAIN_SERDES_0.IP1_LN0_REFCLK	1	eDP0_LN0_REFCLK	0	0	
Main_DSS_EDP_0	phy_ln0_refclk	MAIN_SERDES_0.IP1_LN2_REFCLK	1	eDP0_LN0_REFCLK	1	0	
Main_DSS_EDP_0	phy_ln0_rxclk	MAIN_SERDES_0.IP1_LN0_RXCLK	1	eDP0_LN0_RXCLK	0	0	
Main_DSS_EDP_0	phy_ln0_rxclk	MAIN_SERDES_0.IP1_LN2_RXCLK	1	eDP0_LN0_RXCLK	1	0	
Main_DSS_EDP_0	phy_ln0_rxfclk	MAIN_SERDES_0.IP1_LN0_RXFCLK	1	eDP0_LN0_RXFCLK	0	0	
Main_DSS_EDP_0	phy_ln0_rxfclk	MAIN_SERDES_0.IP1_LN2_RXFCLK	1	eDP0_LN0_RXFCLK	1	0	
Main_DSS_EDP_0	phy_ln0_txfclk	MAIN_SERDES_0.IP1_LN0_TXFCLK	1	eDP0_LN0_TXFCLK	0	0	
Main_DSS_EDP_0	phy_ln0_txfclk	MAIN_SERDES_0.IP1_LN2_TXFCLK	1	eDP0_LN0_TXFCLK	1	0	
Main_DSS_EDP_0	phy_ln0_txmclk	MAIN_SERDES_0.IP1_LN0_TXMCLK	1	eDP0_LN0_TXMCLK	0	0	
Main_DSS_EDP_0	phy_ln0_txmclk	MAIN_SERDES_0.IP1_LN2_TXMCLK	1	eDP0_LN0_TXMCLK	1	0	
Main_DSS_EDP_0	phy_ln1_refclk	MAIN_SERDES_0.IP1_LN1_REFCLK	1	eDP0_LN1_REFCLK	0	0	
Main_DSS_EDP_0	phy_ln1_refclk	MAIN_SERDES_0.IP1_LN3_REFCLK	1	eDP0_LN1_REFCLK	1	0	

Copyright © 2024 Texas Instruments Incorporated

Table 5-33. Clock Mapping (continued)

Module	Clock Port	Clock	Divider	Control Register	Mux Value	Default Mux Value	Typical Frequency
Main_DSS_EDP_0	phy_ln1_rxclk	MAIN_SERDES_0.IP1_LN1_RXCLK	1	eDP0_LN1_RXCLK	0	0	
Main_DSS_EDP_0	phy_ln1_rxclk	MAIN_SERDES_0.IP1_LN3_RXCLK	1	eDP0_LN1_RXCLK	1	0	
Main_DSS_EDP_0	phy_ln1_rxfclk	MAIN_SERDES_0.IP1_LN1_RXFCLK	1	eDP0_LN1_RXFCLK	0	0	
Main_DSS_EDP_0	phy_ln1_rxfclk	MAIN_SERDES_0.IP1_LN3_RXFCLK	1	eDP0_LN1_RXFCLK	1	0	
Main_DSS_EDP_0	phy_ln1_txfclk	MAIN_SERDES_0.IP1_LN1_TXFCLK	1	eDP0_LN1_TXFCLK	0	0	
Main_DSS_EDP_0	phy_ln1_txfclk	MAIN_SERDES_0.IP1_LN3_TXFCLK	1	eDP0_LN1_TXFCLK	1	0	
Main_DSS_EDP_0	phy_ln1_txmclk	MAIN_SERDES_0.IP1_LN1_TXMCLK	1	eDP0_LN1_TXMCLK	0	0	
Main_DSS_EDP_0	phy_ln1_txmclk	MAIN_SERDES_0.IP1_LN3_TXMCLK	1	eDP0_LN1_TXMCLK	1	0	
Main_DSS_EDP_0	phy_ln2_refclk	MAIN_SERDES_0.IP1_LN2_REFCLK	1				
Main_DSS_EDP_0	phy_ln2_rxclk	MAIN_SERDES_0.IP1_LN2_RXCLK	1				
Main_DSS_EDP_0	phy_ln2_rxfclk	MAIN_SERDES_0.IP1_LN2_RXFCLK	1				
Main_DSS_EDP_0	phy_ln2_txfclk	MAIN_SERDES_0.IP1_LN2_TXFCLK	1				
Main_DSS_EDP_0	phy_ln2_txmclk	MAIN_SERDES_0.IP1_LN2_TXMCLK	1				
Main_DSS_EDP_0	phy_ln3_refclk	MAIN_SERDES_0.IP1_LN3_REFCLK	1				
Main_DSS_EDP_0	phy_ln3_rxclk	MAIN_SERDES_0.IP1_LN3_RXCLK	1				
Main_DSS_EDP_0	phy_ln3_rxfclk	MAIN_SERDES_0.IP1_LN3_RXFCLK	1				
Main_DSS_EDP_0	phy_ln3_txfclk	MAIN_SERDES_0.IP1_LN3_TXFCLK	1				
Main_DSS_EDP_0	phy_ln3_txmclk	MAIN_SERDES_0.IP1_LN3_TXMCLK	1				
Main_DSS_EDP_0	pll_ctrl_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	1				500
Main_ECAP_0	vbus_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_ECAP_1	vbus_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_ECAP_2	vbus_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_ELM_0	vbusp_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_EMMC4_0	emmcsdss_io_clk_i	MMC1_CLKLB	1	EMMCSD1_LB_CLKSEL	0	0	
Main_EMMC4_0	emmcsdss_io_clk_i	MMC1_CLK	1	EMMCSD1_LB_CLKSEL	1	0	
Main_EMMC4_0	emmcsdss_vbus_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	2				250
Main_EMMC4_0	emmcsdss_xin_clk	MAIN_PLL0.HSDIV2	1	EMMCSD_REFCLK_SEL	0	1	200
Main_EMMC4_0	emmcsdss_xin_clk	MAIN_PLL1.HSDIV2	1	EMMCSD_REFCLK_SEL	1	1	192
Main_EMMC4_0	emmcsdss_xin_clk	MAIN_PLL2.HSDIV2	1	EMMCSD_REFCLK_SEL	2	1	200
Main_EMMC4_0	emmcsdss_xin_clk	MAIN_PLL3.HSDIV2	1	EMMCSD_REFCLK_SEL	3	1	200
Main_EMMC8_0	emmcss_vbus_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	2				250
Main_EMMC8_0	emmcss_xin_clk	MAIN_PLL0.HSDIV2	1	EMMCSD_REFCLK_SEL	0	0	200
Main_EMMC8_0	emmcss_xin_clk	MAIN_PLL1.HSDIV2	1	EMMCSD_REFCLK_SEL	1	0	192
Main_EMMC8_0	emmcss_xin_clk	MAIN_PLL2.HSDIV2	1	EMMCSD_REFCLK_SEL	2	0	200
Main_EMMC8_0	emmcss_xin_clk	MAIN_PLL3.HSDIV2	1	EMMCSD_REFCLK_SEL	3	0	200
Main_EPWM_0	vbusp_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_EPWM_1	vbusp_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_EPWM_2	vbusp_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_EPWM_3	vbusp_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_EPWM_4	vbusp_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_EPWM_5	vbusp_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125

Copyright © 2024 Texas Instruments Incorporated

Table 5-33. Clock Mapping (continued)

Module	Clock Port	Clock	Divider	Control Register	Mux Value	Default Mux Value	Typical Frequency
Main_EQEP_0	vbus_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_EQEP_1	vbus_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_EQEP_2	vbus_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_ESM_0	clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_GPIO_0	mmr_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_GPIO_2	mmr_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_GPIO_4	mmr_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_GPIO_6	mmr_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_GPMC_0	func_clk	MAIN_PLL0.HSDIV3	1	gpmc_fclk_sel	0	0	133
Main_GPMC_0	func_clk	MAIN_PLL2.HSDIV1	6	gpmc_fclk_sel	1	0	100
Main_GPMC_0	func_clk	MAIN_PLL2.HSDIV1	4	gpmc_fclk_sel	2	0	150
Main_GPMC_0	func_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4	gpmc_fclk_sel	3	0	125
Main_GPMC_0	func_clk	GPMC_FCLK	1				150
Main_GPMC_0	pi_gpmc_ret_clk	GPMC0_CLK	1				
Main_GPMC_0	vbusm_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	2				250
Main_GPU_0	fclk	main_scan_test_clk	1				
Main_GPU_0	gpu_pll_clk	MAIN_PLL6.HSDIV0	1				800
Main_GPU_0	j7aep_gpu_bxs464_dust_fclk	main_scan_test_clk	1				
Main_GPU_0	j7aep_gpu_bxs464_rslc256_fclk	main_scan_test_clk	1				
Main_GPU_0	pll_ctrl_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	1				500
Main_GTC_0	gtc_clk	MAIN_PLL3.HSDIV1	1	GTC_CLK_MUX	0	0	250
Main_GTC_0	gtc_clk	MAIN_PLL0.HSDIV6	1	GTC_CLK_MUX	1	0	250
Main_GTC_0	gtc_clk	MCU_CPTS0_RFT_CLK	1	GTC_CLK_MUX	2	0	
Main_GTC_0	gtc_clk	CPTS0_RFT_CLK	1	GTC_CLK_MUX	3	0	
Main_GTC_0	gtc_clk	MCU_EXT_REFCLK0	1	GTC_CLK_MUX	4	0	100
Main_GTC_0	gtc_clk	EXT_REFCLK1	1	GTC_CLK_MUX	5	0	
Main_GTC_0	gtc_clk	MAIN_SERDES_0.IP2_LN0_TXMCLK	1	GTC_CLK_MUX	6	0	
Main_GTC_0	gtc_clk	MAIN_SERDES_0.IP2_LN1_TXMCLK	1	GTC_CLK_MUX	7	0	
Main_GTC_0	gtc_clk	MAIN_SERDES_0.IP2_LN2_TXMCLK	1	GTC_CLK_MUX	8	0	
Main_GTC_0	gtc_clk	MAIN_SERDES_0.IP2_LN3_TXMCLK	1	GTC_CLK_MUX	9	0	
Main_GTC_0	gtc_clk	MCU_PLL2.HSDIV1	1	GTC_CLK_MUX	14	0	500
Main_GTC_0	gtc_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	1	GTC_CLK_MUX	15	0	500
Main_GTC_0	vbusp_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_I2C_0	clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_I2C_0	piscl	I2C0_SCL	1				
Main_I2C_0	pisys_clk	MAIN_PLL1.HSDIV0	2				96
Main_I2C_1	clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_I2C_1	piscl	I2C1_SCL	1				
Main_I2C_1	pisys_clk	MAIN_PLL1.HSDIV0	2				96
Main_I2C_2	clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_I2C_2	piscl	I2C2_SCL	1				
Main_I2C_2	pisys_clk	MAIN_PLL1.HSDIV0	2				96
Main_I2C_3	clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_I2C_3	piscl	I2C3_SCL	1				

Copyright © 2024 Texas Instruments Incorporated

Table 5-33. Clock Mapping (continued)

Module	Clock Port	Clock	Divider	Control Register	Mux Value	Default Mux Value	Typical Frequency
Main_I2C_3	pisys_clk	MAIN_PLL1.HSDIV0	2				96
Main_I2C_4	clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_I2C_4	piscl	I2C4_SCL	1				
Main_I2C_4	pisys_clk	MAIN_PLL1.HSDIV0	2				96
Main_I2C_5	clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_I2C_5	piscl	I2C5_SCL	1				
Main_I2C_5	pisys_clk	MAIN_PLL1.HSDIV0	2				96
Main_I2C_6	clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_I2C_6	piscl	I2C6_SCL	1				
Main_I2C_6	pisys_clk	MAIN_PLL1.HSDIV0	2				96
Main_MCANSS_0	mcanss_can_rxd	MCAN0_RX	1				
Main_MCANSS_0	mcanss_cclk_clk	MAIN_PLL0.HSDIV4	1	MCAN_CLK_SEL	0	0	80
Main_MCANSS_0	mcanss_cclk_clk	MCU_EXT_REFCLK0	1	MCAN_CLK_SEL	1	0	100
Main_MCANSS_0	mcanss_cclk_clk	HFOSC1	1	MCAN_CLK_SEL	2	0	[19.2 - 27]
Main_MCANSS_0	mcanss_cclk_clk	HFOSC0	1	MCAN_CLK_SEL	3	0	[19.2, 20, 24, 25, 26, 27]
Main_MCANSS_0	mcanss_hclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_MCANSS_1	mcanss_can_rxd	MCAN1_RX	1				
Main_MCANSS_1	mcanss_cclk_clk	MAIN_PLL0.HSDIV4	1	MCAN_CLK_SEL	0	1	80
Main_MCANSS_1	mcanss_cclk_clk	MCU_EXT_REFCLK0	1	MCAN_CLK_SEL	1	1	100
Main_MCANSS_1	mcanss_cclk_clk	HFOSC1	1	MCAN_CLK_SEL	2	1	[19.2 - 27]
Main_MCANSS_1	mcanss_cclk_clk	HFOSC0	1	MCAN_CLK_SEL	3	1	[19.2, 20, 24, 25, 26, 27]
Main_MCANSS_1	mcanss_hclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_MCANSS_2	mcanss_can_rxd	MCAN2_RX	1				
Main_MCANSS_2	mcanss_cclk_clk	MAIN_PLL0.HSDIV4	1	MCAN_CLK_SEL	0	2	80
Main_MCANSS_2	mcanss_cclk_clk	MCU_EXT_REFCLK0	1	MCAN_CLK_SEL	1	2	100
Main_MCANSS_2	mcanss_cclk_clk	HFOSC1	1	MCAN_CLK_SEL	2	2	[19.2 - 27]
Main_MCANSS_2	mcanss_cclk_clk	HFOSC0	1	MCAN_CLK_SEL	3	2	[19.2, 20, 24, 25, 26, 27]
Main_MCANSS_2	mcanss_hclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_MCANSS_3	mcanss_can_rxd	MCAN3_RX	1				
Main_MCANSS_3	mcanss_cclk_clk	MAIN_PLL0.HSDIV4	1	MCAN_CLK_SEL	0	3	80
Main_MCANSS_3	mcanss_cclk_clk	MCU_EXT_REFCLK0	1	MCAN_CLK_SEL	1	3	100
Main_MCANSS_3	mcanss_cclk_clk	HFOSC1	1	MCAN_CLK_SEL	2	3	[19.2 - 27]
Main_MCANSS_3	mcanss_cclk_clk	HFOSC0	1	MCAN_CLK_SEL	3	3	[19.2, 20, 24, 25, 26, 27]
Main_MCANSS_3	mcanss_hclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_MCANSS_4	mcanss_can_rxd	MCAN4_RX	1				
Main_MCANSS_4	mcanss_cclk_clk	MAIN_PLL0.HSDIV4	1	MCAN_CLK_SEL	0	4	80
Main_MCANSS_4	mcanss_cclk_clk	MCU_EXT_REFCLK0	1	MCAN_CLK_SEL	1	4	100
Main_MCANSS_4	mcanss_cclk_clk	HFOSC1	1	MCAN_CLK_SEL	2	4	[19.2 - 27]
Main_MCANSS_4	mcanss_cclk_clk	HFOSC0	1	MCAN_CLK_SEL	3	4	[19.2, 20, 24, 25, 26, 27]
Main_MCANSS_4	mcanss_hclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_MCANSS_5	mcanss_can_rxd	MCAN5_RX	1				
Main_MCANSS_5	mcanss_cclk_clk	MAIN_PLL0.HSDIV4	1	MCAN_CLK_SEL	0	5	80
Main_MCANSS_5	mcanss_cclk_clk	MCU_EXT_REFCLK0	1	MCAN_CLK_SEL	1	5	100
Main_MCANSS_5	mcanss_cclk_clk	HFOSC1	1	MCAN_CLK_SEL	2	5	[19.2 - 27]

Copyright © 2024 Texas Instruments Incorporated

Table 5-33. Clock Mapping (continued)

Module	Clock Port	Clock	Divider	Control Register	Mux Value	Default Mux Value	Typical Frequency
Main_MCANSS_5	mcanss_cclk_clk	HFOSC0	1	MCAN_CLK_SEL	3	5	[19.2, 20, 24, 25, 26, 27]
Main_MCANSS_5	mcanss_hclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_MCANSS_6	mcanss_can_rxd	MCAN6_RX	1				
Main_MCANSS_6	mcanss_cclk_clk	MAIN_PLL0.HSDIV4	1	MCAN_CLK_SEL	0	6	80
Main_MCANSS_6	mcanss_cclk_clk	MCU_EXT_REFCLK0	1	MCAN_CLK_SEL	1	6	100
Main_MCANSS_6	mcanss_cclk_clk	HFOSC1	1	MCAN_CLK_SEL	2	6	[19.2 - 27]
Main_MCANSS_6	mcanss_cclk_clk	HFOSC0	1	MCAN_CLK_SEL	3	6	[19.2, 20, 24, 25, 26, 27]
Main_MCANSS_6	mcanss_hclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_MCANSS_7	mcanss_can_rxd	MCAN7_RX	1				
Main_MCANSS_7	mcanss_cclk_clk	MAIN_PLL0.HSDIV4	1	MCAN_CLK_SEL	0	7	80
Main_MCANSS_7	mcanss_cclk_clk	MCU_EXT_REFCLK0	1	MCAN_CLK_SEL	1	7	100
Main_MCANSS_7	mcanss_cclk_clk	HFOSC1	1	MCAN_CLK_SEL	2	7	[19.2 - 27]
Main_MCANSS_7	mcanss_cclk_clk	HFOSC0	1	MCAN_CLK_SEL	3	7	[19.2, 20, 24, 25, 26, 27]
Main_MCANSS_7	mcanss_hclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_MCANSS_8	mcanss_can_rxd	MCAN8_RX	1				
Main_MCANSS_8	mcanss_cclk_clk	MAIN_PLL0.HSDIV4	1	MCAN_CLK_SEL	0	8	80
Main_MCANSS_8	mcanss_cclk_clk	MCU_EXT_REFCLK0	1	MCAN_CLK_SEL	1	8	100
Main_MCANSS_8	mcanss_cclk_clk	HFOSC1	1	MCAN_CLK_SEL	2	8	[19.2 - 27]
Main_MCANSS_8	mcanss_cclk_clk	HFOSC0	1	MCAN_CLK_SEL	3	8	[19.2, 20, 24, 25, 26, 27]
Main_MCANSS_8	mcanss_hclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_MCANSS_9	mcanss_can_rxd	MCAN9_RX	1				
Main_MCANSS_9	mcanss_cclk_clk	MAIN_PLL0.HSDIV4	1	MCAN_CLK_SEL	0	9	80
Main_MCANSS_9	mcanss_cclk_clk	MCU_EXT_REFCLK0	1	MCAN_CLK_SEL	1	9	100
Main_MCANSS_9	mcanss_cclk_clk	HFOSC1	1	MCAN_CLK_SEL	2	9	[19.2 - 27]
Main_MCANSS_9	mcanss_cclk_clk	HFOSC0	1	MCAN_CLK_SEL	3	9	[19.2, 20, 24, 25, 26, 27]
Main_MCANSS_9	mcanss_hclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_MCANSS_10	mcanss_can_rxd	MCAN10_RX	1				
Main_MCANSS_10	mcanss_cclk_clk	MAIN_PLL0.HSDIV4	1	MCAN_CLK_SEL	0	10	80
Main_MCANSS_10	mcanss_cclk_clk	MCU_EXT_REFCLK0	1	MCAN_CLK_SEL	1	10	100
Main_MCANSS_10	mcanss_cclk_clk	HFOSC1	1	MCAN_CLK_SEL	2	10	[19.2 - 27]
Main_MCANSS_10	mcanss_cclk_clk	HFOSC0	1	MCAN_CLK_SEL	3	10	[19.2, 20, 24, 25, 26, 27]
Main_MCANSS_10	mcanss_hclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_MCANSS_11	mcanss_can_rxd	MCAN11_RX	1				
Main_MCANSS_11	mcanss_cclk_clk	MAIN_PLL0.HSDIV4	1	MCAN_CLK_SEL	0	11	80
Main_MCANSS_11	mcanss_cclk_clk	MCU_EXT_REFCLK0	1	MCAN_CLK_SEL	1	11	100
Main_MCANSS_11	mcanss_cclk_clk	HFOSC1	1	MCAN_CLK_SEL	2	11	[19.2 - 27]
Main_MCANSS_11	mcanss_cclk_clk	HFOSC0	1	MCAN_CLK_SEL	3	11	[19.2, 20, 24, 25, 26, 27]
Main_MCANSS_11	mcanss_hclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_MCANSS_12	mcanss_can_rxd	MCAN12_RX	1				
Main_MCANSS_12	mcanss_cclk_clk	MAIN_PLL0.HSDIV4	1	MCAN_CLK_SEL	0	12	80
Main_MCANSS_12	mcanss_cclk_clk	MCU_EXT_REFCLK0	1	MCAN_CLK_SEL	1	12	100
Main_MCANSS_12	mcanss_cclk_clk	HFOSC1	1	MCAN_CLK_SEL	2	12	[19.2 - 27]

Copyright © 2024 Texas Instruments Incorporated

Table 5-33. Clock Mapping (continued)

Module	Clock Port	Clock	Divider	Control Register	Mux Value	Default Mux Value	Typical Frequency
Main_MCANSS_12	mcanss_cclk_clk	HFOSC0	1	MCAN_CLK_SEL	3	12	[19.2, 20, 24, 25, 26, 27]
Main_MCANSS_12	mcanss_hclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_MCANSS_13	mcanss_can_rxd	MCAN13_RX	1				
Main_MCANSS_13	mcanss_cclk_clk	MAIN_PLL0.HSDIV4	1	MCAN_CLK_SEL	0	13	80
Main_MCANSS_13	mcanss_cclk_clk	MCU_EXT_REFCLK0	1	MCAN_CLK_SEL	1	13	100
Main_MCANSS_13	mcanss_cclk_clk	HFOSC1	1	MCAN_CLK_SEL	2	13	[19.2 - 27]
Main_MCANSS_13	mcanss_cclk_clk	HFOSC0	1	MCAN_CLK_SEL	3	13	[19.2, 20, 24, 25, 26, 27]
Main_MCANSS_13	mcanss_hclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_MCANSS_14	mcanss_can_rxd	MCAN14_RX	1				
Main_MCANSS_14	mcanss_cclk_clk	MAIN_PLL0.HSDIV4	1	MCAN_CLK_SEL	0	14	80
Main_MCANSS_14	mcanss_cclk_clk	MCU_EXT_REFCLK0	1	MCAN_CLK_SEL	1	14	100
Main_MCANSS_14	mcanss_cclk_clk	HFOSC1	1	MCAN_CLK_SEL	2	14	[19.2 - 27]
Main_MCANSS_14	mcanss_cclk_clk	HFOSC0	1	MCAN_CLK_SEL	3	14	[19.2, 20, 24, 25, 26, 27]
Main_MCANSS_14	mcanss_hclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_MCANSS_15	mcanss_can_rxd	MCAN15_RX	1				
Main_MCANSS_15	mcanss_cclk_clk	MAIN_PLL0.HSDIV4	1	MCAN_CLK_SEL	0	15	80
Main_MCANSS_15	mcanss_cclk_clk	MCU_EXT_REFCLK0	1	MCAN_CLK_SEL	1	15	100
Main_MCANSS_15	mcanss_cclk_clk	HFOSC1	1	MCAN_CLK_SEL	2	15	[19.2 - 27]
Main_MCANSS_15	mcanss_cclk_clk	HFOSC0	1	MCAN_CLK_SEL	3	15	[19.2, 20, 24, 25, 26, 27]
Main_MCANSS_15	mcanss_hclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_MCANSS_16	mcanss_can_rxd	MCAN16_RX	1				
Main_MCANSS_16	mcanss_cclk_clk	MAIN_PLL0.HSDIV4	1	MCAN_CLK_SEL	0	16	80
Main_MCANSS_16	mcanss_cclk_clk	MCU_EXT_REFCLK0	1	MCAN_CLK_SEL	1	16	100
Main_MCANSS_16	mcanss_cclk_clk	HFOSC1	1	MCAN_CLK_SEL	2	16	[19.2 - 27]
Main_MCANSS_16	mcanss_cclk_clk	HFOSC0	1	MCAN_CLK_SEL	3	16	[19.2, 20, 24, 25, 26, 27]
Main_MCANSS_16	mcanss_hclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_MCANSS_17	mcanss_can_rxd	MCAN17_RX	1				
Main_MCANSS_17	mcanss_cclk_clk	MAIN_PLL0.HSDIV4	1	MCAN_CLK_SEL	0	17	80
Main_MCANSS_17	mcanss_cclk_clk	MCU_EXT_REFCLK0	1	MCAN_CLK_SEL	1	17	100
Main_MCANSS_17	mcanss_cclk_clk	HFOSC1	1	MCAN_CLK_SEL	2	17	[19.2 - 27]
Main_MCANSS_17	mcanss_cclk_clk	HFOSC0	1	MCAN_CLK_SEL	3	17	[19.2, 20, 24, 25, 26, 27]
Main_MCANSS_17	mcanss_hclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_MCASP_0	aux_clk	MAIN_PLL4.HSDIV0	1	McASP_AUXCLK_SEL	0	0	197
Main_MCASP_0	aux_clk	MAIN_PLL2.HSDIV2	1	McASP_AUXCLK_SEL	1	0	200
Main_MCASP_0	aux_clk	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT[0]	1	McASP_AUXCLK_SEL	4	0	
Main_MCASP_0	aux_clk	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT[1]	1	McASP_AUXCLK_SEL	5	0	
Main_MCASP_0	aux_clk	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT[2]	1	McASP_AUXCLK_SEL	6	0	
Main_MCASP_0	aux_clk	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT[3]	1	McASP_AUXCLK_SEL	7	0	
Main_MCASP_0	mcasp_aclkr_pin	MCASP0_ACLKR	1				
Main_MCASP_0	mcasp_aclkx_pin	MCASP0_ACLKX	1				
Main_MCASP_0	mcasp_ahclkr_pin	HFOSC1	1	mcasp_ahclkr_mux	0	0	[19.2 - 27]

Copyright © 2024 Texas Instruments Incorporated

Table 5-33. Clock Mapping (continued)

Module	Clock Port	Clock	Divider	Control Register	Mux Value	Default Mux Value	Typical Frequency
Main_MCASP_0	mcasp_ahclkr_pin	HFOSC0	1	mcasp_ahclkr_mux	1	0	[19.2, 20, 24, 25, 26, 27]
Main_MCASP_0	mcasp_ahclkr_pin	AUDIO_EXT_REFCLK0	1	mcasp_ahclkr_mux	2	0	
Main_MCASP_0	mcasp_ahclkr_pin	AUDIO_EXT_REFCLK1	1	mcasp_ahclkr_mux	3	0	
Main_MCASP_0	mcasp_ahclkr_pin	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT[0]	1	mcasp_ahclkr_mux	8	0	
Main_MCASP_0	mcasp_ahclkr_pin	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT[1]	1	mcasp_ahclkr_mux	9	0	
Main_MCASP_0	mcasp_ahclkr_pin	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT[2]	1	mcasp_ahclkr_mux	10	0	
Main_MCASP_0	mcasp_ahclkr_pin	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT[3]	1	mcasp_ahclkr_mux	11	0	
Main_MCASP_0	mcasp_ahclkx_pin	HFOSC1	1	mcasp_ahclkx_mux	0	0	[19.2 - 27]
Main_MCASP_0	mcasp_ahclkx_pin	HFOSC0	1	mcasp_ahclkx_mux	1	0	[19.2, 20, 24, 25, 26, 27]
Main_MCASP_0	mcasp_ahclkx_pin	AUDIO_EXT_REFCLK0	1	mcasp_ahclkx_mux	2	0	
Main_MCASP_0	mcasp_ahclkx_pin	AUDIO_EXT_REFCLK1	1	mcasp_ahclkx_mux	3	0	
Main_MCASP_0	mcasp_ahclkx_pin	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT[0]	1	mcasp_ahclkx_mux	8	0	
Main_MCASP_0	mcasp_ahclkx_pin	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT[1]	1	mcasp_ahclkx_mux	9	0	
Main_MCASP_0	mcasp_ahclkx_pin	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT[2]	1	mcasp_ahclkx_mux	10	0	
Main_MCASP_0	mcasp_ahclkx_pin	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT[3]	1	mcasp_ahclkx_mux	11	0	
Main_MCASP_0	vbusp_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	2				250
Main_MCASP_1	aux_clk	MAIN_PLL4.HSDIV0	1	McASP_AUXCLK_SEL	0	1	197
Main_MCASP_1	aux_clk	MAIN_PLL2.HSDIV2	1	McASP_AUXCLK_SEL	1	1	200
Main_MCASP_1	aux_clk	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT	1	McASP_AUXCLK_SEL	4	1	
Main_MCASP_1	aux_clk	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT	1	McASP_AUXCLK_SEL	5	1	
Main_MCASP_1	aux_clk	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT	1	McASP_AUXCLK_SEL	6	1	
Main_MCASP_1	aux_clk	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT	1	McASP_AUXCLK_SEL	7	1	
Main_MCASP_1	mcasp_aclkr_pin	MCASP1_ACLKR	1				
Main_MCASP_1	mcasp_aclkx_pin	MCASP1_ACLKX	1				
Main_MCASP_1	mcasp_ahclkr_pin	HFOSC1	1	mcasp_ahclkr_mux	0	1	[19.2 - 27]
Main_MCASP_1	mcasp_ahclkr_pin	HFOSC0	1	mcasp_ahclkr_mux	1	1	[19.2, 20, 24, 25, 26, 27]
Main_MCASP_1	mcasp_ahclkr_pin	AUDIO_EXT_REFCLK0	1	mcasp_ahclkr_mux	2	1	
Main_MCASP_1	mcasp_ahclkr_pin	AUDIO_EXT_REFCLK1	1	mcasp_ahclkr_mux	3	1	
Main_MCASP_1	mcasp_ahclkr_pin	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT	1	mcasp_ahclkr_mux	8	1	
Main_MCASP_1	mcasp_ahclkr_pin	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT	1	mcasp_ahclkr_mux	9	1	
Main_MCASP_1	mcasp_ahclkr_pin	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT	1	mcasp_ahclkr_mux	10	1	
Main_MCASP_1	mcasp_ahclkr_pin	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT	1	mcasp_ahclkr_mux	11	1	
Main_MCASP_1	mcasp_ahclkx_pin	HFOSC1	1	mcasp_ahclkx_mux	0	1	[19.2 - 27]
Main_MCASP_1	mcasp_ahclkx_pin	HFOSC0	1	mcasp_ahclkx_mux	1	1	[19.2, 20, 24, 25, 26, 27]
Main_MCASP_1	mcasp_ahclkx_pin	AUDIO_EXT_REFCLK0	1	mcasp_ahclkx_mux	2	1	
Main_MCASP_1	mcasp_ahclkx_pin	AUDIO_EXT_REFCLK1	1	mcasp_ahclkx_mux	3	1	
Main_MCASP_1	mcasp_ahclkx_pin	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT	1	mcasp_ahclkx_mux	8	1	
Main_MCASP_1	mcasp_ahclkx_pin	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT	1	mcasp_ahclkx_mux	9	1	

Copyright © 2024 Texas Instruments Incorporated

Table 5-33. Clock Mapping (continued)

Module	Clock Port	Clock	Divider	Control Register	Mux Value	Default Mux Value	Typical Frequency
Main_MCASP_1	mcasp_ahclkx_pin	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT	1	mcasp_ahclkx_mux	10	1	
Main_MCASP_1	mcasp_ahclkx_pin	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT	1	mcasp_ahclkx_mux	11	1	
Main_MCASP_1	vbusp_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	2				250
Main_MCASP_2	aux_clk	MAIN_PLL4.HSDIV0	1	McASP_AUXCLK_SEL	0	2	197
Main_MCASP_2	aux_clk	MAIN_PLL2.HSDIV2	1	McASP_AUXCLK_SEL	1	2	200
Main_MCASP_2	aux_clk	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT	1	McASP_AUXCLK_SEL	4	2	
Main_MCASP_2	aux_clk	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT	1	McASP_AUXCLK_SEL	5	2	
Main_MCASP_2	aux_clk	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT	1	McASP_AUXCLK_SEL	6	2	
Main_MCASP_2	aux_clk	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT	1	McASP_AUXCLK_SEL	7	2	
Main_MCASP_2	mcasp_aclkr_pin	MCASP2_ACLKR	1				
Main_MCASP_2	mcasp_aclkx_pin	MCASP2_ACLKX	1				
Main_MCASP_2	mcasp_ahclkr_pin	HFOSC1	1	mcasp_ahclkr_mux	0	2	[19.2 - 27]
Main_MCASP_2	mcasp_ahclkr_pin	HFOSC0	1	mcasp_ahclkr_mux	1	2	[19.2, 20, 24, 25, 26, 27]
Main_MCASP_2	mcasp_ahclkr_pin	AUDIO_EXT_REFCLK0	1	mcasp_ahclkr_mux	2	2	
Main_MCASP_2	mcasp_ahclkr_pin	AUDIO_EXT_REFCLK1	1	mcasp_ahclkr_mux	3	2	
Main_MCASP_2	mcasp_ahclkr_pin	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT	1	mcasp_ahclkr_mux	8	2	
Main_MCASP_2	mcasp_ahclkr_pin	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT	1	mcasp_ahclkr_mux	9	2	
Main_MCASP_2	mcasp_ahclkr_pin	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT	1	mcasp_ahclkr_mux	10	2	
Main_MCASP_2	mcasp_ahclkr_pin	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT	1	mcasp_ahclkr_mux	11	2	
Main_MCASP_2	mcasp_ahclkx_pin	HFOSC1	1	mcasp_ahclkx_mux	0	2	[19.2 - 27]
Main_MCASP_2	mcasp_ahclkx_pin	HFOSC0	1	mcasp_ahclkx_mux	1	2	[19.2, 20, 24, 25, 26, 27]
Main_MCASP_2	mcasp_ahclkx_pin	AUDIO_EXT_REFCLK0	1	mcasp_ahclkx_mux	2	2	
Main_MCASP_2	mcasp_ahclkx_pin	AUDIO_EXT_REFCLK1	1	mcasp_ahclkx_mux	3	2	
Main_MCASP_2	mcasp_ahclkx_pin	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT	1	mcasp_ahclkx_mux	8	2	
Main_MCASP_2	mcasp_ahclkx_pin	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT	1	mcasp_ahclkx_mux	9	2	
Main_MCASP_2	mcasp_ahclkx_pin	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT	1	mcasp_ahclkx_mux	10	2	
Main_MCASP_2	mcasp_ahclkx_pin	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT	1	mcasp_ahclkx_mux	11	2	
Main_MCASP_2	vbusp_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	2				250
Main_MCASP_3	aux_clk	MAIN_PLL4.HSDIV0	1	McASP_AUXCLK_SEL	0	3	197
Main_MCASP_3	aux_clk	MAIN_PLL2.HSDIV2	1	McASP_AUXCLK_SEL	1	3	200
Main_MCASP_3	aux_clk	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT	1	McASP_AUXCLK_SEL	4	3	
Main_MCASP_3	aux_clk	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT	1	McASP_AUXCLK_SEL	5	3	
Main_MCASP_3	aux_clk	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT	1	McASP_AUXCLK_SEL	6	3	
Main_MCASP_3	aux_clk	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT	1	McASP_AUXCLK_SEL	7	3	
Main_MCASP_3	mcasp_aclkr_pin	MCASP3_ACLKR	1				
Main_MCASP_3	mcasp_aclkx_pin	MCASP3_ACLKX	1				
Main_MCASP_3	mcasp_ahclkr_pin	HFOSC1	1	mcasp_ahclkr_mux	0	3	[19.2 - 27]
Main_MCASP_3	mcasp_ahclkr_pin	HFOSC0	1	mcasp_ahclkr_mux	1	3	[19.2, 20, 24, 25, 26, 27]
Main_MCASP_3	mcasp_ahclkr_pin	AUDIO_EXT_REFCLK0	1	mcasp_ahclkr_mux	2	3	
Main_MCASP_3	mcasp_ahclkr_pin	AUDIO_EXT_REFCLK1	1	mcasp_ahclkr_mux	3	3	

Copyright © 2024 Texas Instruments Incorporated

Table 5-33. Clock Mapping (continued)

Module	Clock Port	Clock	Divider	Control Register	Mux Value	Default Mux Value	Typical Frequency
Main_MCASP_3	mcasp_ahclkr_pin	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT	1	mcasp_ahclkr_mux	8	3	
Main_MCASP_3	mcasp_ahclkr_pin	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT	1	mcasp_ahclkr_mux	9	3	
Main_MCASP_3	mcasp_ahclkr_pin	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT	1	mcasp_ahclkr_mux	10	3	
Main_MCASP_3	mcasp_ahclkr_pin	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT	1	mcasp_ahclkr_mux	11	3	
Main_MCASP_3	mcasp_ahclkx_pin	HFOSC1	1	mcasp_ahclkx_mux	0	3	[19.2 - 27]
Main_MCASP_3	mcasp_ahclkx_pin	HFOSC0	1	mcasp_ahclkx_mux	1	3	[19.2, 20, 24, 25, 26, 27]
Main_MCASP_3	mcasp_ahclkx_pin	AUDIO_EXT_REFCLK0	1	mcasp_ahclkx_mux	2	3	
Main_MCASP_3	mcasp_ahclkx_pin	AUDIO_EXT_REFCLK1	1	mcasp_ahclkx_mux	3	3	
Main_MCASP_3	mcasp_ahclkx_pin	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT	1	mcasp_ahclkx_mux	8	3	
Main_MCASP_3	mcasp_ahclkx_pin	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT	1	mcasp_ahclkx_mux	9	3	
Main_MCASP_3	mcasp_ahclkx_pin	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT	1	mcasp_ahclkx_mux	10	3	
Main_MCASP_3	mcasp_ahclkx_pin	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT	1	mcasp_ahclkx_mux	11	3	
Main_MCASP_3	vbusp_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	2				250
Main_MCASP_4	aux_clk	MAIN_PLL4.HSDIV0	1	McASP_AUXCLK_SEL	0	4	197
Main_MCASP_4	aux_clk	MAIN_PLL2.HSDIV2	1	McASP_AUXCLK_SEL	1	4	200
Main_MCASP_4	aux_clk	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT	1	McASP_AUXCLK_SEL	4	4	
Main_MCASP_4	aux_clk	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT	1	McASP_AUXCLK_SEL	5	4	
Main_MCASP_4	aux_clk	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT	1	McASP_AUXCLK_SEL	6	4	
Main_MCASP_4	aux_clk	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT	1	McASP_AUXCLK_SEL	7	4	
Main_MCASP_4	mcasp_aclkr_pin	MCASP4_ACLKR	1				
Main_MCASP_4	mcasp_aclkx_pin	MCASP4_ACLKX	1				
Main_MCASP_4	mcasp_ahclkr_pin	HFOSC1	1	mcasp_ahclkr_mux	0	4	[19.2 - 27]
Main_MCASP_4	mcasp_ahclkr_pin	HFOSC0	1	mcasp_ahclkr_mux	1	4	[19.2, 20, 24, 25, 26, 27]
Main_MCASP_4	mcasp_ahclkr_pin	AUDIO_EXT_REFCLK0	1	mcasp_ahclkr_mux	2	4	
Main_MCASP_4	mcasp_ahclkr_pin	AUDIO_EXT_REFCLK1	1	mcasp_ahclkr_mux	3	4	
Main_MCASP_4	mcasp_ahclkr_pin	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT	1	mcasp_ahclkr_mux	8	4	
Main_MCASP_4	mcasp_ahclkr_pin	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT	1	mcasp_ahclkr_mux	9	4	
Main_MCASP_4	mcasp_ahclkr_pin	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT	1	mcasp_ahclkr_mux	10	4	
Main_MCASP_4	mcasp_ahclkr_pin	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT	1	mcasp_ahclkr_mux	11	4	
Main_MCASP_4	mcasp_ahclkx_pin	HFOSC1	1	mcasp_ahclkx_mux	0	4	[19.2 - 27]
Main_MCASP_4	mcasp_ahclkx_pin	HFOSC0	1	mcasp_ahclkx_mux	1	4	[19.2, 20, 24, 25, 26, 27]
Main_MCASP_4	mcasp_ahclkx_pin	AUDIO_EXT_REFCLK0	1	mcasp_ahclkx_mux	2	4	
Main_MCASP_4	mcasp_ahclkx_pin	AUDIO_EXT_REFCLK1	1	mcasp_ahclkx_mux	3	4	
Main_MCASP_4	mcasp_ahclkx_pin	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT	1	mcasp_ahclkx_mux	8	4	
Main_MCASP_4	mcasp_ahclkx_pin	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT	1	mcasp_ahclkx_mux	9	4	
Main_MCASP_4	mcasp_ahclkx_pin	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT	1	mcasp_ahclkx_mux	10	4	
Main_MCASP_4	mcasp_ahclkx_pin	MAIN_ATL_0.ATL_IO_PORT_ATCLK_OUT	1	mcasp_ahclkx_mux	11	4	
Main_MCASP_4	vbusp_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	2				250

Copyright © 2024 Texas Instruments Incorporated

Table 5-33. Clock Mapping (continued)

Module	Clock Port	Clock	Divider	Control Register	Mux Value	Default Mux Value	Typical Frequency
Main_NAVSS_0	cpts_rclk_clk	MAIN_PLL3.HSDIV1	1	NAVSS_CPTS_RCLK_SEL	0	0	250
Main_NAVSS_0	cpts_rclk_clk	MAIN_PLL0.HSDIV6	1	NAVSS_CPTS_RCLK_SEL	1	0	250
Main_NAVSS_0	cpts_rclk_clk	MCU_CPTS0_RFT_CLK	1	NAVSS_CPTS_RCLK_SEL	2	0	
Main_NAVSS_0	cpts_rclk_clk	CPTS0_RFT_CLK	1	NAVSS_CPTS_RCLK_SEL	3	0	
Main_NAVSS_0	cpts_rclk_clk	MCU_EXT_REFCLK0	1	NAVSS_CPTS_RCLK_SEL	4	0	100
Main_NAVSS_0	cpts_rclk_clk	EXT_REFCLK1	1	NAVSS_CPTS_RCLK_SEL	5	0	
Main_NAVSS_0	cpts_rclk_clk	MAIN_SERDES_0.IP2_LN0_TXMCLK	1	NAVSS_CPTS_RCLK_SEL	6	0	
Main_NAVSS_0	cpts_rclk_clk	MAIN_SERDES_0.IP2_LN1_TXMCLK	1	NAVSS_CPTS_RCLK_SEL	7	0	
Main_NAVSS_0	cpts_rclk_clk	MAIN_SERDES_0.IP2_LN2_TXMCLK	1	NAVSS_CPTS_RCLK_SEL	8	0	
Main_NAVSS_0	cpts_rclk_clk	MAIN_SERDES_0.IP2_LN3_TXMCLK	1	NAVSS_CPTS_RCLK_SEL	9	0	
Main_NAVSS_0	cpts_rclk_clk	MCU_PLL2.HSDIV1	1	NAVSS_CPTS_RCLK_SEL	14	0	500
Main_NAVSS_0	cpts_rclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	1	NAVSS_CPTS_RCLK_SEL	15	0	500
Main_NAVSS_0	modss_vd2clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	1				500
Main_NAVSS_0	nbss_vclk	MAIN_PLL7.HSDIV0	1				1000
Main_NAVSS_0	nbss_vd2clk	MAIN_PLL7.HSDIV0	2				500
Main_NAVSS_0	pdma_main_debug_clk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	2				250
Main_NAVSS_0	pdma_main_misc_clk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_NAVSS_0	pdma_main_usart_clk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_NAVSS_0	udmass_vd2clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	1				500
Main_NAVSS_0	virtss_vd2clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	1				500
Main_NBSS_0	clk	MAIN_PLL7.HSDIV0	2				500
Main_PCIe_0	pcie_cba_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	2				250
Main_PCIe_0	pcie_cpts_rclk_clk	MAIN_PLL3.HSDIV1	1	PCIEn_CPTS_RCLK_MUX	0	1	250
Main_PCIe_0	pcie_cpts_rclk_clk	MAIN_PLL0.HSDIV6	1	PCIEn_CPTS_RCLK_MUX	1	1	250
Main_PCIe_0	pcie_cpts_rclk_clk	MCU_CPTS0_RFT_CLK	1	PCIEn_CPTS_RCLK_MUX	2	1	
Main_PCIe_0	pcie_cpts_rclk_clk	CPTS0_RFT_CLK	1	PCIEn_CPTS_RCLK_MUX	3	1	
Main_PCIe_0	pcie_cpts_rclk_clk	MCU_EXT_REFCLK0	1	PCIEn_CPTS_RCLK_MUX	4	1	100
Main_PCIe_0	pcie_cpts_rclk_clk	EXT_REFCLK1	1	PCIEn_CPTS_RCLK_MUX	5	1	
Main_PCIe_0	pcie_cpts_rclk_clk	MAIN_SERDES_0.IP2_LN0_TXMCLK	1	PCIEn_CPTS_RCLK_MUX	6	1	
Main_PCIe_0	pcie_cpts_rclk_clk	MAIN_SERDES_0.IP2_LN1_TXMCLK	1	PCIEn_CPTS_RCLK_MUX	7	1	
Main_PCIe_0	pcie_cpts_rclk_clk	MAIN_SERDES_0.IP2_LN2_TXMCLK	1	PCIEn_CPTS_RCLK_MUX	8	1	
Main_PCIe_0	pcie_cpts_rclk_clk	MAIN_SERDES_0.IP2_LN3_TXMCLK	1	PCIEn_CPTS_RCLK_MUX	9	1	
Main_PCIe_0	pcie_cpts_rclk_clk	MCU_PLL2.HSDIV1	1	PCIEn_CPTS_RCLK_MUX	14	1	500
Main_PCIe_0	pcie_cpts_rclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	1	PCIEn_CPTS_RCLK_MUX	15	1	500
Main_PCIe_0	pcie_lane0_refclk	MAIN_SERDES_0.IP2_LN0_REFCLK	1				
Main_PCIe_0	pcie_lane0_rxclk	MAIN_SERDES_0.IP2_LN0_RXCLK	1				
Main_PCIe_0	pcie_lane0_rxfclk	MAIN_SERDES_0.IP2_LN0_RXFCLK	1				

Copyright © 2024 Texas Instruments Incorporated

Table 5-33. Clock Mapping (continued)

Module	Clock Port	Clock	Divider	Control Register	Mux Value	Default Mux Value	Typical Frequency
Main_PCIe_0	pcie_lane0_txfclk	MAIN_SERDES_0.IP2_LN0_TXFCLK	1				
Main_PCIe_0	pcie_lane0_txmclk	MAIN_SERDES_0.IP2_LN0_TXMCLK	1				
Main_PCIe_0	pcie_lane1_refclk	MAIN_SERDES_0.IP2_LN1_REFCLK	1				
Main_PCIe_0	pcie_lane1_rxclk	MAIN_SERDES_0.IP2_LN1_RXCLK	1				
Main_PCIe_0	pcie_lane1_rxfclk	MAIN_SERDES_0.IP2_LN1_RXFCLK	1				
Main_PCIe_0	pcie_lane1_txfclk	MAIN_SERDES_0.IP2_LN1_TXFCLK	1				
Main_PCIe_0	pcie_lane1_txmclk	MAIN_SERDES_0.IP2_LN1_TXMCLK	1				
Main_PCIe_0	pcie_lane2_refclk	MAIN_SERDES_0.IP2_LN2_REFCLK	1				
Main_PCIe_0	pcie_lane2_rxclk	MAIN_SERDES_0.IP2_LN2_RXCLK	1				
Main_PCIe_0	pcie_lane2_rxfclk	MAIN_SERDES_0.IP2_LN2_RXFCLK	1				
Main_PCIe_0	pcie_lane2_txfclk	MAIN_SERDES_0.IP2_LN2_TXFCLK	1				
Main_PCIe_0	pcie_lane2_txmclk	MAIN_SERDES_0.IP2_LN2_TXMCLK	1				
Main_PCIe_0	pcie_lane3_refclk	MAIN_SERDES_0.IP2_LN3_REFCLK	1				
Main_PCIe_0	pcie_lane3_rxclk	MAIN_SERDES_0.IP2_LN3_RXCLK	1				
Main_PCIe_0	pcie_lane3_rxfclk	MAIN_SERDES_0.IP2_LN3_RXFCLK	1				
Main_PCIe_0	pcie_lane3_txfclk	MAIN_SERDES_0.IP2_LN3_TXFCLK	1				
Main_PCIe_0	pcie_lane3_txmclk	MAIN_SERDES_0.IP2_LN3_TXMCLK	1				
Main_PCIe_0	pcie_pm_clk	CLK_12M_RC	1				13
Main_PCIe_0	pcie_pm_clk	RCOSC	1				
Main_PDMA_CPSW_0	main_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_PDMA_Debug_0	main_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	2				250
Main_PDMA_Debug_G0	vclk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	2				250
Main_PDMA_Debug_G1	vclk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	2				250
Main_PDMA_MCAN_0	vclk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_PDMA_McASP_0	vclk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	2				250
Main_PDMA_SPI_G0	vclk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_PDMA_SPI_G1	vclk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_PDMA_UART_0	main_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_PDMA_UART_G0	vclk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_PDMA_UART_G1	vclk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_PDMA_UART_G2	vclk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_Pulsar_0	cpu0_clk	MAIN_PLL14.HSDIV0	1				1000
Main_Pulsar_0	cpu1_clk	MAIN_PLL14.HSDIV0	1				1000
Main_Pulsar_0	interface0_clk	MAIN_PLL14.HSDIV0	1				1000
Main_Pulsar_0	interface0_phase	MAIN_TIEOFF HIGH	1				
Main_Pulsar_0	interface1_clk	MAIN_PLL14.HSDIV0	1				1000
Main_Pulsar_0	interface1_phase	MAIN_TIEOFF HIGH	1				
Main_Pulsar_1	cpu0_clk	MAIN_PLL14.HSDIV1	1				1000
Main_Pulsar_1	cpu1_clk	MAIN_PLL14.HSDIV1	1				1000
Main_Pulsar_1	interface0_clk	MAIN_PLL14.HSDIV1	1				1000
Main_Pulsar_1	interface0_phase	MAIN_TIEOFF HIGH	1				
Main_Pulsar_1	interface1_clk	MAIN_PLL14.HSDIV1	1				1000
Main_Pulsar_1	interface1_phase	MAIN_TIEOFF HIGH	1				
Main_RTI_A72_0	rti_clk	HFOSC0	1	MAIN_WWDT_RTICLK_SEL	0	0	[19.2, 20, 24, 25, 26, 27]
Main_RTI_A72_0	rti_clk	LFXOSC	1	MAIN_WWDT_RTICLK_SEL	1	0	0.032768
Main_RTI_A72_0	rti_clk	RCOSC	1	MAIN_WWDT_RTICLK_SEL	2	0	

Copyright © 2024 Texas Instruments Incorporated

Table 5-33. Clock Mapping (continued)

Module	Clock Port	Clock	Divider	Control Register	Mux Value	Default Mux Value	Typical Frequency
Main_RTI_A72_0	rti_clk	CLK_32K	1	MAIN_WWDT_RTICLK_SEL	3	0	0.032000
Main_RTI_A72_0	rti_clk	HFOSC1	1	MAIN_WWDT_RTICLK_SEL	4	0	[19.2 - 27]
Main_RTI_A72_0	rti_clk	HFOSC1	1	MAIN_WWDT_RTICLK_SEL	5	0	[19.2 - 27]
Main_RTI_A72_0	rti_clk	HFOSC1	1	MAIN_WWDT_RTICLK_SEL	6	0	[19.2 - 27]
Main_RTI_A72_0	rti_clk	HFOSC1	1	MAIN_WWDT_RTICLK_SEL	7	0	[19.2 - 27]
Main_RTI_A72_0	vbusp_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_RTI_A72_1	rti_clk	HFOSC0	1	MAIN_WWDT_RTICLK_SEL	0	1	[19.2, 20, 24, 25, 26, 27]
Main_RTI_A72_1	rti_clk	LFXOSC	1	MAIN_WWDT_RTICLK_SEL	1	1	0.032768
Main_RTI_A72_1	rti_clk	RCOSC	1	MAIN_WWDT_RTICLK_SEL	2	1	
Main_RTI_A72_1	rti_clk	CLK_32K	1	MAIN_WWDT_RTICLK_SEL	3	1	0.032000
Main_RTI_A72_1	rti_clk	HFOSC1	1	MAIN_WWDT_RTICLK_SEL	4	1	[19.2 - 27]
Main_RTI_A72_1	rti_clk	HFOSC1	1	MAIN_WWDT_RTICLK_SEL	5	1	[19.2 - 27]
Main_RTI_A72_1	rti_clk	HFOSC1	1	MAIN_WWDT_RTICLK_SEL	6	1	[19.2 - 27]
Main_RTI_A72_1	rti_clk	HFOSC1	1	MAIN_WWDT_RTICLK_SEL	7	1	[19.2 - 27]
Main_RTI_A72_1	vbusp_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_SerDes_0	clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_SerDes_0	dphy_ref_clk	HFOSC0	1	DPHY_CLK_SEL	0	0	[19.2, 20, 24, 25, 26, 27]
Main_SerDes_0	dphy_ref_clk	HFOSC1	1	DPHY_CLK_SEL	1	0	[19.2 - 27]
Main_SerDes_0	dphy_ref_clk	MAIN_PLL3.HSDIV4	1	DPHY_CLK_SEL	2	0	156
Main_SerDes_0	dphy_ref_clk	MAIN_PLL2.HSDIV4	1	DPHY_CLK_SEL	3	0	100
Main_SerDes_0	ip1_ppi_M_TxClkEsc_clk	MAIN_PLL1.HSDIV8	1				20
Main_SerDes_0	ip2_ppi_M_TxClkEsc_clk	MAIN_PLL1.HSDIV8	1				20
Main_SerDes_0	ip3_ppi_M_TxClkEsc_clk	MAIN_TIEOFF LOW	1				
Main_SerDes_0	ip4_ppi_M_TxClkEsc_clk	MAIN_TIEOFF LOW	1				
Main_SerDes_0	psm_clk	MAIN_PLL1.HSDIV8	1				20
Main_SerDes_0	tap_tck	MAIN_TAP_BS_JTAG__CLK	1				
Main_RTI_C7x_0	rti_clk	HFOSC0	1	MAIN_WWDT_RTICLK_SEL	0	16	[19.2, 20, 24, 25, 26, 27]
Main_RTI_C7x_0	rti_clk	LFXOSC	1	MAIN_WWDT_RTICLK_SEL	1	16	0.032768
Main_RTI_C7x_0	rti_clk	RCOSC	1	MAIN_WWDT_RTICLK_SEL	2	16	
Main_RTI_C7x_0	rti_clk	CLK_32K	1	MAIN_WWDT_RTICLK_SEL	3	16	0.032000
Main_RTI_C7x_0	rti_clk	HFOSC1	1	MAIN_WWDT_RTICLK_SEL	4	16	[19.2 - 27]
Main_RTI_C7x_0	rti_clk	HFOSC1	1	MAIN_WWDT_RTICLK_SEL	5	16	[19.2 - 27]
Main_RTI_C7x_0	rti_clk	HFOSC1	1	MAIN_WWDT_RTICLK_SEL	6	16	[19.2 - 27]
Main_RTI_C7x_0	rti_clk	HFOSC1	1	MAIN_WWDT_RTICLK_SEL	7	16	[19.2 - 27]
Main_RTI_C7x_0	vbusp_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125

Copyright © 2024 Texas Instruments Incorporated

Table 5-33. Clock Mapping (continued)

Module	Clock Port	Clock	Divider	Control Register	Mux Value	Default Mux Value	Typical Frequency
Main_SerDes_1	clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_SerDes_1	dphy_ref_clk	HFOSC0	1	DPHY_CLK_SEL	0	1	[19.2, 20, 24, 25, 26, 27]
Main_SerDes_1	dphy_ref_clk	HFOSC1	1	DPHY_CLK_SEL	1	1	[19.2 - 27]
Main_SerDes_1	dphy_ref_clk	MAIN_PLL3.HSDIV4	1	DPHY_CLK_SEL	2	1	156
Main_SerDes_1	dphy_ref_clk	MAIN_PLL2.HSDIV4	1	DPHY_CLK_SEL	3	1	100
Main_SerDes_1	ip1_ppi_M_TxClkEsc_clk	MAIN_PLL1.HSDIV8	1				20
Main_SerDes_1	ip2_ppi_M_TxClkEsc_clk	MAIN_TIEOFF LOW	1				
Main_SerDes_1	ip3_ppi_M_TxClkEsc_clk	MAIN_TIEOFF LOW	1				
Main_SerDes_1	ip4_ppi_M_TxClkEsc_clk	MAIN_TIEOFF LOW	1				
Main_SerDes_1	psm_clk	MAIN_PLL1.HSDIV8	1				20
Main_SerDes_1	tap_tck	MAIN_TAP_BS_JTAG__CLK	1				
Main_RTI_C7x_1	rti_clk	HFOSC0	1	MAIN_WWDT_RTICLK_SEL	0	17	[19.2, 20, 24, 25, 26, 27]
Main_RTI_C7x_1	rti_clk	LFXOSC	1	MAIN_WWDT_RTICLK_SEL	1	17	0.032768
Main_RTI_C7x_1	rti_clk	RCOSC	1	MAIN_WWDT_RTICLK_SEL	2	17	
Main_RTI_C7x_1	rti_clk	CLK_32K	1	MAIN_WWDT_RTICLK_SEL	3	17	0.032000
Main_RTI_C7x_1	rti_clk	HFOSC1	1	MAIN_WWDT_RTICLK_SEL	4	17	[19.2 - 27]
Main_RTI_C7x_1	rti_clk	HFOSC1	1	MAIN_WWDT_RTICLK_SEL	5	17	[19.2 - 27]
Main_RTI_C7x_1	rti_clk	HFOSC1	1	MAIN_WWDT_RTICLK_SEL	6	17	[19.2 - 27]
Main_RTI_C7x_1	rti_clk	HFOSC1	1	MAIN_WWDT_RTICLK_SEL	7	17	[19.2 - 27]
Main_RTI_C7x_1	vbusp_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_SPI_0	clkspiref_clk	MAIN_PLL0.HSDIV5	1				50
Main_SPI_0	io_clkspii_clk	SPI0_CLK	1	SPI0_CLK_LPBK_MUX	0	0	
Main_SPI_0	io_clkspii_clk	MAIN_SPI_0.IO_CLKSPIO_CLK	1	SPI0_CLK_LPBK_MUX	1	0	
Main_SPI_0	vbusp_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_RTI_GPU	rti_clk	HFOSC0	1	MAIN_WWDT_RTICLK_SEL	0	15	[19.2, 20, 24, 25, 26, 27]
Main_RTI_GPU	rti_clk	LFXOSC	1	MAIN_WWDT_RTICLK_SEL	1	15	0.032768
Main_RTI_GPU	rti_clk	RCOSC	1	MAIN_WWDT_RTICLK_SEL	2	15	
Main_RTI_GPU	rti_clk	CLK_32K	1	MAIN_WWDT_RTICLK_SEL	3	15	0.032000
Main_RTI_GPU	rti_clk	HFOSC1	1	MAIN_WWDT_RTICLK_SEL	4	15	[19.2 - 27]
Main_RTI_GPU	rti_clk	HFOSC1	1	MAIN_WWDT_RTICLK_SEL	5	15	[19.2 - 27]
Main_RTI_GPU	rti_clk	HFOSC1	1	MAIN_WWDT_RTICLK_SEL	6	15	[19.2 - 27]
Main_RTI_GPU	rti_clk	HFOSC1	1	MAIN_WWDT_RTICLK_SEL	7	15	[19.2 - 27]
Main_RTI_GPU	vbusp_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_SPI_1	clkspiref_clk	MAIN_PLL0.HSDIV5	1				50
Main_SPI_1	io_clkspii_clk	SPI1_CLK	1	SPI1_CLK_LPBK_MUX	0	0	
Main_SPI_1	io_clkspii_clk	MAIN_SPI_1.IO_CLKSPIO_CLK	1	SPI1_CLK_LPBK_MUX	1	0	
Main_SPI_1	vbusp_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125

Copyright © 2024 Texas Instruments Incorporated

Table 5-33. Clock Mapping (continued)

Module	Clock Port	Clock	Divider	Control Register	Mux Value	Default Mux Value	Typical Frequency
Main_RTI_Pulsar_R5F_0_0	rti_clk	HFOSC0	1	MAIN_WWDT_RTICLK_SEL	0	28	[19.2, 20, 24, 25, 26, 27]
Main_RTI_Pulsar_R5F_0_0	rti_clk	LFXOSC	1	MAIN_WWDT_RTICLK_SEL	1	28	0.032768
Main_RTI_Pulsar_R5F_0_0	rti_clk	RCOSC	1	MAIN_WWDT_RTICLK_SEL	2	28	
Main_RTI_Pulsar_R5F_0_0	rti_clk	CLK_32K	1	MAIN_WWDT_RTICLK_SEL	3	28	0.032000
Main_RTI_Pulsar_R5F_0_0	rti_clk	HFOSC1	1	MAIN_WWDT_RTICLK_SEL	4	28	[19.2 - 27]
Main_RTI_Pulsar_R5F_0_0	rti_clk	HFOSC1	1	MAIN_WWDT_RTICLK_SEL	5	28	[19.2 - 27]
Main_RTI_Pulsar_R5F_0_0	rti_clk	HFOSC1	1	MAIN_WWDT_RTICLK_SEL	6	28	[19.2 - 27]
Main_RTI_Pulsar_R5F_0_0	rti_clk	HFOSC1	1	MAIN_WWDT_RTICLK_SEL	7	28	[19.2 - 27]
Main_RTI_Pulsar_R5F_0_0	vbusp_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_SPI_2	clkspiref_clk	MAIN_PLL0.HSDIV5	1				50
Main_SPI_2	io_clkspii_clk	SPI2_CLK	1	SPI2_CLK_LPBK_MUX	0	0	
Main_SPI_2	io_clkspii_clk	MAIN_SPI_2.IO_CLKSPIO_CLK	1	SPI2_CLK_LPBK_MUX	1	0	
Main_SPI_2	vbusp_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_RTI_Pulsar_R5F_0_1	rti_clk	HFOSC0	1	MAIN_WWDT_RTICLK_SEL	0	29	[19.2, 20, 24, 25, 26, 27]
Main_RTI_Pulsar_R5F_0_1	rti_clk	LFXOSC	1	MAIN_WWDT_RTICLK_SEL	1	29	0.032768
Main_RTI_Pulsar_R5F_0_1	rti_clk	RCOSC	1	MAIN_WWDT_RTICLK_SEL	2	29	
Main_RTI_Pulsar_R5F_0_1	rti_clk	CLK_32K	1	MAIN_WWDT_RTICLK_SEL	3	29	0.032000
Main_RTI_Pulsar_R5F_0_1	rti_clk	HFOSC1	1	MAIN_WWDT_RTICLK_SEL	4	29	[19.2 - 27]
Main_RTI_Pulsar_R5F_0_1	rti_clk	HFOSC1	1	MAIN_WWDT_RTICLK_SEL	5	29	[19.2 - 27]
Main_RTI_Pulsar_R5F_0_1	rti_clk	HFOSC1	1	MAIN_WWDT_RTICLK_SEL	6	29	[19.2 - 27]
Main_RTI_Pulsar_R5F_0_1	rti_clk	HFOSC1	1	MAIN_WWDT_RTICLK_SEL	7	29	[19.2 - 27]
Main_RTI_Pulsar_R5F_0_1	vbusp_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_SPI_3	clkspiref_clk	MAIN_PLL0.HSDIV5	1				50
Main_SPI_3	io_clkspii_clk	SPI3_CLK	1	SPI3_CLK_LPBK_MUX	0	0	
Main_SPI_3	io_clkspii_clk	MAIN_SPI_3.IO_CLKSPIO_CLK	1	SPI3_CLK_LPBK_MUX	1	0	
Main_SPI_3	io_clkspii_clk	MAIN_SPI_3.IO_CLKSPIO_CLK	1	SPI3_CLK_MUX	0	0	
Main_SPI_3	io_clkspii_clk	MAIN_SPI_3.IO_CLKSPIO_CLK	1	SPI3_CLK_MUX	1	0	
Main_SPI_3	io_clkspii_clk	SPI3_CLK	1	SPI3_CLK_MUX	1	0	
Main_SPI_3	vbusp_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_RTI_Pulsar_R5F_1_0	rti_clk	HFOSC0	1	MAIN_WWDT_RTICLK_SEL	0	30	[19.2, 20, 24, 25, 26, 27]
Main_RTI_Pulsar_R5F_1_0	rti_clk	LFXOSC	1	MAIN_WWDT_RTICLK_SEL	1	30	0.032768
Main_RTI_Pulsar_R5F_1_0	rti_clk	RCOSC	1	MAIN_WWDT_RTICLK_SEL	2	30	
Main_RTI_Pulsar_R5F_1_0	rti_clk	CLK_32K	1	MAIN_WWDT_RTICLK_SEL	3	30	0.032000
Main_RTI_Pulsar_R5F_1_0	rti_clk	HFOSC1	1	MAIN_WWDT_RTICLK_SEL	4	30	[19.2 - 27]
Main_RTI_Pulsar_R5F_1_0	rti_clk	HFOSC1	1	MAIN_WWDT_RTICLK_SEL	5	30	[19.2 - 27]
Main_RTI_Pulsar_R5F_1_0	rti_clk	HFOSC1	1	MAIN_WWDT_RTICLK_SEL	6	30	[19.2 - 27]
Main_RTI_Pulsar_R5F_1_0	rti_clk	HFOSC1	1	MAIN_WWDT_RTICLK_SEL	7	30	[19.2 - 27]

Copyright © 2024 Texas Instruments Incorporated

Table 5-33. Clock Mapping (continued)

Module	Clock Port	Clock	Divider	Control Register	Mux Value	Default Mux Value	Typical Frequency
Main_RTI_Pulsar_R5F_1_0	vbusp_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_SPI_4	clkspiref_clk	MAIN_PLL0.HSDIV5	1				50
Main_SPI_4	io_clkspii_clk	MAIN_SPI_4.IO_CLKSPIO_CLK	1				
Main_SPI_4	vbusp_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_RTI_Pulsar_R5F_1_1	rti_clk	HFOSC0	1	MAIN_WWDT_RTICLK_SEL	0	31	[19.2, 20, 24, 25, 26, 27]
Main_RTI_Pulsar_R5F_1_1	rti_clk	LFXOSC	1	MAIN_WWDT_RTICLK_SEL	1	31	0.032768
Main_RTI_Pulsar_R5F_1_1	rti_clk	RCOSC	1	MAIN_WWDT_RTICLK_SEL	2	31	
Main_RTI_Pulsar_R5F_1_1	rti_clk	CLK_32K	1	MAIN_WWDT_RTICLK_SEL	3	31	0.032000
Main_RTI_Pulsar_R5F_1_1	rti_clk	HFOSC1	1	MAIN_WWDT_RTICLK_SEL	4	31	[19.2 - 27]
Main_RTI_Pulsar_R5F_1_1	rti_clk	HFOSC1	1	MAIN_WWDT_RTICLK_SEL	5	31	[19.2 - 27]
Main_RTI_Pulsar_R5F_1_1	rti_clk	HFOSC1	1	MAIN_WWDT_RTICLK_SEL	6	31	[19.2 - 27]
Main_RTI_Pulsar_R5F_1_1	rti_clk	HFOSC1	1	MAIN_WWDT_RTICLK_SEL	7	31	[19.2 - 27]
Main_RTI_Pulsar_R5F_1_1	vbusp_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_SPI_5	clkspiref_clk	MAIN_PLL0.HSDIV5	1				50
Main_SPI_5	io_clkspii_clk	SPI5_CLK	1	SPI5_CLK_LPBK_MUX	0	0	
Main_SPI_5	io_clkspii_clk	MAIN_SPI_5.IO_CLKSPIO_CLK	1	SPI5_CLK_LPBK_MUX	1	0	
Main_SPI_5	vbusp_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_SA2_UL_0	pka_in_clk	MAIN_PLL0.HSDIV1	1				400
Main_SA2_UL_0	x1_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_SA2_UL_0	x2_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	2				250
Main_SPI_6	clkspiref_clk	MAIN_PLL0.HSDIV5	1				50
Main_SPI_6	io_clkspii_clk	SPI6_CLK	1	SPI6_CLK_LPBK_MUX	0	0	
Main_SPI_6	io_clkspii_clk	MAIN_SPI_6.IO_CLKSPIO_CLK	1	SPI6_CLK_LPBK_MUX	1	0	
Main_SPI_6	vbusp_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_SPI_7	clkspiref_clk	MAIN_PLL0.HSDIV5	1				50
Main_SPI_7	io_clkspii_clk	SPI7_CLK	1	SPI7_CLK_LPBK_MUX	0	0	
Main_SPI_7	io_clkspii_clk	MAIN_SPI_7.IO_CLKSPIO_CLK	1	SPI7_CLK_LPBK_MUX	1	0	
Main_SPI_7	vbusp_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_Timer_0	timer_hclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_Timer_0	timer_tclk_clk	HFOSC0	1	MAIN_TIMER_CLKSEL	0	0	[19.2, 20, 24, 25, 26, 27]
Main_Timer_0	timer_tclk_clk	HFOSC1	1	MAIN_TIMER_CLKSEL	1	0	[19.2 - 27]
Main_Timer_0	timer_tclk_clk	MAIN_PLL0.HSDIV8	1	MAIN_TIMER_CLKSEL	2	0	250
Main_Timer_0	timer_tclk_clk	RCOSC	1	MAIN_TIMER_CLKSEL	3	0	
Main_Timer_0	timer_tclk_clk	MAIN_PLL3.HSDIV3	1	MAIN_TIMER_CLKSEL	4	0	250
Main_Timer_0	timer_tclk_clk	MCU_EXT_REFCLK0	1	MAIN_TIMER_CLKSEL	5	0	100
Main_Timer_0	timer_tclk_clk	EXT_REFCLK1	1	MAIN_TIMER_CLKSEL	6	0	
Main_Timer_0	timer_tclk_clk	LFXOSC	1	MAIN_TIMER_CLKSEL	7	0	0.032768
Main_Timer_0	timer_tclk_clk	CPTS0_RFT_CLK	1	MAIN_TIMER_CLKSEL	8	0	
Main_Timer_0	timer_tclk_clk	MAIN_PLL1.HSDIV3	1	MAIN_TIMER_CLKSEL	9	0	192
Main_Timer_0	timer_tclk_clk	MAIN_PLL2.HSDIV6	1	MAIN_TIMER_CLKSEL	10	0	225
Main_Timer_0	timer_tclk_clk	MAIN_PLL4.HSDIV2	1	MAIN_TIMER_CLKSEL	11	0	197
Main_Timer_0	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF2	1	MAIN_TIMER_CLKSEL	12	0	
Main_Timer_0	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF3	1	MAIN_TIMER_CLKSEL	13	0	

Copyright © 2024 Texas Instruments Incorporated

Table 5-33. Clock Mapping (continued)

Module	Clock Port	Clock	Divider	Control Register	Mux Value	Default Mux Value	Typical Frequency
Main_Timer_0	timer_tclk_clk	MAIN_CPSW_0.CPTS_GENF0	1	MAIN_TIMER_CLKSEL	14	0	
Main_Timer_1	timer_hclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_Timer_1	timer_tclk_clk	HFOSC0	1	MAIN_TIMER_CLKSEL	0	1	[19.2, 20, 24, 25, 26, 27]
Main_Timer_1	timer_tclk_clk	HFOSC1	1	MAIN_TIMER_CLKSEL	1	1	[19.2 - 27]
Main_Timer_1	timer_tclk_clk	MAIN_PLL0.HSDIV8	1	MAIN_TIMER_CLKSEL	2	1	250
Main_Timer_1	timer_tclk_clk	RCOSC	1	MAIN_TIMER_CLKSEL	3	1	
Main_Timer_1	timer_tclk_clk	MAIN_PLL3.HSDIV3	1	MAIN_TIMER_CLKSEL	4	1	250
Main_Timer_1	timer_tclk_clk	MCU_EXT_REFCLK0	1	MAIN_TIMER_CLKSEL	5	1	100
Main_Timer_1	timer_tclk_clk	EXT_REFCLK1	1	MAIN_TIMER_CLKSEL	6	1	
Main_Timer_1	timer_tclk_clk	LFXOSC	1	MAIN_TIMER_CLKSEL	7	1	0.032768
Main_Timer_1	timer_tclk_clk	CPTS0_RFT_CLK	1	MAIN_TIMER_CLKSEL	8	1	
Main_Timer_1	timer_tclk_clk	MAIN_PLL1.HSDIV3	1	MAIN_TIMER_CLKSEL	9	1	192
Main_Timer_1	timer_tclk_clk	MAIN_PLL2.HSDIV6	1	MAIN_TIMER_CLKSEL	10	1	225
Main_Timer_1	timer_tclk_clk	MAIN_PLL4.HSDIV2	1	MAIN_TIMER_CLKSEL	11	1	197
Main_Timer_1	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF2	1	MAIN_TIMER_CLKSEL	12	1	
Main_Timer_1	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF3	1	MAIN_TIMER_CLKSEL	13	1	
Main_Timer_1	timer_tclk_clk	MAIN_CPSW_0.CPTS_GENF0	1	MAIN_TIMER_CLKSEL	14	1	
Main_Timer_1	timer_tclk_clk	LFXOSC	1	TIMER1_CASCADE	0	0	0.032768
Main_Timer_1	timer_tclk_clk	MCU_EXT_REFCLK0	1	TIMER1_CASCADE	0	0	100
Main_Timer_1	timer_tclk_clk	MAIN_PLL1.HSDIV3	1	TIMER1_CASCADE	0	0	192
Main_Timer_1	timer_tclk_clk	MAIN_PLL4.HSDIV2	1	TIMER1_CASCADE	0	0	197
Main_Timer_1	timer_tclk_clk	MAIN_PLL2.HSDIV6	1	TIMER1_CASCADE	0	0	225
Main_Timer_1	timer_tclk_clk	MAIN_PLL0.HSDIV8	1	TIMER1_CASCADE	0	0	250
Main_Timer_1	timer_tclk_clk	MAIN_PLL3.HSDIV3	1	TIMER1_CASCADE	0	0	250
Main_Timer_1	timer_tclk_clk	HFOSC1	1	TIMER1_CASCADE	0	0	[19.2 - 27]
Main_Timer_1	timer_tclk_clk	HFOSC0	1	TIMER1_CASCADE	0	0	[19.2, 20, 24, 25, 26, 27]
Main_Timer_1	timer_tclk_clk	CPTS0_RFT_CLK	1	TIMER1_CASCADE	0	0	
Main_Timer_1	timer_tclk_clk	EXT_REFCLK1	1	TIMER1_CASCADE	0	0	
Main_Timer_1	timer_tclk_clk	MAIN_CPSW_0.CPTS_GENF0	1	TIMER1_CASCADE	0	0	
Main_Timer_1	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF2	1	TIMER1_CASCADE	0	0	
Main_Timer_1	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF3	1	TIMER1_CASCADE	0	0	
Main_Timer_1	timer_tclk_clk	RCOSC	1	TIMER1_CASCADE	0	0	
Main_Timer_1	timer_tclk_clk	MAIN_TIMER_0.TIMER_PWM	1	TIMER1_CASCADE	1	0	
Main_Timer_2	timer_hclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_Timer_2	timer_tclk_clk	HFOSC0	1	MAIN_TIMER_CLKSEL	0	2	[19.2, 20, 24, 25, 26, 27]
Main_Timer_2	timer_tclk_clk	HFOSC1	1	MAIN_TIMER_CLKSEL	1	2	[19.2 - 27]
Main_Timer_2	timer_tclk_clk	MAIN_PLL0.HSDIV8	1	MAIN_TIMER_CLKSEL	2	2	250
Main_Timer_2	timer_tclk_clk	RCOSC	1	MAIN_TIMER_CLKSEL	3	2	
Main_Timer_2	timer_tclk_clk	MAIN_PLL3.HSDIV3	1	MAIN_TIMER_CLKSEL	4	2	250
Main_Timer_2	timer_tclk_clk	MCU_EXT_REFCLK0	1	MAIN_TIMER_CLKSEL	5	2	100
Main_Timer_2	timer_tclk_clk	EXT_REFCLK1	1	MAIN_TIMER_CLKSEL	6	2	
Main_Timer_2	timer_tclk_clk	LFXOSC	1	MAIN_TIMER_CLKSEL	7	2	0.032768
Main_Timer_2	timer_tclk_clk	CPTS0_RFT_CLK	1	MAIN_TIMER_CLKSEL	8	2	
Main_Timer_2	timer_tclk_clk	MAIN_PLL1.HSDIV3	1	MAIN_TIMER_CLKSEL	9	2	192
Main_Timer_2	timer_tclk_clk	MAIN_PLL2.HSDIV6	1	MAIN_TIMER_CLKSEL	10	2	225
Main_Timer_2	timer_tclk_clk	MAIN_PLL4.HSDIV2	1	MAIN_TIMER_CLKSEL	11	2	197
Main_Timer_2	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF2	1	MAIN_TIMER_CLKSEL	12	2	
Main_Timer_2	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF3	1	MAIN_TIMER_CLKSEL	13	2	
Main_Timer_2	timer_tclk_clk	MAIN_CPSW_0.CPTS_GENF0	1	MAIN_TIMER_CLKSEL	14	2	
Main_Timer_3	timer_hclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125

Copyright © 2024 Texas Instruments Incorporated

Table 5-33. Clock Mapping (continued)

Module	Clock Port	Clock	Divider	Control Register	Mux Value	Default Mux Value	Typical Frequency
Main_Timer_3	timer_tclk_clk	HFOSC0	1	MAIN_TIMER_CLKSEL	0	3	[19.2, 20, 24, 25, 26, 27]
Main_Timer_3	timer_tclk_clk	HFOSC1	1	MAIN_TIMER_CLKSEL	1	3	[19.2 - 27]
Main_Timer_3	timer_tclk_clk	MAIN_PLL0.HSDIV8	1	MAIN_TIMER_CLKSEL	2	3	250
Main_Timer_3	timer_tclk_clk	RCOSC	1	MAIN_TIMER_CLKSEL	3	3	
Main_Timer_3	timer_tclk_clk	MAIN_PLL3.HSDIV3	1	MAIN_TIMER_CLKSEL	4	3	250
Main_Timer_3	timer_tclk_clk	MCU_EXT_REFCLK0	1	MAIN_TIMER_CLKSEL	5	3	100
Main_Timer_3	timer_tclk_clk	EXT_REFCLK1	1	MAIN_TIMER_CLKSEL	6	3	
Main_Timer_3	timer_tclk_clk	LFXOSC	1	MAIN_TIMER_CLKSEL	7	3	0.032768
Main_Timer_3	timer_tclk_clk	CPTS0_RFT_CLK	1	MAIN_TIMER_CLKSEL	8	3	
Main_Timer_3	timer_tclk_clk	MAIN_PLL1.HSDIV3	1	MAIN_TIMER_CLKSEL	9	3	192
Main_Timer_3	timer_tclk_clk	MAIN_PLL2.HSDIV6	1	MAIN_TIMER_CLKSEL	10	3	225
Main_Timer_3	timer_tclk_clk	MAIN_PLL4.HSDIV2	1	MAIN_TIMER_CLKSEL	11	3	197
Main_Timer_3	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF2	1	MAIN_TIMER_CLKSEL	12	3	
Main_Timer_3	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF3	1	MAIN_TIMER_CLKSEL	13	3	
Main_Timer_3	timer_tclk_clk	MAIN_CPSW_0.CPTS_GENF0	1	MAIN_TIMER_CLKSEL	14	3	
Main_Timer_3	timer_tclk_clk	LFXOSC	1	TIMER3_CASCADE	0	0	0.032768
Main_Timer_3	timer_tclk_clk	MCU_EXT_REFCLK0	1	TIMER3_CASCADE	0	0	100
Main_Timer_3	timer_tclk_clk	MAIN_PLL1.HSDIV3	1	TIMER3_CASCADE	0	0	192
Main_Timer_3	timer_tclk_clk	MAIN_PLL4.HSDIV2	1	TIMER3_CASCADE	0	0	197
Main_Timer_3	timer_tclk_clk	MAIN_PLL2.HSDIV6	1	TIMER3_CASCADE	0	0	225
Main_Timer_3	timer_tclk_clk	MAIN_PLL0.HSDIV8	1	TIMER3_CASCADE	0	0	250
Main_Timer_3	timer_tclk_clk	MAIN_PLL3.HSDIV3	1	TIMER3_CASCADE	0	0	250
Main_Timer_3	timer_tclk_clk	HFOSC1	1	TIMER3_CASCADE	0	0	[19.2 - 27]
Main_Timer_3	timer_tclk_clk	HFOSC0	1	TIMER3_CASCADE	0	0	[19.2, 20, 24, 25, 26, 27]
Main_Timer_3	timer_tclk_clk	CPTS0_RFT_CLK	1	TIMER3_CASCADE	0	0	
Main_Timer_3	timer_tclk_clk	EXT_REFCLK1	1	TIMER3_CASCADE	0	0	
Main_Timer_3	timer_tclk_clk	MAIN_CPSW_0.CPTS_GENF0	1	TIMER3_CASCADE	0	0	
Main_Timer_3	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF2	1	TIMER3_CASCADE	0	0	
Main_Timer_3	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF3	1	TIMER3_CASCADE	0	0	
Main_Timer_3	timer_tclk_clk	RCOSC	1	TIMER3_CASCADE	0	0	
Main_Timer_3	timer_tclk_clk	MAIN_TIMER_2.TIMER_PWM	1	TIMER3_CASCADE	1	0	
Main_Timer_4	timer_hclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_Timer_4	timer_tclk_clk	HFOSC0	1	MAIN_TIMER_CLKSEL	0	4	[19.2, 20, 24, 25, 26, 27]
Main_Timer_4	timer_tclk_clk	HFOSC1	1	MAIN_TIMER_CLKSEL	1	4	[19.2 - 27]
Main_Timer_4	timer_tclk_clk	MAIN_PLL0.HSDIV8	1	MAIN_TIMER_CLKSEL	2	4	250
Main_Timer_4	timer_tclk_clk	RCOSC	1	MAIN_TIMER_CLKSEL	3	4	
Main_Timer_4	timer_tclk_clk	MAIN_PLL3.HSDIV3	1	MAIN_TIMER_CLKSEL	4	4	250
Main_Timer_4	timer_tclk_clk	MCU_EXT_REFCLK0	1	MAIN_TIMER_CLKSEL	5	4	100
Main_Timer_4	timer_tclk_clk	EXT_REFCLK1	1	MAIN_TIMER_CLKSEL	6	4	
Main_Timer_4	timer_tclk_clk	LFXOSC	1	MAIN_TIMER_CLKSEL	7	4	0.032768
Main_Timer_4	timer_tclk_clk	CPTS0_RFT_CLK	1	MAIN_TIMER_CLKSEL	8	4	
Main_Timer_4	timer_tclk_clk	MAIN_PLL1.HSDIV3	1	MAIN_TIMER_CLKSEL	9	4	192
Main_Timer_4	timer_tclk_clk	MAIN_PLL2.HSDIV6	1	MAIN_TIMER_CLKSEL	10	4	225
Main_Timer_4	timer_tclk_clk	MAIN_PLL4.HSDIV2	1	MAIN_TIMER_CLKSEL	11	4	197
Main_Timer_4	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF2	1	MAIN_TIMER_CLKSEL	12	4	
Main_Timer_4	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF3	1	MAIN_TIMER_CLKSEL	13	4	
Main_Timer_4	timer_tclk_clk	MAIN_CPSW_0.CPTS_GENF0	1	MAIN_TIMER_CLKSEL	14	4	
Main_Timer_5	timer_hclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_Timer_5	timer_tclk_clk	HFOSC0	1	MAIN_TIMER_CLKSEL	0	5	[19.2, 20, 24, 25, 26, 27]
Main_Timer_5	timer_tclk_clk	HFOSC1	1	MAIN_TIMER_CLKSEL	1	5	[19.2 - 27]

Copyright © 2024 Texas Instruments Incorporated

Table 5-33. Clock Mapping (continued)

Module	Clock Port	Clock	Divider	Control Register	Mux Value	Default Mux Value	Typical Frequency
Main_Timer_5	timer_tclk_clk	MAIN_PLL0.HSDIV8	1	MAIN_TIMER_CLKSEL	2	5	250
Main_Timer_5	timer_tclk_clk	RCOSC	1	MAIN_TIMER_CLKSEL	3	5	
Main_Timer_5	timer_tclk_clk	MAIN_PLL3.HSDIV3	1	MAIN_TIMER_CLKSEL	4	5	250
Main_Timer_5	timer_tclk_clk	MCU_EXT_REFCLK0	1	MAIN_TIMER_CLKSEL	5	5	100
Main_Timer_5	timer_tclk_clk	EXT_REFCLK1	1	MAIN_TIMER_CLKSEL	6	5	
Main_Timer_5	timer_tclk_clk	LFXOSC	1	MAIN_TIMER_CLKSEL	7	5	0.032768
Main_Timer_5	timer_tclk_clk	CPTS0_RFT_CLK	1	MAIN_TIMER_CLKSEL	8	5	
Main_Timer_5	timer_tclk_clk	MAIN_PLL1.HSDIV3	1	MAIN_TIMER_CLKSEL	9	5	192
Main_Timer_5	timer_tclk_clk	MAIN_PLL2.HSDIV6	1	MAIN_TIMER_CLKSEL	10	5	225
Main_Timer_5	timer_tclk_clk	MAIN_PLL4.HSDIV2	1	MAIN_TIMER_CLKSEL	11	5	197
Main_Timer_5	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF2	1	MAIN_TIMER_CLKSEL	12	5	
Main_Timer_5	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF3	1	MAIN_TIMER_CLKSEL	13	5	
Main_Timer_5	timer_tclk_clk	MAIN_CPSW_0.CPTS_GENF0	1	MAIN_TIMER_CLKSEL	14	5	
Main_Timer_5	timer_tclk_clk	LFXOSC	1	TIMER5_CASCADE	0	0	0.032768
Main_Timer_5	timer_tclk_clk	MCU_EXT_REFCLK0	1	TIMER5_CASCADE	0	0	100
Main_Timer_5	timer_tclk_clk	MAIN_PLL1.HSDIV3	1	TIMER5_CASCADE	0	0	192
Main_Timer_5	timer_tclk_clk	MAIN_PLL4.HSDIV2	1	TIMER5_CASCADE	0	0	197
Main_Timer_5	timer_tclk_clk	MAIN_PLL2.HSDIV6	1	TIMER5_CASCADE	0	0	225
Main_Timer_5	timer_tclk_clk	MAIN_PLL0.HSDIV8	1	TIMER5_CASCADE	0	0	250
Main_Timer_5	timer_tclk_clk	MAIN_PLL3.HSDIV3	1	TIMER5_CASCADE	0	0	250
Main_Timer_5	timer_tclk_clk	HFOSC1	1	TIMER5_CASCADE	0	0	[19.2 - 27]
Main_Timer_5	timer_tclk_clk	HFOSC0	1	TIMER5_CASCADE	0	0	[19.2, 20, 24, 25, 26, 27]
Main_Timer_5	timer_tclk_clk	CPTS0_RFT_CLK	1	TIMER5_CASCADE	0	0	
Main_Timer_5	timer_tclk_clk	EXT_REFCLK1	1	TIMER5_CASCADE	0	0	
Main_Timer_5	timer_tclk_clk	MAIN_CPSW_0.CPTS_GENF0	1	TIMER5_CASCADE	0	0	
Main_Timer_5	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF2	1	TIMER5_CASCADE	0	0	
Main_Timer_5	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF3	1	TIMER5_CASCADE	0	0	
Main_Timer_5	timer_tclk_clk	RCOSC	1	TIMER5_CASCADE	0	0	
Main_Timer_5	timer_tclk_clk	MAIN_TIMER_4.TIMER_PWM	1	TIMER5_CASCADE	1	0	
Main_Timer_6	timer_hclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_Timer_6	timer_tclk_clk	HFOSC0	1	MAIN_TIMER_CLKSEL	0	6	[19.2, 20, 24, 25, 26, 27]
Main_Timer_6	timer_tclk_clk	HFOSC1	1	MAIN_TIMER_CLKSEL	1	6	[19.2 - 27]
Main_Timer_6	timer_tclk_clk	MAIN_PLL0.HSDIV8	1	MAIN_TIMER_CLKSEL	2	6	250
Main_Timer_6	timer_tclk_clk	RCOSC	1	MAIN_TIMER_CLKSEL	3	6	
Main_Timer_6	timer_tclk_clk	MAIN_PLL3.HSDIV3	1	MAIN_TIMER_CLKSEL	4	6	250
Main_Timer_6	timer_tclk_clk	MCU_EXT_REFCLK0	1	MAIN_TIMER_CLKSEL	5	6	100
Main_Timer_6	timer_tclk_clk	EXT_REFCLK1	1	MAIN_TIMER_CLKSEL	6	6	
Main_Timer_6	timer_tclk_clk	LFXOSC	1	MAIN_TIMER_CLKSEL	7	6	0.032768
Main_Timer_6	timer_tclk_clk	CPTS0_RFT_CLK	1	MAIN_TIMER_CLKSEL	8	6	
Main_Timer_6	timer_tclk_clk	MAIN_PLL1.HSDIV3	1	MAIN_TIMER_CLKSEL	9	6	192
Main_Timer_6	timer_tclk_clk	MAIN_PLL2.HSDIV6	1	MAIN_TIMER_CLKSEL	10	6	225
Main_Timer_6	timer_tclk_clk	MAIN_PLL4.HSDIV2	1	MAIN_TIMER_CLKSEL	11	6	197
Main_Timer_6	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF2	1	MAIN_TIMER_CLKSEL	12	6	
Main_Timer_6	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF3	1	MAIN_TIMER_CLKSEL	13	6	
Main_Timer_6	timer_tclk_clk	MAIN_CPSW_0.CPTS_GENF0	1	MAIN_TIMER_CLKSEL	14	6	
Main_Timer_7	timer_hclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_Timer_7	timer_tclk_clk	HFOSC0	1	MAIN_TIMER_CLKSEL	0	7	[19.2, 20, 24, 25, 26, 27]
Main_Timer_7	timer_tclk_clk	HFOSC1	1	MAIN_TIMER_CLKSEL	1	7	[19.2 - 27]
Main_Timer_7	timer_tclk_clk	MAIN_PLL0.HSDIV8	1	MAIN_TIMER_CLKSEL	2	7	250
Main_Timer_7	timer_tclk_clk	RCOSC	1	MAIN_TIMER_CLKSEL	3	7	
Main_Timer_7	timer_tclk_clk	MAIN_PLL3.HSDIV3	1	MAIN_TIMER_CLKSEL	4	7	250

Copyright © 2024 Texas Instruments Incorporated

Table 5-33. Clock Mapping (continued)

Module	Clock Port	Clock	Divider	Control Register	Mux Value	Default Mux Value	Typical Frequency
Main_Timer_7	timer_tclk_clk	MCU_EXT_REFCLK0	1	MAIN_TIMER_CLKSEL	5	7	100
Main_Timer_7	timer_tclk_clk	EXT_REFCLK1	1	MAIN_TIMER_CLKSEL	6	7	
Main_Timer_7	timer_tclk_clk	LFXOSC	1	MAIN_TIMER_CLKSEL	7	7	0.032768
Main_Timer_7	timer_tclk_clk	CPTS0_RFT_CLK	1	MAIN_TIMER_CLKSEL	8	7	
Main_Timer_7	timer_tclk_clk	MAIN_PLL1.HSDIV3	1	MAIN_TIMER_CLKSEL	9	7	192
Main_Timer_7	timer_tclk_clk	MAIN_PLL2.HSDIV6	1	MAIN_TIMER_CLKSEL	10	7	225
Main_Timer_7	timer_tclk_clk	MAIN_PLL4.HSDIV2	1	MAIN_TIMER_CLKSEL	11	7	197
Main_Timer_7	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF2	1	MAIN_TIMER_CLKSEL	12	7	
Main_Timer_7	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF3	1	MAIN_TIMER_CLKSEL	13	7	
Main_Timer_7	timer_tclk_clk	MAIN_CPSW_0.CPTS_GENF0	1	MAIN_TIMER_CLKSEL	14	7	
Main_Timer_7	timer_tclk_clk	LFXOSC	1	TIMER7_CASCADE	0	0	0.032768
Main_Timer_7	timer_tclk_clk	MCU_EXT_REFCLK0	1	TIMER7_CASCADE	0	0	100
Main_Timer_7	timer_tclk_clk	MAIN_PLL1.HSDIV3	1	TIMER7_CASCADE	0	0	192
Main_Timer_7	timer_tclk_clk	MAIN_PLL4.HSDIV2	1	TIMER7_CASCADE	0	0	197
Main_Timer_7	timer_tclk_clk	MAIN_PLL2.HSDIV6	1	TIMER7_CASCADE	0	0	225
Main_Timer_7	timer_tclk_clk	MAIN_PLL0.HSDIV8	1	TIMER7_CASCADE	0	0	250
Main_Timer_7	timer_tclk_clk	MAIN_PLL3.HSDIV3	1	TIMER7_CASCADE	0	0	250
Main_Timer_7	timer_tclk_clk	HFOSC1	1	TIMER7_CASCADE	0	0	[19.2 - 27]
Main_Timer_7	timer_tclk_clk	HFOSC0	1	TIMER7_CASCADE	0	0	[19.2, 20, 24, 25, 26, 27]
Main_Timer_7	timer_tclk_clk	CPTS0_RFT_CLK	1	TIMER7_CASCADE	0	0	
Main_Timer_7	timer_tclk_clk	EXT_REFCLK1	1	TIMER7_CASCADE	0	0	
Main_Timer_7	timer_tclk_clk	MAIN_CPSW_0.CPTS_GENF0	1	TIMER7_CASCADE	0	0	
Main_Timer_7	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF2	1	TIMER7_CASCADE	0	0	
Main_Timer_7	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF3	1	TIMER7_CASCADE	0	0	
Main_Timer_7	timer_tclk_clk	RCOSC	1	TIMER7_CASCADE	0	0	
Main_Timer_7	timer_tclk_clk	MAIN_TIMER_6.TIMER_PWM	1	TIMER7_CASCADE	1	0	
Main_Timer_8	timer_hclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_Timer_8	timer_tclk_clk	HFOSC0	1	MAIN_TIMER_CLKSEL	0	8	[19.2, 20, 24, 25, 26, 27]
Main_Timer_8	timer_tclk_clk	HFOSC1	1	MAIN_TIMER_CLKSEL	1	8	[19.2 - 27]
Main_Timer_8	timer_tclk_clk	MAIN_PLL0.HSDIV8	1	MAIN_TIMER_CLKSEL	2	8	250
Main_Timer_8	timer_tclk_clk	RCOSC	1	MAIN_TIMER_CLKSEL	3	8	
Main_Timer_8	timer_tclk_clk	MAIN_PLL3.HSDIV3	1	MAIN_TIMER_CLKSEL	4	8	250
Main_Timer_8	timer_tclk_clk	MCU_EXT_REFCLK0	1	MAIN_TIMER_CLKSEL	5	8	100
Main_Timer_8	timer_tclk_clk	EXT_REFCLK1	1	MAIN_TIMER_CLKSEL	6	8	
Main_Timer_8	timer_tclk_clk	LFXOSC	1	MAIN_TIMER_CLKSEL	7	8	0.032768
Main_Timer_8	timer_tclk_clk	CPTS0_RFT_CLK	1	MAIN_TIMER_CLKSEL	8	8	
Main_Timer_8	timer_tclk_clk	MAIN_PLL1.HSDIV3	1	MAIN_TIMER_CLKSEL	9	8	192
Main_Timer_8	timer_tclk_clk	MAIN_PLL2.HSDIV6	1	MAIN_TIMER_CLKSEL	10	8	225
Main_Timer_8	timer_tclk_clk	MAIN_PLL4.HSDIV2	1	MAIN_TIMER_CLKSEL	11	8	197
Main_Timer_8	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF2	1	MAIN_TIMER_CLKSEL	12	8	
Main_Timer_8	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF3	1	MAIN_TIMER_CLKSEL	13	8	
Main_Timer_8	timer_tclk_clk	MAIN_CPSW_0.CPTS_GENF0	1	MAIN_TIMER_CLKSEL	14	8	
Main_Timer_9	timer_hclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_Timer_9	timer_tclk_clk	HFOSC0	1	MAIN_TIMER_CLKSEL	0	9	[19.2, 20, 24, 25, 26, 27]
Main_Timer_9	timer_tclk_clk	HFOSC1	1	MAIN_TIMER_CLKSEL	1	9	[19.2 - 27]
Main_Timer_9	timer_tclk_clk	MAIN_PLL0.HSDIV8	1	MAIN_TIMER_CLKSEL	2	9	250
Main_Timer_9	timer_tclk_clk	RCOSC	1	MAIN_TIMER_CLKSEL	3	9	
Main_Timer_9	timer_tclk_clk	MAIN_PLL3.HSDIV3	1	MAIN_TIMER_CLKSEL	4	9	250
Main_Timer_9	timer_tclk_clk	MCU_EXT_REFCLK0	1	MAIN_TIMER_CLKSEL	5	9	100
Main_Timer_9	timer_tclk_clk	EXT_REFCLK1	1	MAIN_TIMER_CLKSEL	6	9	
Main_Timer_9	timer_tclk_clk	LFXOSC	1	MAIN_TIMER_CLKSEL	7	9	0.032768

Copyright © 2024 Texas Instruments Incorporated

Table 5-33. Clock Mapping (continued)

Module	Clock Port	Clock	Divider	Control Register	Mux Value	Default Mux Value	Typical Frequency
Main_Timer_9	timer_tclk_clk	CPTS0_RFT_CLK	1	MAIN_TIMER_CLKSEL	8	9	
Main_Timer_9	timer_tclk_clk	MAIN_PLL1.HSDIV3	1	MAIN_TIMER_CLKSEL	9	9	192
Main_Timer_9	timer_tclk_clk	MAIN_PLL2.HSDIV6	1	MAIN_TIMER_CLKSEL	10	9	225
Main_Timer_9	timer_tclk_clk	MAIN_PLL4.HSDIV2	1	MAIN_TIMER_CLKSEL	11	9	197
Main_Timer_9	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF2	1	MAIN_TIMER_CLKSEL	12	9	
Main_Timer_9	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF3	1	MAIN_TIMER_CLKSEL	13	9	
Main_Timer_9	timer_tclk_clk	MAIN_CPSW_0.CPTS_GENF0	1	MAIN_TIMER_CLKSEL	14	9	
Main_Timer_9	timer_tclk_clk	LFXOSC	1	TIMER9_CASCADE	0	0	0.032768
Main_Timer_9	timer_tclk_clk	MCU_EXT_REFCLK0	1	TIMER9_CASCADE	0	0	100
Main_Timer_9	timer_tclk_clk	MAIN_PLL1.HSDIV3	1	TIMER9_CASCADE	0	0	192
Main_Timer_9	timer_tclk_clk	MAIN_PLL4.HSDIV2	1	TIMER9_CASCADE	0	0	197
Main_Timer_9	timer_tclk_clk	MAIN_PLL2.HSDIV6	1	TIMER9_CASCADE	0	0	225
Main_Timer_9	timer_tclk_clk	MAIN_PLL0.HSDIV8	1	TIMER9_CASCADE	0	0	250
Main_Timer_9	timer_tclk_clk	MAIN_PLL3.HSDIV3	1	TIMER9_CASCADE	0	0	250
Main_Timer_9	timer_tclk_clk	HFOSC1	1	TIMER9_CASCADE	0	0	[19.2 - 27]
Main_Timer_9	timer_tclk_clk	HFOSC0	1	TIMER9_CASCADE	0	0	[19.2, 20, 24, 25, 26, 27]
Main_Timer_9	timer_tclk_clk	CPTS0_RFT_CLK	1	TIMER9_CASCADE	0	0	
Main_Timer_9	timer_tclk_clk	EXT_REFCLK1	1	TIMER9_CASCADE	0	0	
Main_Timer_9	timer_tclk_clk	MAIN_CPSW_0.CPTS_GENF0	1	TIMER9_CASCADE	0	0	
Main_Timer_9	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF2	1	TIMER9_CASCADE	0	0	
Main_Timer_9	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF3	1	TIMER9_CASCADE	0	0	
Main_Timer_9	timer_tclk_clk	RCOSC	1	TIMER9_CASCADE	0	0	
Main_Timer_9	timer_tclk_clk	MAIN_TIMER_8.TIMER_PWM	1	TIMER9_CASCADE	1	0	
Main_Timer_10	timer_hclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_Timer_10	timer_tclk_clk	HFOSC0	1	MAIN_TIMER_CLKSEL	0	10	[19.2, 20, 24, 25, 26, 27]
Main_Timer_10	timer_tclk_clk	HFOSC1	1	MAIN_TIMER_CLKSEL	1	10	[19.2 - 27]
Main_Timer_10	timer_tclk_clk	MAIN_PLL0.HSDIV8	1	MAIN_TIMER_CLKSEL	2	10	250
Main_Timer_10	timer_tclk_clk	RCOSC	1	MAIN_TIMER_CLKSEL	3	10	
Main_Timer_10	timer_tclk_clk	MAIN_PLL3.HSDIV3	1	MAIN_TIMER_CLKSEL	4	10	250
Main_Timer_10	timer_tclk_clk	MCU_EXT_REFCLK0	1	MAIN_TIMER_CLKSEL	5	10	100
Main_Timer_10	timer_tclk_clk	EXT_REFCLK1	1	MAIN_TIMER_CLKSEL	6	10	
Main_Timer_10	timer_tclk_clk	LFXOSC	1	MAIN_TIMER_CLKSEL	7	10	0.032768
Main_Timer_10	timer_tclk_clk	CPTS0_RFT_CLK	1	MAIN_TIMER_CLKSEL	8	10	
Main_Timer_10	timer_tclk_clk	MAIN_PLL1.HSDIV3	1	MAIN_TIMER_CLKSEL	9	10	192
Main_Timer_10	timer_tclk_clk	MAIN_PLL2.HSDIV6	1	MAIN_TIMER_CLKSEL	10	10	225
Main_Timer_10	timer_tclk_clk	MAIN_PLL4.HSDIV2	1	MAIN_TIMER_CLKSEL	11	10	197
Main_Timer_10	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF2	1	MAIN_TIMER_CLKSEL	12	10	
Main_Timer_10	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF3	1	MAIN_TIMER_CLKSEL	13	10	
Main_Timer_10	timer_tclk_clk	MAIN_CPSW_0.CPTS_GENF0	1	MAIN_TIMER_CLKSEL	14	10	
Main_Timer_11	timer_hclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_Timer_11	timer_tclk_clk	HFOSC0	1	MAIN_TIMER_CLKSEL	0	11	[19.2, 20, 24, 25, 26, 27]
Main_Timer_11	timer_tclk_clk	HFOSC1	1	MAIN_TIMER_CLKSEL	1	11	[19.2 - 27]
Main_Timer_11	timer_tclk_clk	MAIN_PLL0.HSDIV8	1	MAIN_TIMER_CLKSEL	2	11	250
Main_Timer_11	timer_tclk_clk	RCOSC	1	MAIN_TIMER_CLKSEL	3	11	
Main_Timer_11	timer_tclk_clk	MAIN_PLL3.HSDIV3	1	MAIN_TIMER_CLKSEL	4	11	250
Main_Timer_11	timer_tclk_clk	MCU_EXT_REFCLK0	1	MAIN_TIMER_CLKSEL	5	11	100
Main_Timer_11	timer_tclk_clk	EXT_REFCLK1	1	MAIN_TIMER_CLKSEL	6	11	
Main_Timer_11	timer_tclk_clk	LFXOSC	1	MAIN_TIMER_CLKSEL	7	11	0.032768
Main_Timer_11	timer_tclk_clk	CPTS0_RFT_CLK	1	MAIN_TIMER_CLKSEL	8	11	
Main_Timer_11	timer_tclk_clk	MAIN_PLL1.HSDIV3	1	MAIN_TIMER_CLKSEL	9	11	192
Main_Timer_11	timer_tclk_clk	MAIN_PLL2.HSDIV6	1	MAIN_TIMER_CLKSEL	10	11	225

Copyright © 2024 Texas Instruments Incorporated

Table 5-33. Clock Mapping (continued)

Module	Clock Port	Clock	Divider	Control Register	Mux Value	Default Mux Value	Typical Frequency
Main_Timer_11	timer_tclk_clk	MAIN_PLL4.HSDIV2	1	MAIN_TIMER_CLKSEL	11	11	197
Main_Timer_11	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF2	1	MAIN_TIMER_CLKSEL	12	11	
Main_Timer_11	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF3	1	MAIN_TIMER_CLKSEL	13	11	
Main_Timer_11	timer_tclk_clk	MAIN_CPSW_0.CPTS_GENF0	1	MAIN_TIMER_CLKSEL	14	11	
Main_Timer_11	timer_tclk_clk	LFXOSC	1	TIMER11_CASCADE	0	0	0.032768
Main_Timer_11	timer_tclk_clk	MCU_EXT_REFCLK0	1	TIMER11_CASCADE	0	0	100
Main_Timer_11	timer_tclk_clk	MAIN_PLL1.HSDIV3	1	TIMER11_CASCADE	0	0	192
Main_Timer_11	timer_tclk_clk	MAIN_PLL4.HSDIV2	1	TIMER11_CASCADE	0	0	197
Main_Timer_11	timer_tclk_clk	MAIN_PLL2.HSDIV6	1	TIMER11_CASCADE	0	0	225
Main_Timer_11	timer_tclk_clk	MAIN_PLL0.HSDIV8	1	TIMER11_CASCADE	0	0	250
Main_Timer_11	timer_tclk_clk	MAIN_PLL3.HSDIV3	1	TIMER11_CASCADE	0	0	250
Main_Timer_11	timer_tclk_clk	HFOSC1	1	TIMER11_CASCADE	0	0	[19.2 - 27]
Main_Timer_11	timer_tclk_clk	HFOSC0	1	TIMER11_CASCADE	0	0	[19.2, 20, 24, 25, 26, 27]
Main_Timer_11	timer_tclk_clk	CPTS0_RFT_CLK	1	TIMER11_CASCADE	0	0	
Main_Timer_11	timer_tclk_clk	EXT_REFCLK1	1	TIMER11_CASCADE	0	0	
Main_Timer_11	timer_tclk_clk	MAIN_CPSW_0.CPTS_GENF0	1	TIMER11_CASCADE	0	0	
Main_Timer_11	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF2	1	TIMER11_CASCADE	0	0	
Main_Timer_11	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF3	1	TIMER11_CASCADE	0	0	
Main_Timer_11	timer_tclk_clk	RCOSC	1	TIMER11_CASCADE	0	0	
Main_Timer_11	timer_tclk_clk	MAIN_TIMER_10.TIMER_PWM	1	TIMER11_CASCADE	1	0	
Main_Timer_12	timer_hclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_Timer_12	timer_tclk_clk	HFOSC0	1	MAIN_TIMER_CLKSEL	0	12	[19.2, 20, 24, 25, 26, 27]
Main_Timer_12	timer_tclk_clk	HFOSC1	1	MAIN_TIMER_CLKSEL	1	12	[19.2 - 27]
Main_Timer_12	timer_tclk_clk	MAIN_PLL0.HSDIV8	1	MAIN_TIMER_CLKSEL	2	12	250
Main_Timer_12	timer_tclk_clk	RCOSC	1	MAIN_TIMER_CLKSEL	3	12	
Main_Timer_12	timer_tclk_clk	MAIN_PLL3.HSDIV3	1	MAIN_TIMER_CLKSEL	4	12	250
Main_Timer_12	timer_tclk_clk	MCU_EXT_REFCLK0	1	MAIN_TIMER_CLKSEL	5	12	100
Main_Timer_12	timer_tclk_clk	EXT_REFCLK1	1	MAIN_TIMER_CLKSEL	6	12	
Main_Timer_12	timer_tclk_clk	LFXOSC	1	MAIN_TIMER_CLKSEL	7	12	0.032768
Main_Timer_12	timer_tclk_clk	CPTS0_RFT_CLK	1	MAIN_TIMER_CLKSEL	8	12	
Main_Timer_12	timer_tclk_clk	MAIN_PLL1.HSDIV3	1	MAIN_TIMER_CLKSEL	9	12	192
Main_Timer_12	timer_tclk_clk	MAIN_PLL2.HSDIV6	1	MAIN_TIMER_CLKSEL	10	12	225
Main_Timer_12	timer_tclk_clk	MAIN_PLL4.HSDIV2	1	MAIN_TIMER_CLKSEL	11	12	197
Main_Timer_12	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF2	1	MAIN_TIMER_CLKSEL	12	12	
Main_Timer_12	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF3	1	MAIN_TIMER_CLKSEL	13	12	
Main_Timer_12	timer_tclk_clk	MAIN_CPSW_0.CPTS_GENF0	1	MAIN_TIMER_CLKSEL	14	12	
Main_Timer_13	timer_hclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_Timer_13	timer_tclk_clk	HFOSC0	1	MAIN_TIMER_CLKSEL	0	13	[19.2, 20, 24, 25, 26, 27]
Main_Timer_13	timer_tclk_clk	HFOSC1	1	MAIN_TIMER_CLKSEL	1	13	[19.2 - 27]
Main_Timer_13	timer_tclk_clk	MAIN_PLL0.HSDIV8	1	MAIN_TIMER_CLKSEL	2	13	250
Main_Timer_13	timer_tclk_clk	RCOSC	1	MAIN_TIMER_CLKSEL	3	13	
Main_Timer_13	timer_tclk_clk	MAIN_PLL3.HSDIV3	1	MAIN_TIMER_CLKSEL	4	13	250
Main_Timer_13	timer_tclk_clk	MCU_EXT_REFCLK0	1	MAIN_TIMER_CLKSEL	5	13	100
Main_Timer_13	timer_tclk_clk	EXT_REFCLK1	1	MAIN_TIMER_CLKSEL	6	13	
Main_Timer_13	timer_tclk_clk	LFXOSC	1	MAIN_TIMER_CLKSEL	7	13	0.032768
Main_Timer_13	timer_tclk_clk	CPTS0_RFT_CLK	1	MAIN_TIMER_CLKSEL	8	13	
Main_Timer_13	timer_tclk_clk	MAIN_PLL1.HSDIV3	1	MAIN_TIMER_CLKSEL	9	13	192
Main_Timer_13	timer_tclk_clk	MAIN_PLL2.HSDIV6	1	MAIN_TIMER_CLKSEL	10	13	225
Main_Timer_13	timer_tclk_clk	MAIN_PLL4.HSDIV2	1	MAIN_TIMER_CLKSEL	11	13	197
Main_Timer_13	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF2	1	MAIN_TIMER_CLKSEL	12	13	
Main_Timer_13	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF3	1	MAIN_TIMER_CLKSEL	13	13	

Copyright © 2024 Texas Instruments Incorporated

Table 5-33. Clock Mapping (continued)

Module	Clock Port	Clock	Divider	Control Register	Mux Value	Default Mux Value	Typical Frequency
Main_Timer_13	timer_tclk_clk	MAIN_CPSW_0.CPTS_GENF0	1	MAIN_TIMER_CLKSEL	14	13	
Main_Timer_13	timer_tclk_clk	LFXOSC	1	TIMER13_CASCADE	0	0	0.032768
Main_Timer_13	timer_tclk_clk	MCU_EXT_REFCLK0	1	TIMER13_CASCADE	0	0	100
Main_Timer_13	timer_tclk_clk	MAIN_PLL1.HSDIV3	1	TIMER13_CASCADE	0	0	192
Main_Timer_13	timer_tclk_clk	MAIN_PLL4.HSDIV2	1	TIMER13_CASCADE	0	0	197
Main_Timer_13	timer_tclk_clk	MAIN_PLL2.HSDIV6	1	TIMER13_CASCADE	0	0	225
Main_Timer_13	timer_tclk_clk	MAIN_PLL0.HSDIV8	1	TIMER13_CASCADE	0	0	250
Main_Timer_13	timer_tclk_clk	MAIN_PLL3.HSDIV3	1	TIMER13_CASCADE	0	0	250
Main_Timer_13	timer_tclk_clk	HFOSC1	1	TIMER13_CASCADE	0	0	[19.2 - 27]
Main_Timer_13	timer_tclk_clk	HFOSC0	1	TIMER13_CASCADE	0	0	[19.2, 20, 24, 25, 26, 27]
Main_Timer_13	timer_tclk_clk	CPTS0_RFT_CLK	1	TIMER13_CASCADE	0	0	
Main_Timer_13	timer_tclk_clk	EXT_REFCLK1	1	TIMER13_CASCADE	0	0	
Main_Timer_13	timer_tclk_clk	MAIN_CPSW_0.CPTS_GENF0	1	TIMER13_CASCADE	0	0	
Main_Timer_13	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF2	1	TIMER13_CASCADE	0	0	
Main_Timer_13	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF3	1	TIMER13_CASCADE	0	0	
Main_Timer_13	timer_tclk_clk	RCOSC	1	TIMER13_CASCADE	0	0	
Main_Timer_13	timer_tclk_clk	MAIN_TIMER_12.TIMER_PWM	1	TIMER13_CASCADE	1	0	
Main_Timer_14	timer_hclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_Timer_14	timer_tclk_clk	HFOSC0	1	MAIN_TIMER_CLKSEL	0	14	[19.2, 20, 24, 25, 26, 27]
Main_Timer_14	timer_tclk_clk	HFOSC1	1	MAIN_TIMER_CLKSEL	1	14	[19.2 - 27]
Main_Timer_14	timer_tclk_clk	MAIN_PLL0.HSDIV8	1	MAIN_TIMER_CLKSEL	2	14	250
Main_Timer_14	timer_tclk_clk	RCOSC	1	MAIN_TIMER_CLKSEL	3	14	
Main_Timer_14	timer_tclk_clk	MAIN_PLL3.HSDIV3	1	MAIN_TIMER_CLKSEL	4	14	250
Main_Timer_14	timer_tclk_clk	MCU_EXT_REFCLK0	1	MAIN_TIMER_CLKSEL	5	14	100
Main_Timer_14	timer_tclk_clk	EXT_REFCLK1	1	MAIN_TIMER_CLKSEL	6	14	
Main_Timer_14	timer_tclk_clk	LFXOSC	1	MAIN_TIMER_CLKSEL	7	14	0.032768
Main_Timer_14	timer_tclk_clk	CPTS0_RFT_CLK	1	MAIN_TIMER_CLKSEL	8	14	
Main_Timer_14	timer_tclk_clk	MAIN_PLL1.HSDIV3	1	MAIN_TIMER_CLKSEL	9	14	192
Main_Timer_14	timer_tclk_clk	MAIN_PLL2.HSDIV6	1	MAIN_TIMER_CLKSEL	10	14	225
Main_Timer_14	timer_tclk_clk	MAIN_PLL4.HSDIV2	1	MAIN_TIMER_CLKSEL	11	14	197
Main_Timer_14	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF2	1	MAIN_TIMER_CLKSEL	12	14	
Main_Timer_14	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF3	1	MAIN_TIMER_CLKSEL	13	14	
Main_Timer_14	timer_tclk_clk	MAIN_CPSW_0.CPTS_GENF0	1	MAIN_TIMER_CLKSEL	14	14	
Main_Timer_15	timer_hclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_Timer_15	timer_tclk_clk	HFOSC0	1	MAIN_TIMER_CLKSEL	0	15	[19.2, 20, 24, 25, 26, 27]
Main_Timer_15	timer_tclk_clk	HFOSC1	1	MAIN_TIMER_CLKSEL	1	15	[19.2 - 27]
Main_Timer_15	timer_tclk_clk	MAIN_PLL0.HSDIV8	1	MAIN_TIMER_CLKSEL	2	15	250
Main_Timer_15	timer_tclk_clk	RCOSC	1	MAIN_TIMER_CLKSEL	3	15	
Main_Timer_15	timer_tclk_clk	MAIN_PLL3.HSDIV3	1	MAIN_TIMER_CLKSEL	4	15	250
Main_Timer_15	timer_tclk_clk	MCU_EXT_REFCLK0	1	MAIN_TIMER_CLKSEL	5	15	100
Main_Timer_15	timer_tclk_clk	EXT_REFCLK1	1	MAIN_TIMER_CLKSEL	6	15	
Main_Timer_15	timer_tclk_clk	LFXOSC	1	MAIN_TIMER_CLKSEL	7	15	0.032768
Main_Timer_15	timer_tclk_clk	CPTS0_RFT_CLK	1	MAIN_TIMER_CLKSEL	8	15	
Main_Timer_15	timer_tclk_clk	MAIN_PLL1.HSDIV3	1	MAIN_TIMER_CLKSEL	9	15	192
Main_Timer_15	timer_tclk_clk	MAIN_PLL2.HSDIV6	1	MAIN_TIMER_CLKSEL	10	15	225
Main_Timer_15	timer_tclk_clk	MAIN_PLL4.HSDIV2	1	MAIN_TIMER_CLKSEL	11	15	197
Main_Timer_15	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF2	1	MAIN_TIMER_CLKSEL	12	15	
Main_Timer_15	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF3	1	MAIN_TIMER_CLKSEL	13	15	
Main_Timer_15	timer_tclk_clk	MAIN_CPSW_0.CPTS_GENF0	1	MAIN_TIMER_CLKSEL	14	15	
Main_Timer_15	timer_tclk_clk	LFXOSC	1	TIMER15_CASCADE	0	0	0.032768
Main_Timer_15	timer_tclk_clk	MCU_EXT_REFCLK0	1	TIMER15_CASCADE	0	0	100

Copyright © 2024 Texas Instruments Incorporated

Table 5-33. Clock Mapping (continued)

Module	Clock Port	Clock	Divider	Control Register	Mux Value	Default Mux Value	Typical Frequency
Main_Timer_15	timer_tclk_clk	MAIN_PLL1.HSDIV3	1	TIMER15_CASCADE	0	0	192
Main_Timer_15	timer_tclk_clk	MAIN_PLL4.HSDIV2	1	TIMER15_CASCADE	0	0	197
Main_Timer_15	timer_tclk_clk	MAIN_PLL2.HSDIV6	1	TIMER15_CASCADE	0	0	225
Main_Timer_15	timer_tclk_clk	MAIN_PLL0.HSDIV8	1	TIMER15_CASCADE	0	0	250
Main_Timer_15	timer_tclk_clk	MAIN_PLL3.HSDIV3	1	TIMER15_CASCADE	0	0	250
Main_Timer_15	timer_tclk_clk	HFOSC1	1	TIMER15_CASCADE	0	0	[19.2 - 27]
Main_Timer_15	timer_tclk_clk	HFOSC0	1	TIMER15_CASCADE	0	0	[19.2, 20, 24, 25, 26, 27]
Main_Timer_15	timer_tclk_clk	CPTS0_RFT_CLK	1	TIMER15_CASCADE	0	0	
Main_Timer_15	timer_tclk_clk	EXT_REFCLK1	1	TIMER15_CASCADE	0	0	
Main_Timer_15	timer_tclk_clk	MAIN_CPSW_0.CPTS_GENF0	1	TIMER15_CASCADE	0	0	
Main_Timer_15	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF2	1	TIMER15_CASCADE	0	0	
Main_Timer_15	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF3	1	TIMER15_CASCADE	0	0	
Main_Timer_15	timer_tclk_clk	RCOSC	1	TIMER15_CASCADE	0	0	
Main_Timer_15	timer_tclk_clk	MAIN_TIMER_14.TIMER_PWM	1	TIMER15_CASCADE	1	0	
Main_Timer_16	timer_hclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_Timer_16	timer_tclk_clk	HFOSC0	1	MAIN_TIMER_CLKSEL	0	16	[19.2, 20, 24, 25, 26, 27]
Main_Timer_16	timer_tclk_clk	HFOSC1	1	MAIN_TIMER_CLKSEL	1	16	[19.2 - 27]
Main_Timer_16	timer_tclk_clk	MAIN_PLL0.HSDIV8	1	MAIN_TIMER_CLKSEL	2	16	250
Main_Timer_16	timer_tclk_clk	RCOSC	1	MAIN_TIMER_CLKSEL	3	16	
Main_Timer_16	timer_tclk_clk	MAIN_PLL3.HSDIV3	1	MAIN_TIMER_CLKSEL	4	16	250
Main_Timer_16	timer_tclk_clk	MCU_EXT_REFCLK0	1	MAIN_TIMER_CLKSEL	5	16	100
Main_Timer_16	timer_tclk_clk	EXT_REFCLK1	1	MAIN_TIMER_CLKSEL	6	16	
Main_Timer_16	timer_tclk_clk	LFXOSC	1	MAIN_TIMER_CLKSEL	7	16	0.032768
Main_Timer_16	timer_tclk_clk	CPTS0_RFT_CLK	1	MAIN_TIMER_CLKSEL	8	16	
Main_Timer_16	timer_tclk_clk	MAIN_PLL1.HSDIV3	1	MAIN_TIMER_CLKSEL	9	16	192
Main_Timer_16	timer_tclk_clk	MAIN_PLL2.HSDIV6	1	MAIN_TIMER_CLKSEL	10	16	225
Main_Timer_16	timer_tclk_clk	MAIN_PLL4.HSDIV2	1	MAIN_TIMER_CLKSEL	11	16	197
Main_Timer_16	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF2	1	MAIN_TIMER_CLKSEL	12	16	
Main_Timer_16	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF3	1	MAIN_TIMER_CLKSEL	13	16	
Main_Timer_16	timer_tclk_clk	MAIN_CPSW_0.CPTS_GENF0	1	MAIN_TIMER_CLKSEL	14	16	
Main_Timer_16	timer_tclk_clk	LFXOSC	1	MAIN_TIMER16_AFS_EN	0	0	0.032768
Main_Timer_16	timer_tclk_clk	MCU_EXT_REFCLK0	1	MAIN_TIMER16_AFS_EN	0	0	100
Main_Timer_16	timer_tclk_clk	MAIN_PLL1.HSDIV3	1	MAIN_TIMER16_AFS_EN	0	0	192
Main_Timer_16	timer_tclk_clk	MAIN_PLL4.HSDIV2	1	MAIN_TIMER16_AFS_EN	0	0	197
Main_Timer_16	timer_tclk_clk	MAIN_PLL2.HSDIV6	1	MAIN_TIMER16_AFS_EN	0	0	225
Main_Timer_16	timer_tclk_clk	MAIN_PLL0.HSDIV8	1	MAIN_TIMER16_AFS_EN	0	0	250
Main_Timer_16	timer_tclk_clk	MAIN_PLL3.HSDIV3	1	MAIN_TIMER16_AFS_EN	0	0	250
Main_Timer_16	timer_tclk_clk	HFOSC1	1	MAIN_TIMER16_AFS_EN	0	0	[19.2 - 27]
Main_Timer_16	timer_tclk_clk	HFOSC0	1	MAIN_TIMER16_AFS_EN	0	0	[19.2, 20, 24, 25, 26, 27]
Main_Timer_16	timer_tclk_clk	CPTS0_RFT_CLK	1	MAIN_TIMER16_AFS_EN	0	0	
Main_Timer_16	timer_tclk_clk	EXT_REFCLK1	1	MAIN_TIMER16_AFS_EN	0	0	
Main_Timer_16	timer_tclk_clk	MAIN_CPSW_0.CPTS_GENF0	1	MAIN_TIMER16_AFS_EN	0	0	
Main_Timer_16	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF2	1	MAIN_TIMER16_AFS_EN	0	0	

Copyright © 2024 Texas Instruments Incorporated

Table 5-33. Clock Mapping (continued)

Module	Clock Port	Clock	Divider	Control Register	Mux Value	Default Mux Value	Typical Frequency
Main_Timer_16	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF3	1	MAIN_TIMER16_AFS_EN	0	0	
Main_Timer_16	timer_tclk_clk	RCOSC	1	MAIN_TIMER16_AFS_EN	0	0	
Main_Timer_16	timer_tclk_clk	MCASP0_AFSR	1	MAIN_TIMER16_AFS_EN	1	0	
Main_Timer_16	timer_tclk_clk	MCASP0_AFSX	1	MAIN_TIMER16_AFS_EN	1	0	
Main_Timer_16	timer_tclk_clk	MCASP1_AFSR	1	MAIN_TIMER16_AFS_EN	1	0	
Main_Timer_16	timer_tclk_clk	MCASP1_AFSX	1	MAIN_TIMER16_AFS_EN	1	0	
Main_Timer_16	timer_tclk_clk	MCASP2_AFSR	1	MAIN_TIMER16_AFS_EN	1	0	
Main_Timer_16	timer_tclk_clk	MCASP2_AFSX	1	MAIN_TIMER16_AFS_EN	1	0	
Main_Timer_16	timer_tclk_clk	MCASP3_AFSR	1	MAIN_TIMER16_AFS_EN	1	0	
Main_Timer_16	timer_tclk_clk	MCASP3_AFSX	1	MAIN_TIMER16_AFS_EN	1	0	
Main_Timer_16	timer_tclk_clk	MCASP4_AFSR	1	MAIN_TIMER16_AFS_EN	1	0	
Main_Timer_16	timer_tclk_clk	MCASP4_AFSX	1	MAIN_TIMER16_AFS_EN	1	0	
Main_Timer_16	timer_tclk_clk	MCASP0_AFSR	1	MAIN_TIMER16_AFS_SEL	0	0	
Main_Timer_16	timer_tclk_clk	MCASP0_AFSX	1	MAIN_TIMER16_AFS_SEL	1	0	
Main_Timer_16	timer_tclk_clk	MCASP1_AFSR	1	MAIN_TIMER16_AFS_SEL	2	0	
Main_Timer_16	timer_tclk_clk	MCASP1_AFSX	1	MAIN_TIMER16_AFS_SEL	3	0	
Main_Timer_16	timer_tclk_clk	MCASP2_AFSR	1	MAIN_TIMER16_AFS_SEL	4	0	
Main_Timer_16	timer_tclk_clk	MCASP2_AFSX	1	MAIN_TIMER16_AFS_SEL	5	0	
Main_Timer_16	timer_tclk_clk	MCASP3_AFSR	1	MAIN_TIMER16_AFS_SEL	6	0	
Main_Timer_16	timer_tclk_clk	MCASP3_AFSX	1	MAIN_TIMER16_AFS_SEL	7	0	
Main_Timer_16	timer_tclk_clk	MCASP4_AFSR	1	MAIN_TIMER16_AFS_SEL	8	0	
Main_Timer_16	timer_tclk_clk	MCASP4_AFSX	1	MAIN_TIMER16_AFS_SEL	9	0	
Main_Timer_17	timer_hclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_Timer_17	timer_tclk_clk	HFOSC0	1	MAIN_TIMER_CLKSEL	0	17	[19.2, 20, 24, 25, 26, 27]
Main_Timer_17	timer_tclk_clk	HFOSC1	1	MAIN_TIMER_CLKSEL	1	17	[19.2 - 27]
Main_Timer_17	timer_tclk_clk	MAIN_PLL0.HSDIV8	1	MAIN_TIMER_CLKSEL	2	17	250
Main_Timer_17	timer_tclk_clk	RCOSC	1	MAIN_TIMER_CLKSEL	3	17	
Main_Timer_17	timer_tclk_clk	MAIN_PLL3.HSDIV3	1	MAIN_TIMER_CLKSEL	4	17	250
Main_Timer_17	timer_tclk_clk	MCU_EXT_REFCLK0	1	MAIN_TIMER_CLKSEL	5	17	100
Main_Timer_17	timer_tclk_clk	EXT_REFCLK1	1	MAIN_TIMER_CLKSEL	6	17	
Main_Timer_17	timer_tclk_clk	LFXOSC	1	MAIN_TIMER_CLKSEL	7	17	0.032768
Main_Timer_17	timer_tclk_clk	CPTS0_RFT_CLK	1	MAIN_TIMER_CLKSEL	8	17	
Main_Timer_17	timer_tclk_clk	MAIN_PLL1.HSDIV3	1	MAIN_TIMER_CLKSEL	9	17	192
Main_Timer_17	timer_tclk_clk	MAIN_PLL2.HSDIV6	1	MAIN_TIMER_CLKSEL	10	17	225
Main_Timer_17	timer_tclk_clk	MAIN_PLL4.HSDIV2	1	MAIN_TIMER_CLKSEL	11	17	197
Main_Timer_17	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF2	1	MAIN_TIMER_CLKSEL	12	17	
Main_Timer_17	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF3	1	MAIN_TIMER_CLKSEL	13	17	
Main_Timer_17	timer_tclk_clk	MAIN_CPSW_0.CPTS_GENF0	1	MAIN_TIMER_CLKSEL	14	17	
Main_Timer_17	timer_tclk_clk	LFXOSC	1	TIMER17_CASCADE	0	0	0.032768
Main_Timer_17	timer_tclk_clk	MCU_EXT_REFCLK0	1	TIMER17_CASCADE	0	0	100

Copyright © 2024 Texas Instruments Incorporated

Table 5-33. Clock Mapping (continued)

Module	Clock Port	Clock	Divider	Control Register	Mux Value	Default Mux Value	Typical Frequency
Main_Timer_17	timer_tclk_clk	MAIN_PLL1.HSDIV3	1	TIMER17_CASCADE	0	0	192
Main_Timer_17	timer_tclk_clk	MAIN_PLL4.HSDIV2	1	TIMER17_CASCADE	0	0	197
Main_Timer_17	timer_tclk_clk	MAIN_PLL2.HSDIV6	1	TIMER17_CASCADE	0	0	225
Main_Timer_17	timer_tclk_clk	MAIN_PLL0.HSDIV8	1	TIMER17_CASCADE	0	0	250
Main_Timer_17	timer_tclk_clk	MAIN_PLL3.HSDIV3	1	TIMER17_CASCADE	0	0	250
Main_Timer_17	timer_tclk_clk	HFOSC1	1	TIMER17_CASCADE	0	0	[19.2 - 27]
Main_Timer_17	timer_tclk_clk	HFOSC0	1	TIMER17_CASCADE	0	0	[19.2, 20, 24, 25, 26, 27]
Main_Timer_17	timer_tclk_clk	CPTS0_RFT_CLK	1	TIMER17_CASCADE	0	0	
Main_Timer_17	timer_tclk_clk	EXT_REFCLK1	1	TIMER17_CASCADE	0	0	
Main_Timer_17	timer_tclk_clk	MAIN_CPSW_0.CPTS_GENF0	1	TIMER17_CASCADE	0	0	
Main_Timer_17	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF2	1	TIMER17_CASCADE	0	0	
Main_Timer_17	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF3	1	TIMER17_CASCADE	0	0	
Main_Timer_17	timer_tclk_clk	RCOSC	1	TIMER17_CASCADE	0	0	
Main_Timer_17	timer_tclk_clk	MAIN_TIMER_16.TIMER_PWM	1	TIMER17_CASCADE	1	0	
Main_Timer_18	timer_hclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_Timer_18	timer_tclk_clk	HFOSC0	1	MAIN_TIMER_CLKSEL	0	18	[19.2, 20, 24, 25, 26, 27]
Main_Timer_18	timer_tclk_clk	HFOSC1	1	MAIN_TIMER_CLKSEL	1	18	[19.2 - 27]
Main_Timer_18	timer_tclk_clk	MAIN_PLL0.HSDIV8	1	MAIN_TIMER_CLKSEL	2	18	250
Main_Timer_18	timer_tclk_clk	RCOSC	1	MAIN_TIMER_CLKSEL	3	18	
Main_Timer_18	timer_tclk_clk	MAIN_PLL3.HSDIV3	1	MAIN_TIMER_CLKSEL	4	18	250
Main_Timer_18	timer_tclk_clk	MCU_EXT_REFCLK0	1	MAIN_TIMER_CLKSEL	5	18	100
Main_Timer_18	timer_tclk_clk	EXT_REFCLK1	1	MAIN_TIMER_CLKSEL	6	18	
Main_Timer_18	timer_tclk_clk	LFXOSC	1	MAIN_TIMER_CLKSEL	7	18	0.032768
Main_Timer_18	timer_tclk_clk	CPTS0_RFT_CLK	1	MAIN_TIMER_CLKSEL	8	18	
Main_Timer_18	timer_tclk_clk	MAIN_PLL1.HSDIV3	1	MAIN_TIMER_CLKSEL	9	18	192
Main_Timer_18	timer_tclk_clk	MAIN_PLL2.HSDIV6	1	MAIN_TIMER_CLKSEL	10	18	225
Main_Timer_18	timer_tclk_clk	MAIN_PLL4.HSDIV2	1	MAIN_TIMER_CLKSEL	11	18	197
Main_Timer_18	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF2	1	MAIN_TIMER_CLKSEL	12	18	
Main_Timer_18	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF3	1	MAIN_TIMER_CLKSEL	13	18	
Main_Timer_18	timer_tclk_clk	MAIN_CPSW_0.CPTS_GENF0	1	MAIN_TIMER_CLKSEL	14	18	
Main_Timer_18	timer_tclk_clk	LFXOSC	1	MAIN_TIMER18_AFS_EN	0	0	0.032768
Main_Timer_18	timer_tclk_clk	MCU_EXT_REFCLK0	1	MAIN_TIMER18_AFS_EN	0	0	100
Main_Timer_18	timer_tclk_clk	MAIN_PLL1.HSDIV3	1	MAIN_TIMER18_AFS_EN	0	0	192
Main_Timer_18	timer_tclk_clk	MAIN_PLL4.HSDIV2	1	MAIN_TIMER18_AFS_EN	0	0	197
Main_Timer_18	timer_tclk_clk	MAIN_PLL2.HSDIV6	1	MAIN_TIMER18_AFS_EN	0	0	225
Main_Timer_18	timer_tclk_clk	MAIN_PLL0.HSDIV8	1	MAIN_TIMER18_AFS_EN	0	0	250
Main_Timer_18	timer_tclk_clk	MAIN_PLL3.HSDIV3	1	MAIN_TIMER18_AFS_EN	0	0	250
Main_Timer_18	timer_tclk_clk	HFOSC1	1	MAIN_TIMER18_AFS_EN	0	0	[19.2 - 27]
Main_Timer_18	timer_tclk_clk	HFOSC0	1	MAIN_TIMER18_AFS_EN	0	0	[19.2, 20, 24, 25, 26, 27]
Main_Timer_18	timer_tclk_clk	CPTS0_RFT_CLK	1	MAIN_TIMER18_AFS_EN	0	0	
Main_Timer_18	timer_tclk_clk	EXT_REFCLK1	1	MAIN_TIMER18_AFS_EN	0	0	
Main_Timer_18	timer_tclk_clk	MAIN_CPSW_0.CPTS_GENF0	1	MAIN_TIMER18_AFS_EN	0	0	
Main_Timer_18	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF2	1	MAIN_TIMER18_AFS_EN	0	0	

Copyright © 2024 Texas Instruments Incorporated

Table 5-33. Clock Mapping (continued)

Module	Clock Port	Clock	Divider	Control Register	Mux Value	Default Mux Value	Typical Frequency
Main_Timer_18	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF3	1	MAIN_TIMER18_AFS_EN	0	0	
Main_Timer_18	timer_tclk_clk	RCOSC	1	MAIN_TIMER18_AFS_EN	0	0	
Main_Timer_18	timer_tclk_clk	MCASP0_AFSR	1	MAIN_TIMER18_AFS_EN	1	0	
Main_Timer_18	timer_tclk_clk	MCASP0_AFSX	1	MAIN_TIMER18_AFS_EN	1	0	
Main_Timer_18	timer_tclk_clk	MCASP1_AFSR	1	MAIN_TIMER18_AFS_EN	1	0	
Main_Timer_18	timer_tclk_clk	MCASP1_AFSX	1	MAIN_TIMER18_AFS_EN	1	0	
Main_Timer_18	timer_tclk_clk	MCASP2_AFSR	1	MAIN_TIMER18_AFS_EN	1	0	
Main_Timer_18	timer_tclk_clk	MCASP2_AFSX	1	MAIN_TIMER18_AFS_EN	1	0	
Main_Timer_18	timer_tclk_clk	MCASP3_AFSR	1	MAIN_TIMER18_AFS_EN	1	0	
Main_Timer_18	timer_tclk_clk	MCASP3_AFSX	1	MAIN_TIMER18_AFS_EN	1	0	
Main_Timer_18	timer_tclk_clk	MCASP4_AFSR	1	MAIN_TIMER18_AFS_EN	1	0	
Main_Timer_18	timer_tclk_clk	MCASP0_AFSR	1	MAIN_TIMER18_AFS_SEL	0	0	
Main_Timer_18	timer_tclk_clk	MCASP0_AFSX	1	MAIN_TIMER18_AFS_SEL	1	0	
Main_Timer_18	timer_tclk_clk	MCASP1_AFSR	1	MAIN_TIMER18_AFS_SEL	2	0	
Main_Timer_18	timer_tclk_clk	MCASP1_AFSX	1	MAIN_TIMER18_AFS_SEL	3	0	
Main_Timer_18	timer_tclk_clk	MCASP2_AFSR	1	MAIN_TIMER18_AFS_SEL	4	0	
Main_Timer_18	timer_tclk_clk	MCASP2_AFSX	1	MAIN_TIMER18_AFS_SEL	5	0	
Main_Timer_18	timer_tclk_clk	MCASP3_AFSR	1	MAIN_TIMER18_AFS_SEL	6	0	
Main_Timer_18	timer_tclk_clk	MCASP3_AFSX	1	MAIN_TIMER18_AFS_SEL	7	0	
Main_Timer_18	timer_tclk_clk	MCASP4_AFSR	1	MAIN_TIMER18_AFS_SEL	8	0	
Main_Timer_18	timer_tclk_clk	MCASP4_AFSR	1	MAIN_TIMER18_AFS_SEL	9	0	
Main_Timer_19	timer_hclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_Timer_19	timer_tclk_clk	HFOSC0	1	MAIN_TIMER_CLKSEL	0	19	[19.2, 20, 24, 25, 26, 27]
Main_Timer_19	timer_tclk_clk	HFOSC1	1	MAIN_TIMER_CLKSEL	1	19	[19.2 - 27]
Main_Timer_19	timer_tclk_clk	MAIN_PLL0.HSDIV8	1	MAIN_TIMER_CLKSEL	2	19	250
Main_Timer_19	timer_tclk_clk	RCOSC	1	MAIN_TIMER_CLKSEL	3	19	
Main_Timer_19	timer_tclk_clk	MAIN_PLL3.HSDIV3	1	MAIN_TIMER_CLKSEL	4	19	250
Main_Timer_19	timer_tclk_clk	MCU_EXT_REFCLK0	1	MAIN_TIMER_CLKSEL	5	19	100
Main_Timer_19	timer_tclk_clk	EXT_REFCLK1	1	MAIN_TIMER_CLKSEL	6	19	
Main_Timer_19	timer_tclk_clk	LFXOSC	1	MAIN_TIMER_CLKSEL	7	19	0.032768
Main_Timer_19	timer_tclk_clk	CPTS0_RFT_CLK	1	MAIN_TIMER_CLKSEL	8	19	
Main_Timer_19	timer_tclk_clk	MAIN_PLL1.HSDIV3	1	MAIN_TIMER_CLKSEL	9	19	192
Main_Timer_19	timer_tclk_clk	MAIN_PLL2.HSDIV6	1	MAIN_TIMER_CLKSEL	10	19	225
Main_Timer_19	timer_tclk_clk	MAIN_PLL4.HSDIV2	1	MAIN_TIMER_CLKSEL	11	19	197
Main_Timer_19	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF2	1	MAIN_TIMER_CLKSEL	12	19	
Main_Timer_19	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF3	1	MAIN_TIMER_CLKSEL	13	19	
Main_Timer_19	timer_tclk_clk	MAIN_CPSW_0.CPTS_GENF0	1	MAIN_TIMER_CLKSEL	14	19	
Main_Timer_19	timer_tclk_clk	LFXOSC	1	TIMER19_CASCADE	0	0	0.032768
Main_Timer_19	timer_tclk_clk	MCU_EXT_REFCLK0	1	TIMER19_CASCADE	0	0	100
Main_Timer_19	timer_tclk_clk	MAIN_PLL1.HSDIV3	1	TIMER19_CASCADE	0	0	192
Main_Timer_19	timer_tclk_clk	MAIN_PLL4.HSDIV2	1	TIMER19_CASCADE	0	0	197

Copyright © 2024 Texas Instruments Incorporated

Table 5-33. Clock Mapping (continued)

Module	Clock Port	Clock	Divider	Control Register	Mux Value	Default Mux Value	Typical Frequency
Main_Timer_19	timer_tclk_clk	MAIN_PLL2.HSDIV6	1	TIMER19_CASCADE	0	0	225
Main_Timer_19	timer_tclk_clk	MAIN_PLL0.HSDIV8	1	TIMER19_CASCADE	0	0	250
Main_Timer_19	timer_tclk_clk	MAIN_PLL3.HSDIV3	1	TIMER19_CASCADE	0	0	250
Main_Timer_19	timer_tclk_clk	HFOSC1	1	TIMER19_CASCADE	0	0	[19.2 - 27]
Main_Timer_19	timer_tclk_clk	HFOSC0	1	TIMER19_CASCADE	0	0	[19.2, 20, 24, 25, 26, 27]
Main_Timer_19	timer_tclk_clk	CPTS0_RFT_CLK	1	TIMER19_CASCADE	0	0	
Main_Timer_19	timer_tclk_clk	EXT_REFCLK1	1	TIMER19_CASCADE	0	0	
Main_Timer_19	timer_tclk_clk	MAIN_CPSW_0.CPTS_GENF0	1	TIMER19_CASCADE	0	0	
Main_Timer_19	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF2	1	TIMER19_CASCADE	0	0	
Main_Timer_19	timer_tclk_clk	MAIN_NAVSS_0.CPTS0_GENF3	1	TIMER19_CASCADE	0	0	
Main_Timer_19	timer_tclk_clk	RCOSC	1	TIMER19_CASCADE	0	0	
Main_Timer_19	timer_tclk_clk	MAIN_TIMER_18.TIMER_PWM	1	TIMER19_CASCADE	1	0	
Main_UART_0	fclk_clk	MAIN_PLL1.HSDIV0	1				192
Main_UART_0	sclki_clk	MAIN_TIEOFF LOW	1				
Main_UART_0	vbusp_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_UART_1	fclk_clk	MAIN_PLL1.HSDIV0	1				192
Main_UART_1	sclki_clk	MAIN_TIEOFF LOW	1				
Main_UART_1	vbusp_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_UART_2	fclk_clk	MAIN_PLL1.HSDIV0	1				192
Main_UART_2	sclki_clk	MAIN_TIEOFF LOW	1				
Main_UART_2	vbusp_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_UART_3	fclk_clk	MAIN_PLL1.HSDIV0	1				192
Main_UART_3	sclki_clk	MAIN_TIEOFF LOW	1				
Main_UART_3	vbusp_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_UART_4	fclk_clk	MAIN_PLL1.HSDIV0	1				192
Main_UART_4	sclki_clk	MAIN_TIEOFF LOW	1				
Main_UART_4	vbusp_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_UART_5	fclk_clk	MAIN_PLL1.HSDIV0	1				192
Main_UART_5	sclki_clk	MAIN_TIEOFF LOW	1				
Main_UART_5	vbusp_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_UART_6	fclk_clk	MAIN_PLL1.HSDIV0	1				192
Main_UART_6	sclki_clk	MAIN_TIEOFF LOW	1				
Main_UART_6	vbusp_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_UART_7	fclk_clk	MAIN_PLL1.HSDIV0	1				192
Main_UART_7	sclki_clk	MAIN_TIEOFF LOW	1				
Main_UART_7	vbusp_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_UART_8	fclk_clk	MAIN_PLL1.HSDIV0	1				192
Main_UART_8	sclki_clk	MAIN_TIEOFF LOW	1				
Main_UART_8	vbusp_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_UART_9	fclk_clk	MAIN_PLL1.HSDIV0	1				192
Main_UART_9	sclki_clk	MAIN_TIEOFF LOW	1				
Main_UART_9	vbusp_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_USB3p0SS_0	aclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	1				500
Main_USB3p0SS_0	buf_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	2				250
Main_USB3p0SS_0	clk_lpm_clk	MAIN_PLL1.HSDIV7	1				24
Main_USB3p0SS_0	pclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_USB3p0SS_0	pipe_refclk	MAIN_SERDES_0.IP3_LN3_REFCLK	1	USB0 SerDes refclk Mux	0	0	

Copyright © 2024 Texas Instruments Incorporated

Table 5-33. Clock Mapping (continued)

Module	Clock Port	Clock	Divider	Control Register	Mux Value	Default Mux Value	Typical Frequency
Main_USB3p0SS_0	pipe_refclk	MAIN_SERDES_0.IP3_LN1_REFCLK	1	USB0 SerDes refclk Mux	1	0	
Main_USB3p0SS_0	pipe_rxclk	MAIN_SERDES_0.IP3_LN3_RXCLK	1	USB0 SerDes rxclk Mux	0	0	
Main_USB3p0SS_0	pipe_rxclk	MAIN_SERDES_0.IP3_LN1_RXCLK	1	USB0 SerDes rxclk Mux	1	0	
Main_USB3p0SS_0	pipe_rxfclk	MAIN_SERDES_0.IP3_LN3_RXFCLK	1	USB0 SerDes rxfclk Mux	0	0	
Main_USB3p0SS_0	pipe_rxfclk	MAIN_SERDES_0.IP3_LN1_RXFCLK	1	USB0 SerDes rxfclk Mux	1	0	
Main_USB3p0SS_0	pipe_txfclk	MAIN_SERDES_0.IP3_LN3_TXFCLK	1	USB0 SerDes txfclk Mux	0	0	
Main_USB3p0SS_0	pipe_txfclk	MAIN_SERDES_0.IP3_LN1_TXFCLK	1	USB0 SerDes txfclk Mux	1	0	
Main_USB3p0SS_0	pipe_txmclk	MAIN_SERDES_0.IP3_LN3_TXMCLK	1	USB0 SerDes txmclk Mux	0	0	
Main_USB3p0SS_0	pipe_txmclk	MAIN_SERDES_0.IP3_LN1_TXMCLK	1	USB0 SerDes txmclk Mux	1	0	
Main_USB3p0SS_0	usb2_apb_pclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_USB3p0SS_0	usb2_refclock_clk	HFOSC0	1	USB0_REFCLK_SEL	0	0	[19.2, 20, 24, 25, 26, 27]
Main_USB3p0SS_0	usb2_refclock_clk	HFOSC1	1	USB0_REFCLK_SEL	1	0	[19.2 - 27]
Main_USB3p0SS_0	usb2_tap_tck	MAIN_TAP_BS_JTAG__CLK	1				
Main_AC_ECC_Aggr_6	clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	2				250
Main_AC_ECC_Aggr_9	clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	2				250
Main_CPT2_Aggr_AC_0	vclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	2				250
Main_CPT2_Aggr_ACP_0	vclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	2				250
Main_CPT2_Aggr_HC_0	vclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	2				250
Main_CPT2_Aggr_MI_0	vclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	2				250
Main_CPT2_Aggr_MV_0	vclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	2				250
Main_CPT2_Aggr_RC_0	vclk_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	2				250
Main_CPT2_Probe_ACP_SRAM_SLV_0	probe_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	1				500
Main_CPT2_Probe_ACP_SRAM_SLV_0	vbus_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	2				250
Main_CPT2_Probe_ACP_SRAM_SLV_1	probe_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	1				500
Main_CPT2_Probe_ACP_SRAM_SLV_1	vbus_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	2				250
Main_CPT2_Probe_DMPAC_AC_SRAM_SLV_0	probe_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	1				500
Main_CPT2_Probe_DMPAC_AC_SRAM_SLV_0	vbus_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	2				250
Main_CPT2_Probe_NAVSS_AC_DDR_SLV_0	probe_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	1				500
Main_CPT2_Probe_NAVSS_AC_DDR_SLV_0	vbus_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	2				250
Main_CPT2_Probe_NAVSS_AC_DDR_SLV_1	probe_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	1				500
Main_CPT2_Probe_NAVSS_AC_DDR_SLV_1	vbus_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	2				250
Main_CPT2_Probe_NAVSS_AC_DDR_SLV_2	probe_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	1				500
Main_CPT2_Probe_NAVSS_AC_DDR_SLV_2	vbus_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	2				250
Main_CPT2_Probe_NAVSS_AC_SRAM_SLV_0	probe_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	1				500
Main_CPT2_Probe_NAVSS_AC_SRAM_SLV_0	vbus_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	2				250
Main_CPT2_Probe_VPAC_AC_SRAM_SLV_0	probe_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	1				500
Main_CPT2_Probe_VPAC_AC_SRAM_SLV_0	vbus_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	2				250

Copyright © 2024 Texas Instruments Incorporated

Table 5-33. Clock Mapping (continued)

Module	Clock Port	Clock	Divider	Control Register	Mux Value	Default Mux Value	Typical Frequency
Main_CTRL_MMR_0	vbusp_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_DebugSS_0	atb_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	2				250
Main_DebugSS_0	core_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_DebugSS_0	jtag_tck	MAIN_TAP_BS_JTAG__CLK	1				
Main_DebugSS_0	p1500_wrck	main_dft_clk	1				
Main_DebugSS_0	trexpt_clk	MAIN_PLL2.HSDIV3	1				200
Main_ECC_Aggr_R5_0_0	clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_ECC_Aggr_R5_0_1	clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_ECC_Aggr_R5_1_0	clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_ECC_Aggr_R5_1_1	clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_EFUSE_CTRL_0	pll_ctrl_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_EFUSE_CTRL_0	wkup_osc0_clk	HFOSC0	1	MAIN_eFUSE_SYSCLK SEL	0	0	[19.2, 20, 24, 25, 26, 27]
Main_EFUSE_CTRL_0	wkup_osc0_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4	MAIN_eFUSE_SYSCLK SEL	1	0	125
Main_HC_ECC_Aggr_5	clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	2				250
Main_Infra_ECC_Aggr_0	clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	2				250
MAIN_INTROUTER_CMP_EVENT_0	intr_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
MAIN_INTROUTER_GPIOMUX_0	intr_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_IP_ECC_Aggr_6	clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	2				250
Main_LVL_INTROUTER_Main2MCU_0	intr_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_NAVSS_ECC_Aggr_10	clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	2				250
Main_PBIST_AC_DMPAC	clk1_clk	MAIN_PBIST_CLK	1				
Main_PBIST_AC_DMPAC	clk2_clk	MAIN_PBIST_CLK	1				
Main_PBIST_AC_DMPAC	clk3_clk	MAIN_PBIST_CLK	1				
Main_PBIST_AC_DMPAC	clk4_clk	MAIN_PBIST_CLK	1				
Main_PBIST_AC_DMPAC	clk5_clk	MAIN_PBIST_CLK	1				
Main_PBIST_AC_DMPAC	clk6_clk	MAIN_PBIST_CLK	1				
Main_PBIST_AC_DMPAC	clk7_clk	MAIN_PBIST_CLK	1				
Main_PBIST_AC_DMPAC	clk8_clk	MAIN_PBIST_CLK	1				
Main_PBIST_AC_DMPAC	tclk_clk	MAIN_PBIST_CLK	1				
Main_PBIST_AC_EDP_DSI	clk1_clk	MAIN_PBIST_CLK	1				
Main_PBIST_AC_EDP_DSI	clk2_clk	MAIN_PBIST_CLK	1				
Main_PBIST_AC_EDP_DSI	clk3_clk	MAIN_PBIST_CLK	1				
Main_PBIST_AC_EDP_DSI	clk4_clk	MAIN_PBIST_CLK	1				
Main_PBIST_AC_EDP_DSI	clk5_clk	MAIN_PBIST_CLK	1				
Main_PBIST_AC_EDP_DSI	clk6_clk	MAIN_PBIST_CLK	1				
Main_PBIST_AC_EDP_DSI	clk7_clk	MAIN_PBIST_CLK	1				
Main_PBIST_AC_EDP_DSI	clk8_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_PBIST_AC_EDP_DSI	tclk_clk	MAIN_PBIST_CLK	1				
Main_PBIST_AC_ENC_DEC	clk1_clk	MAIN_PBIST_CLK	1				
Main_PBIST_AC_ENC_DEC	clk2_clk	MAIN_PBIST_CLK	1				
Main_PBIST_AC_ENC_DEC	clk3_clk	MAIN_PBIST_CLK	1				
Main_PBIST_AC_ENC_DEC	clk4_clk	MAIN_PBIST_CLK	1				
Main_PBIST_AC_ENC_DEC	clk5_clk	MAIN_PBIST_CLK	1				
Main_PBIST_AC_ENC_DEC	clk6_clk	MAIN_PBIST_CLK	1				

Copyright © 2024 Texas Instruments Incorporated

Table 5-33. Clock Mapping (continued)

Module	Clock Port	Clock	Divider	Control Register	Mux Value	Default Mux Value	Typical Frequency
Main_PBIST_AC_ENC_DEC	clk7_clk	MAIN_PBIST_CLK	1				
Main_PBIST_AC_ENC_DEC	clk8_clk	MAIN_PBIST_CLK	1				
Main_PBIST_AC_ENC_DEC	tclk_clk	MAIN_PBIST_CLK	1				
Main_PBIST_AC_VPAC	clk1_clk	MAIN_PBIST_CLK	1				
Main_PBIST_AC_VPAC	clk2_clk	MAIN_PBIST_CLK	1				
Main_PBIST_AC_VPAC	clk3_clk	MAIN_PBIST_CLK	1				
Main_PBIST_AC_VPAC	clk4_clk	MAIN_PBIST_CLK	1				
Main_PBIST_AC_VPAC	clk5_clk	MAIN_PBIST_CLK	1				
Main_PBIST_AC_VPAC	clk6_clk	MAIN_PBIST_CLK	1				
Main_PBIST_AC_VPAC	clk7_clk	MAIN_PBIST_CLK	1				
Main_PBIST_AC_VPAC	clk8_clk	MAIN_PBIST_CLK	1				
Main_PBIST_AC_VPAC	tclk_clk	MAIN_PBIST_CLK	1				
Main_PBIST_HC	clk1_clk	MAIN_PBIST_CLK	1				
Main_PBIST_HC	clk2_clk	MAIN_PBIST_CLK	1				
Main_PBIST_HC	clk3_clk	MAIN_PBIST_CLK	1				
Main_PBIST_HC	clk4_clk	MAIN_PBIST_CLK	1				
Main_PBIST_HC	clk5_clk	MAIN_PBIST_CLK	1				
Main_PBIST_HC	clk6_clk	MAIN_PBIST_CLK	1				
Main_PBIST_HC	clk7_clk	MAIN_PBIST_CLK	1				
Main_PBIST_HC	clk8_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_PBIST_HC	tclk_clk	MAIN_PBIST_CLK	1				
Main_PBIST_Infra_0	clk1_clk	MAIN_PBIST_CLK	1				
Main_PBIST_Infra_0	clk2_clk	MAIN_PBIST_CLK	1				
Main_PBIST_Infra_0	clk3_clk	MAIN_PBIST_CLK	1				
Main_PBIST_Infra_0	clk4_clk	MAIN_PBIST_CLK	1				
Main_PBIST_Infra_0	clk5_clk	MAIN_PBIST_CLK	1				
Main_PBIST_Infra_0	clk6_clk	MAIN_PBIST_CLK	1				
Main_PBIST_Infra_0	clk7_clk	MAIN_PBIST_CLK	1				
Main_PBIST_Infra_0	clk8_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_PBIST_Infra_0	tclk_clk	MAIN_PBIST_CLK	1				
Main_PBIST_Infra_1	clk1_clk	MAIN_PBIST_CLK	1				
Main_PBIST_Infra_1	clk2_clk	MAIN_PBIST_CLK	1				
Main_PBIST_Infra_1	clk3_clk	MAIN_PBIST_CLK	1				
Main_PBIST_Infra_1	clk4_clk	MAIN_PBIST_CLK	1				
Main_PBIST_Infra_1	clk5_clk	MAIN_PBIST_CLK	1				
Main_PBIST_Infra_1	clk6_clk	MAIN_PBIST_CLK	1				
Main_PBIST_Infra_1	clk7_clk	MAIN_PBIST_CLK	1				
Main_PBIST_Infra_1	clk8_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_PBIST_Infra_1	tclk_clk	MAIN_PBIST_CLK	1				
Main_PBIST_NAVSS	clk1_clk	MAIN_PBIST_CLK	1				
Main_PBIST_NAVSS	clk2_clk	MAIN_PBIST_CLK	1				
Main_PBIST_NAVSS	clk3_clk	MAIN_PBIST_CLK	1				
Main_PBIST_NAVSS	clk4_clk	MAIN_PBIST_CLK	1				
Main_PBIST_NAVSS	clk5_clk	MAIN_PBIST_CLK	1				
Main_PBIST_NAVSS	clk6_clk	MAIN_PBIST_CLK	1				
Main_PBIST_NAVSS	clk7_clk	MAIN_PBIST_CLK	1				
Main_PBIST_NAVSS	clk8_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_PBIST_NAVSS	tclk_clk	MAIN_PBIST_CLK	1				
Main_PBIST_Pulsar_0	clk1_clk	MAIN_PBIST_CLK	1				
Main_PBIST_Pulsar_0	clk2_clk	MAIN_PBIST_CLK	1				
Main_PBIST_Pulsar_0	clk3_clk	MAIN_PBIST_CLK	1				
Main_PBIST_Pulsar_0	clk4_clk	MAIN_PBIST_CLK	1				
Main_PBIST_Pulsar_0	clk5_clk	MAIN_PBIST_CLK	1				

Copyright © 2024 Texas Instruments Incorporated

Table 5-33. Clock Mapping (continued)

Module	Clock Port	Clock	Divider	Control Register	Mux Value	Default Mux Value	Typical Frequency
Main_PBIST_Pulsar_0	clk6_clk	MAIN_PBIST_CLK	1				
Main_PBIST_Pulsar_0	clk7_clk	MAIN_PBIST_CLK	1				
Main_PBIST_Pulsar_0	clk8_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_PBIST_Pulsar_0	tclk_clk	MAIN_PBIST_CLK	1				
Main_PBIST_Pulsar_1	clk1_clk	MAIN_PBIST_CLK	1				
Main_PBIST_Pulsar_1	clk2_clk	MAIN_PBIST_CLK	1				
Main_PBIST_Pulsar_1	clk3_clk	MAIN_PBIST_CLK	1				
Main_PBIST_Pulsar_1	clk4_clk	MAIN_PBIST_CLK	1				
Main_PBIST_Pulsar_1	clk5_clk	MAIN_PBIST_CLK	1				
Main_PBIST_Pulsar_1	clk6_clk	MAIN_PBIST_CLK	1				
Main_PBIST_Pulsar_1	clk7_clk	MAIN_PBIST_CLK	1				
Main_PBIST_Pulsar_1	clk8_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_PBIST_Pulsar_1	tclk_clk	MAIN_PBIST_CLK	1				
Main_PLL_MMR_0	vbusp_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_PLS_INTROUTER_Main2MCU_0	intr_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_PSC_Wrap_0	clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_PSC_Wrap_0	slow_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	24				21
Main_RC_ECC_Aggr_4	clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	2				250
Main_SEC_MMR_0	vbusp_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_TimeSync_INTROUTER_0	intr_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_SerDes_0	clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	4				125
Main_SerDes_0	cmn_refclk_m	MAIN_SERDES_0_REFCLK_N	1				
Main_SerDes_0	cmn_refclk_p	MAIN_SERDES_0_REFCLK_P	1				
Main_SerDes_0	core_ref_clk	HFOSC0	1	SERDES0_CORE_REF CLK	0	0	[19.2, 20, 24, 25, 26, 27]
Main_SerDes_0	core_ref_clk	HFOSC1	1	SERDES0_CORE_REF CLK	1	0	[19.2 - 27]
Main_SerDes_0	core_ref_clk	MAIN_PLL3.HSDIV4	1	SERDES0_CORE_REF CLK	2	0	156
Main_SerDes_0	core_ref_clk	MAIN_PLL2.HSDIV4	1	SERDES0_CORE_REF CLK	3	0	100
Main_SerDes_0	ip1_ln0_txclk	MAIN_DSS_EDP_0.PHY_LN0_TXCL K	1				
Main_SerDes_0	ip1_ln1_txclk	MAIN_DSS_EDP_0.phy_ln1_txclk	1				
Main_SerDes_0	ip1_ln2_txclk	MAIN_DSS_EDP_0.phy_ln2_txclk	1	SerDes0_IP1_LN2_TXC LK	0	0	
Main_SerDes_0	ip1_ln2_txclk	MAIN_DSS_EDP_0.PHY_LN0_TXCL K	1	SerDes0_IP1_LN2_TXC LK	1	0	
Main_SerDes_0	ip1_ln3_txclk	MAIN_DSS_EDP_0.phy_ln3_txclk	1	SerDes0_IP1_LN3_TXC LK	0	0	
Main_SerDes_0	ip1_ln3_txclk	MAIN_DSS_EDP_0.phy_ln1_txclk	1	SerDes0_IP1_LN3_TXC LK	1	0	
Main_SerDes_0	ip2_ln0_txclk	MAIN_PCIE_0.pcie_lane0_txclk	1				
Main_SerDes_0	ip2_ln1_txclk	MAIN_PCIE_0.pcie_lane1_txclk	1				
Main_SerDes_0	ip2_ln2_txclk	MAIN_PCIE_0.pcie_lane2_txclk	1				
Main_SerDes_0	ip2_ln3_txclk	MAIN_PCIE_0.pcie_lane3_txclk	1				
Main_SerDes_0	ip3_ln0_txclk	MAIN_TIEOFF LOW	1				
Main_SerDes_0	ip3_ln1_txclk	Main_USB3p0SS_0.pipe_txclk	1				
Main_SerDes_0	ip3_ln2_txclk	MAIN_TIEOFF LOW	1				
Main_SerDes_0	ip3_ln3_txclk	Main_USB3p0SS_0.pipe_txclk	1				
Main_SerDes_0	ip4_ln0_txclk	Ivusr_dual_main_0.vusrx_ln0_txclk	1				
Main_SerDes_0	ip4_ln1_txclk	Ivusr_dual_main_0.vusrx_ln1_txclk	1				

Copyright © 2024 Texas Instruments Incorporated

Table 5-33. Clock Mapping (continued)

Module	Clock Port	Clock	Divider	Control Register	Mux Value	Default Mux Value	Typical Frequency
Main_SerDes_0	ip4_ln2_txclk	lvusr_dual_main_0.vusrx_ln2_txclk	1				
Main_SerDes_0	ip4_ln3_txclk	lvusr_dual_main_0.vusrx_ln3_txclk	1				
Main_SerDes_0	tap_tck	MAIN_TAP_BS_JTAG__CLK	1				
Main_Hyperlink_0	v0_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	1				500
Main_Hyperlink_0	v0_rxpm_clk	MAIN_HYPERLINK_0.RXPMCLK	1				
Main_Hyperlink_0	v0_txfl_clk	MAIN_HYPERLINK_0.TXFLCLK	1				
Main_Hyperlink_0	v1_clk	MAIN_SYSCLK0 (MAIN_PLL0.HSDIV0)	1				500
Main_Hyperlink_0	v1_rxpm_clk	MAIN_HYPERLINK_1.RXPMCLK	1				
Main_Hyperlink_0	v1_txfl_clk	MAIN_HYPERLINK_1.TXFLCLK	1				
Main_Hyperlink_0	vusrx_ln0_refclk	MAIN_SERDES_0.ip4_ln0_refclk	1				
Main_Hyperlink_0	vusrx_ln0_rxclk	MAIN_SERDES_0.ip4_ln0_rxclk	1				
Main_Hyperlink_0	vusrx_ln0_rxfclk	MAIN_SERDES_0.ip4_ln0_rxfclk	1				
Main_Hyperlink_0	vusrx_ln0_txfclk	MAIN_SERDES_0.ip4_ln0_txfclk	1				
Main_Hyperlink_0	vusrx_ln0_txmclk	MAIN_SERDES_0.ip4_ln0_txmclk	1				
Main_Hyperlink_0	vusrx_ln1_refclk	MAIN_SERDES_0.ip4_ln1_refclk	1				
Main_Hyperlink_0	vusrx_ln1_rxclk	MAIN_SERDES_0.ip4_ln1_rxclk	1				
Main_Hyperlink_0	vusrx_ln1_rxfclk	MAIN_SERDES_0.ip4_ln1_rxfclk	1				
Main_Hyperlink_0	vusrx_ln1_txfclk	MAIN_SERDES_0.ip4_ln1_txfclk	1				
Main_Hyperlink_0	vusrx_ln1_txmclk	MAIN_SERDES_0.ip4_ln1_txmclk	1				
Main_Hyperlink_0	vusrx_ln2_refclk	MAIN_SERDES_0.ip4_ln2_refclk	1				
Main_Hyperlink_0	vusrx_ln2_rxclk	MAIN_SERDES_0.ip4_ln2_rxclk	1				
Main_Hyperlink_0	vusrx_ln2_rxfclk	MAIN_SERDES_0.ip4_ln2_rxfclk	1				
Main_Hyperlink_0	vusrx_ln2_txfclk	MAIN_SERDES_0.ip4_ln2_txfclk	1				
Main_Hyperlink_0	vusrx_ln2_txmclk	MAIN_SERDES_0.ip4_ln2_txmclk	1				
Main_Hyperlink_0	vusrx_ln3_refclk	MAIN_SERDES_0.ip4_ln3_refclk	1				
Main_Hyperlink_0	vusrx_ln3_rxclk	MAIN_SERDES_0.ip4_ln3_rxclk	1				
Main_Hyperlink_0	vusrx_ln3_rxfclk	MAIN_SERDES_0.ip4_ln3_rxfclk	1				
Main_Hyperlink_0	vusrx_ln3_txfclk	MAIN_SERDES_0.ip4_ln3_txfclk	1				
Main_Hyperlink_0	vusrx_ln3_txmclk	MAIN_SERDES_0.ip4_ln3_txmclk	1				

Copyright © 2024 Texas Instruments Incorporated

5.4.4 Clock Inputs

5.4.4.1 Overview

Various external clock inputs are needed to drive the device. Summary of these input clock signals is given below. Some of those inputs are bidirectional and are marked as inputs/outputs in the summary:

- High frequency oscillators
  - WKUP_OSC0_XO/WKUP_OSC0_XI — external main crystal interface pins of the internal oscillator which sources a reference clock. Provides reference clock to PLLs within WKUP/MCU and MAIN domain.
  - OSC1_XO/OSC1_XI — external main crystal interface pins connected to internal oscillator which sources reference clock. Provides reference clock to PLLs within MAIN domain. This high-frequency oscillator is used to provide audio clock frequencies to MCASPs.
- Low frequency oscillator
  - Low frequency digital input – WKUP_LF_CLKIN - Low Frequency 32k digital clock input, optionally sourced from an external PMIC or other clock source. This SoC does not support a LFOSC crystal input.
- General purpose clock inputs
  - MCU_EXT_REFCLK0 — optional external system clock input (MCU domain).
  - EXT_REFCLK1— optional external system clock input (MAIN domain). Optionally PLL4 (AUDIO0 PLL).
- External video pixel clock inputs
  - VOUT0_EXTPCLKIN — optional for the DPI0 port of DSS.
- External CPTS reference clock inputs
  - MCU_CPTS0_RFT_CLK — CPTS reference clock inputs for MCU_CPTS_RFT_CLK.
  - CPTS0_RFT_CLK — CPTS reference clock inputs for CPTS_RFT_CLK.
- External audio reference clock input/output pins. For more information which clocks may be sourced as an outputs to the pins, see Section 12.5.2, *Multichannel Audio Serial Port (MCASP)*.
  - AUDIO_EXT_REFCLK0
  - AUDIO_EXT_REFCLK1

Note

Other clock inputs that are routed directly to the subsystems are described in the respective subsystem chapters.

5.4.4.2 Mapping of Clock Inputs

Table 5-34 lists the mapping for the device clock inputs.

Table 5-34. Mapping for Input Sources

Domain	Clock	Ball Mapping	Frequency List / Range	Type
WKUP	WKUP_HFOSC0_CLK	WKUP_OSC0_XI WKUP_OSC0_XO	19.2, 20, 24, 25, 26, or 27 MHz	External Clock / Crystal Pins
	WKUP_LF_CLKIN	WKUP_GPIO0_67	Intended to be low frequency ~32KHz	LVCMOS
	MCU_EXT_REFCLK0	WKUP_GPIO0_10	Up to 100 MHz	LVCMOS
MAIN	HFOSC1_CLK	OSC1_XI OSC1_X0	19.2, 20, 24, 25, 26, or 27 MHz For audio applications: 22.5792 MHz or 24.576 MHz	External Clock / Crystal Pins
	EXT_REFCLK1	EXT_REFCLK1	Up to 100 MHz	LVCMOS

Table 5-35 lists the internal RC-oscillator sources.

Table 5-35. Internal RC Oscillator Input Sources

Domain	RC Oscillator	Clock	Frequency
WKUP	WKUP_RC_OSC_12M	CLK_12M_RC	12.5 MHz

Copyright © 2024 Texas Instruments Incorporated

5.4.5 Clock Outputs

The device provides several system clock outputs. Summary of these output clock signals is as follows:

- **MCU_CLKOUT0**
  - Reference clock output for Ethernet PHYs (50 MHz or 25 MHz)
- **MCU_SYSCLKOUT0**
  - MCU_SYSCLK0 is divided by 4 and then sent out of the device as a LVCMOS clock signal (MCU_SYSCLKOUT0). This signal can be used to test if the main chip clock is functioning or not. This signal should not be used as a clock source for external devices on a board.
- **MCU_OBSCLK0**
  - On the clock output MCU_OBSCLK0, oscillators and PLLs clocks can be observed for tests and debug. This signal should not be used as a clock source for external devices on a board.
- **SYSCLKOUT0**
  - SYSCLK0 is divided by 4 and then sent out of the device as a LVCMOS clock signal (SYSCLKOUT0). This signal can be used to test if the main chip clock is functioning or not. This signal should not be used as a clock source for external devices on a board.
- **CLKOUT**
  - Reference clock output for Ethernet PHYs (50 MHz)
- **OBSCLK[1:0]**
  - On the clock output OBSCLK0/1, oscillators and PLLs clocks can be observed for tests and debug.

Other clock outputs, that are routed directly from subsystems to device pins, are described in the respective module chapter.

Observation clock pins - MCU_OBSCLK0, OBSCLK0, OBSCLK1, and OBSCLK2 serve the following purposes:

- During testing, PLLs on the device are configured to output all operating frequencies desired by each module to characterize PLL performance.
- System debug: during debug, clocks from various PLLs can be inspected for possible clues. Example clock glitches, lock loss etc.

Note

Maximum frequency supported on both MCU_OBSCLK0, OBSCLK0, OBSCLK1, and OBSCLK2 pins is 200 MHz. Hence the divider at the output of each OBSCLK mux must be programmed to meet this limitation.

Unlike the OBSCLK pins which can select several different clocks for output, system clock pins (MCU_SYSCLKOUT0 and SYSCLKOUT0) are hardwired to dedicated clock resources (see Section 5.4.5.2)

5.4.5.1 Observation Clock Pins

5.4.5.1.1 MCU_OBSCLK0 Pin

MCU_OBSCLK0 output is controlled by CTRLMMR_WKUP_MCU_OBSCLK_CTRL register in the WKUP_CTRL_MMR0 module; for more information about control registers, see *Control Module (CTRL_MMR)*. Two muxes are connected in series - MCU_OBSCLK0_MUX0 and MCU_OBSCLK0_MUX1, see Figure 5-10.

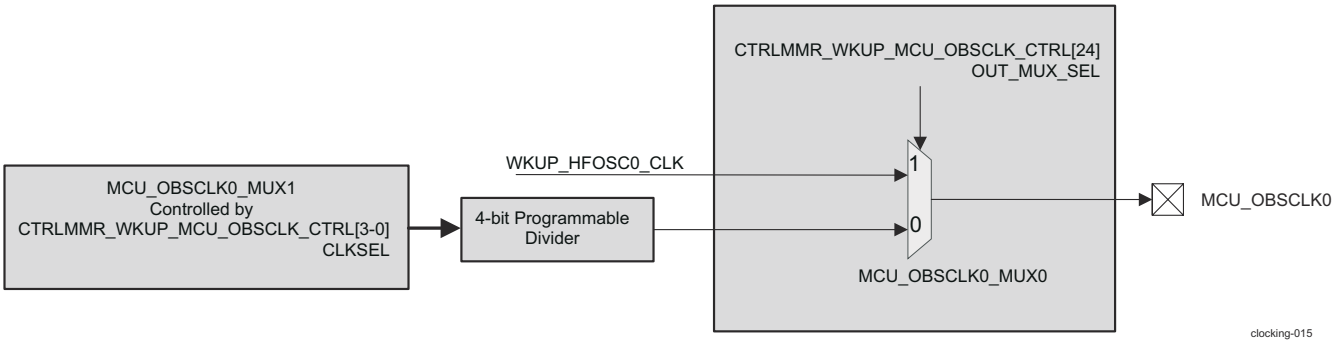


Figure 5-10. MCU_OBSCLK Muxes Diagram

Copyright © 2024 Texas Instruments Incorporated

How to select the output of MCU_OBSCLK0_MUX1 is described in Table 5-36.

Table 5-36. MCU_OBSCLK0_MUX1 Output Clock Selection

CTRLMMR_WKUP_MCU_OBSCLK_CTRL[2][3-0] CLK_SEL	MCU_OBSCLK0_MUX1 Output Clock Selection[1]
0x0	CLK_12M_RC
0x1	0 (GND)[3]
0x2	MCU_PLL0_HSDIV0_CLKOUT
0x3	WKUP_PLLCTL_OBSCLK (MCU_PLL0 input reference clock)
0x4	MCU_PLL1_HSDIV1_CLKOUT
0x5	MCU_PLL1_HSDIV2_CLKOUT
0x6	MCU_PLL1_HSDIV3_CLKOUT
0x7	MCU_PLL1_HSDIV4_CLKOUT
0x8	MCU_PLL2_HSDIV0_CLKOUT
0x9	CLK_32K
0xA	MCU_PLL2_HSDIV1_CLKOUT
0xB	MCU_PLL2_HSDIV2_CLKOUT
0xC	MCU_PLL2_HSDIV3_CLKOUT
0xD	MCU_PLL2_HSDIV4_CLKOUT
0xE	WKUP_HFOSC0_CLKOUT
0xF	WKUP_LF_CLKIN

(1) Software-controlled 4-bit divider is used to obtain the divided versions of the clocks passed to MCU_OBSCLK1 mux
(2) For more information about control registers, see *Control Module (CTRL_MMR)*.
(3) GND - ground

The value of the software-controlled 4-bit divider is determined by register CTRLMMR_WKUP_MCU_OBSCLK_CTRL[11-8] MCU_OBSCLK_CTRL_CLK _DIV in the WKUP_CTRL_MMR0 module; for more information about control registers, see *Control Module (CTRL_MMR)*.

Note

MCU_OBSCLK1_MUX0 is provided as a low jitter output for WKUP_HFOSC0_CLK. In this configuration, CTRLMMR_WKUP_MCU_OBSCLK_CTRL[3:0] should be configured as 0001b (1, selecting a logical low signal) and CTRLMMR_WKUP_MCU_OBSCLK_CTRL[24] should be configured as 1.

5.4.5.1.2 OBSCLK0, OBSCLK1, and OBSCLK2 Pins

The OBSCLK0, OBSCLK1, and OBSCLK2 output pins are controlled simultaneously,so that the three pins are connected to the same signal. OBSCLK0, OBSCLK1, and OBSCLK2 outputs are controlled by CTRLMMR_OBSCLK0_CTRL register in the CTRL_MMR0 module; for more information about control registers, see *Control Module (CTRL_MMR)*. Figure 5-11 shows a block diagram of internal OBSCLK0 mux connections.

Note

OBSCLK2 has lot more jitter due to PD limitations/long routing. If the intent is to use OBSCLK2 to do anything beyond just observe clock toggle, customers should switch to OBSCLK0 or OBSCLK1.

Figure 5-11. OBSCLK0 Muxes Diagram

Copyright © 2024 Texas Instruments Incorporated

Table 5-37. OBSCLK0_MUX1_CLKOUT

CTRLMMR_OBSCLK1_CTRL[1][1-0] CLK_SEL	OBSCLK0_MUX1_CLKOUT Selection
0x0	MAIN_PLL7_HSDIV0_CLKOUT / 4
0x1	MAIN_PLL8_HSDIV0_CLKOUT / 8
0x2	
0x3	0 (GND)[2]

(1) For more information about control registers, see *Control Module (CTRL_MMR)*.
(2) GND - ground

Table 5-38. OBSCLK0, OBSCLK1, and OBSCLK2 Clock Selection

CTRLMMR_OBSCLK0_CTRL[2][4-0] CLK_SEL	OBSCLK0, OBSCLK1, and OBSCLK2 Selection [1]
0x0	MAIN_PLL0_HSDIV0_CLKOUT
0x1	MAIN_PLL1_HSDIV0_CLKOUT
0x2	MAIN_PLL2_HSDIV0_CLKOUT
0x3	MAIN_PLL3_HSDIV0_CLKOUT
0x4	MAIN_PLL4_HSDIV0_CLKOUT
0x5	MAIN_PLL5_HSDIV0_CLKOUT
0x6	MAIN_PLL6_HSDIV0_CLKOUT
0x7	0 (GND)[3]
0x8	0 (GND)[3]
0x9	0 (GND)[3]
0xA	0 (GND)[3]
0xB	0 (GND)[3]
0xC	MAIN_PLL12_HSDIV0_CLKOUT
0xD	Input from OBSCLK0_MUX1_CLKOUT
0xE	MAIN_PLL14_HSDIV0_CLKOUT
0xF	
0x10	MAIN_PLL16_HSDIV0_CLKOUT
0x11	MAIN_PLL17_HSDIV0_CLKOUT
0x12	
0x13	MAIN_PLL19_HSDIV0_CLKOUT
0x14	UFS MPHY_TX_REF_SYMBOLCLK
0x15	UFS MPHY_M31_VCO_19P2M_CLK
0x16	UFS MPHY_M31_VCO_26M_CLK
0x17	
0x18	MAIN_PLL24_HSDIV0_CLKOUT
0x19	MAIN_PLL25_HSDIV0_CLKOUT
0x1A	CPTS_GENF3
0x1B	CLK_12M_RC
0x1C	WKUP_LF_CLKIN
0x1D	PLLCTRL_OBSCLK (PLL0 input reference clock)
0x1E	HFOSC1_CLK
0x1F	WKUP_HFOSC0_CLKOUT

(1) Software-controlled 8-bit divider is used to obtain the divided versions of the clocks passed to outpt pins
(2) For more information about control registers, see *Control Module (CTRL_MMR)*.
(3) GND - ground

The value of the software-controlled 8-bit divider is determined by register CTRLMMR_OBSCLK0_CTRL[15-8] OBSCLK_CTRL_CLK_DIV; for more information about control registers, see *Control Module (CTRL_MMR)*.

Copyright © 2024 Texas Instruments Incorporated

5.4.5.2 System Clock Pins

5.4.5.2.1 MCU_SYSCLKOUT0

MCU_SYSCLK0 is divided by 4 and then send out of the device as a LVCMOS clock signal (MCU_SYSCLKOUT0).

Note

MCU_SYSCLKOUT0 cannot be used as a clock source for external devices on the board.

5.4.5.2.2 SYSCLKOUT0

SYSCLK0 is divided by 4 and then send out of the device as a LVCMOS clock signal (SYSCLKOUT0). This signal can be used to test if the specific device clock is functioning or not.

Note

SYSCLKOUT0 cannot be used as a clock source for external devices on the board.

5.4.6 Device Oscillators

The device has the possibility to source clocks of two external high-frequency oscillators - WKUP_HFOSC0 and HFOSC1 and one external low-frequency clock input - WKUP_LF_CLKIN. The device has one internal RC oscillator - WKUP_RC_OSC_12M.

5.4.6.1 Device Oscillators Integration

5.4.6.1.1 Oscillators with External Crystal

By default, HFOSC1 is disabled, while WKUP_HFOSC0 is enabled after PORz.

All device oscillators with external crystals support a bypass mode. In this mode, an external clock can be driven on the XI Pin.

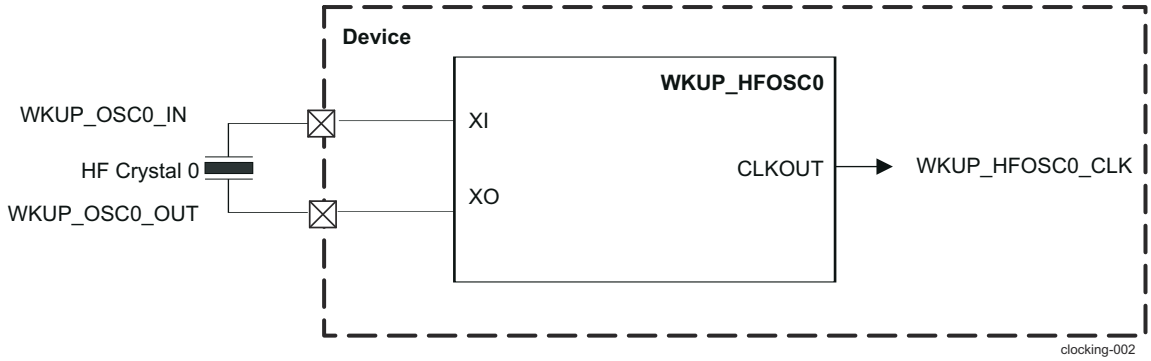


Figure 5-12. WKUP_HFOSC0 Integration Diagram

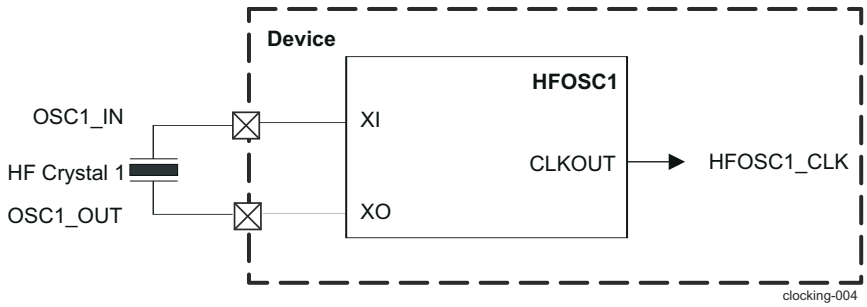


Figure 5-13. HFOSC1 Integration Diagram

Copyright © 2024 Texas Instruments Incorporated

5.4.6.1.2 Internal RC Oscillator

WKUP_RC_OSC_12M in always oscillate mode.

The RC clock output (WKUP_RC_OSC_12M) is used by the on-chip Reset Glue logic, Power Reset Generator (PRG) circuits, WKUP_HFOSC0 Loss Circuits, Dual Clock Comparator (DCC) and Timer Modules. Its frequency is targeted at 12.55MHz +/-10%.

Note

An output clock of WKUP_RC_OSC_12M is further divided down to generate a 32-kHz clock (CLK_32K). This clock is used by WKUP_DMSC0 and Timer Modules.

If an application needs accurate 32.768-kHz clock, then use the WKUP_LF_CLKIN external clock input.

5.4.6.2 Oscillator Clock Loss Detection

The device supports WKUP_HFOSC0 clock loss circuitry to detect when WKUP_HFOSC0_CLK stops toggling. If CTRLMMR_WKUP_MCU_PLL_CLKSEL[8] CLKLOSS_SWTCH_EN is set and WKUP_HFOSC0_CLK stops toggling condition is detected, the reference clock is switched from WKUP_HFOSC0_CLKOUT to WKUP_CLK_12M_RC to allow the device to operate with a slower clock. The WKUP_HFOSC0_CLK stops toggling condition is reported as an error to WKUP_ESM0 regardless of the value of CTRLMMR_WKUP_MCU_PLL_CLKSEL[8] CLKLOSS_SWTCH_EN. Integration diagram of WKUP_HFOSC0 clock loss detection is presented in Figure 5-14.

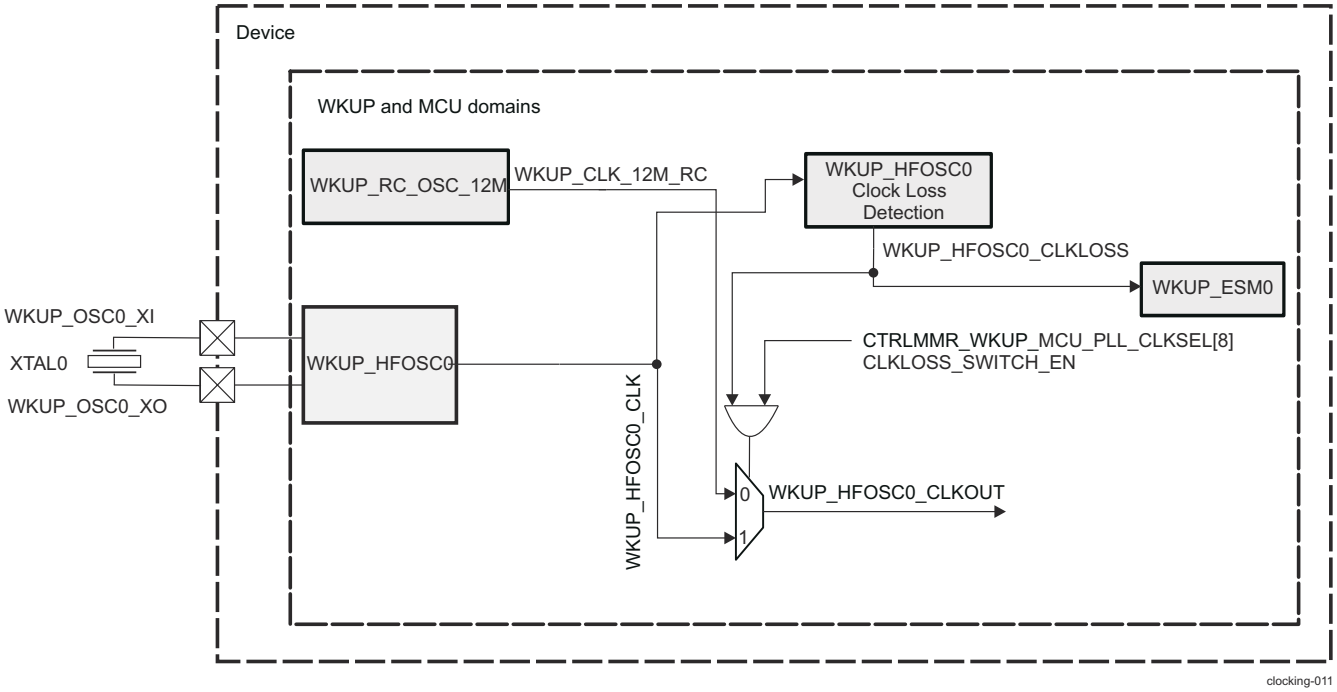


Figure 5-14. WKUP_HFOSC0 Clock Loss Detection Integration Diagram

WKUP_ESM0 can optionally generate an interrupt to WKUP_DMSC0 and MCU_R5FSS so they can save some critical contents such as error logging to scratch pad memory or external flash. ESM must also be configured to report this error on the SAFETY_ERRORn pin.

WKUP_HFOSC0 clock loss is a catastrophic failure since this clock is the most critical system clock. During the clock loss condition an external system intervention is required.

The clock loss mux is not a glitch free mux. The mux control is synchronized to the CLK_12M_RC clock. Since WKUP_HFOSC0_CLK has stopped switching the mux should transition to RC Clock without producing glitches.

Copyright © 2024 Texas Instruments Incorporated

Hence it is important to ensure that MCU_SAFETY_ERRORn is pulled low during this error condition. In clock-loss condition, the device reports the error to the external device through MCU_SAFETY_ERRORn pin - the pin is driven Low. The recovery mechanism is up to the external system (such as a PMIC to take action). For example, it can try a full system power cycle to see if the system recovers. If the system does not recover then, it has to take some other action such as to check system clocks, external crystal, supply rails.

PLL will lose the lock when clock loss is detected. During this time 12.5-MHz RC clock is output through the bypass muxes. The PLL will relock to a new frequency based on 12.5 MHz.

In an event of clock glitch which may potentially hang the device, then the WKUP_DMSC0 watchdog timer will expire and will generate an internal reset for the whole device.

Copyright © 2024 Texas Instruments Incorporated

5.4.7 PLLs

Phase-Locked Loop circuits (PLLs) in the device are clock generator PLLs, which multiply the lower-frequency reference clock up to the operating frequency of the respective subsystem(s).

5.4.7.1 WKUP and MCU Domains PLL Overview

There are total three PLLs in the device in WKUP/MCU domain:

- MCU_PLL0 (MCU R5FSS PLL) with WKUP_PLLCTRL0
- MCU_PLL1 (MCU PERIPHERAL PLL)
- MCU_PLL2 (MCU Command Platform Ethernet Switch (CPSW) PLL).

Overview of the device PLLs with their reference clock options in WKUP/MCU domain is shown on Figure 5-15. For more specific information about PLLs see Section 5.4.7.5, *PLLs Device-Specific Information*.

Copyright © 2024 Texas Instruments Incorporated

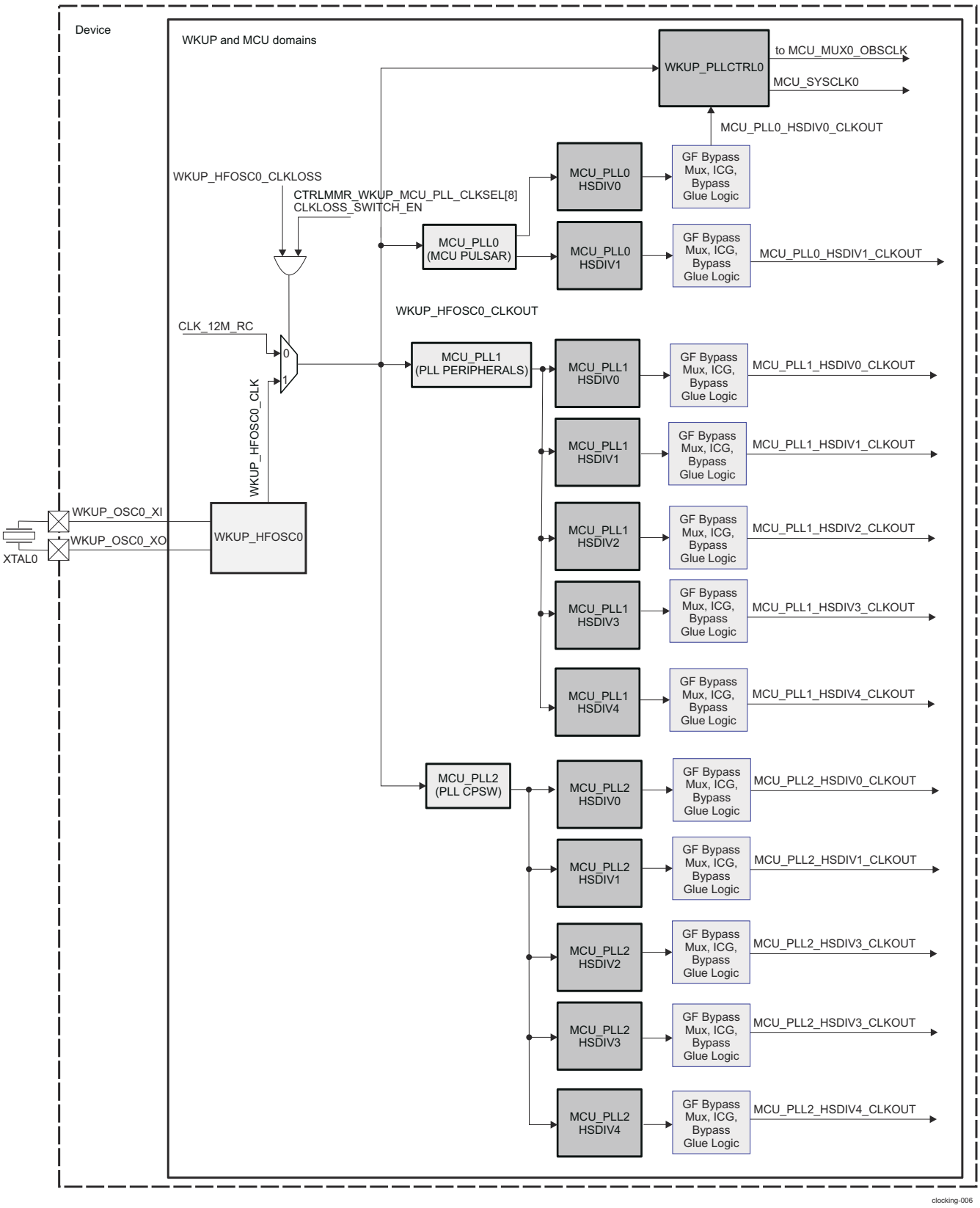


Figure 5-15. WKUP/MCU Domain PLLs Integration

5.4.7.2 MAIN Domain PLLs Overview

There are total sixteen PLLs in the device in MAIN domain:

Copyright © 2024 Texas Instruments Incorporated

- PLL0 (MAIN PLL) with PLLCTRL0
- PLL1 (PER0 PLL)
- PLL2 (PER1 PLL)
- PLL3 (CPSW9G PLL)
- PLL4 (AUDIO0 PLL)
- PLL5 (VIDEO PLL)
- PLL6 (GPU PLL)
- PLL7 (C7x PLL)
- PLL8 (ARM0 PLL)
- PLL12 (DDR PLL)
- PLL14 (R5FSS PLL)
- PLL16 (DSS PLL0)
- PLL17 (DSS PLL1)
- PLL19 (DSS PLL3)
- PLL25 (VISION PLL)
- PLL26 (DDR1)

Overview of the device PLLs with their reference clock options in MAIN domain is shown on Figure 5-16 and Figure 5-17. For more specific information about PLLs see Section 5.4.7.5, *PLLs Device-Specific Information*.

Note

The external muxes of choosing the reference clocks are glitch-free muxes.

Copyright © 2024 Texas Instruments Incorporated

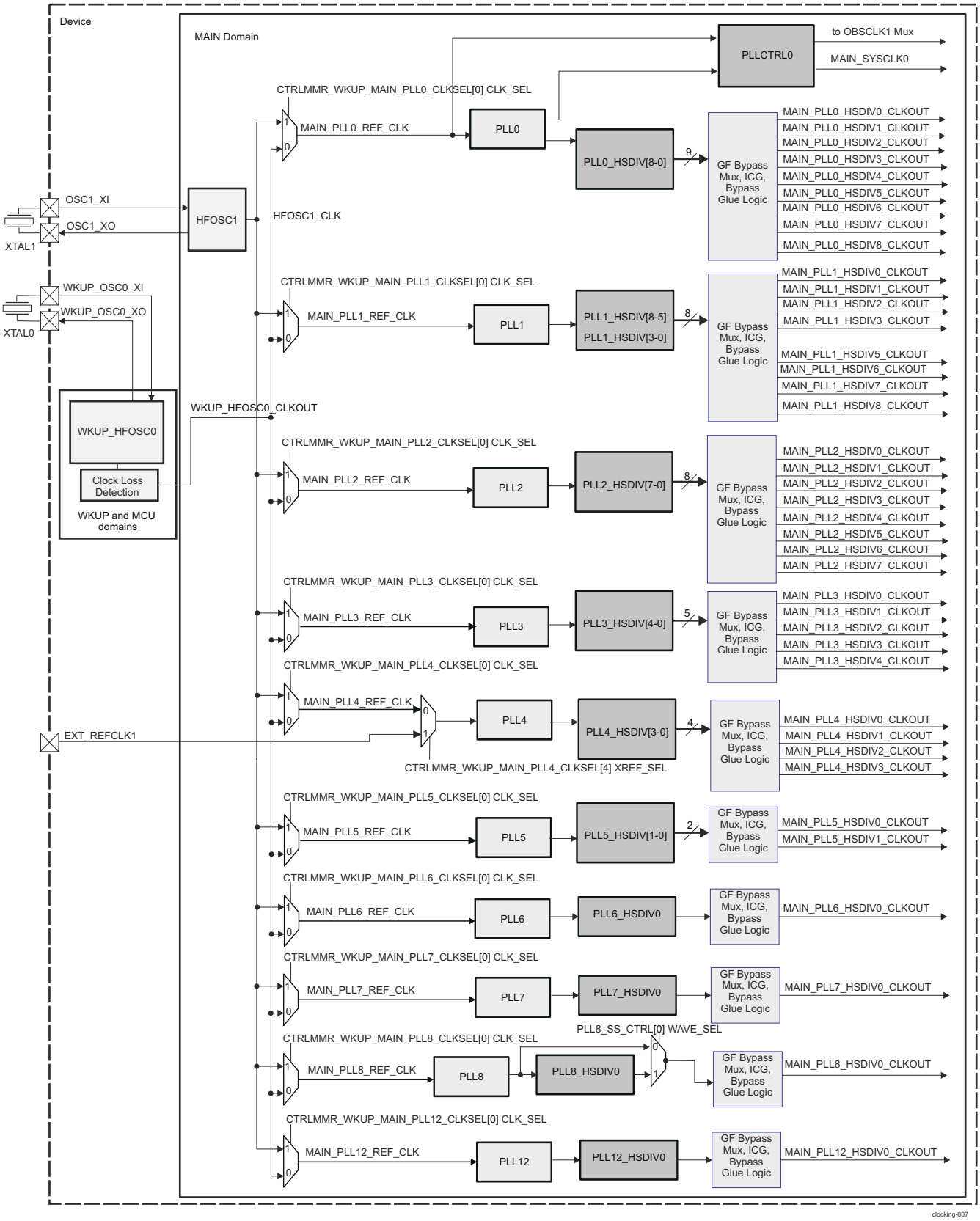


Figure 5-16. MAIN Domain PLLs Integration - Part 1

Copyright © 2024 Texas Instruments Incorporated

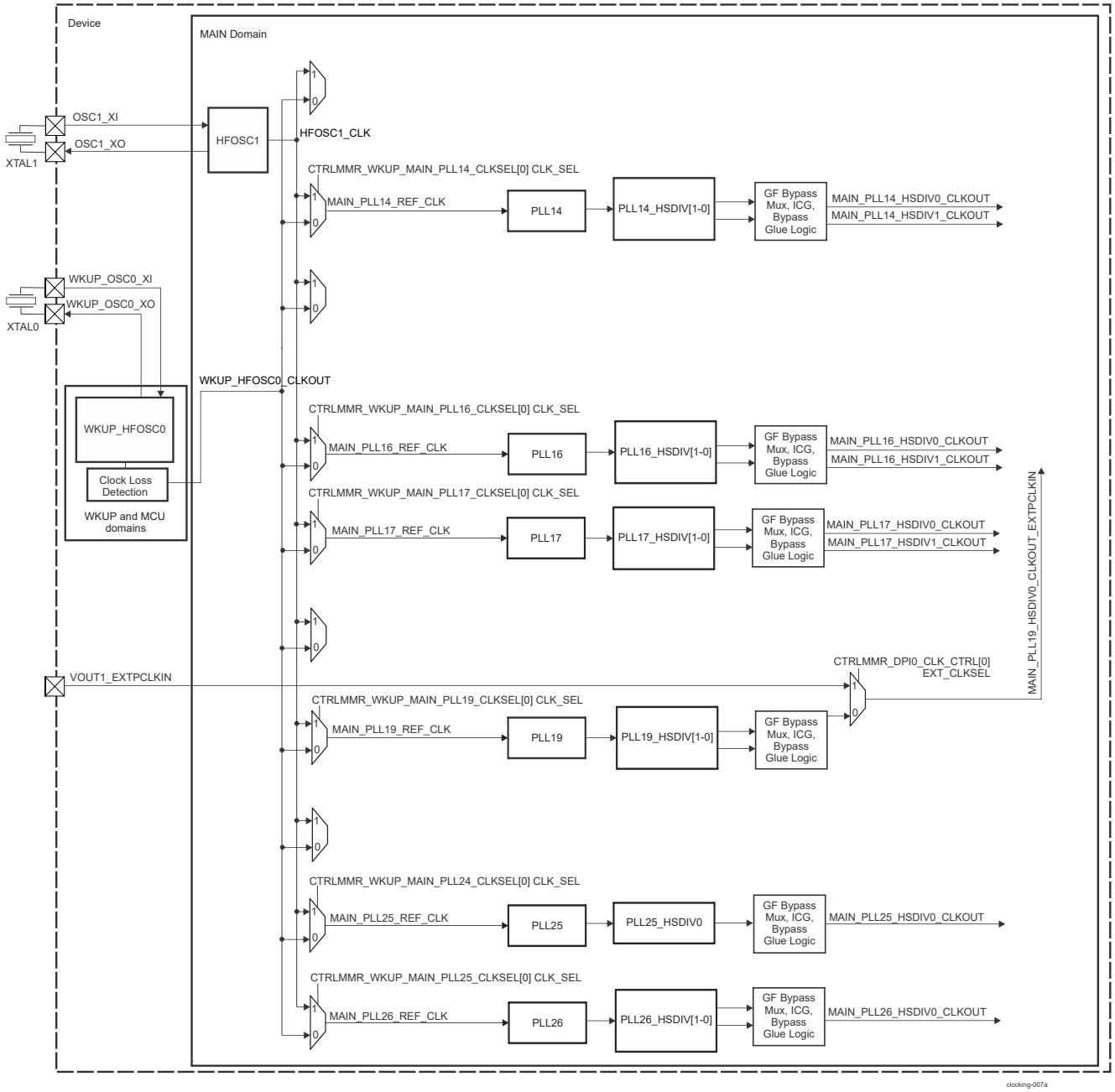


Figure 5-17. MAIN Domain PLLs Integration - Part 2

5.4.7.3 PLL Reference Clocks

5.4.7.3.1 PLLs in MCU Domain

The reference clocks MCU_PLL0_REF_CLK, MCU_PLL1_REF_CLK and MCU_PLL2_REF_CLK for the PLLs in MCU domain is chosen between the internal high-frequency (HF) oscillator with external crystal, WKUP_HFOSC0, and 12.5-MHz free-running RC oscillator. The selection for all PLLs is made through CTRLMMR_WKUP_MCU_PLL_CLKSEL[8] CLKLOSS_SWTCH_EN, see Table 5-39.

Table 5-39. PLL MCU Domain Reference Clock Selection - WKUP_HFOSC0_CLKOUT Selection

CTRLMMR_WKUP_MCU_PLL_CLKSEL[2][8] CLKLOSS_SWTCH_EN	MCU_PLLn_REF_CLK[1]
0 (default)	WKUP_HFOSC0_CLK

Copyright © 2024 Texas Instruments Incorporated

Table 5-39. PLL MCU Domain Reference Clock Selection - WKUP_HFOSC0_CLKOUT Selection (continued)

CTRLMMR_WKUP_MCU_PLL_CLKSEL[2][8] CLKLOSS_SWTCH_EN	MCU_PLLn_REF_CLK[1]
1	WKUP_CLK_12M_RC if clock loss is detected or WKUP_HFOSC0_CLK if clock loss is not detected

(1) n = 0, 1, 2
(2) For more information about control registers, see *Control Module (CTRL_MMR)*.

5.4.7.3.2 PLLs in MAIN Domain

Each PLL in MAIN domain has a dedicated register CTRLMMR_WKUP_MAIN_PLLn_CLKSEL in WKUP_CTRL_MMR0 to choose its reference clock.

MAIN_PLLn_REF_CLK (n = 0 to 8, 12 to 19, 23 to 25) clock source is selected by means of CTRLMMR_WKUP_MAIN_PLLn_CLKSEL[0] CLK_SEL bit; for more information about control registers, see *Control Module (CTRL_MMR)*. The encoding of CLK_SEL is shown in Table 5-40.

Table 5-40. PLL MAIN Domain Reference Clock Selection

CTRLMMR_WKUP_MAIN_PLLn_CLKSEL[1][0] CLK_SEL	MAIN_PLLn_ REF_CLK Selection [1]
0 (default)	WKUP_HFOSC0_CLK
1	HFOSC1_CLK

(1) n = 0 to 8, 12 to 19, 23 to 25

PLL4 (AUDIO0 PLL) input reference clocks have additional selection via mux configured in the CTRLMMR_WKUP_MAIN_PLL4_CLKSEL[4] XREF_SEL in WKUP_CTRL_MMR0 (see Table 5-41); for more information about control registers, see *Control Module (CTRL_MMR)*.

Table 5-41. PLL4 (AUDIO0 PLL) Clock Selection

CTRLMMR_WKUP_MAIN_PLL4_CLKSEL[4] XREF_SEL	PLL4 (AUDIO0 PLL) Ref Selection
0 (default)	MAIN_PLL4_REF_CLK already selected via MAIN_PLL4_REF_CLK[0] CLK_SEL
1	EXT_REFCLK1 (external clock input on device pins)

5.4.7.4 Generic PLL Overview

To generate high-frequency clocks, the device supports multiple on-chip PLLs controlled directly by the Top-level Clocking:PLLTS16FFCLAFRACF2.

Note

This chapter discusses only the PLLs that are directly controlled by the Top-level Clocking. The other PLLs embedded in and managed by other subsystems are described in their respective subsystems.

5.4.7.4.1 PLLs Output Clocks Parameters

Figure 5-18 shows the functional architecture of a generic PLL for PLLTS16FFCLAFRACF2.

Copyright © 2024 Texas Instruments Incorporated

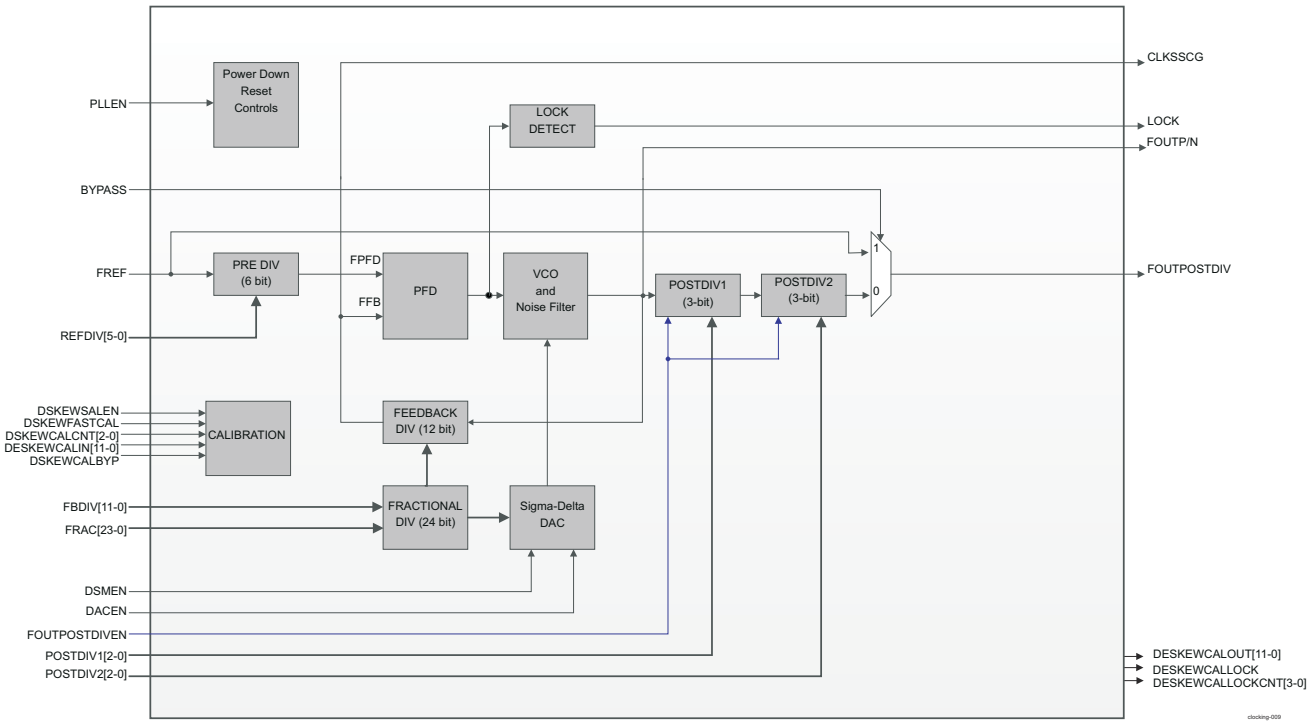


Figure 5-18. Generic PLL Functional Diagram for PLLTS16FFCLAFRACF2 Type

5.4.7.4.1.1 PLLs Input Clocks

As shown in Figure 5-18 for each type of PLL:

- FREF: Reference clock input is used to generate the synthesized clock but can also be used as the bypass clock for some outputs of the PLL whenever the PLL enters bypass mode. It is mandatory for the PLL clock synthesis.

5.4.7.4.1.2 PLL Output Clocks

5.4.7.4.1.2.1 PLLTS16FFCLAFRAC2 Type Output Clocks

Table 5-42 describes the output clocks of PLLTS16FFCLAFRAC2.

Table 5-42. PLLTS16FFCLAFRAC2 Output Clocks

Output	Description	Frequency
FOUTP	Positive phase VCO output (no post divider)	((FREF / REFDIV) * (FBDIV + FRAC))
FOUTN	Negative phase VCO output (no post divider)	((FREF / REFDIV) * (FBDIV + FRAC))
FOUTPOSTDIV	VCO-divided clock output.	FOUTP / (POSTDIV1*POSTDIV2)
CLKSSCG	Clock to SSMOD	(FREF / REFDIV)

Where:

- REFDIV is the software-configured division ratio binary value.
- FBDIV is the software-configured division ratio binary value.
- FRAC is the software-configured fractional division.
- POSTDIV1 is the software-configured division ratio binary value.
- POSTDIV2 is the software-configured division ratio binary value.

Copyright © 2024 Texas Instruments Incorporated

Note

POSTDIV1 and POSTDIV2 valid values are from 1 to 7. To ensure correct operation, POSTDIV1 must always be programmed to a value equal to or greater than POSTDIV2.

Note

For device-specific information about clock output parameters and syntesized clocks, see Table 5-47 and Table 5-49.

5.4.7.4.1.2.2 PLL Lock

PLL module outputs a lock status signal to indicate that the PLL has achieved frequency lock. When the PLL detects no cycle slips between the feedback clock (FFB) and reference clock FPFD (FREF/ REFDIV[5-0]) for 128 consecutive cycles, it asserts the lock signal high. When the PLL detects any cycle slip, lock signal will go low and stays low until it detects no cycle slip for 128 consecutive cycles. This lock signal is captured in <PLL_name>_STAT[0] LOCK bit. Software can read this bit to determine if the PLL has achieved frequency lock before selecting the PLL clock through external bypass mux.

When PLL losses lock, there is a hardware mechanism to automatically bypass the PLL clock to the reference clock (FREF) using the external glitch free mux. BYP_ON_LOCKLOSS bit in <PLL_name>_CTRL register enables this automatic bypass mode on PLL lock loss. When the PLL re-locks, the glitch free mux switches to PLL clock out.

The PLL lock signal is also routed to ESM module for error reporting. ESM can be configured to generate interrupts to MCU_R5FSS/R5FSS and WKUP_DMSC0 and assert Low on SAFETY_ERRORn pin on PLL Lock loss for further action.

5.4.7.4.1.2.3 HSDIVIDER

A PLL may contain up to 16 HSDIVIDER modules to produce more clocks with divided ratio based on the PLL synthesized clock frequency. HSDIVIDER provides only one output. The HSDIVIDER output clock frequency is given by the equations in Table 5-43.

Table 5-43. HSDIV_CLKOUT Frequency With PLL State

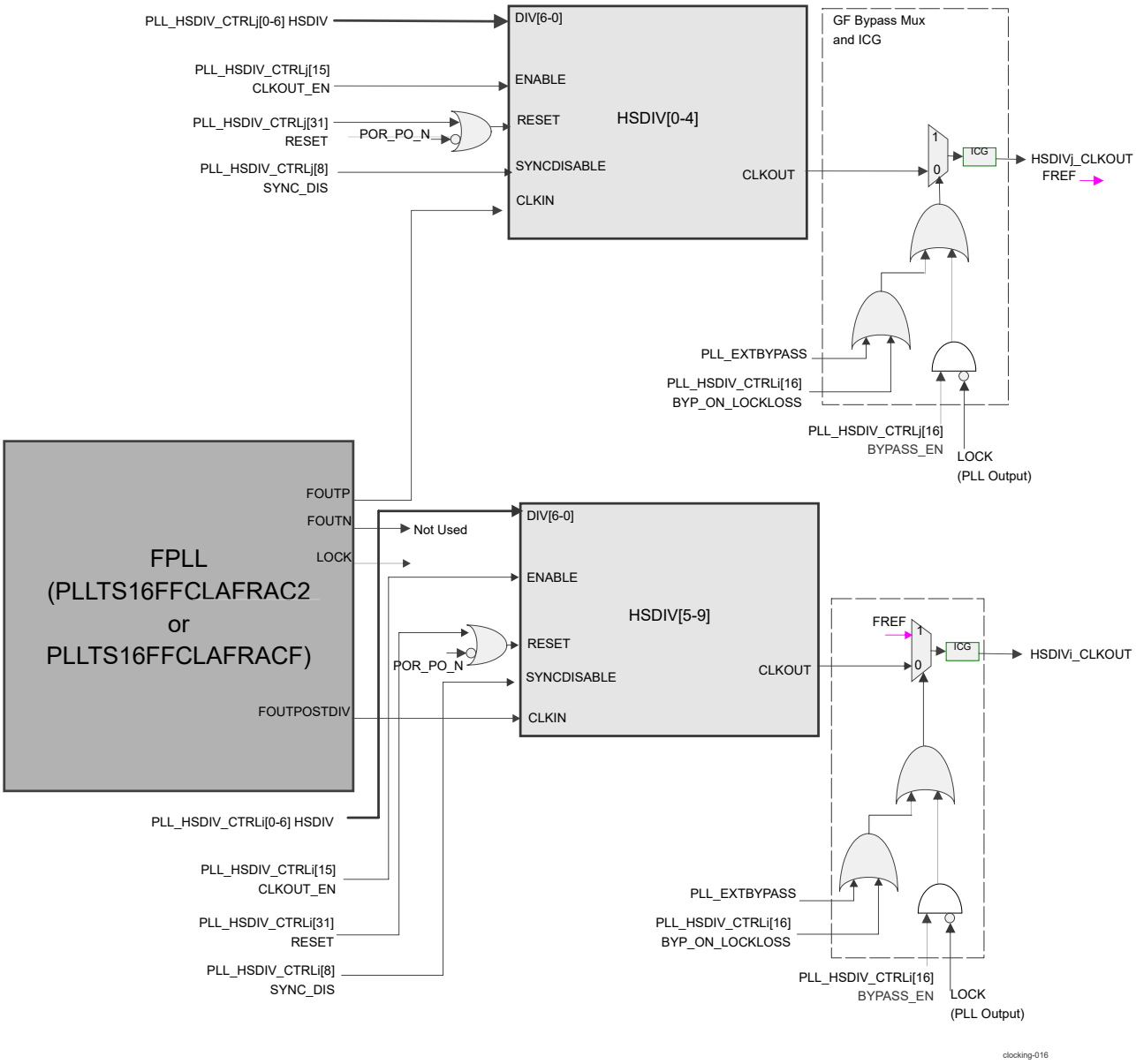
Equation	PLL Mode
HSDIV_CLKOUT = FOUTP / (HSDIV + 1)	Locked
HSDIV_CLKOUT = FREF	Before lock or during relock

Where:

- FOUTP is the PLL lock frequency. In case of PLLTS16FFCLVDESKEWC PLL, FOUTP is substituted with FOUT1X.
- HSDIV is the software-configured division ratio binary value.

Figure 5-19 describe the connection between a HSDIV and a specific type of PLL. Signals related only to this connections are shown in both figures.

Copyright © 2024 Texas Instruments Incorporated



(1) i = 0 to 4; j = 5 to 9

Figure 5-19. PLLTS16FFCLAFRAC2 / PLLTS16FFCLAFRACF and HSDIV Connection

5.4.7.4.1.2.4 ICG Module

ICG module ensures that when a particular HSDIV is not enabled the HSDIV clock output is gated off. This is a hardware module with no software control.

5.4.7.4.1.2.5 PLL Power Down

PLLTS16FFCLAFRACF2 is disabled with MCU_PLLn_CTRL[15] PLL_EN or PLLn_CTRL[15] PLL_EN.

5.4.7.4.1.2.6 PLL Calibration

PLLTS16FFCLAFRACF 2 has acalibration feature. The calibration settings are done via <PLL_Name>_CAL_CTRL register. The calibration status can be monitored by reading

Copyright © 2024 Texas Instruments Incorporated

<PLL_Name>_CAL_STAT register. For more information about device-specific information about calibration feature, see Table 5-51.

5.4.7.4.2 PLL Spread Spectrum Modulation Module

Spread Spectrum Modulation (SSMOD) modules on the device reduces Electro Magnetic Interference (EMI). It is a fully-digital circuit, used to modulate the frequency of the selected fractional PLLs. Programming options include selection of center spread or down spread, modulation depth, and modulation shape. The default modulation profile is triangular.

All PLLs in this family of device have SSMOD module, except for PLL24, which is the only PLL of PLLTS16FFCLVDESKEWC type.

SSMOD in the PLL is performed by changing the feedback divider (FRAC and FBDIV) in a triangular pattern. This varies the frequency of the output clock in a triangular pattern. The frequency of this pattern is the modulation frequency (f_m). It is programmed as a ratio of the CLKIN/4.

Note

Spread Spectrum modulation and calibration cannot be enabled together because calibration cannot be enabled when the PLL is configured for fractional mode.

Figure 5-20 shows a diagram of the spreading generation block.

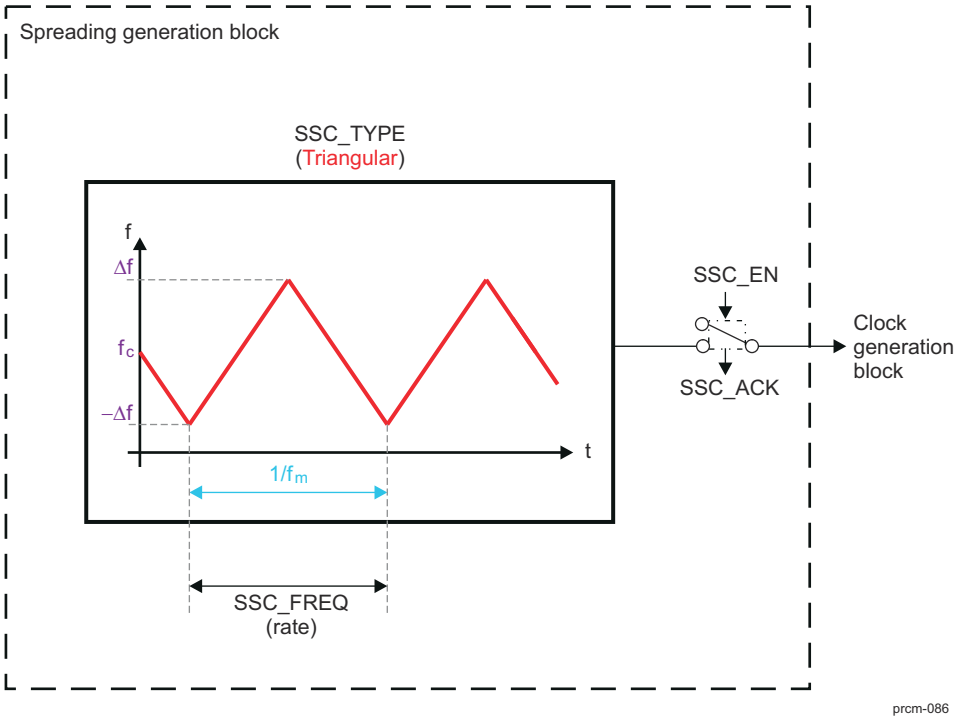


Figure 5-20. Spreading Generation Block Diagram

Note

f_c is the original output clock frequency.

f_m is the spreading frequency.

This additional block generates the required waveform used to reduce EMI. This waveform is then modulated with the initial signal to add some controlled deviation to the clock signal frequency, which spreads the energy of the clock and its harmonics into a band of frequencies, and then reduces EMI. SSMOD_DOWNSPREAD can control the position of the generated signal. It is controlled by the <PLL_name>_CTRL[4] DOWNSPREAD_EN

Copyright © 2024 Texas Instruments Incorporated

bit of the corresponding registers. If the DOWNSPREAD_EN bit is set to 1, the frequency spread on the lower side is twice the programmed value. The frequency spread on the higher side is 0.

The value of SSMOD_FREQ controls the rate of the generated signal. It can be programmed as a ratio of the reference clock: f_{ref} / 4. The value that must be programmed is calculated as follows:
ModFreqDivider = f_{ref} / (4 × f_m), where $f_m < f_{ref}$ / 70, $f_{ref} = f_{inp}$ / (1+N), and f_{inp} is the input clock for the PLL.

5.4.7.4.2.1 Definition of SSMOD

The aim of the SSMOD is to add a variation to the frequency of an original clock, which spreads the generated interference over a larger band of frequencies.
In theory, SSMOD means that the clock signal is varied around the desired frequency. For example, for a 1 GHz clock, the frequency might be 999.5 MHz at one time and 1.0005 GHz at another. Doing this constantly causes the power of the tone to be spread out more over a broader band of tight frequencies (centered at the desired tone). To realize this constant variation on the original signal, a modulation with an additional signal (called spreading waveform) is realized.
Creating an SSMOD by spreading the initial clock frequency is done by defining the following parameters:

- The spreading frequency (deviation), which is the ratio of the range of spreading frequency over the original clock frequency.

5.4.7.4.2.2 SSMOD Configuration

Table 5-44 describes the PLL SSMOD control bitfields. Figure 5-21 describes the connection between a SSMOD and a PLL.

Table 5-44. SSMOD related bitfields

Parameter	Register	Description
SSMOD enable control	<PLL_name>n_SS_CTRL[31] BYPASS_EN (For example, MCU_PLL0 - MCU_PLL0_SS_CTRL[31] BYPASS_EN)	Enable/disable the PLL SSMOD feature.
Type of spread	<PLL_name>n_SS_CTRL[4] DOWNSPREAD_EN (For example, MCU_PLL0 - MCU_PLL0_SS_CTRL[4] DOWNSPREAD_EN)	Selects center spread or down spread clock variance
Modulation dvider	<PLL_name>n_SS_SPREAD[19-16] MOD_DIV (For example, MCU_PLL0 - MCU_PLL0_SS_SPREAD[19-16] MOD_DIV)	Input clock divider. This divider sets the modulation frequency.
Spread Depth	<PLL_name>n_SS_SPREAD[4-0] SPREAD (For example, MCU_PLL0 - MCU_PLL0_SS_SPREAD[4-0] SPREAD)	Sets the spread modulation depth.

Copyright © 2024 Texas Instruments Incorporated

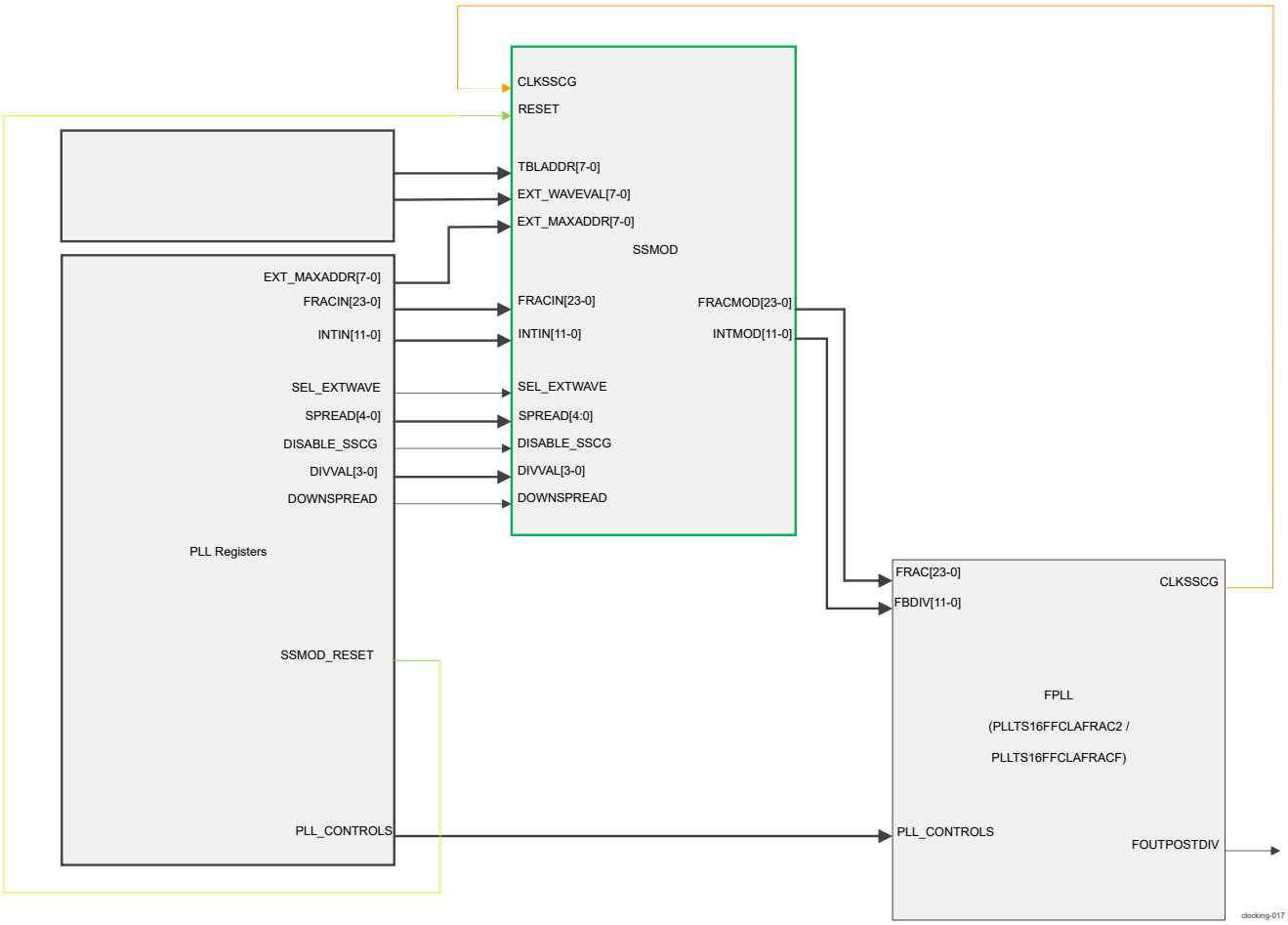


Figure 5-21. PLL and SSMOD Connection

Copyright © 2024 Texas Instruments Incorporated

5.4.7.5 PLLs Device-Specific Information

Table 5-45 lists the basic information of each of the three MCU PLLs.

Table 5-45. Basic Information of MCU_PLL0, MCU_PLL1, and MCU_PLL2

PLL name	Type	SSMOD Available	HSDIV Available	Input clock	Comments
MCU_PLL0	PLLTS16FFCLAFRACF2	Yes, see Table 5-44 for more information on SSMOD related bitfields/bits	HSDIV0 and HSDIV1	WKUP_HFOSC0_CLKOUT	See (1), (2), and (3)
MCU_PLL1	PLLTS16FFCLAFRACF2	Yes, see Table 5-44 for more information on SSMOD related bitfields/bits	HSDIV0, HSDIV1, HSDIV2, HSDIV3, and HSDIV4	WKUP_HFOSC0_CLKOUT	See (1), (2), and (3)
MCU_PLL2	PLLTS16FFCLAFRACF2	Yes, see Table 5-44 for more information on SSMOD related bitfields/bits	HSDIV0, HSDIV1, HSDIV2, HSDIV3, and HSDIV4	WKUP_HFOSC0_CLKOUT	See (1), (2), and (3)

(1) See *Generic PLL Functional Diagram for PLLTS16FFCLAFRAC2 type* for overview of the PLL.
(2) See Section 5.4.7.5.2 for synthezised clock parameters.
(3) See Section 5.4.7.5.3 for clock output parameters.

Table 5-46 lists the basic information of each of the MAIN PLLs.

Table 5-46. Basic Information of MAIN Domain PLLs

PLL name	Type	SSMOD Available	HSDIV Available	Input clock	Comments
PLL0	PLLTS16FFCLAFRACF2	Yes, see Table 5-44 for more information on SSMOD related bitfields/bits	HSDIV0, HSDIV1, HSDIV2, HSDIV3, HSDIV4, HSDIV5, HSDIV6, HSDIV7, and HSDIV8	MAIN_PLL0_REF_CLK	See (1), (2), and (3)
PLL1	PLLTS16FFCLAFRACF2	Yes, see Table 5-44 for more information on SSMOD related bitfields/bits	HSDIV0, HSDIV1, HSDIV2, HSDIV3, HSDIV5, HSDIV6, HSDIV7, and HSDIV8	MAIN_PLL1_REF_CLK	See (1), (2), and (3)
PLL2	PLLTS16FFCLAFRACF2	Yes, see Table 5-44 for more information on SSMOD related bitfields/bits	HSDIV0, HSDIV1, HSDIV2, HSDIV3, HSDIV4, HSDIV5, HSDIV6, and HSDIV7	MAIN_PLL2_REF_CLK	See (1), (2), and (3)
PLL3	PLLTS16FFCLAFRACF2	Yes, see Table 5-44 for more information on SSMOD related bitfields/bits	HSDIV0, HSDIV1, HSDIV2, HSDIV3, and HSDIV4	MAIN_PLL3_REF_CLK	See (1), (2), and (3)
PLL4	PLLTS16FFCLAFRACF2	Yes, see Table 5-44 for more information on SSMOD related bitfields/bits	HSDIV0, HSDIV1, HSDIV2, and HSDIV3	MAIN_PLL4_REF_CLK	See (1), (2), and (3)
PLL5	PLLTS16FFCLAFRACF2	Yes, see Table 5-44 for more information on SSMOD related bitfields/bits	HSDIV0 and HSDIV1	MAIN_PLL5_REF_CLK	See (1), (2), and (3)
PLL6	PLLTS16FFCLAFRACF2	Yes, see Table 5-44 for more information on SSMOD related bitfields/bits	HSDIV0	MAIN_PLL6_REF_CLK	See (1), (2), and (3)
PLL7	PLLTS16FFCLAFRACF2	Yes, see Table 5-44 for more information on SSMOD related bitfields/bits	HSDIV0	MAIN_PLL7_REF_CLK	See (1), (2), and (3)
PLL8	PLLTS16FFCLAFRACF2	Yes, see Table 5-44 for more information on SSMOD related bitfields/bits	HSDIV0	MAIN_PLL8_REF_CLK	See (1), (2), and (3)
PLL12	PLLTS16FFCLAFRACF2	Yes, see Table 5-44 for more information on SSMOD related bitfields/bits	HSDIV0	MAIN_PLL12_REF_CLK	See (4), (2), and (3)

Copyright © 2024 Texas Instruments Incorporated

Table 5-46. Basic Information of MAIN Domain PLLs (continued)

PLL name	Type	SSMOD Available	HSDIV Available	Input clock	Comments
PLL14	PLLTS16FFCLAFRACF2	Yes, see Table 5-44 for more information on SSMOD related bitfields/bits	HSDIV0 and HSDIV1	MAIN_PLL14_REF_CLK	See (1), (2), and (3)
PLL16	PLLTS16FFCLAFRACF2	Yes, see Table 5-44 for more information on SSMOD related bitfields/bits	HSDIV0 and HSDIV1	MAIN_PLL16_REF_CLK	See (1), (2), and (3)
PLL17	PLLTS16FFCLAFRACF2	Yes, see Table 5-44 for more information on SSMOD related bitfields/bits	HSDIV0 and HSDIV1	MAIN_PLL17_REF_CLK	See (1), (2), and (3)
PLL19	PLLTS16FFCLAFRACF2	Yes, see Table 5-44 for more information on SSMOD related bitfields/bits	HSDIV0	MAIN_PLL19_REF_CLK	See (1), (2), and (3)
PLL25	PLLTS16FFCLAFRAC2	Yes, see Table 5-44 for more information on SSMOD related bitfields/bits	HSDIV0 and HSDIV1	MAIN_PLL25_REF_CLK	See (1), (2), and (3)
PLL26	PLLTS16FFCLAFRACF	No	HSDIV0	MAIN_PLL26_REF_CLK MAIN_PLL26 HSDIV0_CLK_OUT	See (5), (2), and (3)

(1) See *Generic PLL Functional Diagram for PLLTS16FFCLAFRAC2 type* for overview of the PLL.
(2) See Section 5.4.7.5.2 for synthezised clock parameters.
(3) See Section 5.4.7.5.3 for clock output parameters.
(4) See Figure 5-18, *Generic PLL Functional Diagram for PLLTS16FFCLAFRAC2F type* for overview of the PLL.
(5) See *Generic PLL Functional Diagram for PLLTS16FFCLVDESKEWC type* for overview of the PLL.

Copyright © 2024 Texas Instruments Incorporated

5.4.7.5.1 SSMOD Related Bitfields Table

Table 5-44 lists SSMOD related bitfields for all three types of PLLs.

5.4.7.5.2 Clock Synthesis Inputs to the PLLs

Table 5-47 lists the clock synthesis parameters for PLLTS16FFCLAFRAC2 and PLLTS16FFCLAFRACF types.

Table 5-47. Clock synthesis Parameters for PLLTS16FFCLAFRAC2 and PLLTS16FFCLAFRACF Types

Parameter Name	Register
FBDIV	<PLL_name>_FREQ_CTRL0[11-0] FB_DIV_INT (For example, MCU_PLL0 - MCU_PLL0_FREQ_CTRL0[11-0] FB_DIV_INT)
FRAC	<PLL_name>_FREQ_CTRL1[23-0] FB_DIV_FRAC (For example, MCU_PLL0 - MCU_PLL0_FREQ_CTRL1[23-0] FB_DIV_FRAC)
POSTDIV2	<PLL_name>_DIV_CTRL[26-24] POST_DIV2 (For example, MCU_PLL0 - MCU_PLL0_DIV_CTRL[26-24] POST_DIV2)
POSTDIV1	<PLL_name>_DIV_CTRL[18-16] POST_DIV1 (For example, MCU_PLL0 - MCU_PLL0_DIV_CTRL[18-16] POST_DIV1)
REFDIV	<PLL_name>_DIV_CTRL[5-0] REF_DIV (For example, MCU_PLL0 - MCU_PLL0_DIV_CTRL[5-0] REF_DIV)

Table 5-48 lists the clock synthesis parameters for PLLTS16FFCLVDESKEWC type.

Note

For PLLTS16FFCLAFRAC2 and PLLTS16FFCLAFRACF types - POSTDIV1 and POSTDIV2 values are from 1 to 7. To ensure correct operation, POSTDIV1 must always be programmed to a value equal to or greater that POSTDIV2.

Table 5-48. Clock synthesis Parameters for PLLTS16FFCLVDESKEWC Type

Parameter Name	Register
FBDIV	<PLL_name>_DIV_CTRL[13-12] FB_DIV (For example, PLL24 - PLL24_PLL_DIV_CTRL[13-12] FB_DIV)
POSTDIV	<PLL_name>_DIV_CTRL[10-8] POST_DIV (For example, PLL24 - PLL24_PLL_DIV_CTRL[10-8] POST_DIV)
REFDIV	<PLL_name>_DIV_CTRL[1-0] REF_DIV (For example, PLL24 - PLL24_PLL_DIV_CTRL[1-0] REF_DIV)

Note

For PLLTS16FFCLVDESKEWC type:

- REFDIV and FBDIV valid values are 1, 2, and 4.
- POSTDIV valid values are 1, 2, 4, 8, 16, 32, 64, and 128.

5.4.7.5.3 Clock Output Parameter

Table 5-49 lists the control/status bit fields for output clocks.

Copyright © 2024 Texas Instruments Incorporated

Table 5-49. Clock Output Parameter for PLLTS16FFCLAFRAC2 and PLLTS16FFCLAFRACF Types

Clock Output/Divider	Parameter Name	Control/Status Bit Field
<PLL_name>_CLKOUT, <PLL_name>_HSDIVj_CLKOUT(1)	Control	<PLL_name>_CTRL[31] BYPASS_EN (For example, MCU_PLL0 - MCU_PLL0_CTRL[31] BYPASS_EN)
<PLL_name>_HSDIVj_CLKOUT(1)	Control	<PLL_name>_HSDIV_CTRLj15] CLKOUT_EN (For example, MCU_PLL0 - MCU_PLL0_HSDIV_CTRL0[5] CLKOUT_EN and MCU_PLL0_HSDIV_CTRL1[5] CLKOUT_EN)
<PLL_name>_HSDIVj_CLKOUT(1)	Divider control	<PLL_name>_HSDIV_CTRLj[6-0] HSDIV (For example, MCU_PLL0 - MCU_PLL0_HSDIV_CTRL0[6-0] HSDIV and MCU_PLL0_HSDIV_CTRL1[6-0] HSDIV)
PLL bypass mux	Control	<PLL_name>_CTRL[16] BYP_ON_LOCKLOSS (For example, MCU_PLL0 - MCU_PLL0_CTRL[16] BYP_ON_LOCKLOSS)
Enable 4-phase clock generator	Control	<PLL_name>_CTRL[5] CLK_4PH_EN (ignored if <PLL_name>_CTRL[4] CLK_POSTDIV_EN = 0) (For example, MCU_PLL0 - MCU_PLL0_CTRL[5] CLK_4PH_EN)
Post divide CLK enable	Control	<PLL_name>_CTRL[4] CLK_POSTDIV_EN (For example, MCU_PLL0 - MCU_PLL0_CTRL[4] CLK_POSTDIV_EN)
Delta-Sigma modulator enable	Control	<PLL_name>_CTRL[1] DSM_EN (For example, MCU_PLL0 - MCU_PLL0_CTRL[1] DSM_EN)
Enable fractional noise canceling DAC	Control	<PLL_name>_CTRL[0] DAC_EN (For example, MCU_PLL0 - MCU_PLL0_CTRL[0] DAC_EN)

(1) j is the number of HSDIV for the current PLL.

Table 5-50. Clock Output Parameter for PLLTS16FFCLVDESKEWC Type

Clock Output/Divider	Parameter Name	Control/Status Bit Field
<PLL_name>_CLK_OUT, <PLL_name>_HSDIVj_CLK_OUT	Control	<PLL_name>_CTRL[31] BYPASS_EN (For example, PLL24 - PLL24_CTRL[31] BYPASS_EN)

5.4.7.5.4 Calibration Related Bitfields

Table 5-51 lists the calibration related bitfields/bits.

Table 5-51. Recalibration Feature Parameters

Recalibration feture	Parameter Name	Control/Status Bit Field
Calibration enable to actively adjust for input skew	Control	<PLL_name>_CAL_CTRL[31] CAL_EN (For example, PLL12 - PLL12_CAL_CTRL[31] CAL_EN)
Fast calibration enabled	Control	<PLL_name>_CAL_CTRL[20] FAST_CAL (For example, PLL12 - PLL12_CAL_CTRL[20] FAST_CAL)
Calibration loop programmable counter	Control	<PLL_name>_CAL_CTRL[18-16] CAL_CNT (For example, PLL12 - PLL12_CAL_CTRL[18-16] CAL_CNT)
Calibration bypass	Control	<PLL_name>_CAL_CTRL[15] CAL_BYP (For example, PLL12 - PLL12_CAL_CTRL[15] CAL_BYP)
Calibration input	Control	<PLL_name>_CAL_CTRL[11-0] CAL_IN (For example, PLL12 - PLL12_CAL_CTRL[11-0] CAL_IN)
Output of the calibration block	Status	<PLL_name>_CAL_STAT[11-0] CAL_OUT (For example, PLL12 - PLL12_CAL_STAT[11-0] CAL_OUT)

Copyright © 2024 Texas Instruments Incorporated

5.4.7.6 PLL and PLL Controller Connection

Two PLL Controllers are implemented in the device - PLLCTRL0 and WKUP_PLL_CTRL0. They are respectively related to PLL0 and MCU_PLL0. The PLL Controllers manage the clock ratios, alignment, and gating for the system clocks.

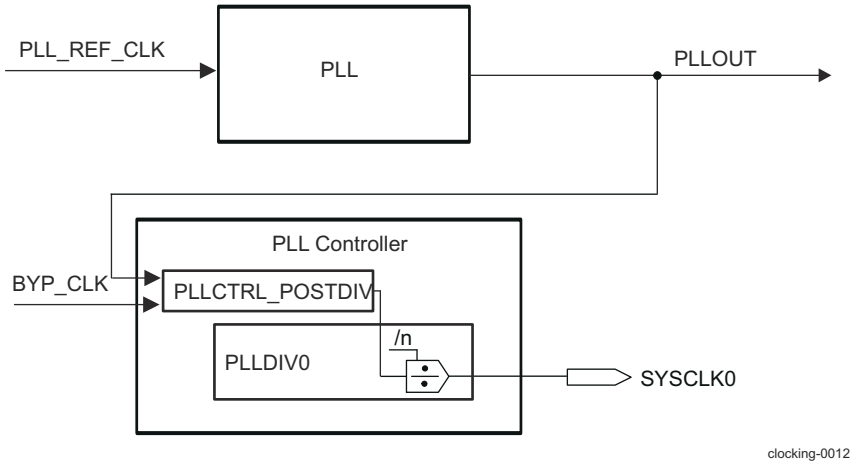


Figure 5-22. PLL and PLL Controller

Note

PLLCTRL_POSTDIV is not supported in this family of devices.

Note

The PLL Controllers registers can be accessed by any master in the device.

A SYSCLK0 clock out of a PLLCTRL is the only synchronous clock in that domain. That means - MCU_SYSCLK0 out of WKUP_PLLCTRL is the synchronous CBASS clock in MCU domain and SYSCLK0 out of Main PLLCTRL is the synchronous CBASS clock in Main domain.

5.4.7.7 System Clocks Operating Frequency Ranges

Table 5-52 lists the operating frequency ranges for the system clocks of the device.

Table 5-52. System Clocks Operating Frequency Range

System Clocks	Minimum Operating Frequency (MHz)
MCU_PLL0 (MCU TBD PLL) with WKUP_PLLCTRL0	1500 MHz to 3200 MHz
MCU_PLL1 (MCU PERIPHERAL PLL)	1500 MHz to 3200 MHz
MCU_PLL2 (MCU CPSW PLL)	1500 MHz to 3200 MHz
PLL0 (MAIN PLL) with PLLCTRL0	1500 MHz to 3200 MHz
PLL1 (PER0 PLL)	1500 MHz to 3200 MHz
PLL2 (PER1 PLL)	1500 MHz to 3200 MHz
PLL3 (CPSW9G PLL)	1500 MHz to 3200 MHz
PLL4 (AUDIO0 PLL)	1500 MHz to 3200 MHz
PLL5 (VIDEO PLL)	1500 MHz to 3200 MHz
PLL6 (GPU PLL)	1500 MHz to 3200 MHz
PLL7 (C7x PLL)	1500 MHz to 3200 MHz
PLL8 (ARM0 PLL)	1500 MHz to 3200 MHz
PLL12 (DDR PLL)	1500 MHz to 3200 MHz
PLL14 (PULSAR PLL)	1500 MHz to 3200 MHz
PLL16 (DSS PLL0)	1500 MHz to 3200 MHz

Copyright © 2024 Texas Instruments Incorporated

Table 5-52. System Clocks Operating Frequency Range (continued)

System Clocks	Minimum Operating Frequency (MHz)
PLL17 (DSS PLL1)	1500 MHz to 3200 MHz
PLL19 (DSS PLL3)	1500 MHz to 3200 MHz
PLL25 (VISION PLL)	1500 MHz to 3200 MHz
PLL26 (DDR1)	1500 MHz to 3200 MHz

(1) Supported input reference clock frequencies to the PLL are 19.2/24/25/26 MHz only.

(2) Interconnect clock on DSS is CPU/4. This will range from 100 MHz to 250 MHz.

(3) When Main PLL is configured to 400 MHz mode, DSS can only support a max pixel clock of 74.25 MHz. For lower resolution displays the DSS clock can be lower than 74.25 MHz.

5.4.7.8 Recommended Clock and Control Signal Transition Behavior

All clocks and strobe signals must transition between V_{IH} and V_{IL} (or between V_{IL} and V_{IH}) in a monotonic manner. Monotonic transitions are more easily ensured with faster switching signals. Slower input transitions are more susceptible to glitches due to noise, and special care must be taken for slow input clocks.

5.4.7.9 Interface Clock Specifications

Interface Clock Terminology

The interface clock is used at the system level to sequence the data and to control transfers accordingly with the interface protocol.

Interface Clock Frequency

The two interface clock characteristics are:

- The maximum clock frequency
- The maximum operating frequency

The interface clock frequency documented here is the maximum clock frequency, which corresponds to the maximum frequency programmable on this output clock. This frequency defines the maximum limit supported by the Device IC and does not take into account any system consideration (PCB, peripherals).

The system designer must take into account these system considerations and the Device IC timing characteristics to properly define the maximum operating frequency that corresponds to the maximum frequency supported to transfer the data on this interface.

5.4.7.10 PLL, PLLCTRL, and HSDIV Controllers Programming Guide

5.4.7.10.1 PLL Initialization

5.4.7.10.1.1 Kick Protection Mechanism

Note

PLL configuration registers are write-protected at power-up. Software must first un-lock the PLLn_LOCKKEY0 and PLLn_LOCKKEY1 registers prior to writing to any chip-level registers.

The un-lock process shall follow the steps:

1. Write value 0x68EF3490 to PLLn_LOCKKEY0 register.
2. Write value 0xD172BC5A to PLLn_LOCKKEY1 register.

After these two steps a write access to the PLL registers is allowed. Writing any other data value to either of these two registers locks the kicker mechanism and blocks any writes to the PLL registers.

Copyright © 2024 Texas Instruments Incorporated

Note

In order to ensure that all PLL registers are write protected, software must always re-lock the kicker mechanism after completing the register writes.

5.4.7.10.1.2 PLL Initialization to PLL Mode

Note

For validated set of parameters that can be programmed to PLLs, please refer to the device-specific Datasheet.

Initially, the device powers up in PLL bypass mode. The bypass mux is a glitch free mux and is located outside the PLL module. PLL_EXTBYPASS bit controls the bypass mux section. During Power-up, the bypass mux defaults to select FREF clock (PLL clock bypass mode).

During boot, WKUP_DMSC0 ROM programs the MCU_PLL0 to a valid frequency based on the crystal frequency and BOOTMODE pin settings. WKUP_DMSC0 then waits for PLL to be locked by checking the PLL LOCK status bit. WKUP_DMSC0 ROM programs PLL_EXTBYPASS bit to ‘0’ to disable the bypass mode to propagate MCU_PLL0 clock to WKUP_PLLCTRL0. At this point WKUP_DMSC0 brings MCU_R5FSS out of reset to complete the reset of the boot process. R5FSS software configures the remaining PLLs and bring them out of bypass mode.

5.4.7.10.1.3 PLL Programming Requirements

- A PLL VCO frequency must be set to > 1500 MHz
  - For best PLL performance, maximize VCO frequency where possible.
- A PLL must always operate in Fractional Mode (DACEN and DSMEN set to ‘1’). This is true even when the FBDIV is a integer.
- A PLL Reference clock frequency must be > 10 MHz.
- REFDIV[5:0] must be set to “000001”.

5.4.7.10.1.3.1 PLL Calibration Procedure

PLL calibration is intended to reduce a specific form of PLL jitter in which a large correction is present on each reference clock edge. In this case, the calibration logic smooths the correction so that the change to the PLL operation is not as abrupt and therefore it reduces jitter related to the reference clock.

In <PLL_name>_CAL_CTRL register:

- CAL_IN = 0
- CAL_CNT = 2
- CAL_BYP = 0
- FAST_CAL = 1

Then, set CAL_EN = 1

Wait for <PLL_name>_CAL_STAT[31] CAL_LOCK to be asserted. The CAL_LOCK signal requires a timeout; if the signal is NOT asserted within 2.3ms, the user may continue to use the PLL but should not change CAL_CNT or FAST_CAL (in the next step).

Then, the use may (it is not required):

- change CAL_CNT from 2 to 7
- change FAST_CAL from 1 to 0

These optional changes increase the duration for which the calibration solution is measured from 2^{2+5} reference clock periods to 2^{7+5} reference clock periods.

Calibration cannot be used when the PLL is configured for fractional mode; specifically, this restriction means that PLL_CTRL[1] = 0.

Copyright © 2024 Texas Instruments Incorporated

5.4.7.10.2 Entire Sequence for Programming PLLCTRL, HSDIV, and PLL

Table 5-53 shows the entire sewuence of programming PLLCTRL, HSDIV, and PLL from an unknown state to a defined non-spread-spectrum state.

Table 5-53. Programming Sequence of PLLCTRL, HSDIV, and PLL

Step	Description
Unlock PLL registers	<PLL_Name>_LOCKKEY0 (= 0x68EF3490)
	<PLL_Name>_LOCKKEY1 (= 0xD172BC5A)
If PLL0	Configure PLLCTRL block into bypass mode
	PLLCTL[0]PLLEN = 0
	PLLCTL[5] PLLENSRC = 0
Configure external bypass so that no transient clock propagates	<PLL_Name>_CTRL[31] BYPASS_EN = 1
Delay	
Disable HSDIV(s)	<PLL_Name>_HSDIV_CTRLk[15] CLKOUT_EN =0
Delay	
Disable PLL	<PLL_Name>_CTRL[15] PLL_EN =0
Delay	
Reset HSDIV(s)	<PLL_Name>_HSDIV_CTRLk[31] RESET = 1
If PLL0	Check PLLSTAT[0] GOSTAT to make sure that divider change is not currently in-progress GOSTAT should equal 0.
	Clear GOSET bit PLLCMD[0] GOSET = 0
	Configure divider in PLLCTRL to /1 PLLDIV1 = 0x8000 (enable divider and divide by 1) PLLDIV2 = 0x00 (disable divider and divide by 1)
	Set alignment control bits ALNCTL = 3
	Set GOSET bit to initiate the divider change PLLCMD[0] GOSET = 1
	Check PLLSTAT[0] GOSTAT to make sure that divider change has completed GOSTAT should equal 0.
Configure HSDIV(s) divider value	<PLL_Name>_HSDIV_CTRLk[6-0] HSDIV (=val)
Clear HSDIV(s) SYNC_DIS	<PLL_Name>_HSDIV_CTRLk[8] SYNC_DIS = 0
Delay	
Clear Reset HSDIV(s)	<PLL_Name>_HSDIV_CTRLk[31] RESET = 0
Configure PLL multiplier	Integer portion of divider <PLL_Name>_FREQ_CTRL0[11-0] (=val) Fractional portion of divider <PLL_Name>_FREQ_CTRL1[23-0] (=val)
Configure PLL dividers	Reference clock divider <PLL_Name>_DIV_CTRL[5-0] REF_DIV = 1
	Post divider 1 <PLL_Name>_DIV_CTRL[18-16] POST_DIV1 = 1
	Post divider 2 <PLL_Name>_DIV_CTRL[26-24] POST_DIV2 = 1
Configure “random” PLL controls	<PLL_Name>_CTRL[8] INTL_BYP_EN = 0
	<PLL_Name>_CTRL[5] CLK_4PH_EN = 0
	<PLL_Name>_DIV_CTRL[26-24] POST_DIV2 = 1
	<PLL_Name>_DIV_CTRL[18-16] POST_DIV1 = 1
	<PLL_Name>_DIV_CTRL[5-0] REF_DIV = 1
	<PLL_Name>_CTRL[1] DSM_EN = 1
	<PLL_Name>_CTRL[0] DAC_EN = 1

Copyright © 2024 Texas Instruments Incorporated

Table 5-53. Programming Sequence of PLLCTRL, HSDIV, and PLL (continued)

Step	Description
	<PLL_Name>_SS_CTRL[31] BYPASS_EN (SSMOD) = 1
	<PLL_Name>_CTRL[4] CLK_POSTDIV_EN = 1
	<PLL_Name>_CTRL[16] BYP_ON_LOCKLOSS (= {0, 1})
Delay	1 us
Enable PLL	<PLL_Name>_CTRL[15] PLL_EN = 1
Wait for Lock	Wait for MCU_PLL0_STAT[0] LOCK = 1
Enable HSDIV(s)	<PLL_Name>_HSDIV_CTRLk[15] CLKOUT_EN = 1
Delay	
Configure external bypass to pass PLL	<PLL_Name>_CTRL[31] BYPASS_EN = 0
Delay	
If PLL0	Configure PLLCTRL block into PLL mode PLLCTL[0] PLLEN = 0
Lock PLL registers	<PLL_Name>_LOCKKEY0 (= any value)
	<PLL_Name>_LOCKKEY1 (= any value)

5.5 Module Integration

5.5.1 ADC

Table 5-54. ADC Instances

Instances	WKUP Domain	MCU Domain	MAIN Domain
MCU_ADC[1:0]		√	

5.5.1.1 ADC Unsupported Features

The integration of the ADC module in this Device supports all module features.

5.5.1.2 ADC Integration Details

ADC Hardware Event Selection

Each ADC module has a HW EVENT input that provides the ability to trigger ADC sampling based on a hardware event rather than through software. Muxes are provided to select the hardware trigger from external pins, periodic timer events, or eHRPWM start of conversion events.

This event selection is described in controlled by the MCU_ADCn_CTRL_TRIG_SEL (n=0,1) field of the MCU_CTRL_MMR0_MCU_ADCn_CTRL (n=0,1) registers in the Control Module (CTRL_MMR), with the default selection being each ADC's associated EXT_TRIGGER pin.

ADC GPI Integration

Each ADC module can be configured either as 8 analog inputs, or 8 digital inputs. To enable digital input mode, the MCU_ADCn_CTRL_GPI_MODE_EN (n=0,1) bit of the MCU_CTRL_MMR0_MCU_ADCn_CTRL (n=0,1) registers in the Control Module (CTRL_MMR) can be set to '1'. This enables the ADC pins to be used as general purpose digital inputs with some restrictions:

- ADC pins support only 1.8V logic levels
- These may be used as GPI only not as GPIO (no output buffer)
- Speed is limited to a 200 MHz switching rate
- All ADC inputs will operate as either digital or analog inputs as a group (no individual selection)
- The GPI_MODE_EN selection cannot be switched dynamically (desired operation should be selected before enabling the ADC)

ADC Reference Integration

Copyright © 2024 Texas Instruments Incorporated

The ADC0/1 REFN and REFP voltage references are not pinned out on this device. See the below table for details on the internal connections for each reference.

ADC Reference	Internal Device Connection[(1)]
ADC0 REFP	VDDA_ADC0
ADC0 REFN	VSS
ADC1 REFP	VDDA_ADC1
ADC1 REFN	VSS

(1) TI recommends minimizing the shared impedance of the ADC References and VDDA_ADC[1:0] / VSS.

5.5.2 ATL

Instance	WKUP Domain	MCU Domain	MAIN Domain
ATL0			√

5.5.2.1 ATL Unsupported Features

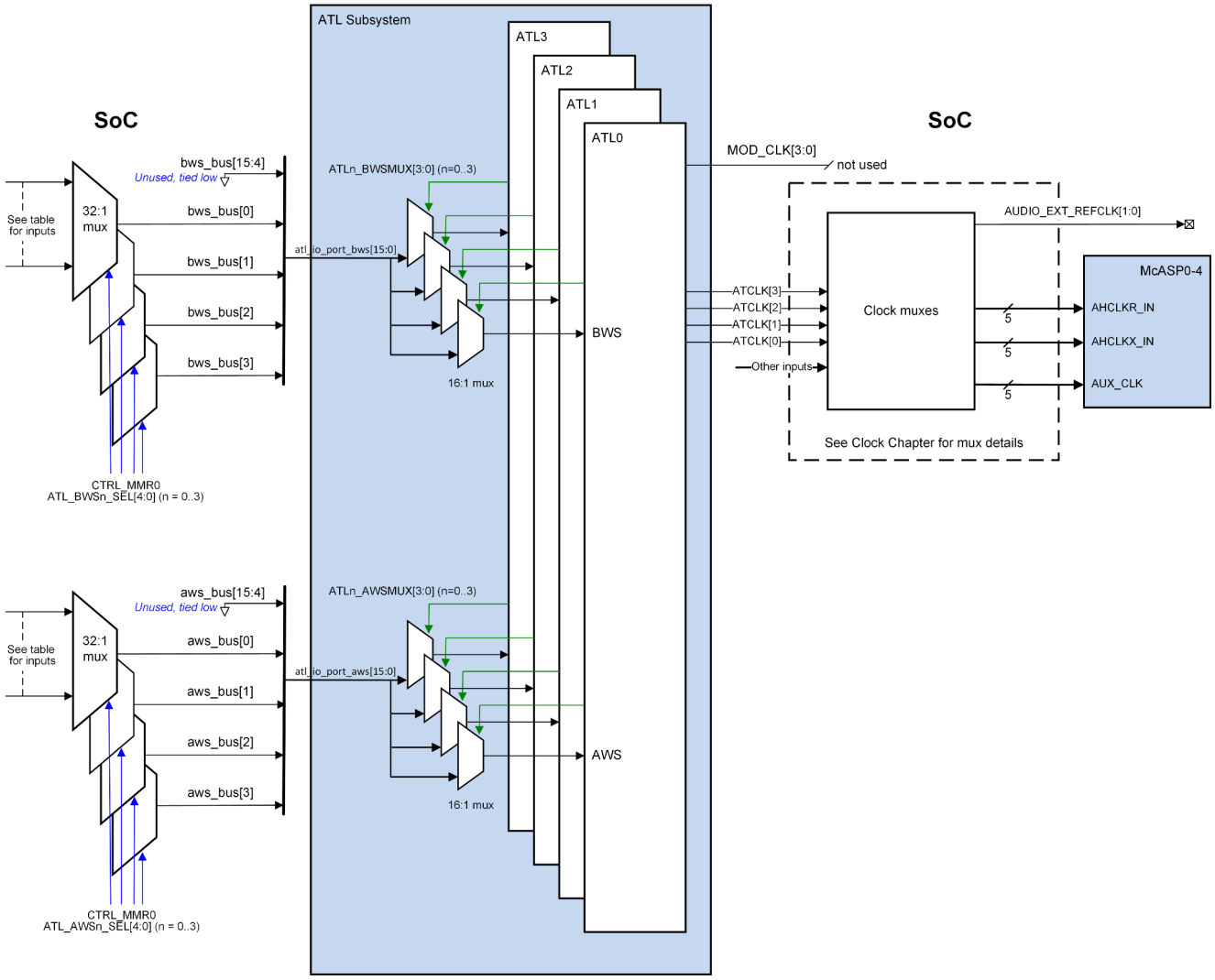
The ATL module features listed below are not supported by the integration on this device:

- MOD_OUT[3:0] Modem Clock (MODCLK) outputs are unused and unconnected

5.5.2.2 ATL Integration Details

The adjusted audio clock (ATCLK) outputs of the ATLSS may be selected to drive the AUDIO_EXT_REFCLK[1:0] pins to serve as the master clock of an audio dac, fed back to McASP AHCLKX/R inputs to drive AFSX/R and ACLKX/R dividers, or selected as McASP AUXX_CLK sources as illustrated below. See the clocking section for additional details.

Copyright © 2024 Texas Instruments Incorporated



The ATL subsystem includes internal muxes to choose 1 of 16 possible sources of AWS and BWS inputs for each of the four ATL instances. These internal muxes are controlled by the ATL module's ATLn_AWSMUX and ATLn_BWSMUX registers and are only used to select one AWS and one BWS source from atl_io_port_aws and atl_io_port_bws for each of ATL instance. Device level muxes are used to choose a word select source from up to 32 possible inputs for each.

The following table shows Baseband I2S word select (BWS) input sources for each of the 4 ATL modules in the ATL subsystem (*n* = 0-3):

CTRL_MMR0 ATL_BWS*n*_SEL[4:0]	Source Signal Name	Mode	CTRL_MMR0 ATL_BWS*n*_SEL[4:0]	Source Signal Name	Mode
0	mcasp0_afsr_pad_in	McASP0 ASYNC	16	mcasp4_afsx_pad_in1 2	McASP4 SYNC
1	mcasp1_afsr_pad_in	McASP1 ASYNC	17	tied 1'b0	Reserved
2	mcasp2_afsr_pad_in	McASP2 ASYNC	18	tied 1'b0	Reserved
3	mcasp3_afsr_pad_in	McASP3 ASYNC	19	tied 1'b0	Reserved
4	mcasp4_afsr_pad_in(1)	McASP4 ASYNC	20	tied 1'b0	Reserved
5	tied 1'b0	Reserved	21	tied 1'b0	Reserved
6	tied 1'b0	Reserved	22	tied 1'b0	Reserved
7	tied 1'b0	Reserved	23	tied 1'b0	Reserved
8	tied 1'b0	Reserved	24	AUDIO_EXT_REFCLK0_IN	External Clock Source 0

Copyright © 2024 Texas Instruments Incorporated

CTRL_MMR0 ATL_BWS*n*_SEL[4:0]	Source Signal Name	Mode	CTRL_MMR0 ATL_BWS*n*_SEL[4:0]	Source Signal Name	Mode
9	tied 1'b0	Reserved	25	AUDIO_EXT_REFCLK1_IN	External Clock Source 1
10	tied 1'b0	Reserved	26	tied 1'b0	Reserved
11	tied 1'b0	Reserved	27	tied 1'b0	Reserved
12	mcasp0_afsx_pad_in(2)	McASP0 SYNC	28	tied 1'b0	Reserved
13	mcasp1_afsx_pad_in(2)	McASP1 SYNC	29	tied 1'b0	Reserved
14	mcasp2_afsx_pad_in(2)	McASP2 SYNC	30	tied 1'b0	Reserved
15	mcasp3_afsx_pad_in(2)	McASP3 SYNC	31	tied 1'b0	Reserved

1. Usage Note: McASP4 is intended for use with eDP but may be used as an additional general purpose transmit/receive audio stream if the clock pins are available in the pinmux.
2. FAQ: In SYNC mode, McASP uses the AFSX signal to clock both receiver and transmitter, which is why AFSX shows up in the BWS table.

The following table shows Audio I2S word select (AWS) input sources for each of the 4 ATL modules in the ATL subsystem (*n* = 0-3):

CTRL_MMR0 ATL_AWS*n*_SEL[4:0]	Source Signal Name	Mode	CTRL_MMR0 ATL_AWS*n*_SEL[4:0]	Source Signal Name	Mode
0	mcasp0_afsx_pad_in	McASP0 ASYNC	16	mcasp4_afsx_pad_in1	McASP4 SYNC
1	mcasp1_afsx_pad_in	McASP1 ASYNC	17	tied 1'b0	Reserved
2	mcasp2_afsx_pad_in	McASP2 ASYNC	18	tied 1'b0	Reserved
3	mcasp3_afsx_pad_in	McASP3 ASYNC	19	tied 1'b0	Reserved
4	mcasp4_afsx_pad_in(1)	McASP4 ASYNC	20	tied 1'b0	Reserved
5	tied 1'b0	Reserved	21	tied 1'b0	Reserved
6	tied 1'b0	Reserved	22	tied 1'b0	Reserved
7	tied 1'b0	Reserved	23	tied 1'b0	Reserved
8	tied 1'b0	Reserved	24	AUDIO_EXT_REFCLK0_IN	External Clock Source 0
9	tied 1'b0	Reserved	25	AUDIO_EXT_REFCLK1_IN	External Clock Source 1
10	tied 1'b0	Reserved	26	tied 1'b0	Reserved
11	tied 1'b0	Reserved	27	tied 1'b0	Reserved
12	mcasp0_afsx_pad_in	McASP0 SYNC	28	tied 1'b0	Reserved
13	mcasp1_afsx_pad_in	McASP1 SYNC	29	tied 1'b0	Reserved
14	mcasp2_afsx_pad_in	McASP2 SYNC	30	tied 1'b0	Reserved
15	mcasp3_afsx_pad_in	McASP3 SYNC	31	tied 1'b0	Reserved

1. Usage Note: McASP4 is intended for use with eDP but may be used as an additional general purpose transmit/receive audio stream if the clock pins are available in the pinmux.

5.5.3 CPSW2G

Table 5-55. CPSW2G Instances

Instance	WKUP Domain	MCU Domain	MAIN Domain
CPSW2G0			√
MCU_CPSW2G0		√	

Copyright © 2024 Texas Instruments Incorporated

5.5.3.1 CPSW2G Unsupported Features

The CPSW2G module features listed below are not supported by the integration on this device:

- Maximum frame size of 9600 bytes
- GMII Mode
- SGMII Mode
- MACSEC
- Synchronous Ethernet

5.5.3.2 MCU_CPSW2G0 Integration Details

RMII Clocking

The Device integrates both internal and external clock source options in RMII mode. However, the internal clock source option is UNSUPPORTED by the Device integration. Figure 5-23 presents the high level view of the two clocking options.

Figure 5-23(a) shows the UNSUPPORTED internal clock source for RMII clock. It is a 50 MHz clock source that is provided by the Device on the MCU_CLKOUT0 pin. This clock source routes on the PCB to both the Device MCU_RMII1_REF_CLK input and the external Phy RMII clock input.

Figure 5-23(b) shows the SUPPORTED external clock source option for the RMII clock. In this case a 50 MHz clock is available on the PCB (possibly from an oscillator or from the Ethernet Phy). This externally generated clock source is routed to both the Device MCU_RMII1_REF_CLK input and the external Phy RMII clock input. In both cases, the Device MCU_RMII1_REF_CLK is an input. The difference between the two cases is simply that in the UNSUPPORTED Figure 5-23(a) the Device provides a 50 MHz clock source on a separate pin.

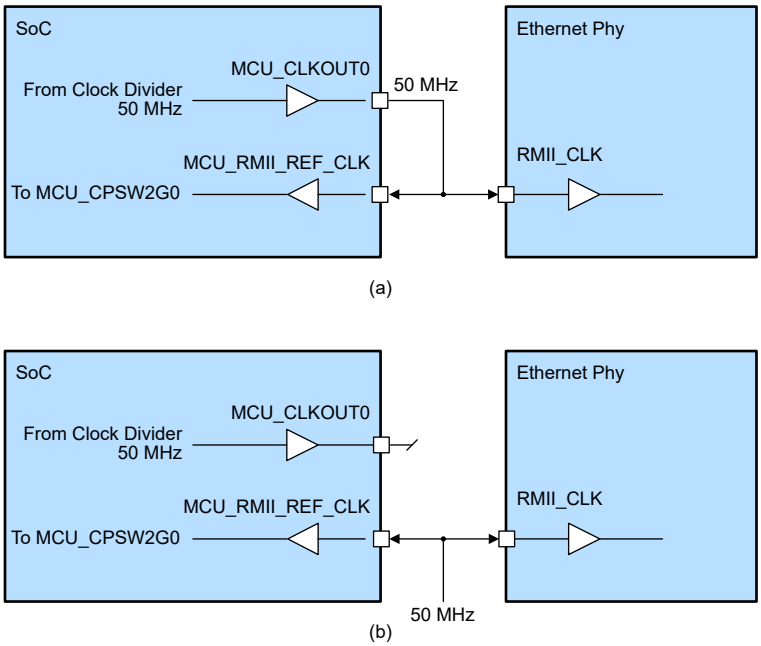


Figure 5-23. Clocking Options in RMII Mode

5.5.3.3 CPSW2G0 Integration Details

RMII Clocking

The Device integrates both internal and external clock source options in RMII mode. However, the internal clock source option is UNSUPPORTED by the Device integration. Figure 5-24 presents the high level view of the two clocking options.

Copyright © 2024 Texas Instruments Incorporated

Figure 5-24(a) shows the UNSUPPORTED internal clock source for RMII clock. It is a 50 MHz clock source that is provided by the Device on the CLKOUT pin. This clock source routes on the PCB to both the Device RMII_REF_CLK input and the external Phy RMII clock input.

Figure 5-24(b) shows the SUPPORTED external clock source option for the RMII clock. In this case a 50 MHz clock is available on the PCB (possibly from an oscillator or from the Ethernet Phy). This externally generated clock source is routed to both the Device RMII_REF_CLK input and the external Phy RMII clock input. In both cases, the Device RMII_REF_CLK is an input. The difference between the two cases is simply that in the UNSUPPORTED Figure 5-24(a) the Device provides a 50 MHz clock source on a separate pin.

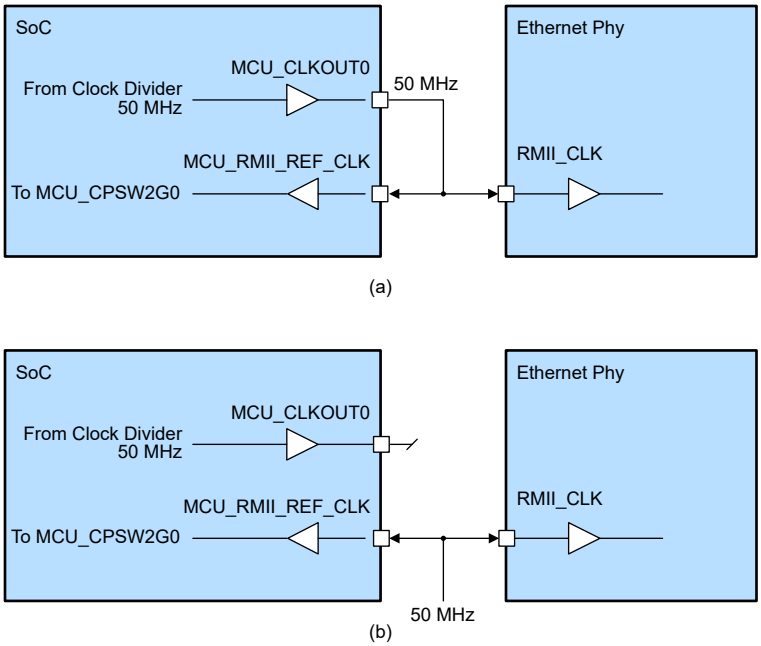


Figure 5-24. Clocking Options for RMII Mode

5.5.4 CSI_RX

Table 5-56. CSI_RX Instances

Instances	WKUP Domain	MCU Domain	MAIN Domain
CSI_RX[1:0]			√

5.5.4.1 CSI_RX Unsupported Features

The CSI_RX module features listed below are not supported by the integration on this device:

- Line Count Error Interrupt (Streams 1-3)
- Frame Mismatch Error Interrupt (Streams 1-3)
- Frame Count Error Interrupt (Streams 1-3)
- FCC Stop Interrupt (Streams 1-3)
- FCC Start Interrupt (Streams 1-3)
- Line/Byte Interrupt (Streams 1-3)
- Timer / Timer Interrupt (Streams 1-3)

5.5.4.2 CSI_RX Integration Details

Each CSI_RX is driven by a dedicated instance of a MIPI D-PHY Receive module. The D-PHY is contains its own bus interface for configuration rather than being configured through the CSI_RX module to which it is connected. All instances of CSI_RX have dedicated DMA interfaces (PSI-L) and therefore can stream to memory. The DMA interface supports up to 32 channels. Likewise, all instances of the CSI_RX module can be configured to retransmit to the CSI_TX module for diagnostic purposes. The streaming interface between each CSI_RX and CSI_TX supports up to 4 channels.

Copyright © 2024 Texas Instruments Incorporated

The Device supports flexible mapping of VP0/VP1 to any CAL0_i streaming input of the VPAC instances. Any one of the CSI_RX VP0 ports can be selected to feed the CAL0_i streaming input of the VPAC0 directly.

5.5.5 CSI_TX

Table 5-57. CSI_TX Instances

Instances	WKUP Domain	MCU Domain	MAIN Domain
CPSW9G0			√

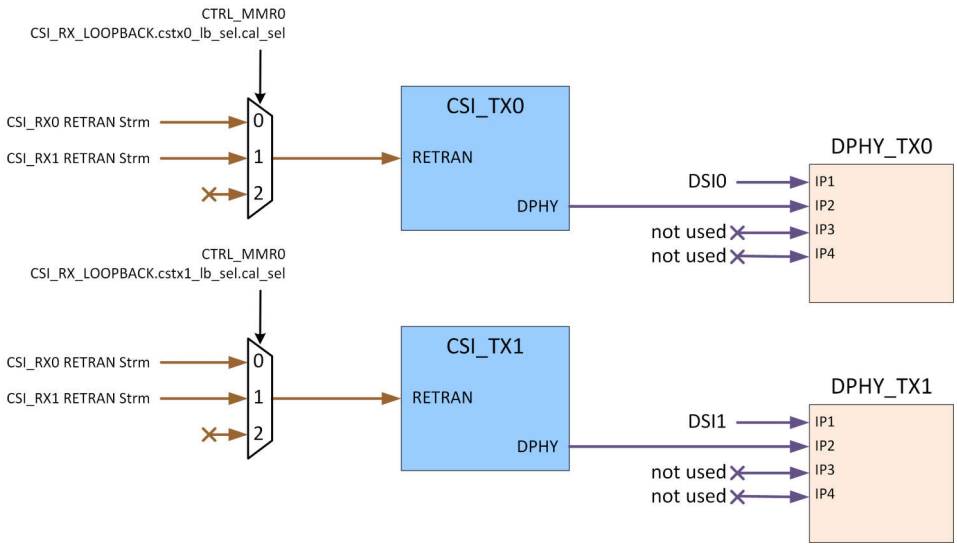
5.5.5.1 CSI_TX Unsupported Features

The CSI_TX module features listed below are not supported by the integration on this device:

- CSI2 ver2.0 scrambling
- CSI2 ver2.0 optional no LP (low power state) between packets
- Cropping
- Pixel Processing
- Multiple Active Input Streams
- 3 Lane Support

5.5.5.2 CSI_TX Integration Details

The CSI_TX module is able to transmit CSI-2 protocol image data from memory or retransmit the video stream of either CSI_RX module. The CTRL_MMR0 CSI_RX_LOOPBACK register is used to select the mapping of CSI_RX receiver stream to each CSI_TX transmitter. Each CSI_TX shares a MIPI transmit PHY (DPHY_TX) with a DSI Video output.



5.5.6 DCC

Table 5-58. DCC Instances

Instances	WKUP Domain	MCU Domain	MAIN Domain
MCU_DCC[1:0]		√	
DCC[9:0]			√

5.5.6.1 DCC Unsupported Features

The integration of the DCC module in this Device supports all module features.

5.5.6.2 DCC Integration Details

There are no additional DCC module integration details.

Copyright © 2024 Texas Instruments Incorporated

5.5.7 DMTIMER (Timer)

Table 5-59. DMTIMER Instances

Instances	WKUP Domain	MCU Domain	MAIN Domain
MCU_DMTIMER[9:0]		√	
DMTIMER[19:0]			√

5.5.7.1 DMTIMER (Timer) Unsupported Features

The DMTIMER (Timer) module features listed below are not supported by the integration on this device:

- Atomic 64-bit timer value read of cascaded timers

5.5.7.2 DMTIMER (Timer) Integration Details

Timer Resolution

The Timer resolution and interrupt period are dependent on the selected input clock and clock prescale value. Example resolutions for common clock values are shown in Table 5-60.

Table 5-60. Timer Resolution and Maximum Range

Clock	Prescaler	Resolution	Interrupt Period Range
32.768 KHz	1 (min)	31.25 µs	31.25 µs to ~36 hr, 35 mins
	256 (max)	8 ms	8 ms to ~391 days, 22 hr, 48 mins
27 MHz	1 (min)	~42 ms	~42 ms to ~179 sec
	256 (max)	~10.67 µs	~10.67 µs to ~12 hr, 44 mins

Timer Cascading

Each odd numbered Timer instance within the MCU Domain may be optionally cascaded with the previous even numbered Timer instance to form an up to 64-bit timer.

- MCU_DMTIMER1 may be cascaded to MCU_DMTIMER0
- MCU_DMTIMER3 may be cascaded to MCU_DMTIMER2
- ...
- MCU_DMTIMER9 may be cascaded to MCU_DMTIMER8

Each odd numbered Timer instance within the MAIN Domain may be optionally cascaded with the previous even numbered Timer instance to form an up to 64-bit timer.

- DMTIMER1 may be cascaded to DMTIMER0

  DMTIMER3 may be cascaded to DMTIMER2

  ...

  DMTIMER19 may be cascaded to DMTIMER18

When cascaded, MCU_DMTIMERn acts as a 32-bit prescaler to MCU_DMTIMER[n+1]. The MCU_DMTIMERn must be configured to generate a PWM output edge at the desired rate to increment the MCU DMTIMER[n+1] counter. The same cascading rules apply for MAIN Domain DMTIMERs.

Timer IO Multiplexing

The Device provides 10 MCU_TIMERIO pins and 8 TIMERIO pins to be used as Timer Capture inputs or as Timer PWM outputs. In order to provide maximum flexibility, the 10 MCU_TIMERIO pins may be used with any of the 10 MCU_DMTIMER[9:0] instances, and the 8 TIMERIO pins may be used with any of the 20 DMTIMER[19:0] instances. System-level muxes configured in the CTRL_MMRs are used to control the capture source pin and PWM ouptut source for each MCU_DMTIMER and DMTIMER instance.

5.5.8 DPHY_RX

Copyright © 2024 Texas Instruments Incorporated

Table 5-61. DPHY_RX Instances

Instances	WKUP Domain	MCU Domain	MAIN Domain
DPHY_RX[1:0]			√

5.5.8.1 DPHY_RX Unsupported Features

The DPHY_RX module features listed below are not supported by the integration on this device:

- Swapping of Clock & Data Lanes

5.5.8.2 DPHY_RX Integration Details

The DPHY_RX is a point-to-point connection to the CSI_RX interface.

5.5.9 DPHY_TX

Table 5-62. DPHY_TX Instances

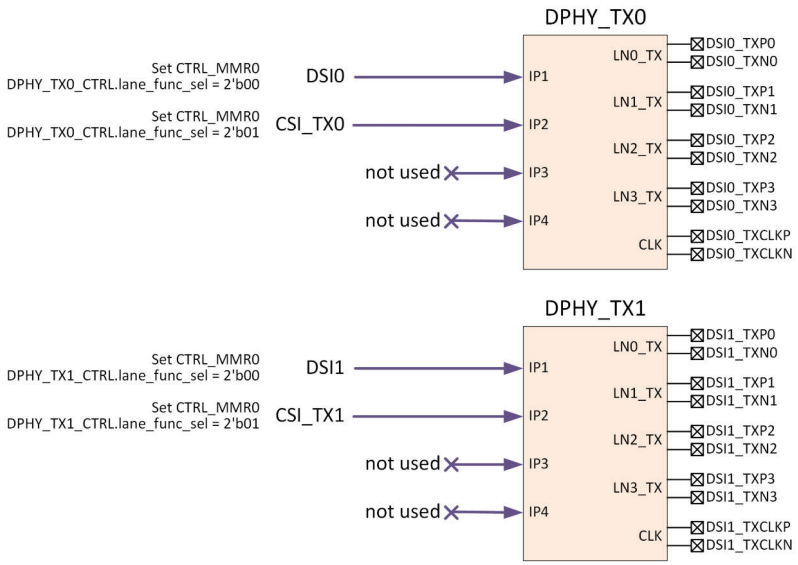
Instances	WKUP Domain	MCU Domain	MAIN Domain
DPHY_TX[1:0]			√

5.5.9.1 DPHY_TX Unsupported Features

The integration of the DPHY_TX module in this Device supports all module features.

5.5.9.2 DPHY_TX Integration Details

Each DPHY_TX provides the physical interface for either a MIPI DSI display or a MIPI CSI_TX video output. The output source is selected using the appropriate DPHY_TX*n*_CTRL register in CTRL_MMR0.



5.5.10 DSS/DSI

Instance	WKUP Domain	MCU Domain	MAIN Domain
DSS			√
DSI[1:0]			√

5.5.10.1 DSS Unsupported Features

None.

5.5.10.2 DSI Unsupported Features

- DSI1 does not support 30-bit or 36-bit data

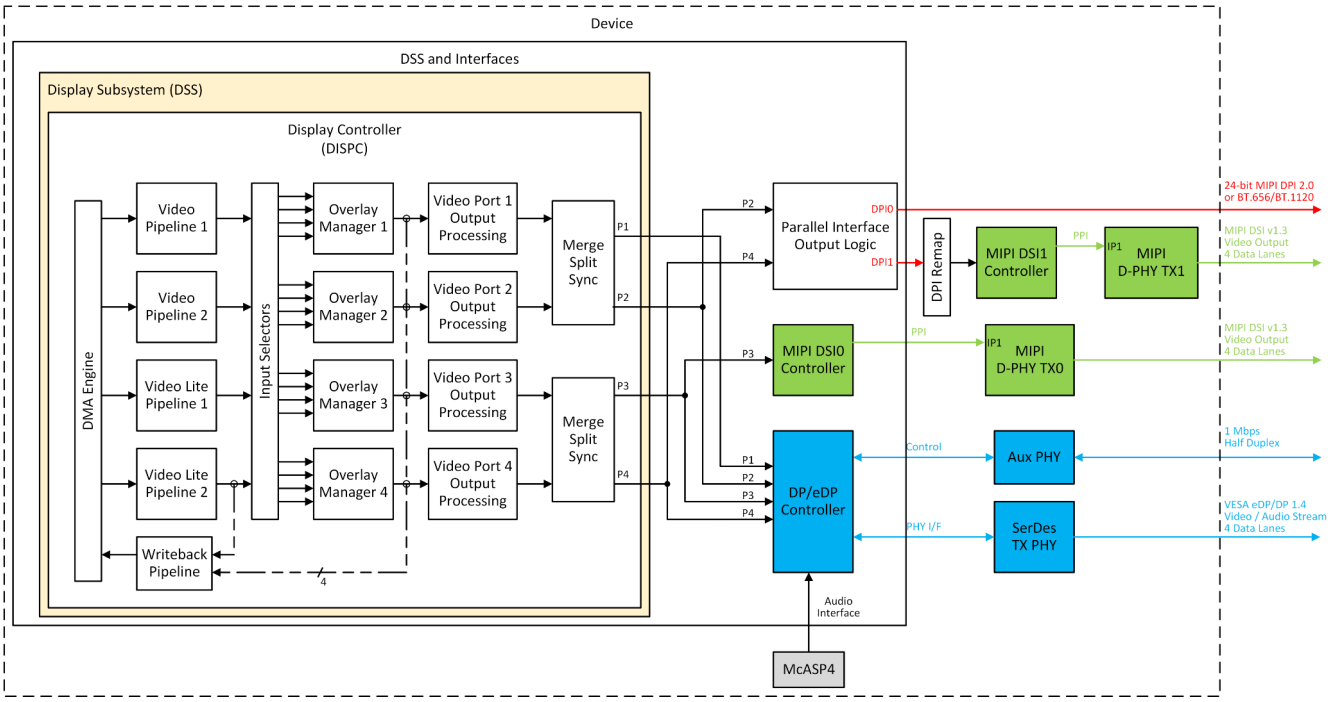
Copyright © 2024 Texas Instruments Incorporated

- DSI1 does not support security enforcement

5.5.10.3 DSS/DSI Integration Details

DSS7 is capable of driving multiple displays in parallel through a combination of interfaces:
- DPI Parallel Interface
- Up to two MIPI Display Serial Interface (DSI) ports
- One embedded Display Port (eDP) controller

The following diagram shows how the 4 video outputs (P[4:1]) of the DSS are mapped to the various display peripherals on the device. The DSS internal registers determine which of the display peripherals are allowed to output content at any time so that content protection is supported through the DSS built-in security capabilities.



5.5.10.3.1 DSS Pixel Clock Sourcing

Three PLLs (PLL16, PLL17, and PLL19) are allocated to the DSS for sourcing the pixel clocks (dpi_[3:0]_in_clk.) (See clocking chapter for details.) PLL16/PLL17 should be the primary high-resolution display pclk sources (e.g. enabling 2.5K or 4K display output). PLL19 is associated closely with the DPI0 port. When dpi_1 or dpi_3 is selected to drive the DPI0 port, the dpi_0/3_in_clk must be sourced from this PLL. When DPI0 is not enabled, PLL19 can be used to source the pixel clock of other display outputs.

The PLL19 clock source can alternatively be driven by an external pixel clock input pin. This allows the DSS DPI output to be generated in genlock with an external clock source. The external pixel clock source for PLL19 HSDIV0_CLKOUT is selected by setting MMR_CTRL0 DPI0_CLK_CTRL[EXT_CLKSEL]=1.

5.5.11 eCAP

Instance	WKUP Domain	MCU Domain	MAIN Domain
eCAP[2:0]			√

5.5.11.1 eCAP Unsupported Features

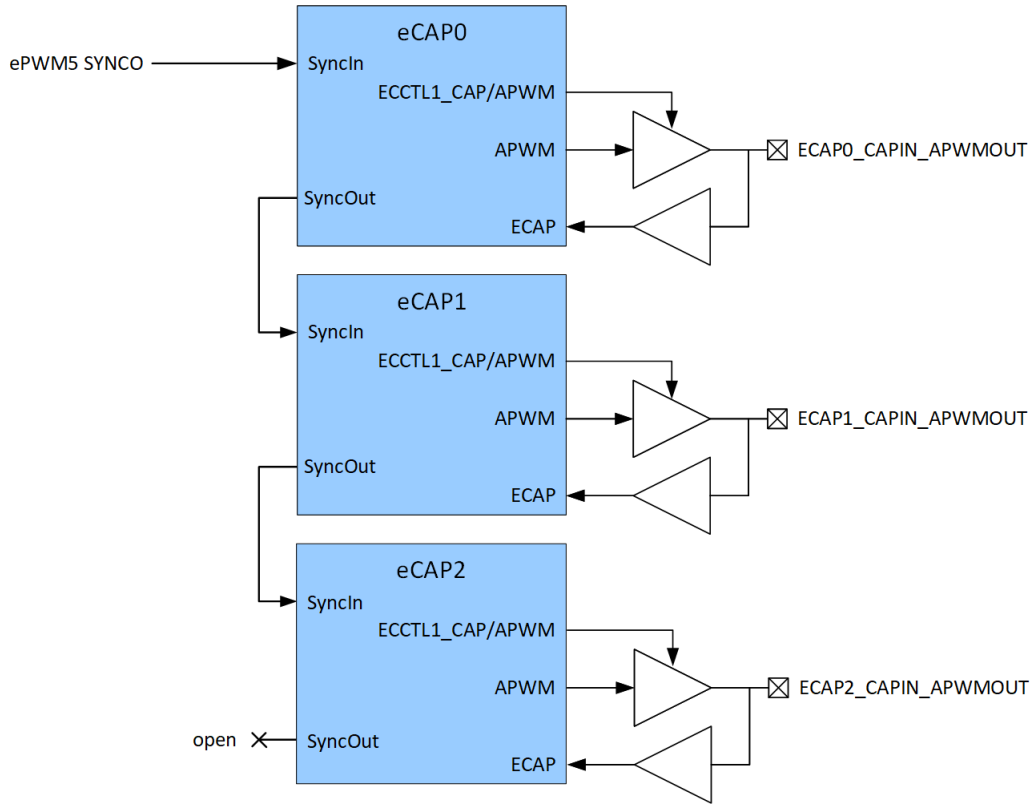
None.

5.5.11.2 eCAP Integration Details

Each eCAP capture event comes from a dedicated external pin which can optionally be configured as an asymmetrical PWM output.

Copyright © 2024 Texas Instruments Incorporated

The eCAP module timebases can be synchronized using sync input signals. The eCAP module synchronization signals are connected in a daisy-chain fashion allowing each eCAP to be synchronized to the previous instance in the chain. Additionally, eCAP0 can be hardware synchronized to the ePWM5 as shown in the synchronization detail below.



5.5.12 ePWM

Instance	WKUP Domain	MCU Domain	MAIN Domain
ePWM[5:0]			√

5.5.12.1 ePWM Unsupported Features

- High-resolution extension
- No on-chip digital comparators are provided. Module comparator high/low limit inputs for ePWMA/B are not used. Trip events are supported through TripZOne inputs.

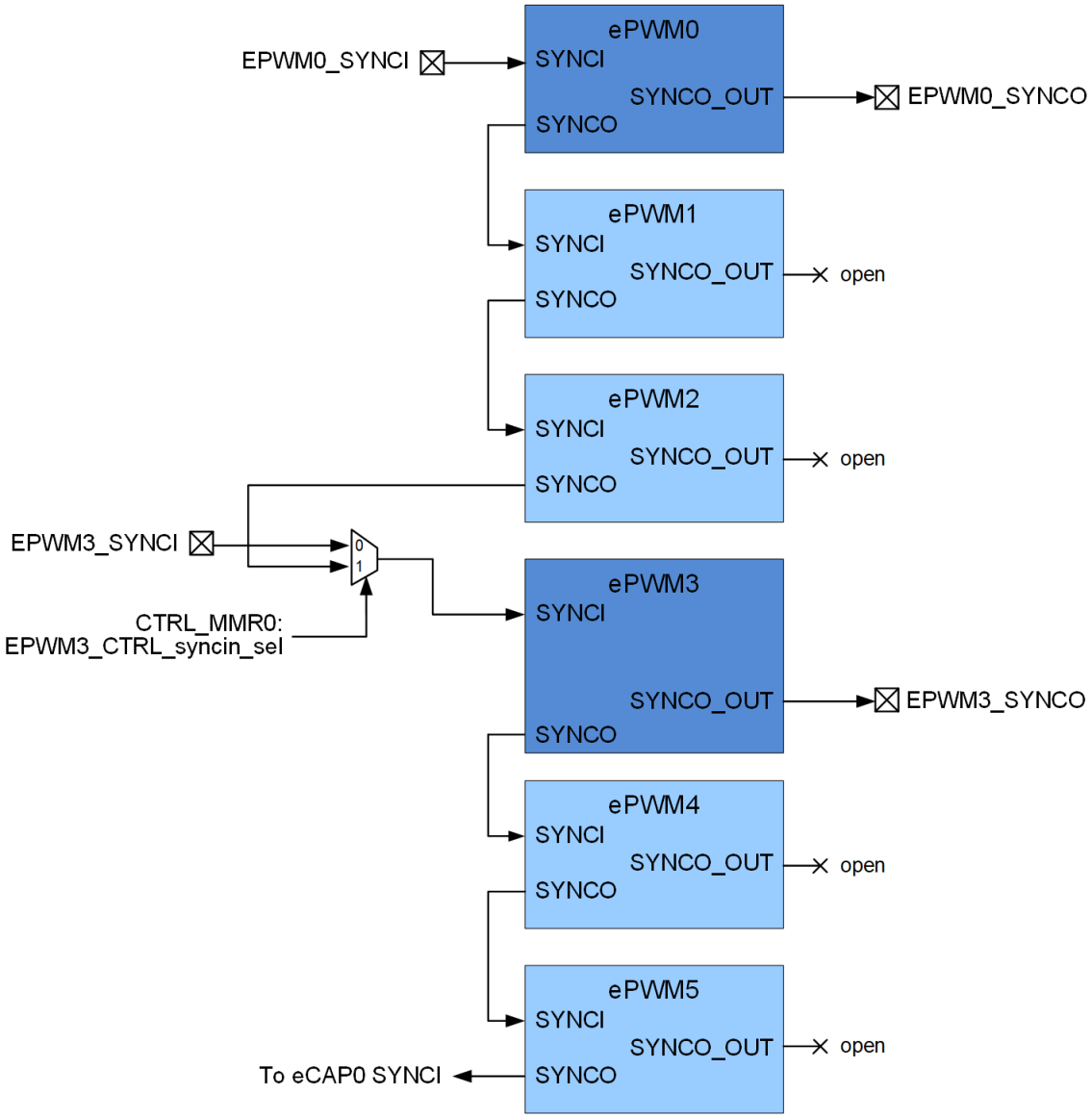
5.5.12.2 ePWM Integration Details

Additional integration details for the ePWM modules on J7AEP and J7AHP are detailed in the following sections.

ePWM Synchronization

The ePWM SYNCI inputs allow the timebase (TBCNT) of an ePWM module to be synchronized with another epWM module or other event. This allows the addition of lead/lag phase control to waveforms generated by the other ePWM module. On J7AEP/J7AHP, ePWM[2:0] SYNCI inputs are daisy-chained together to allow synchronization to each other or to an external SYNCI input pin. The ePWM[5:3] SYNCI inputs are also daisy-chained together and may be synchronized to a separate external SYNCI input pin , or to ePWM[2:0] as shown below:

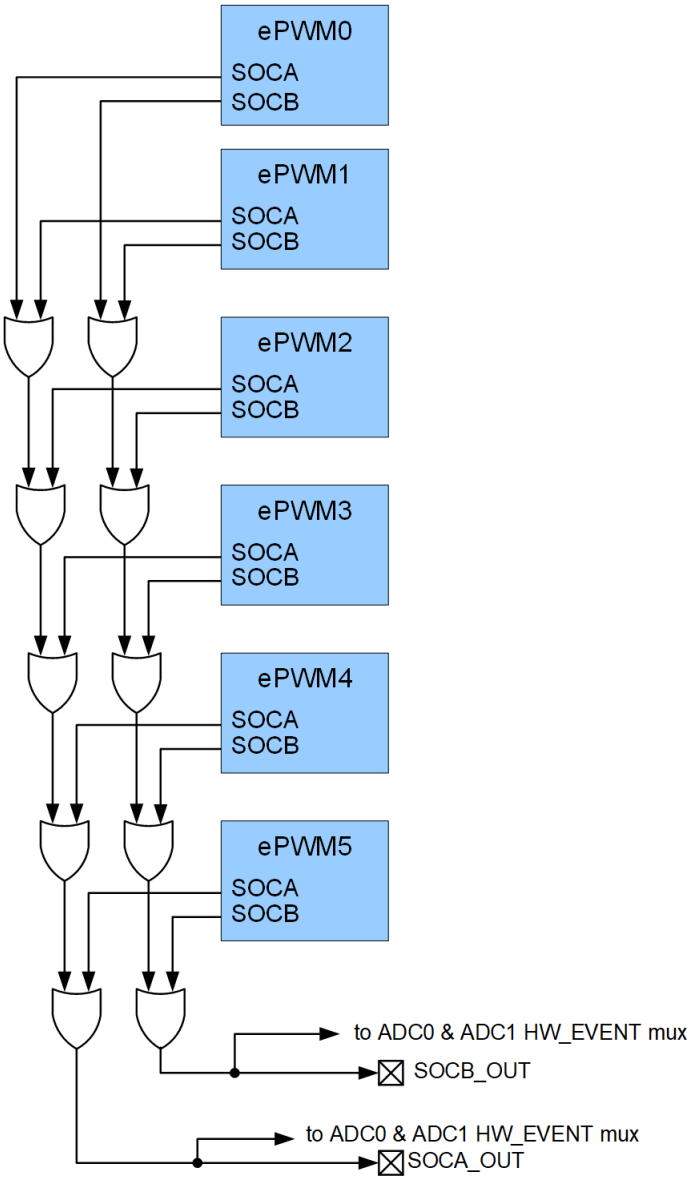
Copyright © 2024 Texas Instruments Incorporated



ePWM SOC Outputs

The ePWM modules provide Start of Conversion (SOCA, SOCB) events that can be used to trigger the on-chip ADCs or off-chip modules. On J7AEP/J7AHP, the SOCA and SOCB events from each ePWM module are ORed together allowing any of the 6 ePWMS to act as the SOCA or SOCB event generator:

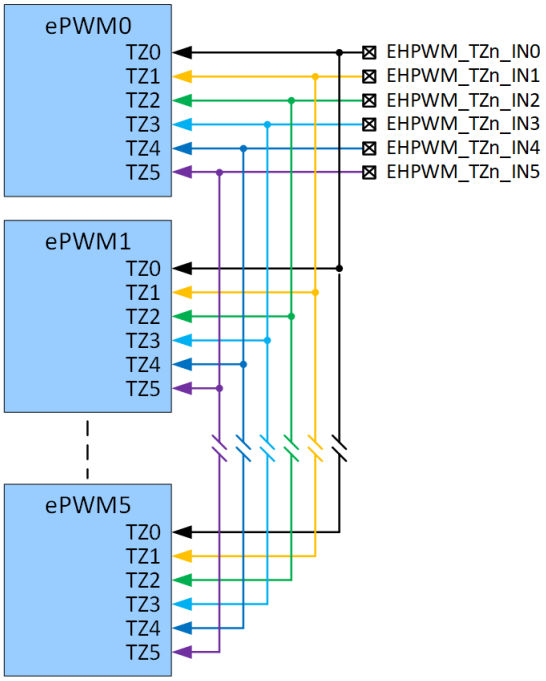
Copyright © 2024 Texas Instruments Incorporated



ePWM Tripzone

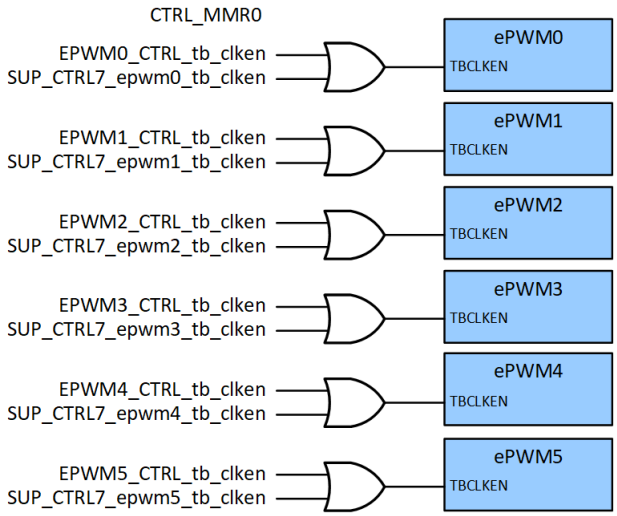
Each ePWM supports 6 tripzone event inputs. On J7AEP/J7AHP, each tripzone input to all six ePWM modules are tied to a common TZn input pin allowing any ePWM trip event to be triggered by any of the six input pins.

Copyright © 2024 Texas Instruments Incorporated



ePWM Timebase Clock Synchronization

The ePWM modules include a timebase counter for pre-scaling the input clock to generate the TBCLK (timebase clock) used to clock the PWM. The timebase counter is enabled via the ePWM TBCLKEN input. This input is driven (high) through the associated CTRL_MMR0 PWM*n*_CTRL_tb_clken register bit (where *n* is the ePWM instance). In order for multiple ePWM module timebase counters to be started synchronously, the TBCLKEN inputs can also be driven by the CTRL_MMR0 SUP_CTRL7_epwm*n*_tb_clken bits. Using the SUP_CTRL7 register allows multiple ePWM TBCLKEN inputs to be enabled with a single register write. The two tb_clken sources are ORed together so that setting the bit in either location will enable the associated ePWM's timebase counter:



5.5.13 ESM

Instance	WKUP Domain	MCU Domain	Main Domain
WKUP_ESM0	√		
MCU_ESM0		√	
ESM0			√

Copyright © 2024 Texas Instruments Incorporated

5.5.13.1 ESM Unsupported Features

The integration of the ESM module in this WKUP and MAIN Domains supports all module features.

The ESM module features listed below are not supported by the integration in the MCU Domain on this Device:

- ERR_O output to a device pin (ERR_O is instead connected as an event to WKUP_ESM0), along with ERR_O output PWM Mode

5.5.13.2 ESM Integration Details

The ERR_O output of the MAIN Domain ESM0 drives the SOC_SAFETY_ERRORn pin directly to indicate a MAIN Domain safety error condition to an external device. The ESM0 ERR_O error output is also connected to the MCU_ESM0 as an error event input. Both Level and PWM Mode are supported on the SOC_SAFETY_ERRORn output signal if the ESM0 ERR_O is not monitored by MCU_ESM0. However, only Level Mode is allowed if the ESM0 ERR_0 error signal is actively monitored by the MCU_ESM0 module.

The MCU_ESM0 ERR_O error output is connected to the WKUP_ESM0 as an error event input, but it is not directly connected to a Device pin. Only Level Mode is allowed for the MCU_ESM0 ERR_0 error output.

The ERR_O output of WKUP_ESM0 drives the MCU_SAFETY_ERRORn pin directly to indicate a WKUP Domain safety error condition to an external device. This pin can also indicate MCU Domain and MAIN Domain safety error conditions via daisy-chaining of the Device ESM modules, provided that the Events corresponding to those daisy-chained ERR_O outputs are enabled.

Figure 5-25 describes the integration of the ESM Modules in the Device.

Copyright © 2024 Texas Instruments Incorporated

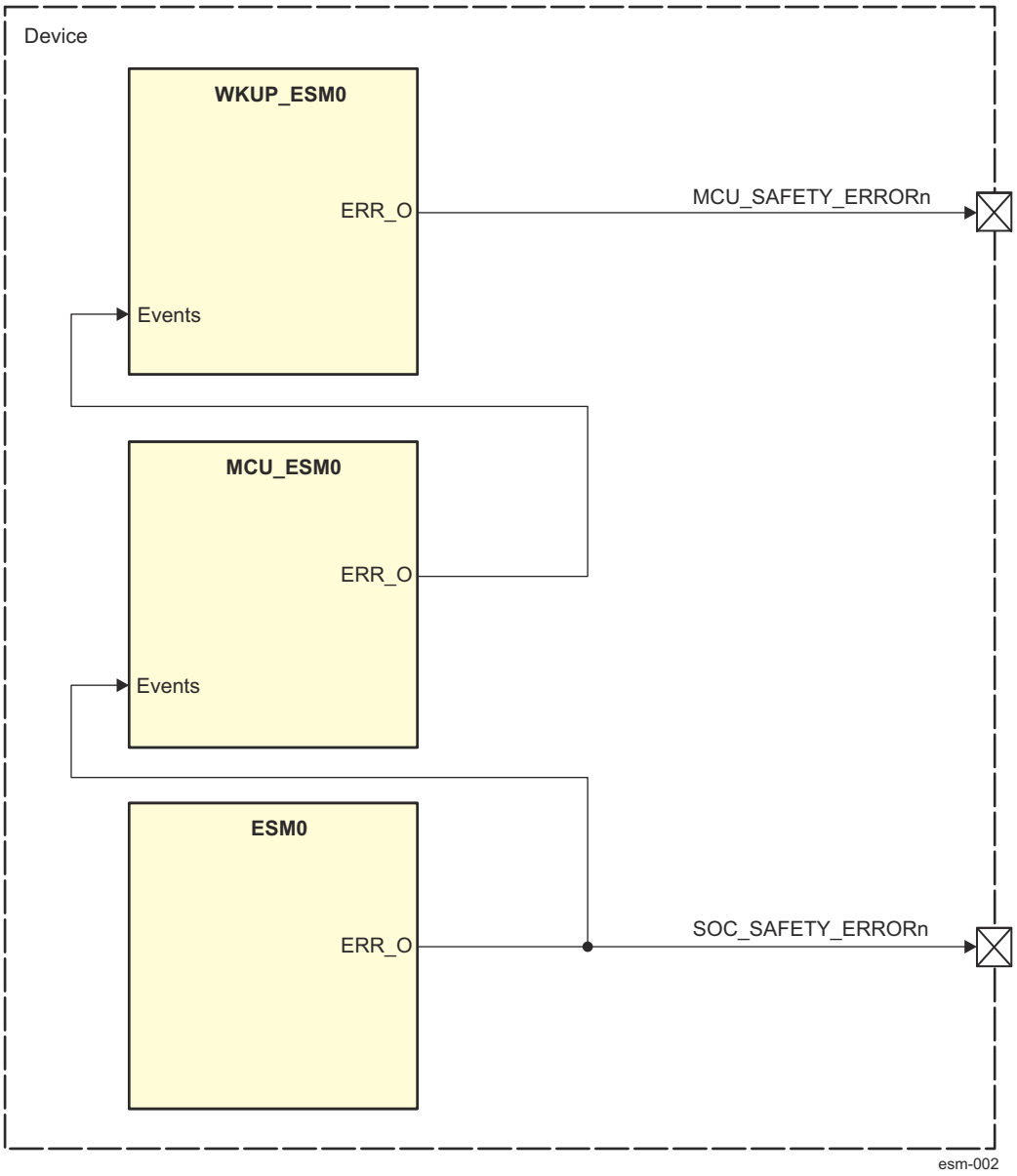


Figure 5-25. ESM Module Integration

5.5.14 FSS

Instance	WKUP Domain	MCU Domain	Main Domain
MCU_FSS0		√	
MCU_OSPI[1:0]		√	

5.5.14.1 FSS Unsupported Features

The FSS module features listed below are not supported by the integration on this Device:

- OSPI DMA in INDAC mode

5.5.14.2 FSS Integration Details

There are no additional FSS integration details.

Copyright © 2024 Texas Instruments Incorporated

5.5.15 GPIO

Instance	WKUP Domain	MCU Domain	Main Domain
WKUP_GPIO[1:0]	√		
GPIO[6,4,2,0]			√

5.5.15.1 GPIO Unsupported Features

The GPIO module features listed below are not supported by the integration on this Device:

- Not all GPIOs are pinned out

5.5.15.2 GPIO Integration Details

GPIO Interrupts

All GPIO interrupts are mapped to a series of event multiplexing Interrupt Router (INTRTR) modules. These multiplexors allow any one of the available GPIO interrupts to be selected and passed on as an event to the various processor interrupt controllers and DMA controllers. Event selection is controlled through associated registers within each INTRTR.

The GPIO Bank interrupts represent a consolidation of all 16 GPIO interrupts associated with each bank. These are also selectable through the INTRTR event multiplexors and can be used to increase the total number of events available to a processor at the expense of full event granularity.

GPIO Multiplexing

The device has six instances of GPIO modules. The GPIO modules are integrated in two groups.

- Group one: WKUP_GPIO0 and WKUP_GPIO1. On I/O pins, this group is named WKUP_GPIO0.
- Group two: GPIO0, GPIO2, GPIO4, and GPIO6. On I/O pins, this group is named GPIO0.

The corresponding groups I/O pins are multiplexed within each group modules.

The GPIO pins are grouped into banks (16 pins per bank and 9 banks per module), which means that each GPIO module provides up to 144 dedicated general-purpose pins with input and output capabilities; thus, the general-purpose interface supports up to 288 (2 group instances x (9 banks x 16 pins)) I/O pins. Because WKPU_GPIOu_[89:143] (u = 0, 1), and GPIOn_[66:143] (n = 0, 2, 4, 6) are reserved in this device, general purpose interface supports up to 155 I/O pins.

GPIO Virtualization

The GPIO module does not support virtualization. Therefore, two or more physical GPIO modules are instantiated within the MAIN and WKUP Domains to provide virtualization and/or isolation of GPIO control between safety-critical and non-safety-critical subsystems. The Device maps each GPIO signal from each GPIO Module in a Domain to the same pins, allowing any of the GPIO Modules in the Domain to control the pin via the VGPIO_SEL fields of the associated PADCONFIG registers. The GPIO interrupts/events are selected between the GPIO instances based on the same VGPIO_SEL fields. The bank interrupts from the GPIO instances are not multiplexed.

Enabling GPIO as a Wakeup Source

During Device DeepSleep power saving mode, the GPIO functional clock is powered down. This would prevent the WKUP_GPIO module from detecting transitions on GPIO pins to be used as wakeup from DeepSleep events. In order to prevent this issue, special clocking and vbus control are implemented as part of the WKUP_GPIO integration to allow GPIO transition to remain detectable.

A clock mux is provided to allow the GPIO VBUS_CLK to be switched to an on-chip clock source prior to gating of the standard clock source (MCU_SYSCLK0/8) and power-down of the off-chip HFOSC0 oscillator. This clock mux is controlled by WKUP_CTRL_MMR0 register bits. Because there is no asynchronous bridge between the WKUP_GPIO module and the WKUP CBASS, the module register may only be accessed when it is clocked using the synchronous MCU_SYSCLK0/8 clock source. Prior to switching the clock source to prepare for DeepSleep, all VBUS accesses to the WKUP_GPIO module must be blocked through the dedicated LPSC using a clock stop request. Note that when wakeup functionality for the WKUP_GPIO is enabled, (through a

Copyright © 2024 Texas Instruments Incorporated

WKUP_CTRL_MMR bit), this clock stop request will not actually propagate to the WKUP_GPIO module (or stop its clock.) Instead, it will be fed back to the LPSC as a clock stop acknowledged. This will cause the associated WKUP_CBASS0 to route all future WKUP_GPIO register accesses to a null endpoint. The WKUP_GPIO LPSC must be maintained in CLKSTOP mode until MCU_SYSCLK0/8 is fully restored upon wakeup from DeepSleep and the WKUP_GPIO clock mux has finished switching back to the normal VBUS clock source.

The following are the steps required (expected to be performed by the SMS) to enable WKUP_GPIO wakeup events prior to DeepSleep entry:

- Determine which events are to be used for DeepSleep wakeup and enable the interrupts through the corresponding GPIO SET_RIS_TRIG and SET_FALL_TRIG registers. Disable non-wakeup interrupts through the corresponding CLR_RIS_TRIG and CLR_FALL_TRIG registers.
- Set the WKUP_GPIO_CTRL_wake_en bit in the WKUP_CTRL_MMR to enable DeepSleep LPSC operation.
- Disable further VBUS access to the WKUP_GPIO through the LPSC clock stop request.
- Change the WKUP_GPIO clock source to CLK_32K or CLK_12M_RC as desired by setting the WAKE_CLK_SEL field in the CTRL_MMR_WKUP_GPIO_CLKSEL register. (Wait at least one clock cycle of the new clock to ensure switch has occurred.)
- Perform normal DeepSleep entry routine (enable SMS wakeup events, clock gate MCU clocks, power down oscillators, etc.)

Once in deep sleep mode, any GPIO transition (low to high or high to low) intended to cause a wakeup must be maintain at its new value for at least 2 of the selected functional clock (CLK_32K or CLK_12M_RC) maximum periods to ensure proper detection. (Note that process variance of the RC clocks must be taken into account). The detected event will be latched in the GPIO INSTAT register and the gpio_lvl_intr wakeup event sent to the SMS to trigger the wakeup FSM.

After wakeup, the SMS must restore the MCU_SYSCLK0/8 operation to allow the wakeup source event to be latched within the SMS. Once this is done the SMS may restore access to the WKUP_GPIO module by reversing the steps above:

- Change the WKUP_GPIO clock source back to MCU_SYSCLK0/8 using the WKUP_GPIO_CLKSEL_clksel bit in the WKUP_CTRL_MMR. (Wait at least 1 of the previously selected clock cycles to ensure glitch-free switch is complete.)
- Re-enable vbus access to the WKUP_GPIO through the LPSC clkstop request.

The specific GPIO source(s) of the wakeup event may be determined by reading the GPIO INSTAT register and cleared by writing 1's to the set bits of the same register.

5.5.16 GPMC

Instance	WKUP Domain	MCU Domain	MAIN Domain
GPMC0			√

5.5.16.1 GPMC Unsupported Features

- 32-bit data path to external memory device (only 16 data lines are pinned out)
- Chip select regions 4-7 (only CS0-3 are pinned out)
- 28-bit memory device addressing (only A[24:0] are pinned out)
- DMA FIFO operation (FIFO operation may be controlled through interrupt synchronization only)

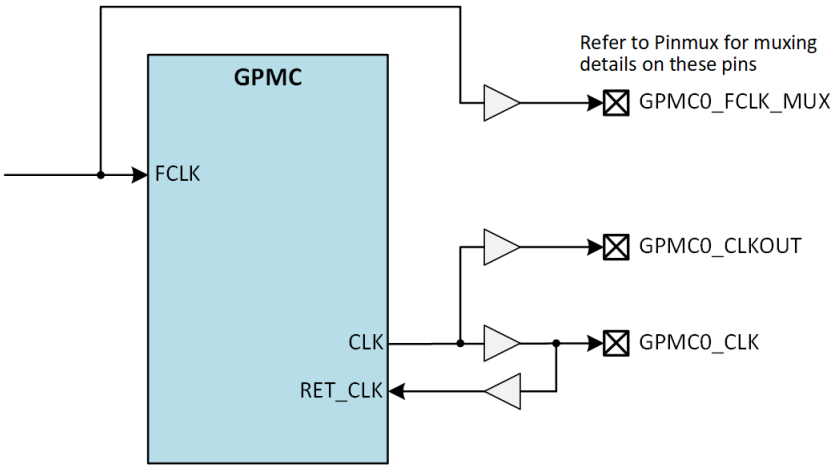
5.5.16.2 GPMC Integration Details

When using synchronous interface protocols, the GPMC_CLK output only toggles during the read or write access cycle. In some applications, it may be desirable to have a continuous clock running at the GPMC interface clock frequency for clocking attached devices. This option is enabled by an optional clock path from the GPMC functional clock input to the GPMC_CLKOUT pin. This clock output (GPMC_FCLK_MUX) can be selected through the standard MUXMODE selection of the GPMC_CLKOUT pin PADCONFIG control register. (See device Pin Muxing for details.).

Note that when using synchronous interface protocols with the continuous clock option described above, the programmer should ensure that the GPMC outputs are timed to the same frequency. (GPMC_CONFIG1_x

Copyright © 2024 Texas Instruments Incorporated

GPMCFCLKDIVIDER = 0). The GPMC_CLK is also output to two pads for signal integrity reasons. One pad is intended for board level use and one is for IO loopback.



5.5.17 GPU

Instance	WKUP Domain	MCU Domain	MAIN Domain
GPU0			√

5.5.17.1 GPU Unsupported Features

- No memory coherency is provided between the GPU and ARM72 or C7x processor cores.

5.5.17.2 GPU Integration Details

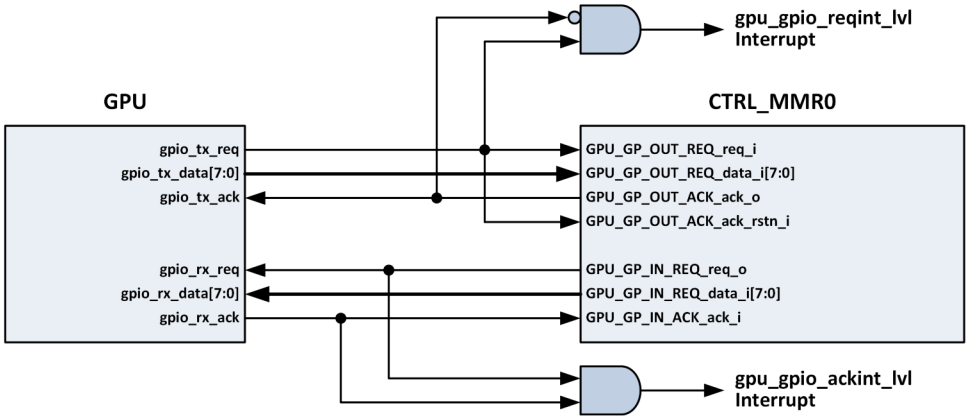
GPU Firmware Processor General Purpose Interface Details

The Firmware Processor General Purpose Interface of the GPU allows for messages to be passed between the GPU and another processor Core on the SoC. The interface consists of two channels:

- Firmware Processor Transmit (Output) Channel
  - Data (8 bits) provided by GPU to another CPU on the SOC
  - Data transfers are initiated by the GPU asserting gpio_tx_req
  - Receipt of Data is Acknowledged by the other CPU asserting gpio_tx_ack
  - When the GPU receives the Acknowledge from the other CPU, it deasserts gpio_tx_req
  - When the other CPU sees gpio_tx_req deasserted, it deasserts gpio_tx_ack
- Firmware Processor Receive (Input) Channel
  - Data (8 bits) provided by another CPU to the GPU
  - Data transfers are initiated by the other CPU asserting gpio_rx_req
  - Receipt of Data is Acknowledged by the GPU asserting gpio_rx_ack

On J7AEP and J7AHP, the GPU's general purpose interface is exposed to each of the SoC CPUs through CTRLMMR0 registers with interrupt signaling in each direction:

Copyright © 2024 Texas Instruments Incorporated

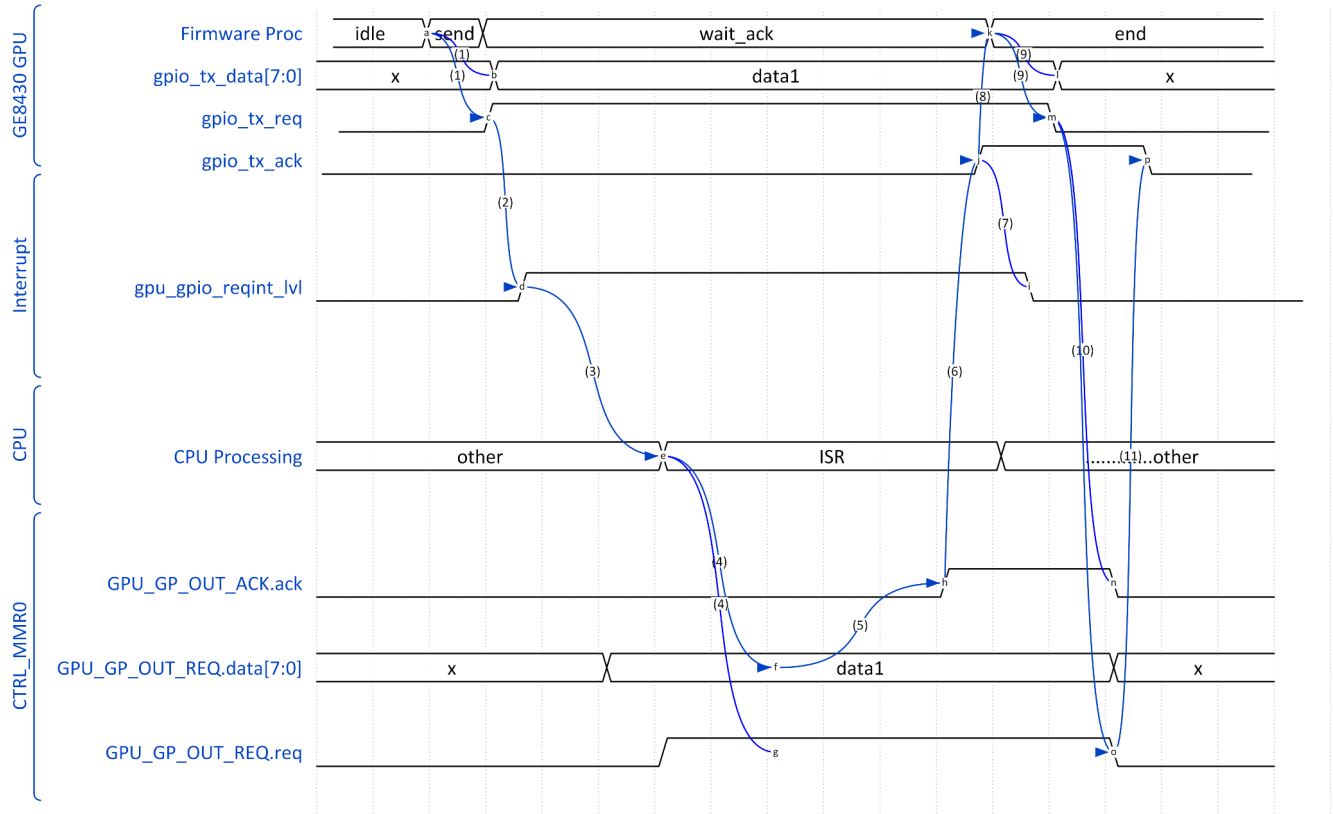


GPU Firmware Transmit (output) Channel Operation

The GPU Transmit Channel allows the GPU firmware processor to send data to an external CPU on the SoC. It operates as follows (see diagram):

1. When the GPU Firmware Processor enters the 'send' state, it begins by driving gpio_tx_data[7:0] with valid data ('data1') and driving gpio_tx_req active. The GPU will drive these signals in the same GPU clock cycle. The firmware then transitions to the wait_ack state where it remains until it sees gpio_tx_ack asserted.
2. Assertion of the gpio_tx_rq signal while gpio_tx_ack is low causes an interrupt request on gpu_gpio_reqint.
3. One of the other CPUs on the SoC responds to the interrupt by entering an ISR
4. As a result of the interrupt (shown here as within the ISR, but not required to be) the CPU reads from the GPU_GP_OUT_REQ register in the CTRL_MMR0 module. The read returns the output data and the active state of the gpio_tx_req signal.
5. After reading the transmitted data, the CPU writes to the GPU_GP_OUT_ACK register to set the ack bit.
6. Setting GPU_GP_OUT_ACK_ack bit causes the gpio_tx_ack signal to be asserted.
7. The interrupt gpu_gpio_reqint interrupt is cleared by the assertion of gpio_tx_ack.
8. The GPU Firmware processor enters the ‘end’ state to terminate the transaction.
9. The GPU Firmware processor terminates the transaction by de-asserting gpio_tx_req. Valid data is no longer required to be output on gpio_tx_data[7:0]
10. The gpio_tx-req de-assertion resets the CTRL_MMR0 GPU_GP_OUT_ACK_ack bit and clears the GPU_GP_OUR_REQ_req bit.
11. The GPU_GP_OUT_ACK_ack bit reset de-asserts the gpio_tx_ack signal completing the handshake and returning the GPU GPIO to the idle state from which a new transfer may begin.

Copyright © 2024 Texas Instruments Incorporated

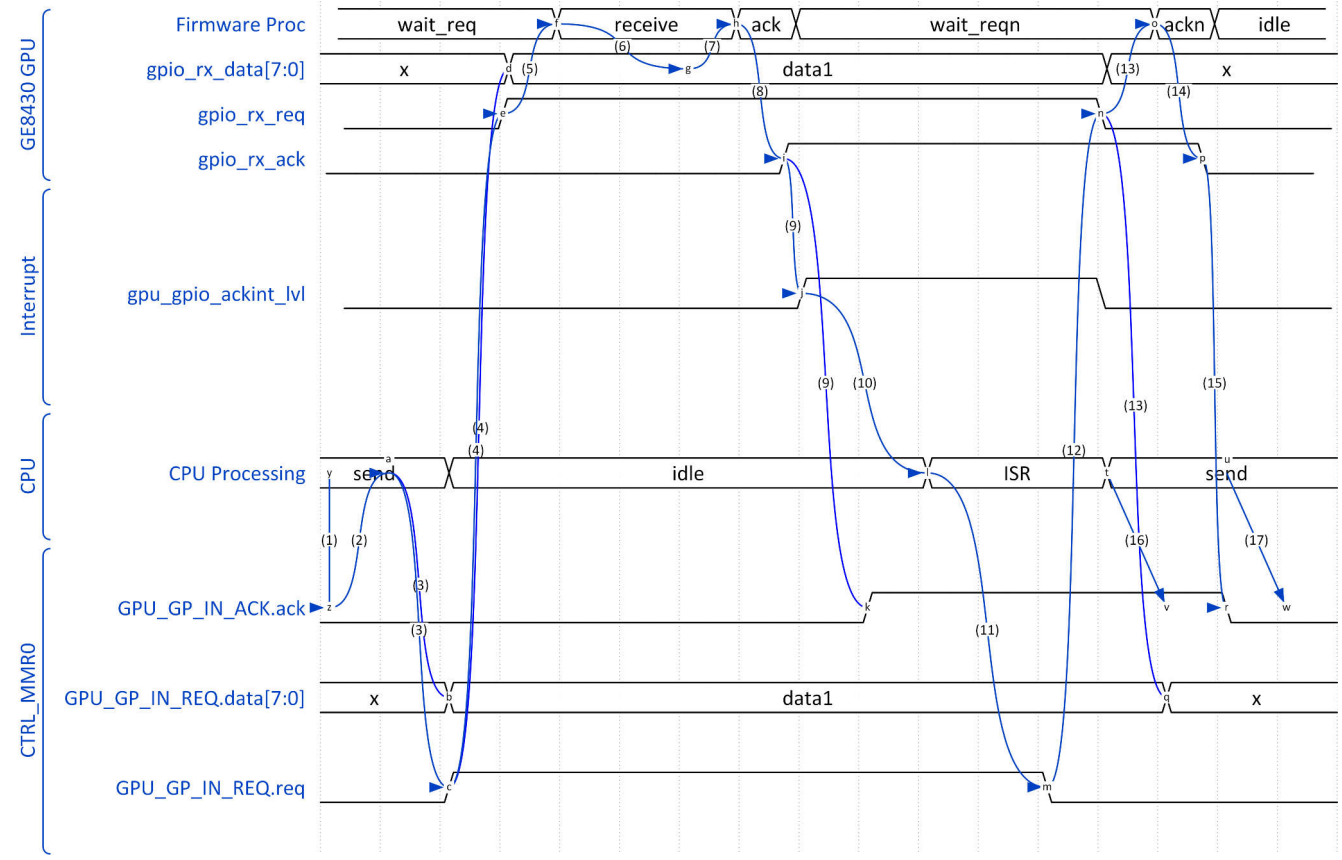


GPU Firmware Receive (Input) Channel Operation

The GPU Receive Channel allows an external CPU on the SoC to send data to the GPU firmware processor. It operates as follows (see diagram):

1. The CPU must poll the GPU_GP_IN_ACK.ack bit for ‘0’ at the beginning of the send operation, to ensure that the GPU has released the gpio_rx_ack signal. If this step is skipped and the GPU still has gpio_rx_ack asserted, then a gpu_rx_ackint interrupt will be sent to the CPU immediately before the GPU has read the GPIO data.
2. After verifying that gpio_rx_ack is de-asserted the CPU may continue the GPIO send process.
3. The CPU writes the send data in GPU_GP_IN_REQ_data[7:0] bits and sets the GPU_GP_IN_REQ_req bit.
4. The setting of the GPU_GP_IN_REQ.req bit causes the gpio_rx_req GPU input to be asserted.
5. The GPU Firmware processor sees gpio_rx_req asserted and enters the GPIO receive state.
6. The GPU Firmware processor reads the valid data from the gpio_rx_data[7:0] inputs
7. The GPU Firmware processor enters the GPIO ack state
8. The GPU Firmware processor acknowledges data read completion by asserting gpio_rx_ack which sets the GPU_GP_IN_ACK.ack bit.
9. The assertion of both gpio_rx_req and gpio_rx_ack activates the gpu_gpio_ackint interrupt
10. The enabled interrupt causes the CPU to enter into the ISR
11. In the ISR, the CPU clears the GPU_GP_IN_REQ_req bit
12. Clearing of GPU_IN_REQ_req de-asserts gpio_rx_req.
13. Upon gpio_rx_req de-assertion, the GPU Firmware processor enters the ackn state
14. The GPU Firmware processor deasserts the gpio_rx_ack signal.
15. De-assertion of gpio_rx_ack clears the GPU_GP_IN_ACK_ack bit enabling initiation of a new transfer
16. Shows that CPU must poll the GPU_GP_IN_ACK_ack bit for ‘0’ at the beginning of the send operation, to ensure that the GPU has released the gpio_rx_ack signal. It must not set GPU_GP_IN_REQ.req here or an extra gpu_gpio_ackint will be generated.
17. Shows that the CPU has polled GPU_GP_IN_ACK_ack low and may begin a new transfer.

Copyright © 2024 Texas Instruments Incorporated



GPU Power Management

The GPU includes a power management control interface to allow the GPU integrated firmware processor to interact with the SoC power management solution in order to throttle GPU power during idle time. A general description of the power management procedure is a follows:

- The GPU Firmware processor initiates a power control operation by outputting the type, domain, and gpu mask values to indicate the type of request and activating the pwrctrl_req output. This generates a gpu_pwrctrl_req interrupt to the SoC Power Management (PM) processor.
- The PM processor reads the requested command from the associated GPU_PWR_REQ register.
- The PM processor will execute the requested Power Control request (through standard LPSC control processes).
- Upon completion of the command the PM processor will set the complete (if successful) or abort (if unsuccessful) bits of the associated GPU_PWR_ACK register.
- Upon receiving an active complete or abort input, the GPU Firmware processor will de-assert the pwrctrl_req output cleating the interrupt and clearing the GPU_PWR_ACK complete and abort bits.

5.5.18 I2C

Table 5-63. I2C Instances

Instances	WKUP Domain	MCU Domain	MAIN Domain
WKUP_I2C0	√		
MCU_I2C[1:0]		√	
MCSPI[6:0]			√

5.5.18.1 WKUP_I2C0 Unsupported Features

The WKUP_I2C0 module features listed below are not supported by the integration on this device:

- SCCB protocol
- DMA mode

Copyright © 2024 Texas Instruments Incorporated

- Debug suspend mode request real time

5.5.18.2 MCU_I2C[1:0] Unsupported Features

The MCU_I2C[1:0] module features listed below are not supported by the integration on this device:
- SCCB protocol
- High speed (3.4 MBPS) operation not supported on MCU_I2C1
- Full I2C electrical compliance on MCU_I2C1, which is implemented using standard LVCMOS I/Os
- DMA mode
- Debug suspend mode request real time
- Clockstop Wakeup Interrupt Connected to DMSC, but instances are in an always on domain.

5.5.18.3 I2C[6:0] Unsupported Features

The I2C[6:0] module features listed below are not supported by the integration on this device:
- SCCB protocol
- High speed (3.4 MBPS) operation on I2C[6:1]
- Full I2C electrical compliance on I2C[6:1], which are implemented using standard LVCMOS I/Os
- DMA mode
- Debug suspend mode request real time

5.5.18.4 I2C Integration Details

WKUP_I2C0, MCU_I2C0, and I2C0 use a an I2C Open Drain Buffer that supports I2C High Speed Mode when operating at 1.8V. At 3.3V the IO does not operate in HS Mode nor does it enable its current source, regardless of the HSMODE and HSMCSE state. There is also a current source on the I2C Open Drain Buffers that will increase the rise time of the open-drain pin by actively pulling up the pin. This signal is controlled by an MMR bit because only one device (the HS Mode Master) may enable its pullup on SCL. The SDA pin uses the same I2C Open Drain Buffer, but the active pullup on SDA is never enabled.

The remaining I2C interfaces on the Device use a standard LVCMOS buffers rather than I2C Open Drain Buffers, and therefore do not support I2C High Speed Mode.

5.5.19 I3C

Instance	WKUP Domain	MCU Domain	Main Domain
MCU_I3C0		√	

5.5.19.1 I3C Unsupported Features

The I3C module features listed below are not supported by the integration on this Device:
- DMA service of I3C FIFOs
- Legacy I2C devices with true Open-Drain I/Os or emulated Open-Drain I/Os
- GPIO Interface

5.5.19.2 I3C Integration Details

I3C Clock Frequency

The MCU_I3C0 interface operates at the MCU R5FSS clock frequency /6 which is nominally 166 MHz. The recommended clock divider stings for the MCU_I3C0 instance based on this clock for the SDR/HDR-DDR push-pull mode are:
- I3C Effective clock frequency = 11.9 MHz
  - 166 MHz / 14
  - CTRL0_i3c value = 1
  - CTRL1_pp_low value = 3
  - Duty Cycle – 28.6% High
- I3C Effective clock frequency = 11.1 MHz
  - 166 MHz / 15
  - CTRL0_i3c value = 2

Copyright © 2024 Texas Instruments Incorporated

  - CTRL1_pp_low value = 1
  - Duty Cycle – 40% High
- I3C Effective clock frequency = 10.4 MHz
  - 166 MHz / 16
  - CTRL0_i3c value = 3
  - CTRL1_pp_low value = 0
  - Duty Cycle – 50% High

5.5.20 MCAN

Instance	WKUP Domain	MCU Domain	Main Domain
MCU_MCAN[1:0]		√	
MCAN[17:0]			√

5.5.20.1 MCAN Unsupported Features

The MCAN module features listed below are not supported by the integration on this Device:
- Debug DMA (Instead, debug messages can be read through the RX FIFO)
- TX DMA Channels 3-31 (Only TX DMA Channels 0-2 have associated PDMA channels)

5.5.20.2 MCAN Integration Details

There are no additional MCAN module integration details.

5.5.21 MMCSD

Instance	WKUP Domain	MCU Domain	MAIN Domain
MMCSD0 (8-bit)			√
MMCSD1 (4-bit)			√

5.5.21.1 MMCSD Unsupported Features

Note that MMCSD0 and MMCSD1 are actually two different IP modules. MMCSD0 IO is fixed at 1.8V for eMMC only. MMCSD1 supports both 1.8V and 3.3V IO and can be used for eMMC, SD, or SDIO interface.

- MMCSD1 does not pin out the SD Card busy LED control signal

5.5.21.2 MMCSD Integration Details

The MMCSD1 module interfaces directly to IO cells that support the SDIO standard. By default, these IOs are controlled by other functions muxed onto these pins. The MMCSD1 will need to be released from reset and an MMR (PHY_CTRL_1_REG) inside EMMCSD4 will need to be programmed to switch the final stage pin muxing before MMC1 may be used.

5.5.22 McASP

Instance	WKUP Domain	MCU Domain	MAIN Domain
McASP[4:0]			√

5.5.22.1 McASP Unsupported Features

- Direct Audio Muting functions (AMUTE and AMUTEIN signals are not pinned out)
- Full 16 channel support on all McASP instances. (Note that channel availability on any McASP instance ultimately a function of pin muxing selection)
  - 16 Channels on McASP1 (only 5 channels, AXR[4:0] are pinned out)
  - 16 Channels on McASP2 (only 5 channels, AXR[4:0] are pinned out)
  - 16 Channels on McASP3 (only 3 channels, AXR[2:0] are pinned out)
  - 16 Channels on McASP4 (only 5 channels, AXR[4:0] are pinned out)

Copyright © 2024 Texas Instruments Incorporated

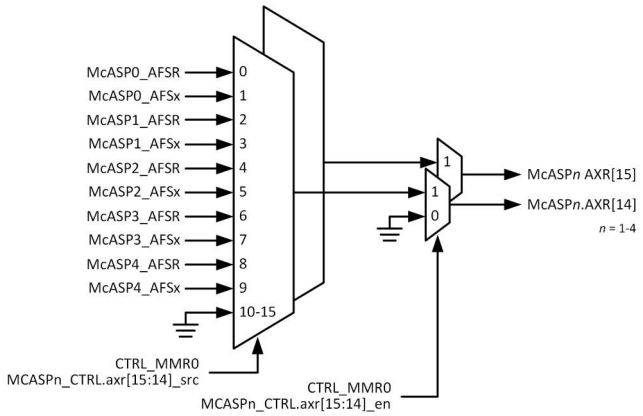
5.5.22.2 McASP Integration Details

AHCLKX and AHCLKR Clock Sources

The McASP AHCLKX and AHCLKR inputs may be driven either by device clock input pins (those with associated AHCLK clock muxing) or by the ATL output clocks. The AHCLKX and AHCLKR outputs do not have explicit output pins. Instead the device AUDIO_EXT_REFCLK[1:0] pins may be driven either by the associated McASP AHCLK outputs or by the ATL output clocks. See the clocking section for details.

Cross-McASP Synchronization using Frame Sync Feedback

In order to align frame syncs across McASPs that may be feeding DACs with different formats but the same frequency, the frame sync of McASPx can be optionally fed into a receive data pin of McASPy. The device includes a multiplexor to allow any McASP transmit or receive frame syn to be connected to receive channels 14 & 15 of any other McASP:



If this serializer is enabled as a receiver, then the frame sync of McASPx will appear as receive data for McASPy. For example, with both McASPs configured for 16-bit data, 2 slots, one might make the conclusions shown in the table below based on receive data. Note that the purpose here is not to actually 'receive' the accurate frame sync as data that is processed. Rather, it is to inspect the relative timing of the frame syncs, and then make an adjustment to the width of the bit clock of McASPx until perfect (ex. #4 from Table 31-9) or good enough alignment (ex. #3, #5) is achieved.

To adjust the McASP transmit bit clock, the CLKADJ bits 17:16 of the ACLKXCTL register can be used to lengthen (write 2'b10) or reduce (write 2'b01) the McASP bit clock period in a one-shot fashion. After each adjustment the frame sync should be checked again and the process repeated until the two frame syncs are aligned. For example, if McASPy frame sync is leading McASPx frame sync, the expectation would be to write 2'b01 to McASPx's CLKADJ bits, which will shorten one bit clock period of McASPx and cause the next frame sync to occur one input clock cycle earlier (input clock to McASP internal transmit clock divider). This would be repeated until the two Frame Syncs are within optimal alignment.

This process is expected to be carried out once, before any audio transfers occur. It is not intended to be used for sample rate conversion but rather initial phase alignment of transmit data across McASPs. It should also be noted that exact alignment as represented in the table may not actually be ideal. For example, there may be filtering delay inside the DACs that are fed by the McASPs. If the filter delay between two DACs is different, an offset in alignment of the frame syncs may actually be what is needed to align the analog audio output across channels. One positive of this mechanism is that any arbitrary alignment of frame sync signals can be achieved to within +/- 1 bit clock by adjusting the target value for the "Received At McASPy" column.

Sample	Received at McASPy	Alignment Conclusion (assumes active low frame sync)
1	1110_0000_0000_0000_0001_1111_1111_1111	Frame Sync of McASPy Leads Frame Sync of McASPx Slightly
2	0000_0000_0000_0011_1111_1111_1111_1100	Frame Sync of McASPy Lags Frame Sync of McASPx Slightly

Copyright © 2024 Texas Instruments Incorporated

Sample	Received at McASPy	Alignment Conclusion (assumes active low frame sync)
3	1000_0000_0000_0000_0111_1111_1111_1111	Frame Syncs are aligned to within +/- 1 Bit Clock
4	0000_0000_0000_0000_1111_1111_1111_1111	
5	0000_0000_0000_0001_1111_1111_1111_1110	

Enhanced Display Port Audio

Note that the eDP uses McASP4 AXR[3:0] exclusively to provide audio data to the DisplayPort interface. If the eDP function is not required, McASP4 is available for general usage.

5.5.23 McSPI

Table 5-64. McSPI Instances

Instances	WKUP Domain	MCU Domain	MAIN Domain
MCU_MCSPI[2:0]		√	
MCSPI[7:0]			√

5.5.23.1 MCSPI Unsupported Features

The MCSPI module features listed below are not supported by the integration on this device:

- Target mode wakeup
- Retention during power down
- MCU_MCSPI2 External pins
- MCSPI4 External pins and Initiator mode and Channels 1,2,3

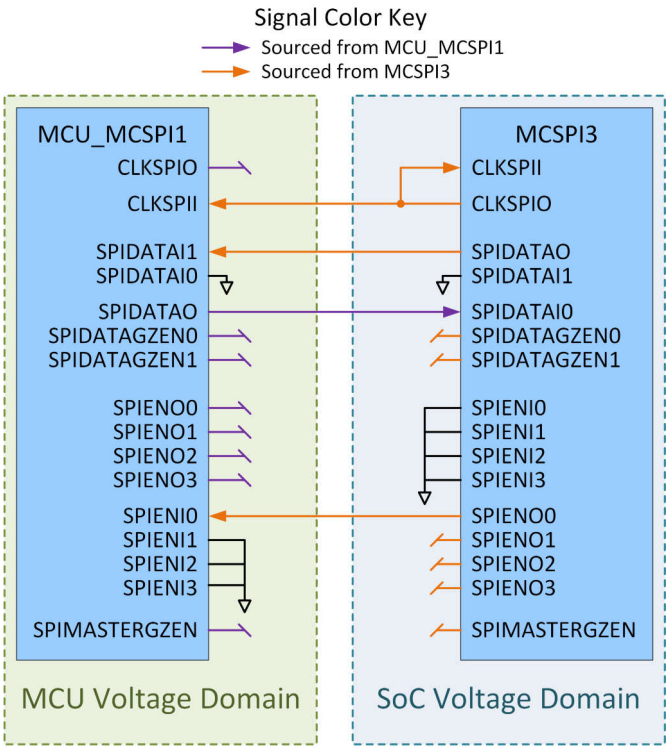
5.5.23.2 MCSPI Integration Details

MAIN-MCU SPI Connectivity

The MCU_MCSPI1 and MCU_MCSPI2 modules include internal connectivity to the MCSPI3 and MCSPI4 modules in the MAIN SoC domain. By default, the MAIN MCSPI3 is connected as Initiator to MCU_MCSPI1 as Target. Alternatively, MAIN MCSPI3 and MCU_MCSPI1 can be pinned out externally. System-level muxes controlled by MCU_SPI1_CTRL[SPI1_LINKDIS], MCU_SPI1_CLKSEL[MSTR_LB_CLKSEL], and SPI3_CLKSEL[MSTR_LB_CLKSEL] allow MCU_MCSPI1 inputs to be controlled from either the I/O pin inputs or the MCSPI3 module.

When the MCU_SPI_CTRL[SPI1_LINKDIS] bit is 1'b0 (default), MCU_MCSIP1 is linked internally to MCSPI3, as shown below:

Copyright © 2024 Texas Instruments Incorporated



MCU_MCSPI1 Connectivity
MCU_SPI_CTRL[SPI_LINKDIS] = 1'b0

When the MCU_SPI_CTRL[SPI1_LINKDIS] bit is 1'b1, both MCU_MCSPI1 and MCSPI3 operate independently using device pins. The device PINMUX must be appropriately programmed to select the desired MCSPI signal functions.

Copyright © 2024 Texas Instruments Incorporated

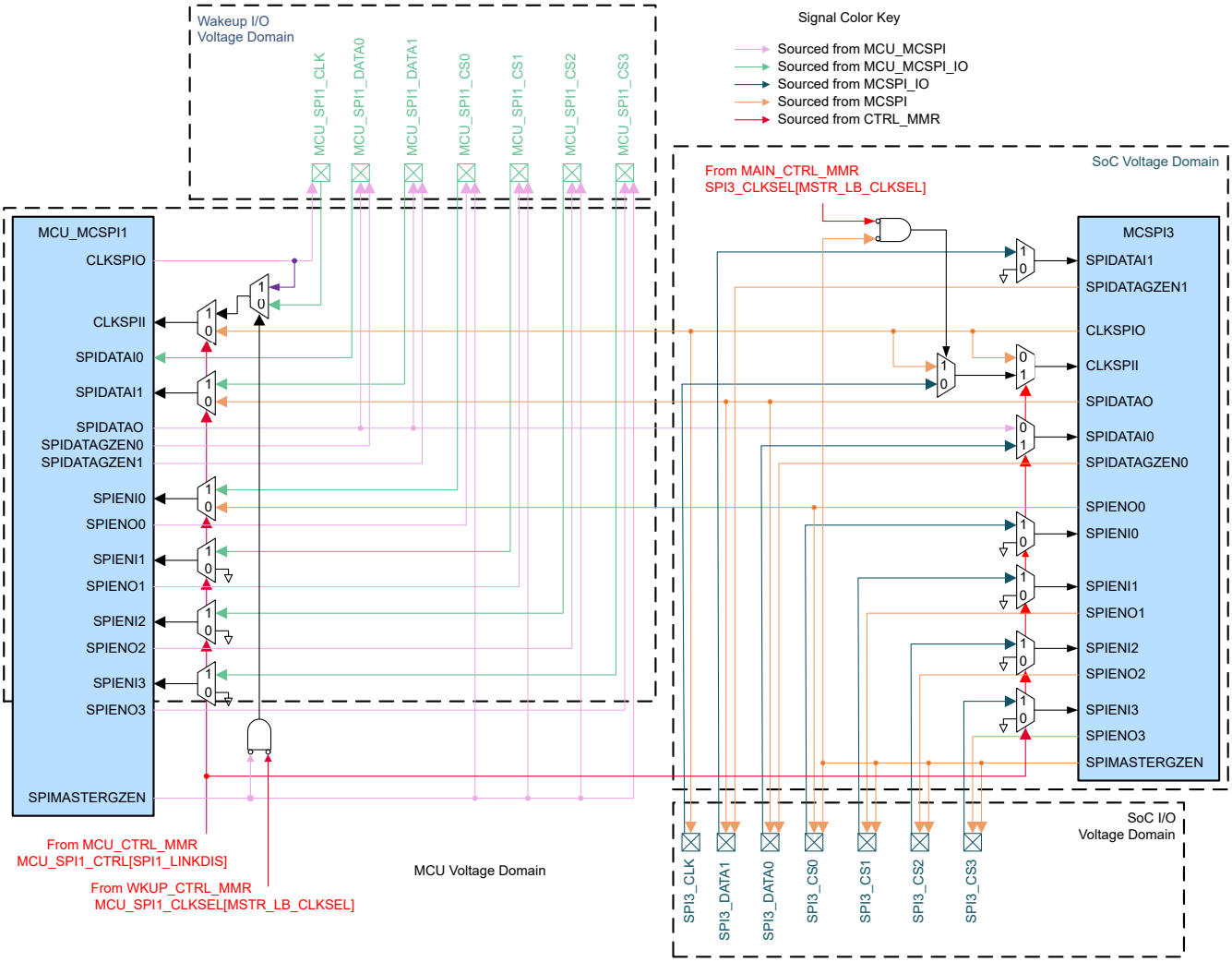


Figure 5-26. MCU_MCSPI1 Connectivity Details

The MCU_MCSPI2 and MCSPI4 modules are not pinned out externally, but MCSPI4 is connected directly as a Target to the MCU_MCSPI2 module as an Initiator as shown in Figure 5-27.

Copyright © 2024 Texas Instruments Incorporated

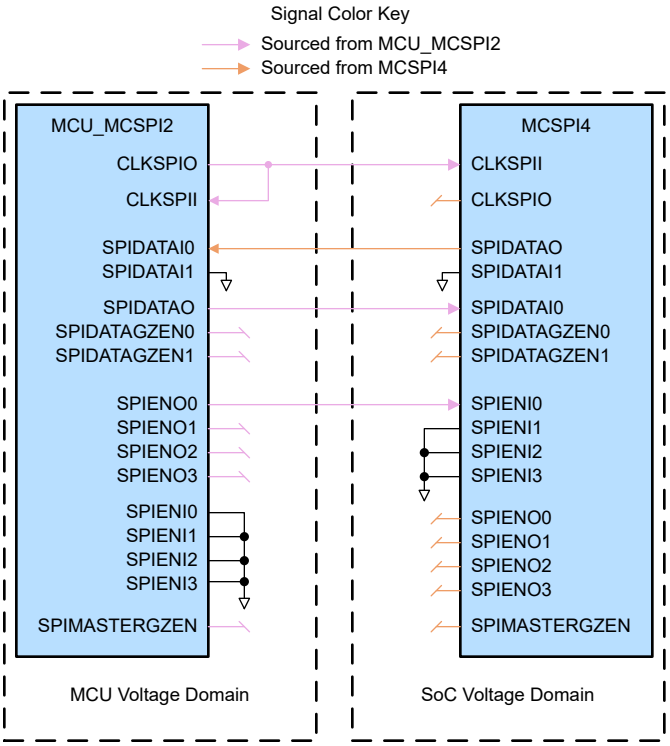


Figure 5-27. MCSPI4 Connectivity Details

Copyright © 2024 Texas Instruments Incorporated

5.5.24 Navigator Subsystem (NAVSS)

This section contains the integration details for the NavSS modules on this device. For Further information, see the *Navigator Subsystem (NAVSS)* section of the *Data Movement Architecture* chapter.

5.5.24.1 Module Allocations

Table 5-65. NAVSS Allocation within Device Domains

Instance	Domain		
	WKUP	MCU	MAIN
MCU_NAVSS0	-	√	-
NAVSS0	-	-	√

5.5.24.2 Main Navigator SubSystem (NAVSS)

5.5.24.2.1 Global Event Map (All NavSS)

The global event map for all Navigator events is shown in Table 5-66.

Table 5-66. Global Event Map

Destination		Offset	PSIL Routed Slots	Actual Slots
NAVSS or External	Port			
NAVSS0	UDMASS_INTR_AGGR0 SEVI(1)	0 (0k)	8192 (8k)	4608 (4.5k)
-	Reserved	8192 (8k)	-	-
MCU_NAVSS0	UDMASS_INTR_AGGR0 SEVI	16384 (16k)	2048 (2k)	1536 (1.5k)
external	DMSC INTR_AGGR0 SEVI	18432 (18k)	2048 (2k)	-
NAVSS0	INTR_AGGR0 SEVI	20480 (20k)	2048 (2k)	1024 (1k)
NAVSS0	INTR_AGGR1 SEVI	22528 (22k)	1024 (1k)	1024 (1k)
-	Reserved	23552 (23k)	-	-
NAVSS0	UDMASS_INTR_AGGR0 MEVI(2)	32768 (32k)	2048 (2k)	512 (0.5k)
MCU_NAVSS0	UDMASS_INTR_AGGR0 MEVI	34816 (34k)	1024 (1k)	128 (0.125k)
-	Reserved	35840 (35k)	-	-
NAVSS0	UDMASS_INTR_AGGR0 GEVI(3)	36864 (36k)	2048 (2k)	512 (0.5k)
-	Reserved	38912 (38k)	-	-
MCU_NAVSS0	UDMASS_INTR_AGGR0 GEVI	39936 (39k)	1024 (1k)	256 (0.25k)
external	PDMA_MCU0 LEVI(4)	40960(40.000k)	128	-
external	PDMA_MCU1 LEVI	41088 (40.125k)	128	-
external	PDMA_MCU2 LEVI	41216 (40.250k)	128	-
external	PDMA_ADC LEVI	41344 (40.375k)	128	-
external	PDMA_MAIN_DEBUG LEVI	41984 (41.000k)	128	-
external	PDMA_MAIN_MCASP LEVI	42112 (41.125k)	128	-
external	PDMA_MAIN_MISC LEVI	42240 (41.250k)	-	-
external	PDMA_MAIN_USART LEVI	42368 (41.375k)	128	-
external	VPAC	42496 (41.500k)	512	-
external	Reserved	43008 (42k)	-	-
NAVSS0	UDMA Triggers	49152 (48k)	1024 (1k)	1024 (1k)
NAVSS0	BCDMA Triggers	50176 (49k)	1024 (1k)	1024 (1k)
MCU_NAVSS0	UDMA Triggers	56320 (55k)	256 (0.25k)	256 (0.25k)
external	MSMC DRU	61440 (60k)	1024 (1k)	320 (.3125k)
-	Reserved for MSMC DRU	62464 (61k)	-	-

(1) SEVI – Mappable source input
(2) MEVI – Multicast event input

Copyright © 2024 Texas Instruments Incorporated

(3) GEVI – Global counting event input
(4) LEVI – Local event input

5.5.24.2.2 PSIL System Thread Map (All NAVSS)

Endpoints in MCU_NAVSS0 can be paired with NAVSS0 and vice versa as shown in Table 5-67 (source/destination pairs). The configuration proxy in each subsystem can configure any endpoint in the map on either subsystem without restriction.

Table 5-67. System Thread MAP for all PSIL Threads

Thread Number	NAVSS Instance	Endpoint
0x0000	NAVSS0	Configuration Proxy
0x0001	MCU_NAVSS0	Configuration Proxy
0x0002-0x0007	-	Reserved
0x0008	NAVSS0	UDMAP0 TRSTRM
0x0009-0x001F	NAVSS0	Reserved for future UDMA TRSTRM instances
0x0020	NAVSS	UDMAP0 CFGSTRM
0x0021	MCU_NAVSS0	UDMAP0 CFGSTRM
0x0022	NAVSS0	BCDMA0 CFGSTRM
0x0021-0x003F	-	Reserved for future UDMA CFGSTRM instances
0x0040-0x0FFF	-	Reserved
0x1000-0x1FFF	NAVSS0	UDMAP0 Threads
0x2000-0x2FFF	NAVSS0	BCDMA0 Threads
0x3000-0x42FF	NAVSS0	Reserved
0x4300-0x43FF	NAVSS0	PDMA_MAIN_DEBUG
0x4400-0x44FF	NAVSS0	PDMA_MAIN_MCASP
0x4500-0x45FF	NAVSS0	Reserved
0x4600-0x46FF	NAVSS0	PDMA_MAIN_MISC
0x4700-0x47FF	NAVSS0	PDMA_MAIN_USART
0x4800-0x481F	NAVSS0	MSMC
0x4820-0x48FF	NAVSS0	VPAC/DMPAC
0x4900-0x49FF	NAVSS0	CSI
0x4A00-0x4AFF	NAVSS0	CPSW
0x4B00-0x5FFF	-	Reserved
0x6000-0x6FFF	MCU_NAVSS0	UDMAP0 Threads
0x7000-0x70FF	MCU_NAVSS0	CPSW0 (2-port MCU_CPSW0)
0x7100-0x71FF	MCU_NAVSS0	MCU_PDMA0
0x7200-0x72FF	MCU_NAVSS0	MCU_PDMA1
0x7300-0x73FF	MCU_NAVSS0	MCU_PDMA2
0x7400-0x74FF	MCU_NAVSS0	MCU_PDMA_ADC
0x7500-0x75FF	MCU_NAVSS0	SAUL0
0x7600-0x7FFF	-	Reserved

5.5.24.2.3 VBUSM Route ID Table

The VBUSM Master routeIds are listed in Table 5-68.

Table 5-68. VBUSM Route IDs

VBUSM Interface	Route Id(s)	Order Id(s)
MSMC0_MST	0-127	8-15
MSMC1_MST	255	0-15
NAV_DDR0		

Copyright © 2024 Texas Instruments Incorporated

Table 5-68. VBUSM Route IDs (continued)

VBUSM Interface	Route Id(s)	Order Id(s)
NAV_DDR1		
NAV_SRAM0		
NAV_SRAM1		
PROXY0	160	0-15
SEC_PROXY0	216	0-15
RINGACC0.DST	200	0-15
UDMA0.MEM0	168	0-15
UDMA0.MEM1	169	0-15
UDMA0.UMEMW	170	0-15
UDMA0.UMEMR	171	0-15

5.5.24.2.4 NAVSS Interrupt Router Configuration

The Interrupt Router is not configured with an INTD (no_intd = 1), it performs only interrupt muxing.

Please see the *Interrupts (inputs)* sheet of the Appendix Spreadsheet for the NAVSS Interrupt Router (NAVSS0_INTR_0) Input Mappings, and the *Interrupts (outputs)* sheet for the connections of the NAVSS Interrupt Router (NAVSS0_INTR_0) Outputs.

5.5.24.2.5 NAVSS Ring Accelerator Configuration

Each UDMA-P has an associated dedicated Ring Accelerator that is accessible by it alone (not accessible by other subsystems). The UDMA-P configuration is listed in Table 5-69.

Table 5-69. RINGACC Configuration Parameters

Module Instance	Parameters		
	Ring Count	Number of Monitors	Proxy Target Base
NAVSS0_UDMASS_RINGACC0	1024	32	0x000038000000

Table 5-70 shows the RINGACC ring mapping.

Table 5-70. RINGACC Ring Mapping

Mapping	Rings	Description
NAVSS0_UDMASS0_UDMAP0 Transmit	ring[340:0]	UDMAP0 Transmit Channels
NAVSS0_UDMASS0_UDMAP0 Receive	ring[422:341]	UDMAP0 Receive Channels
General Purpose	ring[767:423]	General-purpose rings
NAVSS0_SEC_PROXY0	ring[877:768]	SEC_PROXY0 rings
General Purpose	ring[1023:878]	General-purpose rings

5.5.24.3 MCU Navigator Subsystem (MCU NAVSS)

5.5.24.3.1 Global Event Map

The global event map for all navigator events is shown in Table 5-66.

5.5.24.3.2 PSIL System Thread Map (All NAVSS)

Endpoints in MCU_NAVSS0 can be paired with NAVSS0 and visa versa as shown in Table 5-67 (source/destination pairs). The configuration proxy in each subsystem can configure any endpoint in the map on either subsystem without restriction.

5.5.24.3.3 MCU NAVSS VBUSM Route ID Table

The VBUSM Master routeIds are listed in Table 5-71.

Copyright © 2024 Texas Instruments Incorporated

Table 5-71. MCU VBUSM Route IDs

VBUSM Interface	Route ID(s)	Order ID(s)
MSMC0_MST	0-127	8-15
MSMC1_MST	255	0-15
NAV_MCU_DST0		
PROXY0	3616	0-15
SEC_PROXY0	3640	0-15
RINGACC0.DST	3624	0-15
UDMA0.MEM0	3584	0-15
UDMA0.MEM1	3585	0-15
UDMA0.UMEMW	3586	0-15
UDMA0.UMEMR	3587	0-15

5.5.24.3.4 MCU NAVSS Interrupt Router Configuration

The Interrupt Router is not configured with an INTD (no_intd = 1), it performs only interrupt muxing.

See the *Interrupts (inputs)* sheet of the Appendix Spreadsheet for the MCU NAVSS Interrupt Router (MCU_NAVSS0_INTR_ROUTER_0) Input Mappings, and the *Interrupts (outputs)* sheet for the connections of the MCU NAVSS Interrupt Router (MCU_NAVSS0_INTR_ROUTER_0) Outputs.

5.5.24.3.5 MCU NAVSS UDMASS Interrupt Aggregator Configuration

Table 5-72. Interrupt Aggregators Parameters

Module Instance	Parameters				
	VINTR (1)	SEVI (2)	GEVI (3)	LEVI (4)	MEVI (5)
MCU_NAVSS0_UDMASS_INTR_AGGR0	256	1536	256	12 (4 + 8 external)	128

(1) VINTR – Number of Virtual Interrupt outputs. These are connected to the interrupt router inputs. Value can be read from INTA_INTCAP/UDMA_INTA_INTCAP register.
(2) SEVI – Number of Main (Steerable) Events. Value can be read from INTA_INTCAP/UDMA_INTA_INTCAP register.
(3) GEVI – Number of Countable Events. Value can be read from INTA_AUXCAP/UDMA_INTA_AUXCAP register.
(4) LEVI – Number of Local Event inputs. Value can be read from INTA_AUXCAP/UDMA_INTA_AUXCAP register.
(5) MEVI – Number of Multicast Events. Value can be read from INTA_AUXCAP/UDMA_INTA_AUXCAP register.

Note

4 local (LEVI) pulse interrupts are from the MCRC0 module.

If EVENT_PEND_INTR[3:0] needs to be converted to level interrupts, then UDMASS_INTR_AGGR0 can be used to generate events to the PSILSS. PSILSS would route events back to the interrupt aggregator to be turned into level output interrupts.

See the *Interrupts (inputs)* sheet of the Appendix Spreadsheet for the MCU NAVSS UDMASS Interrupt Aggregator (MCU_NAVSS0_UDMASS_INTA_0) Input Mappings, and the *Interrupts (outputs)* sheet for the connections of the MCU NAVSS UDMASS Interrupt Aggregator (MCU_NAVSS0_UDMASS_INTA_0) Outputs.

5.5.24.3.6 MCU NAVSS UDMA Configuration

Table 5-73 shows the UDMA configuration parameters in this SoC.

Table 5-73. UDMA Configuration Parameters

Module Instance	Parameters(1)				
	tchan_cnt	rchan_cnt	echan_cnt (2)	hchan_cnt	rflow_cnt(3)
MCU_NAVSS0_USMASS_UDMAP0	48	48	0	2	96

(1) Parameter values can be read from the capabilities registers. See CAP2 and CAP3 registers.
(2) External UTC channel count
(3) Rx flow table entry count

Copyright © 2024 Texas Instruments Incorporated

5.5.24.3.7 MCU NAVSS Ring Accelerator Configuration

Each UDMA-P has an associated dedicated Ring Accelerator that is accessible by it alone (not accessible by other subsystems). The UDMA-P configuration is listed in Table 5-74.

Table 5-74. RINGACC Configuration Parameters

Module Instance	Parameters		
	Ring Count	Number of Monitors	Proxy Target Base
MCU_NAVSS0_UDMASS_RINGACC0	286	32	0x00002B000000

Table 5-75 shows the RINGACC ring mapping.

Table 5-75. RINGACC Ring Mapping

Mapping	Rings	Description
MCU_NAVSS0_UDMASS_UDMAP0 Transmit	ring[47:0]	48 UDMAP0 transmit channels
MCU_NAVSS0_UDMASS_UDMAP0 Receive	ring[95:48]	48 UDMAP0 receive channels
General purpose	ring[255:96]	General-purpose rings
MCU_NAVSS0_SEC_PROXY0	ring[285:256]	SEC_PROXY0 rings

Note

For Ring Accelerator functional description, see *Ring Accelerator*.

Table 5-76 shows the MSRAM configuration parameters set during SoC design. MSRAM0 is accessible only from the secure proxy (dst) and ring accelerator (DST port). MSRAM1 is accessible only from the ring accelerator (DST port).

Table 5-76. MSRAM Configuration Parameters

Module Instance	Parameters		
	Depth	Width	Base Address
MCU_NAVSS0_UDMASS_MSRAM0	3594	64	0x000028000000
MCU_NAVSS0_UDMASS_MSRAM1	4096	64	0x000028010000

5.5.24.3.8 MCU NAVSS Proxy Configuration

Table 5-77 shows the Proxy configuration parameters set during SoC design.

Table 5-77. Proxy Configuration Parameters

Module Instance	Parameters				
	Proxies (1)	Buffer Size (2)	Target	Channels (3)	Sizes (4)
MCU_NAVSS0_PROXY0	64	256	MCU_NAVSS0_RINGACC0	286	4096

(1) Number of proxy threads supported. Can be read off from the CONFIG read-only register
(2) Number of bytes per proxy buffer
(3) Number of channels for the target
(4) Number of bytes per channel supported for the target

Note

For Proxy functional description, see *Proxy*.

5.5.24.3.9 MCU NAVSS Secure Proxy Configuration

Table 5-78 shows the Secure Proxy configuration parameters set during SoC design.

Table 5-78. Secure Proxy Configuration Parameters

Module Instance	Parameters				
	Proxies (1)	Message Size (2)	Target	Channels (3)	Sizes (4)

Copyright © 2024 Texas Instruments Incorporated

Table 5-78. Secure Proxy Configuration Parameters (continued)

MCU_NAVSS0_SEC_PROXY0	90	64	MCU_NAVSS0_RINGACC0	30	4096

(1) Number of proxy threads supported. Can be read off from the CONFIG read-only register
(2) Number of bytes per message. Can be read off from the CONFIG read-only register
(3) Number of channels for the target
(4) Number of bytes per channel supported for the target

Note

For Secure Proxy functional description, see *Secure Proxy*.

5.5.24.4 Block Copy DMA (BCDMA)

This section contains the integration details for the BCDMA module on this device. For Further information, see the *Block Copy DMA (BCDMA)* section of the *Data Movement Architecture* chapter.

5.5.24.4.1 Features Not Supported

The following features are not supported on this family of devices:

- The BCDMA instance does not have any Block-Copy only channels
- BCDMA channels cannot be used with any generic peripherals, they are dedicated for CSI-Rx and CSI-Tx only

5.5.24.4.2 Module Allocations

Table 5-79. BCDMA Allocation Across Device Domains

Instance	WKUP Domain	MCU Domain	Main Domain
NAVSS0_BCDMA0	-	-	√ (NAVSS)

5.5.24.4.3 BCDMA Configuration

Table 5-80 shows the UDMA configuration parameters in this SoC.

Table 5-80. BCDMA Configuration Parameters

Module Instance	Parameters(1)					
	tchan_cnt(2)	rchan_cnt(3)	echan_cnt (4)	hchan_cnt(6)	uchan_cnt(7)	rflow_cnt(5)
NAVSS0_BCDMA0	0	32	16	0	0	0

(1) Parameter values can be read from the capabilities registers. See UDMA_CAP2 and UDMA_CAP3 registers.
(2) Total internal Tx channel count (Normal- + High- + Ultra-high capacities)
(3) Total internal Rx channel count (Normal- + High- + Ultra-high capacities)
(4) External UTC channel count
(5) Rx flow table entry count
(6) High-capacity + Ultra-high capacity channel count
(7) Ultra-high capacity channel count

5.5.25 PCIE

Table 5-81. PCIE Instances

Instances	WKUP Domain	MCU Domain	MAIN Domain
PCIE1			√

5.5.25.1 PCIE Unsupported Features

The PCIE module features listed below are not supported by the integration on this device:

- Address translation services (ATS)
- Quality of Service

5.5.25.2 PCIE Integration Details

See "Serializer/Deserializer (SerDes)" for PCIE connections to SerDes in this Device.

Copyright © 2024 Texas Instruments Incorporated

5.5.26 R5FSS

Instance	WKUP Domain	MCU Domain	Main Domain
MCU_R5FSS[0]		√	
R5FSS[1:0]			√

5.5.26.1 R5FSS and MCU_R5FSS Unsupported Features

The MCU_R5FSS module features listed below are not supported by the integration on this Device:
- VBUSM Peripheral Port
- VBUSM Virtual Peripheral Port

The integration of the R5FSS modules in this Device supports all module features.

5.5.26.2 MCU_R5FSS Integration Details

Requirements for Changing PULSAR Clock Frequency

MCU_R5FSS supports either 1 GHZ or 333 MHz for its core speed, and the interface clock is fixed at 333MHz. Core frequency switching between 1 GHz and 333 MHz is controlled through a CTRL_MMR bit to change the core to interface clock ratio from 3:1 to 1:1. The below software sequence shall be followed to prevent clock misalignment:
1. Place the R5 cores in WFI mode. (Alternatively, the MCU_R5FSS power domain can be powered down.)
2. Change the core to interface clock ratio by modifying the associated CTRL_MMR register.
3. Generate an interrupt to the R5 cores to release them from WFI mode (If alternate Step 1 method was used, Power up MCU_R5FSS domain, and follow the MCU bring up sequence based on split or lock-step mode.)

5.5.26.3 R5FSS Integration Details

The R5FSS has two physical and three logical low latency peripheral ports:
- VBUSM Peripheral Port - Split logically into Normal and Virtual Ports
- VBUSP Peripheral Port

The CPU requires the Virtual Peripheral Port to be mapped to a subset of (or complete) range of the Normal Peripheral Port. The Virtual Peripheral Port is disabled by default, and must be enabled by a System Control Coprocessor register write.

On this Device, the Virtual Peripheral Port is mapped to the same base address and size of the normal peripheral port. Although possible, the user should not enable the virtual peripheral port because the virtual port supports fewer (3 versus 15) outstanding writes, and enabling it will impact performance.

Interrupt outputs: FPIXCm, FPUFCm, FPOFCm, FPDZCm, FPIDCm, and FPIOCm from R5F are not connected. Consequently, the only way to check the status of the cumulative exception status flags is by reading the FPSCR register.

5.5.27 RAT

Table 5-82. RAT Clocks and Resets

Clocks	
Module Clock Input	**Description**
RAT_ICLK	RAT Interface Clock
Resets	
Module Reset Input	**Description**
RAT_RST	RAT reset

Table 5-83. RAT Hardware Requests

Interrupt Requests		
Module Interrupt Signal	**Description**	**Type**
RAT_PLS_INTR	RAT Pulse Interrupt Output	Pulse

Copyright © 2024 Texas Instruments Incorporated

Table 5-83. RAT Hardware Requests (continued)

RAT_INTR	RAT Interrupt Output	Level

5.5.27.1 RAT Integration Details

5.5.27.1.1 RAT Source IDs

Each RAT (Region Address Translation) instance is assigned a unique 16-bit RAT Source ID Value to identify the transaction source of any exception which may occur.

The following table shows the RAT source ID mapping associated with the EXCEPTION_LOGGING_HEADER0[23-8] SRC_ID field for each RAT module.

Module Instance	Source ID Value (Decimal)
MCU_ARMSS0_CPU0	2
MCU_ARMSS0_CPU1	3
R5FSS0_CORE0	4
R5FSS0_CORE1	5
R5FSS1_CORE0	6
R5FSS1_CORE1	7

5.5.28 RTI

Instance	WKUP Domain	MCU Domain	Main Domain
MCU_RTI[1:0]		√	
R5FSS[31:28,17:15,1:0]			√

5.5.28.1 RTI Unsupported Features

The RTI module features listed below are not supported by the integration on this Device:

- Analog watchdog timer
- External clock supervision
- Periodic interrupt / DMA events

Only the Digital Windowed Watchdog function of the RTI should be used.

5.5.28.2 RTI Integration Details

The Main domain instances of the RTI module are intended to function as a Digital Windowed Watchdog for the CPU core that they are associated with in Table 5-84. It is not intended to use an RTI that is provisioned for a particular CPU core with a different CPU core.

Table 5-84. RTI Integration

RTI Instance	Associated CPU
RTI0	A72SS0_CORE0
RTI1	A72SS0_CORE1
RTI2	None. Use of this RTI is Reserved.
RTI3	None. Use of this RTI is Reserved.
RTI4	None. Use of this RTI is Reserved.
RTI5	None. Use of this RTI is Reserved.
RTI6	None. Use of this RTI is Reserved.
RTI7	None. Use of this RTI is Reserved.
RTI15	GPU
RTI16	C71SS_CORE0
RTI17	C71SS_CORE1
RTI18	None. Use of this RTI is Reserved.

Copyright © 2024 Texas Instruments Incorporated

Table 5-84. RTI Integration (continued)

RTI Instance	Associated CPU
RTI19	None. Use of this RTI is Reserved.
RTI28	R5FSS0_CORE0
RTI29	R5FSS0_CORE1
RTI30	R5FSS1_CORE0
RTI31	R5FSS1_CORE1
RTI32	None. Use of this RTI is Reserved.
RTI33	None. Use of this RTI is Reserved.

The MCU_RTI Modules serve as a windowed watchdog for MCU_R5FSS0 as described in Table 5-85.

Table 5-85. MCU_RTI Module Instances

RTI Instance	Associated CPU
MCU_RTI0	MCU_R5FSS0 (Lockstep Mode) MCU_R5FSS0_CORE0 (Unlocked Mode)
MCU_RTI1	MCU_R5FSS0_CORE1 (Unlocked Mode)

5.5.29 UART

Instance	WKUP Domain	MCU Domain	Main Domain
WKUP_UART0	√		
MCU_UART0		√	
UART[9:0]			√

5.5.29.1 UART Unsupported Features

The UART module features listed below are not supported by the integration on this Device:

- Synchronous mode
- ISO7816 Mode
- DMA Mode (for WKUP and MCU Domain UART modules)
- SCR DMA Mode 2 (for Main Domain UART modules)
- Full modem handshaking (except for Main Domain UART0, which does support Full modem handshaking)

5.5.29.2 UART Integration Details

The UART module controls the function of interface pins based on the selected operating mode. The pin functions selected in each operating mode are shown in the table below.

Mode	TXD Pin Function	RTSn Pin Function	CTSn Pin Function	RXD Pin Function
UART / USART	TXD	RTSn	CTS	RXD
RS-485	TXD	RTSn (DIR)	Not used	RXD
IrDA	IRTX	SD	Not used	RXD
CIR	RCTX	SD	Not used	RXD

5.5.30 USBSS

Instance	WKUP Domain	MCU Domain	MAIN Domain
USB3SS0			√

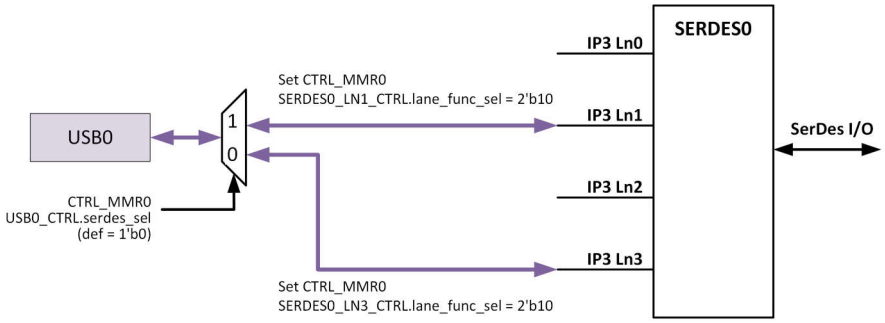
5.5.30.1 USB Unsupported Features

- The USB2 PHY implements ESD at 3.3V on the analog VBUS pin. For 5V VBUS usage, an external resistor divider.

Copyright © 2024 Texas Instruments Incorporated

5.5.30.2 USB Integration Details

The USB3 PHY interface can be connected to either SerDes0 Lane 3 or SerDes0 Lane 1 as required by the application. The CTRL_MMR0 USB0_CTRL.serdes_sel bit determines the SerDes0 lane usage. (This defaults to a value of 1'b0 which selects SerDes0 Ln3.) The appropriate SerDes lane interface (IP3) must also be correctly configured. (e.g. CTRL_MMR0 SERDES0_LN1_CTRL.lane_func_sel = 2'b10 if Lane 1 is used or CTRL_MMR0 SERDES0_LN3_CTRL.lane_func_sel = 2'b10 if Lane 3 is used.)



5.5.31 Video CODEC

Instance	WKUP Domain	MCU Domain	MAIN Domain
CODEC0			√

5.5.31.1 CODEC Unsupported Features

- UART debug interface

5.5.31.2 CODEC Integration Details

None.

5.5.32 VPAC

Instance	WKUP Domain	MCU Domain	MAIN Domain
VPAC0			√

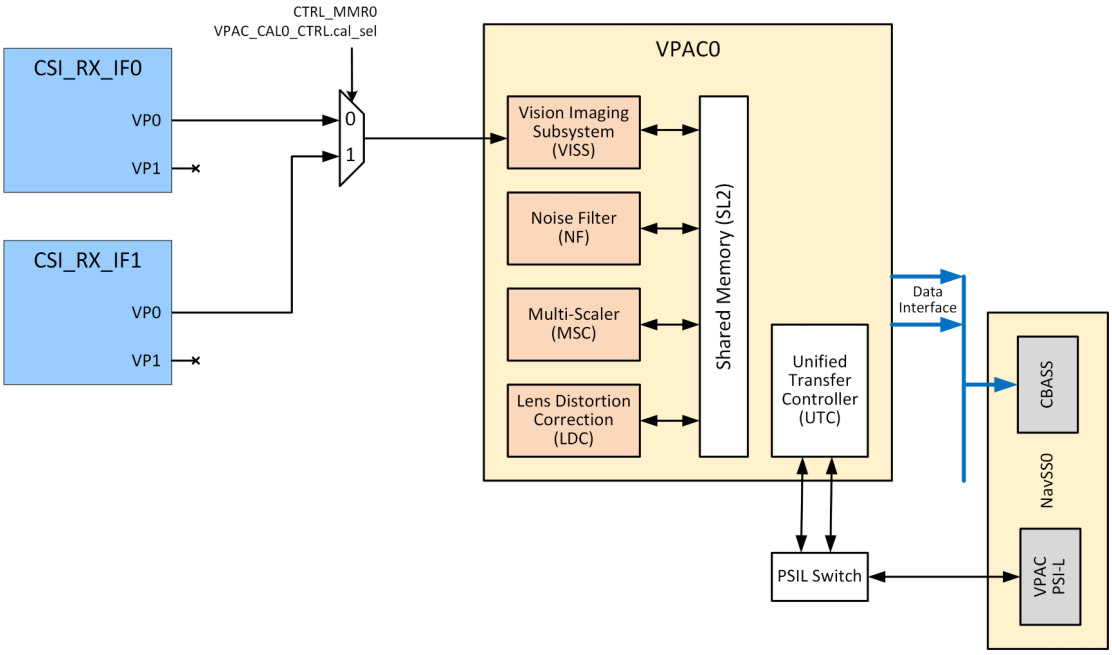
5.5.32.1 VPAC Unsupported Features

- Byte Swap of 10b 420 format

5.5.32.2 VPAC Integration Details

The VPAC0 VISS can receive camera data on its Camera Adaptive Layer (CAL) input port from either CSI_RX interface. The VPAC includes 2 Master interfaces for accessing DDR through the NavSS and includes PSIL interfaces to allow the NavSS UDMA to push data into the UTCs.

Copyright © 2024 Texas Instruments Incorporated



6 Processors and Accelerators

6.1 Compute Cluster....395
6.2 Dual-A72 MPU Subsystem....397
6.3 Dual-R5F MCU Subsystem....408
6.4 C71x DSP Subsystem....479
6.5 Graphics Accelerator (GPU)....488
6.6 Video Accelerator....489
6.7 Vision Pre-processing Accelerator (VPAC)....496
6.8 Depth and Motion Perception Accelerator (DMPAC)....677

Copyright © 2024 Texas Instruments Incorporated

6.1 Compute Cluster

This section describes the Compute Cluster (COMPUTE_CLUSTER0).

6.1.1 Compute Cluster Overview

The COMPUTE_CLUSTER encompasses the Arm® Cortex®-A72 subsystems, C71x DSP subsystems, Cluster Level Event Controller (CLEC), Generic Interrupt Controller (GIC), Multicore Shared Memory Controller (MSMC), Data Routing Units (DRUs), AXI to VBUSM.C bridges, ECC aggregators and debug components. Figure 6-1 shows an overview of the COMPUTE_CLUSTER and its surrounding modules.

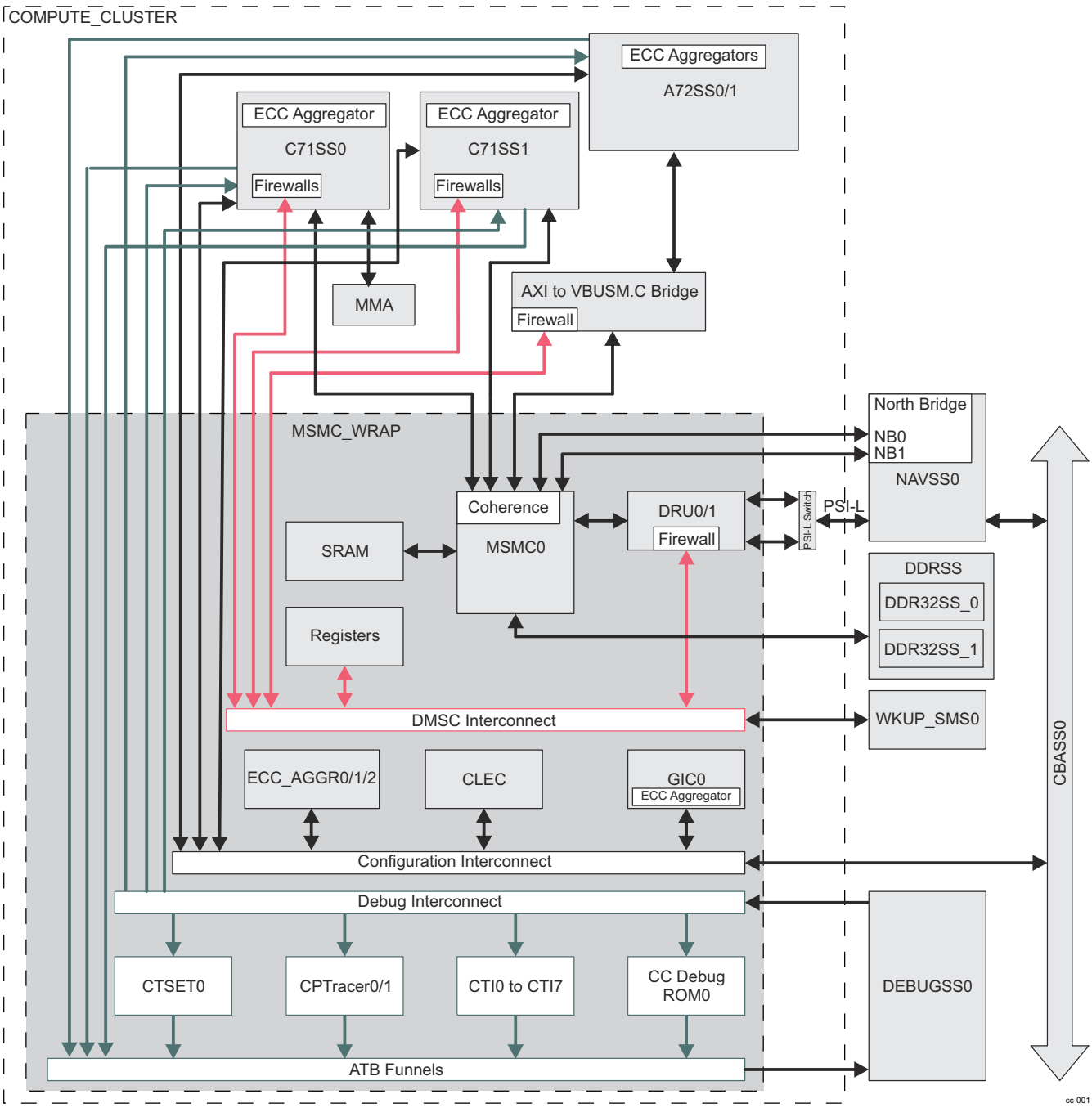


Figure 6-1. COMPUTE_CLUSTER Overview

Copyright © 2024 Texas Instruments Incorporated

Table 6-1. Compute Cluster MSMC Port Mapping

MSMC Port	DESCRIPTION	Core
COMPUTE_CLUSTER0_CPU0	MPU CLUSTER 0 (DUAL CORES)	A72SS0
COMPUTE_CLUSTER0_CPU4	C7 DSP SUBSYSTEM with MMA	C71SS0 + MMA
COMPUTE_CLUSTER0_CPU5	C7 DSP SUBSYSTEM	C71SS1

Copyright © 2024 Texas Instruments Incorporated

6.2 Dual-A72 MPU Subsystem

This section describes the Dual-A72 Microprocessor Unit (MPU) Subsystem (A72SS) in the device.

6.2.1 A72SS Overview

6.2.1.1 A72SS Introduction

The device implements one Dual-core Arm® Cortex®-A72 MPU, which is integrated inside the Compute Cluster, along with other modules. The Cortex-A72 cores are general-purpose processors that can be used for running customer applications.

The A72SS is built around the Arm Cortex-A72 MPCore (A72 cluster), which is provided by Arm and configured by TI. It is based on the symmetric multiprocessor (SMP) architecture, and thus it delivers high performance and optimal power management and debug capabilities.

The A72 processor is a multi-issue out-of-order superscalar execution engine with integrated L1 instruction and data caches, compatible with Armv8-A architecture. The Armv8-A architecture brings a number of new features. These include 64-bit data processing, extended virtual addressing and 64-bit general purpose registers.

The A72 processor features an in-order, 8-stage, dual-issue pipeline, and improved integer, Arm Neon™, Floating-Point Unit (FPU) and memory performance. It supports two execution states: AArch32 and AArch64. The AArch64 state gives the A72 CPU its ability to execute 64-bit applications, while the AArch32 state allows the processor to execute existing Armv7-A applications.

For more details on the Compute Cluster and its internal architecture, see *Compute Cluster*.

6.2.1.2 A72SS Features

The A72SS supports the following key features:

- Arm Dual-A72 Cluster (Cortex-A72 MPCore) level features:
  - A72 CPU
    - Full Armv8-A architecture compliance
      - AArch32 and AArch64 execution states
        - AArch32 for full backward compatibility with Armv7
        - AArch64 for 64b support and new architectural features
      - All exception levels EL0-3
      - A32 instruction set (previously Arm instruction set)
      - T32 instruction set (previously Thumb instruction set)
      - A64 instruction set
    - Advanced SIMD and floating point extensions (Neon)
    - Armv8 cryptography extensions
    - Superscalar, variable length, out-of-order pipeline
    - Dynamic branch prediction with Branch Target Buffer (BTB) and Global History Buffer (GHB) RAMs, return stack, and indirect predictor
  - A72 L1/L2 cache memory and MMU
    - 48-entry, fully associative, L1 instruction TLB with native support for 4KB, 64KB, and 1MB page sizes
    - 32-entry, fully associative, L1 data TLB with support for 4KB, 64KB, and 1MB page sizes
    - 4-way, set associative, unified 1K entry L2 TLB per processor
    - 48KB L1 Instruction Cache per processor with parity protection
    - 32KB L1 Data Cache per processor with ECC protection
    - 1MB Shared L2 Cache with ECC protection
  - Arm GICv3 architecture
  - Generic timers
  - Debug
    - Arm CoreSight™ architecture
    - Embedded Trace Macrocell (ETM)
    - Performance Monitor Unit (PMUv3 architecture)
- TI A72SS subsystem level features:
  - 512-bit wide, asynchronous VBUSM.C master interface

Copyright © 2024 Texas Instruments Incorporated

    - AXI2VBUSM_MASTER bridge
    - Cache pre-warming via use of ACP port
  - Timebase input interfaces
    - 64-bit graycoded global time
    - 48-bit graycoded debug time
  - 32-bit VBUSP slave interface for debug (internally converted to APB)
  - 32-bit ATB output port for debug/trace
  - Interface with Arm GIC-500 interrupt controller
  - Support for ECC on internal RAMs via ECC aggregators
- SoC level features:
  - Supports the SoC multi-core cache coherency architecture
  - Dedicated A72SS clocking (Arm PLL) for full flexibility in performance trade-offs
  - Advanced power management with fine-grained control of individual A72 CPU power domains, coarse grained cluster-level power management, and low-power standby modes (WFI/WFE modes)
  - Dedicated RTI windowed watchdog timer per core

Copyright © 2024 Texas Instruments Incorporated

6.2.2 A72SS Functional Description

6.2.2.1 A72SS Block Diagram

Figure 6-2 shows the block diagram of the A72SS subsystem.

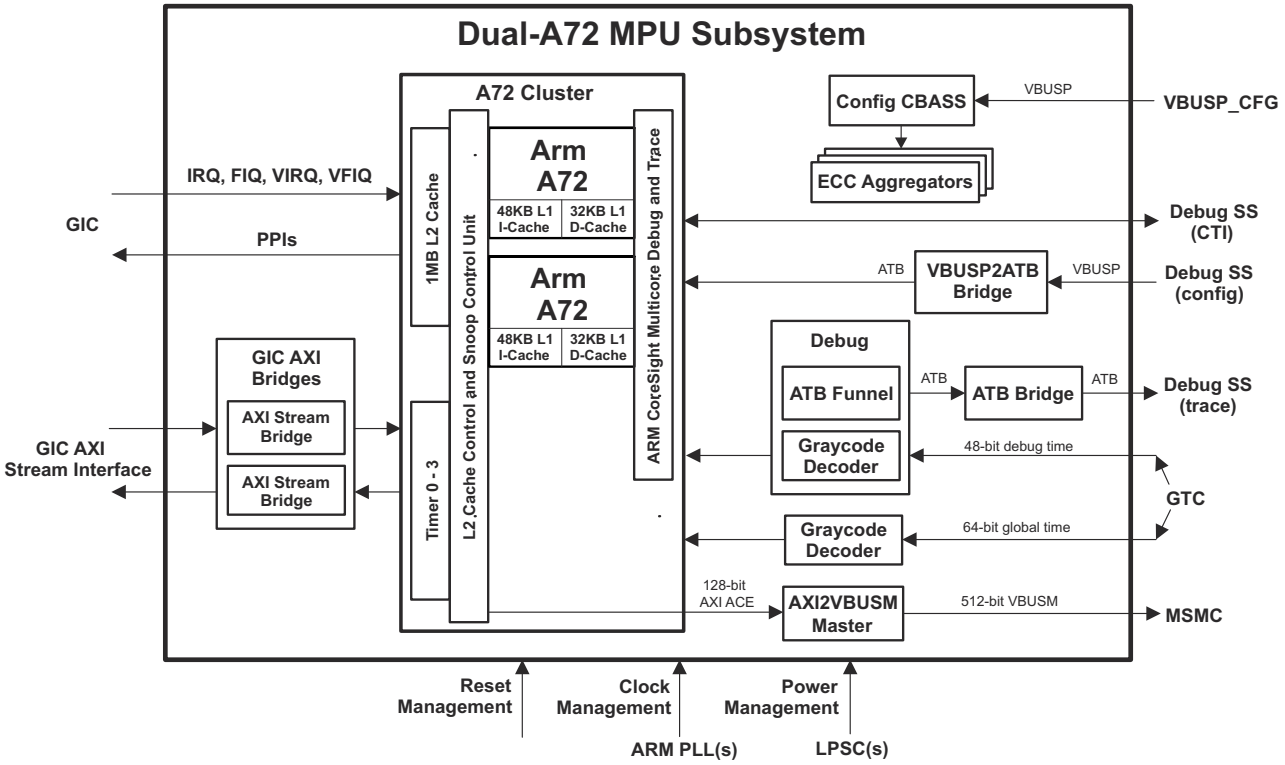


Figure 6-2. A72SS Block Diagram

6.2.2.2 A72SS A72 Cluster

The A72 cluster is provided by Arm and configured by TI. Table 6-2 summarizes the configuration of the A72 cluster for this device.

Table 6-2. A72 Cluster Configuration

Parameter	Value
Core type	A72[(1)]
Number of cores	2
Bus width	512
L1 instruction cache size	48K
L1 data cache size	32K
L2 cache size	1M
ECC protection for L2 cache	Included
ECC/parity protection for CPU cache	Included
Advanced SIMD and Floating Point Extension	Included
Cryptography extension	Included
L2 FEQ size	28

(1) The A72 core revision used in this device is r1p0.

6.2.2.3 A72SS Interfaces and Async Bridges

The A72SS has the following main interfaces:

- 512-bit wide VBUSM.C master interface

Copyright © 2024 Texas Instruments Incorporated

  - Supported by AXI2VBUSM_MASTER Bridge, which performs the following primary functions (among others):
    - Provides an asynchronous voltage domain crossing boundary for the AXI ACE master and ACP slave ports
    - Provides protocol conversion between AXI ACE (128-bit) and CBA VBUSM.C (512-bit)
    - Provides support for cache pre-warming
- 64-bit graycoded system input time
  - Graycode value provided by Global Timebase Counter (GTC)
  - Dedicated decoder (64-bit input) for graycode-to-binary conversion
- 48-bit graycoded debug input time
  - Graycode value provided by GTC
  - Dedicated decoder (48-bit input) for graycode-to-binary conversion
- 32-bit VBUSP slave interface for configuration of ECC aggregators
- 32-bit VBUSP slave interface for debug
  - Supported by VBUSP2APB Bridge, which performs VBUSP-to-APB conversion (for controlling the Arm A72 Cluster internal debug logic)
- 32-bit ATB output port for debug/trace
  - Supported by ATB Bridge, which performs clock and voltage level conversion on the combined ATB interface
  - Connected to the Debug Subsystem
- Cross Trigger Interface (CTI) for debug
  - Connected to the Debug Subsystem
- Interface(s) with Arm GIC-500 Interrupt Controller
  - Supported by GIC AXI Streaming Bridge, which performs clock and voltage level conversion on the AXI streaming protocol
  - Interrupts (PPIs) from Arm A72 Cluster to GIC-500
  - Interrupts (IRQ, FIQ, VIRQ, VFIQ) from GIC-500 to Arm A72 Cluster
- Power/clock interface(s)
  - Dedicated PLL for Arm A72 Cluster
  - Dedicated LPSC for Arm A72 Cluster, and also for each A72 core

6.2.2.4 A72SS Interrupts

6.2.2.4.1 A72SS Interrupt Inputs

The A72 CPU recieves interrupts at its inputs (IRQ, FIQ, VIRQ, VFIQ) via the dedicated Arm GIC-500 Interrupt Controller which is integrated inside the Compute Cluster and is tightly-coupled to the A72. The GIC-500 supports both A72 cores in the system. The GIC-500 is compliant to the Arm GICv3 specification and supports four types of interrupts:

- *Software Generated Interrupts (SGI)*
  - There are 16 SGIs (ID0-ID15)
  - These are inter-processor interrupts
- *Private Peripheral Interrupts (PPI)*
  - There are 16 PPIs (ID16-ID31)
  - These are wired interrupts dedicated to a specific CPU
  - Many are reserved to specific functions via convention
- *Shared Peripheral Interrupts (SPIs)*
  - There are 960 SPIs (ID32-ID991)
  - These are wired interrupts that can be routed to any core or cluster, based on the programming of that interrupt in the GIC-500
- *Locality-Specific Peripheral Interrupts (LPI)*
  - There are 57344 LPIs
  - These interrupts are used for message-based interrupts from a peripheral

The mapping of PPIs and SPIs to the GIC-500 interrupt inputs can be found in TBD.

Copyright © 2024 Texas Instruments Incorporated

Note

For a brief list of features supported by the GIC-500 module, see *Generic Interrupt Controller (GIC)*. For detailed description of the GIC-500 module, see the *Arm® CoreLink™ GIC-500 Generic Interrupt Controller Technical Reference Manual*.

6.2.2.4.2 A72SS Interrupt Outputs

Table 6-3 lists the interrupts generated by the A72SS

Table 6-3. A72SS Interrupt Outputs

TI Interrupt Name[(1)]	Arm Interrupt Name	Interrupt Description
A72 Core Interrupts (PPIs)[(2)]		
A72SS0_VCPUMNTIRQ*y*_0	nVCPUMNTIRQ	This interrupt indicates that a virtual CPU interface on the corresponding core needs serviced. This interrupt is serviced by software switching to the virtual CPU and finding what specific service needs done.
A72SS0_CNTHPIRQ*y*_0	nCNTHPIRQ	This interrupt indicates a physical timer event at the EL2 (hypervisor) exception level. Service of this interrupt is dependent on what the timer was set for. Software should take exception to EL2 and service accordingly.
A72SS0_CNTPNSIRQ*y*_0	nCNTPNSIRQ	This interrupt indicates a physical timer event at the EL1 non-secure exception level. Service of this interrupt is dependent on what the timer was set for. Software should take exception to EL1 non-secure and service accordingly.
A72SS0_CNTPSIRQ*y*_0	nCNTPSIRQ	This interrupt indicates a physical timer event at the EL1 secure exception level. Service of this interrupt is dependent on what the timer was set for. Software should take exception to EL1 secure and service accordingly.
A72SS0_CNTVIRQ*y*_0	nCNTVIRQ	This interrupt indicates a virtual timer event. Service of this interrupt is dependent on what the timer was set for. Software should take exception and service accordingly.
A72SS0_PMUIRQ*y*_0	nPMUIRQ	This interrupt is generated by the Performance Monitor Unit (PMU). The PMU can generate an interrupt based on several conditions, depending on programming. Software should take exception and query the PMU as to the cause of the interrupt, and act accordingly.
A72SS0_COMMIRQ*y*_0	nCOMMIRQ	Communications channel receive or transmit interrupt
A72SS0_CTIIRQ*y*_0	CTIIRQ	Cross Trigger Interface (CTI) interrupt
A72 Cluster Interrupts		
A72SS0_EXTERRIRQ_0	nEXTERRIRQ	This error indicates that an error has occurred on the memory bus. It is considered an external abort (or asynchronous error), because it cannot be attributed to a specific instruction. Depending on the system, software may try to recover, or reset the system.
A72SS0_INTERRIRQ_0	nINTERRIRQ	This error indicates an unrecoverable error in the cache system (L1 or L2 data RAM, L2 tag RAM, or SCU L1 duplicate tag RAM). It is considered an external abort (or asynchronous error), because it cannot be attributed to a specific instruction. Depending on the system, software can try to recover, or reset.

(1) In this column, *y* is the A72 core index (0 or 1)
(2) These are *per core* interrupts

Note

The mapping of these interrupts in the system is summarized in *A72SS Integration*, and can also be found in *Interrupts*.

For more detailed description of these interrupts and their handling, see the *Arm® Cortex®-A72 MPCore Processor Technical Reference Manual*.

6.2.2.5 A72SS Power Management, Clocking and Reset

6.2.2.5.1 A72SS Power Management

The A72 cluster and each A72 CPU reside in a separate power domain, as follows:

Copyright © 2024 Texas Instruments Incorporated

- PD14 (PD_A72_CLUSTER_0): Power domain of A72 cluster
- PD15 (PD_A72_0): Power domain of A72SS0_CORE0
- PD16 (PD_A72_1): Power domain of A72SS0_CORE1

There is a dedicated Local Power Sleep Controller (LPSC) for the A72 cluster and for each A72 core, as well. The LPSC assignment is as follows:

- LPSC78 (LPSC_A72_CLUSTER_0): Dedicated for A72 cluster
- LPSC80 (LPSC_A72_0): Dedicated for A72SS0_CORE0
- LPSC81 (LPSC_A72_1): Dedicated for A72SS0_CORE1

For more details on these LPSCs, including power-up/down sequences, see *Power*.

6.2.2.5.2 A72SS Clocking

There is a dedicated PLL for the A72 cluster: PLL8 (ARM0 PLL). For more details on this PLL, see *Clocking*.

6.2.2.6 A72SS Debug Support

The A72SS supports the standard Arm CoreSight debug architecture. Details on Arm debug can be found *On-chip Debug*, and the relevant Arm specifications.

6.2.2.7 A72SS Timestamps

The A72SS has two timebase input interfaces:

- 64-bit global timestamp: Used for synchronizing the Arm internal timers. This is done via the Arm CNTVALUEB[63:0] bus
- 48-bit debug timestamp: Can be embedded in Arm trace streams at periodic and strategic locations, allowing the temporal relationship of different trace sources to be determined

Both of them are fed by the Global Timebase Counter (GTC), which provides a 64-bit graycode value. The MPU includes two graycode decoders (for global time and debug time, respectively), which take the asynchronous graycoded times, synchronize them to the appropriate clock, and convert them to binary time, as required by Arm.

Note

Both graycode decoders are 64-bit but the one dedicated to debug time has a 48-bit input (the upper 16 bits are tied to 0).

For more details on the GTC, see *Global Timebase Counter (GTC)*.

6.2.2.8 A72SS Watchdog

The A72SS does not have an integrated watchdog timer. Instead, this feature is provided externally by the Real Time Interrupt (RTI) module, which implements a windowed watchdog timer capable of issuing warm reset to the SoC, when necessary.

There are two RTI modules in the MAIN domain that are assigned to the A72 cores:

- RTI0 used as windowed watchdog for A72SS0_CORE0
- RTI1 used as windowed watchdog for A72SS0_CORE1

For more details on the RTI windowed watchdog feature, see *Real Time Interrupt Module (RTI)*.

6.2.2.9 A72SS Internal Diagnostics

The A72SS integrates three ECC aggregators for internal diagnostics and to stimulate errors in associated RAMs for ECC testing purposes:

- A72SS0_CORE0_ECC_AGGR: ECC aggregator for A72SS0_CORE0 level
- A72SS0_CORE1_ECC_AGGR: ECC aggregator for A72SS0_CORE1 level
- A72SS0_CLUSTER_ECC_AGGR: ECC aggregator for A72_CLUSTER (L2) level

Copyright © 2024 Texas Instruments Incorporated

6.2.2.9.1 A72SS ECC Aggregators During Low Power States

When a CPU core is in WFI/WFE, the corresponding CPU ECC aggregator MMR region is not accessible and will return error status. Similarly, when L2 is in WFI, the L2 ECC aggregator MMR region is not accessible and will return error status.

6.2.2.9.2 A72SS CBASS Diagnostics

A72SS0_CORE0_ECC_AGGR integrates all the SVBUS slave endpoints for A72SS0_CORE0.

Table 6-4. A72SS0_CORE0_ECC_AGGR RAM ID Mapping for Interface Diagnostics

Endpoint	Memory ID	Controller
A72SS0_CORE0	24	EDC controller for A72SS0_CORE0_ECC_AGGR

A72SS0_CORE1_ECC_AGGR integrates all the SVBUS slave endpoints for A72SS0_CORE1.

Table 6-5. A72SS0_CORE1_ECC_AGGR RAM ID Mapping for Interface Diagnostics

Endpoint	Memory ID	Controller
A72SS0_CORE1	24	EDC controller for A72SS0_CORE1_ECC_AGGR

A72SS0_CLUSTER_ECC_AGGR integrates all the SVBUS slave endpoints for CBASS and cluster.

Table 6-6. A72SS0_CLUSTER_ECC_AGGR RAM ID Mapping for Interface Diagnostics

Endpoint	Memory ID	Controller
A72SS0_CLUSTER	32	EDC controller for A72SS0_CLUSTER_ECC_AGGR
	33-41	Internal bridges

6.2.2.9.3 A72SS SRAM Diagnostics

The A72 cluster natively supports ECC/parity protection on some of its internal SRAMs. However, error injection for the memories is not natively supported by the A72 cluster. Instead, this capability is added at A72SS level via ECC aggregator. This is a valuable TI addition to the Arm native implementation.

Table 6-7 shows the A72SS SRAM ECC support details.

Table 6-7. A72SS SRAM ECC Support

A72 RAM	ARM Native Support	TI Error Injection Support
L1 I-Cache Data RAM	Parity protection	Single error injection(1)
L1 I-Cache Tag RAM	Parity protection	Single error injection(1)
L1 I-Cache BTB RAM	No support	N/A
L1 I-Cache GHB RAM	No support	N/A
L1 I-Cache IP RAM	No support	N/A
L1 D-Cache Data RAM	ECC	Single and double error injection(1)
L1 D-Cache Tag RAM	ECC	Single and double error injection(1)
L1 PF PHT RAM	No support	N/A
L2 TLB RAM	Parity	Single error injection(1)
L2 Snoop Tag RAM	ECC	Single and double error injection(2)
L2 Tag RAM	ECC	Single and double error injection(2)
L2 Data RAM	ECC	Single and double error injection(2)
L2 Dirty RAM	ECC	Single and double error injection(2)
L2 Inclusion PLRU RAM	ECC	Single and double error injection(2)

(1) Supported by A72SS0_CORE0/1_ECC_AGGR
(2) Supported by A72SS0_CLUSTER_ECC_AGGR

Copyright © 2024 Texas Instruments Incorporated

6.2.2.9.4 A72SS SRAM ECC Aggregator Configurations

There are several schemas of memory protection employed inside of the processors. Table 6-8 and Table 6-9 describe the schema and arrangement. This describes what bits are protected, how they are protected, the behavior when an error is detected and what bits can be disturbed for ECC testing purposes. This also lists the RAM_ID associated with each memory to the corresponding ECC aggregator.

The key for the protection schemes is as follows:

- Parity: Parity bit(s) to protect data
- ECC: Error Correction Code to protect data
- SECDED: Single Error Correction, Double Error Detection

Table 6-8. A72SS0_CORE0/1_ECC_AGGR Mapping

Memory	Protection	Parity/ECC Granularity	RAM Arrangement	RAM_ID	Notes
L1 I-Cache Data	Parity	1 parity bit / 16 bits of Instruction data	[71] – Parity of all odd bits of [63:32] and [67] [70] – Parity of all even bits of [63:32] and [66] [69] – Parity of all odd bits of [31:0] and [65] [68] – Parity of all even bits of [31:0] and [64]	0-5	Parity error invalidates the offending cache line, force a fetch from the L2 cache on the next access. No aborts are generated, location of error is repored in the A72 CPUMERR register.
L1 I-Cache Tag	Parity	2 parity bits / 36 bits tag entry	[107] – Parity of all odd bits of [104:72] [71] – Parity of all odd bits of [68:36] [35] – Parity of all odd bits of [32:0] [106] – Parity of all even bits of [104:72] [70] – Parity of all even bits of [68:36] [34] – Parity of all even bits of [32:0] [105], [69], [33] – Valid bit [104:72], [68:36], [32:0] – Tag	6-7	Parity error invalidates the offending cache line, force a fetch from the L2 cache on the next access. No aborts are generated, location of error is repored in the A72 CPUMERR register.
L1 D-cache Data	ECC SECDED	32-bit word	[155:149] – ECC [148:117] – Data [116:110] – ECC [109:78] – Data [77:71] – ECC [70:39] – Data [38:32] – ECC [31:0] – Data	8-15	Single bit errors are corrected in the background and the line is removed from the cache as part of the correction process. No exception or interrupt is generated, but the A72 MERR register is updated to indicate a non-fatal error. Double bit errors are detected and an imprecise data abort is triggered and the line is evicted from the cache.
L1 D-Cache Tag	ECC SECDED	Tag for a single cache line	[39-33] – ECC for [32:2] Tag and [1:0] State	16-19	Single bit errors are corrected in the background and the line is removed from the cache as part of the correction process. No exception or interrupt is generated, but the A72 MERR register is updated to indicate a nonfatal error. Double bit errors are detected and an imprecise data abort is triggered and the line is evicted from the cache.
L2 TLB	Parity		[129] – Parity Bit [128:0] – TLB data	20-23	Error detection results in entry invalidated, new pagewalk to refetch.

Copyright © 2024 Texas Instruments Incorporated

Table 6-9. A72SS0_CLUSTER_ECC_AGGR Mapping

Memory	Protection	RAM Arrangement	RAM_ID	Notes
L2 Tag	ECC SECDED	Odd way: [77:71] – ECC for [70:41] Tag and [40:39] State Even Way: [38:32] – ECC for [31:2] Tag and [1:0] State	12-27	Accesses resulting in an L2 cache hit, where a single-bit error is detected on the data array, the L2 memory system supports in-line ECC correction. Uncorrected data is forwarded to the requesting unit, and in parallel, the ECC circuitry checks for accuracy. If a single-bit error is detected, any uncorrected data returned within two cycles before the error indicator must be discarded.
L2 Data	ECC SECDED	[143:128] – ECC [127:0] - Data	4-11	Accesses resulting in an L2 cache hit, where a single-bit error is detected on the Data array, the L2 memory system supports in-line ECC correction. Uncorrected data is forwarded to the requesting unit, and in parallel, the ECC circuitry checks for accuracy. If a single-bit error is detected, any uncorrected data returned within two cycles before the error indicator must be discarded.
L2 Snoop Tag RAM	ECC SECDED	[79:73] – ECC for [72:42] Tag and [41:40] State [39:33] – ECC for [32:2] Tag and [1:0] State	0-3	For single-bit ECC errors detected, the request is flushed from the L2 pipeline and is forced to reissue. The tag bank where the single-bit error occurred, performs a read-modify-write sequence to correct the single-bit error in the array. The request is then reissued.
L2 Inclusion PLRU RAM	ECC SECDED	[38:32] – ECC for [30:16] PLRU and [15:0] Inclusion	30-31	For single-bit ECC errors detected, the request is flushed from the L2 pipeline and is forced to reissue. The tag bank where the single-bit error occurred, performs a read-modify-write sequence to correct the single-bit error in the array. The request is then reissued.
L2 Dirty RAM	ECC SECDED	[47:44] – ECC [43:40] – Page Attribute [39:37] – ECC [36] – Dirty bit [35:32] – ECC [31:28] – Page Attribute [27:25] – ECC [24] – Dirty bit [23:20] – ECC [19:16] – Page Attribute [15:13] – ECC [12] – Dirty bit [11:8] – ECC [7:4] – Page Attribute [3:1] – ECC [0] – Dirty bit	28-29	For single-bit ECC errors detected, the request is flushed from the L2 pipeline and is forced to reissue. The tag bank where the single-bit error occurred, performs a read-modify-write sequence to correct the single-bit error in the array. The request is then reissued.

Copyright © 2024 Texas Instruments Incorporated

6.2.2.10 A72SS Cache Pre-Warming

Cache pre-warming is a method by which data/instructions that will be needed in the near future can be loaded into the L2 cache before the MPU itself does a fetch. The goal is for the cache to have the data ready before the MPU actually needs it, so that software does not need to stall waiting for cache fetches. The mechanism is the MPU's ACP port combined with the Data Routing Unit (DRU).

The basic steps are as follows:

1. Software identifies some data or instruction it knows it will need in the near future
2. Software sets up a DRU pre-warm transfer for this data
3. DRU performs reads through the ACP port of the MPU (through AXI2VBUSM_MASTER)
4. MPU initiaites reads over the memory interface that load the data into L2 cache
5. MPU returns the read data through the ACP port
   - AXI2VBUSM_MASTER discards this data (there is no need for the DRU to see it)

Now, when software tries to access this data, it will hit in the L2 cache.

For details on DRU programming to perform cache pre-warming, see *Multicore Shared Memory Controller (MSMC)*.

6.2.2.11 A72SS Boot

For Armv8 cores such as A72, two boot modes are supported: legacy 32-bit address boot mode, and 64-bit address boot mode. The selection is asserted before the A72 core is out of reset and is based on the configuration of the following Compute Cluster register:

- CC_CP_CONFIG_0[3-0] AARCH bit field

When the A72 core is configured for a 32-bit address boot mode, the boot vector is fixed at address 0x0000_0000, which corresponds to the start address of the 2KB boot RAM (PSRAM2KECC0_RAM).

When the A72 core is configured for a 64-bit address boot mode, the boot vector is determined by the RVBARADDRx tie-off signal, which is configured via associated Compute Cluster registers:

- For A72SS0_CORE0: CC_RST_VEC_LO_CP0_0 and CC_RST_VEC_HI_CP0_0
- For A72SS0_CORE1: CC_RST_VEC_LO_CP1_0 and CC_RST_VEC_HI_CP1_0

The Compute Cluster RVBARAADRx setting is latched into the A72 RVBAR register, where reset is applied. Once reset is released, the A72 RVBAR register becomes read-only and the latched value is used as a boot vector location.

Table 6-10 summarizes the A72 boot mode specifics.

Table 6-10. A72 Boot Mode Specifics

32-bit Address Boot Mode	64-bit Address Boot Mode
Boot vector is part of vector table	Boot vector is not part of vector table
Boot vector has a fixed address: 0x0000_0000. It corresponds to the 2KB boot RAM (PSRAM2KECC0_RAM)	Boot vector is determined by RVBARADDRx[39:2] tie-off setting per core.(1)
Location of the vector table is fixed by default (0x0000_0000)	No default location of vector table
No need to program A72 VBAR register if vector table stays at the default address	Each A72 core boot code must program VBAR_ELn core registers to initialize its location of the vector tables for the various Armv8 exception levels
Vector table is 32B	Vector table is 128B
Vector table location can be reprogrammed after boot	Vector table location is unknown until it is programmed after reset

(1) Although the associated Compute Cluster register provides support for a 50-bit wide reset base vector, the actual support is for a 40-bit reset base vector per SoC restriction.

6.2.2.12 A72SS IPC with Other CPUs

The A72 cores can communicate with other device cores (R5F MCU, C66x DSP, C7x DSP, etc) by supporting interrupt generation to and from these cores. The interprocessor communication (IPC) interrupts are assigned in

Copyright © 2024 Texas Instruments Incorporated

the corresponding Control Module memory-mapped registers (MMRs) called IPC_SET*x* / IPC_CLR*x*. For more information, see *Control Module (CTRL_MMR)*.

Copyright © 2024 Texas Instruments Incorporated

6.3 Dual-R5F MCU Subsystem

This chapter describes the dual-R5F microcontroller unit subsystem (R5FSS) in the device.

6.3.1 R5FSS Overview

The R5FSS is a dual-core implementation of the Arm® Cortex®-R5F processor configured for split/lock operation. It also includes accompanying memories (L1 caches and tightly-coupled memories), standard Arm CoreSight™ debug and trace architecture, integrated vectored interrupt manager (VIM), ECC aggregators, and various other modules for protocol conversion and address translation for easy integration into the SoC.

6.3.1.1 R5FSS Features

Each R5FSS supports the following features:

- Dual-core Arm Cortex-R5F
  - Core revision: r1p3
  - Armv7-R profile
  - Split/lock operation
    - Split mode: Two independently operating cores (asymmetric multi processing, no coherence)
    - Lock (lockstep) mode: One main operating core with the other operating in lockstep
    - Boot-time configurable to be in split or lock mode
  - L1 memory system
    - 32KB instruction cache
      - 4x8KB ways
      - SECDED ECC protected per 64 bits
    - 32KB data cache
      - 4x8KB ways
      - SECDED ECC protected per 32 bits
    - 64KB tightly-coupled memory (TCM) per CPU in split mode
      - SECDED ECC protected per 32 bits
      - Readable/writable from system
      - Split into A and B banks (with B further splitting into B0 and B1 interleaved banks)
        - 32KB TCMA (ATCM)
        - 16KB TCMB0 (B0TCM)
        - 16KB TCMB1 (B1TCM)
    - 128KB of TCM for CPU0 in lock mode
  - Low interrupt latency with restartable instructions
  - Non-maskable interrupt (NMI)
  - Full-precision floating point (VFPv3)
  - 16 region memory protection unit (MPU)
  - 8 breakpoints
  - 8 watchpoints
  - Dynamic branch prediction with global history buffer and 4-entry return stack
  - CoreSight debug access port (DAP)
  - CoreSight embedded trace macrocell (ETM-R5) interface
  - Performance monitoring unit (PMU)
- Interfaces
  - 64-bit VBUSM master pair (1 read, 1 write) for L3 memory accesses (per core)
  - 64-bit VBUSM slave for TCM access (per core)
    - Also allows access to cache for debug purposes
  - 32-bit VBUSM master pair (1 read, 1 write) for peripheral access
    - Supported for R5FSS0 ; not supported for MCU_R5FSS0
  - 32-bit VBUSP master for peripheral access (per core)
  - 32-bit VBUSP slave configuration port (per core)
  - 32-bit VBUSP slave debug port
    - Allows access to all R5FSS internal debug logic
- Synchronous clock domain crossing on all interfaces

Copyright © 2024 Texas Instruments Incorporated

  - Interfaces can run at an integer division of the core frequency
- Error detection logic (in lockstep mode only)
- 32-bit to 48-bit region-based address translation (RAT) on memory access masters
  - 16 regions
    - Base address + size
    - Must be size aligned
- Integrated vectored interrupt manager (VIM)
  - 512 interrupts per core
    - Only interrupts connected to R5F core 0 are available in lock mode
    - Each interrupt programmable as either IRQ or FIQ
    - Each interrupt has a programmable enable mask
    - Each interrupt has a programmable 4-bit priority
  - Priority interrupt supported
  - Vectored interrupt interface
    - Compatible with R5F VIC port
    - Programmable 32-bit vector address per interrupt
      - Address is SECDED error protected
      - Default vector addresses provided on DED
    - Split or lockstep capable
    - Software interrupt generation
- Integrated ECC aggregators
  - Support for error injection to all supported ECC memory blocks to test ECC functionality (add-on function from TI)
  - One ECC aggregator per core to cover all RAMs and caches associated with that core
- Standard Arm CoreSight debug and trace architecture at the R5FSS level
  - Cross triggering: Supported by cross trigger interface (CTI) (per CPU) and cross trigger matrix (CTM) components
  - Processor trace: Supported by embedded trace macrocell (ETM) (per CPU) and advanced trace bus (ATB) funnel components
- Boot
  - From ROM or external memory
  - From TCM

Unsupported Features:

- See the Module Integration section of this document for a list of module features not supported by the integration on this Device.

6.3.1.2 R5FSS Ports

Table 6-11. R5FSS Clocks and Resets

Module Clock Input	Description
Clocks	
R5FSS_CORE0_FCLK	R5FSS_CORE0 functional clock
R5FSS_CORE0_ICLK	R5FSS_CORE0 interface clock
R5FSS_CORE1_FCLK	R5FSS_CORE1 functional clock
R5FSS_CORE1_ICLK	R5FSS_CORE1 interface clock
Resets	
Module Reset Input	**Description**
R5FSS_CORE0_RST	R5FSS_CORE0 main reset
R5FSS_CORE0_DBG_RST	R5FSS_CORE0 debug reset (APB excluded)
R5FSS_CORE1_RST	R5FSS_CORE1 main reset
R5FSS_CORE1_DBG_RST	R5FSS_CORE1 debug reset (APB excluded)

Copyright © 2024 Texas Instruments Incorporated

Table 6-12. R5FSS Hardware Requests

Module Interrupt Signal	Description	Type
R5FSS_CORE0 Interrupts		
R5FSS_CORE0_PMU_0	R5FSS_CORE0 performance monitor interrupt	Level
R5FSS_CORE0_VALIRQ_0	R5FSS_CORE0 validation IRQ interrupt	Level
R5FSS_CORE0_VALFIQ_0	R5FSS_CORE0 validation FIQ interrupt	Level
R5FSS_CORE0_CTI_0	R5FSS_CORE0 cross trigger interrupt	Level
R5FSS_COMMON0_COMMRX_LEVEL_0_0	R5FSS_CORE0 DTRRX full interrupt	Level
R5FSS_COMMON0_COMMTX_LEVEL_0_0	R5FSS_CORE0 DTRTX empty interrupt	Level
R5FSS_CORE0_ECC_CORRECTED_LEVEL_0	R5FSS_CORE0 SEC ECC interrupt	Level
R5FSS_CORE0_ECC_UNCORRECTED_LEVEL_0	R5FSS_CORE0 DED ECC interrupt	Level
R5FSS_CORE0_EXP_INTR_0	R5FSS_CORE0 RAT exception interrupt	Level
R5FSS_COMMON0_ECC_SE_TO_ESM_0_0	R5FSS_CORE0 ECC single-bit error interrupt (cache and TCM RAMs)	Level
R5FSS_COMMON0_ECC_DE_TO_ESM_0_0	R5FSS_CORE0 ECC double-bit error interrupt (cache and TCM RAMs)	Level
R5FSS_CORE1 Interrupts		
R5FSS_CORE1_PMU_0	R5FSS_CORE1 performance monitor interrupt	Level
R5FSS_CORE1_VALIRQ_0	R5FSS_CORE1 validation IRQ interrupt	Level
R5FSS_CORE1_VALFIQ_0	R5FSS_CORE1 validation FIQ interrupt	Level
R5FSS_CORE1_CTI_0	R5FSS_CORE1 cross trigger interrupt	Level
R5FSS_COMMON0_COMMRX_LEVEL_1_0	R5FSS_CORE1 DTRRX full interrupt	Level
R5FSS_COMMON0_COMMTX_LEVEL_1_0	R5FSS_CORE1 DTRTX empty interrupt	Level
R5FSS_CORE1_ECC_CORRECTED_LEVEL_0	R5FSS_CORE1 SEC ECC interrupt	Level
R5FSS_CORE1_ECC_UNCORRECTED_LEVEL_0	R5FSS_CORE1 DED ECC interrupt	Level
R5FSS_CORE1_EXP_INTR_0	R5FSS_CORE1 RAT exception interrupt	Level
R5FSS_COMMON0_ECC_SE_TO_ESM_1_0	R5FSS_CORE1 ECC single-bit error interrupt (cache and TCM RAMs)	Level
R5FSS_COMMON0_ECC_DE_TO_ESM_1_0	R5FSS_CORE1 ECC double-bit error interrupt (cache and TCM RAMs)	Level
R5FSS_CCMR5 Interrupts		
R5FSS_COMMON0_SELFTEST_ERR_PULSE_0	R5FSS self test failure interrupt	Pulse
R5FSS_COMMON0_COMPARE_ERR_PULSE_0	R5FSS CPU bus compare failure interrupt	Pulse
R5FSS_COMMON0_BUS_MONITOR_ERR_PULSE_0	R5FSS inactivity monitor failure interrupt	Pulse
R5FSS_COMMON0_VIM_COMPARE_ERR_PULSE_0	R5FSS VIM bus compare failure interrupt	Pulse
R5FSS_CCM_COMPARE_STAT_PULSE_INTR_0	R5FSS CCMR5 in self test or split mode interrupt	Pulse
R5FSS_ESM Interrupts		
R5FSS_ECC_DE_TO_ESM_0	R5FSS_CORE0 ESM Interrupt for double bit errors	Level
R5FSS_ECC_DE_TO_ESM_1	R5FSS_CORE1 ESM Interrupt for double bit errors	Level
R5FSS_ECC_SE_TO_ESM_0	R5FSS_CORE0 ESM Interrupt for single bit errors	Level
R5FSS_ECC_SE_TO_ESM_1	R5FSS_CORE1 ESM Interrupt for single bit errors	Level

Copyright © 2024 Texas Instruments Incorporated

6.3.2 R5FSS Functional Description

6.3.2.1 R5FSS Block Diagram

Figure 6-3 shows the R5FSS block diagram.

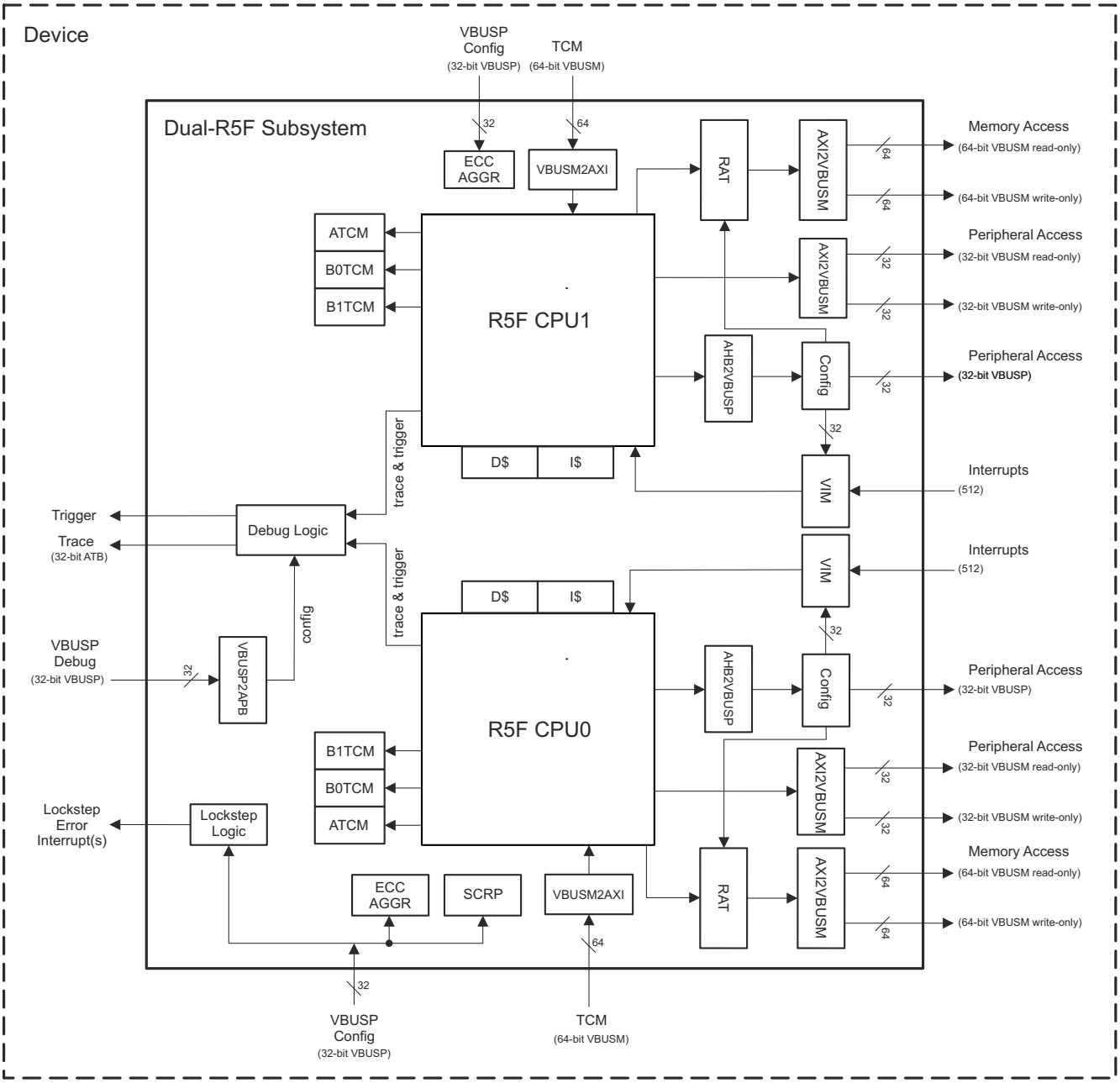


Figure 6-3. R5FSS Block Diagram

6.3.2.2 R5FSS Cortex-R5F Core

The Cortex-R5F is a processor from Arm, which is based on the Armv7-R profile. Each R5FSS implements two R5F cores, CPU0 and CPU1, each with their own RAMs and interfaces. While in reset, they can be bootstrapped to work in one of two modes: split or lockstep.

In split mode, each R5F core works completely independent from the other (asymmetric multi-processing, or AMP). Each core uses its own RAMs and interfaces, with no coherence between the two cores. The only restriction is that CPU0 must be in a higher power/reset state than CPU1. For instance, CPU1 cannot be out of reset if CPU0 is not.

Copyright © 2024 Texas Instruments Incorporated

In lock mode, the core logic from CPU1 is used as redundant logic to check for errors in CPU0. Comparison logic automatically checks the redundant logic against the primary logic and flags any errors. The CPU1 interfaces are not used in this mode. In previous generations of R5FSS, the CPU1 TCMs were unused in lock mode. In this generation of R5FSS, for more efficient use of available memories, the CPU1 TCMs are stacked on CPU0 TCMs in lock mode and are accessible only by CPU0 and CPU0 TCM interface. The TCM size for CPU0 is essentially doubled in lock mode (128KB).

For more detailed description of this processor, see the *Arm Cortex-R5 Technical Reference Manual*. A brief list of features supported by the R5F processor in this device is given in Section 6.3.1.1, *R5FSS Features*.

6.3.2.2.1 L1 Caches

The R5F has a Harvard cache architrecture, which means it has an independent L1 instruction cache (32KB) and L1 data cache (32KB). The instruction cache is protected by SECDED ECC per 64 bits. The data cache is protected by SECDED ECC per 32 bits.

6.3.2.2.2 Tightly-Coupled Memories (TCMs)

The R5F has two tightly-coupled memories (TCMs), ATCM and BTCM. The BTCM is further broken down into two interleaved banks, B0TCM and B1TCM.

TCMs are low-latency, tightly integrated memories for the R5F to use. Either TCM can be used for any combination of instruction and/or data. TCM performance is equal to performance on instructions/data that are in cache. However, TCMs have some additional advantages over cache. TCMs can be loaded with instructions that do not cache well (such as ISRs) or preloaded with code by an external source, before that code is needed, to save cache miss time. TCMs are also a good place for blocks of data for intense processing. They can be loaded (or pre-loaded by an external source) before the data is needed, saving cache miss time. The data can then be directly accessed by an external source, instead of needing to do cache evicts.

As mentioned, TCMs can be accessed (either read or written) by an external source over the TCM VBUSM slave interface. This allows instructions or data to be preloaded, or for data to be read out after the R5F has processed it. The VBUSM slave has a lower priority to accessing TCMs than the R5F but care must be taken to keep an external source from reading or writing TCM data that the R5F is working on. This handshaking is external to any of the R5FSS hardware.

TCMs are protected by ECC per 32 bits. For this to work, ECC must be enabled before data is written in to the TCMs (either externally or from the R5F). ECC is enabled via the following R5F system control bits: ACTLR.ATCMPCEN, ACTLR.B0TCMPCEN, and ACTLR.B1TCMPCEN, respectively.

Whether or not the TCMs are enabled is controlled by the ENABLE bit in the corresponding ATCM/BTCM region register. The default (reset) value of this bit is determined by the CPUn_INITRAMA and CPUn_INITRAMB bootstraps, respectively. Both ATCM and BTCM are configured for a size of 32KB in this device. Note that the BTCM size is the total of both B0TCM and B1TCM (16KB each).

If a TCM is not enabled, then it does not appear in the R5F memory view, but it can be accessed by an external source. If a TCM is enabled, then its place in the R5F memory map is determined by a combination of bootstrap signal and system register. If the CPUn_LOCZRAMA bootstrap signal is high, then the initial base address of ATCM is 0x0000_0000 and the initial address of BTCM is 20’h41010. If the CPUn_LOCZRAMA bootstrap signal is low, then the initial base address of BTCM is 0x0000_0000 and the initial base address of ATCM is 20’h41010.

Note

This base address of 0x41010 for ATCM/BTCM based on the CPUn_LOCZRAMA bootstrap only affects the R5F’s memory view. The SoC will see the ATCM/BTCM based on the TCM slave interface regions, as defined in Section 6.3.2.3.2. The base address of either TCM may be overwritten via the ATCM or BTCM region register. Care must be taken not to move the base address of a TCM when it may be being accessed.

It is possible to preload a TCM with instructions and boot from it. For more details, see Section 6.3.2.11, *R5FSS Boot Options*.

Copyright © 2024 Texas Instruments Incorporated

6.3.2.2.3 R5FSS Special Signals

The following tables list some R5FSS features associated with special signals. Note that in lockstep mode only those for CPU0 apply.

Table 6-13. MCU_R5FSS0 Special Features

Feature	Comment
Cluster affinity group ID	R5F cluster 0 (ID = 0x0)
Exception handling state at reset 0 = Arm 1 = Thumb	Controlled via MCU_SEC_MMR register setting. Defaults to Arm mode
Split or lockstep mode 0 = Split mode 1 = Lockstep mode	Controlled via MCU_SEC_MMR register setting. Defaults to a value defined by eFuse
CPUn execution halt when coming out of reset (CPUn_HALT)	Controlled via MCU_SEC_MMR register setting. Defaults to halted state
CPUn exception vectors base address	Controlled via MCU_SEC_MMR register setting. Defaults to ROM address 0x0000_41810_0000
CPUn VIM base address	0x40F8_0000
CPUn RAT base address	0x40F9_0000
CPUn RAT accesses ID	0x2 (CPU0); 0x3 (CPU1)
CPUn ATCM enable at reset (CPUn_INITRAMA)	Controlled via MCU_SEC_MMR register setting. Defaults to disabled state
CPUn BTCM enable at reset (CPUn_INITRAMB)	Controlled via MCU_SEC_MMR register setting. Defaults to enabled state
CPUn A/BTCM reset base address indicator (CPUn_LOCZRAMA) 0 = B at 0x0 1 = A at 0x0	Controlled via MCU_SEC_MMR register setting. Defaults to 1
CPUn non-maskable fast interrupts enable	Controlled via MCU_SEC_MMR register setting. Defaults to disabled state
CPUn VBUSM peripheral port enabled at reset	VBUSM port not used
CPUn VBUSP peripheral port enable at reset	Enabled
CPUn VBUSP peripheral port base address	Mapped to 0x0_4000_0000 MCU peripheral base address
CPUn VBUSP peripheral port size	16MB for MCUSS peripherals (0x0_4000_0000 to 0x0_40FF_FFFF)
CPUn VBUSM normal peripheral port base address	VBUSM port not used
CPUn VBUSM normal peripheral port size	VBUSM port not used
CPUn VBUSM virtual peripheral port base address	VBUSM port not used
CPUn VBUSM virtual peripheral port size	VBUSM port not used
CPUn clock stopped indication	Status logged into MCU_SEC_MMR register bit
CPUn WFI state	Status logged into MCU_SEC_MMR register bit
CPUn WFE state	Status logged into MCU_SEC_MMR register bit
CPU clockstop behavior 0: CPU clocks stopped in standby 1: CPU clocks not stopped in standby	Controlled via MCU_SEC_MMR register setting. Defaults to 0

Table 6-14. R5FSS0 Special Features

Feature	Comment
Cluster affinity group ID	R5F Cluster 1 (ID = 0x1)
Exception handling state at reset 0 = Arm 1 = Thumb	Controlled via MAIN_SEC_MMR register setting. Defaults to Arm mode
Split or lockstep mode 0 = Split mode 1 = Lockstep mode	Controlled via MAIN_SEC_MMR register setting. Defaults to a value defined by eFuse
CPUn execution halt when coming out of reset (CPUn_HALT)	Controlled via MAIN_SEC_MMR register setting. Defaults to halted state

Copyright © 2024 Texas Instruments Incorporated

Table 6-14. R5FSS0 Special Features (continued)

Feature	Comment
CPUn exception vectors base address	Controlled via MAIN_SEC_MMR register setting. Defaults to Bootvector RAM address 0x0000_0000_0200
CPUn VIM base address	0x0FF8_0000
CPUn RAT base address	0x0FF9_0000
CPUn RAT accesses ID	0x4 (CPU0); 0x5 (CPU1)
CPUn ATCM enable at reset (CPUn_INITRAMA)	Controlled via MAIN_SEC_MMR register setting. Defaults to disabled state
CPUn BTCM enable at reset (CPUn_INITRAMB)	Controlled via MAIN_SEC_MMR register setting. Defaults to enabled state
CPUn A/BTCM reset base address indicator (CPUn_LOCZRAMA) 0 = B at 0x0 1 = A at 0x0	Controlled via MAIN_SEC_MMR register setting. Defaults to 1
CPUn non-maskable fast interrupts enable	Controlled via MAIN_SEC_MMR register setting. Defaults to disabled state
CPUn VBUSM peripheral port enabled at reset	Enabled
CPUn VBUSP peripheral port enable at reset	Enabled
CPUn VBUSP peripheral port base address	Mapped to 0x0_0F80_0000 for low latency MAIN peripherals
CPUn VBUSP peripheral port size	8MB for MAIN peripherals (0x0_0F80_0000 to 0x0_0FFF_FFFF)
CPUn VBUSM normal peripheral port base address	Mapped to 0x0200_0000 for MAIN peripherals
CPUn VBUSM normal peripheral port size	32MB for MAIN peripherals (0x0_0200_0000 to 0x0_03FF_FFFF)
CPUn VBUSM virtual peripheral port base address	Mapped to 0x0200_0000 for MAIN peripherals
CPUn VBUSM virtual peripheral port size	32MB for MAIN peripherals (0x0_0200_0000 to 0x0_03FF_FFFF)
CPUn clock stopped indication	Status logged into MAIN_SEC_MMR register bit
CPUn WFI state	Status logged into MAIN_SEC_MMR register bit
CPUn WFE state	Status logged into MAIN_SEC_MMR register bit
CPU clockstop behavior 0: CPU clocks stopped in standby 1: CPU clocks not stopped in standby	Controlled via MAIN_SEC_MMR register setting. Defaults to 0

6.3.2.3 R5FSS Interfaces

6.3.2.3.1 R5FSS Master Interfaces

The R5FSS has several master interfaces per core:

- 64-bit VBUSM master pair (1 read – “RMST”, 1 write – “WMST”) for L3 memory accesses; this is the main memory interface
  - Includes region-based address translation (RAT)
- 32-bit VBUSM master pair (1 read – “PRMST”, 1 write – “PWMST”) for peripheral access
  - Enabled at reset
  - Supported for R5FSS0 ; not supported for MCU_R5FSS0
- 32-bit VBUSP master “PMST” for peripheral access
  - Includes logic that provides the R5F CPU with a private access to VIM and RAT
  - Enabled at reset

6.3.2.3.2 R5FSS Slave Interfaces

The R5FSS has several slave interfaces that define its internal memory space:

- 32-bit VBUSP configuration slave (per core)
  - Region [0]: ECC aggregator block
  - Region [1]: Lockstep comparator block (CCMR5); this region is only applicable to CORE0
- 64-bit TCM slave (per core)
  - Region [0]: ATCM
  - Region [1]: BTCM

Copyright © 2024 Texas Instruments Incorporated

  - Region [2]: Instruction cache RAMs
  - Region [3]: Data cache RAMs
- 32-bit VBUSP debug slave
  - Provides access to all R5FSS internal debug logic

Regions [0] and [1] of the TCM slave interface provide direct access to the TCM RAMs. Access to the RAMs is arbitrated with access from the R5F's L1 memory system. Excessive access while the R5F is also attempting access will degrade performance.

Regions [2] and [3] of the TCM slave interface provide access to the cache RAMs for testing purposes. Access to the cache RAMs can only be done while the caches are disabled and should only be done for test purposes.

In addition to the slave interfaces, there are peripherals (RAT and VIM) that are only accessible by the R5F. The R5F has an access to these modules via the VBUSP peripheral interface.

6.3.2.4 R5FSS Power, Clocking and Reset

6.3.2.4.1 R5FSS Power

The following R5FSS power considerations should be noted:

- R5FSS has a single power domain for all its internal logic
- When operating in split mode, CPU0 must be in a higher power/reset state than CPU1.

For more details on R5FSS power management, including power-up and power-down sequences, see *Power*.

6.3.2.4.2 R5FSS Clocking

The R5FSS has four clock inputs:

- CPU0_CLK: This is the clock for CPU0 logic
- CPU1_CLK: This is the clock for CPU1 logic
- CPU0_ICLK: This is the clock for CPU0 interfaces
- CPU1_ICLK: This is the clock for CPU1 interfaces

CPU0_CLK and CPU1_CLK are the clocks for all of the internal CPU logic. They are provided separately so that:

- In split mode, CPU1's clock may be turned off, while CPU0 is still in active mode
- In lock mode, they are sourced separately

CPU0_ICLK and CPU1_ICLK are the clocks for all of the interfaces for their associated CPU (for example: VBUSM and VBUSP bridges, exception generation, debug and trace logic). They are provided separately so that CPU1_ICLK can be gated if CPU1 is in a lower power state, while CPU0 is ON, or when in lock mode.

The interface clock is an integer ratio of the CPU clock. See *R5FSS Integration*.

6.3.2.4.2.1 Changing MCU_R5FSS0 CPU Clock Frequency

For each MCU_R5FSS0 CPU, the interface clock is an integer ratio of the CPU0 clock. The exact core to interface clock ratio is configured via associated Control Module register:

- CTRLMMR_MCU_R5_CORE0_CLKSEL for both MCU_R5FSS0 cores

The interface clock has a fixed frequency, so this register is essentially used to select the CPU clock frequency. Due to some design limitation, the core frequency switch for the MCU_R5FSS0 may cause clock misalignment. To avoid this this clock misalignment, the below software sequence should be implemented by a core other than MCU_R5FSS0_CORE0 or MCU_R5FSS0_CORE1 when changing the CPU clock frequency:

1. Power down MCU_R5FSS0
2. Change the core to interface ratio by modifying the associated CTRL_MMR register
3. Power up MCU_R5FSS0 and follow the bring up sequence based on split or lockstep mode

Note that the sequence above is a static configuration change and any context in the MCU_R5FSS0 will be not be preserved. There is no dynamic clock reconfiguration available.

6.3.2.4.3 R5FSS Reset

The R5FSS has four reset inputs:

Copyright © 2024 Texas Instruments Incorporated

- CPU0_RST: This is the reset for the non-debug logic of CPU0
- CPU0_DBG_RST: This resets the CPU0 debug logic, excluding the APB interface
- CPU1_RST: This is the reset for the non-debug logic of CPU1
- CPU1_DBG_RST: This resets the CPU1 debug logic, excluding the APB interface

In addition to the reset signals, there are two halt signals:

- CPU0_HALT
- CPU1_HALT

These halt signals keep the CPUs from fetching instructions when they come out of reset. The main use is to have the CPUs halted until the TCMs are loaded (when booting from TCM), though halt could be used for any other purpose.

6.3.2.5 R5FSS Lockstep Error Detection Logic

When bootstrapped to lockstep mode, there is a logic that compares the outputs of the two cores. Whenever an error is detected, an interrupt is asserted. A logic is also included to test the comparison logic.

In lockstep mode, the logic from CPU1 is used to check CPU0. CPU1's RAMs and L1 memory system are not used. Instead, all inputs to CPU0 are copied, delayed by two cycles, and fed into CPU1's logic. All outputs from CPU0 are copied, delayed by two cycles, and compared to the outputs of CPU1.

The lockstep error detection logic in the R5FSS is implemented via the CPU compare module (CCMR5). The CCMR5 performs two main functions:

- CPU output compare: Compares the core compare bus outputs of the two R5F processor cores in lockstep mode
- Inactivity monitoring: This is to detect any transaction initiated by the diagnostic R5F core (CPU1) in lockstep mode. The diagnostic core is expected *not* to initiate any transaction in lockstep mode

6.3.2.5.1 CPU Output Compare Block

To implement a lockstep CPU cluster, a diagnostic checker block is included. The diagnostic checker block compares the CPU compare outputs on every cycle. A difference in the CPU output signals is indicated by an error signal.

As the CPUs are working with a two-cycle phase difference, there should be no compare errors generated when the CPUs are reset two cycles apart. For the various CPU interfaces, the comparison will start on the first valid transaction on key control signals.

Note

While R5 is in lockstep and CCM is active, debug using external debugger is not supported.

6.3.2.5.1.1 Operating Modes

The CPU compare block can run in one out of four operating modes:

- Compare active block
- Self test
- Error forcing
- Self test error forcing

To select an operating mode for the CPU compare block, a dedicated key must be written to the key register R5FSS_CCMKEYR1. R5FSS_CCMSR1 is the status register associated with this block.

Table 6-15 provides details on the operating modes of the CPU compare block.

Table 6-15. CPU Compare Block Operating Modes

Mode	Key(1)	Self Test Error Signal	Compare Error Signal	CMPE1(2)	STC1(3)	STET1(4)	STE1(5)
Compare block active	0000	Enabled	Enabled	Enabled	Disabled	Disabled	Disabled
Self test	0110	Enabled	Disabled	Disabled	Enabled	Enabled	Enabled
Error forcing	1001	Error	Error	Disabled	Disabled	Disabled	Disabled

Copyright © 2024 Texas Instruments Incorporated

Table 6-15. CPU Compare Block Operating Modes (continued)

Mode	Key(1)	Self Test Error Signal	Compare Error Signal	CMPE1(2)	STC1(3)	STET1(4)	STE1(5)
Self test error forcing	1111	Error	Enabled	Enabled	Disabled	Disabled	Disabled

(1) R5FSS_CCMKEYR1[3:0] MKEY1 bit field
(2) R5FSS_CCMSR1[16] CMPE1: Compare error flag
(3) R5FSS_CCMSR1[8] STC1: Self test complete flag
(4) R5FSS_CCMSR1[1] STET1: Self test error type flag
(5) R5FSS_CCMSR1[0] STE1: Self test error flag

6.3.2.5.1.2 Compare Block Active Mode

In compare block active mode, the output signals of both CPUs are compared, and any difference in the outputs is indicated by the compare error signal. Additionally, as indicated in Table 6-15, the self test error signal is also asserted.

Note

The self test error signal is shared by both the CCMR5's CPU compare and inactivity monitor blocks.

Note

Not all flops inside the R5F CPU(s) are initialized at reset. To avoid an erroneous CCMR5 compare error, the application software needs to ensure that the CPU registers of both CPUs are initialized with the same values before the registers are used. This is to ensure that the programmer's model CPU registers of both CPU are initialized properly.

6.3.2.5.1.3 Self Test Mode

In self test mode, the CCMR5 compare block is checked for faults by applying internally generated series of test patterns. During self test, the core compare disabled signal is deactivated. If a fault on the CCMR5 module gets detected, a self test error is generated.

When self test mode is entered, the CCMR5 module will generate predetermined test patterns to look for any hardware faults inside it. If a fault is detected, then a self test error flag (STE1) is set, a self test error signal is asserted, and the self test is terminated immediately after entering compare mismatch mode. If no fault is found during self test, the self test complete flag (STC1) will be set. The user needs to poll the R5FSS_CCMSR1 status register to find out the self test status. In both cases – self test terminated and self test completed – the CCMR5 will remain in self test mode and will be idle, and therefore the R5FSS_CCMKEYR1 key register will show the self test key until the mode is switched by writing another key to this register. During the self test operation, the compare error signal output is inactive, irrespective of the compare result. When switched out of self test mode, the core compare disabled signal is de-asserted.

There are two types of patterns generated by CCMR5 during self test mode: compare match test, and compare mismatch test. The CCMR5 first generates compare match test patterns followed by compare mismatch test patterns. During self test, each test pattern is applied on CPU output signal ports of the CCMR5's compare block and clocked for one cycle.

Self test of CPU output compare logic is done with respect to CPU clock. As mentioned, self test error is indicated by the self test error signal. Whether the self test failed during compare match test or compare mismatch test is indicated by the self test error type flag (STET1) in the R5FSS_CCMSR1 status register. When the block's self test is completed, the corresponding self test complete flag (STC1) is set.

Note

During self test, both CPUs can execute normally, but the compare logic will not be checking any CPU signals. Also, during self test, only the compare unit logic is tested. The self test is not interruptible.

Copyright © 2024 Texas Instruments Incorporated

6.3.2.5.1.4 Compare Match Test

During compare match test, there are four different test patterns generated to stimulate the hardware. An identical vector is applied to both input ports (CPU0 and CPU1 output signals, which are inputs to the CCMR5 block) at the same time expecting a compare match. These patterns cause the self test logic to exercise every CPU output signal. This test can be independently enabled by programming the CCMR5 registers. If the compare unit produces a compare mismatch then the self test error flag is set and the self test error signal is generated. If an error is detected during compare match, then CCMR5 enters compare mismatch mode before terminating the test.

The four test patterns are:

- All 0's on both CPUs
- All 1's on both CPUs
- 0xA's on both CPUs
- 0x5's on both CPUs

These four test patterns will take four clock cycles to complete.

Table 6-16 illustrates the compare match test sequence. The core compare disabled signal is asserted on entry and de-asserted when complete.

Table 6-16. Compare Match Test Sequence

CPU0 Signal Position									CPU1 Signal Position									Cycle
n:8	7	6	5	4	3	2	1	0	n:8	7	6	5	4	3	2	1	0	
0x5	0	1	0	1	0	1	0	1	0x5	0	1	0	1	0	1	0	1	0
0xA	1	0	1	0	1	0	1	0	0xA	1	0	1	0	1	0	1	0	1
0's	0	0	0	0	0	0	0	0	0's	0	0	0	0	0	0	0	0	2
1's	1	1	1	1	1	1	1	1	1's	1	1	1	1	1	1	1	1	3

6.3.2.5.1.5 Compare Mismatch Test

In compare mismatch test, the number of test patterns is equal to two times the number of CPU output signals to compare in compare mode active. The core compare disabled signal is asserted on entry and de-asserted when complete. An 'all-ones' vector is applied to the CCMR5's CPU0 input port and the same pattern is also applied to the CCMR5's CPU1 input port but with one bit flipped starting from signal position 0. The un-equal vector should cause the CCMR5 module to expect a compare mismatch at signal position 0. Note that a mismatch is an expected good result, while a match will indicate hardware fault. When a fault is detected, which means that a compare match is produced by the compare logic, the self test error flag is set and the self test error signal is asserted.

The above compare mismatch test algorithm repeats itself again in a domino fashion with next signal position flipped, while forcing all other signals to logic level '1'. This process is repeated until every single signal position is verified on both CPU signal ports.

The compare mismatch test is terminated if a compare match is detected and the module becomes idle. The compare mismatch test ensures that the compare unit is able to detect a mismatch on every CPU signal being compared.

Table 6-17 illustrates the compare mismatch test sequence.

Table 6-17. Compare Mismatch Test Sequence

CPU0 Signal Position										CPU1 Signal Position										Cycle
n	...	7	6	5	4	3	2	1	0	n	...	7	6	5	4	3	2	1	0	
1	1's	1	1	1	1	1	1	1	1	1	1's	1	1	1	1	1	1	1	0	0
1	1's	1	1	1	1	1	1	1	1	1	1's	1	1	1	1	1	1	0	1	1
1	1's	1	1	1	1	1	1	1	1	1	1's	1	1	1	1	1	0	1	1	2
1	1's	1	1	1	1	1	1	1	1	1	1's	1	1	1	1	0	1	1	1	3
. .																				

Copyright © 2024 Texas Instruments Incorporated

Table 6-17. Compare Mismatch Test Sequence (continued)

CPU0 Signal Position										CPU1 Signal Position										Cycle
n	...	7	6	5	4	3	2	1	0	n	...	7	6	5	4	3	2	1	0	
1	1's	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	n-1
1	1's	1	1	1	1	1	1	1	1	0	1's	1	1	1	1	1	1	1	1	n
1	1's	1	1	1	1	1	1	1	0	1	1's	1	1	1	1	1	1	1	1	n+1
1	1's	1	1	1	1	1	1	0	1	1	1's	1	1	1	1	1	1	1	1	n+2
1	1's	1	1	1	1	1	0	1	1	1	1's	1	1	1	1	1	1	1	1	n+3
1	1's	1	1	1	1	0	1	1	1	1	1's	1	1	1	1	1	1	1	1	n+4
. .																				
0	1's	1	1	1	1	1	1	1	1	1	1's	1	1	1	1	1	1	1	1	2n

6.3.2.5.1.6 Error Forcing Mode

In error forcing mode, a test pattern is applied to the CPU related inputs of the compare logic to force an error at the compare error signal of the compare unit. The core compare disabled signal is asserted on entry and de-asserted when complete.

This mode is enabled by writing the dedicated key in the R5FSS_CCMKEYR1 key register (see Table 6-15). The error signal is generated by asserting the CPU compare error signal.

Error forcing mode is similar to the compare mismatch test operation of self test mode, in which an un-equal vector is injected into the CCMR5 CPU signal ports. Instead of setting a self test error flag and asserting self test error signal, the error forcing mode forces the compare mismatch to set the compare error flag (CMPE1) and assert the compare error signal.

Only one hardcoded test pattern is applied into CCMR5 during error forcing mode. A repeated 0x5 pattern is applied to CPU0 signal port, while a repeated 0xA pattern is applied to the CPU1 signal port. The error forcing mode takes one cycle to complete. Hence, the failing signature is presented for one clock cycle and the mode is automatically switched to lockstep mode and the R5FSS_CCMKEYR1 key register will show the lockstep key (0000). During this cycle the CPUs are not compared. The user should expect to receive the error signal from CCMR5 module once the error forcing mode is entered. If no error signal is set, then a hardware fault is present.

6.3.2.5.1.7 Self Test Error Forcing Mode

In self test error forcing mode, an error is forced at the self test error signal. The compare unit is still running in lockstep mode and the key is switched to lockstep after one clock cycle. The core compare disabled signal is asserted on entry and de-asserted when complete.

6.3.2.5.2 Inactivity Monitor Block

Another function of the CCMR5 module is to monitor the inputs of bus matrix coming from the diagnostic R5F core (CPU1) in lockstep mode, to detect any transaction initiated by diagnostic core. Output signals from the diagnostic core, which indicate a valid transaction on a master interface, are compared against their clamped values. If any signal value is different from its clamped value, an error signal is generated.

Table 6-18 shows the R5F diagnostic core output signals, which are being monitored to detect any activity in the processor bus.

Table 6-18. Inactivity Monitor Block Signals

Signal Name	Interface	Description	Clamp Value
AWVALIDM1	L2 AXI Master	Indicates write address and control are valid	0
ARVALIDM1	L2 AXI Master	Indicates read address and control are valid	0
AWVALIDP1	AXI PP[(1)]	Indicates write address and control are valid	0
ARVALIDP1	AXI PP[(1)]	Indicates read address and control are valid	0
HTRANSP1[1:0]	AHB PP	Indicates the type of transfer – idle (00), busy (01), non-sequential (10), sequential (11)	0
BVALIDS1	AXI Slave	Indicates valid write response is available	0

Copyright © 2024 Texas Instruments Incorporated

Table 6-18. Inactivity Monitor Block Signals (continued)

Signal Name	Interface	Description	Clamp Value
RVALIDS1	AXI Slave	Indicates Read Data Channel Address and Control are valid	0
ATCEN01	ATCM	Enable for ATCM lower word	0
ATCEN11	ATCM	Enable for ATCM upper word	0
B0TCEN01	B0TCM	Enable for B0TCM lower word	0
B0TCEN11	B0TCM	Enable for B0TCM upper word	0
B1TCEN01	B1TCM	Enable for B1TCM lower word	0
B1TCEN11	B1TCM	Enable for B1TCM upper word	0

(1) Not supported for MCU_R5FSS0

More details on these signals can be found in the *Arm Cortex-R5 Technical Reference Manual*.

The error response in case of a detected transaction is indicated by the bus monitor error signal.

6.3.2.5.2.1 Operating Modes

The inactivity monitor block can run in one out of four operating modes:

- Compare active block
- Self test
- Error forcing
- Self test error forcing

To select an operating mode for the inactivity monitor block, a dedicated key must be written to the key register R5FSS_CCMKEYR3. R5FSS_CCMSR3 is the status register associated with this block.

Table 6-19 provides details on the operating modes of the inactivity monitor block.

Table 6-19. Inactivity Monitor Block Operating Modes

Mode	Key(1)	Self Test Error Signal	Compare Error Signal(2)	CMPE3(3)	STC3(4)	STET3(5)	STE3(6)
Compare block active	0000	Enabled	Enabled	Enabled	Disabled	Disabled	Disabled
Self test	0110	Enabled	Disabled	Disabled	Enabled	Enabled	Enabled
Error forcing	1001	Error	Error	Disabled	Disabled	Disabled	Disabled
Self test error forcing	1111	Error	Enabled	Enabled	Disabled	Disabled	Disabled

(1) R5FSS_CCMKEYR3[3:0] MKEY3 bit field
(2) Corresponds to bus monitor error signal for inactivity monitoring
(3) R5FSS_CCMSR3[16] CMPE3: Compare error (bus monitor error) flag
(4) R5FSS_CCMSR3[8] STC3: Self test complete flag
(5) R5FSS_CCMSR3[1] STET3: Self test error type flag
(6) R5FSS_CCMSR3[0] STE3: Self test error flag

6.3.2.5.2.2 Compare Block Active Mode

In compare block active mode, the output signals of CPU1 (after clamping) are compared against their clamped values, and a mismatch is indicated by the bus monitor error signal. Additionally, as indicated in Table 6-19, the self test error signal is also asserted.

Note

The self test error signal is shared by both the CCMR5's CPU compare and inactivity monitor blocks.

6.3.2.5.2.3 Self Test Mode

Inactivity monitor has a self test feature and self test is done by applying a predetermined set of internally generated test patterns on the Inactivity Monitor inputs. For the fault detection, the comparison is done against the clamped values. The generated patterns are same as output compare block test patterns, but they are applied on every HCLK cycles. Any fault detected is indicated by the self test error signal. If no fault is found during self test, the self test complete flag (STC3) will be set. The user needs to poll the R5FSS_CCMSR3 status register to find out the self test status. In both cases – self test terminated and self test completed

Copyright © 2024 Texas Instruments Incorporated

– inactivity monitor will remain in self test mode and will be idle, and therefore the R5FSS_CCMKEYR3 key register will show the self test key until the mode is switched by writing another key to this register.

Note

Bus monitor error is disabled during self test mode.

Two types of test patterns are applied for inactivity monitor self test:

- Compare match pattern
- Compare mismatch pattern

Self test takes 16 cycles to complete. Self test error is indicated by the self test error signal. Whether the self test failed during compare match test or compare mismatch test is indicated by the self test error type flag (STET3) in the R5FSS_CCMSR3 status register. When the block's self test is completed, the corresponding self test complete flag (STC3) is set.

6.3.2.5.2.4 Compare Match Test

In compare match test, patterns are applied such that no compare error is generated. Because the comparison is done against the clamped values, and all compared signals are clamped to zero, only one test pattern is applicable for compare match test. A pattern of 'all-zeros' is applied for compare match test. If a compare mismatch is found, then self test error flag (STE3) is set and self test error is asserted. If an error is detected during compare match, then CCMR5 enters compare mismatch mode before terminating the test. If no mismatch is found, compare mismatch test is done next.

6.3.2.5.2.5 Compare Mismatch Test

In compare mismatch test, a series of un-equal test patterns are applied on inactivity monitor inputs. Un-equal vectors are expected to generate a compare mismatch and a compare match indicates a fault. If a compare match is found, self test error flag is set and a self test error is indicated. Self test is terminated as well. If all patterns are generating mismatch, self test complete flag (STC3) is set.

Table 6-20 shows the patterns used for compare mismatch test.

Table 6-20. Inactivity Monitor Compare Mismatch Patterns

Inactivity Monitor Compare Mismatch Patterns												Cycle
11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0	1	0	1
0	0	0	0	0	0	0	0	0	1	0	0	2
0	0	0	0	0	0	0	0	1	0	0	0	3
0	0	0	0	0	0	0	1	0	0	0	0	4
0	0	0	0	0	0	1	0	0	0	0	0	5
0	0	0	0	0	1	0	0	0	0	0	0	6
0	0	0	0	1	0	0	0	0	0	0	0	7
0	0	0	1	0	0	0	0	0	0	0	0	8
0	0	1	0	0	0	0	0	0	0	0	0	9
0	1	0	0	0	0	0	0	0	0	0	0	10
1	0	0	0	0	0	0	0	0	0	0	0	11

6.3.2.5.2.6 Error Forcing Mode

In error forcing mode, a test pattern is applied to inactivity monitor inputs of the compare logic to force an error at the compare error signal of the compare unit. Only one hardcoded test pattern of 'all-ones' is applied into CCMR5 during error forcing mode. The error forcing mode takes one cycle to complete. Hence, the failing signature is presented for one clock cycle and the mode is automatically switched to lockstep mode and the

Copyright © 2024 Texas Instruments Incorporated

R5FSS_CCMKEYR3 key register will show the lockstep key (0000). During this cycle the bus monitoring is disabled.

6.3.2.5.2.7 Self Test Error Forcing Mode

In self test error forcing mode, an error is forced at the self test error signal. The compare unit is still running in lockstep mode and the key is switched to lockstep after one clock cycle.

6.3.2.5.3 Polarity Inversion Logic

As already mentioned, there is logic (implemented via XOR) to test the comparison logic itself. The CCMR5 block can be configured to be in self test to check the integrity of the final XOR error generation logic, and can be switched back to functional compare mode after self test completion. To ensure that switching back to functional mode has happened, user can program CCMR5 polarity control register (R5FSS_CCMPOLCNTRL) to invert the polarity of eight CPU output signals, which will cause a CPU compare error generation. The eight signals that are chosen to be able to control the polarity are: AWVALIDMm, RVALIDSm, ATCEN0m, B0TCEN0m, BITCWEm, IRQACKm, AWVALIDPm, HWRITEP. See the *Arm Cortex-R5 Technical Reference Manual* for details on these signals.

The XOR inversion is placed between the two back-to-back delay flops of CPU1 outputs. Note that the XOR inversion logic is a diagnostic feature. If user does not observe a CPU compare error generation after programming the R5FSS_CCMPOLCNTRL[7:0] POL_INV inversion bits, this means that:

1. All eight signals of the inversion logics (XOR) are faulty and CPU compare error signal is not asserted. This is a very difficult case to hit but theoretically it could happen if a hard fault gets accumulated over times on these polarity inversion XOR gates; OR
2. The CCMR5 block has not switched back to functional mode yet

User can perform self test again to identify whether the issue is #1 or #2. If the second self test passes that means the issue is #1.

6.3.2.6 R5FSS Vectored Interrupt Manager (VIM)

6.3.2.6.1 VIM Overview

The VIM aggregates device interrupts and sends them to the R5F CPU(s). It can be used in either split or lockstep configuration. In split, it has two independent interrupt cores, one per CPU. In lockstep, CPU1 acts as a diagnostic on CPU0; only CPU0's outputs are used but all outputs are compared to CPU1 to provide diagnostic coverage.

The VIM module supports the following features:

- 512 interrupt inputs per R5F core
- Each interrupt has its own 4-bit programmable priority
  - Defined via the R5FSS_VIM_PRI_INT_j register
  - The VIM provides support for priority interruption of interrupts
- Each interrupt has its own enable mask
  - Interrupt enable is done via the R5FSS_VIM_INTR_EN_SET_j register
  - Interrupt disable is done via the R5FSS_VIM_INTR_EN_CLR_j register
- Each interrupt can be programmed as either an IRQ or FIQ
  - Defined via the R5FSS_VIM_INTMAP_j register
- Each interrupt has its own programmable 32-bit vector address associated with it
  - Defined via the R5FSS_VIM_VEC_INT_j register
  - Protected with SECDED
- One IRQn and one FIQn output per core
- Vectored interrupt interface
  - Compatible with R5F VIC port
- Default vector provided when a double-bit error is detected
- Split or lockstep capable
  - In lockstep mode, only interrupts connected to VIM interrupt core 0 are available
- Software interrupt generation

Copyright © 2024 Texas Instruments Incorporated

6.3.2.6.2 VIM Interrupt Inputs

The VIM supports 512 interrupt inputs per core. Each interrupt can be either a level or a pulse (both active-high). The interrupt mapping for the two R5F cores can be found in *Interrupt Sources*.

6.3.2.6.3 VIM Interrupt Outputs

The VIM has two interrupt outputs per core:

- *CoreN_IRQn*: This is a normal interrupt for core *N* (active-low level). It can be serviced via the VIC interface or through the MMR interface. Whenever an interrupt input goes high, if that interrupt is mapped as an IRQ (via the R5FSS_VIM_INTMAP_j register) and is enabled (via the R5FSS_VIM_INTR_EN_SET_j register), then it will cause an IRQ to assert
- *CoreN_FIQn*: This is a fast (or non-maskable) interrupt for core *N* (active-low level). FIQs always have priority over IRQs. An FIQ can be serviced through the MMR interface. Whenever an interrupt input goes high, if that interrupt is mapped as an FIQ and is enabled, then it will cause an FIQ to assert

6.3.2.6.4 VIM Interrupt Vector Table (VIM RAM)

For each VIM interrupt core, there is an associated interrupt vector table (VIM RAM) that is used to store the address of ISRs. During register vectored interrupt and hardware vectored interrupt, VIM accesses the interrupt vector table using the vector value to fetch the address of the corresponding ISR. Note that both interrupt vector tables are identical in their memory organization.

The VIM RAM is basically comprised of a set of interrupt vector registers (R5FSS_VIM_VEC_INT_j). Hence, the interrupt vector table is organized in 512 words of 30 bits, with a base address corresponding to the physical address of the first register in the group.

Note

The lower two bits of the 32-bit interrupt vector are always 0s.

Figure 6-4 shows the VIM RAM interrupt vector map.

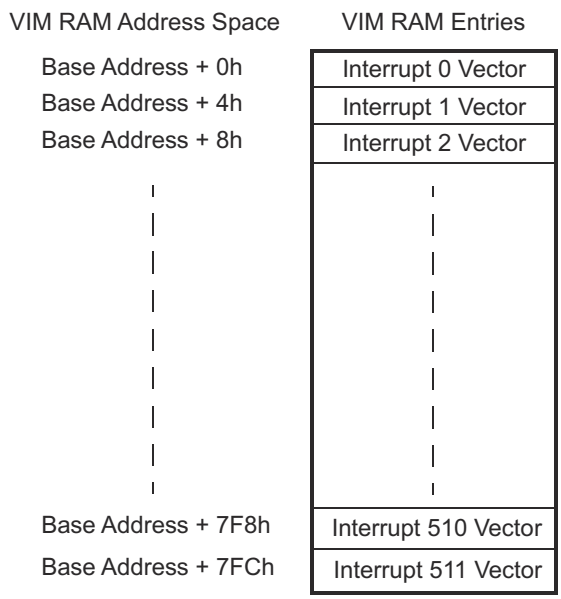


Figure 6-4. VIM RAM Interrupt Vector Map

The interrupt vector table has protection by ECC to indicate corruption due to soft errors. The ECC logic inside VIM supports SECDED. See Table 6-21 for the VIM RAM ID in the ECC aggregator map.

6.3.2.6.5 VIM Interrupt Prioritization

Each interrupt has a priority number assigned to it (set using the R5FSS_VIM_PRI_INT_j register). Legal values are 0 to 15, where 0 is the highest priority and 15 is the lowest priority. The highest priority interrupt is the

Copyright © 2024 Texas Instruments Incorporated

pending interrupt with the smallest priority number. If two pending interrupts have the same priority, the interrupt lowest numerically (0 through maximum number of interrupts) is prioritized. IRQs and FIQs are prioritized separately.

The VIM supports the interruption of the currently active interrupt by one with a higher priority. FIQs and IRQs are completely separate but both use the same mechanism.

When an interrupt goes from pending to active (FIQ: reading the R5FSS_VIM_FIQVEC register; IRQ: reading the R5FSS_VIM_IRQVEC register, or the *coreN_IRQACK* going high), then the interrupt is loaded into the corresponding active register (R5FSS_VIM_ACTFIQ / R5FSS_VIM_ACTIRQ), and all interrupts of an equal or lesser priority are masked (discarded). If prior to this interrupt being cleared (by writing to the R5FSS_VIM_FIQVEC register, or R5FSS_VIM_IRQVEC register) another interrupt of higher priority arrives, then the FIQn/IRQn will be asserted and that interrupt made pending as normal. The CPU may or may not service the higher priority interrupt. If the CPU switches this interrupt to active (by reading the R5FSS_VIM_FIQVEC / R5FSS_VIM_IRQVEC register, or (or *coreN_IRQACK* going high for an IRQ), then the currently active interrupt will be pushed onto a stack. When an interrupt is cleared by writing to the R5FSS_VIM_FIQVEC / R5FSS_VIM_IRQVEC register, if there are any interrupts on the stack, the first entry is popped off and put back into the R5FSS_VIM_ACTFIQ / R5FSS_VIM_ACTIRQ register, so that software may continue where it left off. Note that the R5FSS_VIM_IRQVEC / R5FSS_VIM_FIQVEC address registers are not repopulated with the old vector as it is assumed that the ISR is picking back up where it left off. Only the interrupt number and priority are restored to the R5FSS_VIM_ACTIRQ / R5FSS_VIM_ACTFIQ registers. If software needs the vector again, it will have to read it by using the interrupt number.

6.3.2.6.6 VIM ECC Support

The memory that holds the interrupt vector for each interrupt is protected by SECDED ECC. Single-bit errors are corrected and written back. Double-bit errors are not corrected. If a double-bit error occurs while trying to load a vector, then the R5FSS_VIM_DEDVEC register is used to provide the default vector for the *coreN_IRQADDRV* signal, the R5FSS_VIM_IRQVEC register, and the R5FSS_VIM_FIQVEC register. The R5FSS_VIM_DEDVEC should point to an ISR that handles the fact that there was an uncorrectable error in the interrupt handling.

Some possible remediating actions would be to:

1. Reconstruct the vector table and re-start the application
   a. Potentially switch to a completely software interrupt handler in the mean time
2. Restart the application from scratch
3. Reset the device
4. Sit in a loop (or WFI) while something external (for example, the ESM) responds to the DED interrupt that will be generated

It is up to the user and the application to determine the appropriate action.

Note

An interrupt that has an uncorrectable vector error (and thus uses the DED vector) will still have the priority of the original interrupt . This makes it possible for a higher priority interrupt to supercede the handling of the error.

Control and reporting are done by the R5FSS ECC aggregator.

6.3.2.6.7 VIM Lockstep Mode

In lockstep mode, CPU1 is used as a diagnostic for CPU0. In this mode, only the interrupt inputs for CPU0 are used. Besides to CPU0, these interrupt inputs are also internally routed to CPU1 (through the level-sync / edge-detect logic dedicated to CPU1, and additionally through some delay circuits). The outputs from both VIM interrupt cores are then sent to the R5FSS CCMR5 module through dedicated compare buses (with CPU0's outputs delayed). The CCMR5 module is responsible for comparing the two sets of output signals and for reporting any mismatches by generating a VIM bus compare interrupt.

Copyright © 2024 Texas Instruments Incorporated

Note

In lockstep mode, only the VIM RAM dedicated to CPU0 is used, so software *must not* do anything with the ECC interface on the VIM RAM dedicated to CPU1.

6.3.2.6.8 VIM IDLE State

The VIM will indicate IDLE when there are no pending unmasked interrupts or MMR accesses. The VIM does not have a clock stop interface.

6.3.2.6.9 VIM Interrupt Handling

There are multiple ways to service an interrupt depending on how much of the hardware assistance offered by the VIM the software wants to take advantage of.

For IRQs, it is recommended to use the procedure in Section 6.3.2.6.9.1, but the procedures in Section 6.3.2.6.9.2 or Section 6.3.2.6.9.3 (if a user wants to implement a fully software prioritization scheme) may be used as alternatives.

For FIQs, it is recommended to use the procedure in Section 6.3.2.6.9.4, but the procedure in Section 6.3.2.6.9.5 may be used as an alternative.

Note

These descriptions do not include steps such as stack pushes and state retention that software must take in order to return from the ISR. It is assumed that the programmer is aware of these steps.

6.3.2.6.9.1 Servicing IRQ Through Vector Interface

If the associated CPU has the vector (VIC) interface enabled, then the following method is used for servicing IRQs:

1. Hardware handshake
   a. CPU asserts *coreN_IRQACK* high
   b. VIM asserts *coreN_IRQADDRV* to indicate that the *coreN_IRQADDR* bus is stable with the correct vector address
   c. CPU reads *coreN_IRQADDR*, jumps to that address, and de-asserts *coreN_IRQACK* low
   d. VIM de-asserts *coreN_IRQn* and *coreN_IRQADDRV*, VIM masks (discards) all IRQs with the same or lower priority
   e. VIM loads the value from the R5FSS_VIM_PRIIRQ[9:0] NUM bit field (which corresponds to the vector address) into the R5FSS_VIM_ACTIRQ[9:0] NUM bit field, which causes the R5FSS_VIM_ACTIRQ[31] VALID bit to be set
2. Service the interrupt
3. Depending on whether the original source of the interrupt was a pulse or a level (determined by reading the R5FSS_VIM_ACTIRQ[9:0] NUM bit field to determine number, and reading the appropriate bit in the R5FSS_VIM_INTTYPE_j register to determine type)
   a. Pulse
      i. Clear the status by writing a '1' to the appropriate bit in the R5FSS_VIM_IRQSTS_j register, or R5FSS_VIM_STS_j register
      ii. Clear the interrupt at the source. This way, the source can generate another pulse, if it needs to, and the VIM will process this as a new interrupt
   b. Level
      i. Clear the interrupt at the source
      ii. Clear the status by writing a '1' to the appropriate bit in the R5FSS_VIM_IRQSTS_j register, or R5FSS_VIM_STS_j register. This way, the level should be gone at the input to the VIM, it will avoid falsely re-calling the interrupt. If the source maintains the level, then it means there is another interrupt
4. Write any value to the R5FSS_VIM_IRQVEC register
   a. This will clear the priority mask and will cause all interrupts to be re-evaluated for the new highest priority interrupt

Copyright © 2024 Texas Instruments Incorporated

b. This will also clear the R5FSS_VIM_ACTIRQ[31] VALID bit

6.3.2.6.9.2 Servicing IRQ Through MMR Interface

When an IRQ interrupt is received, the CPU should follow these steps if not using the vector interface:

1. Read the R5FSS_VIM_IRQVEC register and jump to that address to service the ISR
   a. Reading this register will mask (discard) all interrupts of an equal or lower priority and de-assert the *coreN_IRQn* output. If another interrupt of a higher priority becomes available, the *coreN_IRQn* will re-assert, allowing priority interruption of an interrupt
   b. Reading this register will cause the value from the R5FSS_VIM_PRIIRQ[9:0] NUM bit field to be loaded into the R5FSS_VIM_ACTIRQ[9:0] NUM bit field, and the R5FSS_VIM_ACTIRQ[31] VALID bit to be set
2. Service the interrupt
3. Depending on whether the original source of the interrupt was a pulse or a level
   a. Pulse
      i. Clear the status by writing a '1' to the appropriate bit in the R5FSS_VIM_STS_j register, or R5FSS_VIM_IRQSTS_j register
      ii. Clear the interrupt at the source
   b. Level
      i. Clear the interrupt at the source
      ii. Clear the status by writing a '1' to the appropriate bit in the R5FSS_VIM_STS_j register, or R5FSS_VIM_IRQSTS_j register
4. Write any value to the R5FSS_VIM_IRQVEC register
   a. This will clear the priority mask and will cause all interrupts to be re-evaluated for the new highest priority interrupt
   b. This will also clear the R5FSS_VIM_ACTIRQ[31] VALID bit

6.3.2.6.9.3 Servicing IRQ Through MMR Interface (Alternative)

If a user does not want to use the R5FSS_VIM_IRQVEC register, the VIM may be used as a more traditional interrupt controller. Note that in this mode, priority masking will not work if route 1b is used (see below) as the hardware prioritization may not match the software prioritization scheme. In this case, software would be responsible for doing all priority operations.

1. Determine which interrupt to service
   a. Read the R5FSS_VIM_PRIIRQ register to determine which interrupt is the highest priority IRQ currently asserted, OR
   b. Optionally read the R5FSS_VIM_IRQGSTS register to determine which groups have IRQs pending, then read the R5FSS_VIM_IRQSTS_j register and use a software prioritization scheme to determine which IRQ to service
2. Service the interrupt
3. Read the R5FSS_VIM_IRQVEC register
   a. Note that this step can be done any time before step 4
   b. Value is ignored
4. Depending on whether the original source of the interrupt was a pulse or a level
   a. Pulse
      i. Clear the status by writing a '1' to the appropriate bit in the R5FSS_VIM_STS_j register, or R5FSS_VIM_IRQSTS_j register
      ii. Clear the interrupt at the source.
   b. Level
      i. Clear the interrupt at the source
      ii. Clear the status by writing a '1' to the appropriate bit in the R5FSS_VIM_STS_j register, or R5FSS_VIM_IRQSTS_j register
5. Write any value to the R5FSS_VIM_IRQVEC register

6.3.2.6.9.4 Servicing FIQ

When an FIQ interrupt is received, the CPU should follow these steps:

1. Read the R5FSS_VIM_FIQVEC register and jump to that address to service the ISR

Copyright © 2024 Texas Instruments Incorporated

   a. Reading this register will mask (discard) all interrupts of an equal or lower priority and de-assert the *coreN_FIQn* output. If another interrupt of a higher priority becomes available, the *coreN_FIQn* will re-assert, allowing priority interruption of an interrupt.
   b. Reading this register will cause the value from the R5FSS_VIM_PRIFIQ[9:0] NUM bit field to be loaded into the R5FSS_VIM_ACTFIQ[9:0] NUM bit field, and the R5FSS_VIM_ACTFIQ[31] VALID bit to be set
2. Service the interrupt
3. Depending on whether the original source of the interrupt was a pulse or a level (determined by reading the R5FSS_VIM_ACTFIQ[9:0] NUM bit field to determine number, and reading the appropriate bit in the R5FSS_VIM_INTTYPE_j register to determine type)
   a. Pulse
      i. Clear the status by writing a '1' to the appropriate bit in the R5FSS_VIM_STS_j register, or R5FSS_VIM_FIQSTS_j register
      ii. Clear the interrupt at the source. This way, the source can generate another pulse, if it needs to, and the VIM will process this as a new interrupt
   b. Level
      i. Clear the interrupt at the source
      ii. Clear the status by writing a '1' to the appropriate bit in the R5FSS_VIM_STS_j register, or R5FSS_VIM_FIQSTS_j register. This way, the level should be gone at the input to the VIM, it will avoid falsely re-calling the interrupt. If the source maintains the level, then it means there is another interrupt
4. Write any value to the R5FSS_VIM_FIQVEC register
   a. This will clear the priority mask and will cause all interrupts to be re-evaluated for the new highest priority interrupt
   b. This will also clear the R5FSS_VIM_ACTFIQ[31] VALID bit

6.3.2.6.9.5 Servicing FIQ (Alternative)

If a user does not want to use the R5FSS_VIM_FIQVEC register, the VIM may be used as a more traditional interrupt controller. Note that in this mode, there is no hardware priority masking (because the R5FSS_VIM_FIQVEC register is never read). Software would be responsible for doing all priority operations.

1. Determine which interrupt to service
   a. Read the R5FSS_VIM_FIQVEC register to determine which interrupt is the highest priority FIQ currently asserted, OR
   b. Optionally read the R5FSS_VIM_FIQGSTS register to determine which groups have IRQs pending, then read the R5FSS_VIM_FIQSTS_j register and use a software prioritization scheme to determine which FIQ to service
2. Service the interrupt
3. Depending on whether the original source of the interrupt was a pulse or a level
   a. Pulse
      i. Clear the status by writing a '1' to the appropriate bit in the R5FSS_VIM_STS_j register, or R5FSS_VIM_FIQSTS_j register
      ii. Clear the interrupt at the source.
   b. Level
      i. Clear the interrupt at the source
      ii. Clear the status by writing a '1' to the appropriate bit in the R5FSS_VIM_STS_j register, or R5FSS_VIM_FIQSTS_j register.

6.3.2.7 R5FSS Region Address Translation (RAT)

6.3.2.7.1 RAT Overview

The R5F is a 32-bit processor, which means it can only access 4GB of directly addressable memory. The SoC is a 64-bit virtual (48-bit physical) address system. The R5FSS integrates a standard region-based address translation (RAT) unit (per core) to allow some of its address space to be remapped into the 64-bit world.

The RAT has 16 regions, with each region having dedicated MMRs that define its attributes:

- Base address of the region, defined via R5FSS_RAT_BASE_j[31-0] BASE
- Size of the region, defined via R5FSS_RAT_CTRL_j[5-0] SIZE

Copyright © 2024 Texas Instruments Incorporated

- Translated base address, defined via R5FSS_RAT_TRANS_U_j[15-0] UPPER and R5FSS_RAT_TRANS_L_j[31-0] LOWER

Regions can have any base address and size, as long as the base address is size aligned. The maximum region size is 4GB.

Note

The RAT does not provide any logic for checking if base address is size aligned, so it is software's responsibility to ensure that. Regions that are not aligned may have unpredictable results.

6.3.2.7.2 RAT Operation

The RAT works by translating a 32-bit input address into a 48-bit output address. Any input transaction that starts inside of a programmed region will have its address translated, if the region is enabled. Any disabled region is ignored from the translation lookup.

The input addresses are compared against all the enabled regions in parallel. The region size defines how many of the lowest address bits are included in the region space, starting from a 1B space to a 4GB space. Then the upper bits not part of the region are used for the lookup and comparison, matching those bits of the input addresses against the region base address. The region matches when it is enabled and the compare matches. The first region that matches has the output address set with the region's translated base address upper bits. The lower bits that are part of the region size are copied from the input address to the output address. If there are no region matches then the input address is copied to the output address entirely.

Note

Multiple region definitions must not overlap in their covered address space. The RAT does not check for this, so it is software's responsibility to take care of it. Overlapping regions may have unpredictable results.

Here is an example of a region-based address translation, assuming that region [0] is programmed as follows:

- Region [0] base address = 0x1234_5XXX
- Region [0] size = 4KB
- Region [0] translation upper address = 0x0000_ABCD
- Region [0] translation lower address = 0x5432_1XXX

If a transaction comes in with the address 0x1234_5678, then the output transaction will have the address 0x0000_ABCD_5432_1678.

6.3.2.7.3 RAT Error Logging

The RAT does not provide any protection against transactions that cross region boundaries, that is, start inside of a region and end outside of it, or start outside of a region and end inside of it. If a boundary crossing transaction is detected, then the transaction will be flagged as an error (exception) and a log will be made. The log consists of a series of MMRs that captures the details of the transaction, including the input address that caused the error. The RAT module can capture one error before it is cleared by software. Here is the list of exception logging registers:

- Control:
  - R5FSS_RAT_EXCEPTION_LOGGING_CONTROL
- Header:
  - R5FSS_RAT_EXCEPTION_LOGGING_HEADER0
  - R5FSS_RAT_EXCEPTION_LOGGING_HEADER1
- Data:
  - R5FSS_RAT_EXCEPTION_LOGGING_DATA0
  - R5FSS_RAT_EXCEPTION_LOGGING_DATA1
  - R5FSS_RAT_EXCEPTION_LOGGING_DATA2
  - R5FSS_RAT_EXCEPTION_LOGGING_DATA3

Copyright © 2024 Texas Instruments Incorporated

The log also produces an interrupt. To clear the log, software must either read the final error logging register, or manually clear the interrupt status bit. This will clear the error status, and not the actual log MMRs, but it does allow the next error to be captured into the log MMRs. If the status is not cleared and additional errors are detected, they are not logged.

The error reports use the value from a dedicated bootstrap (see Section 6.3.2.2.3) as the value for the R5FSS_RAT_EXCEPTION_LOGGING_HEADER0[23-8] SRC_ID field.

6.3.2.7.4 RAT Protection

As mentioned, the RAT module does not provide any protection by itself. The intention is that RAT is used in conjunction with memory protection unit (MPU) and system firewalls for protection. It is recommended that one or more RAT regions are directly mapped to an MPU region to provide 'MMU-like' behavior.

6.3.2.8 R5FSS ECC Support

The R5F provides native ECC and parity support on all related memories, generating and checking the redundancy automatically. The methods for checking and reporting errors are available in the *Arm Cortex-R5 Technical Reference Manual*.

The R5FSS adds the capability of testing this logic by allowing errors (single and double bit) to be injected into memories (for testing purposes) via an ECC aggregator (per core). Note that because the R5FSS ECC aggregator is only used in error-injection mode, it only supports a subset of the generic ECC aggregator functionality in the device.

For a detailed description of the generic ECC aggregator functionality, see Section 12.10.4, *ECC Aggregator*. For register descriptions of R5FSS CPU0 and CPU1 ECC aggregators, see *ECC Registers*.

Table 6-21 provides the RAM ID for each core. This is needed for bit field [10-0] ECC_VECTOR in the corresponding R5FSS_CPU0_VECTOR / R5FSS_CPU1_VECTOR register.

Table 6-21. RAM ID Map for ECC Aggregator (Per Core)

RAM ID	Memory Name
0	CPU0/1 ITAG RAM0
1	CPU0/1 ITAG RAM1
2	CPU0/1 ITAG RAM2
3	CPU0/1 ITAG RAM3
4	CPU0/1 IDATA BANK0
5	CPU0/1 IDATA BANK1
6	CPU0/1 IDATA BANK2
7	CPU0/1 IDATA BANK3
8	CPU0/1 DTAG RAM0
9	CPU0/1 DTAG RAM1
10	CPU0/1 DTAG RAM2
11	CPU0/1 DTAG RAM3
12	CPU0/1 DDIRTY RAM
13	CPU0/1 DDATA RAM0
14	CPU0/1 DDATA RAM1
15	CPU0/1 DDATA RAM2
16	CPU0/1 DDATA RAM3
17	CPU0/1 DDATA RAM4
18	CPU0/1 DDATA RAM5
19	CPU0/1 DDATA RAM6
20	CPU0/1 DDATA RAM7
21	CPU0/1 ATCM BANK0
22	CPU0/1 ATCM BANK1

Copyright © 2024 Texas Instruments Incorporated

Table 6-21. RAM ID Map for ECC Aggregator (Per Core) (continued)

RAM ID	Memory Name
23	CPU0/1 B0TCM BANK0
24	CPU0/1 B0TCM BANK1
25	CPU0/1 B1TCM BANK0
26	CPU0/1 B1TCM BANK1
27	CPU0/1 VIM RAM

6.3.2.9 R5FSS Memory View

The memory view of each R5F (that is, the memory map as seen by each R5F) is a function of several things:

- Exception vector bootstrap: The R5F exception table (including boot vector) is always 32 bytes at address 0x00000000 as seen by the R5F. If not booting from a TCM, then boot is done over the main memory interface. The exception vector bootstrap is under software control, which allows these 32 bytes at address 0x00000000 to be remapped somewhere else in the SoC memory map.
- TCM locations: TCMs can be enabled or disabled and located at different places in the memory map, depending on bootstrap configuration. For more details, see Table 6-13, and Section 6.3.2.2.2.
- Peripheral interface locatons: The RF5 natively supports three interfaces for peripheral access. Each can be enabled/disabled and located based on bootstrap configuration. Note that the VBUSP peripheral interface must be enabled in order to use RAT and VIM.
- RAT base address: This is determined by a bootstrap. This address is located within the VBUSP peripheral interface address space.
- VIM base address: This is determined by a bootstrap. This address is located within the VBUSP peripheral interface address space.
- RAT programming: The RAT can take regions of memory accessible by the main memory interface and map them to different addresses.

The combination of the above determines what the R5F sees where in the memory map, and over what interface different transactions come out. Every transaction that does not directly address a TCM or a peripheral interface comes over the main memory interface. Transactions on the main memory interface can be further remapped with the RAT.

See *Memory Map*, for the complete R5F memory view for this device.

6.3.2.10 R5FSS Debug and Trace

The R5FSS supports standard Arm CoreSight debug and trace architecture. For more details, see the *On-chip Debug* chapter.

6.3.2.11 R5FSS Boot Options

There are two methods of booting the R5F, or rather, two methods of placing the exception vectors (of which the boot vector is one).

The first method is to have the exception vectors external to the R5F. The user can place the exception vectors at the address indicated by the exception vector bootstrap and then program the boot vector there. When the processor exits reset, it will fetch the boot vector from this location.

The second method is to boot from a TCM. To do this, software should take the following steps:

1. Assert the correct bootstraps
   a. To boot from ATCM, set CPUn_INITRAMA (or CPUn_INITRAMB to boot from BTCM)
   b. Assert CPUn_LOCZRAMA properly for the desired TCM
2. Assert CPUn_HALT
3. Release the CPU from reset
4. Load the desired code into the TCM via the TCM slave port
   a. Exception vectors should be located at address 0x00000000 of the TCM
5. De-assert CPUn_HALT

Copyright © 2024 Texas Instruments Incorporated

6.3.2.12 R5FSS Core Memory ECC Events

The R5F core generates several events as part of event bus that can be monitored by the PMU for debugging. The memory ECC related events from the event bus are exported to ESM for monitoring.

There are four ECC interrupts to the ESM that aggregate different categories of ECC events – CPU0 single error, CPU0 multi error, CPU1 single error, and CPU1 multi error events. Each ECC event has a 2-bit event bus counter associated with it. Everytime an event occurs, the counter is incremented by 1 till it reaches the max value of 3. The interrupt is asserted if the bus counter of any event associated with the interrupt is non-zero.

Each event bus counter has a MMR decrement control to decrement the counter by 1. So, for example, if a counter value is 2, the MMR to decrement the counter would need to be written 2 times to decrement the counter to 0. The reason decrement control has been added instead of clear control is if a new error occurs between the time the status register is read and the clear MMR is written, the new error would be lost. Write-to-decrement ensures that this does not happen.

When all event bus counters of an associated interrupt are zero, the interrupt is cleared. It takes three clock cycles for the event bus counter to be decremented once the write to the decrement control MMR presents itself at the R5FSS boundary.

Since each of the four ECC interrupts have single bit control to set the interrupt but multiple bits for clearing it, note that once an interrupt is set using the R5FSS_EVNT_BUS_ESM_SET register, it can be cleared by setting all the bits of the R5FSS_EVNT_BUS_ESM_CLR register that correspond to that particular interrupt. For example, if bit [0] of the R5FSS_EVNT_BUS_ESM_SET register is set, it can be cleared by setting bits [7-0] of the R5FSS_EVNT_BUS_ESM_CLR register to clear the interrupt.

The R5F core event bus only signals event when it is enabled. Non-invasive or invasive debug mode needs to be enabled to enable the PMU counters.

The export of the events to the event bus can be enabled by setting the X bit in the Performance Monitor Control Register of the R5F core. For more details, refer to *Arm R5 TRM*.

Table 6-22. R5FSS Event Bus Single-Bit Error Events

Event Bus Bit #	Description	Associated Status Register
22	Instruction cache tag RAM parity or correctable ECC error	R5FSS_CPU0_EVNT_BUS_SB_ERR_CNT_STATUS[1:0] EVNT_BUS0 R5FSS_CPU1_EVNT_BUS_SB_ERR_CNT_STATUS[1:0] EVNT_BUS0
23	Instruction cache data RAM parity or correctable ECC error	R5FSS_CPU0_EVNT_BUS_SB_ERR_CNT_STATUS[3:2] EVNT_BUS1 R5FSS_CPU1_EVNT_BUS_SB_ERR_CNT_STATUS[3:2] EVNT_BUS1
24	Data cache tag or dirty RAM parity error or correctable ECC error, from data-side or ACP	R5FSS_CPU0_EVNT_BUS_SB_ERR_CNT_STATUS[5:4] EVNT_BUS2 R5FSS_CPU1_EVNT_BUS_SB_ERR_CNT_STATUS[5:4] EVNT_BUS2
25	Data cache data RAM parity error or correctable ECC error	R5FSS_CPU0_EVNT_BUS_SB_ERR_CNT_STATUS[7:6] EVNT_BUS3 R5FSS_CPU1_EVNT_BUS_SB_ERR_CNT_STATUS[7:6] EVNT_BUS3
40	ATCM single-bit ECC error	R5FSS_CPU0_EVNT_BUS_SB_ERR_CNT_STATUS[9:8] EVNT_BUS4 R5FSS_CPU1_EVNT_BUS_SB_ERR_CNT_STATUS[9:8] EVNT_BUS4
41	B0TCM single-bit ECC error	R5FSS_CPU0_EVNT_BUS_SB_ERR_CNT_STATUS[11:10] EVNT_BUS5 R5FSS_CPU1_EVNT_BUS_SB_ERR_CNT_STATUS[11:10] EVNT_BUS5
42	B1TCM single-bit ECC error	R5FSS_CPU0_EVNT_BUS_SB_ERR_CNT_STATUS[13:12] EVNT_BUS6 R5FSS_CPU1_EVNT_BUS_SB_ERR_CNT_STATUS[13:12] EVNT_BUS6
43	TCM correctable ECC error reported by load/store unit	R5FSS_CPU0_EVNT_BUS_SB_ERR_CNT_STATUS[15:14] EVNT_BUS7 R5FSS_CPU1_EVNT_BUS_SB_ERR_CNT_STATUS[15:14] EVNT_BUS7
44	TCM correctable ECC error reported by prefetch unit	R5FSS_CPU0_EVNT_BUS_SB_ERR_CNT_STATUS[17:16] EVNT_BUS8 R5FSS_CPU1_EVNT_BUS_SB_ERR_CNT_STATUS[17:16] EVNT_BUS8

Table 6-23. R5FSS Event Bus Multi-Bit Error Events

Event Bus Bit #	Description	Associated Status Register
26	TCM fatal ECC error reported from the prefetch unit	R5FSS_CPU0_EVNT_BUS_MB_ERR_CNT_STATUS[1:0] EVNT_BUS0 R5FSS_CPU1_EVNT_BUS_MB_ERR_CNT_STATUS[1:0] EVNT_BUS0
27	TCM fatal ECC error reported from the load/store unit	R5FSS_CPU0_EVNT_BUS_MB_ERR_CNT_STATUS[3:2] EVNT_BUS1 R5FSS_CPU1_EVNT_BUS_MB_ERR_CNT_STATUS[3:2] EVNT_BUS1

Copyright © 2024 Texas Instruments Incorporated

Table 6-23. R5FSS Event Bus Multi-Bit Error Events (continued)

Event Bus Bit #	Description	Associated Status Register
33	Data cache data RAM fatal ECC error	R5FSS_CPU0_EVNT_BUS_MB_ERR_CNT_STATUS[5:4] EVNT_BUS2 R5FSS_CPU1_EVNT_BUS_MB_ERR_CNT_STATUS[5:4] EVNT_BUS2
34	Data caches tag/dirty RAM fatal ECC error, from data-side or ACP	R5FSS_CPU0_EVNT_BUS_MB_ERR_CNT_STATUS[7:6] EVNT_BUS3 R5FSS_CPU1_EVNT_BUS_MB_ERR_CNT_STATUS[7:6] EVNT_BUS3
37	ATCM multi-bit ECC error	R5FSS_CPU0_EVNT_BUS_MB_ERR_CNT_STATUS[9:8] EVNT_BUS4 R5FSS_CPU1_EVNT_BUS_MB_ERR_CNT_STATUS[9:8] EVNT_BUS4
38	B0TCM multi-bit ECC error	R5FSS_CPU0_EVNT_BUS_MB_ERR_CNT_STATUS[11:10] EVNT_BUS5 R5FSS_CPU1_EVNT_BUS_MB_ERR_CNT_STATUS[11:10] EVNT_BUS5
39	B1TCM multi-bit ECC error	R5FSS_CPU0_EVNT_BUS_MB_ERR_CNT_STATUS[13:12] EVNT_BUS6 R5FSS_CPU1_EVNT_BUS_MB_ERR_CNT_STATUS[13:12] EVNT_BUS6

Copyright © 2024 Texas Instruments Incorporated

6.3.2.13 R5FSS_VIM Registers

Table 6-25 lists the memory-mapped registers for the R5FSS_VIM module. All register offset addresses not listed in Table 6-25 should be considered as reserved locations and the register contents should not be modified.

Table 6-24. R5FSS_VIM Instances

Instance	Base Address
MCU_R5FSS_VIC_CFG	40F8 0000h(1)
R5FSS_VIC_CFG	0FF8 0000h(2)

(1) MCU_R5FSS0 private memory-mapped register space. This region is only accessible by its associated R5F core; it is not accessible by any other SoC master. The base address is the same for each CPU view inside MCU_R5FSS0.
(2) R5FSS0/1 private memory-mapped register space. This region is only accessible by its associated R5F core; it is not accessible by any other SoC master. The base address is the same for each CPU view inside R5FSS0/1.

Table 6-25. R5FSS_VIM Registers (Part 1)

Offset	Acronym	Register Name	MCU_R5FSS_VIC_CFG Physical Address
0h	Section 6.3.2.13.1	Revision register	40F8 0000h
4h	Section 6.3.2.13.2	Info register	40F8 0004h
8h	Section 6.3.2.13.3	Prioritized IRQ register	40F8 0008h
Ch	Section 6.3.2.13.4	Prioritized FIQ register	40F8 000Ch
10h	Section 6.3.2.13.5	IRQ group status register	40F8 0010h
14h	Section 6.3.2.13.6	FIQ group status register	40F8 0014h
18h	Section 6.3.2.13.7	IRQ vector address register	40F8 0018h
1Ch	Section 6.3.2.13.8	FIQ vector address register	40F8 001Ch
20h	Section 6.3.2.13.9	Active IRQ register	40F8 0020h
24h	Section 6.3.2.13.10	Active FIQ register	40F8 0024h
30h	Section 6.3.2.13.11	DED vector address register	40F8 0030h
400h + formula	Section 6.3.2.13.12	Raw status/set register	40F8 0400h + formula
404h + formula	Section 6.3.2.13.13	Interrupt enable status/clear register	40F8 0404h + formula
408h + formula	Section 6.3.2.13.14	Interrupt enable set register	40F8 0408h + formula
40Ch + formula	Section 6.3.2.13.15	Interrupt enabled clear register	40F8 040Ch + formula
410h + formula	Section 6.3.2.13.16	IRQ interrupt enable status/clear register	40F8 0410h + formula
414h + formula	Section 6.3.2.13.17	FIQ interrupt enable status/clear register	40F8 0414h + formula
418h + formula	Section 6.3.2.13.18	Interrupt map register	40F8 0418h + formula
41Ch + formula	Section 6.3.2.13.19	Interrupt type register	40F8 041Ch + formula
1000h + formula	Section 6.3.2.13.20	Interrupt priority register	40F8 1000h + formula
2000h + formula	Section 6.3.2.13.21	Interrupt vector register	40F8 2000h + formula

Table 6-26. R5FSS_VIM Registers (Part 2)

Offset	Acronym	Register Name	R5FSS_VIC_CFG Physical Address
0h	Section 6.3.2.13.1	Revision register	0FF8 0000h
4h	Section 6.3.2.13.2	Info register	0FF8 0004h
8h	Section 6.3.2.13.3	Prioritized IRQ register	0FF8 0008h
Ch	Section 6.3.2.13.4	Prioritized FIQ register	0FF8 000Ch
10h	Section 6.3.2.13.5	IRQ group status register	0FF8 0010h
14h	Section 6.3.2.13.6	FIQ group status register	0FF8 0014h
18h	Section 6.3.2.13.7	IRQ vector address register	0FF8 0018h
1Ch	Section 6.3.2.13.8	FIQ vector address register	0FF8 001Ch

Copyright © 2024 Texas Instruments Incorporated

Table 6-26. R5FSS_VIM Registers (Part 2) (continued)

Offset	Acronym	Register Name	R5FSS_VIC_CFG Physical Address
20h	Section 6.3.2.13.9	Active IRQ register	0FF8 0020h
24h	Section 6.3.2.13.10	Active FIQ register	0FF8 0024h
30h	Section 6.3.2.13.11	DED vector address register	0FF8 0030h
400h + formula	Section 6.3.2.13.12	Raw status/set register	0FF8 0400h + formula
404h + formula	Section 6.3.2.13.13	Interrupt enable status/clear register	0FF8 0404h + formula
408h + formula	Section 6.3.2.13.14	Interrupt enable set register	0FF8 0408h + formula
40Ch + formula	Section 6.3.2.13.15	Interrupt enabled clear register	0FF8 040Ch + formula
410h + formula	Section 6.3.2.13.16	IRQ interrupt enable status/clear register	0FF8 0410h + formula
414h + formula	Section 6.3.2.13.17	FIQ interrupt enable status/clear register	0FF8 0414h + formula
418h + formula	Section 6.3.2.13.18	Interrupt map register	0FF8 0418h + formula
41Ch + formula	Section 6.3.2.13.19	Interrupt type register	0FF8 041Ch + formula
1000h + formula	Section 6.3.2.13.20	Interrupt priority register	0FF8 1000h + formula
2000h + formula	Section 6.3.2.13.21	Interrupt vector register	0FF8 2000h + formula

Copyright © 2024 Texas Instruments Incorporated

6.3.2.13.1 R5FSS_VIM_PID Register (Offset = 0h) [reset = 60900001h]

R5FSS_VIM_PID is shown in Figure 6-5 and described in Table 6-28.

Return to Summary Table.

This register contains the major and minor revisions for the module.

Table 6-27. R5FSS_VIM_PID Instances

Instance	Physical Address
MCU_R5FSS_VIC_CFG	40F8 0000h
R5FSS_VIC_CFG	0FF8 0000h

Figure 6-5. R5FSS_VIM_PID Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
REV																															
R-60900001h																															

LEGEND: R = Read Only; *-n* = value after reset

Table 6-28. R5FSS_VIM_PID Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	REV	R	60900001h	TI internal data. Identifies revision of peripheral.

Copyright © 2024 Texas Instruments Incorporated

6.3.2.13.2 R5FSS_VIM_INFO Register (Offset = 4h) [reset = 200h]

R5FSS_VIM_INFO is shown in Figure 6-6 and described in Table 6-30.

Return to Summary Table.

This contains information about the configuration of the R5FSS_VIM.

Table 6-29. R5FSS_VIM_INFO Instances

Instance	Physical Address
MCU_R5FSS_VIC_CFG	40F8 0004h
R5FSS_VIC_CFG	0FF8 0004h

Figure 6-6. R5FSS_VIM_INFO Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																					INTERRUPTS										
R-0h																					R-200h										

LEGEND: R = Read Only; *-n* = value after reset

Table 6-30. R5FSS_VIM_INFO Register Field Descriptions

Bit	Field	Type	Reset	Description
31-11	RESERVED	R	0h	Reserved. Reads return 0. Writes have no effect.
10-0	INTERRUPTS	R	200h	Indicates the number of interrupts supported by the VIM.

Copyright © 2024 Texas Instruments Incorporated

6.3.2.13.3 R5FSS_VIM_PRIIRQ Register (Offset = 8h) [reset = 0h]

R5FSS_VIM_PRIIRQ is shown in Figure 6-7 and described in Table 6-32.

Return to Summary Table.

This register contains the number of the highest priority pending IRQ.

Table 6-31. R5FSS_VIM_PRIIRQ Instances

Instance	Physical Address
MCU_R5FSS_VIC_CFG	40F8 0008h
R5FSS_VIC_CFG	0FF8 0008h

Figure 6-7. R5FSS_VIM_PRIIRQ Register

31	30	29	28	27	26	25	24
VALID	RESERVED						
R-0h	R-0h						
23	**22**	**21**	**20**	**19**	**18**	**17**	**16**
RESERVED				PRI			
R-0h				R-0h			
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**
RESERVED						NUM	
R-0h						R-0h	
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
NUM							
R-0h							

LEGEND: R = Read Only; *-n* = value after reset

Table 6-32. R5FSS_VIM_PRIIRQ Register Field Descriptions

Bit	Field	Type	Reset	Description
31	VALID	R	0h	This field indicates if the NUM field of this register is valid. 0h = NUM field is invalid (no pending IRQ interrupts) 1h = NUM field is valid (pending IRQ interrupt)
30-20	RESERVED	R	0h	Reserved. Reads return 0. Writes have no effect.
19-16	PRI	R	0h	This field indicates the priority of the pending IRQ interrupt. 0h = Highest priority ... Fh = Lowest priority This field is only valid if the VALID flag of this register is set. Otherwise the value is unpredictable.
15-10	RESERVED	R	0h	Reserved. Reads return 0. Writes have no effect.
9-0	NUM	R	0h	This field indicates the interrupt number of the pending IRQ interrupt with the highest priority. 0h = Interrupt 0 1h = Interrupt 1 ... 3FFh = Interrupt 1023 (Note: The highest supported value is device-specific.) This field is only valid if the VALID flag of this register is set. Otherwise the value is unpredictable.

Copyright © 2024 Texas Instruments Incorporated

6.3.2.13.4 R5FSS_VIM_PRIFIQ Register (Offset = Ch) [reset = 0h]

R5FSS_VIM_PRIFIQ is shown in Figure 6-8 and described in Table 6-34.

Return to Summary Table.

This register contains the number of the highest priority pending FIQ.

Table 6-33. R5FSS_VIM_PRIFIQ Instances

Instance	Physical Address
MCU_R5FSS_VIC_CFG	40F8 000Ch
R5FSS_VIC_CFG	0FF8 000Ch

Figure 6-8. R5FSS_VIM_PRIFIQ Register

31	30	29	28	27	26	25	24
VALID	RESERVED						
R-0h	R-0h						
23	**22**	**21**	**20**	**19**	**18**	**17**	**16**
RESERVED				PRI			
R-0h				R-0h			
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**
RESERVED						NUM	
R-0h						R-0h	
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
NUM							
R-0h							

LEGEND: R = Read Only; *-n* = value after reset

Table 6-34. R5FSS_VIM_PRIFIQ Register Field Descriptions

Bit	Field	Type	Reset	Description
31	VALID	R	0h	This field indicates if the NUM field of this register is valid. 0h = NUM field is invalid (no pending FIQ interrupts) 1h = NUM field is valid (pending FIQ interrupt)
30-20	RESERVED	R	0h	Reserved. Reads return 0. Writes have no effect.
19-16	PRI	R	0h	This field indicates the priority of the pending FIQ interrupt. 0h = Highest priority ... Fh = Lowest priority This field is only valid if the VALID flag of this register is set. Otherwise the value is unpredictable.
15-10	RESERVED	R	0h	Reserved. Reads return 0. Writes have no effect.
9-0	NUM	R	0h	This field indicates the interrupt number of the pending FIQ interrupt with the highest priority. 0h = Interrupt 0 1h = Interrupt 1 ... 3FFh = Interrupt 1023 (Note: The highest supported value is device-specific.) This field is only valid if the VALID flag of this register is set. Otherwise the value is unpredictable.

Copyright © 2024 Texas Instruments Incorporated

6.3.2.13.5 R5FSS_VIM_IRQGSTS Register (Offset = 10h) [reset = 0h]

R5FSS_VIM_IRQGSTS is shown in Figure 6-9 and described in Table 6-36.

Return to Summary Table.

This register indicates which groups of interrupts have pending, unmasked IRQ interrupts.

Table 6-35. R5FSS_VIM_IRQGSTS Instances

Instance	Physical Address
MCU_R5FSS_VIC_CFG	40F8 0010h
R5FSS_VIC_CFG	0FF8 0010h

Figure 6-9. R5FSS_VIM_IRQGSTS Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STS																															
R-0h																															

LEGEND: R = Read Only; *-n* = value after reset

Table 6-36. R5FSS_VIM_IRQGSTS Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	STS	R	0h	This field indicates that one or more interrupts in group M are mapped to IRQ, unmasked, and pending. Bit 0 corresponds to group 0, bit 1 corresponds to group 1, etc. The interrupts associated with each group are [(M*32)+31:M*32]

Copyright © 2024 Texas Instruments Incorporated

6.3.2.13.6 R5FSS_VIM_FIQGSTS Register (Offset = 14h) [reset = 0h]

R5FSS_VIM_FIQGSTS is shown in Figure 6-10 and described in Table 6-38.

Return to Summary Table.

This register indicates which groups of interrupts have pending, unmasked FIQ interrupts.

Table 6-37. R5FSS_VIM_FIQGSTS Instances

Instance	Physical Address
MCU_R5FSS_VIC_CFG	40F8 0014h
R5FSS_VIC_CFG	0FF8 0014h

Figure 6-10. R5FSS_VIM_FIQGSTS Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STS																															
R-0h																															

LEGEND: R = Read Only; *-n* = value after reset

Table 6-38. R5FSS_VIM_FIQGSTS Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	STS	R	0h	This field indicates that one or more interrupts in group M are mapped to FIQ, unmasked, and pending. Bit 0 corresponds to group 0, bit 1 corresponds to group 1, etc. The interrupts associated with each group are [(M*32)+31:M*32]

Copyright © 2024 Texas Instruments Incorporated

6.3.2.13.7 R5FSS_VIM_IRQVEC Register (Offset = 18h) [reset = 0h]

R5FSS_VIM_IRQVEC is shown in Figure 6-11 and described in Table 6-40.

Return to Summary Table.

This register contains the 32-bit interrupt vector address of the currently pending IRQ.

Table 6-39. R5FSS_VIM_IRQVEC Instances

Instance	Physical Address
MCU_R5FSS_VIC_CFG	40F8 0018h
R5FSS_VIC_CFG	0FF8 0018h

Figure 6-11. R5FSS_VIM_IRQVEC Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADDR																														RESERVED	
R-0h																														R-0h	

LEGEND: R = Read Only; *-n* = value after reset

Table 6-40. R5FSS_VIM_IRQVEC Register Field Descriptions

Bit	Field	Type	Reset	Description
31-2	ADDR	R	0h	This field contains the upper 30 bits of the 32-bit interrupt vector address (addresses must be 32-bit aligned) of the currently pending highest priority IRQ (as indicated by the R5FSS_VIM_PRIIRQ[9-0] NUM field). Reading this register returns the interrupt vector address of the pending IRQ with the highest priority. It also has the following effects: - Mask all IRQ interrupts of an equivalent or lower priority - De-asserts the IRQn signal - De-asserts the coreN_IRQADDRV signal (if set) - Loads R5FSS_VIM_ACTIRQ Writing any value to this register will not alter its contents, but will have the following effect: - Remove the mask on all IRQ priorities This field is only valid if the R5FSS_VIM_PRIIRQ[31] VALID flag is set.
1-0	RESERVED	R	0h	Reserved. Read as 0. The lower 2 bits of the 32-bit vector address are always 0. (Vector addresses must be 32-bit aligned.)

Copyright © 2024 Texas Instruments Incorporated

6.3.2.13.8 R5FSS_VIM_FIQVEC Register (Offset = 1Ch) [reset = 0h]

R5FSS_VIM_FIQVEC is shown in Figure 6-12 and described in Table 6-42.

Return to Summary Table.

This register contains the 32-bit interrupt vector address of the currently pending FIQ.

Table 6-41. R5FSS_VIM_FIQVEC Instances

Instance	Physical Address
MCU_R5FSS_VIC_CFG	40F8 001Ch
R5FSS_VIC_CFG	0FF8 001Ch

Figure 6-12. R5FSS_VIM_FIQVEC Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADDR																														RESERVED	
R-0h																														R-0h	

LEGEND: R = Read Only; *-n* = value after reset

Table 6-42. R5FSS_VIM_FIQVEC Register Field Descriptions

Bit	Field	Type	Reset	Description
31-2	ADDR	R	0h	This field contains the upper 30 bits of the 32-bit interrupt vector address (addresses must be 32-bit aligned) of the currently pending highest priority FIQ (as indicated by the R5FSS_VIM_PRIFIQ[9-0] NUM field). Reading this register returns the interrupt vector address of the pending FIQ with the highest priority. It also has the following effects: - Mask all FIQ interrupts of an equivalent or lower priority - De-asserts the FIQn signal - Loads R5FSS_VIM_ACTFIQ Writing any value to this register will not alter its contents, but will have the following effect: - Remove the mask on all FIQ priorities This field is only valid if the R5FSS_VIM_PRIFIQ[31] VALID flag is set.
1-0	RESERVED	R	0h	Reserved. Read as 0. The lower 2 bits of the 32-bit vector address are always 0. (Vector addresses must be 32-bit aligned.)

Copyright © 2024 Texas Instruments Incorporated

6.3.2.13.9 R5FSS_VIM_ACTIRQ Register (Offset = 20h) [reset = 0h]

R5FSS_VIM_ACTIRQ is shown in Figure 6-13 and described in Table 6-44.

Return to Summary Table.

This register contains the number of the active IRQ.

Table 6-43. R5FSS_VIM_ACTIRQ Instances

Instance	Physical Address
MCU_R5FSS_VIC_CFG	40F8 0020h
R5FSS_VIC_CFG	0FF8 0020h

Figure 6-13. R5FSS_VIM_ACTIRQ Register

31	30	29	28	27	26	25	24
VALID	RESERVED						
R-0h	R-0h						
23	**22**	**21**	**20**	**19**	**18**	**17**	**16**
RESERVED				PRI			
R-0h				R-0h			
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**
RESERVED						NUM	
R-0h						R-0h	
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
NUM							
R-0h							

LEGEND: R = Read Only; *-n* = value after reset

Table 6-44. R5FSS_VIM_ACTIRQ Register Field Descriptions

Bit	Field	Type	Reset	Description
31	VALID	R	0h	This field indicates if the NUM field of this register is valid. This field is set whenever the R5FSS_VIM_IRQVEC is read. It is cleared whenever the R5FSS_VIM_IRQVEC is written. 0h = NUM field is invalid (no active IRQ interrupts) 1h = NUM field is valid (active IRQ interrupt)
30-20	RESERVED	R	0h	Reserved. Reads return 0. Writes have no effect.
19-16	PRI	R	0h	This field indicates the priority of the active IRQ interrupt. 0h = Highest priority ... Fh = Lowest priority This field is only valid if the VALID flag of this register is set. Otherwise the value is unpredictable.
15-10	RESERVED	R	0h	Reserved. Reads return 0. Writes have no effect.

Copyright © 2024 Texas Instruments Incorporated

Table 6-44. R5FSS_VIM_ACTIRQ Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
9-0	NUM	R	0h	This field indicates the interrupt number of the active IRQ interrupt. This field is loaded with the value from the R5FSS_VIM_PRIIRQ whenever the R5FSS_VIM_IRQVEC is read. 0h = Interrupt 0 1h = Interrupt 1 … 3FFh = Interrupt 1023 (Note: The highest supported value is device-specific.) This field is only valid if the VALID flag of this register is set. Otherwise the value is unpredictable.

Copyright © 2024 Texas Instruments Incorporated

6.3.2.13.10 R5FSS_VIM_ACTFIQ Register (Offset = 24h) [reset = 0h]

R5FSS_VIM_ACTFIQ is shown in Figure 6-14 and described in Table 6-46.

Return to Summary Table.

This register contains the number of the active FIQ.

Table 6-45. R5FSS_VIM_ACTFIQ Instances

Instance	Physical Address
MCU_R5FSS_VIC_CFG	40F8 0024h
R5FSS_VIC_CFG	0FF8 0024h

Figure 6-14. R5FSS_VIM_ACTFIQ Register

31	30	29	28	27	26	25	24
VALID	RESERVED						
R-0h	R-0h						
23	**22**	**21**	**20**	**19**	**18**	**17**	**16**
RESERVED				PRI			
R-0h				R-0h			
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**
RESERVED						NUM	
R-0h						R-0h	
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
NUM							
R-0h							

LEGEND: R = Read Only; *-n* = value after reset

Table 6-46. R5FSS_VIM_ACTFIQ Register Field Descriptions

Bit	Field	Type	Reset	Description
31	VALID	R	0h	This field indicates if the NUM field of this register is valid. This field is set whenever the R5FSS_VIM_PRIFIQ is read. It is cleared whenever the R5FSS_VIM_FIQVEC is written. 0h = NUM field is invalid (no active FIQ interrupts) 1h = NUM field is valid (active FIQ interrupt)
30-20	RESERVED	R	0h	Reserved. Reads return 0. Writes have no effect.
19-16	PRI	R	0h	This field indicates the priority of the active FIQ interrupt. 0h = Highest priority ... Fh = Lowest priority This field is only valid if the VALID flag of this register is set. Otherwise the value is unpredictable.
15-10	RESERVED	R	0h	Reserved. Reads return 0. Writes have no effect.

Copyright © 2024 Texas Instruments Incorporated

Table 6-46. R5FSS_VIM_ACTFIQ Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
9-0	NUM	R	0h	This field indicates the interrupt number of the active FIQ interrupt. This field is loaded with the value from the R5FSS_VIM_PRIFIQ whenever the R5FSS_VIM_FIQVEC is read. 0h = Interrupt 0 1h = Interrupt 1 … 3FFh = Interrupt 1023 (Note: The highest supported value is device-specific.) This field is only valid if the VALID flag of this register is set. Otherwise the value is unpredictable.

Copyright © 2024 Texas Instruments Incorporated

6.3.2.13.11 R5FSS_VIM_DEDVEC Register (Offset = 30h) [reset = 0h]

R5FSS_VIM_DEDVEC is shown in Figure 6-15 and described in Table 6-48.

Return to Summary Table.

This register contains the 32-bit interrupt vector address to be used as a default in case of a DED error in any of the vectors.

Table 6-47. R5FSS_VIM_DEDVEC Instances

Instance	Physical Address
MCU_R5FSS_VIC_CFG	40F8 0030h
R5FSS_VIC_CFG	0FF8 0030h

Figure 6-15. R5FSS_VIM_DEDVEC Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADDR																														RESERVED	
R/W-0h																														R-0h	

LEGEND: R = Read Only; R/W = Read/Write; *-n* = value after reset

Table 6-48. R5FSS_VIM_DEDVEC Register Field Descriptions

Bit	Field	Type	Reset	Description
31-2	ADDR	R/W	0h	This field contains the upper 30 bits of the 32-bit interrupt vector address (the address must be 32-bit aligned) of an interrupt to be used if an uncorrectable double-bit error (DED) is detected in any of the interrupt vector addresses. If there is a DED, both the R5FSS_VIM_IRQVEC and R5FSS_VIM_FIQVEC registers (along with the VECADDR output) will be populated with the value in this field instead of their normal vector. - ECC Aggregator will indicate where there was a DED. Software may go correct that one location. - Software may keep an ISR at the DED vector which corrects the vectors or otherwise deals with the scenario. - This mechanism is provided because there is no way to indicate on the coreN_IRQADDR to the CPU that the vector is invalid, and it is undesirable to have the CPU jump to a random address.
1-0	RESERVED	R	0h	Reserved. Read as 0. The lower 2 bits of the 32-bit vector address are always 0. (Vector addresses must be 32-bit aligned.)

Copyright © 2024 Texas Instruments Incorporated

6.3.2.13.12 R5FSS_VIM_RAW_j Register (Offset = 400h + formula) [reset = 0h]

R5FSS_VIM_RAW_j is shown in Figure 6-16 and described in Table 6-50.

Return to Summary Table.

This register indicates the raw status of the events in group M.

Offset = 400h + (j * 20h); where j = 0h to Fh.

Table 6-49. R5FSS_VIM_RAW_j Instances

Instance	Physical Address
MCU_R5FSS_VIC_CFG	40F8 0400h + formula
R5FSS_VIC_CFG	0FF8 0400h + formula

Figure 6-16. R5FSS_VIM_RAW_j Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STS																															
R/W1S-0h																															

LEGEND: R/W1S = Read/Write 1 to Set Bit; *-n* = value after reset

Table 6-50. R5FSS_VIM_RAW_j Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	STS	R/W1S	0h	This is the raw status of the events in group M. Each bit corresponds to event Q, where Q = M*32+bit (example: bit 0 is event M*32+0, bit 1 is M*32+1, etc). Read 0h = Inactive Read 1h = Active/pending Write 0h = No effect Write 1h = Set to interrupt raw status

Copyright © 2024 Texas Instruments Incorporated

6.3.2.13.13 R5FSS_VIM_STS_j Register (Offset = 404h + formula) [reset = 0h]

R5FSS_VIM_STS_j is shown in Figure 6-17 and described in Table 6-52.

Return to Summary Table.

This register indicates the masked status of the events in group M.

Offset = 404h + (j * 20h); where j = 0h to Fh.

Table 6-51. R5FSS_VIM_STS_j Instances

Instance	Physical Address
MCU_R5FSS_VIC_CFG	40F8 0404h + formula
R5FSS_VIC_CFG	0FF8 0404h + formula

Figure 6-17. R5FSS_VIM_STS_j Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSK																															
R/W1C-0h																															

LEGEND: R/W1C = Read/Write 1 to Clear Bit; *-n* = value after reset

Table 6-52. R5FSS_VIM_STS_j Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	MSK	R/W1C	0h	This is the masked status of the events in group M. Each bit corresponds to event Q, where Q = M*32+bit (example: bit 0 is event M*32+0, bit 1 is M*32+1, etc). Read 0h = Inactive or disabled Read 1h = Active/pending and enabled Write 0h = No effect Write 1h = Clear interrupt raw status

Copyright © 2024 Texas Instruments Incorporated

6.3.2.13.14 R5FSS_VIM_INTR_EN_SET_j Register (Offset = 408h + formula) [reset = 0h]

R5FSS_VIM_INTR_EN_SET_j is shown in Figure 6-18 and described in Table 6-54.

Return to Summary Table.

This register is used to enable the mask for the events in group M.

Offset = 408h + (j * 20h); where j = 0h to Fh.

Table 6-53. R5FSS_VIM_INTR_EN_SET_j Instances

Instance	Physical Address
MCU_R5FSS_VIC_CFG	40F8 0408h + formula
R5FSS_VIC_CFG	0FF8 0408h + formula

Figure 6-18. R5FSS_VIM_INTR_EN_SET_j Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSK																															
R/W1S-0h																															

LEGEND: R/W1S = Read/Write 1 to Set Bit; *-n* = value after reset

Table 6-54. R5FSS_VIM_INTR_EN_SET_j Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	MSK	R/W1S	0h	This field is used to enable the mask of events in group M. Each bit corresponds to event Q, where Q = M*32+bit (example: bit 0 is event M*32+0, bit 1 is M*32+1, etc). Read 0h = Disabled Read 1h = Enabled Write 0h = No effect Write 1h = Set enable

Copyright © 2024 Texas Instruments Incorporated

6.3.2.13.15 R5FSS_VIM_INTR_EN_CLR_j Register (Offset = 40Ch + formula) [reset = 0h]

R5FSS_VIM_INTR_EN_CLR_j is shown in Figure 6-19 and described in Table 6-56.

Return to Summary Table.

This register is used to disable the mask for the events in group M.

Offset = 40Ch + (j * 20h); where j = 0h to Fh.

Table 6-55. R5FSS_VIM_INTR_EN_CLR_j Instances

Instance	Physical Address
MCU_R5FSS_VIC_CFG	40F8 040Ch + formula
R5FSS_VIC_CFG	0FF8 040Ch + formula

Figure 6-19. R5FSS_VIM_INTR_EN_CLR_j Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSK																															
R/W1C-0h																															

LEGEND: R/W1C = Read/Write 1 to Clear Bit; *-n* = value after reset

Table 6-56. R5FSS_VIM_INTR_EN_CLR_j Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	MSK	R/W1C	0h	This field is used to disable the mask of events in group M. Each bit corresponds to event Q, where Q = M*32+bit (example: bit 0 is event M*32+0, bit 1 is M*32+1, etc). Read 0h = Disabled Read 1h = Enabled Read 0h = No effect Read 1h = Clear enable

Copyright © 2024 Texas Instruments Incorporated

6.3.2.13.16 R5FSS_VIM_IRQSTS_j Register (Offset = 410h + formula) [reset = 0h]

R5FSS_VIM_IRQSTS_j is shown in Figure 6-20 and described in Table 6-58.

Return to Summary Table.

This register indicates the masked status of the events in Group M that are also mapped as IRQs.

Offset = 410h + (j * 20h); where j = 0h to Fh.

Table 6-57. R5FSS_VIM_IRQSTS_j Instances

Instance	Physical Address
MCU_R5FSS_VIC_CFG	40F8 0410h + formula
R5FSS_VIC_CFG	0FF8 0410h + formula

Figure 6-20. R5FSS_VIM_IRQSTS_j Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSK																															
R/W1C-0h																															

LEGEND: R/W1C = Read/Write 1 to Clear Bit; *-n* = value after reset

Table 6-58. R5FSS_VIM_IRQSTS_j Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	MSK	R/W1C	0h	This is the masked status of the events in group M that are mapped to IRQ. Each bit corresponds to event Q, where Q = M*32+bit (example: bit 0 is event M*32+0, bit 1 is M*32+1, etc). Read 0h = Inactive, disabled, or not an IRQ Read 1h = Active/pending, enabled, and IRQ Write 0h = No effect Write 1h = Clear interrupt raw status (if IRQ)

Copyright © 2024 Texas Instruments Incorporated

6.3.2.13.17 R5FSS_VIM_FIQSTS_j Register (Offset = 414h + formula) [reset = 0h]

R5FSS_VIM_FIQSTS_j is shown in Figure 6-21 and described in Table 6-60.

Return to Summary Table.

This register indicates the masked status of the events in group M that are also mapped as FIQs.

Offset = 414h + (j * 20h); where j = 0h to Fh.

Table 6-59. R5FSS_VIM_FIQSTS_j Instances

Instance	Physical Address
MCU_R5FSS_VIC_CFG	40F8 0414h + formula
R5FSS_VIC_CFG	0FF8 0414h + formula

Figure 6-21. R5FSS_VIM_FIQSTS_j Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSK																															
R/W1C-0h																															

LEGEND: R/W1C = Read/Write 1 to Clear Bit; *-n* = value after reset

Table 6-60. R5FSS_VIM_FIQSTS_j Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	MSK	R/W1C	0h	This is the masked status of the events in group M that are mapped to FIQ. Each bit corresponds to event Q, where Q = M*32+bit (example: bit 0 is event M*32+0, bit 1 is M*32+1, etc). Read 0h = Inactive, disabled, or not an FIQ Read 1h = Active/pending, enabled, and FIQ Write 0h = No effect Write 1h = Clear interrupt raw status (if FIQ)

Copyright © 2024 Texas Instruments Incorporated

6.3.2.13.18 R5FSS_VIM_INTMAP_j Register (Offset = 418h + formula) [reset = 0h]

R5FSS_VIM_INTMAP_j is shown in Figure 6-22 and described in Table 6-62.

Return to Summary Table.

This register is used to map interrupts as IRQ or FIQ.

Offset = 418h + (j * 20h); where j = 0h to Fh.

Table 6-61. R5FSS_VIM_INTMAP_j Instances

Instance	Physical Address
MCU_R5FSS_VIC_CFG	40F8 0418h + formula
R5FSS_VIC_CFG	0FF8 0418h + formula

Figure 6-22. R5FSS_VIM_INTMAP_j Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSK																															
R/W-0h																															

LEGEND: R/W = Read/Write; *-n* = value after reset

Table 6-62. R5FSS_VIM_INTMAP_j Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	MSK	R/W	0h	This field is used to indicate which interrupt the corresponding event influences (if enabled) for event group M. Each bit corresponds to event Q, where Q = M*32+bit (example: bit 0 is event M*32+0, bit 1 is M*32+1, etc). 0h = IRQ interrupt (default) 1h = FIQ interrupt

Copyright © 2024 Texas Instruments Incorporated

6.3.2.13.19 R5FSS_VIM_INTTYPE_j Register (Offset = 41Ch + formula) [reset = 0h]

R5FSS_VIM_INTTYPE_j is shown in Figure 6-23 and described in Table 6-64.

Return to Summary Table.

This register indicates whether an interrupt is a pulse or level source.

Offset = 41Ch + (j * 20h); where j = 0h to Fh.

Table 6-63. R5FSS_VIM_INTTYPE_j Instances

Instance	Physical Address
MCU_R5FSS_VIC_CFG	40F8 041Ch + formula
R5FSS_VIC_CFG	0FF8 041Ch + formula

Figure 6-23. R5FSS_VIM_INTTYPE_j Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSK																															
R/W-0h																															

LEGEND: R/W = Read/Write; *-n* = value after reset

Table 6-64. R5FSS_VIM_INTTYPE_j Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	MSK	R/W	0h	This field is used to indicate whether the source of an interrupt is a level (default) or a pulse for event group M. Each bit corresponds to event Q, where Q = M*32+bit (example: bit 0 is event M*32+0, bit 1 is M*32+1, etc). 0h = Level (default) 1h = Pulse Note: This is only informational so that an ISR may query this register and know whether it has to clear a pulse event or a level event. The value has no effect on how the VIM hardware functions. The input interrupts are agnostic as to whether they are pulse or level.

Copyright © 2024 Texas Instruments Incorporated

6.3.2.13.20 R5FSS_VIM_PRI_INT_j Register (Offset = 1000h + formula) [reset = 0h]

R5FSS_VIM_PRI_INT_j is shown in Figure 6-24 and described in Table 6-66.

Return to Summary Table.

This register is used to set the priority of interrupt Q.

Offset = 1000h + (j * 4h); where j = 0h to 1FFh.

Table 6-65. R5FSS_VIM_PRI_INT_j Instances

Instance	Physical Address
MCU_R5FSS_VIC_CFG	40F8 1000h + formula
R5FSS_VIC_CFG	0FF8 1000h + formula

Figure 6-24. R5FSS_VIM_PRI_INT_j Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																												VAL			
R-0h																												R/W-Fh			

LEGEND: R = Read Only; R/W = Read/Write; *-n* = value after reset

Table 6-66. R5FSS_VIM_PRI_INT_j Register Field Descriptions

Bit	Field	Type	Reset	Description
31-4	RESERVED	R	0h	Reserved. Reads return 0. Writes have no effect.
3-0	VAL	R/W	Fh	This is the priority for interrupt Q. If two interrupts have the same priority, then whichever interrupt has the lower number Q wins arbitration. 0h = Highest priority … Fh = Lowest priority (default)

Copyright © 2024 Texas Instruments Incorporated

6.3.2.13.21 R5FSS_VIM_VEC_INT_j Register (Offset = 2000h + formula) [reset = 0h]

R5FSS_VIM_VEC_INT_j is shown in Figure 6-25 and described in Table 6-68.

Return to Summary Table.

This register contains the vector address associated with interrupt Q.

Offset = 2000h + (j * 4h); where j = 0h to 1FFh.

Table 6-67. R5FSS_VIM_VEC_INT_j Instances

Instance	Physical Address
MCU_R5FSS_VIC_CFG	40F8 2000h + formula
R5FSS_VIC_CFG	0FF8 2000h + formula

Figure 6-25. R5FSS_VIM_VEC_INT_j Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
VAL																														RESERVED	
R/W-0h																														R-0h	

LEGEND: R = Read Only; R/W = Read/Write; *-n* = value after reset

Table 6-68. R5FSS_VIM_VEC_INT_j Register Field Descriptions

Bit	Field	Type	Reset	Description
31-2	VAL	R/W	0h	These are the upper 30 bits of the 32-bit vector address associated with interrupt Q. It is the address that will be reflected in the R5FSS_VIM_IRQVEC or R5FSS_VIM_FIQVEC and the VECADDR output when interrupt Q is the active interrupt. Internally, these values are kept in a RAM. The FIQ and IRQ state machines have priority access to this RAM. Writes to this register will be piped internally, but further writes to the MMR interface may be stalled until this write has a chance to complete in the RAM. The new vector address will not take effect until this write completes to the RAM. In order to tell if this write has completed, software may read this register back. That read will not be able to complete unless the write has landed. Reads to this register will stall the MMR interface until the read is able to be completed at the RAM.
1-0	RESERVED	R	0h	Reserved. Read as 0. The lower 2 bits of the 32-bit vector address are always 0. (Vector addresses must be 32-bit aligned.)

Copyright © 2024 Texas Instruments Incorporated

6.3.2.14 R5FSS_RAT Registers

Table 6-70 lists the memory-mapped registers for the R5FSS_RAT. All register offset addresses not listed in Table 6-70 should be considered as reserved locations and the register contents should not be modified.

Table 6-69. R5FSS_RAT Instances

Instance	Base Address
MCU_R5FSS_RAT_CFG	40F9 0000h[(1)]
R5FSS_RAT_CFG	0FF9 0000h[(2)]

(1) MCU_R5FSS0 private memory-mapped register space. This region is only accessible by its associated R5F core; it is not accessible by any other SoC master. The base address is the same for each CPU view inside MCU_R5FSS0.
(2) R5FSS0/1 private memory-mapped register space. This region is only accessible by its associated R5F core; it is not accessible by any other SoC master. The base address is the same for each CPU view inside R5FSS0/1.

Table 6-70. R5FSS_RAT Registers (Part 1)

Offset	Acronym	Register Name	MCU_R5FSS_RAT_CFG Physical Address
0h	Section 6.3.2.14.1	Revision register	40F9 0000h
4h	Section 6.3.2.14.2	Config register	40F9 0004h
20h + formula	Section 6.3.2.14.3	Region control register	40F9 0020h + formula
24h + formula	Section 6.3.2.14.4	Region base register	40F9 0024h + formula
28h + formula	Section 6.3.2.14.5	Region translated lower address register	40F9 0028h + formula
2Ch + formula	Section 6.3.2.14.6	Region translated upper address register	40F9 002Ch + formula
804h	Section 6.3.2.14.7	Destination ID register	40F9 0804h
820h	Section 6.3.2.14.8	Exception logging control register	40F9 0820h
824h	Section 6.3.2.14.9	Exception logging header 0 register	40F9 0824h
828h	Section 6.3.2.14.10	Exception logging header 1 register	40F9 0828h
82Ch	Section 6.3.2.14.11	Exception logging data 0 register	40F9 082Ch
830h	Section 6.3.2.14.12	Exception logging data 1 register	40F9 0830h
834h	Section 6.3.2.14.13	Exception logging data 2 register	40F9 0834h
838h	Section 6.3.2.14.14	Exception logging data 3 register	40F9 0838h
840h	Section 6.3.2.14.15	Exception logging interrupt pending set register	40F9 0840h
844h	Section 6.3.2.14.16	Exception logging interrupt pending clear register	40F9 0844h
848h	Section 6.3.2.14.17	Exception logging interrupt enable set register	40F9 0848h
84Ch	Section 6.3.2.14.18	Exception logging interrupt enable clear register	40F9 084Ch
850h	Section 6.3.2.14.19	EOI register	40F9 0850h

Table 6-71. R5FSS_RAT Registers (Part 2)

Offset	Acronym	Register Name	R5FSS_RAT_CFG Physical Address
0h	Section 6.3.2.14.1	Revision register	0FF9 0000h
4h	Section 6.3.2.14.2	Config register	0FF9 0004h
20h + formula	Section 6.3.2.14.3	Region control register	0FF9 0020h + formula

Copyright © 2024 Texas Instruments Incorporated

Table 6-71. R5FSS_RAT Registers (Part 2) (continued)

Offset	Acronym	Register Name	R5FSS_RAT_CFG Physical Address
24h + formula	Section 6.3.2.14.4	Region base register	0FF9 0024h + formula
28h + formula	Section 6.3.2.14.5	Region translated lower address register	0FF9 0028h + formula
2Ch + formula	Section 6.3.2.14.6	Region translated upper address register	0FF9 002Ch + formula
804h	Section 6.3.2.14.7	Destination ID register	0FF9 0804h
820h	Section 6.3.2.14.8	Exception logging control register	0FF9 0820h
824h	Section 6.3.2.14.9	Exception logging header 0 register	0FF9 0824h
828h	Section 6.3.2.14.10	Exception logging header 1 register	0FF9 0828h
82Ch	Section 6.3.2.14.11	Exception logging data 0 register	0FF9 082Ch
830h	Section 6.3.2.14.12	Exception logging data 1 register	0FF9 0830h
834h	Section 6.3.2.14.13	Exception logging data 2 register	0FF9 0834h
838h	Section 6.3.2.14.14	Exception logging data 3 register	0FF9 0838h
840h	Section 6.3.2.14.15	Exception logging interrupt pending set register	0FF9 0840h
844h	Section 6.3.2.14.16	Exception logging interrupt pending clear register	0FF9 0844h
848h	Section 6.3.2.14.17	Exception logging interrupt enable set register	0FF9 0848h
84Ch	Section 6.3.2.14.18	Exception logging interrupt enable clear register	0FF9 084Ch
850h	Section 6.3.2.14.19	EOI register	0FF9 0850h

Copyright © 2024 Texas Instruments Incorporated

6.3.2.14.1 R5FSS_RAT_PID Register (Offset = 0h) [reset = 66801100h]

R5FSS_RAT_PID is shown in Figure 6-26 and described in Table 6-73.

Return to Summary Table.

This register contains the major and minor revisions for the module.

Table 6-72. R5FSS_RAT_PID Instances

Instance	Physical Address
MCU_R5FSS_RAT_CFG	40F9 0000h
R5FSS_RAT_CFG	0FF9 0000h

Figure 6-26. R5FSS_RAT_PID Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
REV																															
R-66801100h																															

Table 6-73. R5FSS_RAT_PID Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	REV	R	66801100h	TI internal data. Identifies revision of peripheral.

Copyright © 2024 Texas Instruments Incorporated

6.3.2.14.2 R5FSS_RAT_CONFIG Register (Offset = 4h) [reset = 300210h]

R5FSS_RAT_CONFIG is shown in Figure 6-27 and described in Table 6-75.

Return to Summary Table.

This register contains the configuration values for the module.

Table 6-74. R5FSS_RAT_CONFIG Instances

Instance	Physical Address
MCU_R5FSS_RAT_CFG	40F9 0004h
R5FSS_RAT_CFG	0FF9 0004h

Figure 6-27. R5FSS_RAT_CONFIG Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RESERVED								ADDR_WIDTH							
R-0h								R-30h							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADDRS								REGIONS							
R-2h								R-10h							

Table 6-75. R5FSS_RAT_CONFIG Register Field Descriptions

Bit	Field	Type	Reset	Description
31-24	RESERVED	R	0h	Reserved.
23-16	ADDR_WIDTH	R	30h	Number of address bits.
15-8	ADDRS	R	2h	Number of addresses.
7-0	REGIONS	R	10h	Number of regions.

Copyright © 2024 Texas Instruments Incorporated

6.3.2.14.3 R5FSS_RAT_CTRL_j Register (Offset = 20h + formula) [reset = 0h]

R5FSS_RAT_CTRL_j is shown in Figure 6-28 and described in Table 6-77.

Return to Summary Table.

This region controls the size and the enable for a region.

Offset = 20h + (j * 10h); where j = 0h to Fh.

Table 6-76. R5FSS_RAT_CTRL_j Instances

Instance	Physical Address
MCU_R5FSS_RAT_CFG	40F9 0020h + formula
R5FSS_RAT_CFG	0FF9 0020h + formula

Figure 6-28. R5FSS_RAT_CTRL_j Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EN	RESERVED														
R/W-0h	R-0h														
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
RESERVED										SIZE					
R-0h										R/W-0h					

Table 6-77. R5FSS_RAT_CTRL_j Register Field Descriptions

Bit	Field	Type	Reset	Description
31	EN	R/W	0h	Enable for the region.
30-6	RESERVED	R	0h	Reserved.
5-0	SIZE	R/W	0h	Size of the region in address bits. 0h = 1B 1h = 2B 2h = 4B ... 20h = 4GB

Copyright © 2024 Texas Instruments Incorporated

6.3.2.14.4 R5FSS_RAT_BASE_j Register (Offset = 24h + formula) [reset = 0h]

R5FSS_RAT_BASE_j is shown in Figure 6-29 and described in Table 6-79.

Return to Summary Table.

This register is used for the base address for a region. This is the source address for matching to a region.

Offset = 24h + (j * 10h); where j = 0h to Fh.

Table 6-78. R5FSS_RAT_BASE_j Instances

Instance	Physical Address
MCU_R5FSS_RAT_CFG	40F9 0024h + formula
R5FSS_RAT_CFG	0FF9 0024h + formula

Figure 6-29. R5FSS_RAT_BASE_j Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BASE																															
R/W-0h																															

Table 6-79. R5FSS_RAT_BASE_j Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	BASE	R/W	0h	Base address for the region. It must be aligned to the programmed size.

Copyright © 2024 Texas Instruments Incorporated

6.3.2.14.5 R5FSS_RAT_TRANS_L_j Register (Offset = 28h + formula) [reset = 0h]

R5FSS_RAT_TRANS_L_j is shown in Figure 6-30 and described in Table 6-81.

Return to Summary Table.

This register contains the translated lower address bits for a region.

Offset = 28h + (j * 10h); where j = 0h to Fh.

Table 6-80. R5FSS_RAT_TRANS_L_j Instances

Instance	Physical Address
MCU_R5FSS_RAT_CFG	40F9 0028h + formula
R5FSS_RAT_CFG	0FF9 0028h + formula

Figure 6-30. R5FSS_RAT_TRANS_L_j Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LOWER																															
R/W-0h																															

Table 6-81. R5FSS_RAT_TRANS_L_j Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	LOWER	R/W	0h	Translated lower address bits for the region. It must be aligned to the programmed size.

Copyright © 2024 Texas Instruments Incorporated

6.3.2.14.6 R5FSS_RAT_TRANS_U_j Register (Offset = 2Ch + formula) [reset = 0h]

R5FSS_RAT_TRANS_U_j is shown in Figure 6-31 and described in Table 6-83.

Return to Summary Table.

This register contains the translated upper address bits for a region.

Offset = 2Ch + (j * 10h); where j = 0h to Fh

Table 6-82. R5FSS_RAT_TRANS_U_j Instances

Instance	Physical Address
MCU_R5FSS_RAT_CFG	40F9 002Ch + formula
R5FSS_RAT_CFG	0FF9 002Ch + formula

Figure 6-31. R5FSS_RAT_TRANS_U_j Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																UPPER															
R-0h																R/W-0h															

Table 6-83. R5FSS_RAT_TRANS_U_j Register Field Descriptions

Bit	Field	Type	Reset	Description
31-16	RESERVED	R	0h	Reserved.
15-0	UPPER	R/W	0h	Translated upper address bits for the region.

Copyright © 2024 Texas Instruments Incorporated

6.3.2.14.7 R5FSS_RAT_DESTINATION_ID Register (Offset = 804h) [reset = 0h]

R5FSS_RAT_DESTINATION_ID is shown in Figure 6-32 and described in Table 6-85.

Return to Summary Table.

This register defines the destination ID value for error messages.

Table 6-84. R5FSS_RAT_DESTINATION_ID Instances

Instance	Physical Address
MCU_R5FSS_RAT_CFG	40F9 0804h
R5FSS_RAT_CFG	0FF9 0804h

Figure 6-32. R5FSS_RAT_DESTINATION_ID Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								DEST_ID							
R-0h																								R/W-0h							

Table 6-85. R5FSS_RAT_DESTINATION_ID Register Field Descriptions

Bit	Field	Type	Reset	Description
31-8	RESERVED	R	0h	Reserved.
7-0	DEST_ID	R/W	0h	Destination ID.

Copyright © 2024 Texas Instruments Incorporated

6.3.2.14.8 R5FSS_RAT_EXCEPTION_LOGGING_CONTROL Register (Offset = 820h) [reset = 0h]

R5FSS_RAT_EXCEPTION_LOGGING_CONTROL is shown in Figure 6-33 and described in Table 6-87.

Return to Summary Table.

This register controls the exception logging.

Table 6-86. R5FSS_RAT_EXCEPTION_LOGGING_CONTROL Instances

Instance	Physical Address
MCU_R5FSS_RAT_CFG	40F9 0820h
R5FSS_RAT_CFG	0FF9 0820h

Figure 6-33. R5FSS_RAT_EXCEPTION_LOGGING_CONTROL Register

31	30	29	28	27	26	25	24
RESERVED							
R-0h							
23	22	21	20	19	18	17	16
RESERVED							
R-0h							
15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
RESERVED						DISABLE_INTR	DISABLE_F
R-0h						R/W-0h	R/W-0h

Table 6-87. R5FSS_RAT_EXCEPTION_LOGGING_CONTROL Register Field Descriptions

Bit	Field	Type	Reset	Description
31-2	RESERVED	R	0h	Reserved.
1	DISABLE_INTR	R/W	0h	Disables logging interrupt when set.
0	DISABLE_F	R/W	0h	Disables logging when set.

Copyright © 2024 Texas Instruments Incorporated

6.3.2.14.9 R5FSS_RAT_EXCEPTION_LOGGING_HEADER0 Register (Offset = 824h) [reset = 0h]

R5FSS_RAT_EXCEPTION_LOGGING_HEADER0 is shown in Figure 6-34 and described in Table 6-89.

Return to Summary Table.

This register contains the first word of the header.

Table 6-88. R5FSS_RAT_EXCEPTION_LOGGING_HEADER0 Instances

Instance	Physical Address
MCU_R5FSS_RAT_CFG	40F9 0824h
R5FSS_RAT_CFG	0FF9 0824h

Figure 6-34. R5FSS_RAT_EXCEPTION_LOGGING_HEADER0 Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TYPE_F								SRC_ID																DEST_ID							
R-0h								R-0h																R-0h							

Table 6-89. R5FSS_RAT_EXCEPTION_LOGGING_HEADER0 Register Field Descriptions

Bit	Field	Type	Reset	Description
31-24	TYPE_F	R	0h	Type. 4h = RAT.
23-8	SRC_ID	R	0h	Source ID.
7-0	DEST_ID	R	0h	Destination ID.

Copyright © 2024 Texas Instruments Incorporated

6.3.2.14.10 R5FSS_RAT_EXCEPTION_LOGGING_HEADER1 Register (Offset = 828h) [reset = 0h]

R5FSS_RAT_EXCEPTION_LOGGING_HEADER1 is shown in Figure 6-35 and described in Table 6-91.

Return to Summary Table.

This register contains the second word of the header.

Table 6-90. R5FSS_RAT_EXCEPTION_LOGGING_HEADER1 Instances

Instance	Physical Address
MCU_R5FSS_RAT_CFG	40F9 0828h
R5FSS_RAT_CFG	0FF9 0828h

Figure 6-35. R5FSS_RAT_EXCEPTION_LOGGING_HEADER1 Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GROUP								CODE								RESERVED															
R-0h								R-0h								R-0h															

Table 6-91. R5FSS_RAT_EXCEPTION_LOGGING_HEADER1 Register Field Descriptions

Bit	Field	Type	Reset	Description
31-24	GROUP	R	0h	Group.
23-16	CODE	R	0h	Code. 1h = Boundary crossing error.
15-0	RESERVED	R	0h	Reserved.

Copyright © 2024 Texas Instruments Incorporated

6.3.2.14.11 R5FSS_RAT_EXCEPTION_LOGGING_DATA0 Register (Offset = 82Ch) [reset = 0h]

R5FSS_RAT_EXCEPTION_LOGGING_DATA0 is shown in Figure 6-36 and described in Table 6-93.

Return to Summary Table.

This register contains the first word of the data.

Table 6-92. R5FSS_RAT_EXCEPTION_LOGGING_DATA0 Instances

Instance	Physical Address
MCU_R5FSS_RAT_CFG	40F9 082Ch
R5FSS_RAT_CFG	0FF9 082Ch

Figure 6-36. R5FSS_RAT_EXCEPTION_LOGGING_DATA0 Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADDR_L																															
R-0h																															

Table 6-93. R5FSS_RAT_EXCEPTION_LOGGING_DATA0 Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	ADDR_L	R	0h	Address lower 32 bits.

Copyright © 2024 Texas Instruments Incorporated

6.3.2.14.12 R5FSS_RAT_EXCEPTION_LOGGING_DATA1 Register (Offset = 830h) [reset = 0h]

R5FSS_RAT_EXCEPTION_LOGGING_DATA1 is shown in Figure 6-37 and described in Table 6-95.

Return to Summary Table.

This register contains the second word of the data.

Table 6-94. R5FSS_RAT_EXCEPTION_LOGGING_DATA1 Instances

Instance	Physical Address
MCU_R5FSS_RAT_CFG	40F9 0830h
R5FSS_RAT_CFG	0FF9 0830h

Figure 6-37. R5FSS_RAT_EXCEPTION_LOGGING_DATA1 Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																ADDR_H															
R-0h																R-0h															

Table 6-95. R5FSS_RAT_EXCEPTION_LOGGING_DATA1 Register Field Descriptions

Bit	Field	Type	Reset	Description
31-16	RESERVED	R	0h	Reserved.
15-0	ADDR_H	R	0h	Address upper 12 bits.

Copyright © 2024 Texas Instruments Incorporated

6.3.2.14.13 R5FSS_RAT_EXCEPTION_LOGGING_DATA2 Register (Offset = 834h) [reset = 0h]

R5FSS_RAT_EXCEPTION_LOGGING_DATA2 is shown in Figure 6-38 and described in Table 6-97.

Return to Summary Table.

This register contains the third word of the data.

Table 6-96. R5FSS_RAT_EXCEPTION_LOGGING_DATA2 Instances

Instance	Physical Address
MCU_R5FSS_RAT_CFG	40F9 0834h
R5FSS_RAT_CFG	0FF9 0834h

Figure 6-38. R5FSS_RAT_EXCEPTION_LOGGING_DATA2 Register

31	30	29	28	27	26	25	24
RESERVED				ROUTEID			
R-0h				R-0h			
23	22	21	20	19	18	17	16
ROUTEID							
R-0h							
15	14	13	12	11	10	9	8
RESERVED		WRITE	READ	DEBUG	CACHEABLE	PRIV	SECURE
R-0h		R-0h	R-0h	R-0h	R-0h	R-0h	R-0h
7	6	5	4	3	2	1	0
PRIV_ID							
R-0h							

Table 6-97. R5FSS_RAT_EXCEPTION_LOGGING_DATA2 Register Field Descriptions

Bit	Field	Type	Reset	Description
31-28	RESERVED	R	0h	Reserved.
27-16	ROUTEID	R	0h	Route ID.
15-14	RESERVED	R	0h	Reserved.
13	WRITE	R	0h	Write.
12	READ	R	0h	Read.
11	DEBUG	R	0h	Debug.
10	CACHEABLE	R	0h	Cacheable.
9	PRIV	R	0h	Priv.
8	SECURE	R	0h	Secure.
7-0	PRIV_ID	R	0h	Priv ID.

Copyright © 2024 Texas Instruments Incorporated

6.3.2.14.14 R5FSS_RAT_EXCEPTION_LOGGING_DATA3 Register (Offset = 838h) [reset = 0h]

R5FSS_RAT_EXCEPTION_LOGGING_DATA3 is shown in Figure 6-39 and described in Table 6-99.

Return to Summary Table.

This register contains the fourth word of the data. Reading this register will clear the error pending bit.

Table 6-98. R5FSS_RAT_EXCEPTION_LOGGING_DATA3 Instances

Instance	Physical Address
MCU_R5FSS_RAT_CFG	40F9 0838h
R5FSS_RAT_CFG	0FF9 0838h

Figure 6-39. R5FSS_RAT_EXCEPTION_LOGGING_DATA3 Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																						BYTECNT									
R-0h																						R-0h									

Table 6-99. R5FSS_RAT_EXCEPTION_LOGGING_DATA3 Register Field Descriptions

Bit	Field	Type	Reset	Description
31-10	RESERVED	R	0h	Reserved.
9-0	BYTECNT	R	0h	Byte count.

Copyright © 2024 Texas Instruments Incorporated

6.3.2.14.15 R5FSS_RAT_EXCEPTION_PEND_SET Register (Offset = 840h) [reset = 0h]

R5FSS_RAT_EXCEPTION_PEND_SET is shown in Figure 6-40 and described in Table 6-101.

Return to Summary Table.

This register allows to set the exception pending signal.

Table 6-100. R5FSS_RAT_EXCEPTION_PEND_SET Instances

Instance	Physical Address
MCU_R5FSS_RAT_CFG	40F9 0840h
R5FSS_RAT_CFG	0FF9 0840h

Figure 6-40. R5FSS_RAT_EXCEPTION_PEND_SET Register

31	30	29	28	27	26	25	24
RESERVED							
R-0h							
23	22	21	20	19	18	17	16
RESERVED							
R-0h							
15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
RESERVED							PEND_SET
R-0h							R/W1S-0h

Table 6-101. R5FSS_RAT_EXCEPTION_PEND_SET Register Field Descriptions

Bit	Field	Type	Reset	Description
31-1	RESERVED	R	0h	Reserved.
0	PEND_SET	R/W1S	0h	Write a 1 to set the exception pending signal.

Copyright © 2024 Texas Instruments Incorporated

6.3.2.14.16 R5FSS_RAT_EXCEPTION_PEND_CLEAR Register (Offset = 844h) [reset = 0h]

R5FSS_RAT_EXCEPTION_PEND_CLEAR is shown in Figure 6-41 and described in Table 6-103.

Return to Summary Table.

This register allows to clear the pend signal.

Table 6-102. R5FSS_RAT_EXCEPTION_PEND_CLEAR Instances

Instance	Physical Address
MCU_R5FSS_RAT_CFG	40F9 0844h
R5FSS_RAT_CFG	0FF9 0844h

Figure 6-41. R5FSS_RAT_EXCEPTION_PEND_CLEAR Register

31	30	29	28	27	26	25	24
RESERVED							
R-0h							
23	22	21	20	19	18	17	16
RESERVED							
R-0h							
15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
RESERVED							PEND_CLR
R-0h							R/W1C-0h

Table 6-103. R5FSS_RAT_EXCEPTION_PEND_CLEAR Register Field Descriptions

Bit	Field	Type	Reset	Description
31-1	RESERVED	R	0h	Reserved.
0	PEND_CLR	R/W1C	0h	Write a 1 to clear the exception pending signal.

Copyright © 2024 Texas Instruments Incorporated

6.3.2.14.17 R5FSS_RAT_EXCEPTION_ENABLE_SET Register (Offset = 848h) [reset = 0h]

R5FSS_RAT_EXCEPTION_ENABLE_SET is shown in Figure 6-42 and described in Table 6-105.

Return to Summary Table.

This register allows to set the interrupt enable signal.

Table 6-104. R5FSS_RAT_EXCEPTION_ENABLE_SET Instances

Instance	Physical Address
MCU_R5FSS_RAT_CFG	40F9 0848h
R5FSS_RAT_CFG	0FF9 0848h

Figure 6-42. R5FSS_RAT_EXCEPTION_ENABLE_SET Register

31	30	29	28	27	26	25	24
RESERVED							
R-0h							
23	22	21	20	19	18	17	16
RESERVED							
R-0h							
15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
RESERVED							ENABLE_SET
R-0h							R/W1S-0h

Table 6-105. R5FSS_RAT_EXCEPTION_ENABLE_SET Register Field Descriptions

Bit	Field	Type	Reset	Description
31-1	RESERVED	R	0h	Reserved.
0	ENABLE_SET	R/W1S	0h	Write a 1 to set the exception interrupt enable signal.

Copyright © 2024 Texas Instruments Incorporated

6.3.2.14.18 R5FSS_RAT_EXCEPTION_ENABLE_CLEAR Register (Offset = 84Ch) [reset = 0h]

R5FSS_RAT_EXCEPTION_ENABLE_CLEAR is shown in Figure 6-43 and described in Table 6-107.

Return to Summary Table.

This register allows to clear the interrupt enable signal.

Table 6-106. R5FSS_RAT_EXCEPTION_ENABLE_CLEAR Instances

Instance	Physical Address
MCU_R5FSS_RAT_CFG	40F9 084Ch
R5FSS_RAT_CFG	0FF9 084Ch

Figure 6-43. R5FSS_RAT_EXCEPTION_ENABLE_CLEAR Register

31	30	29	28	27	26	25	24
RESERVED							
R-0h							
23	22	21	20	19	18	17	16
RESERVED							
R-0h							
15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
RESERVED							ENABLE_CLR
R-0h							R/W1C-0h

Table 6-107. R5FSS_RAT_EXCEPTION_ENABLE_CLEAR Register Field Descriptions

Bit	Field	Type	Reset	Description
31-1	RESERVED	R	0h	Reserved.
0	ENABLE_CLR	R/W1C	0h	Write a 1 to clear the exception interrupt enable signal.

Copyright © 2024 Texas Instruments Incorporated

6.3.2.14.19 R5FSS_RAT_EOI_REG Register (Offset = 850h) [reset = 0h]

R5FSS_RAT_EOI_REG is shown in Figure 6-44 and described in Table 6-109.

Return to Summary Table.

EOI Register.

The EOI register is used to re-trigger the pulse interrupt signal to ensure that any nested interrupt events are serviced. The software interrupt handler must write to the EOI register at the end of the current interrupt processing routine, so that new events can re-trigger the pulse interrupt signal again. For level interrupt signals the EOI register is not functional and must not be used.

Table 6-108. R5FSS_RAT_EOI_REG Instances

Instance	Physical Address
MCU_R5FSS_RAT_CFG	40F9 0850h
R5FSS_RAT_CFG	0FF9 0850h

Figure 6-44. R5FSS_RAT_EOI_REG Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																EOI_WR															
R-0h																R/W-0h															

Table 6-109. R5FSS_RAT_EOI_REG Register Field Descriptions

Bit	Field	Type	Reset	Description
31-16	RESERVED	R	0h	Reserved.
15-0	EOI_WR	R/W	0h	EOI value.

Copyright © 2024 Texas Instruments Incorporated

6.4 C71x DSP Subsystem

This section describes the C71x Digital Signal Processor Subsystem (C71SS) in the device.

6.4.1 C71SS Overview

The TMS320C71x is the next-generation fixed and floating-point DSP platform. The C71x DSP is a new core in the Texas Instruments' DSP family. The C71x DSP supports vector signal processing, providing significant lift in DSP processing power over a broad range of general signal processing tasks in comparison to the C6x DSP family. In addition, the C71x provides several specialized functions which accelerate targeted functions by more than 30 times. Besides expanding vector processing capabilities, the new C71x core also incorporates advanced techniques to improve control code efficiency and ease of programming such as branch prediction, protected pipeline, precise exception and virtual memory management.

The C71x builds on the C6x's legacy but it is not binary compatible with any previous C6x DSPs. Many of the C71x architecture features are new and unique. The C71x is designed as a platform with these characteristics:

- True 64-bit machine with 64-bit memory addressing (64-bit virtual memory addressing, 40-bit physical addressing) and single-cycle 64-bit base arithmetic operations
- Achieves 4 to 32 times DSP processing capacity compared to C66x
- Preserves out-of-the-box performance parity for C code which has been optimized for C66x with a few exceptions
- Provides a direct software migration path from C66x
- Provides direct architecture support for accelerating OpenCL

Some instances of C671SS are tightly coupled with a matrix multiply accelerator (MMA).

Note

The C71SS is also referred to as C71x CorePac.

Figure 6-45 shows an overview of the C71SS.

Copyright © 2024 Texas Instruments Incorporated

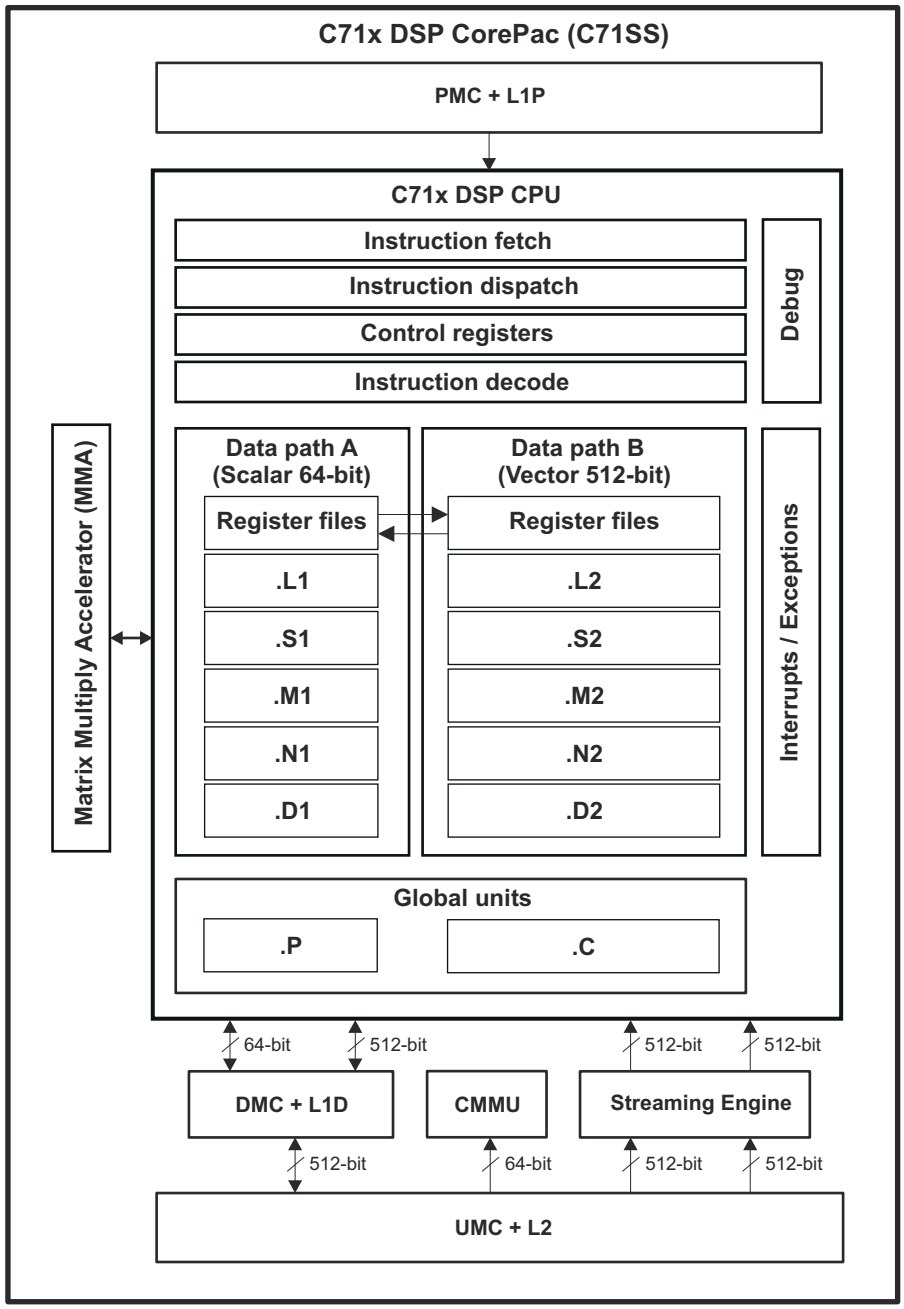


Figure 6-45. C71SS Overview

6.4.1.1 C71SS Features

The C71SS (C71x CorePac) supports the following key features:

- C7120 CPU
- Matrix multiplication accelerator (MMA) available on some instances
- Hierarchical cache memory system
  - Level 1 (L1):
    - L1 program memory controller (PMC) with associated L1 program memory (L1P)
    - L1 data memory controller (DMC) with associated L1 data memory (L1D), either L1 data cache or L1 scratch memory
  - Level 2 (L2):
    - L2 unified memory controller (UMC) with associated L2 memory, either L2 cache or L2 scratch memory
- Streaming engine (SE)

Copyright © 2024 Texas Instruments Incorporated

- CorePac memory management unit (CMMU), fully compliant with Armv8-A memory architecture
- Power/clock/reset management unit
- ECC/parity support
  - ECC protection of L2 memory and L1D memory
  - Parity protection of L1P memory
- PBIST and LBIST support
- Cluster Level Event Controller (CLEC) CPU interface
- C71x debug subsystem
- Extended configuration register (ECR) interface for register access (instead of MMR)

Copyright © 2024 Texas Instruments Incorporated

6.4.2 C71SS Functional Description

Note

This chapter provides only a brief description of C71SS supported features. Full description of C71x DSP functionality is provided in standalone TI documentation.

6.4.2.1 C71x DSP CPU

The C71x CPU is a true 64-bit core which provides the following key features:

- Instruction fetch unit
- Instruction dispatch unit, advanced instruction packing
- Instruction decode unit
- CPU dual datapath
  - One 64-bit scalar side (side A) and one 512-bit vector side (side B)
  - Side A includes the following functional units
    - Four main scalar processing units (.L1, .S1, .M1, .N1) capable of operating on up to 64-bit wide data
    - Two units (.D1, .D2) for address calculations, enabling parallel load/store operations
    - The following operations can be executed at the same time in a single clock cycle
      - One non-aligned 64-bit load or store operation
      - Two 64-bit arithmetic/logical operations (non-multiply arithmetic instructions)
      - One 128-bit multiply operation
  - Side B includes the following functional units
    - Five main vector processing units (.L2, .S2, .M2, .N2, .C) capable of operating on up to 512-bit wide vector data
    - A predication processing unit (.P) for vector predication
    - The following vector operations can be executed at the same time in a single clock cycle
      - One non-aligned 512-bit load or store operation
      - Two 512-bit arithmetic/logical operations (non-multiply arithmetic instructions)
      - One 1024-bit multiply operation
      - One 512-bit correlation operation or regular arithmetic operation
      - One vector predicate manipulation operation
  - Vector side B can perform up to 128×16-bit fixed-point multiply-accumulate (MAC), 4.0 times the MAC capacity compared to C66x
  - Vector side B can perform up to 80 single precision FLOPs/cycle or 32 double precision FLOPs/cycle, 5.0 times the floating point compared to C66x
  - The C71x CPU can load or store in parallel 64 bits and 512 bits of data per clock cycle, more than four times the bandwidth compared to C66x. In addition, a novel streaming data interface can read an additional 1024 bits of data per clock cycle, providing a total of 12 times more bandwidth compared to the C66x
  - Large set of machine registers:
    - Up to 16x512-bit global vector registers
    - Up to 16x64-bit global scalar registers
    - Local registers
  - Register file cross paths
    - Two cross paths, 64-bit each
    - Allow functional units from one data path to access a 64-bit operand from the opposite side global register file
    - The cross paths can not access the local register files directly
- CPU control logic
  - Support for two security states
    - Secure state: CPU can access both secure and non-secure space
    - Non-secure state: CPU can only access non-secure space, and cannot access secure space
  - Support for six privilege and execution levels for security and virtualization support with banked control registers for full isolation and protection

Copyright © 2024 Texas Instruments Incorporated

    - Secure root supervisor
    - Secure root user
    - Non-secure root supervisor
    - Non-secure root user
    - Non-secure guest supervisor
    - Non-secure guest user
  - Pipeline can operate in both unprotected and protected modes
    - Unprotected mode: Traditional C6x VLIW DSP operating mode with exposed instruction delay slots
    - Protected mode: Control code execution mode where instructions delay slots are not exposed
  - Scoreboarding mechanism to help hide memory system stalls and to allow the compiler to generate more efficient control codes
  - Supports recoverable interrupts and internal precise exception
    - No need to disable interrupts during multiple assignment code. In-flight results in multiple-assignment code are recorded by pipeline capture queues. These queues allow all programs to be interrupted at any arbitrary cycle, even during software pipelined loops, and then to be restarted correctly upon return from the interrupt or exception handler
    - Pipeline capture queues may be unloaded and reloaded, allowing the OS to swap out tasks regardless of whether the task was operating in protected or unprotected pipeline mode
    - Programmable event priority, hardware automatic event mask based on priority levels
    - Hardware performs automatic stacking at even handler entry and automatic unstacking at event exiting
  - Nested loop counters (NLC): Hardware mechanism to facilitate low-overhead loop collapsing by providing the computations associated with nested loop counters and predicates, avoid the needs of using explicit instructions
- Test, debug, and interrupt logic
- Enhanced instruction set architecture (ISA)
  - Single-cycle 64-bit arithmetic, logical, and shift instructions
  - Improvements to load/store instructions
    - Endian aware vector load instructions
    - Unpacking load / packing store instructions
    - Vector load/store with element reversal
    - Vector load with data duplications
  - Specialized instructions to speed up key benchmarks
    - Horizontal MAX/MIN search acceleration instructions
    - Horizontal ADD, SUB instructions
    - Dedicated FIR instructions
    - Sliding window sum of absolute differences (SAD)
    - Maskable complex dot products
  - Circular comparison instructions to accelerate Viterbi
  - New and improved atomic instructions to replace LL/SC/CL combination
  - Increases orthogonality between 32-bit, 64-bit, and vector operations compared to C66x
  - Supports binlog instruction for vision algorithms
  - Specialized instructions only allowed to execute at required privilege level and secure state
- C71x OpenCL features
  - Full compliance with IEEE 754 floating point standard
    - Support subnormal numbers
  - DVM support
    - Support fencing for uTLB transactions

The matrix multiply accelerator (MMA) is included as a special functional unit in the C71x CorePac and is tightly coupled to CPU operation.

6.4.2.2 C71x DSP Matrix Multiply Accelerator

The matrix multiply accelerator (MMA) provides the following key features:

- Support for fully connected layer using matrix multiply with arbitrary dimension
- Support for convolution layer using 2D convolution with matrix multiply with read panel
- Support for ReLU non-linearity layer OTF

Copyright © 2024 Texas Instruments Incorporated

- Support for high utilization (>85%) for typical convolutional neural network (CNN), such as AlexNet, ResNet, and others
- Ability to support any CNN network topologies limited only by memory size and bandwidth
- Coupled with C71x CPU using DSP chassis for data formatting
- MMA and C71x CPU are on a shared power domain
  - MMA cannot be independently powered off or clock gated at LPSC level (although it has extensive clock gating within the IP)

6.4.2.3 C71x DSP Cache Memory System

The C71x CorePac implements a hierarchical cache memory system with the following levels:

- Level 1 (L1):
  - L1 program memory controller (PMC) with associated L1 program memory (L1P)
  - L1 data memory controller (DMC) with associated L1 data memory (L1D)
- Level 2 (L2):
  - L2 unified memory controller (UMC) with associated L2 memory

6.4.2.3.1 C71x DSP L1 Program Memory

The L1P cache controller supports a fixed cache size of 32KB. The purpose of the L1P cache is to maximize performance of the code execution. L1P cache is necessary to facilitate fetching program code at a fast clock rate in order to maintain a large system memory. The cache is responsible for hiding the latency associated with executing code from the slower system memory.

The L1P memory system provides the following key features:

- L1 program memory controller (PMC) with 32KB L1P memory, all cache (no support for L1P SRAM)
- L1P cache
  - 4-way set associative
  - 64-byte line size
  - Virtually indexed, virtually tagged (48-bit virtual address)
  - Auto-prefetching on L1P misses from L2
- Prefetch and branch prediction
- ECC SECDED support
- Software initiated coherence operations
  - Single cycle invalidate with support for three modes: all cache lines, only user cache lines, only supervisor cache lines
- Virtual memory
  - Virtual-to-physical addressing on misses
  - Micro-TLB (uTLB) to handle address translation and for code protection
- Extended control register (ECR) access
  - L1P ECR registers are not memory mapped and instead are mapped to a MOVC CPU instruction

6.4.2.3.2 C71x DSP L1 Data Memory

The L1D includes a non-blocking cache controller, which has a main cache and a small fully associative victim cache. Main cache is read-allocate and supports write-back and write-through. Victim cache services read and write hit directly with no performance overhead as main cache.

The purpose of the L1D memory/cache is to maximize performance of the data processing. The cache is necessary to facilitate reading and writing data at the full CPU clock rate, while still having a large system memory. It is the cache's responsibility to hide much of the latency associated with reading from and writing to the slower system memory. The L1D memory includes two logical sections:

- L1D partition 0: Can support 32KB (max) / 8KB (min) of cache. Cache may only reside in partition 0. A cache size of 0KB is not supported, which means that disabling L1D cache functionality is not possible
- L1D partition 1: Can support 40KB (max) / 16KB (min) of SRAM (may include LUT and histogram)

Copyright © 2024 Texas Instruments Incorporated

Note

The minimum cache size (8KB) can only be achieved via L1DMODE register programming.

The L1D memory system provides the following key features:

- L1 data memory controller (DMC) with 48KB L1D memory, configurable as cache and/or SRAM
- L1D partition 0 (cache/SRAM) + L1D partition 1 (SRAM only)
- L1D cache
  - 32KB cache (max), configurable down to 8KB cache (min)
  - Dual datapath (DP0 + DP1) support
  - Direct-mapped (1-way set-associative) main cache
  - 128-byte cache line size
  - Read-allocate cache
  - Support for both write-back and write-through modes
  - Support for victim cache
    - 16 cache lines, fully associative
  - Physically indexed, physically tagged (40-bit physical address)
  - Support for speculative loads
  - Hit under miss
  - Posted write miss support
  - Write merging on all outstanding write transactions inside L1D
- L1D SRAM
  - 16KB (min), configurable up to 40KB (max)
  - Accessible from CPU or DMA
- Lookup table (LUT) and histogram
  - LUT
    - Up to 4 sets of look up tables can be specified simultaneously
    - Supports interpolation mode
    - Indices are unsigned values at 32-bit lanes of the source register
    - Look up table elements can be signed or unsigned, bytes, half-words or words
  - Histogram
    - Up to 4 sets of histograms can be specified simultaneously
    - Supports regular and weighted histogram operations
    - Indices are unsigned values at 32-bit lanes of the source register
    - Histogram weights are at 32-bit lanes of the source register
    - Histogram weights can only be signed bytes or signed half-words
    - Histogram bin data saturates to the minimum or maximum values of its data bin type
- Bandwidth
  - 1024-bit data throughput
  - 16×64-bit banks
- ECC SECDED support
- Coherence
  - Full MESI support
  - Support for global cache coherence operations
  - Snoops and cache maintenance operation support from L2
  - Snoops for L2 SRAM, MSMC SRAM and external (DDR) addresses
- Virtual memory support
  - Support for wider (40-bit) physical address
- ECR access
  - L1D ECR registers are not memory mapped and instead are mapped to a MOVC CPU instruction

6.4.2.3.3 C71x DSP L2 Memory

The L2 memory system provides the following key features:

- L2 unified memory controller (UMC) with 512KB L2 memory, configurable as cache and/or SRAM

Copyright © 2024 Texas Instruments Incorporated

- L2 cache
  - 8-way set-associative
  - 128-byte cache line size
  - Write-allocate cache
  - Support for both write-back and write-through modes
  - Physically indexed, physically tagged (40-bit physical address)
  - Supports 2×64-byte streams from one streaming engine
  - External MMR and MDMA accesses on an unified interface to MSMC
  - Caches MMU page tables
  - Cache pre-warming from SE and/or MSMC
- L2 SRAM
  - Security firewall on L2 SRAM accesses
  - DMA access to L2 SRAM on merged MSMC I/F
- Bandwidth
  - 2048-bit data throughput (see Section 6.4.2.4, *C71x DSP Streaming Engine*)
  - 4×512-bit banks, with 2 virtual banks each
- ECC SECDED support
- Coherence
  - Full MESI support
  - Support for global coherence operations
  - Snoops for L2 SRAM, MSMC SRAM and external (DDR) addresses
  - User coherence commands from SE
  - Full coherence between L1D cache, SE, L2 SRAM, MSMC SRAM and DDR
- ECR access
  - L2 ECR registers are not memory mapped and instead are mapped to a MOVC CPU instruction

6.4.2.4 C71x DSP Streaming Engine

The streaming engine (SE) supports the following key features:

- Provides a flexible, high bandwidth mechanism for reading large quantities of data into C71x CPU
- Supplies two 512-bit data streams (S0, S1) directly from L2 to the vector units
- Provides two stream address generators, supporting the two data streams
  - Flexible multi-dimensional address calculators
  - Provide offset addresses for load and store instructions
- Supports element promotion, decimation, duplication, transpose loads, predication
- LEZR feature can be enabled to return N number of zero vectors to CPU after selected dimension ends. LEZR and transposed mode are not supported together. LEZR is still supported when transposed mode is disabled..
- Provides 6D addressing
  - Access patterns up to 6D can be programmed ahead
  - 6D data is presented as 512-bit vector per cycle
- Communicates with L2 memory controller for requests beyond L2 (MSMC, DDR)
- Coherent with L1D data at stream open/close boundaries
- ECC SECDED support

The SE can sustain 1024-bits/cycle total bandwidth (512 bits/cycle on each of two streams) to the DSP CPU. When combined with its 512-bit/cycle vector load unit and 512-bit/cycle store unit, the DSP CPU can access 2048 bits/cycle of data.

6.4.2.5 C71x DSP CorePac Memory Management Unit

The CorePac Memory Management Unit (CMMU) extends the C71x architecture with support for address translation, access permission and protections and memory attributes determination and checking. It is implemented per C71x cluster as a two-level TLB structure. The CMMU works with the C71x L1 caches, stream buffers in each processor and CorePac memory system of CorePac cluster to translate virtual addresses to physical addresses, controls tablewalk hardware that accesses translation tables in main memory. The CMMU enables fine-grained memory system control through a set of virtual-to-physical address mappings and memory

Copyright © 2024 Texas Instruments Incorporated

attributes held in the L1 level micro translation look-aside buffer (uTLB) and CorePac cluster level translation look-aside buffers (TLBs).

The CMMU provides the following key features:

- Supports AArch32 LPAE and AArch64 page table format
  - Programmable levels of page table translation
  - 4KB, 16KB, 64KB translation granules
- Supports multiple page sizes: 4KB, 16KB, 64KB, 2MB, 32MB, 512MB, 1GB, 16GB
- Supports both hardware and software managed table walks
- Hierarchical Table Lookaside Buffer (TLB) caching page entries, intermediate page walk pointers for optimum address translation performance
  - Micro-TLB (uTLB) as first level TLB
    - 16-entry fully associative dual interface L1 data uTLB
    - 8-entry fully associative L1 instruction uTLB
  - CorePac (L2) TLB as second level TLB
    - 512-entry 4-way associative TLB cache, caching both instruction and data page translations with virtual-to-physical address mapping
    - Intermediate table walk caches to reduce memory access during multi-level page walk
    - Allows caching and sharing of intermediate table walks and page tables in the system cache hierarchy
- Security extensions that facilitate the development of secure applications
- Soft-error protection on all RAM and data storage structures

6.4.2.6 C71x DSP ECC Support

Table 6-110 provides the RAM ID for the C71x DSP ECC aggregator.

Table 6-110. RAM ID Map for C71x DSP ECC Aggregator

RAM ID	Endpoint
0	C71x Corepac
1	CBASS P2P
2	CBASS SCR
3	CBASS SCR
4	Reserved
5	Reserved
6	UMC pipe 0
7	UMC pipe 0
8	UMC pipe 1
9	UMC pipe 1
10	UMC pipe 2
11	UMC pipe 2
12	UMC pipe 3
13	UMC pipe 3
14	DMC
15	DMC tag RAM
16	SE, stream 0
17	SE, stream 1
18	PMC

6.4.2.7 C71x DSP Boot Configuration

In this device, the C71x boots from system memory instead of booting from ROM. A 1KB boot RAM (PSRAM1KECC) is used to store boot vectors for the C71x CPU.

Copyright © 2024 Texas Instruments Incorporated

6.4.2.8 C71x DSP Power-Up/Down Sequences

The power-up and power-down sequences for the C71SS can be found in *Power*.

6.4.2.9 C71x DSP Interrupt Control

The cluster level event controller (CLEC) serves the role of an interrupt controller for the C71x DSP. For more details, see *Interrupts*.

6.5 Graphics Accelerator (GPU)

This chapter describes the integration of the Graphics Processing Unit (GPU) subsystem in the device.

6.5.1 GPU Overview

The Graphics Processing Unit (GPU) accelerates 3-dimensional (3D), 2-dimensional (2D) graphics, and compute applications.

The GPU is based on the PowerVR® BXS 4-64 MC1 core from Imagination Technologies.

6.5.2 Features Supported

- Base architecture, fully compliant with the following APIs:
  - OpenGL ES 3.2
  - OpenCL 1.2 EP
  - Vulkan 1.2
- Tile-based deferred rendering architecture for 3D graphics workloads, with concurrent processing of multiple tiles
- Support for DRM security
- Support for GPU virtualization
  - Up to 8 virtual GPUs
  - Separate IRQ for each OSID
- Multi-threaded Unified Shading Cluster (USC) engine incorporating pixel shader, vertex shader and GP-GPU (compute shader) functionality
- USC incorporates an ALU architecture with high SIMD efficiency
- Fully virtualized memory addressing (up to 64 GB address space), supporting unified memory architecture
- Fine-grained task switching, workload balancing and power management
- Advanced DMA driven operation for minimum host CPU interaction
- 256KB System Level Cache (SLC)
- Specialized Texture Cache Unit (TCU)
- Compressed Texture Decoding
- Lossless data compression (PVRGC) - The PowerVR's geometry compression, which is performed in the Geometry Processing phase of the 3D graphics workload.
- Dedicated RISC-V processor for firmware execution
- Separate power island for the firmware processor for low latency power domain transitions
- On-Chip Performance, Power and Statistics Registers.
- Resolution Support
  - Frame buffer max size = 8K × 8K
  - Texture max size = 8K × 8K
- Anti-aliasing
  - Maximum 4× multisampling
- Performance
  - Floating Point Operations (F32) - 64 operations per clock
  - Floating Point Operations (F16) - 128 operations per clock
  - Texture performance - 4 texels per clock (@32 BPP)
  - Pixel performance - 4 pixel(s) per clock (@32 BPP)
  - Geometry Performance - 0.25 triangles per clock
  - Max performance
    - ~50 GFLOPS, 4 GTex/s or 4 GPix/s @ 800MHz

Copyright © 2024 Texas Instruments Incorporated

Unsupported Features:

- See the Module Integration section of this document for a list of module features not supported by the integration on this Device.

6.6 Video Accelerator

The following sections describe the video accelerator details for the device.

6.6.1 Introduction

The Video Accelerator is a 4K codec that supports both HEVC and H.264/AVC video formats. It provides high performance encode and decode capability up to 8bit 4K@60fps with a single-core architecture.

The Video Accelerator can encode and/or decode any resolution up to 8192 x 4320. It guarantees real-time performance for encoding/decoding 4K 60fps based on its sophisticated, latency tolerant hardware architecture. The Video Accelerator is highly optimized for memory bandwidth loading and excellent power management.

The Video Accelerator contains a 32-bit processor called V-CPU, which is responsible for parsing bitstream syntax in decoder orencoding bitstream syntax in encoder from sequence to slice header unit, pre-scanning slice data, controlling the underlying video hardware blocks called V-CORE. The V-CPU also communicates with host CPU through host register interface. The V-CORE performs actual processing of coded slice data: entropy decoding, inverse scan, inverse transform/quantization, motion compensation, and loop filtering in decoder and motion estimation, intra prediction, RDO, and entropy coding in encoder. This software and hardware combined architecture can provide flexibility and high throughput at the same time.

6.6.2 Features

6.6.2.1 Performance

H.265/HEVC Encoder

- Capable of encoding HEVC Main and Main Still Picture Profile @ L5.1 High tier
  - Maximum resolution: 8192x8192
    - The maximum resolution means the largest image size that the hardware can process with. Performance is not considered in this term.
  - Minimum resolution: 256x128
  - Constraints
    - A picture width shall be multiple of 8.
    - A picture height shall be multiple of 8.

Table 6-111. HEVC Encoder Performance

PicWidth	PicHeight	MHz/60fps 160Mbps
3840	2160	500

H.264/AVC Encoder

- Capable of encoding Baseline/Constrained Baseline/Main/High Profiles Level @ L5.2
  - Maximum resolution: 8192x8192
    - The maximum resolution means the largest image size that the hardware can process with. Performance is not considered in this term.
  - Minimum resolution: 256x128
  - Constraints
    - A picture width shall be multiple of 8.
    - A picture height shall be multiple of 8.

Table 6-112. H.264/AVC Encoder Performance

PicWidth	PicHeight	MHz/60fps 72Mbps
3840	2160	500

H.265/HEVC Decoder

Copyright © 2024 Texas Instruments Incorporated

- Capable of decoding HEVC Main and Main Still Picture Profile @ L5.1 High tier
  - Maximum resolution: 8192x4320
    - The Maximum resolution means the largest image size that the hardware can process with. Performance is not considered in this term.
  - Minimum resolution: 8x8

Table 6-113. H.265/HEVC Decoder Performance

PicWidth	PicHeight	MHz/60fps 160Mbps
3840	2160	450.00

H.264/AVC Decoder

- Capable of decoding Baseline/Constrained Baseline/Main/High Profiles @ L5.2
  - Maximum resolution: 8192x4320
    - The Maximum resolution means the largest image size that the hardware can process with. Performance is not considered in this term.
  - Minimum resolution: 32x32

Table 6-114. H.264/AVC Decoder Performance

PicWidth	PicHeight	MHz/60fps 72Mbps
3840	2160	450.00

6.6.2.2 Codec Related Features

H.265/HEVC Encoder

- Fully compatible with ISO/IEC 23008-2 high efficiency video coding main profile
- I/P slices
- CTU64
  - Supportable prediction unit (PU) size: 32 x 32, 16 x 16, 8 x 8
  - Supportable transform unit (TU) size: 32 x 32 to 4 x 4
- Parallel tools
  - Wavefront parallel processing (WPP) encoding with a single slice
  - Multi slice: Independent slice segment and dependent slice segment
- High performance offline CABAC encoding
- Motion estimation
  - 1/4-pel precision motion vectors
  - Search range [+/-128H, +/-64V] with an adaptive search center
  - Two reference frames for P-slice
  - Long-term reference for P picture
- Custom tuning tools
  - Custom Lambda map and lambda table
  - Custom mode decision
  - Fully programmable user scaling list
- In-loop filter
  - Deblocking filter
  - Sample adaptive offset (SAO)
  - Loop filtering across slices
- Strong intra smoothing on/off
- Transform skip
- Lossless coding
- Picture/CTU/subCTU level of rate control
- Region of interest (ROI) encoding with custom QP map
- 3DNR
- Adaptive intra refresh (AIR) for error resilience
- Support for YUV420 video format
- Support for YUV422 video format (only encoder)

Copyright © 2024 Texas Instruments Incorporated

H.264/AVC Encoder

- Compatible with the ITU-T recommendation H.264 specification. All coding tools in the baseline, con- strained baseline, main, and high profiles are supported.
  - With a few exceptions:
    - Interlaced coding tools are not supported.
    - FMO/ASO tool of H.264 is not supported.
- 16 x 16, 8 x 8 and 4 x 4 block sizes are supported and configurable.
- Motion estimation
  - 1/4-pel accuracy motion estimation with programmable search range up to [+/-64, +/-48]
  - One reference frame for P-slice
- Intra prediction
  - Luma I4 x 4 Mode: 9 modes
  - Luma I8 x 8 Mode: 9 modes
  - Luma I16 x 16 Mode: 4 modes (vertical, horizon, DC, plane)
  - Chroma mode: 3 modes (vertical, horizon, DC)
- Custom tuning tools
  - User-defined mode (skip, intra) map
  - User-defined QP map
  - Lambda tuning for custom mode decision
  - Fully programmable user scaling list
- In-loop deblocking filter
- CABAC/CAVLC support
- Error resilience tools:
  - Cyclic intra refresh (CIR)
  - Multi-slice structure
- A frame level and MB level of rate control
- Region of interest (ROI) encoding with custom QP map
- Support for YUV420 video format
- Support for YUV422 video format (only encoder)

H.265/HEVC decoder

- Fully compatible with ISO/IEC 23008-2 high efficiency video coding main/MSP (main still picture) profile. All coding tools in the profile are supported.
- I/P/B slices
  - All intra-prediction modes
  - All inter-prediction modes
- Variable CTU size: 64 x 64 to 16 x 16
  - Variable prediction unit (PU) size: 64 x 64 to 4 x 4
  - Variable transform unit (TU) size: 32 x 32 to 4 x 4
- Advanced motion vector prediction (AMVP) and merge mode
- A quarter motion compensation with 8 tap filters
- Uniform reconstruction quantization (URQ)
- Parallel coding tools
  - Multi tile
  - Wavefront parallel processing (WPP)-encoded bitstream support
  - Multi slice: Independent slice segment and dependent slice segment
- High performance CABAC decoding
- In-loop deblocking filtering
- Sample adaptive offset (SAO)
- Loop filtering across slice/tile boundaries
- Data reporting to the external host
- Robust error concealment
- Sequence change detection
- Support for YUV420 video format

Copyright © 2024 Texas Instruments Incorporated

H.264/AVC Decoder

- Fully compatible with the ITU-T recommendation H.264 specification. All coding tools in the baseline/constrained baseline/main/high profile are supported.
  - With a few exceptions:
    - Interlaced coding tools are not supported.
    - FMO/ASO tool of H.264 is not supported.
- Variable block size (16 x 16, 16 x 8, 8 x 16, 8 x 8, 8 x 4, 4 x 8 and 4 x 4)
- CABAC/CAVLC support
- In-loop deblocking filter
- Error detection, concealment and error resilience tools
- Support for YUV420 video format

6.6.2.3 Non-Codec Related Features

Rotation and mirroring

The Video Accelerator supports 8 modes of rotation and mirroring. The PRP block can do rotations of the source picture in 90, 180, or 270 degrees and vertical or horizontal flip before starting to encode the operation.

Bit-depth and chroma format conversion

The PRP block is responsible for source format conversion in the encoder IP. 422 (planar/semi-planar) to 420 conversion can be made before encoding operation.

Frame buffer compression

To overcome bandwidth suffering and at the same time to ensure real-time encode/decode performance, TI has carefully designed lossless, great access patterned frame buffer compression technology and employed it into the WAVE IP series. WAVE521CL achieves significant reduction in bandwidth while sustaining fast processing capability.

Long Burst Write

The Video Accelerator includes burst write back (BWB) module that enables 128-byte burst write access to external memory for better bus utilization. It collects a group of short write requests from loop filter block in 128-byte unit and sends it out to the external memory more efficiently. BWB works for an external Display module or other post processing module that needs to read decoded frames in raster scan order.

The FBC/FBD block also writes or reads 128-byte burst length of compressed frame data from/to the externalmemory.

3DNR

The Video Accelerator employs temporal noise reduction technology, also called 3DNR (3D noise reduction), for better quality of image under low light. Video noise appears easily in a low light level. It is an important issue in surveillance camera because images are captured often under dark conditions or indoors, which can make identification difficult.

The Video Accelerator can detect and filter noise for each color component Y, Cb, and Cr of a frame. This achieves both good video quality and enhancement of coding efficiency. Noise reduction is done while encoding. Therefore, additional read/write bandwidth is not required.

Note

3DNR is supported in the H.265/HEVC encoder only.

Latency tolerance

The Video Accelerator can afford to reach real time performance under memory access latency if the number of cycles per CTU retains less than 500 cycles. The Video Accelerator is designed to be less sensitive to pipeline delay, especially at a peak bitrate, which might eventually incur performance drop. With use of decoupling techniques and inter-pipe queues, it can hide this sort of delay and deliver high performance at any situation.

Copyright © 2024 Texas Instruments Incorporated

Subframe synchronization

The Video Accelerator supports subframe synchronization for low latency coding. By receiving dedicated subframe-ready signals from the image process unit (IPU) or by setting host interface registers from host CPU, the Video Accelerator can start encoding process as early as the minimum set of raw video data is ready.

Programmability

The Video Accelerator embeds the 32-bit CPU and 16-bit DSP dedicated to bitstream processing and their video hardware control. Host interface registers and interrupts are provided for communication between a host processor and the Video Accelerator.

Low Power Consumption

The Video Accelerator can ensure ultra-low power operation by gating the clock sources for some internal hardware blocks in an idle state.

Trick mode

The Video Accelerator supports trick mode such as fast/slow forward playback and fast/slow reverse playback for video applications. By seeking an intra random access point (IRAP) picture as entry points of bitstream, the Video Accelerator can skip non-IRAP pictures and start decoding from an IRAP picture or decode a thumbnail of a picture.

Frame-based processing

V-CPU processor operates video operation on a frame by frame basis. While frame operation is running, there is no need for communication between the host processor and the Video Accelerator. This is the key feature for promising the lowest burden to host processor for video operations.

By issuing a picture processing, the host application can perform its own operations until it receives an interrupt from the Video Accelerator informing of the completion of picture processing.

Handling multi-instances

The Video Accelerator supports multiple instances, which can be helpful for multi-channel encoder and/or decoder applications. To support this multi-instance operation, the Video Accelerator uses an internal context parameter set for each instance. While creating a new instance and starting picture processing, a set of these context parameters is created and updated automatically in the Video Accelerator. Because of this internal context management scheme, different encoder and/or decoder tasks running on the host processor can control Video Accelerator operations independently with their own instances.

When creating a new instance, an application task will get a new handle specifying an instance if a new handle is available on the Video Accelerator. All of the following operations for the given application task could be separately handled on the Video Accelerator by using this task-specific handle. Because the Video Accelerator can only perform one picture processing task at a time, each application task should check whether the Video Accelerator is ready or not before starting a new picture operation. By calling a function for closing a certain instance, the application can easily terminate a single task of video operation on the Video Accelerator.

6.6.3 Block Diagram

Figure 6-46 shows the hierarchical architecture of the Video Accelerator and external bus interfaces.

Copyright © 2024 Texas Instruments Incorporated

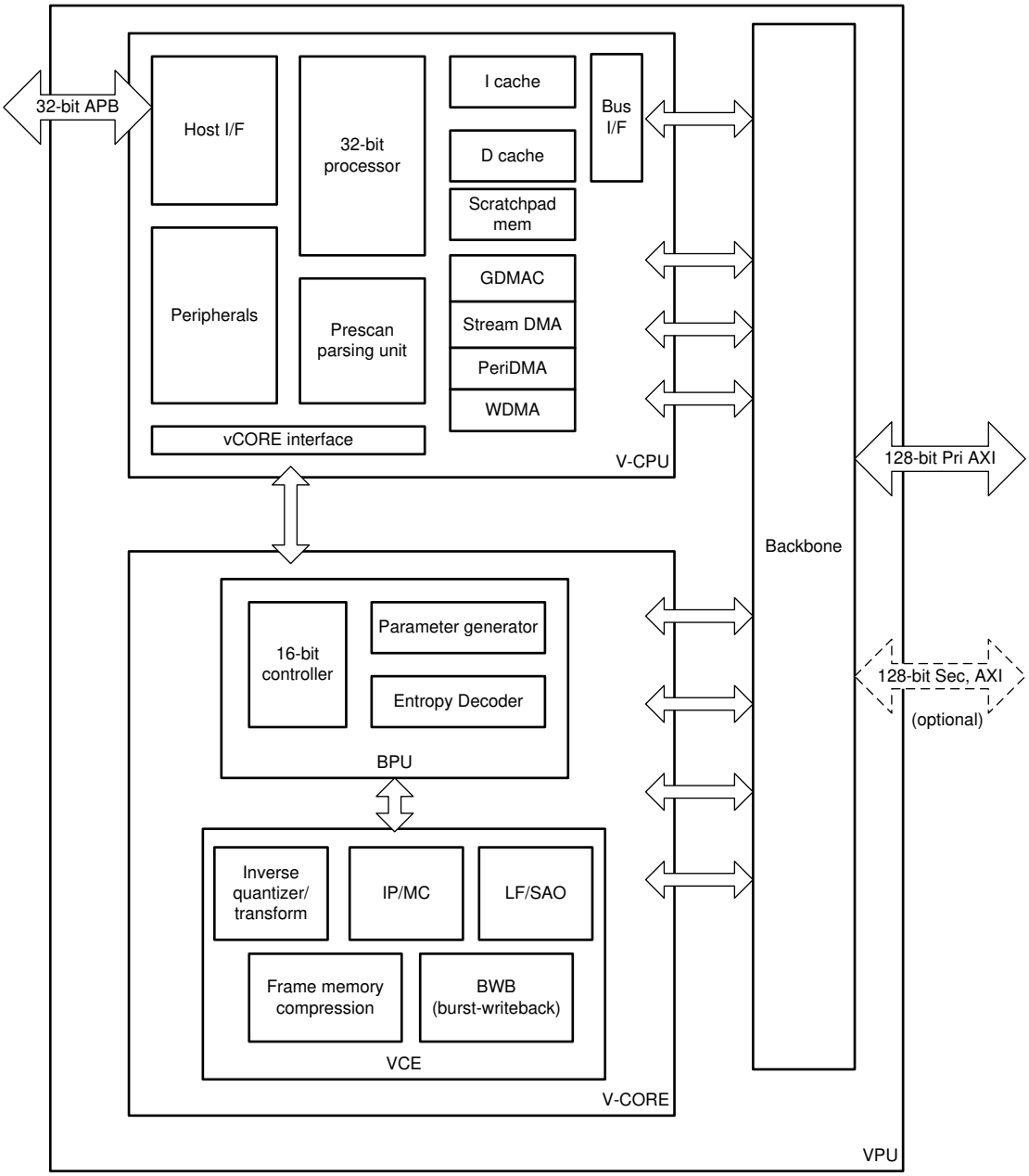


Figure 6-46. Video Accelerator Block Diagram

The Video Accelerator is built on a layer structure for efficient video processing. It is composed of two main parts, V-CPU and V-CORE. V-CPU is a 32-bit processor where the Video Accelerator L1 driver software is running. It mainly encodes and decodes a high-level syntax of bitstream - SPS/PPS/SH, and communicates with the host processor, such as receiving a picture level command from the host processor or sending the result of video task through the 32-bit AMBA3 APB bus. V-CPU can also parse necessary parameters and give necessary slice/tile-level information to the underlying V-CORE hardware.

Particularly in case of an HEVC decoder, V-CPU uses a virtual wavefront parallel processing (VWPP) scheme to balance processing load among V-COREs and to simplify its operation. During the VWPP, V-CPU parses bitstream and extracts some data from bitstream such as CABAC context, QP, and start address for each row and tile. After that, V-CPU reorders the data to meet the WPP processing sequence and dispatches them to the appropriate V-COREs.

V-CORE can encode and decode a slice unit of data. It consists of a 16-bit DSP called a bit processor unit (BPU) and a group of video codec hardware blocks called a video codec engine (VCE).

Copyright © 2024 Texas Instruments Incorporated

- BPU is responsible for packing and parsing of coded slice data, CTU/CU/PU/TU-level parameter derivation, and finally passing slice data over to the VCE. To speed up entropy encoding/decoding, some hardware accelerators are included in the BPU.
- On the other hand, VCE is an actual hardware core executing a series of decoding process - encoding and decoding process - motion estimation, intra-prediction, RDO, and entropy coding with their hardwired logics, motion compensation, inverse-transformation/quantization, and loop filter with their hardwired logics. HEVC and H.264/AVC decoders share most of the internal VCE memories and full logics in certain VCE blocks like intra-prediction for small gate count.

They are doing these tasks with CTU-based pipelines and FIFO queues to ensure real-time speed and performance. When it comes to system connection as shown in the Figure 6-46, there is one 32-bit AMBA3 APB interface connecting to the host processor, and there are two 128-bit AMBA3 AXI bus interfaces connecting to the external memory controller for reading and writing picture data and temporal data.

- Primary AXI: AXI interface to read or write work data and coded picture (bitstream), decoded picture data and so forth.
- Secondary AXI (optional): On-chip-SRAM connected AXI interface to read or write temporal buffer required for V-CORE operation. This is an optional interface that can be used to alleviate bandwidth usage.

Copyright © 2024 Texas Instruments Incorporated

6.7 Vision Pre-processing Accelerator (VPAC)

This chapter describes the Vision Pre-processing Accelerator (VPAC) in the device.

6.7.1 VPAC Overview

The Vision Pre-processing Accelerator (VPAC) subsystem is a set of common vision primitive functions, performing pixel data processing tasks, such as: color processing and enhancement, noise filtering, wide dynamic range (WDR) processing, lens distortion correction, pixel remap for de-warping, on-the-fly scale generation, on-the-fly pyramid generation. The VPAC offloads these common tasks from the main SoC processors (ARM, DSP, etc.), so these CPUs can be utilized for differentiated high-lelvel algorithms. The VPAC is designed to support multiple cameras by working in time-multiplexing mode. The VPAC works as a front end to vision processing pipeline and provides for further processing by other vision accelerators or processor cores inside the SoC. The VPAC also includes an imaging pipe, which can be integrated on-the-fly with external camera sensor, as well as does memory-to-memory (M2M) processing on pixel data.

Figure 6-47 provides an overview of the VPAC subsystem.

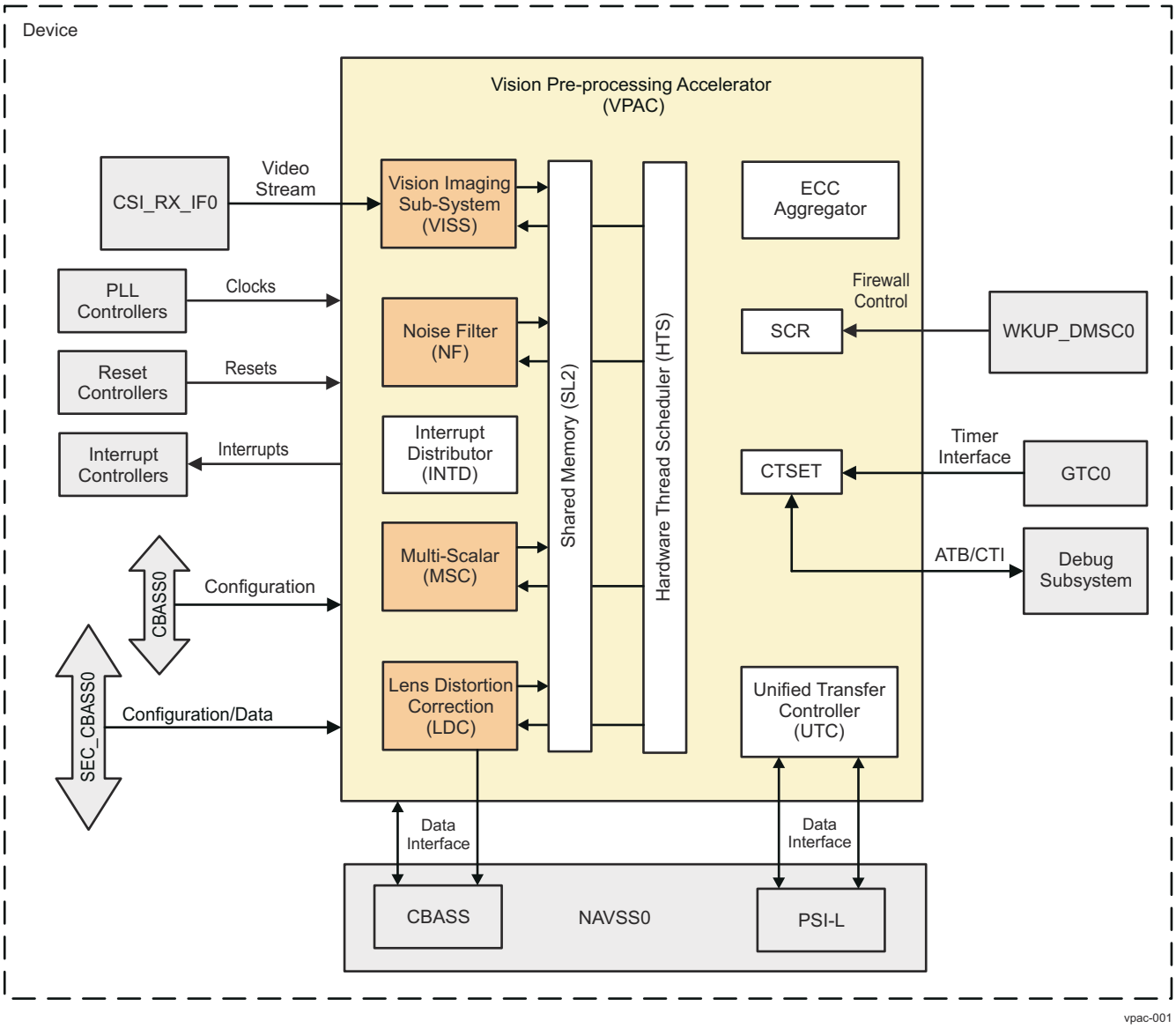


Figure 6-47. VPAC Overview

Copyright © 2024 Texas Instruments Incorporated

6.7.1.1 VPAC Features

The VPAC includes the following processing and infrastructure sub-modules:

- Vision Imaging Sub-System (VISS). The vision imaging pipe does image processing on raw data which includes Wide Dynamic Range (WDR) merge, Defect Pixel Correction (DPC), Lens Shading Correction (LSC), Global/Local Brightness and Contrast Enhancement (GLBCE), demosaicing, color conversion, and Edge Enhancement (EE). It can operate on sensor data either on-the-fly or in memory-to-memory mode. The VISS provides the following main features:
  - Performance is up to 1 pixel/cycle
  - Up to 16-bit input RAW formats (8b, 12b, 14b, 16b)
  - Support for concurrent 12-bit and 8-bit YUV output
  - Support for generic 2x2 RAW format (Bayer, RCCB, RCCC etc.)
  - Support for other color spaces:
    - RGB output
    - Support for color saturation and gray scale for HSV/HSL, etc. (Hue is not supported)
  - Support for multiple WDR/HDR formats:
    - Combined Companded format (up to 16-bit companded)
    - Separated Exposures – up to 3 frame Motion Adaptive HDR Merge
    - Staggered HDR
    - Digital Lateral Overlap (DLO)
  - Support of statistics for Auto-Exposure/Auto-White-Balance or Auto-Focus (configurable option to select either AE/AWB or AF)
  - Advanced Spatial Noise filter with 16-bit support
  - Support for Flexible CFA for generating multi-plane output for any 2x2 RAW format sensor configuration.
    - Supports RCBC, RCCC, Bayer, RGBC as well as other formats
    - Can generate up to 4 independent color planes
    - Support for up two 3 directions per color plane
    - Adaptive threshold based on intensity measure
  - Support for multiple color format output generation
    - Supports traditional 12-bit YUV output
    - Supports RGB output (8-bit only)
    - Supports saturation outputs (8 bits only)
    - Supports Grayscale/Luma/IR/Clear output
  - Support for flexible output format generation including YUV as well as custom
  - Global/Local Brightness and Contrast Enhancement module
    - Up to 4096 pixels/line
    - Local and Global tone mapping (16-bit)
  - Spatial Noise Filter (NSF4V)
    - Supports raw input image in generic 2x2 color pattern format including Bayer pattern
    - Supports 16-bit raw input image
    - Spatially adaptive noise level threshold
    - Configurable soft or hard thresholding
    - Supports elliptical lens shading gain adjustment of adaptive noise threshold with 2 sets of independent settings
    - White Balance gain
    - Automatic image border mirroring
    - Signal level detection with flexible cross-color-channel or independent averaging
  - Option to bypass visual enhancement functions (NSF4V and GLBCE)
  - Edge Enhancement (EE)
    - Enhance the visual quality of the image (Luma only)
    - Edge enhancer filter and the edge sharpener filter can be combined by configuration
    - Option to bypass visual enhancement functions (CAC, NSF4V, and GLBCE)
  - Dynamic White balance gain
  - Linear domain Histogram with multiple ROI
  - Support concurrent Human Vision and Machine Vision tunable FCP instances

Copyright © 2024 Texas Instruments Incorporated

    - Configurable tapping of RFE, NSF4, or GLBCE output for MV path
  - Support Lateral Chromatic Aberration Correction in RAW domain
    - CAC modules uses +-4 lines for correction and placed between RAWFE and NSF4v
  - CFA (2x2 generic RAW format support) :16b pipeline
    - 16□24b decompanding on each filter output (all 4)
    - 24b CCM block
    - 24 □12 companding block on CFA output
- Lens Distortion Correction (LDC) block. The LDC engine is YUV domain processor designed to perform perspective and geometric transforms. This can be used for creating several effects, including but not limited to:
  - Lens Distortion Correction, including Fish Eye Lenses
  - Epipolar rectification for Stereo
  - Generic perspective transforms for rotation scaling or Augmented Reality like effects

  The output of the LDC block can be sent to external memory (DDR) or sent to other VPAC sub-module (Scalar, Noise filter) for further pre-processing via local shared memory (SL2). The LDC block provides the following main features:

  - Performance up to 1 cycle/pixel (bilinear) and 2 cycles/pixel (bicubic)
  - Autonomous memory-to-memory operation
  - Tile based processing
  - Generic mesh based distortion model to correct multiple distortion types
  - Perspective warp transformation for perspective correction; affine transform is a subset of perspective warp and supports scaling and rotation
  - Supported formats (see Section 6.7.2.9):
  - Supports up to 8192 x 8192 image dimension
  - Pixel Interpolation
    - Bicubic interpolation for Y and bilinear interpolation for Cb/Cr
    - Bilinear interpolation mode to offer double throughput
  - Support to process either Luma only or Chroma only modes in YUV420 input data format to save bandwidth
  - Supports variable block size frame processing (9 regions)
  - Data formats with flexible independent data format between Luma and Chroma
  - Support for direct connection with other VPAC HWA on LDC input along with hybrid addressing
    - Mix of circular and linear buffer on pixel data with support for initial line skip
- Multi-Scalar (MSC) block. The MSC block reads data from memory (DDR or on-chip) to shared memory (SL2) and generates up to 10 scaled outputs from a given input with varios scaling ratios (between 1x and x0.25). The output of the MSC block can be sent to external memory (DDR) from the local shared memory (SL2), or can be further noise filtered using the VPAC Noise Filter (NF) block. The MSC supports two independent threads (scalars), which combined can generate up to max 10 scales. The MSC block provides the following main features:
  - Multi-scaling capable: 10 simultaneous scaled outputs from 1 or 2 input planes; up to 2 simultaneous processing thread with total of 10 scaled output with 1 to N and 1 to M configurations (where N+M <=10)
  - Each scaling engine can be configured to perform Pyramid or inter-octave scale generation
  - Scaling ratio supported: 1x ~ x/4 (with 1x scaling, MSC can perform a 3~5-tap generic separable convolution filtering)
  - 5-tap separable filter kernel implementation
  - Programmable kernel sizes for vertical/horizontal filter (3, 4, or 5)
  - Poly phase implementation with support of 64 or 32 phases
  - 4 sets of 5-tap x 32 phase coefficients (two can be combined to support 5-tap x 64 phase coefficients and one of set can hold additional 5-tap customer filter coefficients for non-resizing filter)
  - 2 additional sets of 5-tap Gaussian filter coefficients (single-phase) dedicated for Pyramid (Octave) generation
  - Coefficients in 10-bit (S10Q8 format - signed value of 10 bits with 8-bit fraction)
  - Separate initial horizontal and vertical phase programming option
  - Support for two planes of data processing
    - Data in a plane can be uniform (for example, Y plane) or interleaved (Cb/Cr interleaved)

Copyright © 2024 Texas Instruments Incorporated

  - 8-bit or 12-bit component data
  - Output data rate is dependent on downscaling ratio
  - Programmable input and output ROI (or clipping) for each scaler
  - Support for padding (replication)
  - On-the-fly (OTF) or memory-to-memory (M2M) operation with line buffers in SL2 (frame resolution support is not constrained by MSC line buffer)
  - 2 thread with each thread capable of supporting both luma+Chroma channels
  - Support 8b/12b YUV 422
  - Independent parameter on Luma and Chroma channel in YUV 420 (i.e Y12 + UV8) Y1+Y2
- Noise Filter (NF). The NF block reads data from memory (DDR or on-chip) to shared memory (SL2) and does Bilateral filtering to remove noise. The output of the NF block can be sent to external memory (DDR) from shared memory (SL2) or can be further re-sized using the MSC block. The NF block provides the following main features:
  - Support for bilateral and general filtering
  - Supports filter size up to 5x5
  - Supports true 2D bilateral filtering
  - Supports filter size up to 5x5 of programmable static weights
  - LUT based bilateral weights generation (8-bit for weight)
  - Supported formats: one plane (interleaved plane and non-interleaved) YUV 420, 12-bit
  - Performance up to 1 pixel/cycle
  - Support for ROI: both input and output can be handled via the local Unified Transfer Controller (UTC)
- Hardware Thread Scheduler (HTS-V2). The HTS block used for inter-processor communication among various VPAC sub-modules (VISS, LDC, Scalar, and NF) and with the local DMA Engine (UTC). The HTS block provides the following main features:
  - Scheduling the HWA threads and associated tasks
  - Enables rate-controlled task execution
  - Support running several independent threads asynchronously in the sub-system
  - Debug functions:
    - Halting HWA threads at logical task boundary
    - Handling pipeline suspend and subsequent abort by software intervention
    - Revision2 of HTS
    - Dynamic threshold with SKIP and LDC hybrid addressing support added for Flexconnect feature
- Shared Level 2 (SL2) memory. The SL2 is used as intermediate storage for Hardware Accelerator (HWA) to exchange data across VPAC sub-modules (that is, VISS, LDC, Scalar, and NF) and from DDR/System Memory. VPAC supports Realtime and Non-Realtime processing threads simultaneously.
- DMA Architecture:
  - TR pushed via UDMA-C in NAVSS onto PSIL of UTC (VPAC). Local trigger to initiate transfer is generated by HTS. Block copy data movement happens on VBUSM interface.
  - Two UTC instances inside VPAC: Real-Time traffic UTC0 (32 channels) and Non-Realtime traffic UTC1 (64 channels).

Unsupported Features:

- See the Module Integration section of this document for a list of module features not supported by the integration on this Device.

Copyright © 2024 Texas Instruments Incorporated

6.7.2 VPAC Subsystem Level

This section covers VPAC subsystem-level details.

6.7.2.1 VPAC Subsystem Block Diagram

The VPAC subsystem includes imaging pipe to do image processing on raw pixels (bayer, RCCC, etc.). The VPAC subsystem loads pixel data either from memory (for example, DDR or on-chip memory), or captured from camera sensor (via external module on SoC level), and applies many vision primitive functions such as image processing, scaling to generate image pyramids, noise filtering, correcting lens distortion, applying perpective transformation for stereo rectification, etc. Optionally, the VPAC subsystem can also do dynamic range management on received image data. The output image processed data generated by the VPAC subsystem is written into memory (for example, DDR or on-chip). The VPAC subsysem supports software managed and HTS controlled flexible load/store of data using K3 DMA infrastructure sub-modules.

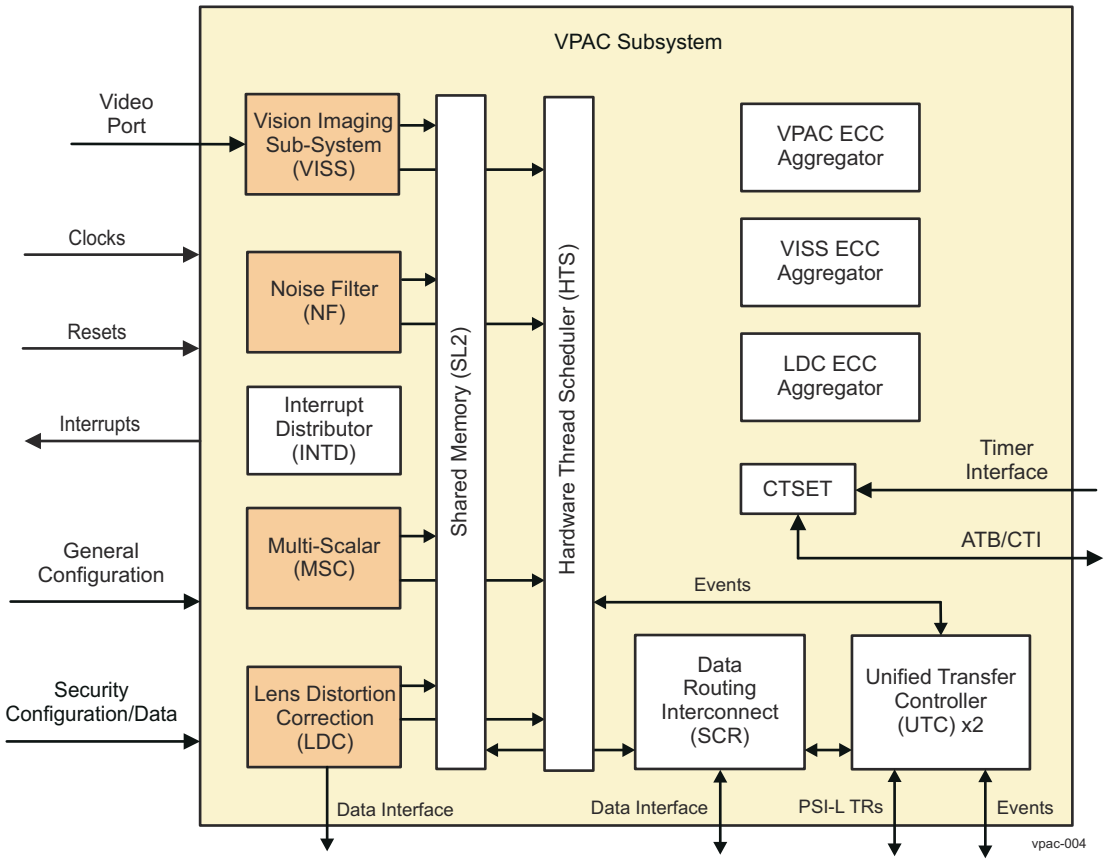


Figure 6-48. VPAC Subsystem Block Diagram

The VPAC subsystem includes the following image processing harware accelerators (HWAs) and infrastructure blocks:

- Vision Imaging Subsystem (VISS). The VISS does on-the-fly (OTF) processing on raw pixels captured from camera sensor. It also does memory to memory processing for data captured from other sources ot its parallel port or video port. The VISS uses VPAC infrastructure modules (HTS, DRU/UTC, interrupt aggregator, etc.) for flow and data management. For more information on the VISS operation and the sub-modules integrated inside, see Section *VPAC Vision Imaging Subsystem (VISS)*. The VISS consists of following Hardware Accelerators (HWA):
  - RAW Front End (RFE) block: The RFE block does RAW pixel (that is, Bayer, RCCC, RGBW, etc.) processing on captured image data from sensor. The processed data is later passed on to the NSF4V block, or otherwise it is passed onto the Flexible Color Proccesing (FCP) block for demosaicing and color conversion. The RFE includes also the H3A block, which supports the control loops for auto focus (AF), auto white balance (AWB) and auto exposure (AE) by computing image statistics.

Copyright © 2024 Texas Instruments Incorporated

  - Noise Filter Engine (NSF4v): The NSF4v is a spatial, 3-level wavelet based noise filter engine, supporting generic 2x2 sensor. It receives data from the RFE block and sends it to the GLBCE module.
  - Global Local Brightness Contrast Enhancement (GLBCE) module: The GLBCE module is used for dynamic range control within image for visual quality. If a contrast enhancement on the input image is required for visual quality, the RFE output is processed by the NSF4v and GLBCE blocks and then passed onto the FCP block. Otherwise, if contrast enhancement is not required, the NSF4v and GLBCE blocks can be bypassed, and the RFE output is directly provided to the FCP block.
  - Flexible Color Processing (FCP) block: The FCP block receives data from GLBCE and does demosaicing and color conversion. The output of the color processing is sent to the internal EE (Edge Enhancer) block, if it is enabled, before being ouput to VPAC shared memory. Otherwise, the FCP output data is directly sent to VPAC shared memory to be written into external memory for further vision processing by programmable processors (that is, DSP or Arm), or other vision hardware accelerators at SoC level.
  - Load/Store Engine (LSE): The LSE is an infrastructure block, which provides the following functions:
    - Receives data, captured through MIPI CSI-2 image sensor (by external module at SoC level), for on-the-fly image processing to reduce DDR bandwidth and latency. LSE also provides horizontal and vertical blanking cycles to allow core data path to settle at proper boundary of line and frame.
    - Provides access to VPAC SL2 memory for loading/storing data. Loaded data from SL2 are passed on to unpacker function of LSE for RFE. Packed data after FCP/H3A processing is written into SL2.
    - Pixel packing/unpacking. Source data (512-bit) for RFE processing is loaded from SL2 and passed onto unpacker function for pixel extraction. Extracted pixels are driven on the video port of RFE. Similarly, FCP produced pixels are driven to packer function for eventual write into SL2. H3A generated data is pseudo mapped as pixel of 32 bits for packing purpose and directly driven by RFE.
    - Event control. HTS events are generated at line level. These events are routed to RFE to start the processing. For each consumed line HTS needs *task done* indication. For some initial lines consumed, there would not be any valid data output. Similarly, H3A generated data will be at paxel/window height number of lines. For consumed lines, which is not producing any valid data to be written into DDR, HTS needs to be indicated with separate mask bits for each output streams. For initial lines, when there is no valid output due to delay lines, LSE will still generates mask output to HTS indicating lack of proper output data.
- Lens Distortion Correction (LDC) module. The LDC module deals with lens geometric distortion issues in the camera system. The distortion can be common optical distortions, such as barrel distortion, pin-cushion distortion, or fisheye distortion. Correction is not limited to just these types of distortions. The LDC module consists of a Back Mapping block (which gives coordinates of the distorted image as a function of coordinates of the undistorted output image), a delta_x/delta_y offset table, frame buffer inetrface, buffer, an interpolation block, and SL2 interface. The frame buffer is external to the LDC module, and is usually in an off-chip SDRAM. The LDC module uses the common VPAC infratructure modules (HTS, DRU/UTC, interrupt aggregator, etc.) for flow and data management, except for loading input data for which it has its own DMA engine. For more details on the LDC operation, see Section *VPAC Lens Distortion Correction (LDC)*.
- Multi-Scaler (MSC) module. The MSC consists of 10 programmable resizers performing multi-thread/multi-scaling operations (2-input to N-outputs, and 2-input to M-outputs, where N+M is 10 or less). Each processing thread of MSC reads its input plane data from the SL2 circular line buffer, performs multi-scaling operations (ratios between X and 0.25X) on the same input, and writes out results to SL2 circular line buffers. In case of on-the-fly operation, the source data is generated from another VPAC HWA. In case of memory-to-memory operation, the source data is read from the external memory. Data transfers from/to SL2 to/from external memory (or another HWA) are handled by the VPAC DMA controller (DRU/UTC), with transfer request events coming from the VPAC HTS module. The MSC module uses the VPAC infrastrucutre modules (HTS, DRU/UTC, interrupt aggregator, etc.) for flow and data management. For more details on the the MSC operation, see Section *VPAC Subsystem Multi-Scaler (MSC)*.
- Noise-filter (NF). The NF module reads data from memory (that is, DDR or on-chip) to the shared memory (SL2) and does bilateral filtering to remove noise. The output of the NF block can be sent to external memory (that is, DDR) from the shared memory (SL2) or can be further re-sized using the MSC module. The NF module supports two modes of filtering: bilateral filtering mode (where the overall weights are 8-bit unsigned weights computed based on center pixels), and generic filtering mode (where all weights are 9-bit signed value read LUT). The NF module uses the VPAC infrastrucutre modules (HTS, DRU/UTC, interrupt aggregator, etc.) for flow and data management. For more details on the NF operation, see Section *VPAC Nosise Filter (NF)*.

Copyright © 2024 Texas Instruments Incorporated

- Hardware Thread Scheduler (HTS). The HTS module is a messaging layer for low-overhead synchronization of the parallel computing tasks and DMA transfers, and is independent from the host processor. It allows autonomous frame level processing for the VPAC subsystem. The HTS module defines various aspects of synchronization and data sharing between the VPAC HWAs. With regards to producer and consumer dependencies, the HTS module ensures that a task starts only when input data and adequate space to write out data is available. In addition to this, it also takes care of pipe-up, debug, abort and interrupt management aspects related to the VPAC HWAs. For more details on the HTS operation, see Section *VPAC Hardware Thread Scheduler (HTS)*.
- Common shared level 2 (SL2) memory. The SL2 memory subsystem implements a full crossbar inteconnect, and serves as input/output scratch memory for the VPAC HWAs.
- Data Router Unit (DRU). The DRUs acts like (and is also referred to as) a Universal Transfer Controllers (UTCs) for managing real-time and nonreal-time DMA transfers.
- Switched Central Resource (SCR) block. The SCR acts like a 256-bit data routing interconnect within the VPAC subsystem, and is used to route UTC traffic onto SL2 memory or for external (system level) access.
- Counter, Timer and System Event Trace (CTSET) module. The CTSET module provides event tracing capabilities for debug purposes.
- ECC Aggregator. The ECC Aggregator modules provide a mechanism to control and monitor certain internal ECC RAMs via Single Error Correction (SEC) and Double Error Detection (DED) functions.

6.7.2.1.1 Notes on VISS RFE H3A Usage

The H3A module supports the control loops for auto focus (AF), auto white balance (AWB) and auto exposure (AE) by computing image statistics. The statistics are typically used by the host CPU to adjust the various parameters for processing the image data. The H3A module comprises two sub modules: the AF and AE/AWB. The H3A module, which is integrated inside RFE module of VISS, can be configured to produce either AE/AWB or AF statistics in a sigle run. Data generated is stored into a separate SL2 memory buffer by writes of 32-Byte data in a single burst. There is support of only one H3A output channel between LSE and HTS.

The H3A output data is a single dimensional data and the amount of data produced depends on number of windows/paxel in a frame, varions functional modes and corner cases of multiple of 8 windows/paxels in a row, finishing row level statistics at 32-Byte boundary, etc. Table 6-115 shows the size of data produced, depending on H3A configuration.

Table 6-115. VPAC Subsystem H3A Output Data Size

Register Settings		Number of Bytes	Special care
AF H3A_PCR[0] AF_EN = ‘1’	Vertical AF disabled H3A_PCR[20] AF_VF_EN = ‘0’	48 Bytes/paxel	If paxel_no_x odd, add 16B per row to make it 32B multiple per row.
	Vertical AF enabled H3A_PCR[20] AF_VF_EN = ‘1’	64 Bytes/paxel	None
AE and AWB H3A_PCR[16] AEW_EN = ‘1’	Sum of square H3A_AEW_CFG[9-8] AEFMT = ‘0’	34 Bytes/window	(+16B) After 8 windows (+16B) At end of row, but no end of windows, if H3A size at Row != 32B (+16B) Num of windows %8 !=0 (+16B) At end of windows, if H3A size at end of windows != 32B
	Min/Max H3A_AEW_CFG[9-8] AEFMT = ‘1’	34 Bytes/window	
	Sum only H3A_AEW_CFG[9-8] AEFMT = ‘2’	18 Bytes/window	

The parameters of H3A output buffer need to be configured via VISS LSE register settings. Some special considerations include:

- Configure the H3A line size to be more or equal to maximum H3A data produced in a row. That is, VISS_LSE_BUF_ATTR[15-6] H3A_LN_SIZE >= ceil((# of output bytes per H3A paxel/window row)/64)
- The H3A output data is 1-dimensional in the external storage (DDR or on-chip memory). For ease of synchronization, this 1-D data is converted into multiple of H3A line data. The TR (transfer request) written to

Copyright © 2024 Texas Instruments Incorporated

transfer H3A data from SL2 to external storage must make sure to configure offset such that data get stored as if 1-D continuous data.
- Configure the minimum VISS LSE_BUF_ATTR[24-16] CBUF_SIZE = 3, to take care of corner cases.

6.7.2.2 VPAC Subsystem Clocks

Each block within the VPAC subsystem receives its own clocking signal:

- MAIN_CLK is the operation clock for all infrastructure modules, such as UTC/DRU, HTS, ECC Aggregator, INTD, etc.
- VISS0_CLK is the operation clock for the VISS and its sub-modules
- LDC0_CLK is the operation clock for the LDC module
- MSC_CLK is the operation clock for the MSC module
- NF_CLK is the operation clock for the NF module

For more details on the source of each clock at SoC level, see *VPAC Integration*.

6.7.2.3 VPAC Subsystem Resets

Each block within the VPAC subsystem can be reset through its own HW reset signal:

- MAIN_RST is the reset for all infrastructure modules, such as UTC/DRU, HTS, ECC Aggregator, etc.
- VISS0_RST is the reset for the VISS and its sub-modules
- LDC0_RST is the reset for the LDC module
- MSC_RST is the reset for the MSC module
- NF_RST is the reset for the NF module

For more details on the source of each reset signal at SoC level, see *VPAC Integration*.

6.7.2.4 VPAC Subsystem Interrupts

The VPAC subsystem outputs six physical system interrupt lines: VPAC_LEVEL_0_INTR through VPAC_LEVEL_5_INTR. For more information on how the interrupt lines are connected at SoC level, see *VPAC Integration*. The interrupt aggregation within the VPAC subsystem is done using the internal INTD module. Each interrupt line can be mapped to the same list of source interrupt events generated by the modules within the VPAC subsystem.

Table 6-116 shows the INTD interrupt control registers for each VPAC interrupt line. Each interrupt line has a separate set of registers for level and pulse input event types. Each bit in a register corresponds to a particular interrupt event. For more details on the mapping of interrupt events to INTD register bits, see Table-XXX further below.

- The ENABLE registers enable the mapping of the internal VPAC interrupt events, generated by a module within the VPAC subsystem, to the VPAC output system interrupt lines. Writing a '1' in a register bit position allows the corresponding module interrupt event to trigger the respective system interrupt. Writing a '0' in the same bit position blocks the module interrupt event from triggering the system interrupt.
- The ENABLE_CLR registers disables the mapping of the internal VPAC interrupt events to the VPAC output system interrupt lines.
- The STATUS registers are used to capture which VPAC sub-module interrupt events, that have been mapped to a system interrupt, are active and pending. This status is used when multiple module interrupts are grouped into a single system interrupt, so the software can read the status register to determine exactly which module interrupt has caused the system interrupt.
  - For level interrupts, the status is based on the active level of the module interrupt input, and when the input is inactive then the status is inactive as well.
  - For pulsed interrupts, the status is set upon the pulse and reflects a pending status and software must clear the pending status since the module does not indicate the inactive status (unlike a level interrupt, which does).
- The STATUS_CLR registers allows software to clear the pending status of interrupt events. This register is only valid and functional for pulsed interrupts and has no effect for level interrupts.

Copyright © 2024 Texas Instruments Incorporated

Table 6-116. VPAC Subsystem Interrupt Control Registers

Output System Interrupt Line	Input Interrupt Event Type	Interrupt Enable Registers	Interrupt Clear Registers	Interrupt Status Registers	Interrupt Status Clear Registers
VPAC_LEVEL_0_INTR	Level	VPAC_INTD_ENABLE_REG_LEVEL_VPAC_OUT_0_0 through VPAC_INTD_ENABLE_REG_LEVEL_VPAC_OUT_0_7	VPAC_INTD_ENABLE_CLR_REG_LEVEL_VPAC_OUT_0_0 through VPAC_INTD_ENABLE_CLR_REG_LEVEL_VPAC_OUT_0_7	VPAC_INTD_STATUS_REG_LEVEL_VPAC_OUT_0_0 through VPAC_INTD_STATUS_REG_LEVEL_VPAC_OUT_0_7	VPAC_INTD_STATUS_CLR_REG_LEVEL_VPAC_OUT_0_0 through VPAC_INTD_STATUS_CLR_REG_LEVEL_VPAC_OUT_0_7
	Pulse	VPAC_INTD_ENABLE_REG_PULSE_VPAC_OUT_0_0 through VPAC_INTD_ENABLE_REG_PULSE_VPAC_OUT_0_7	VPAC_INTD_ENABLE_CLR_REG_PULSE_VPAC_OUT_0_0 through VPAC_INTD_ENABLE_CLR_REG_PULSE_VPAC_OUT_0_7	VPAC_INTD_STATUS_REG_PULSE_VPAC_OUT_0_0 through VPAC_INTD_STATUS_REG_PULSE_VPAC_OUT_0_7	VPAC_INTD_STATUS_CLR_REG_PULSE_VPAC_OUT_0_0 through VPAC_INTD_STATUS_CLR_REG_PULSE_VPAC_OUT_0_7
VPAC_LEVEL_1_INTR	Level	VPAC_INTD_ENABLE_REG_LEVEL_VPAC_OUT_1_0 through VPAC_INTD_ENABLE_REG_LEVEL_VPAC_OUT_1_7	VPAC_INTD_ENABLE_CLR_REG_LEVEL_VPAC_OUT_1_0 through VPAC_INTD_ENABLE_CLR_REG_LEVEL_VPAC_OUT_1_7	VPAC_INTD_STATUS_REG_LEVEL_VPAC_OUT_1_0 through VPAC_INTD_STATUS_REG_LEVEL_VPAC_OUT_1_7	VPAC_INTD_STATUS_CLR_REG_LEVEL_VPAC_OUT_1_0 through VPAC_INTD_STATUS_CLR_REG_LEVEL_VPAC_OUT_1_7
	Pulse	VPAC_INTD_ENABLE_REG_PULSE_VPAC_OUT_1_0 through VPAC_INTD_ENABLE_REG_PULSE_VPAC_OUT_1_7	VPAC_INTD_ENABLE_CLR_REG_PULSE_VPAC_OUT_1_0 through VPAC_INTD_ENABLE_CLR_REG_PULSE_VPAC_OUT_1_7	VPAC_INTD_STATUS_REG_PULSE_VPAC_OUT_1_0 through VPAC_INTD_STATUS_REG_PULSE_VPAC_OUT_1_7	VPAC_INTD_STATUS_CLR_REG_PULSE_VPAC_OUT_1_0 through VPAC_INTD_STATUS_CLR_REG_PULSE_VPAC_OUT_1_7
VPAC_LEVEL_2_INTR	Level	VPAC_INTD_ENABLE_REG_LEVEL_VPAC_OUT_2_0 through VPAC_INTD_ENABLE_REG_LEVEL_VPAC_OUT_2_7	VPAC_INTD_ENABLE_CLR_REG_LEVEL_VPAC_OUT_2_0 through VPAC_INTD_ENABLE_CLR_REG_LEVEL_VPAC_OUT_2_7	VPAC_INTD_STATUS_REG_LEVEL_VPAC_OUT_2_0 through VPAC_INTD_STATUS_REG_LEVEL_VPAC_OUT_2_7	VPAC_INTD_STATUS_CLR_REG_LEVEL_VPAC_OUT_2_0 through VPAC_INTD_STATUS_CLR_REG_LEVEL_VPAC_OUT_2_7
	Pulse	VPAC_INTD_ENABLE_REG_PULSE_VPAC_OUT_2_0 through VPAC_INTD_ENABLE_REG_PULSE_VPAC_OUT_2_7	VPAC_INTD_ENABLE_CLR_REG_PULSE_VPAC_OUT_2_0 through VPAC_INTD_ENABLE_CLR_REG_PULSE_VPAC_OUT_2_7	VPAC_INTD_STATUS_REG_PULSE_VPAC_OUT_2_0 through VPAC_INTD_STATUS_REG_PULSE_VPAC_OUT_2_7	VPAC_INTD_STATUS_CLR_REG_PULSE_VPAC_OUT_2_0 through VPAC_INTD_STATUS_CLR_REG_PULSE_VPAC_OUT_2_7
VPAC_LEVEL_3_INTR	Level	VPAC_INTD_ENABLE_REG_LEVEL_VPAC_OUT_3_0 through VPAC_INTD_ENABLE_REG_LEVEL_VPAC_OUT_3_7	VPAC_INTD_ENABLE_CLR_REG_LEVEL_VPAC_OUT_3_0 through VPAC_INTD_ENABLE_CLR_REG_LEVEL_VPAC_OUT_3_7	VPAC_INTD_STATUS_REG_LEVEL_VPAC_OUT_3_0 through VPAC_INTD_STATUS_REG_LEVEL_VPAC_OUT_3_7	VPAC_INTD_STATUS_CLR_REG_LEVEL_VPAC_OUT_3_0 through VPAC_INTD_STATUS_CLR_REG_LEVEL_VPAC_OUT_3_7
	Pulse	VPAC_INTD_ENABLE_REG_PULSE_VPAC_OUT_3_0 through VPAC_INTD_ENABLE_REG_PULSE_VPAC_OUT_3_7	PAC_INTD_ENABLE_CLR_REG_PULSE_VPAC_OUT_3_0 through VPAC_INTD_ENABLE_CLR_REG_PULSE_VPAC_OUT_3_7	VPAC_INTD_STATUS_REG_PULSE_VPAC_OUT_3_0 through VPAC_INTD_STATUS_REG_PULSE_VPAC_OUT_3_7	VPAC_INTD_STATUS_CLR_REG_PULSE_VPAC_OUT_3_0 through VPAC_INTD_STATUS_CLR_REG_PULSE_VPAC_OUT_3_7
VPAC_LEVEL_4_INTR	Level	VPAC_INTD_ENABLE_REG_LEVEL_VPAC_OUT_4_0 through VPAC_INTD_ENABLE_REG_LEVEL_VPAC_OUT_4_7	VPAC_INTD_ENABLE_CLR_REG_LEVEL_VPAC_OUT_4_0 through VPAC_INTD_ENABLE_CLR_REG_LEVEL_VPAC_OUT_4_7	VPAC_INTD_STATUS_REG_LEVEL_VPAC_OUT_4_0 through VPAC_INTD_STATUS_REG_LEVEL_VPAC_OUT_4_7	VPAC_INTD_STATUS_CLR_REG_LEVEL_VPAC_OUT_4_0 through VPAC_INTD_STATUS_CLR_REG_LEVEL_VPAC_OUT_4_7
	Pulse	VPAC_INTD_ENABLE_REG_PULSE_VPAC_OUT_4_0 through VPAC_INTD_ENABLE_REG_PULSE_VPAC_OUT_4_7	VPAC_INTD_ENABLE_CLR_REG_PULSE_VPAC_OUT_4_0 through VPAC_INTD_ENABLE_CLR_REG_PULSE_VPAC_OUT_4_7	VPAC_INTD_STATUS_REG_PULSE_VPAC_OUT_4_0 through VPAC_INTD_STATUS_REG_PULSE_VPAC_OUT_4_7	VPAC_INTD_STATUS_CLR_REG_PULSE_VPAC_OUT_4_0 through VPAC_INTD_STATUS_CLR_REG_PULSE_VPAC_OUT_4_7

Copyright © 2024 Texas Instruments Incorporated

Table 6-116. VPAC Subsystem Interrupt Control Registers (continued)

Output System Interrupt Line	Input Interrupt Event Type	Interrupt Enable Registers	Interrupt Clear Registers	Interrupt Status Registers	Interrupt Status Clear Registers
VPAC_LEVEL_5_INTR	Level	VPAC_INTD_ENABLE_REG_LEVEL_VPAC_OUT_5_0 through VPAC_INTD_ENABLE_REG_LEVEL_VPAC_OUT_5_7	VPAC_INTD_ENABLE_CLR_REG_LEVEL_VPAC_OUT_5_0 through VPAC_INTD_ENABLE_CLR_REG_LEVEL_VPAC_OUT_5_7	VPAC_INTD_STATUS_REG_LEVEL_VPAC_OUT_5_0 through VPAC_INTD_STATUS_REG_LEVEL_VPAC_OUT_5_7	VPAC_INTD_STATUS_CLR_REG_LEVEL_VPAC_OUT_5_0 through VPAC_INTD_STATUS_CLR_REG_LEVEL_VPAC_OUT_5_7
	Pulse	VPAC_INTD_ENABLE_REG_PULSE_VPAC_OUT_5_0 through VPAC_INTD_ENABLE_REG_PULSE_VPAC_OUT_5_7	VPAC_INTD_ENABLE_CLR_REG_PULSE_VPAC_OUT_5_0 through VPAC_INTD_ENABLE_CLR_REG_PULSE_VPAC_OUT_5_7	VPAC_INTD_STATUS_REG_PULSE_VPAC_OUT_5_0 through VPAC_INTD_STATUS_REG_PULSE_VPAC_OUT_5_7	VPAC_INTD_STATUS_CLR_REG_PULSE_VPAC_OUT_5_0 through VPAC_INTD_STATUS_CLR_REG_PULSE_VPAC_OUT_5_7

The VISS interrupt events are mapped to register bits within the VPAC_INTD_xxx_VPAC_OUT_0_0 through VPAC_INTD_xxx_VPAC_OUT_5_0 registers. Table 6-117 provides more details on the VISS interrupt events. All VISS interrupts are single pulse event signals.

Table 6-117. VPAC Subsystem VISS Interrupt Events

Register Bit	Interrupt Event Name	Description
0	RAWFE_CFG_ERR_INTR	Config read or write memory acccess occurred during functional operation and likely corrupted functional opertion. VISS merges all config error sources from RawFE and refer to RawFE spec for the entire error source list.
1	RAWFE_AEW_PULSE_INTR	H3A AEW interrupt.
2	RAWFE_AF_PULSE_INTR	H3A AF interrupt.
3	RAWFE_H3A_PULSE_INTR	H3A interrupt.
4	RAWFE_H3A_BUF_OVRFLOW_PULSE_INTR	H3A output buffer overflow.
5	NSF4V_LINEMEM_CFG_ERR_INTR	VBUSP diagnostic read access of , while NSF data using RAM for functional purpose.
6	NSF4V_HBLANK_ERR_INTR	Horzontal Blanking too short between lines.
7	NSF4V_VBLANK_ERR_INTR	Vertical Blanking too short between frames.
8	GLBCE_CFG_ERR_INTR	Either non-shadowed registers written or statastic memories are accessed during active window.
9	GLBCE_FILT_START_INTR	GLBCE started filtering. This interrupt is issued at the rising edge of filtering signal.
10	GLBCE_FILT_DONE_INTR	GLBCE ended filtering. This interrupt is issued at the falling edge of filtering signal.
11	GLBCE_HSYNC_ERR_INTR	Generated when delayed HS/HE signals doesn't match with derived signals from GLBCE core.
12	GLBCE_VSYNC_ERR_INTR	Generated when delayed VS/VE signals doesn't match with derived signals from GLBCE core.
13	GLBCE_VP_ERR_INTR	This interrupt is issued, if there is a data input while filtering is high.
14	FCFA_CFG_ERR_INTR	Either non-shadowed registers written or line memories are accessed during active window.
15	FCC_CFG_ERR_INTR	Configuration access to registers/memories has corrupted functional operation.
		Configuration read or write memory acccess occurred during functional operation.
		Merged independent error sources at VISS. Refer to Section *VISS Flexible Color Processing (FCP)* for all sources.
16	FCC_OUTIF_OVF_ERR_INTR	FIFO overflow on FIFO for Y12 LSE I/F. Merged all FCC output overflow error sources at VISS. Refer to Section *VISS Flexible Color Processing (FCP)* for all sources.

Copyright © 2024 Texas Instruments Incorporated

Table 6-117. VPAC Subsystem VISS Interrupt Events (continued)

Register Bit	Interrupt Event Name	Description
17	FCC_HIST_READ_ERR_INTR	Host was not able to read the entire histogram mem between VS-VE window (triggered when the first access to histogram has been performed but the last has not been performed).
18	EE_CFG_ERR	Cinfiguration happened to EE regions causing corruption during frame processing.
19	EE_SYNCOVF_ERR	EE horizontal synchronization FIFO overflow interrupt.
20	LSE_FR_DONE_EVT_INTR	LSE frame done interrupt.
21	LSE_SL2_RD_ERR_INTR	Set whenever there is an error response on VBUSM read command for any input channel.
22	LSE_SL2_WR_ERR_INTR	Set whenever there is an error response on VBUSM write command for any output channel.
23	LSE_CAL_VP_ERR_INTR	Set whenever one of the following input frame errors is detected at VPORT_INPUT.
24	LSE_OUT_FR_START_EVT_INTR	Ouput Frame Start (from VISS top level).

The LDC interrupt events are mapped to register bits within the VPAC_INTD_xxx_VPAC_OUT_0_1 through VPAC_INTD_xxx_VPAC_OUT_5_1 registers. Table 6-118 provides more details on the LDC interrupt events. All LDC interrupts are single pulse event signals.

Table 6-118. VPAC Subsystem LDC Interrupt Events

Register Bit	Interrupt Event Name	Description
0	LDC_PIX_IBLK_OUTOFBOUND_INTR	Back mapped pixel co-ordinate goes out of the pre-computed input pixel bounding box.
1	LDC_MESH_IBLK_OUTOFBOUND_INTR	Block mesh co-ordinate goes out of the pre-computed mesh bounding box.
2	LDC_PIX_IBLK_MEMOVF_INTR	Input Pixel block memory overflow.
3	LDC_MESH_IBLK_MEMOVF_INTR	Mesh block memory overflow.
4	LDC_IFR_OUTOFBOUND_INTR	Back mapped input co-ordinate goes out of input frame range.
5	LDC_INT_SZOVF_INTR	Affine and perspective transform precision overflow error.
6	LDC_FR_DONE_EVT_INTR	Frame done LDC.
7	LDC_SL2_WR_ERR_INTR	Error on SL2 VBSUM Write interface.
8	LDC_VBUSM_RD_ERR_INTR	Error on Input VBUSM Read interface.

The MSC interrupt events are mapped to register bits within the VPAC_INTD_xxx_VPAC_OUT_0_2 through VPAC_INTD_xxx_VPAC_OUT_5_2 registers. Table 6-119 provides more details on the MSC interrupt events. All MSC interrupts are single pulse event signals.

Table 6-119. VPAC Subsystem MSC Interrupt Events

Register Bit	Interrupt Event Name	Description
0	MSC_LSE_FR_DONE_EVT_0_INTR	Frame done MSC processing thread 0.
1	MSC_LSE_FR_DONE_EVT_1_INTR	Frame done MSC processing thread 1.
2	MSC_LSE_SL2_RD_ERR_INTR	Error on SL2 VBSUM Read interface.
3	MSC_LSE_SL2_WR_ERR_INTR	Error on SL2 VBSUM Write interface.

The NF interrupt events are mapped to register bits within the VPAC_INTD_xxx_VPAC_OUT_0_2 through VPAC_INTD_xxx_VPAC_OUT_5_2 registers. Table 6-120 provides more details on the NF interrupt events. All NF interrupts are single pulse event signals.

Table 6-120. VPAC Subsystem NF Interrupt Events

Register Bit	Interrupt Event Name	Description
8	NF_FR_DONE_INTR	Frame done NF.
9	NF_SL2_WR_ERR_INTR	Error on SL2 VBSUM Write interface.

Copyright © 2024 Texas Instruments Incorporated

Table 6-120. VPAC Subsystem NF Interrupt Events (continued)

Register Bit	Interrupt Event Name	Description
10	NF_SL2_RD_ERR_INTR	Error on SL2 VBSUM Read interface.

The HTS interrupt events are mapped to register bits within the VPAC_INTD_xxx_VPAC_OUT_0_3 through VPAC_INTD_xxx_VPAC_OUT_5_3 registers. Table 6-121 provides more details on the HTS interrupt events. All HTS interrupts are single pulse event signals, save for the SPARE_PEND_xxx interrupts, which can be both single pulse events and level events.

Table 6-121. VPAC Subsystem HTS Interrupt Events

Register Bit	Interrupt Event Name	Description
0	PIPE_DONE_0	Pipeline 0 done.
1	PIPE_DONE_1	Pipeline 1 done.
2	PIPE_DONE_2	Pipeline 2 done.
3	PIPE_DONE_3	Pipeline 3 done.
4	PIPE_DONE_4	Pipeline 4 done.
5	PIPE_DONE_5	Pipeline 5 done.
6	PIPE_DONE_6	Pipeline 6 done.
7	TDONE	HWA0 thread done.
8	RESERVED	Reserved.
9	TDONE	HWA2 thread done.
10	RESERVED	Reserved.
11	TDONE	HWA4 thread done.
12	TDONE	HWA5 thread done.
13	TDONE	HWA6 thread done.
14	SPARE_DEC_0	Spare 0 scheduler decrement pulse (ehost mode).
15	SPARE_DEC_1	Spare 1 scheduler decrement pulse (ehost mode).
16	SPARE_PEND_0_PULSE	Spare 0 scheduler pend assertion (ehost mode).
17	SPARE_PEND_0_LEVEL	Spare 0 scheduler pend assertion (ehost mode).
18	SPARE_PEND_1_PULSE	Spare 1 scheduler pend assertion (ehost mode).
19	SPARE_PEND_1_LEVEL	Spare 1 scheduler pend assertion (ehost mode).
20	WATCHDOGTIMER_ERR_0	Watchdog Timeout Error for HWA0.
21	RESERVED	Reserved.
22	WATCHDOGTIMER_ERR_2	Watchdog Timeout Error for HWA2.
23	RESERVED	Reserved.
24	WATCHDOGTIMER_ERR_4	Watchdog Timeout Error for HWA4.
25	WATCHDOGTIMER_ERR_5	Watchdog Timeout Error for HWA5.
26	WATCHDOGTIMER_ERR_6	Watchdog Timeout Error for HWA6.

The UTC interrupt events indicating TR complete are mapped to register bits within the VPAC_INTD_xxx_VPAC_OUT_0_4 through VPAC_INTD_xxx_VPAC_OUT_5_4 registers for UTC0, and VPAC_INTD_xxx_VPAC_OUT_0_5 through VPAC_INTD_xxx_VPAC_OUT_5_5 plus VPAC_INTD_xxx_VPAC_OUT_0_6 through VPAC_INTD_xxx_VPAC_OUT_5_6 registers for UTC1. Table 6-122 provides more details on the UTC TR complete interrupt events. All UTC TR complete interrupts are single pulse event signals.

Table 6-122. VPAC Subsystem UTC TR Complete Interrupt Events

Register Bit	Interrupt Event Name	Description
31-0	UTC0_COMPLETE_INTR[31:0]	TR complete interrupt.
31-0 plus 31-0	UTC1_COMPLETE_INTR[63:0]	TR complete interrupt.

Copyright © 2024 Texas Instruments Incorporated

The UTC interrupt events indicating error are mapped to register bits within the VPAC_INTD_xxx_VPAC_OUT_0_7 through VPAC_INTD_xxx_VPAC_OUT_5_7 registers for both UTC0 and UTC1. Table 6-123 provides more details on the UTC error interrupt events. All UTC error interrupts are single pulse event signals.

Table 6-123. VPAC Subsystem UTC Error Interrupt Events

Register Bit	Interrupt Event Name	Description
0	UTC0_ERR	UTC0 error.
1	UTC1_ERR	UTC1 error.
2	UTC0_PROT_ERR_INTR	UTC0 protocol violation.
3	UTC1_PROT_ERR_INTR	UTC1 protocol violation.

The CTSET interrupt events are mapped to register bits within the VPAC_INTD_xxx_VPAC_OUT_0_7 through VPAC_INTD_xxx_VPAC_OUT_5_7 registers. Table 6-124 provides more details on the CTSET events. All CTSET error interrupts are single pulse event signals.

Table 6-124. VPAC Subsystem CTSET Interrupt Events

Register Bit	Interrupt Event Name	Description
4	CTM_PULSE_INTR	Counter Timer Module (CTM) pulse interrupt.

6.7.2.5 VPAC Subsystem SL2 Memory Infrastructure

The SL2 memory subsystem implements an interconnect with a fixed priority scheme for accessing the memory banks. The slaves are LS banked. The master side (initiator) priority order is as follows (with 0 being the highest):

- 0. 256-bit UTC-RT RD (real-time traffic)
- 1. 256-bit UTC-RT WR (real-time traffic)
- 2. 512-bit VISS0 (real-time)
- 3. 512-bit MSC
- 4. 512-bit LDC0
- 5. 512-bit NF
- 6. 256-bit UTC-NRT RD (nonreal-time traffic)
- 7. 256-bit UTC-NRT WR (nonreal-time traffic)
- 8. 256-bit VBUSM slave interface from NAVSS at SoC level

The SL2 memory does not implement ECC, as it contains mostly pixel data.

The output of the LDC is block (2D). The NF block as well as the MSC, however, are working with lines (1D). An automatic block (2D) to line (1D) conversion is implemented in the HW through a pattern adapter, without any dedicated logic.

6.7.2.6 VPAC Subsystem DMA Infrastructure

The VPAC DRU (that is, UTC), together with the NAVSS UDMA module, are used to implement the K3 Data Movement Architecture (DMA) for the VPAC subsysem. The VPAC UTC implements a Transfer Controller (TC) for managing the DMA transfers and the NAVSS UDMA works as channel controller. The UDMA supports pushing TC into UTC on a PSI-L layer. The UTC module is optimized for transfer types such as 2D/3D/4D, that are typical of VPAC access patterns. The UTC module supports 256-bit data bus, 64 events, and total 64 channels in block copy mode.

Copyright © 2024 Texas Instruments Incorporated

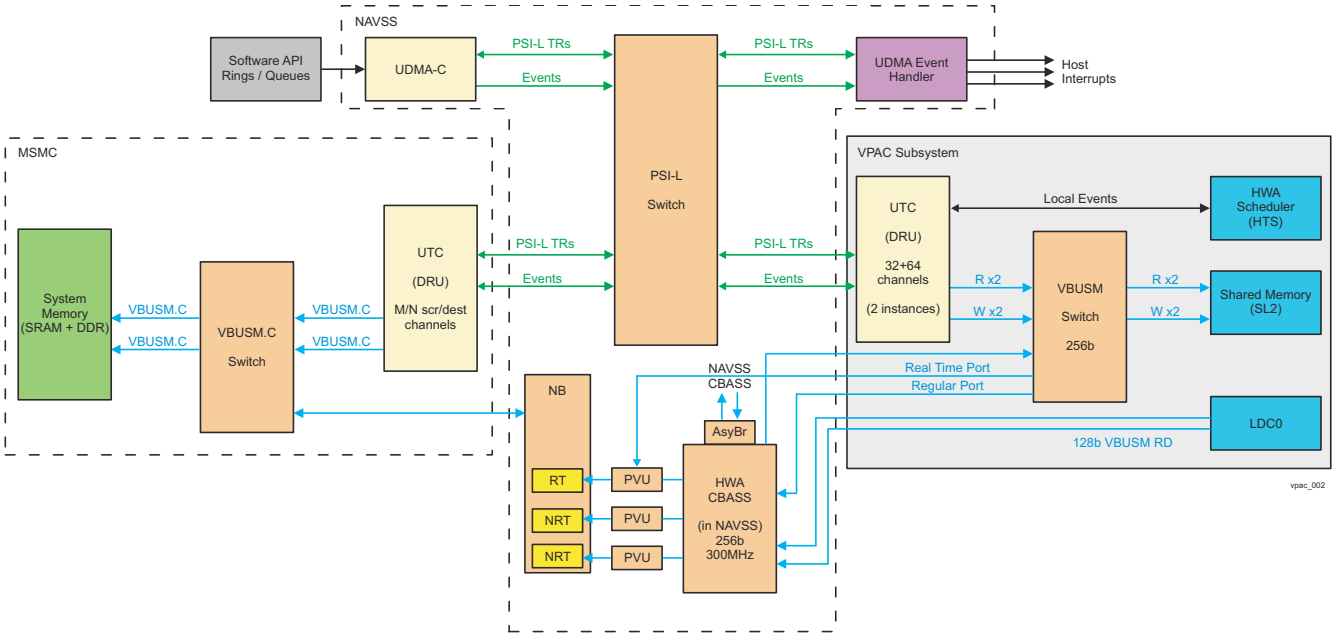


Figure 6-49. VPAC Subsystem Data Movement Flow Diagram

Figure 6-49 captures the key components in the data movement flow for the VPAC subsystem. For further details about their connectivity and functional details, refer to Chapter *Data Movement Architecture (DMA)*.

The use of another (SoC level) external DMA (that is, the SoC level DRU) can be used for long distance reads. Transfer triggers are always generated by the VPAC HTS module, while the VPAC UTC returns back transfer completion event to the HTS. The required set of transfers is programmed via the VPAC UDMA Channel Controller (that is, the UDMA module within NAVSS). However, the actual DMA engine used to carry out the transfers (VPAC UTC or SoC level DRU) remains transparent to the software programming model.

The VPAC UTC receives DMA transfer events to read or write the following set of data:

1. VISS0 input frame (exposure0, exposure1, exposure2)
2. VISS0 output data (Y12, UV12, Y8, UV8, S8, AE/AF)
3. LDC0 output data (Luma12, Chroma12, Luma8, Chroma8)
4. MSC0 input
5. MSC1 input
6. MSC0/MSC1 output (10 scales)
7. NF input
8. NF output
9. Spare scheduler DMA channels (block copy - line buffer data transfer through SoC level MSMC)

Two UTC instances are used inside the VPAC subsystem. UTC0 deals with real time (RT) transfer , and UTC1 is used for non-realtime (NRT) transfer. Both UTC0 and UTC1 data transfer channels are used for DMA transfers in and out of the VPAC SL2 memory. All 32 channels of the UTC0 are mapped on *channel_number = [95:64]* and are recommended to be used for real time data transfer only. All 64 channels of UTC1 are mapped on *channel_number = [63:0]* and are recommended to be used for all other DMA transfer needs of the VPAC subsystem. The VBUSM *chanid* signal carries channel number info for each UTC on its own VBUSM ports. Each UTC can generate up to 64 outstanding transactions.

These data transfer events would be mapped to channel within UTC. All VPAC data transfers are deterministic in nature and hence it does not require any interaction from the software in between a frame processing. TR (transfer requests) should be programmed in such a way that they operate on a circular buffer in SL2 memory and a full frame level buffer in DDR. For more details about the TR format, see Chapter *Data Movement Architecture (DMA)*.

All data movement transactions at VPAC subsystem boundary are multiples of aligned 128-byte transfers.

Copyright © 2024 Texas Instruments Incorporated

6.7.2.7 VPAC Subsystem Data Routing Interconnect

The data routing is based on a system address of the SL2 memory. This is a 256-bit interconnect. The LDC read interface directly connects to AC internal CBASS.

The SL2 master port of this interconnect implements a 512-bit width adaptor. In other words, the write accesses to the SL2 master port accumulate two consecutive 256 bits data phases, pack them to form a 512 bits word and then write it into SL2 memory. The read accesses on this port, on the other hand, are split into two 256 bits data phases before being sent to the requestor port.

The data routing interconnect has four masters on both VPAC SL2 side and on the external VBUSM side.

6.7.2.8 VPAC Subsystem Pipeline Flow Control and Messaging

The VPAC subsystem can run up to 5 independent pipelines using the VPAC HWAs and UTC. These pipelines are completely independent from each other and their input output data management is handled by the HTS module. The thread management is flexible and supports different top level settings of resolution, frame rate, DDR data buffer address, etc.

The VPAC HTS module is used to implement the thread management and control triggering of processing threads within the VPAC subsystem. It is used to manage message transfer and control between VPAC and external host (that is, SoC level processor). HTS can also be used to handle writing LDC block data into the SoC level MSMC (L3 SRAM) and create block to row data for the VPAC MSC and NF processing without going through the external DDR memory.

The mapping of VPAC HWAs to HTS scheduler is shown inTable 6-125. Note that each HWA and spare scheduler has a spare socket on its consumer and producer side (not mentioned in below table) for future expansion. For more details, see Section *VPAC Hardware Thread Scheduler (HTS)*.

Table 6-125. VPAC Subsystem HTS Mapping

HTS scheduler	VPAC HWA
HWA0	VISS0
HWA1	Reserved
HWA2	LDC0
HWA3	Reserved
HWA4	MSC0
HWA5	MSC1
HWA6	NF
HWA7	N/A
HWA8	N/A
DMA0-4	VISS0 [c0,c1,c2,x,x]
DMA8-10	Reserved
DMA32	MSC0 [c0]
DMA40	MSC1 [c0]
DMA48	NF [c0]
DMA56-59	N/A
DMA64-67	N/A
DMA240-245	VISS0 [p0,p1,p2,p3,p4,p5]
DMA256-261	Reserved
DMA272-275	LDC0 [p0,p1,p2,p3]
DMA288-291	Reserved
DMA304-313	MSC0 [p0] / MSC1 [p0],
	MSC0 [p1] / MSC1 [p1],
	...
	MSC0 [p9] / MSC1 [p9],
DMA336	NF [p0]

Copyright © 2024 Texas Instruments Incorporated

Table 6-125. VPAC Subsystem HTS Mapping (continued)

HTS scheduler	VPAC HWA
DMA352-355	N/A
DMA368-371	N/A

6.7.2.8.1 VISS Node Scheduler

The VISS node scheduler comprises 3 consumer sockets and 6 producer sockets. On consumer side, it is only connected to the DMA producer scheduler. On producer socket, it is only connected to the DMA consumer scheduler.

The consumer sockets are mapped to:
1. Exposure #0 for WDR merge (DMA)
2. Exposure #1 for WDR merge (DMA)
3. Exposure #2 for WDR merge (DMA)

The producer sockets are mapped to the following output buffers:
1. Y12 (DMA)
2. UV12 (DMA)
3. Y8 (DMA)
4. UV8 (DMA)
5. S8 (DMA)
6. H3A (DMA)

Pipeline #n (n = 0, …, 6 configurable) is mapped to this scheduler. When a scheduler is enabled, the DMA producer scheduler triggers UTC data loading from DDR memory into the SL2 memory. Once data is available inside the SL2 memory, the VISS scheduler starts a VISS thread. Prior to starting a VISS thread, the buffer availability to write output data is checked. These consumer and producer sockets can optionally be disabled depending on the usecase.

To enable streaming data support, this scheduler node supports a streaming feature. That is, the scheduler does not go into idle state after the end of the pipe, rather it transitions to initialization state to save host intervention between frame in OTF (on-the-fly) mode.

6.7.2.8.2 LDC Node Scheduler

The LDC node scheduler comprises 0 consumer sockets and 7 producer sockets. On consumer side, the LDC uses its own read DMA engine to fetch input data directly into internal buffer of LDC. On producer socket, it can be connected to 5 different consumers.

The producer sockets are mapped to the following output buffers:
1. Luma(#1) -> Pattern Adapter -> DMA
2. Chroma(#1) -> Pattern Adapter -> DMA
3. Luma(#2) -> Pattern Adapter -> DMA
4. Chroma(#2) -> Pattern Adapter -> DMA
5. Luma/Chroma -> Pattern Adapter -> MSC0
6. Luma/Chroma -> Pattern Adapter -> MSC1
7. Luma/Chroma -> Pattern Adapter -> NF

Pipeline #n (n = 0, …, 6 configurable) is mapped to this scheduler. When a scheduler is enabled, the LDC read DMA engine loads required input data into the LDC memory. Prior to starting a LDC thread, the output buffer availability is checked to write output data. These producer sockets can optionally be disabled depending on the usecase.

To support on-the-fly MSC and NF operation after LDC, a pattern adapter is used to convert blocks into lines. Although produced data count is only two, the produced data count is still included, along with pattern adapter pair for unique end consumers.

On-the-fly support for LDC -> MSC / LDC -> NF connection can be achived via the SL2 memory (if space is available) or via the SoC level MSMC SRAM. Both flows are supported using the HTS scheduler.

Copyright © 2024 Texas Instruments Incorporated

6.7.2.8.3 MSC Node Scheduler

The MSC supports two independent threads. For this reason, two MSC schedulers are used. Each MSC node scheduler comprises consumer socket and 10 producer sockets. On consumer side, it is either connected to the DMA producer scheduler or the LDC output. On producer socket, it is to the DMA consumer scheduler.

The consumer socket is mapped to:

1. Scalar data from DDR / LDC output

The producer sockets are mapped to the following output buffers:

1. MSC0-0
2. MSC0-1
3. MSC0-2
4. MSC0-3
5. MSC0-4
6. MSC0-5
7. MSC0-6
8. MSC0-7
9. MSC0-8
10. MSC0-9

Pipeline #n (n = 0, …, 6 configurable) is mapped to this scheduler. When a scheduler is enabled, the DMA producer scheduler triggers UTC data loading from the DDR memory into SL2 memory. Once data is available inside the SL2 meory, the MSC scheduler starts a MSC thread. Prior to starting a MSC thread, the buffer availability to write output data is checked. These consumer and producer sockets can optionally be disabled depending on the usecase.

For both MSC threads, two schedulers are used in VPAC subsystem. Since the MSC core can output maximum 10 scaled outputs (combining MSC0 and MSC1), this fact is utilized in optimizing the DMA consumer sockets and DMA channels required. The need of DMA channels is mutually orthogonal in inverted order, that is, (MSC0-0 and MSC1-9), (MSC0-1 and MSC1-8), etc. are mapped to the same DMA channel. When the MSC needs to run in the same pipeline as LDC, then pipeline number must be same as of the LDC. The consumer socket needs to be connnected to the LDC output (p0).

6.7.2.8.4 NF Node Scheduler

The NF node scheduler comprises 1 consumer socket and 1 producer socket. On consumer side, it is either connected to the DMA producer scheduler or the LDC output. On producer socket, it is only connected to the DMA consumer scheduler.

The consumer socket is mapped to:

1. NF input data from DDR / LDC producer socket (p1)

The producer sockets are mapped to the following output buffer:

1. NF output

Pipeline #n (n = 0, …, 6 configurable) is mapped to this scheduler. When a scheduler is enabled, the DMA producer scheduler triggers UTC data loading from the DDR memory into SL2 memory. Once data is available inside the SL2 memory, the NF scheduler starts a NF thread. Prior to starting a NF thread, the buffer availability to write output data is checked.

When the NF needs to run in the same pipeline as LDC, then the pipeline number must be same as of the LDC. The consumer socket needs to be connnected to the LDC output (p1).

6.7.2.8.5 Spare Scheduler

There are up to spare schedulers in each HTS instance. The key features of each spare scheduler are as follows:

- Store LDC data into L3 memory (SoC level MSMC SRAM) and grow line buffer for next trigger to MSC/NF through spare schedulers.
- Help in inserting any external host through spare schedulers.

Copyright © 2024 Texas Instruments Incorporated

- Capability to trigger DMA and include pattern adapters to support block/line conversion control.
- Spare scheduler 0/1 : Inserting external host
- Spare scheduler 2...7 : Managing growing block to line buffer inside MSMC (L3 SRAM)

6.7.2.9 VPAC Subsystem Data Formats Support

Table 6-126 summarizes the data formats supported in VPAC Subsystem.

Table 6-126. VPAC Subsystem Data Formats Support

No	Module	Input			Output		
		Bit-depth	Chroma (NV12)	Packing	Bit-depth	Chroma (NV12)	Packing
1	VISS	12	RAW	no	12 & 8	420	yes
2		14	RAW	no	12 & 8	420	yes
3		16	RAW	no	12 & 8	420	yes
4		8	RAW	yes	8	420	yes
5		12	RAW	yes	12 & 8	420	yes
5		All of the above			8	422	yes
6	LDC	8	422	yes	8	420	yes
7		8	422	yes	8	422	yes
8		8	420	yes	8	420	yes
9		12	420	yes/no	12	420	yes
10		12	420	yes/no	12 & 8	420	no
11	MSC	12	Planar	yes	12	planar	yes
12		8	Planar	yes	8	planar	yes
13	NF	12	Y/UV (NV12)	yes	12	UV (NV12)	yes

6.7.2.10 VPAC Subsystem Debug Features

The VPAC subsystem is a combination of multiple asynchronous pipelines running concurrently. The Counter, Timer and System Event Trace (CTSET) module within the VPAC susbystem provides the following main features:

- Monitoring up to 255 VPAC events.
- ATB interface compliant trace packets (synchronous to VPAC clock domain).
- Counter (count=8) and Timer (count=4).
- A CoreSight CTI compatible trigger interface for Event Trace Control, Timer Trigger Output and Event Output.
- All debug events can be configured to be either selected for trace packet, timer or counter features.
- In one of the combination, debug events can be selected to for timer event out to be used to generate external CTI compliant trigger.

For more information on the CTSET module operation, see Chapter *On-Chip Debug*.

Table 6-127 lists the VPAC subsystem events that are mapped on the CTSET module.

Table 6-127. VPAC Subsystem CTSET Event List

Index	Event Name	HWA	Event Type	Event Description
0	viss_err	VISS0	pulse	VISS Related Error message (config, ovf, sync, blank, access, vp) [(1)]
1	rawfe_h3a_pulse_intr	VISS0	pulse	H3A interrupt
2	rawfe_dbg_ctl_vs_event	VISS0	pulse	Verticle start in the beginning of the RAWFE pipeline
3	rawfe_dbg_ctl_ve_event	VISS0	pulse	Verticle end in the beginning of the RAWFE pipeline
4	rawfe_dbg_ctl_hs_event	VISS0	pulse	Horizontal start in the beginning of the RAWFE pipeline
5	rawfe_dbg_ctl_he_event	VISS0	pulse	Horizaontal end in the beginning of the RAWFE pipeline

Copyright © 2024 Texas Instruments Incorporated

Table 6-127. VPAC Subsystem CTSET Event List (continued)

Index	Event Name	HWA	Event Type	Event Description
6	rawfe_dbg_ctl_lse_slv_stall_event	VISS0	pulse	RAWFE is stalling lse slave interface due to FCP or MTC stall
7	rawfe_dbg_ctl_lse_mst_stall_event	VISS0	pulse	H3A master interface to LSE is stalled
8	rawfe_dbg_ctl_lse_intf_stall_event	VISS0	pulse	Within a frame (VS to VE at RFE i/p) LSE is not sending data to RFE
9	rawfe_dbg_ctl_x_y_match_event	VISS0	pulse	X,Y pixel position has reached the start of RAWFE pipeline
10	rawfe_dbg_ctl_dpc_otf_corr_event	VISS0	pulse	DPC OTF corrected a pixel position
11	rawfe_dbg_ctl_pipe_adv_event	VISS0	pulse	Active pipeline advancement
12	nsf4v_in_hs_event	VISS0	pulse	Start of Active Line at NSF4V input
13	nsf4v_in_vs_event	VISS0	pulse	Start of Active Frame at NSF4V input
14	nsf4v_in_he_event	VISS0	pulse	End of Active Line at NSF4V output
15	nsf4v_in_ve_event	VISS0	pulse	End of Active Frame at NSF4V output
16	glbce_filt_start_intr	VISS0	pulse	GLBCE filtering start (generated at the rising edge of filtering signal)
17	glbce_filt_done_intr	VISS0	pulse	GLBCE filtering done (generated at falling edge of filtering signal)
18	glbce_sol_event	VISS0	pulse	Start of Active Line at GLBCE input
19	glbce_sof_event	VISS0	pulse	Start of Active Frame at GLBCE input
20	glbce_eol_event	VISS0	pulse	End of Active Line at GLBCE output
21	glbce_eof_event	VISS0	pulse	End of Active Frame at GLBCE output
22	fcfa_sol_event	VISS0	pulse	Start of line processing for CFA input
23	fcfa_sof_event	VISS0	pulse	Start of frame processing for CFA input
24	fcc_stall_event	VISS0	pulse	Stall has occurred on any one of the output LSE interfaces
25	fcc_eol_if_y12_event	VISS0	pulse	End of line on Y12 LSE I/f (only generated for valid lines)
26	fcc_eof_if_y12_event	VISS0	pulse	End of frame on Y12 LSE I/f
27	fcc_eol_if_uv12_event	VISS0	pulse	End of line on UV12 LSE I/f (only generated for valid lines)
28	fcc_eof_if_uv12_event	VISS0	pulse	End of frame on UV12 LSE I/f
29	fcc_eol_if_y8r8_event	VISS0	pulse	End of line on Y8R8 LSE I/f (only generated for valid lines)
30	fcc_eof_if_y8r8_event	VISS0	pulse	End of frame on Y8R8 LSE I/f
31	fcc_eol_if_c8g8_event	VISS0	pulse	End of line on C8G8 LSE I/f (only generated for valid lines)
32	fcc_eof_if_c8g8_event	VISS0	pulse	End of frame on C8G8 LSE I/f
33	fcc_eol_if_s8b8_event	VISS0	pulse	End of line on S8B8 LSE I/f (only generated for valid lines)
34	fcc_eof_if_s8b8_event	VISS0	pulse	End of frame on S8B8 LSE I/f
35	fcc_flexcc_eop_event	VISS0	pulse	End of processing (EOP) for FlexCC
36	lse_fr_done_evt_intr	VISS0	pulse	Frame processing done event
37	lse_out_fr_start_evt_intr	VISS0	pulse	Travelled Frame Start from LSE Core
39	ldc_err	LCD0	pulse	LDC processing error [(2)]
40	mesh_iblk_fetch_start_event	LCD0	pulse	Input mesh block fetch start event
41	mesh_iblk_fetch_done_event	LCD0	pulse	Input mesh block fetch completion event
42	pix_iblk_fetch_start_event	LCD0	pulse	Input pixel block fetch start event (Active for both Y and C)
43	pix_iblk_fetch_done_event	LCD0	pulse	Input pixel block fetch done event (Active for both Y and C)

Copyright © 2024 Texas Instruments Incorporated

Table 6-127. VPAC Subsystem CTSET Event List (continued)

Index	Event Name	HWA	Event Type	Event Description
44	frame_start_event	LCD0	pulse	Frame processing start event
45	lse_stall_event	LCD0	pulse	LSE write stall event
46	coreblk_proc_done_event	LCD0	pulse	LDC Core Block processing done
47	corecblk_wr_done_event	LCD0	pulse	LDC Core block write complete on C12 channel
48	coreyblk_wr_done_event	LCD0	pulse	LDC Core block write complete on Y12 channel
49	vpac_ldc_fr_done_evt_intr	LCD0	pulse	Frame processing done event
51	msc_err	MSC	pulse	Error Response for VBUSM read/Write SL2 access (msc_lse_sl2_rd_err_intr || msc_lse_sl2_wr_err_intr)
52	msc_lse_fr_done_evt_0_intr	MSC	pulse	Frame processing done event for MSC Thread0
53	msc_lse_fr_done_evt_1_intr	MSC	pulse	Frame processing done event for MSC Thread1
55	nf_err	NF	pulse	Error Response for VBUSM read/Write SL2 access (vpac_nf_sl2_wr_err_intr || vpac_nf_sl2_rd_err_intr)
56	vpac_nf_fr_done_evt_intr	NF	pulse	Frame processing start event
58	cal_vp_stall	CAL	pulse	CAL VP stall
60	start	HTS	pulse	HWA-0 (task start)
61	done	HTS	pulse	HWA-0 (task done)
62	init	HTS	pulse	HWA-0 (init)
63	eop	HTS	pulse	HWA-0 (eop)
64	out_ch0_done	HTS	pulse	HWA-0-ch0-done
65	out_ch1_done	HTS	pulse	HWA-0-ch1-done
66	out_ch2_done	HTS	pulse	HWA-0-ch2-done
67	out_ch3_done	HTS	pulse	HWA-0-ch3-done
68	out_ch4_done	HTS	pulse	HWA-0-ch4-done
69	out_ch5_done	HTS	pulse	HWA-0-ch5-done
70	start	HTS	pulse	HWA-2 (task start)
71	done	HTS	pulse	HWA-2 (task done)
72	init	HTS	pulse	HWA-2 (init)
73	eop	HTS	pulse	HWA-2 (eop)
74	out_ch0_done	HTS	pulse	HWA-2-ch0-done
75	out_ch1_done	HTS	pulse	HWA-2-ch1-done
76	out_ch2_done	HTS	pulse	HWA-2-ch2-done
77	out_ch3_done	HTS	pulse	HWA-2-ch3-done
78	start	HTS	pulse	HWA-4 (task start)
79	done	HTS	pulse	HWA-4 (task done)
80	init	HTS	pulse	HWA-4 (init)
81	eop	HTS	pulse	HWA-4 (eop)
82	start	HTS	pulse	HWA-5 (task start)
83	done	HTS	pulse	HWA-5 (task done)
84	init	HTS	pulse	HWA-5 (init)
85	eop	HTS	pulse	HWA-5 (eop)
86	out_ch0_done	HTS	pulse	HWA-4-5-ch0-done
87	out_ch1_done	HTS	pulse	HWA-4-5-ch1-done
88	out_ch2_done	HTS	pulse	HWA-4-5-ch2-done

Copyright © 2024 Texas Instruments Incorporated

Table 6-127. VPAC Subsystem CTSET Event List (continued)

Index	Event Name	HWA	Event Type	Event Description
89	out_ch3_done	HTS	pulse	HWA-4-5-ch3-done
90	out_ch4_done	HTS	pulse	HWA-4-5-ch4-done
91	out_ch5_done	HTS	pulse	HWA-4-5-ch5-done
92	out_ch6_done	HTS	pulse	HWA-4-5-ch6-done
93	out_ch7_done	HTS	pulse	HWA-4-5-ch7-done
94	out_ch8_done	HTS	pulse	HWA-4-5-ch8-done
95	out_ch9_done	HTS	pulse	HWA-4-5-ch9-done
96	start	HTS	pulse	HWA-6 (task start)
97	done	HTS	pulse	HWA-6 (task done)
98	init	HTS	pulse	HWA-6 (init)
99	eop	HTS	pulse	HWA-6 (eop)
100	pipe_done	HTS	pulse	Pipeline-0 Done
101	pipe_done	HTS	pulse	Pipeline-1 Done
102	pipe_done	HTS	pulse	Pipeline-2 Done
103	pipe_done	HTS	pulse	Pipeline-3 Done
104	pipe_done	HTS	pulse	Pipeline-4 Done
105	pipe_done	HTS	pulse	Pipeline-5 Done
106	pipe_done	HTS	pulse	Pipeline-6 Done
107	spare_dec_0	HTS	pulse	Spare 0 sch decrement pulse (ehost mode)
108	spare_dec_1	HTS	pulse	Spare 1 sch decrement pulse (ehost mode)
109	spare_pend_0	HTS	level	Spare 0 sch pend assertion (ehost mode)
110	spare_pend_1	HTS	level	Spare 1 sch pend assertion (ehost mode)
142:111	utc1_channel_start[31:0] if CTSET_HWA_SL2_DBG='0' else [nf,msc,ldc0,viss0] {sl2 sreq,rreq,creq stall,valid creq}	HTS	pulse	MMR config to select HWA SL2 master ports access control signals
173:143	utc1_channel_start[63:32] if CTSET_RT_UTC_IN='0' else utc0_channel_start[31:0]	UTC	pulse	Channel start
206:175	utc1_ctset_intr[31:0] if CTSET_DMA_SOC_DBG='0' else [ldc0_rd,utc1_ext,utc0_ext] {ext sreq,rreq stall, valid rreq,creq stall,valid creq}	HTS	pulse	MMR config to select external master port access control signals
238:207	utc1_ctset_intr[63:32] if CTSET_RT_UTC_OUT='0' else utc0_ctset_intr[31:0]	UTC	pulse	CTSET event output (for HWA channel_done)
254:239	[utc1,utc0] {wrsreq,1'b0,wr creq stall,wr valid creq, rd rreq stall, rd valid rreq, rd creq stall, rd valid creq}	HTS	pulse	MMR config to select external ctset events or UTC SL2 access control signals

(1) The signals combined to generate the viss_err event are: rawfe_cfg_err_intr || glbce_cfg_err_intr || fcfa_cfg_err_intr || fcc_cfg_err_intr || ee_cfg_err || nsfv4_line_cfg_err || rawfe_h3a_buf_ovrflow_pulse_intr || nsf4v_hblank_err_intr || nsf4v_vblank_err_intr || glbce_vp_err_intr || glbce_hsync_err_intr || glbce_vsync_err_intr || fcc_outif_ovf_err_intr || fcc_hist_read_err_intr || ee_syncovf_err || lse_sl2_rd_err_intr || lse_sl2_wr_err_intr || lse_cal_vp_err_intr.

(2) The signals combined to generate the ldc_err event: pix_iblk_outofbound_intr || mesh_iblk_outofbound_intr || pix_iblk_memovf_intr || mesh_iblk_memovf_intr || ifr_outofbound_intr || int_szovf_intr || vpac_ldc_sl2_wr_err_intr || vpac_ldc_vbusm_rd_err_intr.

The VPAC subsystem also supports halting an execution pipeline using external or user halt request. Halting a pipeline is achieved by not granting thread start to HWA. Once halted, VPAC waits for an external sync trigger

Copyright © 2024 Texas Instruments Incorporated

or config resume to restart execution. VISS, MSC and NF modules halt at line boundary, and LDC halts at block boundary.

The VPAC subsystem supports generic debug capability/features and ARM cross trigger interface for debug triggers:

- Halting the HWA threads on debug request input at logical trigger boundary.
- Continue/restart VPAC from halted state (step/continue) based on external sync trigger and HTS_DBG_CNTL register.
- Debug compliant dbg_attn output after processing external halt request and when VPAC halts debug enabled pipelines.
- CTI (Cross Trigger Interface) compliant out halted trigger after processing external halt request and reaching halted state of VPAC.
- Stalling all pipelines of VPAC (with configurable option to disable impact of debug features on a pipeline).
- Indicate to Host through readable debug_rdy bit in HTS to indicate readiness of HWA resource in case of halted (dbgattn asserted)

The VPAC subsystem debug capability supported is as captured in the HTS_DBG_CAP register. Ext_halt, ext_sync, hwa_halted are CTI (Cross Trigger Interface) mapped async trigger interface (refer to the async protocol of trigger interface in the ARM® CoreSight™ Architecture Specification).

The VISS module, when used in OTF mode, must not be included as debuggable module within VPAC. VISS must be disabled (HTS: debug pipeline disable) to respond to halt request, otherwise behavior is undefined. The VISS module, when used in memory to memory mode, can be halted. But VISS LUTs must not be read during halted state, as it can corrupt the design pipeline.

6.7.2.11 VPAC Subsystem Internal Diagnostic Features

The internal diagnostic features for the VPAC subsystem in HW are as follows:

1. SECDED/ECC:
   - ECC on all LUT memories and ECC on control lines of interconnects
   - No ECC on Pixel Buffers, or memories containing predictors, motion vectors or disparity data
   - No ECC on Interconnect data
2. Supports PBIST interfaces for all memories (for periodic runtime memory).
3. Supports logic BIST to detect permanent fault.
4. Parallel Signature Analyzer (PSA) based signature generation for all data outputs for faster BIST on Logic and MMRs. The PSA is 32-bit CRC based.
5. Watchdog Timers (per HW processing threads) to detect hang condition (will raise interrupt to external host).
6. K3 compliant diagnostic MMRs to log fault addresses and/or error/hang reason.
7. Supports external host access to all of its MMR space and critical memories while suspended due to error.
8. Enables firewalling on all access into SL2 and config space through NAVSS. The default configuration path is point-2-point connection from MCU R5F, so it does not need firewalling.

6.7.2.11.1 Parallel Signature Analysis (PSA)

The VPAC subsystem supports generating CRC signature on each output channel. The LSE module inside a HWA, implements a CRC signature capture mechanism, wherein it samples pixel (except H3A where it is fixed 32-bit) from each valid transfer from HWA core to LSE for each output channel. This CRC signature updates stop at end of frame and signature is available for reference comparison.

The VISS, NF and MSC modules are line based, hence in CRC signature calculation it is needed to scan frame in raster order (left to right and top to bottom). LDC being 2D block based processing needs different mechanism to compute signature. For LDC, pixels are scanned in raster order within block (left to right and from top to bottom), then blocks are scanned in raster order inside region and enabled regions are scanned in raster order within frame.

6.7.2.12 VPAC Subsystem Security Features

The VPAC subsystem implements region basead firewall on slave endpoints within config CBASS and SL2 CBASS. Both CBASS is driven by independent DMSC FW interface from SoC level. All internal masters on SL2 SCR are configured with ISC. Figure 6-50 provides FW/ISC summary.

Copyright © 2024 Texas Instruments Incorporated

Module	Interconnet	Security component	Parameter	Endpoints	FW_ID(dec)	FW/ISC/QoS config address offset(hex)	privID reset value(dec)
VPAC	SCRP	Region FW	2 region	ECC AGG	6048	0	
			1 region	VPAC_TOP	6049	400	
			8 region	INTD	6050	800	
			8 region	HTS	6051	C00	
			1 region	CTSET	6052	1000	
			4 region	VISS0	6053	1400	
			4 region	LDC0	6055	1C00	
			2 region	MSC	6057	2400	
			1 region	NF	6058	2800	
			8 region	UTC0	6059	2C00	
			8 region	UTC1	6060	3000	
	SCRM (SL2)	ISC	1 port x 1 initiator	VISS0		0	213
			1 port x 1 initiator	LDC0		800	215
			1 port x 2 initiator	MSC		1000	217
			1 port x 1 initiator	NF		1400	218
		Region FW	8 region	SL2 Bank0	6080	0	
			8 region	SL2 Bank1	6081	400	
			8 region	SL2 Bank2	6082	800	
			8 region	SL2 Bank3	6083	C00	

vpac-005

Figure 6-50. VPAC Subsystem FW/ISC Configuration

6.7.2.13 VPAC Subsystem Programmer's Guide

6.7.2.13.1 Initialization Sequence

The VPAC subsystem implements 6 configurable pipelines. A pipeline is a combination of several input DMA channels, compute logic and output DMA channels. Few possible configurations for a pipeline are provided in Table 6-128. Each accelerator DMA connection (consumer or producer) is fixed, if enabled. These pipelines will operate independent of each other and their numbering can be configured differently, but supported combination remains the same. One accelerator thread cannot be part of two pipelines simultaneously (for example combination 6 cannot be combined with 2, 3 and 5).

Table 6-128. VPAC Subsystem Pipeline Options

Pipeline	0	1	2	3	4	5	6	7
UTC-VISS0 IN	Y	N	N	N	N	N	N	N
VISS0	Y	N	N	N	N	N	N	N
UTC-VISS0 OUT	Y	N	N	N	N	N	N	N
LDC0	N	N	Y	N	N	N	Y	Y
UTC-LDC0 OUT	N	N	Y	N	N	N	LDC-Y & UV	LDC-Y & UV
UTC-MSC0 IN	N	N	N	Y	N	N	N	Y
MSC0	N	N	N	Y	N	N	LDC-Y/UV	LDC-Y/UV
UTC-MSC0 OUT	N	N	N	Y	N	N	Y	Y
UTC-MSC1 IN	N	N	N	N	Y	N	N	Y
MSC1	N	N	N	N	Y	N	LDC-Y/UV	LDC-Y/UV
UTC-MSC1 OUT	N	N	N	N	Y	N	Y	Y
UTC-NF IN	N	N	N	N	N	Y	N	Y
NF	N	N	N	N	N	Y	LDC-Y/UV	LDC-Y/UV
UTC-NF OUT	N	N	N	N	N	Y	Y	Y
Spare 0	-	-	-	-	-	-	-	Y
Spare 1	-	-	-	-	-	-	-	Y
Spare 2	-	-	-	-	-	-	-	Y
Spare 3	-	-	-	-	-	-	-	Y

The sequence in Figure 6-51 describes a high level initialization sequence for the VPAC pipelines. The parameters pertaining to frame resolution, input and output dependencies, data movement and algorithm need to be updated for every pipeline.

Copyright © 2024 Texas Instruments Incorporated

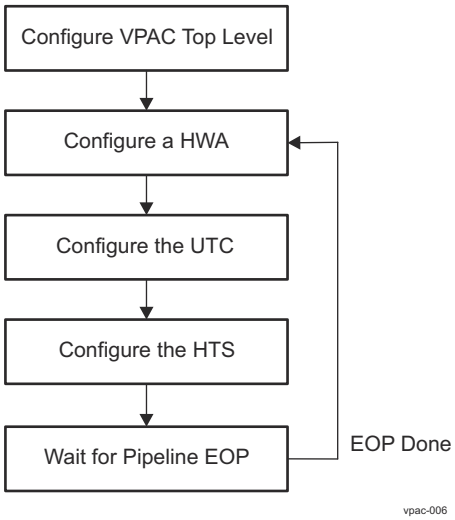


Figure 6-51. VPAC Subsystem Pipeline Processing Initialization Sequence

6.7.2.13.2 VISS Configuration

Refer to Section *VPAC Vision Imaging Subsystem (VISS)* for complete details about its configuration.

The VISS operates in line mode or frame mode:

- Line mode (when VISS_LSE_CFG_LSE[5] IN_CH_SYNC_MODE = 0): Every ‘tstart’ starts with loading a line and produces a complete line (except due to line delay through VISS pipeline there is no output data for ‘tstart’ count equals to VISS_LSE_DST_BUF_ATTR0_j[31-25] LOUT_SKIP_INIT register field setting for each channel and no input buffer loading after ‘tstart’ count goes beyond frame height.
  - Minimum horizontal blanking needs to be configured in LSE (HzlatencyLine)= aggregation of horizontal latency of enabled components: 50(RFE) + 62(GLBCE) + 25 (CFA) + 25(FCC) + 26(EE) + 44(NSF4) ~ for all enabled VISS sub-modules.
- Frame mode, recommended, VISS_LSE_CFG_LSE[5] IN_CH_SYNC_MODE = 0): Every ‘tstart’ starts with loading a line till ‘tstart’ count equals to VISS_LSE_DST_BUF_ATTR0_j[31-25] LOUT_SKIP_INIT register field setting for any of the enabled output channels. After this, input line loading is independent from ‘tstart’ and only output data is synchronized with ‘tstart’. Minimum buffer depth on input channels are so chosen to enforce input data availability throughout frame.
  - Minimum horizontal blanking needs to be configured in the LSE module = MAX of horizontal blanking of enabled components:
    - MaxVertBlanking = 2(DPC) + 1(GLBCE) + 3(CFA) + 15(NSFV4)
    - LOUT_SKIP_INIT(S8) = MaxVertBlanking
    - LOUT_SKIP_INIT(Y12) = MaxVertBlanking + 2(EE enabled for Y12)
    - LOUT_SKIP_INIT(UV12) = MaxVertBlanking + 1(420 Chroma)
    - LOUT_SKIP_INIT(Y8) = MaxVertBlanking + 2(EE enabled for Y8)
    - LOUT_SKIP_INIT(UV8) = MaxVertBlanking + 1(420 Chroma)

6.7.2.13.2.1 VISS UTC Configuration

The VISS HWA in a non-streaming mode (memory to memory mode) needs a maximum of 3 input buffers to be loaded by UTC before it can start its processing. The 6 VISS HWA outputs produced need to be stored by UTC into DDR. Refer to chapter *Data Routing Unit (DRU)*, about configuring the UTC for data load and store for VISS processing.

The 3 UTC channels needs to be set up for input data fetch:

- VISS Exposure-0
- VISS Exposure-1
- VISS Exposure-2

The 6 VISS outputs iare mapped to independent UDMA channels, as follows:

Copyright © 2024 Texas Instruments Incorporated

- VISS-Y-12b
- VISS-UV-12b
- VISS-Y-8b
- VISS-UV-8b
- VISS-S-8b
- VISS-H3A-AE/AF

The UTC channel configuration assumes different channels for each input and output data. SL2 storage needs to allocate minimum ping-pong buffer for each input/output data. Buffer depth is configurable. For each UTC trigger, single line of data is either loaded from DDR or stored into DDR. The HTS module manages start of UTC channels.

6.7.2.13.2.2 VISS HTS Configuration for Line Mode

The HWA0 scheduler in HTS is used for VISS. One example for HTS programming to map VISS on pipeline=0 is as below:

- Assume frame resolution to be 1920x1080 with all components in VISS pipeline enabled, and EE on Y12.
- postLoad = max(LOUT_SKIP_INIT for Y12, LOUT_SKIP_INIT for Y8, LOUT_SKIP_INIT for S8), thr = 1;
- Producer DMA Depth min = Th+1;

```
// HWA0 Scheduler Programming (keep default programing for not included parameters)
HTS->HWA0_wdtimer->wdtimer_en = 1; //Activate WD
HTS->HWA0_scheduler_control->pipeline_num = 0; // Belongs to pipeline 0
```

```
// HWA0 consumer and producer control
HTS->HWA0_cons_control->cons0_en = 1; // Enable exp-0 Fetch
HTS->HWA0_cons_control->cons1_en = 1; // Enable exp-1 Fetch
HTS->HWA0_cons_control->cons2_en = 1; // Enable exp-2 Fetch
HTS->HWA0_cons_control->prod0_select = 0; // Fixed to UTC
HTS->HWA0_cons_control->prod1_select = 0; // Fixed to UTC
HTS->HWA0_cons_control->prod2_select = 0; // Fixed to UTC
```

```
HTS->HWA0_prod0_control->prod_en = 1; // Enable Producer socket
HTS->HWA0_prod0_buf_control->depth = 2; // ping-pong buffer
HTS->HWA0_prod0_control->cons_select = 0; // Fixed to UTC
HTS->HWA0_prod1_control->prod_en = 1; // Enable Producer socket
HTS->HWA0_prod1_buf_control->depth = 2; // ping-pong buffer
HTS->HWA0_prod1_control->cons_select = 0; // Fixed to UTC
HTS->HWA0_prod2_control->prod_en = 1; // Enable Producer socket
HTS->HWA0_prod2_buf_control->depth = 2; // ping-pong buffer
HTS->HWA0_prod2_control->cons_select = 0; // Fixed to UTC
HTS->HWA0_prod3_control->prod_en = 1; // Enable Producer socket
HTS->HWA0_prod3_buf_control->depth = 2; // ping-pong buffer
HTS->HWA0_prod3_control->cons_select = 0; // Fixed to UTC
HTS->HWA0_prod4_control->prod_en = 1; // Enable Producer socket
HTS->HWA0_prod4_buf_control->depth = 2; // ping-pong buffer
HTS->HWA0_prod4_control->cons_select = 0; // Fixed to UTC
HTS->HWA0_prod5_control->prod_en = 1; // Enable Producer socket
HTS->HWA0_prod5_buf_control->depth = 2; // ping-pong buffer
HTS->HWA0_prod5_control->cons_select = 0; // Fixed to UTC
```

```
// Exp-0 Fetch
HTS->DMA0_scheduler_control->pipeline_num = 0; // Belongs to pipeline 0
HTS->DMA0_hop->hop = 1; // Head of the pipe scheduler
HTS->DMA0_hop->hop_thread_count = 1080; // Assuming frame height of 1080
HTS->DMA0_scheduler_control->dma_channel_no = 0; // Assign appropriate UDMA channel
HTS->DMA0_prod0_control->prod_en = 1; // Enable producer socket
HTS->DMA0_prod0_buf_control->depth = 2; // ping-pong
HTS->DMA0_prod0_count->count_postload = 23; // Additional triggers for vertical blanking to flush
out the VISS.
```

```
// Exp-1 Fetch
HTS->DMA1_scheduler_control->pipeline_num = 0; // Belongs to pipeline 0
HTS->DMA1_hop->hop = 1; // Head of the pipe scheduler
HTS->DMA1_hop->hop_thread_count = 1080; // Assuming frame height of 1080
HTS->DMA1_scheduler_control->dma_channel_no = 1; // Assign appropriate DMA channel
HTS->DMA1_prod0_control->prod_en = 1; // Enable producer socket
```


Copyright © 2024 Texas Instruments Incorporated

```
HTS->DMA1_prod0_buf_control->depth = 2; // ping-pong
HTS->DMA1_prod0_count->count_postload = 23; // Additional triggers for vertical blanking to flush
out the VISS.
```

```
// Exp-2 Fetch
HTS->DMA2_scheduler_control->pipeline_num = 0; // Belongs to pipeline 0
HTS->DMA2_hop->hop = 1; // Head of the pipe scheduler
HTS->DMA2_hop->hop_thread_count = 1080; // Assuming frame height of 1080
HTS->DMA2_scheduler_control->dma_channel_no = 2; // Assign appropriate DMA channel
HTS->DMA2_prod0_control->prod_en = 1; // Enable producer socket
HTS->DMA2_prod0_buf_control->depth = 2; // ping-pong
HTS->DMA2_prod0_count->count_postload = 23; // Additional triggers for vertical blanking to flush
out the VISS.
```

```
// VISS output Y-12
HTS->DMA240_scheduler_control->pipeline_num = 0; // Belongs to pipeline 0
HTS->DMA240_scheduler_control->DMA_channel_no = 3; // Assign appropriate DMA channel
HTS->DMA240_cons0_control->cons_en = 1; // Enable consumer socket
// VISS output UV-12
HTS->DMA241_scheduler_control->pipeline_num = 0; // Belongs to pipeline 0
HTS->DMA241_scheduler_control->DMA_channel_no = 4; // Assign appropriate DMA channel
HTS->DMA241_cons0_control->cons_en = 1; // Enable consumer socket
// VISS output Y-8
HTS->DMA242_scheduler_control->pipeline_num = 0; // Belongs to pipeline 0
HTS->DMA242_scheduler_control->DMA_channel_no = 5; // Assign appropriate DMA channel
HTS->DMA242_cons0_control->cons_en = 1; // Enable consumer socket
// VISS output UV-8
HTS->DMA243_scheduler_control->pipeline_num = 0; // Belongs to pipeline 0
HTS->DMA243_scheduler_control->DMA_channel_no = 6; // Assign appropriate DMA channel
HTS->DMA243_cons0_control->cons_en = 1; // Enable consumer socket
// VISS output S-8
HTS->DMA244_scheduler_control->pipeline_num = 0; // Belongs to pipeline 0
HTS->DMA244_scheduler_control->DMA_channel_no = 7; // Assign appropriate DMA channel
HTS->DMA244_cons0_control->cons_en = 1; // Enable consumer socket
// VISS output H3A-AE
HTS->DMA245_scheduler_control->pipeline_num = 0; // Belongs to pipeline 0
HTS->DMA245_scheduler_control->DMA_channel_no = 8; // Assign appropriate DMA channel
HTS->DMA245_cons0_control->cons_en = 1; // Enable consumer socket
```

```
// Enable Required Schedulers
HTS->HWA0_scheduler_control->sch_en = 1; //VISS0 Scheduler Enable
HTS->DMA0_scheduler_control->sch_en = 1; //Prod DMA for expo0 Enable
HTS->DMA1_scheduler_control->sch_en = 1; //Prod DMA for expo1 Enable
HTS->DMA1_scheduler_control->sch_en = 1; //Prod DMA for expo2 Enable
HTS->DMA240_scheduler_control->sch_en = 1; //Cons DMA for Y12 Enable
HTS->DMA241_scheduler_control->sch_en = 1; //Cons DMA for UV12 Enable
HTS->DMA242_scheduler_control->sch_en = 1; //Cons DMA for Y8 Enable
HTS->DMA243_scheduler_control->sch_en = 1; //Cons DMA for UV8 Enable
HTS->DMA244_scheduler_control->sch_en = 1; //Cons DMA for S8 Enable
HTS->DMA245_scheduler_control->sch_en = 1; //Cons DMA for H3A Enable
```

```
// Enable Pipeline
HTS-> PIPELINE_CONTROL_0 ->pipe_en = 1; // Enable VISS0 pipeline# 0
```

6.7.2.13.2.3 VISS HTS Configuration for Frame Mode

The HWA0 scheduler in HTS is used for VISS0. The HTS programming is as follows:

- Assume frame resolution to be 1920x1080 with all components in VISS pipeline enabled.
- Var = ceil((1024/in_pix_cntrsz + 808/out_pix_cntrz + 8 + HzlatencyLine) / (Framewidth in pixels + lse.vp_hblnk_cnt));
- postLoad = max(lout_init_skip for Y12, lout_init_skip for Y8, lout_init_skip for S8) + Var;
- Th = Var + 1;
- Producer DMA Depth min = Th + 1;

Copyright © 2024 Texas Instruments Incorporated

The rest of the HTS registers configuration remains same as in Line Mode, except for below fields (based on earlier equations):

```
HTS-> dma0_prod0_buf_control.threshold = Var+1;
HTS-> dma0_prod0_buf_control.depth = Var+2;
HTS-> dma0_prod0_count.count_postload = 23+Var;
```

6.7.2.13.3 VISS OTF Configuration

Refer to the programming information within Section 6.7.2.13.2, *VISS Configuration*, for common details.

The HTS configuration for OTF operation will change as follows:

```
HTS->HWA0_cons_control->cons0_en = 0; // Enable exp-0 Fetch
HTS->HWA0_cons_control->cons1_en = 0; // Enable exp-1 Fetch
HTS->HWA0_cons_control->cons2_en = 0; // Enable exp-2 Fetch
HTS->HWA0_scheduler_control-> strm_en = 1; // streaming input enabled
HTS->DMA0_prod0_control->prod_en = 0
HTS->DMA1_prod0_control->prod_en = 0
HTS->DMA2_prod0_control->prod_en = 0
// Enable Required Schedulers
HTS->DMA0_scheduler_control->sch_en = 0; //Prod DMA for expo0 Enable
HTS->DMA1_scheduler_control->sch_en = 0; //Prod DMA for expo1 Enable
HTS->DMA1_scheduler_control->sch_en = 0; //Prod DMA for expo2 Enable
```

6.7.2.13.4 LDC Configuration (LDC Connected to MSC0, NF and DMA)

In this example configuration, pipeline 2 is used for LDC. The LDC output is stored into DDR, and connected to MSC0 and NF concurrently.

The LDC DMA connection are these: DMA272 (Y12), DMA273 (UV12), DMA274 (Y8), DMA275 (UV8)

The DMA connections for MSC and NF are as follows:

- MSC0: DMA[304:313]
- MSC1: DMA[304:313]
- NF: DMA336

The parameters assumed for this example is: LDC OBW = 32, OBH = 32, Frame width = 1920

For more details on LDC configuration, see Section *VPAC Lens Distortion Correction (LDC)*.

6.7.2.13.4.1 LDC DMA Configuration

The LDC does not need thread management on its input buffer. The LDC uses its own DMA engine to fetch required input data. The LDC needs DMA to write out its output data and needs HTS interface. Refer to chapter *Data Routing Unit (DRU)*, about DMA configuration for data store. Maximum 13 DMA channels need to be set up for writing out output data.

6.7.2.13.4.2 LDC HTS Configuration

The HWA2 scheduler in HTS is used for LDC, the HWA4 is used for MSC0, and HWA6 is used for NF. The HTS programming is as follows:

```
// HWA2/LDC Scheduler Programming (keep default programing for rest of parameters)
HTS->HWA2_wdtimer->wdtimer_en = 1; //Activate WD
HTS->HWA2_scheduler_control->pipeline_num = 2; // Belongs to pipeline 2
```

```
// HWA2/LDC Producer control
HTS->HWA2_prod0_control->prod_en = 1;        // Enable Producer socket
HTS->HWA2_prod0_buf_control->depth = 120;        // (FW/OBW)*2; ping-pong buffer for 2 rows of
32x32 blocks
HTS->HWA2_prod0_control->cons_select = 0;    // Fixed to DMA
HTS->HWA2_prod0_buf_control->threshold = 1;      // threshold value
HTS->HWA2_prod0_count-> count_postload=0;
HTS->HWA2_prod0_count-> count_preload=0;
HTS->HWA2_pa0_control-> pa_ps_maxcount = 32; // OBH
HTS->HWA2_pa0_control-> pa_cs_maxcount = 60; // (FW/OBW)
```


Copyright © 2024 Texas Instruments Incorporated

```
HTS->HWA2_pa0_control-> pa_buf_cntl = ‘0’;   //
HTS->HWA2_pa0_control-> pa_enable = ‘1’ ;
```

```
// HWA2/LDC Producer control
HTS->HWA2_prod1_control->prod_en = 1;        // Enable Producer socket
HTS->HWA2_prod1_buf_control->depth = 2;         // ping-pong buffer for chroma blocks
HTS->HWA2_prod1_control->cons_select = 0;    // Fixed to DMA
HTS->HWA2_prod1_buf_control->threshold = 1;      // threshold value
HTS->HWA2_prod1_count-> count_postload=0;
HTS->HWA2_prod1_count-> count_preload=0;
HTS->HWA2_pa1_control-> pa_enable = ‘0’;     // No pattern adaptation for Chroma
```

```
// HWA2/LDC Producer control
HTS->HWA2_prod4_control->prod_en = 1;        // Enable Producer socket
HTS->HWA2_prod4_buf_control->depth = 120;        // (FW/OBW)*2; ping-pong buffer for 2 rows of
32x32 blocks
HTS->HWA2_prod4_control->cons_select = 33;    // Select MSC0-IN
HTS->HWA2_prod4_buf_control->threshold = 5;      // threshold value
HTS->HWA2_prod4_count-> count_postload=2;
HTS->HWA2_prod4_count-> count_preload=2;
HTS->HWA2_pa4_control-> pa_ps_maxcount = 32; // OBH
HTS->HWA2_pa4_control-> pa_cs_maxcount = 60; // (FW/OBW)
HTS->HWA2_pa4_control-> pa_buf_cntl = ‘1’;   // Apply threshold/pos/preload on PA output
HTS->HWA2_pa4_control-> pa_enable = ‘1’;
```

```
// HWA2/LDC Producer control
HTS->HWA2_prod6_control->prod_en = 1;        // Enable Producer socket
HTS->HWA2_prod6_buf_control->depth = 120;        // (FW/OBW)*2; ping-pong buffer for 2 rows of
32x32 blocks
HTS->HWA2_prod6_control->cons_select = 49;    // Select NF-IN
HTS->HWA2_prod6_buf_control->threshold = 5;      // threshold value
HTS->HWA2_prod6_count-> count_postload=2;
HTS->HWA2_prod6_count-> count_preload=2;
HTS->HWA2_pa6_control-> pa_ps_maxcount = 32; // OBH
HTS->HWA2_pa6_control-> pa_cs_maxcount = 60; // (FW/OBW)
HTS->HWA2_pa6_control-> pa_buf_cntl = ‘1’;   // Apply threshold/pos/preload on PA output
HTS->HWA2_pa6_control-> pa_enable = ‘1’ ;
```

```
// configure MSC0 (10 output)
HTS->HWA4_scheduler_control->pipeline_num = 2; // Belongs to pipeline 2
HTS->HWA4_cons_control->cons0_en = 1; // MSC in
HTS->HWA4_cons_control->prod0_select = 37; // connect to LDC-MSC producer
HTS->HWA4_prod0_control->prod_en = 1; // Enable Producer socket
HTS->HWA4_prod0_buf_control->depth = 2; // ping-pong buffer
HTS->HWA4_prod0_control->cons_select = 0; // Fixed to DMA
HTS->HWA4_prod1_control->prod_en = 1; // Enable Producer socket
HTS->HWA4_prod1_buf_control->depth = 2; // ping-pong buffer
HTS->HWA4_prod1_control->cons_select = 0; // Fixed to UDMA
HTS->HWA4_prod2_control->prod_en = 1; // Enable Producer socket
HTS->HWA4_prod2_buf_control->depth = 2; // ping-pong buffer
HTS->HWA4_prod2_control->cons_select = 0; // Fixed to DMA
HTS->HWA4_prod3_control->prod_en = 1; // Enable Producer socket
HTS->HWA4_prod3_buf_control->depth = 2; // ping-pong buffer
HTS->HWA4_prod3_control->cons_select = 0; // Fixed to DMA
HTS->HWA4_prod4_control->prod_en = 1; // Enable Producer socket
HTS->HWA4_prod4_buf_control->depth = 2; // ping-pong buffer
HTS->HWA4_prod4_control->cons_select = 0; // Fixed to DMA
HTS->HWA4_prod5_control->prod_en = 1; // Enable Producer socket
HTS->HWA4_prod5_buf_control->depth = 2; // ping-pong buffer
HTS->HWA4_prod5_control->cons_select = 0; // Fixed to DMA
HTS->HWA4_prod6_control->prod_en = 1; // Enable Producer socket
HTS->HWA4_prod6_buf_control->depth = 2; // ping-pong buffer
HTS->HWA4_prod6_control->cons_select = 0; // Fixed to DMA
HTS->HWA4_prod7_control->prod_en = 1; // Enable Producer socket
HTS->HWA4_prod7_buf_control->depth = 2; // ping-pong buffer
HTS->HWA4_prod7_control->cons_select = 0; // Fixed to DMA
HTS->HWA4_prod8_control->prod_en = 1; // Enable Producer socket
HTS->HWA4_prod8_buf_control->depth = 2; // ping-pong buffer
HTS->HWA4_prod8_control->cons_select = 0; // Fixed to DMA
HTS->HWA4_prod9_control->prod_en = 1; // Enable Producer socket
```


Copyright © 2024 Texas Instruments Incorporated

```
HTS->HWA4_prod9_buf_control->depth = 2; // ping-pong buffer
HTS->HWA4_prod9_control->cons_select = 0; // Fixed to DMA
```

```
// configure NF
HTS->HWA6_scheduler_control->pipeline_num = 2; // Belongs to pipeline 2
HTS->HWA6_cons_control->cons0_en = 1; // NF in
HTS->HWA6_cons_control->prod_select = 39; // connect to LDC-NF producer
HTS->HWA6_prod0_control->prod_en = 1; // Enable Producer socket
HTS->HWA6_prod0_buf_control->depth = 2; // ping-pong buffer
HTS->HWA6_prod0_control->cons_select = 0; // Fixed to DMA
```

```
// configure DMA
HTS->DMA304_scheduler_control->pipeline_num = 2; // Belongs to pipeline 0
HTS->DMA304_scheduler_control->dma_channel_no = 0x12; // Assign appropriate DMA channel
HTS->DMA304_cons0_control->cons_en = 1; // Enable consumer socket
HTS->DMA305_scheduler_control->pipeline_num = 2; // Belongs to pipeline 0
HTS->DMA305_scheduler_control->dma_channel_no = 0x13; // Assign appropriate DMA channel
HTS->DMA305_cons0_control->cons_en = 1; // Enable consumer socket
HTS->DMA306_scheduler_control->pipeline_num = 2; // Belongs to pipeline 0
HTS->DMA306_scheduler_control->dma_channel_no = 0x14; // Assign appropriate DMA channel
HTS->DMA306_cons0_control->cons_en = 1; // Enable consumer socket
HTS->DMA307_scheduler_control->pipeline_num = 2; // Belongs to pipeline 0
HTS->DMA307_scheduler_control->dma_channel_no = 0x15; // Assign appropriate DMA channel
HTS->DMA307_cons0_control->cons_en = 1; // Enable consumer socket
HTS->DMA308_scheduler_control->pipeline_num = 2; // Belongs to pipeline 0
HTS->DMA308_scheduler_control->dma_channel_no = 0x16; // Assign appropriate DMA channel
HTS->DMA308_cons0_control->cons_en = 1; // Enable consumer socket
HTS->DMA309_scheduler_control->pipeline_num = 2; // Belongs to pipeline 0
HTS->DMA309_scheduler_control->dma_channel_no = 0x17; // Assign appropriate DMA channel
HTS->DMA309_cons0_control->cons_en = 1; // Enable consumer socket
HTS->DMA310_scheduler_control->pipeline_num = 2; // Belongs to pipeline 0
HTS->DMA310_scheduler_control->dma_channel_no = 0x18; // Assign appropriate DMA channel
HTS->DMA310_cons0_control->cons_en = 1; // Enable consumer socket
HTS->DMA311_scheduler_control->pipeline_num = 2; // Belongs to pipeline 0
HTS->DMA311_scheduler_control->dma_channel_no = 0x19; // Assign appropriate DMA channel
HTS->DMA311_cons0_control->cons_en = 1; // Enable consumer socket
HTS->DMA312_scheduler_control->pipeline_num = 2; // Belongs to pipeline 0
HTS->DMA312_scheduler_control->dma_channel_no = 0x1A; // Assign appropriate DMA channel
HTS->DMA312_cons0_control->cons_en = 1; // Enable consumer socket
HTS->DMA313_scheduler_control->pipeline_num = 2; // Belongs to pipeline 0
HTS->DMA313_scheduler_control->dma_channel_no = 0x1B; // Assign appropriate DMA channel
HTS->DMA313_cons0_control->cons_en = 1; // Enable consumer socket
HTS->DMA336_scheduler_control->pipeline_num = 2; // Belongs to pipeline 0
HTS->DMA336_scheduler_control->dma_channel_no = 0x1C; // Assign appropriate DMA channel
HTS->DMA336_cons0_control->cons_en = 1; // Enable consumer socket
HTS->DMA272_scheduler_control->pipeline_num = 2; // Belongs to pipeline 0
HTS->DMA272_scheduler_control->dma_channel_no = 0x1D; // Assign appropriate DMA channel
HTS->DMA272_cons0_control->cons_en = 1; // Enable consumer socket
HTS->DMA273_scheduler_control->pipeline_num = 2; // Belongs to pipeline 0
HTS->DMA273_scheduler_control->dma_channel_no = 0x1E; // Assign appropriate DMA channel
HTS->DMA273_cons0_control->cons_en = 1; // Enable consumer socket
```

```
// Enable Required Schedulers
HTS->HWA2_scheduler_control->sch_en = 1; //LDC0 Scheduler Enable
HTS->HWA4_scheduler_control->sch_en = 1; //MSC0 Scheduler Enable
HTS->HWA6_scheduler_control->sch_en = 1; //NF Scheduler Enable
HTS->DMA272_scheduler_control->sch_en = 1; //Cons DMA for LDC Y12 Enable
HTS->DMA273_scheduler_control->sch_en = 1; //Cons DMA for LDC UV12 Enable
HTS->DMA304_scheduler_control->sch_en = 1; //Cons DMA for MSC0-O0 Enable
HTS->DMA305_scheduler_control->sch_en = 1; //Cons DMA for MSC0-O1 Enable
HTS->DMA306_scheduler_control->sch_en = 1; //Cons DMA for MSC0-O2 Enable
HTS->DMA307_scheduler_control->sch_en = 1; //Cons DMA for MSC0-O3 Enable
HTS->DMA308_scheduler_control->sch_en = 1; //Cons DMA for MSC0-O4 Enable
HTS->DMA309_scheduler_control->sch_en = 1; //Cons DMA for MSC0-O5 Enable
HTS->DMA310_scheduler_control->sch_en = 1; //Cons DMA for MSC0-O6 Enable
HTS->DMA311_scheduler_control->sch_en = 1; //Cons DMA for MSC0-O7 Enable
HTS->DMA312_scheduler_control->sch_en = 1; //Cons DMA for MSC0-O8 Enable
HTS->DMA313_scheduler_control->sch_en = 1; //Cons DMA for MSC0-O9 Enable
HTS->DMA336_scheduler_control->sch_en = 1; //Cons DMA for NF Output Enable
```

```
// Enable Pipeline
HTS->HTS_control->pipe_en2 = 1; // Enable LDC-MSC-NF pipeline# 2
```


Copyright © 2024 Texas Instruments Incorporated

6.7.2.13.5 Real-time Operating Requirements

SW needs to interfere at frame level for each pipeline.

VPAC subsystem can have a maximum of 6 pipelines at any given time. Refer to Section *VPAC Hardware Thread Scheduler (HTS)*, for further details.

Copyright © 2024 Texas Instruments Incorporated

6.7.3 VPAC Vision Imaging Subsystem (VISS)

Vision ISS (Imaging sub-system) is set of raw pixel processing functions to enhance and reconstruct image and convert to various color formats for rest of processing. This is mainly tuned for ADAS market space, specifically front camera vision processing. Vision ISS is Hardware IP that will be used in ADAS chips. Vision ISS does on-the-fly (OTF) processing on raw pixels captured from camera sensor (via CSI-RX). It also does memory to memory processing (via DDR) for data captured from other sources e.g. parallel port or video port. Vision ISS is integrated inside VPAC and uses Hardware thread scheduler (HTS) and UTC for control and data transfer.

6.7.3.1 VISS Top Level

This section describes the Vision ISS (VISS) top level details.

6.7.3.1.1 Features Supported

- Performance: up to 1 pixel per clock per HW Instance
- Support for 16 bit input RAW format
- Support for concurrent 12 bit YUV output & 8b YUV output
- Support for multiple RAW formats (e.g. Bayer, RCCC)
- Flexible to handle any 2x2 RAW format
- Enables Support for other color spaces
  - Enables RGB output
  - Support other color spaces e.g. Support for Color saturation and Gray scale for HSV/HSL etc (Hue is not supported)
- Support for multiple WDR /HDR Formats and Up-to 3 exposures merge
- Companded format (12 & 14 bit formats)
- Stagger format of WDR
- Split pixel format of WDR
- Rest of WDR format
- Support of statistics for Auto-Exposure / Auto-white-balance or Auto-Focus (configurable option to select either AE/AWB or AF not both concurrently)
- Advanced Spatial Noise filter with 16-bit support
- Dynamic White balance adaptation
- Histogram on Raw Bayer data
- Support for Flexible CFA for generating multi-plane output for any 2x2 RAW Format sensor configuration
  - Supports RCBC, RCCC, Bayer, RGBC as well as other formats
  - Can generate up to 4 independent color planes
  - Support for up two 3 directions per color plane
  - Adaptive Threshold based on intensity measure
- Support for multiple color format output generation
  - Support traditional 12 bit YUV Output
  - Supports RGB output
  - Supports Saturation outputs (8 bits only)
  - Supports Grayscale/Luma/IR/Clear Output
- Support for flexible output format generation including YUV as well as custom
- Edge enhancement Option to bypass visual enhancement functions (NSF4V and GLBCE) for ADAS vision applications
- Concurrent Human Vision (HV) and Mission Vision(MV) outputs
- Chromatic Abaration Correction

6.7.3.1.2 VISS Block Diagram

The VISS module does on-the-fly processing on raw pixels captured from camera sensor (via CSI RX module). It also does memory to memory processing (via DDR memory) for data captured from other sources at SoC level. Figure 6-52 is simplified block diagram of VISS top level.

Copyright © 2024 Texas Instruments Incorporated

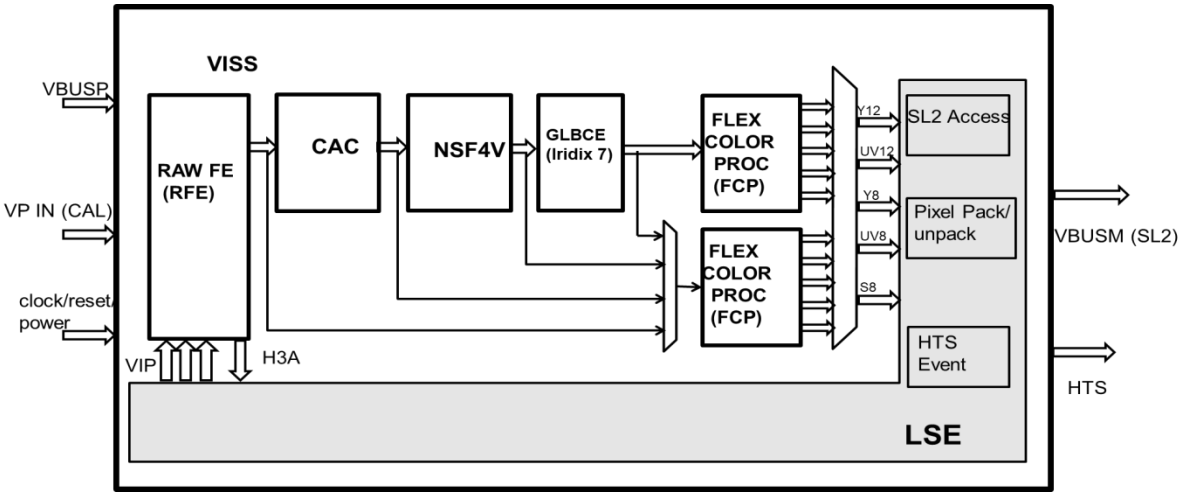


Figure 6-52. VISS Block Diagram

The VISS consists of the following Hardware Accelerators (HWAs):

- RFE (RAW-FE): The RAW Frond End HW block does RAW pixel (that is, Bayer, RCCC, RGBW, etc.) processing on captured image data from sensor and pass-on to NSF4V, GLBCE block, and then to Flexible Color Processing (FCP) HW block for demosaicing and color conversion.
- CAC: Performs Chromatic aberration correction on Red and Blue pixels. CAC IP can be bypassed in the applications where Chromatic aberration correction is not required.
- NSF4V: Spatial Noise Filter supporting generic 2x2 pixel format with 16-bit pixel size. The NSF4V module can be bypassed in the applications where visual enhancement is not desired.
- GLBCE: The GLBCE module is used for dynamic range control within image for visual quality. If contrast enhancement on input image is required for visual quality, the RFE output is processed by the NSF4v and GLBCE blocks. Otherwise, the RFE output is bypassed to FCP.
- FCP: The Flexible Color Processing HW receives data from the GLBCE and does demosaincing and color conversion. The output of FCP is sent to VPAC shared memory to be written into DDR for the rest of the vision processing by programmable processors (for example, DSP or Arm), or other Vision HW blocks (for example, DMPAC).
- LSE: The Load and Store Engine is and infrastructure block, which performs the following functions:
  - CAL video port: The CSI RX module at SoC level receives CSI-2 sensor data, extracts pixels and drives the result data on its video port interface. The video port is mapped onto one of LSE input interfaces for on-the-fly image processing to reduce DDR bandwidth and latency. The LSE also provides horizontal and vertical blanking cycles to allow core data path to settle at proper boundary of line and frame.
  - SL2 memory access: This module supports load/store data from/to SL2 using VBUSM interface. Loaded data from SL2 are passed on to unpacker function of LSE for RFE. Packed data after FCP/H3A processing is written into SL2.
  - Pixel pack/unpack: Source data (512-bit) for RFE processing is loaded from SL2 and passed onto unpacker function for pixel extraction. Extracted pixels are driven on to the video port of RFE. Similarly, the FCP produced pixels are passed through packer function for eventual write into SL2. The H3A generated data is pseudo mapped as pixels of 32-bit for packing purpose and directly driven by the RFE.
  - Event control: The HTS events are generated at line level. These events are routed to LSE to start the processing. For each consumed line, the HTS needs a *task done* indication. For some initial lines of image, there would not be any valid data output. Similarly, the H3A generated data will be at paxel/window height number of lines. For initial lines not producing any valid data, the HTS needs to be indicated with separate mask bits for each output streams. For these initial lines, when there is no valid output due to lines delay inside VISS, the LSE still generates mask output to HTS indicating lack of proper output data.

6.7.3.1.3 VISS Data Flow within VPAC

VISS receives data from sensor through:

- CSI RX video port for OTF processing (Sensor -> CSI RX -> LSE -> RFE -> CAC/NSF4V/GLBCE -> FCP)

Copyright © 2024 Texas Instruments Incorporated

- DMA of VPAC (Sensor -> CAL -> DDR -> VPAC_SL2 -> LSE -> RFE -> CAC/NSF4V/GLBCE -> FCP)

6.7.3.1.3.1 VISS On-the-fly Processing

The VISS operation is controlled through a single HTS thread. For details of the HTS operation refer to Section *VPAC Hardware Thread Scheduler (HTS)*. The VISS configuration is valid for a single or multiple frame. External-host manages VISS configuration at end of each frame, if required. The HTS controls image processing between lines and handles managing shared memory for VISS inputs/outputs. Sub-frame processing is not natively supported. SW must configure VISS and HTS to start initialization process before enabling sensor capture. After the initialization sequence, the HTS will generate a start trigger to VISS. The generation of the start trigger depends on the availability of H3A data out buffer and FCP data out buffer inside the SL2 memory.

At this stage, the LSE waits for CSI RX video port (VP) to start sending valid pixels. As and when VP data starts streaming in, it is routed to the RFE video port. As RFE, CAC/NSF4V/GLBCE, and FCP are running in streaming mode, the FCP indicates to LSE about completion of one line of operation . After receiving valid data out, the HTS will trigger UTC for data store into DDR. Each 'thread done' accompanies output mask for all output buffers. Mask bits indicate to HTS about validity of output buffer as for each thread done all output buffers do not need to be produced. The HTS will issue next start after checking all output buffers availability. The streaming data from CSI RX is stallable upto 2KB storage inside VP memory of CSI RX. Any temporary stall can be absorbed, but prolonged stall is buffer overflow inside CSI RX. In VPAC, VISS produced data is transferred using Real Time fabric of NAVSS/MSMC and low predictable latency needs to be guaranteed.

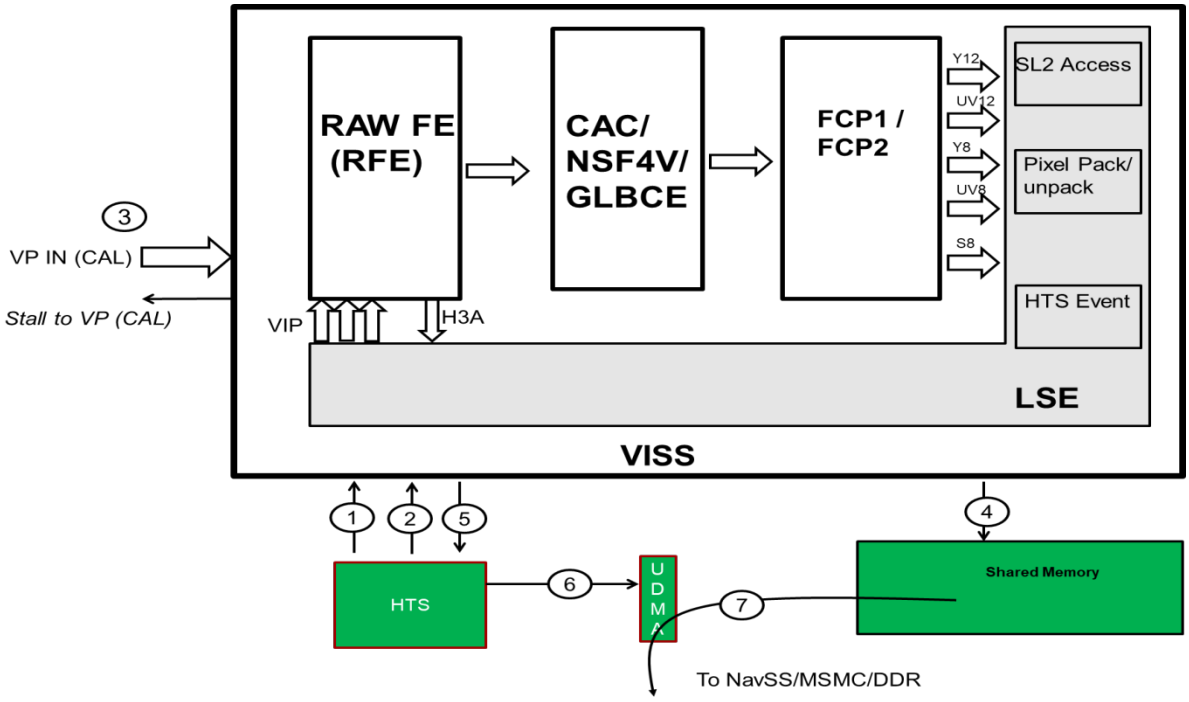


Figure 6-53. VISS On-the-fly Operation

The following OTF operation steps are illustrated in Figure 6-53:

1. *Init* from HTS to initialize VISS.
2. Thread start. Step 1+2 prior to enabling CSI RX + PHY + sensor.
3. Video port data streaming. Unit of operation inside VISS at line level.
4. Output data produced by FCP and H3A at line level. H3A will produce data only at those lines which aligns with paxel_height/window_height.
5. Production of 1 line depends on consumption of 1 line on VP interface. 'Thread done' triggered after completion of 1 processed line write into SL2. In case there is no valid data to be written into SL2, 'thread done' is generated after consumption of 1 line of data from VP. Note: *Horizontal blanking will be configured inside LSE to fecilitate writing complete line out of FCP*.

Copyright © 2024 Texas Instruments Incorporated

6. HTS triggers DMA to store output buffer from SL2 to DDR.
7. DMA loads data from SL2 and writes into DDR.

The following assumptions are made for the OTF operation described above:

- UDMA write is mapped onto Real Time Thread for guaranteed QoS. Channel done by UTC must be generated after completion of write into DDR. Next thread start depend on availability of minimum one buffer for each output line in SL2. More buffers in SL2 can take care of instantaneous drop in UDMA transfer capacity.
- Cycle gap between HTS done and next start must be minimal. CSI RX VP interface will be stalled whenever LSE input buffer on VP interface reaches near full condition.
- Configurable vertical latency times 'Tstart' generated by HTS to let RFE + NSF4V + GLBCE + FCP flush its internal pipe, even though last line of current frame is already fed into RFE.
- Buffer dependencies:
  - H3A output buffer (AE or AF)
  - FCP output buffer (Y12, YV12, Y8, UV8, S8)

6.7.3.1.3.1.1 Non-WDR or Companded WDR Sensors

VISS is enabled for single input data when either sensor does not support WDR or sensor does on-chip WDR merge and pass on merged data after companding to 12 bit / 14 bits as shown in Figure 6-54.

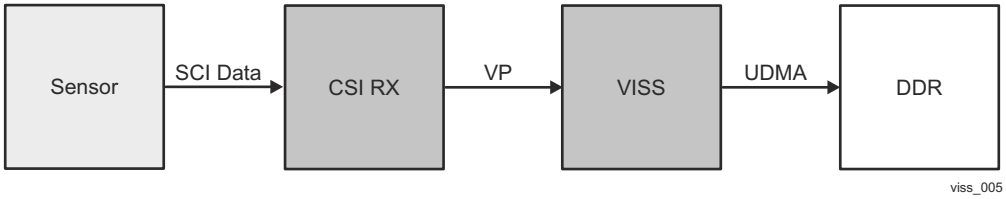


Figure 6-54. VISS Non-WDR/Companded: High Level

6.7.3.1.3.2 VISS Memory to Memory Image Processing

CSI RX writes CSI-2 received pixel data into DDR. Once a frame is written, host is triggered to initiate VISS processing. HTS will go through 'init' sequence to initialize VISS and its SL2 pointers. As UDMA is head of pipeline for memory to memory mode of image processing, its scheduler inside HTS needs to be configured for generating "frame height" number of triggers to fetch image lines one by one. This is essential to bring determinism in pipeline as there is no producer for UDMA load thread.

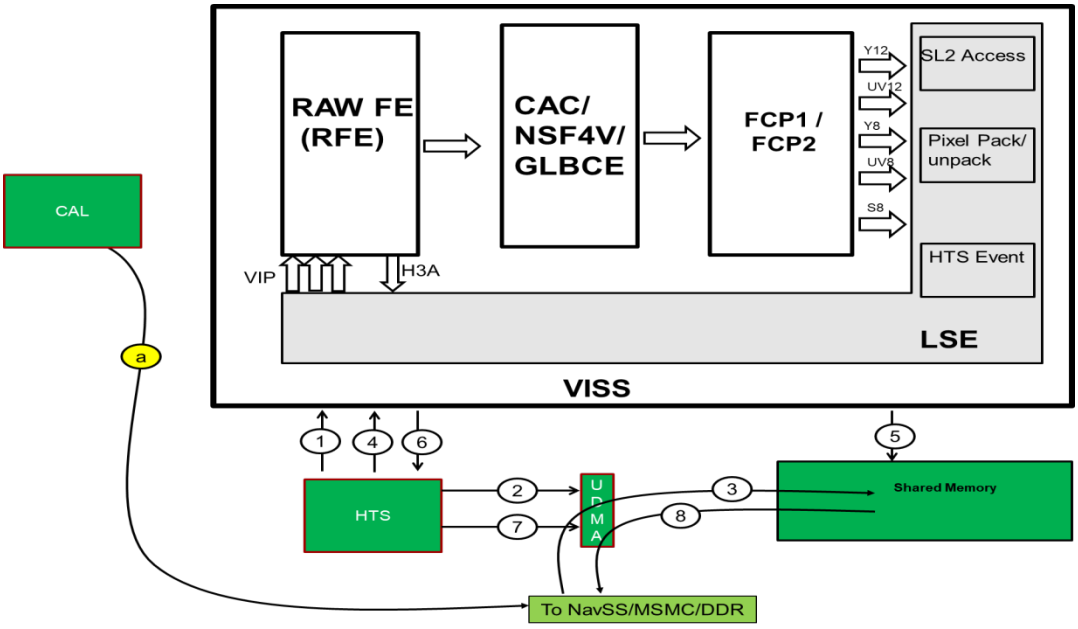


Figure 6-55. VISS Memory to Memory Operation

Copyright © 2024 Texas Instruments Incorporated

The following memory to memory operation steps are illustrated in Figure 6-55:

1. CSi-2 stream captured via CSI RX into DDR. This traffic must be routed through Real Time fabric of NAVSS/MSMC. CSI RX capture happens independent of VISS processing.

1. *Init* from HTS to initialize VISS.
2. HTS triggers UDMA to fetch data for processing. There could be max 3 independent input data.
3. UDMA loads the data and stores into SL2.
4. Upon completion of all input buffer loading (at line level) UDMA channel 'done' event triggers VISS thread start (still needs to ensure there are buffer available for writing VISS output data).
5. VISS produces line #N corresponding to loading of line #N + vertical_latency.
6. 'Tdone' triggered to HTS.
7. HTS triggers UDMA for output buffer transfer into DDR.
8. UDMA loads data from SL2 and write into DDR.

The following asumptions are made for the memory to memory operation described above:

- CSI RX write is mapped onto hard real time fabric of NavSS/MSMC.
- Configurable vertical blanking times 'Tstart' generated by HTS to let RFE + NSF4V + GLBCE + FCP flush its internal pipe, eventhough last line of current frame is already fed into RFE.
- Buffer dependencies:
  - RFE input buffer (exp0, exp1, exp2)
  - H3A output Buffer (AE or AF)
  - FCP output Buffer (Y12, UV12, Y8, UV8, S8)

6.7.3.1.4 Concurret Machine Vision and Human Vision Output

Machine vision (MV) and human vision (HV) output requires different tuning of alogorthimic blocks. For HV output, emphasis is on visual appearance while machine vision output requires output close to the original scene. A general machine vision application would not need NSF4V and GLBCE functions. Thus, to support concurrent output generation for MV and HV, a second instance of FCP is added. Raw image processing performed by RFE and CAC are common functions needed by both MV and HV.

When all outputs are generated to one FCP path, it should be done using a legacy FCP1 instance. Though it is not a hard requirement, FCP2 is targeted to be used in machine vision path and legacy FCP1 path is targeted for human vision path. FCP2 can be clock gated (by default) when not required to be used using the *FCP2_CNTL.PIX_CLKE_N* register control.

To enable flexible use cases, FCP2 can receive inputs from any IPs prior in the pipe line. This includes RFE, CAC, NSF4V, and GLBCE. Though each instance of FCP generates 5 output channel data, it is mux'ed with flexible combinations (mux control is in the LSEOUT_*MUX_CNTL* register) and 5 output channel data is generated.

6.7.3.1.5 VISS Clocking

The clock frequency requirement for the on-the-fly VISS pipe is 83 MHz to support 2M@30fps with 25% blanking and 10% DMA overhead. Clock frequency to support 4 pipes of 2MP@30fps image processing in memory to memory mode is 300 MHz with 25% system/DMA overhead. In memory to memory image processing, the high performing image pipe will create instantaneous peak bandwidth.

All modules inside VISS has 2 clock inputs:

- Functional clock : Used for all non-data path operation, like configuration logic and non-pixel data interfaces.
- Gated pixel clock : Pixel clock, that is synchronous to the functional clock, but is enabled when input data is available from LSE. All data path operations are done on the pixel clock. All the operations move in streaming fashion till the final output generation.

6.7.3.1.6 VISS Data Formats Support

Table 6-129 lists the data formats supported by VISS. VISS does not support YUV Input. In case of external YUV Sensor, CSI RX directly writes into DDR and tha data is subsequently processed by the rest of VPAC.

See Table 6-126 for more details.

Copyright © 2024 Texas Instruments Incorporated

Table 6-129. VISS Data Formats

Direction	Pixel Width	Pixel Container Width (M2M)	Packing	Video port (OTF)	Comments
VISS Input	RAW12/14/16	16b	No	16b / pixel	1. Companded data/External ISP 2. MSB bits are padded with zeros within fixed width pipeline
	RAW8	8b	Yes	16b / pixel	
	RAW12	12b	Yes	16b / pixel	
VISS Output	YUV420 (12b) NV12	12b	Yes	N/A	
	YUV420 (12b) NV12	16b	No	N/A	
	YUV420 (8b) NV12	8b	Yes	N/A	
	YUV422 (8b) UYVY/YUY2/NV16	8b	Yes	N/A	
	YUV422 (12b)* UYVY/YUY2/NV16	12b	Yes	N/A	
	YUV422 (12b)* UYVY/YUY2/NV16	16b	No	N/A	
	RGB (planar)	8b	Yes	N/A	With RGB output, limited YUV data outputs are available (only 12-bit YUV data is available)
	S8	8bit	Yes	N/A	

6.7.3.1.7 VISS VPORT Interface

The video port (VPORT) interface is used to stream the pixel data from up stream to down stream module. In VISS, the VPORT interface is used to receive data from CSI RX (in streaming mode) and in between VISS sub-modules (RFE -> CAC -> NSF4V -> GLBCE -> FCP).

The VPORT supports stall signal (VP_STALL) and also either one pixel or 2 pixel data per cycle.

The pixel data (VP_DATA) received on the VPORT is organized in 32 bits as follows:

- VP_DATA[15:0] = pixel (n)
- VP_DATA[31:16] = pixel (n+1). This is ignored, when VPORT_TWO_PIXEL = 0
- MSBs are padded with 0's when less than 16 bits are used.

6.7.3.1.8 VISS Submodule Integration Specifics

6.7.3.1.8.1 LSE Integration

Figure 6-56 captures the channel integration between LSE and rest of VISS submodules. Though digram shows three input channels between LSE to RFE, 3 channels data is transferred using single VBUSP interface.

Copyright © 2024 Texas Instruments Incorporated

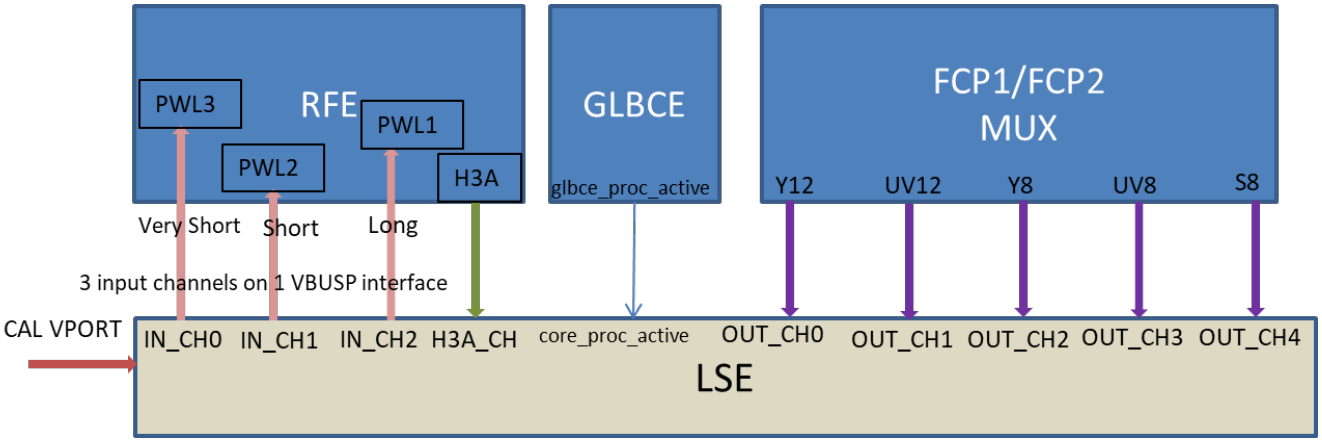


Figure 6-56. VISS LSE Channel Integration

The LSE configuration for VISS can be read in VISS_LSE_STATUS_PARAM register fields.

For more information on LSE module operation, see Chapter *Load and Store Engine (LSE)*.

6.7.3.1.8.2 Chromatic Aberration Correction

Figure 6-57 captures the CAC, NSF4V, and GLBCE integration in the VISS pipe line. All CAC, NSF4V, and GLBCE can be enabled or disabled based on use case. By default, all are disabled. General Visual use cases require NSF4V and GLBCE (along with FCP.EE) to be enabled. Low cost lenses with purple fringe artifact require CAC to be enabled.

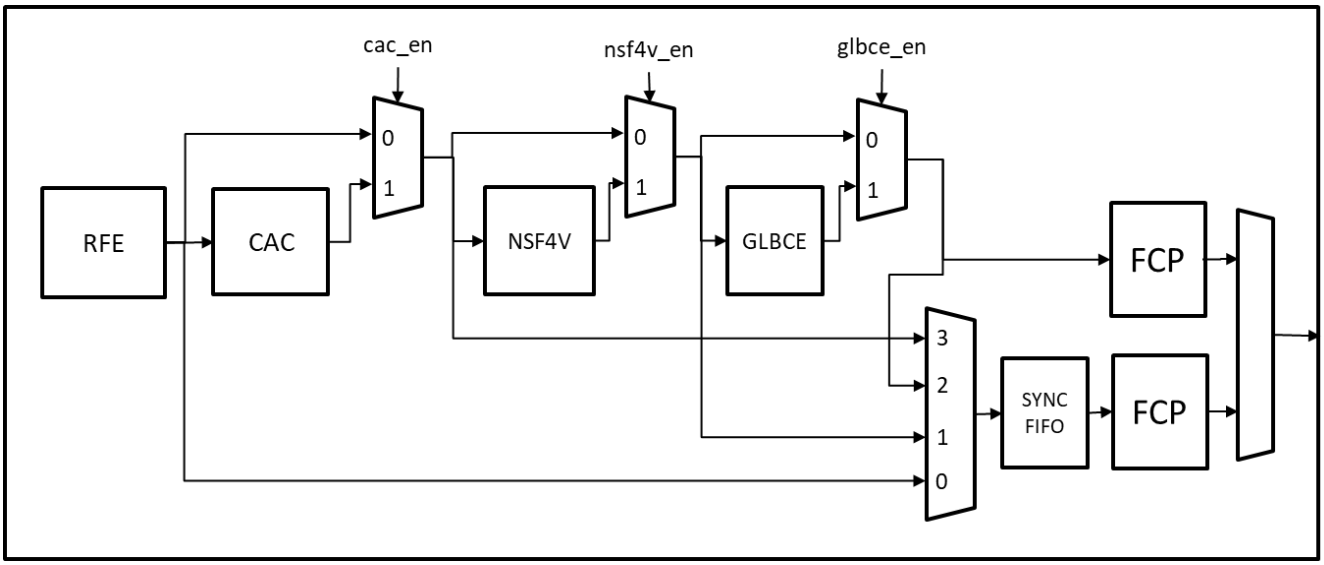


Figure 6-57. CAC, NSF4V, and GLBCE Integration

6.7.3.1.8.3 Spatial Noise Filter (NSF4V)

Figure 6-57 captures the NSF4V and GLBCE integration in the VISS pipe line. Both NSF4V and GLBCE can be enabled or disabled based on use case. By default, they are disabled. General Visual use cases require NDF4V and GLBCE (along with FCP.EE) to be enabled.

Figure 6-58 captures the RawHistogram and Dynamic White Balance block integration in NSF4V. Both DWB and RawHistogram can be enabled/disabled independent of NSF4V enable. When Histogram is enabled, **minimum frame width must be 128** (required to initialize the histogram memory at the frame start).

Copyright © 2024 Texas Instruments Incorporated

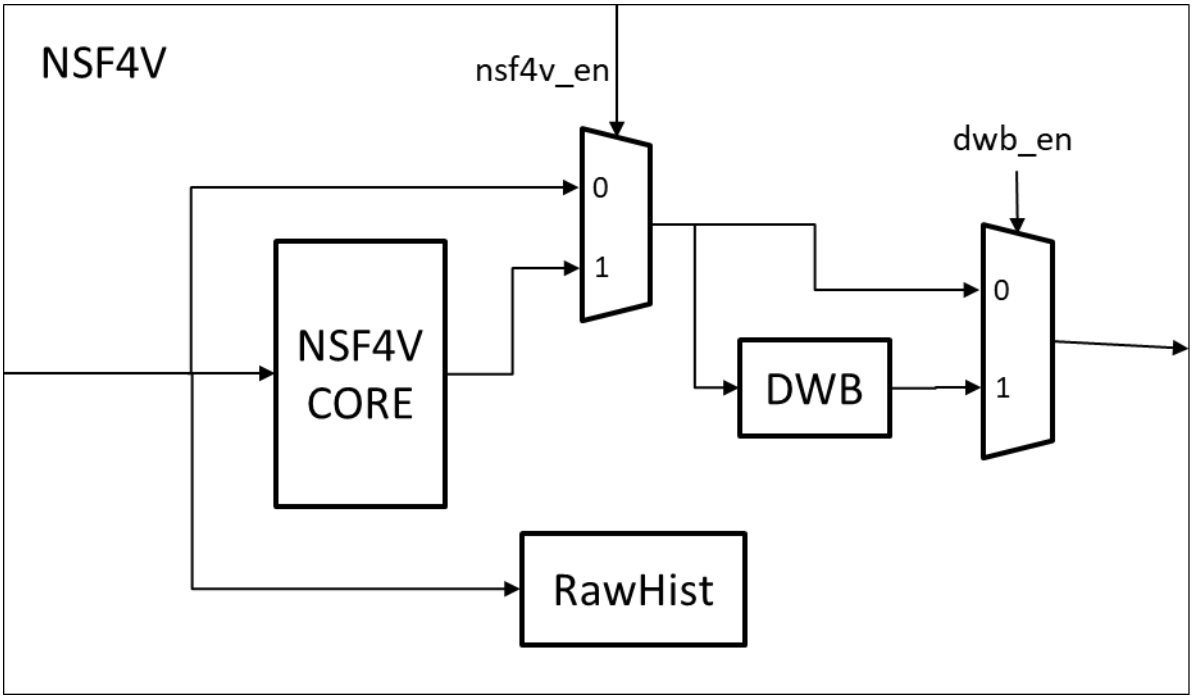


Figure 6-58. RawHistogram and DWB integration in NSF4V

6.7.3.1.8.4 GLBCE Integration

GLBCE is Global/Local Brightness and Contrast Enhancement module.

The GLBCE module supports the following main features:

- One pixel/cycle operation
- Up to 4096 pixels/line (line size)
- Bayer image
- When GLBCE is off, the RFE/CAC/NSF4V output is bypassed to FCP

For more information on GLBCE operation, see Section 6.7.3.5, *VISS Global/Local Brightness and Contrast Enhancement (GLBCE) Module*.

6.7.3.1.8.4.1 GLBCE Startup

GLBCE requires approximately 5400 clock pulses after startup before it could receive data on its video port. Therefore, before RFE -> GLBCE -> FCP path could be used after a GLBCE reset, the SW shall use the following sequence to provide those required clock pulses:

- Prior to enabling rest of VISS pipe, configure VISS clock control to allow free running clock to GLBCE.
- Wait for the 'glbce_filtering_done' event. It indicates that GLBCE has received enough PCLK pulses for internal initialization and that it is now ready to receive pixels.
- Configure GLBCE clock to normal mode. Enable the rest of VISS pipeline.

GLBCE has a minimum input restriction of 480x240 pixels. The input frame must be at least this size.

GLBCE LUTs must not be changed during the active frames. They can only be updated in vertical blanking. GLBCE_LUT_FI may be changed frame by frame depending on the GLBCE tuning method. Other LUTs (GLBCE_REV_PERCEPT_LUT_xx, GLBCE_FWD_PERCEPT_LUT_xx, and GLBCE_WDR_GAMMA_LUT_xx) are not expected to be updated dynamically except in very limited case.

6.7.3.1.8.4.2 GLBCE Bypass

When the GLBCE is disabled (by VISS_CNTL[0] GLBCE_EN = 0), any access targeted to GLBCE registers or GLBCE statistics memory will respond with error status, and 'glbce_cfg_err' interrupt will be generated.

Copyright © 2024 Texas Instruments Incorporated

6.7.3.1.8.5 Flexible Color Processing (FCP)

Figure 6-59 captures top level overview of FCP. LUT in front on CFA can map 16-bit data to 12-bit when CFA is operating in 12-bit mode. It can also be used to 16b to 16b remap when CFA is operating in 16-bit mode. EE data path is clock gated when EE module is not used. EE has bypass condition (YEE = 0) where data flows thorough it unchanged with the same latency as the enable condition.

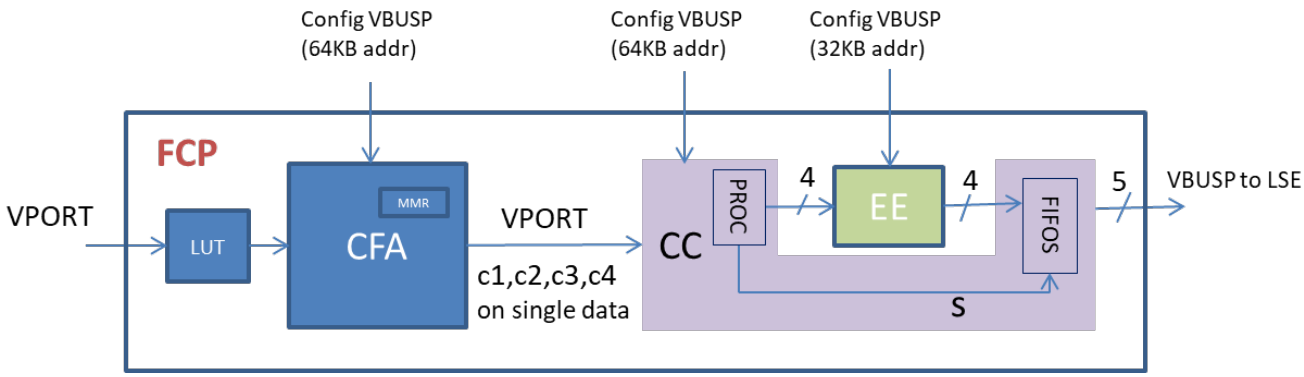


Figure 6-59. FCP Top Level Overview

In the CC module, Y12 to Y8 and U12 to UV8 LUTs should be enabled or disabled together (i.e *CFG_2.C8LutEn* = *CFG_2.C8LutEn*).

Program *EE_CFG_1.Yuv8_cl_align* = 1 while generating interleaved 422 8-bit and EE*_CFG_1.Yuv12_cl_align* = 1 while generating interleaved 422 12-bit output. This function is available even when EE path is not used.

6.7.3.1.9 VISS Stall Handling

VISS can handle momentarily stall condition created due to lack of enough band width.

6.7.3.1.10 VISS Blanking Requirements

Table 6-130 lists some blanking and latency details for VIS submodules.

Table 6-130. VISS Blanking Requirements

IP	Blanking		Latency	
	Vertical (in number of lines)	Horizontal (in pixel clock cycles)	Vertical (in number of lines)	Horizontal (in pixel clock cycles)
RawFE	2 - DPC Enable 0 - DPC Disable	8	2 - DPC Enable 0 - DPC Disable	50
CAC	6	20	6	24
NSF4V	15	28	15	44
NSF4V.DWB	0	7	0	7
GLBCE	- (1)	5	1	61
CFA	1	5	3	
FCC	0	20	1 (2)	25
EE	2	2	2	29
NSF4V	15	28	15	44

(1) GLBCE requires (1033*7 + 2066*FLOOR(VARIANCEINTENSITY/2,1) + VARIANCESPACE*2066+30) cycles for vertical blanking.
(2) In 420 mode, Chroma is vertically downsampled. For the first line, Chroma channel does not generate any dummy VBUSP requests. Hence, 1 line vertical delay is required.

The VARIANCEINTENSITY and VARIANCESPACE parameters (in Table 6-130 notes) affect the sensitivity of the transforms across different parts of the frame. Both parameters can be configured through the GLBCE_VARIANCE register.

In Table 6-130, the vertical blanking/latencies are expressed in number of lines (except for GLBCE), and horizontal blanking/latency is expressed in number of pixel clock cycles.

Copyright © 2024 Texas Instruments Incorporated

Horizontal blanking, buffer depth and HTS programming parameters (Threshold and PostLoad) depend on VISS frame mode or line mode of operation (selected via LSE_CFG_LSE[5] IN_CH_SYNC_MODE register field). Frame mode is the recommended mode of operation, and line mode is primarly used for debug purposes.

- Frame mode:
  - LSE input line fetch starts once the previous input line plus horizontal blanking cycle are provided to RFE. Horizontal blanking should meet two requirements:
    - Greater than minimum horizontal blanking of each enabled modules
    - Greater than maximum cycle gap between the last line pixel write on all LSE enabled output channel line ends
  - Horizontal Blanking of 35 is expected to be sufficient for most use cases. Horizontal blanking of 55 should be used, if EE is enabled and any of the CFA outputs is directly written out.
- Line mode:
  - LSE input line fetch start on HTS Tstart
  - Horizontal blanking is addition of all enabled modules horizontal delay (that is, latency)

LSE will automatically generate required vertical blanking until all output channels produce HTS_EOP, and GLBCE processing is complete (when GLBCE is enabled). In the vertical blanking window, LSE generates dummy lines (data = 0x0) following frame width paramter. LSE prematurely ends line w/o HE marker once vertical blanking requirement is met (that is, processing is done and HTS_EOP is generated).

6.7.3.1.11 FCP2 Sync FIFO

The second instance of FCP (referred as FCP2) can receive data from different IPs prior to being in the pipeline. It creates difference in both veritical and horizontal latency in the pipe line compared to the FCP1 instance. Vertical latency can be handled using the *DST[a]_BUF_ATTR0. Iout_skip_init* parameter in the LSE. However, horizontal latency difference in the pipe line needs higher horizontal lanking in frame mode of operation, which can result in reduced efficiency.

To align the VPORT timing at both instances of FCP, SyncFIFO block is added, which delays the FCP2 input VPORT interface by a programmable number of pixel cycles. The delay parameter can be programmed in *FCP2_CNTL.IN_PIPEDLY.* The delay value should be equal to the addition of the horizontal latency of all the IPs included in the FCP1 path but not in FCP2 input. Examples:

- If both FCP1 and FCP2 are getting data from GLBCE then program FCP2_CNTL.IN_PIPEDLY = 0
- If FCP2 is getting data from CAC and FCP1 is getting data from GLBCE with NSF4V and DWB enabled, program FCP2_CNTL.IN_PIPEDLY = 114 (44 [NSF4V] + 8 [DWB] + 62 [GLBCE])

6.7.3.1.12 VISS Interrupts

The VISS submodules generate the interrupt events described in Section *VPAC Subsystem Interrupts*.

6.7.3.1.12.1 Interrupts Merging

The configuration error interrupts are merged at VISS level. The following list describe the configuration error eventss and the corresponding merged interrupts.

- RAWFE_CFG_ERR_INTR: Generated when configuration interface access the RFE end points, while frame processing is active. Merged interrupts from RFE:
  - lut1_cfg_err_intr
  - lut2_cfg_err_intr
  - lut3_cfg_err_intr
  - wdr_lut_cfg_err_intr
  - dpc_line_cfg_err_intr
  - dpc_lut_cfg_err_intr
  - h3a_lut_cfg_err_intr
  - h3a_line_cfg_err_intr
- GLBCE_CFG_ERR_INTR: Generated when the configuration interface access the GLBCE register region or statastics memory, while frame processing is active. Merged interrupts from GLBCE:
  - glbce_statmem_cfg_err_intr
  - glbce_mmr_cfg_err_intr

Copyright © 2024 Texas Instruments Incorporated

- FCFA_CFG_ERR_INTR: Generated when the configuration interface access the CFA register region, or LUT or line memory, while frame processing is active. Merged interrupts from CFA for both instances (FCP1 and FCP2):
  - lut_cfg_err_intr
  - cfa_mmr_err_intr
  - cfa_pix_err_intr
- FCC_OUTIF_OVF_ERR_INTR: Generated when any of the FCC output interfaces overflows. Merged interrupts from CC for both instances (FCP1 and FCP2):
  - overflow_if_y12_intr
  - overflow_if_uv12_intr
  - overflow_if_y8r8_intr
  - overflow_if_c8g8_intr
  - overflow_if_s8b8_intr
- FCC_CFG_ERR_INTR: Generated when the configuration interface access FCC register region or its LUTs, while frame processing is active. Merged interrupts from FCC for both instances (FCP1 and FCP2):
  - contrast0_cfg_err_intr
  - contrast1_cfg_err_intr
  - contrast2_cfg_err_intr
  - lut_12to80_cfg_err_intr
  - lut_12to81_cfg_err_intr
  - lut_12to82_cfg_err_intr
- EE_CFG_ERR_INTR: Configuration happened to EE regions causing corruption during frame processing. Merged EE sources are (for both instances FCP):
  - eelut_cfg_err_intr
  - ee_pix_err_intr
- EE_SYNCOVF_ERR_INTR (for both instances FCP):
  - ee_hz_align12_intr
  - ee_hz_align8_intr
- fcc_hist_read_err_intr
  - FCP1.fcc_hist_read_err_intr
  - FCP2.fcc_hist_read_err_intr

6.7.3.1.12.2 Handling of Configuration Error Interrupts

Table 6-131 give hints on handling configuration error interrupts.

Table 6-131. VISS Configuration Error Interrupts Handling

Memory	Error Generation Window	HWA-level Register Status Clearing Mechanism
RAWFE		
PWL1_LUT, PWL2_LUT, PWL3_LUT	Config read access during active line OR write access during active frame.	Write ‘1’ to corresponding bit in RAWFE_INT_STAT register.
WDR_LUT		
H3A_LUT		
H3A_ACCM	Config read/write access during active frame at H3A boundary.	
H3A_LINE		
DPC_LUT	Config read/write access during active frame.	
DPC_LINE	Config read access during active line OR write access during active frame.	
LSC Table	Config read/write access during active frame. Active frame in lsc case is VS at pwl to VE at lsc input delayed by 1 cycle.	
NSF4V		
NSF4V_LINE	Config read/write access when datapath is accessing the corresponding memory on the same cycle.	No register status in NSF4V.
GLBCE		

Copyright © 2024 Texas Instruments Incorporated

Table 6-131. VISS Configuration Error Interrupts Handling (continued)

Memory	Error Generation Window	HWA-level Register Status Clearing Mechanism
GLBCE non-shadowed registers	Config write access during active frame. Active frame window is from VS_IN to Filter done.	Write ‘1’ to GLBCE_INT_STAT[0] MMR_CFG_ERR register bit.
GLBCE_STAT	Config read/write access during active frame. Active frame window is from VS_IN to Filter done.	Write ‘1’ into GLBCE_INT_STAT[1] STATMEM_CFG_ERR register bit.
GLBCE_LINE	Not mapped to config.	N/A
CFA		
CFA FIR Filter registers	Config write access during active frame.	Write ‘1’ to CFA_INT_STATUS[2] CFA_MMR_ERR register bit.
CFA_LUT	Config read/write access during active frame.	Write ‘1’ into CFA_INT_STATUS[0] LUT_CFG_ERR register bit.
CFA_LINE	Config read/write access during active frame.	Write ‘1’ to CFA_INT_STATUS[1] CFA_PIX_ERR register bit.
FCC		
LUT_CONTRAST	Config access (read/write) occurs during active line when the LUT is enabled.	Write ‘1’ to corresponding bit in FCC_FLEXCC_INT_STATUS register.
HISTOGRAM	Config access has occurred to the first location but not to the last location till the start of next frame implying that full histogram was not read.	
LUT_COLOR	Config access (read/write) occurs during active line when the LUT is enabled.	
CC_LINE	No error generation logic.	N/A
EE		
EE_LINE	Config accesses during active frame.	Write ‘1’ to EE_INT_STATUS[1] EE_PIX_ERR register bit.
LUT	Config accesses during active frame.	Write ‘1’ to EE_INT_STATUS[0] EELUT_CFG_ERR register bit.

6.7.3.1.13 VISS Error Correcting Code (ECC) Support

ECC is a mechanism for providing increased system reliability (via reduction of memory soft errors) by allowing single bit errors to be detected and corrected and double bit errors to be detected.

One or more memories within VISS are ECC protected using an ECC Memory Wrapper which implements Single Error Correction and Double Error Detection (SECDED).

The ECC wrapper provides Single Error Detection and Correction (SED/SEC). This logic detects and corrects a single bit error (1 bit error per ECC word or per ECC data segment). For memories that contain critical and/or persistent data, automatic (immediate or delayed) write-back of the corrected data to the corresponding memory address is supported. In addition, the ECC wrapper also supports multiple options for partial word writes, such as read-modify-write or multiple ECC code segments per word.

The ECC wrapper also provides Double Error Detection (DED). This logic only detects (does not correct) double errors (2 bit errors per ECC word or per ECC data segment).

An ECC Aggregator at the VISS level consolidates the ECC configuration and status bits for all the ECC supported memories in the subsystem. It provides a single EOI-handshake based interrupt to the host (for both single and double error detections) and a standard 32-bit VBUSP interface for configuring and querying the various ECC wrappers via their ECC register set.

The following VISS memories are ECC protected:

- GLBCE statastics memory.
- RAWFE non pixel line buffer memories. See Section *VISS RAW Frond-End (RAWFE)*, for more information.
- CAC 2D LUT
- FCP non pixel line buffer memories

For more information on the ECC Aggregator operation, see Section *ECC Aggregator* in Chapter *Safety Modules*.

Copyright © 2024 Texas Instruments Incorporated

6.7.3.1.14 VISS Programmer's Guide

6.7.3.1.14.1 VISS Initialization Sequence

VISS initialization is part of any other HWA initialization on VPAC. Prior to starting HWA initialization through HTS, VISS must be configured in below order.

1. Configure RFE registers, GLBCE registers, NSF4V registers and FCP registers:
   - The VISS_CNTL[0] GLBCE_EN register bit needs to be set, in order to configure the GLBCE registers.
2. Select input / output stream properties inside LSE.
3. After reset and prior to enabling LSE, apply the following configuration:
   a. Make sure VISS_CNTL[0] GLBCE_EN bit is '1'.
   b. Enable IRQ for interrupt event 'glbce_filtering_done'.
   c. Set VISS_GLBCECONFIG[0] GLBCE_PCLKFREE register bit to '1'.
   d. Wait for interrupt 'glbce_filtering_done'.
   e. Clear VISS_GLBCECONFIG[0] GLBCE_PCLKFREE register bit.
4. Enable LSE input and output buffer lines.
5. Enable pipeline and schedulers associated with VISS HTS configuration (pipeline enable must happen after enabling all attached schedulers).
6. Enabling HTS will trigger init sequence which will initialize LSE, RFE and FCP. This will also trigger head of pipe for NAVSS UDMA fetch (in case of memory to memory operation). Otherwise,VISS waits for streaming data from CSI RX.
7. After initializing VISS/VPAC, configure and enable CSI RX, CSI RX PHY and camera sensor in order.
8. Once streaming data starts, it will continue till the end of frame. Blanking (set in VISS_LSE_CFG[31-22] VP_HBLNK_CNT register field) must be configured more than VISS blanking needs (see *VISS Blanking Requirements*).
9. Once a complete frame is fully written into DDR, the HTS module comes back to init state and restart from step #6. For some usesecase, LSE input/out enables after each frame might be considered as well (step #5).
10. After each 'frame_done', if the configuration needs to be changed, it must be done within vertical blanking period for not shadowed registers (refer to Sections *VISS RAW Frond-End (RAWFE)* and *VISS Flexible Color Processing (FCP) Module*, for more details).
11. At any stage, if there is a need to stop the streaming interface, the HTS must first be paused and then aborted, and then reset and re-configure the entire VISS as part or recovery from abort.

6.7.3.1.14.2 VISS Configuration Restrictions

Frame size restriction:

- If GLBCE is enabled in the VISS pipeline, the minimum frame required is 480x240 pixels.
- If GLBCE is disabled in the VISS pipeline, the minimum frame required is 64x16 pixels.
- Frame width and height has to be even.
- Histogram can only be enabled when the frame width is 256 pixels minimum.

LSE H3A channel enabling restriction:

- When the H3A output channel is enabled in LSE, at least one other output channel needs to be enabled.

6.7.3.1.14.3 VISS Real-time Operating Requirements

SW needs to configure UTC/UDMA channel registers for write out buffers.

SW needs to reconfigure RAWFE LSE LUT, if needed, and any other non-shadowed registers during vertical blanking period only. Some control bits would be shadowed, which can be configured during a complete frame.

Copyright © 2024 Texas Instruments Incorporated

6.7.3.2 VISS RAW Frond-End (RAWFE)

This section describes the Vision Imaging Sub-system (VISS) RAW Frond-End (RAWFE) module in the device.

6.7.3.2.1 RAWFE Overview

The RAWFE is a set of raw pixel processing functions to enhance the RAW image signal received from the sensor. The key application of the RAWFE is in the ADAS space where in it is used for both analytics as well as visual use cases.

6.7.3.2.1.1 RAWFE Supported Features

Each RAWFE module implements the following features:

- Support for any 2x2 RAW data pattern.
- Support for merge up to 3 separate exposures.
- PWL (piecewise linear) and LUT based de-companding at the front end of each exposure channel
- Support for WDR merge, including 2 exposure merge, 3 exposure merge and 2 exposure hybrid (12 bit and 16 bit) merge.
- Support for companding LUT from 24 bits down to 12 bits.
  - LUTs support higher precision in lower end.
- Support for LUT as well as adaptive Defect Pixel Correction.
- Support for table based lens shading correction.
- Support for 3A statistics
  - Statistics can be generated on independent exposures or merged data after tone map.

6.7.3.2.1.2 RAWFE Not Supported Features

The RAWFE does not support the following features:

- Motion compensated WDR Merge
- Simultaneous statistics using H3A on all 3 exposures
- More than 3 exposures for WDR

6.7.3.2.2 RAWFE Functional Description

6.7.3.2.2.1 RAWFE Functional Operation

The RAWFE processes captured image data from sensor and passes the data to the Flexible Color processing module (FCP) for demosaicing and color conversion.

Figure 6-60 shows a high level conceptual block diagram of the RAWFE module.

The RAWFE consists of the following main components

- Decompanding block: This is used to decompand the sensor compressed raw to native bit depth. The same block can be used for tone mapping as well. The decompanding block can take sensor compressed RAW data and can decompand to up to 24 bits.
- WDR Merge: Each instance of the WDR merge block can take in 2 independent exposures (up to 16 bits) and generate up to 20 bit data.
- LSC Block: The block performs lens shading correction by applying gains stored in a look up memory.
- DPC: The block performs defect pixel correction using either LUT based memory or adaptive defect correction.
- H3A: The H3A block generates 2 different set of statistics, one for AWB and AE and the other for auto focus.

Copyright © 2024 Texas Instruments Incorporated

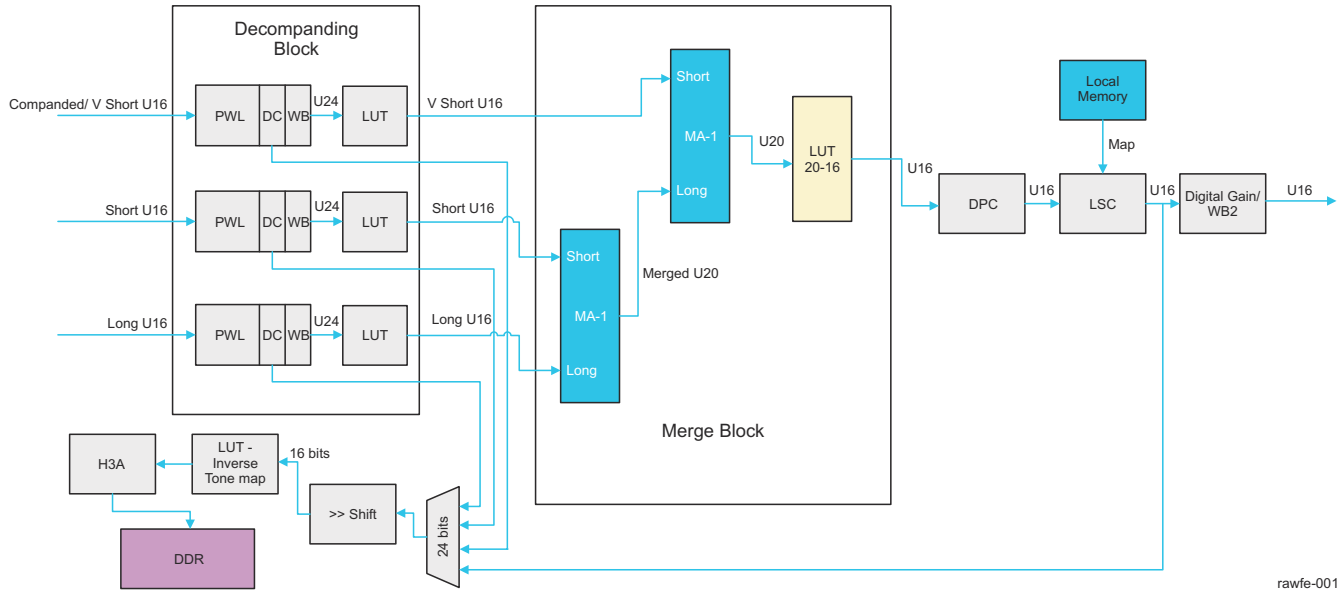


Figure 6-60. RAWFE Block Diagram

6.7.3.2.2.2 RAWFE Integration in VISS

Figure 6-61 show the VISS Block Diagram integrating the RAWFE module.

The VISS Consist of following hardware accelerators

1. RAWFE: The RAWFE block does RAW pixel (e.g. Bayer, Clear Channel) processing on captured image data from sensor and later passes to FCP block for demosaicing and color conversion.
2. FCP (Flexible Color Processing): The FCP receives data from RAWFE and does demosaicing and color conversion. The output of FCP is sent to external memory for vision processing by programmable processors (e.g. DSP or ARM) as well as Vision HW Blocks (e.g. VPAC and DMPAC).
3. GLBCE (Global Local Brightness and Contrast Enhancement): This is local adaptive tone mapping block.

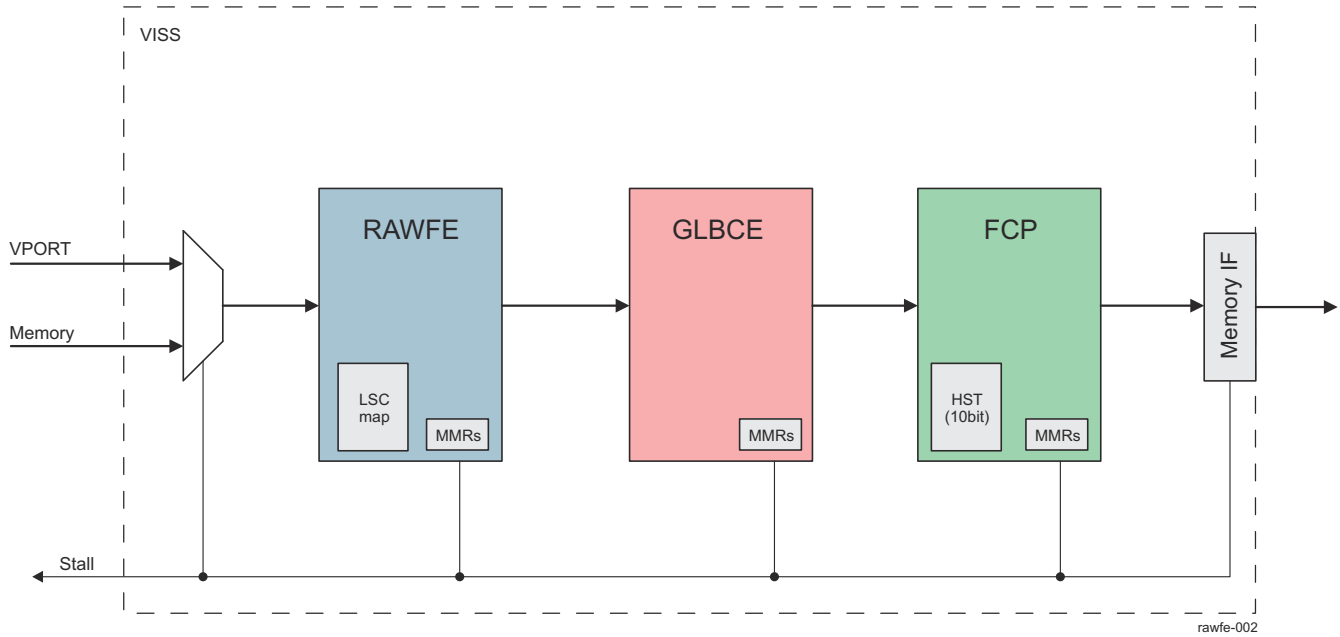


Figure 6-61. RAWFE Integration in the VISS

Copyright © 2024 Texas Instruments Incorporated

6.7.3.2.2.3 RAWFE Memory Map

Table 6-132 lists the RAWFE memory mapped RAMs.

Table 6-132. RAWFE Memory Map

Address Offset	Register Space	Size
0x00000	RAWFE registers	
0x00400	H3A registers	
0x00800	V-Short LUT (LUT3-RAM)(1)	639 x16 bits
0x01000	Short LUT (LUT2-RAM)(1)	639 x16 bits
0x01800	Long LUT (LUT1-RAM)(1)	639 x16 bits
0x02000	Merge LUT (WDR_LUT-RAM)(1)	639 x 16 bits
0x02800	H3A LUT (H3a_LUT-RAM)(1)	639 x 10 bits
0x03000	DPC LUT (DPC-RAM)(1)	256x 29 bits
0x08000	LSC LUT(RAM)(1)	19032 Bytes

(1) Partial writes to RAMs are NOT supported. Software must write the full word. This restriction is an artifact of simplifying ECC support.

Most registers in RAWFE MMR space are shadowed. On frame start, the non shadowed registers will update the shadowed registers and used for the remainder of the frame for data processing. The following registers are not shadowed:

- VISS_RAWFE_INT_STAT
- VISS_RAWFE_DBG_STAT1
- VISS_RAWFE_DBG_STAT2
- VISS_RAWFE_DBG_STAT3
- VISS_RAWFE_DBG_STAT4
- VISS_RAWFE_DBG_CTL

H3A has partial shadowing, refer to Section 6.7.3.2.4.6 for details.

LUT3 ram is shadowed. LUT3 and LUT2 will be multiplexed for shadowing capabilities. Software must use a control register to select which lut is to be used during frame. Software must fully configure LUT2 in this case as well.

Note

Accesses to non shadowed registers or rams during active window can cause corruption. During an active phase, non blanking region, any accesses to a ram, read or write, will corrupt the output data. H3a rams, lsc lut, dpc lut rams are exceptions. These rams must be outside the full frame, including horizontal blanking regions, otherwise they will cause corruption. Any write to mmr's that are not shadowed will result in data corruption in non blanking area, the interrupt status and debug registers are exception. Mmr reads to any type of register are ok and will not result in frame output corruption. Interrupt events will be asserted if config accesses corrupt frame data.

6.7.3.2.2.4 RAWFE ECC for RAMs

Table 6-133 lists the RAWFE ECC RAMs

Table 6-133. Table 3: RAM ECC capabilities

Module	# Rams	ECC
PWL LUT1	2	ECC, with single bit correction, including writeback.
PWL LUT2	2	ECC, with single bit correction, including writeback.
PWL LUT3	2	ECC, with single bit correction, including writeback.
WDR LUT	2	ECC, with single bit correction, including writeback.
H3A LUT	2	ECC, with single bit correction, including writeback.
DPC (line buffer)	4	No ecc
LSC LUT	1	ECC, with single bit correction, including writeback.
LSC GAIN	1	ECC, with single bit correction, including writeback.

Copyright © 2024 Texas Instruments Incorporated

Table 6-133. Table 3: RAM ECC capabilities (continued)

Module	# Rams	ECC
H3A accum	1	No ecc
H3A line	1	No ecc

6.7.3.2.3 RAWFE Interrupts

The following section lists the RAWFE interrupts

6.7.3.2.3.1 RAWFE CPU Interrupts

Table 6-134 lists the interrupts generated by the RAWFE sub-modules.

Table 6-134. RAWFE Interrupts

Interrupt	Description
LUT1_CFG_ERR	Config access to LUT1 ram has corrupted functional operation, config read or write memory access occurred during functional operation. Only asserted if any (read/write) cfg access occurs during active line OR write occurs in horizontal blanking within a frame.
LUT2_CFG_ERR	Config access to LUT2 ram has corrupted functional operation, config read or write memory access occurred during functional operation. Only asserted if any (read/write) cfg access occurs during active line OR write occurs in horizontal blanking within a frame.
LUT3_CFG_ERR	Config access to LUT3 ram has corrupted functional operation, config read or write memory access occurred during functional operation. Only asserted if cfg access occurs during active line. Only asserted if any (read/write) cfg access occurs during active line OR write occurs in horizontal blanking within a frame.
WDR_LUT_CFG_ERR	Config access to WDR LUT ram has corrupted functional operation, config read or write memory access occurred during functional operation. Only asserted if any (read/write) cfg access occurs during active line OR write occurs in horizontal blanking within a frame.
H3A_LUT_CFG_ERR	Config access to H3A LUT ram has corrupted functional operation, config read or write memory access occurred during functional operation. Only asserted if any (read/write) cfg access occurs during active line OR write occurs in horizontal blanking within a frame.
H3A_ACCM_CFG_ERR	Config access to H3A accum ram has corrupted functional operation, config read or write memory access occurred during functional operation. Asserted if any cfg access occurs during active frame.
H3A_LINE_CFG_ERR	Config access to H3A line ram has corrupted functional operation, config read or write memory access occurred during functional operation. Asserted if any cfg access occurs during active frame.
DPC_LUT_CFG_ERR	Config access to DPC LUT ram has corrupted functional operation, config read or write memory access occurred during functional operation. Asserted if any cfg access occurs during active frame.
DPC_LINE_CFG_ERR	Config access to DPC line ram has corrupted functional operation, config read or write memory access occurred during functional operation. Only asserted if any cfg access occurs during active line OR write occurs in horizontal blanking within a frame.
LSC_CFG_ERR	Config access to LSC ram has corrupted functional operation, config read or write memory access occurred during functional operation. Asserted if any cfg access occurs during active frame. Active frame in lsc case is VS at pwl to ve at lsc input delayed by 1 cycle.
H3A_AEW	h3a aew interrupt.
H3A_AF	h3a af interrupt
H3A	h3a interrupt
H3A_BUF_OVRFLOW_PULSE_INTR	Generated when an overflow occurs in the H3a output buffer

6.7.3.2.3.2 RAWFE Debug Events

Table 6-135 lists the RAWFE debug events. Debug events are used for hardware debug as well as performance measurement. They are not intended to be used as interrupts where the cpu needs to process them.

Copyright © 2024 Texas Instruments Incorporated

Table 6-135. RAWFE Debug events

Event	Description
VS_EVENT	ventricle start in the beginning of the rawfe pipeline
VE_EVENT	ventricle end at the end of the rawfe pipeline
HS_EVENT	horizontal start in the beginning of the rawfe pipeline
HE_EVENT	horizontal end at the end of the rawfe pipeline
LSE_SLAVE_STALL	rawfe is stalling lse slave interface due to ext or mtc stall
LSE_MASTER_STALL	lse master interface is stalled
LSE_INTERFACE_IDLE	within a frame lse is not sending data
X_Y_EVENT_MATCH	x-y pixel position has reached the start of rawfe pipeline
DPC_OTF_CORR_EVENT	dpc otf corrected a pixel position
PIPE_ADV_EVENT	active pipeline advancement

6.7.3.2.3.3 RAWFE Interrupt Handling: High Priority

When a high priority interrupt occurs, the CPU starts shutting down the RAWFE and stopping its usage. The next steps of actions are under software control.

6.7.3.2.3.4 RAWFE Interrupt Handling: Low Priority

When a low priority interrupt occurs, the CPU notices these conditions and maintain statistics of such occurrences. The next steps of actions are under Software control.

6.7.3.2.4 RAWFE Sub-Modules Details

The following sections describe in detail the sub-modules present within the RAWFE module.

6.7.3.2.4.1 RAWFE Decompanding Block

The decompanding block is responsible for decompanding pwl (or LUT) compressed data back to its linear bit width. The decompanding block can generate up to 24 bits of data, however the data needs to be tone mapped down to 16 bits before additional processing can take place. There are 3 instances of the Decompanding block each implementing the same functionality.

The input interface of the decompanding block is 16 bits.

Figure 6-62 shows a high level block diagram of the decompanding block.

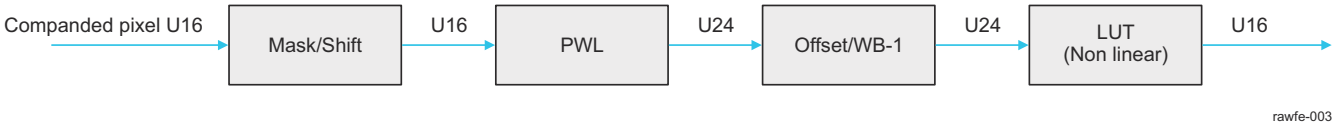


Figure 6-62. RAWFE Decompanding Block Diagram

6.7.3.2.4.1.1 RAWFE Mask & Shift

The mask and shift block is used to mask out any control bits present in the data. The ‘Mask’ is a 16 bit configuration register which will perform a bit wise ‘and’ operation on the incoming data. (For example for sensor data with data in 12 bits and 4 bits of control information, the mask value would be programmed to 0xFFF). The Mask operation is followed by a shift operation. The ‘Shift’ field is a 3 bit configuration register which can perform a right shift (>>) from 0 to 7 bits.

6.7.3.2.4.1.2 RAWFE Piece Wise Linear Operation

The PWL block is used to apply a piece-wise-linear gain on the incoming pixel data and is generally used to uncompress sensor compressed data. The knee points of the PWL curve should be programmed to correspond with the compression knee points used by the sensor configuration.

The PWL block supports an input size from 8 – 16 bits and can support an output up to 24 bits.

Figure 6-63 depicts a PWL curve applied on incoming data

Copyright © 2024 Texas Instruments Incorporated

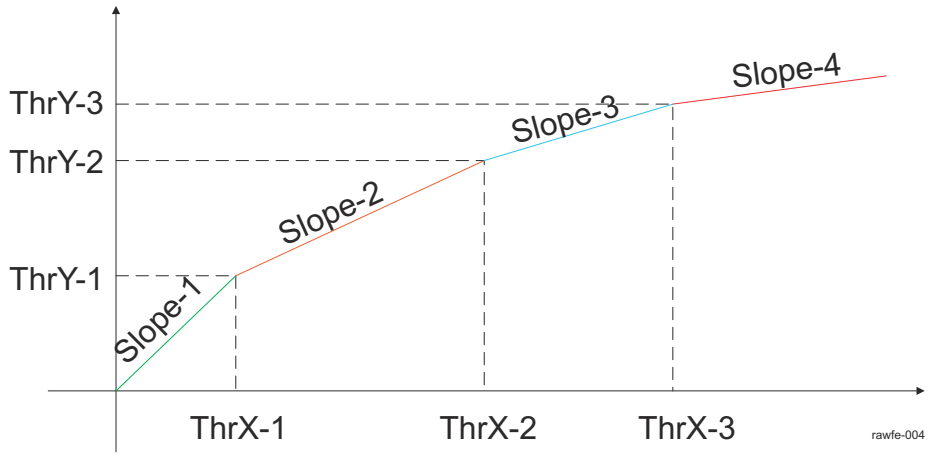


Figure 6-63. RAWFE PWL Curve implementation

The PWL block implements the following equation assuming the input lies in the nth segment.

$$\text{Out_Pixel} = ((X - \text{ThrX_n}) \times \text{Slope}) + \text{ThrY_n} \quad (2)$$

The figure below shows the PWL block architecture

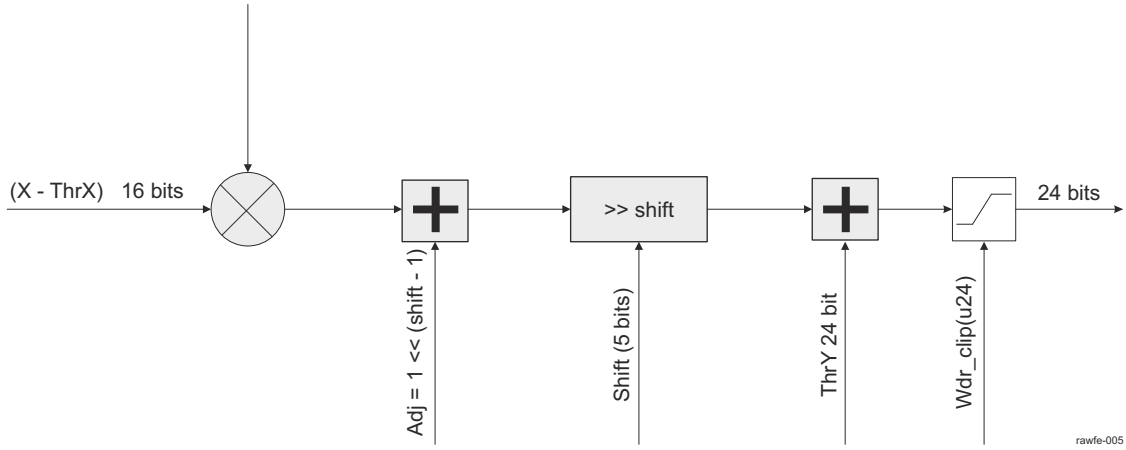


Figure 6-64. RAWFE PWL Block Diagram

The Wdr_clip parameter can be set to clip the output at the desired bit width (typically 20 bits).

6.7.3.2.4.1.3 RAWFE Offset/WB-1 Block

This block is used for applying White balance correction in linear domain as well as for subtracting the DC offset using the signed 24 bit offsets. The functionality is similar to the WB2 block. Note, there is a tap out point to H3A after the offset application, but prior to the gain block. This allows the H3A to receive DC subtracted (but not WB corrected) data.

6.7.3.2.4.1.4 RAWFE LUT Based compression

The LUT based compression is used for addressing multiple use cases.

- The primary function of the LUT is to take the PWL decompounded data (up to 24 bits and after DC subtraction) and compress it using an x^n curve to a bitwidth of 16 bits. This is required since the downstream path is only capable of supporting up to 16 bits of data.
- It can be used to decompand data when the compression curve cannot be addressed using 3 knee points or if the compression is true curve instead of PWL. In this scenario the DC subtraction IP is disabled and the LUT is used to implement both the decompression as well as the DC subtraction functionality.

Copyright © 2024 Texas Instruments Incorporated

The LUT compression is modified from the implementation in previous generation ISPs. The previous LUTs had 512 entries leading to a step size of 2^20/512 = 2048. As such there were only 2 LUT points in the critical low range (0-4095) of the image. The LUT implementation is optimized for a 20-bit decompounded data and allows for a step size of 32 in the critical range (lower 12 bits) and a step size of 2048 in the successive ranges.

The Non Linear LUT implementation supports 3 zones for the input range

- For bitWidth < 12 bits, the LUT offers a linear range with a step size of 32. (Only the first 128 locations (plus 129th location for supporting interpolation) are utilized in this mode
- For bitWidths between 13 and 20: This is the most common operating zone for the LUT. In this mode, the range between 0 and 4k is split in 128 locations with a step size of 32. Beyond 4k range, the step is variable and is set to 2k for the highest bit width of 20 and subsequently reduces by half as the bitwidth is reduced (E.g. 1k for 19 bits etc)
- For bitwidths between 21 and 24: In this operation zone, the first 128 locations are used with a varying step size based on bit width such that the step size is 64 for 21 bits of data and doubles with increase in bitWidth. (E.g. 128 for 22 bits etc). The remaining locations implement a varying step size between 4k (for 21 bits) and 32k (for 24 bits).
  - Example-1: Bitwidth=21 bits:
    - Range 0-8k-> Step size = 64;
    - Range 8k- 2^21 -> Step Size = 4096
  - Example-2: Bitwidth = 22 bits
    - Range 0 -16k -> Step Size = 128
    - Range 16k – 2^22 -> Step_Size = 8192
  - Example 3: Bitwidth = 24 bits
    - Range 0 – 64K -> Step Size = 512
    - Range 64K - 2^24 -> Step Size = 32k

The LUT logic provides a generic table which can support any bit width and can be used in different regions in the RAWFE as well as other modules.

The LUT3 supports shadowing for very short frame only. If the MMR shadow enable is set then the LUT2 is used on the LUT3 data.

6.7.3.2.4.2 RAWFE WDR Merge Block

The merge block supports merging up to 3 exposures to generate a cumulative 20 bit frame.

The merge block is based on 2 instances of the Motion-Adaptive merge block, each of which is independently capable of supporting the merging of 2 exposures (10 – 16 bits each) to a combined bit width of up to 20 bits. In a more generic scenario the merge block should be designed and verified to support the following scenarios

- Merge 2 exposures of 10 bits each to 16 bits
- Merge 2 exposures of 12 bits each to 16 bits
- Merge 2 exposures of 12 bits each to 20 bits
- Merge 2 exposures (12 bit and 16 bit) to 20 bits.
- Merge 2 exposures of 16 bit each to 20 bits.

Using two instances of the motion adaptive merge block provides the functionality to merge up to 3 independent exposures, each of 12 bits, to a combined bitwidth of 20 bits. The block also provides capability to only merge 2 exposures using a combination of mux and clipping blocks.

The merge block comprises of two instances of the motion adaptive merge (MA1/MA2) as well as an LUT based companding block and mux structures to enable different data flow. To keep the design complexity under control some restrictions are applied to the data connectivity on the merge blocks. A key restriction is that for the two exposure merge only the MA-2 block should be used and the MA-2 should be used along with MA-1 for 3 exposure merge. Further the position of the long and short exposures on each block are fixed. Table 6-136 summarizes some of the possible combinations which can be realized using the logic. (Note, this is not an exhaustive list from a DV perspective).

The MA block when set to bypass/disabled mode, will pass the input stream marked as ‘short’ (refer to Figure 6-60) to the output.

Copyright © 2024 Texas Instruments Incorporated

Table 6-136. Figure 7:RAWFE Different merge modes

Merge Mode	MA-1	MA-2	LUT	
2 Exposures (12 bit) to 20 bit	Bypassed	Merge Long / Short	Use VS and Short Exposures	Set to 20-16
2 Exposures (12 bit) to 16 bit	Bypassed	Merge Long /Short	Use VS and Short Exposures	Bypassed
2 Exposures (12 bit) to 16 bit with tone mapping	Bypassed	Merge Long /Short	Use VS and Short Exposures	Set to 16-16 mode
2 Exposures (12 & 16) to 20 bit	Bypassed	Merge Long /Short	Use VS and Short Exposures	Set to 20-16
3 Exposures (12 bit each) to 20 bit	Merge Long and Short	Merge VS / MA-1 output	Use all 3 exposures	Set to 20-16
Single exposure companded	Bypass	Bypass	Use VS exposure only	Bypassed

6.7.3.2.4.2.1 RAWFE WDR Motion Adaptive Merge (MA1 / MA2)

This section provides the detailed description of the two instances of the motion adaptive merge.

Figure 6-65 shows the high level block diagram of the MA merge block

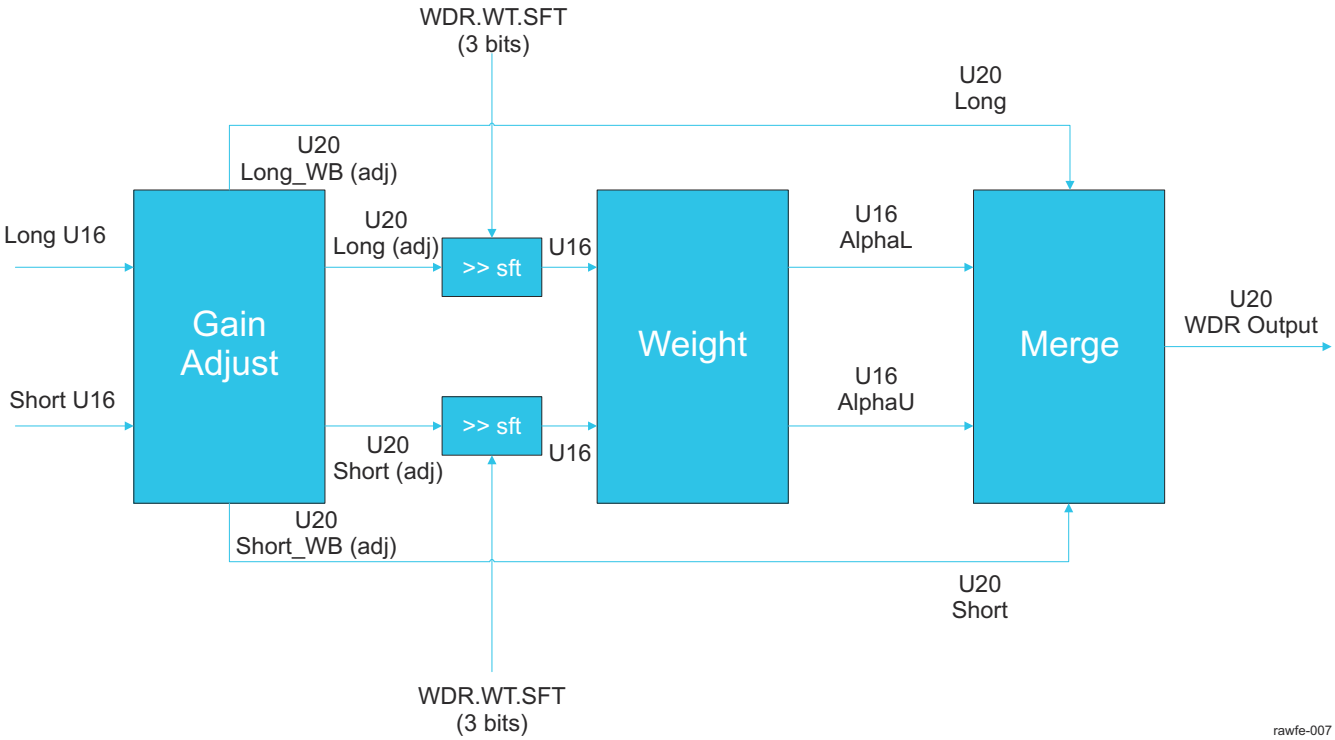


Figure 6-65. RAWFE WDR Motion Adaptive Merge

The MA block receives two inputs each of which can be up to 16 bits. As mentioned earlier, the ordering of the long and the short exposure cannot be changed and should be positioned as it's depicted in the figure above. The MA block consists of 3 main sub-blocks, which are described in the following sub-sections.

6.7.3.2.4.2.2 RAWFE Companding LUT

The companding LUT is used to perform a global tone mapping on the image and reduce the bit depth from 20 bits down to 16 bits. The LUT is implemented as a non uniformly spaced memory, with higher precision in the lower range of the image intensity. The LUT architecture is similar to that used for the decompanding LUT in the previous section.

Copyright © 2024 Texas Instruments Incorporated

6.7.3.2.4.3 RAWFE Defective Pixel Correction (DPC) Block

The defect pixel correction (DPC) block is responsible for correcting defective pixels present on the sensor as an artifact of the manufacturing process. The defect pixel detection can either be LUT based or on the fly (OTF – DPC). For LUT based DPC, the defect map is stored in a memory.

Figure 6-66 shows a high level block diagram of the DPC module. The DPC comprises of two different methods of performing the functionality, each of which is highlighted below. The LUT based defect correction mechanism requires the sensor to be calibrated and the defect positioned stored in memory, whereas the Adaptive DPC automatically detects defects and corrects them. If required the LUT DPC and adaptive DPC can be enabled together in sequence.

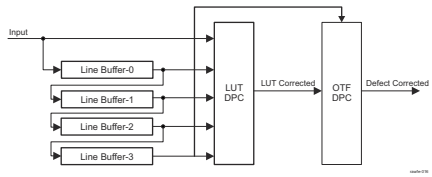


Figure 6-66. RAWFE DPC Block Diagram

6.7.3.2.4.3.1 RAWFE LUT Based DPC

LUT based defect pixel correction provides the highest level of accuracy since all defects can be identified perfectly without any false positives. The LUT based DPC mechanism however suffers from the limitation that the sensor has to be calibrated (each sample has to be individually calibrated) and that can add cost to the testing and qualification process. Some automotive grade sensor vendors provide the defect pixel locations already in an on chip memory and that can additionally be utilized to program the defect LUT without adding the cost. The LUT size for the DPC is constrained to 256 entries, since it is assume for a sensor of 2 -3 Mpix resolution, 256 entries are more than sufficient. Each entry in the LUT is 29 bits with 13 bits for x coordinate, 13 bits for Y coordinate and 3 bits for the method. Figure 6-67 illustrates the organization of the LUT memory

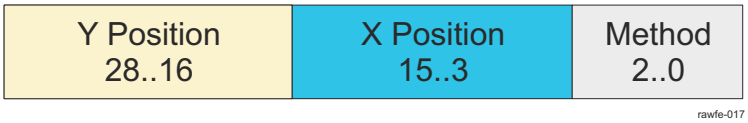


Figure 6-67. RAWFE DPC LUT Memory Organization

A sample table is provided below for 7 entries. The LUT is specified in left to right and top to down fashion (Raster Scan order). As such design only has to look at the current LUT pointer to be able to determine if the current location is a defect or not.

Table 6-137. Figure 14: RAWFE Sample DPC Table

10	1400	0
236	1107	1
236	1200	1
488	10	1
488	100	2
800	138	3
900	1000	1
900	1100	3

The method filed pertains to how the defect pixel is treated once it has been detected. Several possibilities arise including copying from one of the neighbors or using an average of the neighbors. The table below summarizes different methods that can be programmed for treating defect pixels.

Copyright © 2024 Texas Instruments Incorporated

Table 6-138. Figure 15:RAWFE LUT DPC Method(s)

Method Value	Functional equivalent
0	Replace with Black or white dot based on config
1	Dout = d4
2	Dout = d5
3	Dout = (d4 + d5)/2
4	Dout = (d2 + d7)/2
5	Dout = d2
6	Dout = d7
7	Dout = ((d2 + d7)/2 + (d4 + d5)/2)/2

The pixel arrangement and numbering convention is depicted in the image below with ‘d0’ being considered the center pixel.

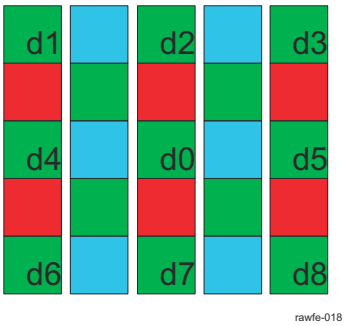


Figure 6-68. RAWFE DPC Pixel numbering convention

When the DPC method is set to ‘0’, the output is either a black or a white pixel (depending on a separate configuration register). With this method, the LUT dpc can be used in conjunction with OTF DPC in a way that the LUT programmed defects are almost always identified by the adaptive DPC. This can be used to accurately correct for all known defects and then using adaptive DPC to correct for temperature or other operating condition based defects. (Defect pixels are strongly correlated with analog gain).

6.7.3.2.4.3.2 RAWFE On-The-Fly (OTF) DPC

The LUT based DPC suffers from the drawback that each sensor die has to be calibrated for defects. The calibration process adds cost to the sensor dies, as such the preferred DPC method relies on automatically detecting the defects based on thresholds.

The OTF DPC algorithm detects defective pixels by comparing the current pixel (d0 on the left of the figure below) against its 8 neighboring pixels with the same color (d1 ~ d8 on the left of the figure below). Let’s use dmax and dmin to denote the maximum and minimum of d1 ~ d8 respectively. If the current pixel d0 is greater than the sum of dmax and a detection threshold T (i.e., d0 > dmax + T) as shown on the right of the figure below, then the current pixel is considered a defect and its value is replaced by dmax. Similarly, if d0 < dmin – T, then the current pixel is replaced by dmin.

Copyright © 2024 Texas Instruments Incorporated

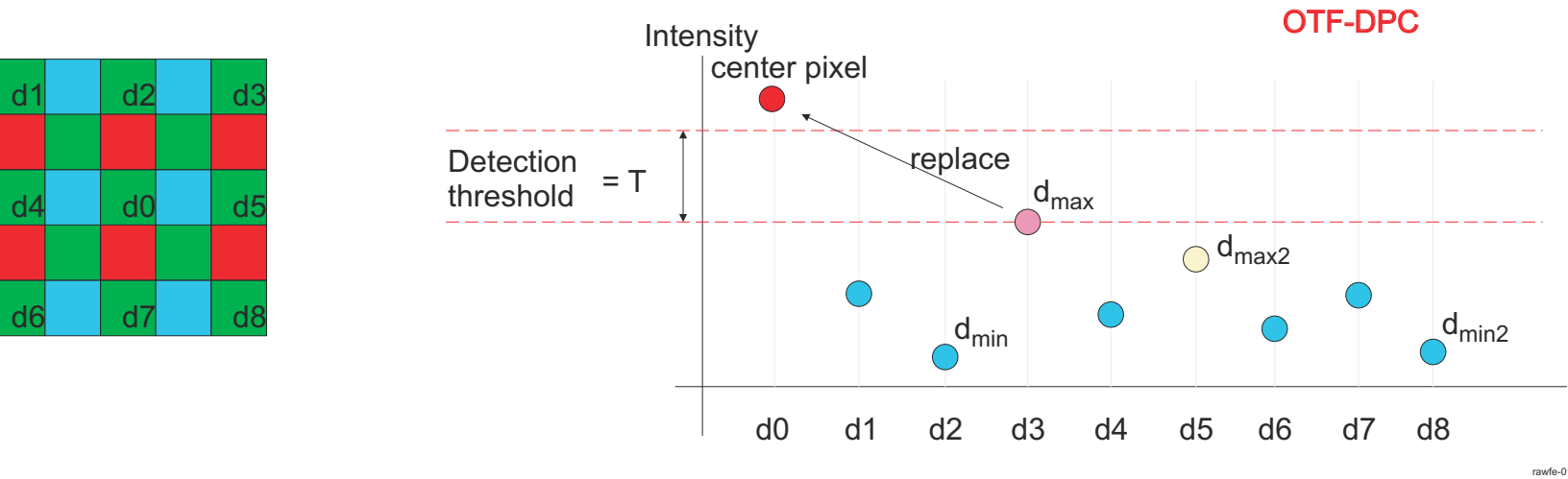


Figure 6-69. RAWFE OTF DPC

The detection threshold T is made adaptive by calculating the local image intensity and using a programmable look-up-table. The local intensity is approximated by the average of the second largest and the second smallest pixel values among d1 ~ d8 (dmax2 and dmin2 respectively in the figure above). With this average value, the threshold T is obtained from the look-up-table with linear interpolation. The look-up-table contains the detection thresholds (U16) at the average values of 0, 512, 1024, 2048, 4096, 8192, 16384, 32768 and the corresponding slope values (S12Q8) for linear interpolation.

6.7.3.2.4.4 RAWFE Lens Shading Correction (LSC) and Digital Gain (DG) Block

The lens shading module corrects for the lens fall of (loss of light) at the corners of the lens. Figure 6-70 shows a sample image of this phenomenon and the corresponding result after treating with LSC operation.

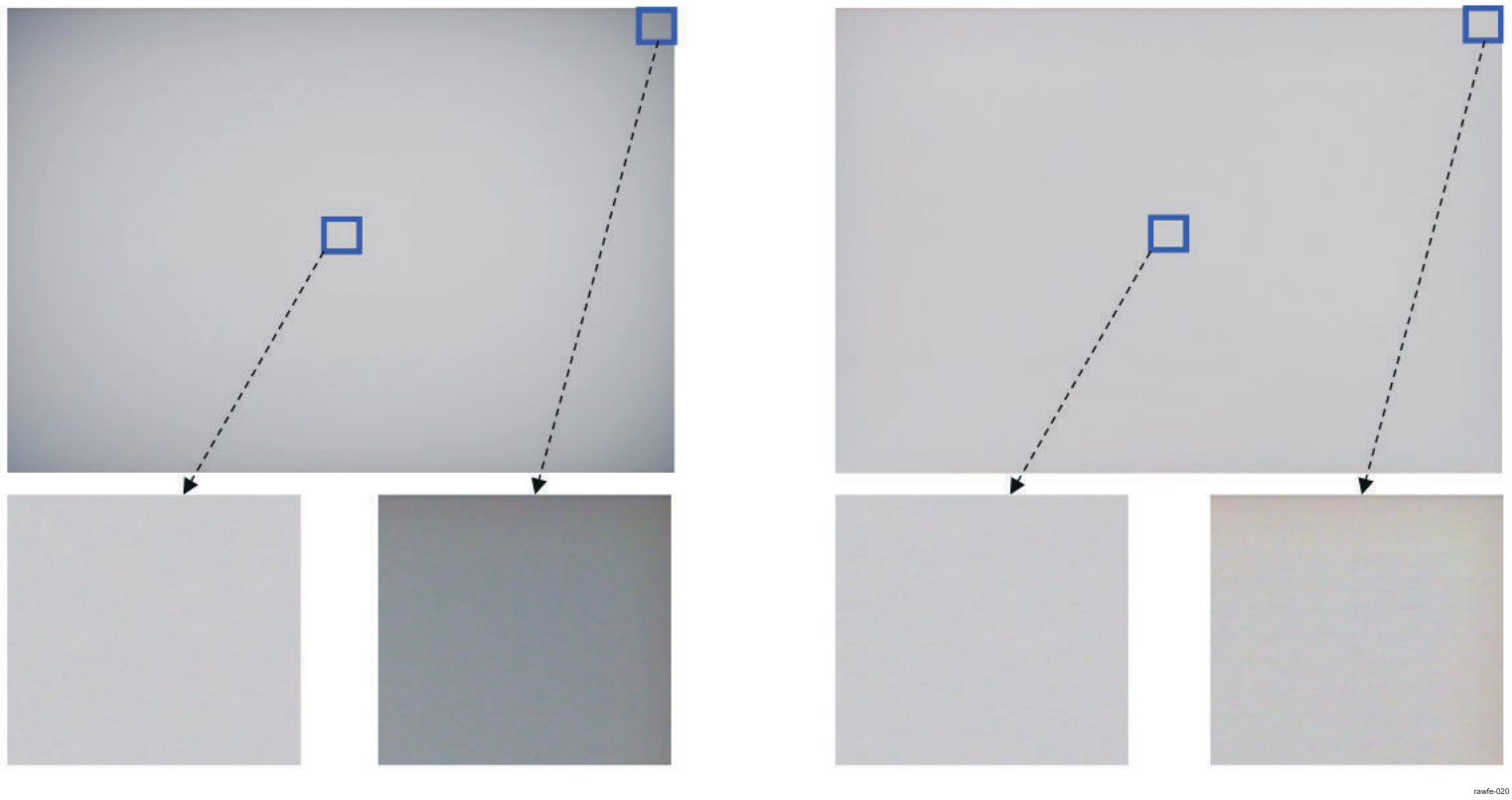


Figure 6-70. RAWFE Before and After LSC

Copyright © 2024 Texas Instruments Incorporated

The image on the left shows a block at the center of the lens and a corresponding block at the edge of the lens. Both blocks are equally illuminated, as such should appear equally bright on the image. However, due to lens fall off, the block at the edge appears significantly darker than the block at the center. The image on the right shows the corresponding result after LSC gains have been applied.

This block (2D-LSC) contains lens shading correction by multiplying an image with a gain factor 2-D map, pixel by pixel. The image is conceived to be in Bayer CFA format having a 2x2 color pattern. The gain factor map is stored in internal MEMORY down-sampled, and is accessed and up-sampled by the LSC module to pixel resolution before being applied to the pixel data. The key difference from previous generation LSC is that the gain map is stored internally in the memory instead of relying on DRAM fetch.

The LSC module does not implement any Region of Interest (ROI) functionality and the LSC gain mask is applied on the entire image. Further, only an 8 bit gain is stored in the mask. The gain can be stored in different formats depending on the range. The 8 bits of gain can be used to represent different ranges based on the lsccfg.GAIN_FORMAT register as per the table below

- 0 : U8Q8 -> 0 to 0.996
- 1 : U8Q8 +1 -> 1 to 1.996
- 2 : U8Q7 -> 0 to 1.992
- 3 : U8Q7 +1 -> 1 to 2.992
- 4 : U8Q6 -> 0 to 2.984
- 5 : U8Q6 +1 -> 1 to 3.984
- 6 : U8Q5 -> 0 to 6.968
- 7 : U8Q5 +1 -> 1 to 7.968

Note

The offset field is removed from the implementation of the LUT and only the gain field is retained. This is different than previous implementations.

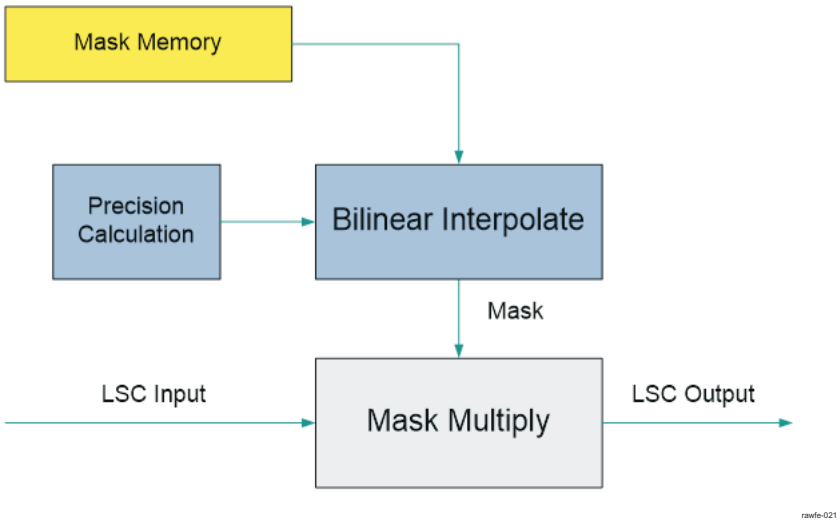


Figure 6-71. RAWFE LSC block diagram

6.7.3.2.4.4.1 RAWFE LSC Features Supported

- Gain map is MxN down-sampled, M being the horizontal sampling factor, N being the vertical sampling factor, M and N being {8,16,32.64,128 } independently.
- The size of the internal memory is based on a 2Mpix sensor with 16x32 downscaling ratio. As such the size is approximately 19 Kbytes-(4758 locations with 4 Bytes/location) -> Each entry has gains for all 4 colors. This corresponds to a sensor resolution of 1928 x 1208
- Support 8-bit entry in the gain map (in U8Q8, U8Q7, U8Q6, and U8Q5 format with optional base of 1.0 to shift the range up)

Copyright © 2024 Texas Instruments Incorporated

Figure 6-70 illustrates the LSC active region and map up-scaling feature. (The maps are stored independently per color channel)

6.7.3.2.4.4.2 RAWFE LSC Image Framing with Respect to Gain Map Samples

The gain maps are stored MxM downsampled, and needs to be upsampled by LSC hardware before being applied to the image. It is most straightforward for LSC hardware if the gain map grid is aligned with the input image grid. In other words, first pixel corresponds to a source gain map entry, rather than being upsampled through interpolation. Figure 6-72 shows the LSC active region with respect to the gain map grid.

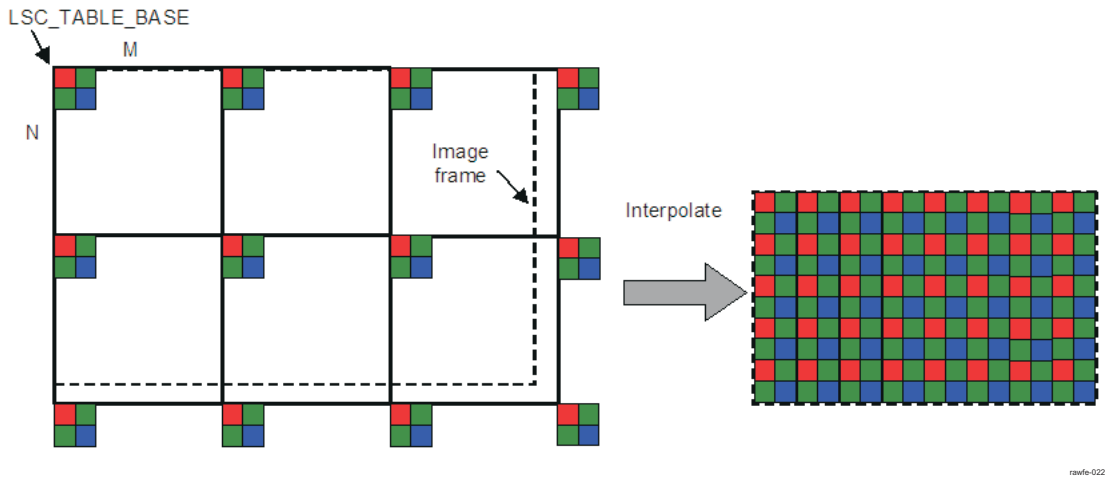


Figure 6-72. RAWFE LSC active region with respect to gain map samples

For an image frame size of W x H (output and input are the same size), gain map being MxN downsampled, it needs (ceil(W / M) +1) x (ceil(H / N) +1) x 4 bytes of gain map data in internal memory. (NOTE: ceil() represents the ceiling function).

The LSC module is designed to work with Bayer CFA data, having R/Gr/Gb/B color pattern. For the purpose of functional description, we assume Red is the starting color, but any other starting color or other 2x2 color pattern can be used by placing color gains in the appropriate order.

Each 2x2 set of samples is stored together in a 32-bit word in internal memory. For R/Gr/Gb/B CFA data and 8x8 downsampled gains, the following order is assumed in the gain map, using conventional (X, Y) coordinate notation (X for horizontal offset and Y for vertical offset):

Line 0: R(0,0) Gr(1,0) Gb(0,1) B(1,1) R(8,0) Gr(9,0) Gb(8,1) B(9,1)...

Line 1: R(0,8) Gr(1,8) Gb(0,9) B(1,9) R(8,8) Gr(9,8) Gb(8,9) B(9,9)...

6.7.3.2.4.5 RAWFE Gain & Offset Block

The gain block allow 2 primary functions

- Allow for digital gain to be applied if the image is too dark even after analog gain and exposure time have been set to the maximum
- Independent gain setting for each color channel in 13 bit format U13Q9 with 9 bits of fraction.
  - Allows a gain value from 0 to 31.996 in steps of 1/256.

The Gain block can be configured to apply an offset on top of the gain. This is to support the log compressed image mode, where in the traditional WB Gain becomes an offset.

6.7.3.2.4.6 RAWFE H3A

6.7.3.2.4.6.1 RAWFE H3A Overview

The H3A module supports the control loops for autofocus, auto white balance, and auto exposure by collecting statistics about the raw image. The metrics are used to adjust parameters for processing the imaging/video data. There are two main blocks in the H3A module:

- Autofocus (AF) engine:

Copyright © 2024 Texas Instruments Incorporated

The AF submodule extracts and filters the red, green, and blue data from input image data and provides the accumulation or peaks of the data in a specified region. The specified region is a 2D block of data referred to as a paxel. The AF engine supports the following features:

- Peak mode in a paxel: Accumulation of the maximum focus value (FV) of each line in a paxel
- Accumulation mode in a paxel
- Accumulation of horizontal and vertical focus value in a paxel
- Up to 12 paxels in the horizontal direction and up to 12 paxels in the vertical direction with vertical focus
- Up to 36 paxels in the horizontal direction and up to 128 paxels in the vertical direction with horizontal focus only
- Programmable width and height for the paxel/window
- Programmable red, green, and blue position within a 2 × 2 matrix
- Separate horizontal start for paxel and filtering
- Programmable vertical and horizontal line increments within a paxel
- Horizontal FV uses parallel infinite impulse response (IIR) filters configured in a dual-biquad configuration with individual coefficients (two filters with 11 coefficients each). The filters are intended to compute the sharpness/peaks in the frame on which to focus.
- Vertical FV uses a 5-tap FIR filter with 8-bit coefficients. With horizontal steps each paxel has up to 32 columns to be maintained for vertical FV calculation.

• Auto exposure and auto white balance (AE/AWB) engine:

  The AE/AWB engine accumulates values and checks for saturated values in a subsampling of the video data. In the case of the AE/AWB, the 2D block of data is referred to as a window. Thus, other than having different names, paxels and windows are essentially the same. However, the numbers, dimensions, and starting positions of AF paxels and AE/AWB windows are programmable separately. AE/AWB supports the following features:

  - Accumulate clipped pixels along with all nonsaturated pixels in each window per color
  - Accumulate the sum of squared pixels in each window per color
  - Minimum and maximum pixel values in each window per color
  - Support for up to 36 horizontal windows with sum + { sum_sq or min+max} output
  - Support for up to 56 horizontal windows with sum output
  - Support for up to 128 vertical windows
  - Programmable width and height for the windows. All windows in the frame are the same size.
  - Separate vertical start coordinate and height for a black row of paxels that is different than the remaining color paxels
  - Programmable horizontal sampling points in a window
  - Programmable vertical sampling points in a window

• Double-buffer for paxel/window accumulation
• H3A data path is 10 bits.
• Maximum input size is 4096 pixels.

6.7.3.2.4.6.2 RAWFE H3A Top-Level Block Diagram

The block diagram in Figure 6-73 shows the process of the AF and AE/AWB data paths through the H3A module.

Copyright © 2024 Texas Instruments Incorporated

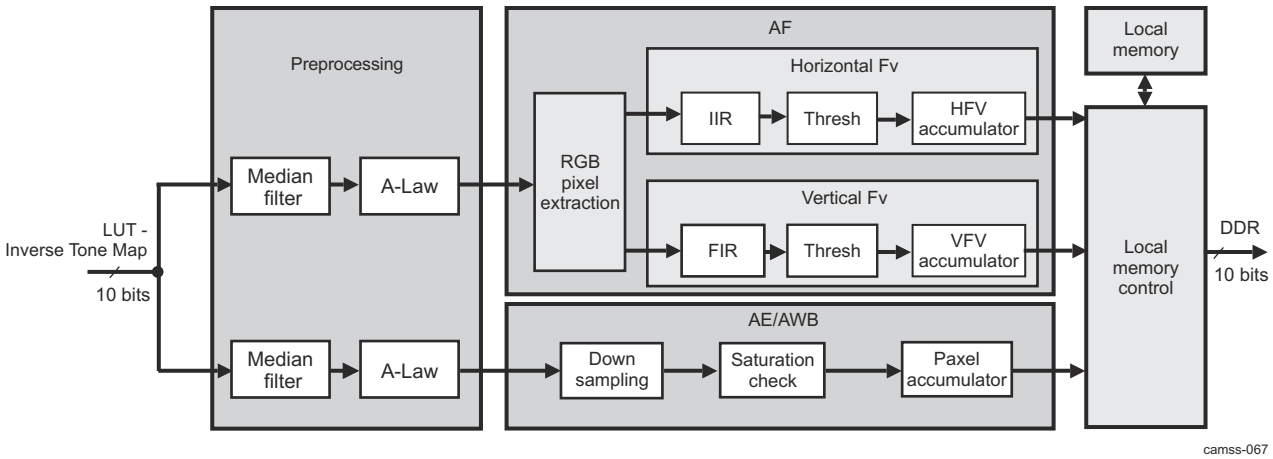


Figure 6-73. RAWFE H3A Top-Level Block Diagram

6.7.3.2.4.6.3 RAWFE H3A Line Framing Logic

In certain cases the number of clock cycles between HD pulses is greater than the line buffer included in the H3A. To solve this problem a framing module was added before the line buffer. The framing module uses the VISS_RAWFE_H3A_LINE_START register to find the position of the first pixel to place into the line buffer. All other registers reference this point as the 0 pixel for their start positions. The line size is 4096 pixels. After 4096 clock cycles the framing logic disables the line buffer and waits until the next HD. If the next HD comes before 4096 clock cycles, then the active region ends immediately and the counter waits for the VISS_RAWFE_H3A_LINE_START register count to be reached again. For the vertical position the VISS_RAWFE_H3A_LINE_START[31:16] SLV bit field can be used to determine where the start point of the frame is relative to the rising edge of VD. This logic allows for an active frame to cross VD boundaries and remain in the same frame.

Figure 6-74 shows the RAWFE H3A frame format settings.

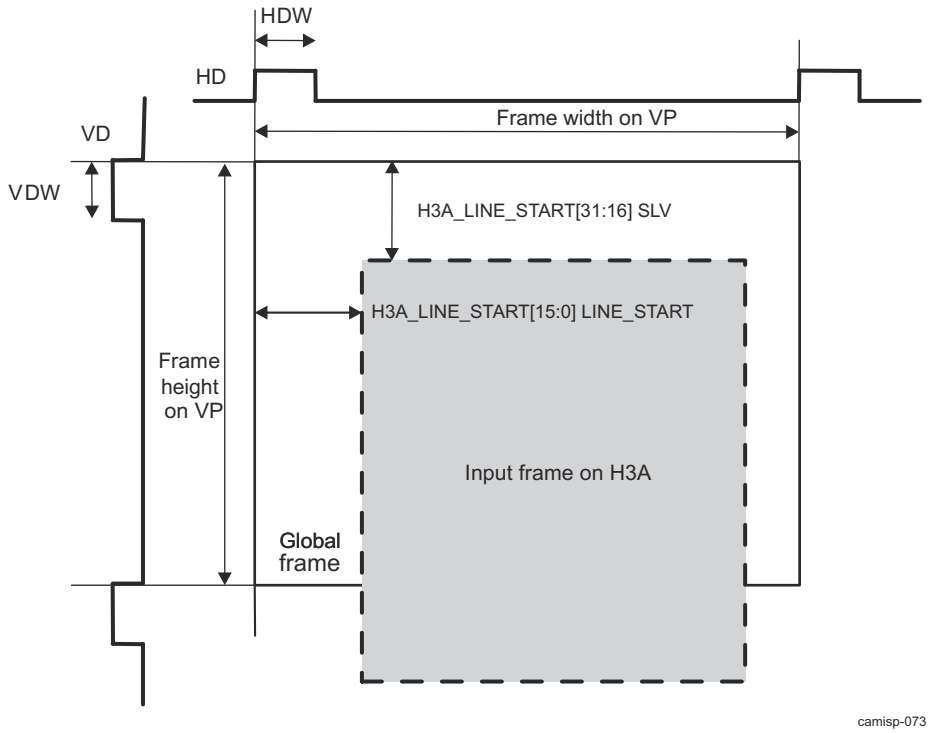


Figure 6-74. RAWFE H3A Frame Format Settings

Copyright © 2024 Texas Instruments Incorporated

Note

(Frame width on VP) - (VISS_RAWFE_H3A_LINE_START[15:0] LINE_START) must be less than or equal to 4096, because the H3A memory lines are limited to 4096 pixels.

6.7.3.2.4.6.4 RAWFE H3A Optional Preprocessing

The input to the H3A module is 10-bit RAW data from the IPIPEIF. A 10-bit to 8-bit A-Law compression step can be enabled and disabled separately for the AF engine (the VISS_RAWFE_H3A_PCR[1] AF_ALAW_EN bit) and the AE/AWB engine (the VISS_RAWFE_H3A_PCR[17] AEW_ALAW_EN bit). A-Law compression offers added protection against overflowing the accumulators.

If the A-Law table is enabled, the output is 10 bits, with the upper two bits filled with 0.

For the AF process, a horizontal median filter can be enabled and disabled (the VISS_RAWFE_H3A_PCR[2] AF_MED_EN bit) before A-Law compression. This filter is useful for reducing temperature-induced noise. The horizontal median filter calculates the absolute difference between the current pixel (i) and pixel (i – 2), and between the current pixel (i) and pixel (i + 2). If the absolute difference exceeds a threshold, and the sign of the differences is the same, the average of pixel (i – 2) and pixel (i + 2) replaces pixel (i). The threshold of the horizontal median filter can be set in the VISS_RAWFE_H3A_PCR[10:3] MED_TH bit field.

6.7.3.2.4.6.5 RAWFE H3A Autofocus Engine

The AF engine works by extracting each green (Gr or Gb) pixel from the video stream and subtracts a fixed offset of 128 or 512 (depending of whether A-Law is enabled or disabled) from the pixel value. The offset value is then passed through an IIR filter and the absolute value of the filter output is the focus value (FV). Both FV and FV^2 are produced. The FV and FV^2 values can be accumulated or the maximum for each line/column can be accumulated. The following sections describe this process in more detail.

6.7.3.2.4.6.5.1 RAWFE H3A Paxel Extraction

From the paxel starting coordinate (the VISS_RAWFE_H3A_AFPAXSTART[27:16] PAXSH and VISS_RAWFE_H3A_AFPAXSTART[11:0] PAXSV bit fields) specifies the starting point of the paxel grid, with respect to first pixel of the input image frame.

The paxel starting coordinate also indicates which color pixels are extracted if VF is enabled (that is, if VISS_RAWFE_H3A_PCR[20] AF_VF_EN = 1). Normally, either Gr or Gb is used for AF, but it is not important to the hardware whether it is red, green, or blue. If VF is not enabled, then the red, green, and blue pixel extraction is controlled by the VISS_RAWFE_H3A_PCR[13:11] RGBPOS bit field to extract the correct colors from the input stream. Figure 6-75 shows the available options for this bit field. The red and blue pixel positions are interchangeable. For each 2 × 2 grid, the green pixels are summed to create a single value. Because of this, the amplitude of the green output contains 2 pixels, while the red and blue outputs each contain 1 pixel.

Copyright © 2024 Texas Instruments Incorporated

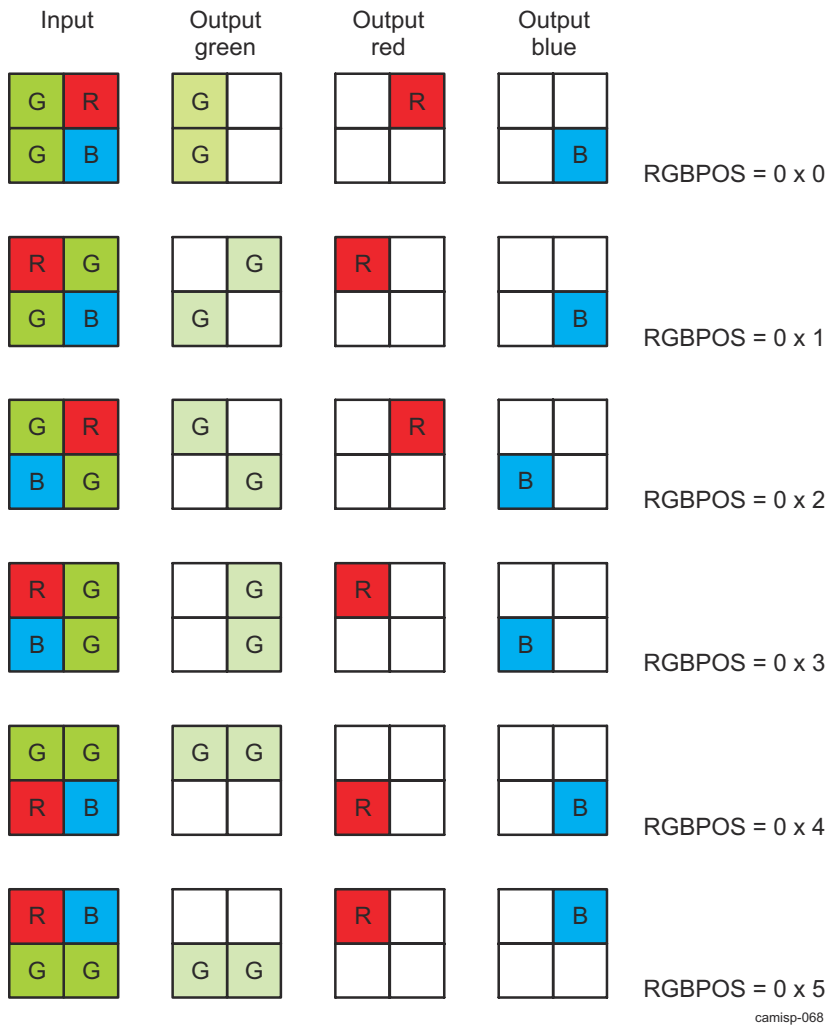


Figure 6-75. RAWFE H3A Red, Green, and Blue Pixel Extraction Examples

Each paxel is VISS_RAWFE_H3A_AFPAX1[23:16] PAXW × VISS_RAWFE_H3A_AFPAX1[7:0] PAXH (width × height) pixels. Inside each paxel, horizontal FV can skip lines, operating on one every VISS_RAWFE_H3A_AFPAX2[16:13] AFINCV lines. Vertical FV can skip columns, operating on one every VISS_RAWFE_H3A_AFPAX2[20:17] AFINCH columns. Up to 32 columns are supported for each paxel. If floor (PAXW/AFINCH) >= 32, only the first 32 designated columns are operated on. Because PAXW, PAXH, AFINCV, and AFINCH are all even numbers, AF always operates on the same green color, Gr or Gb. IIR filters for the horizontal FVs start operation at column VISS_RAWFE_H3A_AFIIRSH[11:0] IIRSH.

Figure 6-76 shows the RAWFE H3A horizontal/vertical fv paxel configuration.

Copyright © 2024 Texas Instruments Incorporated

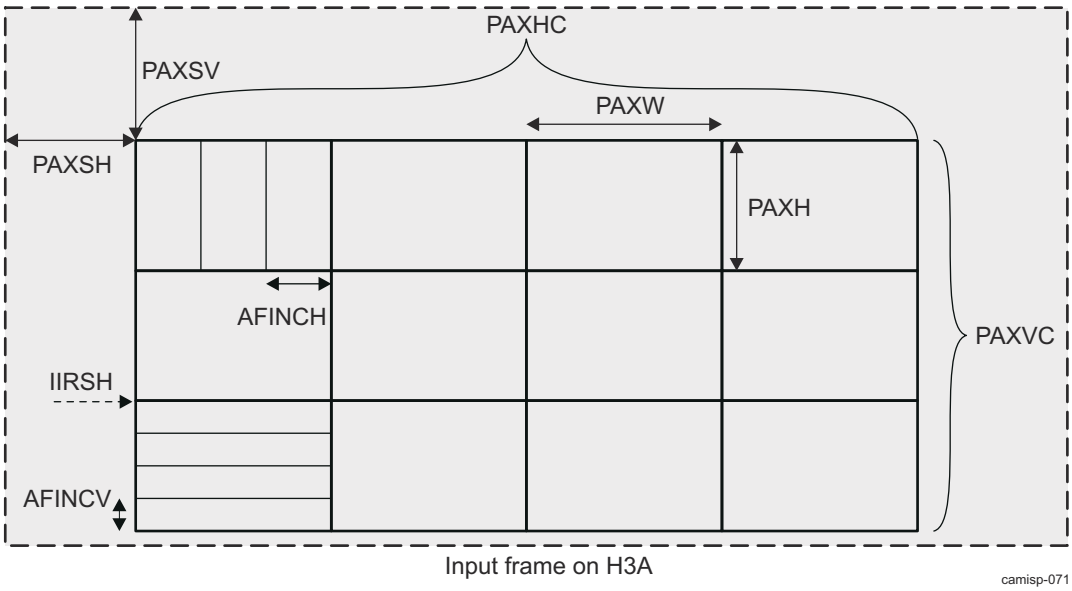


Figure 6-76. RAWFE H3A Horizontal/Vertical FV Paxel Configuration

Note

(VISS_RAWFE_H3A_AFPAXSTART[27:16] PAXSH) + (VISS_RAWFE_H3A_AFPAX2[5:0] PAXHC) × (VISS_RAWFE_H3A_AFPAX1[23:16] PAXW) <= [(Frame width on VP) – (VISS_RAWFE_H3A_LINE_START[15:0] LINE_START)] <= 4096

Table 6-139 lists the bit fields that configure the size and number of paxels.

Table 6-139. RAWFE H3A Paxel Register Field Descriptions

Bit Field	Bit Width	Description
VISS_RAWFE_H3A_AFPAX1[23:16] PAXW	8	Paxel width (in pixels)
VISS_RAWFE_H3A_AFPAX1[7:0] PAXH	8	Paxel height (in lines)
VISS_RAWFE_H3A_AFPAX2[5:0] PAXHC	6	Paxel count for horizontal direction
VISS_RAWFE_H3A_AFPAX2[12:6] PAXVC	7	Paxel count for vertical direction
VISS_RAWFE_H3A_AFPAX2[16:13] AFINCV	4	Line increments in a paxel
VISS_RAWFE_H3A_AFPAX2[20:17] AFINCH	4	Column increments in a paxel
VISS_RAWFE_H3A_AFPAXSTART[27:16] PAXSH	12	Paxel start position H
VISS_RAWFE_H3A_AFPAXSTART[11:0] PAXSV	12	Paxel start position V
VISS_RAWFE_H3A_AFIIRSH[11:0] IIRSH	12	IIR filter start position

The H3A AF engine also has an option for an advanced or normal stats collection mode. When 0xCA00 is written to the VISS_RAWFE_H3A_ADVANCED[31:15] ID bit field, then VISS_RAWFE_H3A_ADVANCED[0] AF_MODE can be used to toggle between normal and advanced AF stats collection mode. When the advanced AF stats collection mode is enabled, the ZEROS section of the AF paxel packet is filled with the sum of the maximum FVs, regardless of the color, from HFV_1 and HFV_2.

6.7.3.2.4.6.5.2 RAWFE H3A Horizontal FV Calculator

The FV calculator takes the unsigned red/green/blue extracted data and subtracts 128 or 512 (depending on whether A-Law is enabled) to place the data in the range –128 to 127 or –512 to 511.

After removing the offset, the data is sent through two parallel IIR filters configured in a dual-biquad configuration. Each filter uses a unique set of 11 programmable coefficients. Each coefficient is 12-bits-wide with 6 bits of decimal, S12Q6 (VISS_RAWFE_H3A_AFCOEF010 to VISS_RAWFE_H3A_AFCOEF0010 for SET0, and VISS_RAWFE_H3A_AFCOEF110 to VISS_RAWFE_H3A_AFCOEF1010 for SET1). The filter-shift

Copyright © 2024 Texas Instruments Incorporated

registers are cleared on each horizontal line at the position set by the register IIR horizontal start register (the VISS_RAWFE_H3A_AFIIRSH[11:0] IIRSH bit field). The absolute values of the output (16 bits wide with 4 bits of decimal, U16Q4) of both filters are then sent to the AF accumulator module. Signed clipping is performed during the FV calculation. If the input value is m bits (signed) and the required output value is n bits, clipping transforms the input to between -2^1 and 2^1. Values lower than -2^1 are set to -2^1, and values higher than 2^1 are set to 2^1.

6.7.3.2.4.6.5.3 RAWFE H3A HFV Accumulator

The horizontal focus value (HFV) accumulator takes the output of the horizontal IIR filter and accumulates values for each paxel. The size and number of paxels is configurable by registers.

Table 6-139 lists the register fields that configure the size and number of paxels:

- In peak mode (VISS_RAWFE_H3A_PCR[14] FVMODE = 0x1), the maximum value is accumulated.
- In sum mode (VISS_RAWFE_H3A_PCR[14] FVMODE = 0x0), all HFV_n are accumulated in a paxel.

The following equations detail the calculation for:

- Sum of pixel values used in HFV: The pixel values that are used for filtering and accumulation of HFV are also accumulated in this sum of pixel values.
- HFV_n (HFV_n_peak for peak mode or HFV_n_sum for sum mode)
- HFV_count_n
- HFV_sq_n (HFV_sq_n_peak for peak mode or HFV_sq_n_sum for sum mode)

n = 1 or 2 for IIR1 and IIR2, respectively.

For each paxel, these six values are available for each R, G, and B component.

```
for (k=0; k<PAXH) // Loop on paxel rows
{
 rowpeak_n = 0;
 for (l=0; l<PAXW; l++) // Loop on values within a row
 {
 aIIRout_n = ABS(IIRout_n);
 if (aIIRout_n >= threshold_n)
 {
 hfval = aIIRout_n - threshold_n;
 HFV_count_n++;
 }
 else hfval = 0;
 if (hfval > rowpeak_n)
 {
 rowpeak_n = HFV_n;
 }
 HFV_n_sum += hfval;
 HFV_sq_n_sum += (hfval* hfval + RNDADD) >> RNDSHIFT;
 } // Finished looping on values in a row
 HFV_n_peak += rowpeak_n;
 HFV_sq_n_peak += (rowpeak_n * rowpeak_n + RNDADD) >> RNDSHIFT;
}
```

- threshold_n is VISS_RAWFE_H3A_HVF_THR[15:0] HTHR1 and VISS_RAWFE_H3A_HVF_THR[31:16] HTHR2, respectively.
- IIRout_n is the IIRout_1 and IIRout_2 outputs, respectively.
- HFV_count_n and HFV_sq_n are not sent to the DMA interface if VF is disabled.
- RNDADD and RNDSHIFT depend on whether input pixels are 8-bit or 10-bit, and achieves rounding. This is automatically performed by the module.
- If VF is enabled, only the green color channel values are output to the DMA interface.
- In sum mode:
  - HFV_n = HFV_n_sum
  - HFV_sq_n = HFV_sq_n_sum
- In peak mode:
  - HFV_n = HFV_n_peak
  - HFV_sq_n = HFV_sq_n_peak

Copyright © 2024 Texas Instruments Incorporated

6.7.3.2.4.6.5.4 RAWFE H3A VFV Calculator

The VFV calculator takes the unsigned extracted data through two FIR filters, each with a set of five coefficients (VCOEF1_x, where x = 0 to 4, in the VISS_RAWFE_H3A_VFV_CFG1 and VISS_RAWFE_H3A_VFV_CFG2 registers for FIR 1, and VCOEF2_x, where x = 0 to 4, in the VISS_RAWFE_H3A_VFV_CFG3 and VISS_RAWFE_H3A_VFV_CFG4 registers). Each coefficient is 8 bits wide with 4 bits of decimal (S8Q4). The filter outcome is downshifted by 4 bits and takes an absolute value to produce a 16-bit unsigned value. This is then sent to threshold VISS_RAWFE_H3A_VFV_CFG2[31:16] VTHR1 for FIR 1, and VISS_RAWFE_H3A_VFV_CFG4[31:16] VTHR2 for FIR 2, and square logic to produce VFV_n and VFV_sq_n.

6.7.3.2.4.6.5.5 RAWFE H3A VFV Accumulator

The VFV accumulator takes the output of the vertical FIR filters and accumulates values for each paxel. The size and number of paxels is configurable by registers.

Table 6-139 lists the bitfields that configure the size and number of paxels.

The following equations detail the calculation for:

- VFV_n
- VFV_count_n
- VFV_sq_n

n = 1 or 2 for FIR1 and FIR2, respectively.

For each paxel, these six values are available for each R, G, and B component.

```
FIR_coef_n = [VCOEFn_0, VCOEFn_1, VCOEFn_2, VCOEFn_3, VCOEFn_4]; /* coefficient values in S8.4
format */
aFIRout_n = (ABS(inner_product(extracted_G, FIR_coef_n)) + 8) >> 4;
if (aFIRout_n >= threshold_n)
{
 VFV_n = aFIRout_n - threshold_n
 VFV_count_n++;
}
else VFV_n = 0;
VFV_sq_n = (VFV_n * VFV_n + RNDADD) >> RNDSHIFT;
```

- threshold_n is VISS_RAWFE_H3A_VFV_CFG2[31:16] VTHR1 and VISS_RAWFE_H3A_VFV_CFG4[31:16] VTHR2, respectively.
- FIRout_n is the FIRout_1 and FIRout_2 outputs, respectively.
- RNDADD and RNDSHIFT depend on whether the input pixels are 8-bit or 10-bit, and achieves rounding. This is automatically performed by the module.

6.7.3.2.4.6.6 RAWFE H3A AE/AWB Engine

The AE/AWB engine starts by dividing the frames into windows, and then subsamples each window into 2 × 2 blocks. For each subsampled 2 × 2 block, each pixel is accumulated. Also, each pixel is compared to a limit set in a register. If any pixels in a 2 × 2 block are greater than or equal to the limit, the block is not counted in the unsaturated block counter. Pixels greater than the limit are replaced by the limit, and the value of the pixel is accumulated.

The AE/AWB module has three output format modes, which are set through the VISS_RAWFE_H3A_AEWCFG[9:8] AEFMT bit field:

- Sum of square mode: VISS_RAWFE_H3A_AEWCFG[9:8] AEFMT = 0x0
- Min/max mode: VISS_RAWFE_H3A_AEWCFG[9:8] AEFMT = 0x1
- Sum-only mode: VISS_RAWFE_H3A_AEWCFG[9:8] AEFMT = 0x2

6.7.3.2.4.6.6.1 RAWFE H3A Subsampler

The subsampler partitions the frame into windows using the size, count, and starting location parameters shown on the left in Figure 6-77. Each window is further sampled down to a set of 2 × 2 blocks. The horizontal and vertical distances between the start of blocks within a window is programmable using the parameters shown on the right in Figure 6-77.

Copyright © 2024 Texas Instruments Incorporated

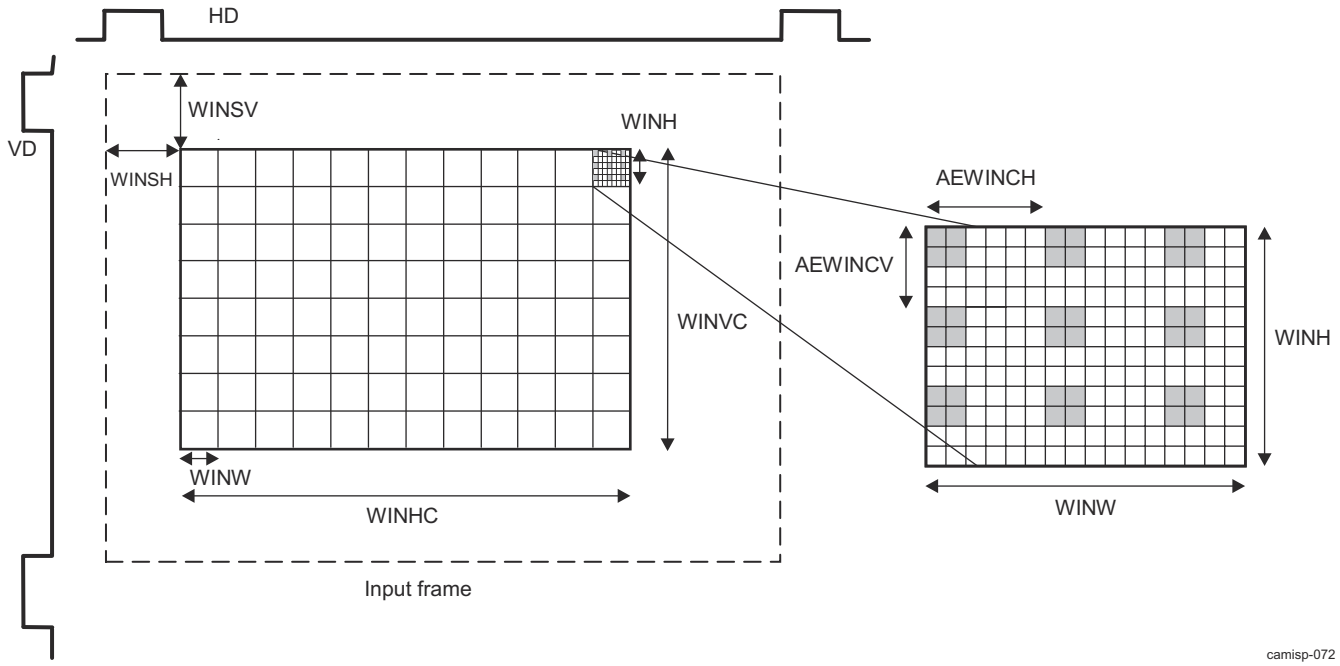


Figure 6-77. RAWFE H3A AE/AWB Window Configurations

Table 6-140 lists the bit fields that configure the window and block sizes, counts, and starting positions.

Table 6-140. RAWFE H3A AE/AWB Window Register Field Descriptions

Bit Field	Bit Width	Description
VISS_RAWFE_H3A_AEWWIN1[20:13] WINW	7	Window width (in pixels)
VISS_RAWFE_H3A_AEWWIN1[31:24] WINH	7	Window height (in lines)
VISS_RAWFE_H3A_AEWWIN1[5:0] WINHC	6	Window count for horizontal direction
VISS_RAWFE_H3A_AEWWIN1[12:6] WINVC	7	Window count for vertical direction
VISS_RAWFE_H3A_AEWINSTART[11:0] WINSH	12	Window start position H
VISS_RAWFE_H3A_AEWINSTART[27:16] WINSV	12	Window start position V
VISS_RAWFE_H3A_AEWSUBWIN[3:0] AEWINCH	4	Horizontal distance between subsamples
VISS_RAWFE_H3A_AEWSUBWIN[11:8] AEWINCV	4	Vertical distance between subsamples

6.7.3.2.4.6.6.2 RAWFE H3A Additional Black Row of AE/AWB Windows

In addition to the 128 rows of windows, the AE/AWB module provides support for an additional row of windows for black data. This data may be useful in determining the DC offset noise of the rest of the data. The black row of windows can be before or after the regular rows of windows. The vertical start line for the black row of windows is specified in the VISS_RAWFE_H3A_AEWINBLK[27:16] WINSV bit field, and the height is specified in the VISS_RAWFE_H3A_AEWINBLK[6:0] WINH bit field. The horizontal starting pixel and horizontal width of the black row of windows are the same as for the regular rows of windows.

Figure 6-78 shows a black row of windows before rows of windows.

Copyright © 2024 Texas Instruments Incorporated

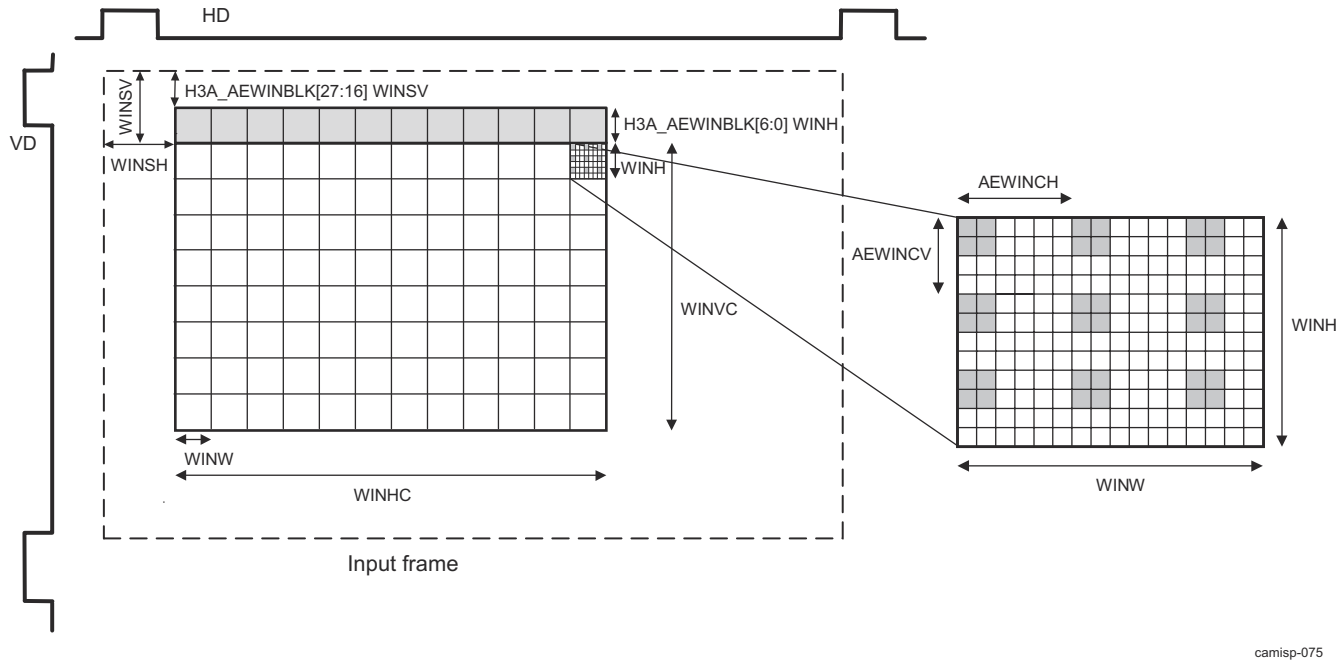


Figure 6-78. RAWFE H3A Black Row of Windows Before Regular Rows of Windows

Table 6-141 lists the bit fields that configure the window and block sizes, counts, and starting positions.

Table 6-141. RAWFE H3A AE/AWB Window With Additional Black Row Register Field Descriptions

Bit Field	Bit Width	Description
VISS_RAWFE_H3A_AEWWIN1[20:13] WINW	7	Window width (in pixels)
VISS_RAWFE_H3A_AEWWIN1[31:24] WINH	7	Window height (in lines)
VISS_RAWFE_H3A_AEWWIN1[5:0] WINHC	6	Window count for horizontal direction
VISS_RAWFE_H3A_AEWWIN1[12:6] WINVC	7	Window count for vertical direction
VISS_RAWFE_H3A_AEWINSTART[11:0] WINSH	12	Window start position H
VISS_RAWFE_H3A_AEWINSTART[27:16] WINSV	12	Window start position V
VISS_RAWFE_H3A_AEWSUBWIN[3:0] AEWINCH	4	Horizontal distance between subsamples
VISS_RAWFE_H3A_AEWSUBWIN[11:8] AEWINCV	4	Vertical distance between subsamples
VISS_RAWFE_H3A_AEWINBLK[27:16] WINSV	12	Window start position H for single black line
VISS_RAWFE_H3A_AEWINBLK[6:0] WINH	7	Window height (in lines) for single black line

6.7.3.2.4.6.6.3 RAWFE H3A Saturation Check

The saturation check module compares the data from the subsampler to the value programmed in the VISS_RAWFE_H3A_PCR[31:22] AVE2LMT bit field. This value is the maximum clipping value. If all 4 pixels in the 2 × 2 block are less than the AVE2LMT value, the value of the unsaturated block counter is incremented. There is one unsaturated block counter per window. The unsaturated block counters are later written to memory.

6.7.3.2.4.6.6.4 RAWFE H3A AE/AWB Accumulators

The output from the saturation check module and the subsampler module are separately accumulated for each pixel in every 2 × 2 pixel block for each window. Therefore, there are eight accumulators per window (one accumulator for each pixel in a 2 × 2 pixel block, times two sets of accumulators: clipped/saturated data and presaturated data). Each of the 4 pixels in the 2 × 2 pixel grid is associated with a color R, Gr, B, Gb); however, the output of these accumulators is referenced by position in the grid, not color.

The accumulators are 16 bits wide, and the accumulated data is 10 bits wide. Therefore, when a window contains more than 64 pixels of the same color, an overflow risk exists. This risk can be reduced by enabling the A-Law conversion in the preprocessing stage. See Section 6.7.3.2.4.6.4 for details.

Copyright © 2024 Texas Instruments Incorporated

The AE/AWB module has a shift value for the accumulation of pixel values that is set in the VISS_RAWFE_H3A_AEWCFG[3:0] SUMSHFT bit field.

6.7.3.2.4.6.7 RAWFE H3A DMA Interface

The DMA interface module takes the data from the AF engine and AE/AWB engine and builds packets to be sent to the memory through the BL module.

The data interface has separate start pointers for the AF and AE/AWB engines.

- The starting address for the AF engine is the VISS_RAWFE_H3A_AFBUFST[31:5] AFBUFST bit field.
- The starting address for the AE/AWB engine is the VISS_RAWFE_H3A_AFBUFST[31:5] AEWBUFST bit field.

The DMA interface module continuously loops through this data as it builds the packets. To optimize the transfer sizes, the DMA interface sends out an AF or AE transfer for each row of paxels or windows. This requires that each horizontal row of paxels or windows starts and ends on a 32-byte boundary. If a horizontal row of paxels or windows ends on a non-32 byte boundary, the hardware packs zeroes. The counts for the AEW that occur every eight windows is sent in the row with the eighth consecutive window.

Table 6-142 shows the packet formats for AF with vertical AF enabled.

Table 6-142. RAWFE H3A AF Packet Format With Vertical AF Enabled

Buffer Start Address (Byte Address) VISS_RAWFE_H3A_AFBUFST	31 16 15 0	
	Sum of pixel values used in HFV	(Paxel 0)
	HFV_1 (peak or sum)	(Paxel 0)
	HFV_sq_1 (peak or sum)	(Paxel 0)
	HFV_count_1	(Paxel 0)
	HFV_2 (peak or sum)	(Paxel 0)
	HFV_sq_2 (peak or sum)	(Paxel 0)
	HFV_count_2	(Paxel 0)
	Zeroes	(Paxel 0)
	VFV_1	(Paxel 0)
	VFV_sq_1	(Paxel 0)
	VFV_count_1	(Paxel 0)
	Zeroes	(Paxel 0)
	VFV_2	(Paxel 0)
	VFV_sq_2	(Paxel 0)
	VFV_count_2	(Paxel 0)
	Zeroes	(Paxel 0)
	Sum of pixel values used in HFV	(Paxel 1)
	HFV_1 (peak or sum)	(Paxel 1)
	HFV_sq_1 (peak or sum)	(Paxel 1)
	HFV_count_1	(Paxel 1)
	...	

Table 6-143 lists the packet formats for AE/AWB for sum of square mode (VISS_RAWFE_H3A_AEWCFG[9:8] AEFMT = 0x0) .

Copyright © 2024 Texas Instruments Incorporated

Table 6-143. RAWFE H3A AE/AWB Packet Format for Sum of Square Mode

	31 16	15 0	
Buffer address (byte address) VISS_RAWFE_H3A_AEWBUFST	Subsample Accum[1]	Subsample Accum[0]	Window 0 data
	Subsample Accum[3]	Subsample Accum[2]	
	Saturator Accum[1]	Saturator Accum[0]	
	Saturator Accum[3]	Saturator Accum[2]	
	Sum of squares[0]		
	Sum of squares[1]		
	Sum of squares[2]		
	Sum of squares[3]		
VISS_RAWFE_H3A_AEWBUFST + 32 bytes	Subsample Accum[1]	Subsample Accum[0]	Window 1 data
	Subsample Accum[3]	Subsample Accum[2]	
	Saturator Accum[1]	Saturator Accum[0]	
	Saturator Accum[3]	Saturator Accum[2]	
	Sum of squares[0]		
	Sum of squares[1]		
	Sum of squares[2]		
	Sum of squares[3]		
VISS_RAWFE_H3A_AEWBUFST + 64 bytes	Subsample Accum[1]	Subsample Accum[0]	Window 2 data
	Subsample Accum[3]	Subsample Accum[2]	
	Saturator Accum[1]	Saturator Accum[0]	
	Saturator Accum[3]	Saturator Accum[2]	
	Sum of squares[0]		
	Sum of squares[1]		
	Sum of squares[2]		
	Sum of squares[3]		
VISS_RAWFE_H3A_AEWBUFST + 96 bytes	Subsample Accum[1]	Subsample Accum[0]	Window 3 data
	Subsample Accum[3]	Subsample Accum[2]	
	Saturator Accum[1]	Saturator Accum[0]	
	Saturator Accum[3]	Saturator Accum[2]	
	Sum of squares[0]		
	Sum of squares[1]		
	Sum of squares[2]		
	Sum of squares[3]		
VISS_RAWFE_H3A_AEWBUFST + 128 bytes	Subsample Accum[1]	Subsample Accum[0]	Window 4 data
	Subsample Accum[3]	Subsample Accum[2]	
	Saturator Accum[1]	Saturator Accum[0]	
	Saturator Accum[3]	Saturator Accum[2]	

Copyright © 2024 Texas Instruments Incorporated

Table 6-143. RAWFE H3A AE/AWB Packet Format for Sum of Square Mode (continued)

	31	16 15	0
	Sum of squares[0]		
	Sum of squares[1]		
	Sum of squares[2]		
	Sum of squares[3]		
VISS_RAWFE_H3A_AEWBUFST + 160 bytes	Subsample Accum[1]	Subsample Accum[0]	Window 5 data
	Subsample Accum[3]	Subsample Accum[2]	
	Saturator Accum[1]	Saturator Accum[0]	
	Saturator Accum[3]	Saturator Accum[2]	
	Sum of squares[0]		
	Sum of squares[1]		
	Sum of squares[2]		
	Sum of squares[3]		
VISS_RAWFE_H3A_AEWBUFST + 192 bytes	Subsample Accum[1]	Subsample Accum[0]	Window 6 data
	Subsample Accum[3]	Subsample Accum[2]	
	Saturator Accum[1]	Saturator Accum[0]	
	Saturator Accum[3]	Saturator Accum[2]	
	Sum of squares[0]		
	Sum of squares[1]		
	Sum of squares[2]		
	Sum of squares[3]		
VISS_RAWFE_H3A_AEWBUFST + 224 bytes	Subsample Accum[1]	Subsample Accum[0]	Window 7 data
	Subsample Accum[3]	Subsample Accum[2]	
	Saturator Accum[1]	Saturator Accum[0]	
	Saturator Accum[3]	Saturator Accum[2]	
	Sum of squares[0]		
	Sum of squares[1]		
	Sum of squares[2]		
	Sum of squares[3]		
VISS_RAWFE_H3A_AEWBUFST + 256 bytes	Unsaturated count, win 1	Unsaturated count, win 0	Unsaturated block count for the above 8 windows
	Unsaturated count, win 3	Unsaturated count, win 2	
	Unsaturated count, win 5	Unsaturated count, win 4	
	Unsaturated count, win 7	Unsaturated count, win 6	
	Data for next eight windows, and so on. If the total number of windows is not a multiple of 8, the unsaturated counters are written immediately following the last window data. For example, if the total number of windows (including the black row) is 43, the first 40 windows are written out as per the 272-byte boundary above. Then the remaining three windows are written at +0, +32, and +64 bytes. The counts are written out at +96 instead of +256-byte boundary.		

Copyright © 2024 Texas Instruments Incorporated

Table 6-143. RAWFE H3A AE/AWB Packet Format for Sum of Square Mode (continued)

31	16 15	0

Table 6-144 lists the packet formats for AE/AWB in minimum-maximum mode (VISS_RAWFE_H3A_AEWCFG[9:8] AEFMT = 0x1).

Table 6-144. RAWFE H3A AE/AWB Packet Format for Minimum-Maximum Mode

	31	16 15	0
Buffer address (byte address) VISS_RAWFE_H3A_AEWBUFST	Subsample Accum[1]	Subsample Accum[0]	Window 0 data
	Subsample Accum[3]	Subsample Accum[2]	
	Saturator Accum[1]	Saturator Accum[0]	
	Saturator Accum[3]	Saturator Accum[2]	
	Minimum[1]	Minimum[0]	
	Minimum[3]	Minimum[2]	
	Maximum[1]	Maximum[0]	
	Maximum[3]	Maximum[2]	
VISS_RAWFE_H3A_AEWBUFST + 32 bytes	Subsample Accum[1]	Subsample Accum[0]	Window 1 data
	Subsample Accum[3]	Subsample Accum[2]	
	Saturator Accum[1]	Saturator Accum[0]	
	Saturator Accum[3]	Saturator Accum[2]	
	Minimum[1]	Minimum[0]	
	Minimum[3]	Minimum[2]	
	Maximum[1]	Maximum[0]	
	Maximum[3]	Maximum[2]	
VISS_RAWFE_H3A_AEWBUFST + 64 bytes	Subsample Accum[1]	Subsample Accum[0]	Window 2 data
	Subsample Accum[3]	Subsample Accum[2]	
	Saturator Accum[1]	Saturator Accum[0]	
	Saturator Accum[3]	Saturator Accum[2]	
	Minimum[1]	Minimum[0]	
	Minimum[3]	Minimum[2]	
	Maximum[1]	Maximum[0]	
	Maximum[3]	Maximum[2]	
VISS_RAWFE_H3A_AEWBUFST + 96 bytes	Subsample Accum[1]	Subsample Accum[0]	Window 3 data
	Subsample Accum[3]	Subsample Accum[2]	
	Saturator Accum[1]	Saturator Accum[0]	
	Saturator Accum[3]	Saturator Accum[2]	
	Minimum[1]	Minimum[0]	
	Minimum[3]	Minimum[2]	
	Maximum[1]	Maximum[0]	
	Maximum[3]	Maximum[2]	

Copyright © 2024 Texas Instruments Incorporated

Table 6-144. RAWFE H3A AE/AWB Packet Format for Minimum-Maximum Mode (continued)

	31 16	15 0	
VISS_RAWFE_H3A_AEWBUFST + 128 bytes	Subsample Accum[1]	Subsample Accum[0]	Window 4 data
	Subsample Accum[3]	Subsample Accum[2]	
	Saturator Accum[1]	Saturator Accum[0]	
	Saturator Accum[3]	Saturator Accum[2]	
	Minimum[1]	Minimum[0]	
	Minimum[3]	Minimum[2]	
	Maximum[1]	Maximum[0]	
	Maximum[3]	Maximum[2]	
VISS_RAWFE_H3A_AEWBUFST + 160 bytes	Subsample Accum[1]	Subsample Accum[0]	Window 5 data
	Subsample Accum[3]	Subsample Accum[2]	
	Saturator Accum[1]	Saturator Accum[0]	
	Saturator Accum[3]	Saturator Accum[2]	
	Minimum[1]	Minimum[0]	
	Minimum[3]	Minimum[2]	
	Maximum[1]	Maximum[0]	
	Maximum[3]	Maximum[2]	
VISS_RAWFE_H3A_AEWBUFST + 192 bytes	Subsample Accum[1]	Subsample Accum[0]	Window 6 data
	Subsample Accum[3]	Subsample Accum[2]	
	Saturator Accum[1]	Saturator Accum[0]	
	Saturator Accum[3]	Saturator Accum[2]	
	Minimum[1]	Minimum[0]	
	Minimum[3]	Minimum[2]	
	Maximum[1]	Maximum[0]	
	Maximum[3]	Maximum[2]	
VISS_RAWFE_H3A_AEWBUFST + 224 bytes	Subsample Accum[1]	Subsample Accum[0]	Window 7 data
	Subsample Accum[3]	Subsample Accum[2]	
	Saturator Accum[1]	Saturator Accum[0]	
	Saturator Accum[3]	Saturator Accum[2]	
	Minimum[1]	Minimum[0]	
	Minimum[3]	Minimum[2]	
	Maximum[1]	Maximum[0]	
	Maximum[3]	Maximum[2]	
VISS_RAWFE_H3A_AEWBUFST + 256 bytes	Unsaturated count, win 1	Unsaturated count, win 0	Unsaturated block count for the above 8 windows
	Unsaturated count, win 3	Unsaturated count, win 2	
	Unsaturated count, win 5	Unsaturated count, win 4	
	Unsaturated count, win 7	Unsaturated count, win 6	

Copyright © 2024 Texas Instruments Incorporated

Table 6-144. RAWFE H3A AE/AWB Packet Format for Minimum-Maximum Mode (continued)

31	16 15	0
	Data for next eight windows, and so on. If the total number of windows is not a multiple of 8, the unsaturated counters are written immediately following the last window data. For example, if the total number of windows (including the black row) is 43, the first 40 windows are written out as per the 272-byte boundary above. Then the remaining three windows are written at +0, +32, and +64 bytes. The counts are written out at +96 instead of +256-byte boundary.	

Table 6-145 lists the packet formats for AE/AWB in sum-only mode (VISS_RAWFE_H3A_AEWCFG[9:8] AEFMT = 0x2).

Table 6-145. RAWFE H3A AE/AWB Packet Format for Sum-Only Mode

31	16 15	0
Buffer address (byte address) VISS_RAWFE_H3A_AEWBUFST	Subsample Accum[1]	Subsample Accum[0]	Window 0 data
	Subsample Accum[3]	Subsample Accum[2]	
	Saturator Accum[1]	Saturator Accum[0]	
	Saturator Accum[3]	Saturator Accum[2]	
VISS_RAWFE_H3A_AEWBUFST + 32 bytes	Subsample Accum[1]	Subsample Accum[0]	Window 1 data
	Subsample Accum[3]	Subsample Accum[2]	
	Saturator Accum[1]	Saturator Accum[0]	
	Saturator Accum[3]	Saturator Accum[2]	
VISS_RAWFE_H3A_AEWBUFST + 64 bytes	Subsample Accum[1]	Subsample Accum[0]	Window 2 data
	Subsample Accum[3]	Subsample Accum[2]	
	Saturator Accum[1]	Saturator Accum[0]	
	Saturator Accum[3]	Saturator Accum[2]	
VISS_RAWFE_H3A_AEWBUFST + 96 bytes	Subsample Accum[1]	Subsample Accum[0]	Window 3 data
	Subsample Accum[3]	Subsample Accum[2]	
	Saturator Accum[1]	Saturator Accum[0]	
	Saturator Accum[3]	Saturator Accum[2]	
VISS_RAWFE_H3A_AEWBUFST + 128 bytes	Subsample Accum[1]	Subsample Accum[0]	Window 4 data
	Subsample Accum[3]	Subsample Accum[2]	
	Saturator Accum[1]	Saturator Accum[0]	
	Saturator Accum[3]	Saturator Accum[2]	
VISS_RAWFE_H3A_AEWBUFST + 160 bytes	Subsample Accum[1]	Subsample Accum[0]	Window 5 data
	Subsample Accum[3]	Subsample Accum[2]	

Copyright © 2024 Texas Instruments Incorporated

Table 6-145. RAWFE H3A AE/AWB Packet Format for Sum-Only Mode (continued)

	31 — 16	15 — 0	
VISS_RAWFE_H3A_AEWBUFST + 192 bytes	Saturator Accum[1]	Saturator Accum[0]	Window 6 data
	Saturator Accum[3]	Saturator Accum[2]	
	Subsample Accum[1]	Subsample Accum[0]	
	Subsample Accum[3]	Subsample Accum[2]	
VISS_RAWFE_H3A_AEWBUFST + 224 bytes	Saturator Accum[1]	Saturator Accum[0]	Window 7 data
	Saturator Accum[3]	Saturator Accum[2]	
	Subsample Accum[1]	Subsample Accum[0]	
	Subsample Accum[3]	Subsample Accum[2]	
VISS_RAWFE_H3A_AEWBUFST + 256 bytes	Saturator Accum[1]	Saturator Accum[0]	Unsaturated block count for the above 8 windows
	Saturator Accum[3]	Saturator Accum[2]	
	Unsaturated count, win 1	Unsaturated count, win 0	
	Unsaturated count, win 3	Unsaturated count, win 2	
	Unsaturated count, win 5	Unsaturated count, win 4	
	Unsaturated count, win 7	Unsaturated count, win 6	
	Data for next eight windows, and so on. If the total number of windows is not a multiple of 8, the unsaturated counters are written immediately following the last window data. For example, if the total number of windows (including black row) is 43, the first 40 windows are written out as per the 272-byte boundary above. Then the remaining three windows are written at +0, +32, and +64 bytes. The counts are written out at +96 instead of +256-byte boundary.		

6.7.3.2.4.6.8 RAWFE H3A Events and Status Checking

The AF and AEW engines generate an interrupt event at the end of processing each frame. However, these two interrupts are internally tied together so that only one H3A interrupt signal is generated. If the AF engine and AEW engine do not process the same frame concurrently, this should not be an issue. However, if they do run concurrently, one of two outcomes may occur:

- The H3A interrupt may seem to trigger only once for each frame. This can happen when the processing for the AF and AEW engines finishes at or near the same time. The interrupt service routine (ISR) does not have enough time to clear the interrupt flag for the first interrupt before the second interrupt occurs.
- The H3A interrupt may trigger twice for each frame. This can happen when the AF engine or the AEW engine finishes processing the frame much earlier than the other engine. In this case, the ISR does have enough time to clear the interrupt flag for the first interrupt by the time the second interrupt occurs.

The outcome depends on the difference in location of the last paxel/window in the frame (determines when processing is finished), the frequency of the relative clocks in the system, the occurrence and triggering of other interrupts in the system, and the latencies of the context switching and ISR execution.

The VISS_RAWFE_H3A_PCR[15] BUSYAF and/or VISS_RAWFE_H3A_PCR[18] BUSYAEAWB status bits are set when the start of frame occurs (if the VISS_RAWFE_H3A_PCR[0] AF_EN and/or VISS_RAWFE_H3A_PCR[16] AEW_EN bits are 1 at that time). They are automatically reset to 0 at the end of processing a frame. The VISS_RAWFE_H3A_PCR[15] BUSYAF and/or VISS_RAWFE_H3A_PCR[18] BUSYAEAWB status bits may be polled to determine the end of frame status.

Copyright © 2024 Texas Instruments Incorporated

6.7.3.2.4.6.9 RAWFE H3A Interface Mux

The H3A logic also implements an LUT or Shift /clip block. The LUT logic is same as the decompanding LUT in the previous section(s) with some minor modifications. The input bit width to the LUT cannot exceed 16 and the LUT entry size is only 10 bits, since the H3A logic works on 10 bit data.

The logic also implements a shift and clipU10 functionality for converting data from up to 24 bits down to 10. The shift parameter has to be programmed to implement the desired shift value. The shift register programming must ensure that the output of the shift is not greater than 16bits when the LUT is enabled and it should not be greater than 10 bits when the LUT is disabled.

6.7.3.2.4.6.10 RAWFE H3A interface to LSE

The H3A interface to LSE is VBUSP. It uses channel id to indicate data or control information. Please refer to LSE functional spec on control information mapping to channel id bits.

6.7.3.2.4.6.11 RAWFE H3A Erratas

The following hardware restrictions apply and need to be taken care of by software:

1. When AF is enabled software can not use the first or last pixels in the window. Start at position 2 end at line -2.
2. When AEW is enabled software can not use the first pixel in a window. Start at position 2.
3. (PAXW+1) % (AFINCH+1) != 1 when VF enabled (PAXW and AFINCH are raw MMR parameters)

6.7.3.2.5 RAWFE Programmer's Guide

6.7.3.2.5.1 RAWFE Core programming details

There are multiple use cases that can be enabled with this module, however broadly the use case are broken in to two categories, one for Companded Data sensors and the other for Separate Exposures sensors. The following fields should be programmed correctly for the IP to function

- PWL Core
- Set slopes and thresholds as per sensor settings only for Companded sensors

- Decompanding LUT
- This LUT is used for global tone map of decompounded data from 24 bits to 16 bits. (This is typically not used for separate exposures sensor)

- Merge Logic
- Program the merge block based on the exposure rations of the incoming data.
- This block is typically not used for companed sensors.
- LUT after merge should be programmed to tone map from 20 bits down to 16 (Separate exposure sensors only)

- DPC/ LSC
- Set as per tuning parameters for both companded/separate exposures sensors

- H3A LUT and settings
- Use LUT if H3A input comes after LSC. Program LUT to apply inverse global curve.
- Use shift if data is coming from exposures directly.

6.7.3.2.5.2 RAWFE HTS programming details

See *HWA Thread Scheduler* and Section 6.7.2, *VPAC Subsystem* for details.

6.7.3.2.5.3 RAWFE Data transfer programming details

See , *UDMA* for details of programming for PARAM Space.

6.7.3.2.5.4 RAWFE Initialization Sequence

See Section 6.7.2, *VPAC Subsystem* for details.

6.7.3.2.5.5 RAWFE Real-time Operating Requirements

See Section 6.7.2, *VPAC Subsystem* for details.

Copyright © 2024 Texas Instruments Incorporated

6.7.3.2.5.6 RAWFE Power up/down Sequence

See Section 6.7.2, *VPAC Subsystem* for details.

6.7.3.3 Chromatic Aberration Correction (CAC) Module

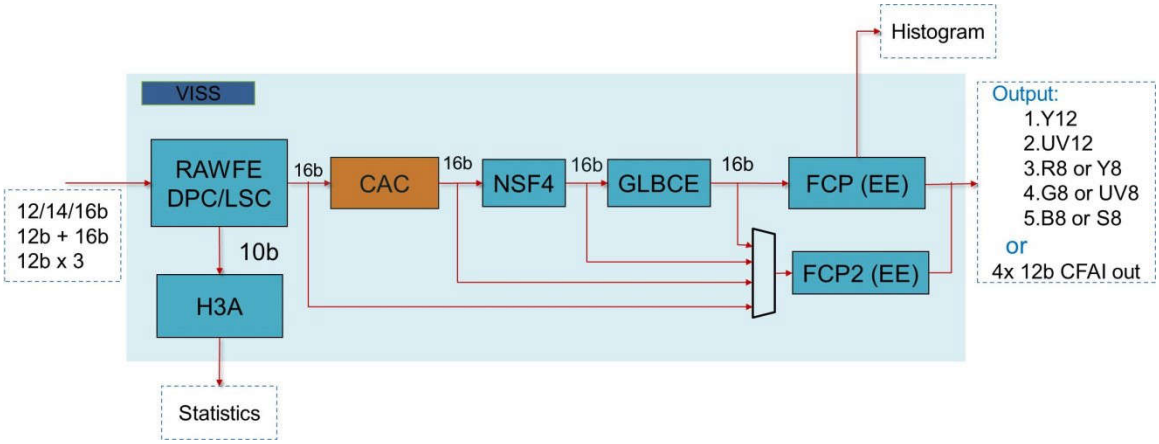
6.7.3.3.1 Overview and Feature List

6.7.3.3.1.1 Features Supported

- Chromatic Aberration Correction (CAC) for Red and Green color in Bayer color formats
- Generic mesh based model for CAC
- LUT to store programmable differential correction with 1/8th pixel precision
- Programmable Block size with each block having 2 LUT entries covering entire frame
  - Intermediate correction values are calculated using bilinear interpolation
- Bi-cubic pixel interpolation on corrected back mapped pixel
- Supports Horizontal Correction in the range of -8.0 to +8.0 and Vertical Correction in the range of -4.0 to +4.0
- Performance: up to 1 Pix/cycle (excluding blanking)
- Streaming based processing architecture targeted to integrate with VISS
- Support 16-bit pixel
- Logic support frame dimension of 8Kx8K. Line memory support is limited to 4K line width

6.7.3.3.2 Functional Description

6.7.3.3.2.1 CAC Integration in VISS



The VISS consist of following hardware accelerators:

- RFE (RAW-FE): The RAW-FRONT HW Block does RAW pixel (e.g. Bayer, RCCC, RGBW etc) processing on captured image data from sensor and pass-on to CAC, NSF4V, GLBCE block, and then to Flexible ColorProc HW block for demosaicing and color conversion.
- CAC (Chromatic Aberration Correction): Captured in this spec.
- NSF4V: Spatial Noise filter supporting generic 2x2 pixel format with 16-bit pixel size. NSF4V IP can be bypassed in the applications where visual enhancement is not desired.
- GLBCE: Iridix7 Ip of Apical is used for dynamic range control within image for visual quality. Iridix Ip is integrated within GLBCE module to adapt its interface with rest of image pipe and integrate its configuration registers. If contrast enhancement on input image is required for visual quality RFE/CAC/NSF4V output is processed by GLBCE block. Otherwise GLBCE is bypassed to FCP.
- FCP (FLEXible COLOR PROCessing): The FLEX-COLOR-PROC HW receives data from GLBCE and does demosaincing and color conversion. The output of COLOR-PROC is sent to VPAC shared memory to be written into DDR for rest of vision processing by programmable processors (e.g. DSP or ARM) or Vision HW Blocks (e.g. VPAC and DMPAC).

Copyright © 2024 Texas Instruments Incorporated

6.7.3.3.2.2 Introduction

Chromatic aberration, also known as “color fringing” or “purple fringing”, is a common optical problem that occurs in low quality lens when a lens is either unable to bring all wavelengths of color to the same focal plane (***Longitudinal*** *Chromatic aberration*), and/or when wavelengths of color are focused at different positions in the focal plane (*Lateral Chromatic aberration)* .

Figure 6-79 shows one example of Chromatic Aberration referred as *Lateral Chromatic Aberration* (LCA). In this case, all colors are in focus in the same plane, but the foci are not placed along the optical axis. CAC Hardware accelerator performs *Lateral Chromatic aberration* correction is as part of VISS Pipe line.

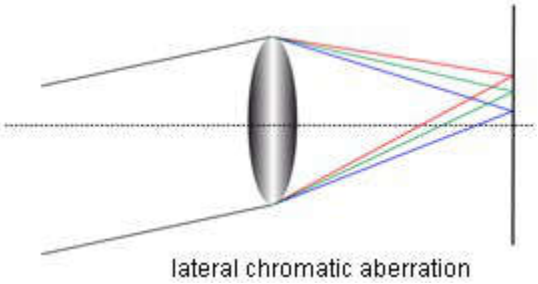


Figure 6-79. Lateral Chromatic Aberration

6.7.3.3.2.3 Functional Operation

CAC correction requires remapping locations of Red and Blue pixels to align with Green pixel. Green pixel is taken as reference and assumed no correction is needed on Green pixel. Legacy block based correction wouldn’t fit into the streaming VISS pipe line and incurs additional bandwidth and latency. CAC HWA is implemented using line based back mapping architecture. Correction involves back-mapping each output pixel to pixel position in the source image, and thus the corrected image is fully populated. As the back mapped pixel locations mostly fall onto fractional coordinates, correction involves bi-cubic interpolation among the nearest available pixels.

As shown in Figure 6-80, CAC consists of Timing control module, 8KB LUT with mesh correction offset table, Mesh LUT bi-linear interpolation block, Pixel interface module, pixel bi-cubic interpolation block and 10 pixel line memories.

Copyright © 2024 Texas Instruments Incorporated

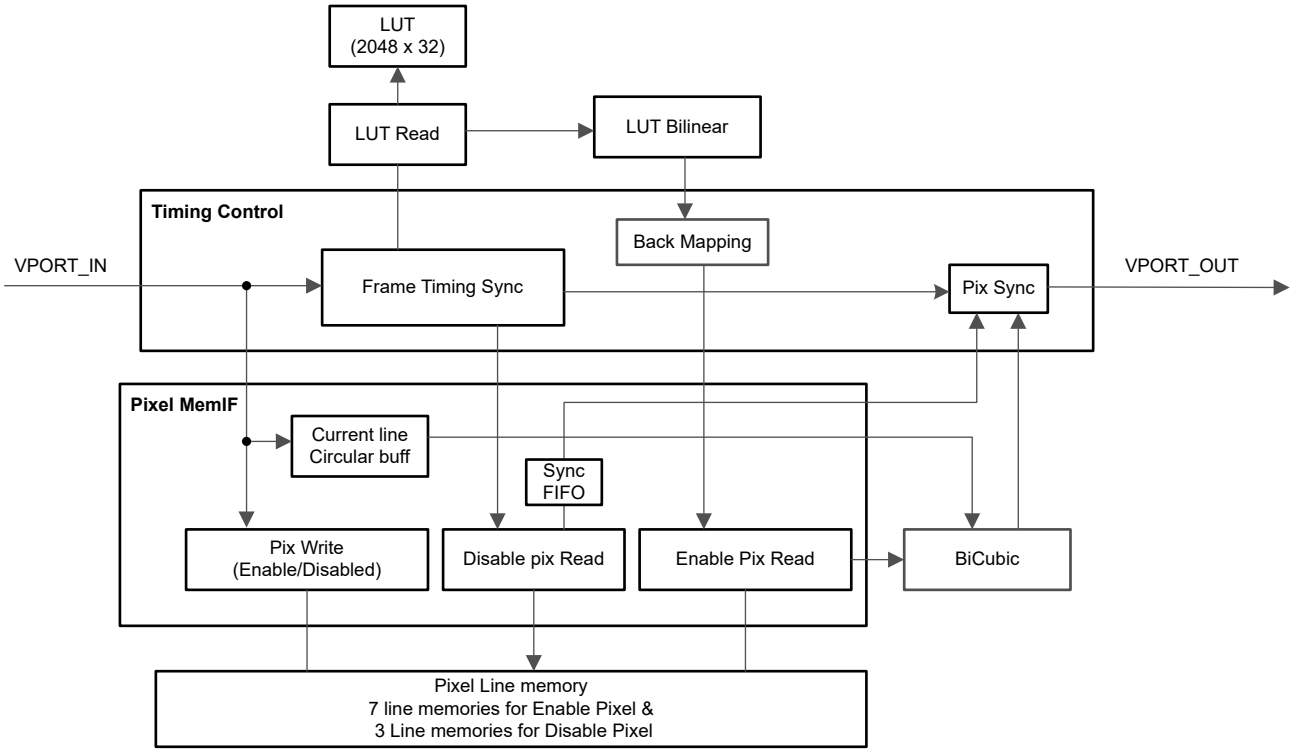


Figure 6-80. CAC Functional Block Diagram

Timing Control module handles all the timing synchronization aspects of CAC. It generates the Output frame co-ordinates, initiates the Mesh LUT operation on receiving 7th input valid line. It also handles timing synchronization between all the blocks in the design.

6.7.3.3.2.3.1 CAC Back Mapping

Offset table defines a (□x, □y) vector for a regular grid of output points. Each grid will define 2 offset vectors one each for EVEN and ODD line(s), defining how far output pixel needs to be moved in either direction. Final pixel location in input source image is used to compute the output pixel. Offset table can have precise definition and can capture rapidly changing offset tables. However it will require a large amount of memory. Since most offset tables are not expected to change rapidly in a small spatial region, CAC uses a subsampled offset table. Grid is down sampled with down-sampling factor in the range of 8 to 128 with increments of 4 in both horizontal and vertical directions. Subsampling factor is set in the register; BLOCKSZ.SIZE This enables reducing the amount of memory required to describe the offset vectors, but requires hardware to interpolate the missing offset vectors. CAC supports bilinear interpolation to interpolate missing offset vectors.

The mapping procedure is described by the following series of equations. Given an output pixel at location (x_0,y_0), we compute the input image pixel location (x_1,y_1).

$$ind_x = \frac{X_o}{Blk_{size}};\ \ f_x = MOD(X_o, Blk_{size})$$
$$ind_y = \frac{Y_o}{Blk_{size}};\ \ f_y = MOD(Y_o, Blk_{size})$$

Clip new index to Offset grid boundaries.

Copyright © 2024 Texas Instruments Incorporated

$$i_x = CLIP(ind_x, Grid_{HCnt})$$
$$i_y = CLIP(ind_y, Grid_{VCnt})$$

$$\Delta x_p = \frac{\left((blk_{size} - f_y) \cdot T_{x0} + f_y \cdot T_{x1} + \frac{square(blk_{size})}{2}\right)}{squae(block_{size})}$$

$$\Delta y_p = \frac{\left((blk_{size} - f_y) \cdot T_{y0} + f_y \cdot T_{y1} + \frac{sqaure(blk_{size})}{2}\right)}{square(blk_{size})}$$

Clip to Horizontally to -8.0 to +8.0 pixel range (-64 to +64 with Q3 precision).

Clip to Vertically to -4.0 to +4.0 pixel range (-32 to +32 with Q3 precision).

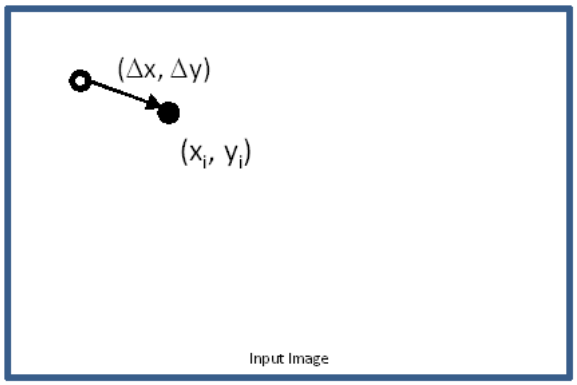


Figure 6-81. Back Mapping Procedure Using Offset Table

The input coordinate (x1,y1) refers to final input frame co-ordinates.

6.7.3.3.2.3.1.1 Offset Table Storage Format

The size of the offset table is dependent on the output frame size. The size must be:

$$Table\ Width = CEIL\left(\frac{Frame\ Width}{Blk_{size}}\right) + 1$$

$$Table\ Height = CEIL\left(\frac{Frame\ Height}{Blk_{size}}\right) + 1.$$

The Offset table should be stored with □x and □y offsets interleaved and stored in raster order. Each offset is S8Q3, so a table entry is 32-bits, including both □x and □y for both Even and Odd line. Each line captures offset for one color either Red or Green.

Copyright © 2024 Texas Instruments Incorporated

6.7.3.3.2.3.2 Pixel Interpolation

As the coordinates calculated by the back mapping function are not integer values in most cases, b-cubic interpolation is applied to the back mapped pixels. Interpolation is done on the pixel of same color (either Red or Blue).

Output pixel is interpolated from the 16 pixels in the 4x4 grid around the back mapped location, as shown in Figure 6-82. *Ri* and *Bi* indicate integer co-ordinates of back mapped pixels. Bi-cubic interpolation is used first along the horizontal direction, then the vertical direction.

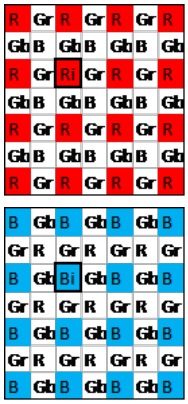


Figure 6-82. Bi-Cubic Interpolation Input Data

6.7.3.3.2.3.3 Bi-cubic Coefficients

The following coefficients are used for bi-cubic operation. Though back-mapped pixel has Q3 precision, there are 16 fractional phases due to Bayer pattern. The filter coefficients should be in S12Q10 format (signed 12-bit with 10 bits of fraction).

Phase	Coefficients (S12Q10)			
0/16	0	1024	0	0
1/16	-28	1014	40	-2
2/16	-49	987	93	-7
3/16	-63	944	158	-15
4/16	-72	888	232	-24
5/16	-76	821	313	-34
6/16	-75	745	399	-45
7/16	-71	663	487	-55
8/16	-64	576	576	-64
9/16	-55	487	663	-71
10/16	-45	399	745	-75
11/16	-34	313	821	-76
12/16	-24	232	888	-72
13/16	-15	158	944	-63
14/16	-7	93	987	-49
15/16	-2	40	1014	-28

Figure 6-83. Bi-cubic Coefficients

6.7.3.3.2.4 Interrupt Conditions

6.7.3.3.2.4.1 Interrupts

Name	Width	Type	Polarity	Category	Clock	Reset	Description
cac_lut_cfg_err	1	pulse	high	func	func_clk	srstn_mod_g_rst_n	CAC cac_lut_cfg_err event

Copyright © 2024 Texas Instruments Incorporated

A ***cac_lut_cfg_err*** interrupt is generated when configuration access happens on LUT memory while CAC is operating on a valid frame. A valid frame is defined as Input Frame Start (VS_IN) to output frame end (VE_OUT).

6.7.3.3.2.4.2 Debug Events

Table 6-146. Debug Events

Event	Description
cac_in_vs_event	Frame Start Input. Optionally can be enabled by setting DBG_CTL.DBG_EN && DBG_CTL.SOF_EN. Status can be observed in DBG_STAT.SOF
cac_in_hs_event	Frame Start Input. Optionally can be enabled by setting DBG_CTL.DBG_EN && DBG_CTL.SOL_EN. Status can be observed in DBG_STAT.SOL.
cac_out_ve_event	Frame End Output. Optionally can be enabled by setting DBG_CTL.DBG_EN && DBG_CTL.EOF_EN. Status can be observed in DBG_STAT.EOF.
cac_out_he_event	Line End Output. Optionally can be enabled by setting DBG_CTL.DBG_EN && DBG_CTL.EOL_EN. Status can be observed in DBG_STAT.EOL.

Copyright © 2024 Texas Instruments Incorporated

6.7.3.4 VISS Spatial Noise Filter (NSF4V)

6.7.3.4.1 NSF4V Introduction

6.7.3.4.1.1 NSF4V Features

For a list of features supported by the NSF4V module, see Section *VPAC Features*.

Some additional NSF4V parameters include:

- Input image format: any 2x2 raw color pattern
- Frame Width: limited to 4096
- Throughput:
  - 1 pixel in/out per clock
- Support raw input image in generic 2x2 color pattern format including Bayer pattern
- Support 16-bit raw input image with or without companding
- Automatic image border mirroring
- Signal level detection with flexible cross-color-channel or independent averaging
- Spatially adaptive noise level threshold
- Configurable soft or hard thresholding
- Support elliptical lens shading gain adjustment of adaptive noise threshold with 2 sets of independent settings
- Dynamic White balance gainadoptation (independent of Noise Filter operation)
- Histogram on Raw Bayer data (independent of Noise Filter operation)

6.7.3.4.2 NSF4V Overview

The advanced noise filter NSF4v decomposes the input image into low-frequency, mid-frequency, and high-frequency bands first and then filter out the noise in frequency domain. NSF4v can adapt its noise filtering strength to local image intensity according to the user programmable noise threshold registers. NSF4 is also able to adjust its strength according to the amount of previously applied lens shading correction gains with a user programmable approximate radial model.

NSF4V includes the function to generate a histogram on the raw input data and dynamic white balance at the output independent of NSF4V core noise filtering function.

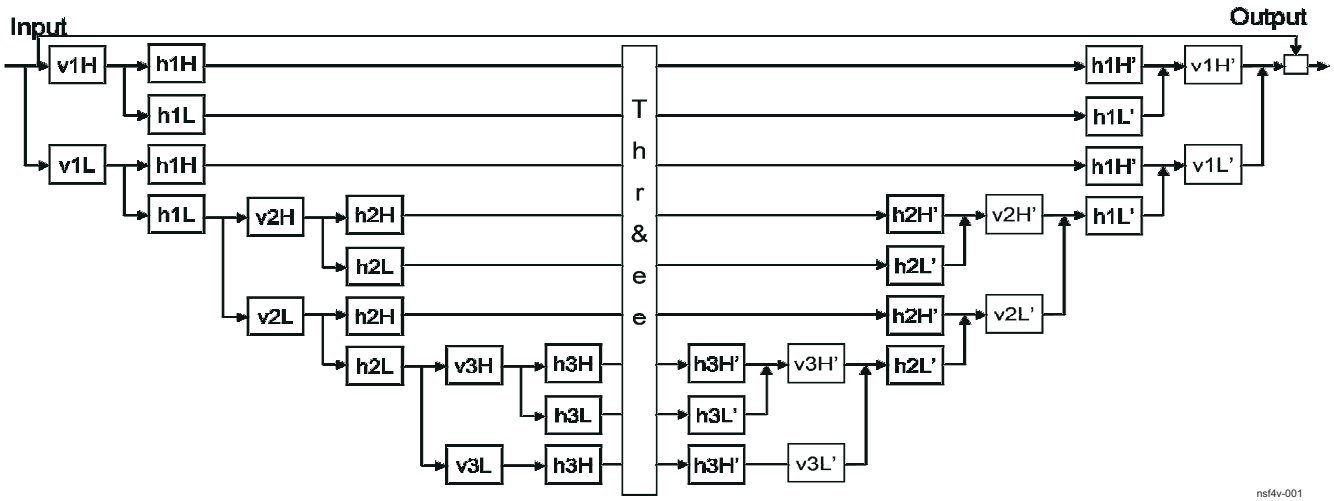


Figure 6-84. NSF4V Noise Filter Algorithm

6.7.3.4.2.1 Decomposition Kernel Representation

With combination of just level 1, levels 1+2, and levels 1+2+3, horizontally and vertically, the result feature detection filter kernels are shown in Figure 6-85. Note that all coefficients are powers of 2, and therefore do not require multiplication.

Copyright © 2024 Texas Instruments Incorporated

$$F1 = \begin{matrix} 1 & -1 \\ 1 & -1 \end{matrix} \; /4$$

$$F2 = \begin{matrix} 1 & 1 \\ -1 & -1 \end{matrix} \; /4$$

$$F3 = \begin{matrix} 1 & -1 \\ -1 & 1 \end{matrix} \; /4$$

$$F4 = \begin{matrix} 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 \end{matrix} \; /16$$

$$F5 = \begin{matrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 \end{matrix} \; /16$$

$$F6 = \begin{matrix} 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & 1 \\ -1 & -1 & 1 & 1 \end{matrix} \; /16$$

$$F7 = \begin{matrix} 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \end{matrix} \; /64$$

$$F8 = \begin{matrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \end{matrix} \; /64$$

$$F9 = \begin{matrix} 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \end{matrix} \; /64$$

nsf4v-002

Figure 6-85. NSF4V Kernel Representation for Decomposition Filters

G1...9 are the counter-part reconstruction kernels. Each of G1...9 is a row-wise and column-wise mirror of the counterpart F function.

The kernel representation in Figure 6-86 illustrates a filter alignment.

Copyright © 2024 Texas Instruments Incorporated

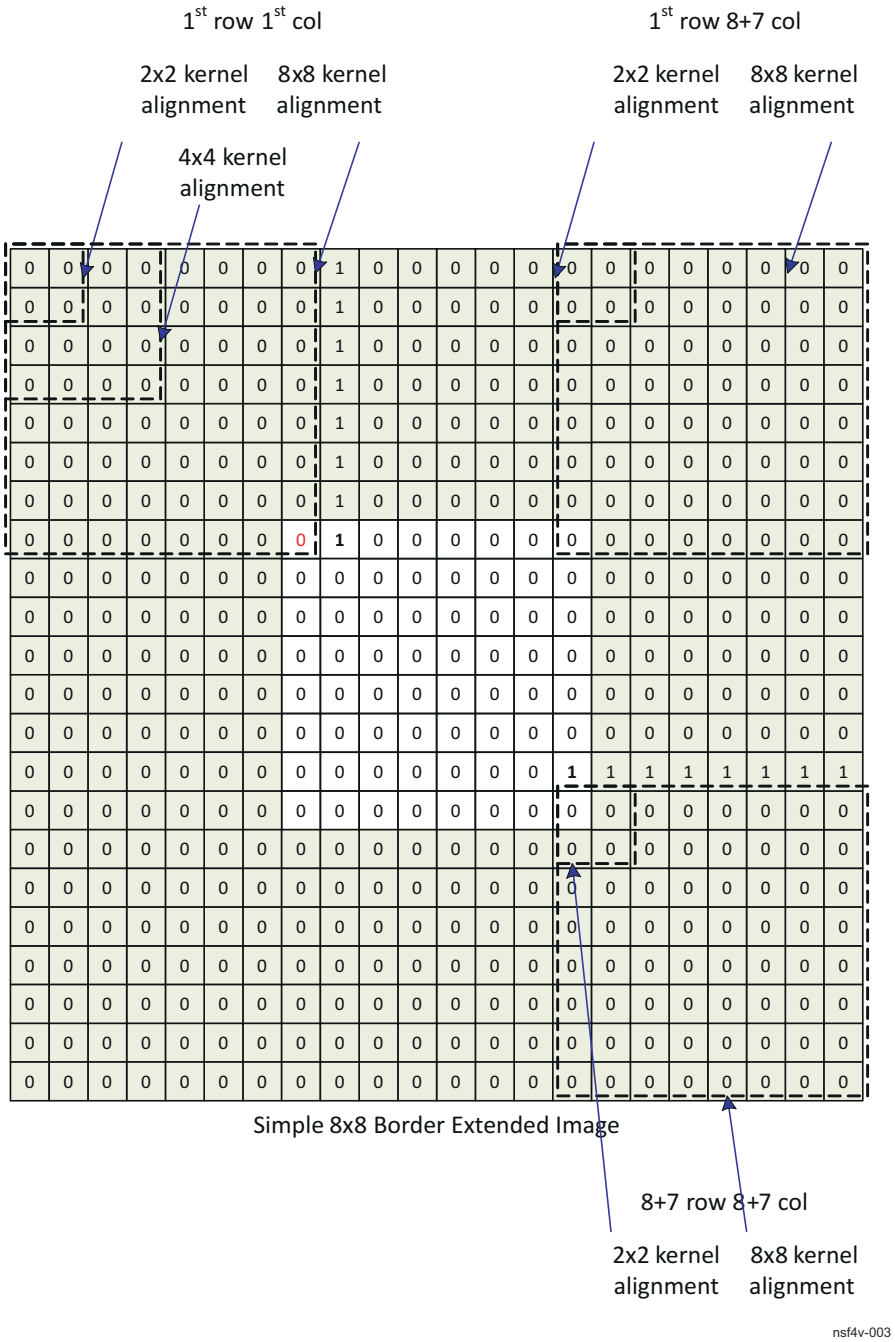


Figure 6-86. NSF4V Kernel Representation Filter Alignment

Figure 6-86 is showing an academic 8x8 pixel image frame which has been border extended on all 4 edges. Superposed onto this border extended frame (shown in dotted lines) is the kernel alignment for all 3 different kernel sizes for each of the three levels:

- Level 1: 2x2 kernel
- Level 2: 4x4 kernel
- Level 3: 8x8 kernel

For each of the 9 different kernels, a matrix of (width + 7) x (height +7) decomposition values result. All of these result matrixes are exactly aligned.

Copyright © 2024 Texas Instruments Incorporated

0	0	0	0	0	0	0	**-2**	**2**	0	0	0	0	0	0
0	0	0	0	0	0	0	**-2**	**2**	0	0	0	0	0	0
0	0	0	0	0	0	0	**-2**	**2**	0	0	0	0	0	0
0	0	0	0	0	0	0	**-2**	**2**	0	0	0	0	0	0
0	0	0	0	0	0	0	**-2**	**2**	0	0	0	0	0	0
0	0	0	0	0	0	0	**-2**	**2**	0	0	0	0	0	0
0	0	0	0	0	0	0	**-2**	**2**	0	0	0	0	0	0
0	0	0	0	0	0	0	**-1**	**1**	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0
0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

nsf4v-004

Figure 6-87. NSF4V Example F1 Kernel Result

The kernel alignment is top-left aligned for the first pixel. Which translates in the recursive approach as a X-Y shift. In order to be mathematically aligned in the Recursive approach, the Level 1 output needs to be shifted 6 positions in both the X and Y direction relative to the Level 3 output. Additionally, the Level 2 output needs to be shifted 4 positions in both the X and Y direction relative to the Level 3 output. These shifts are critical such that Level 1, 2, and 3 values are cycle aligned when presented to the thresholding circuit.

One can reach the same conclusion by looking at the figure and thinking about which line delay or which horizontal pipelined pixel position is used in order for the kernels to output their results in the same pipeline cycle.

6.7.3.4.3 NSF4V Lens Shading Correction Compensation

The Lens Shading Correction (LSC) is implemented prior to NSF. This is not an optimal order of operation and therefore an adjustment factor is implemented to compensate for the order of operations. The LSCC (Lens Shading Correction Compensation) first approximates the gain of LSC and calculates both that gain and it's inverse. Within the T_n calculation, the inverse gain is applied to the LL2 average, then the T_n is calculated. Last, the result is multiplied by the gain. In this way, the LSC is approximately reversed temporarily in order that T_n is calculated properly.

LSCC calculates the radius (distance) from the center of the image, then uses a programmable curve to look up the gain value.

While calculating the radius, the center on the frame is programmable. Also a correction gain (kh and kv) separately for X and Y are programmable which allows for elliptical shaped lenses. Because the SQRT is such an expensive function, the resulting radius is used in the SQR form and as such has a very large dynamic range. A programmable shift value effectively selects which portion of this dynamic range is to use for the curve lookup.

The curve lookup is a 16 segment curve. The SQR(Radius) is lookup on the X axis and the resulting gain is supplied on the Y axis.

Copyright © 2024 Texas Instruments Incorporated

6.7.3.4.4 NSF4V Noise Threshold Adaptation to Local Image Intensity

Figure 6-88 shows the local image intensity to noise threshold T_n mapping via the 12-segment interpolated lookup. Twelve sets of (X, Y, S), S = slope, are provided per curve, one set per color plane, with the first X coordinate fixed at zero.

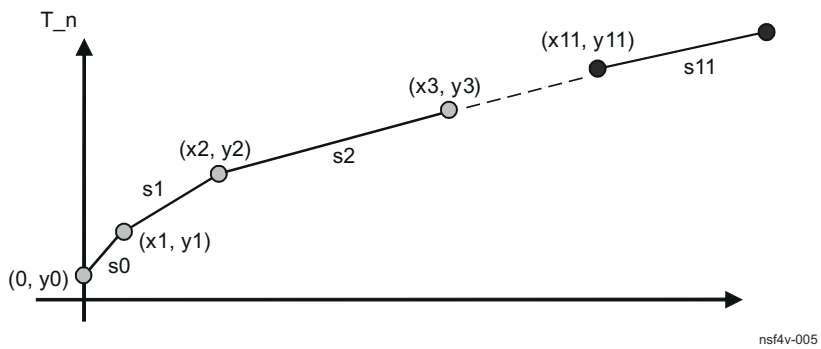


Figure 6-88. NSF4V Local Image Intensity to Noise Threshold Piece-wise Linear Function

The slopes should be implemented with signed value, as some image sensors might require higher noise threshold at the low intensity range than high intensity range.

6.7.3.4.5 Delta Features

- Dynamic White Balance
  - Applies white gain calculated based on intensity of neighboring 6 pixels horizontally
  - Intensity is calculated on even pixel but used for that pixel and next odd pixel
  - Gain is calculated from intensity using 8-segment PWL implementation
  - Each 2x2 color component will have independent PWL parameters
- Raw Bayer domain Histogram
  - Convert 16-bit tone mapped to 12-bit log domain using LUT
  - Phase select to enable histogram on up to 2 color channels in 2x2 pattern. If 2 channels are enabled, then they need to be in different lines.
  - Total 128 bins of 22-bits each (step size of 32)
  - Capture the histogram of up to 8 independent programmable non-overlapping regions
  - Histogram output is mapped to MMR space. Data needs to be read once Output frame is completed

Copyright © 2024 Texas Instruments Incorporated

6.7.3.5 VISS Global/Local Brightness and Contrast Enhancement (GLBCE) Module

6.7.3.5.1 GLBCE Overview

The Global Local Brightness Contrast Enhancement module performs dynamic range compression or tone mapping of an input image based on a model of the human visual system. It is used to produce most natural images under a wide range of capture conditions, typically by revealing shadow detail which would otherwise be under-exposed in high contrast situations.

6.7.3.5.2 GLBCE Interface

Data flow control of GLBCE module is done by PCLK. GLBCE processes the image cycle by cycle, and output the data after certain cycles of latency.

6.7.3.5.3 GLBCE Core

The GLBCE Core is an iridix® core which performs "dynamic range compression" or "tone mapping" on an input image based on a model of the human visual system. It is used in a camera pipeline to produce the most natural images under a wide range of capture conditions, typically by revealing shadow detail which would otherwise be under-exposed in high contrast situations. The purpose of the transform is to map the image content from an input source such that it remains fully visible on an output display without loss of content. Note that iridix is specifically designed to preserve color, sharpness and boost contrast of the source image.

The iridix core is based on space-variant or pixel-by-pixel processing algorithms which construct a family of transforms based on the analysis of image content. The algorithms inside iridix are adaptive, meaning that a single set of parameters can be set to process any source image or any video sequence under different lighting conditions.

Iridix generates effective tone curves by analyzing the regions surrounding each pixel. The tone curves generated by iridix are based on an asymmetry curve which is then shaped and controlled by a number of parameters that control the gain limitation weighting towards shadows, mid tones and highlights.

6.7.3.5.3.1 GLBCE Core Key Parameters

Table 6-147 shows the key parameters within the iridix core. It is recommended that all parameters be set statically to their recommended values other than the iridix strength parameter, which should be controlled dynamically with scene content. The calculation of iridix strength is explained in Section 6.7.3.5.3.2.

Table 6-147. GLBCE Core Key Parameters

Parameter	Description
iridix strength	This register sets iridix processing strength.
	0x00: Video data will not be processed at all, and will pass to the output unchanged.
	0xFF: Maximum processing strength.
Variance space	This register affects the sensitivity of the transform to different areas of the image, and can be increased in order to emphasize small regions (e.g. faces). If this parameter is set to zero, the sensitivity to small areas is maximized (i.e. the transform becomes more local). When this parameter is set to 0xF, the transform deemphasizes small local details, and the transform becomes more global
Variance intensity	For a high-contrast image with shadows and highlights, a small value will cause iridix to adjust shadows and highlights almost independently. A large variance causes iridix to become progressively more global, meaning that the presence of highlights affects the processing of shadows and vice-versa.
Slope min limit	This register is used to restrict the slope of the tone curve generated by iridix. When this parameter is set to 0x00, the tone curve slope generated by iridix is not limited.
	When this value is set to 0xFF, iridix will not generate a tone curve with a slope less than 3/4. Intermediate values of slope limit are linear interpolation between minimum and maximum values.
	The recommended range of this parameter is 0-128
Slope max limit	This register is used to restrict the slope of the tone-curve generated by iridix. When this parameter is set to 0xFF, the tone curve slope generated by iridix is not limited.
	When this value is set to 0x00, iridix will not generate a tone-curve with slope greater than 16/15 (48°). Intermediate values of slope limit are linear interpolation between minimum and maximum values.
	The recommended range of this parameter is 0-160. For noisy images, lower values should be used.
Black level	The value stored in this register will be used as the zero level for iridix processing in all data channels. Data below this value will not be processed and remain unchanged.

Copyright © 2024 Texas Instruments Incorporated

Table 6-147. GLBCE Core Key Parameters (continued)

Parameter	Description
White level	The value stored in this register will be used as the white level for iridix processing in all data channels. Data above this value will not be processed and remain unchanged.
Asymmetry LUT	This look-up-table is 33 words, each of which is 16-bits. The asymmetry function is used to balance the iridix effect between the dark and bright regions of the image.
Forward perceptual LUT	The iridix core works in a perceptually uniform space and thus when linear data is presented to the core, this look-up-table must be used to map the linear data to the perceptual space so that iridix can perform tis perceptual functions.
Reverse perceptual LUT	The iridix core works in a perceptually uniform space and thus when linear data is required from the output of the core, this look-up-table must be used to map the data from the perceptual space back to linear space for further processing in the pipeline.

6.7.3.5.3.2 GLBCE Iridix Strength Calculation

This section describes how the dynamic parameter, iridix strength, is calculated.

Note

Dynamic range of a digital camera can be described as the ratio of maximum light intensity measurable (at pixel saturation), to minimum light intensity measurable (but above read-out noise). This is a very important concept to consider when computing iridix strength, because the gain applied by iridix is also proportional to the visible boost of noise in the image. The maximum gain applied by iridix should be determined by the dynamic range of the camera system. This is governed by the equation presented in this section.

With iridix strength is zero, all tone curves are linear for all images. As iridix strength is increased, the variation between curves in shadows, mid tones and highlights increases, resulting in increasing compression of global dynamic range.

Iridix strength is calculated as follows:

$$\text{irridix strength} = \text{Max}\left(0, \text{Min}\left(\frac{\text{IR_Gain_Target} - 1}{\text{IR_Gain}_{\text{Max}}-1}, \text{f(analog gain)}\right)\right)*255$$

glbce-843

Figure 6-89. GLBCE Iridix Strength Formula

Where:

- IR_Gain_Target is the desired iridix gain to be applied to the image. This is calculated from the image's histogram on a per-frame basis by the auto exposure algorithm (in RAWFE H3A module). The histogram is analysed to compute an exposure ratio for the desired global histogram shift. This computed exposure ratio is then used to feed a look-up-table which in turn generates the IR_Gain_Target parameter.
- IR_GainMax is the maximum iridix gain. If the fwd_percep_lut is disabled, then the IR_GainMax should be set to 64, and if the fwd_percep_lut is enabled, then the IR_Gainmax should be set to 16.
- f(analog gain) is a function that determines the maximum iridix strength in relation to noise. f(analog gain) is characterised as follows:

Copyright © 2024 Texas Instruments Incorporated

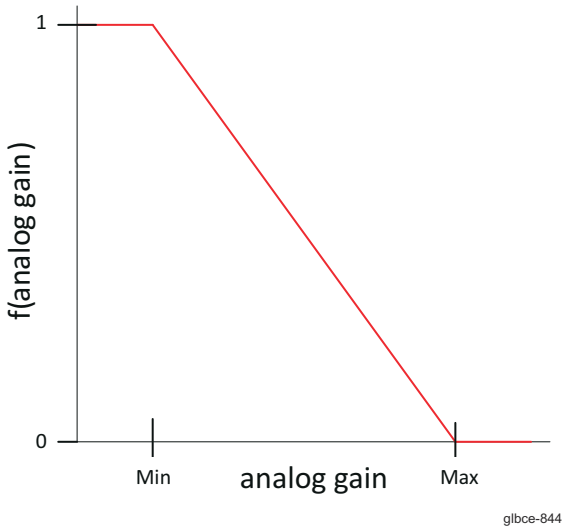


Figure 6-90. GLBCE Iridix f(Analog Gain) Function

f(analog gain) should only be used when the RAWFE is in linear mode. Otherwise, it should be set to 1

6.7.3.5.3.3 GLBCE Iridix Configuration Registers

6.7.3.5.3.3.1 GLBCE Iridix Frame Width

Frame Width is the number of pixels in an active line and is controlled from GLBCE_FRAME_WIDTH[15:0] VAL. Recommended value 2560 (decimal)

6.7.3.5.3.3.2 GLBCE Iridix Frame Height

Frame Height is the number of active lines in a frame and is controlled from GLBCE_FRAME_HEIGHT[15:0] VAL. Recommended value 1920 (decimal)

6.7.3.5.3.3.3 GLBCE Iridix Control 0

This is the main control register for the iridix core is GLBCE_CONTROL0 where GLBCE is enabled/disabled and the type of processing algorithm is selected.

6.7.3.5.3.3.4 GLBCE Iridix Control 1

This register is reserved for debugging purposes. Recommended value 6 (decimal)

6.7.3.5.3.3.5 GLBCE Iridix Strength

GLBCE_STRENGTH_IR[7:0] VAL register sets the processing strength of the iridix core. Recommended value 128 (decimal)

6.7.3.5.3.3.6 GLBCE Iridix Variance

The GLBCE_VARIANCE parameter affects the sensitivity of the transform to different areas of the image, and can be increased in order to emphasize small regions (e.g. faces). GLBCE_VARIANCE[7:4] VARIANCEINTENSITY recommended value 12 (decimal). GLBCE_VARIANCE[3:0] VARIANCESPACE recommended value 7 (decimal).

6.7.3.5.3.3.7 GLBCE Iridix Dither

When the number of color gradations of a display device is small (for example 6 bits per color), pixel dithering can make gradients look smother. Dithering of the least significant bits is applied and then these bits can be later truncated. The three least significant bits (D2, D1, D0) of this GLBCE_DITHER register are responsible for the strength of dithering. There are 4 possible levels of dithering. Recommended value 0 (decimal)

6.7.3.5.3.3.8 GLBCE Iridix Amplification Limit

The parameters dark amplification limit and bright amplification limit are used to restrict the luminance space in which iridix can adaptively generate tone curves for each

Copyright © 2024 Texas Instruments Incorporated

pixel. GLBCE_LIMIT_AMPL[7:4] BRIGHTAMPLIFICATIONLIMIT recommended value 0 (decimal). GLBCE_LIMIT_AMPL[3:0] DARKAMPLIFICATIONLIMIT recommended value 0 (decimal).

6.7.3.5.3.3.9 GLBCE Iridix Slope Min and Max

GLBCE_SLOPE_MIN[7:0] SLOPEMINLIMIT register is used to restrict the slope of the tone curve generated by iridix. Recommended value 64 (decimal).

GLBCE_SLOPE_MAX[7:0] SLOPEMAXLIMIT register is used to restrict the slope of the tone curve generated by iridix. Recommended value 72 (decimal).

6.7.3.5.3.3.10 GLBCE Iridix Black Level

The value stored in the Black Level register GLBCE_BLACK_LEVEL[15:0] VAL will be used as the zero level for iridix processing in all data channels. Data below the Black level will not be processed and remain unchanged. Recommended value 0 (decimal).

6.7.3.5.3.3.11 GLBCE Iridix White Level

The value stored in White Level register GLBCE_WHITE_LEVEL[15:0] VAL will be used as white level for iridix processing in all data channels. Data above White level will not be processed and stay unchanged. Recommended value 65535 (decimal).

6.7.3.5.3.3.12 GLBCE Iridix Asymmetry Function Look-up-table

The Asymmetry function is used to balance the iridix effect between the dark and bright regions of the image. The Asymmetry Function Lookup Table geometry is 33 words, each of which is 16-bits wide.
GLBCE_LUT_FI_00
GLBCE_LUT_FI_01
... repeat until...
GLBCE_LUT_FI_32
Recommended values (decimal):
0:5377:10218:14600:18585:22224:25561:28631:31466:34092:36530:38801:40921:42904
:44764:46511:48156:49706:51171:52557:53870:55116:56299:57425:58498:59520:60497
:61429:62322:63176:63995:64781:65535

The C-code for generating the Asymmetry LUT is shown below:

```
#include <math.h>
int AsymmetryTable[33];
void GenerateAsymmetry(int Asymmetry, int SecondPole)
{
    double x  = (double(Asymmetry)+1)/257 * 2 - 1;
    int ai = (int)floor(0.5 + 255 * (1- 1/(1000*x*x*x) + x - ((x >= 0)*2)));
    double as = fabs(double(ai) / 255);
    double dp = double(SecondPole) / 255;
    int ii;
    for (ii = 0 ; ii < 33 ; ++ii)
    {
            if(ai >= 0)
            {
                    x = double(ii) / 32;
            }
            else
            {
                    x = double(32 - ii) / 32;
            }
            int y = (int)floor((dp+(1-dp)*pow((fabs(1-dp-x)/dp), 3)) * (x*(as+1)/(as+x) ) *
                 65535.0 + 0.5);
            y = y < 0 ? 0 : y > 65535 ? 65535 : y;

            if (ai < 0)
            {
                    y = 65535 - y;
            }
            AsymmetryTable[ii] = y;
        }
}
```


Copyright © 2024 Texas Instruments Incorporated

6.7.3.5.3.3.13 GLBCE Iridix Forward and Reverse Perceptual Functions Look-up-tables

The iridix core works in a perceptually uniform space and thus when linear data is presented to the core it needs to be processed with special perceptual functions. These are implemented as two perceptual LUTs namely: Forward Perceptual LUT and Reverse Perceptual LUT. Each LUT is made up of sixty-five 32-bit words.
GLBCE_REV_PERCEPT_LUT_00
GLBCE_REV_PERCEPT_LUT_01
... repeat untill ...
GLBCE_REV_PERCEPT_LUT_64

Recommended values (in decimal) for Reverse LUT:
0:228:455:683:910:1138:1369:1628:1912:2221:2556:2916:3304:3717:4158:4626:5122:
5645:6197:6777:7386:8024:8691:9387:10113:10869:11654:12471:13317:14194:15103
:16042:17012:18014:19048:20113:21210:22340:23501:24696:25922:27182:28475
:29800:31159:32552:33977:35437:36930:38458:40019:41615:43245:44910:46609
:48343:50112:51916:53755:55630:57539:59485:61466:63482:65535

GLBCE_FWD_PERCEPT_LUT_00
GLBCE_FWD_PERCEPT_LUT_01
... repeat untill ...
GLBCE_FWD_PERCEPT_LUT_64

Recommended values (in decimal) for Forward LUT:
0:4622:8653:11684:14195:16380:18335:20118:21766:23304:24751:26119:27422:28665
:29857:31003:32108:33176:34209:35211:36185:37132:38055:38955:39834:40693:41533
:42355:43161:43951:44727:45488:46236:46971:47694:48405:49106:49795:50475:51145
:51805:52456:53099:53733:54360:54978:55589:56193:56789:57379:57963:58539:59110
:59675:60234:60787:61335:61877:62414:62946:63473:63996:64513:65026:65535

6.7.3.5.3.3.14 GLBCE Iridix WDR Look-up-table

If an image sensor with built-in WDR functionality is used, and that sensor outputs data in a companded format, this LUT can be used to reverse the companding function in the sensor and feed linear data into the iridix core. This LUT has 257 entries.

GLBCE_WDR_GAMMA_LUT_00
GLBCE_WDR_GAMMA_LUT_01
... repeat untill ...
GLBCE_WDR_GAMMA_LUT_256

Recommended values (in decimal):
0:5887:10263:13117:15287:17058:18564:19881:21055:22117:23088:23984:24818:25597:26330
:27021:27677:28301:28896:29466:30011:30536:31040:31527:31997:32451:32891:33318:33733
:34135:34527:34908:35280:35643:35996:36342:36680:37010:37333:37650:37960:38264:38562
:38854:39141:39423:39700:39972:40239:40502:40761:41016:41266:41513:41756:41996:42232
:42465:42694:42921:43144:43364:43582:43797:44009:44218:44425:44629:44831:45030:45228
:45423:45615:45806:45995:46181:46366:46548:46729:46908:47085:47260:47434:47606:47776
:47945:48112:48277:48441:48604:48765:48924:49083:49239:49395:49549:49702:49854:50004
:50153:50301:50448:50593:50738:50881:51024:51165:51305:51444:51582:51719:51855:51990
:52124:52257:52389:52521:52651:52781:52909:53037:53164:53290:53415:53540:53663:53786
:53908:54030:54150:54270:54389:54507:54625:54742:54858:54974:55089:55203:55316:55429
:55541:55653:55764:55874:55984:56093:56202:56310:56417:56524:56630:56736:56841:56945
:57049:57153:57256:57358:57460:57561:57662:57763:57863:57962:58061:58159:58257:58355
:58452:58548:58645:58740:58835:58930:59025:59118:59212:59305:59398:59490:59582:59673
:59764:59855:59945:60035:60124:60213:60302:60390:60478:60566:60653:60740:60827:60913
:60999:61084:61169:61254:61338:61422:61506:61589:61673:61755:61838:61920:62002:62083
:62164:62245:62326:62406:62486:62566:62645:62724:62803:62882:62960:63038:63115:63193
:63270:63347:63423:63500:63576:63651:63727:63802:63877:63952:64026:64101:64175:64248
:64322:64395:64468:64541:64613:64685:64757:64829:64901:64972:65043:65114:65185:65255
:65325:65395:65465:65535

Copyright © 2024 Texas Instruments Incorporated

6.7.3.5.4 GLBCE Embedded Memory

GLBCE needs 2 lines of line memories and 1 set of cache memories.

Table 6-148. GLBCE Memories

Memory Type	Qty	Size	Clock	ECC Supported	Description
DELAY LINE	1	2122 x 64 (GLBCE_LINE_SIZE /2+10)x64	FCLK	No	GLBCE line memory
STATMEM	8 banks	16 bit x 1024	FCLK	Yes	GLBCE Cache memory (computes and stores the statistics from the frame)

6.7.3.5.5 GLBCE General Processing

GLBCE block needs appropriate statistics information to be stored to its cache memory in prior to processing incoming frame. Usually, this takes a two-pass process.

1. GLBCE is configured for input frame-A
2. Frame-A is given to GLBCE
3. GLBCE processes the input frame-A
4. GLBCE generates statistic (statistics-A) data based on frame-A in vertical blanking period
5. (Optional process) statistics-A is read from cache memory and moved to SDRAM
6. GLBCE is configured for input frame-B
7. Frame-B is given to GLBCE
8. GLBCE processes the input frame-B with the statistics-A generated from frame-A
9. GLBCE generates statistics-B based on frame-B in vertical blanking period
10. (Optional process) statistics-B is read from cache memory and moved to SDRAM

Alternatively, GLBCE can process in the following manner

1. Pre-calculated Statistics-C is copied to GLBCE cache memory from SDRAM
2. GLBCE is programmed for input frame-D
3. Frame-D is given to GLBCE
4. GLBCE processes frame-D based on statistics-C
5. GLBCE generates statistics-D based on frame-D in vertical blanking period
6. (Optional process) statistics-D is read from cache memory and moved to SDRAM

In Section 6.7.3.5.6, some examples closer to real world use cases are explained based on the above two processes.

6.7.3.5.6 GLBCE Continuous Frame Processing

This is a very basic case, which does not need transferring statistics between cache memory and SDRAM. The most common example is video capture. In this case, the most straight forward way to run GLBCE, is to generate statistics from N-th frame, and apply it to (N+1)-th frame. The effect from using statistics of different frame should be small if the difference between frames is small. For example, it is expected this does not generate artifact, if the video frame rate is 30fps or larger. Below this rate, the single frame processing programming model explained in Section 6.7.3.5.7 should be considered.

In the case of continuous frame processing, GLBCE is operated in the following manner:

1. GLBCE is programmed for incoming frame
2. Input frame comes in
3. GLBCE processes Input frame based on the statistics from the previous frame
4. GLBCE generates statistics based on the current input frame in the vertical blanking period
5. Repeat Step 1 through 4 as required

For the first frame of a set of sequential frames, Strength parameter should be 0, so GLBCE does not affect the image based on wrong statistics. SW may need to increase the strength value gradually after the first frame.

Copyright © 2024 Texas Instruments Incorporated

6.7.3.5.7 GLBCE Single Image Processing

For a single image, GLBCE needs to process the image twice, once for statistics generation, and once for actual GLBCE process.

1. GLBCE is configured for the input image.
2. The image is given to GLBCE. (Output image from GLBCE should be surpressed, for example, by disabling the next module in the path)
3. Statistics is generated in vertical blanking period.
4. The input image is given to GLBCE again. Output image with right GLBCE is generated.

   Also, GLBCE can take in small size image in the first pass to reduce processing time.
5. A small size version (IMG-S) of the input image (IMG-IN) is generated.
6. GLBCE is configured for the small size image (IMG-S).
7. The small size image (IMG-S) is given to GLBCE.
8. Statistics for the small size image is generated in vertical blanking period.
9. GLBCE is configured for the original input image (IMG-IN).
10. The original input image (IMG-IN) is given to GLBCE.

Copyright © 2024 Texas Instruments Incorporated

6.7.3.6 VISS Flexible Color Processing (FCP) Module

6.7.3.6.1 FCP Overview

The Flexible Color Processing module is responsible for performing CFA Interpolation as well as performing a host of color processing functions for generating different color outputs.

6.7.3.6.1.1 FCP Features Supported

The FCP include the following main features:

- Flexible CFA for generating multi-plane output for any 2x2 raw format sensor configuration.
  - Supports RCBC, RCCC, Bayer, RGBC as well as other formats
  - Can generate up to 4 independent color planes
  - Support for up to 3 directions per color plane
  - Adaptive Threshold based on intensity measure
  - Support for 16 bit processing in CFA
- Support for multiple color format output generation:
  - Support 12-bit YUV output
  - Supports RGB output
  - Supports Saturation outputs (8 bits only)
  - Supports Grayscale/Luma/IR/Clear output.
- Support for 16 bit to 12-16 bit compression/translation (LUT based) at the entry of the blockCFA
- Support for flexible output format generation including YUV as well as RGB, SV, and so forth.
- Single cycle/pixel performance
- Support for Edge Enhancement

6.7.3.6.2 FCP Functional Description

The Flexible Color Processor (FCP) receives data from RAW-FE and does demosaicing and color conversion. The output of FCP is sent to external memory.

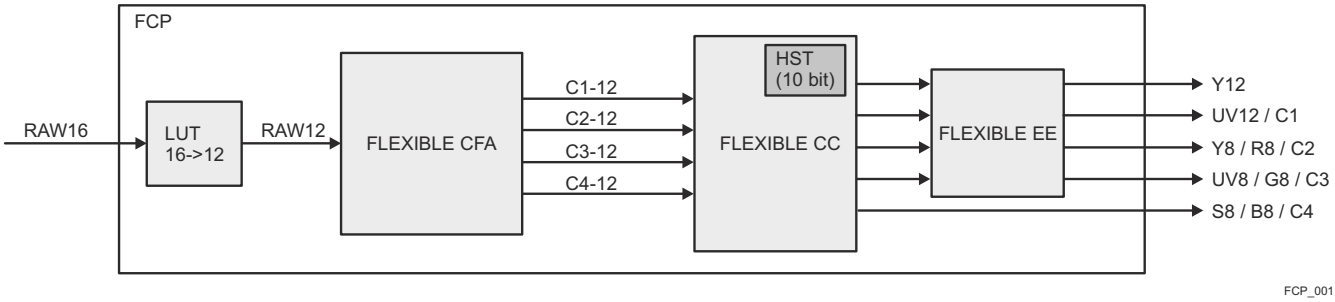


Figure 6-91. FCP Block Diagram

The FCP consists of the following major sub-blocks for supporting different sensor and output formats:

- LUT based compressionor translation: used to reduce or translate the bit width from 16 bits to 12 bits while still preserving the dynamic range.
- Flexible CFA: this block takes in as input any 2x2 raw sensor pattern and is capable of generating up to 4 full resolution color planes.
- Flexible Color Conversion (Flexible CC): this block takes in the output of the Flexible CFA and generates multiple standard as well as custom data formats.
- Flexible Edge Enhancer (Flexible EE): this block can take the output of the Color Conversion and align the Y and UV channels as well as enhance the edges in the luma channel.

6.7.3.6.3 FCP Submodule Details

6.7.3.6.3.1 Flexible CFA / Demosaicing

6.7.3.6.3.1.1 Feature-set

The Flexible CFA include the following main features:

- Support for Flexible CFA for generating multi-plane output for any 2x2 raw format sensor configuration.

Copyright © 2024 Texas Instruments Incorporated

  - Supports RCBC, RCCC, Bayer, RGBC as well as other formats.
  - Can generate up to 4 independent color planes.
  - Supports Clear (C) pixel.
  - Supports Infrared (IR) pixels.
- Edge aware CFA interpolation.
- Software programmable filter for interpolation up to 4 color channels.
- Support for up to 4 phases and 3 directions per color channel.
- Software controlled algorithm for direction selection.
- Adaptive threshold calculation for direction selection.
- Support of up to 2 sets of programmable mask based gradient selection
- Support for 16-bit pixel processing

6.7.3.6.3.1.2 Block Diagram of Flexible CFA

The Flexible CFA module follows the VPORT (internal) interface for both the input/output to the RAW-FE or Flexible CC module.

6.7.3.6.3.1.2.1 Gradient/Threshold Calculation

The Flexible CFA incorporates 2 sets of gradient and direction selection logic. Typically for symmetric 2x2 patterns like Bayer, only a single set is sufficient; however in sensors wherein one of the channels is working on a different data type, the second set can be used for that channel. An example of such sensors is the RGB-IR sensor, wherein the 2x2 pattern consists of one pixel of each color channel and one pixel of the IR channel. In this case, mixing the gradients and intensity of the color channels with the IR channel could be meaningless.

The gradient selection uses bit masks to select the desired pixel position for calculating the horizontal and the vertical gradient.

Figure 6-92 shows the notation for the bit-mask values. The same notation is used for vertical masks as well and the notation implies that setting a ‘1’ in any field of the bit Mask enables the absolute difference of those 2 pixels to be summed in the gradient calculation.

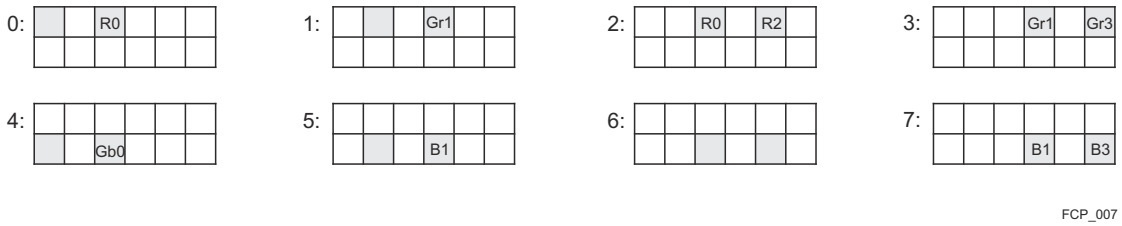


Figure 6-92. Gradient Bit Masks

Similar masks are also used to calculate the intensity (see Figure 6-93), however instead of taking the absolute difference, the bit masks are used to sum up the pixels in those locations.

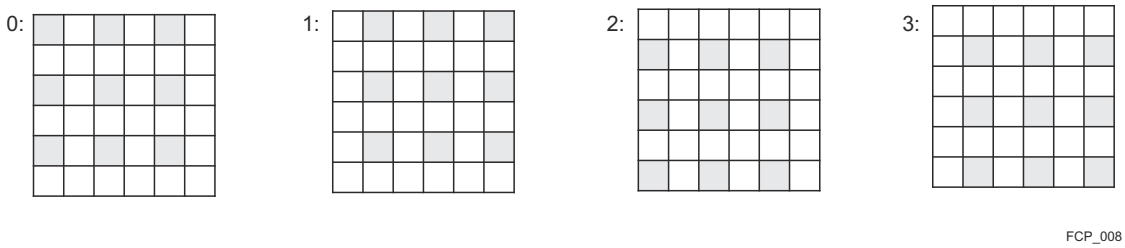


Figure 6-93. Intensity Bit Mask

The 4 bit Intensity mask is used to choose which pixels in the 6 rows should be used for creating the intensity sum. The same masks are used for every 2x2 window within the 6x6 buffer. (Thus setting all 4 bits to ‘1’ will be equivalent to summing up the entire 6x6 window.) A programmable shift is then used to bring the intensity down to the 0-16k range. The maximum value of intensity can be 6x6 x4k = 144k, so the maximum value of shift is 4 (divide by 16). The shift value can be programmed to any value between 0 and 4 and should be chosen based

Copyright © 2024 Texas Instruments Incorporated

on the mask. Note that dependent on the shift value, it's possible that the full range of 0-16k of the intensity may not be reached, hence the adaptive threshold should be programmed accordingly.

The adaptive threshold calculation uses the calculated intensity. For more information, see Section 6.7.3.6.3.1.2.2, *Software Controlled Direction Selection*.

6.7.3.6.3.1.2.2 Software Controlled Direction Selection

Software can control or guide direction selection at frame or sequence level as shown in Figure 6-94.

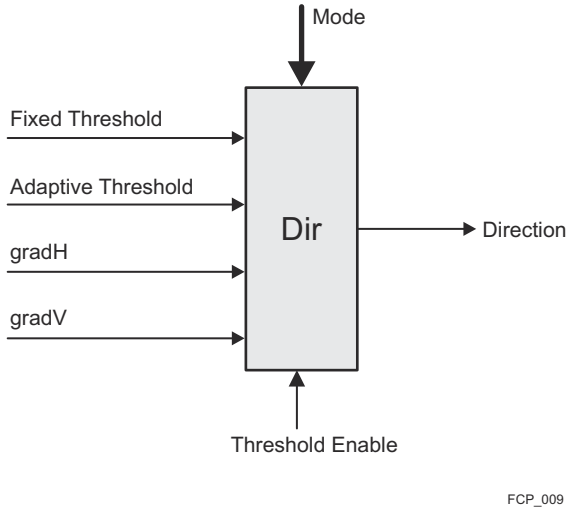


Figure 6-94. Software Controlled Direction Selection

Table 6-149 summarizes number of option available to software.

Table 6-149. Summary of HSX Spaces

Mode	Name	Details
0	Horizontal	Dir = 0
1	Vertical	Dir = 1
2	HV	Dir = 2
3	HV_Threshold (Fixed threshold is MMR)	in Dir = 0: gradH ≥ gradV + threshold in Dir = 1: gradV ≥ gradH + threshold in Dir = 2: Otherwise
4	HV_AdaptiveThreshold	in Dir = 0: gradH ≥ gradV + threshold in Dir = 1: gradV ≥ gradH + threshold in Dir = 2: Otherwise

For calculation of adaptive threshold, 7 options are used. Each one defines a threshold to be used as a different intensity level as it follows:

- Entry-0 -> Intensity = 0
- Entry-1 -> Intensity = 512
- Entry-2 -> Intensity = 1024
- Entry-3 -> Intensity = 2048
- Entry 4 -> Intenisty = 4096
- Entry 5 -> Intensity = 8192
- Entry 6 -> Intensity = 16384

The adaptive threshold can then be calculated as a linear interpolation between 2 successive entries based on the given intensity.

6.7.3.6.3.1.3 Example Filter Coefficients - Green Interpolation

Figure 6-95 shows an example Green filter for Bayer.

Copyright © 2024 Texas Instruments Incorporated

Four left thermograms show the directional filter for green interpolation. Combines early cutoff and extended cutoff filters.

While the two on the right shows the non-directional filter. Used when no direction is preferred. Provides reasonable passband in horizontal and vertical directions without extending too far to avoid artifacts.

Note that all phases of same component (for example Green) have identical frequency responce but phase shifterd based on position.

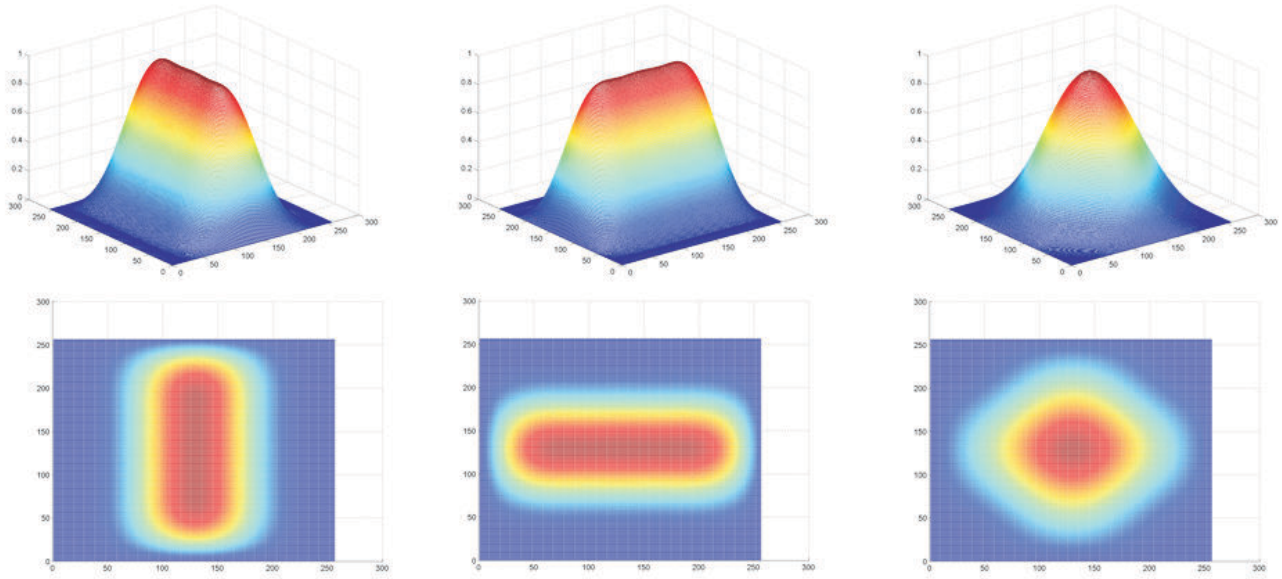


Figure 6-95. Example Green Filter for Bayer

6.7.3.6.3.1.3.1 Example Filter Coefficients - Red/Blue Interpolation

Figure 6-95 shows an example Red/Blue filter for Bayer.

Two left thermograms show the lowpass filter used on red/blue images.

While the four right shows the directional highpass filter used on green image to add additional high frequency content to red/blue images. In non-directional case, these are not used.

Note that all phases of same component (for example Red/Blue) have identical frequency responce but phase shifterd based on position.

Copyright © 2024 Texas Instruments Incorporated

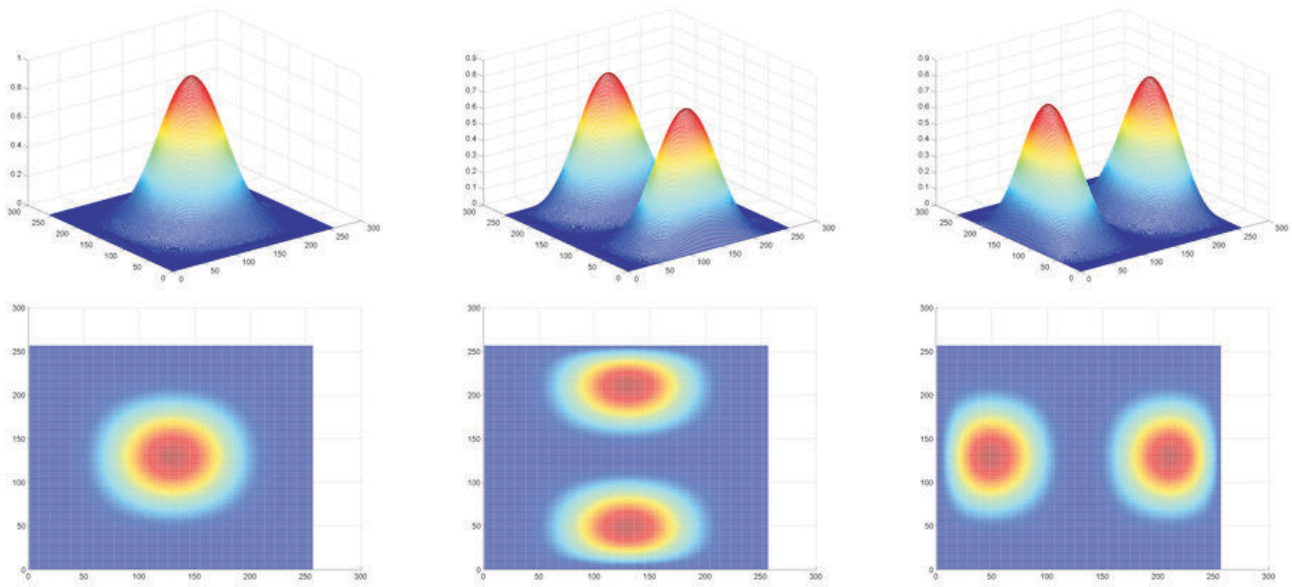


Figure 6-96. Example Red/Blue Filter for Bayer

6.7.3.6.3.1.4 CFA 16-Bit Upgrade

The following changes are done to make the CFA 16 bit compliant:

- Core filter data path is updated to 16 bits
- Gradient block
  - The gradients are normalized by adding a right shift of 4, this retains the range of gradients as the older implementation
  - Gradients are compared with thresholds and wrt each other
- FLXD_Intensity
  - Increated the size of the shift (after intensity calculation) by 1 bit (4 bits now)
  - Refer to the description of SET0_INTENSITY0 (Intensity BitField), (0x1d98)
- FLXD_ThrCalc()
  - No update since intensity remains in the same range as previous implementation

6.7.3.6.3.1.5 FIR Filter Output Scaling

Prior to the final output of the C1-C4 FIR Filter there is a scaling and offset addition.

That is the sum of the selected coefficients (derived from the intensity, gradient logic and controls above) times the pixel values is then sent through a scaling multiplier U14Q8 and then added an offset U16.

6.7.3.6.3.1.6 Decopanding, 24-bit Color Conversion Matrix and Companding Blocks

Figure 6-97 shows the block receives the internal Flex-CFA outputs and allows the pixel data to be decompanded up to 24-bits prior to going through a Color Conversion Matrix and then Companded back to 12-bit for the Flex-CC.

Copyright © 2024 Texas Instruments Incorporated

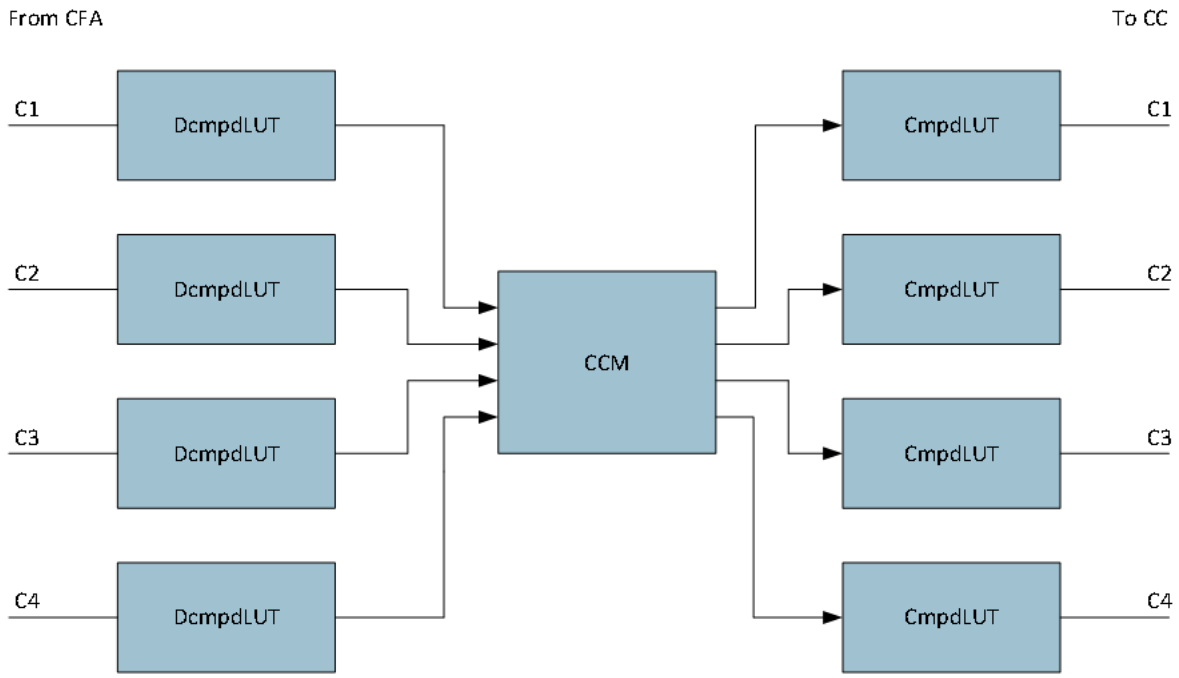


Figure 6-97. Block Diagram

6.7.3.6.3.1.6.1 The DcmpdLUT Block

The DcmpdLUT block can expand the 12- or 16-bit pixel data up to 24 bits for the Color Conversion Matrix block to process. Due to the input bit width being tied to 16, only 609 entries in the DLUT table are used. Normally used when the 16-bit data path is enabled.

6.7.3.6.3.1.6.2 The CCM Block

The CCM block uses the following formula to determine the output data Ex; the input data Cx is from the DcmpdLUT blocks.

E1(x,y)=(W11*C1(x,y))+ =(W12*C2(x,y))+ =(W13*C1(x,y))+ =(W14*C4(x,y))+OFFSET1

E2(x,y)=(W21*C1(x,y))+ =(W22*C2(x,y))+ =(W23*C1(x,y))+ =(W24*C4(x,y))+OFFSET2

E3(x,y)=(W31*C1(x,y))+ =(W32*C2(x,y))+ =(W33*C1(x,y))+ =(W34*C4(x,y))+OFFSET3

E4(x,y)=(W41*C1(x,y))+ =(W42*C2(x,y))+ =(W43*C1(x,y))+ =(W44*C4(x,y))+OFFSET4

Where W*ab* is CCM_OCH*b*_ICH*a* register fields. And OFFSET*c* is CCM_OCH*c*_OFFSET register fields.

6.7.3.6.3.1.6.3 The CmpdLUT Block

The CmpdLUT block compresses the selected data from the CCM back to 12-bit pixel data for the CC to process. The input width is controlled by the dandc_com_ctrl.linearbitwdth register setting.

6.7.3.6.3.1.6.4 Controls for the Decompanding, CCM, and Companding Blocks

- dCmpdLutEn - enable decompanding LUT
- LinearBitWidth - bit width at which DLUT, CCM and CLUT Input operations (Can be 16-24 range)
- ccmEn // Enable CCM logic
- cmpdLutEn // enable companding LUT
- CCM Controls with offset
  - Weights (4x4) are S12Q8
  - Offset is 26 bits signed (S26Q24) and can span a range of -2 to +1 for 24 bit data
  - Offsets need to be programmed by the user correctly for the desired LinearBitWidth

Copyright © 2024 Texas Instruments Incorporated

– For instance, if the CCM output is Oout0 = W0R0 + W1G0 + W2B0 + 1 // Offset should be set to 2^24 if the data is 24 bits or 2^16 if the data is 16 bits

6.7.3.6.3.1.6.5 Example Use Cases

The addition of these registers enables multiple use cases, not all of which are valid.

Table 6-150 highlights some possible use cases and the following MMR settings.

Table 6-150. Use Cases

	Name	Cfg_1.en16bitmode	LutEn (Legacy LUT)	Dcmpd En	Cmpd En	Linear BitWidth	CCM	Comments
0	Legacy	0	Valid	0	0	12	Valid	Legacy mode but can use an optional CCM
1	Basic 16 bit	1	0	0	0	16	Valid	Enhanced mode which can use a 16 bit CFA, however no companding / decompanding. A CCM may be used. In this usecase the data is shifted down to 12 bits after CCM by the logic automatically
2	Basic 16 bit – inverse tone map (Visual)	1	0	1	0	16	Valid	Similar mode as basic 16 bit, however cmpnding LUT is used to apply an inverse gamma. CCM may be optionally used. This mode is similar to enhanced legacy visual mode but utilizes the new 16 bit CFA
3	20 Bit RCCB analytics	1	0	1	1	20	RCB -> RGB	This is 20 bit RCCB mode. The decompanding LUT maps to 20 bit, linear CCM converts RCB to RGB and then a LOG curve is applied with companding LUT
4	24 Bit RCCB analytics	1	0	1	1	24	RCB -> RGB	This is 20 bit RCCB mode. The decompanding LUT maps to 24 bit, linear CCM converts RCB to RGB and then a LOG curve is applied with companding LUT

6.7.3.6.3.2 Edge Enhancer Module Wrapper (WEE)

Figure 6-98 shows a high level block diagram of the Edge Enhancer (EE) module wrapper. For more information about the Edge Enhancer algorithm, see *Edge Enhancer (EE)*. The wrapper places the Edge Enhancer within the FCP structure along with assotiated muxes.

Copyright © 2024 Texas Instruments Incorporated

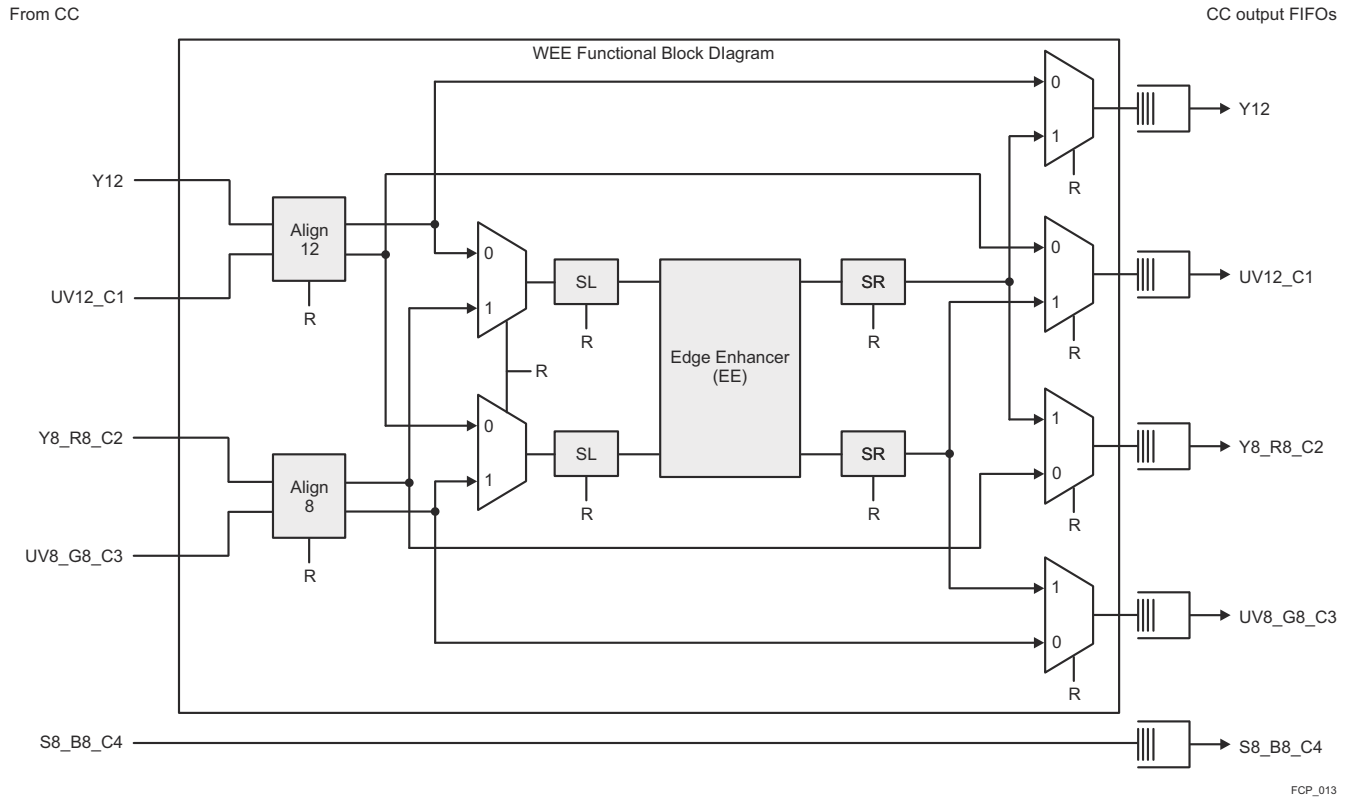


Figure 6-98. Block Diagram of Edge Enhancer Module Wrapper

6.7.3.6.3.2.1 Align 12 Block

The Align 12 block is responsible for aligning the 12-bit Chroma and Luma channels. It supports a missalignment of up to ±19 pixels. The Align 12 block can be used independently from the EE block so that Y12 and UV12 can be output simultaneously.

6.7.3.6.3.2.2 Align 8 Block

The Align 8 block is responsible for aligning the 8-bit Chroma and Luma channels. It supports a missalignment of up to ±19 pixels. The Align 8 block can be used independently from the EE block so that Y12 and UV12 can be output simultaneously.

6.7.3.6.3.2.3 Mux Blocks

The six mux blocks muxes one of two frames to the destination port. This is not a simple mux, but it only switches on frame boundaries. This means overlapping frames will not create frame fragments. Frame B is the only one to be forwarded and only after frame A completes. So if the frames overlap, a partial frame may be dropped during the switch. When the output MUX blocks are all bypassing the EE, the EE clock is gated to save power.

6.7.3.6.3.2.4 SL - Shift Left Block

This block shifts the data left by the programmed amount. Changes to the shifted left value only occurs prior to SOF (start-of-frame) received at the block. Which means the shifted amount is not changed during a frame.

6.7.3.6.3.2.5 EE - Edge Enhancer Block

The Edge Enhancer module is used to enhance the visual quality of the image by increasing its sharpness. The module only works on Y (Luma) data and uses a combination of 2D High-Pass Filtering to detect edges and then apply them on the input image.

6.7.3.6.3.2.6 SR - Shift Right Block

This block shifts the data right by the programmed amount. Changes to the shifted right value only occurs prior to SOF received at the block. Which means the shifted amount is not changed during a frame.

Copyright © 2024 Texas Instruments Incorporated

6.7.3.6.3.2.7 Edge Enhancer Module Wrapper (WEE) Registers

The WEE control registers are effectively shadowed which means changes take effect at the start-of-frame. The only exception is the WEE muxes which may have frame overlap. This happens if a switch occurs prior to frame B start-of-frame, but frame A is still in transit when the frame B arrives, then the first frame of frame B is dropped. In such case the Mux does not send a fragmented frame due to a mux switch occurring.

6.7.3.6.3.3 Flexible Color Conversion (CC)

Figure 6-99 shows a high level block diagram of the Flexible Color Conversion (CC) module.

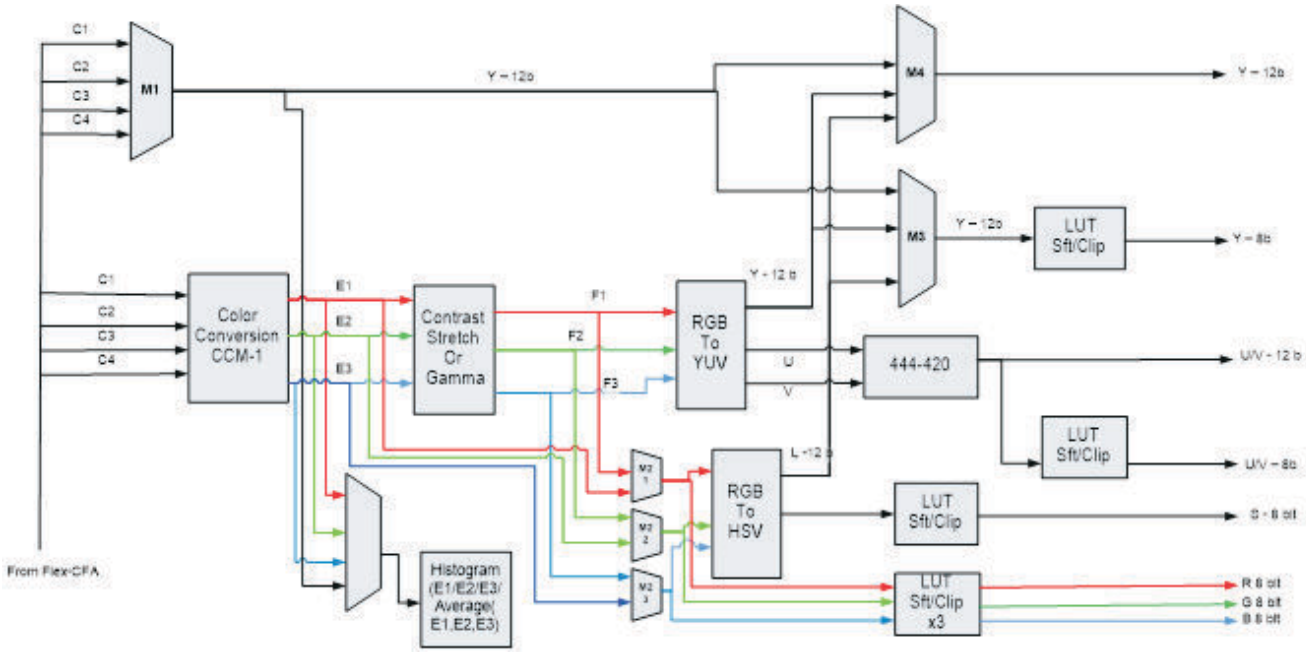


Figure 6-99. Flexible CC Block Diagram (Logical View)

The Flexible CC enables multiple use cases and allows different color formats to be generated.

The Flexible CC module follows the VPORT (Internal) Interface for both the input to the Flexible CC module.

6.7.3.6.3.3.1 Interface Mux

The Flexible CC logical block diagram has 5 primary outputs connecting to the LSE modules in the VISS. For more information about LSE modules, see *Load Store Engine (LSE)*. A multiplex scheme is used to connect multiple outputs on to 5 interfaces using the interface Mux. The Flexible CC output can optionally also allow data from Flexible CFA to be tapped out on to one of the primary interfaces. Note that 12-8 LUTs are shared between the RGB generation and the Y/UV 8-bit generation logic.

Figure 6-100 shows the details of the interface mux. There are only 3 LUTs shared in different mode and only 5 concurrent output streams can be activated at any point of time. For 8-bit outputs, the data is stored in the MSB 11-4 bits of the 12-bit bus. Further, the Y12 output is dedicated and not multiplexed with other interface signals.

Copyright © 2024 Texas Instruments Incorporated

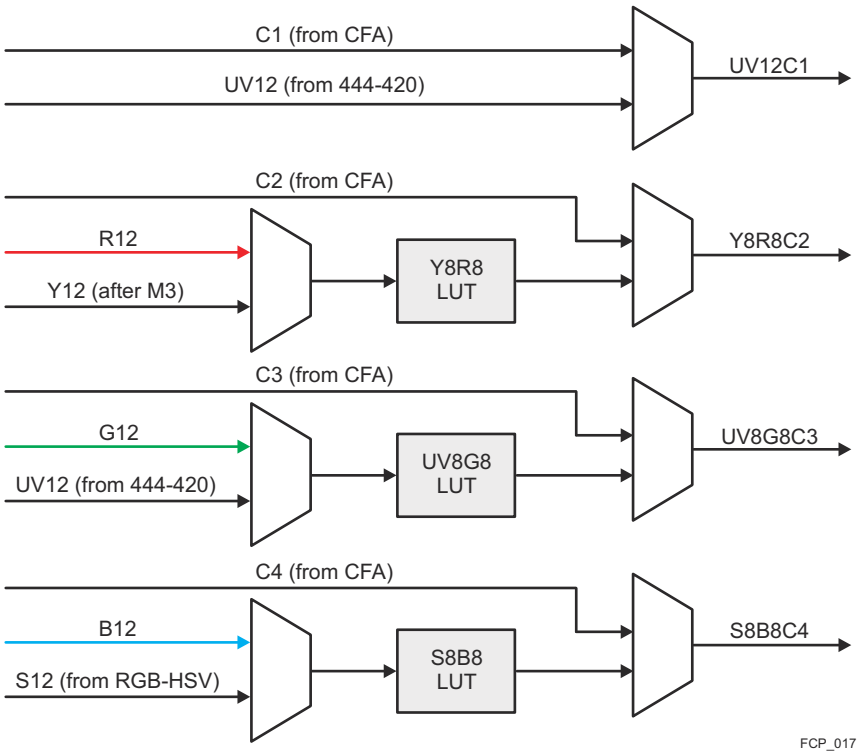


Figure 6-100. Flexible CC Interface Mux

6.7.3.6.3.3.2 Color Conversion (CCM-1)

The Color Conversion (CCM-1) is defined using the mathematical equations from Figure 6-101.

$$E1(x,y) = W11 \times C1(x,y) + W12 \times C2(x,y) + W13 \times C3(x,y) + W14 \times C4(x,y) + Offset_1$$
$$E2(x,y) = W21 \times C1(x,y) + W22 \times C2(x,y) + W23 \times C3(x,y) + W24 \times C4(x,y) + Offset_2$$
$$E3(x,y) = W31 \times C1(x,y) + W32 \times C2(x,y) + W33 \times C3(x,y) + W34 \times C4(x,y) + Offset_3_$$

Figure 6-101. CCM-1

The CCM matrices weights (W**) are 12 bit signed each (S12Q8) representing a range from -8 to +7.996 with 8 bits of fraction. The offsets are 13 bit signed notation (S13Q11) representing a range of -4096 to +4095. The offset is effectively shifted by one (left shifted/ multiplied by 2) before being applied. The effective range of offset is -2 to +1.995 with 11 bits of fraction.

The 4 component CCM allows for a power combination of color format generation to be addressed.

6.7.3.6.3.3.3 RGB to HSX Conversion

The RGB-HSV module is used for converting color spaces and creating custom luminance data plane. This way the 8-bit Saturation and 12-bit Luminance (L or V) plane are generated.

Figure 6-102 shows the block which generates the Luminance / Gray Scale Plane as well as the Saturation (8-bit) plane.

Copyright © 2024 Texas Instruments Incorporated

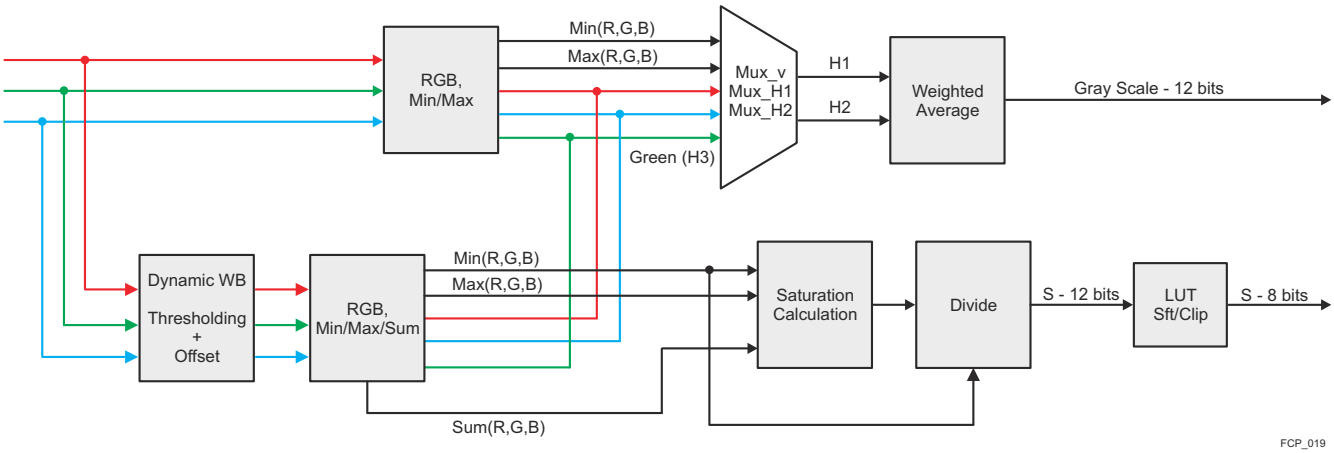


Figure 6-102. S & V Generation

6.7.3.6.3.3.3.1 Weighted Average Block

The Weighted Average block implements the equation shown on Figure 6-103.

$$GrayScale(x,y) = W1 \times H1(x,y) + W2 \times H2(x,y) + W3 \times H3(x,y) + Offset_1$$

Figure 6-103. Gray Scale Computation

In the equation on Figure 6-103 the Weights (W1*) are 12 bits signed with a range of -8 to +7.9996 in S12Q8 format. The offset is Signed 13 bits in S13Q11 format with a range of -2 to +1.995.

Figure 6-104 depicts the archtiectue of the Weighted Average block:

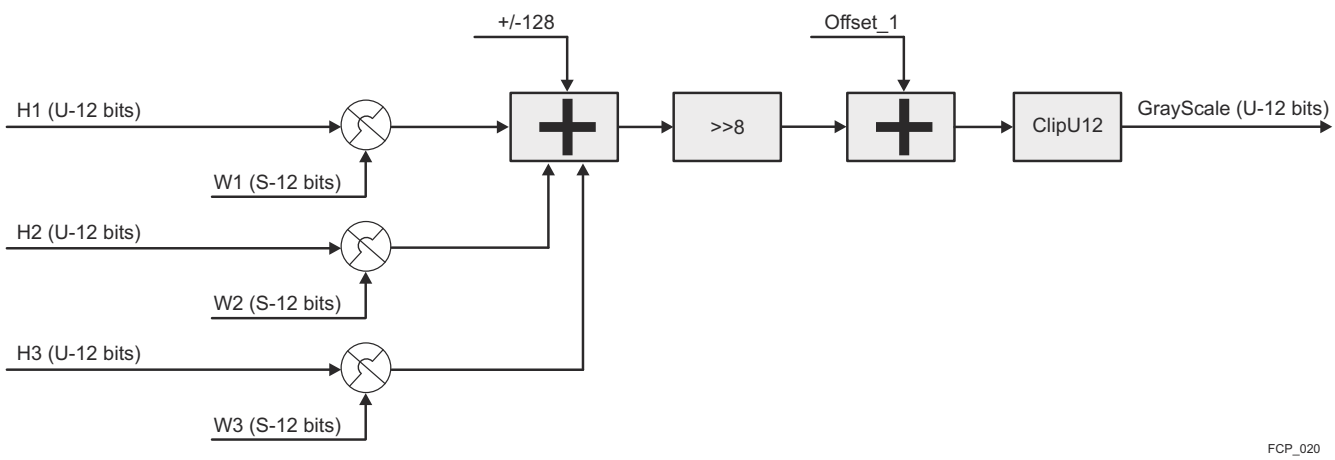


Figure 6-104. Weighted Average Block

The mux before the matrix chooses whether the native R/B channels are transmitted or alternatively the Min/Max (RGB) from the Min-Max block are transmitted. Further, the calculation of V can work on either the non-WB corrected data or the WB corrected version. For more information about WB correction, see Section 6.7.3.6.3.3.3.2, *Saturation Block*.

The combination of the Min-Max block, the mux and the Weighted Matrix allows any of the Gray Scale calculation as shown in Table 6-151.

Copyright © 2024 Texas Instruments Incorporated

Table 6-151. Summary of HSX Spaces

No	Formula	Configuration
Grey scale (HSI)	(R+G+B) / 3	H1 = R, H2 = B; Offset_1 = 0; W1 = W2 = W3 = 1/3 = 85;
Gray Scale (HSV)	Max(RGB)	H1 = Max(RGB); H2 = NA Offset_1 = 0; W2/W3 = 0; W1 = 1 (256);
Gey Scale HSL	(Max(RGB) + Min (RGB)) / 2	H1 = Max(RGB), H2 = Min(RGB); W1 = W2 = 0.5 (128) Offset_1 = 0; W3 = 0;
		H1 = R, H2 = G; W1 = W3 = 0.25 (64); W2 = 0.5 (128) Offset_1 =0

Table 6-152. Grey Scale and Saturation Calcuation

No	HSI Color Space	HSV Color Space	SHL Color Space	Customer Requirements
Grey Scale Computation	(R+G+B) / 3	Max(R,G,B)	(Max(R,G,B) + Min(R,G,B)) / 2	(R+2*G+B) / 3
Saturation (S) Computation	1-(Min(R,G,B) / (R+G+B))	(Max(R,G,B) - Min(R,G,B)) / Max(R,G,B)	(Max(R,G,B) - Min(R,G,B)) / 255-Gray value	S = Max(R,G,B) - Min(R,G,B)
Hue (H) Computation	Complex formula involving (R-G) (R-B) & (G-B)	Division formula livolving (R-G) (R-B) & (G-B)	Complex formula involving all component	Separate output for (R-G) and Yellowness

6.7.3.6.3.3.3.2 Saturation Block

The Saturation block first applies a dynamic white balance offset to correct the saturation plane when the image is in log domain and is used for analytics data flow. Once the WB offset is applied, the saturation calculation proceeds as described below. Note that the White balance offset is only applied if the independent pixel values are below a threshold (VISS_FCP_FCC_RGBHSV_WB_LINLOGTHR_1/2). Further, even the minimum of RGB is compared against a threshold (VISS_FCP_FCC_RGBHSV_WB_LINLOGTHR_2[27-16] SATMINTHR bit field) and the higher of the two is used. The saturation calculation always uses the data with the WB correction applied, however the V calculation can choose between WB corrected or uncorrected data using VISS_FCP_FCC_CFG_1[26] MUXRGBHSV_MUX_V.

Table 6-152 shows the condensed form for the Grey Scale and Saturation calculation. In the Flexible CC the saturation calculation is supported using two modes based on the VISS_FCP_FCC_CFG_2[6] HSVSATMODE bit as follows.

- HSVSATMODE = 0: Max(RGB) - Min(RGB)
- HSVSATMODE = 1: SUM(RGB) - Min(RGB)

Similarly the denominator for the dvision is chosen using the following selection on the VISS_FCP_FCC_CFG_2[5-4] HSVSATDIVMODE bit field.

- HSVSATDIVMODE = 0: No division, denominator = 1
- HSVSATDIVMODE = 1: Max(RGB)
- HSVSATDIVMODE = 2: 4095 – Gray value (computed in Figure 6-103)
- HSVSATDIVMODE = 3: Sum(RGB)

6.7.3.6.3.3.3.3 Division Block

The division operation is implemented to calculate 1/x followed by a multiplication with the value. The input to the LUT is 12-bit unsigned value and the output is 8-bits fraction representing 1/x calculation. The divide LUT is implemented as a non-linear ROM with 1216 (1280 for aligning to 128 size boundary) entries and up to 2 segments to reduce error. Ideally to implement a full ROM for an input space of 12 bits would require 4k entires, however to save space and reduce error, the ROM is partitioned such that it is full resolution for the first 256 entries where the curve is highly non linear, whereas the next 4k - 256 entires have a step size of 4. Each ROM entry has a bit size of 20 bits in U20Q20.

Copyright © 2024 Texas Instruments Incorporated

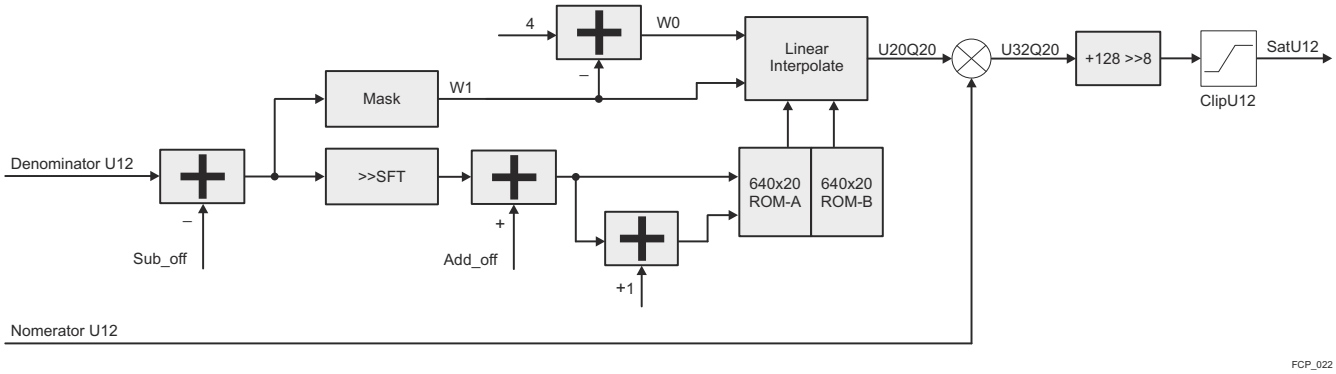


Figure 6-105. Division LUT

These are the parameters that need to be calculated based on the range of the input.

- SFT: ‘0’ if input <256, 2 otherwise
- Mask: ‘0’ if input <256, 4 otherwise
- Add_off: ‘0’ if input <256, 256 otherwise
- Sub_off: ‘0’ if input <256, 256 otherwise

The ‘shift’ parameter at the end is a configuration register which needs to be programmed. Typical programming value is 8 to generate a result in Q12 format (such that the range is 0 to1). This is usually the case when the denominator is greater than the numerator.

6.7.3.6.3.3.3.4 LUT Based 12 to 8 Downsampling

In the final step the generated saturation value (12 bit) or the incoming RGB values (12 bit) needs to be scaled down to 8 bits using the linear LUT with 513 entries.

Figure 6-106 is a high level block diagram of the LUT block. It shows the weight calculation for the LUT assuming the incoming data can be of any bit depth between 8 and 12 bits.

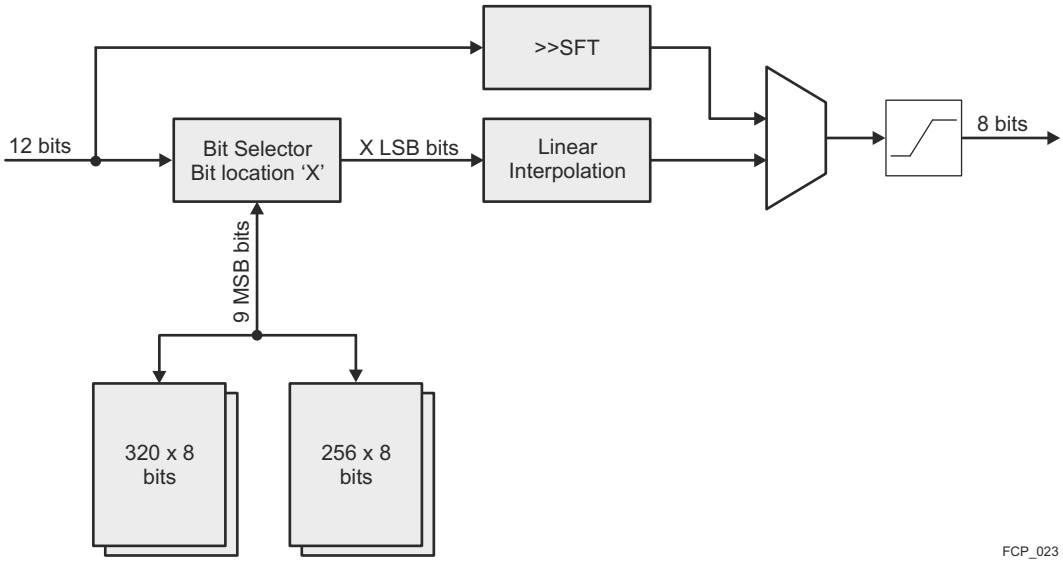


Figure 6-106. LUT Based 12-8 Compression

The LUTs are sized to provide 513 locations of data, to enable linear interpolation for all locations. The internal weights are sized as 3 bits since the maximum delta between 2 steps is only 4095/512 = 8.

Figure 6-107 shows the high level block diagram of the interpolation process. The logic is designed to scale for bit width and can support anything from 8 to 12 bits of input to support different usecases. However, since the

Copyright © 2024 Texas Instruments Incorporated

step size is different depending on the input bit width, the bit selection and weight calculation logic is designed to account for that.

A shift operation can be performed instead of LUT to reduce bitwidth from 12 down to 8 bits.

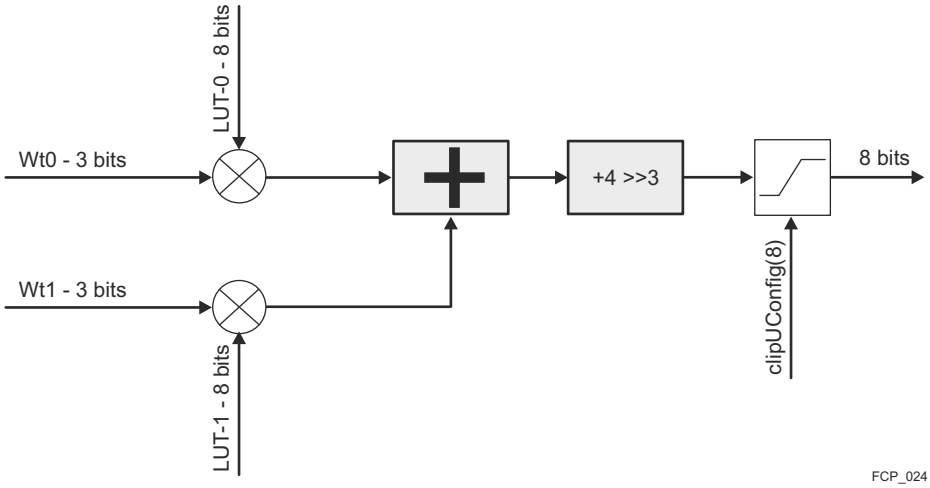


Figure 6-107. Linear Interpolation on LUT

The code below shows the addressing for the LUT as well as the weight calculation logic for supporting multiple bit widths at the input from 8 – 12 bits.

12-8 Bit LUT Curve

```
CASE BIT_SEL
12 (12 bit data): Addr = Inp[3:11]; Wt1 = Inp[2:0]
11 (11 bit data): Addr = Inp[2:10]; Wt1 = Inp[1:0] & '0'
10 (10 bit data): Addr = Inp[1:9]; Wt1 = Inp[0] & '00'
9 (9 bit data): Addr = Inp[0:8]; Wt1 = '000'
8 (8 bit data): Addr = Inp[0:7]; Wt1 = '000';
//Wt0 is always calculated at 8 - Wt1
Wt0 = 8 - Wt1
```

The input bit width (BIT_SEL) is specified using a dedicated register for generating Y8 output. However for other paths (RGB/UV/Sat) the 8 bit conversion is done using either 12 bits (if data is tapped prior to contrast block) as the bitwidth or the clipping value after contrast block (VISS_FCP_FCC_CFG_2[12-9] CONTRASTBITCLIP, if data is tapped after contrast block).

6.7.3.6.3.3.4 Histogram

The histogram can work on either one of the color components after the CCM-1 or the fourth color component routed directly from the Flexible CFA. The Histogram can also work on internally generated Luma value using simple averaging of RGB components. The Histogram module supports a local memory which can be read by the host processor. The Histogram provides data which can be used by the software to generate tone curves for the Contrast Stretch LUT.

The input to the Histogram module is calculated based on the control MMR as it follows:

- 0: R
- 1: G
- 2: B
- 3: MuxC1_4
- 4: (R+2*G+B) / 4

The histogram module downsamples the image in the horizontal direction by a factor of 2 using simple averaging filter. This ensures that the memory access requirements of the module are limited to 1 read + 1 write access every 2 cycles (1 access/cycle) and thus can be achieved using a simple memory architecture.

Copyright © 2024 Texas Instruments Incorporated

The histogram module requires a ping-pong memory which is mapped to the configuration bus. This allows for histogram operation to run in concurrent with the Read operation from the host. For ease of software complexity, both the Ping/Pong memories map to the same address and it is the responsibility of the design to ensure that the muxing logic always take care of the correct access from both the host and the module. To detect error conditions, if the host is not able to read fast enough from the memory, an error interrupt should be generated whenever the host has initiated a read transfer from the first location of the memory but has not initiated a read transfer from the last location of the memory.

The Histogram module has 2 banks of memores each of 256x20 bits which are used in ping pong fashion. The input pixel has to be downshifted to match the bin size. The Histogram module has separate registers for creating an ROI using horizontal/vertical start postions as well as horizontal/vertical size.

Typically the ROI registers (VISS_FCP_FCC_CFG_HIST_1 and VISS_FCP_FCC_CFG_HIST_2) have to be set that at least one line is skipped for histogram calculation. The one line of time can be used by the histogram module to set the value of each of the bins to zero. This also puts a constraint on the histogram ROI to have a minmum width of 256 pixels.

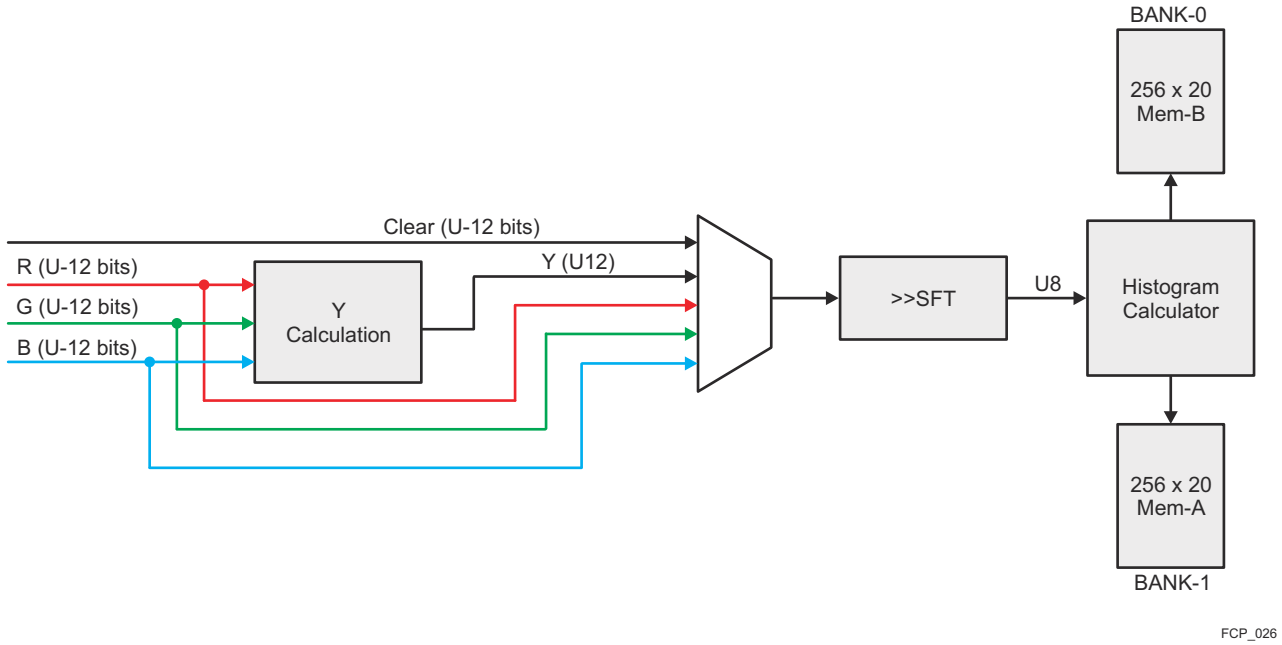


Figure 6-108. Histogram Module

6.7.3.6.3.3.5 Contrast Stretch / Gamma

The logic of the Contrast/Gamma unit is based on the same as that of the 12-8 bit conversion LUTs. There are 3 copies of the logic, one for each color channel. The standard data flow allows for a 12 bit input and 12 bit unsigned output, however different bit width combinations are possible using the select bit of the LUT and the clip value. Each entry of the LUT table is 12 bits.

Figure 6-109 and Figure 6-110 show the block diagram and implementation of linear interpolation for the Contrast Stretch/Gamma module.

Copyright © 2024 Texas Instruments Incorporated

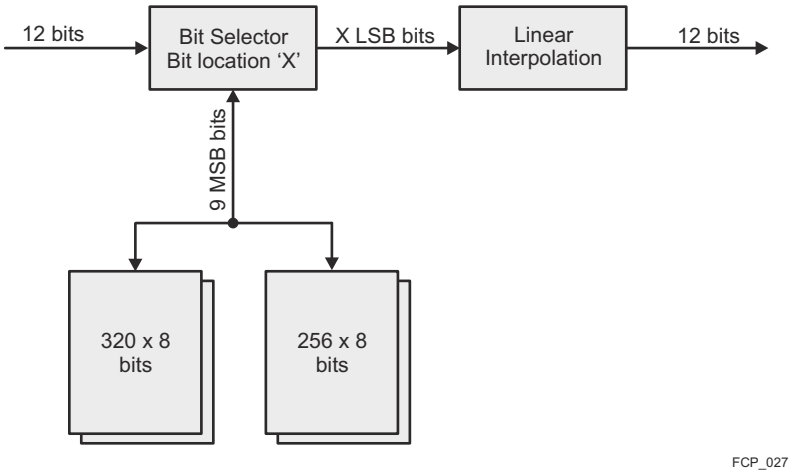


Figure 6-109. Contrast Enhancement Module

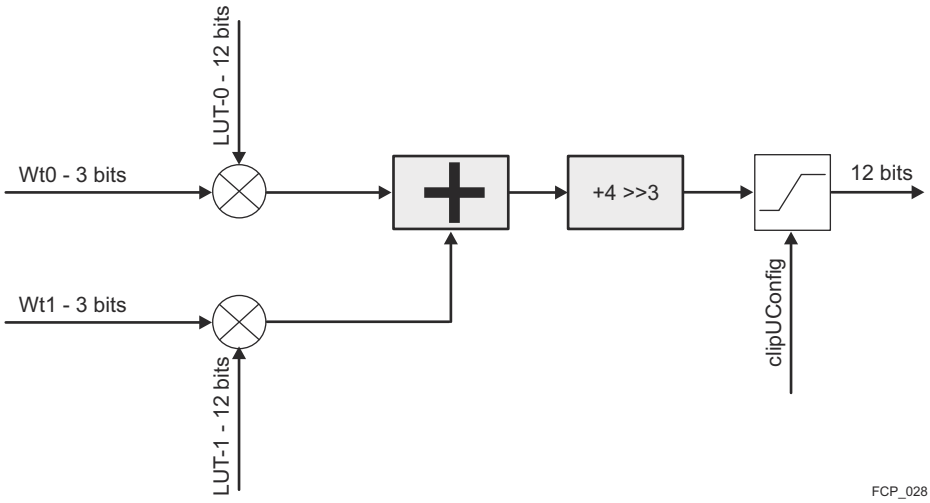


Figure 6-110. Contrast Enhancement Linera Interpolation

The code below shows the LUT addressing scheme which is similar to the 12-8 bit conversion module. The configurable clip at the end of processing allows for less than 12 bit output to be supported directly from the module.

Contrast Enhancement, LUT Addressing

```
CASE BIT_SEL
0 (12 bit data): Addr = Inp[3:11]; Wt1 = Inp[2:0]
1 (11 bit data): Addr = Inp[2:10]; Wt1 = Inp[1:0] & '0'
2 (10 bit data): Addr = Inp[1:9]; Wt1 = Inp[0] & '00'
3 (9 bit data): Addr = Inp[0:8]; Wt1 = '000'
4 (8 bit data): Addr = Inp[0:7]; Wt1 = '000';
//Wt0 is always calculated at 8 - Wt1
Wt0 = 8 - Wt1
```

6.7.3.6.3.3.6 RGB-YUV Conversion

The RGB-YUV conversion is a matrix based conversion from 12-bit RGB to 12-bit YUV. The output after conversion is YUV444.

The conversion is carried out using the equation on Figure 6-111.

Copyright © 2024 Texas Instruments Incorporated

$$Y(x,y) = WRY \times R(x,y) + WGY \times G(x,y) + WBY \times B(x,y) + Offset_1$$
$$Cb(x,y) = WRCb \times R(x,y) + WGCb \times G(x,y) + WBCb \times B(x,y) + Offset_2$$
$$Cr(x,y) = WRCr \times R(x,y) + WGCr \times G(x,y) + WBCr \times B(x,y) + Offset_3_$$

Figure 6-111. RGB-to-YUV Conversion

In this equation, the weights are signed 12 bits (S12Q8) with 8 bit of fraction precision, representing a range of -8 to +7.996. The offsets are signed 13 bits.

Figure 6-112 shows the implementation of the RGB to YUV module. The output after conversion is YUV444, as such it is followed by a chroma downsampling stage where in the Chroma resolution is reduced by half in both the horizontal and vertical direction.

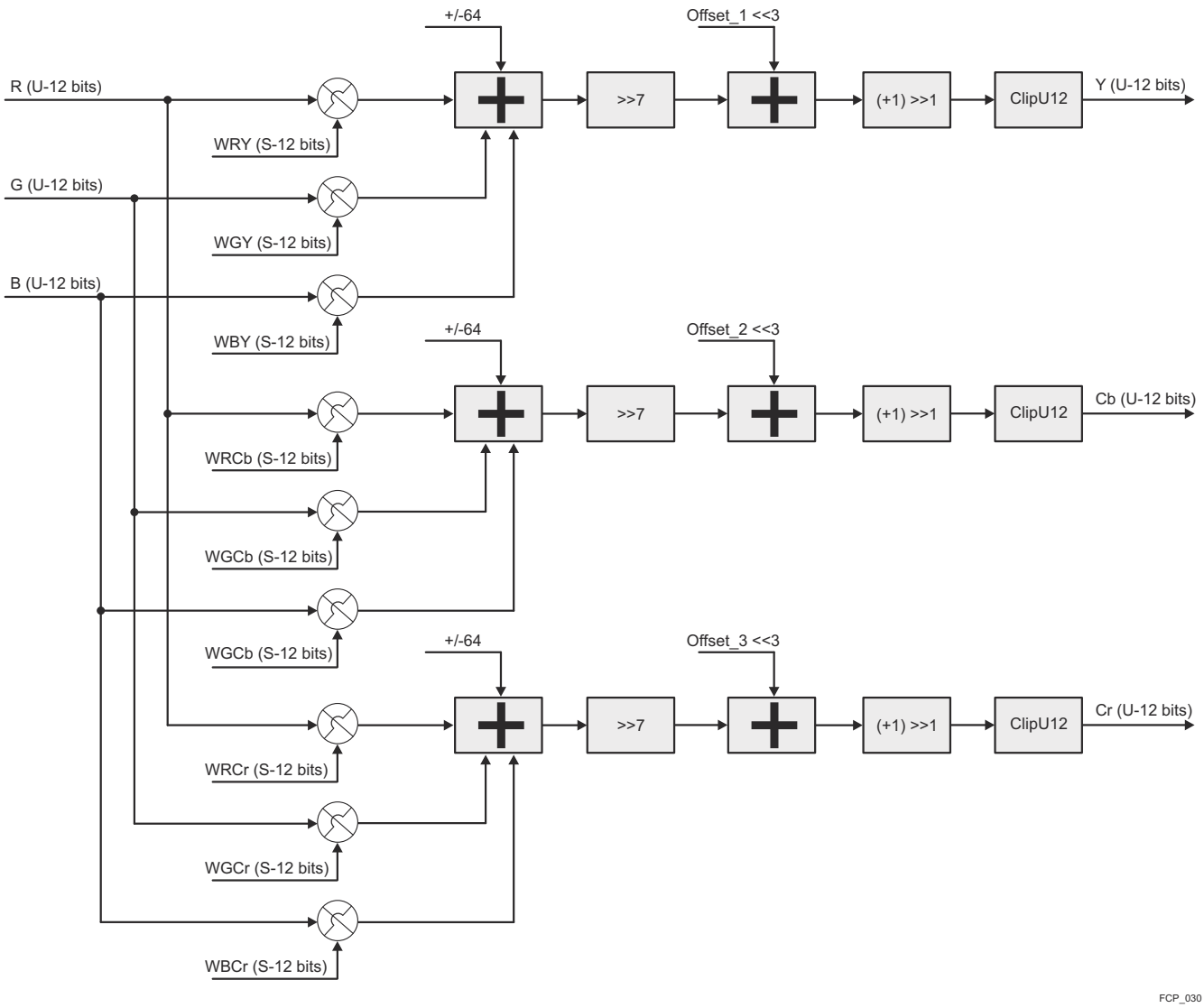


Figure 6-112. RGB to YUV

Copyright © 2024 Texas Instruments Incorporated

6.7.3.6.3.4 444-422/420 Chroma Down-sampler

This module down-samples the chroma components from 444-420 format; effectively reducing the chroma resolution to 50% in both the horizontal and vertical direction.

The downsampling is performed using a 3 Tap fitering [1 2 1] followed by decimation. The conversion logic generates chroma in co-sited/co-located format in both the horizontal and vertical direction.

Figure 6-113 shows the block diagram of the format conversion module. The Chorma output can be optionally downshifted from 12 bits to 8 bits using the LUT/SFT module. Only one copy of the Horizontal averaging filter is required since there is a decimation step following the filtering. Only even locations of both Cb/Cr are retained while the odd locations are decimated, however from design perspective the Hz Averager module works on Cb on even cycles and Cr on odd cycles.

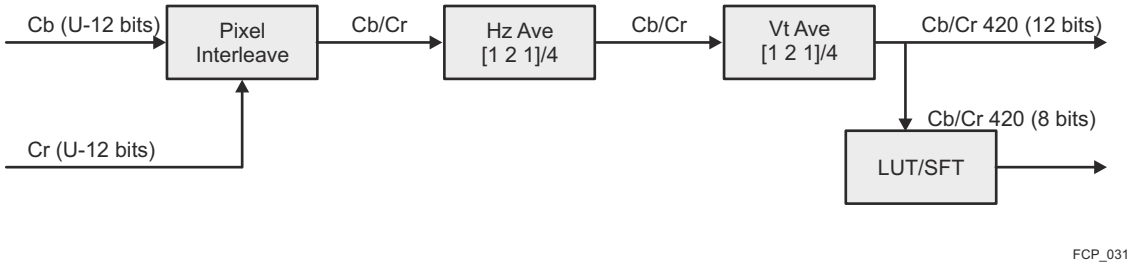


Figure 6-113. 444 to 420 Module

Additionally another mode of operation is to support planar YUV422 mode. In this case the chroma is only downsampled in the horizontal direction and the vertical averaging/downsampling is disabled.

The mode is referred to pseudo 422 since actual 422 mode requires the data to be interleaved between the Y and the chroma channels. In this case the interleaving of Y & C is handled by LSE whereas the Flexible CC module will only generate Y & Cb/Cr (interleaved) planes separately.

6.7.3.6.3.5 Blanking and Latency

Table 6-153 shows the blanking and latency functions in the FCP module.

Table 6-153. Blanking and Latency

Horizontal Blanking	Vertical Blanking	Horizontal Latency	Vertical Latency
10 cycles	0 line	25 cycles	1 lines – 420 Chroma 0 lines – Other Ch(s)

6.7.3.6.4 FCP Clocking

The FCP module has two clocks - CLK_A and VBUSP_CLK as it follows:

- CLK_A is 60 MHz.
- VBUSP_CLK must be greater than 30 MHz.

6.7.3.6.5 FCP Interrupts

The FCP module has three interrupts, as it follows:

- LUT_CFG_ERR - 16-12 LUT is written during active window
- CFA_PIX_ERR - Access to line memories by CFG during active window
- CFA_MMR_ERR - Non-Shadowed registers are written during active window

There are two additional events:

- SOL_EVENT - Start of line processing for Flexible CFA, triggered when the line enters the Flexible CFA
- SOF_EVENT - Start of frame processing for Flexible CFA, triggered when the frame enters the Flexible CFA

For information about the FCP interrupt events, see Section *VISS Top Level Functional Description*.

6.7.3.6.6 FCP Programmer's Guide

Copyright © 2024 Texas Instruments Incorporated

6.7.3.6.6.1 HWA Core Programming Details

The following fields should be programmed correctly for the FCP module to function.

- Input LUT
  - Use the LUT to reduce the bitwidth of the data to 12 bits so that it can pass through the Flexible CFA pipe.
  - If the GLBCE module is disabled, set the bit width and enable the LUT.
- Flexible CFA
  - Program the Flexible CFA kernels based on the input data pettern. Program only 3 channels for Bayer sensors and up to 4 for other non-standard formats like RGB-IR.
  - Set up the correct thresholds (use only one set for Bayer type sensors and two sets for non-standard formats like RGB-IR).
- Flexible CC
  - Set up the Flexible CC for either visual or analytics data flow.
    - For visual, program the CCM-1 to be used as RGB2RGB and Contrast to be used as Gamma with clip at 10 bits. Only the 8-bit outputs are used and the Y8/C8 data is generated using shift down from 10 to 8 (instead of LUT).
    - For analytics, CCM-1 is progrmmed depending on sensor input format and the hisogram is used to populate the contrast table. 12-bit outputs are used however Y8/C8 can be generated using 12 to 8 LUT based downshifting. The correct Saturation generation parameters should also be set.

6.7.3.6.6.2 HWA HTS Programming Details

For details of programming, see *HWA Thread Scheduler (HTS)* and Section 6.7.2, *VPAC Subsystem*.

6.7.3.6.6.3 HWA Data Transfer Programming Details

For details of programming for PARAM space, see Section *UDMA Specification*.

6.7.3.6.6.4 Initialization Sequence

For information, see Section 6.7.2, *VPAC Subsystem*.

6.7.3.6.6.5 Real-time Operating Requirements

For information, see Section 6.7.2, *VPAC Subsystem*.

6.7.3.6.6.6 Power Up/Down Sequence

For information, see Section 6.7.2, *VPAC Subsystem*.

Copyright © 2024 Texas Instruments Incorporated

6.7.3.7 VISS Edge Enhancer (EE)

This section describes the Edge Enhancer module.

6.7.3.7.1 Edge Enhancer Introduction

The Edge Enhancer module is used to enhance the visual quality of the image by increasing the sharpness of the image. The module only works on Y (Luma) data and uses a combination of 2D HPF (High-Pass Filter) filtering to detect edges and then apply them on the input image.

The Edge Enhancer consists of two separate blocks internally, the edge enhancer filter and the edge sharpener filter. The module can be programmed to either select the output of one of the filters or to use a blend of both.

Figure 6-114 below shows a conceptual block diagram of the edge enhancer filter. The input data is assumed to be a 12 bits/pixel. The EE module is tuned to process 8-bit images (MSB aligned) but has support to process 10-bit and 12-bit images as well.

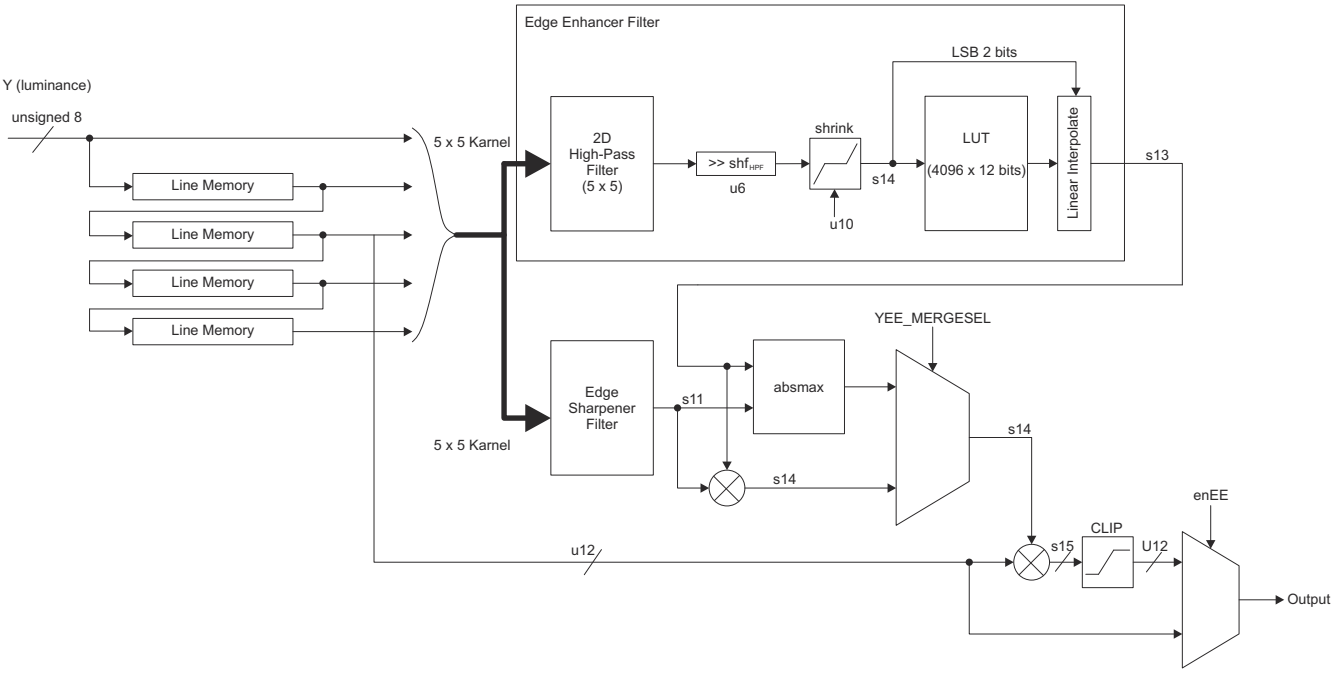


Figure 6-114. Edge Enhancer Block Diagram

6.7.3.7.1.1 Edge Enhancer Filter

The edge enhancer filter works on a 5 × 5 HPF filter (M) with programmable coefficients. The filter is symmetric as such only 3 × 3 coefficients are programmable.

Here, M is a 5 × 5 matrix with programmable coefficients specified by VISS_FCP_EE_YEE_COEF_Ri_Cj (i and j are 0, 1, or 2). The shift value (shf_{HPF}) is specified by VISS_FCP_EE_YEE_SHIFT[5-0] YEE_SHIFT.

Copyright © 2024 Texas Instruments Incorporated

$$HPF(h,v)=\left(\sum_{j=-2}^{2}\sum_{i=-2}^{2}M_{i,j}Y(h+i,v+j)\right)>>shf_{HPF}$$

$$M=\begin{pmatrix} M_{2,2} & M_{1,2} & M_{0,2} & M_{1,2} & M_{2,2} \\ M_{2,1} & M_{1,1} & M_{0,1} & M_{1,1} & M_{2,1} \\ M_{2,0} & M_{1,0} & M_{0,0} & M_{1,0} & M_{2,0} \\ M_{2,1} & M_{1,1} & M_{0,1} & M_{1,1} & M_{2,1} \\ M_{2,2} & M_{1,2} & M_{0,2} & M_{1,2} & M_{2,2} \end{pmatrix}$$

Figure 6-115. Edge Enhancer Linear Filter

The HPF value is shrink by a threshold value, $threshold_{HPF}$ (u6 = 10), specified by VISS_FCP_EE_YEE_E_THR register, and clipped to signed 14 bits to get the index for the LUT. The MSB 12 bits of the index are used for the LUT address (4k locations) whereas the LSB 2 bits acts as weight to the linear interpolation logic.

$$index=\text{clip}(\text{shrink}(HPF,threshold_{HPF}),-8k,8k)$$

$$\text{shrink}(x,threshold)=\begin{cases} x+threshold & x<-threshold \\ 0 & -threshold\le x\le threshold \\ x-threshold & threshold<x \end{cases}$$

$$\text{clip}(x,limit_{LOW},limit_{HIGH})=\begin{cases} -limit_{LOW} & x<-limit_{LOW} \\ x & -limit_{LOW}\le x\le limit_{HIGH} \\ limit_{HIGH} & limit_{HIGH}<x \end{cases}$$

Figure 6-116. Edge Enhancer Indexing

The edge enhancement intensity is looked up from the LUT, where A is the MSB 12 bits of the index.

$$E_{\text{int}}=Interpolate(LUT[A],LUT[A+1])$$

Figure 6-117. Edge Intensity LUT Formula

The 5 × 5 HPF filter is not fully programmable, rather only the top left quadrant is programmable coefficients. The rest of the kernel is symmetric along the y and the x axis.

Figure 6-118 below depicts how the kernels are implemented, with the kernel locations marked inside the boxes. The green color shows the kernels that are programmable.

Copyright © 2024 Texas Instruments Incorporated

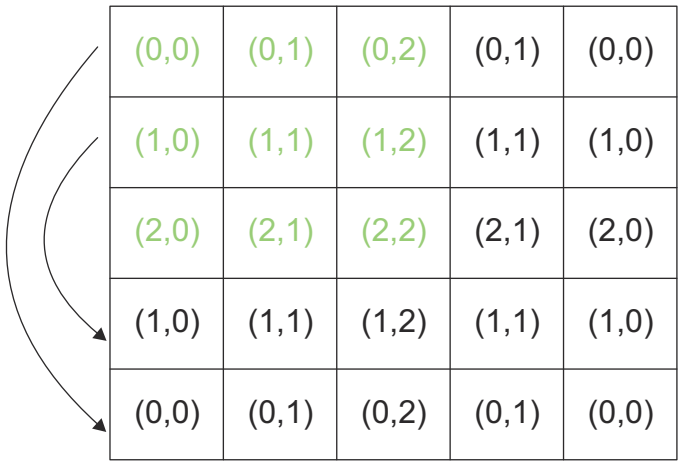


Figure 6-118. 5 × 5 HPF Filter

The border conditions are handled by data duplication on each of the borders when the desired pixel go out of range.

The output of the LUT is the final output of the edge enhancer filter.

6.7.3.7.1.2 Edge Sharpener Filter

Figure 6-119 below shows a high level block diagram of the edge sharpener function.

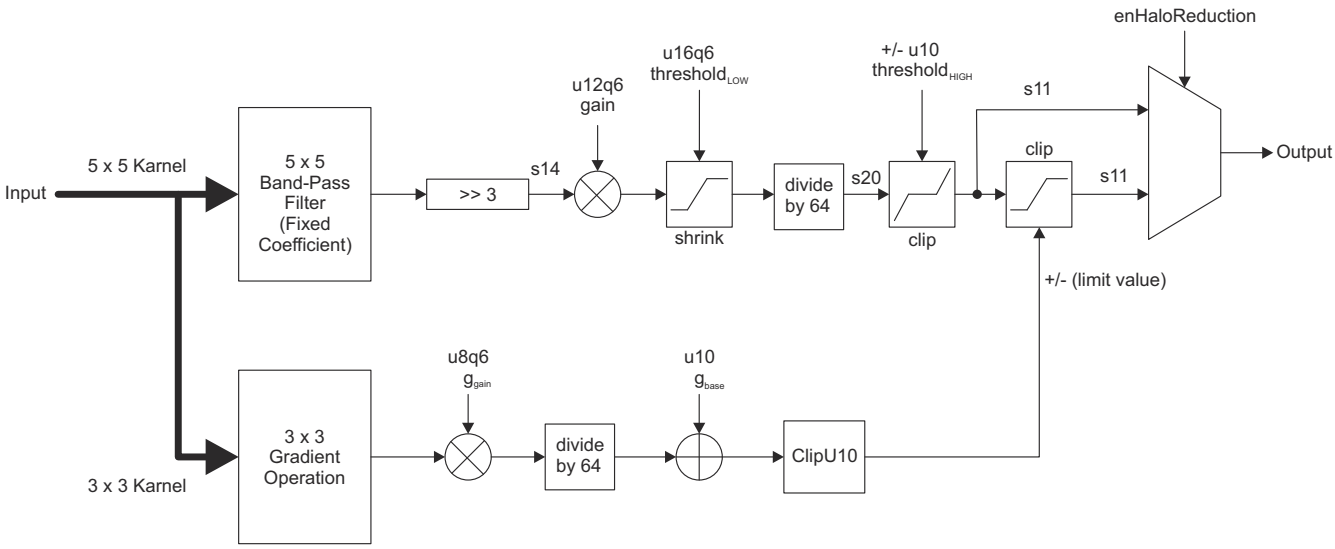


Figure 6-119. Edge Sharpener Function

In edge sharpener filter module, enabled when VISS_FCP_EE_YEE_MERGESEL[0] YEE_MERGESEL = 1, edge clarity is enhanced without producing a halo artifact. In this module, edge intensity is derived by the following 2D linear filter with fixed coefficients shown in Figure 6-120.

Copyright © 2024 Texas Instruments Incorporated

$$S_{i,j} = \begin{pmatrix} 0 & -1 & -2 & -1 & 0 \\ -1 & 0 & 2 & 0 & -1 \\ -2 & 2 & 8 & 2 & -2 \\ -1 & 0 & 2 & 0 & -1 \\ 0 & -1 & -2 & -1 & 0 \end{pmatrix}$$

$$sharpness(h,v) = \text{clip}\left(\text{shrink}\left(g \times \left(\left(\sum_{j=-2}^{2}\sum_{i=-2}^{2} S_{i,j} Y(h+i, v+j)\right) >> 3\right), threshold_{LOW}\right) >> 6, -threshold_{HIGH}, threshold_{HIGH}\right)$$

Figure 6-120. Edge Sharpener Details

The gain (*g*) and threshold values for shrink/clip function (*threshold* $_{LOW}$, *threshold* $_{HIGH}$) are determined by register values (VISS_FCP_EE_YES_E_GAIN, VISS_FCP_EE_YES_E_THR1, VISS_FCP_EE_YES_E_THR2). The precision of *g* is in U12Q6, *threshold* $_{LOW}$ is in U16Q6 and that of *threshold* $_{HIGH}$ is in U10.

This edge intensity is then clipped by a threshold value in the formula shown in Figure 6-121.

$$S_{int} = \begin{cases} clip(sharpness, -grad, grad) & \text{Halo reduction on} \\ sharpness & \text{Halo reduction off} \end{cases}$$

Figure 6-121. Edge Intensity Clipping Formula

The threshold value (grad) is a function of the activity around the target pixel, which is derived from gradient values. Gain (g_{grad}) and offset (g_{base}) are specified by VISS_FCP_EE_YES_G_GAIN and VISS_FCP_EE_YES_G_OFT.

$$grad = g_{grad}\left(\min(gx_L, gx_R) + \min(gy_T, gy_B)\right) >> 6 + g_{base}$$

$$gx_L = abs\left(\sum_{j=-1}^{1}\sum_{i=-1}^{1} G_{xL}(i,j) \cdot y(h+i, v+j)\right)$$

$$gx_R = abs\left(\sum_{j=-1}^{1}\sum_{i=-1}^{1} G_{xR}(i,j) \cdot y(h+i, v+j)\right)$$

$$gy_T = abs\left(\sum_{j=-1}^{1}\sum_{i=-1}^{1} G_{yT}(i,j) \cdot y(h+i, v+j)\right)$$

$$gy_B = abs\left(\sum_{j=-1}^{1}\sum_{i=-1}^{1} G_{yB}(i,j) \cdot y(h+i, v+j)\right)$$

$$GxL = \frac{1}{4}\begin{pmatrix} 1 & -1 & 0 \\ 2 & -2 & 0 \\ 1 & -1 & 0 \end{pmatrix}$$

$$GxR = \frac{1}{4}\begin{pmatrix} 0 & 1 & -1 \\ 0 & 2 & -2 \\ 0 & 1 & -1 \end{pmatrix}$$

$$GyT = \frac{1}{4}\begin{pmatrix} 1 & 2 & 1 \\ -1 & -2 & -1 \\ 0 & 0 & 0 \end{pmatrix}$$

$$GyB = \frac{1}{4}\begin{pmatrix} 0 & 0 & 0 \\ 1 & 2 & 1 \\ -1 & -2 & -1 \end{pmatrix}$$

Figure 6-122. Edge Threshold Value Formula

Capping with gradient value prevents overly enhancing edges, and suppresses halo artifacts around edges.

6.7.3.7.1.3 Merge Block

The output from edge enhancer filter and edge sharpener filter are merged with the following function.

Copyright © 2024 Texas Instruments Incorporated

$$E_{merge} = \begin{cases} E_{\text{int}} + S_{\text{int}} & VISS_FCP_EE_YEE_MERGESEL[0]YEE_MERGESEL = 0 \\ \text{absmax}(E_{\text{int}}, S_{\text{int}}) & VISS_FCP_EE_YEE_MERGESEL[0]YEE_MERGESEL = 1 \end{cases}$$

$$\text{absmax}(x, y) = \begin{cases} x & abs(y) \le abs(x) \\ y & otherwise \end{cases}$$

Figure 6-123. Edge Enhancer and Sharpener Merger Formula

The E_{merge} value is added to the *Y* input value to make the final output.

The contribution of each of the filters can be modified. The edge sharpener filter block can be turned off or reduced in strength using the gain fields. The edge enhancer filter block can be disabled using the LUT.

Copyright © 2024 Texas Instruments Incorporated

6.7.3.7.2 Edge Enhancer Programming Model

Table 6-154 captures the programming parameters and the programming model of the Edge Enahncer module.

Note

UMQN implies M bits with N bits of fraction and M-N bits of Integer.

For more information, see *Edge Enhancer Registers*.

Table 6-154. Edge Enhancer Programming Parameters

Parameter Name	New Precision	Tuning Methodology
VISS_FCP_EE_EE_CFG_0[12-0] WIDTH	13 bits	Width of the image
VISS_FCP_EE_EE_CFG_0[28-16] HEIGHT	13 bits	Height of the image
VISS_FCP_EE_EE_ENABLE[0] YEE_ENABLE	1	Enable Module
VISS_FCP_EE_YEE_SHIFT[5-0] YEE_SHIFT	U6	-
VISS_FCP_EE_YEE_COEF_R0/1/2_C0/1/2[9-0] YEE_COEF_R0/1/2_C0/1/2	Signed 10	-
VISS_FCP_EE_YEE_E_THR[9-0] YEE_E_THR	U10	Shrink Threshold before LUT, scale by 16×
VISS_FCP_EE_YEE_MERGESEL[0] YEE_MERGESEL	1 bit	Mergesel: Off(0), On(1)
VISS_FCP_EE_YES_E_HAL[0] YES_E_HAL	1 bit	Halo Reduction; Off(0)/ On(1)
VISS_FCP_EE_YES_G_GAIN[7-0] YES_G_GAIN	U8Q6	Usually 1.5 times YES_E_GAIN
VISS_FCP_EE_YES_G_OFT[9-0] YES_G_OFT	U10	Scale by 16
VISS_FCP_EE_YES_E_GAIN[11-0] YES_E_GAIN	U12Q6	Same as previous
VISS_FCP_EE_YES_E_THR1[15-0] YES_E_THR1	U16Q6	Scale by 16
VISS_FCP_EE_YES_E_THR2[9-0] YES_E_THR2	U10	Scale by 16

Copyright © 2024 Texas Instruments Incorporated

6.7.4 VPAC Lens Distortion Correction (LDC) Module

This section describes the Lens Distortion Correction (LDC) module.

6.7.4.1 LDC Overview

The LDC module deals with lens geometric distortion issues in the camera system. The distortion can be common optical distortions, such as barrel, pin-cushion, or fisheye distortion. LDC is not limited to just these types of distortions. Affine transformation support is also added to support the image and video stabilization application, where multiple images of the same scene need to be aligned. Perspective warp is an extension to affine transform and offers additional capabilities. Perspective transformations can align two images that are captured from different camera viewpoints or locations. This can also be used to rectify the left and right images of a stereo camera to reduce the complexity of a disparity computation. Perspective transformations can also generate new viewpoints.

6.7.4.1.1 LDC Features

- Autonomous memory-to-memory operation
  - Tile based processing
- Generic mesh based distortion model to correct multiple distortion types
- Perspective warp transformation for perspective correction; affine transform is a subset of perspective warp and supports scaling and rotation
- Support multiple input and output YUV data formats:
- Support up to 8192 x 8192 image dimension
- Pixel Interpolation
  - Bi-cubic interpolation for Y and bilinear interpolation for Cb/Cr
  - Bilinear interpolation mode to offer double throughput
- Performance
  - 1 cyc/pixel (bilinear)
  - 2 cyc/pixel (Bi-cubic)
- Support to process either Luma only or Chroma only modes in YUV420 input data format to save bandwidth
- Support for independent block size in multiple regions (up to 9 regions)
- Dual output channels for programmable output pixel size
- ECC support on Mesh Data internal storage memories and width conversion LUTs on dual output channel
- Interface to HTS module for block level synchronization with other IPs
- Circular addressing support for Output Write (for SL2 interface) and Input Read
- Event for Block/Frame completion and Error scenarios
- Multiple data formats support with flexible independent data format between Luma and Chroma
- Support for direct connection with other VPAC HWA along with hybrid addressing
  - Mix of circular and linear buffer on pixel data with support for initial line skip

6.7.4.2 LDC Functional Description

6.7.4.2.1 LDC Integration in VPAC

Figure 6-124 shows the LDC integration within the device

The list of hardware blocks in VPAC are as follow:

1. LDC (Lens distortion correction): LDC block reads data from memory (e.g. DDR or on-chip) and applies perspective transform as well as correction of lens distortion (including fisheye lenses). The output of LDC block can be sent to external memory (e.g. DDR) or sent to other hardware block (e.g. Scalar, Noise filter) for further pre-processing via local shared memory (SL2).
2. Scalar: Scalar block reads data from shared memory (SL2) and generates up to 10 scaled output from 2 inputs with various scaling ratios (between x and x0.5). The output of Scalar to Shared memory (SL2) can be further noise filtered using NF block or written to DDR.
3. Noise Filter (NF): NF block reads data from memory (e.g. DDR or on-chip) to shared memory (SL2) and does bilateral filtering to remove noise. The output of NF block can be sent to external memory (e.g. DDR) from Shared memory (SL2) or can be further re-sized using Scalar HW.
4. VISS (Vision ISS): There are two instance of on-the-fly processing for sensor related processing in VISS.

Copyright © 2024 Texas Instruments Incorporated

The list of infrastructure blocks in VPAC are following as follow:

1. Shared Level 2 (SL2) memory: This is used to exchange data across HW IP block (e.g. LDC, Scalar and NF) as well as to DMA Engine (e.g. UDMA).
2. HTS (Hardware Thread Scheduler): HTS is used for IPC communication among various HW IP Blocks (e.g. LDC, Scalar and NF as well as DMA Engine (e.g. UDMA)). Message Manager shown in Figure 1 is implemented as HTS.

VPAC has dedicated UDMA for DMA transfers except for LDC Read Data. For more top level integration and functional details see Section 6.7.2, *VPAC Subsystem* for details.

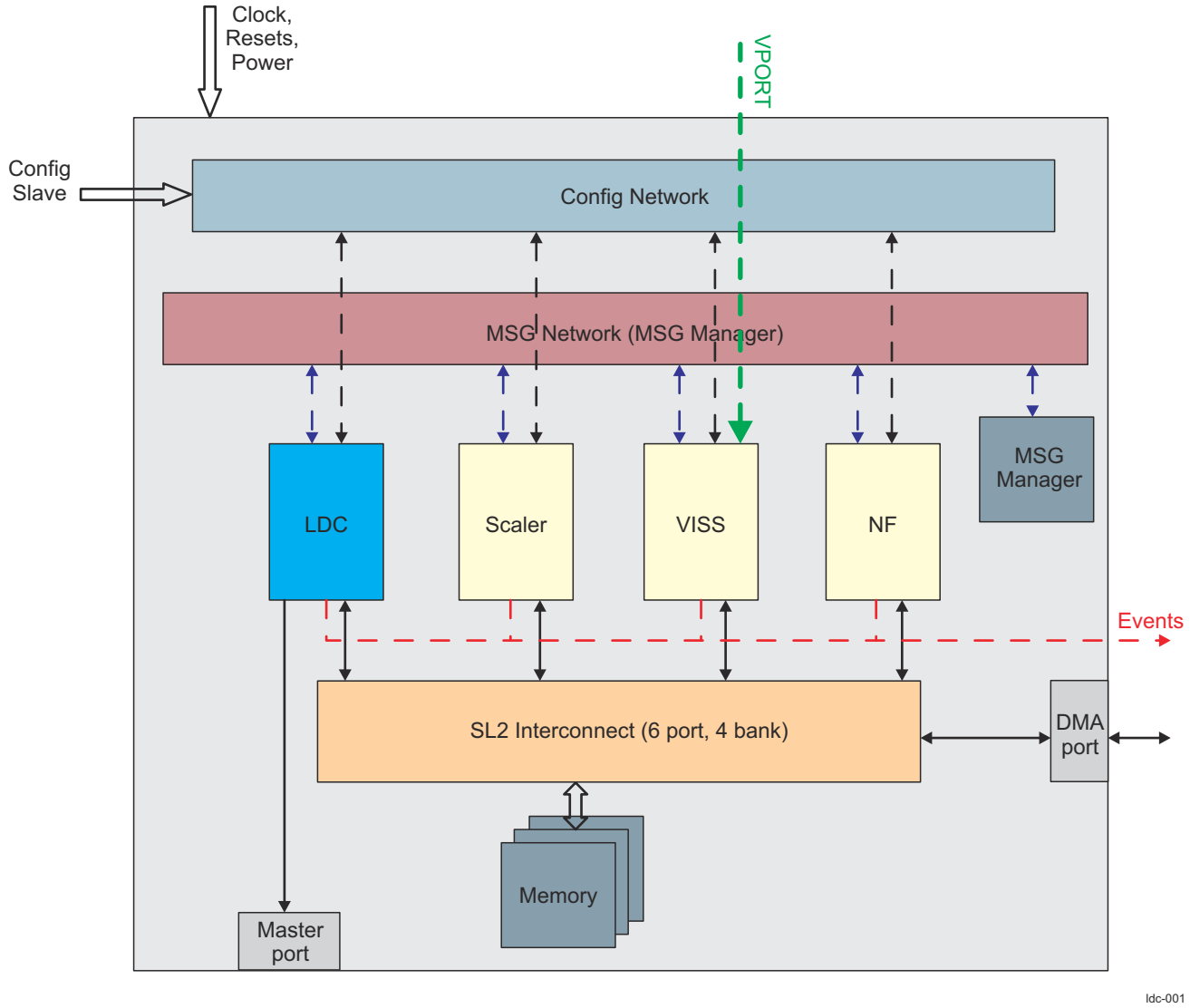


Figure 6-124. LDC Integration in VPAC

6.7.4.2.2 LDC Block Diagram

Most digital cameras suffer from some degree of nonlinear geometric distortion. A spatial transformation is required to correct the distortion. In automotive applications, cameras use wide angle lenses, including fisheye lenses to provide 180+° field of view. To visually present the scene to the user in an easy-to-consume representation, these distortions need to be corrected

Back mapping gives coordinates of the distorted image as a function of coordinates of the undistorted output image. Correction involves back-mapping each output pixel to a location in the source distorted image, and thus

Copyright © 2024 Texas Instruments Incorporated

the corrected image is fully populated. As the distorted pixel locations mostly fall onto fractional coordinates, correction involves interpolation.

As shown in Figure 6-125, the LDC consists of a back mapping block, a x/y offset table, image buffer interface, buffer, an interpolation block, and SL2 interface.

Figure 6-125 is a block diagram of the LDC.

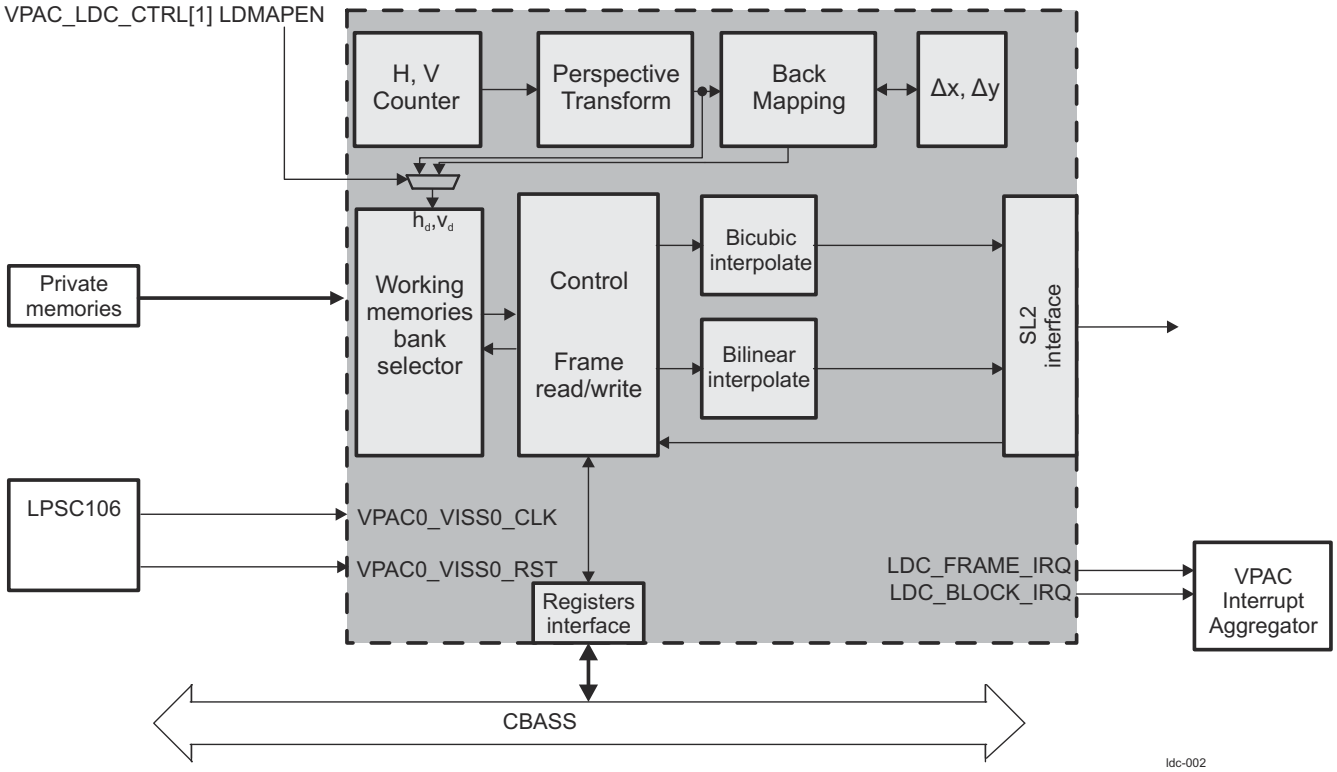


Figure 6-125. LDC Block Diagram

Given the coordinates of the undistorted image, the corresponding coordinates of the distorted image are calculated by combining the output coordinates and the offsets from the offset table. Distorted pixels are read from the image buffer, and buffered for the bilinear interpolation. After the interpolation, corrected image is written back to the SL2 memory.

The LDC processes the image in small two-dimensional (2D) blocks. The software configures appropriate parameters, then initiates the LDC function by writing to an LDC register. The LDC controls the sequencing through 2D blocks, DMA transfers, and computation to process an entire image autonomously. Interrupt, if enabled, is asserted at the completion of the image.

The LDC write is controlled by HTS at block level.

To start the LDC, software must set the VPAC_LDC_CTRL[0] LDC_EN bit to 1. The VPAC_LDC_CTRL[2] BUSY bit is a status that reflects LDC activity.

6.7.4.2.3 LDC Clocks

Complete VPAC LDC operates on single clock (that is, VPAC0_LDC0_CLK). Except the signals that are coming from LPSC, all other LDC sub-block clock domains are synchronous to VPAC0_LDC0_CLK.

6.7.4.2.4 LDC Interrupts

LDC generates interrupt events and these are made available to the VPAC. VPAC implements compatible interrupt function via the interrupt aggregator, see chapter Section 6.7.2 *VPAC Subsystem*.

Copyright © 2024 Texas Instruments Incorporated

All the interrupts are active high and pulsed for one VPAC0_VISS0_CLK cycle. Following interrupts events are generated. Out of these only ECC interrupt should be treated as real interrupt and others are used as events for the interrupt aggregator at VPAC level.

Table 6-155. LDC Interrupt Events

Interrupt Event	Type	Category	Description
PIX_IBLK_OUTOFBOUND	pulse	func	PIX_IBLK_OUTOFBOUND pulse interrupt event
MESH_IBLK_OUTOFBOUND	pulse	func	MESH_IBLK_OUTOFBOUND pulse interrupt event
IFR_OUTOFBOUND	pulse	func	IFR_OUTOFBOUND pulse interrupt event
INT_SZOVF	pulse	func	INT_SZOVF pulse interrupt event
VPAC_LDC_FR_DONE_EVT	pulse	func	LSE frame done event
VPAC_LDC_SL2_WR_ERR	pulse	func	LSE SL2 VBUSM Write Error interrupt event
PIX_IBLK_MEMOVF	pulse	func	Input pixel block internal memory overflow event
MESH_IBLK_MEMOVF	pulse	func	Input mesh block internal memory overflow event
VPAC_LDC_VBUSM_RD_ERR	pulse	func	Error event on VBUSM Read Interface
UNCORR_PULSE	pulse	error	ECC Aggregator uncorrectable error, pulse
UNCORR_LEVEL	level	error	ECC Aggregator uncorrectable error, level
CORR_PULSE	pulse	error	ECC Aggregator correctable error, pulse
CORR_LEVEL	level	error	ECC Aggregator correctable error, level

6.7.4.2.4.1 LDC Interrupt Events Description

6.7.4.2.4.1.1 PIX_IBLK_OUTOFBOUND

Generated when back mapping of block non-corner pixel co-ordinate goes out of the pre-computed input bounding box (i.e. data required for processing is not available in private pixel storage). This event is generated for the first occurrence of error in each frame. Upon this error, PIX_PAD setting needs to be reviewed. Upon pixel out of bound, pixel co-ordinate will be clipped to the pre-fetched block boundary. Usually no special treatment is required on this error event. Only few pixels are expected to be have this condition..

6.7.4.2.4.1.2 MESH_IBLK_OUTOFBOUND

Generated when mesh data required for non-corner mesh co-ordinate goes out of the pre-computed mesh bounding box (i.e. mesh data required for processing is not available in internal mesh storage). This event is generated for the first occurrence of error in each frame. This error condition is handled inside design by clipping the mesh data location to the pre-fetched block boundary. No special treatment is expected on this error event.

6.7.4.2.4.1.3 IFR_OUTOFBOUND

Generated when back mapped input pixel co-ordinate goes out of valid input frame boundary or co-ordinates post Perspective warping goes out of valid Mesh frame size (Frame size for which mesh data is available). This event is generated for the first occurrence of error in each frame. In the event of error condition, design will clip the co-ordinates to valid frame boundary. It is recommended to review the source of the error to make sure that it is expected.

6.7.4.2.4.1.4 INT_SZOVF

Generated when intermediate variables of affine or perspective transform operation goes above what hardware supports (U16Q3). This event is generated for the first occurrence of error in each frame. In the event of error condition, design will clip the variables the size to what HW supports. Upon this error, Affine and Perspective coefficients need to be reviewed.

Copyright © 2024 Texas Instruments Incorporated

6.7.4.2.4.1.5 VPAC_LDC_FR_DONE_EVT

Generated on once complete output frame is written into SL2.

6.7.4.2.4.1.6 VPAC_LDC_SL2_WR_ERR

This event is generated whenever the LDC SL2 Output VBUSM Interface write status is something other than 0 (success).

6.7.4.2.4.1.7 PIX_IBLK_MEMOVF

Generated when input pixel block memory storage requirement is more than internal pixel memory or each side of input block size goes above 1023. This event will be generated once per block in case of error. In the event of error condition, one or more output pixel blocks corruption. Upon this error, Output Block size setting needs to be reviewed.

6.7.4.2.4.1.8 MESH_IBLK_MEMOVF

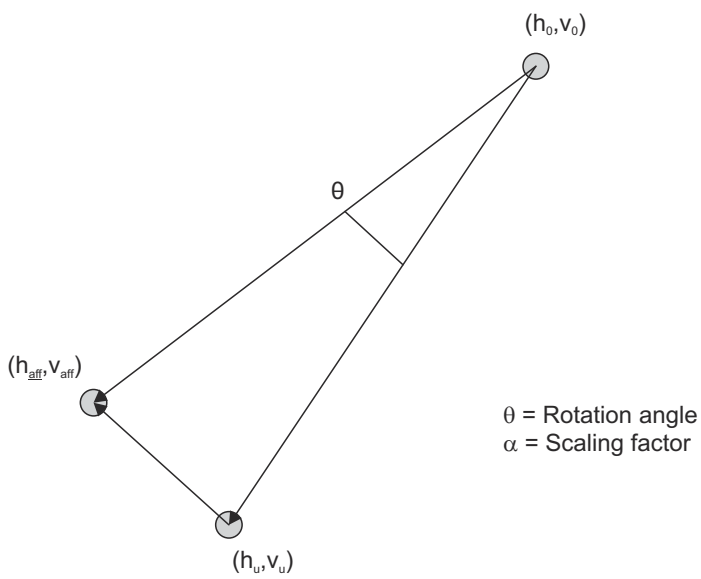
Generated when Mesh block internal storage requirement is more than internal Mesh memory or either side of input block size goes above 1010. This event will be generated once per block in case of error. In the event of error condition, one or more output pixel blocks corruption. Upon this error, Output Block size or Affine/Pwarp parameters or mesh sub-sampling factor setting needs to be reviewed.

6.7.4.2.4.1.9 VPAC_LDC_VBUSM_RD_ERR

This event is generated whenever the LDC Read VBUSM Interface status is something other than 0 (success).

6.7.4.2.5 LDC Affine Transform

The LDC first performs an affine transform, which can be used for rotations and scaling, as shown in Figure 6-126. The output of the pixel coordinate is used as the input address to the affine transform.



$$\begin{bmatrix} h_{aff} \\ v_{aff} \end{bmatrix} = \begin{bmatrix} a \times \cos(\theta) & a \times \sin(\theta) \\ -a \times \sin(\theta) & a \times \cos(\theta) \end{bmatrix} \begin{bmatrix} h_u \\ v_u \end{bmatrix} + \begin{bmatrix} h0 - h0 \times a \times \cos(\theta) - v0 \times a \times \sin(\theta) \\ v0 + h0 \times a \times \cos(\theta) - v0 \times a \times \sin(\theta) \end{bmatrix}$$

ldc-010

Figure 6-126. LDC Affine Transformation

The mapping from destination coordinate to the source coordinate is expressed as:

$h_{aff} = a \times h_u + b \times v_u + c$

$v_{aff} = d \times h_u + e \times v_u + f$

Copyright © 2024 Texas Instruments Incorporated

This transform allows full rotation, expansion, contraction, and translation. The parameters {a, b, d, e} are in S16Q12 (signed number on 14 bits with 2 bits for integer and 12 bits of fraction) format, and parameters {c, f} are in S16Q3 format (signed number on 16 bits with 13 bits for integer and 3 bits of fraction). These parameters must be set in following bit fields:

- a = VPAC_LDC_AFF_AB[15:0] A
- b = VPAC_LDC_AFF_AB[31:16] B
- c = VPAC_LDC_AFF_CD[15:0] C
- d = VPAC_LDC_AFF_CD[31:16] D
- e = VPAC_LDC_AFF_EF[15:0] E
- f = VPAC_LDC_AFF_EF[31:16] F

6.7.4.2.6 LDC Perspective Transformation

When a camera is viewing a scene from two different positions or when multiple cameras are viewing the scene from different positions, a transformation between the two viewing angles is needed to align the images. Under specific conditions, the class of geometric transformations known as homography, or planar-perspective transformation, will capture the geometric relationship between the images accurately. Common applications of homography transforms are to align (or stitch) multiple frames of the same scene to compute a panoramic output image. A second application is the alignment of planar surfaces in the world. Finally, perspective transforms are also useful in computing depth maps from a stereo image pair. By rectifying the two views, the search to compute disparity between the two views is simplified to a 1-D search problem. The homography is defined by a 3x3 transformation matrix, as in

$$h_{aff} = a * h_u + b * v_u + c \quad (3)$$

$$v_{aff} = d * h_u + e * v_u + f \quad (4)$$

$$z = \max(0, g * h_u + h * v_u + 1) \quad (5)$$

$$h_p = h_{aff} / z \quad (6)$$

$$v_p = v_{aff} / z. \quad (7)$$

The affine transform is a subset of the perspective transformation. By setting g = h = 0, $h_p = h_{aff}$ and $v_p = v_{aff}$.

In image alignment applications, the homography matrices are computed by locating corresponding points in the two frames and estimating the matrix parameters to transform the set of points in one frame onto the corresponding points in the second frame. In the stereo rectification application, the matrix is determined (pre-computed) at the calibration step and remains fixed.

When LDC is requested to provide a change in the frame size between the input and output, the affine/perspective parameters must be programmed to implement the coordinate scaling (see for programming details). In the following sections, the mesh table is used to program distortion correction. The table sizes are defined based on the total output frame size.

6.7.4.2.7 LDC Lens Distortion Back Mapping

In YCbCr mode, the offset table defines a (x,y) vector for a regular grid of output points. The grid can be fully sampled or down sampled. A fully sampled grid will define an offset vector for every output pixel, defining exactly where to fetch the input data to compute the output pixel. This is the most precise definition and can capture rapidly changing offset tables. The drawback is that it will require a large amount of memory bandwidth as the LDC engine will be reading offset values for every output pixel. Since most offset tables are not expected to change rapidly in a small spatial region, LDC supports reading a subsampled offset table. Offset tables can be subsampled by powers of two in both horizontal and vertical directions and the subsampling factor is set in the register, VPAC_LDC_MESHTABLE_CFG[2-0] M. This mode conserves memory bandwidth by reducing the amount of data read to describe the offset vectors, but requires more hardware to interpolate the missing offset vectors. LDC supports bilinear interpolation to interpolate missing offset vectors.

The mapping procedure is described by the following series of equations. Given an output pixel at location, we compute the input pixel location.

Copyright © 2024 Texas Instruments Incorporated

x_c=CLIP(x_o,0,8×W-1), W: Mesh Frame Width (8)

y_c=CLIP(y_o,0,8×H-1), H: Mesh Frame Height (9)

Mask=(1 << (3+M)) -1 (10)

i_{x0} = (x_c >> (3 + M)), f_x=x_c & Mask (11)

i_{y0} = (y_c >> (3 + M)), f_y=y_c & Mask (12)

Clip new locations to previously fetched mesh block boundaries.

i_x=CLIP(i_{x0},block_startmx,block_endmx-1) (13)

i_y=CLIP(i_{y0},block_startmy,block_endmy-1) (14)

Copyright © 2024 Texas Instruments Incorporated

$$\mathrm{Ofst0} = \mathrm{READ32}\left(\mathrm{OfstTable}\left[i_x,\ i_y\right]\right)$$
$$\Delta x_0 = \mathrm{MSB16}(\mathrm{Ofst0})$$
$$\Delta y_0 = \mathrm{LSB16}(\mathrm{Ofst0})$$

$$\mathrm{Ofst1} = \mathrm{READ32}\left(\mathrm{OfstTable}\left[i_x + 1,\ i_y\right]\right)$$
$$\Delta x_1 = \mathrm{MSB16}(\mathrm{Ofst1})$$
$$\Delta y_1 = \mathrm{LSB16}(\mathrm{Ofst1})$$

$$\mathrm{Ofst2} = \mathrm{READ32}\left(\mathrm{OfstTable}\left[i_x,\ i_y + 1\right]\right)$$
$$\Delta x_2 = \mathrm{MSB16}(\mathrm{Ofst2})$$
$$\Delta y_2 = \mathrm{LSB16}(\mathrm{Ofst2})$$

$$\mathrm{Ofst3} = \mathrm{READ32}\left(\mathrm{OfstTable}\left[i_x + 1,\ i_y + 1\right]\right)$$
$$\Delta x_3 = \mathrm{MSB16}(\mathrm{Ofst3})$$
$$\Delta y_3 = \mathrm{LSB16}(\mathrm{Ofst3})$$
$$T_{x0} = \left(\left(1 \ll (M + 3)\right) - f_x\right) \cdot \Delta x_0 + f_x \cdot \Delta x_1$$
$$T_{x1} = \left(\left(1 \ll (M + 3)\right) - f_x\right) \cdot \Delta x_2 + f_x \cdot \Delta x_3$$

$$T_{y0} = \left(\left(1 \ll (M + 3)\right) - f_x\right) \cdot \Delta y_0 + f_x \cdot \Delta y_1$$
$$T_{y1} = \left(\left(1 \ll (M + 3)\right) - f_x\right) \cdot \Delta y_2 + f_x \cdot \Delta y_3$$

$$\Delta x = \left(\left(\left(1 \ll (M + 3)\right) - f_y\right) \cdot T_{x0} + f_y \cdot T_{x1} + (1 \ll 2M + 5)\right) \gg (2M + 6)$$
$$\Delta y = \left(\left(\left(1 \ll (M + 3)\right) - f_y\right) \cdot T_{y0} + f_y \cdot T_{y1} + (1 \ll 2M + 5)\right) \gg (2M + 6)$$

$$x_i = x_o + \Delta x$$
$$y_i = y_o + \Delta y$$

$$x_o, y_o:\ \mathrm{U16Q3}$$
$$x_i, y_i:\ \mathrm{U16Q3}$$
$$\Delta x, \Delta y:\ \mathrm{S16Q3}$$

ldc-023

Figure 6-127. LDC Lens Distortion Back Mapping Offset Calculation

Graphically, this procedure is shown in Figure 6-128.

Copyright © 2024 Texas Instruments Incorporated

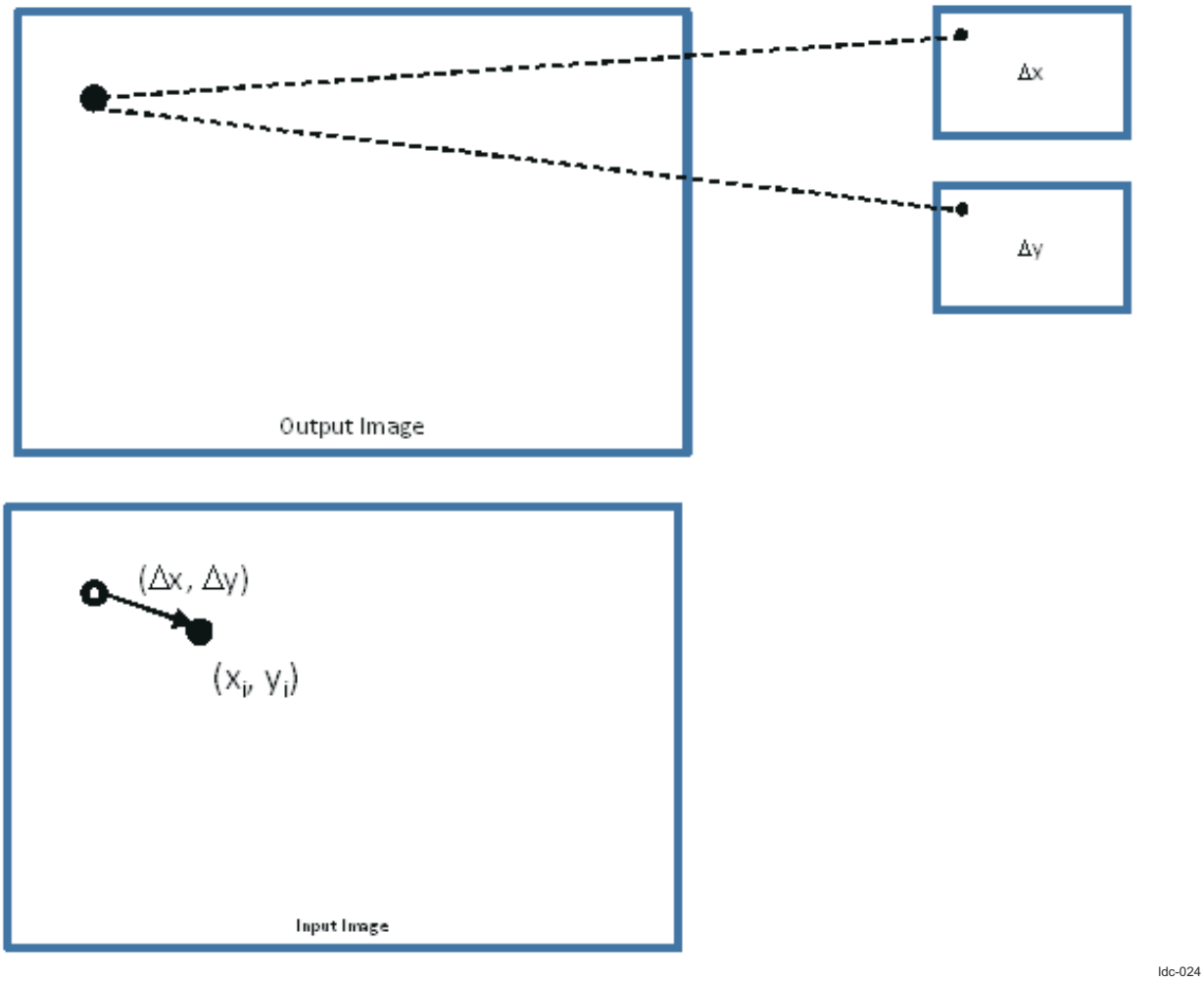


Figure 6-128. LDC Back Mapping Procedure Using Offset Table

The input coordinate (x_i,y_i) refers to final input frame co-ordinates and is used to fetch the appropriate input pixels for interpolation. The interpolation procedure is described in . Before fetching data(x_i,y_i), is clipped to the input frame boundary.

6.7.4.2.7.1 LDC Mesh Table Storage Format

The size of the mesh table is dependent on the Total output frame size. The size must be:

$$\text{Table Width}=\text{CEIL}((\text{Mesh Frame Width})/2^M)+1 \quad (15)$$

$$\text{Table Height}=\text{CEIL}((\text{Mesh Frame Height})/2^M)+1. \quad (16)$$

Mesh frame width and height are specified by registers: VPAC_LDC_MESH_FRSZ[13-0] W and VPAC_LDC_MESH_FRSZ[29-16] H. The downsampling factor, M, is specified by register VPAC_LDC_MESHTABLE_CFG[2] M.

The mesh table should be stored with delta(x) and delta(y) offsets interleaved and stored in raster order. Each offset is S16Q3, so a table entry is 32-bits, including both delta(x) and delta(y). The location of the mesh table is configured by registers VPAC_LDC_MESH_BASE_H and VPAC_LDC_MESH_BASE_I. The buffer width (>= Table Width), is provided by VPAC_LDC_MESH_OFST[15-0] OFST. The mesh table must be aligned on a 16-byte boundary.

.

Copyright © 2024 Texas Instruments Incorporated

6.7.4.2.8 LDC Pixel Interpolation

In UYVY and NV12 mode, the pixel interpolation type can be set to bi-cubic interpolation for best quality or bilinear interpolation for faster performance.

As the coordinates (h_d,v_d) calculated by the back-mapping function are not generally integer values, bi-cubic or bilinear interpolation is applied to the distorted pixels.

Depending on register configuration, bi-cubic or bilinear interpolation is used to interpolate the output Y pixels:

- VPAC_LDC_CFG[6] YINT_TYP = 0, bi-cubic interpolation for Y, bilinear for other components
- VPAC_LDC_CFG[6] YINT_TYP = 1, bi-linear interpolation for all components

In the case of bi-cubic interpolation, the distorted pixel is interpolated from the 16 Y pixels in the 4 x 4 grid around the distorted location, as shown in Figure 6-129. Bi-cubic interpolation is used first along the horizontal direction, then the vertical direction.

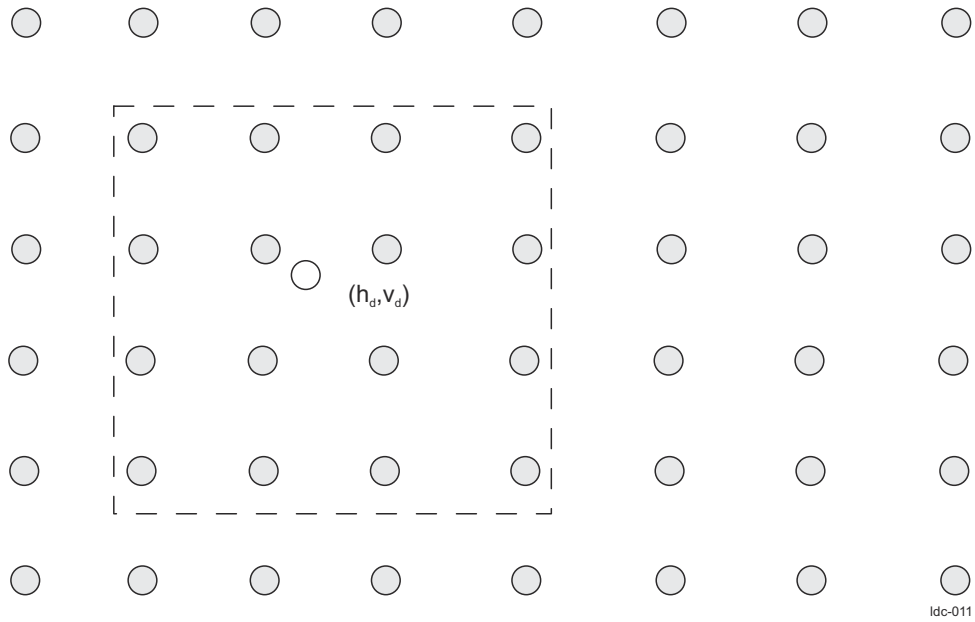


Figure 6-129. LDC Bi-cubic Interpolation for Y

If bi-linear interpolation is selected, the distorted pixel is interpolated from the four Y pixels in the 2 x 2 grid around the distorted location, as shown in Figure 6-130.

Copyright © 2024 Texas Instruments Incorporated

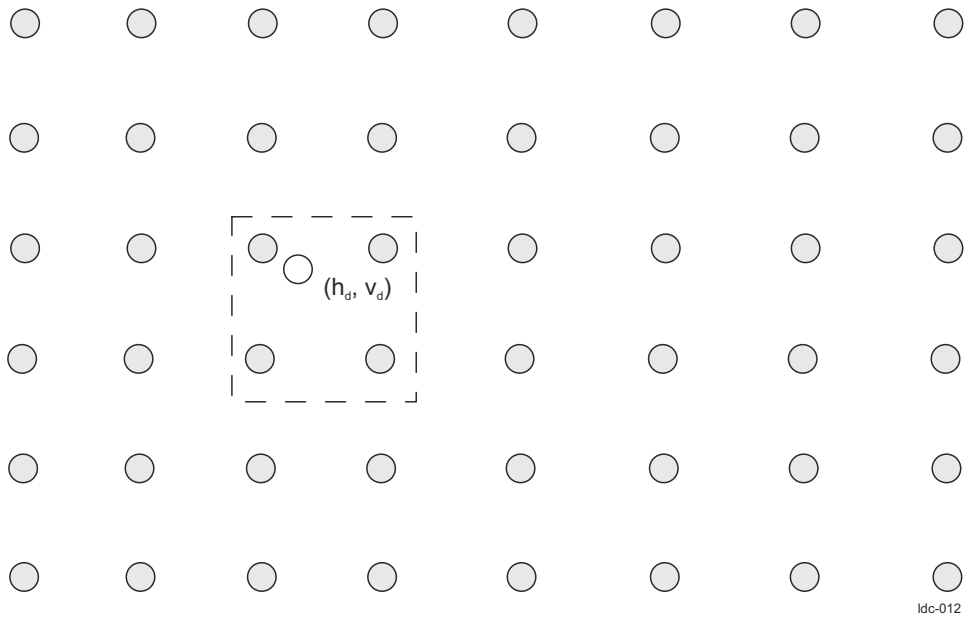


Figure 6-130. LDC Bi-linear Interpolation for Y

For Cb and Cr components, simple bilinear interpolation is used. Each distorted pixel is interpolated from the four same-color pixels on the 2 x 2 grid around the distorted location. For UYVY data, the Cb/Cr grid is not square, but is 2x wider compared to the height. This is shown in Figure 6-131.

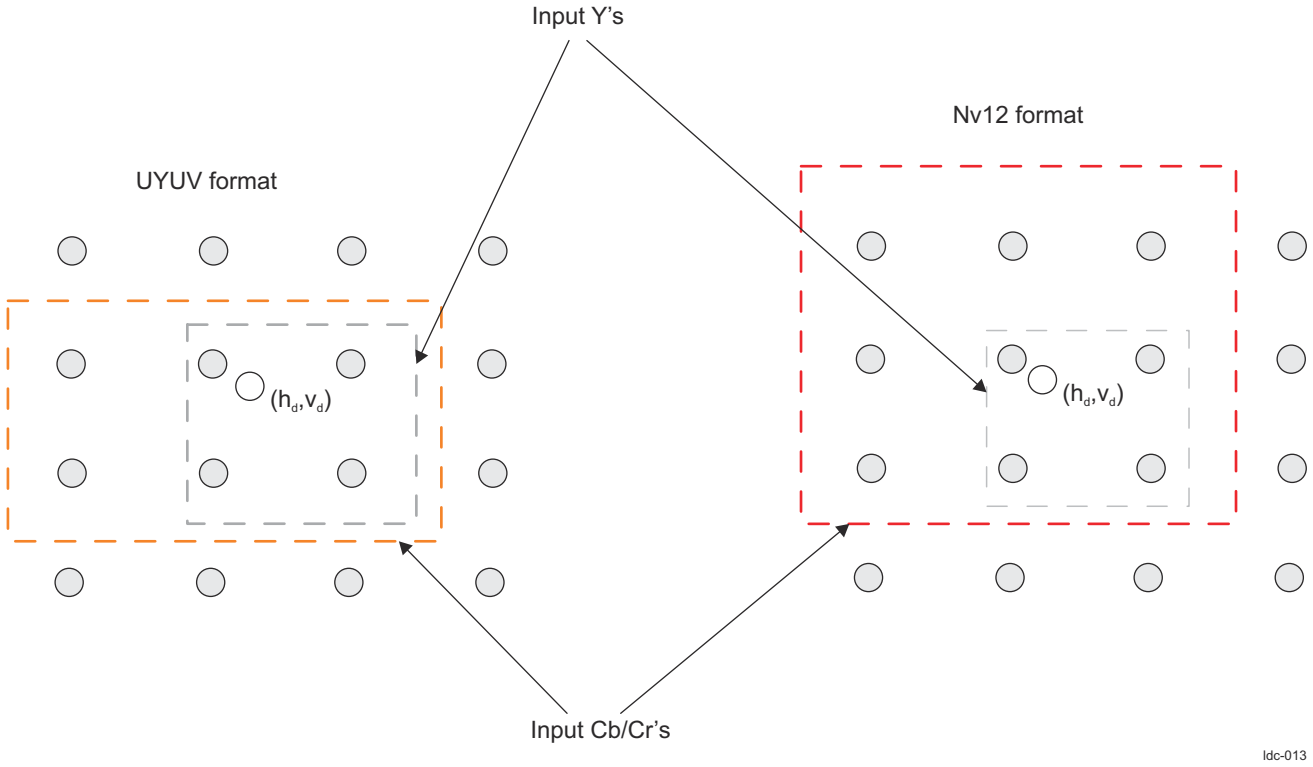


Figure 6-131. LDC Bilinear Interpolation for CB/Cr in UYVY and NV12 Format

6.7.4.2.9 LDC Buffer Management

Nonlinear nature of the Back Mapping suggests 2-D processing to make more efficient use of external memory bandwidth. Since the mapping computation is non-trivial, the hardware requires the software to provide a number of data management parameters to assist hardware with data transfers.

Copyright © 2024 Texas Instruments Incorporated

6.7.4.2.9.1 LDC Buffer Management

Once mesh block fetch is completed, final input co-ordinates are calculated by applying back mapping on previously calculated perspective warp corner pixel co-ordinates. Additional padding is applied on top of these back mapped corner co-ordinates based on the interpolation type.

The LDC processes a block of VPAC_LDC_OUT_BLKSZ[7:0] OBW x VPAC_LDC_OUT_BLKSZ[15:8] OBH (output block width by output block height) pixels at a time in a raster-scan order throughout the image. OBW is constrained to be a multiple of a base width, depending on the operating mode. This is described in Table 6-156. OBH should be even. The LDC uses four corners of the output block plus some padding to fetch the input block. Due to extreme scenarios, there will be scenarios where above bounding box calculated by hardware doesn't cover all the input data required to generate particular output block. To cover those cases, software needs to supply additional PixelPad, the amount of padding in input block in all directions. Software must supply OBW, OBH, and VPAC_LDC_OUT_BLKSZ[19:16] PIXPAD, the amount of padding in input pixels. Software must guarantee that, for EVERY output pixel in the OBW x OBH output block, the input pixels required are bounded by back mapping of the four corners plus/minus the padding. More precisely, the input block is determined by:

- IBX_start = min(truncate(distortx(corner1)), truncate(distortx(corner3))) - HwPad - PixelPad
- IBX_end = max(truncate(distortx(corner2)), truncate(distortx(corner4))) + HwPad + PixelPad
- IBY_start = min(truncate(distorty(corner1)), truncate(distorty(corner2))) - HwPad - PixelPad
- IBY_end = maxn(truncate(distorty(corner3)), truncate(distorty(corner4))) + HwPad + PixelPad

where corner1, corner2, corner3, and corner4 are upper-left, upper-right, lower-left, and lower-right corners of the OBW x OBH output block, and distortx(.), distorty(.) are X and Y distorted coordinates of the corners. Distorted coordinates computed by the back-mapping process described in Section 6.7.4.2.7, *LDC Lens Distortion Back-Mapping*. Figure 6-132 shows the input block bound calculation. Table 6-156 lists the OBW requirements.

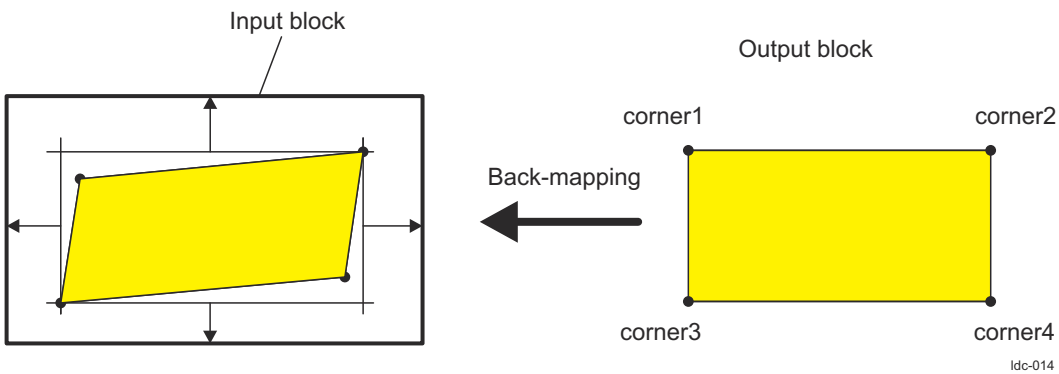


Figure 6-132. LDC Input Block Bound

Table 6-156. LDC OBW Requirements

Data Format	OBW must be a multiple of
YUV422	8
YUV420	8

Typically for geometric distortion, the LDC_BLOCK[19:16] PIXPAD bit field must be 4 to accommodate neighbor sets for all colors. Software must set the PIXPAD bit field correctly.

OBH and OBW must be large enough for efficient operation of the lens distortion operation. As shown in Table 6-156, OBW is constrained to ensure efficient external memory write. Another constraint is that input block size, (IBX_end – IBX_start + 1) x (IBY_end - IBY_start + 1), for EVERY input block of the image, must fit the allocated input buffer. These parameters must not be specified pessimistically (OBH, OBW too small, or PixelPad too large). Pessimistic parameter setting degrades performance and incurs unnecessary external memory transfer.

The LDC can process a portion of the image, rather than the entire image. This saves time by letting an image process through multiple software/LDC interactions to correct only a portion of the image (See Multi-pass Frame processing).

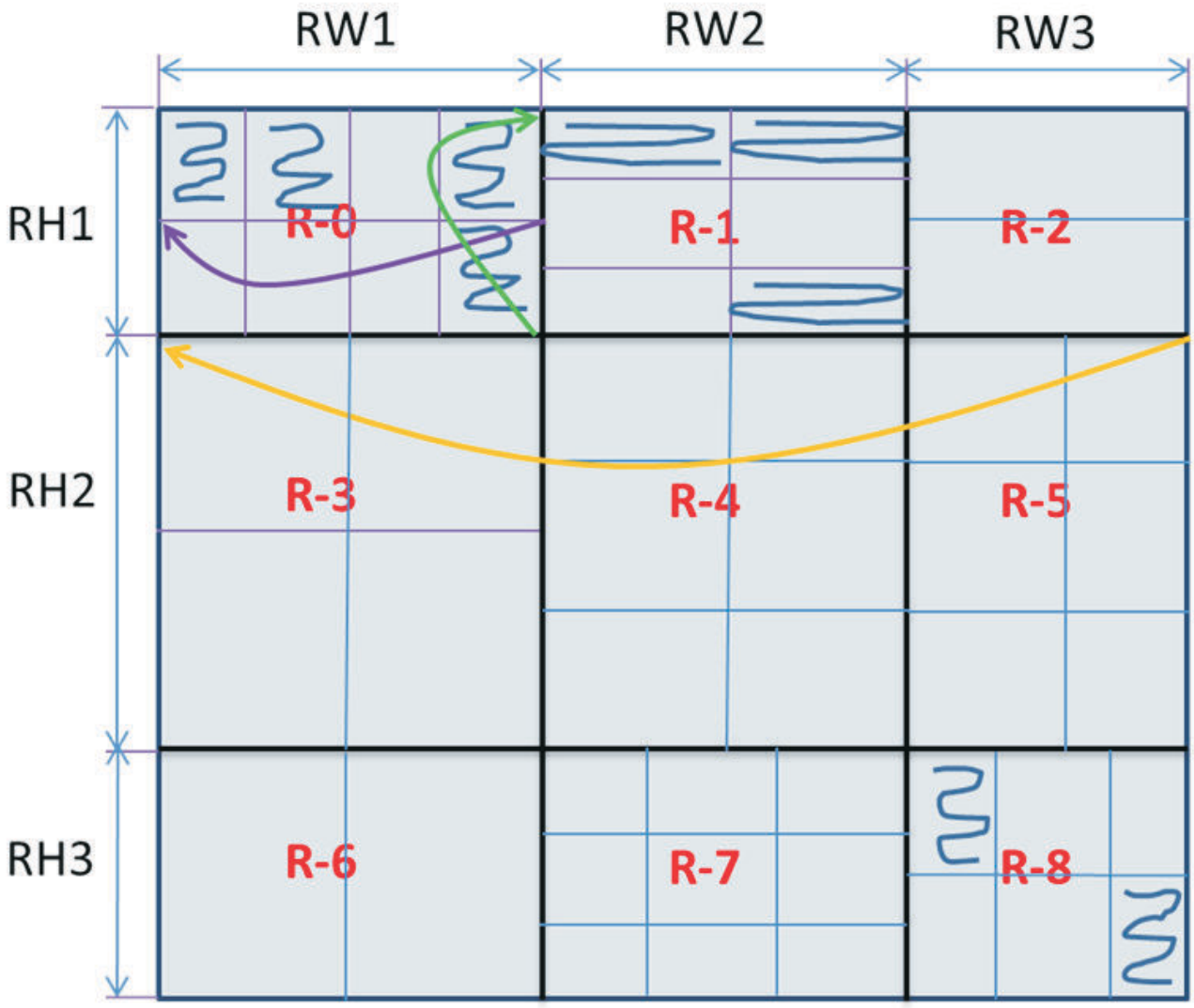
Copyright © 2024 Texas Instruments Incorporated

An intermediate event, *hts_tdone*, is also provided on completion of each macro block output write. This allows the LDC operation to be pipelined with other tasks. The LDC output write stall after this event, waiting for a pulse on the hts_tstart signal to begin writing the next macro block.

6.7.4.2.10 LDC Multi Region with Variable Block size

Distortion in wide angle lenses causes output block size to be very small due to extreme scaling in particular regions of the Image. This could be caused by high field-of-view or viewing position. The range of “Ratio of Output block to input pixel block size” is very high across the frame.

In order to solve this, LDC supports dividing the frame into multiple regions and output block size can be programmed independently for each region. Frame can be divided up to 9 regions. Frame can be divided into maximum of three slices horizontally and maximum of three slices vertically as shown in Figure 6-133. All the regions in a vertical slice share the common horizontal start location and all the regions in a horizontal slice share the common vertical start location. Width of vertical slices can be configured using VPAC_LDC_REGN_W12_SZ[13-0] W1, VPAC_LDC_REGN_W12_SZ[29-16] W2 and PAC_LDC_REGN_W3_SZ[13-0] W3 register fields. Height of horizontal slices can be configured using VPAC_LDC_REGN_H12_SZ[13-0] H1, VPAC_LDC_REGN_H12_SZ[29-16] H2 and VPAC_LDC_REGN_H3_SZ[13-0] H3 register fields.



ldc-026

Figure 6-133. LDC Variable Block Size with Multiple Regions

Copyright © 2024 Texas Instruments Incorporated

Hardware will process the each region as independent frame, hence processing of the pixels will in 2D sequence with in the region. Up on completing processing of one region, it will process next region in raster direction.

6.7.4.2.10.1 LDC Region Skip Feature

In surround view use case, some regions of the frame won't be used for stitching of final frame. By programming VPAC_LDC_REGION_CTRL_j[0] ENABLE to low, hardware will skip the processing of particular region.

Note

Hardware has to make sure that FRAME_DONE information to LSE is generated on last pixel of last valid region.

As an example shown in Figure 6-134, Region-5, 7 and 8 are skipped for processing. FRAME_DONE is passed on last pixel of Region-6. Ordering and numbering of the regions will remain the same even in the case of skipping some regions.

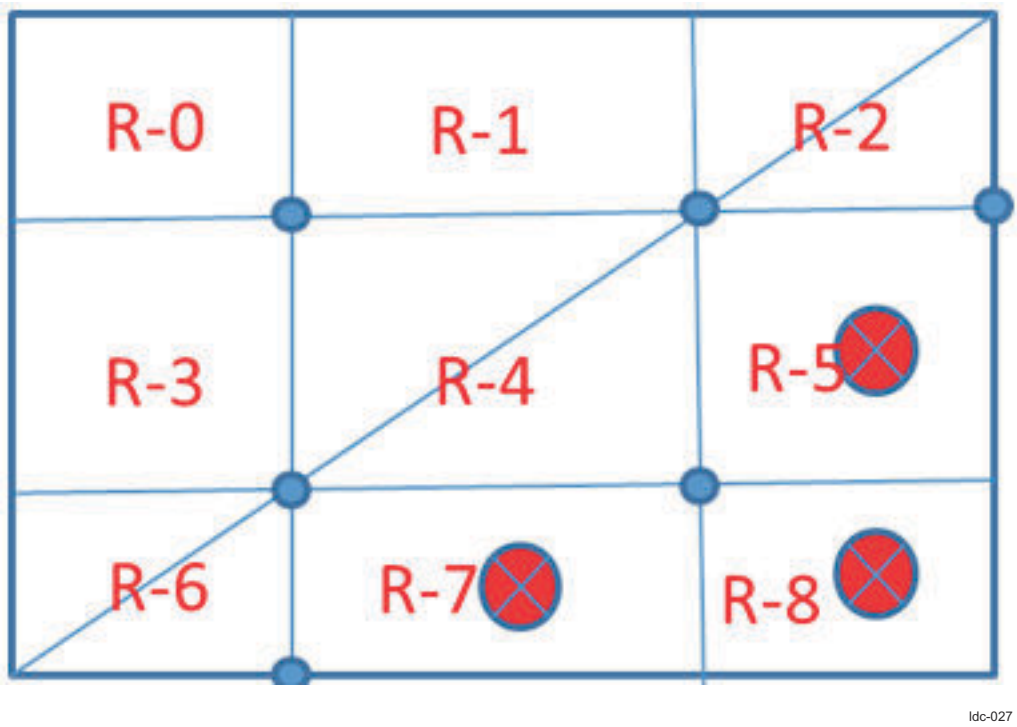


Figure 6-134. LDC Region Skipping Example

6.7.4.2.10.2 LDC Support for sub-set of 3x3 regions

HW supports dividing the frame into sub-set of 3x3 regions. Following combinations of regions are supported.

- 3 vertical slices
  - RW1 = x , RW2 = y and RW3 = z
- 2 vertical slices
  - RW1 = x , RW2 = y and RW3 = 0
  - Last region width has to be zero
- 1 vertical slice
  - RW1 = x , RW2 = 0 and RW3 = 0
  - Last two regions width has to be zero
- 3 horizontal slices
  - RH1 = x , RH2 = y and RH3 = z
- 2 horizontal slices
  - RH1 = x , RH2 = y and RH3 = 0

Copyright © 2024 Texas Instruments Incorporated

  - Last region height has to be zero
- 1 horizontal slice
  - RH1 = x , RH2 = 0 and RH3 = 0
  - Last two regions height has to be zero

As shown in Figure 6-135, 2x2 region partitioning can be done by programming RW3=RH3=0.

Note

Region numbering remains as in 3x3 region partitioning with R3, R6-R8 being null regions.

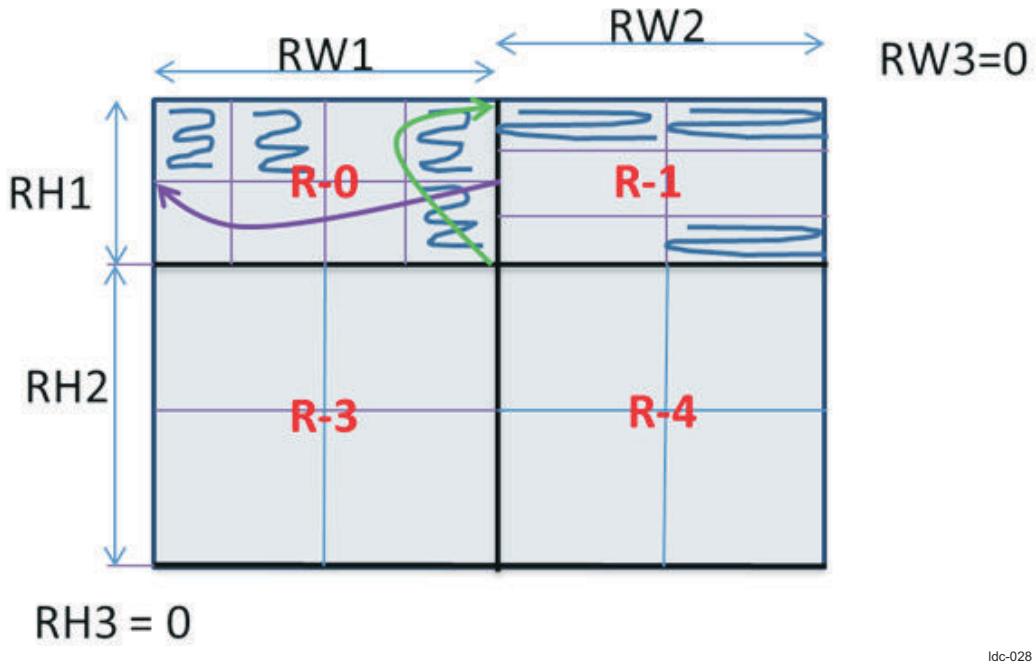


Figure 6-135. LDC 2x2 Region Partitioning Example

6.7.4.2.10.3 LDC Limitations of Multi Region Scheme

Following features can't be supported when multi region processing is enabled.

- Circular buffer support can't be enabled on Input data fetch
- HTS synchronization can't be enabled on Input data fetch
- LDC can't be joined with other HWAs in the VPAC. LDC output needs to be written to external memory.
- HTS can't make use of multiple BPR buffers available in SL2 circular buffer

6.7.4.2.10.4 LDC Multi Region Block Constrains

- RW1+RW2+RW3 must be equal to OFW
- RH1+RH2+RH3 must be equal to OFH
- SW to program the LSE buffer Stride, at-least the size of largest output block width among all regions.
- SW to program buffer height at-least twice the maximum block height among all regions

6.7.4.2.11 LDC Multi-pass Frame processing

The LDC can process a portion of the image, rather than the entire image. This saves time by letting an image process through multiple software/LDC interactions to correct only a portion of the image.

An intermediate interrupt can occur after processing each macro block. This lets the LDC operation be pipelined with other tasks.

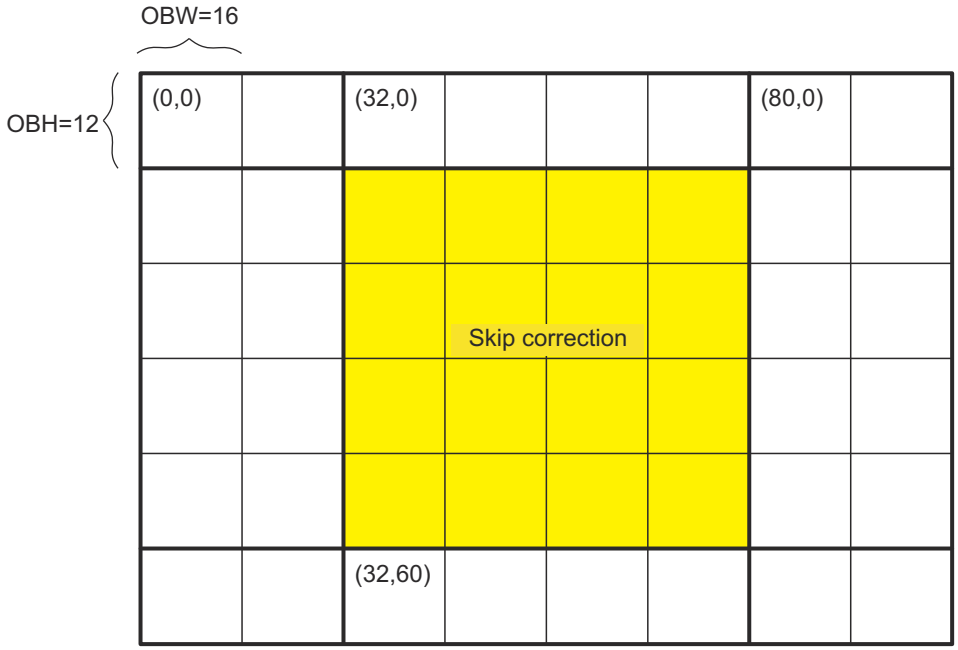
To process only a portion of the image, the following parameters are required:

- VPAC_LDC_INITXY[12:0] INITX: X coordinate of upper-left corner of output frame
- VPAC_LDC_INITXY[28:16] INITY: Y coordinate of upper-left corner of output frame

Copyright © 2024 Texas Instruments Incorporated

- VPAC_LDC_INPUT_FRSZ[13:0] W: Width of output frame
- VPAC_LDC_INPUT_FRSZ[29:16] H: Height of output frame
- FrameBase SDRAM address of upper-left corner of output frame (Y and Cb/Cr base needed in case of YCbCr420)
- FrameOfst SDRAM frame width (in bytes)

Figure 6-136 is an example of multiple-pass processing with middle-of-the-image skipped.



INITX	INITY	W	H	WBASE
0	0	32	72	0x10000
32	0	48	12	0x10040
80	0	32	72	0x100A0
32	60	48	12	0x134C0

ldc-015

Figure 6-136. LDC Multiple-Pass Correction Example

The LDC does not copy skipped blocks from the input frame to the output frame (necessary task unless output frame = input frame); software must set up and initiate this memory copy.

A note on in-place operation to conserve external memory: Depending on the offset table and configuration parameters, it may not be feasible to point the output image at the input image and have corrected pixels overwriting the source image. Software must determine whether it is feasible to overwrite, and allocate a separate frame buffer when it is not feasible. The LDC does not care if the input and output image pointers coincide, nor does it check any dependency violations.

The starting address of the input frame, corresponding to input coordinate (0, 0), is specified in the LDC_RD_BASE[31:0] RBASE bit field and offset in the LDC_RD_OFST[15:0] ROFST bit field ((Y and Cb/Cr base required for YCbCr4:2:0: LDC_420C_RD_BASE[31:0] RBASE)). The LDC does not clips input block to input frame size if any of the input block falls outside the input frame.

Copyright © 2024 Texas Instruments Incorporated

Note

- Any number of input blocks CAN fall partially or totally outside the input frame by assuming coordinates that are outside valid range (negative or exceeding maximum image width defined by the line offset).
- If any output pixel refers to any input pixel outside the input frame, the computed value for that output pixel would be same as neighbor pixel which falls into valid input frame. The other output pixels can still have correct outcomes.

6.7.4.2.12 LDC Input/Output Data Formats

Table 6-157 captures valid number of LDC input and output data mode/format combinations.

Table 6-157. LDC Supported Data Format Combinations

Input		Output	
Data Mode	Data format	Data Mode	Data format
420 420_Y 420_UV	8-bit	Same as input	8bit
			12-bit - packed
	12-bit - packed		12-bit - packed
	12-bit - unpacked		12-bit - unpacked
			8-bit (on channel[2/3] using LUT)
	12-bit - packed		8-bit (on channel[0/1] with shift) 8-bit (on channel[2/3] using LUT)
422	8-bit	420	8-bit
			12-bit – packed
		422	8-bit

Output data mode will be same as input data mode except when input data is 422, output data can also be converted to 420.

The following list contains the register fields used to configure data modes and formats:

- Input data mode - VPAC_LDC_CTRL[6-3] IP_DATAMODE
- Input data format - VPAC_LDC_CTRL[8-7] IP_DFMT
- Input data independent control for Chroma VPAC_LDC_CTR[16].CH_CHANCTRL_EN
- Input Chroma data format - VPAC_LDC_CTRL[18-17] CH_IP_DFMT
- Output Luma data format – VPAC_LDC_LSE_BUF_CFG_j (j = 0, 2) PIX_FMT_PW
- Output Chroma data format – VPAC_LDC_LSE_BUF_CFG_j (j = 1, 3) PIX_FMT_PW

The sample grid for YCbCr 422_UYVY is shown in Figure 6-137.

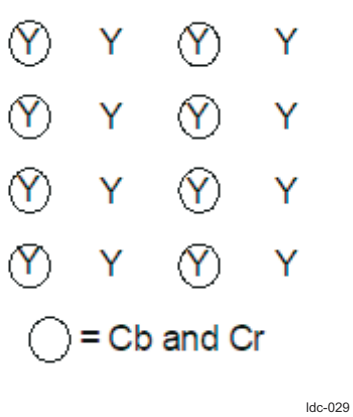


Figure 6-137. LDC Sample Grids for UYVY

UYVY data are stored in 8-bit per color components as shown in Table 6-158.

Copyright © 2024 Texas Instruments Incorporated

Table 6-158. LDC UYVY 8-bit Per Color Components

31			0
Y(1)	Cr(0)	Y(0)	Cb(0)

The sample grid for YUV420 NV12 is shown in Figure 6-138.

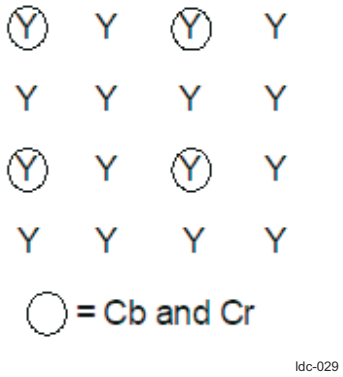


Figure 6-138. LDC Sample Grids for NV12

YCbCr 420 data are stored 8-bit per color components, and in two planes, Y by itself and Cb/Cr interleaved as shown in Table 6-159

Table 6-159. LDC YCbCr 420 data stored 8-bit per color components

31			0
(Y plane) Y(3)	Y(2)	Y(1)	Y(0)
(Cb/Cr plane) Cr(2)	Cb(2)	Cr(0)	Cb(0)

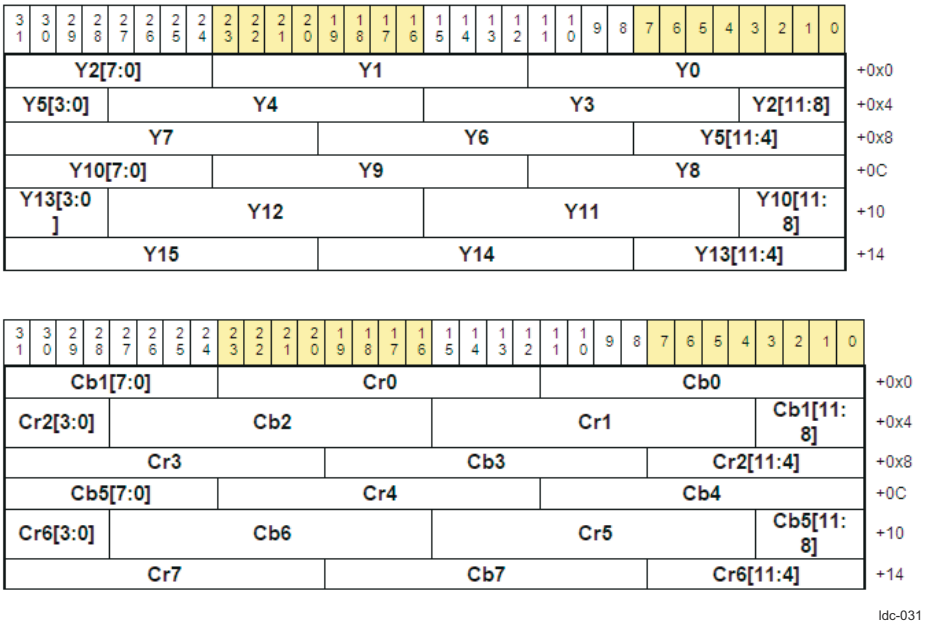


Figure 6-139. LDC YUV420 (2-plane 12-bit Fully Packed Format)

6.7.4.2.13 LDC YUV422 to YUV420 Conversion

Supporting input YUV422 format enables external sensors providing the YUV data directly. YUV422 to YUV420 conversion is done after pixel interpolation and achieved by dropping odd Chroma lines in the block. This retains the Chroma co-sited relation with Luma as at LDC input.

Copyright © 2024 Texas Instruments Incorporated

Figure 6-140. LDC YUV422 to YUV420 Conversion

6.7.4.2.14 Independent Channel Control

Some of the applications require high bit depth (up to 12-bit) for Luma but 8-bit is sufficient for Chroma data. This can be achieved by configuring independent Chroma data format parameter. Independent parameters are enabled by setting *CTRL.CH_CHANCTRL_EN.* This can be enabled when Luma and Chroma are stored in separate buffer (i.e. all the data modes except *422_UYVY* mode).

When *CTRL.CH_CHANCTRL_EN* is high, following parameters will take effect and all parameters should be programmed even if they have same value as Luma counterpart.

- CTRL.CH_IP_DFMT – Chroma Data format
- CH_RD_OFST.OFST – Chroma buffer Read Offset (aka Stride)
- CH_RD_OFST.MOD – Chroma Circular Buffer Height
- HYBD_CHCBUFF_PARAM – Chroma Hybrid addressing parameters

Following Input data control parameters are common between Luma and Chroma.

- CTRL.HYBD_ADDREN – Input Hybrid addressing Enable
- CTRL.IP_CIRCEN – Input Circular buffer Enable (CBuff height is independent)
- IBUF_PIX_START – Input buffer Start Co-ordinates

In 422 and 420 modes, Chroma is sub-sampled compared to Luma. In RGB data, all three color components are present at full resolution (i.e. no sub-sampling). To process RGB 2 new data modes are introduced. RGB data can be processed *Y1_Y2* data mode as 2 pass solution; with 2 of 3 RGB color components processed in Y1_Y2 mode and other one in 420_Y mode. In Y1_Y2 data mode, Y1 and Y2 stored in separate buffers. RGB can be processed in single pass in Y1_Y2Y3 mode with Y1 in one buffer and interleaved Y2Y3 separate buffer.

6.7.4.2.15 LDC SL2 Interface (LSE)

LDC Output write interface is handled by LSE sub-module. LSE (Load Store Engine) module is a common component integrated in VPAC to perform data load and store tasks on the SL2 memory for the HWA algorithm core.

VBUSP interface is be used to transfer the data from LDC Core to LSE.

For more LSE details, see *Load Store Engine (LSE)*.

6.7.4.2.15.1 LDC PSA (Parallel Signature Analysis)

LSE integrates a CRC signature capture mechanism on each output channel data to verify the integrity of the HWA's internal paths. When enabled, every valid pixel data from the CORE is sampled by a CRC module to update the signature for the channel before the data is packed and sent to the SL2 buffer. Note that data used for PSA is raw data from the CORE rather than data written into SL2 with container alignment. For LDC, PSA signature is calculated in the same order as CORE data transfer to LSE which is 3D block order as against frame order. At the end of frame condition, the final signature is saved in a separate read-only signature register (VPAC_LDC_LSE_PSA_SIGNATURE_y y=0..#_of_Output Channel-1) till the next end of frame condition for software to read and compare it against the golden reference signature. The read-only register keeps the last saved value until a reset (set to 0) or is replaced by another frame data signature.

Copyright © 2024 Texas Instruments Incorporated

6.7.4.2.16 LDC LUT Mapped Dual Output

LDC produces parallel 8-bit and 12-bit outputs to support surround view and ADAS applications concurrently. LDC produces parallel 8-bit output from 12-bit input using LUT based implementation. This 8-bit output is written into SL2 using LSE output channels (channel[2] for Luma and channel[3] for Chroma). Though use case requires 8-bit output, hardware is designed to be generic which supports any 8-12bit to 8-12bit conversion.

12 or 8 bit generated by data interpolation block is mapped to 8 to 12 bits using a LUT based mapping. The LUT is similar to the LUTs used in VISS and is a linear LUT with 513 entries.

The LUTs are sized to provide 513 locations of data, to enable linear interpolation for all locations. The internal weights are sized as 3 bits since the maximum delta between 2 steps is only 4095/512 = 8. The logic is designed to scale for bit width and can support anything from 8 to 12 bits of input to support different use cases. However, since the step size is different depending on the input bit width, the bit selection and weight calculation logic is designed to account for that. LSE Shift operation can be instead of LUT to reduce bit width from 12 down to 8.

The input bit width (IN_BITDEPTH explained above) is specified using a dedicated register. Output is clipped to programmable clip value ($2^{BitClip\ -1}$). Combination of IN_BITDEPTH, LUT values and Clip value programation allows any 8-12bit to 8-12bit conversion.

6.7.4.2.17 LDC Band Width Controller

A Bandwidth Limiter is required to limit the mean BW that LDC can request over the VBUSM Read interface. The software sets the maximum allowed bytes per cycle in the register, VPAC_LDC_VBUSMR_CFG[27-16] BW_CTRL.

To determine the maximum allowed bytes per cycle, the software needs to know the configuration of LDC, which determines the maximum input block size. Given a specific configuration, the ldc_findMaxInputBuffer utility provided with the functional C model, can be used to give the maximum input block size. The performance requirements determine the maximum number of cycles allowed per block.

6.7.4.2.18 LDC Input Block Fetch Limit

In order to limit the band width hogging by LDC in case of wrong mesh table data, maximum amount of input block pixel data being fetched will be capped at maximum internal memory available for respective data.

In YUV420* modes, limit is imposed by Luma data. Pixel data fetch is stopped on detecting the overflow condition and interrupt event is generated. Maximum of 2 requests can be issued upon overflow detection. In case of mesh data, only interrupt event is generated on detecting overflow condition and there is no stalling of fetch. These events are generated for the first overflow case per frame.

6.7.4.2.19 LDC HTS Interface

The HTS interface is used to synchronize between blocks in VPAC Subsystem. In the case of LDC, it controls the start of LDC operation by issuing hts_init. LDC will be waiting for hts_init once it is enabled, so LDC is expected to be enabled before issuing hts_init.

HTS also controls the LDC output write which will indirectly control all LDC operations (that is, fetch and processing). LDC will start writing output pixel data when only it receives hts_tstart for that particular block. Once all output data is written, LDC will generate hts_tdone (also hts_eop if it is last block of the frame).

6.7.4.2.20 LDC VBUSM Read Interface

VBUSM Read interface has following features

- Data Width – 128 Bit
- Address width – 48 Bit
- Maximum of number of Tags – 32
  - Design can be constrained to use lesser number of Tags
- Maximum Command Size: Programmable
  - 32/64/128/256 Bytes
  - Command will split at maximum command size address boundary
- Chanid Encode: chanid[1:0] identify the source/type of the data interface is requesting. Chanid[11:2] is tied low.

Copyright © 2024 Texas Instruments Incorporated

- Chanid[1:0] decoding
  - 00 – Mesh Data.
  - 01 – Luma data
  - 10 – Chroma data

6.7.4.2.21 Partial Input Frame Storage

LDC can be used to generate images in variety of views. Depending on the use case, some parts of the input image are not used. In order to reduce on-chip memory size requirement in circular buffer mode, only required part of the image is stored. Starting co-ordinate of the stored frame can be configured in *IBUF_PIX_START* register. Default values on of (*StartX,StartY*) are zero making backward compatible with previous version (J7ES PG1).

Using programmable StartY, Circular buffer start can be aligned with image stored in the local memory. Due to 16-byte alignment requirement on Base address, following restrictions apply on StartX.

- 420/422SP/Y1_Y2/Y1_Y2Y3 mode
  - 8-bit format – Multiple of 16
  - 12-bit unpack fomat – Multiple of 8
  - 12-bit packed format – Multiple of 32
- 422_UYVY mode – Multiple of 8
  - 8-bit format – Multiple of 8
  - 12-bit unpack format – Multiple of 4
  - 12-bit packed format – Multiple of 16

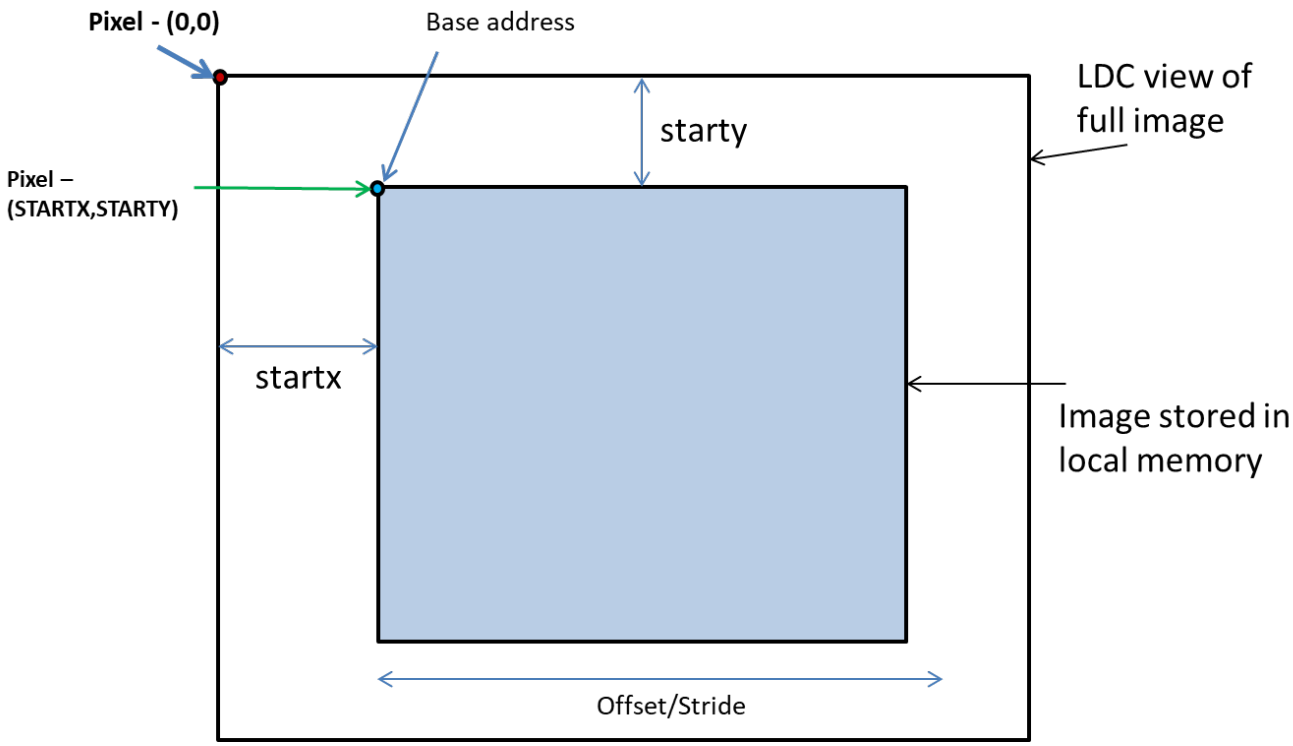


Figure 6-141. Partial Image Buffer View of LDC

6.7.4.2.22 Hybrid Addressing

Circular buffer feature enables other VPAC HWAs feeding data directly to LDC. In this case, input data to LDC is stored on-chip to save BW primarily and latency from DDR. However, in some use cases, on-chip memory allocation won't be sufficient to store required circular buffer. By storing extreme distortion regions in DDR, on-chip circular buffer size requirement can be reduced. This requires hybrid addressing support where some part of image is stored in linear addressing buffer and rest is stored in circular buffer. Hybrid addressing can be enabled by setting *CTRL.HYBD_ADDREN.* Programmable initial and end range of the image can be stored in

Copyright © 2024 Texas Instruments Incorporated

linear addressing buffers. Start line of circular buffer is configured by *HYBD_CBUFF_PARAM.STARTLINE* and End line by *HYBD_CBUFF_PARAM.ENDLINE.* Image outside of this range is stored in 2 separate linear buffers.

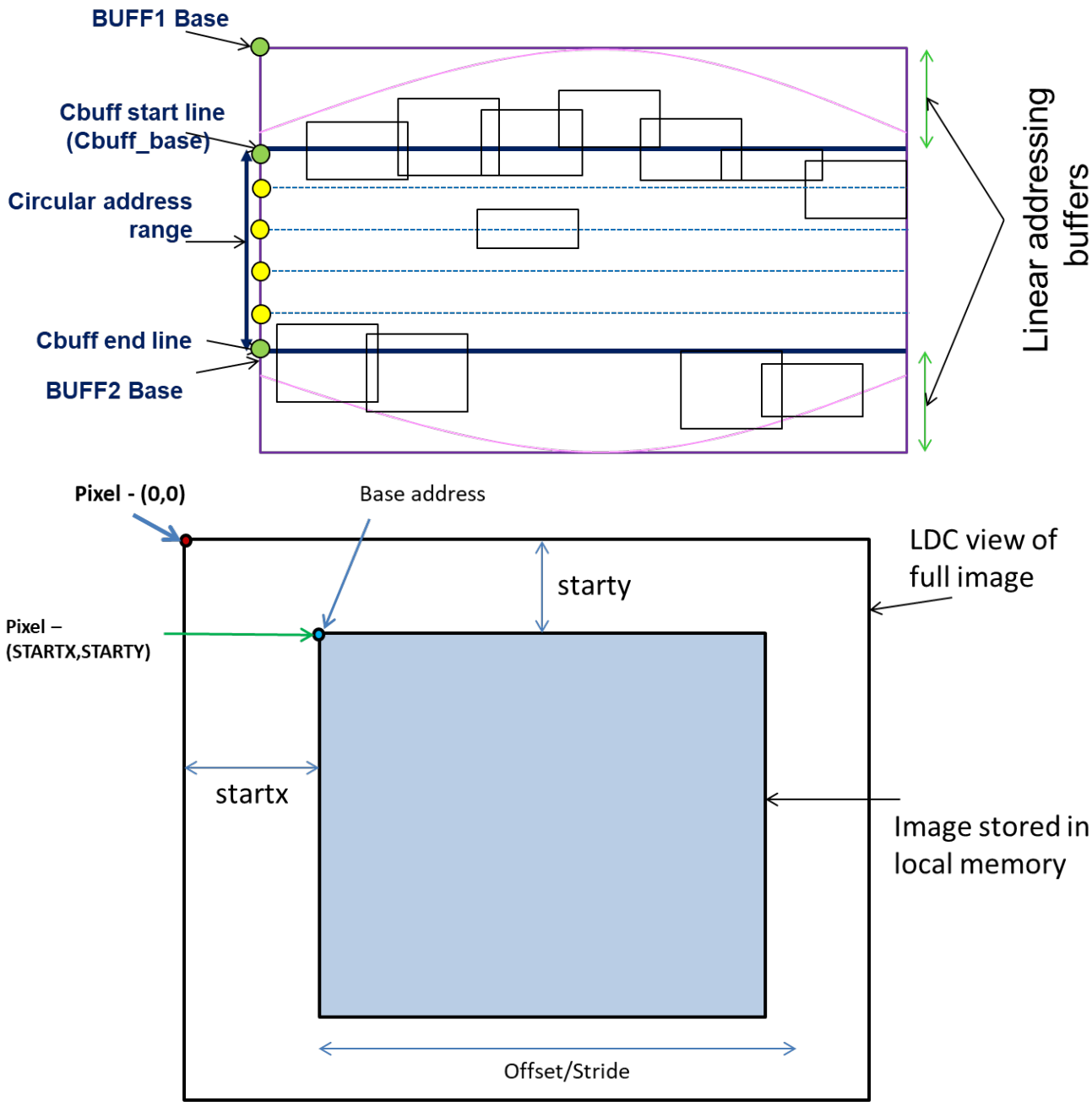


Figure 6-142. Partial Image Buffer View of LDC

Linear buffers can be stored in independent addresses and each buffer has separate base address. Circular buffer enable should be set when hybrid addressing is enabled.

6.7.4.3 LDC Programmers Guide

6.7.4.3.1 LDC Programming Geometric Distortion Mode

Geometric distortions, barrel and pincushion distortion, can be corrected on UYVY and NV12 data formats. The Mesh based back mapping is used in this mode. An additional affine transform can be configured at the same time.

1. Check and wait for VPAC_LDC_CTRL[2] BUSY to become IDLE (0).

Copyright © 2024 Texas Instruments Incorporated

2. Set VPAC_LDC_CTRL[4-3] IP_DATAMODE for UYVY (0) or NV12 data (2). Program VPAC_LDC_CTRL[6-5] IP_DFMT to required data storage format.
3. Set VPAC_LDC_CTRL[1] LDMAPEN = 1 to enable lens distortion back mapping.
4. Set the input frame base address in VPAC_LDC_RD_BASE_H / VPAC_LDC_RD_BASE_L registers. Note that the frame base address must be aligned on a 16-byte boundary. In case of YUV420, YUV422SP, Y1_Y2 and Y1_Y2Y3 modes, configure VPAC_LDC_RD_420C_BASE_H / VPAC_LDC_RD_420C_BASE_L registers for Chroma Data.
5. Set the input frame line offset in VPAC_LDC_RD_OFST[15-0] OFST.
6. If reading the input image from a circular buffer:
   a. Set VPAC_LDC_CTRL[9] IP_CIRCEN = 1.
   b. LDC computes the row number and applies modulo operation. The absolute address in the buffer is computed from this wrap-around row and the column number and then data is fetched from the circular buffer. Set the circular buffer size in VPAC_LDC_RD_OFST[29-16] MOD register field. In YUV420 modes, the number of rows in the buffer is MOD/2. In others modes, the number of rows in the buffer is MOD for both Y buffer and Cb/Cr buffer.
7. Set the tile size in VPAC_LDC_REGION_OUT_BLKSZ_j[15-8] OBH and VPAC_LDC_REGION_OUT_BLKSZ_j[7:0] OBW. Note the constraints on OBW in Table 6-156.
8. Set the pixel pad in VPAC_LDC_REGION_OUT_BLKSZ_j[19-16] PIXPAD.
9. Set the input frame size in VPAC_LDC_INPUT_FRSZ[29-16] H and VPAC_LDC_INPUT_FRSZ[13-0] W.
10. Set the output frame size in VPAC_LDC_MESH_FRSZ[29-16] H and VPAC_LDC_MESH_FRSZ[13:0] W. Mesh data has to be present for entire frame after transform.
11. Set the output compute frame size in VPAC_LDC_COMPUTE_FRSZ[29-16] H and VPAC_LDC_COMPUTE_FRSZ[13-0] W.
12. Set the starting output point in VPAC_LDC_INITXY[12-0] INITX and VPAC_LDC_INITXY[28-16] INITY.
13. Configure SL2 Interface programming registers. Details of LSE configuration is captured in Section 6.7.4.3.4.
14. Set the mesh offset table pointer to the correct address (VPAC_LDC_MESH_BASE_H / VPAC_LDC_MESH_BASE_l[31-0] ADDR and VPAC_LDC_MESH_OFST[15-0] OFST). Set the table down sampling factor for MxM down sampling in VPAC_LDC_MESHTABLE_CFG[2-0] M.
15. Set the Y plane interpolation type to bilinear or bi-cubic in VPAC_LDC_CFG[6] YINT_TYP.
16. If also using affine transform, set the six affine transform parameters in VPAC_LDC_AFF_AB.A, VPAC_LDC_AFF_AB.B, VPAC_LDC_AFF_CD.C, VPAC_LDC_AFF_CD.D, VPAC_LDC_AFF_EF.E, and VPAC_LDC_AFF_EF.F. If affine transform is not used, then these need to be set to the following values: A = 4096, B = 0, C = 0, D = 0, E = 4096, and F = 0.
17. If also using perspective transform, enable VPAC_LDC_CTRL[7] PWARPEN and set VPAC_LDC_PWARP_GH[15-0] G and VPAC_LDC_PWARP_GH[31-16] H. If perspective transform is not used, disable VPAC_LDC_CTRL[7] PWARPEN and set VPAC_LDC_PWARP_GH[15-0] G = 0 and VPAC_LDC_PWARP_GH[31-16] H = 0.
18. Set VPAC_LDC_CTRL[0] LDC_EN = 1 to start the LDC operation.
19. Start the HTS init sequencing. LDC fetch/processing is gated with hts_init.
20. Wait for VPAC_LDC_CTRL[2] BUSY to become IDLE (0) or wait for the end of frame completion interrupt.

6.7.4.3.2 LDC Programming Rotational Video Stabilization (Affine Transformation)

In the rotational video stabilization use case, consecutive frames capturing the same scene are aligned to minimize camera shake. Camera motions cause the frames to have translation or rotation differences. Thus, the frames need to be aligned to maintain a stable output video. The affine transform offers control to account for scaling, rotation, and translation transforms. For example, the rotation use case can be parameterized with the following affine transform parameters:

$$\begin{bmatrix} h_{aff} \\ v_{aff} \end{bmatrix} = \begin{bmatrix} a & b \\ d & e \end{bmatrix} \begin{bmatrix} h_d \\ v_d \end{bmatrix} + \begin{bmatrix} c \\ f \end{bmatrix}$$

$$\begin{bmatrix} h_{aff} \\ v_{aff} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} h_d \\ v_d \end{bmatrix} + \begin{bmatrix} h_o - h_o \cos(\theta) - v_o \sin(\theta) \\ v_o - v_o \cos(\theta) - h_o \sin(\theta) \end{bmatrix}$$

ldc-035

Copyright © 2024 Texas Instruments Incorporated

where theta is the angle of rotation. In the scaling case, the affine transform uses:

$$\begin{bmatrix} h_{aff} \\ v_{aff} \end{bmatrix} = \begin{bmatrix} \alpha & 0 \\ 0 & \alpha \end{bmatrix} \begin{bmatrix} h_d \\ v_d \end{bmatrix} + \begin{bmatrix} h_o(1-\alpha) \\ v_o(1-\alpha) \end{bmatrix}$$

ldc-036

where alpha is the scale factor. Translation uses an identity matrix along with the translation vector as the affine transform parameters, as in:

$$\begin{bmatrix} h_{aff} \\ v_{aff} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} h_d \\ v_d \end{bmatrix} + \begin{bmatrix} T_h \\ T_v \end{bmatrix}$$

ldc-037

where Th and Tv are the translation parameters. More complex transforms can be derived and the affine transformation is general so that optimization routines may be used to provide the best parameters to align two frames. In all affine transforms, the perspective warp parameters, g and h, should be set to 0.

The following steps need to be performed by the application to initialize and run LDC to perform affine transform.

1. Check and wait for VPAC_LDC_CTRL[2] BUSY to become IDLE (0).
2. Set VPAC_LDC_CTRL[4-3] IP_DATAMODE and VPAC_LDC_CTRL[6-5] IP_DFMT to required data mode and format.
3. If lens distortion correction is performed at the same time as the affine transform, set VPAC_LDC_CTRL[1] LDMAPEN = 1. Otherwise, set VPAC_LDC_CTRL[1] LDMAPEN = 0.
4. Set the input frame base address in VPAC_LDC_RD_BASE_H / VPAC_LDC_RD_BASE_I. Note that the frame base address must be aligned on a 16-byte boundary.
5. Set the input frame line offset in VPAC_LDC_RD_OFST[15-0] OFST.
6. If reading the input image from a circular buffer:
   a. Set VPAC_LDC_CTRL[9] IP_CIRCEN = 1.
   b. If using row address, LDC computes the row number and applies a modulo operation. The absolute address in the buffer is computed from this wrap-around row and the column number and then data is fetched from the circular buffer. Set the circular buffer size in VPAC_LDC_RD_OFST[29-16] MOD. In YUV420 modes, the number of rows in the Y buffer is MOD and the number of rows in the Cb/Cr buffers is MOD/2. In others modes, the number of rows in the buffer is MOD for both Y buffer and Cb/Cr buffer.
7. Set the tile size in VPAC_LDC_OUT_BLKSZ[15-8] OBH and VPAC_LDC_OUT_BLKSZ[7-0] OBW. Note the constraints on OBW in Table 6-156.
8. Set the pixel pad in VPAC_LDC_OUT_BLKSZ[19-16] PIXPAD.
9. Set the input frame size in VPAC_LDC_INPUT_FRSZ[29-16] H and VPAC_LDC_INPUT_FRSZ[13-0] W.
10. Set the output frame size in VPAC_LDC_MESH_FRSZ[29-16] H and VPAC_LDC_MESH_FRSZ[13-0] W. Mesh data has to be present for entire frame after transform.
11. Set the output compute frame size in VPAC_LDC_COMPUTE_FRSZ[29-16] H and VPAC_LDC_COMPUTE_FRSZ[13-0] W.
12. Set the starting output point in VPAC_LDC_INITXY[12-0] INITX and VPAC_LDC_INITXY[28-16] INITY.
13. Configure SL2 Interface programming registers. Details of LSE configuration is captured in Section 6.7.4.3.4.
14. If the data format is NV12, set the 420 UV input plane base address in VPAC_LDC_RD_420C_BASE_H / VPAC_LDC_RD_420C_BASE_I. This address must be 16-byte aligned.
15. Set the mesh offset table pointer to the correct address (VPAC_LDC_MESH_BASE_H / VPAC_LDC_MESH_BASE_I and VPAC_LDC_MESH_OFST. Set the table down sampling factor for MxM down sampling in MESHTABLE_CFG.M.
16. Set the Y plane interpolation type to bilinear or bicubic in VPAC_LDC_CFG[6] YINT_TYP.
17. Set the six affine transform parameters in VPAC_LDC_AFF_AB.A, VPAC_LDC_AFF_AB.B, VPAC_LDC_AFF_CD.C, VPAC_LDC_AFF_CD.D, VPAC_LDC_AFF_EF.E, and VPAC_LDC_AFF_EF.F. If

Copyright © 2024 Texas Instruments Incorporated

affine transform is not used, then these need to be set to the following values: A = 4096, B = 0, C = 0, D = 0, E = 4096, and F = 0.
18. Disable perspective warp transform with VPAC_LDC_CTRL[7] PWARPEN = 0. Set the two perspective transform parameters VPAC_LDC_PWARP_GH[15-0] G = 0 and VPAC_LDC_PWARP_GH[31-16] H = 0.
19. Set VPAC_LDC_CTRL[0] LDC_EN = 1 to start the LDC operation.
20. Start the HTS init sequencing. LDC fetch/processing is gated with hts_init.
21. Wait for VPAC_LDC_CTRL[2] BUSY to become IDLE (0) or wait for the end of frame completion interrupt.

6.7.4.3.3 LDC Programming Perspective Transformation

Perspective transforms are used to align the image data of two different cameras viewing the same scene from different positions. It can also be used in the case of stereo image rectification to align the epipolar lines of the left/right image pair with their scan lines.

The following steps need to be performed by the application to initialize and run LDC to perform perspective transform.

1. Check and wait for VPAC_LDC_CTRL[2] BUSY to become IDLE (0).
2. Set VPAC_LDC_CTRL[4-3] IP_DATAMODE and VPAC_LDC_CTRL[6-5] IP_DFMT to required data mode and format.
3. If lens distortion correction is performed at the same time as the perspective transform, set VPAC_LDC_CTRL[1] LDMAPEN = 1. Otherwise, set LDMAPEN = 0.
4. Set the input frame base address in VPAC_LDC_RD_BASE_H / VPAC_LDC_RD_BASE_I. Note that the frame base address must be aligned on a 16-byte boundary.
5. Set the input frame line offset in VPAC_LDC_RD_OFST[15-0] OFST.
6. Set the tile size in VPAC_LDC_OUT_BLKSZ[15-8] OBH and VPAC_LDC_OUT_BLKSZ[7-0] OBW. Note the constraints on OBW in Table 6-156.
7. Set the pixel pad in VPAC_LDC_OUT_BLKSZ[19-16] PIXPAD.
8. Set the input frame size in VPAC_LDC_INPUT_FRSZ[29-16] H and VPAC_LDC_INPUT_FRSZ[13-0] W.
9. Set the output frame size in VPAC_LDC_MESH_FRSZ[29-16] H and VPAC_LDC_MESH_FRSZ[13-0] W. Mesh data has to be present for entire frame after transform.
10. Set the output compute frame size in VPAC_LDC_COMPUTE_FRSZ[29-16] H and VPAC_LDC_COMPUTE_FRSZ[13-0] W.
11. Set the starting output point in VPAC_LDC_INITXY[12-0] INITX and VPAC_LDC_INITXY[28-16] INITY.
12. Configure SL2 Interface programming registers. Details of LSE configuration is captured in Section 6.7.4.3.4.
13. If the data format is NV12, set the 420 UV input plane base address in VPAC_LDC_RD_420C_BASE_H / VPAC_LDC_RD_420C_BASE_I. This address must be 16-byte aligned.
14. Set the mesh offset table pointer to the correct address (VPAC_LDC_MESH_BASE_H / VPAC_LDC_MESH_BASE_I and VPAC_LDC_MESH_OFST. Set the table down sampling factor for MxM down sampling in MESHTABLE_CFG.M.
15. Set the Y plane interpolation type to bilinear or bicubic in VPAC_LDC_CFG[6] YINT_TYP.
16. Set the eight perspective transform parameters in VPAC_LDC_AFF_AB.A, VPAC_LDC_AFF_AB.B, VPAC_LDC_AFF_CD.C, VPAC_LDC_AFF_CD.D, VPAC_LDC_AFF_EF.E, and VPAC_LDC_AFF_EF.F. If affine transform is not used, then these need to be set to the following values: A = 4096, B = 0, C = 0, D = 0, E = 4096, F = 0,G = 0 and H = 0. Enable VPAC_LDC_CTRL[7] PWARPEN = 1.
17. Set VPAC_LDC_CTRL[0] LDC_EN = 1 to start the LDC operation.
18. Start the HTS init sequencing. LDC fetch/processing is gated with hts_init.
19. Wait for VPAC_LDC_CTRL[2] BUSY to become IDLE (0) or wait for the end of frame completion interrupt.

6.7.4.3.4 LDC Programming LSE

Following provides the LSE general programming guideline.

It also captures an additional secondary specific use case configuration values for some registers.

- Frame Size = 1920x1080
- Output Block size = 64x32
- Luma data being consumed by MSC and Chroma is written out by UDMA
  - Block to line conversion done for Luma data via SL2.
  - Chroma data is written out at block level.

Copyright © 2024 Texas Instruments Incorporated

- Circular buffer size just to do ping-pong between LDC – MSC(Y) and LDC – UDMA(C)
- 12-bit pixel data

1. Pixel Data Format
   a. VPAC_LDC_LSE_BUF_CFG_j (PIX_FMT_PW, PIX_FMT_CNTRSZ, and PIX_FMT_ALIGN) parameters define the pixel output data format for this output channel.
   b. Common data formats:
      i. Fully packed 12-bit : PIX_FMT_PW=1, PIX_FMT_CNTRSZ=1, PIX_FMT_ALIGN=0
      ii. 12-bit unpacked (MSB aligned) : PIX_FMT_PW = 1, PIX_FMT_CNTRSZ=2, PIX_FMT_ALIGN=1
2. Output SL2 Circular Buffer Configuration
   a. VPAC_LDC_LSE_BUF_ATTR0_j (buf_stride) – define the line stride size (must be 64 byte multiple)
      i. VPAC_LDC_LSE_BUF_ATTR0_j[15-6] BUF_STRIDE = 2880 (1920 * 1.5)
      ii. VPAC_LDC_LSE_BUF_ATTR0_j[15-6] BUF_STRIDE = 96 (64 * 1.5)
   b. VPAC_LDC_LSE_BUF_ATTR0_j (cbuf_size) – define the size of the circular buffer in the SL2 (number of lines)
      i. VPAC_LDC_LSE_BUF_ATTR0_j[24-16] CBUF_SIZE = 68 (2*OBH+4)
      ii. VPAC_LDC_LSE_BUF_ATTR0_j[24-16] CBUF_SIZE = 64 (2*OBH)
3. Output SL2 Circular Buffer 2D mode Configuration (applicable for LDC only - TBD)
   a. VPAC_LDC_LSE_BUF_ATTR1_j (buf_blk_width) – defines the number of bytes equivalent to OBW (output block width) pixels.
      i. VPAC_LDC_LSE_BUF_ATTR1_j.buf_blk_width = 96 (64*1.5)
   b. VPAC_LDC_LSE_BUF_ATTR1_j (cbuf_bpr) – defines the number of 2D blocks per row in the SL2 circular buffer.
      i. VPAC_LDC_LSE_BUF_ATTR1_j[25-16] CBUF_BPR_CHAN = 30
      ii. VPAC_LDC_LSE_BUF_ATTR1_j[25-16] CBUF_BPR_CHAN = 1
4. Base Address and Channel Enable:
   a. VPAC_LDC_LSE_BUF_BA_j[23-6] ADDR – defines the SL2 base address of the Circular buffer for this output channel.
   b. VPAC_LDC_LSE_BUF_BA_j[31] ENABLE – enables this output channel.

6.7.4.3.5 LDC Programming Restrictions and Special Cases

The following list of LDC programming restrictions should be considered:

1. VPAC_LDC_LSE_BUF_ATTR1_j[25-16] CBUF_BPR_CHAN supports 1023 number of block per row. Hence, with output block width of 8, maximum line size supported in SL2 memory is 8184 pixels.
2. In 422 mode, always program as data format as 8-bit (design does not select data format automatically based on mode)
3. Due to hardware limitations, make sure that input block size (height and width) does not cross 1023 (pixels/lines).
   - The hardware generates either PIX_IBLK_MEMOVF or MESH_IBLK_MEMOVF events in this case.
4. VPAC_LDC_CTRL[0] LDC_EN should be high before issuing hts_init.

Copyright © 2024 Texas Instruments Incorporated

6.7.5 VPAC Multi-Scaler (MSC)

6.7.5.1 MSC Overview

VPAC_MSC (VPAC Multi-Scaler, hereinafter simply referred to also as MSC) is a Hardware Acceleration (HWA) module in Vision Pre-processing Accelator (VPAC) . MSC supports two sets of multi-output scaling filters to accelerate generation of pyramid (Octave) and intra-octave scales required by Optical Flow processing and DSP vision algorithms.

6.7.5.1.1 MSC Features

- 10 scaled outputs independently mapped to 1 or 2 processing threads (any number up to 10 scaler outputs can be mapped to one thread, while the remaining scaler outputs can be mapped to the other thread)
- Two input sub-channels (primary and secondary) per thread to support 1 or 2-plane input sources
- Programmable scaling engine:
  - Pyramid or intra-octave scale generation
  - 1x ~ x/4 image downscaling
  - 1x generic separable convolution filtering
  - On-the-fly (OTF) or memory-to-memory (M2M) operation with line buffers in SL2 (thus, frame width not constrained by internal line buffer width).
- Polyphase filters:
  - 5-tap separable filter kernel structure
  - Programmable kernel sizes for vertical/horizontal filter (3, 4, or 5)
  - Polyphase implementation with support of 64 or 32 phases
  - Independent programming of vertical/horizontal filters with initial phase programming option
- Filter Coefficients (shared by all scalers):
  - Coefficients in signed 10-bit number (typically 1-bit integer with 8 fractional bits)
  - Programmable Coefficient shift size to set precision of coefficient number (shift size represents the fractional bit width)
  - 2 dedicated sets of 5-tap single phase filter coefficients (for example, Gaussian) for Pyramid (Octave) generation or custom convolution filtering.
  - 4 sets of 5-tap x 32 phase coefficients
    - Two can be combined to support 5-tap x 64-phase filters
    - First can be configured to hold additional 5-tap single-phase convolution filters for non-resizing application.
- Input/output format support:
  - Each thread can support the following input source formats:
    - Primary sub-channel: One plane (uniform or interleaved format - Y, CbCr interleaved, R)
    - Secondary sub-channel: One plane (uniform Y or interleaved CbCr, GB interleaved) with option to extract CbCr data from YUV422 source data read by primary channel
  - Each Input channel supports 8/12-bit pixel data in 8/12/16-bit pixel container with programmable LSB/MSB alignment
  - Each Output channel supports 8/12-bit pixel data in 8/12/16-bit pixel container with programmable LSB/MSB alignment
  - Luma dedicated output channels (chan 0/2/4/6) can support YUV422 interleaved output (8/12-bit in 8/12/16-bit container per component) by combining Y with UV data from the Chroma dedicated output channels (chan 1/3/5/7)
- Performance:
  - 1 cycle/pixel per plane
  - Input line skip for thread performing only pyramid-generation (50% less processing time)
- Other features:
  - Programmable Input and Output Regio of Interest (ROI) (or clipping) for each scaler
  - Support for Vertical and Horizontal edge Padding (replication) for both luma/chroma data plane inputs
- Functional Safety Support: CRC frame data signature capture on all output channels for fault detection in the data path.

Copyright © 2024 Texas Instruments Incorporated

6.7.5.2 MSC Functional Description

6.7.5.2.1 MSC Functional Overview

VPAC_MSC consists of 10 programmable resizers performing multi-thread/multi-scaling operations (1-input to N-outputs and 1-input to M-outputs where N+M is 10 or less). Each processing thread of MSC HWA reads its input plane data from a Shared Memory buffers (SL2) circular line buffers, performs multi-scaling operations (ratios between X and 0.25X) on the selected thread channel input, and writes out results to SL2 circular line buffers. In case of OTF operation, the source data is generated from another HWA. In case of M2M operation, the source data is read from the DDR memory. Data transfers from/to SL2 to/from external memory (or another HWA) are handled by VPAC level DMA controller with transfer request events coming from VPAC top level HWA Thread Scheduler (HTS), see *HWA Threads Scheduler (HTS)*.

A typical configuration of the MSC module uses one input plane data to generate 7 intra-octave scales and the next octave image. Multi-pass processing is then utilized to generate up to next 6 or 7 sets of scales/octaves. Since the resolution after each octave is essentially reduced by 4x, the performance requirement for generating an infinite number of scales is bounded by 1.33x of the base input image pixel rate. The second input can be used to enable a pyramid generation processing for another input data or a set of scales/pyramid generation for chroma data as long as the total number of scales needed by both threads is 10 or less. Each resizer in a processing thread can access any coefficient set.

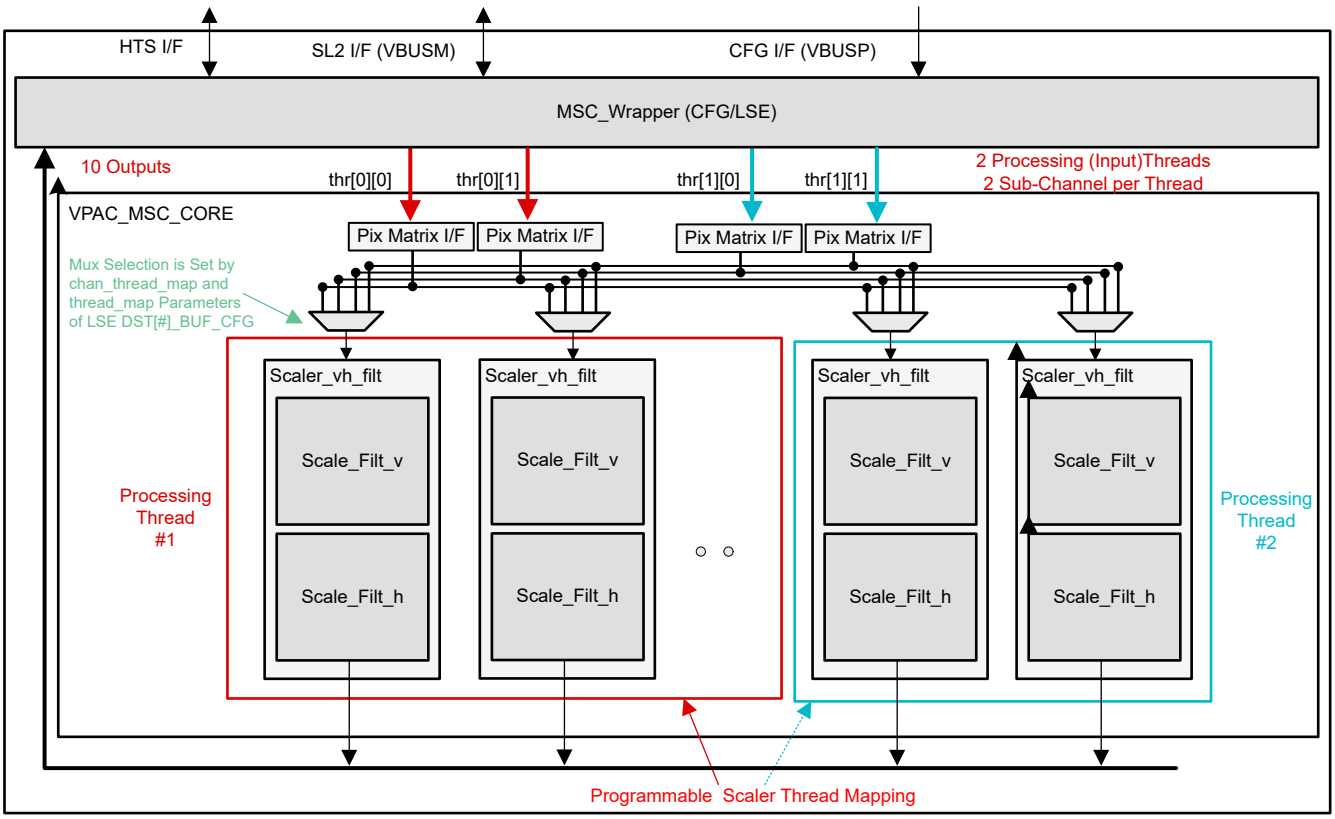


Figure 6-143. Block Diagram of Multi-Scaler Filter Core

For details on MSC integration in VPAC, see Section 6.7.2,*VPAC Subsystem Level*.

6.7.5.2.2 Resizer Algorithm Details

This section describes the algorithm details of the Horizontal and Vertical Down-Scaling engines.

6.7.5.2.2.1 Multiple Scales Generations

For a high number of scales needed by vision image processing algorithm (for example, 32 or more), there are many options in generating the multiple scales as shown in Figure 6-144. Among the options, the “linear rescale from octave resolution” gives the best tradeoff in video quality and design simplicity. The MSC implements 1-to-N

Copyright © 2024 Texas Instruments Incorporated

multi-rescaling (that is, from one base image to multi-intra-scales and the next octave image generation) as shown as option B in Figure 6-144.

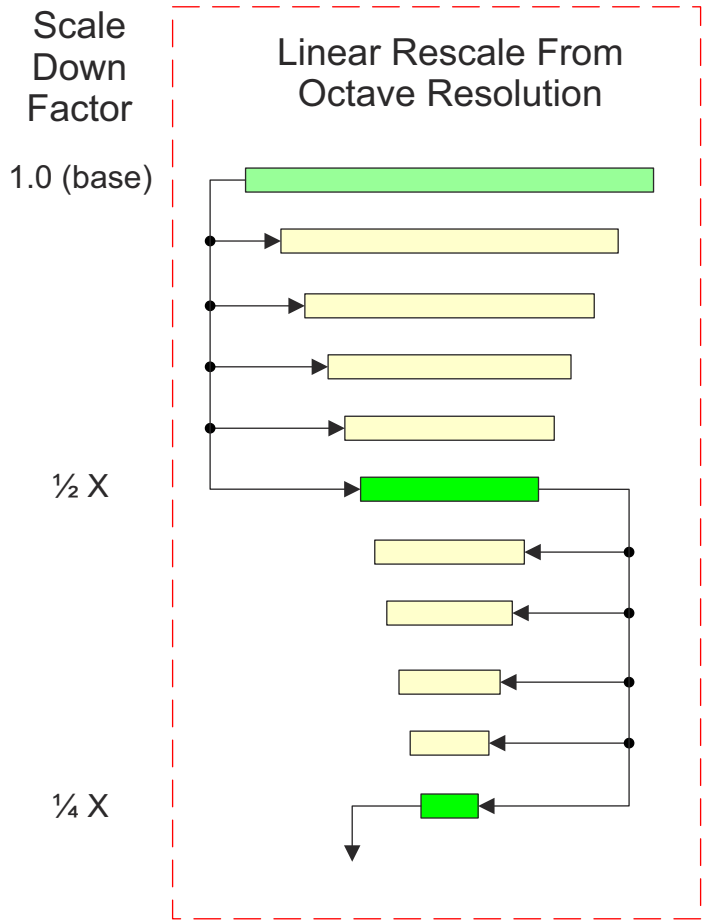


Figure 6-144. Multiple Scale Generation

6.7.5.2.2.2 Polyphase Filter

Both vertical and horizontal resizers in MSC hardware are implemented using a programmable Polyphase filter structure supporting 5-tap kernel with either 64 or 32 phases.

6.7.5.2.2.2.1 Interpolation/Resampling

Figure 6-145 illustrates the interpolation principle. The assumptions are that the input pixels are evenly spaced. The distance between each input pixel is assumed to be 1. The magnification ratio is given by 64/N and p/64 is the initial phase of the output data. The output pixels are evenly spaced as well. The distance between each output pixel is given by N/64 (in the example below, N > 64). The position of the n-th output pixel is given by (n x N + p) / 64.

Copyright © 2024 Texas Instruments Incorporated

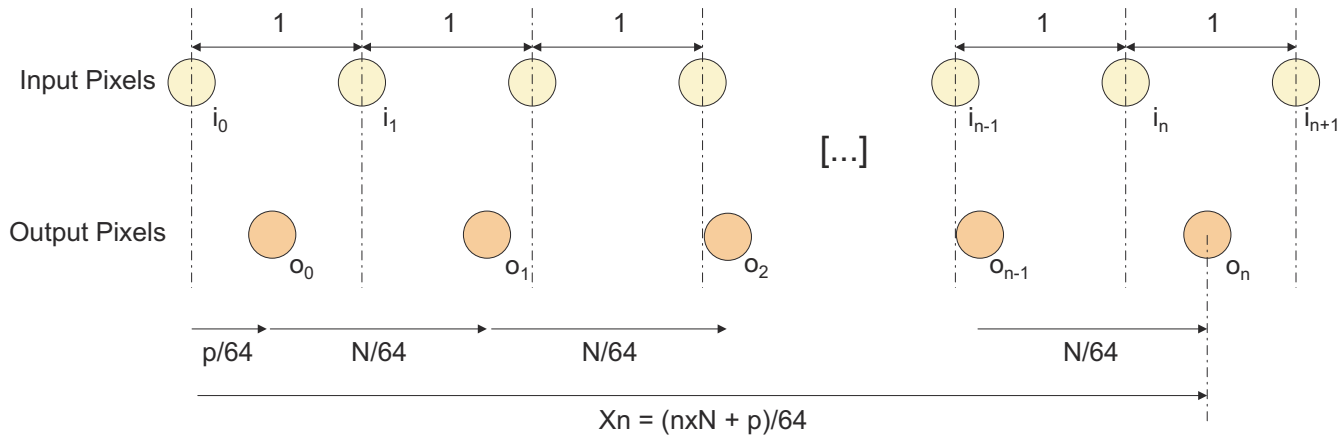


Figure 6-145. Interpolation Principle

Figure 6-146 illustrates the interpolation principle at the n-th output pixel (on) at position x_n. Interpolation method with m and d parameters, are described as:

m = floor((n x N + p) / 64)

d = ((n x N + p) / 64) - m

Figure 6-146 illustrates the concept when the filters are configured in 4-tap mode for non-integer rescaling (the first coefficient corresponding to O_{m-2} is set to zero).

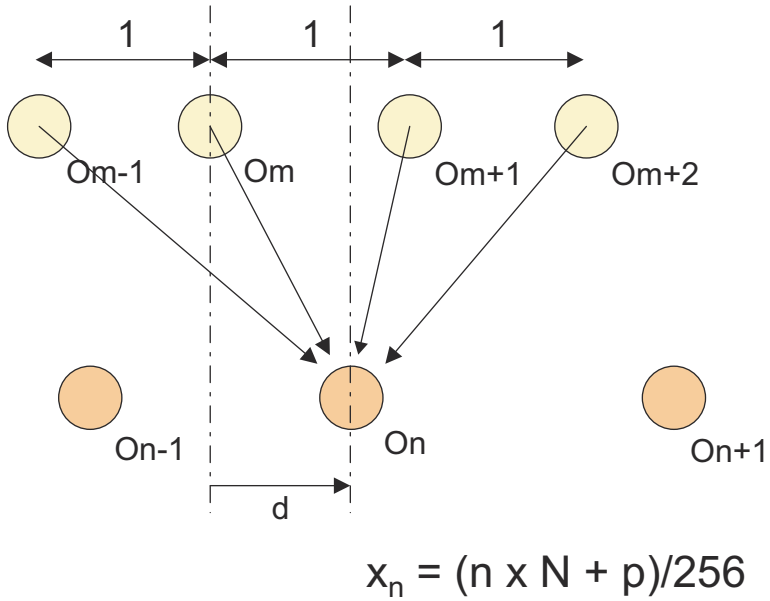


Figure 6-146. Interpolation Process

Figure 6-147 illustrates another explanation of the concept of a Polyphase filter using a simple 2-tap linear filter in a design which supports 5 phases.

Copyright © 2024 Texas Instruments Incorporated

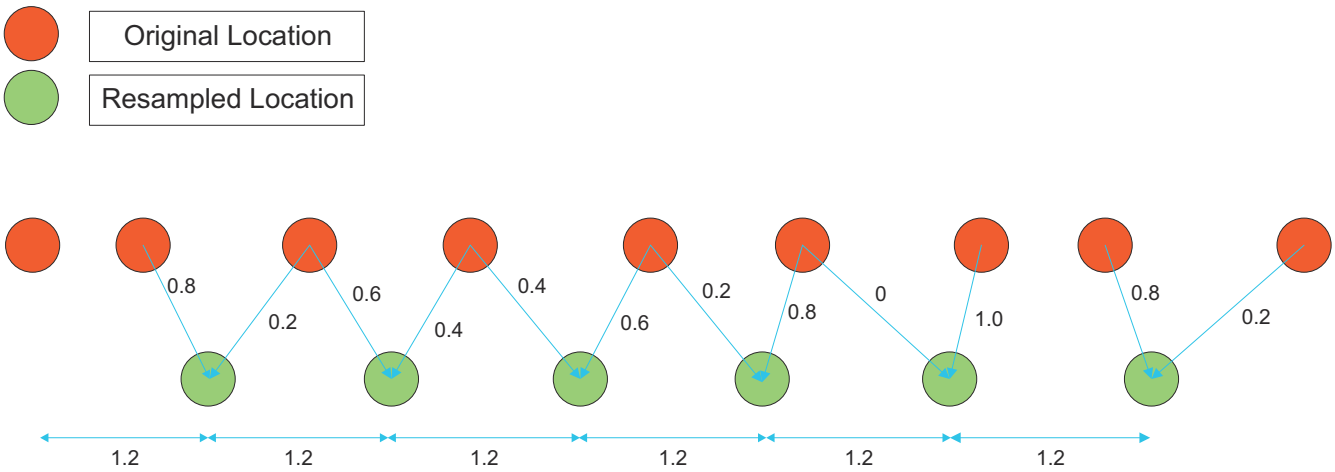


Figure 6-147. Polyphase Filtering

For each one of the 5 phases, the weights (or filter coefficients) are different and after the 5 phases the weights start repeating, see Figure 6-147.

Figure 6-148 depicts the architecture for the independent polyphase filter unit.

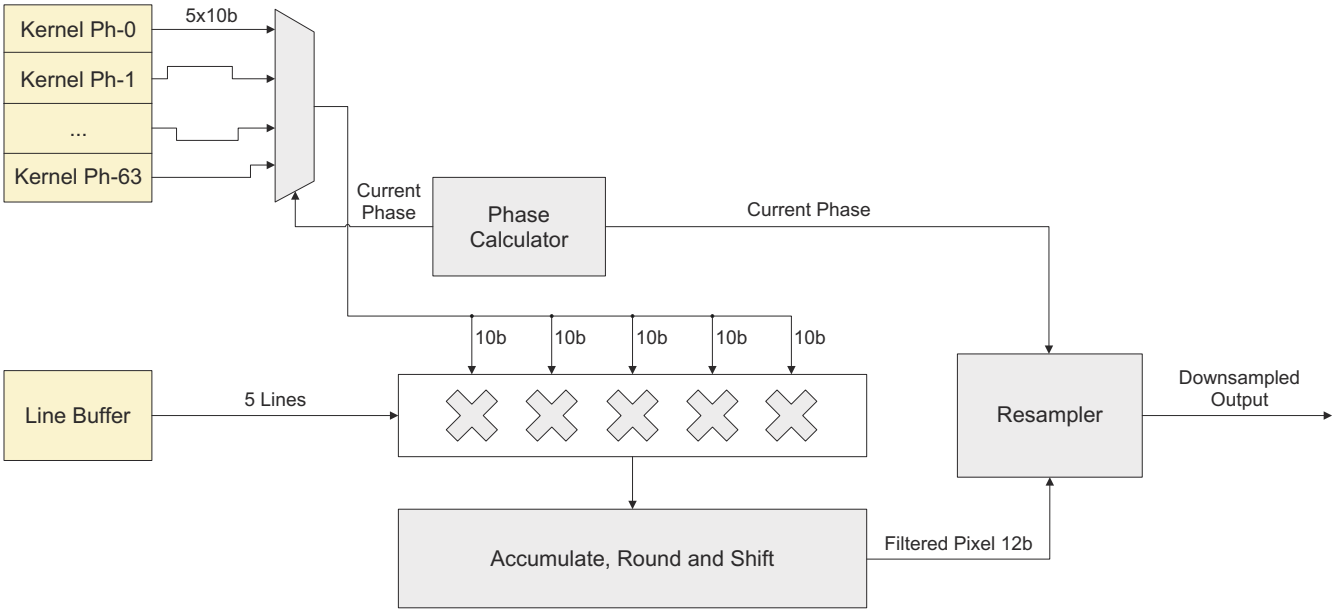


Figure 6-148. Polyphase Filter Unit

Both the horizontal and vertical filter described earlier are an instance of the Polyphase filter unit, though only the vertical filter requires line memories (implemented in SL2 memory space). In all, there will be 20 instances of the Polyphase filter unit, 2 filters per channel and 10 total channels in the VPAC_MSC module.

6.7.5.2.2.2 Phase Calculation and Re-sampler

The phase calculation block calculates the current phase which in turn is used for selecting the right coefficient set and for the output qualifier in the Re-sampler block. The Re-sampler engine then decides whether or not the output created by the interpolation unit is a valid output or not.

6.7.5.2.2.3 Shared Coefficient Buffers

The MSC module supports up to 4 sets of multi-phase coefficients for generic non-integer resizing and up to 10 sets of single-phase coefficients (5x1 entries) for Octave generation and integer resizing applications. The multi-phase coefficients can also be configured in one of two following options:

- 4 sets of 5-tap x 32 phase coefficients

Copyright © 2024 Texas Instruments Incorporated

- 1 set of 5-tap x 64 phase coefficients and 2 sets of 5-tap x 32 phase coefficients
- 2 sets of 5-tap x 64 phase coefficients

The 32 phase configuration option allows using independent coefficients for vertical and horizontal resizer to achieve aspect ratio change during the scaling.

6.7.5.2.2.2.4 Border Pixel Padding

The MSC module replicates missing lines and pixels at top/bottom and left/right sides of the input frame for non-interleaved (luma) and interleaved (chroma) data.

The missing lines at the top/bottom are replicated in the LSE sub-module prior to the vertical scaler and the missing pixels at the left/right sides of each line are replicated in the core prior to the horizontal scaler. For more information about LSE module, see *Load Store Engine (LSE)*.

Replication scheme simply repeats the edge line or pixel to create missing lines/pixels. Figure 6-149 shows horizontal pixel replication.

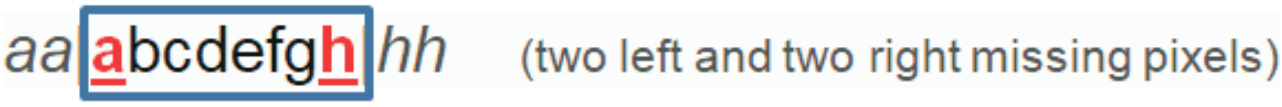


Figure 6-149. Horizontal Pixel Replication

If a scaling is performed on a sub-frame (ROI) within the input frame, no replicated lines/pixels will be used since the lines/pixels outside the sub-frame are already present in the input frame.

For interleaved data format, the horizontal replication is done separately for U and V data. For U-data, edge U-pixel will be replicated. For V-data, edge V-pixel will be replicated. There is no distinction in the vertical edge padding.

6.7.5.2.2.3 ROI Handling

- ROI supported during scaling generation
  - Only one ROI at Input Frame and one ROI at Output frame per scale
  - ROI definitions (for input as well as all output scales) can be changed at every input frame under software control.
  - ROI for each scale (for output) can be different among scales (as well as with respect to the input). It is set under software Control.
  - The key purpose of ROI is DDR bandwidth reduction. There is additional help by reduction on processing power on DSP and hardware.
- The input ROI is a superset area considering following two scenarios.
  - Multiple ROIs in given Input Frame, if there are multiple ROI for input frame
  - Input Frame area from Overlapped ROI across scales

Figure 6-150 shows an example of multiple ROIs in the source superset ROI area.

Copyright © 2024 Texas Instruments Incorporated

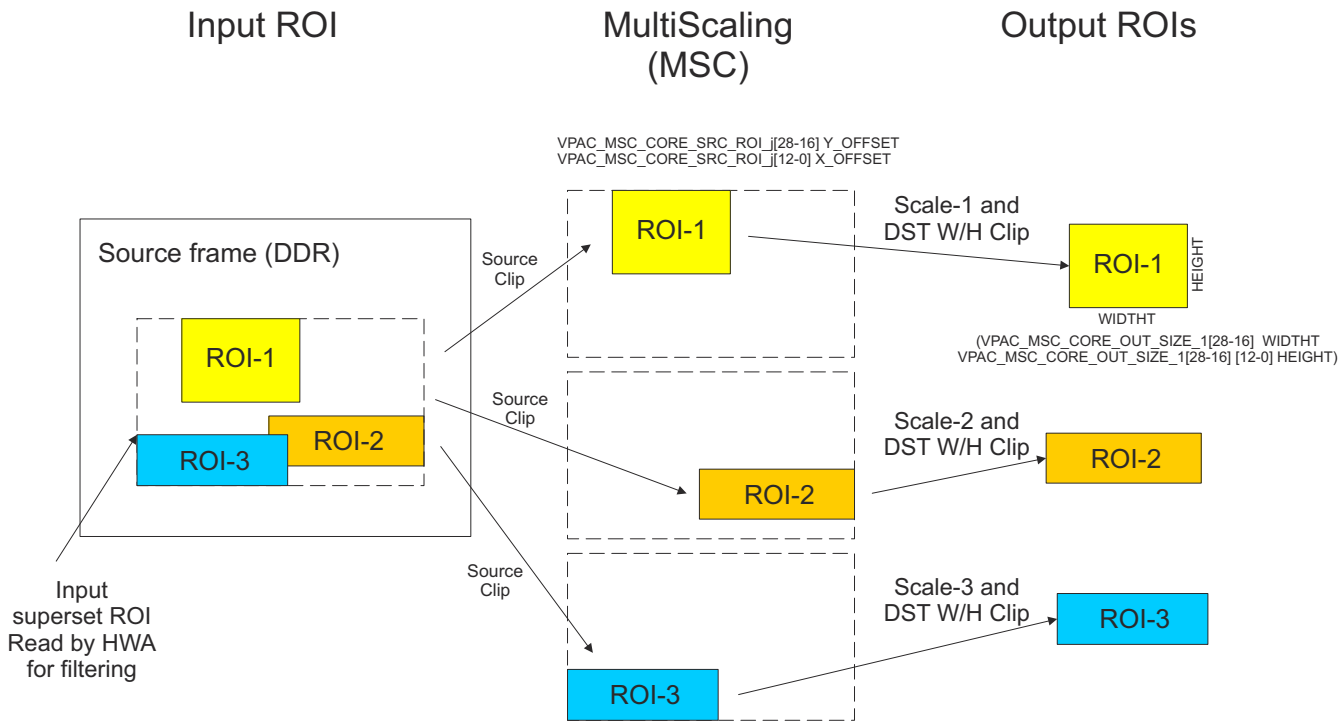


Figure 6-150. Multiple-ROI Support Illustration

Figure 6-151 illustrates another application of generating output ROIs with different scales from a common source region.

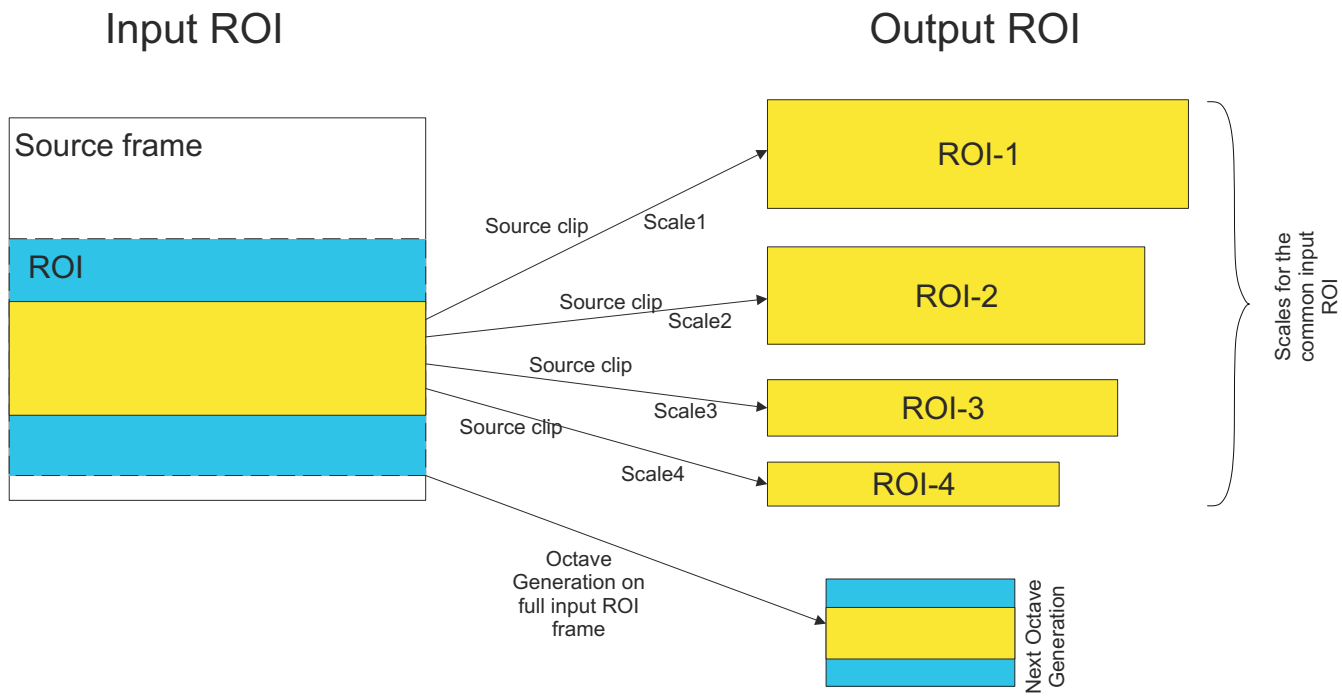


Figure 6-151. Multiple-ROI From a Common Source ROI

By defining output ROIs, only the data within ROI in the “superset ROI” scaled output image is saved thus reducing the DDR traffic significantly.

6.7.5.2.3 MSC Data Formats Supported

Table 6-160 summarizes data format supported by MSC.

Copyright © 2024 Texas Instruments Incorporated

Table 6-160. MSC Input/Output Data Formats

Module (IO)	Bit-Depth	Chroma Format	Packing
MSC Input	8-bit	YUV420	Fully Packed
	12-bit	YUV420	Fully Packed
	12-bit	YUV420	Unpacked in 16-bit
MSC Output	8-bit	YUV420	Fully Packed
	12-bit	YUV420	Fully Packed
	12-bit	YUV420	Unpacked in 16-bit

Table 6-161 and Table 6-162 show the pixel data memory organization for YUV420 (2-plane 12-bit fully packed format).

Table 6-161. YUV420 (2-plane 12-bit Fully Packed Format), First Plane

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0	
Y2[7:0] Y1 Y0	+0x0
Y5[3:0] Y4 Y3 Y2[11:8]	+0x4
Y7 Y6 Y5[11:4]	+0x8
Y10[7:0] Y9 Y8	+0C
Y13[3:0] Y12 Y11 Y10[11:8]	+10
Y15 Y14 Y13[11:4]	+14

Table 6-162. YUV420 (2-plane 12-bit Fully Packed Format), Second Plane

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0	
Cb1[7:0] Cr0 Cb0	+0x0
Cr2[3:0] Cb2 Cr1 Cb1[11:8]	+0x4
Cr3 Cb3 Cr2[11:4]	+0x8
Cb5[7:0] Cr4 Cb4	+0C
Cr6[3:0] Cb6 Cr5 Cb5[11:8]	+10
Cr7 Cb7 Cr6[11:4]	+14

Table 6-163 and Table 6-164 show the pixel data memory organization for YUV420 (2-plane 8-bit fully packed format)

Table 6-163. YUV420 (2-plane 8-bit Fully Packed Format), YUV 4:2:0 – NV12, First Plane

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0	
Y3 Y2 Y1 Y0	+0x0
Y7 Y6 Y5 Y4	+0x4
Y11 Y10 Y9 Y8	+0x8
Y15 Y14 Y13 Y12	+0xC

Copyright © 2024 Texas Instruments Incorporated

Table 6-164. YUV420 (2-plane 8-bit Fully Packed format), YUV 4:2:0 – NV12, Second Plane

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0	
Cr1 Cb1 Cr0 Cb0	+0 x0
Cr3 Cb3 Cb2 Cb2	+0 x4
Cr5 Cb5 Cr4 Cb4	+0 x8
Cr7 Cb7 Cb6 Cb6	+0 xC

Note

MSC supports both NV12 and NV21 chroma component ordering. Since there is no color processing, the ordering of Cb and Cr is irrelevant. Simply, if a NV12 chroma plane is the input, the output will be a NV12 chroma plane.

12-bit unpacked format (16-bit container) with LSB/MSB alignments is shown below

Table 6-165 and Table 6-166 show the pixel data memory organization for YUV420 (2-plane 12-bit unpacked format – LSB aligned)

Table 6-165. YUV420 (2-plane 12-bit Unpacked Format – LSB aligned), YUV 4:2:0 – NV12 (12-bit) (0x3D), First Plane

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0	
unused Y1 unused Y0	+0 x0
unused Y3 unused Y2	+0 x4

Table 6-166. YUV420 (2-plane 12-bit Unpacked Format – LSB aligned), YUV 4:2:0 – NV12 (12-bit) (0x3D), Second Plane

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0	
unused Cr0 unused Cb0	+0 x0
unused Cr1 unused Cb1	+0 x4

Table 6-167 and Table 6-168 show the pixel data memory organization for YUV420 (2-plane 12-bit unpacked format – MSB aligned)

Table 6-167. YUV420 (2-plane 12-bit Unpacked Format – MSB aligned), YUV 4:2:0 – NV12 (12-bit) (0x3D), First Plane

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0	
Y1 unused Y0 unused	+0x0
Y3 unused Y2 unused	+0x4

Table 6-168. YUV420 (2-plane 12-bit Unpacked Format – MSB aligned), YUV 4:2:0 – NV12 (12-bit) (0x3D), Second Plane

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0	
Cr0 unused Cb0 unused	+0 x0
Cr1 unused Cb1 unused	+0 x4

Copyright © 2024 Texas Instruments Incorporated

6.7.5.3 MSC Interrupt Conditions

6.7.5.3.1 CPU Interrupts

Table 6-169 lists the interupts generated by the MSC module.

Table 6-169. Interrupts

Interrupt	Type	Description
VPAC_MSC_LSE_FR_DONE_EVT_0	Pulse	Frame Processing complete for all filters in the processing thread 0.
VPAC_MSC_LSE_FR_DONE_EVT_1	Pulse	Frame Processing complete for all filters in the processing thread 1.
VPAC_MSC_LSE_SL2_RD_ERR	Pulse	Set whenever there is an error response on VBUSM read command request for any input channel
VPAC_MSC_LSE_SL2_WR_ERR	Pulse	Set whenever there is an error response on VBUSM write command request for any output channel

All interrupts are single pulse event signals that are mapped to the VPAC level interrupt aggregation logic. The MSC has no mask/set/clear registers for these events. For more information see Section 6.7.2, *VPAC Subsystem*.

6.7.5.3.2 Interrupt Event Description

6.7.5.3.2.1 VPAC_MSC_LSE_FR_DONE_EVT_0/1 Events

These events are generated when all filters associated with the thread 0 or 1 complete the frame processing. This is set when input EOF (end of frame) event and output EOF (end of frame) events are detected.

6.7.5.3.2.2 VPAC_MSC_LSE_SL2_RD_ERR Interrupt Event

This event is generated whenever the VBUSM read status is something other than 0 (success).

The VPAC status register VPAC_MSC_STATUS_ERROR[4-0] VM_RD_ERR stores the last non-zero RSTATUS value and the input channel number for user to see what type of error has occurred on which channel.

Table 6-170. Encoding for RSTATUS

RSTATUS	Meaning
0	Success
1	Addressing error
2	Protection error
3	Timeout error
4	Data error
5	Unsupported Addressing Mode error
6	RESERVED
7	Exclusive read fail

6.7.5.3.2.3 VPAC_MSC_LSE_SL2_WR_ERR Interrupt Event

This event is generated whenever the VBUSM write status is something other than 0 (success).

The VPAC status register VPAC_MSC_STATUS_ERROR[14-8] VM_WR_ERR stores the last non-zero WSTATUS value and the output channel number for user to see what type of error has occurred on which channel.

Table 6-171. Encoding for WSTATUS

WSTATUS	Meaning
0	Success
1	Addressing error
2	Protection error
3	Timeout error
4	Data error

Copyright © 2024 Texas Instruments Incorporated

Table 6-171. Encoding for WSTATUS (continued)

WSTATUS	Meaning
5	Unsupported Addressing Mode error
6	RESERVED
7	Exclusive write fail

6.7.5.4 MSC Submodule Details

The MSC_LSE (Load Store Engine) handles all interfacing to the SL2 memory space and performs following functions:

- Provides interface to SL2 circular buffers via a common VBUSM master interface
- Manages input and output channel updates in sync with HTS
- Provides data unpacking for core and packing for SL2 interface
- Manages LSE specific configuration registers

The MSC_CORE performs all filtering operations for various 2-in/10-out multi-scaling/multi-thread configurations. It has no direct connection to external interface and it works on a frame data coming in and out in raster scan order.

The internal data transfers between the MSC_LSE and MSC_CORE are done over VBUSP write-only streaming interfaces.

6.7.5.4.1 MSC Configuration Interface (MSC_CFG)

The CFG interface supports a standard 32-bit VBUSP slave interface for rgister configuration. The MSC configuration registers consist of LSE configuration registers and CORE filter configuration registers. A simple VBUSP switch (CBA SCR) is used to route VBUSP commands to these two configuration sub-modules.

6.7.5.4.2 MSC Load Store Engine (MSC_LSE)

6.7.5.4.2.1 MSC_LSE Overview

Figure 6-152 shows the MSC_LSC block diagram.

Copyright © 2024 Texas Instruments Incorporated

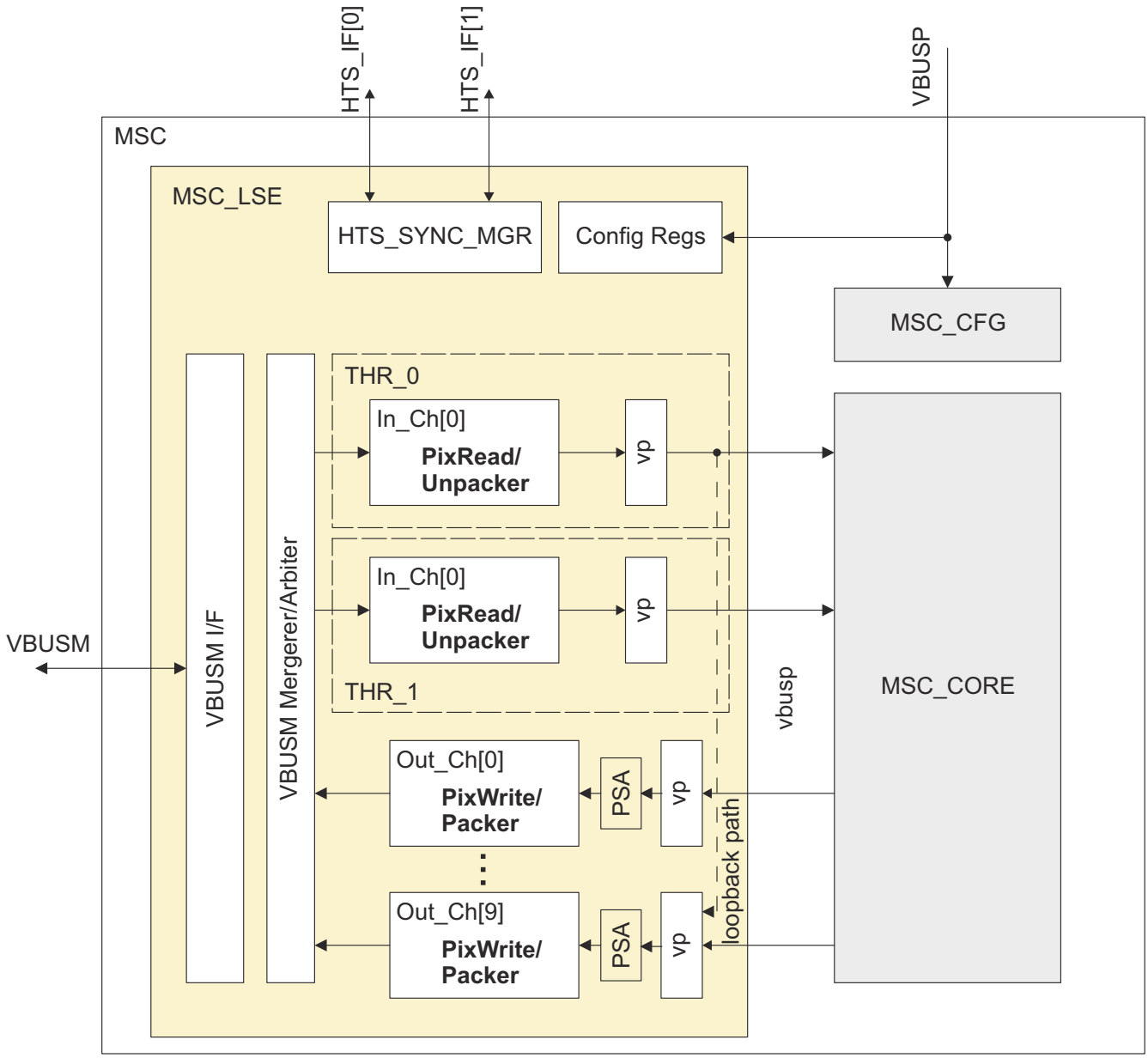


Figure 6-152. MSC_LSC Block Diagram

6.7.5.4.2.1.1 MSC_LSE Features

The MSC_LSE supports following VPAC_LSE features:

- Up to 2 processing threads supported (a thread is a processing chain with its own HTS start/done).
- Input channel features:
  - One to two SL2 input channels per thread with following features for the input channel:
    - Single data plane support
    - 8/12 packed and 12-bit unpacked source format support
    - Up to 5 lines - input kernel height support
    - 1D addressing mode only with CBUF address handling
    - Input Line Skip Support
    - Vertical boundary Edge Padding (Replication only) support
    - IndividualChannel enable
    - Multiple independent sub-channel support within each thread
      - Option to specify src_fmt for each channel

Copyright © 2024 Texas Instruments Incorporated

  - Option to skip line processing on UV channel when Y & UV channels are active.
  - Support for interleaved YUV22 or 2-plane YUV422/YUV420 and R+GB formats
- Output Channel Features:
  - Up to 10 SL2 output channels (pixel data output only)
  - Single data plane support for each output channel
  - Interleaved YUV422 output support - YUV422 Y and U/V outputs (2 designated output channels from the core) merged and output as a single YUV422 interleaved output. 8/12b component sizes supported.
  - Programmable thread mapping for each output channel
  - 8/12 packed and 12-bit unpacked output pixel data format support
  - 1D addressing mode support with CBUF address handling
  - Channel enable
  - Video frame data with frame sync signals (along with optional in-band control signals) over a separate vbusp streaming write-slave interface
- HTS synchronization support
- LSE internal data bypass mode support
  - Input channel 0 can be configured to bypass/loopback mode the data to the core and/or directly to the output channel 9 (bypassing the core).
- CRC frame data signature capture on all output channels for fault detection in the data path

6.7.5.4.2.2 MSC_LSE Internal Data Loopback Channel

The MSC_LSE provides a data loopback channel.

When enabled (VPAC_MSC_LSE_CFG_LSE[0] LOOPBACK_EN), the LSE sends the “loopback-enabled” input channel unpacked data directly to the designated output channel packing logic without going through the HWA_CORE. The looped back data can be packed in a different format in the output channel to perform a data format conversion or can be packed in the same format as the input.

When the loopback mode is enabled, the “loopback” designated output channel is no longer available for HWA_CORE output data transfer. But, the input channel can still operate normally to read in the data for the HWA_CORE (provided that the other output channel is used for transferring out the HWA_CORE output data) by setting VPAC_MSC_LSE_CFG_LSE[1] LOOPBACK_CORE_EN. In this mode, the input channel flow is throttled by both the core and the loopback data path.

The data flow of the loopback mode is also controlled by the HTS synchronization signals.

6.7.5.4.2.3 MSC_LSE PSA Support

As a safety function, LSE integrates a CRC signature capture mechanism on each output channel data to verify the integrity of the HWA internal paths. When enabled (VPAC_MSC_LSE_CFG_LSE[8] PSA_EN), every valid transfer frame data from the MSC is passed through a CRC module to update the signature for the channel before the data is packed and sent to the SL2 buffer. At the end of frame condition, the final signature is saved in a separate read-only signature register (VPAC_MSC_LSE_PSA_SIGNATURE_y[31-0] VAUE = 0,...,Number_of_Output Channel-1) till the next end of frame condition for software to read and compare it against the golden reference signature.

The PSA signature register is based on the following IEEE 802.3 standard 32-bit CRC polynomial:

$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + x + 1$

The PSA module is inserted in the LSE data path as shown in Figure 6-152.

6.7.5.4.2.4 MSC_LSE Feature Detailed Description

For detailed LSE feature description, see *Load Store Engine (LSE)*.

6.7.5.4.3 MSC_CORE (HWA Core)

The MSC_CORE (HWA_CORE) is configured as two multi-scaling engines each of which can perform 1-to many (multi) scaling operations on an independent input source. The number of outputs (scaler filters) assigned to each multi-scaling engine is user-configurable.

Copyright © 2024 Texas Instruments Incorporated

6.7.5.4.3.1 MSC_CORE Overview

Figure 6-153 shows the architectural overview of the MSC_CORE. Each reconfigurable multi-scaling engine in the MSC_CORE receives its input frame data from its own dedicated VP (vbusp write only slave) interface. VP_IN_0 port is used for the thread #0 (multi-scaling engine-0) and VP_IN_1 port is used for the thread #1 (multi-scaling engine-1). The THREAD_MAP parameter of the MSC_LSE output channel configuration register (VPAC_MSC_LSE_DST_BUF_CFG_j[7] THREAD_MAP) controls which input port is connected to which scaling filter (filter_0 ~ filter_9).

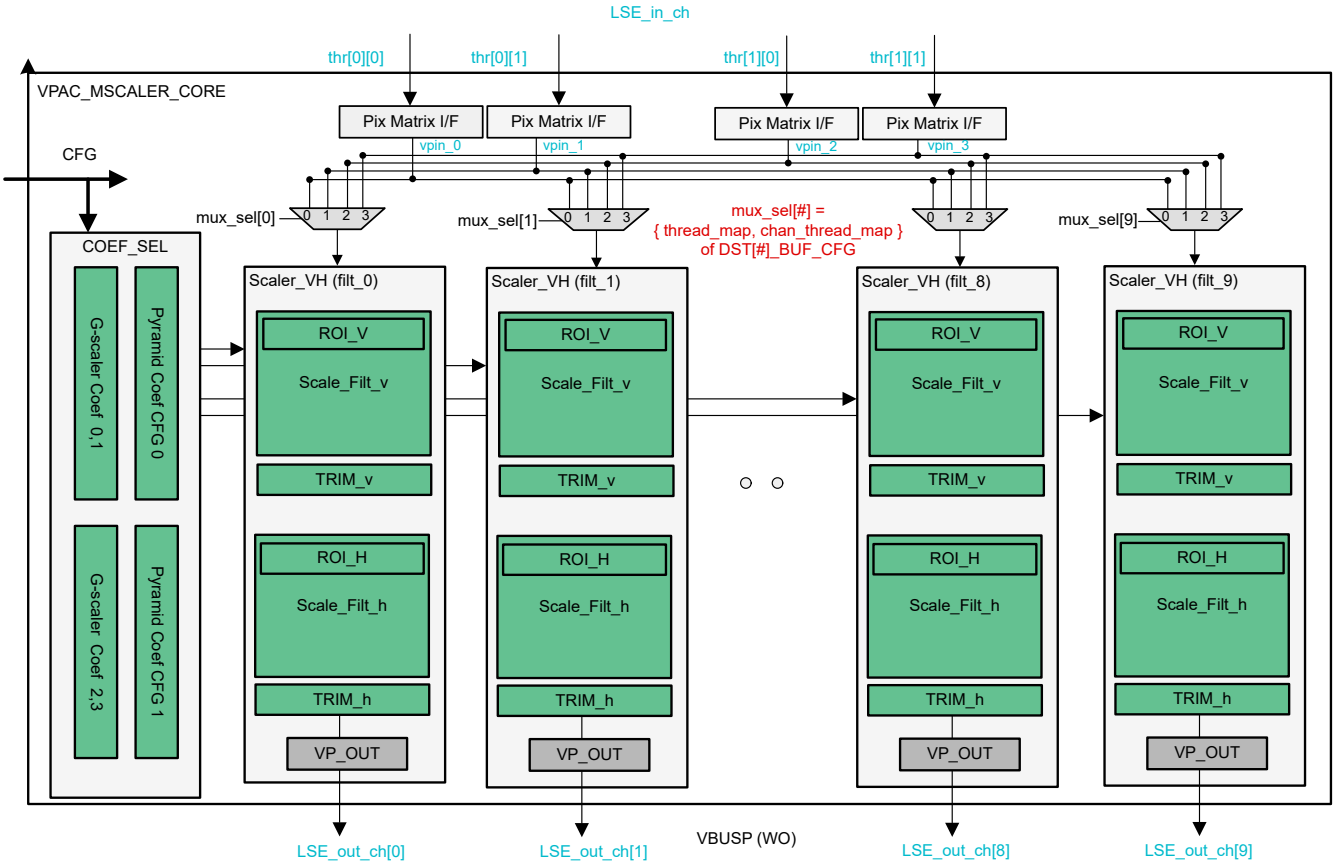


Figure 6-153. MSC_CORE Block Diagram

The scaling filter implements the vertical-before-horizontal architecture in order to avoid intermediate result line buffers that would be needed in the horizontal-before-vertical configuration. This architecture also allows the source frame width to be not restricted by internal line buffer sizes.

Each scaling filter in the MSC_CORE consists of 5-tap (32/64-phase) Polyphase filter based vertical and horizontal resizers. At the input of each resizer, a ROI checker determines the input start position (that is, marks the input data as valid). Output of each resizer is then trimmed to the configured output size of the scaling filter in each respective direction.

For MSC, the vertical input edge padding is performed in the MSC_LSE while the horizontal input boundary padding is done within the MSC_CORE before the horizontal resizer.

6.7.5.4.3.2 Polyphase Filter of Vertical/Horizontal Resizers

6.7.5.4.3.2.1 Filter Data Path Logic

Figure 6-154 shows the micro-architecture of the polyphase filter data path. The 5-tap filter structure can be programmed to perform a 5-tap Gaussian filter (for Octave generation) or a 4-tap bi-cubic downscaling filter (for scale generation). Or, it can be configured to perform a custom convolution filter without resizing. Filter coefficients for any unused taps should be set to 0. For example, in case of a 4-tap filtering, C2(Fi) filter coefficient should be set to 0. For 3-tap filtering, C-2(Fi) and C2(Fi) filter coefficients should be set to 0.

Copyright © 2024 Texas Instruments Incorporated

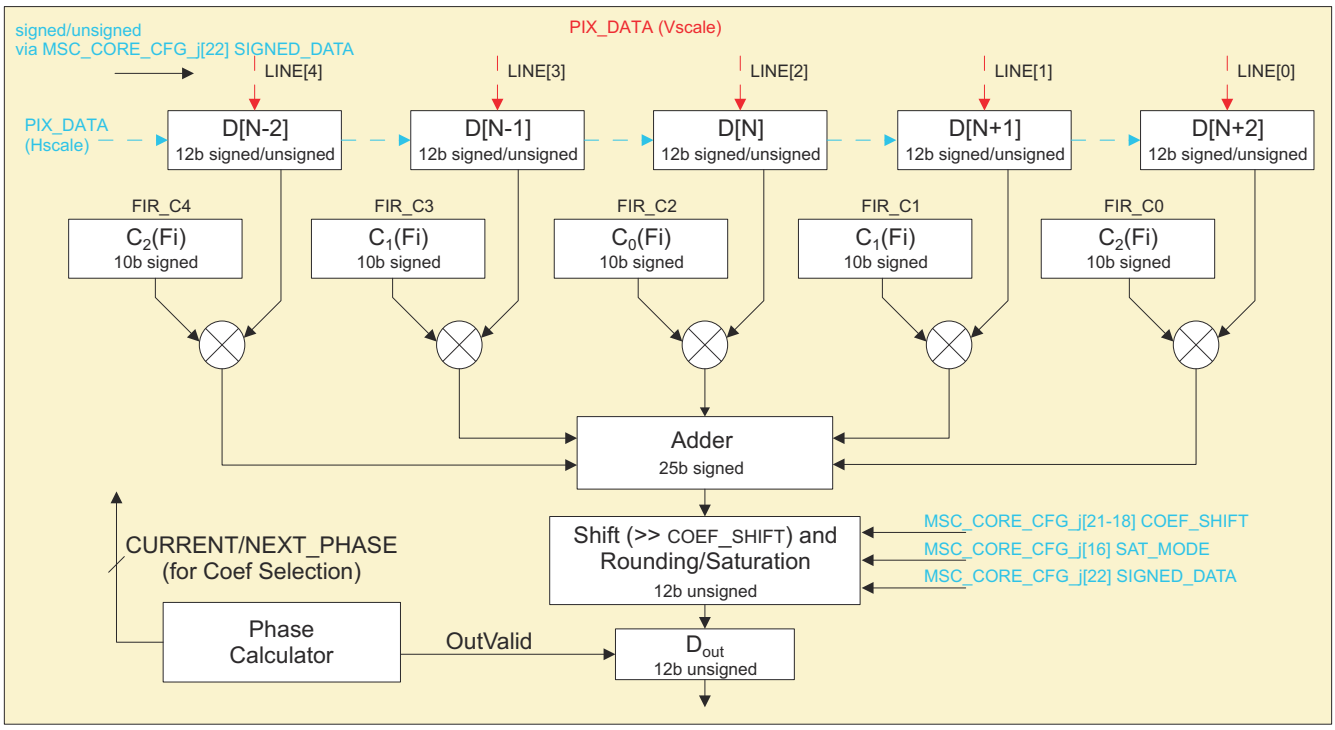


Figure 6-154. Polyphase Filter Micro-Architecture

Kernel Size Configuration

There is no separate configuration required to set the filter kernel size (number of taps) within the polyphase filter. The filter always works as a 5-tap filter but requires unused taps to be masked using the zero coefficients.

Coefficient Precision Selection

The precision of the coefficients is user-configurable using VPAC_MSC_CORE_CFG_j[21-18] COEF_SHIFT which determines the number of fractional bits of the coefficient.

Unsigned/Signed Data Type support

Typically for image resizing, the input and output data are in unsigned (positive) integer numbers in the range of [0..4095]. But, in some non-image convolution filtering (for example, Sobel filter), the input and output data may be in signed integer data format. The MSC supports a user-selectable data type mode bit (VPAC_MSC_CORE_CFG_j[22] SIGNED_DATA) per resizer to specify whether the input/output data is in signed or unsigned data format. The selection determines the input range and output clipping limits as shown below:

VPAC_MSC_CORE_CFG_0[22] SIGNED_DATA = 0 ->[0..4095]

VPAC_MSC_CORE_CFG_0[22] SIGNED_DATA = 1 -> [-2048..2047]

Note that all filters in the same processing thread should have the SIGNED_DATA bit set to a same value.

Rounding Logic

The rounding is done by “adding 0.5 and dropping fractional bits”. This achieves the following:

For data >= 0: “round to nearest, ties away from zero”

For data < 0: “round to nearest, ties toward zero”

This rounding is applied to the 12-bit MSC filter output data. If the result of a filter output is to be stored in SL2 as 8-bit data, an additional rounding in LSE output channel can be optionally enabled to apply rounding during 12bit to 8bit mapping (instead of simple truncation). See DST_BUF_CFG. enable_output_pixel_rounding in LSE_CFG.

Output Saturation Logic

Copyright © 2024 Texas Instruments Incorporated

For some non-image unsigned data filtering, the result of the final adder could be negative numbers. To avoid clipping all negative numbers to 0, the MSC filter data processing supports a mode (VPAC_MSC_CORE_CFG_j[22] SAT_MODE = 1) to offset the result by 2048 and then clip the final result to [0..4095] in the output saturation logic to preserve the full data range.

Edge Pixel Replication

Pixel replication in vertical direction (edge line replication) is done in the MSC_LSE.

Pixel replication in horizontal direction (edge pixel replication) is performed in the horizontal filter. For chroma data plane, Cb/Cr pixel replication is done independently with edge Cb/Cr data respectively.

6.7.5.4.3.2.2 Filter Phase Calculation

Figure 6-155 shows the polyphase filter phase calculation logic.

Each polyphase filter includes an accumulator (16b unsigned with 12b fractional bits) to support both the integer and fractional rate decimation filtering. ACC_INIT parameter, defined in VPAC_MSC_CORE_ACC_INIT_j registers for both for vertical and horizontal fitelring, is used to initialize the initial phase of the filter. FIRINC parameter, defined in VPAC_MSC_CORE_FIRINC_j registers for both for vertical and horizontal fitelring, is used to increment the accumulator after each valid output. The ACCUM, see Figure 6-155, is also decremented by 4096 (equivalent to 1 input sample period) after each valid input line/pixel.

The output of the filter is valid when the integer part of the ACCUM (bits[16-12]) is equal to 0 (meaning the center filter tap has the correct input data). Otherwise, the output of the filter is invalid and no data is written to the SL2 memory.

The fractional part is used to determine which coefficient phase to use. For 64-bit phase mode, COEF_PHASE_SELECT = ACCUM[11-6]. For 32-bit phase mode, COEF_PHASE_SELECT = ACCUM[11-7]. Additional fractional bits are included to minimize the input phase error.

Copyright © 2024 Texas Instruments Incorporated

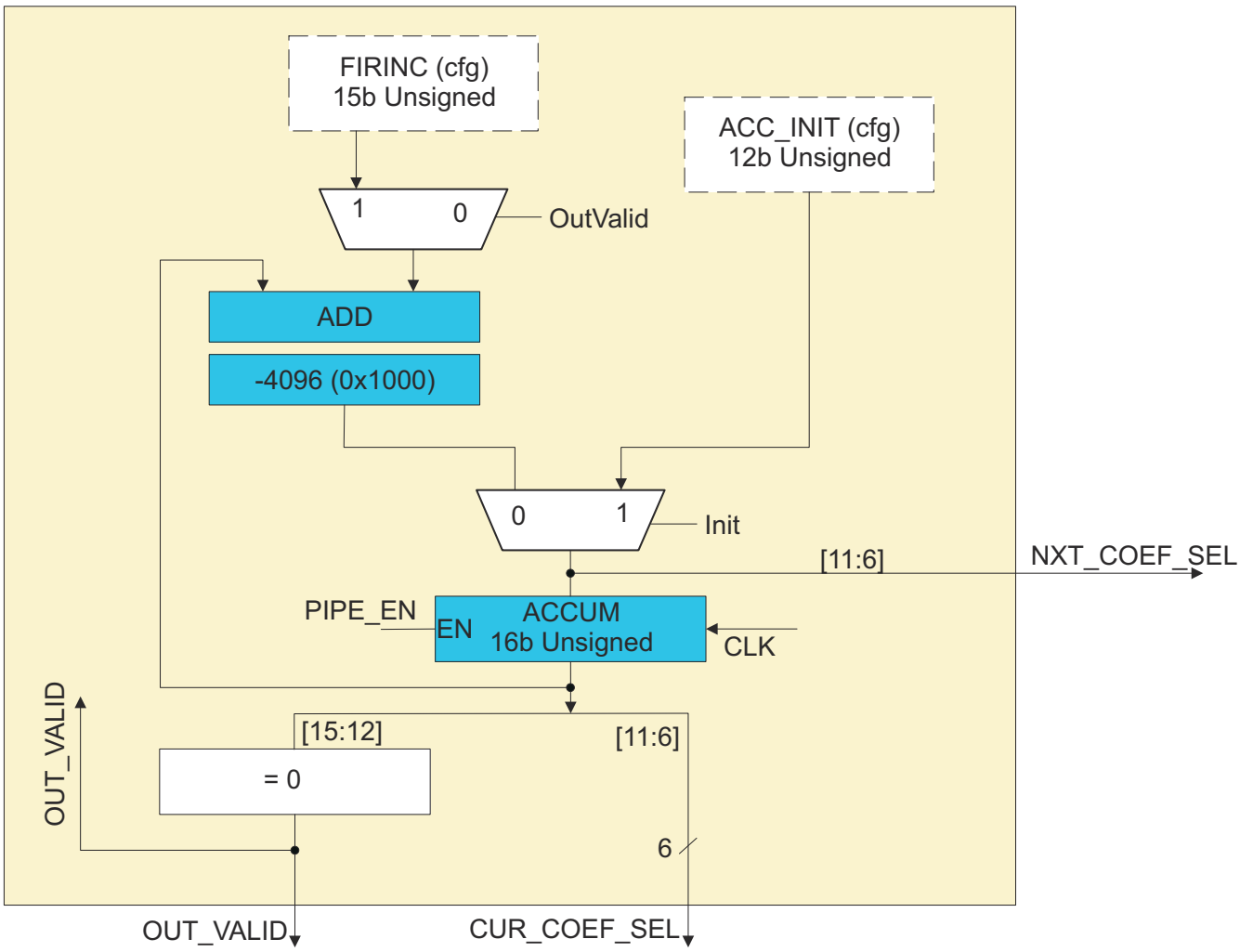


Figure 6-155. Polyphase Filter Phase Calculation Logic

NXT_COEF_SEL is used to pre-fetch the next coefficient from the table in order minimize the critical path delay (particularly for horizontal scaler in which coefficients change on every cycle).

6.7.5.4.3.2.3 Filter Parameters

Table 6-172 lists the parameters used to define the configuration of the resizing (downsampling) vertical/horizontal filters.

Table 6-172. Parameters of the Resizing (Downsampling) Vertical/Horizontal Filters

Parameter	Bit Precision	Description
FIRINC_V VPAC_MSC_CORE_FIRINC_j[30-16] VS	U15Q12 (Unsigned 15b number with 12-bit fraction)	Vertical Resizing Ratio value (vertical filter increment) Valid Range: 4096 (1x scaling) < FIRINC_V < 16384 (1/4x scaling) +1 in the above equation is optional. +1 forces non-repeating coefficients in integer ratio scaling cases.
FIRINC_H VPAC_MSC_CORE_FIRINC_j[14-0] HS	U15Q12 (Unsigned 15b number with 12-bit fraction)	Horizontal Resizing Ratio value (Horizontal filter increment)) Valid Range: 4096 (1x scaling) < FIRINC_V < 16384 (1/4x scaling)
ACC_INIT_V VPAC_MSC_CORE_ACC_INIT_j[27-16] VS	U12Q12 (Unsigned 12b number with 12b fraction)	Initial Vertical Resizer Phase Valid Range: 0 < ACC_INIT_V < 4095

Copyright © 2024 Texas Instruments Incorporated

Table 6-172. Parameters of the Resizing (Downsampling) Vertical/Horizontal Filters (continued)

Parameter	Bit Precision	Description
ACC_INIT_H VPAC_MSC_CORE_ACC_INIT_j[11-0] HS	U12Q12 (Unsigned 12b number with 12b fraction)	Initial Horizontal Resizer Phase Valid Range: 0 < ACC_INIT_V < 4095

Additionally, Table 6-173 lists the mode parameters defined the filter operation and coefficient selection.

Table 6-173. Filter Mode and Coefficient Selection via VPAC_MSC_CORE_CFG_j Registers

Parameter	Bit Width	Description
SIGNED_DATA VPAC_MSC_CORE_CFG_j[22] SIGNED_DATA	1-bit	Integer type of input and output frame data: 0 : Unsigned 12-bit (default) 1 : Signed 12-bit
UV_MODE VPAC_MSC_CORE_CFG_j[17] UV_MODE	1-bit	Source data interleave format: 0 - non-interleaved (Y data) 1 - interleaved (UV data)
SP_VS_COEF_SEL VPAC_MSC_CORE_CFG_j[15-12] SP_VS_COEF_SEL	4-bits	Single Phase – Vertical Filter Coef Selection N : 5-tap/32-phase Filter Coef Set #0 (Entry N) where N=0-9 10 : Gaussian Filter #0 11 : Gaussian Filter #1 12-15 : Invalid When FilterMode=0 and N <10, then the multi-phase coef set #0 is dedicated for one or more single phase filters.
SP_HS_COEF_SEL VPAC_MSC_CORE_CFG_j[10-7] SP_HS_COEF_SEL	4-bits	Single Phase – Horizontal Filter Coef Selection N : 5-tap/32-phase Filter Coef Set #0 (Entry N+10) where N=0-9 10 : Gaussian Filter #0 11 : Gaussian Filter #1 12-15 : Invalid When FilterMode=0 and N <10, then the multi-phase coef set #0 is dedicated for one or more single phase filters.
VS_COEF_SEL VPAC_MSC_CORE_CFG_j[5-4] VS_COEF_SEL	2-bits	Multi-phase Vertical Coef Selection 00 : 5-tap/32-phase Filter coef set #0 01 : 5-tap/32-phase Filter coef set #1 10 : 5-tap/32-phase Filter coef set #2 11 : 5-tap/32-phase Filter coef set #3
HS_COEF_SEL VPAC_MSC_CORE_CFG_j[3-2] HS_COEF_SEL	2-bits	Multi-phase Horizontal Coef Selection 00 : 5-tap/32-phase Filter coef set #0 01 : 5-tap/32-phase Filter coef set #1 10 : 5-tap/32-phase Filter coef set #2 11 : 5-tap/32-phase Filter coef set #3
PHASE_MODE VPAC_MSC_CORE_CFG_j[1] PHASE_MODE	1-bit	Filter Phase mode selection 0 - 64 phases 1 - 32 phases
FILTER_MODE VPAC_MSC_CORE_CFG_j[0] FILTER_MODE	1-bit	Filter Mode 0 – Single Phase Filter (for example, Gaussian Filter for Pyramid) 1 – Multi-phase Scaling Filter
FILT_SAT_MODE VPAC_MSC_CORE_CFG_j[16] SAT_MODE	1-bit	Filter Output Saturation Mode 0 - [0..4095] clipping 1 - [-2048.. 2047] clip followed by +2048 This parameter is used only when signed_data =0 (unsigned data type).

Copyright © 2024 Texas Instruments Incorporated

Table 6-173. Filter Mode and Coefficient Selection via VPAC_MSC_CORE_CFG_j Registers (continued)

Parameter	Bit Width	Description
COEF_SHIFT VPAC_MSC_CORE_CFG_j[21-18] COEF_SHIFT	4-bits	Coef Shift Size: configures the precision of the 10-bit signed filter coefficients (valid Shift range: 5~9) : 5: Shift by 5 (5-bit fraction) 6: Shift by 6 (6-bit fraction) 7: Shift by 7 (7-bit fraction) 8: Shift by 8 (8-bit fraction) 9: Shift by 9 (9-bit fraction) Integer size = 9 - #_of_fraction_bits

6.7.5.4.3.2.4 Single-Phase Filter Parameters

For a single-phase filter configuration (VPAC_MSC_CORE_CFG_j[0] FILTER_MODE = 0), software must ensure that FIRINC parameters (VPAC_MSC_CORE_FIRINC_j[30-16] VS and VPAC_MSC_CORE_FIRINC_j[14-0] HS) are programmed with one of the following values only:

- 0x4000 (1/4 X resizing)
- 0x2000 (1/2 X resizing)
- 0x1000 (1X – no scaling)

VPAC_MSC_CORE_ACC_INIT_j[27-16] VS and VPAC_MSC_CORE_ACC_INIT_j[11-0] HS configuration values are ignored by the MSC hardware (internally set to 0x0) when VPAC_MSC_CORE_CFG_j[0] FILTER_MODE = 0. If any phase offsets are needed, the starting pixel/line locations of the frame and the filter coefficient values should be adjusted to implement the desired phase shift.

6.7.5.4.3.2.5 Interleaved Mode Handling

When the input is in the interleaved data format (for example, YUV420 Chroma data), the horizontal filter supports two sets of 5-input buffers (Cb and Cr buffers) and alternates filtering operation between two. Phase calculation and coefficient updates are done every other sample. Edge replications at the start of the line and at the end of the line are handled separately for Cb and Cr data.

There is no change in how the vertical filter handles an interleaved data format source.

6.7.5.4.3.2.6 Input Skip Line Support

When a processing thread is only performing an octave generation (1/2x single phase resizing), the next input lines can be offset by 2 lines since the output is only generated every other input line times. By skipping lines (VPAC_MSC_LSE_SRC_CFG_j[7] SRC_LN_INC_2), the SL2 access is reduced in half for this mode and the cycle time to complete an Octave generation is equal to ½ of the input frame size.

Line N: Read Line N-2, N-1, N, N+1, N+2

Line N+1: Skip

Line N+2: Read Line N, N+1, N+2, N+3, N+4

Line N+3: Skip

Even though “N+1” line processing is skipped, all source lines are still required for proper filtering. Therefore, the DMA transfer from DDR to SL2 should include all lines. VPAC_MSC_CORE_FIRINC_j[30-16] VS should be set to 4096 (1x resizing) to compensate for source line skip (if it is doing a ½ resizing).

Horizontal Skip is done normally as ½ x resizing by the core HScale Filter.

6.7.5.4.3.3 Scaler Filter Thread Mapping

The scaling filter_# (# = 0..9) is enabled and its input is connected to one of the processing thread source VP ports (VP_IN_0 ,VP_IN_1 , VP_IN_2 or VP_IN_3) by configuring the output channel #n buffer configuration:

Copyright © 2024 Texas Instruments Incorporated

LSE_CFG.DST[#]_BUF_CFG.thread_map and chan_thread_map: define the mapping of input thread/source to the filter #.

LSE_CFG.DST[#]_BUF_BA.enable : enables both filter[#] and LSE output channel[#]

By mapping each output to a thread explicitly in the output channel configuration register, MSC guarantees that a filter can only be connected to one processing thread.

6.7.5.4.3.4 Filter Coefficients

Since each resizer needs to access the common coefficients independently, the coefficients are stored in registers (instead of memory). Each resizer selects directly from these register using the coefficient selection scheme as shown in Figure 6-156. The same mux is used to select either vertical or horizontal coefficients.

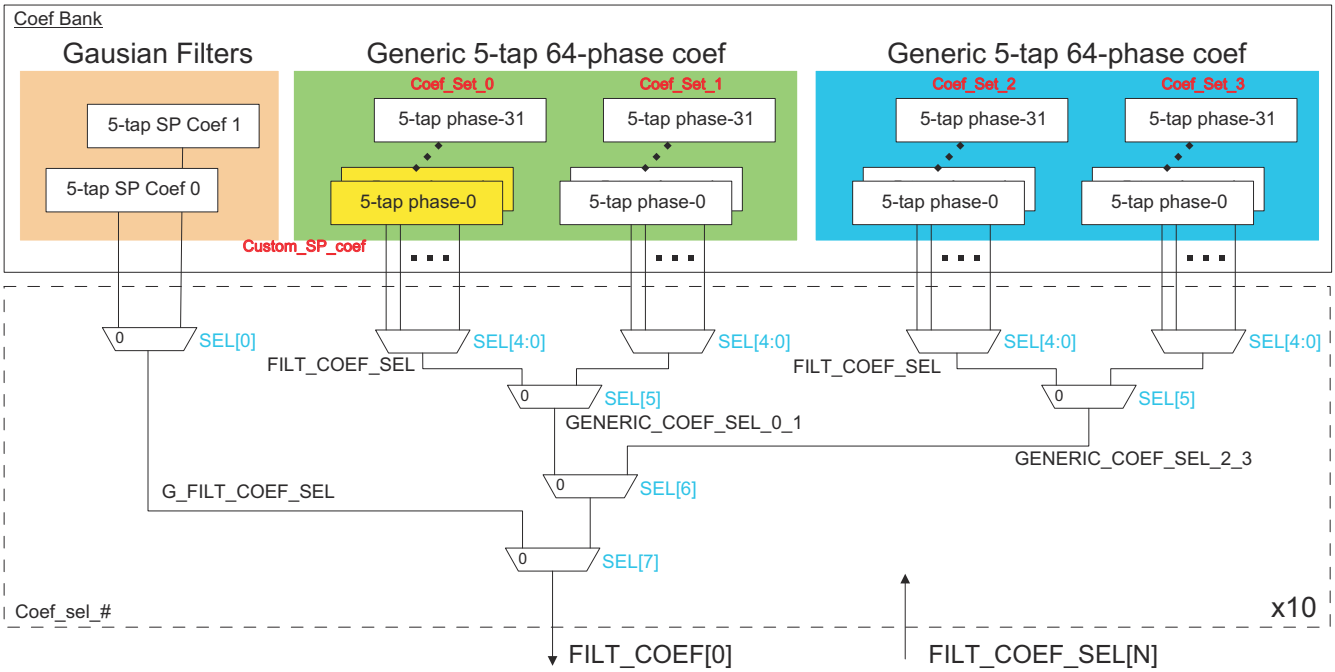


Figure 6-156. Coefficient Selection

6.7.5.4.3.4.1 Filter Coefficient Selection Algorithm

Filter Coefficient Selection is determined using the following algorithm:

```
If
( cfg.filter_mode == 0 ) {
  //
single phase mode
If( vertical_filter )  coef_sel = { cfg.sp_vs_coef_src,cfg.sp_vs_coef_sel}
else coef_sel = { cfg.sp_hs_coef_src, cfg.sp_hs_coef_sel}
   if( coef_sel[4]  == 0 )   filt_coef_sel        =   { 7'b0, coef_sel[0]}    //  use dedicated
Gaussian filter set
          else if ( vertical_filter)   filt_coef_sel =   { 4'b1001, coef_sel[3:0] }   // use one of
second 16 other custom coefs

                else filt_coef_sel =  { 4'b1000,  coef_sel[3:0] }   // use one of 16 other custom
coefs
}
else
{
  //
multiphase mode
   If( vertical_filter ) coef_sel = cfg.vs_coef_sel[1:0]

 else coef_sel = cfg.hs_coef_sel[1:0]
   If ( cfg.phase_mode == 0 )  coef_sel = { 1'b1, coef_sel[1], filter_phase_sel[5:0] } // 64-phase
mode
```


Copyright © 2024 Texas Instruments Incorporated

```
                else coef_sel = { 1'b1, coef_sel[1:0],  filter_phase_sel[5:1]}  // 32-phase
mode
}
```

where filter_phase_sel comes from the Phase Calculation Logic.

The vertical resizer selects one set of coefficients at the start of each line, saves them locally in the vertical scaler to use the same coefficients for the entire pixels in the line.

The Horizontal resizer selects a new coefficient on each input pixel processing. To reduce timing dependency, the horizontal coefficient selection is made with next phase value from the horizontal scaler's phase calculator one cycle ahead of actual filter calculation.

6.7.5.4.3.4.2 Filter Coefficient Parameter Configuration

Each coefficient table entry consists of 5 coefficients stored in two memory mapped configuration registers - VPAC_MSC_CORE_C210_j and VPAC_MSC_CORE_C43_j. FIR_C4-0 parameters map to C-2, C-1, C0, C1, and C2 coefficients (see Figure 6-153).

6.7.5.4.3.4.3 3/4/5-Tap Filter Configuration

All filters operate in 5-tap filter mode – requiring 5-tap coefficients regardless of the filter usage. For less-than-5 tap filter configurations, coefficients corresponding to the unused taps should be set to 0.

For example, FIR_C0 and FIR_C4 will need to be set to 0x0 for a 3-tap configuration.

For 4-tap configuration, FIR_C0 will need to be set to 0x0.

6.7.5.4.3.5 Input/Output ROI Trimmers

Each scaler filter implements the ROI scaling as shown below with input and output trimmers. The input trimmer simply marks the beginning of the ROI to indicate where the valid source pixels and lines start in the source image. Doing the source ROI marker allows the exact specification of a sub-image in a common source image (no position or size restriction) while keeping the original boundary pixels (that is, not use the replicated pixels/ lines).

The output trimmers then uses destination size parameters (VPAC_MSC_CORE_OUT_SIZE_j[28-16] HEIGHT and VPAC_MSC_CORE_OUT_SIZE_j[12-0] WIDTH) to trim out the exact final output image sizes.

VPAC_MSC does not perform output size correction by stuffing extra pixels in the case the programmed VPAC_MSC_CORE_OUT_SIZE_j[28-16] HEIGHT and VPAC_MSC_CORE_OUT_SIZE_j[12-0] WIDTH exceeds the actual output size of the scaler. In this case, VPAC_MSC simply outputs the actual size.

6.7.5.5 MSC Performance

Since the resizer performs only downscaling operations, the throughput of scaler is mostly dictated by the input processing rate – 1 input pixel per 1 clock cycle. Once the data is made available to the scaler filter after the initial set up and data fetching, the scaler should be fully active except for few cycles at the end of each line to flush the line.

If a processing thread is performing a pyramid generation only, the vertical line skip feature can be enabled to reduce the frame processing time approximately by ½.

6.7.5.6 MSC Clocking

Complete VPAC_MSC module operates on single clock - VPAC0_MSC_CLK.

6.7.5.7 MSC Reset

VPAC_MSC has one synchronous active low reset input. The entire module is reset by this reset.

6.7.5.8 MSC Programmer's Guide

6.7.5.8.1 Programming Model

6.7.5.8.1.1 MSC Programming Guidelines

MSC does not support shadow registers for its configuration registers. All changes must be done prior to the HTS.INIT request for a frame processing.

Copyright © 2024 Texas Instruments Incorporated

Input and output channels are individually enabled. It is expected that at least one input and output channels are enabled for each HTS thread enabled.

The MSC does not check for proper programming of all configuration parameters. It is strictly software responsibility to ensure programming of MSC (Core and LSE) registers do not cause conflict with each other.

6.7.5.8.1.2 MSC_Core Programming Details

All filters enabled for a multi-scaling thread are programmed via:

- VPAC_MSC_CORE_CFG_j[0] FILTER_MODE; Coefficient Set Selection - VPAC_MSC_CORE_CFG_j[3-2] HS_COEF_SEL, VPAC_MSC_CORE_CFG_j[5-4] VS_COEF_SEL, VPAC_MSC_CORE_CFG_j[10-7] SP_HS_COEF_SEL, VPAC_MSC_CORE_CFG_j[15-12] SP_VS_COEF_SEL; and filter tap size parameters VPAC_MSC_CORE_CFG_j[11] SP_VS_COEF_SRC and VPAC_MSC_CORE_CFG_j[6] SP_HS_COEF_SRC.
- Coefficients set in VPAC_MSC_CORE_C210_j, VPAC_MSC_CORE_C43_j, VPAC_MSC_CORE_C210_j_k, VPAC_MSC_CORE_C43_j_k registers.
- ROI source offset - VPAC_MSC_CORE_SRC_ROI_j[28-16] Y_OFFSET and VPAC_MSC_CORE_SRC_ROI_j[12-0] X_OFFSET, and size. The size of ROI depends on the X and Y offsets. For full-size set the offsets to 0. Note that the ROI_SIZE may or may not be same as the MSC frame size (set in MSC_FRAME_SIZE_j[28-16] HEIGHT and MSC_FRAME_SIZE_j[12-0] WIDTH) - depending on whether ROI is full or sub-frame sized.
- Output size - VPAC_MSC_CORE_OUT_SIZE_j[28-16] HEIGHT and VPAC_MSC_CORE_OUT_SIZE_j[12-0] WIDTH.
- Filter parameters - VPAC_MSC_CORE_FIRINC_j[14-0] HS, VPAC_MSC_CORE_FIRINC_j[30-16] VS, VPAC_MSC_CORE_ACC_INIT_j[27-16] VS, and VPAC_MSC_CORE_ACC_INIT_j[11-0] HS.

Filter thread mapping is done with output channel configuration parameters (MSC_BUF_CFG_j[7] THREAD_MAP). Specifying the mapping in one register for each channel prevents any resource conflict.

6.7.5.8.1.3 MSC_LSE Programming Details

Two processing threads can be activated independently.

6.7.5.8.1.3.1 Input Thread Configuration:

1. Pixel Data Format
   a. VPAC_MSC_LSE_SRC_CFG_j[1-0] PIX_FMT_PW, VPAC_MSC_LSE_SRC_CFG_j[3-2] PIX_FMT_CNTRSZ, and VPAC_MSC_LSE_SRC_CFG_j[4] PIX_FMT_ALIGN parameters define the pixel input data format for all input channels of the thread.
   b. Common data formats:
      i. Fully packed 12-bit: PIX_FMT_PW = 1, PIX_FMT_CNTRSZ = 1, PIX_FMT_ALIGN = 0
      ii. Fully packed 8-bit: PIX_FMT_PW = 0, PIX_FMT_CNTRSZ = 0, PIX_FMT_ALIGN = 0
2. Pixel Input Matrix Configuration
   a. VPAC_MSC_LSE_SRC_CFG_j[15-12] KERN_LN_OFFSET defines the starting line offset number of the input pixel matrix (for 5-tap filter configuration, offset = 0, for 4-tap configuration, offset = 1)
   b. VPAC_MSC_LSE_SRC_CFG_j[11-8] KERN_SZ_HEIGHT defines the height of the input pixel matrix (for example, MSC = 5, VISS = 1, NF = 5)
   c. VPAC_MSC_LSE_SRC_CFG_j[21-19] KERN_TPAD_SZ, and VPAC_MSC_LSE_SRC_CFG_j[18-16] KERN_BPAD_SZ defines the number of padding lines required at top and bottom of the frame (for example, for 5-tap filter configuration for MSC, both of these parameters are set to 2. For 4-tap filter configuration for MSC, TPAD_SZ =1 while BPAD_SZ = 2).
3. Source Frame Size
   a. VPAC_MSC_LSE_SRC_FRAME_SIZE_j[28-16] HEIGHT and VPAC_MSC_LSE_SRC_FRAME_SIZE_j[12-0] WIDTH – define the width and height (pixels/lines) of the source video frame.
4. Source SL2 Circular Buffer Configuration
   a. VPAC_MSC_LSE_SRC_BUF_ATTR_j[15-6] BUF_STRIDE – defines the line stride size (must be 64 byte multiple)

Copyright © 2024 Texas Instruments Incorporated

   b. VPAC_MSC_LSE_SRC_BUF_ATTR_j[24-16] CBUF_SIZE – define the size of the circular buffer in the SL2 (number of lines)
   c. VPAC_MSC_LSE_SRC_BUF_ATTR_j[31-25] START_NIB_OFFSET – defines the line start offset in smaller resolution (start address within the first SL2 512-bit word in 4-bit resolution).
5. Source BA and Enable Configuration
   a. VPAC_MSC_LSE_SRC_i_BUF_BA_y[23-6] ADDR – defines the SL2 base address of the Circular buffer for the input channel. VPAC_MSC_LSE_SRC_i_BUF_BA_y[31] ENABLE – enables the input channel.

6.7.5.8.1.3.2 Output Channel Configuration

1. Pixel Data Format
   a. VPAC_MSC_LSE_DST_BUF_CFG_j[1-0] PIX_FMT_PW, VPAC_MSC_LSE_DST_BUF_CFG_j[3-2] PIX_FMT_CNTRSZ, and VPAC_MSC_LSE_DST_BUF_CFG_j[4] PIX_FMT_ALIGN parameters define the pixel output data format for this output channel.
   b. Common data formats:
      i. Fully packed 12-bit: PIX_FMT_PW = 1, PIX_FMT_CNTRSZ = 1, PIX_FMT_ALIGN = 0
      ii. Fully packed 8-bit: PIX_FMT_PW = 0, PIX_FMT_CNTRSZ = 0, PIX_FMT_ALIGN = 0
2. Output Thread Map
   a. VPAC_MSC_LSE_DST_BUF_CFG_j[7] THREAD_MAP defines the output channel thread mapping.
3. Output SL2 Circular Buffer Configuration
   a. VPAC_MSC_LSE_DST_BUF_ATTR0_j[15-6] BUF_STRIDE – define the line stride size (must be 64 byte multiple)
   b. VPAC_MSC_LSE_DST_BUF_ATTR0_j[24-16] CBUF_SIZE – define the size of the circular buffer in the SL2 (number of lines)
4. Base Address and Channel Enable:
   a. VPAC_MSC_LSE_DST_BUF_BA_j[23-6] ADDR – defines the SL2 base address of the Circular buffer for this output channel.
   b. VPAC_MSC_LSE_DST_BUF_BA_j[31] ENABLE – enables this output channel.

6.7.5.8.1.4 MSC HTS Programming Details

There is no configuration required for HTS interface in the MSC_HWA. All programming is done in the separate HTS module. It is expected that at least one input and output channel is enabled for each HTS thread. For more details on programming HTS, see *Hardware Accelerator (HWA) Thread Scheduler (HTS)*.

6.7.5.8.1.5 MSC Data Transfer Programming Details

Refer to *VPAC Subsystem Programmer's Guide*.

6.7.5.8.1.6 LSE Interrupt Programming

MSC does not have any interrupt MASK/SET/CLEAR registers . Interrupts are pulse output events that go to the VPAC top level interrupt aggregation logic. For more information about the aggregation logic, see Section 6.7.2, *VPAC Subsystem*.

MSC does, however, provide a set of statuses for VBUSM interface errors. These status are mapped to VPAC_MSC_STATUS_ERROR and they are cleared by writing all 1's to the status bit field.

6.7.5.8.2 Initialization Sequence

For details, see Section 6.7.2, *VPAC Subsystem Level*.

6.7.5.8.3 Real-Time Operating Requirements

For details, see Section 6.7.2, *VPAC Subsystem Level*.

6.7.5.8.4 Power Up/Down Sequence

For details, see Section 6.7.2, *VPAC Subsystem Level*.

Copyright © 2024 Texas Instruments Incorporated

6.7.6 VPAC Noise Filter (NF)

6.7.6.1 NF Overview

The noise filter in VPAC is used for filtering out spatial noise without impacting edges in image.

6.7.6.1.1 NF Supported Features

- Bilateral filtering:
  - Supports filter size up to 5×5
  - Supports true 2D Bilateral filtering
- Generic filtering:
  - Supports filter size up to 5×5 of programmable static weights
- LUT based Bilateral weights generation:
  - 8-bit for weight
- Supported formats:
  - YUV420,12-bit
  - One plan (interleaved plane and non-interleaved)
- Performance: 1 cycle/pixel
- Line based Input and Output
- Memory-to-memory operation
- Support for ROI:
  - Both Input and Output can be handled via DMA

6.7.6.1.2 NF Not Supported Features

- Filter size higher than 5×5

6.7.6.2 NF Functional Description

6.7.6.2.1 Functional Operation

The noise filter hardware (NF HW) block reads data from memory (DDR or on-chip) to shared memory (SL2 with the help of DMA) and does Bilateral filtering to remove noise. The output of NF HW block can be sent to external memory (DDR) from shared memory (SL2) or can be further re-sized using Scalar hardware.

6.7.6.2.1.1 Overview

Figure 6-157 shows a block diagram of Bilateral noise filter with support for filtering size up to 5×5.

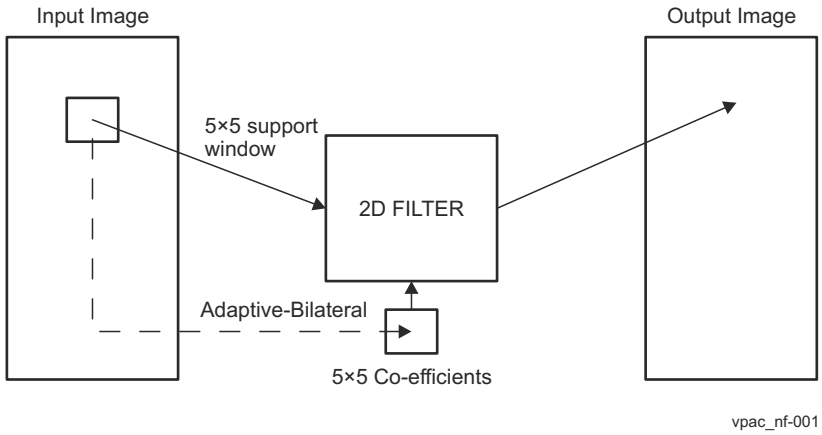


Figure 6-157. Bilateral Noise Filter Block Diagram

6.7.6.2.1.2 NF Integration In VPAC

Figure 6-158 presents VPAC block diagram.

It consists of four hardware module blocks and three infrastucture module blocks.

Copyright © 2024 Texas Instruments Incorporated

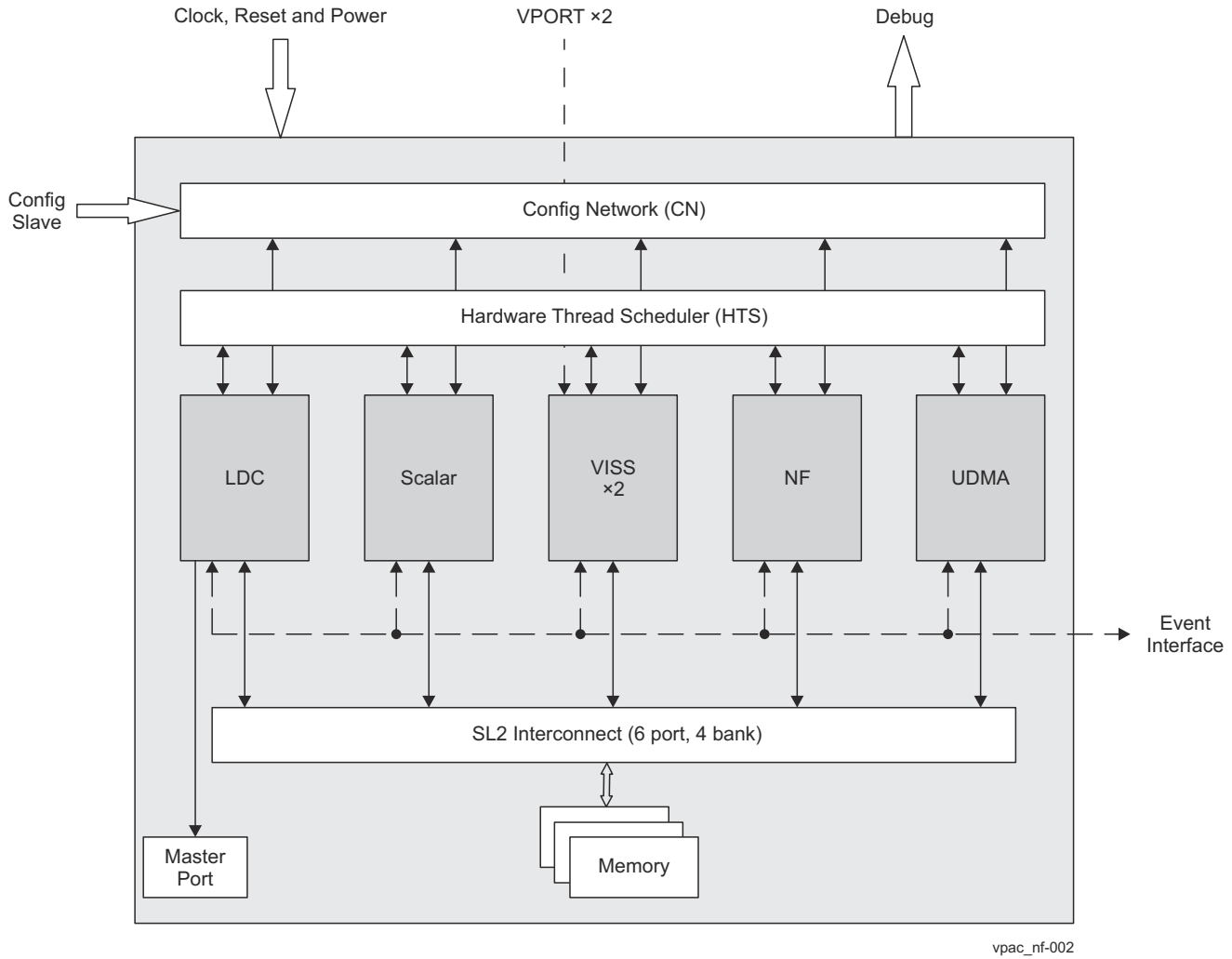


Figure 6-158. VPAC Block Diagram

VPAC hardware blocks:

1. LDC (Lens Distortion Correction): LDC hardware block reads data from memory (DDR or on-chip) and applies perspective transform as well as correction of lens distortion (including fisheye lens). The output of LDC block can be sent to external memory (DDR) or sent to other hardware module (Scalar, Noise filter) for further pre-processing via local shared memory (SL2).
2. Scalar (or Downsampler or Resizer): Scalar hardware block reads data from memory (DDR or on-chip) to shared memory (SL2) and generates up to 10 scaled output from two inputs with varios scaling ratios (between × and ×0.25). The output of Scalar block can be sent to external memory (DDR) from shared memory (SL2) or can be further noise filtered using Noise filter hardware.
3. Noise Filter (NF): NF hardware block reads data from memory (DDR or on-chip) to shared memory (SL2) and does Bilateral filtering to remove noise. The output of NF block can be sent to external memory (DDR) from shared memory (SL2) or can be further re-sized using Scalar hardware.
4. VISS (Vision ISS): There are two instances of on-the-fly processing for sensor related processing in VISS hardware.

VPAC infrastucture blocks:

1. Shared Level 2 (SL2) memory: It is used to exchanging data across hardware module block (for example: LDC, Scalar and NF) as well as to DMA engine (for example: UDMA).
2. Hardware Thread Scheduler (HTS): It is used for IPC communication among various hardware module block for example: LDC, Scalar and NF) as well as to DMA engine (for example: UDMA).
3. DMA engine (UDMA consisting of CC + DRU) is inside VPAC boundary. It consists of one CC and two TC.

Copyright © 2024 Texas Instruments Incorporated

6.7.6.2.1.3 Algorithm Details

The basic equation for Bilateral filtering is shown in Figure 6-159.

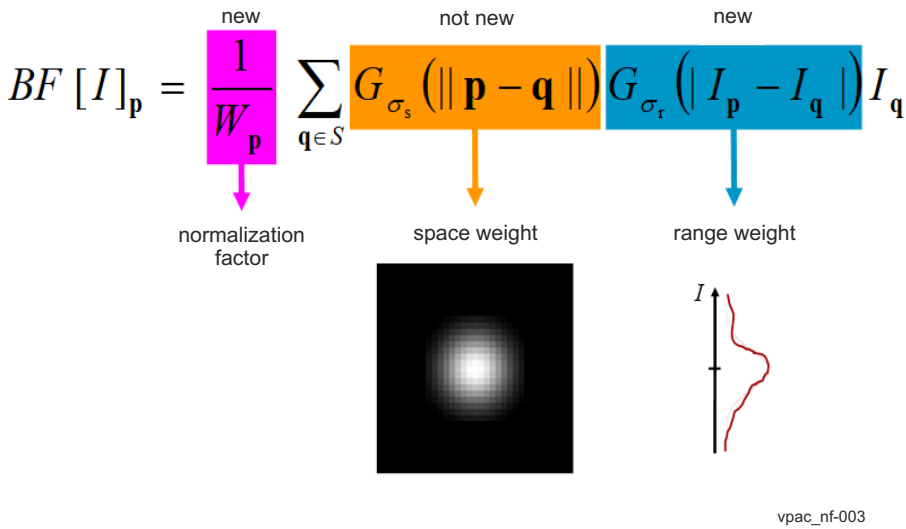


Figure 6-159. Bilateral Filtering Equation

The range and space parameters are shown in Figure 6-160.

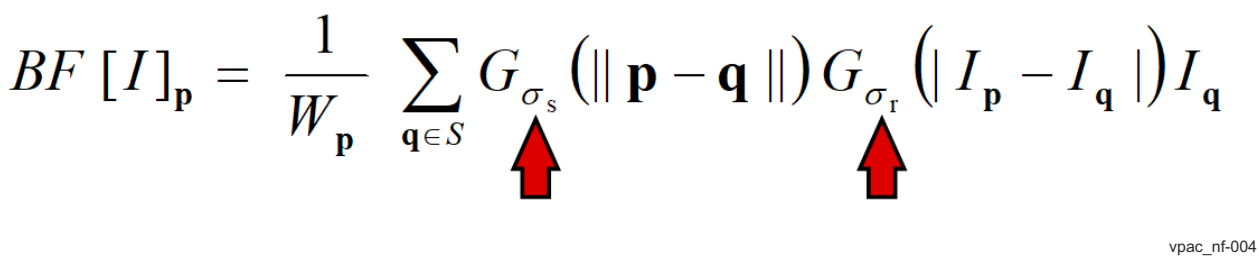


Figure 6-160. Range and Space Parameters

Range and space parameters:

- space ss: spatial extent of the kernel, size of the considered neighborhood.
- range sr: "minimum" amplitude of an edge.

6.7.6.2.1.4 Data Format Support In VPAC

Table 6-174 summarizes data format support in VPAC that needs to be supported by NF hardware block.

Table 6-174. Data Format Across VPAC

No	Type	Bit-Depth	Chroma Format	Packing	Reasoning
1.	LDC Input	8-bit	YUV422	N/A	External sensor ISP
		12-bit	YUV420	Fully packed	
		12-bit	YUV420	Un-packed	Other source in chip
2.	LDC Output	12-bit	YUV420	Fully packed	
3.	NF Input	12-bit	YUV420	Fully packed	
4.	NF Output	12-bit	YUV420	Fully packed	
5.	Scalar Input	12-bit	YUV420	Fully packed	One plane at a time
6.	Scalar Output	12-bit	YUV420	Fully packed	One plane at a time

Figure 6-161 presents YUV420 2-plane 12-bit fully packed format.

Copyright © 2024 Texas Instruments Incorporated

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Y2[7:0]								Y1												Y0												+ 0x0
Y5[3:0]				Y4												Y3												Y2[11:8]				+ 0x4
Y7												Y6												Y5[11:4]								+ 0x8
Y10[7:0]								Y9												Y8												+ 0xC
Y13[3:0]				Y12												Y11												Y10[11:8]				+ 0x10
Y15												Y14												Y13[11:4]								+ 0x14

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Cb1[7:0]								Cr0												Cb0												+ 0x0
Cr2[3:0]				Cb2												Cr1												Cb1[11:8]				+ 0x4
Cr3												Cb3												Cr2[11:4]								+ 0x8
Cb5[7:0]								Cr4												Cb4												+ 0xC
Cr6[3:0]				Cb6												Cr5												Cb5[11:8]				+ 0x10
Cr7												Cb7												Cr6[11:4]								+ 0x14

vpac_nf-005

Figure 6-161. YUV420 (2-Plane 12-bit Fully Packed Format)

6.7.6.3 NF Interrupts

6.7.6.3.1 CPU Interrupts

The following interrupts are generated by the NF module (see Table 6-175):

Table 6-175. Interrupts

Interrupt	Description
NF_FRAME_DONE	Frame processing complete.
NF_SL2_READ_ERROR	Set whenever there is an error response on VBUSM read command request for any input channel.
NF_SL2_WRITE_ERROR	Set whenever there is an error response on VBUSM write command request for any output channel.

All interrupts are single pulse event signals that are mapped to the VPAC level interrupt aggregation logic.

6.7.6.3.2 Interrupt Event Description

6.7.6.3.2.1 NF_FRAME_DONE Event

These events are generated when noise filtering is complete for entire frame. This is set when input end of frame (EOF) event and output EOF events are detected.

6.7.6.3.2.2 NF_SL2_READ_ERROR Event

This event is generated whenever the VBUSM read status is something other than '0' (success).

The VPAC status register (VPAC_NF_LSE_STATUS_ERROR) stores the last non-zero rstatus value for user to see which type of error has occurred.

6.7.6.3.2.3 NF_SL2_WRITE_ERROR Event

This event is generated whenever the VBUSM write status is something other than '0' (success).

The VPAC status register (VPAC_NF_LSE_STATUS_ERROR) stores the last non-zero sstatus value for user to see which type of error has occurred.

6.7.6.4 NF Submodule Details

Following the common hardware partition strategy of VPAC HWA, the Noise Filter is partitioned as shown in Figure 6-162.

Copyright © 2024 Texas Instruments Incorporated

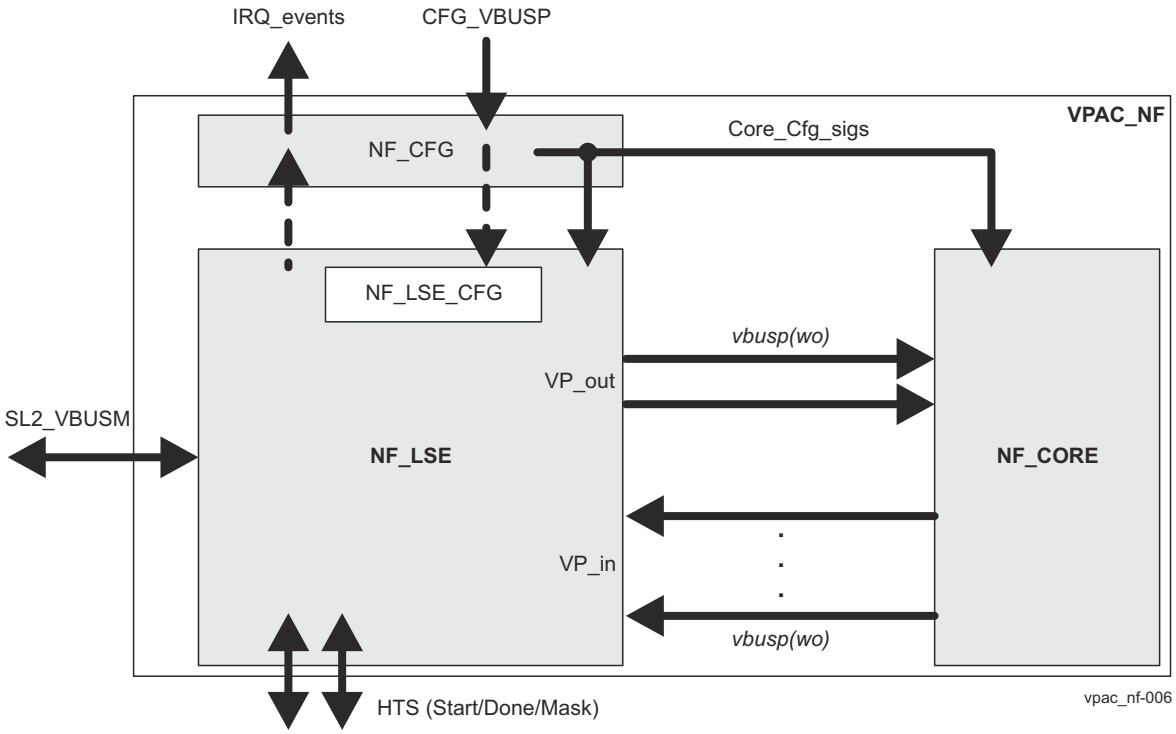


Figure 6-162. NF Hardware Partition

The NF_CFG manages all configuration registers for the NF (top and core) via a common VBUSP slave interface.

The NF_LSE (Load Store Engine) handles all interfacing to the SL2 memory space and performs the following functions:

- Provides interface to SL2 circular buffers via a common VBUSM master interface.
- Manages input and output channel updates in sync with Hardware Thread Scheduler (HTS).
- Provides data unpacking for core and packing for SL2 interface.
- Manages LSE specific configuration registers.

The NF_CORE performs noise filtering operations on given input and produces output. It has no direct connection to external interface and it works on a frame data coming in and out in raster scan order.

6.7.6.4.1 NF_CFG

The NF configuration registers consist of LSE configuration registers (see *VPAC_NF_LSE Registers*) and CORE configuration registers (see *VPAC_NF_CORE Registers*).

Figure 6-163 shows NF_CFG block diagram.

Copyright © 2024 Texas Instruments Incorporated

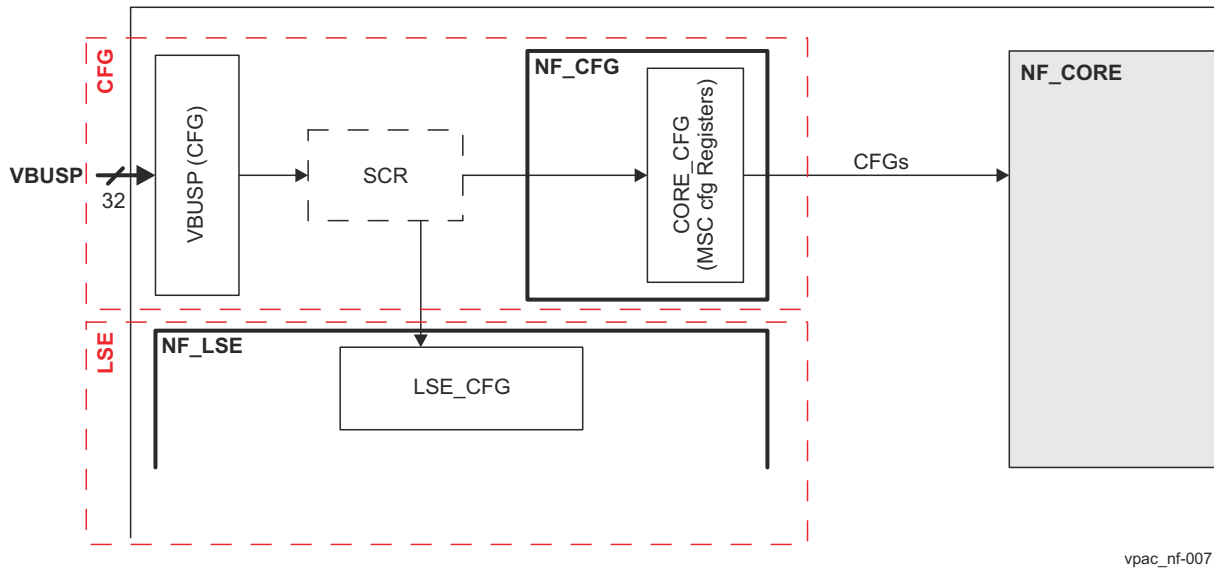


Figure 6-163. NF_CFG Block Diagram

6.7.6.4.1.1 VBUSP Configuration Interface

The 32-bit VBUSP slave interface for MMR configuration supports only the linear incrementing mode (in little endian). It does not provide any write/read error status.

6.7.6.4.1.2 Configuration Register Address Map

For details of MMR registers, see *NF Registers*.

6.7.6.4.2 NF_LSE

The NF integrates a VPAC_LSE (Load Store Engine) .

6.7.6.4.2.1 NF_LSE Overview

The NF_LSE supports the following VPAC_LSE features:

- Input channel features:
  - One SL2 input channel with following features for the input channel:
    - Up to 5 lines - input kernel height support
    - Any source pixel start position support with Line Start Offset (in nibble-address resolution)
    - Input Line Skip support
    - Vertical boundary Edge Padding (Replication only) support
- Output Channel Features:
  - Only one SL2 output channels (pixel data output only)
  - Single data plane support for each output channel with no special chroma data interleaving support on the output
- Hardware Thread Scheduler (HTS) synchronization support
  - Start/Done task synchronization signal handling
  - Programmable Done-mask generation schemes
  - End of processing (EOP) generation

6.7.6.4.2.2 NF_LSE Feature Detailed Description

For detailed LSE feature description, refer to *Load Store Engine (LSE) Overview*.

6.7.6.4.3 HTS Interface And Integration

Figure 6-164 shows interfacing with Hardware Thread Scheduler (HTS).

Copyright © 2024 Texas Instruments Incorporated

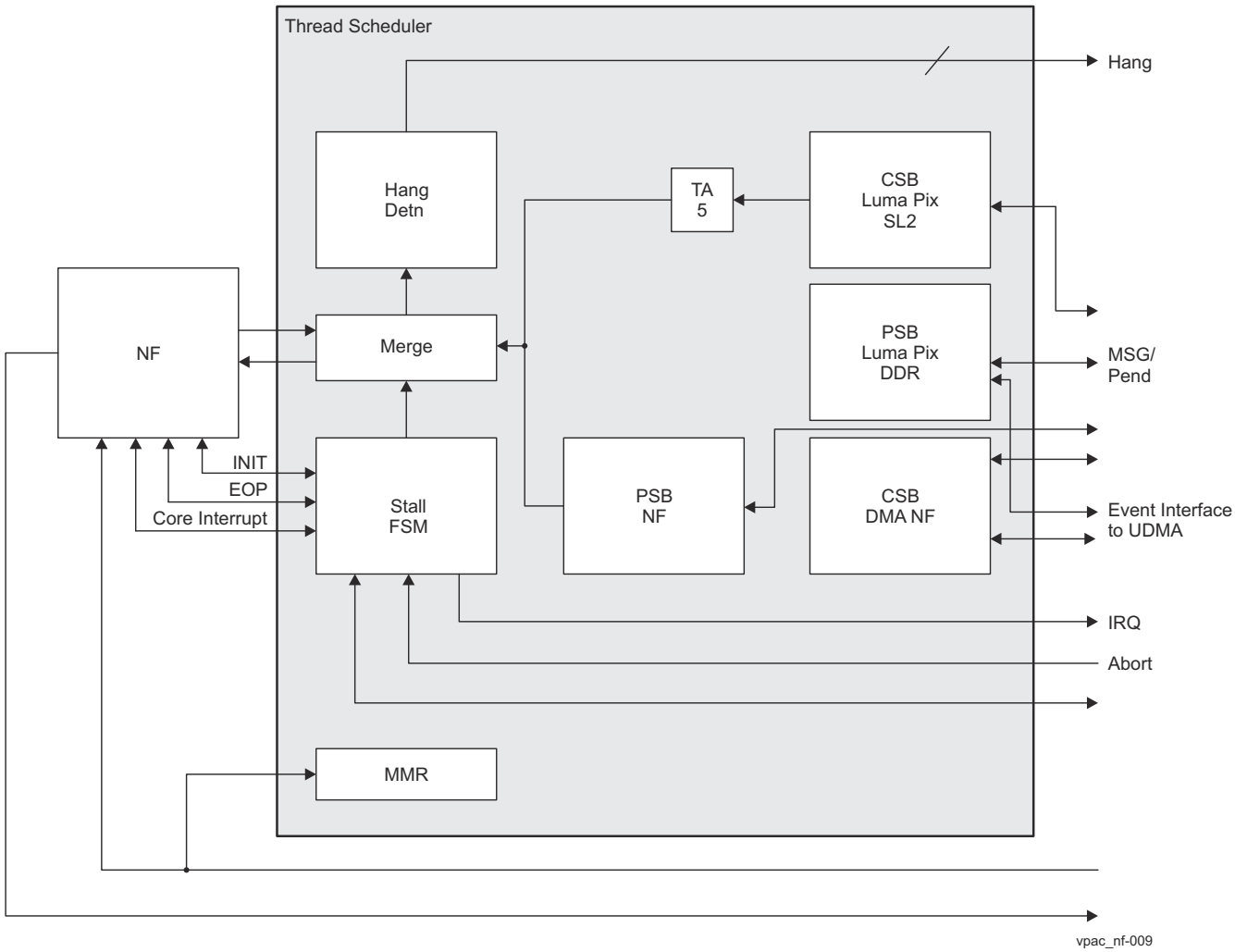


Figure 6-164. Hardware Thread Scheduler (HTS)

6.7.6.4.3.1 Hardware Thread Scheduler (HTS)

The HTS (Hardware Thread Scheduler) supports a 64-bit VBUSM master interface to communicate with a top-level message manager. It performs following functions to flow control data/control for HWA.

- NF HWA Initialization and EOP-Flush
- Handling of IRQ hardware events (that affect flow control)
- Debug/Emulation support (for stalling HTS/DMA transfer)
- Abort/Suspend support

6.7.6.4.3.2 Synchronization With HTS

Scheduling of data transfer in the SL2 interface is controlled by the HTS controller integrated at the VPAC top level. A dedicated HTS is integrated per processing thread to manage start and done flow control signals and to issue DMA service events to trigger DDR from/to SL2 transfers and SL2 from/to HWA.

The following flow control scheme is used:

/* initialization */

1. @ Frame Start up: (HTS) INIT —> Frame Initialization

/* First two line processing */

Copyright © 2024 Texas Instruments Incorporated

Note

'N' is input kernel height.

1. @ N-2 input lines available and output buffer line available: (HTS) START —> Starts filter Line processing (initial line) tasks
2. @ Input Line Consumed and all output Done: (PixCntlFSM) DONE —> Indicates completion of the line processing task
3. @ N-1 input lines available and output buffer line available: (HTS) START —> Starts filter Line processing (initial line) tasks
4. @ Input Line Consumed and output Done: (PixCntlFSM) DONE —> Indicates completion of the line processing task

/* Middle Lines processing */

1. @ N-1 input lines available and output buffer line available: (HTS) START —> Starts filter Line processing (initial line) tasks
2. @ Input Line Consumed and output Done: (PixCntlFSM) DONE —> Indicates completion of the line processing task
3. Repeat (6) and (7) until Frame_H-3 lines

/* Last two line processing */

1. @ N-1 input lines available and output buffer line available: (HTS) START —> Starts filter Line processing (initial line) tasks
2. @ Input Line Consumed and output Done: (PixCntlFSM) DONE —> Indicates completion of the line processing task
3. @ N-2 input lines available and output buffer line available: (HTS) START —> Starts filter Line processing (initial line) tasks
4. @ Input Line Consumed and all output Done: (PixCntlFSM) DONE —> Indicates completion of the line processing task
5. @EOP: (PixCntlFSM) EOP

6.7.6.4.4 Noise Filter Core Block Diagram

Figure 6-165 shows the core of the Noise filter. The line buffers are stored in SL2.

Copyright © 2024 Texas Instruments Incorporated

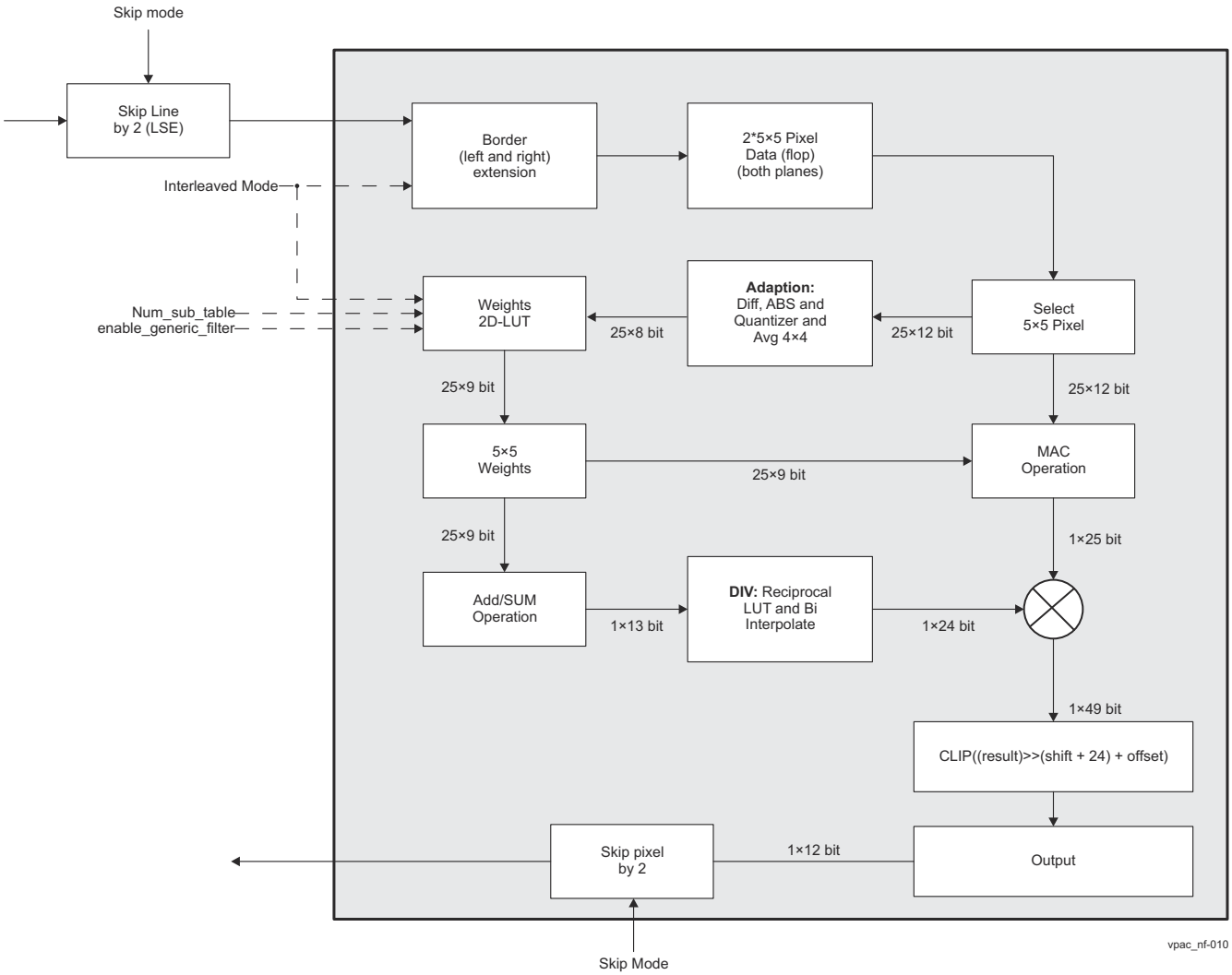


Figure 6-165. NF-CORE Block Diagram

There are two modes of filtering:

1. Bilateral filtering mode: The overall weights are computed based on center pixels. These are 8-bit unsigned weights. The center weight is read from MMR as 8-bit unsigned. The 9th bit for all weight is hardwired to '0'. The MAC operation is signed multiply with addition. As MSB for weight is set '0', it will be act as unsigned multiplier of 12-bit pixel to 8-bit unsigned weight.
2. Generic filtering mode: In this mode, all weights are 9-bit signed value read LUT from initial 24 values read as 9-bit signed value from every 2 bytes (16 bits) locations.

6.7.6.4.4.1 VP Port (NF_LSE To/From NF_CORE Over VBUSP Interface)

The data transfers from/to the MSC_LSE are done through the VP ports with VBUSP write-only streaming protocols.

6.7.6.4.4.2 Space Weight Details

Figure 6-166 shows spatial distance from central pixel to decide spatial weight.

Copyright © 2024 Texas Instruments Incorporated

4	3	2	3	4
3	1	0	1	3
2	0		0	2
3	1	0	1	3
4	3	2	3	4

vpac_nf-011

Figure 6-166. Space Distace For 5×5 Pixels

- Total number pixels = 25
  1. Number of pixel at distance 0 = 4
  2. Number of pixel at distance 1 = 4
  3. Number of pixel at distance 2 = 4
  4. Number of pixel at distance 3 = 8
  5. Number of pixel at distance 4 = 4
- Space range (five, 0-4)
- Please note that Center pixel has 9-bit programmable MMR register for fixed weight

6.7.6.4.4.3 Weight Calculation Logic

6.7.6.4.4.3.1 Combined LUT For Space And Range Weights

In this approach, Bilateral weight is computed for each center pixel based on the input (guide) image:

- i: center pixel index, j: neighbor pixel index
- I_i: center pixel input image intensity, I_j: neighbor pixel input image intensity

$$W_{i,j}^{bf} = \frac{1}{K_i}\exp\left(-\frac{|i-j|^2}{\sigma_s^2}\right)\exp\left(-\frac{|I_i-I_j|^2}{\sigma_c^2}\right),$$

Division LUT Combined LUT

vpac_nf-012

Figure 6-167. LUT For Combined Weight For Space And Range

The following are details of this approach:

- (i-j) is 3-bit for 5×5: use 3 bits
- (I_i - I_j) is 13-bits: quantize to 8 bits
- Pre-compute Wi, j as 8-bit values including normalization and put in a 2D lookup table
- LUT[i][j]: i is 3-bit pix index, j is 8-bit image intensity diff
- Lookup value is 8-bit
- Total storage: 5 × 256 × 8b = 1280 Bytes
- Can be reduced further based on symmetric nature of range
- For 1/Ki (refer to Section 6.7.6.4.4.4)

The LUT organization is shown in Figure 6-168.

Copyright © 2024 Texas Instruments Incorporated

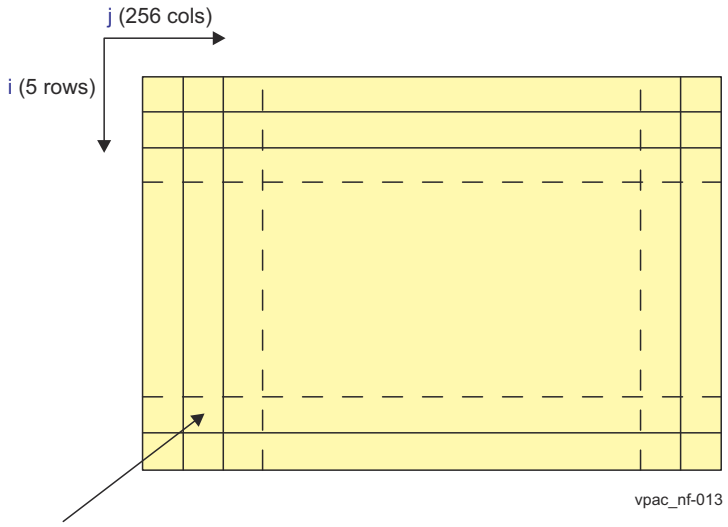


Figure 6-168. LUT Organization For Combined Weights

Figure 6-169 shows details of weight LUT logic for this approach.

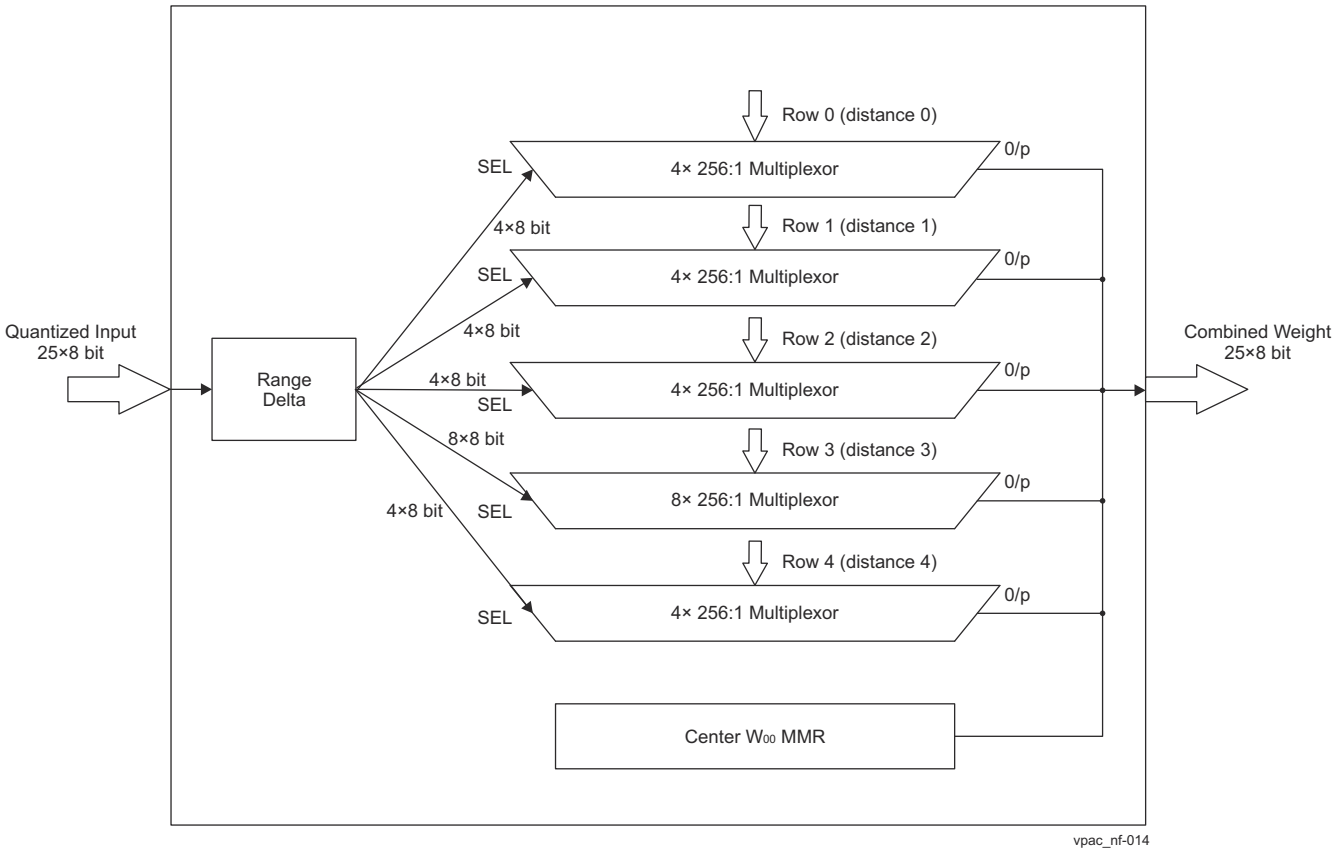


Figure 6-169. Weight LUT Logic

6.7.6.4.4.4 Reciprocal Calculation Logic

1/Ki calculation is done using Reciprocal table.

The following are details of Reciprocal table:

- Table is located in ROM (all values are hardwired)
- 24 bits per entry

Copyright © 2024 Texas Instruments Incorporated

- The first 512 entries are not quantized
- The remaining entries are quantized by 16 with Bilinear interpolation during lookup two neighbouring values
- Total of 993 entries

6.7.6.4.4.5 Border Handling

6.7.6.4.4.5.1 Border Handling (Simple)

The noise filter will support boarder extension. It will process two pixel copy in each directions for padding. The output frame size will be equal to input frame size as shown in Figure 6-170. The output there will be border pixels (two in each direction) will be unfiltered, while rest of middle frame is noise filtered. Please note that the output frame size (or resolution) will be identical to input frame (resolution).

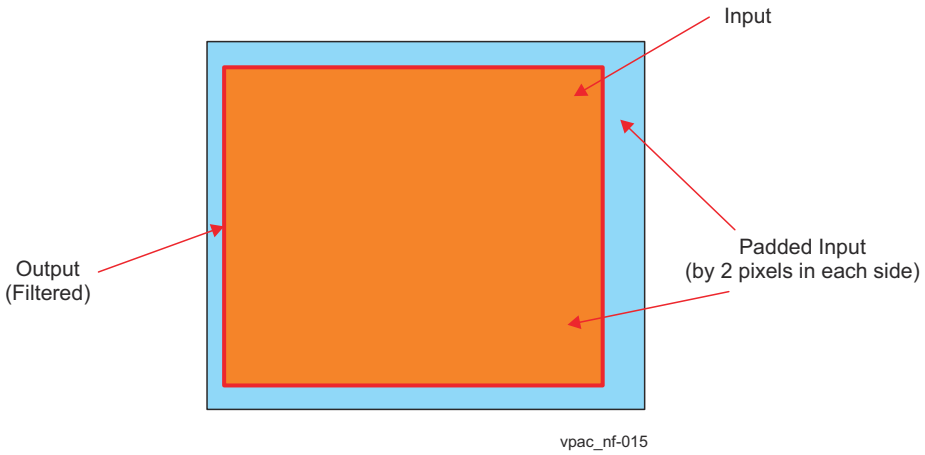


Figure 6-170. Border Handling

6.7.6.4.5 Usage As Generic 2D Filter Engine

The given noise filter can be used as generic 2D 5×5 filter core. This mode is enabled by selecting ENABLE_GENERIC_FILTER in CNTR MMR. In this case, software needs to set necessary MMR to provide co-efficients for 2D convulation. The overall decision for 5×5 co-efficient is shown in Figure 6-171.

Note the order of the weight is Column first then Row second C0_R0, C1_R0, ... , C4_R5, C5_R5.

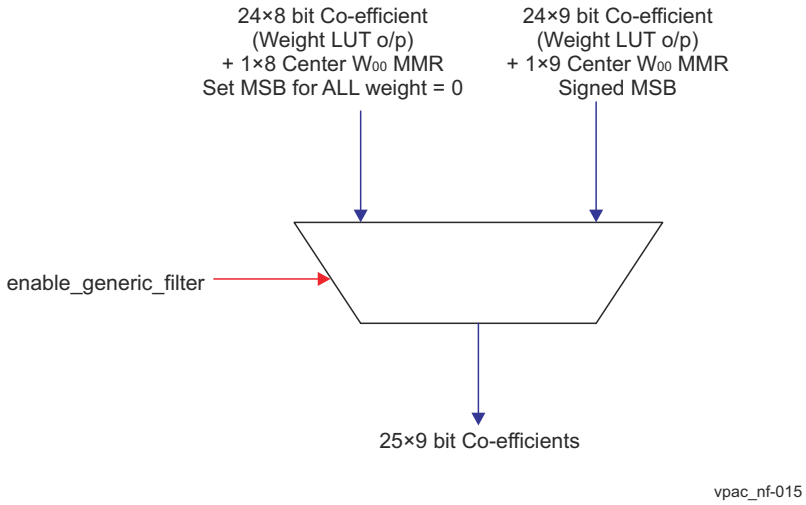


Figure 6-171. 5×5 Weight (Generic 2D Filtering)

Figure 6-172 shows the usage of Weight or Range LUT for Generic Filtering Mode.

Copyright © 2024 Texas Instruments Incorporated

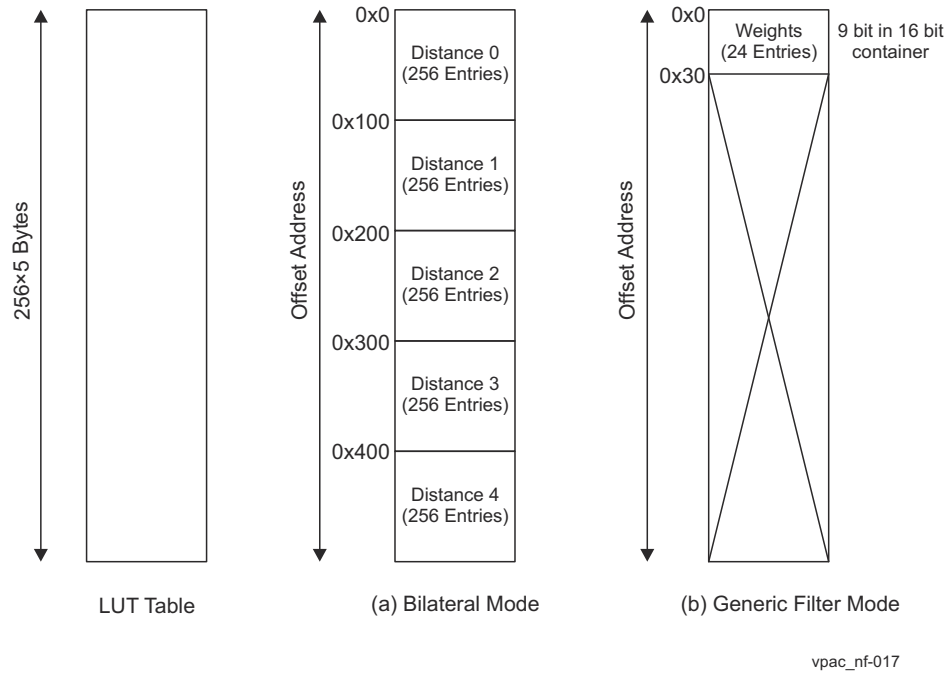


Figure 6-172. Weight LUT Usage In Generic Mode

6.7.6.4.6 Adaptive Bilateral Weight Support

The hardware supports adaptive Bilateral filtering using num_sub_tables > 0. The num_sub_table = 1 means 2 tables withing LUT per distance, num_sub_table = 2 means 4 sub_table within LUT per distance and num_sub_table = 3 means 8 sub_tables per distance. The tables are stored as interlaved within each distance. For example in case num_sum_table = 1, there are two LUT within each distance with even entries coming from LUT1 and odd entries coming from table 2 as shown in Figure 6-173.

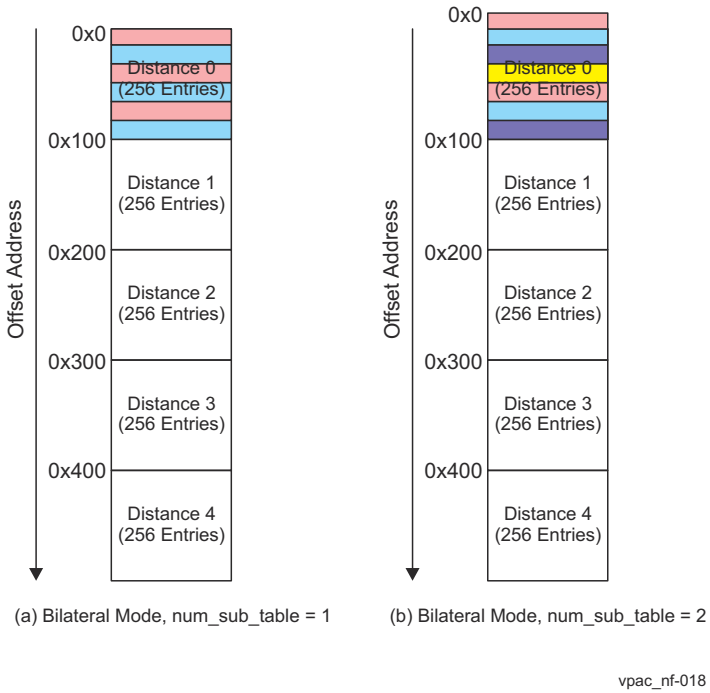


Figure 6-173. Interleaved Entries In Table

The adaptation logic consist using 4×4 average of 5×5 window (top left corner) and num_sub_tables from MMR as shown below using combo logic below:

Copyright © 2024 Texas Instruments Incorporated

- Mask "num_sub_tables" LSB bits out of 8 bits.
- LSB Bits = Average_4×4 >> (12-num_sub_tables)
- 8 bits for LUT look up = (8-num_sub_table) << num_sub_table + LSB Bits
- Please note that this is constant LSB bits for all 25 pixels that will be replaced for all 25×8 pixels.

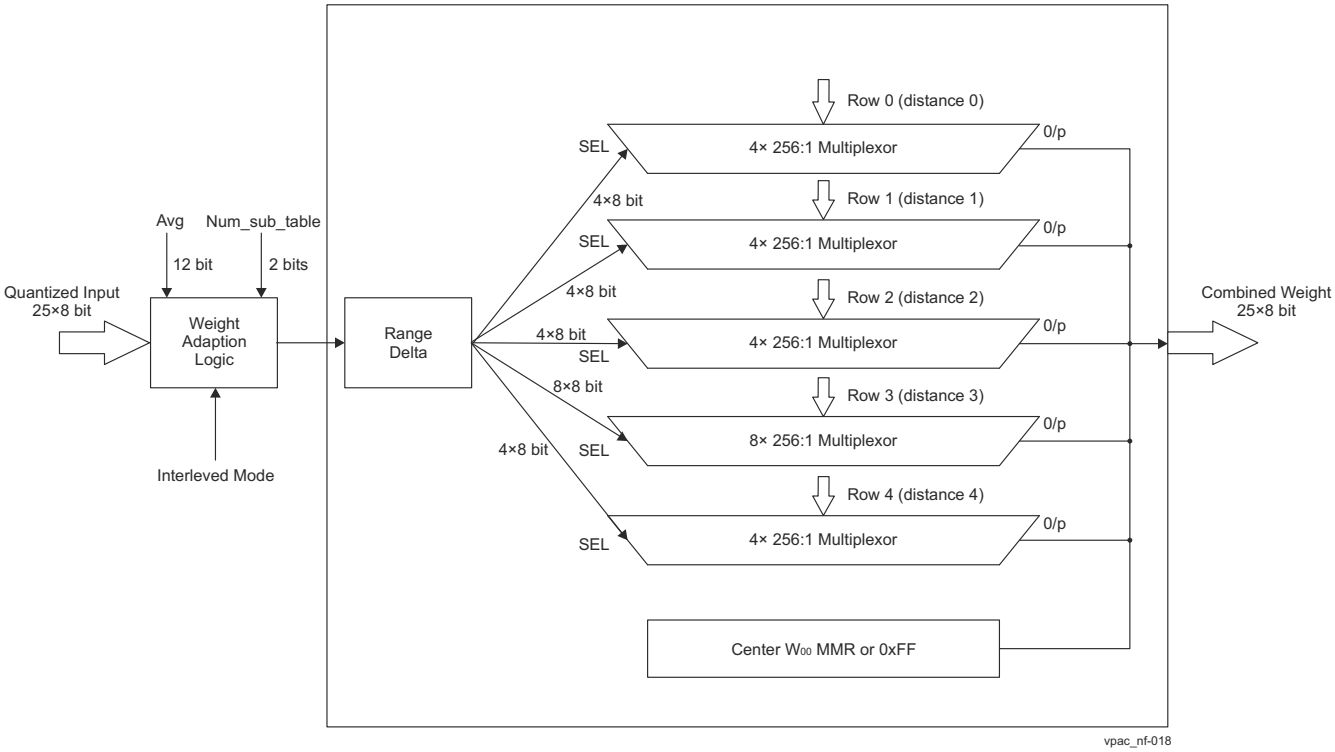


Figure 6-174. Adaptive Bilateral Filtering

6.7.6.4.7 Chroma Handling (Interleaved Mode)

Please note that the given hardware module can support either Luma (Non-Interleaved mode) OR Chroma mode (Interleaved mode) at a time. It can't support two data plane (Luma + Chroma) simultaneously.

When the input is in the interleaved data format (for example: YUV420 Chroma data), the filter core keeps two sets of 5-input samples (Cb and Cr buffers) and alternates filtering operation between two. In this case, core does deinterleaving and maintains two set of data of 6×5 for both planes as shown in block diagram of core. The parameters for Chroma needs to programmed at every frame differently compared to Luma.

Note software can always do a "force" select which sub table to use by clearing adaptive_mode then sub_table_select defines which table to use, otherwise below defines which sub table is selected.

- Mask "num_sub_tables" LSB bits out of 8 bits.
- LSB Bits = Average_4×4>>(12-num_sub_tables)
- If(num_sub_tables != 1):
  - 8 bits for LUT look up = (8-num_sub_table) << num_sub_table + LSB Bits

    Else:
  - 8 bits for LUT look up = (8- inputPlaneInterleaving) << inputPlaneInterleaving + LSB Bits
- Please note that this is constant LSB bits for all 25 pixels that will be replaced for all 25×8 pixels.

6.7.6.5 NF Integration Details

6.7.6.5.1 Performance Requirements

The core performance throughput 1 cycle/pixel. The NF module is designed to support 300 MHz.

Copyright © 2024 Texas Instruments Incorporated

6.7.6.5.2 Slave VBUSP Interface Clock

Table 6-176. Slave VBUSP Interface Clock

Signal Name	Dir	Default Val	Description
NF_CLK	In	1'd0	Single ended clock

6.7.6.5.3 Clocking

The NF module has one input clock which should run at 300 MHz.

It only has 1 clock IPGeneric

kp_ti_ipg_clk_s1c1fps I_kp_ti_ipg_clk_s1c1fps

6.7.6.6 NF Programmer's Guide

6.7.6.6.1 Programming Model

The full programming of the module, consist of the following four steps (see Figure 6-175):

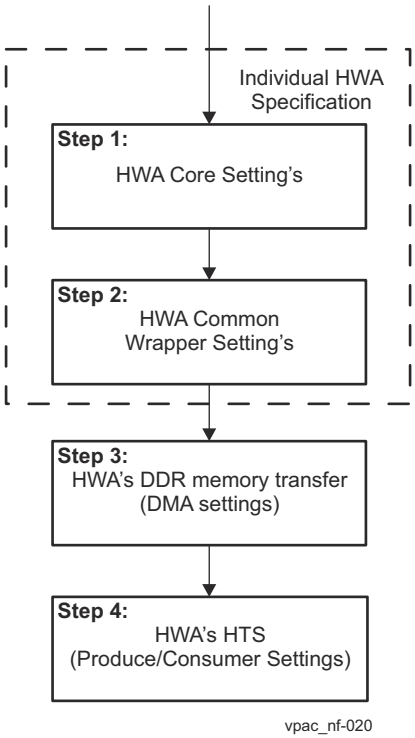


Figure 6-175. Programming Steps

6.7.6.6.1.1 HWA Core Programming Details

There are two use-cases:

- Bilateral Noise Filter Mode: The following core MMR needs to be programmed to enable Bilateral noise filter mode:
  - Set Weight LUT (8-bit unsigned) tables (5×256 values corresponding to various distance from distance in MMR (0x100 onwards))
  - Set Control register fields (Center weight to maximum - 0×255, offset, shift, …)
  - Set generic_filter_mode to '0'
- Generic 2D Convolution (filtering) Mode: The following core MMR needs to be programmed to enable Bilateral noise filter mode:
  - Set Weight LUT (9 bit unsigned) tables (24 values (each 9-bit signed) in 16 bit container from start of LUT
  - Set Control register fields (Center weight, offset, shift, …)

Copyright © 2024 Texas Instruments Incorporated

– Set generic_filter_mode to '1'

6.7.6.6.1.2 NF SL2 Wrapper Interface Programming Details

The following MMR needs to be programmed as per overall Data flow as well as algorithm needs:

- Set Input (SRC) side SL2 configuration, buffer base address, attributes and sizes
- Set Output (DST) side SL2 configuration, buffer base address, attributes and sizes

6.7.6.6.1.3 HWA HTS Programming Details

For more details of programming, refer to HTS and Section VPAC Subsystem.

Although following are Hardcoded configuration for given HWA (Noise filter (NF)) for HTS:

- Consumer score board: Transaction aggregation
- Number of lines of delay = 5
- …

6.7.6.6.1.4 HWA Data Transfer Programming Details

For more details, refer to the VPAC Top/UTC specification.

6.7.6.6.2 Initialization Sequence

Refer to Section VPAC Subsystem.

6.7.6.6.3 Real-Time Operating Requirements

Refer to Section VPAC Subsystem.

6.7.6.6.4 Power Up/Down Sequence

Refer to Section VPAC Subsystem.

6.7.6.6.5 Clock Stop

Note NF requires the software to shutdown all traffic and insure the NF is idle before it can assert nf_clkstop_req signal.

Also, no new work/traffic can occure until nf_clockstop_req de-asserts alogn with nf_clkstop_ack de-assertion. Since the NF will be idle before the assertion of nf_clkstop_req, the assertion of nf_clkstop_ack should occur less than 10 clock cycles in response to the requirest.

Copyright © 2024 Texas Instruments Incorporated

6.8 Depth and Motion Perception Accelerator (DMPAC)

This chapter describes the Depth and Motion Perception Accelerator (DMPAC) in the device.

6.8.1 DMPAC Overview

The Depth and Motion Perception Accelerator (DMPAC) is a power efficient hardware accelerator that computes dense stereo depth maps (*depth*) and dense optical flow vectors (*motion*) from camera inputs.

The image/video sensor-based environmental perception (also known as scene understanding) is at the core of many emerging applications in automotive, industrial and consumer electronics. Typically, this involves detection of all objects in the scene along with their 3D position and motion with regards to the observer or the car by analyzing one or many related input video streams. Various computer vision algorithms are used to achieve these tasks.

A very robust method of obtaining the 3D depth from images is to use two cameras in a stereo setup - two cameras with known relative positions and camera parameters. The two images of the same scene, captured from two different camera poses/perspectives, are analyzed to find disparities among every pixel positions in the images. This is known as the Stereo Disparity map. The disparity values of every pixel can be used to obtain the 3D positions of the object/space they belong to via triangulation.

On the other hand, by analyzing two images from a single camera, captured at two different time instances (that is, two temporal frames in a video), one can determine where each pixel in a past frame moved to in the future frame. This is known as the Optical Flow vector. The flow vectors for each pixel position can be used to obtain 3D structure of the scene, identify moving objects and determine their relative speed and direction of motion.

The DMPAC is dedicated to the aforesaid image processing tasks. The stereo and optical flow processing is partitioned into two top level sub-blocks: the Dense Optical Flow (DOF) engine and the Stereo Disparity Engine (SDE). The DOF and SDE blocks share a common shared local memory, DMA, external messaging and control infrastructure.

6.8.1.1 DMPAC Features

The DMPAC supports the following main features:

- Input image resolution up to 2 MPix with maximum horizontal resolution of up to 2048 pixels and maximum vertical resolution of up to 1024 pixels
- Input pixel format of 12-bit fully packed luminance data, and other formats using the integrated Format Conversion (FOCO) module
- Pixel level output packed in 16bpp and 32bpp formats for stereo disparity estimates and optical flow estimates, respectively
- Simultaneous operation of SDE and DOF, each processing inputs of up to 1 MPix image resolution (1 MPix = 1280 × 720 pixels)
- Resolution vs. frame rate scalability (higher frame rate at lower resolution)
- A Dense Optical Flow (DOF) engine: The DOF engine implements Texas Instruments' proprietary algorithm for estimation/calculation of the optical flow between the input image pair. The algorithm uses hierarchical coarse-to-fine block search strategy leveraging image pyramids in which proprietary binary pixel descriptors are used for pixel correspondence determination. In the scheme accuracy and smoothness of the flow map is enforced using optical flow estimates of the causal neighbors of a pixel in the given and higher (lower resolution) pyramid level (pixels are processed in raster scan order in a pyramid level to refine existing optical flow estimates). The DOF engine supports:
  - Optical flow vector lengths up to +/- 191 pixels in horizontal direction and +/- 62 pixels in vertical direction for each input pixel
  - Dense flow vector map for each input pixel
  - Optical flow vector precision of $1/16^{th}$ of inter pixel distance
  - Post filtering of the optical flow estimates using 2D median filtering
  - 16 level confidence score for every output flow vector
- A Stereo Disparity Engine (SDE): The SDE implements Texas Instruments' proprietary algorithm in order to find the disparity map between the input image pair. The SDE supports:
  - Disparity search range (SR) of 64/128/192. It can support either "0 to SR-1" or "-3 to SR-4"
  - $1/16^{th}$ pixel accurate sub-pixel level disparity estimate

Copyright © 2024 Texas Instruments Incorporated

  - Disparity estimate post processing using 2D median filtering
  - 8 level confidence score for each disparity output
- Common top level infrastructure:
  - Shared Level 2 (SL2) memory sub-system, which serves both DOF and SDE blocks, along with the FOCO modules when required as an intermediate storage exchange data across sub-modules (FOCO to DOF/SDE) and from DDR/System Memory
  - A Unified Transfer Controller (UTC), which serves as a DMA engine
  - Messaging and control mechanism via a Hardware Thread Scheduler (HTS) block
  - A Counter, Timer and System Event Trace (CTSET) module, which provides event tracing capability for the SDE and DOF hardware threads

The DMPAC does not support (but relies on other HWAs/processors on the SoC to perform these tasks):

- Epipolar rectification of the stereo input images
- Image pyramid generation for the optical flow input images

7 Interprocessor Communication

This chapter describes the interprocessor communication modules in the device.

7.1 Mailbox..... 679
7.2 Spinlock..... 691

Copyright © 2024 Texas Instruments Incorporated

7.1 Mailbox

This chapter describes the mailbox module in the device.

7.1.1 Mailbox Overview

Mailbox module serves to facilitate the communication between the various on-chip processors of the device by providing a queued mailbox-interrupt mechanism.

The queued mailbox-interrupt mechanism allows the software to establish a communication channel between two processors (users) through a set of registers and associated interrupt signals.

Table 7-1. Mailbox Allocation Across Device Domains

Instance	Domain		
	WKUP	MCU	MAIN
MAILBOX0	-	-	✓ (NAVSS)
MAILBOX1	-	-	✓ (NAVSS)

7.1.1.1 Mailbox Features

Each mailbox module supports the following features:

- Parameters configured at design time (see Table 7-2):
  - Number of mailbox clusters
    - Each mailbox cluster has the same configuration
    - Clusters are accessed via separate VBUS regions. Clusters can be treated as mailbox module instances.
  - Number of users
  - Number of mailbox message queues
  - Number of messages (FIFO depth) for each message queue
- 32-bit message width
- Partial writes do not advance the FIFO pointers
- Message reception and queue-not-full notification using interrupts
- Non-intrusive emulation

7.1.1.2 Mailbox Parameters

Table 7-2 shows the configuration of the mailbox module in the device.

Table 7-2. Mailbox Configuration Parameters

Module Instance	Parameters			
	Users (Processors)	Clusters (1)	Message Queues (2)	Number of Messages (3)
MAILBOX0	4	12	16	4
MAILBOX1	4	12	16	4

(1) Number of mailbox clusters
(2) Number of mailbox message queues (mailboxes per cluster)
(3) Number of messages (FIFO depth) for each message queue

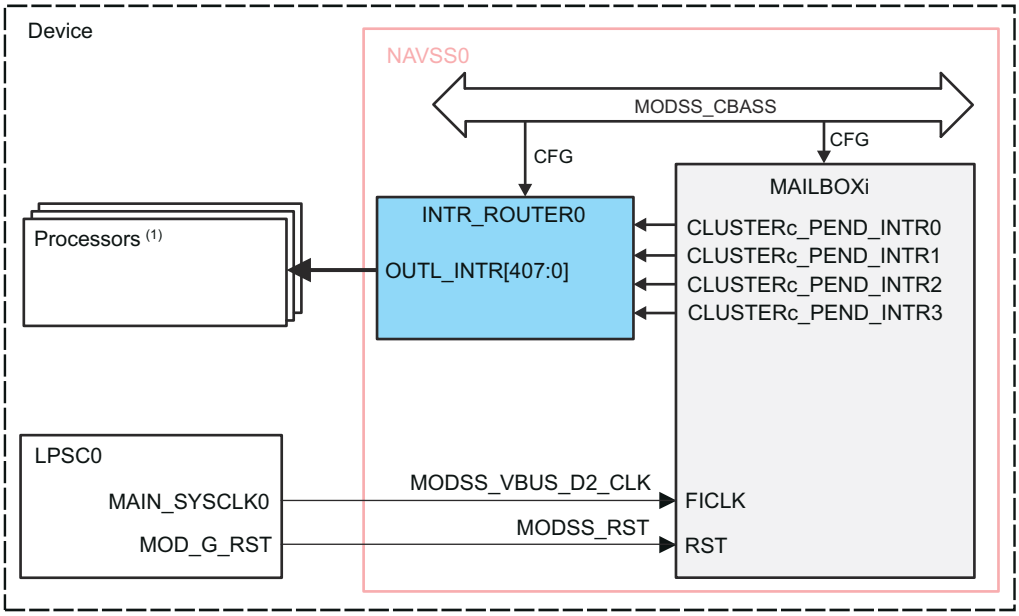
Copyright © 2024 Texas Instruments Incorporated

7.1.2 Mailbox Integration

This section describes the mailbox integration in the device, including information about clocks, resets, and hardware requests.

7.1.2.1 System Mailbox Integration

Figure 7-1 shows the MAILBOX integration in the NAVSS. Each MAILBOX module contains a number of mailbox clusters. Each cluster generates one interrupt per user (processor).



(1) For details, refer to NAVSS0 Clocks, Resets, and Interrupts figure.

mailbox-002

Figure 7-1. Mailbox Integration

c = 0 to 11

i = 0 to 1

Table 7-3 through Table 7-5 summarize the MAILBOX integration in the device.

Table 7-3. Mailbox Integration Attributes

Module Instance	Attributes			
	Power Sleep Controller	Power Domain	Module Domain	Interconnect
MAILBOX0	PSC0	GP	LPSC0	MODSS_CBASS
MAILBOX1	PSC0	GP	LPSC0	MODSS_CBASS

Table 7-4. Mailbox Clocks and Resets

Clocks				
Module Instance	**Module Clock Input**	**Source Clock Signal**	**Source**	**Description**
MAILBOX0	MAILBOX0_FICLK	MODSS_VBUS_D2_CLK	MAIN_SYSCLK0	MAILBOX0 clock. This clock is used for all interface and functional operations.
MAILBOX1	MAILBOX1_FICLK	MODSS_VBUS_D2_CLK	MAIN_SYSCLK0	MAILBOX1 clock. This clock is used for all interface and functional operations.
Resets				
Module Instance	**Module Reset Input**	**Source Reset Signal**	**Source**	**Description**
MAILBOX0	MAILBOX0_RST	MODSS_RST	LPSC0	MAILBOX0 hardware reset
MAILBOX1	MAILBOX1_RST	MODSS_RST	LPSC0	MAILBOX1 hardware reset

Copyright © 2024 Texas Instruments Incorporated

Table 7-5. Mailbox Hardware Requests

Interrupt Requests					
Module Instance	**Module Interrupt Signal**	**Destination Interrupt Input**	**Destination**	**Description**	**Type**
MAILBOX0	CLUSTER0_PEND_INTR3	IN_INTR[439]	INTR_ROUTER0	Cluster 0 user 3 interrupt request.	Level
	CLUSTER0_PEND_INTR2	IN_INTR[438]	INTR_ROUTER0	Cluster 0 user 2 interrupt request.	Level
	CLUSTER0_PEND_INTR1	IN_INTR[437]	INTR_ROUTER0	Cluster 0 user 1 interrupt request.	Level
	CLUSTER0_PEND_INTR0	IN_INTR[436]	INTR_ROUTER0	Cluster 0 user 0 interrupt request.	Level
	CLUSTER1_PEND_INTR3	IN_INTR[435]	INTR_ROUTER0	Cluster 1 user 3 interrupt request.	Level
	CLUSTER1_PEND_INTR2	IN_INTR[434]	INTR_ROUTER0	Cluster 1 user 2 interrupt request.	Level
	CLUSTER1_PEND_INTR1	IN_INTR[433]	INTR_ROUTER0	Cluster 1 user 1 interrupt request.	Level
	CLUSTER1_PEND_INTR0	IN_INTR[432]	INTR_ROUTER0	Cluster 1 user 0 interrupt request.	Level
	CLUSTER2_PEND_INTR3	IN_INTR[431]	INTR_ROUTER0	Cluster 2 user 3 interrupt request.	Level
	CLUSTER2_PEND_INTR2	IN_INTR[430]	INTR_ROUTER0	Cluster 2 user 2 interrupt request.	Level
	CLUSTER2_PEND_INTR1	IN_INTR[429]	INTR_ROUTER0	Cluster 2 user 1 interrupt request.	Level
	CLUSTER2_PEND_INTR0	IN_INTR[428]	INTR_ROUTER0	Cluster 2 user 0 interrupt request.	Level
	CLUSTER3_PEND_INTR3	IN_INTR[427]	INTR_ROUTER0	Cluster 3 user 3 interrupt request.	Level
	CLUSTER3_PEND_INTR2	IN_INTR[426]	INTR_ROUTER0	Cluster 3 user 2 interrupt request.	Level
	CLUSTER3_PEND_INTR1	IN_INTR[425]	INTR_ROUTER0	Cluster 3 user 1 interrupt request.	Level
	CLUSTER3_PEND_INTR0	IN_INTR[424]	INTR_ROUTER0	Cluster 3 user 0 interrupt request.	Level
	CLUSTER4_PEND_INTR3	IN_INTR[423]	INTR_ROUTER0	Cluster 4 user 3 interrupt request.	Level
	CLUSTER4_PEND_INTR2	IN_INTR[422]	INTR_ROUTER0	Cluster 4 user 2 interrupt request.	Level
	CLUSTER4_PEND_INTR1	IN_INTR[421]	INTR_ROUTER0	Cluster 4 user 1 interrupt request.	Level
	CLUSTER4_PEND_INTR0	IN_INTR[420]	INTR_ROUTER0	Cluster 4 user 0 interrupt request.	Level
	CLUSTER5_PEND_INTR3	IN_INTR[419]	INTR_ROUTER0	Cluster 5 user 3 interrupt request.	Level
	CLUSTER5_PEND_INTR2	IN_INTR[418]	INTR_ROUTER0	Cluster 5 user 2 interrupt request.	Level
	CLUSTER5_PEND_INTR1	IN_INTR[417]	INTR_ROUTER0	Cluster 5 user 1 interrupt request.	Level
	CLUSTER5_PEND_INTR0	IN_INTR[416]	INTR_ROUTER0	Cluster 5 user 3 interrupt request.	Level
	CLUSTER6_PEND_INTR3	IN_INTR[415]	INTR_ROUTER0	Cluster 6 user 3 interrupt request.	Level
	CLUSTER6_PEND_INTR2	IN_INTR[414]	INTR_ROUTER0	Cluster 6 user 2 interrupt request.	Level

Copyright © 2024 Texas Instruments Incorporated

Table 7-5. Mailbox Hardware Requests (continued)

	CLUSTER6_PEND_INTR1	IN_INTR[413]	INTR_ROUTER0	Cluster 6 user 1 interrupt request.	Level
	CLUSTER6_PEND_INTR0	IN_INTR[412]	INTR_ROUTER0	Cluster 6 user 0 interrupt request.	Level
	CLUSTER7_PEND_INTR3	IN_INTR[411]	INTR_ROUTER0	Cluster 7 user 3 interrupt request.	Level
	CLUSTER7_PEND_INTR2	IN_INTR[410]	INTR_ROUTER0	Cluster 7 user 2 interrupt request.	Level
	CLUSTER7_PEND_INTR1	IN_INTR[409]	INTR_ROUTER0	Cluster 7 user 1 interrupt request.	Level
	CLUSTER7_PEND_INTR0	IN_INTR[408]	INTR_ROUTER0	Cluster 7 user 0 interrupt request.	Level
	CLUSTER8_PEND_INTR3	IN_INTR[407]	INTR_ROUTER0	Cluster 8 user 3 interrupt request.	Level
	CLUSTER8_PEND_INTR2	IN_INTR[406]	INTR_ROUTER0	Cluster 8 user 2 interrupt request.	Level
	CLUSTER8_PEND_INTR1	IN_INTR[405]	INTR_ROUTER0	Cluster 8 user 1 interrupt request.	Level
	CLUSTER8_PEND_INTR0	IN_INTR[404]	INTR_ROUTER0	Cluster 8 user 0 interrupt request.	Level
	CLUSTER9_PEND_INTR3	IN_INTR[403]	INTR_ROUTER0	Cluster 9 user 3 interrupt request.	Level
	CLUSTER9_PEND_INTR2	IN_INTR[402]	INTR_ROUTER0	Cluster 9 user 2 interrupt request.	Level
	CLUSTER9_PEND_INTR1	IN_INTR[401]	INTR_ROUTER0	Cluster 9 user 1 interrupt request.	Level
	CLUSTER9_PEND_INTR0	IN_INTR[400]	INTR_ROUTER0	Cluster 9 user 0 interrupt request.	Level
	CLUSTER10_PEND_INTR3	IN_INTR[399]	INTR_ROUTER0	Cluster 10 user 3 interrupt request.	Level
	CLUSTER10_PEND_INTR2	IN_INTR[398]	INTR_ROUTER0	Cluster 10 user 2 interrupt request.	Level
	CLUSTER10_PEND_INTR1	IN_INTR[397]	INTR_ROUTER0	Cluster 10 user 1 interrupt request.	Level
	CLUSTER10_PEND_INTR0	IN_INTR[396]	INTR_ROUTER0	Cluster 10 user 0 interrupt request.	Level
	CLUSTER11_PEND_INTR3	IN_INTR[395]	INTR_ROUTER0	Cluster 11 user 3 interrupt request.	Level
	CLUSTER11_PEND_INTR2	IN_INTR[394]	INTR_ROUTER0	Cluster 11 user 2 interrupt request.	Level
	CLUSTER11_PEND_INTR1	IN_INTR[393]	INTR_ROUTER0	Cluster 11 user 1 interrupt request.	Level
	CLUSTER11_PEND_INTR0	IN_INTR[392]	INTR_ROUTER0	Cluster 11 user 0 interrupt request.	Level
MAILBOX1	CLUSTER0_PEND_INTR3	IN_INTR[487]	INTR_ROUTER0	Cluster 0 user 3 interrupt request.	Level
	CLUSTER0_PEND_INTR2	IN_INTR[486]	INTR_ROUTER0	Cluster 0 user 2 interrupt request.	Level
	CLUSTER0_PEND_INTR1	IN_INTR[485]	INTR_ROUTER0	Cluster 0 user 1 interrupt request.	Level
	CLUSTER0_PEND_INTR0	IN_INTR[484]	INTR_ROUTER0	Cluster 0 user 0 interrupt request.	Level
	CLUSTER1_PEND_INTR3	IN_INTR[483]	INTR_ROUTER0	Cluster 1 user 3 interrupt request.	Level
	CLUSTER1_PEND_INTR2	IN_INTR[482]	INTR_ROUTER0	Cluster 1 user 2 interrupt request.	Level

Copyright © 2024 Texas Instruments Incorporated

Table 7-5. Mailbox Hardware Requests (continued)

CLUSTER1_PEND_INTR1	IN_INTR[481]	INTR_ROUTER0	Cluster 1 user 1 interrupt request.	Level
CLUSTER1_PEND_INTR0	IN_INTR[480]	INTR_ROUTER0	Cluster 1 user 0 interrupt request.	Level
CLUSTER2_PEND_INTR3	IN_INTR[479]	INTR_ROUTER0	Cluster 2 user 3 interrupt request.	Level
CLUSTER2_PEND_INTR2	IN_INTR[478]	INTR_ROUTER0	Cluster 2 user 2 interrupt request.	Level
CLUSTER2_PEND_INTR1	IN_INTR[477]	INTR_ROUTER0	Cluster 2 user 1 interrupt request.	Level
CLUSTER2_PEND_INTR0	IN_INTR[476]	INTR_ROUTER0	Cluster 2 user 0 interrupt request.	Level
CLUSTER3_PEND_INTR3	IN_INTR[475]	INTR_ROUTER0	Cluster 3 user 3 interrupt request.	Level
CLUSTER3_PEND_INTR2	IN_INTR[474]	INTR_ROUTER0	Cluster 3 user 2 interrupt request.	Level
CLUSTER3_PEND_INTR1	IN_INTR[473]	INTR_ROUTER0	Cluster 3 user 1 interrupt request.	Level
CLUSTER3_PEND_INTR0	IN_INTR[472]	INTR_ROUTER0	Cluster 3 user 0 interrupt request.	Level
CLUSTER4_PEND_INTR3	IN_INTR[471]	INTR_ROUTER0	Cluster 4 user 3 interrupt request.	Level
CLUSTER4_PEND_INTR2	IN_INTR[470]	INTR_ROUTER0	Cluster 4 user 2 interrupt request.	Level
CLUSTER4_PEND_INTR1	IN_INTR[469]	INTR_ROUTER0	Cluster 4 user 1 interrupt request.	Level
CLUSTER4_PEND_INTR0	IN_INTR[468]	INTR_ROUTER0	Cluster 4 user 0 interrupt request.	Level
CLUSTER5_PEND_INTR3	IN_INTR[467]	INTR_ROUTER0	Cluster 5 user 3 interrupt request.	Level
CLUSTER5_PEND_INTR2	IN_INTR[466]	INTR_ROUTER0	Cluster 5 user 2 interrupt request.	Level
CLUSTER5_PEND_INTR1	IN_INTR[465]	INTR_ROUTER0	Cluster 5 user 1 interrupt request.	Level
CLUSTER5_PEND_INTR0	IN_INTR[464]	INTR_ROUTER0	Cluster 5 user 0 interrupt request.	Level
CLUSTER6_PEND_INTR3	IN_INTR[463]	INTR_ROUTER0	Cluster 6 user 3 interrupt request.	Level
CLUSTER6_PEND_INTR2	IN_INTR[462]	INTR_ROUTER0	Cluster 6 user 2 interrupt request.	Level
CLUSTER6_PEND_INTR1	IN_INTR[461]	INTR_ROUTER0	Cluster 6 user 1 interrupt request.	Level
CLUSTER6_PEND_INTR0	IN_INTR[460]	INTR_ROUTER0	Cluster 6 user 0 interrupt request.	Level
CLUSTER7_PEND_INTR3	IN_INTR[459]	INTR_ROUTER0	Cluster 7 user 3 interrupt request.	Level
CLUSTER7_PEND_INTR2	IN_INTR[458]	INTR_ROUTER0	Cluster 7 user 2 interrupt request.	Level
CLUSTER7_PEND_INTR1	IN_INTR[457]	INTR_ROUTER0	Cluster 7 user 1 interrupt request.	Level
CLUSTER7_PEND_INTR0	IN_INTR[456]	INTR_ROUTER0	Cluster 7 user 0 interrupt request.	Level
CLUSTER8_PEND_INTR3	IN_INTR[455]	INTR_ROUTER0	Cluster 8 user 3 interrupt request.	Level
CLUSTER8_PEND_INTR2	IN_INTR[454]	INTR_ROUTER0	Cluster 8 user 2 interrupt request.	Level

Copyright © 2024 Texas Instruments Incorporated

Table 7-5. Mailbox Hardware Requests (continued)

CLUSTER8_PEND_INTR1	IN_INTR[453]	INTR_ROUTER0	Cluster 8 user 1 interrupt request.	Level
CLUSTER8_PEND_INTR0	IN_INTR[452]	INTR_ROUTER0	Cluster 8 user 0 interrupt request.	Level
CLUSTER9_PEND_INTR3	IN_INTR[451]	INTR_ROUTER0	Cluster 9 user 3 interrupt request.	Level
CLUSTER9_PEND_INTR2	IN_INTR[450]	INTR_ROUTER0	Cluster 9 user 2 interrupt request.	Level
CLUSTER9_PEND_INTR1	IN_INTR[449]	INTR_ROUTER0	Cluster 9 user 1 interrupt request.	Level
CLUSTER9_PEND_INTR0	IN_INTR[448]	INTR_ROUTER0	Cluster 9 user 0 interrupt request.	Level
CLUSTER10_PEND_INTR3	IN_INTR[447]	INTR_ROUTER0	Cluster 10 user 3 interrupt request.	Level
CLUSTER10_PEND_INTR2	IN_INTR[446]	INTR_ROUTER0	Cluster 10 user 2 interrupt request.	Level
CLUSTER10_PEND_INTR1	IN_INTR[445]	INTR_ROUTER0	Cluster 10 user 1 interrupt request.	Level
CLUSTER10_PEND_INTR0	IN_INTR[444]	INTR_ROUTER0	Cluster 10 user 0 interrupt request.	Level
CLUSTER11_PEND_INTR3	IN_INTR[443]	INTR_ROUTER0	Cluster 11 user 3 interrupt request.	Level
CLUSTER11_PEND_INTR2	IN_INTR[442]	INTR_ROUTER0	Cluster 11 user 2 interrupt request.	Level
CLUSTER11_PEND_INTR1	IN_INTR[441]	INTR_ROUTER0	Cluster 11 user 1 interrupt request.	Level
CLUSTER11_PEND_INTR0	IN_INTR[440]	INTR_ROUTER0	Cluster 11 user 0 interrupt request.	Level

DMA Events					
Module Instance	**Module DMA Event**	**Destination DMA Event Input**	**Destination**	**Description**	**Type**
MAILBOX0	-	-	-	No PDMA channels to external DMA engines	-
MAILBOX1	-	-	-	No PDMA channels to external DMA engines	-

Note

For more information on the interconnects, see *System Interconnect*.

For more information on the power, reset, and clock management, see the corresponding sections within *Device Configuration*.

Note

For information about interrupt source description, see *Mailbox Interrupt Requests*.

Copyright © 2024 Texas Instruments Incorporated

7.1.3 Mailbox Functional Description

The mailbox module provides a means of communication through message queues among the users. The individual mailbox modules, or FIFOs, can associate (or de-associate) with any of the processors using the MAILBOX_IRQ_ENABLE_SET_j (or MAILBOX_IRQ_ENABLE_CLR_j) register.

Each user has a dedicated interrupt signal from the corresponding mailbox module instance and dedicated interrupt enabling and status registers.

Each MAILBOX_IRQ_STATUS_RAW_j/MAILBOX_IRQ_STATUS_CLR_j interrupt status register corresponds to a particular user.

7.1.3.1 Mailbox Block Diagram

Figure 7-2 shows the mailbox internal block diagram.

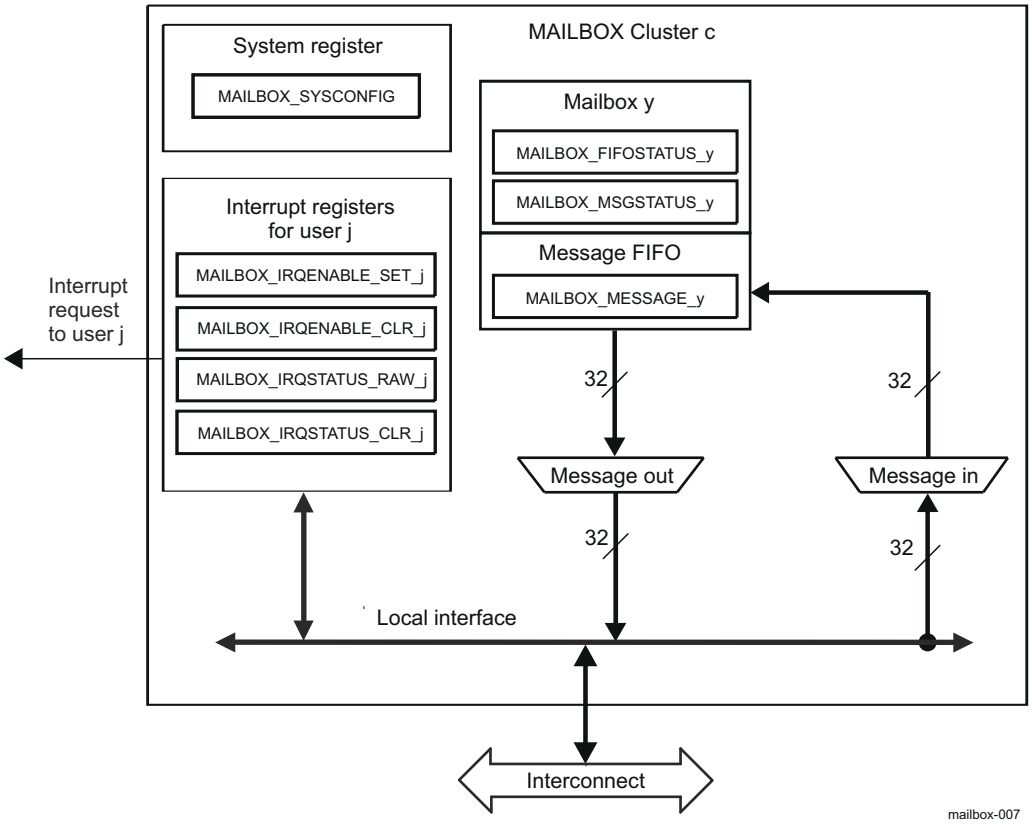


Figure 7-2. Mailbox Block Diagram

j = 0 to 3

y = 0 to 15

7.1.3.2 Mailbox Software Reset

The mailbox module supports a software reset through the MAILBOX_SYSCONFIG[0] SOFTRESET bit. Setting this bit to 1 enables an active software reset that is functionally equivalent to a hardware reset. Reading the MAILBOX_SYSCONFIG[0] SOFTRESET bit gives the status of the software reset:

- Read 1: the software reset is on-going.
- Read 0: the software reset is complete.

The software must ensure that the software reset completes before doing mailbox operations.

7.1.3.3 Mailbox Power Management

The clkstop_idle will be asserted if all of the following are true:

Copyright © 2024 Texas Instruments Incorporated

- The VBUSP interface is inactive
- All mailboxes are empty
- There are no enabled events or outstanding interrupts

An enabled event means there is pending action required from one of the users, and it means one or more mailboxes still expect data.

7.1.3.4 Mailbox Interrupt Requests

An interrupt request allows the user of the mailbox to be notified when a message is received or when the message queue is not full. There is one interrupt per user.

Table 7-6 lists the event flags, and their mask, that can cause module interrupts.

Table 7-6. Interrupt Events

Non-Maskable Event Flag[(1)]	Maskable Event Flag	Event Mask Bit	Event Unmask Bit	Description
MAILBOX_IRQ_STATUS_RAW_j[0+y*2] NEWMSGSTATUSMBy	MAILBOX_IRQ_STATUS_CLR_j[0+y*2] NEWMSG_STATUSMBy	MAILBOX_IRQ_ENABLE_CLR_j[0+y*2] NEWMSG_STATUSMBy	MAILBOX_IRQ_ENABLE_SET_j[0+y*2] NEWMSG_STATUSMBy	Mailbox *y* receives a new message.
MAILBOX_IRQ_STATUS_RAW_j[1+y*2] NOTFULLSTATUSMBy	MAILBOX_IRQ_STATUS_CLR_j[1+y*2] NOTFULLSTATUSMBy	MAILBOX_IRQ_ENABLE_CLR_j[1+y*2] NOTFULLSTATUSMBy	MAILBOX_IRQ_ENABLE_SET_j[1+y*2] NOTFULLSTATUSMBy	Mailbox *y* message queue is not full.

(1) MAILBOX_IRQ_STATUS_RAW_j register is mostly used for debug purposes.

CAUTION

Once an event generating the interrupt request has been processed by the software, it must be cleared by writing a logical 1 in the corresponding bit of the MAILBOX_IRQ_STATUS_CLR_j register.

Writing a logical 1 in a bit of the MAILBOX_IRQ_STATUS_CLR_j register will also clear to 0 the corresponding bit in the appropriate MAILBOX_IRQ_STATUS_RAW_j register.

An event can generate an interrupt request when a logical 1 is written to the corresponding unmask bit in the MAILBOX_IRQ_ENABLE_SET_j register. Events are reported in the appropriate MAILBOX_IRQ_STATUS_CLR_j and MAILBOX_IRQ_STATUS_RAW_j registers.

An event stops generating interrupt requests when a logical 1 is written to the corresponding mask bit in the MAILBOX_IRQ_ENABLE_CLR_j register. Events are only reported in the appropriate MAILBOX_IRQ_STATUS_RAW_j register.

In case of the MAILBOX_IRQ_STATUS_RAW_j register, the event is reported in the corresponding bit even if the interrupt request generation is disabled for this event.

7.1.3.5 Mailbox Assignment

7.1.3.5.1 Description

To assign a receiver to a mailbox, set the new message interrupt enable bit corresponding to the desired mailbox in the MAILBOX_IRQ_ENABLE_SET_j register. The receiver reads the MAILBOX_MESSAGE_y register to retrieve a message from the mailbox.

An alternate method for the receiver that does not use the interrupts is to poll the MAILBOX_FIFO_STATUS_y and/or MAILBOX_MSG_STATUS_y registers to know when to send or retrieve a message to or from the mailbox. This method does not require assigning a receiver to a mailbox. Because this method does not include the explicit assignment of the mailbox, the software must avoid having multiple receivers use the same mailbox, which can result in incoherency.

To assign a sender to a mailbox, set the queue-not-full interrupt enable bit of the desired mailbox in the MAILBOX_IRQ_ENABLE_SET_j register, where *u* is the number of the sending user. However, direct allocation of a mailbox to a sender is not recommended because it can cause the sending processor to be constantly interrupted.

Copyright © 2024 Texas Instruments Incorporated

It is recommended that register polling be used to:

- Check the status of either the MAILBOX_FIFO_STATUS_y or MAILBOX_MSG_STATUS_y registers
- Write the message to the corresponding MAILBOX_MESSAGE_y register, if space is available.

The sender might use the queue-not-full interrupt when the initial mailbox status check indicates the mailbox is full. In this case, the sender can enable the queue-not-full interrupt for its mailbox in the appropriate MAILBOX_IRQ_ENABLE_SET_j register. This allows the sender to be notified by interrupt only when a FIFO queue has at least one available entry.

Reading the MAILBOX_IRQ_STATUS_CLR_j register determines the status of the new message and the queue-not-full interrupts for a particular user. Writing 1 to the corresponding bit in the MAILBOX_IRQ_STATUS_CLR_j register acknowledges, and subsequently clears, an interrupt.

CAUTION

Assigning multiple senders or multiple receivers to the same mailbox is not recommended.

7.1.3.6 Sending and Receiving Messages

7.1.3.6.1 Description

When a 32-bit message is written to the MAILBOX_MESSAGE_y register, the message is appended into the FIFO queue. This queue holds four messages. If the queue is full, the message is discarded.

Queue overflow can be avoided by first reading the MAILBOX_FIFO_STATUS_y register to check that the mailbox message queue is not full before writing a new message to it.

Reading the MAILBOX_MESSAGE_y register returns the message at the beginning of the FIFO queue and removes it from the queue. If the FIFO queue is empty when the MAILBOX_MESSAGE_y register is read, the value 0 is returned.

The new message interrupt is asserted when at least one message is in the mailbox message FIFO queue. To determine the number of messages in the mailbox message FIFO queue, read the MAILBOX_MSG_STATUS_y register.

Copyright © 2024 Texas Instruments Incorporated

7.1.3.7 Example of Communication

Figure 7-3 shows an example of communication between two processors.

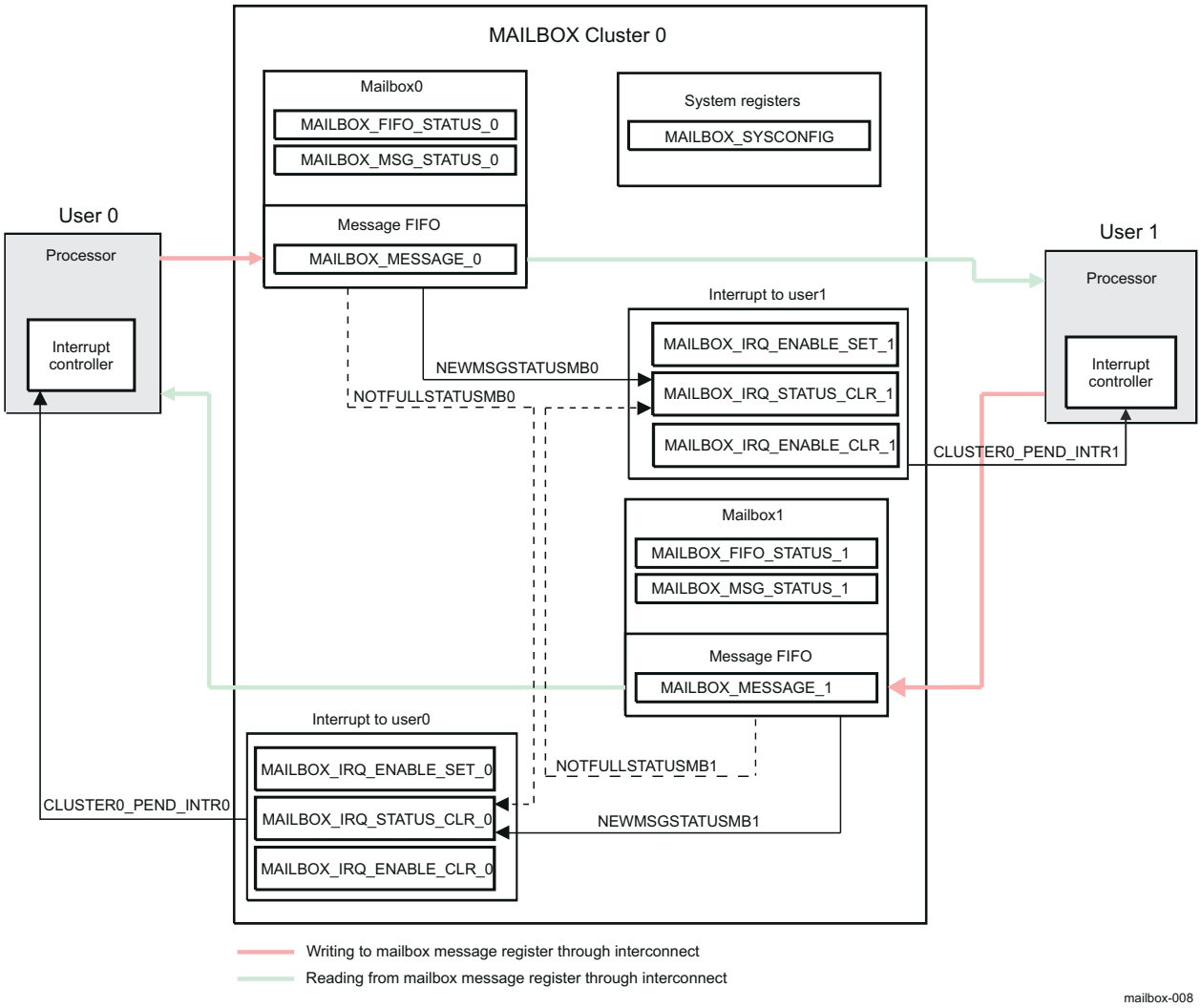


Figure 7-3. Example of Communication

Copyright © 2024 Texas Instruments Incorporated

7.1.4 Mailbox Programming Guide

7.1.4.1 Mailbox Low-level Programming Models

This section covers the low-level hardware programming sequences for configuration and usage of the mailbox module.

7.1.4.1.1 Global Initialization

7.1.4.1.1.1 Surrounding Modules Global Initialization

This section identifies the requirements of initializing the surrounding modules when the mailbox module is to be used for the first time after a device reset. This initialization of surrounding modules is based on the integration of the mailbox.

See Section 7.1.2, *Mailbox Integration*, for further information.

Table 7-7. Global Initialization of Surrounding Modules for MAILBOX

Surrounding Modules	Comments
LPSC	The MAILBOX functional and interface clock must be enabled.
Interrupt Controllers	Processor's (user's) interrupt controller must be configured to enable the interrupt request generation.

7.1.4.1.1.2 Mailbox Global Initialization

7.1.4.1.1.2.1 Main Sequence - Mailbox Global Initialization

This procedure initializes the mailbox module after a power-on or software reset.

Table 7-8. Mailbox Global Initialization

Step	Register/Bit Field/Programming Model	Value
Perform a software reset	MAILBOX_SYSCONFIG[0] SOFT_RESET	0x1
Wait until reset is complete	MAILBOX_SYSCONFIG[0] SOFT_RESET	=0x0

7.1.4.1.2 Mailbox Operational Modes Configuration

7.1.4.1.2.1 Mailbox Processing modes

7.1.4.1.2.1.1 Main Sequence - Sending a Message (Polling Method)

Table 7-9. Sending a Message (Polling Method)

Step	Register/Bit Field/Programming Model	Value
IF: Is FIFO full?	MAILBOX_FIFO_STATUS_y[0] FULL	=0x1
Wait until at least one message slot is available	MAILBOX_FIFO_STATUS_y[0] FULL	=0x0
ELSE		
Write message	MAILBOX_MESSAGE_y[31:0] VALUE	0x-
ENDIF		

7.1.4.1.2.1.2 Main Sequence - Sending a Message (Interrupt Method)

Table 7-10. Sending a Message (Interrupt Method)

Step	Register/Bit Field/Programming Model	Value
IF: Is FIFO full?	MAILBOX_FIFO_STATUS_y[0] FULL	=0x1
Enable interrupt event	MAILBOX_IRQ_ENABLE_SET_j[1+ y*2]	0x1
User (processor) can perform another task until interrupt occurs See Section 7.1.4.1.3.1 for interrupt handling in sending mode		
ELSE		
Write message	MAILBOX_MESSAGE_y[31:0] VALUE	0x-
ENDIF		

Copyright © 2024 Texas Instruments Incorporated

7.1.4.1.2.1.3 Main Sequence - Receiving a Message (Polling Method)

Table 7-11. Receiving a Message (Polling Method)

Step	Register/Bit Field/Programming Model	Value
IF: Number of messages is not equal to 0	MAILBOX_MSG_STATUS_y[2:0] NUM_MESSAGES	≠0x0
Read message	MAILBOX_MESSAGE_y[31:0] VALUE	0x-
ENDIF		

7.1.4.1.2.1.4 Main Sequence - Receiving a Message (Interrupt Method)

Table 7-12. Receiving a Message (Interrupt Method)

Step	Register/Bit Field/Programming Model	Value
Enable interrupt event	MAILBOX_IRQ_ENABLE_SET_j[0 + y*2]	0x1
User (processor) can perform another task until interrupt occurs See Section 7.1.4.1.3.2 for interrupt handling in receiving mode		

7.1.4.1.3 Mailbox Events Servicing

7.1.4.1.3.1 Events Servicing in Sending Mode

Table 7-13 describes the events servicing in sending mode.

Table 7-13. Events Servicing in Sending Mode

Step	Register/Bit Field/Programming Model	Value
Read interrupt status bit	MAILBOX_IRQ_STATUS_CLR_j[1 + y*2]	0x1
Write message	MAILBOX_MESSAGE_y[31:0] VALUE	0x--
Write 1 to acknowledge interrupt	MAILBOX_IRQ_STATUS_CLR_j[1 + y*2]	0x1

7.1.4.1.3.2 Events Servicing in Receiving Mode

Table 7-14 describes the events servicing in receiving mode.

Table 7-14. Events Servicing in Receiving Mode

Step	Register/Bit Field/Programming Model	Value
Read interrupt status bit	MAILBOX_IRQ_STATUS_CLR_j[0 + y*2]	0x1
IF: Number of messages is not equal to 0?	MAILBOX_MSG_STATUS_y[2:0] NUM_MESSAGES	≠0x0
Read message	MAILBOX_MESSAGE_y[31:0] VALUE	0x--
ELSE		
Write 1 to acknowledge interrupt	MAILBOX_IRQ_STATUS_CLR_j[0 + y*2]	0x1
ENDIF		

Copyright © 2024 Texas Instruments Incorporated

7.2 Spinlock

This chapter describes the Spinlock module of the device.

7.2.1 Spinlock Overview

The Spinlock module provides hardware assistance for synchronizing the processes running on multiple processors in the device.

The Spinlock module implements 256 spinlocks (or hardware semaphores), which provide an efficient way to perform a lock operation of a device resource using a single read-access, avoiding the need of a read-modify-write bus transfer that the programmable cores are not capable of.

7.2.2 Spinlock Functional Description

7.2.2.1 Spinlock Software Reset

The Spinlock module can be reset by software through the SPINLOCK_SYSCONFIG[1] SOFTRESET bit. Setting this bit to 1 enables an active software reset that is functionally equivalent to a hardware reset.

7.2.2.2 Spinlock Power Management

The Spinlock module supports the Power Idle interface. Software must arrange to only power off the Spinlock module when no locks would be lost. In general, the steps to powering down the Spinlock module are:

1. Software must check that all controllers that may be using the Spinlock module are either:
   a. Already powered off (all in-use flags are 0)
   b. Notified that Spinlock is not available and the notification is acknowledged
2. If desired, check that no locks are currently held in the Spinlock module. The status of each bank of 32 locks can be read from the SPINLOCK_SYSTATUS register. If any locks are held, they are orphaned because they are not held by any controller that is still active. Alternatively, you may decide to wait a timeout period to allow any active controller to clean up its locks before powering down.

The Spinlock module may now be powered off.

7.2.2.3 About Spinlocks

Spinlocks are present to solve the need for synchronization and mutual exclusion between heterogeneous processors and those not operating under a single, shared operating system. There is no alternative mechanism to accomplish these operations between processors in separate subsystems.

Spinlocks are not the best way to synchronize between tasks or threads on one CPU. Instead, spinlocks are for use in synchronization between different subsystems in the device that don't have any other means of hardware-based synchronization.

Spinlocks do not solve all system synchronization issues. They have limited applicability and should be used with care to implement higher level synchronization protocols.

A spinlock is appropriate for mutual exclusion for access to a shared data structure. It should be used only when:

1. The time to hold the lock is predictable and small (for example, a maximum hold time of less than 200 CPU cycles may be acceptable).
2. The locking task cannot be preempted, suspended, or interrupted while holding the lock (this would make the hold time large and unpredictable).
3. The lock is lightly contended, that is the chance of any other process (or processor) trying to acquire the lock while it is held is small.

If these conditions are met, then the locking code can retry a failed attempt to acquire the lock until success.

If the conditions are not met, then a spinlock is not a good candidate. One alternative is to use a spinlock for critical section control (engineered to meet the conditions) to implement a higher level semaphore that can support preemption, notification, timeout or other higher level properties.

7.2.2.4 Spinlock Functional Operation

The Spinlock module supports 256 spinlocks. It accepts only a single command at a time and processes the command fully before accepting the next command. A lock is requested by reading the

Copyright © 2024 Texas Instruments Incorporated

SPINLOCK_LOCK_REG_i[0] TAKEN bit. There are two states: Taken (SPINLOCK_LOCK_REG_y[0] TAKEN = 1) or Not Taken (SPINLOCK_LOCK_REG_y[0] TAKEN = 0), where y = 0h to FFh.

When the status of lock *y* (where y = 0 to 255) is Not Taken (free), a read from the SPINLOCK_LOCK_REG_y register returns 0 and sets the lock to Taken (locked). When the status of lock *y* is Taken, a read returns 1 and does not change the state of the lock.

A write to the SPINLOCK_LOCK_REG_y register does not change the state of lock, unless when writing 0 when the lock is in Taken state. By doing this, the requester frees the lock.

Figure 7-4 shows the SPINLOCK_LOCK_REG_y register state diagram.

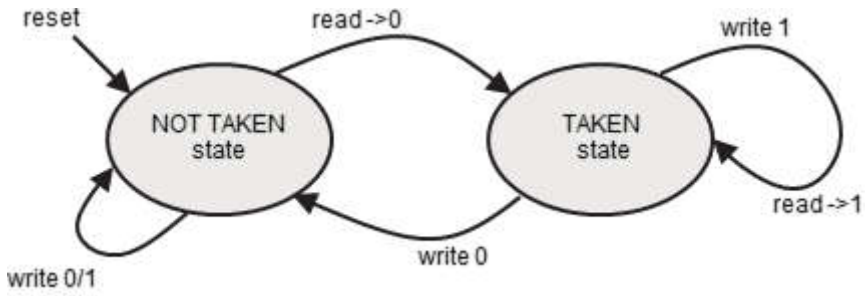


Figure 7-4. Lock Register State Diagram

Note

- There is no support to ensure that a lock register is locked and unlocked by the same process. This must be ensured in software.
- There is no support to check that the same initiator that acquired the lock is the one that is freeing the lock.

Copyright © 2024 Texas Instruments Incorporated

7.2.3 Spinlock Programming Guide

7.2.3.1 Spinlock Low-level Programming Models

This section covers the low-level hardware programming sequences for configuration and usage of the module.

7.2.3.1.1 Basic Spinlock Operations

The main Spinlock operations are:

- Clear all the Taken spinlocks by writing 0 to SPINLOCK_LOCK_REG_y (only after a system bug recovery)
- Take a spinlock by reading the SPINLOCK_LOCK_REG_i[0] TAKEN bit
- Release spinlock by writing 0 to SPINLOCK_LOCK_REG_i[0] TAKEN bit

7.2.3.1.1.1 Spinlocks Clearing After a System Bug Recovery

Module initialization (after reset) is not needed, except after system bug recovery. The following table presents the Spinlock initialization after a system bug recovery. Software should store 0 into each of the SPINLOCK_LOCK_REG_y registers at system startup to insure that all locks are initialized to Not Taken.

Table 7-15. Spinlock System Bug Recovery

Step	Register	Value
IF: SPINLOCK_SYSTATUS[0] IU0 = 1?	SPINLOCK_SYSTATUS[0] IU0	=1
Free the 256 locks	SPINLOCK_LOCK_REG_y[0] TAKEN (y = 0 to 255)	0x0
END		

7.2.3.1.1.2 Take and Release Spinlock

This procedure configures the take and release (free) operations for the Spinlock module. A spinlock should only be held with interrupts disabled. So, before attempting to obtain the spinlock, software must disable interrupts. Then it must read the SPINLOCK_LOCK_REG_y[0] TAKEN bit to attempt to obtain the lock. If it succeeds, it must proceed directly through the critical section then unlock and re-enable interrupts. If the acquisition attempt fails, the acquisition must be reattempted. To prevent unknown interrupt disabled time, interrupts must be re-enabled and then disabled before reattempting to acquire the lock. Figure 7-5 shows the described above procedure.

Copyright © 2024 Texas Instruments Incorporated

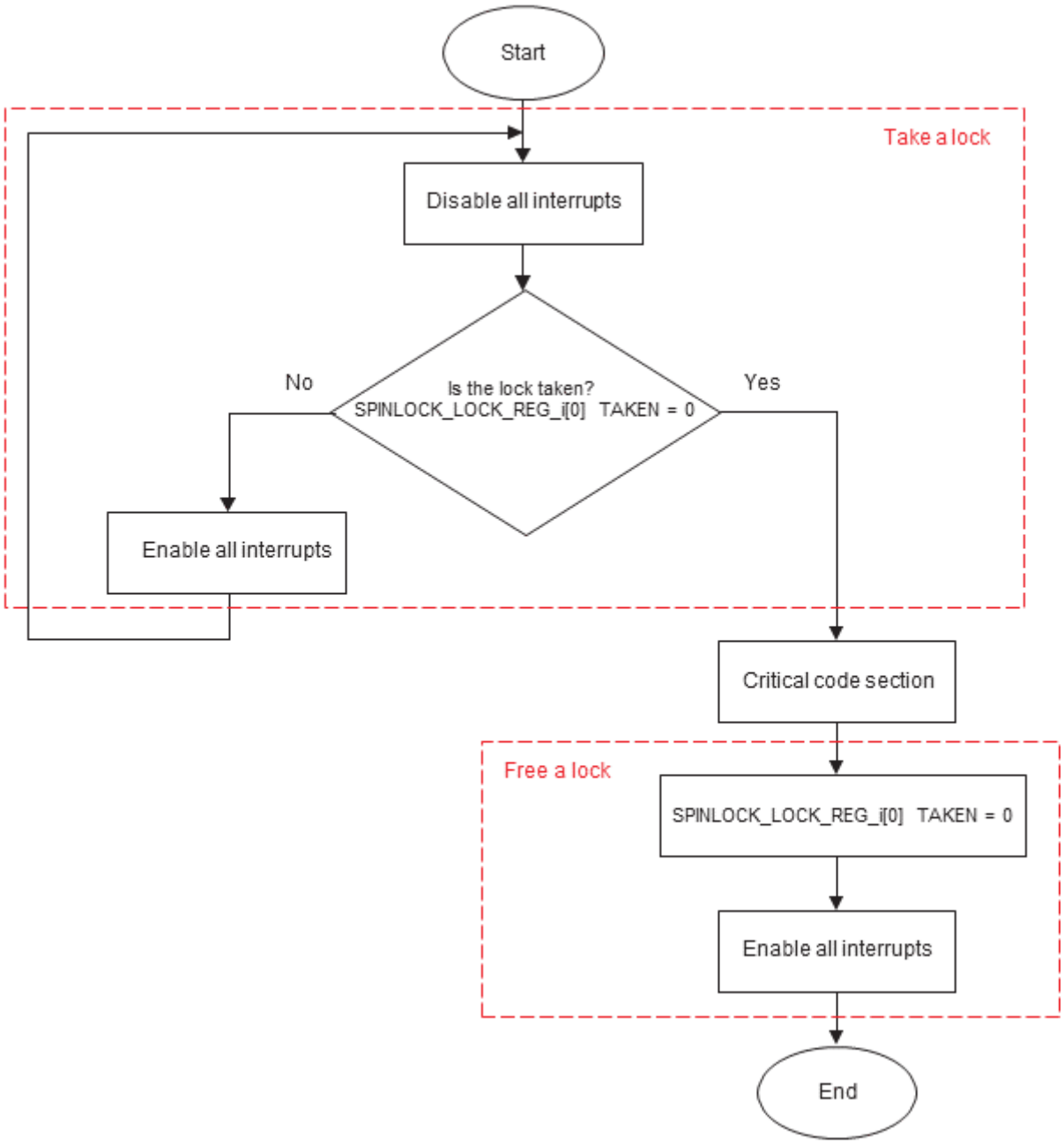


Figure 7-5. Take and Release Spinlock

Table 7-16. Register Call Summary

Register Name
SPINLOCK_LOCK_REG_y[0] TAKEN

Table 7-17. Subprocess Call Summary

Subprocess Name	Description
Disable (Mask) All Interrupts	For information about disabling/enabling all interrupts in an Arm® processor, refer to Arm *Technical Reference Manual*, available at infocenter.arm.com/help/index.jsp. For information about disabling/enabling all interrupts in other processors, refer to the corresponding processor chapter.
Enable (Unmask) All Interrupts	

Copyright © 2024 Texas Instruments Incorporated

8 Memory Controllers

This chapter describes the memory controllers in the device.

8.1 Multicore Shared Memory Controller (MSMC) 696
8.2 DDR Subsystem (DDRSS) 718
8.3 Peripheral Virtualization Unit (PVU) 763
8.4 Region-based Address Translation (RAT) Module 768

Copyright © 2024 Texas Instruments Incorporated

8.1 Multicore Shared Memory Controller (MSMC)

This section describes the Multicore Shared Memory Controller (MSMC) for the device.

8.1.1 MSMC Overview

The Multicore Shared Memory Controller (MSMC) forms the heart of the compute cluster (COMPUTE_CLUSTER0) providing high-bandwidth resource access both to and from all of the connected processing elements and the rest of the system. MSMC serves as the data-movement backbone of the compute cluster.

Table 8-1 shows MSMC modules allocation within device domains.

Table 8-1. MSMC Modules Allocation within Device Domains

Module Instance	Domain		
	WKUP	MCU	MAIN
MSMC	-	-	✓

Figure 8-1 shows an overview of the MSMC and its surrounding modules.

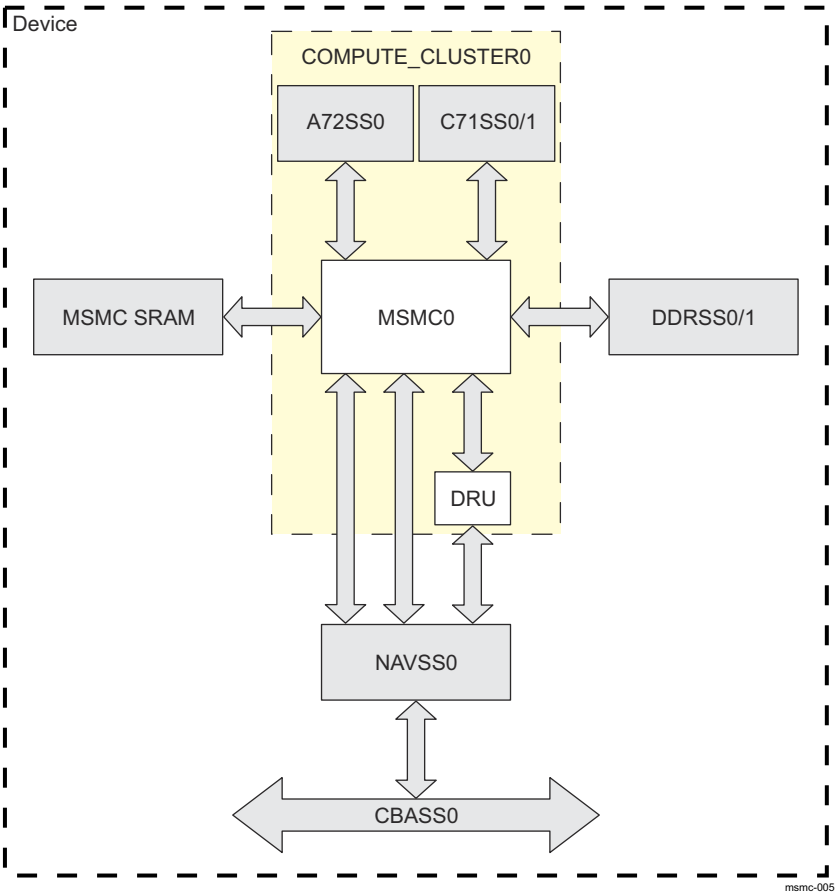


Figure 8-1. MSMC Overview

MSMC supports the following features:

- 4MB (4 banks x 1MB) SRAM with ECC:
  - Shared coherent level 2/level 3 memory-mapped SRAM
  - Shared coherent level 3 cache
- 512-bit processor port bus and 40-bit physical address bus
- Coherent unified bi-directional interfaces to connect to processors or device masters
- One infrastructure master interface

Copyright © 2024 Texas Instruments Incorporated

- Dual external memory master interface
- Supports distributed virtual system
- Supports internal DMA engine – DRU (Data Routing Unit)
  - DMA in/out L2 SRAM, MSMC, DDR and system
  - L2, L3 cache pre-warming and post flushing
- Bandwidth management with starvation bound
- Two-level QoS support for real-time/nonreal-time split
- Security firewall flush support for SRAM/cache and external memory
- Functional reliability:
  - SEC/DED protection on all data and tag memories with hardware scrubbing
  - SEC/DED protection on all data pipelines
  - Data memory address hamming protection
  - Coherent interconnect transaction metadata parity protection
- One interconnect messaging interface that supports DMA/prefetch requests to DRU
- Trace and debugging support
- Supports dynamic clock gating on all logic units
- MSMC is always on when VD_CORE is on
- MSMC R50+ Features
  - CMMU Compression support
  - FFI support between A72/C7x and R5F
  - DDR Interleaving
  - Way group optimization

8.1.1.1 MSMC Not Supported Features

MSMC does not support the following:

- RAM address decode protection
- Direct cache resize changes from one non-zero cache size configuration to another non-zero cache size configuration. In this case software is required to manually transition down to zero cache size configuration first, followed by a second transition from zero cache size configuration to the new non-zero cache size configuration.
- MSMC SRAM or SDRAM traffic during an MSMC cache resize transition. The traffic during this transition has undefined behavior.

Copyright © 2024 Texas Instruments Incorporated

8.1.2 MSMC Functional Description

8.1.2.1 MSMC Block Diagram

Figure 8-2 shows a high-level view of the MSMC module that includes the main interfaces, memory, and subunits.

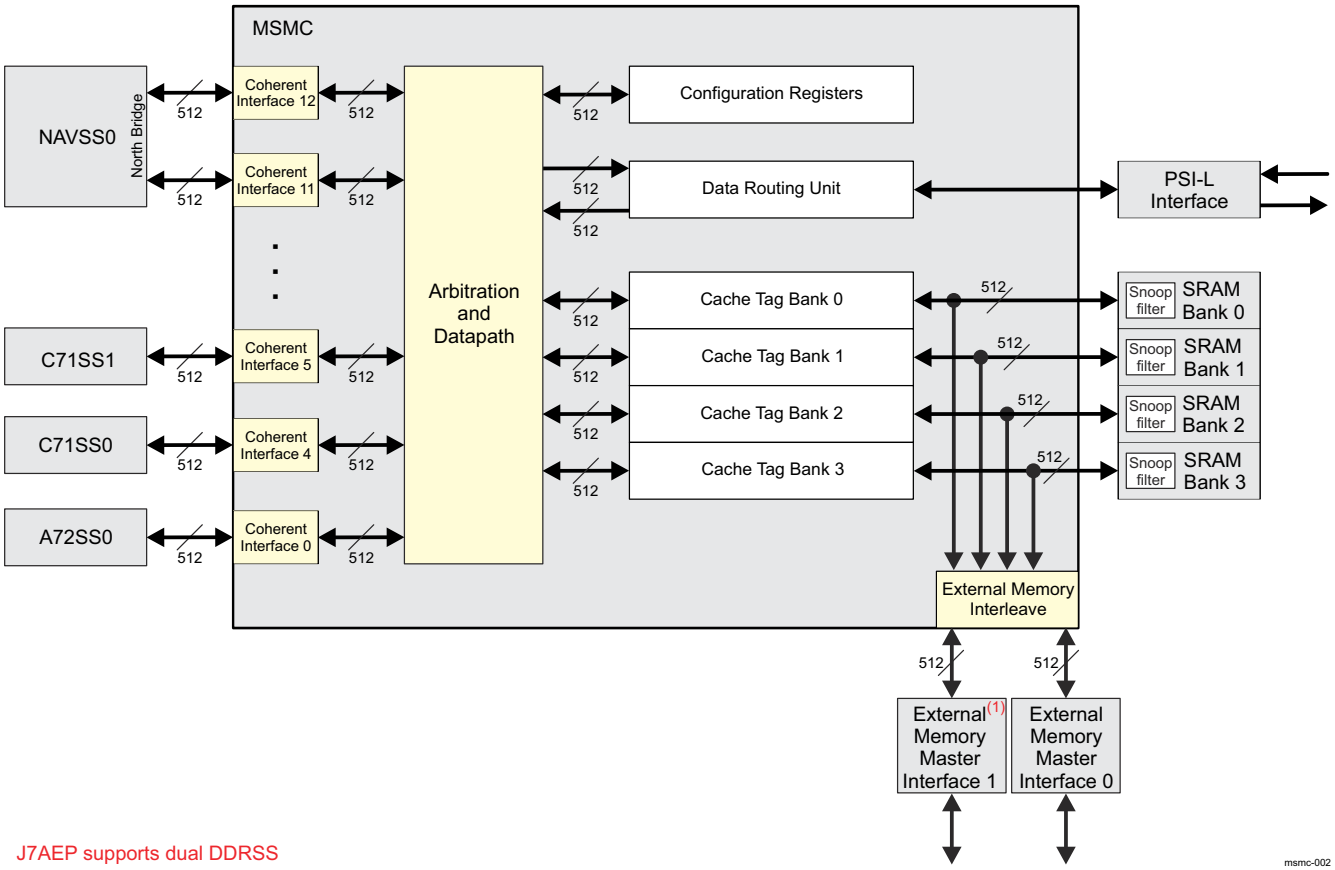


Figure 8-2. MSMC Functional Block Diagram

MSMC directly incorporates on-chip SRAM controllers to provide the compute cluster with low-latency memory. In addition, the individual memory banks can maintain coherence with the connected caching masters as well as the system slave ports.

8.1.2.2 MSMC On-Chip Memory Banking

MSMC on-chip SRAM supports physical and virtual banking to maximize the available bandwidth to all masters. The total SRAM space is divided into four physical banks, and each physical bank contains two virtual sub-banks. These virtual banks provide access to that physical memory bank every MSMC_CLK cycle.

Figure 8-3 shows the MSMC memory organization. Each SRAM bank provides 64 bytes per MSMC_CLK cycle.

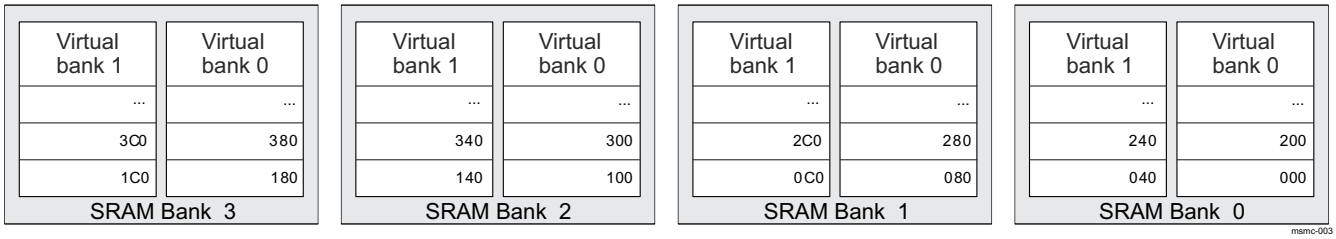


Figure 8-3. MSMC Memory Organization

Copyright © 2024 Texas Instruments Incorporated

8.1.2.3 MSMC Snoop Filter and Data Cache

Each MSMC SRAM bank implements the memory-mapped SRAM snoop filter in its own memory structure similar to a cache tag memory. The snoop filter tracks atomic coherent blocks at the same granularity as the level 3 cache for external memory: 128 bytes.

MSMC also implements a combined snoop filter and level 3 cache for the connected external memory space. As with the memory-mapped SRAM, the snoop filter exists to optimize performance of coherent external memory data cached in the attached level 1 and level 2 coherent caches. The external memory snoop filter is a 32-way set-associative cache structure which tracks coherent data at the same granularity as the SRAM snoop filter: 128 bytes.

SoC and DRU transactions use snoop filter allocation policy which directs allocations strictly into the data-backed portion (ways) of the L3 cache. Random way selection only occurs when the targeted data-backed or non-data-backed portion (ways) of the L3 cache set are fully occupied. This applies also to CPU transactions, if the MSMC_CACHE_CTRL[10] ALLOCATION_POLICY bit is set to 0x0. When the MSMC_CACHE_CTRL[10] ALLOCATION_POLICY bit is set to 0x1, all CPU transactions use a randomized allocation policy which can select between both data-backed and non-data-backed ways.

The level 3 snoop filter/cache randomly chooses a way to replace when allocating a new cache line. The foundation of this random replacement is a 24-bit maximal period Linear Feedback Shift Register (LFSR) tuned to minimize any bias. The state of the LFSR supplies two to five bit allocation pointers depending on the configured cache size/number of ways. Each bank houses its own separately initialized LFSR which updates upon each allocation. For debugging purposes, MSMC supplies a linear mode replacement policy which initializes to the configured number of ways, eventually decrement to zero with allocations, and wrap back around to the total number of ways. Software can enable the linear replacement policy by writing 0x1 to the MSMC_CACHE_CTRL[8] REPLACEMENT_POLICY bit.

8.1.2.3.1 Way Partitioning

The random replacement implementation attempts to minimize any allocation bias throughout the ways for a large number of allocations. However, it can be useful to introduce purposeful bias to help keep the state of certain tasks resident in the cache. To serve this end, MSMC supports programmable way-partitioning based on real-time versus nonreal-time traffic.

To accomplish this without introducing other unpredictable bias MSMC provides groups of AND-OR masks which can be attached to a group of IDs. The AND-OR masks consist of two fields: a two-bit AND field and a two-bit OR field. MSMC applies these two fields to the lower two bits of the randomly chosen allocation way. Figure 8-4 shows how MSMC calculates the final allocation pointer using the initial random state and the user-defined AND-OR mask.

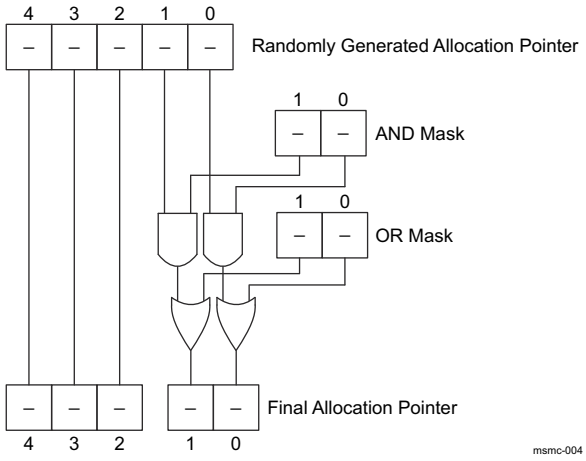


Figure 8-4. Way Partition Allocation Pointer Generation

The two-bit masks allow rough partitioning for quadrants of the cache. Depending on the configured cache size, the cache consists of 0-7 groups of 4 ways each, for a maximum of 32 ways. The upper 3 bits in the allocation

Copyright © 2024 Texas Instruments Incorporated

pointer determine which group of 4 ways. The final two bits determine selection between the 4 ways within that group. Table 8-2 describes the possible values for the AND-OR mask and the resulting available ways.

Table 8-2. AND-OR Mask Way Selection

Way 3	Way 2	Way 1	Way 0	AND Mask	OR Mask
X	X	X	X	11	00
X	X			-1	10
		X	X	01	00
X		X		1-	01
	X		X	10	00
X				--	11
	X			-0	10
		X		0-	01
			X	00	00

The AND-OR masks are programmed via the following bit fields:

- MSMC_RT_WAY_SELECT[6-5] OR_MASK
- MSMC_RT_WAY_SELECT[1-0] AND_MASK
- MSMC_NRT_WAY_SELECT[6-5] OR_MASK
- MSMC_NRT_WAY_SELECT[1-0] AND_MASK

8.1.2.3.2 Cache Size Configuration and Associativity

MSMC has registers to provide software control to determine which portion of the on-chip SRAM will be used as cache data store. To maximize this flexibility, the associativity of the level 3 data cache scales with the desired cache size. The 32 ways in each snoop filter set are separated into 8 groups of 4 ways each as shown in Table 8-3.

Table 8-3. Cache Way Grouping

	Group 0	Group 1	Group 2	Group 3
Way 0	0	4	8	12
Way 1	1	5	9	13
Way 2	2	6	10	14
Way 3	3	7	11	15
	Group 4	**Group 5**	**Group 6**	**Group 7**
Way 0	16	20	24	28
Way 1	17	21	25	29
Way 2	18	22	26	30
Way 3	19	23	27	31

The MSMC_CACHE_CTRL[3-0] CACHE_SIZE bit field controls the cache size by selecting how many way groups are backed with data. Table 8-4 describes the encodings of the CACHE_SIZE field.

Table 8-4. Cache Size Field Encodings

CACHE_SIZE	Number of Active Way Groups	Total Active Data Ways
0x0	0	0 ways
0x1	1	4 ways
0x2	2	8 ways
0x3	3	12 ways
0x4	4	16 ways
0x5	5	20 ways
0x6	6	24 ways
0x7	7	28 ways

Copyright © 2024 Texas Instruments Incorporated

Table 8-4. Cache Size Field Encodings (continued)

CACHE_SIZE	Number of Active Way Groups	Total Active Data Ways
0x8	8	32 ways

Writing a new value to the CACHE_SIZE field begins the cache size transition process. The MSMC hardware sets the MSMC_CACHE_CTRL[4] SZ_TRANSITION bit to 0x1 during this transition process. When the transition process completes hardware sets SZ_TRANSITION to 0x0. During this transition time window:

- Further writes to the CACHE_SIZE field are ignored
- Reads to the CACHE_SIZE field reflect the previous field value.

After setting a new CACHE_SIZE value, software can poll the SZ_TRANSITION bit to know when the transition is complete. At that point the new CACHE_SIZE value can be read in the MSMC_CACHE_CTRL[3-0] CACHE_SIZE bit field.

As the cache scales up in size, the memory-mapped space shrinks by the same amount, appearing as though the cache grows from the top of the memory mapped region down. This provides a single contiguous memory-mapped window starting from the SRAM base address. The memory space occupied by the cache is no longer accessible as memory-mapped SRAM and any accesses to this space return an addressing error.

8.1.2.4 MSMC Access Protection Checks

To support potential software polling/reconfiguration for functional reliability, MSMC does not trigger error status for configuration accesses to unimplemented addresses. All reads of unimplemented addresses return data as zero and writes are ignored. This model should support a simple DMA "read, compare, and write-back" model for checking the MSMC configuration state.

8.1.2.5 MSMC Null Slave

MSMC implements a null slave endpoint to handle accesses which cannot be sent to their intended destination. This can happen for several reasons listed in Table 8-5. Commands routed to null slave return to the originating master with the appropriate response status listed in Table 8-5. MSMC prioritizes protection error status over addressing error status when both can be applied to a single command.

Table 8-5. Null Slave Response Status

Description	Response Status
Firewall Flush	Protection Error
Invalid Memory Range	Addressing Error
Coherent Command to DMA	Success
Disabled Endpoint	Addressing Error
Real-time to DMA	Addressing Error

MSMC captures command information and triggers an event when routing unexpected command to null slave. Any of the conditions in Table 8-5 which respond with a faulted response status (except firewall failures) triggers null slave error interrupt and logs appropriate command information to the MSMC_NULL_SLV_STAT0 and MSMC_NULL_SLV_STAT1 registers. The following bit fields capture that command information:

- ADDR
- BYTECNT
- ROUTEID
- PRIVID
- OPCODE
- MEMTYPE
- EMU
- SEC
- PRIV

For more information about the MSMC interrupts, see Section 8.1.2.8.

MSMC captures a single null slave error event into the MSMC_NULL_SLV_STAT0 and MSMC_NULL_SLV_STAT1 registers.

Copyright © 2024 Texas Instruments Incorporated

8.1.2.6 MSMC Resource Arbitration

When access requests arrive at the MSMC slave ports, the requests may target the same endpoint and hence need arbitration. These endpoint arbiters implement a multi-layer fairness arbitration scheme with starvation bounds to allow the user control to partition system bandwidth.

In each arbitration cycle, each arbiter prioritizes all active requests using the following prioritized layers until it selects a unique winner:

1. Transaction-tagged priority - Each active requestor pushes out a priority based on all of the currently active transactions in its pipeline. The highest priority requestors are chosen.
2. Fair-Share State - The arbiter selects the requestor(s) with the highest internal Fairshare state.
3. Static Priority - Finally, the arbiter uses a static priority to break any remaining ties after levels 1 and 2.

In addition to the three levels the arbiter tracks a starvation count for each requestor to each arbiter. Every time a request wins arbitration, all loser requests have their starvation count decremented. Every time a requestor wins arbitration for an active request, the starvation count resets to the initial state. If the starvation count reaches zero, that requestor priority level elevates to the highest priority level. Once the starved active request wins arbitration the priority level reverts back to the normal value. These starvation bounds can be programmed via the MSMC_SBNDCOH0 through MSMC_SBNDCOH12 registers.

Note

Not all MSMC_SBNDCOHx registers are used on this device.

The three-layer arbitration scheme is not sufficient to ensure a bound on the minimum bandwidth experienced by lower priority requests relative to higher priority requests. Consequently, a starvation bound can be defined to limit that minimum bandwidth experienced by the lower priority requests relative to higher priority requests.

Since MSMC introduces multiple virtual channels feeding the same physical arbiter, the starvation mechanism is expanded to specify bounds for these channels.

Table 8-6. Starvation Bound Definitions

Port	Channel	Resource Target	Register Name and Field
Cache coherent 0	Real-time	External memory	MSMC_SBNDCOH0[55-48] SBNDE_RT
		On-chip resources	MSMC_SBNDCOH0[39-32] SBNDM_RT
	Non-real-time	External memory	MSMC_SBNDCOH0[23-16] SBNDE_NRT
		On-chip resources	MSMC_SBNDCOH0[7-0] SBNDM_NRT
Cache coherent 4	Real-time	External memory	MSMC_SBNDCOH4[55-48] SBNDE_RT
		On-chip resources	MSMC_SBNDCOH4[39-32] SBNDM_RT
	Non-real-time	External memory	MSMC_SBNDCOH4[23-16] SBNDE_NRT
		On-chip resources	MSMC_SBNDCOH4[7-0] SBNDM_NRT
Cache coherent 5	Real-time	External memory	MSMC_SBNDCOH5[55-48] SBNDE_RT
		On-chip resources	MSMC_SBNDCOH5[39-32] SBNDM_RT
	Non-real-time	External memory	MSMC_SBNDCOH5[23-16] SBNDE_NRT
		On-chip resources	MSMC_SBNDCOH5[7-0] SBNDM_NRT
Cache coherent 11	Real-time	External memory	MSMC_SBNDCOH11[55-48] SBNDE_RT
		On-chip resources	MSMC_SBNDCOH11[39-32] SBNDM_RT
	Non-real-time	External memory	MSMC_SBNDCOH11[23-16] SBNDE_NRT
		On-chip resources	MSMC_SBNDCOH11[7-0] SBNDM_NRT
Cache coherent 12	Real-time	External memory	MSMC_SBNDCOH12[55-48] SBNDE_RT
		On-chip resources	MSMC_SBNDCOH12[39-32] SBNDM_RT
	Non-real-time	External memory	MSMC_SBNDCOH12[23-16] SBNDE_NRT
		On-chip resources	MSMC_SBNDCOH12[7-0] SBNDM_NRT

Copyright © 2024 Texas Instruments Incorporated

Table 8-6. Starvation Bound Definitions (continued)

Port	Channel	Resource Target	Register Name and Field
DRU	Real-time	External memory	MSMC_SBNDDRU[55-48] SBNDE_RT
		On-chip resources	MSMC_SBNDDRU[39-32] SBNDM_RT
	Non-real-time	External memory	MSMC_SBNDDRU[23-16] SBNDE_NRT
		On-chip resources	MSMC_SBNDDRU[7-0] SBNDM_NRT
Responses Snoops	Real-time	External memory	MSMC_SBNDRESP[55-48] SBNDE_RT
		On-chip resources	MSMC_SBNDRESP[39-32] SBNDM_RT
	Non-real-time	External memory	MSMC_SBNDRESP[23-16] SBNDE_NRT
		On-chip resources	MSMC_SBNDRESP[7-0] SBNDM_NRT

The starvation bounds specified through the registers in Table 8-6 apply to each trip an access requires through an arbitration point. The number of potential arbitrations should be considered when programming the starvation bounds. The following are few common examples of round-trip command arbitration for reads:

- Non-coherent SRAM read: Command central arbitration > Data SRAM arbitration > Response central arbitration
- Coherent SRAM read: Command central arbitration > Snoop central arbitration > Snoop response central arbitration > Read response central arbitration

Depending on whether caching and coherence are enabled and the state of the system, the number of potential arbitrations for a given access may vary and starvation should be considered accordingly.

8.1.2.7 MSMC Error Detection and Correction

MSMC protects both on-chip memory resources and data busses throughout the MSMC pipelines. A mixture of SEC/DED (single-error correct/double-error detect) hamming code and simple parity protect the on-chip memory resources.

Note

MSMC aligns to the device architecture for system reliability using ECC aggregators to report detected parity and EDC faults both from scrubbing and functional access. The MSMC associated ECC aggregators are part of COMPUTE_CLUSTER0. For more information, see *Compute Cluster ECC Aggregators* in *Compute Cluster*.

For information about the ECC aggregator functionality, see *ECC Aggregator*.

8.1.2.7.1 On-chip SRAM and Pipeline Data Protection

The on-chip data memory and data bus pipelines share the same Hamming EDC strategy to maximize protection across MSMC. Two EDC data/hamming pairs combine to form each 64 bytes of data. When functional accesses read data from the memory the EDC code travels through the MSMC pipeline with the data protecting all of those data pipelines in addition to the memory. Consumers of the protected data pairs should perform the EDC check and correction as needed before utilizing the data.

MSMC does not protect stored memory data all of the time. This requires memory initialization after reset to avoid spurious errors. In addition, read-modify-write combine cycle is also needed in case of any write access to the memory that does not commit to a full EDC quanta.

MSMC stores each 532-bit data + hamming code separately in the on-chip SRAM in the following way:

- SRAM[255:0] -> 256-bit Data Word 0
- SRAM[265:256] -> 10-bit Hamming Word 0
- SRAM[266:521] -> 256-bit Data Word 1
- SRAM[531:522] -> 10-bit Hamming Word 1

8.1.2.7.2 On-chip SRAM L3 Cache Tag and Snoop Filter Protection

The level 3 cache/snoop filter tags stored in on-chip SRAM are also SEC/DED protected similarly to the data storage. Due to timing closure including all of the tag bits for a given set into one hamming data + code word is

Copyright © 2024 Texas Instruments Incorporated

not possible. Instead, the 32 ways are broken into 8 groups of 4 ways each comprising hamming data (100 bits) + code (9 bits). The MSMC core logic outputs 8 logical SRAM interfaces for the tag memories based on the EDC protection quanta sizes. The hamming protection bits are stored at the most significant bit positions of the total data + hamming word inside the logical memory instance.

8.1.2.7.3 On-chip SRAM Memory Mapped SRAM Snoop Filter Protection

MSMC stores the memory mapped SRAM snoop filter state in a separate memory from the tags and protects this information with simple parity. Each snoop filter entry requires five data bits and eight coherent blocks are stored together as a parity data word, resulting in one parity bit to every 40 data bits. The parity bit resides at the most significant bit position in the total parity + data word.

8.1.2.7.4 Background Parity Refresh (Scrubbing)

The MSMC has scrubbing engine that periodically cycles through each location of each memory bank in the MSMC, reading and correcting the data, recalculating the parity bits for the data and storing the data and parity information.

The frequency with which each scrubbing cycle is initiated and the delay between each burst by the scrubbing engine is programmed using the MSMC_SMEDCC register. The MSMC_SMEDCC[7-0] REFDEL bitfield controls the number of MSMC_CLK cycles between each scrub. To prevent the bursts from the scrubbing engine from posing a significant performance impact, the value in the REFDEL field is pre-scaled by 1024 in functional mode. A value of 0 is interpreted to be the same as a value of 1. At reset, REFDEL = 0 and a new scrub burst is issued 1024 cycles after the previous one ends. The operation of the scrubbing engine is enabled by default at reset but may be disabled by setting the MSMC_SMEDCC[31] SEN bit to 0x0.

8.1.2.8 MSMC Interrupts

The MSMC has the following registers for interrupt control:

- MSMC_SMIRSTAT - Raw interrupt status
- MSMC_SMIRWS - Raw interrupt software set
- MSMC_SMIRC - Raw interrupt software clear
- MSMC_SMIESTAT - Enable interrupt status
- MSMC_SMIEWS - Enable interrupt software set
- MSMC_SMIEC - Enable interrupt software clear
- MSMC_SMESTAT - Raw and Enabled Interrupt Status

Each bit in these registers represents a single event or interrupt supported by MSMC, which generates only one interrupt. This is the null slave error interrupt (MSMC_NULL_SLAVE_ERROR). MSMC sets this event when routing a command through the MSMC null slave port which results in an unsuccessful response status. Commands failing firewall checks do not trigger the null slave error interrupt event in MSMC.

8.1.2.8.1 Raw Interrupt Registers

MSMC sets the associated bit in the MSMC_SMIRSTAT register when a software or hardware event occurs. Software can trigger the event by setting the corresponding bit in the MSMC_SMIRWS register. MSMC keeps the MSMC_SMIRSTAT bit asserted until software clears it by setting the corresponding bit in MSMC_SMIRC. Both MSMC_SMIRWS and MSMC_SMIRC are write-only registers and do not implement any actual state.

8.1.2.8.2 Interrupt Enable Registers

Software can enable an event by setting the associated bit in the MSMC_SMIEWS register, and MSMC will assert the same bit in the MSMC_SMIESTAT register in response. The bit of MSMC_SMIESTAT remains asserted until software disables the event by setting the corresponding bit in the MSMC_SMIEC register. Both MSMC_SMIEWS and MSMC_SMIEC are write-only registers and do not implement any actual state.

8.1.2.8.3 Triggered and Enabled Interrupts

The MSMC_SMESTAT register is a read-only reflection of events that are both triggered and enabled. It is a combination of MSMC_SMIRSTAT and MSMC_SMIESTAT.

8.1.2.9 MSMC Memory Regions

Device masters access MSMC controlled resources through two memory map windows - internal and external.

Copyright © 2024 Texas Instruments Incorporated

Table 8-7 shows all MSMC associated memory regions including detailed breakdown of the MSMC internal memory mapped window.

Table 8-7. MSMC Memory Regions

Region	Start Address	End Address	Region Size
Internal memory mapped window			
MSMC Coherent Interface for A72SS0	0x00 6000 0000	0x00 60FF FFFF	16 MB
MSMC Coherent Interface for C71SS0	0x00 6400 0000	0x00 64FF FFFF	16 MB
MSMC Coherent Interface for C7xSS1	0x00 6500 0000	0x00 65FF FFFF	16 MB
MSMC Coherent Interface for the NB0 Port of North Bridge in NAVSS0	0x00 6B00 0000	0x00 6BFF FFFF	16 MB
MSMC Coherent Interface for the NB1 Port of North Bridge in NAVSS0	0x00 6C00 0000	0x00 6CFF FFFF	16 MB
DRU Configuration Registers	0x00 6D00 0000	0x00 6DFF FFFF	16 MB
MSMC Configuration Registers(1)	0x00 6E00 0000	0x00 6EFF FFFF	16 MB
MSMC SRAM	0x00 7000 0000	0x00 703F FFFF	4 MB
External memory mapped window			
2 GB External SDRAM Space	0x00 8000 0000	0x00 FFFF FFFF	2 GB
8 GB External SDRAM Space(2)	0x08 0000 0000	0x09 FFFF FFFF	8 GB
Other MSMC associated memory regions			
Compute Cluster ECC Aggregator 0 Configuration Registers	0x4D 2000 0000	0x4D 2000 03FF	1 KB
Compute Cluster ECC Aggregator 1 Configuration Registers	0x4D 2000 0400	0x4D 2000 07FF	1 KB
Compute Cluster ECC Aggregator 2 Configuration Registers	0x4D 2000 0800	0x4D 2000 0BFF	1 KB

(1) The MSMC configuration space supports 1-, 2-, 4- and 8-byte aligned reads and 4- and 8-byte aligned writes.
(2) The first 2 GB of this region are inaccessible.

As shown in Table 8-7 software accesses the memory-mapped on-chip SRAM as part of the MSMC memory mapped space.

8.1.2.10 MSMC Hardware Coherence

MSMC supports hardware cache coherence between CPU cache-coherent masters and peripherals accessing MSMC through the system slave ports for shared SRAM and DDR data range spaces. Hardware coherence support abstracts away the cache memory systems of supported components from software, simplifying software implementation.

MSMC does not support memory coherence for the following spaces:

- DDRSS configuration space
- MSMC configuration space
- System-on-chip master port peripherals/memory
- CPU local resources such as memory-mapped level 2 SRAM
- Any other memory not directly connected to MSMC

To stay coherent with each other, caches inside coherent masters typically track more states than the traditional valid/dirty combinations. MSMC uses the ACE coherence protocol which supports these five MOESI states:

- Modified - Cache line is unique (no other caches currently have it) and dirty
- Owned - Cache line is shared (other caches may have it) and dirty
- Exclusive - Cache line is unique (no other caches have it) and clean
- Shared - Cache line is shared (other caches may have it) and clean
- Invalid - No allocated line

In order to maintain coherence between cached masters, MSMC can initiate requests to coherent masters for shared regions of memory. These requests are referred to as "snoop requests".

Copyright © 2024 Texas Instruments Incorporated

The following example demonstrates how coherence is maintained between DMA and CPU, when DMA writes to shareable memory space.

1. DMA (noncaching master) issues a write to shareable space.
2. MSMC coherence controller issues Invalidate and Writeback snoop to all CPUs.
3. All CPUs invalidate the line from their cache and respond with dirty data if necessary
4. MSMC merges the DMA write data with the victim and commits to memory.

The following example demonstrates how coherence is maintained between DMA and CPU, when DMA reads from shareable memory space.

1. DMA issues a read to shareable space.
2. MSMC coherence controller issues a ReadOnce snoop to the CPU.
3. CPU responds with cached data.
4. MSMC returns the read data to DMA.

8.1.2.10.1 Snoop Filter Broadcast Mode

As described in Section 8.1.2.3 MSMC implements inclusive snoop filters for both memory-mapped SRAM and external memory shared space to limit the amount of required snoop traffic. Both the SRAM and external snoop filters encode the following states for each coherent memory block in their respective filtering structures:

- INVALID - This memory block must be in Invalid state in all coherent CPUs. Since the external snoop filter is inclusive of all cached coherent data, a miss in the snoop filter is functionally equivalent to this state.
- CPU_SHARED - This memory block may be cached in Shared or Owned state in the associated coherent CPU.
- CPU_UNIQUE - This memory block may be cached in Shared, Owned, Exclusive, or Modified state in the associated coherent CPU.
- BROADCAST_SHARED - This memory block may be cached in Shared or Owned states in multiple coherent CPUs.
- BROADCAST_UNIQUE - This memory block may be cached in Shared, Owned, Exclusive, or Modified states in multiple coherent CPUs. Typically lines only reach this state when MSMC encounters a snoop filter parity error or broadcast mode is enabled.

As a safeguard MSMC provides a "Broadcast Mode" configuration that forces MSMC to treat all snoop filter entries as BROADCAST_UNIQUE. This results in snoops generated to all applicable coherent masters for snoop filter hits. This mode can be toggled on and off, but the user should be aware that the snoop filter becomes no longer strictly inclusive once enabled since snoop filter entries accessed during this mode are no longer accurately tracking line ownership. Coherency is still maintained in this mode, though additional potentially unnecessary snoop activity may occur. To enable the broadcast mode feature the MSMC_COHCTRL[0] BCM bit should be set to 0x1.

8.1.2.11 MSMC Quality-of-Service

As an orthogonal axis to arbitration priority MSMC provides two classes of traffic, real-time (RT) and nonreal-time (NRT), to maintain quality-of-service (QoS) guarantees and alleviate head-of-line blocking issues. The currently implemented dataflows supported by the QoS hardware are:

- SoC coherent slave - MSMC SRAM
- SoC coherent slave - External memory
- DRU coherent slave - MSMC SRAM
- DRU coherent slave - External memory

At a high-level priority and QoS represent orthogonal axes controlling the latency (priority) and bandwidth protection (QoS) for multiple data flows routed through the MSMC. To accomplish this MSMC provides a dedicated buffering at each arbitration point that can only be consumed by RT traffic. This way NRT traffic cannot completely starve out RT requests.

The SoC coherent, DRU coherent, and external memory interfaces need dedicated credits to support QoS. Each of these interfaces implements additional threading to exchange these dedicated credits.

Copyright © 2024 Texas Instruments Incorporated

Table 8-8. QoS Threading

Thread Index	[0]	[2]
SoC Coherent Slave	CPU	RT_CPU
DRU Coherent Slave	CPU	RT_CPU
External Memory Master	CPU	RT_CPU

The RT_CPU thread provides a completely separate pool of credits, where necessary, to implement the dedicated buffers for RT traffic.

Note

There is no software control over the MSMC QoS hardware.

Note

Interfaces which do not support QoS features denote all traffic as nonreal-time.

8.1.2.12 MSMC Memory Regions Protection

There is no built-in firewall to protect the on-chip and external memory regions. MSMC relies on the system distributed firewall structure to protect these memory windows. MSMC receives a "pass/fail" attribute on each of its entry points including the DRU interfaces which it uses to route the transaction to either the addressed endpoint (pass) or to the internal null slave (fail). Transactions carrying the "fail" return Protection Error on read or write status. Since these failed transactions route through the null slave endpoint no snoops, allocations, or other hardware side-effects occur as a result of these transactions.

8.1.2.13 MSMC Cache Tag View

MSMC implements debug hardware to allow software to query the state of the SRAM snoop filter and L3 cache tag/snoop filter. Software can query the tag structures using the MSMC_DBGTAGCTL and MSMC_DBGTAGVIEW registers in the following sequence:

1. Write the desired tag or snoop filter information to the MSMC_DBGTAGCTL register and wait for status return
2. Read the MSMC_DBGTAGVIEW register to collect the values populated from step 1

The write in step 1 triggers population of the MSMC_DBGTAGVIEW register with the appropriate values from the tag or snoop filter memory. The following are important notes about this feature:

- Writing invalid values into the MSMC_DBGTAGCTL register results in undefined values in the MSMC_DBGTAGVIEW register.
- All writes to MSMC_DBGTAGCTL triggers re-population of MSMC_DBGTAGVIEW, regardless of whether MSMC_DBGTAGVIEW has been read since the last write to MSMC_DBGTAGCTL
- This feature is intended for debug use when the system is halted or quiet. If software writes MSMC_DBGTAGCTL during active operation MSMC will find an empty cycle on the tag/snoop filter memory to read the contents and return to MSMC_DBGTAGVIEW. However, if there are accesses in flight which operate on the desired tag or snoop filter block the contents of MSMC_DBGTAGVIEW will reflect whatever was in the memory at the time of read.
- Security considerations for cache tag view must be comprehended in the firewalls present before all MSMC request interfaces. MSMC does not perform additional security checks.

8.1.2.14 MSMC R50+ Features

8.1.2.14.1 Way Group Partitioning

Real-time and non-real-time transactions are an example of transactions that should, ideally, have orthogonal processing, so it is important that they are allocated in separate way groups to take advantage of per-way-group hazarding.

In addition to having a configurable number of cache way groups be backed with data,MSMC also provides software control over the number of cache way groups real time and non-real time transactions are each allowed

Copyright © 2024 Texas Instruments Incorporated

to allocate into. The DATABACKED_GROUPS fields within the RT_WAY_SELECT and NRT_WAY_SELECT memory-mapped registers control the number of data backed way groups to be allocatable by real-time and non-real-time transactions, respectively. Similarly, the NON_DATABACKED_GROUPS fields within the RT_WAY_SELECT and NRT_WAY_SELECT memory-mapped registers control the number of way groups without data backing that can be allocated by real-time and non-real-time transactions, respectively.

For any setting of these values MSMC will minimize the number of way groups that are simultaneously allocatable by both real-time and non-real-time transactions. For example, if there are 3 way groups with data backing and both real-time and non-real-time transactions are configured to be allocateable in 2 data backed way groups, then one way group will be allocateable by only real-time transactions, one only by non-realtime transactions, and one that is allocateable by both real-time non-real-time transactions. This is shown in Figure 8-5.

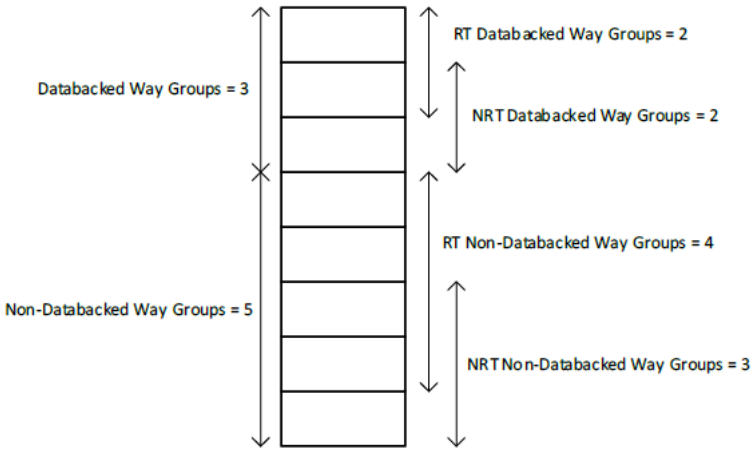
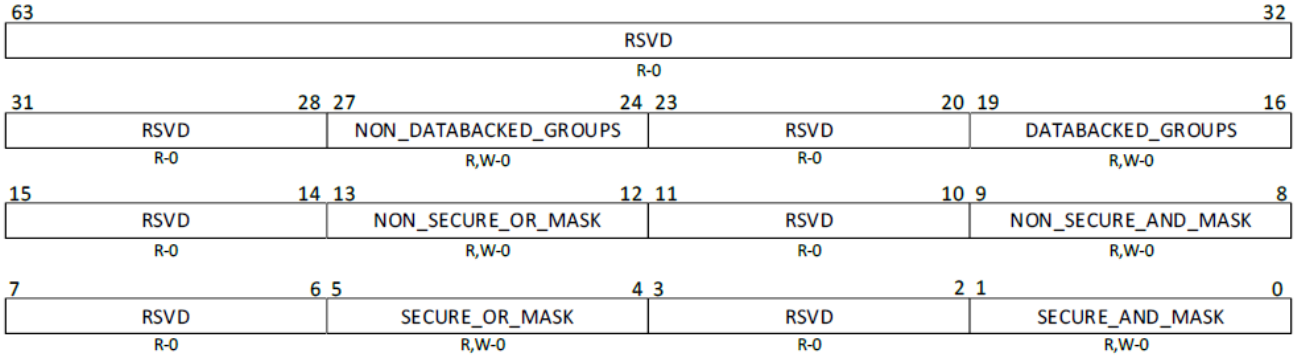


Figure 8-5. Way Group Partitioning

If DATABACKED_GROUPS is set to ‘0’ or is a value greater than the value in the CACHE_SIZE field within the CACHE_CTRL memory mapped register, then MSMC defers to the CACHE_SIZE setting to determine what way groups are available for any data-allocating transactions. Similarly, if NON_DATABACKED_GROUPS is set to ‘0’ or is a value greater than the value of (8-CACHE_SIZE), then MSMC defers to (8- CACHE_SIZE) when determining what way groups are available for any non-data-allocating transactions.

8.1.2.14.1.1 MMRs Related to Way Group Partitioning Feature

8.1.2.14.1.1.1 RT_WAY_SELECT [Address = 0x8000]

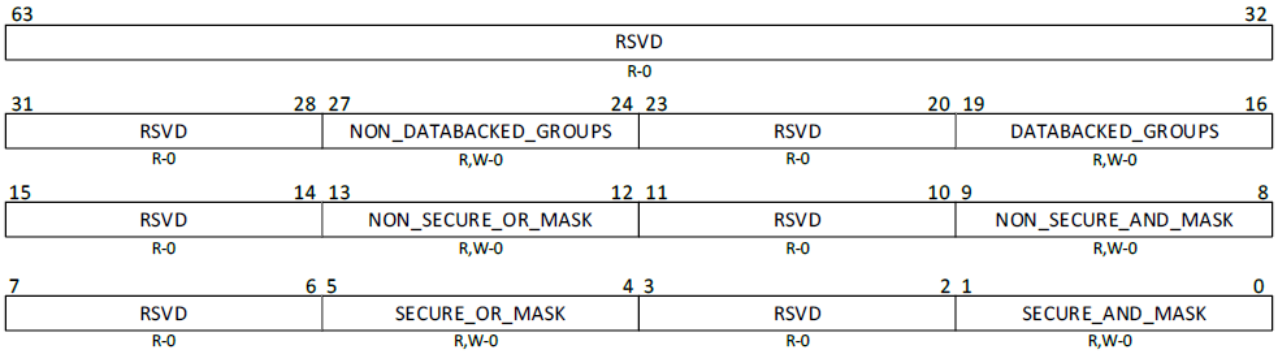


Bits	Name	Description
63:28	Reserved	Reserved
27:24	NON_DATABACKED_GROUPS	Numberof non-data backed way groups availableto RT allocations

Copyright © 2024 Texas Instruments Incorporated

Bits	Name	Description
23:20	Reserved	Reserved
19:16	DATABACKED_GROUPS	Number of data backed way groups available toRT allocations
15:14	Reserved	Reserved
13:12	NON_SECURE_OR_MASK	ORmask for way-select
11:10	Reserved	Reserved
9:8	NON_SECURE_AND_MASK	ANDmask for way-select
7:6	Reserved	Reserved
5:4	SECURE_OR_MASK	ORmask for way-select
3:2	Reserved	Reserved
1:0	SECURE_AND_MASK	ANDmask for way-select

8.1.2.14.1.1.2 NRT_WAY_SELECT [Address = 0x8008]



Bits	Name	Description
63:28	Reserved	Reserved
27:24	NON_DATABACKED_GROUPS	Number of non-data backed way groups availableto NRT allocations
23:20	Reserved	Reserved
19:16	DATABACKED_GROUPS	Number of data backed way groups available to NRTallocations
15:14	Reserved	Reserved
13:12	NON_SECURE_OR_MASK	ORmask for way-select
11:10	Reserved	Reserved
9:8	NON_SECURE_AND_MASK	ANDmask for way-select
7:6	Reserved	Reserved
5:4	SECURE_OR_MASK	ORmask for way-select
3:2	Reserved	Reserved
1:0	SECURE_AND_MASK	ANDmask for way-select

8.1.2.14.2 Write Back Invalidate

Through the WBINV_CTRL memory-mapped-register MSMC provides control for software to trigger a coherent invalidation of MSMC's external and/or internal SRAM snoop filter(s). If a '1' is written to the EMIF_SF_WBINV field in the register, then MSMC will invalidate the external snoop filter, writing back any dirty data to EMIF. If a '1' is written to the SRAM_SFINV field in the register, then MSMC will invalidate the internal SRAM snoop filter and write back any dirty data into its own SRAM. If both fields are written to simultaneously, then MSMC will invalidate the external snoop filter before invalidating the internal SRAM snoop filter immediately afterwards. As part of the write back invalidation procedure, MSMC will request all coherent masters to invalidate any lines that MSMC is tracking. The behavior tree is shown in the figure below.

Copyright © 2024 Texas Instruments Incorporated

When the write back invalidation of the external and/or internal SRAM SF is triggered theWBINV_ACTIVE field of the WBINV_CTRL memory mapped register will be set to ‘1’ until all invalidations are complete at which time it will be reset to ‘0’. Due to shared hardware betweensnoop filter invalidation and cache resizing for looping though all of the snoop filter entries, any write to EMIF_SF_WBINV, SRAM_SF_WBINV, or the CACHE_SIZE field within the CACHE_CTRL memory mapped register will be ignored if the WBINV_ACTIVE bit is ‘1’. Similarly, writes to EMIF_SF_WBINV and SRAM_SF_WBINV are ignored if a cache resize is in progress.

MSMC will maintain coherency for any transactions that arrive during a write back invalidation procedure, although the snoop filter state for any addresses they access is not ensured to be invalid until the operation is complete. The snoop filter state for any address accessed during the invalidation of the associated snoop filter is dependent on whether or not the access occurs before or after MSMC has performed its write back invalidation on that particular snoop filter entry.

Copyright © 2024 Texas Instruments Incorporated

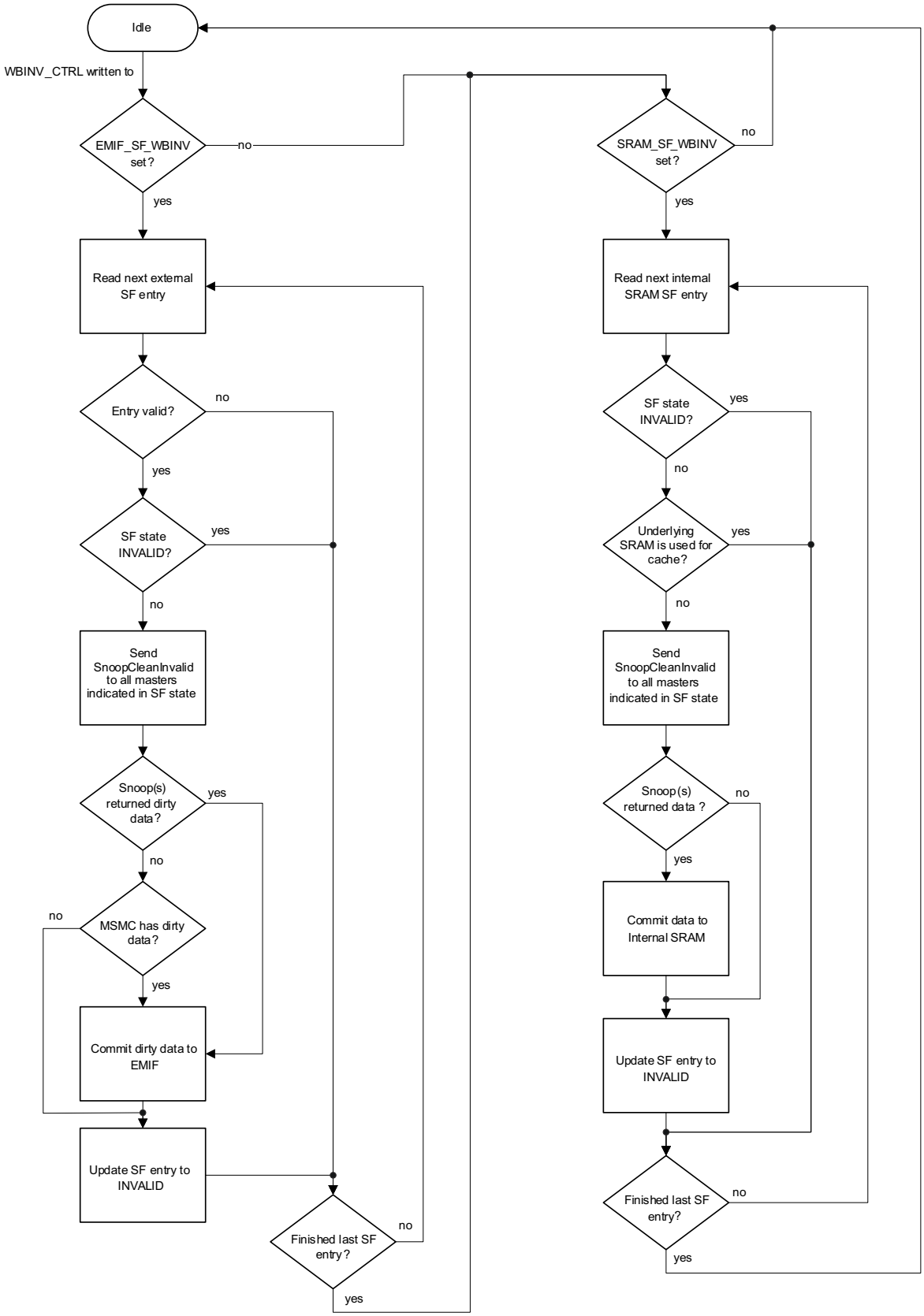
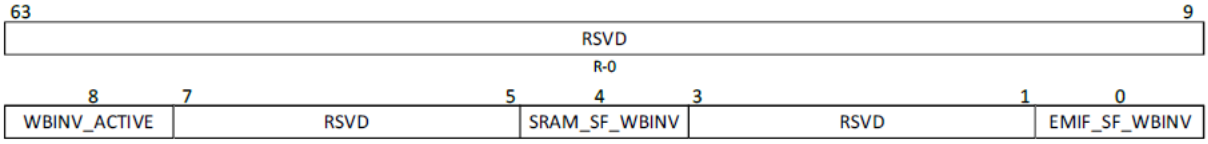


Figure 8-6. WBINV Flow Chart

8.1.2.14.2.1 MMR Related to Snoop Filter Invalidate Feature

Copyright © 2024 Texas Instruments Incorporated

8.1.2.14.2.1.1 WBINV_CTRL [Address = 0x4000]



Bits	Name	Description
63:9	Reserved	Reserved
8	WBINV_ACTIVE	WriteBack Invalidation in progress
7:5	Reserved	Reserved
4	SRAM_SF_WBINV	Trigger internal SRAM write back invalidation
3:1	Reserved	Reserved
0	EMIF_SF_WBINV	Trigger external memory write back invalidation

8.1.2.14.3 FFI Support

In the scenarios where ASIL-B endpoint is hung, MSMC supports unwinding of this situation by auto responding/ dropping pending commands. Below is the description of how MSMC handles different command types in FFI scenario. Note: For details on FFI sequence please refer Compute Cluster Spec.

1. CPU read/write: The CPU read/write requests to SRAM/EMIF and responses will be killed at the interface when FFI is active for the corresponding CPU.
2. SDMA and cache warm transactions from DRU: DRU (ASIL-B) can be reset if any of the ASIL-B CPUs are in FFI, since SDMA/cwarm transactions are not tracked in MSMC. The SDMA/cwarm requests and responses will be killed at the interface when FFI is active.
3. SDMA transactions from SoC ports: Traffic mastered by ASIL-D SoC ports and targeted to any ASIL- B CPU is not tracked in MSMC. If the slave CPU is in FFI, then the master SoC will not receive responses. In that case ASIL-D timeout gaskets will initiate recovery of ASIL-D SoC master.
4. Snoops: Snoop responses are faked in RMW queue so that the state machine is not stuck waiting for snoop responses.
5. DVM Sync-complete: DVM completes are faked in RMW queue so that the state machine is not stuck waiting for DVM completes.
6. CPU, SDMA,Snoop transactions: When in FFI, make sure local arb is free from partial transactions by reverting the local arb state machines back to reset/idle states when FFI is active.
7. Timeout gaskets: Timeout gaskets in MSMC track master timeout as well as slave far-side credit return timeout and generate timeout interrupts when either of them times out.

Timeout gaskets are located at CPU and DRU interfaces.

timeout_val[2:0](binary encoding)	timeout cycles (binary values)
"000"	"00000000000010000000000"
"001"	"00000000001000000000000"
"010"	"00000001000000000000000"
"011"	"00000100000000000000000"
"100"	"00010000000000000000000"
"101"	"01000000000000000000000"
"110"	"10000000000000000000000"
"111"	"11111111111111111111111"

8.1.2.14.3.1 FFI Event Sequence

The following is the sequence of events that FFI go through from failure to recovery:

Copyright © 2024 Texas Instruments Incorporated

1. CPU crash: One or more of the masters in ASIL-B domain (C7x/ARM/DRU) crashes. The crash is manifested as the master not being able to respond to commands. For example, CPU1 crashes and stops responding to commands.
2. Crash detection: The timeout gasket connected to the malfunctioning CPU1 generates timeout interrupt gskt_cpu1_if_timed_out_intr. For proper interrupt generation make sure gskt_cpu1_if_enable_timeout_detect is driven high and a valid timeout value on gskt_cpu1_if_timeout_val is driven as well. These MSMC-level tieoffs are driven by SoC level control registers. gskt_cpu1_if_timed_out_intr is sticky and needs to be cleared during recovery.
3. FFI req generation: When the SoC interrupt controller receives the timeout interrupt, gskt_cpu1_if_timed_out_intr, it will trigger the corresponding FFI req, ffi_cpu1_if_ffi_req.
4. MSMC FFI mode: MSMC receives ffi_cpu1_if_ffi_req and enters FFI mode where it will kill the commands from and responses to CPU1 at the interface. It will internally unwind snoop responses and credit returns that CPU1 is otherwise responsible for. During this time ASIL-D to ASIL-D traffic will be uninterrupted. For example, SoC to DDR or MSMC SRAM will proceed. Any access to CPU1 (SDMA) from other masters will not receive responses during this time. Since software is aware that CPU1 is FFI, it will stop sending new commands to CPU1. There is a scenario where ASIL-B master could have sent a command to CPU1 just before it entered FFI, that's the reason all ASIL-B masters are forced to FFI in a typical use case. If ASIL-D master sends the command to CPU1 just before it enters FFI, then the ASIL-D master's timeout gaskets will trigger unwinding of that ASIL-D master.
5. Power-down and Reset: FFI req is followed by the usual power-down and reset sequence. So, ffi_cpu1_if_ffi_req is followed by issuing cpu1_pwr_scr_disable_req and waiting for cpu1_pwr_scr_disable_ack. Once the ack is received CPU1 can be safely reset.
6. Recovery: When CPU1 is in reset, ffi_cpu1_if_ffi_req will be de-asserted , followed by reset de-assertion and cpu1_pwr_scr_disable_req de-assertion. Now, CPU1 can resume normal functional operation.

8.1.2.14.4 Broadcast Mode

This is not a new feature but a behavior update in existing feature. Previously, when broadcast mode is set (i.e. snoop filter is disabled) MSMC only sends broadcast snoop for cache hits. This new update keeps cache hit behavior as is; however, broadcast snoop is sent for cache misses as well. Note that all allocations are killed when broadcast mode is set in this new update.

Because there are no cache allocations in this mode, inclusivity is lost. Coherency is maintained as long as broadcast mode is set but dynamic change of broadcast mode from high to low (i.e. Broadcast mode enable to disable) is not supported.

System must invalidate L2 caches of all snoopable coherent masters connected to MSMC before they can disable Broadcast mode again after enabling it to maintain coherence.

This mode is expected to be used as a debug feature only, as performance is significantly impacted once this mode is enabled.

8.1.2.14.5 DRU and SDMA Access Constraints (Access ARC Removal)

The following SDMA access restrictions exist:

- ARM to DRU CFG accesses are restricted – This means:
  - ARMs cannot program DRU TRs or use DRU
  - ARMs cannot access CLEC
  - Cache warm operation for ARM l2 caches is not supported
- For all CPU coherent ports, CPU to CPU SDMA accesses are restricted
- SOC (non-LDP) port to DRU and CPU L2 SDMA accesses are restricted

This is a build time configuration, so restriction may be different for a different MSMC configuration.

8.1.2.14.6 EMIF Interleaving

There are two separate paths an EMIF access can take from the master to EMIF slave. 1.Through RMW queue, 2. Direct path (Bypass RMW queue). Coherent master accesses go through RMW queue path and non-coherent/SoC master accesses take a direct path to EMIF, bypassing RMW queue.

Copyright © 2024 Texas Instruments Incorporated

Table 8-9. Interleave Region Size

region_size[4:0]	Interleave Region Size
0	0MB (no interleave region)
1	128MB
2	256MB
3	512MB
4	1 GB
5	2 GB
6	3 GB
7	4 GB
8	6 GB
9	8 GB
10	12GB
11	16GB
12	32GB
13-31	Reserved

Table 8-10. Interleave Granule Size

granule_size[4:0]	Interleave Granule Size	Reason for Selection
0	128B	Highest performance for single master
1	512B	Highest performance for single master/ECC
2	2 KB	Pagesize interleaving
3	4 KB	Pagesize interleaving
4	16KB	Mostbank efficient for 2K size
5	32KB	Mostbank efficient for 4K size
6	512MB	
7	1 GB	Coarse memory interleaving
8	1.5GB	
9	2 GB	
10	3 GB	
11	4 GB	
12	6 GB	
13	8 GB	
14	16GB	Full memory interleaving
15-31	Reserved	

Table 8-11. emifs_active

emifs_active[3:0]	Description
0000	Reserved
0001	EMIF-0active
0010	EMIF-1active
0100	EMIF-2active
1000	EMIF-3active
0011	EMIF-0,1active
0101	EMIF-0,2active
1001	EMIF-0,3active
0110	EMIF-1,2active
1010	EMIF-1,3active
1100	EMIF-2,3active
0111	EMIF-0,1,2active

Copyright © 2024 Texas Instruments Incorporated

Table 8-11. emifs_active (continued)

emifs_active[3:0]	Description
1011	EMIF-0,1,3active
1101	EMIF-0,2,3active
1110	EMIF-1,2,3active
1111	EMIF-0,1,2,3active

Table 8-12. Hybrid_select

hybrid_select[4:0]	Description for 2 EMIFs	Description for 3 EMIFs.	Description for 4 EMIFs
00000	Entire space interleaved	Entire space interleaved	Entire space interleaved
00001	EMIF-0 separated,EMIFs-0,1 interleaved	EMIF-0 separated,EMIFs-1,2 interleaved	EMIF-0 separated,EMIF-1,2,3 interleaved
00010	EMIF-1 separated,EMIFs-0,1 interleaved	EMIF-1 separated,EMIFs-0,2 interleaved	EMIF-1 separated,EMIF-0,2,3 interleaved
00100	Reserved	EMIF-2 separated,EMIFs-0,1 interleaved	EMIF-2 separated,EMIF-0,1,3 interleaved
01000	Reserved	Reserved	EMIF-3 separated,EMIF-0,1,2 interleaved
10000	Reserved	Reserved	Reserved
10001	EMIF-0 separated,EMIFs-0,1 interleaved	EMIF-0 separated,EMIFs-0,1,2 interleaved	EMIF-0 separated,EMIF-0,1,2,3 interleaved
10010	EMIF-1 separated,EMIFs-0,1 interleaved	EMIF-1 separated,EMIFs-0,1,2 interleaved	EMIF-1 separated, EMIF-0,1,2,3interleaved
10100	Reserved	EMIF-2 separated,EMIFs-0,1,2 interleaved	EMIF-2 separated, EMIF-0,1,2,3interleaved
11000	Reserved	Reserved	EMIF-3 separated, EMIF-0,1,2,3interleaved

Table 8-13. ecc_enable

ecc_enable[3:0]	Description
0000	ECC disabled on all EMIFs
0001	ECC enabled on EMIF-0
0010	ECC enabled on EMIF-1
0100	ECC enabled on EMIF-2
1000	ECC enabled on EMIF-3
0011	ECC enabled on EMIF-0,1
0101	ECC enabled on EMIF-0,2
1001	ECC enabled on EMIF-0,3
0110	ECC enabled on EMIF-1,2
1010	ECC enabled on EMIF-1,3
1100	ECC enabled on EMIF-2,3
0111	ECC enabled on EMIF-0,1,2
1011	ECC enabled on EMIF-0,1,3
1101	ECC enabled on EMIF-0,2,3
1110	ECC enabled on EMIF-1,2,3
1111	ECC enabled on EMIF-0,1,2,3

When ECC is enabled, the corresponding DDR size will be effectively reduced. Max_memory=(ddr_ram_size/576)*512. The div-by-576 is an integer division. If the address falls out of max_memory range, the access will land in the next EMIF.

Programming Model for 2xEMIF case:

Copyright © 2024 Texas Instruments Incorporated

Case-1:DDR0 = DDR1 :2x {512M, 1G, 1.5G, 2G, 3G, 4G, 6G, 8G}

Separated DDRs:

region_size= DDR0+DDR1 sizes granule_size = DDR0 size emifs_active = both DDRs active

hybrid_select = Choose hybrid_select such that anything that is outside the region goes to the EMIF specified in hybrid_select. If its default(0), MSMC will route it out via its EMIF0 port.

ecc_enable= enabled/disabled on any DDR For example, if we have 2x1G DDRs then,

region_size= 2G, granule_size = 1G, emifs_active = 3, hybrid_select = x, ecc_enable = {0, 1, 2, 3}

Interleaved DDRs:

region_size= DDR0+DDR1 sizes granule_size < DDR0 size emifs_active = both DDRs active

hybrid_select = Choose hybrid_select such that anything that is outside the region goes to the EMIF specified in hybrid_select. If its default(0), MSMC will route it out via its EMIF0 port.

ecc_enable= enabled on both DDRs or disabled on both DDRs For example, if we have DDR0 = 1G and DDR1 = 1G then,

region_size= 2G, granule_size < 1G , emifs_active = 3, hybrid_select = x, ecc_enable = {0, 3}

Case-2:DDR0 < DDR1 :(512M,1G) ; (512M, 2G) ; (1G, 2G) ; (1G, 3G) ; (2G, 3G)

There are two ways to accomplish separated DDRs: case-A and case-B.Separated DDRs-A:

region_size = 2xDDR0 size granule_size = DDR0 size emifs_active. = both DDRs active hybrid_select. = DDR1

ecc_enable = enabled/disabled on any DDR

For example, if we have DDR0 =1G and DDR1 = 2G then,

region_size= 2G, granule_size = 1G, emifs_active = 3, hybrid_select = 2, ecc_enable = {0, 1, 2, 3}

Separated DDRs-B:

region_size = DDR0 size granule_size = DDR0 size emifs_active. = both DDRs active hybrid_select. = DDR1

ecc_enable = enabled/disabled on any DDR

For example, if we have DDR0 =1G and DDR1 = 2G then,

region_size= 1G, granule_size = 1G, emifs_active = 3, hybrid_select = 2, ecc_enable = {0, 1, 2, 3}

Interleaved DDRs:

region_size = 2xDDR0 size granule_size < DDR0 size emifs_active. = both DDRs active hybrid_select. = DDR1

ecc_enable = enabled on both DDRs or disabled on both DDRs. For example, if we have DDR0 =1G and DDR1 = 2G then,

region_size= 2G, granule_size < 1G, emifs_active = 3, hybrid_select = 2, ecc_enable = {0, 3}

Case-3:DDR0 > DDR1 : (1G,512M) ; (2G, 512M) ; (2G, 1G) ; (3G, 1G) ; (3G, 2G). *This case is not officially supported to have consistent system-level constraints for 2, 3, 4 EMIFs interleaving cases.

MSMC RTL supports it with the following programming model.Separated DDRs:

region_size = DDR0 size granule_size = DDR0 size emifs_active. = both DDRs active hybrid_select. = DDR1

ecc_enable = enabled/disabled on any DDR

For example, if we have DDR0 =2G and DDR1 = 1G then,

region_size= 2G, granule_size = 2G, emifs_active = 3, hybrid_select = 2, ecc_enable = {0, 1, 2, 3}

Interleaved DDRs:

region_size = 2xDDR1 size granule_size < DDR1 size emifs_active. = both DDRs active hybrid_select. = DDR0

Copyright © 2024 Texas Instruments Incorporated

ecc_enable = enabled on both DDRs or disabled on both DDRs. For example, if we have DDR0 =2G and DDR1 = 1G then,

region_size= 2G, granule_size < 1G, emifs_active = 3, hybrid_select = 1, ecc_enable = {0, 3}

8.1.2.14.7 QoS Fix/RT Hazarding

MSMC can send information to EMIF, about real time transactions that are currently stalled in MSMC so EMIF can drain the original transaction that is being hazarded upon. Once the transaction makes progress from EMIF, eventually the hazard is removed in MSMC.

Every command that is sent from MSMC to EMIF will have a queue ID encoded in its cmsband. If a command is stuck in EMIF, and there is another transaction in MSMC that is hazarded on the command that is stuck in EMIF, then MSMC sends the encoded queue ID of MSMC hazard to EMIF as and when the hazard is detected. EMIF compares the cmsband encoding of the originally stalled command in EMIF and the queue ID of hazard map encoding which it receives from MSMC when a hazard is detected. If EMIF finds a match between these two encoded queue IDs then it will trigger a drain such that the command originally stuck in EMIF makes progress and thus ultimately recovering MSMC from hazard.

For MSMC→EMIF commands that bypass the RMW queue, queue ID is meaningless so encode it in such a way that EMIF does not match any of the hazard queue ID encodings.

For EMIF compare MSBAND bits [8:5] for Queue ID/Transaction ID. Compare MSBAND bits [12:9]for Bank ID, if 12:9 = 0xF means it’s via EMIF bypass path and not coming from a bank.

Copyright © 2024 Texas Instruments Incorporated

8.2 DDR Subsystem (DDRSS)

This section describes the DDR Subsystem (DDRSS) for the device.

8.2.1 DDRSS Overview

The DDR subsystem in this device comprises DDR controller, DDR PHY and wrapper logic to integrate these blocks in the device. The DDR subsystem is referred to as DDRSS and is used to provide an interface to external SDRAM devices which can be utilized for storing program or data. DDRSS is accessed via MSMC, and not directly through the system interconnect.

The DDRSS supports:

- Memory Types:
  - LPDDR4
- Memory Bus Features:
  - 32-bit width with in-line ECC
  - Up to 2 ranks (in one package)
  - SDRAM address range up to 8 GB
- System Bus Interface:
  - 512-bit data width
  - Little endian only
  - Address aliasing prevention to block accesses to unpopulated SDRAM region
  - Clock asynchronous to DDR clock
- Configuration Bus Interface:
  - 32-bit data width
  - Linear incrementing addressing mode
  - 32-bit aligned accesses only
  - Little endian only
  - Clock asynchronous to DDR clock
- Key Features:
  - Full coherency across all commands
  - Bank interleaving
  - Priority based scheduling
  - Scheduling based on bank openness
  - Class of Service (CoS) - Three latency classes supported
  - Read/write scheduling to avoid turn-around time
  - Prioritized refresh scheduling
  - Dynamic change of refresh rate via software for extended temperatures
  - Statistical counters for performance management
- SDRAM ECC Features:
  - In-line ECC
  - Read-modify-write ECC for sub-word writes
  - ECC address error logging
  - Statistical counters for counting ECC errors
  - Injecting ECC errors during normal operation for validation
- Low Power Features:
  - Self-refresh entry and exit via software or hardware clock stop request
  - System bus clock stop via hardware clock stop request when controller is idle
  - Automatic idle power saving mode when no or low activity is detected
  - DDR and system bus clock frequency change using self-refresh via software or hardware clock stop request
  - Turning off SoC power after DDR is put into self-refresh (DDR reset and CKE I/O retention)
  - Tri-stating of all DDR I/O cells via software while driving CKE and RSTN pins during self-refresh
  - LPDDR4 Frequency Set Point (FSP)
  - I/O retention managed by an external device through dedicated pin
- Functional Reliability Features:

Copyright © 2024 Texas Instruments Incorporated

  - ECC on data pipe
  - Parity on address and command pipes
  - Data EDC on the master port interface
  - Command parity on the master port interface
  - Data EDC on the configuration port interface
  - Command parity on the configuration port interface
  - MSMC2DDR bridge AXI bus timeout
- DDR PHY Features:
  - Automatic and software controllable initialization and calibration (ZQ) for the DDR PHY and I/O cells
  - Automatic and software controllable delay line calibrations with Voltage and Temperature (VT) compensation
  - Automatic and software controllable write levelling with VT compensation
  - Automatic read DQS gate training per rank with VT compensation
  - Automatic and software controllable DQ/DQS eye training per rank
  - Automatic and software controllable read and write data bit deskew
  - Automatic and software controllable Command/Address (CA) levelling with VT compensation for LPDDR4
  - Automatic and software controllable CA bit deskew for LPDDR4
  - Refreshes to SDRAM during leveling and training
  - No seeding requirement based on board topology for any of the leveling and training algorithms
  - Dynamic/automatic I/O Receiver disable when read transfer is not on going
  - Capability of disabling data macros and I/O cells when not in use

Copyright © 2024 Texas Instruments Incorporated

8.2.2 DDRSS Environment

This section describes the DDRSS0 external connections (environment).

Copyright © 2024 Texas Instruments Incorporated

Table 8-14 describes the DDRSS0 I/O signals used for connection to SDRAM devices.

Table 8-14. DDRSS0 I/O signals

Module Pin	I/O[1]	Description
RSTN	I/O	SDRAM reset
CKE[1-0]	I/O	SDRAM CKE[1-0] signals
CK	I/O	SDRAM differential clock pair
CKN		
CSN0_0	I/O	SDRAM chip select 0 (two copies of CS0)[2]
CSN0_1		
CSN1_0	I/O	SDRAM chip select 1 (two copies of CS1)[2]
CSN1_1		
CA[5-0]	I/O	SDRAM address and command bus
DQS[3-0]	I/O	SDRAM data strobe
DQSN[3-0]	I/O	SDRAM data strobe invert
DQ[31-0]	I/O	SDRAM data bus
DM[3-0]	I/O	SDRAM data mask/DBI
RET	I/O	External I/O retention enable

(1) I = Input; O = Output
(2) Chip select for each channel is controlled independently during CA training, but they run in locked-step during normal operation.

Note

For more information about device level signals (pull-up/down resistors, buffer type, multiplexing and others), see tables *Pin Attributes* and *Pin Multiplexing* in the device-specific Datasheet.

Copyright © 2024 Texas Instruments Incorporated

8.2.3 DDRSS Functional Description

8.2.3.1 DDRSS MSMC2DDR Bridge

The MSMC2DDR bridge is part of the DDRSS and connects the DDR controller and MSMC data paths.

8.2.3.1.1 VBUSM.C Threads

Note

VBUS (VBUSM and VBUSP) is the device interconnect communication protocol. The letter "C" in "VBUSM.C" stands for Coherence Extension and denotes the interface is dedicated to COMPUTE_CLUSTER0. The VBUS protocol is beyond the scope of the device documentation as it is purely hardware oriented. For details about the device interconnect and the associated software controls, see *System Interconnect*.

The MSMC2DDR bridge supports two threads on the VBUSM.C interface. They are as follows:

- High Priority Thread (HPT) - traffic received on VBUSM.C thread 2 belongs to HPT.
- Low Priority Thread (LPT) - traffic received on VBUSM.C thread 2 belongs to LPT.

HPT has priority over LPT. Therefore, the execution of commands from the command FIFO can be out-of-order. As a result, the read data can also be returned out-of-order. This ensures that the HPT will be guaranteed execution even when LPT is fully blocked.

The MSMC2DDR bridge maintains data coherency across threads. Therefore, any HPT transactions that depend on the LPT transactions due to address conflicts are blocked until execution of the corresponding LPT transactions. This might result in a scenario where HPT can be blocked because of blockage on LPT. Once the LPT blockage has cleared, the blockage on the dependent HPT commands is also cleared.

HPT transactions can use global credits as well as thread 2 credits. LPT transactions can only use global credits.

On the command interface, the MSMC2DDR bridge gives back global credits on the VBUSM.C interface only if the available credits on the command queue in the DDR controller are greater than the DDRSS_V2A_CTL_REG[16-12] CRIT_THRESH field.

On the read data return interface, HPT traffic is prioritized over LPT traffic. The MSMC2DDR bridge sends HPT read data if there is at least 1 global or 1 thread 2 credit available.

The MSMC2DDR bridge sends LPT read data if there is no HPT data to send, and:

- There are at least 2 global credits available, or
- There is at least 1 global credit available and it is the last data phase

8.2.3.1.2 Class of Service (CoS)

Commands arriving on LPT and HPT to the DDRSS0 carry the VBUSM.C priority whereas the DDR controller uses another priority. The MSMC2DDR bridge has the following registers for flexible mapping of the VBUSM.C priority to DDR controllers's priority:

- Range match registers:
  - DDRSS_V2A_R1_MAT_REG
  - DDRSS_V2A_R2_MAT_REG
  - DDRSS_V2A_R3_MAT_REG
- Priority map registers:
  - DDRSS_V2A_LPT_DEF_PRI_MAP_REG
  - DDRSS_V2A_LPT_R1_PRI_MAP_REG
  - DDRSS_V2A_LPT_R2_PRI_MAP_REG
  - DDRSS_V2A_LPT_R3_PRI_MAP_REG
  - DDRSS_V2A_HPT_DEF_PRI_MAP_REG
  - DDRSS_V2A_HPT_R1_PRI_MAP_REG
  - DDRSS_V2A_HPT_R2_PRI_MAP_REG
  - DDRSS_V2A_HPT_R3_PRI_MAP_REG

Copyright © 2024 Texas Instruments Incorporated

This allows the system to essentially create different classes of service based on the initiator (Route ID) and the priority of the commands.

Each thread has a set of corresponding priority map registers to map the incoming priority on that thread to appropriate proirity for the DDR controller as shown in Figure 8-7. The default values of these registers are such that inherently the HPT traffic has higher priority than the LPT traffic inside the DDR controller, but these settings can be changed via software per system needs.

Note

For information about Route ID, see *Route ID* in *System Interconnect*.

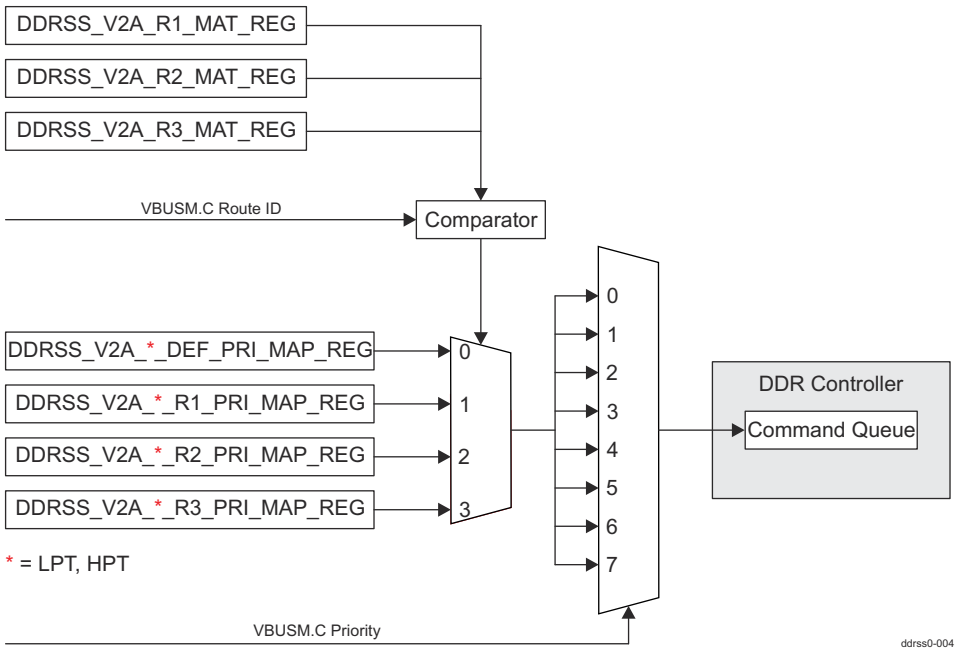


Figure 8-7. DDRSS CoS Mapping Diagram

8.2.3.1.3 AXI Write Data All-Strobes

The DDR controller data path is connected to the MSMC2DDR bridge through AXI interface. To enable higher performance and lower latency, the bridge drives the AXI AWALLSTRB signal to a "1" when the AXI write data do not have any byte enable holes. This allows the DDR controller to accept and start processing the write command as soon as possible without waiting for all data. The DDRSS_CTL_377[16] AXI0_ALL_STROBES_USED_ENABLE bit must be set to a 0x1 to utilize this feature. The controller ignores the AWALLSTRB signal if AXI0_ALL_STROBES_USED_ENABLE is set to 0x0.

8.2.3.1.4 Inline ECC for SDRAM Data

For SDRAM data integrity, the MSMC2DDR bridge supports inline ECC on the data written to or read from the SDRAM. ECC is enabled by programming the bits in the DDRSS_ECC_CTRL_REG register. ECC is stored together with the data so that a dedicated SDRAM device for ECC is not required. When using this inline ECC feature the inline ECC inside the DDR controller must be disabled by setting to 0x0 the DDRSS_CTL_206[17-16] ECC_ENABLE field.

8-bit single error correction double error detection (SECDED) ECC is calculated over 64-bit data quanta. For every 512-byte data block 64 bytes of ECC is stored inline. Thus 1/9th of the total SDRAM space is used for ECC storage and the rest 8/9th is available for system use. From system point of view that 8/9th of the SDRAM data space are seen as consecutive byte addresses. Even if there are non-ECC protected regions the previously described 1/9th-8/9th rule still applies and consecutive byte addresses are seen from system point of view.

Copyright © 2024 Texas Instruments Incorporated

The ECC is calculated for all accesses that are within the address ranges protected by ECC. The address ranges are specified through the following registers:

- DDRSS_ECC_R0_STR_ADDR_REG
- DDRSS_ECC_R0_END_ADDR_REG
- DDRSS_ECC_R1_STR_ADDR_REG
- DDRSS_ECC_R1_END_ADDR_REG
- DDRSS_ECC_R2_STR_ADDR_REG
- DDRSS_ECC_R2_END_ADDR_REG

The ECC is read and verified during reads if both the DDRSS_ECC_CTRL_REG[0] ECC_EN and DDRSS_ECC_CTRL_REG[2] ECC_CK bits are set to 0x1. For 1-bit ECC error, the MSMC2DDR bridge corrects the data and returns it to the requestor. Although the error is corrected on the returned data, the SDRAM is not corrected. It is responsibility of the system software to correct the ECC error at that location.

It is also responsibility of the system software to pre-load the ECC protected region with known data before functional reads and writes are performed. This can be done by writing to the SDRAM with ECC enabled (DDRSS_ECC_CTRL_REG[0] ECC_EN = 0x1 and DDRSS_ECC_CTRL_REG[1] RMW_EN = 0x1) and ECC check disabled (DDRSS_ECC_CTRL_REG[2] ECC_CK = 0x0). Once the data is loaded in the SDRAM, ECC check must be enabled (DDRSS_ECC_CTRL_REG[2] ECC_CK = 0x1) before using the DDR interface.

8.2.3.1.4.1 ECC Cache

The MSMC2DDR bridge implements a 128-line deep ECC cache for improving inline ECC performance. Each cache line can be allocated to an initiator in the system based on its Route ID. The Route ID allocation to cache line can be done by writing to the DDRSS_ECC_RID_INDX_REG and DDRSS_ECC_RID_VAL_REG registers. Since the bridge uses unallocated cache lines for all read accesses without Route ID allocation, one or more locations in the cache should be kept unallocated for better performance. To ensure that at least one cache line remains unallocated, in the case all cache lines are allocated by software, the bridge automatically unallocates the 127th cache line. Write accesses without Route ID allocation result in ECC and data writes to the SDRAM, when the DDRSS_ECC_CTRL_REG[4] WR_ALLOC bit is set to 0x0. When DDRSS_ECC_CTRL_REG[4] WR_ALLOC is set to 0x1, an unassigned cache-line is allocated to write accesses without Route ID allocation.

8.2.3.1.4.2 ECC Statistics

For 1-bit ECC error, the MSMC2DDR bridge logs the address location for the error in an internal 2-level deep FIFO. It stores the first two 1-bit ECC errors. The DDRSS_ECC_1B_ERR_ADR_LOG_REG register shows the address on top of the internal FIFO. Software should write 0x1 to the DDRSS_ECC_1B_ERR_ADR_LOG_REG[28-0] ECC_1B_ERR_ADR field to pop the FIFO and display the next address stored. The FIFO is loaded with the address for the next 1-bit ECC error if it is not full. If a single address is associated with more than one ECC error, that address is logged twice.

The number of 1-bit ECC errors can be counted using the DDRSS_ECC_1B_ERR_CNT_REG register. The MSMC2DDR bridge also supports programming a threshold in the DDRSS_ECC_1B_ERR_THRSH_REG register. When the 1-bit error count is equal to or greater than the programmed threshold, the bridge sets the DDRSS_V2A_INT_RAW_REG[3] ECC1BERR bit and also triggers the DDR0_DDRSS_DRAM_ECC_CORR_ERR_LVL_0 interrupt. When servicing the interrupt, software needs to clear the error count otherwise further interrupts will not be triggered. For 2-bit ECC errors, the bridge sets the DDRSS_V2A_INT_RAW_REG[4] ECC2BERR bit and also triggers the DDR0_DDRSS_DRAM_ECC_UNCORR_ERR_LVL_0 interrupt. The bridge does not correct the data for these uncorrectable errors. Along with generating the interrupt, the bridge also reports an error to the requesting initiatior and sends all zeros for the data. The bridge also logs the address location for the error in the DDRSS_ECC_2B_ERR_ADR_LOG_REG register.

The MSMC2DDR bridge sets the DDRSS_V2A_INT_RAW_REG[5] ECCM1BERR bit and triggers the DDR0_DDRSS_DRAM_ECC_UNCORR_ERR_LVL_0 interrupt whenever it receives multiple 1-bit errors in different data words of the same SDRAM burst. This is done because the probability of receiving multiple 1-bit errors in different data words of the same SDRAM burst is very low. If this occurs, the chances are that there were multi-bit errors in each data word. Therefore, for reliability, pessimistic approach is taken to report these as a fatal error. The threshold for the number of 1-bit errors that result in an uncorrected error, reporting can

Copyright © 2024 Texas Instruments Incorporated

be set by writing to the DDRSS_ECC_CTRL_REG[12-8] COR_ECC_THRESH field. For reliability, the threshold is always kept at 0/1 meaning 1-bit error in 2 or more data words is reported as a 2-bit error. For debug, the threshold can be changed to a higher value. Note that since these are multiple 1-bit errors, the statistic logging is done in the DDRSS_ECC_1B_ERR_CNT_REG and DDRSS_ECC_1B_ERR_ADR_LOG_REG registers.

8.2.3.1.5 Opcode Checking

The MSMC2DDR bridge checks the VBUSM.C opcode received for unsupported opcodes and flags error for debug purposes. Only opcode values for read and write operations are supported.

If an unsupported VBUSM.C opcode is received for an access, the bridge sets the DDRSS_V2A_INT_RAW_REG[0] OERR bit to 0x1 and triggers the DDR0_DDRSS_V2A_OTHER_ERR_LVL_0 interrupt. The opcode and the Route ID for the command caused error are logged in the DDRSS_V2A_OERR_LOG_REG[17-12] OERR_OP_CODE and DDRSS_V2A_OERR_LOG_REG[11-0] OERR_ROUTE_ID fields.

Since the bridge does not know how to process the command with an unsupported opcode, it discards the command and does not respond with any status on the write status or the read status buses.

8.2.3.1.6 Address Alias Prevention

The MSMC2DDR bridge checks the VBUSM.C address received against the valid SDRAM address space programmed in the DDRSS_V2A_CTL_REG register. If the VBUSM.C address for an access falls outside the programmed range the bridge sets to 0x1 the DDRSS_V2A_INT_RAW_REG[1] AERR bit and also triggers the DDR0_DDRSS_V2A_OTHER_ERR_LVL_0 interrupt. The address and the Route ID for the command caused error are logged in the DDRSS_V2A_AERR_LOG1_REG and DDRSS_V2A_AERR_LOG2_REG registers.

The valid address range can be programmed using the DDRSS_V2A_CTL_REG[9-5] SDRAM_IDX and DDRSS_V2A_CTL_REG[4-0] REGION_IDX fields. Table 8-15 summarizes the scenarios for determining valid address range and interrupt generation.

Table 8-15. REGION_IDX and SDRAM_IDX Scenarios

Condition	Description
REGION_IDX = SDRAM_IDX	DDR region size is equal to the connected SDRAM size. It is SoC responsibility to ensure that the addresses received by the DDR subsystem do not fall outside the region size. No address error is generated if the address received falls outside the region size.
REGION_IDX < SDRAM_IDX	DDR region size is less than the connected SDRAM size. It is SoC responsibility to ensure that the addresses received by the DDR subsystem do not fall outside the region size. No address error is generated if the address received falls outside the region size.
REGION_IDX > SDRAM_IDX	DDR region size is greater than the connected SDRAM size. Address error is generated if the address received falls outside the connected SDRAM size.

A write access outside the programmed range is discarded. A write error associated status is sent back to the VBUSM.C interface.

A read access outside the programmed range is executed on the SDRAM interface as a normal read, but data will contain all zeroes before forwarding it to the VBUSM.C interface. A read error associated status is sent back to the VBUSM.C interface along with the read data containing all zeroes.

When inline ECC is enabled, the available SDRAM size is reduced by 1/9th of the size programmed in the SDRAM_IDX field. Therefore, the bridge reports an error if an access falls outside the reduced SDRAM size when inline ECC is enabled. The reduced SDRAM size limit applies to all accesses, both to the protected and non-protected ECC regions.

8.2.3.1.7 Data Error Detection and Correction

The MSMC2DDR bridge performs data error detection and correction (EDC) on write data received. The EDC is performed over 256-bit data quanta and uses a Hamming code algorithm supporting single error correction and double error detection.

If the write data has a single bit error the bridge indicates a single bit error through the ECC aggregator interrupts. The write data will be corrected and written to the SDRAM. The address, the error position,

Copyright © 2024 Texas Instruments Incorporated

and the Route ID for the command caused error are logged in the DDRSS_V2A_1B_ERR_LOG1_REG and DDRSS_V2A_1B_ERR_LOG2_REG registers. The number of 1-bit EDC errors is kept in the DDRSS_V2A_1B_ERR_CNT_REG register.

If the write data has a double bit error the bridge indicates a double bit error through the ECC aggregator interrupts. The write will be executed on the SDRAM interface as a normal write and the data inside the SDRAM will be corrupted. The address and the route ID for the command caused the error are logged in the DDRSS_V2A_2B_ERR_LOG1_REG and DDRSS_V2A_2B_ERR_LOG2_REG registers. The bridge does not respond with a write error associated status for 2-bit EDC errors.

The bridge implements four EDC checkers that check EDC on each 256-bit quanta of a 1024-bit data word, in a single clock cycle. Therefore, when ECC errors occur on multiple 256-bit quanta in the same 1024-bit word, the error reported and logged is for the least significant 256-bits. If different quanta in the same 1024-bit word have single and double bit errors simultaneously, both errors are reported via interrupts but the address and Route ID associated with only the double bit error are logged.

The bridge generates an EDC code for the read data and sends it along with its read response for checking by the receiving system master.

8.2.3.1.8 AXI Bus Timeout

The MSMC2DDR bridge has a timeout counter that expires when no AXI transaction can be sent to the DDR controller or no AXI response is received from the controller for a programmed time interval. The counter only counts when there are pending commands in the bridge and the AXI bus is idle. Therefore, the bridge will not time out during low-power modes. The time interval can be programmed by writing to the DDRSS_V2A_BUS_TO[23-0] BUS_TIMER field.

Upon the expiry of the counter, the bridge terminates all pending commands in its internal FIFOs, returns error responses, sets the DDRSS_V2A_INT_RAW_REG[2] TOERR bit and triggers the DDR0_DDRSS_V2A_OTHER_ERR_LVL_0 interrupt.

After a timeout occurs, the bridge terminates any new commands received and returns error response. Writing a value of 0x0 to the DDRSS_V2A_BUS_TO[23-0] BUS_TIMER field exits the timeout mode. The other method to exit the timeout mode is to reset the DDRSS0, thus resetting the MSMC2DDR bridge, the DDR controller, and the DDR PHY.

8.2.3.2 DDRSS Interrupts

The DDRSS0 generates the following interrupts:

- DDR0_DDRSS_PLL_FREQ_CHANGE_REQ_0
- DDR controller interrupts:
  - DDR0_DDRSS_CONTROLLER_GLOBAL_ERROR_NONFATAL_0
  - DDR0_DDRSS_CONTROLLER_GLOBAL_ERROR_FATAL_0
  - DDR0_DDRSS_CONTROLLER_0
- DDR PHY interrupts:
  - DDR0_DDRSS_HS_PHY_GLOBAL_ERROR_0
- ECC aggregators interrupts:
  - DDR0_DDRSS_CTL_ECC_AGGR_CORR_ERR_LVL_0
  - DDR0_DDRSS_CTL_ECC_AGGR_UNCORR_ERR_LVL_0
  - DDR0_DDRSS_VBUS_ECC_AGGR_CORR_ERR_LVL_0
  - DDR0_DDRSS_VBUS_ECC_AGGR_UNCORR_ERR_LVL_0
  - DDR0_DDRSS_CFG_ECC_AGGR_CORR_ERR_LVL_0
  - DDR0_DDRSS_CFG_ECC_AGGR_UNCORR_ERR_LVL_0
- MSMC2DDR bridge interrupts:
  - DDR0_DDRSS_DRAM_ECC_CORR_ERR_LVL_0
  - DDR0_DDRSS_DRAM_ECC_UNCORR_ERR_LVL_0
  - DDR0_DDRSS_V2A_OTHER_ERR_LVL_0

The MSMC2DDR bridge sets to 0x1 the DDRSS_V2A_INT_RAW_REG[0] OERR bit, if the VBUSM.C access has an unsupported opcode.

Copyright © 2024 Texas Instruments Incorporated

The MSMC2DDR bridge sets to 0x1 the DDRSS_V2A_INT_RAW_REG[1] AERR bit, if the VBUSM.C address for an access that falls outside the DDR space programmed in the DDRSS_V2A_CTL_REG register.

The MSMC2DDR bridge sets to 0x1 the DDRSS_V2A_INT_RAW_REG[2] TOERR bit, if it detects a hang on its interface to the DDR controller.

The MSMC2DDR bridge sets to 0x1 the DDRSS_V2A_INT_RAW_REG[3] ECC1BERR bit, if the threshold for 1-bit ECC errors is met.

The MSMC2DDR bridge sets to 0x1 the DDRSS_V2A_INT_RAW_REG[4] ECC2BERR bit, in case of 2-bit errors for a read access performed within the SDRAM address range protected by ECC.

The DDRSS0 asserts particular interrupt line only if the interrupts are enabled by writing 0x1 to the corresponding bit in the DDRSS_V2A_INT_SET_REG register. The interrupts can be disabled by writing 0x1 to the corresponding bit in the DDRSS_V2A_INT_CLR_REG register.

When interrupts are enabled, the corresponding bits in the DDRSS_V2A_INT_STAT_REG register are also set if an interrupt condition occurs. The interrupts can be cleared once serviced by writing 0x1 to the corresponding bit in the DDRSS_V2A_INT_STAT_REG register as well as writing to the DDRSS_V2A_EOI_REG register.

Table 8-16 shows the events that are generated by the MSMC2DDR bridge.

Table 8-16. MSMC2DDR Bridge Events

Event Flag	Event Mask	Description
DDRSS_V2A_INT_RAW_REG[4] ECC2BERR DDRSS_V2A_INT_STAT_REG[4] ECC2BERR	DDRSS_V2A_INT_SET_REG[4] ECC2BERR_EN DDRSS_V2A_INT_CLR_REG[4] ECC2BERR_EN	Generated in case of 2-bit errors for a read access within the ECC protected SDRAM address range. For more information, see Section 8.2.3.1.4.2.
DDRSS_V2A_INT_RAW_REG[3] ECC1BERR DDRSS_V2A_INT_STAT_REG[3] ECC1BERR	DDRSS_V2A_INT_SET_REG[3] ECC1BERR_EN DDRSS_V2A_INT_CLR_REG[3] ECC1BERR_EN	Generated in case of meeting the threshold for 1-bit ECC errors. For more information, see Section 8.2.3.1.4.2.
DDRSS_V2A_INT_RAW_REG[2] TOERR DDRSS_V2A_INT_STAT_REG[2] TOERR	DDRSS_V2A_INT_SET_REG[2] TOERR_EN DDRSS_V2A_INT_CLR_REG[2] TOERR_EN	Generated in case of a hang of the interface between the MSMC2DDR bridge and the DDR controller. For more information, see Section 8.2.3.1.8.
DDRSS_V2A_INT_RAW_REG[1] AERR DDRSS_V2A_INT_STAT_REG[1] AERR	DDRSS_V2A_INT_SET_REG[1] AERR_EN DDRSS_V2A_INT_CLR_REG[1] AERR_EN	Generated if the VBUSM.C address for an access falls outside the programmed range. For more information, see Section 8.2.3.1.6.
DDRSS_V2A_INT_RAW_REG[0] OERR DDRSS_V2A_INT_STAT_REG[0] OERR	DDRSS_V2A_INT_SET_REG[0] OERR_EN DDRSS_V2A_INT_CLR_REG[0] OERR_EN	Generated if an unsupported VBUSM.C opcode is received for an access. For more information, see Section 8.2.3.1.5.

8.2.3.3 DDRSS Memory Regions

Table 8-17 shows all memory regions associated with the DDRSS0.

Table 8-17. DDRSS0 Memory Regions

Region	Start Address	End Address	Region Size
DDRSS0 wrapper logic registers	0x00 0298 0000	0x00 0298 01FF	512 B
DDR controller registers	0x00 0299 0000	0x00 0299 1FFF	8 KB
DDR PHY independent module registers	0x00 0299 2000	0x00 0299 3FFF	8 KB
DDR PHY registers	0x00 0299 4000	0x00 0299 7FFF	16 KB
DDRSS0_ECC_AGGR_CTL registers	0x4D 200B 0000	0x4D 200B 03FF	1 KB
DDRSS0_ECC_AGGR_VBUS registers	0x4D 200B 0400	0x4D 200B 07FF	1 KB

Copyright © 2024 Texas Instruments Incorporated

Table 8-17. DDRSS0 Memory Regions (continued)

Region	Start Address	End Address	Region Size
DDRSS0_ECC_AGGR_CFG registers	0x4D 200B 0800	0x4D 200B 0BFF	1 KB
External SDRAM data space - region 0 (32-bit low memory space)	0x00 8000 0000	0x00 FFFF FFFF	2 GB
External SDRAM data space - region 1 (high memory space)(1)	0x08 0000 0000	0x09 FFFF FFFF	8 GB

(1) The first 2 GB of this region are inaccessible.

8.2.3.4 DDRSS ECC Support

The DDRSS0 aligns to the device architecture for system reliability and uses the following ECC aggregators for detection and reporting of ECC errors:

- DDRSS0_ECC_AGGR_CTL
- DDRSS0_ECC_AGGR_VBUS
- DDRSS0_ECC_AGGR_CFG

For information about the ECC aggregator functionality, see *ECC Aggregator*.

8.2.3.5 DDRSS Dynamic Frequency Change Interface

This interface is used for handshaking between DDRSS0 and CTRL_MMR0 to dynamically change DDR clock frequency to support LPDDR4 Frequency Set Point (FSP). The frequency change can be initiated either by a SoC processor or by the DDRSS0.

The associated CTRL_MMR0 registers for processor initiated frequency change are:

- CTRLMMR_CHNG_DDR4_FSP_REQ
- CTRLMMR_CHNG_DDR4_FSP_ACK

The associated CTRL_MMR0 registers for DDRSS0 initiated frequency change are:

- CTRLMMR_DDR4_FSP_CLKCHNG_REQ
- CTRLMMR_DDR4_FSP_CLKCHNG_ACK

The sequence for an SoC processor initiated frequency change is as follows:

1. The processor writes the desired frequency to the CTRLMMR_CHNG_DDR4_FSP_REQ[1-0] REQ_TYPE field.
2. The processor sets to 0x1 the CTRLMMR_CHNG_DDR4_FSP_REQ[8] REQ bit to initiate frequency change.
3. The processor programs PLL12 according to the CTRLMMR_CHNG_DDR4_FSP_REQ[1-0] REQ_TYPE value.
4. The processor polls the CTRLMMR_CHNG_DDR4_FSP_ACK[7] ACK and CTRLMMR_CHNG_DDR4_FSP_ACK[0] ERROR bits to check if the frequency change has been completed successfully.

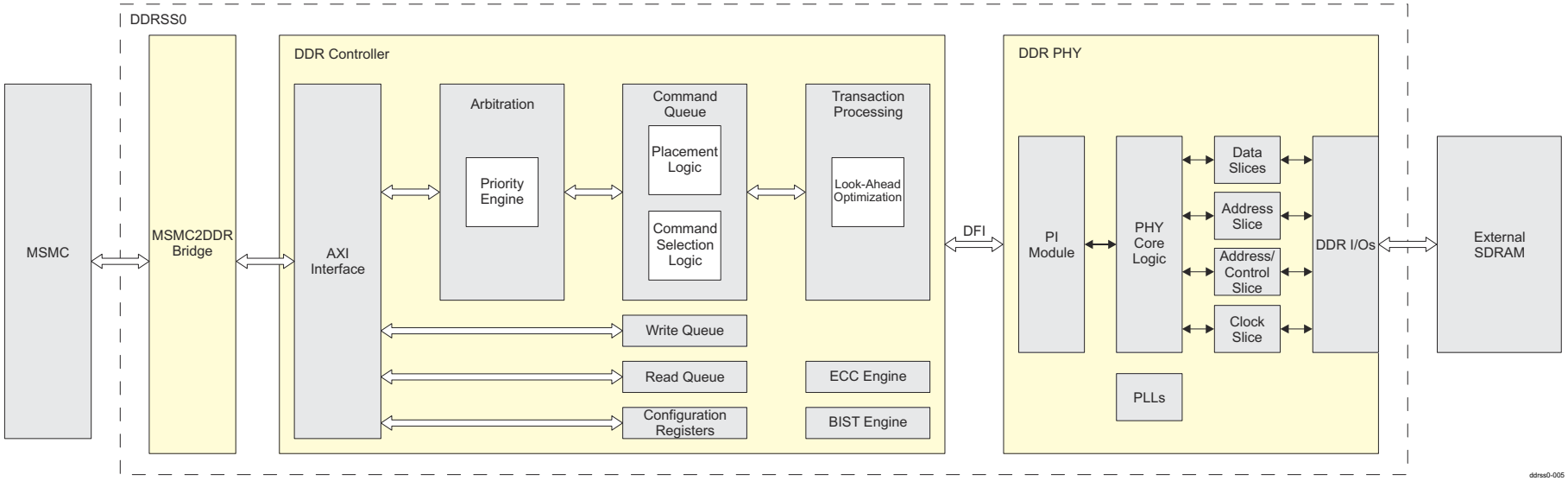
The sequence for DDRSS0 initiated frequency change is as follows:

1. DDRSS0 requests frequency change by asserting the CTRLMMR_DDR4_FSP_CLKCHNG_REQ[7] REQ bit and the CTRLMMR_DDR4_FSP_CLKCHNG_REQ[1-0] REQ_TYPE field. The REQ bit is also connected to the DDR0_DDRSS_PLL_FREQ_CHANGE_REQ_0 interrupt.
2. Software reads the CTRLMMR_DDR4_FSP_CLKCHNG_REQ[1-0] REQ_TYPE value and programs PLL12 accordingly.
3. After the DDRSS0_FCLK has been changed software should set to 0x1 the CTRLMMR_DDR4_FSP_CLKCHNG_ACK[0] ACK bit.

Copyright © 2024 Texas Instruments Incorporated

8.2.3.6 DDR Controller Functional Description

Figure 8-8 shows the DDR controller blocks along with the DDR PHY and MSMC2DDR bridge.



The ECC Engine block of the DDR controller is not supported.

BIST Engine is not supported.

Figure 8-8. DDR Controller Functional Blocks

Copyright © 2024 Texas Instruments Incorporated

8.2.3.6.1 DDR PHY Interface (DFI)

The DDR controller and DDR PHY are connected via DDR PHY Interface (DFI) which is used for transferring control information and data between the PHY and the controller. There are DFI programmable parameters which software can interact with through some of the DDR controller (see *DDR Controller Registers*), PI (see *PI Registers*) and DDR PHY (see *DDR PHY Registers*) registers. For more information about DFI, see http://www.ddr-phy.org/.

8.2.3.6.2 Command Queue

The DDR controller contains a command queue which uses a placement algorithm to determine the order of commands to be placed into the command queue. The placement logic follows many rules to determine where new commands should be inserted into the queue, relative to the contents of the command queue at a time. Placement is determined by considering address collisions, source collisions, data collisions, command types and priorities. The placement logic also attempts to maximize efficiency of the DDR controller through command grouping and bank splitting.

8.2.3.6.2.1 Placement Logic

The placement logic is a 2-stage queue which determines the order of commands that run in the DDR controller. The placement logic follows rules for determining placement of new commands into the queue, relative to the contents of the command queue at that time. Placement is determined by considering coherency, address collisions, source collisions, data collisions, user assigned priority, latency, age, and command type to offer low latency for critical masters while optimizing bandwidth for all masters. A second reordering stage allows ready-to-run commands to start even if the head-of-queue command is not yet ready to run. Some of the rules used in the placement logic can be individually enabled or disabled via the DDRSS_CTL_274 through DDRSS_CTL_277 registers. In addition, the queue can be disabled completely, resulting in an in-line queue that services requests in the order that they arrive.

The DDR controller also has a full look-ahead facility that reduces the effect of page misses by pre-conditioning rows for upcoming requests by using “spare” cycles in preceding transactions.

8.2.3.6.2.2 Command Selection Logic

After a command is in the command queue, command selection logic determines the method of pulling commands from the queue for running. On each clock cycle, the selection logic scans the entries of the command queue for determining the command to run. Commands for running are based on bank readiness, availability of at least 1 burst of data (writes), availability of storage for at least 1 burst of data (reads), bus turnaround timing, and conflicts. Similar to the placement rules, a command does not run before a command that was placed ahead of it in the command queue if it conflicts with address, source ID, or bank commands. Lower priority commands can run ahead of higher priority commands if the higher priority commands are not ready to run, as long as they do not conflict with commands that are ahead in the command queue.

8.2.3.6.3 Low Power Control

The DDR controller contains an arbitration block that has software programmable interface for low power control. Placement of and removal from the various memory low power modes is controlled through programming of registers in the DDR controller. The user can also monitor the status of the memory devices through a programmable register. This interface supports a lock option allowing software to execute additional commands through this interface without worrying about state changes through other interfaces.

8.2.3.6.4 Transaction Processing

The transaction command processing logic is used to process the commands in the command queue. The logic organizes the commands to the memories in such a way that data throughput is maximized. Bank opening and closing cycles are used for data transfers. The logic reviews the entire command queue for look-ahead of which banks have to be accessed in the future and ensures that the programed memory timing conditions are met. This flexibility allows the controller to be tuned to extract the maximum performance out of memories. During processing, the controller examines the commands and issues the appropriate set of signals to the memory.

Copyright © 2024 Texas Instruments Incorporated

8.2.3.6.5 BIST Engine

Note

BIST Engine is not supported.

The DDR controller has a BIST engine that supports MOVI3N and limited MOVI algorithms, a self-refresh retention test, an idle retention test and a memory initialization test for detecting the following faults:

- All address decoder faults
- All memory cell stuck-at faults
- All transition faults
- Most unlinked inversion coupling faults
- Most unlinked idem potent coupling faults
- Most idem potent coupling that are linked with inversion coupling faults

The DDR controller also supports BIST on ECC lanes for out-of-band ECC configuration. The following are the BIST related registers:

- DDRSS_CTL_194 through DDRSS_CTL_205
- DDRSS_CTL_302 through DDRSS_CTL_311

8.2.3.6.6 ECC Engine

Note

The ECC Engine block of the DDR controller is not supported.

The ECC engine can confirm the data accuracy and remove or identify bit errors if they occur. It checks for errors in both the data and the check code on all read transactions. The DDR controller can detect single-bit and double-bit data errors and can correct single-bit errors. The ECC engine supports interrupts, register storage of ECC error signatures, signaling of ECC errors, ECC scrubbing and write-back, automatic ECC corruption and ECC error forcing. The DDR controller supports inline ECC, which means a portion of the memory device connected to the controller is reserved for storing the ECC check codes and is not available for user data storage. ECC can be enabled or disabled through the DDRSS_CTL_206[17-16] ECC_ENABLE field. When enabled, all read data is checked for ECC (and optionally corrected) and ECC is computed and stored on all write data. Single-bit errors can be corrected and double-bit errors can be flagged. ECC information is not returned and ECC scrubbing is supported to maintain memory contents. The DDRSS_CTL_206 through DDRSS_CTL_227 registers contain the ECC related software controls.

8.2.3.6.7 Address Mapping

The DDR controller automatically maps SoC addresses to the SDRAM memory in a contiguous block. Addressing starts at address 0x0 and ends at the highest available address according to the size and number of SDRAM memories present. This mapping depends on the programmed values in the DDR controller registers and is based on the actual size of the available SDRAM memories. The size is configured via the DDRSS_CTL_286[2-0] MEMDATA_RATIO_0 and DDRSS_CTL_287[10-8] MEMDATA_RATIO_1 fields and that must be initialized at power-up.

8.2.3.6.8 Paging Policy

The DDR controller offers a flexible paging policy that allows open page operation, closed page operation, or an auto-precharge-per-command option that allows both modes simultaneously. Auto-precharge-per-command allows marking a particular master or transaction (for example, a CPU cache) as a closed-page transaction. This type of transaction reduces power and improves latency for the next transaction to a different row in the same SDRAM bank. Other transactions can be marked as open page transactions. This type of transaction reduces power and improves latency and bandwidth to the next transaction to the same row in the same SDRAM bank.

8.2.3.6.9 DDR Controller Initialization

Once the power to the SDRAM and SoC is stable, the DDR controller must be initialized. It then automatically initializes the external memory. The general steps to initialize the DDR controller are as follows:

1. Reset the DDRSS0 through the DDRSS0_RST signal. This resets all DDR controller registers.

Copyright © 2024 Texas Instruments Incorporated

2. Issue write register commands to configure the SDRAM protocols and the settings for the DDR controller. Keep the DDRSS_CTL_0[0] START bit to 0x0 during this initialization step.
3. Set to 0x1 the DDRSS_CTL_0[0] START bit. This triggers the controller to execute initialization sequence using the settings written to its registers. If the DDRSS_CTL_88[0] PWRUP_SREFRESH_EXIT bit is set to 0x0, the DDR controller issues MRWs and requests a ZQCL.

The DDR controller waits for the DDR PHY to assert the dfi_init_complete signal, which indicates that the PHY and SDRAM are ready to accept commands. When this signal asserts, the controller begins its initialization routine. The DDR controller sets to 0x1 bit [9] in the DDRSS_CTL_293[31-0] INT_STATUS_0 field when initialization is complete and it is ready to accept commands.

For information about the DDR PHY initialization, see Section 8.2.3.7.5.

8.2.3.6.10 Programming LPDDR4 Memories

LPDDR4 memories are specified to run at high frequency and include additional features such as CA training, write leveling, and ODT. At these high speeds, the DDR PHY also supports read data-eye training and gate training. Leveling and training operations are all performed through the PI module.

8.2.3.6.10.1 Frequency Set Point (FSP)

LPDDR4 memories use frequency set points to allow the memory to easily switch between two different frequencies without ever being in an un-trained state. There are two frequency set points - frequency set point 0 and frequency set point 1.

FSP is controlled through the following bits:

- DDRSS_CTL_192[0] DFS_ALWAYS_WRITE_FSP
- DDRSS_CTL_163[16] DISABLE_UPDATE_TVRCG
- DDRSS_CTL_192[16] FSP_OP_CURRENT
- DDRSS_CTL_191[24] FSP_PHY_UPDATE_MRW
- DDRSS_CTL_192[8] FSP_STATUS
- DDRSS_CTL_192[24] FSP_WR_CURRENT
- DDRSS_CTL_193[17-16] FSP0_FRC
- DDRSS_CTL_193[25-24] FSP1_FRC
- DDRSS_CTL_190[8] MR_FSP_DATA_VALID_F0
- DDRSS_CTL_190[16] MR_FSP_DATA_VALID_F1
- DDRSS_CTL_190[24] MR_FSP_DATA_VALID_F2
- DDRSS_CTL_166[4-0] TCKFSPE_F0
- DDRSS_CTL_168[4-0] TCKFSPE_F1
- DDRSS_CTL_171[4-0] TCKFSPE_F2
- DDRSS_CTL_166[12-8] TCKFSPX_F0
- DDRSS_CTL_168[28-24] TCKFSPX_F1
- DDRSS_CTL_171[12-8] TCKFSPX_F2
- DDRSS_CTL_166[31-16] TVREF_LONG_F0
- DDRSS_CTL_169[15-0] TVREF_LONG_F1
- DDRSS_CTL_171[31-16] TVREF_LONG_FN

8.2.3.6.10.1.1 FSP Mode Register Programming During Initialization

Before initialization, the user should program the FSP-related mode registers (MR1, MR2, MR3, MR11, MR12, MR13, MR14 and MR22) with the relevant values, and the DDRSS_CTL_21[28-24] DFIBUS_FREQ_F2, DDRSS_CTL_21[20-16] DFIBUS_FREQ_F1 and DDRSS_CTL_21[12-8] DFIBUS_FREQ_F0 fields with the frequency set to use. These mode registers are programmed through the following fields:

- DDRSS_CTL_175[7-0] MR1_DATA_F0_0
- DDRSS_CTL_175[23-16] MR1_DATA_F1_0
- DDRSS_CTL_176[7-0] MR1_DATA_F2_0
- DDRSS_CTL_182[23-16] MR1_DATA_F0_1
- DDRSS_CTL_183[7-0] MR1_DATA_F1_1
- DDRSS_CTL_183[23-16] MR1_DATA_F2_1

Copyright © 2024 Texas Instruments Incorporated

- DDRSS_CTL_175[15-8] MR2_DATA_F0_0
- DDRSS_CTL_175[31-24] MR2_DATA_F1_0
- DDRSS_CTL_176[15-8] MR2_DATA_F2_0
- DDRSS_CTL_182[31-24] MR2_DATA_F0_1
- DDRSS_CTL_183[15-8] MR2_DATA_F1_1
- DDRSS_CTL_183[31-24] MR2_DATA_F2_1
- DDRSS_CTL_176[31-24] MR3_DATA_F0_0
- DDRSS_CTL_177[7-0] MR3_DATA_F1_0
- DDRSS_CTL_177[15-8] MR3_DATA_F2_0
- DDRSS_CTL_184[15-8] MR3_DATA_F0_1
- DDRSS_CTL_184[23-16] MR3_DATA_F1_1
- DDRSS_CTL_184[31-24] MR3_DATA_F2_1
- DDRSS_CTL_178[23-16] MR11_DATA_F0_0
- DDRSS_CTL_178[31-24] MR11_DATA_F1_0
- DDRSS_CTL_179[7-0] MR11_DATA_F2_0
- DDRSS_CTL_186[7-0] MR11_DATA_F0_1
- DDRSS_CTL_186[15-8] MR11_DATA_F1_1
- DDRSS_CTL_186[23-16] MR11_DATA_F2_1
- DDRSS_CTL_179[15-8] MR12_DATA_F0_0
- DDRSS_CTL_179[23-16] MR12_DATA_F1_0
- DDRSS_CTL_179[31-24] MR12_DATA_F2_0
- DDRSS_CTL_186[31-24] MR12_DATA_F0_1
- DDRSS_CTL_187[7-0] MR12_DATA_F1_1
- DDRSS_CTL_187[15-8] MR12_DATA_F2_1
- DDRSS_CTL_180[7-0] MR13_DATA_0
- DDRSS_CTL_180[15-8] MR14_DATA_F0_0
- DDRSS_CTL_180[23-16] MR14_DATA_F1_0
- DDRSS_CTL_180[31-24] MR14_DATA_F2_0
- DDRSS_CTL_187[31-24] MR14_DATA_F0_1
- DDRSS_CTL_188[7-0] MR14_DATA_F1_1
- DDRSS_CTL_188[15-8] MR14_DATA_F2_1
- DDRSS_CTL_181[31-24] MR22_DATA_F0_0
- DDRSS_CTL_182[7-0] MR22_DATA_F1_0
- DDRSS_CTL_182[15-8] MR22_DATA_F2_0
- DDRSS_CTL_189[15-8] MR22_DATA_F0_1
- DDRSS_CTL_189[23-16] MR22_DATA_F1_1
- DDRSS_CTL_189[31-24] MR22_DATA_F2_1

In addition, the following fields must also be programmed before asserting the DDRSS_CTL_0[0] START bit:

- DDRSS_CTL_165[25-16] TFC_F0
- DDRSS_CTL_166[4-0] TCKFSPE_F0
- DDRSS_CTL_166[12-8] TCKFSPX_F0
- DDRSS_CTL_164[25-16] TVRCG_ENABLE_F0
- DDRSS_CTL_165[9-0] TVRCG_DISABLE_F0
- DDRSS_CTL_168[9-0] TFC_F1
- DDRSS_CTL_168[20-16] TCKFSPE_F1
- DDRSS_CTL_160[28-24] TCKFSPX_F1
- DDRSS_CTL_167[9-0] TVRCG_ENABLE_F1
- DDRSS_CTL_167[25-16] TVRCG_DISABLE_F1
- DDRSS_CTL_170[25-16] TFC_F2
- DDRSS_CTL_171[4-0] TCKFSPE_F2
- DDRSS_CTL_171[12-8] TCKFSPX_F2
- DDRSS_CTL_169[25-16] TVRCG_ENABLE_F2
- DDRSS_CTL_170[9-0] TVRCG_DISABLE_F2
- DDRSS_CTL_21[1-0] DFIBUS_BOOT_FREQ

Copyright © 2024 Texas Instruments Incorporated

- DDRSS_CTL_20[25-24] DFIBUS_FREQ_INIT
- DDRSS_CTL_149[24] DFS_ENABLE
- DDRSS_CTL_191[24] FSP_PHY_UPDATE_MRW

The following bits may or may not be modified, depending on what operations are needed after a frequency change occurs:

- DDRSS_CTL_192[0] DFS_ALWAYS_WRITE_FSP
- DDRSS_CTL_89[16] DFS_ZQ_EN

Once the DDRSS_CTL_0[0] START bit is asserted, the DDR controller programs the mode registers in the memory devices with these values using the FSP-OP and FSP-WR values from the DDRSS_CTL_180[7-0] MR13_DATA_0 field. Once initialization is complete, either the PHY, PI, or controller will have updated the frequency set points with the needed values.

8.2.3.6.10.1.2 FSP Mode Register Programming During Normal Operation

During normal operation, the DDR controller may modify the FSP mode register values in memory. To do this, the user should program the DDRSS_CTL_164[1-0] MRW_DFS_UPDATE_FRC field with the frequency set that the DDR controller is using and program the appropriate new values in the FSP-related mode registers previously listed. Since there are two copies of each of these mode registers in the memory device (other than MR13), the user should also identify the appropriate frequency set points for operation and writing in the DDRSS_CTL_192[16] FSP_OP_CURRENT and DDRSS_CTL_192[24] FSP_WR_CURRENT bits.

Generally, the user should not modify the mode registers for the frequency set being used, and therefore should program the DDRSS_CTL_192[24] FSP_WR_CURRENT with the opposite value of the DDRSS_CTL_192[16] FSP_OP_CURRENT bit.

Once established the values to write, the frequency set in the DDR controller to use, and the frequency set points, the user should program the DDRSS_CTL_159[26-0] WRITE_MODEREG field for the operation:

- Set bit [24] to 1h for all chips to be written with new values, or identify the target chip select in bits [15-8].
- Set bit [26] to 1h for a FSP-write operation. A write to this bit targets the mode registers MR1, MR2, MR3, MR11 and MR22 for writing. MR12 and MR14 are also written if the DDRSS_CTL_191[24] FSP_PHY_UPDATE_MRW bit is set to 0h.
- Set bit [25] to 1h to trigger the operation.

All of these bits may be programmed simultaneously.

Note

When the DDRSS_CTL_159[26-0] WRITE_MODEREG field is written with bit [26] set to 1h, the corresponding MR13 mode register is updated automatically with the value in the DDRSS_CTL_180[7-0] MR13_DATA_0 or DDRSS_CTL_187[23-16] MR13_DATA_1 fields, except that the value in the DDRSS_CTL_192[24] FSP_WR_CURRENT bit will define the value written in MR13 [6] and the value in the DDRSS_CTL_192[16] FSP_OP_CURRENT bit will define the value written in MR13 [7]. This write occurs before the other mode register writes, which means that the user may inadvertently write the wrong frequency set point mode or switch to the new frequency set point.

8.2.3.6.10.1.3 FSP Mode Register Programming During Dynamic Frequency Scaling

During a hardware DFS operation, the FSP mode registers are used to switch from one frequency set to the other. Before a hardware frequency change can be requested, the following must be programmed:

- DDRSS_CTL_193[17-16] FSP0_FRC
- DDRSS_CTL_193[25-24] FSP1_FRC
- DDRSS_CTL_192[24] FSP_WR_CURRENT
- DDRSS_CTL_192[16] FSP_OP_CURRENT
- DDRSS_CTL_190[24] MR_FSP_DATA_VALID_F2
- DDRSS_CTL_190[16] MR_FSP_DATA_VALID_F1
- DDRSS_CTL_190[8] MR_FSP_DATA_VALID_F0

Copyright © 2024 Texas Instruments Incorporated

FSP mode register writes are required as part of the DFS operation. To accomplish this, three mode register writes are issued to MR13, and additional mode register write to each of the FSP mode registers. These commands are issued at three times:

1. After frequency change entry and before the memory is placed into self refresh the first mode register write to MR13 is issued. This writes the value from the DDRSS_CTL_180[7-0] MR13_DATA_0 and DDRSS_CTL_187[23-16] MR13_DATA_1 fileds for all bits other than the FSP-WR bit which will be defined as opposite of the FSP-OP bit. Then mode register writes to all the other FSP mode registers are issued.
2. After the memory is in self-refresh, the MR13 is written again. This write causes the FSP-OP bit to toggle, and to set the VRCG bit. At this point, the frequency change should take place.
3. After the frequency change, one final mode register write to MR13 is performed to clear the VRCG bit. The DFS operation will update the FSP mode registers and the DDRSS_CTL_192[8] FSP_STATUS bit once it is complete.

8.2.3.6.10.2 Data Bus Inversion (DBI)

LPDDR4 memories support data bus inversion (DBI) for signal integrity and power savings. The DDR controller can support DBI on the read or write path, or on both. DBI functions by ensuring that in each data byte, there are no more than four '1' bits. This can be used to confirm data transmission, and also for power savings.

DBI is controlled through the following:

- The DDRSS_CTL_291[16] RD_DBI_EN bit
- The DDRSS_CTL_291[8] WR_DBI_EN bit

Before enabling controller-level data bus inversion, the DBI mode must be enabled in the memories through a write to bits [7-6] of MR3. The system behavior depends on the value of the memory mode register bits, the DDRSS_CTL_291[16] RD_DBI_EN or DDRSS_CTL_291[8] WR_DBI_EN bit, and the value of each byte of data.

If the DDRSS_CTL_291[8] WR_DBI_EN bit is set to 1h, the user should ensure that the memory mode register WR DBI bit is also set to 1h. When both of these bits are set, the DDR controller examines each byte of the incoming write data and identifies the number of '1' bits within each byte. If a byte of data has 5 or more '1' bits, the DDR controller automatically flips the data bits in that byte and drives the dfi_wrdata_mask/dfi_wrdata_mask_p1 signal that corresponds to that byte to '1'. If the number of ‘1’ bits within a byte is 4 or less or the DDRSS_CTL_291[8] WR_DBI_EN bit is set to 0h, then the data is sent to the DRAMs unchanged. On the DRAM, if the dfi_wrdata_mask/dfi_wrdata_mask_p1 signal is driven to '1', the DRAM re-inverts that byte before storing the data and if the dfi_wrdata_mask/dfi_wrdata_mask_p1 signal is set to '0', the DRAM stores the data as received.

For the read path, if the memory mode register RD DBI bit is set to 1h, the DDRSS_CTL_291[16] RD_DBI_EN bit should also be set to 1h. When both of these bits are set, the DDR controller examines the input signal dfi_rddata_dbi_n/dfi_rddata_dbi_n_w1. If the signal is driven to '1' for a byte, the DDR controller inverts the associated byte before sending it to the requestor. If the dfi_rddata_ dbi_n/dfi_rddata_dbi_n_w1 signal is held to '0' for a byte, the data is sent out to the requestor unchanged.

8.2.3.6.10.3 On-Die Termination

LPDDR4 memories provide two different types of on-die termination (ODT) - ODT for the DQ and ODT for the CA buses.

8.2.3.6.10.3.1 LPDDR4 DQ ODT

LPDDR4 supports termination for the DQ and DQS input signals, but this termination does not utilize an ODT input pin. Instead, the ODT is controlled by the command. The functionality is enabled through bits [2-0] of mode register MR11, where a value of 0h disables the ODT functionality. The DQ ODT settings can be programmed through the following fields:

- DDRSS_CTL_178[23-16] MR11_DATA_F0_0
- DDRSS_CTL_186[7-0] MR11_DATA_F0_1
- DDRSS_CTL_178[31-24] MR11_DATA_F1_0
- DDRSS_CTL_186[15-8] MR11_DATA_F1_1
- DDRSS_CTL_179[7-0] MR11_DATA_F2_0
- DDRSS_CTL_186[23-16] MR11_DATA_F2_1

Copyright © 2024 Texas Instruments Incorporated

These field values are written to the associated mode registers when the DDR controller issues a MRW to the memory.

When DQ ODT is enabled, RTT is asserted with a write or masked write command.

8.2.3.6.10.3.2 LPDDR4 CA ODT

LPDDR4 can terminate the CA bus, including the CS and CLK signals. The value on the ODT input pin, ODT_CA, on the DRAM and the settings in the mode registers determine if CA ODT is enabled in the memories and the RTT termination resistance. The functionality is enabled through bits [6-4] of mode register MR11, where the value of 0h disables ODT functionality. Bits [5-3] of MR22 control individual aspects of the CA ODT when ODT is not disabled. The CA ODT settings can be programmed through the following fields:

- DDRSS_CTL_178[23-16] MR11_DATA_F0_0
- DDRSS_CTL_178[31-24] MR11_DATA_F1_0
- DDRSS_CTL_179[7-0] MR11_DATA_F2_0
- DDRSS_CTL_186[7-0] MR11_DATA_F0_1
- DDRSS_CTL_186[15-8] MR11_DATA_F1_1
- DDRSS_CTL_186[23-16] MR11_DATA_F2_1
- DDRSS_CTL_181[31-24] MR22_DATA_F0_0
- DDRSS_CTL_182[7-0] MR22_DATA_F1_0
- DDRSS_CTL_182[15-8] MR22_DATA_F2_0
- DDRSS_CTL_189[15-8] MR22_DATA_F0_1
- DDRSS_CTL_189[23-16] MR22_DATA_F1_1
- DDRSS_CTL_189[31-24] MR22_DATA_F2_1

These field values are written to the associated mode registers when the DDR controller issues a MRW to the memory.

8.2.3.6.10.4 Byte Lane Swapping

The DDR controller supports byte lane swapping on the board or a DIMM for better timing or routing. In most cases, this swapping is invisible to the user as data byte lanes are swapped when data is being written, and then unswapped as data is read. However, mode register reads does not un-swap the data. Therefore, the user must be aware of the location of the 0th byte of the data in order to process the information from the mode registers correctly.

Whether byte lane swapping is used or not, the user must program the following fields:

- DDRSS_CTL_286[11-8] DEVICE0_BYTE0_CS0
- DDRSS_CTL_286[19-16] DEVICE1_BYTE0_CS0
- DDRSS_CTL_286[27-24] DEVICE2_BYTE0_CS0
- DDRSS_CTL_287[3-0] DEVICE3_BYTE0_CS0
- DDRSS_CTL_287[19-16] DEVICE0_BYTE0_CS1
- DDRSS_CTL_287[27-24] DEVICE1_BYTE0_CS1
- DDRSS_CTL_288[3-0] DEVICE2_BYTE0_CS1
- DDRSS_CTL_288[11-8] DEVICE3_BYTE0_CS1

Unique parameters are provided for each chip select to allow for different byte swapping on each chip select. Each field contains one bit for each byte of the memory data bus. If a device is not being used, the DEVICEx_BYTE0_CSy field should be programmed to 0h, but if the device is being used, then only one bit of the DEVICEx_BYTE0_CSy field should be set to 1h.

8.2.3.6.10.5 DQS Interval Oscillator

The DDR controller includes a feature to control the internal DQS clock tree oscillator in the LPDDR4 memories. This oscillator is used to track the delay variance of the DQS clock tree that will occur over time due to temperature and voltage. This feature allows the oscillator to be initiated and results read. If the result is outside of the programmed allowable variance, the DDR controller informs the PHY. The DDR controller does not perform any training functions as a part of or a result of this oscillator measurement - the PHY is expected to handle all training requirements.

The DQS oscillator is controlled through the following fields:

Copyright © 2024 Texas Instruments Incorporated

- DDRSS_CTL_26[16] DQS_OSC_ENABLE
- DDRSS_CTL_28[31-24] DQS_OSC_HIGH_THRESHOLD
- DDRSS_CTL_28[23-16] DQS_OSC_NORM_THRESHOLD
- DDRSS_CTL_27[14-0] DQS_OSC_PERIOD
- DDRSS_CTL_29[15-8] DQS_OSC_PROMOTE_THRESHOLD
- DDRSS_CTL_30[0] DQS_OSC_REQUEST
- DDRSS_CTL_29[7-0] DQS_OSC_TIMEOUT
- DDRSS_CTL_27[19-16] FUNC_VALID_CYCLES
- DDRSS_CTL_30[23-8] OSC_BASE_VALUE_0_CS0
- DDRSS_CTL_31[15-0] OSC_BASE_VALUE_1_CS0
- DDRSS_CTL_31[31-16] OSC_BASE_VALUE_2_CS0
- DDRSS_CTL_32[15-0] OSC_BASE_VALUE_3_CS0
- DDRSS_CTL_32[31-16] OSC_BASE_VALUE_0_CS1
- DDRSS_CTL_33[15-0] OSC_BASE_VALUE_1_CS1
- DDRSS_CTL_33[31-16] OSC_BASE_VALUE_2_CS1
- DDRSS_CTL_34[15-0] OSC_BASE_VALUE_3_CS1
- DDRSS_CTL_29[31-16] OSC_VARIANCE_LIMIT
- DDRSS_CTL_27[31-24] TOSCO_F0
- DDRSS_CTL_28[7-0] TOSCO_F1
- DDRSS_CTL_28[15-8] TOSCO_F2

Once the DDR controller has been initialized, before the controller reaches its terminal power-on state, a request is generated to the oscillator module to set the initial base values. These values are read from the memories and stored in the OSC_BASE_VALUE_x_CSy fields.

Following a frequency change, the base values to be used for comparison must be updated. As part of the operation, the frequency change task generates a subtask request to the DQS oscillator. This request, like the initialization request, will run the oscillator and then load the resulting values into the OSC_BASE_VALUE_x_CSy fields.

Once the base values have been updated, the DDR controller sets to 1h bit [31] in the DDRSS_CTL_293[31-0] INT_STATUS_0 field indicating DQS oscillator base value update interrupt.

8.2.3.6.10.5.1 Oscillator State Machine

A state machine inside the DDR controller tracks oscillator-related commands and wait times. The basic flow is as follows:

1. If the oscillator function is enabled by setting the DDRSS_CTL_26[16] DQS_OSC_ENABLE bit to 1h, the process begins when a request is received to run the oscillator function. This request can be issued automatically during the initialization process, during a frequency change process, as a software request by setting the DDRSS_CTL_30[0] DQS_OSC_REQUEST bit to 1h, or through a programmable periodic timer (the DDRSS_CTL_28[23-16] DQS_OSC_NORM_THRESHOLD, DDRSS_CTL_28[31-24] DQS_OSC_HIGH_THRESHOLD, or DDRSS_CTL_29[7-0] DQS_OSC_TIMEOUT fields). Once this process has started, the memory is not allowed to enter any low power modes until complete.
2. If the request is through initialization, MR23 is written with the value of the DDRSS_CTL_190[7-0] MR23_DATA field defining the number of cycles to run the oscillator.
3. An MPC is issued to start the oscillator. If there are multiple DRAM devices on a rank, the MPC is issued to all devices on the rank at the same time, and then the state machine proceeds to the next rank and issues MPCs to that rank, and so on.
4. The state machine waits the cycles defined by the DDRSS_CTL_27[14-0] DQS_OSC_PERIOD field and the cycles specified in the DDRSS_CTL_27[31-24] TOSCO_F0, DDRSS_CTL_28[7-0] TOSCO_F1, and DDRSS_CTL_28[15-8] TOSCO_F2 fields to allow the operation to complete and the results to be stored.
5. The state machine issues a read to MR18.
6. The state machine waits the number of cycles specified in the DDRSS_CTL_51[27-24] TMRR field, and then stores the result in a temporary location. If there are multiple DRAM devices on a rank, the MRR captures the mode register content for all devices on that rank.
7. The state machine issues a read to MR19.

Copyright © 2024 Texas Instruments Incorporated

8. The state machine waits the number of cycles specified in the DDRSS_CTL_51[27-24] TMRR field, and then stores the result in a temporary location. If there are multiple DRAM devices on a rank, the MRR captures the mode register content for all devices on that rank.
9. The values read in MR18 and MR19 represent the LSB and MSB of the DQS oscillator count. The DDR controller combines them and compares the result. If multiple devices exist on a rank, the comparison is made between each device and the base value.
   a. If this is the initial read or a read related to a DFS operation, the values are stored in the OSC_BASE_VALUE_x_CSy fields. The DDR controller drives a 1h value on the dfi_function signal and asserts the dfi_function_valid signal for the number of cycles defined by the DDRSS_CTL_27[19-16] FUNC_VALID_CYCLES field.
   b. If the value is FFFFh, an overflow condition occurred during the measurement process. The DDR controller sets to 1h bit [0] in the DDRSS_CTL_294[12-0] INT_STATUS_1 field indicating DQS oscillator measurement overflow interrupt and ignores the value.
   c. If the difference between the measured value and the base value is lower than or equal to the value programmed in the DDRSS_CTL_29[31-16] OSC_VARIANCE_LIMIT field, the DDR controller returns to idle state. If the request was issued through software, the DDR controller sets to 1h bit [30] in the DDRSS_CTL_293[31-0] INT_STATUS_0 field indicationg DQS oscillator request complete interrupt.
   d. If the difference between the measured value and the base value is greater than the value programmed in the DDRSS_CTL_29[31-16] OSC_VARIANCE_LIMIT field, the DDR controller sets to 1h bit [1] in the DDRSS_CTL_294[12-0] INT_STATUS_1 field indicating DQS oscillator out of variance interrupt. The DDR controller also drives a 2h value on the dfi_function signal and asserts the dfi_function_valid signal for the number of cycles defined by the DDRSS_CTL_27[19-16] FUNC_VALID_CYCLES field. The measured value will replace the base value for any device for which the measured value is outside the variance. This base value update does not set to 1h bit [31] in the DDRSS_CTL_293[31-0] INT_STATUS_0 field.

8.2.3.6.10.6 Per-Bank Refresh (PBR)

Refresh commands are important memory operations to preserve memory contents, but they are also disruptive to system transaction flow. To minimize the impact, the DDR controller implements refresh on a per-bank basis. This does increase the quantity of refresh commands, but allows banks that are not targeted by the refresh to be accessed for read and write commands.

PBR is controlled through the following fields:

- DDRSS_CTL_228[12-8] AREF_MAX_CREDIT
- DDRSS_CTL_228[4-0] AREF_MAX_DEFICIT
- DDRSS_CTL_227[20-16] AREF_NORM_THRESHOLD
- DDRSS_CTL_71[28-24] AREF_PBR_CONT_DIS_THRESHOLD
- DDRSS_CTL_71[20-16] AREF_PBR_CONT_EN_THRESHOLD
- DDRSS_CTL_71[3-0] PBR_BANK_SELECT_DELAY
- DDRSS_CTL_71[8] PBR_CONT_REQ_EN
- DDRSS_CTL_67[0] PBR_EN
- DDRSS_CTL_70[31-16] PBR_MAX_BANK_WAIT
- DDRSS_CTL_67[8] PBR_NUMERIC_ORDER
- DDRSS_CTL_59[16] TREF_ENABLE
- DDRSS_CTL_68[15-0] TREFI_PB_F0
- DDRSS_CTL_69[15-0] TREFI_PB_F1
- DDRSS_CTL_70[15-0] TREFI_PB_F2

8.2.3.6.10.6.1 Normal Operation

Per-bank refresh commands may be issued to all banks or to a single bank, and may be issued to each bank in any order if programmed for this behavior. However, all banks must be refreshed before the same bank can be issued a new per-bank refresh command. The DDR controller can issue the commands in any order, and subsequent per-bank refresh sequences can be a completely different order than the last command. The order of banks is specified via the DDRSS_CTL_67[8] PBR_NUMERIC_ORDER bit. It is potentially advantageous to execute out-of-numerical order to prevent an active bank from stalling refresh operations to any other banks.

Copyright © 2024 Texas Instruments Incorporated

The PBR logic is part of the auto-refresh logic. The auto-refresh logic is responsible for maintaining the debit / credit counts for refresh that ensure that refresh commands are occurring as needed. The DRAM allows the user to postpone a maximum of 8 refresh commands which means that the user can wait as much as 9 x TREFI_PB_Fx cycles before issuing a refresh command. The DDR controller keeps track of how many refreshes it has skipped (DDRSS_CTL_228[4-0] AREF_MAX_DEFICIT) or advanced (DDRSS_CTL_228[12-8] AREF_MAX_CREDIT) and ensures that the DRAM requirements are satisfied. The PBR logic issues refresh commands to single banks if enabled via the DDRSS_CTL_67[0] PBR_EN bit and in accordance with the programmed values. The DDRSS_CTL_59[16] TREF_ENABLE bit must also be set to 1h for the PBR logic to work. The auto-refresh logic issues refresh commands to all banks and high priority refresh commands if the refresh count falls below a certain threshold.

When a refresh is required, the PBR logic issues a per-bank refresh command to a specific bank. If that bank is currently being accessed by the DDR controller core, the command will be held off. If the command is held off beyond the number of cycles specified in the DDRSS_CTL_70[31-16] PBR_MAX_BANK_WAIT field, the PBR logic inhibits the active process in the DDR controller core, closes the bank, executes the PBR command and then un-inhibits the core to allow the interrupted process to continue. Once the bank is free, the PBR command is issued. The bank will be blocked from accesses within the number of clocks programmed into the DDRSS_CTL_71[3-0] PBR_BANK_SELECT_DELAY field and the refresh will occur and then release the bank.

8.2.3.6.10.6.2 Continuous Refresh Request Mode

The PBR logic can be temporarily modified to continuously assert refresh requests. This option may be desirable when the auto-refresh logic is running at a deficit and the user wishes to run per-bank refreshes instead of auto-refreshes to restore the refresh count.

Setting the DDRSS_CTL_71[8] PBR_CONT_REQ_EN bit to 1h enables this mode. Automatic mode falls between the normal priority and high priority commands in terms of priority. When enabled, if the number of all-bank refresh commands pending exceeds the value programmed into the DDRSS_CTL_71[20-16] AREF_PBR_CONT_EN_THRESHOLD field, then the DDR controller automatically begins issuing per-bank refresh commands. As refreshes occur, the pending command count drops. Once that value meets the value programmed in the DDRSS_CTL_71[28-24] AREF_PBR_CONT_DIS_THRESHOLD field, the DDR controller stops automatically issuing per-bank refresh commands.

If however the refresh command pending count exceeds the value programmed into the DDRSS_CTL_227[20-16] AREF_NORM_THRESHOLD field, the DDR controller temporarily suspends the PBR logic, and the auto-refresh logic issues all-bank refresh commands until the refresh logic catches up. Once caught up, the PBR logic is resumed.

8.2.3.7 DDR PHY Functional Description

The DDR PHY provides functionality to interface the DDR controller to SDRAM devices. The PHY has a slice based and DQS-delay architecture. It contains data, address, address/control and clock slices and uses programmable clock delay lines to align write data, read data capture, and DQS gating from the I/O pads across the DFI interface to the DDR controller.

8.2.3.7.1 Data Slice

The data slice transfers data between the DDR controller and the SDRAM devices. The data slice is a module that interfaces to the DQ, DM, and DQS signals of the SDRAM and is duplicated as many times as necessary to create the currnet SoC SDRAM data width.

The write data path logic (from DFI to pads) and the read data path logic (from pads to DFI) are contained within the data slice. Termination and directional controls for the data path related I/Os are also contained in this slice.

8.2.3.7.2 Address Slice

The address slice transfers address information between the DDR controller and the SDRAM devices and contains 6 address bits.

The address write data path logic (from DFI to pads) and the read data path logic (loopback only) are contained within the address slice. Termination and directional controls for the address path related I/Os also exist in this slice.

Copyright © 2024 Texas Instruments Incorporated

8.2.3.7.2.1 Address Swapping

The address slice has the ability to programmably change which DFI address bits that are connected to the I/O pads. Address signals can be swapped within an address slice without restrictions. The swapping is done via the DDRSS_PHY_1053[23-0] PHY_ADR_ADDR_SEL_0 field.

8.2.3.7.3 Address/Control Slice

The address/control slice is a module that interfaces to the control, command, and address connections of the SDRAM and is duplicated as many times as necessary to create the appropriate width for the SoC. The address/control signals within a slice have per bit delay line control.

8.2.3.7.4 Clock Slice

The clock slice creates the clock signal used by the SDRAM. The slice also disables the clock when instructed by the DDR controller through the DFI bus. When the clock is disabled, the CK pin drives low and the CKN drives high. Duty cycle correction is also applied in the memory clock slice.

8.2.3.7.5 DDR PHY Initialization

Once the power to the SDRAM and SoC is stable, the DDR PHY must be reset and initialized before normal traffic can commence. Once the initialization has completed, the DDR PHY asserts the dfi_init_complete signal to notify the DDR controller that the PHY and SDRAM are ready to accept commands.

The procedure to initialize the DDR PHY is as follows:

1. Reset the DDRSS0 through the DDRSS0_RST signal. This resets all DDR PHY registers.
2. Program all needed PHY registers.
3. The dfi_init_start is then asserted by the DDR controller. The PHY slice master delay lines locks and then the PHY asserts the dfi_init_complete signal. This indicates that the PHY and SDRAM are ready to accept commands. The PHY is now ready for normal traffic.

For information about the DDR controller initialization, see Section 8.2.3.6.9.

8.2.3.7.6 DDR PHY Dynamic Frequency Scaling (DFS)

The DDR PHY supports Dynamic Frequency Scaling (DFS) as defined by the DFI specification. This feature allows the PHY to be configured for multiple frequency sets, and have the DFI bus dynamically specify which frequency set to use at a given time. Before using a frequency set in normal operation, leveling operation should be performed on that set. Failure to do so may result in invalid operation. Once leveling has been performed with each frequency set, the user can freely switch between frequencies without need to repeat the leveling, unless the leveling environment requires periodic updates for voltage and temeperature compensation.

This DDR PHY contains registers for each frequency set. This feature creates a lookup table that may be accessed through the DDRSS_PHY_1281[17-16] PHY_FREQ_SEL_INDEX and DDRSS_PHY_1281[8] PHY_FREQ_SEL_MULTICAST_EN fields, which gives a flexibility in controlling these frequency set registers with a simple register interface. This fact should be taken into account when reading and writing these registers.

The DDRSS_PHY_1280[1-0] PHY_FREQ_SEL and DDRSS_PHY_1281[17-16] PHY_FREQ_SEL_INDEX fields operate independently, but if the PHY_FREQ_SEL_INDEX field specifies a frequency set different than the one currently being used, accidental write to or read from invalid registers may happen.

Reads from frequency based registers return only the value that correlates to the frequency set identified in the DDRSS_PHY_1281[17-16] PHY_FREQ_SEL_INDEX field. For writes to the frequency based registers, the resulting write depends on the value of the DDRSS_PHY_1281[8] PHY_FREQ_SEL_MULTICAST_EN bit. If it is set to 0x1, all frequency sets are concurrently written. If the PHY_FREQ_SEL_MULTICAST_EN bit is set to 0x0, only the frequency set identified in the PHY_FREQ_SEL_INDEX field is written.

For example, consider a write to DDRSS_PHY_130[9-0] PHY_RDDQS_DM_FALL_SLAVE_DELAY_0 with the following settings:

- DDRSS_PHY_1281[17-16] PHY_FREQ_SEL_INDEX = 0x0001 (frequency set 1)
- DDRSS_PHY_1281[8] PHY_FREQ_SEL_MULTICAST_EN = 0x0.

In this case, only the frequency set 1 version of the DDRSS_PHY_130[9-0] PHY_RDDQS_DM_FALL_SLAVE_DELAY_0 field is written. If the same settings are used,

Copyright © 2024 Texas Instruments Incorporated

but with PHY_FREQ_SEL_MULTICAST_EN = 0x1, then all frequency-indexed versions of PHY_RDDQS_DM_FALL_SLAVE_DELAY_0 are written.

8.2.3.7.7 Chip Select and Frequency Based Register Settings

Some register settings within the PHY are chip select based. The DDRSS_PHY_6[0] PHY_PER_CS_TRAINING_INDEX_0 bit allows access to a particular chip select copy. The PHY_PER_CS_TRAINING_INDEX_0 bit is associated with data slice 0. The following bits are associated with the other data slices:

- DDRSS_PHY_262[0] PHY_PER_CS_TRAINING_INDEX_1
- DDRSS_PHY_518[0] PHY_PER_CS_TRAINING_INDEX_2
- DDRSS_PHY_774[0] PHY_PER_CS_TRAINING_INDEX_3

When writing to chip select based registers, if the DDRSS_PHY_5[24] PHY_PER_CS_TRAINING_MULTICAST_EN_0 bit is set to 0x1, then all chip select copies receive the same data. The PHY_PER_CS_TRAINING_MULTICAST_EN_0 bit is associated with data slice 0. The following bits are associated with the other data slices:

- DDRSS_PHY_261[24] PHY_PER_CS_TRAINING_MULTICAST_EN_1
- DDRSS_PHY_517[24] PHY_PER_CS_TRAINING_MULTICAST_EN_2
- DDRSS_PHY_773[24] PHY_PER_CS_TRAINING_MULTICAST_EN_3

Some register settings in the PHY are frequency based. As already described, the DDRSS_PHY_1281[17-16] PHY_FREQ_SEL_INDEX field allows access to a particular frequency copy. When writing to frequency based registers, if the DDRSS_PHY_1281[8] PHY_FREQ_SEL_MULTICAST_EN bit is set to 0x1, then all frequency copies receive the same data. Table 8-18 shows the frequency based register settings.

Table 8-18. Frequency Based Register Settings

Register Settings	Category
Data Slice	
DDRSS_PHY_103[3-0] PHY_SW_MASTER_MODE_0 DDRSS_PHY_359[3-0] PHY_SW_MASTER_MODE_1 DDRSS_PHY_615[3-0] PHY_SW_MASTER_MODE_2 DDRSS_PHY_871[3-0] PHY_SW_MASTER_MODE_3	Master delay line
DDRSS_PHY_103[18-8] PHY_MASTER_DELAY_START_0 DDRSS_PHY_359[18-8] PHY_MASTER_DELAY_START_0 DDRSS_PHY_615[18-8] PHY_MASTER_DELAY_START_0 DDRSS_PHY_871[18-8] PHY_MASTER_DELAY_START_0	
DDRSS_PHY_103[29-24] PHY_MASTER_DELAY_STEP_0 DDRSS_PHY_359[29-24] PHY_MASTER_DELAY_STEP_0 DDRSS_PHY_615[29-24] PHY_MASTER_DELAY_STEP_0 DDRSS_PHY_871[29-24] PHY_MASTER_DELAY_STEP_0	
DDRSS_PHY_104[7-0] PHY_MASTER_DELAY_WAIT_0 DDRSS_PHY_360[7-0] PHY_MASTER_DELAY_WAIT_1 DDRSS_PHY_616[7-0] PHY_MASTER_DELAY_WAIT_2 DDRSS_PHY_872[7-0] PHY_MASTER_DELAY_WAIT_3	
DDRSS_PHY_104[15-8] PHY_MASTER_DELAY_HALF_MEASURE_0 DDRSS_PHY_360[15-8] PHY_MASTER_DELAY_HALF_MEASURE_1 DDRSS_PHY_616[15-8] PHY_MASTER_DELAY_HALF_MEASURE_2 DDRSS_PHY_872[15-8] PHY_MASTER_DELAY_HALF_MEASURE_3	

Copyright © 2024 Texas Instruments Incorporated

Table 8-18. Frequency Based Register Settings (continued)

Register Settings	Category
DDRSS_PHY_110[1-0] PHY_WRPATH_GATE_DISABLE_0 DDRSS_PHY_366[1-0] PHY_WRPATH_GATE_DISABLE_1 DDRSS_PHY_622[1-0] PHY_WRPATH_GATE_DISABLE_2 DDRSS_PHY_878[1-0] PHY_WRPATH_GATE_DISABLE_3	Write path
DDRSS_PHY_110[10-8] PHY_WRPATH_GATE_TIMING_0 DDRSS_PHY_366[10-8] PHY_WRPATH_GATE_TIMING_1 DDRSS_PHY_622[10-8] PHY_WRPATH_GATE_TIMING_2 DDRSS_PHY_878[10-8] PHY_WRPATH_GATE_TIMING_3	
DDRSS_PHY_139[9-8] PHY_DQ_FFE_0 DDRSS_PHY_395[9-8] PHY_DQ_FFE_1 DDRSS_PHY_651[9-8] PHY_DQ_FFE_2 DDRSS_PHY_907[9-8] PHY_DQ_FFE_3	
DDRSS_PHY_139[17-16] PHY_DQS_FFE_0 DDRSS_PHY_395[17-16] PHY_DQS_FFE_1 DDRSS_PHY_651[17-16] PHY_DQS_FFE_2 DDRSS_PHY_907[17-16] PHY_DQS_FFE_3	
DDRSS_PHY_104[31-24] PHY_WRLVL_DLY_STEP_0 DDRSS_PHY_360[31-24] PHY_WRLVL_DLY_STEP_1 DDRSS_PHY_616[31-24] PHY_WRLVL_DLY_STEP_2 DDRSS_PHY_872[31-24] PHY_WRLVL_DLY_STEP_3	Write leveling
DDRSS_PHY_105[3-0] PHY_WRLVL_DLY_FINE_STEP_0 DDRSS_PHY_361[3-0] PHY_WRLVL_DLY_FINE_STEP_1 DDRSS_PHY_617[3-0] PHY_WRLVL_DLY_FINE_STEP_2 DDRSS_PHY_873[3-0] PHY_WRLVL_DLY_FINE_STEP_3	
DDRSS_PHY_105[13-8] PHY_WRLVL_RESP_WAIT_CNT_0 DDRSS_PHY_361[13-8] PHY_WRLVL_RESP_WAIT_CNT_1 DDRSS_PHY_617[13-8] PHY_WRLVL_RESP_WAIT_CNT_2 DDRSS_PHY_873[13-8] PHY_WRLVL_RESP_WAIT_CNT_3	

Copyright © 2024 Texas Instruments Incorporated

Table 8-18. Frequency Based Register Settings (continued)

Register Settings	Category
DDRSS_PHY_107[7-0] PHY_WDQLVL_DLY_STEP_0 DDRSS_PHY_363[7-0] PHY_WDQLVL_DLY_STEP_1 DDRSS_PHY_619[7-0] PHY_WDQLVL_DLY_STEP_2 DDRSS_PHY_875[7-0] PHY_WDQLVL_DLY_STEP_3	Write DQ training
DDRSS_PHY_107[11-8] PHY_WDQLVL_QTR_DLY_STEP_0 DDRSS_PHY_363[11-8] PHY_WDQLVL_QTR_DLY_STEP_1 DDRSS_PHY_619[11-8] PHY_WDQLVL_QTR_DLY_STEP_2 DDRSS_PHY_875[11-8] PHY_WDQLVL_QTR_DLY_STEP_3	
DDRSS_PHY_86[16] PHY_NTP_TRAIN_EN_0 DDRSS_PHY_342[16] PHY_NTP_TRAIN_EN_1 DDRSS_PHY_598[16] PHY_NTP_TRAIN_EN_2 DDRSS_PHY_854[16] PHY_NTP_TRAIN_EN_3	
DDRSS_PHY_86[31-24] PHY_NTP_WDQ_STEP_SIZE_0 DDRSS_PHY_342[31-24] PHY_NTP_WDQ_STEP_SIZE_1 DDRSS_PHY_598[31-24] PHY_NTP_WDQ_STEP_SIZE_2 DDRSS_PHY_854[31-24] PHY_NTP_WDQ_STEP_SIZE_3	
DDRSS_PHY_87[10-0] PHY_NTP_WDQ_START_0 DDRSS_PHY_343[10-0] PHY_NTP_WDQ_START_1 DDRSS_PHY_599[10-0] PHY_NTP_WDQ_START_2 DDRSS_PHY_855[10-0] PHY_NTP_WDQ_START_3	
DDRSS_PHY_87[26-16] PHY_NTP_WDQ_STOP_0 DDRSS_PHY_343[26-16] PHY_NTP_WDQ_STOP_1 DDRSS_PHY_599[26-16] PHY_NTP_WDQ_STOP_2 DDRSS_PHY_855[26-16] PHY_NTP_WDQ_STOP_3	
DDRSS_PHY_88[17-8] PHY_WDQLVL_DVW_MIN_0 DDRSS_PHY_344[17-8] PHY_WDQLVL_DVW_MIN_1 DDRSS_PHY_600[17-8] PHY_WDQLVL_DVW_MIN_2 DDRSS_PHY_856[17-8] PHY_WDQLVL_DVW_MIN_3	
DDRSS_PHY_88[24] PHY_SW_WDQLVL_DVW_MIN_EN_0 DDRSS_PHY_344[24] PHY_SW_WDQLVL_DVW_MIN_EN_1 DDRSS_PHY_600[24] PHY_SW_WDQLVL_DVW_MIN_EN_2 DDRSS_PHY_856[24] PHY_SW_WDQLVL_DVW_MIN_EN_3	
DDRSS_PHY_89[5-0] PHY_WDQLVL_PER_START_OFFSET_0 DDRSS_PHY_345[5-0] PHY_WDQLVL_PER_START_OFFSET_1 DDRSS_PHY_601[5-0] PHY_WDQLVL_PER_START_OFFSET_2 DDRSS_PHY_857[5-0] PHY_WDQLVL_PER_START_OFFSET_3	
DDRSS_PHY_113[6-0] PHY_WDQ_OSC_DELTA_0 DDRSS_PHY_369[6-0] PHY_WDQ_OSC_DELTA_1 DDRSS_PHY_625[6-0] PHY_WDQ_OSC_DELTA_2 DDRSS_PHY_881[6-0] PHY_WDQ_OSC_DELTA_3	

Copyright © 2024 Texas Instruments Incorporated

Table 8-18. Frequency Based Register Settings (continued)

Register Settings	Category
DDRSS_PHY_136[15-8] PHY_DATA_DC_DQ0_CLK_ADJUST_0 DDRSS_PHY_136[23-16] PHY_DATA_DC_DQ1_CLK_ADJUST_0 DDRSS_PHY_136[31-24] PHY_DATA_DC_DQ2_CLK_ADJUST_0 DDRSS_PHY_137[7-0] PHY_DATA_DC_DQ3_CLK_ADJUST_0 DDRSS_PHY_137[15-8] PHY_DATA_DC_DQ4_CLK_ADJUST_0 DDRSS_PHY_137[23-16] PHY_DATA_DC_DQ5_CLK_ADJUST_0 DDRSS_PHY_137[31-24] PHY_DATA_DC_DQ6_CLK_ADJUST_0 DDRSS_PHY_138[7-0] PHY_DATA_DC_DQ7_CLK_ADJUST_0 DDRSS_PHY_392[15-8] PHY_DATA_DC_DQ0_CLK_ADJUST_1 DDRSS_PHY_392[23-16] PHY_DATA_DC_DQ1_CLK_ADJUST_1 DDRSS_PHY_392[31-24] PHY_DATA_DC_DQ2_CLK_ADJUST_1 DDRSS_PHY_393[7-0] PHY_DATA_DC_DQ3_CLK_ADJUST_1 DDRSS_PHY_393[15-8] PHY_DATA_DC_DQ4_CLK_ADJUST_1 DDRSS_PHY_393[23-16] PHY_DATA_DC_DQ5_CLK_ADJUST_1 DDRSS_PHY_393[31-24] PHY_DATA_DC_DQ6_CLK_ADJUST_1 DDRSS_PHY_394[7-0] PHY_DATA_DC_DQ7_CLK_ADJUST_1 DDRSS_PHY_648[15-8] PHY_DATA_DC_DQ0_CLK_ADJUST_2 DDRSS_PHY_648[23-16] PHY_DATA_DC_DQ1_CLK_ADJUST_2 DDRSS_PHY_648[31-24] PHY_DATA_DC_DQ2_CLK_ADJUST_2 DDRSS_PHY_649[7-0] PHY_DATA_DC_DQ3_CLK_ADJUST_2 DDRSS_PHY_649[15-8] PHY_DATA_DC_DQ4_CLK_ADJUST_2 DDRSS_PHY_649[23-16] PHY_DATA_DC_DQ5_CLK_ADJUST_2 DDRSS_PHY_649[31-24] PHY_DATA_DC_DQ6_CLK_ADJUST_2 DDRSS_PHY_650[7-0] PHY_DATA_DC_DQ7_CLK_ADJUST_2 DDRSS_PHY_904[15-8] PHY_DATA_DC_DQ0_CLK_ADJUST_3 DDRSS_PHY_904[23-16] PHY_DATA_DC_DQ1_CLK_ADJUST_3 DDRSS_PHY_904[31-24] PHY_DATA_DC_DQ2_CLK_ADJUST_3 DDRSS_PHY_905[7-0] PHY_DATA_DC_DQ3_CLK_ADJUST_3 DDRSS_PHY_905[15-8] PHY_DATA_DC_DQ4_CLK_ADJUST_3 DDRSS_PHY_905[23-16] PHY_DATA_DC_DQ5_CLK_ADJUST_3 DDRSS_PHY_905[31-24] PHY_DATA_DC_DQ6_CLK_ADJUST_3 DDRSS_PHY_906[7-0] PHY_DATA_DC_DQ7_CLK_ADJUST_3	Duty cycle correction

Copyright © 2024 Texas Instruments Incorporated

Table 8-18. Frequency Based Register Settings (continued)

Register Settings	Category
DDRSS_PHY_138[15-8] PHY_DATA_DC_DM_CLK_ADJUST_0 DDRSS_PHY_394[15-8] PHY_DATA_DC_DM_CLK_ADJUST_1 DDRSS_PHY_650[15-8] PHY_DATA_DC_DM_CLK_ADJUST_2 DDRSS_PHY_906[15-8] PHY_DATA_DC_DM_CLK_ADJUST_3	Duty cycle correction
DDRSS_PHY_136[7-0] PHY_DATA_DC_DQS_CLK_ADJUST_0 DDRSS_PHY_392[7-0] PHY_DATA_DC_DQS_CLK_ADJUST_1 DDRSS_PHY_648[7-0] PHY_DATA_DC_DQS_CLK_ADJUST_2 DDRSS_PHY_904[7-0] PHY_DATA_DC_DQS_CLK_ADJUST_3	
DDRSS_PHY_110[17-16] PHY_DATA_DC_INIT_DISABLE_0 DDRSS_PHY_366[17-16] PHY_DATA_DC_INIT_DISABLE_1 DDRSS_PHY_622[17-16] PHY_DATA_DC_INIT_DISABLE_2 DDRSS_PHY_878[17-16] PHY_DATA_DC_INIT_DISABLE_3	
DDRSS_PHY_112[0] PHY_DATA_DC_WRLVL_ENABLE_0 DDRSS_PHY_368[0] PHY_DATA_DC_WRLVL_ENABLE_1 DDRSS_PHY_624[0] PHY_DATA_DC_WRLVL_ENABLE_2 DDRSS_PHY_880[0] PHY_DATA_DC_WRLVL_ENABLE_3	
DDRSS_PHY_112[8] PHY_DATA_DC_WDQLVL_ENABLE_0 DDRSS_PHY_368[8] PHY_DATA_DC_WDQLVL_ENABLE_1 DDRSS_PHY_624[8] PHY_DATA_DC_WDQLVL_ENABLE_2 DDRSS_PHY_880[8] PHY_DATA_DC_WDQLVL_ENABLE_3	
DDRSS_PHY_111[26-16] PHY_DATA_DC_DQ_INIT_SLV_DELAY_0 DDRSS_PHY_367[26-16] PHY_DATA_DC_DQ_INIT_SLV_DELAY_1 DDRSS_PHY_623[26-16] PHY_DATA_DC_DQ_INIT_SLV_DELAY_2 DDRSS_PHY_879[26-16] PHY_DATA_DC_DQ_INIT_SLV_DELAY_3	
DDRSS_PHY_111[9-0] PHY_DATA_DC_DQS_INIT_SLV_DELAY_0 DDRSS_PHY_367[9-0] PHY_DATA_DC_DQS_INIT_SLV_DELAY_1 DDRSS_PHY_623[9-0] PHY_DATA_DC_DQS_INIT_SLV_DELAY_2 DDRSS_PHY_879[9-0] PHY_DATA_DC_DQS_INIT_SLV_DELAY_3	
DDRSS_PHY_97[18-16] PHY_DATA_DC_CAL_CLK_SEL_0 DDRSS_PHY_353[18-16] PHY_DATA_DC_CAL_CLK_SEL_1 DDRSS_PHY_609[18-16] PHY_DATA_DC_CAL_CLK_SEL_2 DDRSS_PHY_865[18-16] PHY_DATA_DC_CAL_CLK_SEL_3	
DDRSS_PHY_112[23-16] PHY_DATA_DC_DM_CLK_SE_THRSHLD_0 DDRSS_PHY_368[23-16] PHY_DATA_DC_DM_CLK_SE_THRSHLD_1 DDRSS_PHY_624[23-16] PHY_DATA_DC_DM_CLK_SE_THRSHLD_2 DDRSS_PHY_880[23-16] PHY_DATA_DC_DM_CLK_SE_THRSHLD_3	
DDRSS_PHY_112[31-24] PHY_DATA_DC_DM_CLK_DIFF_THRSHLD_0 DDRSS_PHY_368[31-24] PHY_DATA_DC_DM_CLK_DIFF_THRSHLD_1 DDRSS_PHY_624[31-24] PHY_DATA_DC_DM_CLK_DIFF_THRSHLD_2 DDRSS_PHY_880[31-24] PHY_DATA_DC_DM_CLK_DIFF_THRSHLD_3	

Copyright © 2024 Texas Instruments Incorporated

Table 8-18. Frequency Based Register Settings (continued)

Register Settings	Category
DDRSS_PHY_93[9-0] PHY_RDDQ0_SLAVE_DELAY_0 DDRSS_PHY_93[25-16] PHY_RDDQ1_SLAVE_DELAY_0 DDRSS_PHY_94[9-0] PHY_RDDQ2_SLAVE_DELAY_0 DDRSS_PHY_94[25-16] PHY_RDDQ3_SLAVE_DELAY_0 DDRSS_PHY_95[9-0] PHY_RDDQ4_SLAVE_DELAY_0 DDRSS_PHY_95[25-16] PHY_RDDQ5_SLAVE_DELAY_0 DDRSS_PHY_96[9-0] PHY_RDDQ6_SLAVE_DELAY_0 DDRSS_PHY_96[25-16] PHY_RDDQ7_SLAVE_DELAY_0 DDRSS_PHY_349[9-0] PHY_RDDQ0_SLAVE_DELAY_1 DDRSS_PHY_349[25-16] PHY_RDDQ1_SLAVE_DELAY_1 DDRSS_PHY_350[9-0] PHY_RDDQ2_SLAVE_DELAY_1 DDRSS_PHY_350[25-16] PHY_RDDQ3_SLAVE_DELAY_1 DDRSS_PHY_351[9-0] PHY_RDDQ4_SLAVE_DELAY_1 DDRSS_PHY_351[25-16] PHY_RDDQ5_SLAVE_DELAY_1 DDRSS_PHY_352[9-0] PHY_RDDQ6_SLAVE_DELAY_1 DDRSS_PHY_352[25-16] PHY_RDDQ7_SLAVE_DELAY_1 DDRSS_PHY_605[9-0] PHY_RDDQ0_SLAVE_DELAY_2 DDRSS_PHY_605[25-16] PHY_RDDQ1_SLAVE_DELAY_2 DDRSS_PHY_606[9-0] PHY_RDDQ2_SLAVE_DELAY_2 DDRSS_PHY_606[25-16] PHY_RDDQ3_SLAVE_DELAY_2 DDRSS_PHY_607[9-0] PHY_RDDQ4_SLAVE_DELAY_2 DDRSS_PHY_607[25-16] PHY_RDDQ5_SLAVE_DELAY_2 DDRSS_PHY_608[9-0] PHY_RDDQ6_SLAVE_DELAY_2 DDRSS_PHY_608[25-16] PHY_RDDQ7_SLAVE_DELAY_2 DDRSS_PHY_861[9-0] PHY_RDDQ0_SLAVE_DELAY_3 DDRSS_PHY_861[25-16] PHY_RDDQ1_SLAVE_DELAY_3 DDRSS_PHY_862[9-0] PHY_RDDQ2_SLAVE_DELAY_3 DDRSS_PHY_862[25-16] PHY_RDDQ3_SLAVE_DELAY_3 DDRSS_PHY_863[9-0] PHY_RDDQ4_SLAVE_DELAY_3 DDRSS_PHY_863[25-16] PHY_RDDQ5_SLAVE_DELAY DDRSS_PHY_864[9-0] PHY_RDDQ6_SLAVE_DELAY_3 DDRSS_PHY_864[25-16] PHY_RDDQ7_SLAVE_DELAY_3	Read path
DDRSS_PHY_97[9-0] PHY_RDDM_SLAVE_DELAY_0 DDRSS_PHY_353[9-0] PHY_RDDM_SLAVE_DELAY_1 DDRSS_PHY_609[9-0] PHY_RDDM_SLAVE_DELAY_2 DDRSS_PHY_865[9-0] PHY_RDDM_SLAVE_DELAY_3	Read path
DDRSS_PHY_101[25-24] PHY_RDDATA_EN_IE_DLY_0 DDRSS_PHY_357[25-24] PHY_RDDATA_EN_IE_DLY_1 DDRSS_PHY_613[25-24] PHY_RDDATA_EN_IE_DLY_2 DDRSS_PHY_869[25-24] PHY_RDDATA_EN_IE_DLY_3	
DDRSS_PHY_102[1-0] PHY_IE_MODE_0 DDRSS_PHY_358[1-0] PHY_IE_MODE_1 DDRSS_PHY_614[1-0] PHY_IE_MODE_2 DDRSS_PHY_870[1-0] PHY_IE_MODE_3	
DDRSS_PHY_102[20-16] PHY_RDDATA_EN_TSEL_DLY_0 DDRSS_PHY_358[20-16] PHY_RDDATA_EN_TSEL_DLY_1 DDRSS_PHY_614[20-16] PHY_RDDATA_EN_TSEL_DLY_2 DDRSS_PHY_870[20-16] PHY_RDDATA_EN_TSEL_DLY_3	
DDRSS_PHY_102[28-24] PHY_RDDATA_EN_OE_DLY_0 DDRSS_PHY_358[28-24] PHY_RDDATA_EN_OE_DLY_1 DDRSS_PHY_614[28-24] PHY_RDDATA_EN_OE_DLY_2 DDRSS_PHY_870[28-24] PHY_RDDATA_EN_OE_DLY_3	
DDRSS_PHY_113[20-16] PHY_RDDATA_EN_DLY_0 DDRSS_PHY_369[20-16] PHY_RDDATA_EN_DLY_1 DDRSS_PHY_625[20-16] PHY_RDDATA_EN_DLY_2 DDRSS_PHY_881[20-16] PHY_RDDATA_EN_DLY_3	
DDRSS_PHY_104[19-16] PHY_RPTR_UPDATE_0 DDRSS_PHY_360[19-16] PHY_RPTR_UPDATE_1 DDRSS_PHY_616[19-16] PHY_RPTR_UPDATE_2 DDRSS_PHY_872[19-16] PHY_RPTR_UPDATE_3	

Copyright © 2024 Texas Instruments Incorporated

Table 8-18. Frequency Based Register Settings (continued)

Register Settings	Category
DDRSS_PHY_105[19-16] PHY_GTLVL_DLY_STEP_0 DDRSS_PHY_361[19-16] PHY_GTLVL_DLY_STEP_1 DDRSS_PHY_617[19-16] PHY_GTLVL_DLY_STEP_2 DDRSS_PHY_873[19-16] PHY_GTLVL_DLY_STEP_3	Gate training
DDRSS_PHY_105[28-24] PHY_GTLVL_RESP_WAIT_CNT_0 DDRSS_PHY_361[28-24] PHY_GTLVL_RESP_WAIT_CNT_1 DDRSS_PHY_617[28-24] PHY_GTLVL_RESP_WAIT_CNT_2 DDRSS_PHY_873[28-24] PHY_GTLVL_RESP_WAIT_CNT_3	
DDRSS_PHY_106[9-0] PHY_GTLVL_BACK_STEP_0 DDRSS_PHY_362[9-0] PHY_GTLVL_BACK_STEP_1 DDRSS_PHY_618[9-0] PHY_GTLVL_BACK_STEP_2 DDRSS_PHY_874[9-0] PHY_GTLVL_BACK_STEP_3	
DDRSS_PHY_106[25-16] PHY_GTLVL_FINAL_STEP_0 DDRSS_PHY_362[25-16] PHY_GTLVL_FINAL_STEP_1 DDRSS_PHY_618[25-16] PHY_GTLVL_FINAL_STEP_2 DDRSS_PHY_874[25-16] PHY_GTLVL_FINAL_STEP_3	
DDRSS_PHY_107[27-24] PHY_RDLVL_DLY_STEP_0 DDRSS_PHY_363[27-24] PHY_RDLVL_DLY_STEP_1 DDRSS_PHY_619[27-24] PHY_RDLVL_DLY_STEP_2 DDRSS_PHY_875[27-24] PHY_RDLVL_DLY_STEP_3	Read data eye training
DDRSS_PHY_108[9-0] PHY_RDLVL_MAX_EDGE_0 DDRSS_PHY_364[9-0] PHY_RDLVL_MAX_EDGE_1 DDRSS_PHY_620[9-0] PHY_RDLVL_MAX_EDGE_2 DDRSS_PHY_876[9-0] PHY_RDLVL_MAX_EDGE_3	
DDRSS_PHY_109[9-0] PHY_RDLVL_DVW_MIN_0 DDRSS_PHY_365[9-0] PHY_RDLVL_DVW_MIN_1 DDRSS_PHY_621[9-0] PHY_RDLVL_DVW_MIN_2 DDRSS_PHY_877[9-0] PHY_RDLVL_DVW_MIN_3	
DDRSS_PHY_109[16] PHY_SW_RDLVL_DVW_MIN_EN_0 DDRSS_PHY_365[16] PHY_SW_RDLVL_DVW_MIN_EN_1 DDRSS_PHY_621[16] PHY_SW_RDLVL_DVW_MIN_EN_2 DDRSS_PHY_877[16] PHY_SW_RDLVL_DVW_MIN_EN_3	
DDRSS_PHY_109[29-24] PHY_RDLVL_PER_START_OFFSET_0 DDRSS_PHY_365[29-24] PHY_RDLVL_PER_START_OFFSET_1 DDRSS_PHY_621[29-24] PHY_RDLVL_PER_START_OFFSET_2 DDRSS_PHY_877[29-24] PHY_RDLVL_PER_START_OFFSET_3	
DDRSS_PHY_135[9-0] PHY_RDLVL_RDDQS_DQ_SLV_DLY_START_0 DDRSS_PHY_391[9-0] PHY_RDLVL_RDDQS_DQ_SLV_DLY_START_1 DDRSS_PHY_647[9-0] PHY_RDLVL_RDDQS_DQ_SLV_DLY_START_2 DDRSS_PHY_903[9-0] PHY_RDLVL_RDDQS_DQ_SLV_DLY_START_3	
DDRSS_PHY_85[30-24] PHY_VREF_INITIAL_START_POINT_0 DDRSS_PHY_341[30-24] PHY_VREF_INITIAL_START_POINT_1 DDRSS_PHY_597[30-24] PHY_VREF_INITIAL_START_POINT_2 DDRSS_PHY_853[30-24] PHY_VREF_INITIAL_START_POINT_3	
DDRSS_PHY_86[6-0] PHY_VREF_INITIAL_STOP_POINT_0 DDRSS_PHY_342[6-0] PHY_VREF_INITIAL_STOP_POINT_1 DDRSS_PHY_598[6-0] PHY_VREF_INITIAL_STOP_POINT_2 DDRSS_PHY_854[6-0] PHY_VREF_INITIAL_STOP_POINT_3	
DDRSS_PHY_100[27-16] PHY_PAD_VREF_CTRL_DQ_0 DDRSS_PHY_356[27-16] PHY_PAD_VREF_CTRL_DQ_1 DDRSS_PHY_612[27-16] PHY_PAD_VREF_CTRL_DQ_2 DDRSS_PHY_868[27-16] PHY_PAD_VREF_CTRL_DQ_3	

Copyright © 2024 Texas Instruments Incorporated

Table 8-18. Frequency Based Register Settings (continued)

Register Settings	Category
DDRSS_PHY_98[7-0] PHY_DQ_OE_TIMING_0 DDRSS_PHY_354[7-0] PHY_DQ_OE_TIMING_1 DDRSS_PHY_610[7-0] PHY_DQ_OE_TIMING_2 DDRSS_PHY_866[7-0] PHY_DQ_OE_TIMING_3	Timing
DDRSS_PHY_98[15-8] PHY_DQ_TSEL_RD_TIMING_0 DDRSS_PHY_354[15-8] PHY_DQ_TSEL_RD_TIMING_1 DDRSS_PHY_610[15-8] PHY_DQ_TSEL_RD_TIMING_2 DDRSS_PHY_866[15-8] PHY_DQ_TSEL_RD_TIMING_3	
DDRSS_PHY_98[23-16] PHY_DQ_TSEL_WR_TIMING_0 DDRSS_PHY_354[23-16] PHY_DQ_TSEL_WR_TIMING_1 DDRSS_PHY_610[23-16] PHY_DQ_TSEL_WR_TIMING_2 DDRSS_PHY_866[23-16] PHY_DQ_TSEL_WR_TIMING_3	
DDRSS_PHY_98[31-24] PHY_DQS_OE_TIMING_0 DDRSS_PHY_354[31-24] PHY_DQS_OE_TIMING_1 DDRSS_PHY_610[31-24] PHY_DQS_OE_TIMING_2 DDRSS_PHY_866[31-24] PHY_DQS_OE_TIMING_3	
DDRSS_PHY_99[15-8] PHY_DQS_TSEL_RD_TIMING_0 DDRSS_PHY_355[15-8] PHY_DQS_TSEL_RD_TIMING_1 DDRSS_PHY_611[15-8] PHY_DQS_TSEL_RD_TIMING_2 DDRSS_PHY_867[15-8] PHY_DQS_TSEL_RD_TIMING_3	
DDRSS_PHY_99[31-24] PHY_DQS_TSEL_WR_TIMING_0 DDRSS_PHY_355[31-24] PHY_DQS_TSEL_WR_TIMING_1 DDRSS_PHY_611[31-24] PHY_DQS_TSEL_WR_TIMING_2 DDRSS_PHY_867[31-24] PHY_DQS_TSEL_WR_TIMING_3	
DDRSS_PHY_84[2-0] PHY_DQ_TSEL_ENABLE_0 DDRSS_PHY_340[2-0] PHY_DQ_TSEL_ENABLE_1 DDRSS_PHY_596[2-0] PHY_DQ_TSEL_ENABLE_2 DDRSS_PHY_852[2-0] PHY_DQ_TSEL_ENABLE_3	
DDRSS_PHY_84[23-8] PHY_DQ_TSEL_SELECT_0 DDRSS_PHY_340[23-8] PHY_DQ_TSEL_SELECT_1 DDRSS_PHY_596[23-8] PHY_DQ_TSEL_SELECT_2 DDRSS_PHY_852[23-8] PHY_DQ_TSEL_SELECT_3	
DDRSS_PHY_84[26-24] PHY_DQS_TSEL_ENABLE_0 DDRSS_PHY_340[26-24] PHY_DQS_TSEL_ENABLE_1 DDRSS_PHY_596[26-24] PHY_DQS_TSEL_ENABLE_2 DDRSS_PHY_852[26-24] PHY_DQS_TSEL_ENABLE_3	
DDRSS_PHY_85[15-0] PHY_DQS_TSEL_SELECT_0 DDRSS_PHY_341[15-0] PHY_DQS_TSEL_SELECT_1 DDRSS_PHY_597[15-0] PHY_DQS_TSEL_SELECT_2 DDRSS_PHY_853[15-0] PHY_DQS_TSEL_SELECT_3	
DDRSS_PHY_99[23-16] PHY_DQS_OE_RD_TIMING_0 DDRSS_PHY_355[23-16] PHY_DQS_OE_RD_TIMING_1 DDRSS_PHY_611[23-16] PHY_DQS_OE_RD_TIMING_2 DDRSS_PHY_867[23-16] PHY_DQS_OE_RD_TIMING_3	
DDRSS_PHY_101[15-8] PHY_DQ_IE_TIMING_0 DDRSS_PHY_357[15-8] PHY_DQ_IE_TIMING_1 DDRSS_PHY_613[15-8] PHY_DQ_IE_TIMING_2 DDRSS_PHY_869[15-8] PHY_DQ_IE_TIMING_3	
DDRSS_PHY_101[23-16] PHY_DQS_IE_TIMING_0 DDRSS_PHY_357[23-16] PHY_DQS_IE_TIMING_1 DDRSS_PHY_613[23-16] PHY_DQS_IE_TIMING_2 DDRSS_PHY_869[23-16] PHY_DQS_IE_TIMING_3	
DDRSS_PHY_92[21-16] PHY_PAD_DSLICE_IO_CFG_0 DDRSS_PHY_348[21-16] PHY_PAD_DSLICE_IO_CFG_1 DDRSS_PHY_604[21-16] PHY_PAD_DSLICE_IO_CFG_2 DDRSS_PHY_860[21-16] PHY_PAD_DSLICE_IO_CFG_3	Pad Controls
Address Slice	

Copyright © 2024 Texas Instruments Incorporated

Table 8-18. Frequency Based Register Settings (continued)

Register Settings	Category
DDRSS_PHY_1065[4-0] PHY_ADR0_SW_WRADDR_SHIFT_0 DDRSS_PHY_1065[28-24] PHY_ADR1_SW_WRADDR_SHIFT_0 DDRSS_PHY_1066[20-16] PHY_ADR2_SW_WRADDR_SHIFT_0 DDRSS_PHY_1067[20-16] PHY_ADR3_SW_WRADDR_SHIFT_0 DDRSS_PHY_1068[20-16] PHY_ADR4_SW_WRADDR_SHIFT_0 DDRSS_PHY_1069[20-16] PHY_ADR5_SW_WRADDR_SHIFT_0	Overrides
DDRSS_PHY_1070[19-16] PHY_ADR_SW_MASTER_MODE_0	Master delay line
DDRSS_PHY_1071[10-0] PHY_ADR_MASTER_DELAY_START_0	
DDRSS_PHY_1071[21-16] PHY_ADR_MASTER_DELAY_STEP_0	
DDRSS_PHY_1071[31-24] PHY_ADR_MASTER_DELAY_WAIT_0	
DDRSS_PHY_1072[7-0] PHY_ADR_MASTER_DELAY_HALF_MEASURE_0	
DDRSS_PHY_1064[7-0] PHY_ADR_TSEL_SELECT_0	Termination settings
DDRSS_PHY_1073[3-0] PHY_ADR_CALVL_DLY_STEP_0	CA training
DDRSS_PHY_1074[3-0] PHY_ADR_CALVL_CAPTURE_CNT_0	
DDRSS_PHY_1072[17-8] PHY_ADR_SW_CALVL_DVW_MIN_0	
DDRSS_PHY_1072[24] PHY_ADR_SW_CALVL_DVW_MIN_EN_0	
DDRSS_PHY_1065[18-8] PHY_ADR0_CLK_WR_SLAVE_DELAY_0 DDRSS_PHY_1066[10-0] PHY_ADR1_CLK_WR_SLAVE_DELAY_0 DDRSS_PHY_1067[10-0] PHY_ADR2_CLK_WR_SLAVE_DELAY_0 DDRSS_PHY_1068[10-0] PHY_ADR3_CLK_WR_SLAVE_DELAY_0 DDRSS_PHY_1069[10-0] PHY_ADR4_CLK_WR_SLAVE_DELAY_0 DDRSS_PHY_1070[10-0] PHY_ADR5_CLK_WR_SLAVE_DELAY_0	Address delay lines
DDRSS_PHY_1064[10-8] PHY_ADR_DC_CAL_CLK_SEL_0	Duty cycle correction
DDRSS_PHY_1074[25-16] PHY_ADR_DC_INIT_SLV_DELAY_0	
DDRSS_PHY_1075[0] PHY_ADR_DC_CALVL_ENABLE_0	
DDRSS_PHY_1075[15-8] PHY_ADR_DC_DM_CLK_THRSHLD_0	
DDRSS_PHY_1064[26-16] PHY_PAD_ADR_IO_CFG_0	Miscellaneous
PHY Level	
DDRSS_PHY_1396[12-0] PHY_PLL_CTRL	PLL settings
DDRSS_PHY_1395[0] PHY_PLL_BYPASS	
DDRSS_PHY_1405[2-0] PHY_CLK_DC_CAL_CLK_SEL	Duty cycle correction
DDRSS_PHY_1308[7-0] PHY_CLK_DC_DM_THRSHLD	
DDRSS_PHY_1399[10-0] PHY_GRP0_SLAVE_DELAY_0 DDRSS_PHY_1399[26-16] PHY_GRP1_SLAVE_DELAY_0 DDRSS_PHY_1400[10-0] PHY_GRP2_SLAVE_DELAY_0 DDRSS_PHY_1400[26-16] PHY_GRP3_SLAVE_DELAY_0 DDRSS_PHY_1401[10-0] PHY_GRP0_SLAVE_DELAY_1 DDRSS_PHY_1402[10-0] PHY_GRP1_SLAVE_DELAY_1 DDRSS_PHY_1403[10-0] PHY_GRP2_SLAVE_DELAY_1 DDRSS_PHY_1404[10-0] PHY_GRP3_SLAVE_DELAY_1	Address/control delay lines
DDRSS_PHY_1397[27-24] PHY_CSLVL_DLY_STEP	CS training
DDRSS_PHY_1396[16] PHY_LOW_FREQ_SEL	Low frequency select
DDRSS_PHY_1422[2-0] PHY_CAL_CLK_SELECT_0	Calibration
DDRSS_PHY_1422[30-24] PHY_CAL_SETTLING_PRD_0	
DDRSS_PHY_1422[23-8] PHY_CAL_VREF_SWITCH_TIMER_0	
DDRSS_PHY_1393[17-0] PHY_PAD_CAL_IO_CFG_0	

Copyright © 2024 Texas Instruments Incorporated

Table 8-18. Frequency Based Register Settings (continued)

Register Settings	Category
DDRSS_PHY_1406[29-0] PHY_PAD_FDBK_DRIVE	Pad Controls
DDRSS_PHY_1407[17-0] PHY_PAD_FDBK_DRIVE2	
DDRSS_PHY_1408[30-0] PHY_PAD_DATA_DRIVE	
DDRSS_PHY_1409[31-0] PHY_PAD_DQS_DRIVE	
DDRSS_PHY_1410[29-0] PHY_PAD_ADDR_DRIVE	
DDRSS_PHY_1411[26-0] PHY_PAD_ADDR_DRIVE2	
DDRSS_PHY_1412[31-0] PHY_PAD_CLK_DRIVE	
DDRSS_PHY_1413[17-0] PHY_PAD_CLK_DRIVE2	

Some register settings in the PHY are both chip select based and frequency based. For them, the PHY_PER_CS_TRAINING_INDEX_0/1/2/3 bits and the DDRSS_PHY_1281[17-16] PHY_FREQ_SEL_INDEX field apply to the register access. The DDRSS_PHY_1281[8] PHY_FREQ_SEL_MULTICAST_EN bit applies as well. Table 8-19 shows the chip select based and frequency based register settings.

Table 8-19. Chip Select Based and Frequency Based Register Settings

Register Settings	Category
Data Slice	

Copyright © 2024 Texas Instruments Incorporated

Table 8-19. Chip Select Based and Frequency Based Register Settings (continued)

Register Settings	Category
DDRSS_PHY_628[10-0] PHY_CLK_WRDQ0_SLAVE_DELAY_2 DDRSS_PHY_628[26-16] PHY_CLK_WRDQ1_SLAVE_DELAY_2 DDRSS_PHY_629[10-0] PHY_CLK_WRDQ2_SLAVE_DELAY_2 DDRSS_PHY_629[26-16] PHY_CLK_WRDQ3_SLAVE_DELAY_2 DDRSS_PHY_630[10-0] PHY_CLK_WRDQ4_SLAVE_DELAY_2 DDRSS_PHY_630[26-16] PHY_CLK_WRDQ5_SLAVE_DELAY_2 DDRSS_PHY_631[10-0] PHY_CLK_WRDQ6_SLAVE_DELAY_2 DDRSS_PHY_631[26-16] PHY_CLK_WRDQ7_SLAVE_DELAY_2 DDRSS_PHY_884[10-0] PHY_CLK_WRDQ0_SLAVE_DELAY_3 DDRSS_PHY_884[26-16] PHY_CLK_WRDQ1_SLAVE_DELAY_3 DDRSS_PHY_885[10-0] PHY_CLK_WRDQ2_SLAVE_DELAY_3 DDRSS_PHY_885[26-16] PHY_CLK_WRDQ3_SLAVE_DELAY_3 DDRSS_PHY_886[10-0] PHY_CLK_WRDQ4_SLAVE_DELAY_3 DDRSS_PHY_886[26-16] PHY_CLK_WRDQ5_SLAVE_DELAY_3 DDRSS_PHY_887[10-0] PHY_CLK_WRDQ6_SLAVE_DELAY_3 DDRSS_PHY_887[26-16] PHY_CLK_WRDQ7_SLAVE_DELAY_3 DDRSS_PHY_116[10-0] PHY_CLK_WRDQ0_SLAVE_DELAY_0 DDRSS_PHY_116[26-16] PHY_CLK_WRDQ1_SLAVE_DELAY_0 DDRSS_PHY_117[10-0] PHY_CLK_WRDQ2_SLAVE_DELAY_0 DDRSS_PHY_117[26-16] PHY_CLK_WRDQ3_SLAVE_DELAY_0 DDRSS_PHY_118[10-0] PHY_CLK_WRDQ4_SLAVE_DELAY_0 DDRSS_PHY_118[26-16] PHY_CLK_WRDQ5_SLAVE_DELAY_0 DDRSS_PHY_119[10-0] PHY_CLK_WRDQ6_SLAVE_DELAY_0 DDRSS_PHY_119[26-16] PHY_CLK_WRDQ7_SLAVE_DELAY_0 DDRSS_PHY_372[10-0] PHY_CLK_WRDQ0_SLAVE_DELAY_1 DDRSS_PHY_372[26-16] PHY_CLK_WRDQ1_SLAVE_DELAY_1 DDRSS_PHY_373[10-0] PHY_CLK_WRDQ2_SLAVE_DELAY_1 DDRSS_PHY_373[26-16] PHY_CLK_WRDQ3_SLAVE_DELAY_1 DDRSS_PHY_374[10-0] PHY_CLK_WRDQ4_SLAVE_DELAY_1 DDRSS_PHY_374[26-16] PHY_CLK_WRDQ5_SLAVE_DELAY_1 DDRSS_PHY_375[10-0] PHY_CLK_WRDQ6_SLAVE_DELAY_1 DDRSS_PHY_375[26-16] PHY_CLK_WRDQ7_SLAVE_DELAY_1	Write path
DDRSS_PHY_120[10-0] PHY_CLK_WRDM_SLAVE_DELAY_0 DDRSS_PHY_376[10-0] PHY_CLK_WRDM_SLAVE_DELAY_1 DDRSS_PHY_632[10-0] PHY_CLK_WRDM_SLAVE_DELAY_2 DDRSS_PHY_888[10-0] PHY_CLK_WRDM_SLAVE_DELAY_3	
DDRSS_PHY_120[25-16] PHY_CLK_WRDQS_SLAVE_DELAY_0 DDRSS_PHY_376[25-16] PHY_CLK_WRDQS_SLAVE_DELAY_1 DDRSS_PHY_632[25-16] PHY_CLK_WRDQS_SLAVE_DELAY_2 DDRSS_PHY_888[25-16] PHY_CLK_WRDQS_SLAVE_DELAY_3	
DDRSS_PHY_131[10-8] PHY_WRITE_PATH_LAT_ADD_0 DDRSS_PHY_387[10-8] PHY_WRITE_PATH_LAT_ADD_1 DDRSS_PHY_643[10-8] PHY_WRITE_PATH_LAT_ADD_2 DDRSS_PHY_899[10-8] PHY_WRITE_PATH_LAT_ADD_3	

Copyright © 2024 Texas Instruments Incorporated

Table 8-19. Chip Select Based and Frequency Based Register Settings (continued)

Register Settings	Category
DDRSS_PHY_136[15-8] PHY_DATA_DC_DQ0_CLK_ADJUST_0 DDRSS_PHY_136[23-16] PHY_DATA_DC_DQ1_CLK_ADJUST_0 DDRSS_PHY_136[31-24] PHY_DATA_DC_DQ2_CLK_ADJUST_0 DDRSS_PHY_137[7-0] PHY_DATA_DC_DQ3_CLK_ADJUST_0 DDRSS_PHY_137[15-8] PHY_DATA_DC_DQ4_CLK_ADJUST_0 DDRSS_PHY_137[23-16] PHY_DATA_DC_DQ5_CLK_ADJUST_0 DDRSS_PHY_137[31-24] PHY_DATA_DC_DQ6_CLK_ADJUST_0 DDRSS_PHY_138[7-0] PHY_DATA_DC_DQ7_CLK_ADJUST_0 DDRSS_PHY_392[15-8] PHY_DATA_DC_DQ0_CLK_ADJUST_1 DDRSS_PHY_392[23-16] PHY_DATA_DC_DQ1_CLK_ADJUST_1 DDRSS_PHY_392[31-24] PHY_DATA_DC_DQ2_CLK_ADJUST_1 DDRSS_PHY_393[7-0] PHY_DATA_DC_DQ3_CLK_ADJUST_1 DDRSS_PHY_393[15-8] PHY_DATA_DC_DQ4_CLK_ADJUST_1 DDRSS_PHY_393[23-16] PHY_DATA_DC_DQ5_CLK_ADJUST_1 DDRSS_PHY_393[31-24] PHY_DATA_DC_DQ6_CLK_ADJUST_1 DDRSS_PHY_394[7-0] PHY_DATA_DC_DQ7_CLK_ADJUST_1 DDRSS_PHY_648[15-8] PHY_DATA_DC_DQ0_CLK_ADJUST_2 DDRSS_PHY_648[23-16] PHY_DATA_DC_DQ1_CLK_ADJUST_2 DDRSS_PHY_648[31-24] PHY_DATA_DC_DQ2_CLK_ADJUST_2 DDRSS_PHY_649[7-0] PHY_DATA_DC_DQ3_CLK_ADJUST_2 DDRSS_PHY_649[15-8] PHY_DATA_DC_DQ4_CLK_ADJUST_2 DDRSS_PHY_649[23-16] PHY_DATA_DC_DQ5_CLK_ADJUST_2 DDRSS_PHY_649[31-24] PHY_DATA_DC_DQ6_CLK_ADJUST_2 DDRSS_PHY_650[7-0] PHY_DATA_DC_DQ7_CLK_ADJUST_2 DDRSS_PHY_904[15-8] PHY_DATA_DC_DQ0_CLK_ADJUST_3 DDRSS_PHY_904[23-16] PHY_DATA_DC_DQ1_CLK_ADJUST_3 DDRSS_PHY_904[31-24] PHY_DATA_DC_DQ2_CLK_ADJUST_3 DDRSS_PHY_905[7-0] PHY_DATA_DC_DQ3_CLK_ADJUST_3 DDRSS_PHY_905[15-8] PHY_DATA_DC_DQ4_CLK_ADJUST_3 DDRSS_PHY_905[23-16] PHY_DATA_DC_DQ5_CLK_ADJUST_3 DDRSS_PHY_905[31-24] PHY_DATA_DC_DQ6_CLK_ADJUST_3 DDRSS_PHY_906[7-0] PHY_DATA_DC_DQ7_CLK_ADJUST_3	Duty cycle correction
DDRSS_PHY_138[15-8] PHY_DATA_DC_DM_CLK_ADJUST_0 DDRSS_PHY_394[15-8] PHY_DATA_DC_DM_CLK_ADJUST_1 DDRSS_PHY_650[15-8] PHY_DATA_DC_DM_CLK_ADJUST_2 DDRSS_PHY_906[15-8] PHY_DATA_DC_DM_CLK_ADJUST_3	Duty cycle correction
DDRSS_PHY_136[7-0] PHY_DATA_DC_DQS_CLK_ADJUST_0 DDRSS_PHY_392[7-0] PHY_DATA_DC_DQS_CLK_ADJUST_1 DDRSS_PHY_648[7-0] PHY_DATA_DC_DQS_CLK_ADJUST_2 DDRSS_PHY_904[7-0] PHY_DATA_DC_DQS_CLK_ADJUST_3	
DDRSS_PHY_131[25-16] PHY_WRLVL_DELAY_EARLY_THRESHOLD_0 DDRSS_PHY_387[25-16] PHY_WRLVL_DELAY_EARLY_THRESHOLD_1 DDRSS_PHY_643[25-16] PHY_WRLVL_DELAY_EARLY_THRESHOLD_2 DDRSS_PHY_899[25-16] PHY_WRLVL_DELAY_EARLY_THRESHOLD_3	Write leveling
DDRSS_PHY_132[9-0] PHY_WRLVL_DELAY_PERIOD_THRESHOLD_0 DDRSS_PHY_388[9-0] PHY_WRLVL_DELAY_PERIOD_THRESHOLD_1 DDRSS_PHY_644[9-0] PHY_WRLVL_DELAY_PERIOD_THRESHOLD_2 DDRSS_PHY_900[9-0] PHY_WRLVL_DELAY_PERIOD_THRESHOLD_3	
DDRSS_PHY_132[16] PHY_WRLVL_EARLY_FORCE_ZERO_0 DDRSS_PHY_388[16] PHY_WRLVL_EARLY_FORCE_ZERO_1 DDRSS_PHY_644[16] PHY_WRLVL_EARLY_FORCE_ZERO_2 DDRSS_PHY_900[16] PHY_WRLVL_EARLY_FORCE_ZERO_3	
DDRSS_PHY_121[1-0] PHY_WRLVL_THRESHOLD_ADJUST_0 DDRSS_PHY_377[1-0] PHY_WRLVL_THRESHOLD_ADJUST_1 DDRSS_PHY_633[1-0] PHY_WRLVL_THRESHOLD_ADJUST_2 DDRSS_PHY_889[1-0] PHY_WRLVL_THRESHOLD_ADJUST_3	

Copyright © 2024 Texas Instruments Incorporated

Table 8-19. Chip Select Based and Frequency Based Register Settings (continued)

Register Settings	Category
DDRSS_PHY_134[10-0] PHY_WDQLVL_DQDM_SLV_DLY_START_0 DDRSS_PHY_390[10-0] PHY_WDQLVL_DQDM_SLV_DLY_START_1 DDRSS_PHY_646[10-0] PHY_WDQLVL_DQDM_SLV_DLY_START_2 DDRSS_PHY_902[10-0] PHY_WDQLVL_DQDM_SLV_DLY_START_3	Write DQ training
DDRSS_PHY_134[19-16] PHY_NTP_WRLAT_START_0 DDRSS_PHY_390[19-16] PHY_NTP_WRLAT_START_1 DDRSS_PHY_646[19-16] PHY_NTP_WRLAT_START_2 DDRSS_PHY_902[19-16] PHY_NTP_WRLAT_START_3	
DDRSS_PHY_134[24] PHY_NTP_PASS_0 DDRSS_PHY_390[24] PHY_NTP_PASS_1 DDRSS_PHY_646[24] PHY_NTP_PASS_2 DDRSS_PHY_902[24] PHY_NTP_PASS_3	
DDRSS_PHY_121[17-8] PHY_RDDQS_DQ0_RISE_SLAVE_DELAY_0 DDRSS_PHY_122[25-16] PHY_RDDQS_DQ1_RISE_SLAVE_DELAY_0 DDRSS_PHY_123[25-16] PHY_RDDQS_DQ2_RISE_SLAVE_DELAY_0 DDRSS_PHY_124[25-16] PHY_RDDQS_DQ3_RISE_SLAVE_DELAY_0 DDRSS_PHY_125[25-16] PHY_RDDQS_DQ4_RISE_SLAVE_DELAY_0 DDRSS_PHY_126[25-16] PHY_RDDQS_DQ5_RISE_SLAVE_DELAY_0 DDRSS_PHY_127[25-16] PHY_RDDQS_DQ6_RISE_SLAVE_DELAY_0 DDRSS_PHY_128[25-16] PHY_RDDQS_DQ7_RISE_SLAVE_DELAY_0 DDRSS_PHY_377[17-8] PHY_RDDQS_DQ0_RISE_SLAVE_DELAY_1 DDRSS_PHY_378[25-16] PHY_RDDQS_DQ1_RISE_SLAVE_DELAY_1 DDRSS_PHY_379[25-16] PHY_RDDQS_DQ2_RISE_SLAVE_DELAY_1 DDRSS_PHY_380[25-16] PHY_RDDQS_DQ3_RISE_SLAVE_DELAY_1 DDRSS_PHY_381[25-16] PHY_RDDQS_DQ4_RISE_SLAVE_DELAY_1 DDRSS_PHY_382[25-16] PHY_RDDQS_DQ5_RISE_SLAVE_DELAY_1 DDRSS_PHY_383[25-16] PHY_RDDQS_DQ6_RISE_SLAVE_DELAY_1 DDRSS_PHY_384[25-16] PHY_RDDQS_DQ7_RISE_SLAVE_DELAY_1 DDRSS_PHY_633[17-8] PHY_RDDQS_DQ0_RISE_SLAVE_DELAY_2 DDRSS_PHY_634[25-16] PHY_RDDQS_DQ1_RISE_SLAVE_DELAY_2 DDRSS_PHY_635[25-16] PHY_RDDQS_DQ2_RISE_SLAVE_DELAY_2 DDRSS_PHY_636[25-16] PHY_RDDQS_DQ3_RISE_SLAVE_DELAY_2 DDRSS_PHY_637[25-16] PHY_RDDQS_DQ4_RISE_SLAVE_DELAY_2 DDRSS_PHY_638[25-16] PHY_RDDQS_DQ5_RISE_SLAVE_DELAY_2 DDRSS_PHY_639[25-16] PHY_RDDQS_DQ6_RISE_SLAVE_DELAY_2 DDRSS_PHY_640[25-16] PHY_RDDQS_DQ7_RISE_SLAVE_DELAY_2 DDRSS_PHY_889[17-8] PHY_RDDQS_DQ0_RISE_SLAVE_DELAY_3 DDRSS_PHY_890[25-16] PHY_RDDQS_DQ1_RISE_SLAVE_DELAY_3 DDRSS_PHY_891[25-16] PHY_RDDQS_DQ2_RISE_SLAVE_DELAY_3 DDRSS_PHY_892[25-16] PHY_RDDQS_DQ3_RISE_SLAVE_DELAY_3 DDRSS_PHY_893[25-16] PHY_RDDQS_DQ4_RISE_SLAVE_DELAY_3 DDRSS_PHY_894[25-16] PHY_RDDQS_DQ5_RISE_SLAVE_DELAY_3 DDRSS_PHY_895[25-16] PHY_RDDQS_DQ6_RISE_SLAVE_DELAY_3 DDRSS_PHY_896[25-16] PHY_RDDQS_DQ7_RISE_SLAVE_DELAY_3	Read path

Copyright © 2024 Texas Instruments Incorporated

Table 8-19. Chip Select Based and Frequency Based Register Settings (continued)

Register Settings	Category
DDRSS_PHY_122[9-0] PHY_RDDQS_DQ0_FALL_SLAVE_DELAY_0 DDRSS_PHY_123[9-0] PHY_RDDQS_DQ1_FALL_SLAVE_DELAY_0 DDRSS_PHY_124[9-0] PHY_RDDQS_DQ2_FALL_SLAVE_DELAY_0 DDRSS_PHY_125[9-0] PHY_RDDQS_DQ3_FALL_SLAVE_DELAY_0 DDRSS_PHY_126[9-0] PHY_RDDQS_DQ4_FALL_SLAVE_DELAY_0 DDRSS_PHY_127[9-0] PHY_RDDQS_DQ5_FALL_SLAVE_DELAY_0 DDRSS_PHY_128[9-0] PHY_RDDQS_DQ6_FALL_SLAVE_DELAY_0 DDRSS_PHY_129[9-0] PHY_RDDQS_DQ7_FALL_SLAVE_DELAY_0 DDRSS_PHY_378[9-0] PHY_RDDQS_DQ0_FALL_SLAVE_DELAY_1 DDRSS_PHY_379[9-0] PHY_RDDQS_DQ1_FALL_SLAVE_DELAY_1 DDRSS_PHY_380[9-0] PHY_RDDQS_DQ2_FALL_SLAVE_DELAY_1 DDRSS_PHY_381[9-0] PHY_RDDQS_DQ3_FALL_SLAVE_DELAY_1 DDRSS_PHY_382[9-0] PHY_RDDQS_DQ4_FALL_SLAVE_DELAY_1 DDRSS_PHY_383[9-0] PHY_RDDQS_DQ5_FALL_SLAVE_DELAY_1 DDRSS_PHY_384[9-0] PHY_RDDQS_DQ6_FALL_SLAVE_DELAY_1 DDRSS_PHY_385[9-0] PHY_RDDQS_DQ7_FALL_SLAVE_DELAY_1 DDRSS_PHY_634[9-0] PHY_RDDQS_DQ0_FALL_SLAVE_DELAY_2 DDRSS_PHY_635[9-0] PHY_RDDQS_DQ1_FALL_SLAVE_DELAY_2 DDRSS_PHY_636[9-0] PHY_RDDQS_DQ2_FALL_SLAVE_DELAY_2 DDRSS_PHY_637[9-0] PHY_RDDQS_DQ3_FALL_SLAVE_DELAY_2 DDRSS_PHY_638[9-0] PHY_RDDQS_DQ4_FALL_SLAVE_DELAY_2 DDRSS_PHY_639[9-0] PHY_RDDQS_DQ5_FALL_SLAVE_DELAY_2 DDRSS_PHY_640[9-0] PHY_RDDQS_DQ6_FALL_SLAVE_DELAY_2 DDRSS_PHY_641[9-0] PHY_RDDQS_DQ7_FALL_SLAVE_DELAY_2 DDRSS_PHY_890[9-0] PHY_RDDQS_DQ0_FALL_SLAVE_DELAY_3 DDRSS_PHY_891[9-0] PHY_RDDQS_DQ1_FALL_SLAVE_DELAY_3 DDRSS_PHY_892[9-0] PHY_RDDQS_DQ2_FALL_SLAVE_DELAY_3 DDRSS_PHY_893[9-0] PHY_RDDQS_DQ3_FALL_SLAVE_DELAY_3 DDRSS_PHY_894[9-0] PHY_RDDQS_DQ4_FALL_SLAVE_DELAY_3 DDRSS_PHY_895[9-0] PHY_RDDQS_DQ5_FALL_SLAVE_DELAY_3 DDRSS_PHY_896[9-0] PHY_RDDQS_DQ6_FALL_SLAVE_DELAY_3 DDRSS_PHY_897[9-0] PHY_RDDQS_DQ7_FALL_SLAVE_DELAY_3	Read path
DDRSS_PHY_129[25-16] PHY_RDDQS_DM_RISE_SLAVE_DELAY_0 DDRSS_PHY_385[25-16] PHY_RDDQS_DM_RISE_SLAVE_DELAY_1 DDRSS_PHY_641[25-16] PHY_RDDQS_DM_RISE_SLAVE_DELAY_2 DDRSS_PHY_897[25-16] PHY_RDDQS_DM_RISE_SLAVE_DELAY_3	Read path
DDRSS_PHY_130[9-0] PHY_RDDQS_DM_FALL_SLAVE_DELAY_0 DDRSS_PHY_386[9-0] PHY_RDDQS_DM_FALL_SLAVE_DELAY_1 DDRSS_PHY_642[9-0] PHY_RDDQS_DM_FALL_SLAVE_DELAY_2 DDRSS_PHY_898[9-0] PHY_RDDQS_DM_FALL_SLAVE_DELAY_3	
DDRSS_PHY_130[25-16] PHY_RDDQS_GATE_SLAVE_DELAY_0 DDRSS_PHY_386[25-16] PHY_RDDQS_GATE_SLAVE_DELAY_1 DDRSS_PHY_642[25-16] PHY_RDDQS_GATE_SLAVE_DELAY_2 DDRSS_PHY_898[25-16] PHY_RDDQS_GATE_SLAVE_DELAY_3	
DDRSS_PHY_131[3-0] PHY_RDDQS_LATENCY_ADJUST_0 DDRSS_PHY_387[3-0] PHY_RDDQS_LATENCY_ADJUST_1 DDRSS_PHY_643[3-0] PHY_RDDQS_LATENCY_ADJUST_2 DDRSS_PHY_899[3-0] PHY_RDDQS_LATENCY_ADJUST_3	
DDRSS_PHY_133[9-0] PHY_GTLVL_RDDQS_SLV_DLY_START_0 DDRSS_PHY_389[9-0] PHY_GTLVL_RDDQS_SLV_DLY_START_1 DDRSS_PHY_645[9-0] PHY_GTLVL_RDDQS_SLV_DLY_START_2 DDRSS_PHY_901[9-0] PHY_GTLVL_RDDQS_SLV_DLY_START_3	Gate training
DDRSS_PHY_133[19-16] PHY_GTLVL_LAT_ADJ_START_0 DDRSS_PHY_389[19-16] PHY_GTLVL_LAT_ADJ_START_1 DDRSS_PHY_645[19-16] PHY_GTLVL_LAT_ADJ_START_2 DDRSS_PHY_901[19-16] PHY_GTLVL_LAT_ADJ_START_3	
DDRSS_PHY_135[9-0] PHY_RDLVL_RDDQS_DQ_SLV_DLY_START_0 DDRSS_PHY_391[9-0] PHY_RDLVL_RDDQS_DQ_SLV_DLY_START_1 DDRSS_PHY_647[9-0] PHY_RDLVL_RDDQS_DQ_SLV_DLY_START_2 DDRSS_PHY_903[9-0] PHY_RDLVL_RDDQS_DQ_SLV_DLY_START_3	Read data eye training

Copyright © 2024 Texas Instruments Incorporated

Table 8-19. Chip Select Based and Frequency Based Register Settings (continued)

Register Settings	Category
DDRSS_PHY_138[31-16] PHY_DSLICE_PAD_BOOSTPN_SETTING_0 DDRSS_PHY_394[31-16] PHY_DSLICE_PAD_BOOSTPN_SETTING_1 DDRSS_PHY_650[31-16] PHY_DSLICE_PAD_BOOSTPN_SETTING_2 DDRSS_PHY_906[31-16] PHY_DSLICE_PAD_BOOSTPN_SETTING_3	Miscellaneous
DDRSS_PHY_139[5-0] PHY_DSLICE_PAD_RX_CTLE_SETTING_0 DDRSS_PHY_395[5-0] PHY_DSLICE_PAD_RX_CTLE_SETTING_1 DDRSS_PHY_651[5-0] PHY_DSLICE_PAD_RX_CTLE_SETTING_2 DDRSS_PHY_907[5-0] PHY_DSLICE_PAD_RX_CTLE_SETTING_3	

8.2.3.7.8 Low-Power Modes

The DDR PHY provides a means to reduce the operational power when it is not actively transferring read or write data for an extended periond of time. This is done using the DFI low power interface. Using this interface, the DDR controller communicates to the PHY that it may enter its own low-power state and informs the PHY how quickly it requires the PHY to recover if a low-power state is entered. This DDR PHY has the following power modes:

- **Idle**: This is a period in which no read or write activity is being performed by the DDR controller. The PHY is not in any low power state, and is ready to accept commands at any time.
- **Read/Write**: This describes the time period when the PHY is actively performing a read or write operation triggered by the DDR controller.
- **Light Sleep**: This is a low-power state in which the master delay lines and their logic receive a clock, but all other slice logic is gated off. The PHY PLLs remain in an active state. Register reads and writes to data and address slices are not permitted. The PHY returns quickly to functional operation after an exit from light sleep. After exiting the PHY initializes the slave delay settings automatically. The master delay line macro maintains lock. This mode is entered if the dfi_lp_wakeup signal is driven with a value less than or equal to the programmed value in bits [3:0] of the the DDRSS_PHY_1318[15-8] PHY_LP_WAKEUP field. If low power idle state is enabled, the value must also be greater than the value of bits [7:4] in the DDRSS_PHY_1318[15-8] PHY_LP_WAKEUP field.
- **Deep Sleep**: This is an ultra low-power state in which clocking to the data, address, and address/control slices is disabled. The PLLs are in a power down state. Pad calibration functions are frozen. When in this mode, no DFI accesses or register reads and writes can be performed. This mode is entered if the dfi_lp_wakeup signal is driven with a value greater than the programmed value in bits [3:0] of the the DDRSS_PHY_1318[15-8] PHY_LP_WAKEUP field.
- **Low Power Idle**: This is a low power version of idle where the data slices are placed in a light sleep mode while still allowing accesses to the data slice register address space. The DDRSS_PHY_1318[16] PHY_LS_IDLE_EN bit enables this low power idle state. The mode is entered if the dfi_lp_wakeup signal is driven with a value less than or equal to the programmed value of bits [7:4] in the DDRSS_PHY_1318[15-8] PHY_LP_WAKEUP field.

In general, if the SDRAM is placed into a power-down mode, the DDR PHY should enter light sleep. If the SDRAM is placed in a self-refresh mode, the DDR PHY could enter deep sleep.

The DDRSS_PHY_1319[9-0] PHY_LP_CTRLUPD_CNTR_CFG field controls the number of clock controller cycles between de-assertion of the dfi_lp_req signal and de-assertion of the dfi_lp_ack signal during exit of light sleep and low power idle modes.

Within each data slice, the DQ, DM and DQS I/O's are disabled when not being written to or read from. The gate feedback pads are constantly enabled to control the read DQS gate open logic. The following fields control disabling the gate feedback pad for the corresponding slice during deep sleep mode:

- DDRSS_PHY_79[26-24] PHY_FDBK_PWR_CTRL_0
- DDRSS_PHY_335[26-24] PHY_FDBK_PWR_CTRL_1
- DDRSS_PHY_591[26-24] PHY_FDBK_PWR_CTRL_2
- DDRSS_PHY_847[26-24] PHY_FDBK_PWR_CTRL_3

8.2.3.7.9 Training Support

Training of the DDR PHY and SDRAM is necessary to support high-speed operation. The general training flow as follows:

Copyright © 2024 Texas Instruments Incorporated

1. I/O calibration
2. CA/CS training (VREF CA)
3. Write leveling
4. Read DQS gate training
5. Read data eye training
6. Write data eye training (VREF DQ)

8.2.3.7.9.1 Write Leveling

When the write leveling algorithm has completed, the final write DQS delay settings can be found in the following fields:

- DDRSS_PHY_120[25-16] PHY_CLK_WRDQS_SLAVE_DELAY_0
- DDRSS_PHY_376[25-16] PHY_CLK_WRDQS_SLAVE_DELAY_1
- DDRSS_PHY_632[25-16] PHY_CLK_WRDQS_SLAVE_DELAY_2
- DDRSS_PHY_888[25-16] PHY_CLK_WRDQS_SLAVE_DELAY_3
- DDRSS_PHY_121[1-0] PHY_WRLVL_THRESHOLD_ADJUST_0
- DDRSS_PHY_377[1-0] PHY_WRLVL_THRESHOLD_ADJUST_1
- DDRSS_PHY_633[1-0] PHY_WRLVL_THRESHOLD_ADJUST_2
- DDRSS_PHY_889[1-0] PHY_WRLVL_THRESHOLD_ADJUST_3

After write leveling is complete, the following fields can be checked to obtain the leveling status:

- DDRSS_PHY_50[16-0] PHY_WRLVL_STATUS_OBS_0
- DDRSS_PHY_306[16-0] PHY_WRLVL_STATUS_OBS_1
- DDRSS_PHY_562[16-0] PHY_WRLVL_STATUS_OBS_2
- DDRSS_PHY_818[16-0] PHY_WRLVL_STATUS_OBS_3

After write leveling is complete, software can always override the results by writing directly to the following fields:

- DDRSS_PHY_120[25-16] PHY_CLK_WRDQS_SLAVE_DELAY_0
- DDRSS_PHY_376[25-16] PHY_CLK_WRDQS_SLAVE_DELAY_1
- DDRSS_PHY_632[25-16] PHY_CLK_WRDQS_SLAVE_DELAY_2
- DDRSS_PHY_888[25-16] PHY_CLK_WRDQS_SLAVE_DELAY_3

If the clock path to the SDRAM is longer than the DQS path to the SDRAM, the leveling result from the first SDRAM along the memory clock path always returns a small value greater than zero because the DQS signal needs to be moved slightly to the right to line up with the memory clock. If the clock path to the SDRAM is shorter than the DQS path to the SDRAM, the DQS signal may need to be moved close to a cycle before it lines up with the memory clock. The leveling result indicates to the slice that the DQS needs to be delayed almost a cycle. This would then result in the write data being a cycle later than expected by the SDRAM based on when the write command is received. In reality, the DQS needs to be negatively delayed to align to the memory clock in the proper cycle. To create this “negative” delay, the command bus needs to be delayed one cycle so that when the DQS is delayed almost one cycle, the DQS appears “early” with respect to the write command. For example, if the memory clock path for slice 0 is expected to be about 1/16th cycle shorter than the DQS path, a write leveling result of 0x1E0 would be expected. The DDRSS_PHY_131[25-16] PHY_WRLVL_DELAY_EARLY_THRESHOLD_0 field enables the data slice 0 to understand if this is an expected delay or if this is a “negative” delay condition. When the result contained in DDRSS_PHY_120[25-16] PHY_CLK_WRDQS_SLAVE_DELAY_0 field is greater than the DDRSS_PHY_131[25-16] PHY_WRLVL_DELAY_EARLY_THRESHOLD_0 field, the command bus is delayed by one cycle.Table 8-20 shows examples of programming this field for various expected delays. The following fields are associated with the other data slices:

- DDRSS_PHY_387[25-16] PHY_WRLVL_DELAY_EARLY_THRESHOLD_1
- DDRSS_PHY_643[25-16] PHY_WRLVL_DELAY_EARLY_THRESHOLD_2
- DDRSS_PHY_889[25-16] PHY_WRLVL_DELAY_EARLY_THRESHOLD_3

Table 8-20. PHY Write Level Delay Values

Expected Delay	PHY_WRLVL_DELAY_EARLY_THRESHOLD_x x = 0, 1, 2, 3	Command Bus Impact
0.25 cycle	0x200 (off)	None
1.00 cycle	0x200 (off)	None

Copyright © 2024 Texas Instruments Incorporated

Table 8-20. PHY Write Level Delay Values (continued)

Expected Delay	PHY_WRLVL_DELAY_EARLY_THRESHOLD_x x = 0, 1, 2, 3	Command Bus Impact
0.00 cycles	0x1E0	1 cycle delay if result is > 0x1E0

The DDRSS_PHY_132[16] PHY_WRLVL_EARLY_FORCE_ZERO_0 bit works with the DDRSS_PHY_131[25-16] PHY_WRLVL_DELAY_EARLY_THRESHOLD_0 field for slice 0. The following fields are associated with the other slices:

- DDRSS_PHY_388[16] PHY_WRLVL_EARLY_FORCE_ZERO_1 and DDRSS_PHY_387[25-16] PHY_WRLVL_DELAY_EARLY_THRESHOLD_1
- DDRSS_PHY_644[16] PHY_WRLVL_EARLY_FORCE_ZERO_2 and DDRSS_PHY_643[25-16] PHY_WRLVL_DELAY_EARLY_THRESHOLD_2
- DDRSS_PHY_900[16] PHY_WRLVL_EARLY_FORCE_ZERO_3 and DDRSS_PHY_899[25-16] PHY_WRLVL_DELAY_EARLY_THRESHOLD_3

When the write leveling result satisfies the early threshold condition, the DQS is ideally aligned to clock but there is a cycle of latency added to the command path. Setting the PHY_WRLVL_EARLY_FORCE_ZERO_x bit to 0x1 forces the PHY_CLK_WRDQS_SLAVE_DELAY_x value to 0x0 so there is no command latency impact.

The PHY_WRLVL_THRESHOLD_ADJUST_x fields contain the results of the PHY_WRLVL_DELAY_PERIOD_THRESHOLD_x and PHY_WRLVL_DELAY_EARLY_THRESHOLD_x calculations from the write leveling process. This allows for direct observation of the leveling results. It also allows to rewrite leveling results if desired. This can be useful in case leveling is performed once and the same results are used in the future. The PHY_WRLVL_DELAY_EARLY_THRESHOLD_x and PHY_WRLVL_DELAY_PERIOD_THRESHOLD_x results can be applied through the PHY_WRLVL_THRESHOLD_ADJUST_x fields without having to go through the leveling process again.

The previously mentioned PHY_WRLVL_DELAY_PERIOD_THRESHOLD_x fields are accessed through the following registers:

- DDRSS_PHY_132[9-0] PHY_WRLVL_DELAY_PERIOD_THRESHOLD_0
- DDRSS_PHY_388[9-0] PHY_WRLVL_DELAY_PERIOD_THRESHOLD_1
- DDRSS_PHY_644[9-0] PHY_WRLVL_DELAY_PERIOD_THRESHOLD_2
- DDRSS_PHY_900[9-0] PHY_WRLVL_DELAY_PERIOD_THRESHOLD_3

8.2.3.7.9.2 Read Gate Training

When the read gate training algorithm has completed, the final read DQS slave delay settings can be found in the following fields:

- DDRSS_PHY_130[25-16] PHY_RDDQS_GATE_SLAVE_DELAY_0 (cycle fraction)
- DDRSS_PHY_386[25-16] PHY_RDDQS_GATE_SLAVE_DELAY_1 (cycle fraction)
- DDRSS_PHY_642[25-16] PHY_RDDQS_GATE_SLAVE_DELAY_2 (cycle fraction)
- DDRSS_PHY_898[25-16] PHY_RDDQS_GATE_SLAVE_DELAY_3 (cycle fraction)
- DDRSS_PHY_131[3-0] PHY_RDDQS_LATENCY_ADJUST_0 (cycle offset)
- DDRSS_PHY_387[3-0] PHY_RDDQS_LATENCY_ADJUST_1 (cycle offset)
- DDRSS_PHY_643[3-0] PHY_RDDQS_LATENCY_ADJUST_2 (cycle offset)
- DDRSS_PHY_899[3-0] PHY_RDDQS_LATENCY_ADJUST_3 (cycle offset)

After read gate training is complete, the following fields can be checked to obtain the training status:

- DDRSS_PHY_54[17-0] PHY_GTLVL_STATUS_OBS_0
- DDRSS_PHY_310[17-0] PHY_GTLVL_STATUS_OBS_1
- DDRSS_PHY_566[17-0] PHY_GTLVL_STATUS_OBS_2
- DDRSS_PHY_822[17-0] PHY_GTLVL_STATUS_OBS_3

After read gate training is complete, software can always override the results by writing directly to the PHY_RDDQS_GATE_SLAVE_DELAY_x and PHY_RDDQS_LATENCY_ADJUST_x fields.

Copyright © 2024 Texas Instruments Incorporated

8.2.3.7.9.3 Read Data Eye Training

When the read gate training algorithm has completed, the final read DQS slave delay settings can be found in the following fields:

- DDRSS_PHY_121[17-8] PHY_RDDQS_DQ0_RISE_SLAVE_DELAY_0
- DDRSS_PHY_122[25-16] PHY_RDDQS_DQ1_RISE_SLAVE_DELAY_0
- DDRSS_PHY_123[25-16] PHY_RDDQS_DQ2_RISE_SLAVE_DELAY_0
- DDRSS_PHY_124[25-16] PHY_RDDQS_DQ3_RISE_SLAVE_DELAY_0
- DDRSS_PHY_125[25-16] PHY_RDDQS_DQ4_RISE_SLAVE_DELAY_0
- DDRSS_PHY_126[25-16] PHY_RDDQS_DQ5_RISE_SLAVE_DELAY_0
- DDRSS_PHY_127[25-16] PHY_RDDQS_DQ6_RISE_SLAVE_DELAY_0
- DDRSS_PHY_128[25-16] PHY_RDDQS_DQ7_RISE_SLAVE_DELAY_0
- DDRSS_PHY_377[17-8] PHY_RDDQS_DQ0_RISE_SLAVE_DELAY_1
- DDRSS_PHY_378[25-16] PHY_RDDQS_DQ1_RISE_SLAVE_DELAY_1
- DDRSS_PHY_379[25-16] PHY_RDDQS_DQ2_RISE_SLAVE_DELAY_1
- DDRSS_PHY_380[25-16] PHY_RDDQS_DQ3_RISE_SLAVE_DELAY_1
- DDRSS_PHY_381[25-16] PHY_RDDQS_DQ4_RISE_SLAVE_DELAY_1
- DDRSS_PHY_382[25-16] PHY_RDDQS_DQ5_RISE_SLAVE_DELAY_1
- DDRSS_PHY_383[25-16] PHY_RDDQS_DQ6_RISE_SLAVE_DELAY_1
- DDRSS_PHY_384[25-16] PHY_RDDQS_DQ7_RISE_SLAVE_DELAY_1
- DDRSS_PHY_633[17-8] PHY_RDDQS_DQ0_RISE_SLAVE_DELAY_2
- DDRSS_PHY_634[25-16] PHY_RDDQS_DQ1_RISE_SLAVE_DELAY_2
- DDRSS_PHY_635[25-16] PHY_RDDQS_DQ2_RISE_SLAVE_DELAY_2
- DDRSS_PHY_636[25-16] PHY_RDDQS_DQ3_RISE_SLAVE_DELAY_2
- DDRSS_PHY_637[25-16] PHY_RDDQS_DQ4_RISE_SLAVE_DELAY_2
- DDRSS_PHY_638[25-16] PHY_RDDQS_DQ5_RISE_SLAVE_DELAY_2
- DDRSS_PHY_639[25-16] PHY_RDDQS_DQ6_RISE_SLAVE_DELAY_2
- DDRSS_PHY_640[25-16] PHY_RDDQS_DQ7_RISE_SLAVE_DELAY_2
- DDRSS_PHY_889[17-8] PHY_RDDQS_DQ0_RISE_SLAVE_DELAY_3
- DDRSS_PHY_890[25-16] PHY_RDDQS_DQ1_RISE_SLAVE_DELAY_3
- DDRSS_PHY_891[25-16] PHY_RDDQS_DQ2_RISE_SLAVE_DELAY_3
- DDRSS_PHY_892[25-16] PHY_RDDQS_DQ3_RISE_SLAVE_DELAY_3
- DDRSS_PHY_893[25-16] PHY_RDDQS_DQ4_RISE_SLAVE_DELAY_3
- DDRSS_PHY_894[25-16] PHY_RDDQS_DQ5_RISE_SLAVE_DELAY_3
- DDRSS_PHY_895[25-16] PHY_RDDQS_DQ6_RISE_SLAVE_DELAY_3
- DDRSS_PHY_896[25-16] PHY_RDDQS_DQ7_RISE_SLAVE_DELAY_3
- DDRSS_PHY_122[9-0] PHY_RDDQS_DQ0_FALL_SLAVE_DELAY_0
- DDRSS_PHY_123[9-0] PHY_RDDQS_DQ1_FALL_SLAVE_DELAY_0
- DDRSS_PHY_124[9-0] PHY_RDDQS_DQ2_FALL_SLAVE_DELAY_0
- DDRSS_PHY_125[9-0] PHY_RDDQS_DQ3_FALL_SLAVE_DELAY_0
- DDRSS_PHY_126[9-0] PHY_RDDQS_DQ4_FALL_SLAVE_DELAY_0
- DDRSS_PHY_127[9-0] PHY_RDDQS_DQ5_FALL_SLAVE_DELAY_0
- DDRSS_PHY_128[9-0] PHY_RDDQS_DQ6_FALL_SLAVE_DELAY_0
- DDRSS_PHY_129[9-0] PHY_RDDQS_DQ7_FALL_SLAVE_DELAY_0
- DDRSS_PHY_378[9-0] PHY_RDDQS_DQ0_FALL_SLAVE_DELAY_1
- DDRSS_PHY_379[9-0] PHY_RDDQS_DQ1_FALL_SLAVE_DELAY_1
- DDRSS_PHY_380[9-0] PHY_RDDQS_DQ2_FALL_SLAVE_DELAY_1
- DDRSS_PHY_381[9-0] PHY_RDDQS_DQ3_FALL_SLAVE_DELAY_1
- DDRSS_PHY_382[9-0] PHY_RDDQS_DQ4_FALL_SLAVE_DELAY_1
- DDRSS_PHY_383[9-0] PHY_RDDQS_DQ5_FALL_SLAVE_DELAY_1
- DDRSS_PHY_384[9-0] PHY_RDDQS_DQ6_FALL_SLAVE_DELAY_1
- DDRSS_PHY_385[9-0] PHY_RDDQS_DQ7_FALL_SLAVE_DELAY_1
- DDRSS_PHY_634[9-0] PHY_RDDQS_DQ0_FALL_SLAVE_DELAY_2
- DDRSS_PHY_635[9-0] PHY_RDDQS_DQ1_FALL_SLAVE_DELAY_2
- DDRSS_PHY_636[9-0] PHY_RDDQS_DQ2_FALL_SLAVE_DELAY_2
- DDRSS_PHY_637[9-0] PHY_RDDQS_DQ3_FALL_SLAVE_DELAY_2

Copyright © 2024 Texas Instruments Incorporated

- DDRSS_PHY_638[9-0] PHY_RDDQS_DQ4_FALL_SLAVE_DELAY_2
- DDRSS_PHY_639[9-0] PHY_RDDQS_DQ5_FALL_SLAVE_DELAY_2
- DDRSS_PHY_640[9-0] PHY_RDDQS_DQ6_FALL_SLAVE_DELAY_2
- DDRSS_PHY_641[9-0] PHY_RDDQS_DQ7_FALL_SLAVE_DELAY_2
- DDRSS_PHY_890[9-0] PHY_RDDQS_DQ0_FALL_SLAVE_DELAY_3
- DDRSS_PHY_891[9-0] PHY_RDDQS_DQ1_FALL_SLAVE_DELAY_3
- DDRSS_PHY_892[9-0] PHY_RDDQS_DQ2_FALL_SLAVE_DELAY_3
- DDRSS_PHY_893[9-0] PHY_RDDQS_DQ3_FALL_SLAVE_DELAY_3
- DDRSS_PHY_894[9-0] PHY_RDDQS_DQ4_FALL_SLAVE_DELAY_3
- DDRSS_PHY_895[9-0] PHY_RDDQS_DQ5_FALL_SLAVE_DELAY_3
- DDRSS_PHY_896[9-0] PHY_RDDQS_DQ6_FALL_SLAVE_DELAY_3
- DDRSS_PHY_897[9-0] PHY_RDDQS_DQ7_FALL_SLAVE_DELAY_3
- DDRSS_PHY_129[25-16] PHY_RDDQS_DM_RISE_SLAVE_DELAY_0
- DDRSS_PHY_385[25-16] PHY_RDDQS_DM_RISE_SLAVE_DELAY_1
- DDRSS_PHY_641[25-16] PHY_RDDQS_DM_RISE_SLAVE_DELAY_2
- DDRSS_PHY_897[25-16] PHY_RDDQS_DM_RISE_SLAVE_DELAY_3
- DDRSS_PHY_130[9-0] PHY_RDDQS_DM_FALL_SLAVE_DELAY_0
- DDRSS_PHY_386[9-0] PHY_RDDQS_DM_FALL_SLAVE_DELAY_1
- DDRSS_PHY_642[9-0] PHY_RDDQS_DM_FALL_SLAVE_DELAY_2
- DDRSS_PHY_898[9-0] PHY_RDDQS_DM_FALL_SLAVE_DELAY_3

Read data eye training for LPDDR4 devices does return data on the DM pin so the DBI data is trained along with the other DQ pins. In this case, the PHY_RDDQS_DM_RISE_SLAVE_DELAY_x and PHY_RDDQS_DM_FALL_SLAVE_DELAY_x fields receive their own unique values.

After read data eye training is complete, no error condition is noted in the read leveling status fields. The RISE and FALL field results in the previously described registers can be checked to look for values that appear out of place (for example, near 0x000, near 0x100 or greater) or to look for a DQ that is much different than the rest. After read data eye training is complete, software can always override the results by writing directly to the PHY_RDDQS_DQy_RISE_SLAVE_DELAY_x, PHY_RDDQS_DQy_FALL_SLAVE_DELAY_x, PHY_RDDQS_DM_RISE_SLAVE_DELAY_x and PHY_RDDQS_DM_FALL_SLAVE_DELAY_x fields.

8.2.3.7.9.4 Write DQ Training

The data that is written to the SDRAM during training is created within the data slice. The following data pattern options are available:

- Clock pattern - An alternating pattern of "1" and "0" is applied to each DQ/DM. The phase of the 1/0 pattern is identical on the DQ/DM pins.
- LFSR pattern - The same LFSR pattern that is used for data loopback can be applied to the DQ/DM pins. The LFSR is reset at the start of the pattern and each burst of data is different according to the progression of the LFSR.
- User defined pattern - A user defined pattern can also be applied to the DQ/DM pins. Independent data for each bit covering a full burst of 16 can be loaded into register fields to supply the training data. The same data pattern is repeated for every burst. The pattern is defined through the following fields:
  - DDRSS_PHY_36[31-0] PHY_USER_PATT0_0
  - DDRSS_PHY_292[31-0] PHY_USER_PATT0_1
  - DDRSS_PHY_548[31-0] PHY_USER_PATT0_2
  - DDRSS_PHY_804[31-0] PHY_USER_PATT0_3
  - DDRSS_PHY_37[31-0] PHY_USER_PATT1_0
  - DDRSS_PHY_293[31-0] PHY_USER_PATT1_1
  - DDRSS_PHY_549[31-0] PHY_USER_PATT1_2
  - DDRSS_PHY_805[31-0] PHY_USER_PATT1_3
  - DDRSS_PHY_38[31-0] PHY_USER_PATT2_0
  - DDRSS_PHY_294[31-0] PHY_USER_PATT2_1
  - DDRSS_PHY_550[31-0] PHY_USER_PATT2_2
  - DDRSS_PHY_806[31-0] PHY_USER_PATT2_3

Copyright © 2024 Texas Instruments Incorporated

- DDRSS_PHY_39[31-0] PHY_USER_PATT3_0
- DDRSS_PHY_295[31-0] PHY_USER_PATT3_1
- DDRSS_PHY_551[31-0] PHY_USER_PATT3_2
- DDRSS_PHY_807[31-0] PHY_USER_PATT3_3
- DDRSS_PHY_40[15-0] PHY_USER_PATT4_0
- DDRSS_PHY_296[15-0] PHY_USER_PATT4_1
- DDRSS_PHY_552[15-0] PHY_USER_PATT4_2
- DDRSS_PHY_808[15-0] PHY_USER_PATT4_3

These patterns can be used in any combination and if multiple patterns are requested, the final results are based on the largest leading edge found and the smallest trailing edge found across the patterns selected.

8.2.3.7.9.5 CA Training

CA bits can be swizzled between any bit positions via the DDRSS_PHY_1053[23-0] PHY_ADR_ADDR_SEL_0 field.

When the memory subsystem consists of more than one LPDDR device on the full bus width (two 16 bit devices on a 32 bit bus), all devices receive the same CA bits at some timing relative to each other depending on board placement and routing. In these cases, the DDR PHY can only supply a single CA training value for all devices to use together. Since each device may not receive ideal timing, the CA timing margin is reduced in the overall memory subsystem. Either a single one (any of them) or any combination of the devices participating in the training can be individually selected via the DDRSS_PHY_1357[20-16] PHY_CALVL_DEVICE_MAP field.

In case of two channels, each one can be assigned to a specific chip select and data slices. The DDRSS_PHY_1293[31-24] PHY_CALVL_CS_MAP field is used to map each CS rank to the appropriate DQ slice for training data return.

Bit 1 of the DDRSS_PHY_1038[25-24] PHY_ADR_CALVL_RANK_CTRL_0 field enables current training results to be taken into account along with new training results, instead of discarding previous results. This can be used to train each rank or channel and construct an aggregate for all of them. Each rank or channel then receives the same CA training timing that is the best case possible for all of them. This may result in the setup/hold margin for the CA bits being narrower than any single rank or channel could achieve by itself.

Note

When multiple frequency sets are supported, then all ranks for a given frequency set must be trained before changing to a new frequency. Failure to do so results in rank aggregation not being performed completely with possible data corruption due to rank training being improper.

In case of multiple channels each rank of a channel must use the same CA swizzle. Each rank of a channel must also use the same DQ swizzle.

If the memory device DQ bits are not received in the same order at the PHY as the LPDDR device sent them, then the DQ data has been swizzled, and must be reordered back to its original form before CA training can interpret the results. The following fields specify the DQ swizzling in the corresponding data slice:

- DDRSS_PHY_114[31-0] PHY_DQ_DM_SWIZZLE0_0
- DDRSS_PHY_370[31-0] PHY_DQ_DM_SWIZZLE0_1
- DDRSS_PHY_626[31-0] PHY_DQ_DM_SWIZZLE0_2
- DDRSS_PHY_882[31-0] PHY_DQ_DM_SWIZZLE0_3
- DDRSS_PHY_115[3-0] PHY_DQ_DM_SWIZZLE1_0
- DDRSS_PHY_371[3-0] PHY_DQ_DM_SWIZZLE1_1
- DDRSS_PHY_627[3-0] PHY_DQ_DM_SWIZZLE1_2
- DDRSS_PHY_883[3-0] PHY_DQ_DM_SWIZZLE1_3

The PHY_DQ_DM_SWIZZLE0_x fields contain the bit position to map the received read DQ bits for proper comparison with the training pattern bits.

After CA training is complete the DDRSS_PHY_1043[31-0] PHY_ADR_CALVL_OBS1_0 field can be read to obtain training status information.

Copyright © 2024 Texas Instruments Incorporated

After CA training is complete, software can always override the results by writing to the following fields:

- DDRSS_PHY_1065[18-8] PHY_ADR0_CLK_WR_SLAVE_DELAY_0
- DDRSS_PHY_1066[10-0] PHY_ADR1_CLK_WR_SLAVE_DELAY_0
- DDRSS_PHY_1067[10-0] PHY_ADR2_CLK_WR_SLAVE_DELAY_0
- DDRSS_PHY_1068[10-0] PHY_ADR3_CLK_WR_SLAVE_DELAY_0
- DDRSS_PHY_1069[10-0] PHY_ADR4_CLK_WR_SLAVE_DELAY_0
- DDRSS_PHY_1070[10-0] PHY_ADR5_CLK_WR_SLAVE_DELAY_0

Note

The PHY_ADRx_CLK_WR_SLAVE_DELAY_0 fields can be programmed between 0x000 and 0x600. While training is permitted to set these fields below 0x0C0, the final result is always greater than 0x0C0. If software overrides training results, values less than 0x0C0 should never be programmed. The same restriction applies also to the PHY_GRPy_SLAVE_DELAY_x fields.

8.2.3.7.9.6 CS Training

When the CS training algorithm has completed, the final CS slave delay settings can be found in the following fields:

- DDRSS_PHY_1399[10-0] PHY_GRP0_SLAVE_DELAY_0
- DDRSS_PHY_1399[26-16] PHY_GRP1_SLAVE_DELAY_0
- DDRSS_PHY_1400[10-0] PHY_GRP2_SLAVE_DELAY_0
- DDRSS_PHY_1400[26-16] PHY_GRP3_SLAVE_DELAY_0
- DDRSS_PHY_1401[10-0] PHY_GRP0_SLAVE_DELAY_1
- DDRSS_PHY_1402[10-0] PHY_GRP1_SLAVE_DELAY_1
- DDRSS_PHY_1403[10-0] PHY_GRP2_SLAVE_DELAY_1
- DDRSS_PHY_1404[10-0] PHY_GRP3_SLAVE_DELAY_1

After CS training is complete, the DDRSS_PHY_1289[31-0] PHY_CSLVL_OBS1 field can be read to obtain training status information.

After CS training is complete, software can always override the results by writing directly to the PHY_GRPy_SLAVE_DELAY_x fields.

8.2.3.7.10 Data Bus Inversion (DBI)

Data bus inversion is supported in the DDR PHY. The PHY only passes the DBI information between the DDR controller and SDRAM devices. It does not perform DBI generation and decode. If the DDRSS_PHY_102[8] PHY_DBI_MODE_0 bit is set to 0x1 the SDRAM DBI information for slice 0 is passed to the DDR controller. The following bits are associated with the other slices:

- DDRSS_PHY_358[8] PHY_DBI_MODE_1
- DDRSS_PHY_614[8] PHY_DBI_MODE_2
- DDRSS_PHY_870[8] PHY_DBI_MODE_3

8.2.3.7.11 I/O Pad Calibration

I/O pad calibration runs at initialization time by default but it can also be programmed to operate in a background mode after initialization. Bits [3:1] of the DDRSS_PHY_1330[12-0] PHY_CAL_MODE_0 field enable the periodic calibration mechanism and determine the base clock frequency of the interval counter. The DDRSS_PHY_1331[31-0] PHY_CAL_INTERVAL_COUNT_0 field then sets a timer to indicate how frequently calibration is performed. When the timer expires, the calibration runs in background, independent of other traffic running through the PHY. When the calibration is complete, the updated PVTP/PVTN/PVTR codes are transferred to the PHY I/O cells. Whether or not the update of the memory clock PVTP/PVTN/PVTR codes is applied at the same time as the remaining I/O cells is controlled through the DDRSS_PHY_1298[8] PHY_CONTINUOUS_CLK_CAL_UPDATE bit. If the automatic update of the memory clock codes is disabled through the PHY_CONTINUOUS_CLK_CAL_UPDATE bit, the DDRSS_PHY_1298[0] SC_PHY_UPDATE_CLK_CAL_VALUES bit can be used to trigger a manual update.

Copyright © 2024 Texas Instruments Incorporated

8.2.3.7.12 DQS Error

A DQS error results from a mismatch between the expected number of read DQS pulses and the actual number of read DQS pulses. Counters exist at DDR PHY level to track the number of DQS errors in each data slice. Counter information can be obtained by reading the following fields:

- DDRSS_PHY_1363[31-0] PHY_DS0_DQS_ERR_COUNTER
- DDRSS_PHY_1364[31-0] PHY_DS1_DQS_ERR_COUNTER
- DDRSS_PHY_1365[31-0] PHY_DS2_DQS_ERR_COUNTER
- DDRSS_PHY_1366[31-0] PHY_DS3_DQS_ERR_COUNTER

The DQS error signals from each data slice are accumulated at PHY level and can be used to generate a dfi_error signal. Bit 2 of the DDRSS_PHY_1361[26-24] PHY_ERR_MASK_EN field is used to mask the DQS error from generating a dfi_error or being reported on dfi_error_ info[2].

8.2.3.8 PI Functional Description

The PHY Independent (PI) module is part of the DDR PHY and can perform PHY independent leveling without involvement of the DDR controller. The PI can also initialize the SDRAM independent of the DDR controller. The PI module provides an interface between the PHY core logic and the DDR controller.

8.2.3.8.1 PI Initialization

The PI module can be used to perform SDRAM initialization before training instead of the DDR controller. The following steps represent the main actions of the initialization process:

1. Set to 0x1 the DDRSS_PI_139[8] PI_DRAM_INIT_EN bit.
2. Enable the PI by setting to 0x1 the DDRSS_PI_0[0] PI_START bit.
3. Enable the DDR controller.
4. The PI, DDR controller and PHY core are involved in a process of specific hardware communication between them that does not require software intervention.
5. The PI starts the power-up sequence which includes the SDRAM initialization timing, the SDRAM mode register programming and training.
6. After completion of the power-up sequence, the PI notifies the DDR controller through the dfi_init_complete signal.
7. The DDR controller can now access the SDRAM.

8.2.4 DDRSS Registers

Note

Refer to the DDRSS Registers section of the DRA821 TRM (https://www.ti.com/lit/pdf/spruiu1) for register offset information to be combined with the base address listed below (where applicable for this Device):

- DDRSS0
  - Sub-system registers: 0x0298 0000
  - Controller registers: 0x0299 0000
  - PI registers: 0x0299 2000
  - PHY registers: 0x0299 4000
- DDRSS1
  - Sub-system registers: 0x029A 0000
  - Controller registers: 0x029B 0000
  - PI registers: 0x029B 2000
  - PHY registers: 0x029B 4000

Copyright © 2024 Texas Instruments Incorporated

8.3 Peripheral Virtualization Unit (PVU)

This chapter describes the PVU module, part of the main NAVSS.

8.3.1 PVU Overview

The Peripheral Virtualization Unit (PVU) module provides TLBs (translation look-aside buffers) for a static virtual address translation on a CBA VBUSM bus.

Figure 8-9 shows a top-level overview of the PVU module.

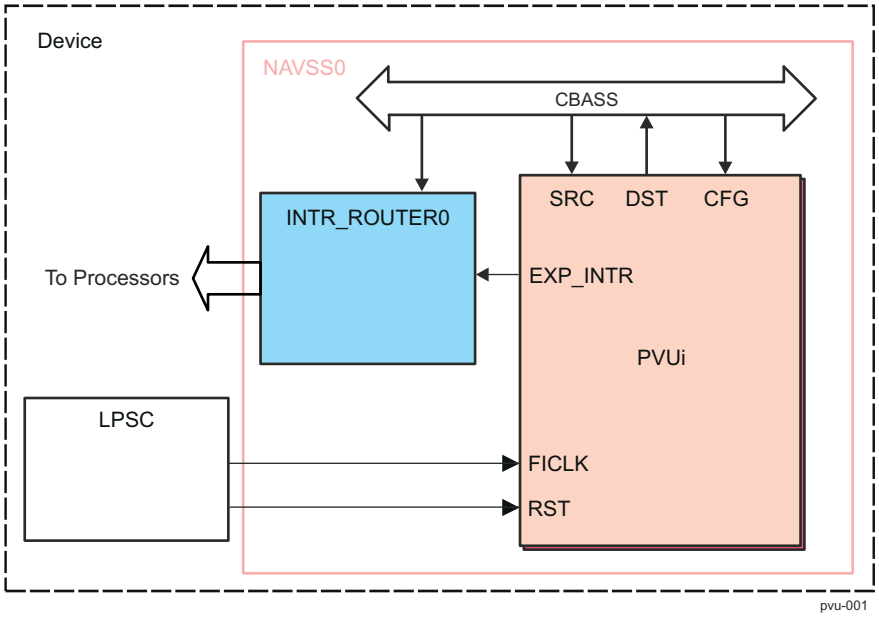


Figure 8-9. PVU Overview

8.3.1.1 PVU Features

Each PVU supports the following features:

- Implements a channelized TLB for virtual address translation
- Supports a pre-configured number of virtIDs (channels). See Section 8.3.1.2.
- Supports mapping of virtIDs to TLBs with a programmable number of DMA channel support
- Supports a pre-configured number of entries per channel of the TLB. See Section 8.3.1.2
- Supports TLB chaining to extend the search but at a latency penalty
- Supports a 48-bit address size
- Supports page sizes of 4 KB, 16 KB, 64 KB, 2 MB, 32 MB, 512 MB, 1 GB, and 16 GB
- Support TLBs in software mode, where they are maintained by software only
- Produces a fault interrupt when the TLB misses or upon a permission error.

8.3.1.2 PVU Parameters

shows the PVU configuration parameters set during SoC design.

Copyright © 2024 Texas Instruments Incorporated

8.3.2 PVU Functional Description

8.3.2.1 Functional Operation Overview

The Peripheral Virtualization Unit, PVU, module provides TLBs, translation look-aside buffers, for a static virtual address translation on a CBA VBUSM bus. This is in contrast to a dynamic translation TLB that works with an MMU, such as the Arm® processor SMMU. The module acts as a VBUSM retiming bridge that also performs the address translation. Each of the TLBs has its own set of private entries. It is a static translation because all the entries must be programmed by software and there is no hardware to fetch new translation entries when a transaction does not match any of the provided entries.

The entire PVU can be enabled or disabled via the PVU_ENABLE register.

8.3.2.2 PVU Channels

The module supports a number of channels (see Section 8.3.1.2), each having its own independent TLB and translation entries. This allows a single PVU to handle the translations for many masters, as well as keep their translations from interfering with each other.

The mapping of transactions to TLBs is made using mapping registers (PVU_VIRTID_MAP1 and PVU_VIRTID_MAP2). Each channel can be mapped to a virtID, or as a sub range of a DMA virtID. This allows the software to decide how the transactions are mapped to TLBs. It can be purely based on virtID, so that each virtual space has its own TLB. Or, it can be partly based on a highly channelized master, such as a DMA, can have 4 TLBs dedicated for channels where the sub-class on the upper chanid bits determine which of the 4 TLBs to use. This DMA mode allows for different classes of DMA traffic, such as descriptors or buffers, to be in their own TLBs and not interfere with each other. Since the TLB mapping is relatively simple, it is software's responsibility to control the definition of virtIDs in peripherals and DMAs so that they map to TLBs as required.

8.3.2.3 TLB

Each PVU channel contains a single traditional TLB. The TLB contains a number of entries that store the translation recipes, and these entries are used only for this TLB. Any activity for other TLBs or PVU channels will not impact this TLB.

8.3.2.4 TLB Entry

A TLB entry consists of the virtual address base and LPAE (Large Physical Address Extension) fields defined by Arm® architecture, stored in TLB Entry registers. This entry then includes the translated base address, size of the page, and other attributes of the page. When written by software, these fields must be written in the same way for correct operation.

The base addresses must be aligned to the defined page size.

8.3.2.5 TLB Selection

The first step of the module is to map the transaction to a specific TLB by comparing the virtID, chanID upper bits, and those programmed in PVU_VIRTID_MAP1 and PVU_VIRTID_MAP2. This mapping determines which TLB to select from the virtID. That TLB is used for the remainder of the pipeline. If there is no match then the PVU automatically misses.

8.3.2.6 DMA Classes

The first N virtIDs can be reserved for DMA class usage in the PVU_VIRTID_MAP1 register. These virtIDs map to TLBs along with classes that are defined in the chanID bits 10 to 11 on the bus. This creates four classes per virtID, that can select from four TLBs, allowing the transaction to restrict its translations to smaller sets from known distinct memory segments (such as the DMA accessing descriptors or buffers). A set of bitfields define the class mapping, which defines the final class value to use for an input class, and can be used to restrict the number of classes if all four are not used (by mapping 2 to 0 and 3 to 1 for example to limit to two classes). Then the final TLB selection is calculated as:

virtID × 4 + final class value

The unused classes cause unused TLBs as entry points, but they can be used in chaining.

Copyright © 2024 Texas Instruments Incorporated

8.3.2.7 General virtIDs

After the first N virtIDs for DMA class usage, the remaining virtIDs up to the maximum allowed (as set in PVU_VIRTID_MAP2) are for general usage. Each of these map one to one to a TLB, where the calculation is:

dma_cnt × 4 + (virtID - dma_cnt)

There are no classes used for this range of virtIDs. An input virtID that is beyond the MAX_CNT bitfield value will produce an error.

8.3.2.8 TLB Lookup

The next step of the module is to check the input address, which uses the bits just above the 4-kB size, bits 12 and up, as the virtual address base. This is compared against all the TLB entries for that TLB, given the size in each entry. The size affects which address bits need comparing for an entry. The minimum size of 4 kB, so address bits 12 and up are compared. But if the size is 64 kB, then bits 16 and up are compared, as the lower bits fall into the size of the page. In addition to the address, the psecure bit is used to match against the secure level of the transaction. Only secure transaction are allowed to match an entry with psecure set, while only non-secure transactions are allowed to match an entry with psecure clear. If the entry is valid and the base compare matches and the secure level matches then the entry is used for the translation and checks.

8.3.2.9 TLB Miss

If the TLB lookup results in no matches and there is no chaining to another TLB (described later), then this is called a TLB miss. If the TLB misses, since there is no back-fill (such as from an MMU), the result is that the translation fails and will fault (a fault is when there is no physical address available). The transaction will be marked as in error with the flush signal, and a later slave will perform the flush that will return a bus error and read data as zeros. The PVU will record the fault as an interrupt and log the fault in the exception logging read-only registers.

8.3.2.10 Multiple Matching Entries

The PVU does not support multiple entries in a TLB that match the same virtual address. This is an illegal condition for software to program, as there is no mechanism to select one over the other. The PVU will have unpredictable results if this occurs, with the one requirement that the input transactions will never transition from a non-secure attribute to a secure attribute even in this illegal condition. This means that there is no support for background and foreground translating entries in the same TLB (which would be the result if a real MMU were used also). If software user wants to use a foreground and background method (that is not MMU compliant), he can place the foreground entries in earlier TLBs that are chained together, and the background entries are in the later TLBs, so that no single TLB has overlapping entries. The background entry would only be hit when the address falls within the background range but not any of the foreground ranges.

8.3.2.11 TLB Disable

A special case is when a TLB should go through a lookup but is disabled through the PVU_CHAIN_j[31] EN bit, or the entire PVU is disabled through the PVU_ENABLE[0] EN bit. If a lookup is requested when the TLB or PVU is disabled, then the transaction will pass and no translation will occur (output address is the same as the input address). Also, if the atype=1 then it will change to atype=0 on the output so that the resulting bypass transaction does not get routed right back to the PVU. Any other atype value will be maintained on the output. Any entries set in the disabled TLB are ignored as well as the chaining setting (described in Section 8.3.2.12) is also ignored, so once a disabled TLB is used then the chain is ended. Software must take care to not include any disabled TLBs in chains that must be fully searched – if a TLB must be disabled then the chain must be updated first to skip over that TLB and any virtIDs that directly map to that TLB initially must not be used.

8.3.2.12 TLB Chaining

If the TLB has a chain enabled by writing a non-zero chain value, then the TLB lookup will continue for the transaction using the chained TLB number. There is a latency penalty to resume the lookup at the chained TLB, but it allows more TLB resources to be used for virtIDs that might need more translations. If a TLB misses and has a chain value of zero then the overall result is a miss. Any TLB logs are always applied to the original TLB of the chain, and not the chained TLB.

TLB chaining can be defined in the PVU_CHAIN_j[11:0] CHAIN register bitfield.

Copyright © 2024 Texas Instruments Incorporated

8.3.2.13 TLB Permissions

The TLB entry contains permissions for the page that must be checked before the translation can be used. These include if the page has *supervisor* access for read, write, or execute, and has *user* access for read, write, or execute. The supervisor or user is based on the transaction priv[0] signal (0 = user, 1 = supervisor). The access type is based on the transaction dtype[0] and dir signals (dtype[0] = 0 and dir = 1 is a read, dtype[0] = 0 and dir = 0 is a write, dtype[0] = 1 and dir = 1 is an execute). The transaction signals are checked against these permissions. If they are allowed then the translation continues. If the page does not allow this transaction then the check fails and flush signal set is on the bus, like for a firewall, indicating that the transaction is no longer valid and must return an error. PVU does not perform the error signaling, so a later module must support it. A fault interrupt is produced.

The pperm entry bits define four additional checks. If pperm[0] is clear then user privilege access is not allowed, and a matching transaction with the CBA bus priv signal set to 0 (indicating user access) will result in an error. If pperm[1] is set then write access is not allowed, and a matching transaction that is a write will result in an error. If pperm[2] is set then instruction execute is not allowed, and a matching transaction that has the CBA bus dtype signal set to 1 (instruction fetch) will result in an error. If pperm[3] is set then supervisor instruction execute is not allowed, and a matching transaction that has the CBA bus priv signal non-zero (above user) and the CBA bus dtype signal set 1 (instruction fetch) will result in an error. If all these checks pass then the translation is allowed.

The pprefetch entry bit determines whether the translation allows prefetch transactions. If set, then any transaction with the pfable signal set are allowed. But if pprefetch=0 then those transactions with pfable set will fault as prefetch is defined as not supported.

8.3.2.14 Translation

If all the checks pass, then the translated address is replaced and the transaction can proceed to the output. Only the address bits above the page size are replaced, and all the lower bits are copied from the transaction. The memory attributes are also replaced on the transaction. These attributes include the memory type, shareability, and allocation policies. In addition, the prefetchable request is only copied if the page allows prefetch, and otherwise it is forced clear, but the transaction is still allowed to continue (just the prefetch will not occur).

8.3.2.15 Memory Attributes

Table 8-21 shows the mapping of page attributes to the bus memory attributes.

Table 8-21. Page Attributes to Bus Memory Attributes Mapping

Bus Signal	Page Attribute	Translated Bus Signal
cinner	piallocpol	cinner = piallocpol
couter	poallocpol	couter = poallocpol
memtype	pmemtype	memtype = pmemtype
sdomain	posable, pisable	sdomain[1] = posable, sdomain[0] = pisable
pable	pprefetch	pable = pable AND pprefetch

8.3.2.16 Faulted Transactions

A faulted transaction will have the flush bit set on the outgoing transaction so that it will return errors by the interconnect. The atype will be set to 0 since the memory attributes have not been replaced. And the outgoing transaction address will have the upper bits set to the fault_addr parameter value (see Section 8.3.1.2), which defines an interconnect address that will always return errors and not interfere with any valid slave traffic (such as to memory).

8.3.2.17 Non-Virtual Transactions

The module must not receive a transaction that does not have the atype set to an intermediate address, but if it does, there is no translation performed for that transaction. The transaction is simply copied to the output without modification.

Copyright © 2024 Texas Instruments Incorporated

8.3.2.18 Allowed virtIDs

Software can control the limit of usable virtIDs in the PVU by PVU_VIRTID_MAP2[11:0] MAX_CNT register. This sets the maximum virtID allowed value. If the PVU receives a transaction with a higher virtID signal, then the transaction will be treated as a fault and error, and no TLB lookup will be performed.

8.3.2.19 Software Control

The software can control all the TLB entries directly. The TLB can be programmed at any time by software, and writes to the TLB take precedence over lookups. This allows the update to be reflected as soon as possible in the translations. But there is no guarantee about the timing of the update against any bus transactions, so the only way to guarantee an endpoint uses the updated table is to stall the endpoint until the update has been completed (using the write status). This allows software to directly populate TLB entries for known translation as well as locking entries down so that they cannot be replaced.

8.3.2.20 Fault Logging

The module will log faults into exception logging registers. It will capture that type of fault, the address that faulted, and the attributes of the transaction that faulted. It will also set an interrupt. The module can only store one fault. When the software clears the interrupt status then the fault is also cleared and another can be captured. If another fault occurs while one is pending then that fault log is lost.

There is a logging Enable/Disable bit per TLB and if disabled then faults from that TLB will not be logged. PVU_EXCEPTION_LOGGING_DISABLE register allows to disable types of faults, and when disabled that type of fault will not be logged. There are additional fault status fields per TLB that capture fault status that would have been logged but could not because another fault was already logged. These status bits stay set until cleared by software.

A TLB fault that occurs after chaining will set the fault status of the original TLB, and not of the last TLB in the chain.

8.3.2.21 Alignment Restrictions

All accesses through the module must not cross 4-kB page boundaries. The module does not perform any checks for this requirement.

Both the input and output addresses must be aligned with the configured size for the TLB.

Copyright © 2024 Texas Instruments Incorporated

8.4 Region-based Address Translation (RAT) Module

This section describes the RAT functionality.

8.4.1 RAT Functional Description

8.4.1.1 RAT Operation

The RAT module performs a region based address translation. It translates a 32-bit input address into a 48-bit output address. Any input transaction that starts inside of a programmed region will have its address translated, if the region is enabled. Any disabled region is ignored from the translation lookup.

RAT has 16 regions, with each region having dedicated registers that define its attributes:

- Base address of the region, via the RAT_BASE_k register.
- Size of the region, via the RAT_CTRL_k register.
- Translated base address (48-bit). It is defined by 32-bit lower address via the RAT_TRANS_L_k register, and 16-bit upper address via the RAT_TRANS_U_k register.

The input addresses are compared against all the enabled regions in parallel. The region size defines how many of the lowest address bits are included in the region space, starting from a 1B space to a 4GB space. Then the upper bits not part of the region are used for the lookup and comparison, matching those bits of the input addresses against the region base address. The region matches when it is enabled and the compare matches. The first region that matches has the output address set with the region translated base address upper bits. The lower bits that are part of the region size are copied from the input address to the output address. If there are no region matches then the input address is copied to the output address entirely.

RAT does not automatically correct transactions that cross region boundaries, that is, start inside of a region and end outside of it, or start outside of a region and end inside of it. If a boundary crossing transaction is detected, then RAT annuls the transaction as illegal, generates an interrupt and logs an error, as explained in *RAT Error Logging*.

Note

The region base address and translated base address must be aligned to the defined region size. For example, if the defined region size is 64KB, then the two addresses must be 64KB aligned. This is software responsibility as RAT does not perform such alignment check. Regions that are not aligned have unpredictable results.

Moreover, multiple region definitions must not overlap in their covered address space. RAT does not check for this, so it is software responsibility to take care of it. Overlapping regions may lead to unpredictable results.

8.4.1.2 RAT Error Logging

The error log consists of a series of registers that capture the details of the transaction, including the input address that caused the error. These registers are the following:

- RAT_EXCEPTION_LOGGING_HEADER0 and RAT_EXCEPTION_LOGGING_HEADER1
- RAT_EXCEPTION_LOGGING_DATA0 through RAT_EXCEPTION_LOGGING_DATA3

The RAT module can capture one error before it is cleared by software. The error logging is enabled by default, but can be disabled via the RAT_EXCEPTION_LOGGING_CONTROL[0] DISABLE_F bit. Upon error logging an interrupt is also generated. To clear the log, software must either read the final error logging register, or manually clear the RAT_EXCEPTION_PEND_CLEAR[0] PEND_CLR bit by setting it to 0x1. This will clear the error status, and not the actual log registers, but it does allow the next error to be captured into the log registers. If the status is not cleared and additional errors are detected, they are not logged.

After an error occurs and is cleared (whether by reading the final error logging register or by clearing the status bit), the RAT_EOI_REG register must be written to guarantee the next interrupt pulse will be produced.

See the RAT section of the Module Integration chapter of this document for the RAT Source ID mapping which is associated with the EXCEPTION_LOGGING_HEADER0[23-8] SRC_ID field for each RAT module.

Copyright © 2024 Texas Instruments Incorporated

9 Interrupts

9.1 Interrupt Architecture

The SoC has many peripherals and a large number of event sources (interrupt, time sync or DMA). The use of events is completely dependent on a user's specific application, which drives a need for maximum flexibility in which event sources are used in the system. It is also completely up to software control as to how the events are serviced.

The SoC includes the following interrupt servicing modules (hosts):

- Security Management Subsystem (WKUP_SMS0):
  - 2 Arm Cortex-M4F cores
  - 2 Nested vectored interrupt controller (NVIC)
- Compute cluster:
  - Three processor subsystems:
    - Dual-core Arm Cortex-A72 microprocessor unit (A72SS0)
    - 2 Single-core C71x DSP subsystems (C71SS0, C71SS1)
  - Two interrupt controllers, both integrated inside the MSMC wrapper:
    - Arm generic interrupt controller (GIC-500) supports both cores in the dual-A72 cluster
    - Cluster level event controller (CLEC) supports the C71x DSPs
- Microcontroller units (R5FSS0, R5FSS1, MCU_R5FSS0), each implementing:
  - Two Arm Cortex-R5F cores
  - Vectored interrupt manager (VIM)

Most of the system events are routed directly to the various proccesing elements but in some cases it is impractical to route all events of a certain group (for example, GPIO events) to each processing element. For this purpose, the SoC integrates several interrupt router (INTRTR) instances. Each interrupt router aggregates a number of system events and can route each event to a given processing element by using simple combinational logic (a set of multiplexors). Event selection is controlled through the associated registers within each interrupt router.

The following interrupt router instances are part of the SoC interrupt architecture:

- WKUP domain GPIO interrupt router (WKUP_GPIOMUX_INTRTR0):
  - Provides selection of active module interrupts for the WKUP_GPIOx modules
- MAIN domain GPIO interrupt router (GPIOMUX_INTRTR0):
  - Provides selection of active module interrupts for the MAIN domain GPIOx modules
- MAIN-to-MCU domain level interrupt router (MAIN2MCU_LVL_INTRTR0):
  - Provides selection of active MAIN domain module *level* interrupts for routing to MCU domain processing elements
  - Reduces the number of event voltage domain crossings
- MAIN-to-MCU domain pulse interrupt router (MAIN2MCU_PLS_INTRTR0):
  - Provides selection of active MAIN domain module *pulse* interrupts for routing to MCU domain processing elements
  - Reduces the number of event voltage domain crossings

In addition, the following interrupt router instances are part of the SoC time sync architecture:

- Time sync event router (TIMESYNC_INTRTR0):
  - Provides selection of active common platform time synchronization (CPTS) events for routing to CPTS capable modules
- Compare event router (CMPEVT_INTRTR0):
  - Provides selection of active CPTS counter compare events for routing as processor interrupts or DMA events

For more details on time sync routers, see *Time Sync Routers*.

Copyright © 2024 Texas Instruments Incorporated

9.2 Interrupt Controllers

9.2.1 Generic Interrupt Controller (GIC)

This section describes the Generic Interrupt Controller (GIC) module in the device.

9.2.1.1 GIC Overview

The device GIC is a TI module that is based on the Arm GIC-500 interrupt controller. The Arm GIC-500 is a high-performance, area-optimized, built-time configurable interrupt controller which detects, manages, and distributes system interrupts to the Arm® Cortex®-A72 processors in the Compute Cluster.

The Arm GIC-500 is compliant to the Arm GICv3 standard. It only supports cores that implement the Armv8 architecture and the GIC CPU interface with the standard GIC Stream Protocol Interface (such as A72).

The GIC includes additional logic (TI wrapper) to fully integrate the Arm GIC-500 into the SoC.

The GIC module is located inside the MSMC_WRAP logical block in the Compute Cluster, along with the cluster level interrupt controller (CLEC), which serves the C71x DSPs.

9.2.1.1.1 GIC Features

Note

This chapter provides only a brief description of the GIC functionality and features. For more details on this module, refer to the *Arm®CoreLink™ GIC-500 Generic Interrupt Controller Technical Reference Manual*.

The Arm GIC-500 supports the following features in the device:

- Module revision: r1p1
- Supports all A72 cores
- 16 software generated interrupts (SGIs) per core
- 16 private peripheral interrupts (PPIs) per core
- 960 shared peripheral interrupts (SPIs)
- Locality-specific peripheral interrupts (LPIs), used for message-based interrupts
- Interrupt translation service (ITS)
  - Device isolation
  - ID translation for message-based interrupts
  - This allows virtual machines to program devices directly
- Interrupt masking and prioritization
- Programmable, affinity-based interrupt routing
- 32 priorities for each interrupt
- Three different interrupt groups
  - Group 0
  - Non-secure group 1
  - Secure group 1

Additionally, the GIC wrapper logic provides the following features:

- Synchronizes interrupt inputs to GIC clock
- Implements AXI2VBUSM and VBUSM2AXI bridges
  - Used for bus protocol conversion (AXI-to-VBUSM and VBUSM-to-AXI, respectively)
- Implements an integrated ECC aggregator
  - Only used to inject errors (for testing purposes)

9.2.1.1.2 GIC Not Supported Features

The GIC does not support the following features:

- GICv2 backwards compatibility
- AWID-based LPI mapping

Copyright © 2024 Texas Instruments Incorporated

9.2.1.2 GIC Functional Description

Note

This chapter provides only a brief description of the GIC functionality and features. For more details on this module, refer to the *Arm® CoreLink™ GIC-500 Generic Interrupt Controller Technical Reference Manual*.

9.2.1.2.1 Arm GIC-500

The Arm GIC-500 is responsible for detecting, managing and communicating system interrupts to all A72 cores in the device. The main difference between the Arm GIC-500 and previous Arm GIC versions is the direct communication with the CPU(s). Previously, an interrupt controller would set a pending bit to a CPU and that CPU would then have to query an interrupt controller MMR to find out what to do. Now, the Arm GIC-500 sends interrupts to a CPU via a dedicated message interface and the CPU communicates back about interrupts through the same interface. The CPU now uses writes and reads to system registers instead of over the memory interface. This leads to reduced latency and the ability to route interrupts to different CPUs based on a set of rules.

The Arm GIC-500 is divided into three main sections:

- **Distributor**: The Distributor receives interrupts from the wire interrupts and the programming interface. It is responsible for prioritizing these interrupts and sending them to the CPU interface via the GIC Stream Protocol Interface.
- **Redistributor**: There is one Redistributor per core. Each Redistributor holds the state that is individual to a particular core (such as the settings for PPIs and SGIs). It also stores the LPIs for that core after they have been generated using the ITS.
- **ITS**: The ITS is responsible for translating message-based interrupts from device peripherals into LPIs. The user can also use the ITS to manage existing LPIs. Note that the ITS is not used for other types of interrupt

9.2.1.2.2 GIC Interrupt Types

The GIC supports four types of interrupts:

- *Software Generated Interrupts (SGIs)*: There are 16 SGIs (ID0-ID15). These are inter-processor interrupts. SGIs can be sent using either Arm system registers or by writing to the Software Generated Interrupt Register (GICD_SGIR).
- *Private Peripheral Interrupts (PPIs)*: There are 16 PPIs (ID16-ID31). These are wired interrupts dedicated to a specific CPU. Many are reserved to specific functions via convention. These can be either active-low levels (by default), or rising edge triggered (software programmable).
- *Shared Peripheral Interrupts (SPIs)*: There are 960 SPIs (ID32-ID991). These are wired interrupts that can be routed to any core or cluster, based on the programming of that interrupt in the GIC. These are active-high levels (by default), or rising edge triggered (software programmable).
- *Locality-Specific Peripheral Interrupts (LPIs)*: There are 57,344 of these. These interrupts are used for message-based interrupts from a peripheral. They are generated through writes to the GIC slave interface. The information associated with these is located in memory. The GIC keeps a small 64-entries cache on-board in order to reduce latency either for specific interrupts or for most-recently-used. LPIs can be routed to different CPUs based on programmed rules.

The mapping of PPIs and SPIs can be found in section TBD .

9.2.1.2.3 GIC Interfaces

The GIC has the following main interfaces:

- A pair of 64-bit VBUSM write-only and read-only master interfaces
  - AXI2VBUSM bridge converts the standard 64-bit AXI4 master interface into this pair of interfaces
  - The AXI4 master interface allows the GIC ITS and redistributors to access main memory
- 32-bit VBUSM slave interface
  - Provides access to Arm GIC-500 internal resources
  - Handles all message-based interrupts. Message-based interrupts can generate SPIs or LPIs, depending on the register that is written

Copyright © 2024 Texas Instruments Incorporated

  - VBUSM2AXI bridge converts this interface into an 32-bit AXI4 slave interface
- 32-bit VBUSP slave interface
  - Provides access to internal ECC aggregator
- Physical interrupt and power signals
  - Inputs: CPU active signals, SPIs and PPIs (can be programmed as either level-sensitive, or edge-triggered)
  - Outputs: Error signals and wake-up requests
- GIC Stream Protocol Interface
  - Consists of a pair of AXI4-Stream interfaces (one upstream and one downstream 16-bit AXI4-Stream interface) that the GIC uses to send interrupts to the core and receive notifications when the core activates interrupts
  - There is a pair of physical interfaces, one in each direction, for each cluster

9.2.1.2.4 GIC Interrupt Outputs

Table 9-1 lists the interrupts that are generated by the GIC.

Table 9-1. GIC Interrupt Outputs

Interrupt Name	Description
GIC_AXIM_ERR	GIC bus error interrupt. This interrupt is triggered if the GIC receives an error on a bus transaction, such as a decode or protection error. This is a pulse-high interrupt. If this interrupt occurs, the GIC might lose interrupts and must be reset. If it is not reset, behavior becomes unpredictable.
GIC_ECC_FATAL	GIC uncorrectable ECC error interrupt. Indicates an uncorrectable 2-bit error detected in one of the ITS cache memories. This is a pulse-high interrupt. If this interrupt occurs, the GIC might lose interrupts and must be reset. If it is not reset, behavior becomes unpredictable.
GIC_PWRm_WAKE_REQUEST_n	GIC wake requests for A72 cores. Asserted state indicates that an interrupt is pending for a processor that has set the PROCESSORSLEEP bit in the GICR_WAKER register. This bit indicates that the SoC should wake up the indicated CPU so that it can process the interrupt.
GIC_PWRm_WAKE_REQUEST_n	

9.2.1.2.5 GIC ECC Support

The Arm GIC-500 has built-in SECDED ECC on its memories to protect against errors. The syndrome generation and checking is done internally.

Additionally, the TI GIC wrapper integrates an ECC aggregator (GIC_ECC_AGGR) in order to allow errors to be injected for testing purposes. The generic ECC aggregator functionality is described in Section 12.10.4, *ECC Aggregator*. Note that the GIC_ECC_AGGR supports only a subset of this functionality.

9.2.1.2.6 GIC AXI2VBUSM and VBUSM2AXI Bridges

The GIC uses the AXI4 master interface to allow the ITS and redistributors to access main memory. It is expected that the main memory (attached to a CBASS VBUSM port) holds the following:

- ITS translation tables
- ITS command queue, which is written by software in order to program the ITS
- LPI configuration table, which holds the priorities and enable bits for LPIs
- LPI pending tables, which can hold information on whether each LPI is pending on each core

The AXI2VBUSM bridge converts the standard 64-bit AXI4 master interface into a pair of 64-bit VBUSM write-only and read-only master interfaces. AXI4 has a separate command channel for read and write, and the AXI2VBUSM bridge keeps them separate, so that the system may prioritize or arbitrate the traffic between them as appropriate.

The GIC uses the AXI4 slave interface to provide access to the programming interfaces of all its parts (distributor, redistributors, ITS). The VBUSM2AXI bridge converts the standard 32-bit VBUSM master interface into an 32-bit AXI4 slave interface.

Copyright © 2024 Texas Instruments Incorporated

Note

The AXI2VBUSM and VBUSM2AXI bridges do not implement any MMRs.

Copyright © 2024 Texas Instruments Incorporated

9.2.2 Cluster Level Event Controller (CLEC)

9.2.2.1 CLEC Overview

The CLEC supports the following features:

- Supports software scalable event routing mechanism
  - RAM-based interrupt event routing table
- Supports aggregation of incoming interrupt events from:
  - SoC peripherals
  - MSMC
  - DRU
  - C71SS
- Supports distribution of each aggregated interrupt event to one or more of the following:
  - SoC (could be routed to the GIC externally)
  - DRU
  - A72SS
  - C71SS
- Supports originating interrupt events through the following mechanisms:
  - MMR writes to trigger new events
  - Reporting errors detected by the CLEC
- 2048 input events
- 128 output events
- Can handle both level and pulse input interrupts
  - SoC out interrupt is always level regardless of input interrupt type (pulse or level)
- Can be used for inter-processor communication (IPC) between A72 and C71
- Shares SPI (shared peripheral interrupt) input events with GIC

Copyright © 2024 Texas Instruments Incorporated

9.2.2.2 CLEC Functional Description

9.2.2.2.1 CLEC Interrupt Event Routing

The CLEC routes events according to a RAM-based interrupt event routing table. This table holds one entry for each supported external interrupt/event number, plus some additional entries for CLEC internal events. External in this context means "external to the CLEC", and corresponds to all interrupt events arriving at the CLEC from other sources. Internal refers to events that the CLEC generates internally, or in response to an MMR write that triggers the event.

The routing table memory has 2048 entires in size. Each routing table entry contains the following details:

- Outgoing event number: Each incoming event can be mapped to any outgoing event number
- Secure / Non-secure: Each incoming event can be claimed as a secure event
- Destination port configuration: This configuration allows for both single-cast and broadcast events

For each incoming interrupt event, the CLEC does the following processing:

- Validates the interrupt event number. For out-of-range event numbers, the CLEC triggers an "invalid event number" event
- Non-secure read/write to registers associated with a secure event will generate a privilege error
- Looks up the event in the routing table
- For each port in the destination bitmap that has a bit set, it queues the event to that destination
  - Queuing to multiple destinations happens in parallel, not sequentially, to minimize latency
  - If no bits are set, the event is dropped silently, but an interrupt dropped event is asserted. This event gets routed as determined by the routing table

Note

For events mapped to DRU and ARM, only event mapping number is looked at and they are not qualified by CLEC_MRR_j[21-16] RTMAP. Hence, this bit field is "don't care" for events that have CLEC_MRR_j[15-8] EXT_EVNUM set to 128 or higher.

Since the routing table is a shared resource, the CLEC only routes one new event per cycle. The routing state machine arbitrates among all incoming event sources round-robin, with the exception that internally generated error events have priority over all other event sources.

The CLEC routes events to their destinations and also maintains pending interrupt information in an CLEC_EFR_k register. Firmware needs to clear this pending interrupt bit in an interrupt service routine (ISR).

9.2.2.2.2 CLEC Virtualization, Isolation and Access Control

The CLEC provides direct support for virtual-machine isolation via a hypervisor. Each event's register window is deliberately spaced 64KB apart. This enables allocating individual event numbers to virtual machines via stage-2 translation. The 64KB spacing allows this to work even with the largest translation granule supported by ARM (64KB).

Individual functions associated with an event within each event's window are spaced 4KB apart. This allows a standard operating system to open up, for instance, write access to an CLEC_ESR_j register for a single task, while blocking access to the event routing details in CLEC_MRR_j register.

The CLEC implements event lock registers (CLEC_GELRS / CLEC_GELRNS) that allow hypervisor to lock down the configuration of events, while still permitting write access to the event send register (CLEC_ESR_j) for that event.

In current implementation, the CLEC provides basic support for isolating virtual machines by partitioning incoming events in the address map. However, it does not provide any support for limiting event numbers assigned for outgoing events in CLEC_MRR_j[15-8] EXT_EVNUM. If such isolation is necessary, software can implement it in the hypervisor via para-virtualization calls, via trap-and-emulate for full virtualization.

Copyright © 2024 Texas Instruments Incorporated

9.2.2.2.3 CLEC Memory Protection

The CLEC implements minimal memory protection checks. It relies on protection mechanisms outside to control accesses from various masters. Chip-level firewalls control which masters may access the CLEC. MMUs provide finer-grain control for programmable processors that access the CLEC.

Specifically, the CLEC returns the following errors:

- Address error on a read or write to unimplemented space
- Address error on a read to a write-only register
- Address error on a write to a read-only register
- Privilege error on a non-secure read or write to registers associated with a secure event (CLEC_MRR_j[31] S = 1 for that event)
- Privilege error on a write to CLEC_ESR_j on an event whose CLEC_MRR_j[30] ESE = 0
- Privilege error on a write to CLEC_MRR_j on an event when CLEC_GELRS[0] LOCK = 1

Any further access control must be implemented with firewalls and/or MMU configuration outside the CLEC.

9.2.2.2.4 CLEC ECC Support

Standard ECC aggregator strategy is used to implement ECC protection on the SRAM used to implement the CLEC routing table. The CLEC has two ports connected to the ECC_AGGR at the MSMC_WRAP level. For RAM_ID values of CLEC ECC-protected memories, see , *Compute Cluster ECC Aggregators*. For details on the generic ECC_AGGR functionality, see Section 12.10.4, *ECC Aggregator*.

9.2.2.2.5 CLEC Intra-Core Communication

In order to support intra-core synchronization inside Compute Cluster, associated C71/A72 events are routed through the CLEC.

9.2.2.2.6 CLEC Event Maps

See *Interrupt Sources* for CLEC event mapping.

9.2.2.2.6.1 CLEC ESM Event Routing

Table 9-2 shows the CLEC ESM event output map.

Table 9-2. CLEC ESM Event Output Map

Destination	Source Interrupt
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_0	MSMC_ECCAGGR0_UNCORRECTED_ERR_LEVEL_INTR
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_1	MSMC_ECCAGGR0_CORRECTED_ERR_LEVEL_INTR
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_2	MSMC_ECCAGGR1_UNCORRECTED_ERR_LEVEL_INTR
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_3	MSMC_ECCAGGR1_CORRECTED_ERR_LEVEL_INTR
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_4	MSMC_ECCAGGR2_UNCORRECTED_ERR_LEVEL_INTR
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_5	MSMC_ECCAGGR2_CORRECTED_ERR_LEVEL_INTR
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_6	MSMC_DFT_PBIST_SAFETY_ERROR_INTR0
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_7	ARM0_ECC_ECCAGGR0_UNCORRECTED_ERR_LEVEL_INTR
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_8	ARM0_ECC_ECCAGGR0_CORRECTED_ERR_LEVEL_INTR
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_9	ARM0_ECC_ECCAGGR1_UNCORRECTED_ERR_LEVEL_INTR
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_10	ARM0_ECC_ECCAGGR1_CORRECTED_ERR_LEVEL_INTR
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_11	ARM0_ECC_ECCAGGR_COREPAC_UNCORRECTED_ERR_LEVEL_INTR
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_12	ARM0_ECC_ECCAGGR_COREPAC_CORRECTED_ERR_LEVEL_INTR
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_13	ARM0_DFT_PBIST_0_SAFETY_ERROR_INTR
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_14	RESERVED
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_15	RESERVED
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_16	RESERVED
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_17	RESERVED

Copyright © 2024 Texas Instruments Incorporated

Table 9-2. CLEC ESM Event Output Map (continued)

Destination	Source Interrupt
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_18	RESERVED
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_19	RESERVED
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_20	RESERVED
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_21	RESERVED
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_22	RESERVED
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_23	RESERVED
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_24	RESERVED
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_25	RESERVED
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_26	RESERVED
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_27	RESERVED
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_28	RESERVED
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_29	RESERVED
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_30	RESERVED
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_31	RESERVED
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_32	RESERVED
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_33	RESERVED
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_34	RESERVED
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_35	C7X_4_ECC_ECCAGGR_COREPAC_UNCORRECTED_ERR_LEVEL_INTR
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_36	C7X_4_ECC_ECCAGGR_COREPAC_CORRECTED_ERR_LEVEL_INTR
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_37	C7X_4_DFT_PBIST_SAFETY_ERROR_INTR
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_38	C7X_5_ECC_ECCAGGR_COREPAC_UNCORRECTED_ERR_LEVEL_INTR
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_39	C7X_5_ECC_ECCAGGR_COREPAC_CORRECTED_ERR_LEVEL_INTR
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_40	C7X_5_DFT_PBIST_SAFETY_ERROR_INTR
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_41	RESERVED
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_42	RESERVED
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_43	RESERVED
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_44	RESERVED
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_45	RESERVED
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_46	RESERVED
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_47	MSMC_DDR1_UNCORRECTED_ERR_LEVEL_INTR
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_48	MSMC_DDR1_CORRECTED_ERR_LEVEL_INTR
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_49	RESERVED
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_50	RESERVED
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_51	RESERVED
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_52	RESERVED
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_53	RESERVED
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_54	RESERVED
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_55	RESERVED
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_56	RESERVED
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_57	RESERVED
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_58	RESERVED
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_59	RESERVED
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_60	RESERVED
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_61	GIC500SS_VBUSM_TO_GASKET_LVL_INTR

Copyright © 2024 Texas Instruments Incorporated

Table 9-2. CLEC ESM Event Output Map (continued)

Destination	Source Interrupt
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_62	GIC500SS_ECC_AGGR_UNCORR_LEVEL_INTR
COMPUTE_CLUSTER0_CLEC_ESM_EVENTS_OUT_LEVEL_63	GIC500SS_ECC_AGGR_CORR_LEVEL_INTR

Copyright © 2024 Texas Instruments Incorporated

9.2.3 Other Interrupt Controllers

All other device interrupt controllers are described in their respective chapters and/or third party documentation.

Copyright © 2024 Texas Instruments Incorporated

9.3 Interrupt Routers

9.3.1 INTRTR Overview

The interrupt router (INTRTR) module provides a mechanism to mux *M* interrupt inputs to *N* interrupt outputs, where all *M* inputs are selectable to be driven per *N* ouput. There is one register per output (MUXCNTL_N) that controls the selection.

There are several INTRTR modules in the device. Their purpose is described in *Interrupt Architecture*.

The user should take the following into account when programming the MUXCNTL_N register:

- Avoid programming this register when input interrupts are active. This could lead to spurious asynchronous output toggles which may lead to unpredictable behavior.

The recommended general programming sequence is as follows:

1. Disable interrupt by writing '0' to the INT_ENABLE bit field. Do not change mux control configuration settings (ENABLE bit field) at this time.
2. Change the mux control configuration settings. INT_ENABLE needs to remain '0' at this time.
3. Enable interrupt by writing '1' to INT_ENABLE. Do not change mux control configuration settings at this time.

9.4 Interrupt Sources

Note

See Appendix Spreadsheet for Interrupt Inputs and Outputs details.

10 Data Movement Architecture (DMA)

10.1 DMA Architecture 781
10.2 Navigator Subsystem (NAVSS) 843
10.3 Peripheral DMA (PDMA) 903
10.4 Data Routing Unit (DRU) 967

Copyright © 2024 Texas Instruments Incorporated

10.1 DMA Architecture

This section describes the DMA (data movement) architecture in the device.

10.1.1 Overview

The DMA architecture specifies the data structures used by Texas Instruments standard communications modules to facilitate direct memory access (DMA) and to provide a consistent application programming interface (API) to the host software in multi-core devices. The data structures and the API used to manipulate them will be jointly referred to as NAVSS.

Frequent tasks are commonly offloaded from the host processor to peripheral hardware to increase system performance. Significant performance gains may result from careful design of the host software and communication module interface. In networking systems, packet transmission and reception are critical tasks. Texas Instruments has developed the NAVSS standard, which is aimed at maximizing the efficiency of interaction between the host software and communications modules.

The design goals for NAVSS are as follows:

- Minimize host interaction
- Maximize memory use efficiency
- Maximize bus burst efficiency
- Maximize symmetry between transmit/receive operations
- Maximize scalability for number of connections, buffer sizes, queue sizes, and protocols supported
- Minimize protocol specific features
- Minimize complexity

NAVSS architecture is shown on Figure 10-1.

Copyright © 2024 Texas Instruments Incorporated

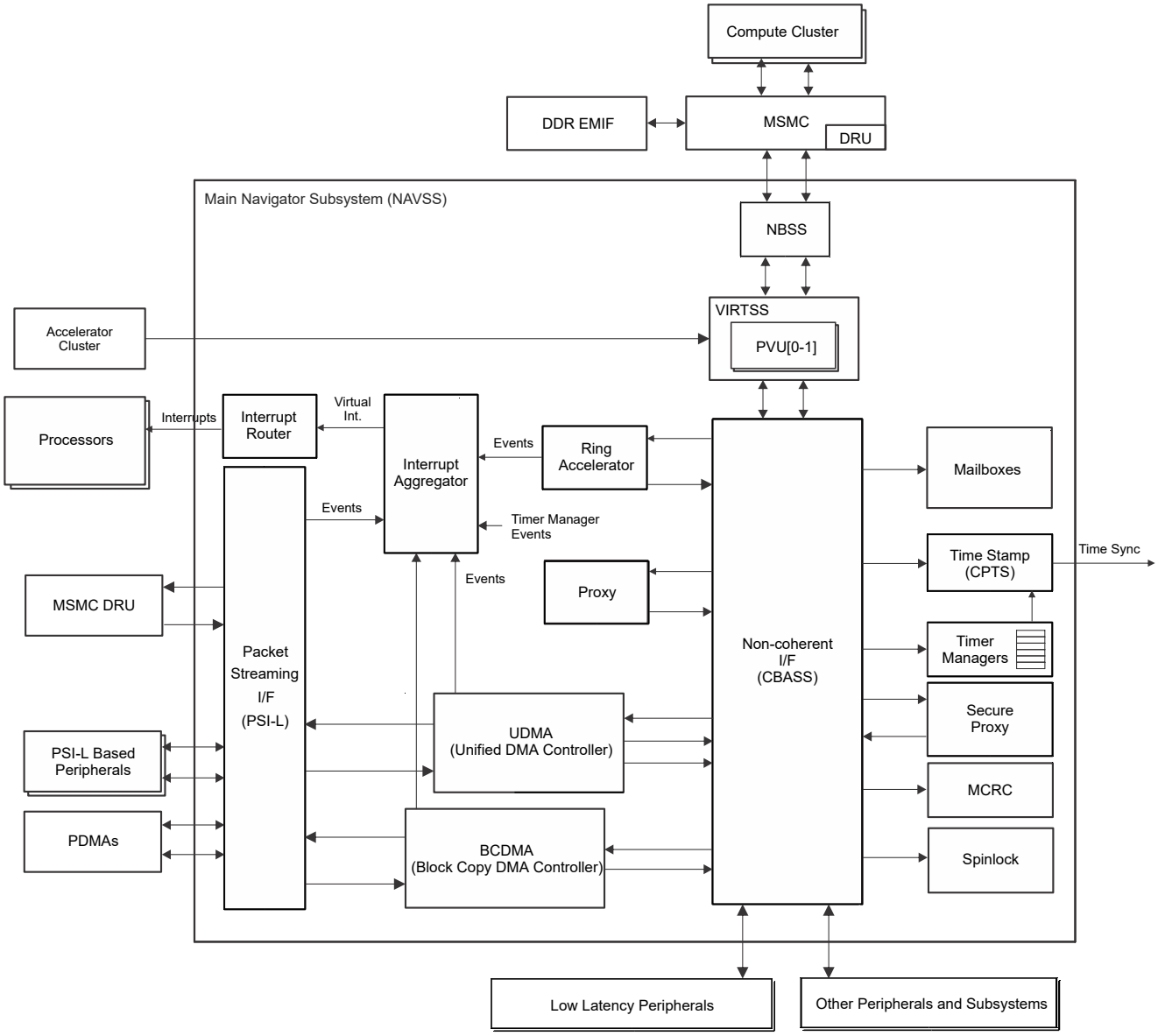


Figure 10-1. NAVSS Overview

10.1.1.1 Navigator Subsystem

The Navigator Subsystem (NAVSS hardware) is a container which groups together as much as possible the components which provide the Hardare to Software boundary for data movement control in the system. All of the components which control work handoff (queuing, interrupts, monitoring and debugging) are included in the NAVSS which in turn is located as close as possible to the various host processors to which services are provided. Other DMA components such as DRUs, UTC, and PDMAs exist outside the NAVSS but all are controlled by the root complexes provided in NAVSS.

The SoC has two NAVSSes: one in main and one in MCU domain.

10.1.1.2 Ring Accelerator (RA)

The ring accelerator (RINGACC or RA) is a hardware module that is responsible for accelerating management of various types of queues in the system. The RA accelerates passing of packets between a producer and consumer in normal system memory (including cached memory). The RA is capable of accelerating the read/write pointer maintenance and occupancy tracking operations.

Copyright © 2024 Texas Instruments Incorporated

The ring accelerator operates like a bus infrastructure bridge in that it passes transactions through it between a source and destination interface. As transactions are passed, the RA modifies the address, byte count, and transaction identifiers and internally updates the occupancies/pointers.

Each queue that the RA provides can operate in either exposed-ring-mode or private-queue-mode.

- The exposed ring mode allows software to directly access the underlying ring structure to add or remove items from the tail or head of the ring respectively. Whenever items are added or removed from a ring, the host software is required to write to a corresponding doorbell register to increment or decrement the ring occupancy. When using the exposed ring mode, no proxy is required between the software and the RA but all queue add operations require a memory fence to be performed to ensure that the data has landed in a snoopable cache before the doorbell register is written.
- The private queue mode provides an abstract view of the ring so that the host software does not need to know the actual physical address location or current read or write indexes of the ring. The private queue mode provides a memory window for each ring which when written redirects the write to the address pointed to by the write pointer and when read redirects the read to the address pointed to by the read pointer. The same address range is always used to push or pop elements from a given queue.

Rings are an implementation of a logical queue with the following limitations:

- Each ring has a finite size. When rings are used in exposed ring mode the following conditions apply:
  - A separate operation is required to write/read the contents of a ring and to update the occupancy of the ring
  - It is not straightforward to allow multiple producers to write to a ring or multiple consumers to read from a ring. Additional synchronization and pointer passing is required since rings require software to manage one of the pointers for the queue.
  - An exposed ring cannot be used when software or hardware needs to both read and write the same queue (such as for a DMA RX free queue where errors can have the DMA write the element back onto the same RX free queue). Since software owns one side of the pointers and hardware owns the other, they cannot be kept coherent if either side needs to update either at any time.

The RA allows each ring to be configured to a different primary element size. Element sized chunks of data are placed onto and retrieved from rings when queuing or de-queueing occurs. Element sizes can be as small as 4 bytes or as large as 256 bytes.

10.1.1.3 Proxy

The proxy provides a mechanism for software to access the RA coherently when the processor does not support large bursts. The RA requires a single burst for each operation so that it maintains atomicity and coherence by relying on the atomicity of the bursts on the bus interconnect fabric, since it only delivers a single burst to the RA before the next. Since processors are usually limited in the size of a burst they can deliver natively, such as 32 to 128 bits, they cannot be used directly with the data access region of the RA for larger element sizes. The proxy solves this gap by providing a temporary space for the software to form a larger burst of data before it is sent to the RA. The proxy allows smaller processor accesses to the data and only forwards to the RA when the entire data burst is complete, as directed by the software. Each proxy hardware can support multiple threads of software (whether on the same or different processors) operating on bursts at the same time, and they do not interfere with each other, as long as each thread of software only operates on its thread in the proxy. Each proxy thread also supports access to any RA queue via different offsets similar to how the RA provides access to the queues via different offsets.

For writing data to the RA, the software will write the data in native sized writes to the proxy thread it has been assigned. The proxy simply accepts the writes into a buffer reserved for that thread. Only when the software writes to the completion byte offset (last byte of the burst) then will the proxy take the entire burst of data written (including the final write data) and send it to the RA as a single write burst. After the completion write, the software can begin building a new data burst. For reading data from the RA, the software will read the data in native sized reads to the proxy thread. For the first read, the proxy will read the entire burst from the RA, so that it atomically gets the next element off the queue. This entire data is stored in the proxy thread buffer, and the requested portion is returned to software. Then software can read any location within the burst and the proxy will read it from the buffer. When software is done reading the data, it must read the completion byte offset so that

Copyright © 2024 Texas Instruments Incorporated

the proxy 'knows' that the read burst is completed, and that for the next read it must read a burst from the RA queue again. A single proxy thread can only be used for a single data burst at a time, so once starting a write it should complete before starting another write or a read, and similarly once starting a read, it should complete before starting another read or a write. If software needs to read and write at the same time, it should use two proxy threads. Similarly if there are multiple threads of software that need to access RA at the same time, they should use separate proxy threads.

10.1.1.4 Secure Proxy

A special version of a proxy is the secure proxy that provides secured access to RA queues for communication to the security master of the system. The secure proxy provides basically the same function as a normal proxy. The data burst size is fixed for message sizes to the security master, so there is no support for all the element sizes of the RA. The security master locks down the RA queue to access and direction for each proxy thread so that the software cannot access anything they shouldn't, so the software only sees a single queue (unlike the normal proxy), and any attempts to access another queue or access it in violation of how the security master defined and an error will be flagged. This allows for a much more controlled setup for passing messages to the security master.

10.1.1.5 Interrupt Aggregator (INTA)

The interrupt aggregator (INTR_AGGR or INTA) is a hardware module which provides a persistent view of the real time status of various DMA components within the overall SoC DMA system. The main function of the module is to convert between 'global events' (assertion and de-assertion events transmitted on the event-transport-lane bus) and 'local events' (level sensitive signals on output, and level or edge sensitive signals on input).

The INTA block provides the following functions:

- Conversion of local event signal lines into global events
- Performing an 'Y-split' operation of global events
- Counting global events
- Steering and aggregation of global events into specified bit positions in one of N interrupt cause registers
- SoC interrupt-achitecture-compliant set/clear and mask set/clear functionality to cover interrupt cause registers
- Output of virtual interrupt sources to the Interrupt Router module

10.1.1.6 Interrupt Router (IR)

The interrupt router (INTR_ROUTER or IR) is a hardware module which allows for virtual interrupt sources which were output from the INTA block to be crossbar switched onto physical interrupts which are connected to the system level Interrupt Controller (ARM GIC or other interrupt controller).

10.1.1.7 Unified DMA – Third Party Channel Controller (UDMA-C)

Note

In this SoC, UDMA-P (see Section 10.1.1.10) serves UDMA-C functions. That is, there is no discrete UDMA-C controller in this SoC device.

The Unified DMA 3rd-Party Channel Controller is intended to perform similar (but significantly upgraded) functions to the EDMA Channel Controller engine on previous SoC devices. The UDMA-C module supports the transmission of Transfer Request packets from memory mapped data structures to data channels within UTCs in the system and reception of Transfer Response packets from data channels in UTCs in the system into memory mapped data structures. Up to 512 (configuration specific) third-party DMA control channels are provided within the UDMA-C. Each control channel corresponds to a single third party DMA data channel in a separate UTC. Transfer Request messages provide transfer parameters which are to be used by the UTC channels to move data in the system (previously implemented as PARAM entries). Transfer Response messages are sent back from the UTC in a one to one relationship with Transfer Request packets and provide an indication of both completion of a data transfer and whether or not any exception or error occurred during the data transfer.

Copyright © 2024 Texas Instruments Incorporated

The UDMA-C controller maintains state information for each of its channels which allows Transfer Request packet transmission and Transfer Response packet reception operations to be time division multiplexed between channels in order to share the underlying DMA hardware. An internal DMA scheduler is used to control the ordering and rate at which this multiplexing occurs for the transmission of Transfer Request packets. The ordering and rate of Transfer Response receive operations is directly controlled by the order in which those packets are received on the Rx PSI-L interface. A block diagram of a UDMA-C module is shown Figure 10-2.

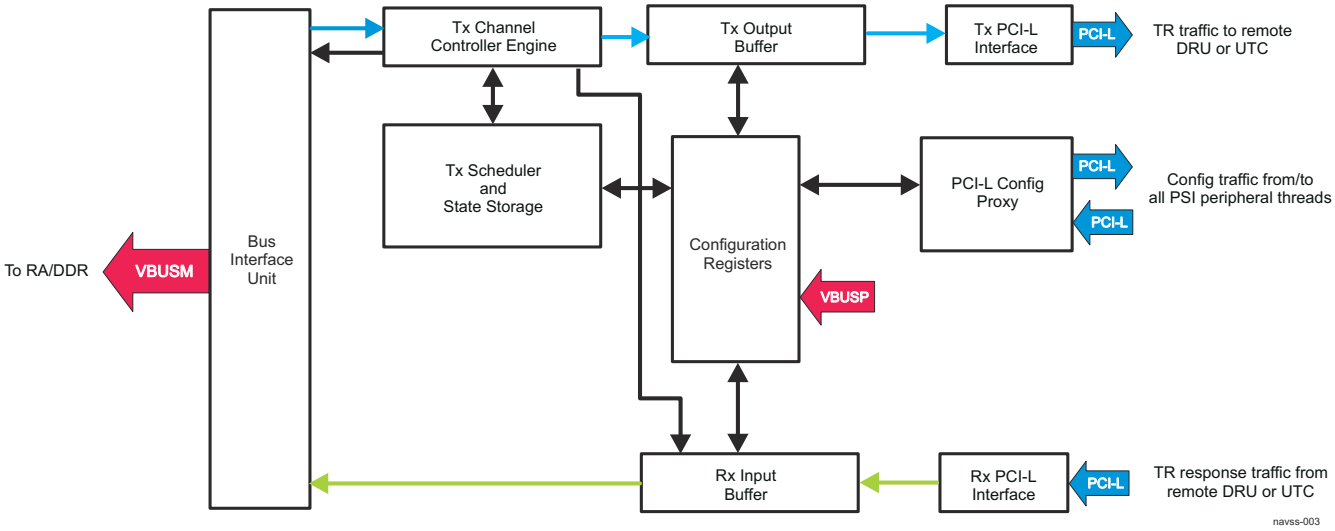


Figure 10-2. UDMA-C Block Diagram

10.1.1.8 Unified Transfer Controller (UTC)

The Unified Transfer Controller is intended to perform similar functions to the EDMA Transfer Controller engine used on previous devices. The UTC engine is generally classified as a third-party DMA. This designation comes from the fact that the engine is not actually the source or sink of the data which is being moved but is instead an intermediary 3rd-party that performs the data move on behalf of the source and sink.

The UTC engine accepts Transfer Response messages from the UDMA-P via a PSI-L interface which provide instructions to copy data between a source read interface and a destination write interface. The sequence of operations that can be instructed includes up to 4-dimensional nested loops. Multiple types of Transfer Request messages are specified and each UTC instance in the system may support all types or any specified subset. The TR formats are specified in detail in a later section. When a Transfer Request has been completed, the UTC returns a Transfer Response message back to the originating UDMA-P. The UTC has the ability to generate events at specified completion points when processing a Transfer Request. These events are sent back to the Interrupt Aggregator block.

An UTC instance may be configured to only support VBUSM to VBUSM block copies or they may also optionally support ‘split’ operations where a read engine is instructed by a TR to read data from a VBUSM interface and send the data to the Packet Streaming fabric via a PSI-L master interface. A similar split operation is supported for writes where data is received into a PSI-L slave interface and is then routed to a write engine which has been instructed by a TR to move the data into a set of memory locations.

10.1.1.9 Data Routing Unit (DRU)

For purposes of the K3 Data Movement architecture, the DRU is essentially a UTC which supports only the block copy mode subset of the Transfer Request message formats. The DRU typically will provide the highest performance block copy data movement capability of any DMA engine within the SoC. The data routing unit (DRU) is a high bandwidth, flexible routing engine with programmable DMA transfer requests. It enables user to perform high speed data transfers between memory mapped slave endpoints, processor caches and shared caches. DRU behaves like a DMA transfer controller, moving data at CC_ARMSS frequency.

Copyright © 2024 Texas Instruments Incorporated

10.1.1.10 Unified DMA – Peripheral Root Complex (UDMA-P)

The UDMA-P is intended to perform similar (but significantly upgraded) functions as the packet-oriented DMA used on previous SoC devices. The UDMA-P module supports the transmission and reception of various packet types. The UDMA-P is architected to facilitate the segmentation and reassembly of SoC DMA data structure compliant packets to/from smaller data blocks that are natively compatible with the specific requirements of each connected peripheral. Multiple Tx and Rx channels are provided within the DMA which allow multiple segmentation or reassembly operations to be ongoing. The DMA controller maintains state information for each of the channels which allows packet segmentation and reassembly operations to be time division multiplexed between channels in order to share the underlying DMA hardware. An external DMA scheduler is used to control the ordering and rate at which this multiplexing occurs for Transmit operations. The ordering and rate of Receive operations is indirectly controlled by the order in which blocks are pushed into the DMA on the Rx PSI-L interface.

The UDMA-P also supports acting as both a UTC and UDMA-C for its internal channels. Channels in the UDMA-P can be configured to be either Packet-Based or Third-Party channels on a channel by channel basis.

A block diagram of the UDMA-P Controller is shown in Figure 10-3.

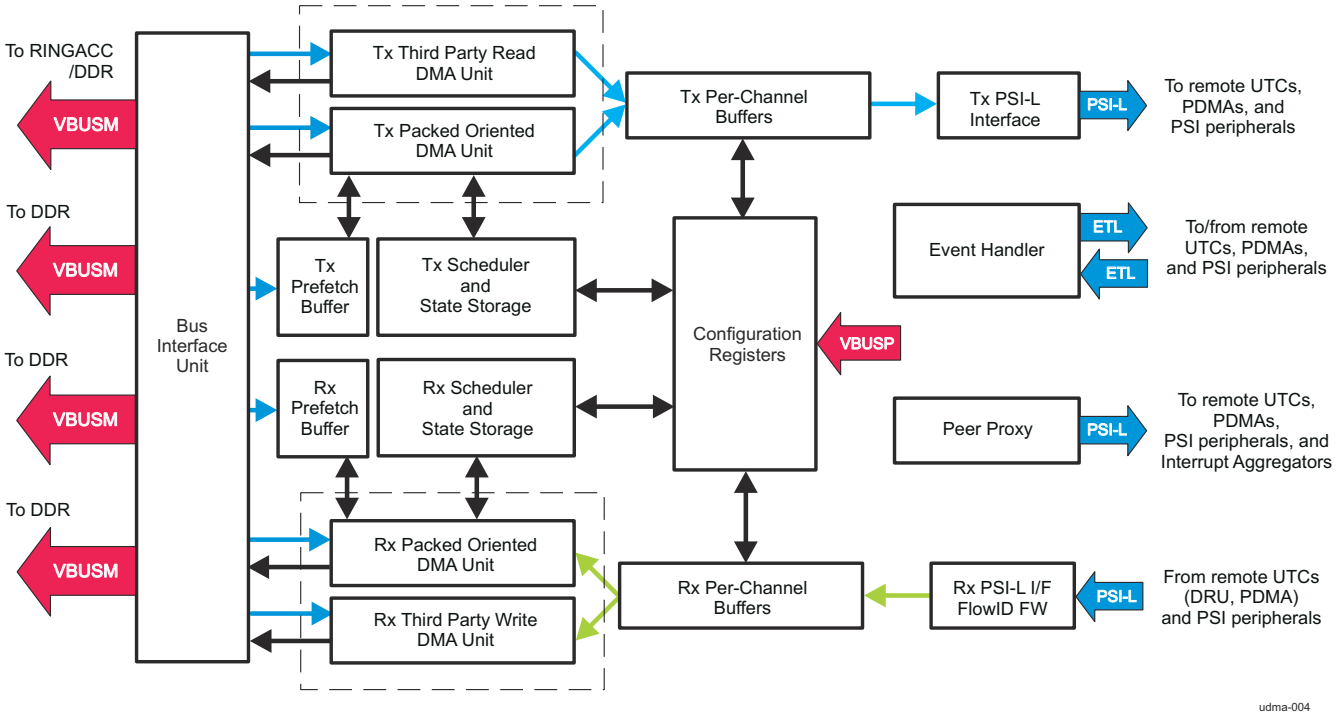


Figure 10-3. UDMA-P Block Diagram

10.1.1.11 Peripheral DMA (PDMA)

The Peripheral DMA is a simple DMA which has been architected to specifically meet the data transfer needs of peripherals which perform data transfers using memory mapped registers accessed via a standard non-coherent bus fabric. The PDMA module is intended to be located close to one or more peripherals which require an external DMA for data movement and is architected to reduce cost by using VBUSP interfaces and supporting only statically configured Transfer Request operations. The PDMA is only responsible for performing the data movement transactions which interact with the peripherals themselves. Data which is read from a given peripheral is packed by a PDMA source channel into a PSI-L data stream which is then sent to a remote peer UDMA-P destination channel which then performs the movement of the data into memory. Likewise, a remote UDMA-P source channel fetches data from memory and transfers it to a peer PDMA destination channel over PSI-L which then performs the writes to the peripheral.

The Peripheral DMA architecture is intentionally heterogeneous (UDMA-P + PDMA) to right size the data transfer complexity at each point in the system to match the requirements of whatever is being transferred to

Copyright © 2024 Texas Instruments Incorporated

or from. Peripherals are typically FIFO based and do not require multi-dimensional transfers beyond their FIFO dimensioning requirements, so the PDMA transfer engines are kept simple with only a few dimensions (typically for sample size and FIFO depth), hardcoded address maps, and simple triggering capabilities.

Multiple source and destination channels are provided within the PDMA which allow multiple simultaneous transfer operations to be ongoing. The DMA controller maintains state information for each of the channels and employs round-robin scheduling between channels in order to share the underlying DMA hardware.

10.1.1.12 Embedded DMA

Embedded DMA refers to a DMA-capable master module which either follows an industry standard hardware/software Application Programming Interface (API) or is a module which does not natively comply with the TI DMA architecture requirements. It is assumed that modules which include Embedded DMA will provide their own dedicated software drivers and interoperability with other DMA entities in the system will be accomplished with software bridging code. None of the remainder of this chapter has any effect on the operation or architecture of Embedded DMA functions.

Copyright © 2024 Texas Instruments Incorporated

10.1.1.13 Definition of Terms

Host The host is an intelligent system resource that configures and manages each communications control module. The host is responsible for allocating memory, initializing all data structures, and responding to port interrupts.

Channel A channel refers to the sub-division of information (flows) that is transported across a DMA engine. Each channel has associated state information. Channels are used to segregate information flows based on the protocol used, scheduling requirements (for example, CBR, VBR, ABR), or concurrency requirements (that is, blocking avoidance). Information flow within a channel is a stream of strongly ordered information.

Data Buffer A data buffer is a single data structure that contains payload information for transmission to or reception from a channel.

Buffer Descriptor A buffer descriptor is a single data structure that contains information about one or more data buffers.

Packet Descriptor A packet descriptor is another name for the first buffer descriptor within a packet. Some fields within a data buffer descriptor are only valid when it is a packet descriptor including the tags, packet length, packet type, and flags. All Monolithic type descriptors are packet descriptors (and are also a Data Buffer).

Queue A queue is a list of strongly ordered entries which is typically used to pass work between a producer and a consumer. Queue entries in most cases are references to a work payload which is being passed but in some cases (Transfer Request Packets for example) queue entries may actually contain data which is being transferred. Queues are used throughout UDMA whenever communication is required between entities. Queues can have multiple different implementations and UDMA uses two of the most common: linked lists and rings.

Linked list A linked list is a data structure in which each entry stores not only the entry data but also a chaining pointer to the next entry in the list. The last entry in each list has its chaining pointer set to NULL (typically encoded as 0x0). The list manager maintains a pointer to the head element on the list and to the tail element on the list. Since the chaining pointer is stored with the entry data, linked lists have a length which is dynamically changeable and limited only by the ability to allocate additional entries which are to be queued/de-queued. Linked lists are present in Host Descriptors to chain multiple descriptors to form a packet

Ring A ring is a data structure in which a contiguous memory block defined by M N-byte entries (total size is M × N bytes) is statically allocated and sequentially written/read in order to pass data or data references. Rings are also referred to as circular buffers because when the last element in the contiguous memory array is written, the pointers wrap back to the beginning address for the ring and start the process all over again. The Ring Accelerator component uses rings in order to implement logical queues

Free Descriptor/ Buffer Queue A free descriptor/buffer queue is a list of available descriptors with pre-linked empty buffers that are to be used by the receive ports for host type descriptors. Free Descriptor/Buffer Queues are implemented by the Ring Accelerator.

Free Descriptor Queue A free descriptor queue is a list of available descriptors that are not yet linked with buffers that are to be used by the receive ports for monolithic type descriptors. Free Descriptor Queues are implemented by the Ring Accelerator.

Packet Queue A packet queue is a list of valid (that is, populated) packet descriptors that is used for forwarding a packet from one entity to another for any number of purposes. Packet Queues are implemented by the Ring Accelerator.

Memory Memory is an area of data storage managed by the host. This area is visible to the port as a 64-bit addressable area.

Device Driver A device driver is application independent software that runs on the host for purposes abstracting the low level hardware so that upper level software can use the hardware without knowing every bit field location or initialization sequence.. General device driver functions include port initialization, transmit packet queuing, and receive packet processing.

Copyright © 2024 Texas Instruments Incorporated

SOP Start of Packet. This refers to the descriptor/buffer that is the first buffer in a packet.

MOP Middle of Packet. This refers to the descriptors/buffers that are neither the first or last buffers in a packet.

EOP End of Packet. This refers to the descriptor/buffer that is the last buffer in a packet.

Copyright © 2024 Texas Instruments Incorporated

10.1.2 DMA Hardware/Software Interface

The intent of the TI Data Movement (DMA) architecture is to provide a uniform hardware/software interface which includes a rich set of mechanisms that software can make use of to transfer data with low overhead and reasonable complexity. To this goal, the number of components which software is required to interact with on an ongoing real time basis has been kept to a minimum and is as follows (in order of anticipated frequency of access):

- Interrupt Aggregator
- Ring Accelerator
- UDMA-P/C (for managed flow control and error/exception handling only)

When interrupts are used, the interrupt aggregator will provide the vast majority of interrupt sources from all DMA components in the system. All non-exception/non-debug packet and TR completion signaling originates from the Interrupt Aggregator and the INTA provides a uniform set of registers that can be queried to quickly determine the cause of a specific interrupt. Events from PSI-L/ETL components are all routed to the host via the Interrupt Aggregator.

The Ring Accelerator provides the primary means by which work is sent to or received from the UDMA-P DMA engines (and the UTC/DRU, and PDMA engines whose control operations are proxied by the UDMA-P).

The UDMA-P are primarily fronted by the RA and the INTA but for real time operations like software controlled flow control (via pause) some registers in the UDMA-P blocks may be directly manipulated.

Data structures are used to pass anytime information between components in the system. These components may be hardware or software. The following sections describe the data structures which are used within a UDMA based system for passing information. These data structures include data buffers, packet descriptors, buffer descriptors, queues (including transmit queues, transmit completion queues, and receive queues) and the configuration registers that are provided in the various components. The following sections provide a detailed description of these data structures.

10.1.2.1 Data Buffers

A data buffer is a byte aligned contiguous block of memory used to store packet payload data. Each buffer is described in an entry in either a packet descriptor or in a buffer descriptor. A data buffer may hold any portion of a packet and may be linked together (via descriptors) with other buffers to form packets. Data buffers may be allocated anywhere within the 64-bit memory space.

The Buffer Length field of the packet/buffer descriptor indicates the number of valid data bytes in the buffer. There may be from 1 to 4M-1 valid data bytes in each buffer.

10.1.2.2 Descriptors

Descriptors are so named because their primary function is to describe other data structures.

The UDMA architecture provides 4 basic types of descriptors each of which has specific characteristics intended to address different requirements. Table 10-1 shows the different types of descriptors and their characteristics.

Table 10-1. UDMA Descriptor Types and Attributes

Descriptor Type	Includes Valid Packet Info	Provided Number of Slots to Link In External Data Buffers	Provides Local Packet Data Storage	Provides Local Protocol Specific Storage	Provides Slot to Link Additional Descriptors Within Same Packet	Description
Host Packet Descriptor	✓	1		✓	✓	Used to describe packet and SOP buffer in applications that require Host OS compatible data structures (i.e. applications where the descriptors and buffers cannot be managed independently but must instead be pre-linked by the Host software). These applications inherently require a separate descriptor for each buffer.

Copyright © 2024 Texas Instruments Incorporated

Table 10-1. UDMA Descriptor Types and Attributes (continued)

Descriptor Type	Includes Valid Packet Info	Provided Number of Slots to Link In External Data Buffers	Provides Local Packet Data Storage	Provides Local Protocol Specific Storage	Provides Slot to Link Additional Descriptors Within Same Packet	Description
Host Buffer Descriptor		1			✓	Used to describe non-SOP buffers in applications that require Host OS compatible data structures
Monolithic Packet Descriptor	✓	0	✓	✓		Used to provide descriptor and data information in one contiguous data structure.
Transfer Request Packet Descriptor		0				Used to feed transfer request sequences to the UDMA third party channel controller

As Table 10-1 shows, two of the four different descriptor types (the Host Packet Descriptor and Monolithic Packet Descriptor) provide packet level information that is useful to both the ports and the Host in order to properly process the packet. These descriptors are referred to as Packet Descriptors and will always appear as the first descriptor (or only descriptor in the case of Monolithic types) within a packet.

Software must allocate descriptors on 16-byte address boundaries. 16-byte is the minimum granularity that UDMA supports. This is intended for better memory utilization by allowing on-chip descriptors which are not a power of 2 in length to be packed into arrays with little wasted memory space.

Even though descriptors and buffers may be allocated on any 16-byte alignment, careful consideration of the alignment effects should be made based on the storage location and any cache related affects that may exist. If data structures are placed in off-chip SDRAM the burst size and alignment restrictions of the memory devices must be considered in order to avoid performance issues related to continually fetching mis-aligned blocks. In this case, the memory efficiency can be reduced to 50% because two memory bank lines are read for every line sized data fetch. Similarly, placing more than one descriptor or buffer object within a single cache line can cause the adjacent object to become corrupted during cache line writeback operations.

Software/application specific control information may be added to the end of any of the packet descriptor types using as many extra words as necessary but the above alignment rules still apply.

10.1.2.2.1 Host Packet Descriptor

Host Packet Descriptors are designed to be used when the application requires support for true, unlimited fragment count scatter/gather type operations. The Host Packet Descriptor contains the following information:

- Indicator which identifies the descriptor as a Host Packet Descriptor
- Source and Destination Tags
- Packet Type
- Packet Length
- Protocol Specific Region Size
- Protocol Specific Control/Status Bits
- Pointer to the first valid byte in the SOP data buffer
- Length of the SOP data buffer
- Pointer to the next buffer descriptor in the packet
- Software specific information

Host Packet Descriptors always contain 48 bytes of required information and may also contain optional software specific information and protocol specific information. How much optional information (and therefore the allocated size of the descriptors) is required is application dependent.

The Host Packet descriptor layout is shown below.

Packet Info (16 bytes)

Copyright © 2024 Texas Instruments Incorporated

Linking Info (8 bytes)
Buffer Info (12 bytes)
Original Buffer Info (12 bytes)
Extended Packet Info Block (Optional) Includes Timestamp and Software Data (16 bytes)
Protocol Specific Data (Optional) (0 to M bytes where M is a multiple of 4)
Other Software Data (Optional and User Defined)

Host Packet Descriptors may be linked with zero or more additional Host Buffer Descriptors in a singly linked list fashion to form packets. Each Host Packet consists of a single Host Packet Descriptor followed by a chain of zero or more Host Buffer Descriptors linked together using the Next Descriptor Pointer fields in the descriptors. The last descriptor in a Host packet has a zero Next Descriptor Pointer.

The 'Other Software Data' portion of the descriptor exists after all of the defined words and is reserved for use by the host software to store completely private data. This region is not used in any way by hardware components in a UDMA system and these modules will not modify any bytes within this region.

The contents of the Host Packet Descriptor words are detailed in Table 10-2 through Table 10-18.

Table 10-2. Host Packet Descriptor Packet Information Word 0 (PD Word 0)

Bits	Name	Description	Rx Overwrite
31:30	2'd1	64-bit Host Packet Descriptor Type Identifier	Yes
29	Extended Packet Info Block Present	This field indicates the presence of the Extended Packet Info Block in the descriptor. 0 = EPIB is not present 1 = 16 byte EPIB is present	Yes
28	Protocol Specific Region Location	This field indicates the location of the Protocol Specific Words: 0 = PS Words are located in the descriptor 1 = PS Words are located in the SOP Buffer immediately prior to the data.	Yes
27:22	Protocol Specific Valid Word Count	This field indicates the valid number of 32-bit words in the protocol specific region. This is encoded in increments of 4 bytes as follows: 0 = 0 bytes 1 = 4 bytes … 16 = 64 bytes ... 32 = 128 bytes 33-63 = RESERVED	Yes
21:0	Packet Length	The length of the packet in bytes. If the Packet Length is less than the sum of the buffer lengths, then the packet data will be truncated. A Packet Length greater than the sum of the buffers is an error unless the packet length is set to 0x3FFFFF. If the packet length is set to all 1s (0x3FFFFF) the Tx port will disable truncation and will transmit as much data as is specified in the Buffer Length fields. On Rx, if the received data exceeds 4M-1 bytes, the packet length field will saturate to a value of 0x3FFFFF. The valid range for an exact packet length is 0 to 4M-1 bytes. If the packet length is set to 0, the port will not actually transmit any information. Instead, the port will perform buffer/descriptor reclamation as instructed in the return information in word 2.	Yes

Table 10-3. Host Packet Descriptor Packet Information Word 1 (PD Word 1)

Bits	Name	Description	Rx Overwrite
31:28	Error Flags	This field contains error flags that can be assigned based on the packet type	Yes

Copyright © 2024 Texas Instruments Incorporated

Table 10-3. Host Packet Descriptor Packet Information Word 1 (PD Word 1) (continued)

Bits	Name	Description	Rx Overwrite
27:24	Protocol Specific Flags	This field contains protocol specific flags/information that can be assigned based on the packet type.	Yes
23:14	Packet ID	Unique Packet ID for packet within FlowID	Yes
13:0	Flow ID	Flow ID within which this packet is being transported. The FlowID is used by downstream blocks to make decisions about packet steering and resource allocations. FlowIDs are also used to allow specific packets to be received into specific sets of buffers.	Yes

Table 10-4. Host Packet Descriptor Packet Information Word 2 (PD Word 2)

Bits	Name	Description	Rx Overwrite
31:27	Packet Type	This field indicates the type of this packet and is encoded as follows: 0-31 = Application specific	Yes
26:19	RESERVED	Reserved	No
18	Return Policy	This field indicates the return policy for this packet. 0 = Entire packet (still linked together) should be returned to queue specified in bits 15:0. 1 = Each buffer should be returned to queue specified in bits 15:0 of Word 2 in their respective descriptors. The Tx DMA will return each buffer in sequence.	No
17	Early Return	This field indicates that each buffer pointer should be immediately returned to the specified queue when data transfer is started from the packet instead of waiting for the entire buffer to be emptied. This flag is used to enable in-place reception of packets on a Receive Channel while the source packet is in the process of being transferred on a Transmit Channel. 0 = Buffer/Packet descriptor pointers should only be returned after all reads have been completed 1 = Buffer/Packet descriptor pointers should be returned immediately upon fetching the descriptor and beginning to transfer data.	No
16	Return Push Policy	This field indicates how a Transmit or Receive DMA should return the descriptor pointers to the free queues This field is encoded as follows: 0 = Descriptor must be returned to tail of queue 1 = Descriptor must be returned to head of queue This bit is only used when the Return Policy bit is set to 1. The Rx DMA will only use this field when an error occurs during reception and the DMA must return descriptors back to the free queue from which they came. This field must be set to 0 for descriptors which will be placed on queues managed by the Ring Accelerator.	No
15:0	Packet Return Queue/ Ring Num	This field indicates the ring number in the RA that the descriptor is to be returned to after transmission is complete. The value 0xFFFF is reserved.	No

Table 10-5. Host Packet Descriptor Packet Information Word 3 (PD Word 3)

Bits	Name	Description	Rx Overwrite
31:24	Source Tag – Hi	This field is application specific. During Packet reception, the DMA controller in the port will overwrite this field as specified in the rx_src_tag_hi_sel field in the flow configuration table entry.	Configurable
23:16	Source Tag – Lo	This field is application specific. During Packet reception, the DMA controller in the port will overwrite this field as specified in the rx_src_tag_lo_sel field in the flow configuration table entry.	Configurable
15:8	Dest Tag – Hi	This field is application specific. During Packet reception, the DMA controller in the port will overwrite this field as specified in the rx_dest_tag_hi_sel field in the flow configuration table entry.	Configurable
15:0	Dest Tag – Lo	This field is application specific. During Packet reception, the DMA controller in the port will overwrite this field as specified in the rx_dest_tag_lo_sel field in the flow configuration table entry.	Configurable

Copyright © 2024 Texas Instruments Incorporated

Table 10-6. Host Packet Descriptor Linking Word 0 (PD Word 4)

Bits	Name	Description	Rx Overwrite
31:0	Next Descriptor Pointer LSB	The 32 LSBs of the 48-bit, 16-byte aligned (min), memory address of the next buffer descriptor in the packet. If the value of this pointer is zero then the current buffer is the last buffer in the packet. The host sets the Next Descriptor Pointer.	Yes

Table 10-7. Host Packet Descriptor Linking Word 1 (PD Word 5)

Bits	Name	Description	Rx Overwrite
31:20	RESERVED	Reserved	Yes
19:16	Next Descriptor Pointer Address Space Select	Effectively bits 51:48 of the address. The value given in this field will be output by the DMA masters on the casel pin which is used by the infrastructure as an identifier for which address space this particular memory region is located within. Address space 0 is the default unified address space for a given device. Address spaces 1-15 are used for alternate address maps which may be external to the device (PCIe/Hyperlink) or in other 'tiles' on large devices.	Yes
15:0	Next Descriptor Pointer MSB	The 16 MSBs of the 48-bit next descriptor pointer	Yes

Table 10-8. Host Packet Descriptor Buffer 0 Info Word 0 (PD Word 6)

Bits	Name	Description	Rx Overwrite
31:0	Buffer 0 Pointer LSB	The Buffer Pointer is the byte aligned memory address of the buffer associated with the buffer descriptor. This value will be written during reception. If the protocol specific words are placed at the beginning of the SOP buffer, this pointer will point to the PS words. The offset to the data in that case must be calculated by the consumer using the Protocol Specific Valid Word Count from Word 2. These are the 32 LSBs of the 48-bit buffer pointer.	Yes

Table 10-9. Host Packet Descriptor Buffer 0 Info Word 1 (PD Word 7)

Bits	Name	Description	Rx Overwrite
31:20	RESERVED	Reserved	Yes
19:16	Buffer 0 Pointer Address Space Select	Effectively bits 51:48 of the address. The value given in this field will be output by the DMA masters on the casel pin which is used by the infrastructure as an identifier for which address space this particular memory region is located within. Address space 0 is the default unified address space for a given device. Address spaces 1-15 are used for alternate address maps which may be external to the device (PCIe/Hyperlink) or in other 'tiles' on large devices.	Yes
15:0	Buffer 0 Pointer MSB	The 16 MSBs of the 48-bit buffer pointer	Yes

Table 10-10. Host Packet Descriptor Buffer 0 Info Word 2 (PD Word 8)

Bits	Name	Description	Rx Overwrite
31:22	RESERVED	Written to 0	Yes
21:0	Buffer 0 Length	The Buffer Length field indicates how many valid data bytes are in the buffer. Unused or protocol specific bytes at the beginning of the buffer are not counted in the Buffer Length field. This value will be overwritten during reception.	Yes

Table 10-11. Host Packet Descriptor Original Buffer Info Word 0 (PD Word 9)

Bits	Name	Description	Rx Overwrite
31:22	RESERVED		No
21:0	Original Buffer 0 Length	The Buffer Length field indicates the original size of the buffer in bytes. Data bytes are in the buffer. This value will not be overwritten during reception. This value is read by the Rx DMA to determine the actual buffer size as allocated by the host at initialization. Since the buffer length in PD Word 8 is overwritten by the Rx port during reception, this field is necessary to permanently store the buffer size information.	No

Copyright © 2024 Texas Instruments Incorporated

Table 10-12. Host Packet Descriptor Original Buffer Info Word 1 (PD Word 10)

Bits	Name	Description	Rx Overwrite
31:0	Original Buffer 0 Pointer LSB	The Buffer Pointer is the byte aligned memory address of the buffer associated with the buffer descriptor. This value will not be overwritten during reception. This value is read by the RX DMA to determine the actual buffer location as allocated by the host at initialization. Since the buffer pointer in PD Words 6/7 is overwritten by the Rx port during reception, this field is necessary to permanently store the buffer pointer information. This field contains the 32 LSBs of the 48-bit original buffer pointer	No

Table 10-13. Host Packet Descriptor Original Buffer Info Word 2 (PD Word 11)

Bits	Name	Description	Rx Overwrite
31:20	RESERVED	Reserved	Yes
19:16	Original Buffer 0 Pointer Address Space Select	Effectively bits 51:48 of the address. The value given in this field will be output by the DMA masters on the casel pin which is used by the infrastructure as an identifier for which address space this particular memory region is located within. Address space 0 is the default unified address space for a given device. Address spaces 1-15 are used for alternate address maps which may be external to the device (PCIe/Hyperlink) or in other 'tiles' on large devices.	Yes
15:0	Original Buffer 0 Pointer MSB	The 16 MSBs of the 48-bit original buffer pointer	No

Table 10-14. Host Packet Descriptor Extended Packet Info Block Word 0 (Optional)

Bits	Name	Description	Rx Overwrite
31:0	Timestamp Info	This field contains an application specific timestamp which can be used for traffic shaping in a QoS enabled system.	Configurable

Table 10-15. Host Packet Descriptor Extended Packet Info Block Word 1 (Optional)

Bits	Name	Description	Rx Overwrite
31:0	Software Info 0	This field stores software centric information that needs to travel with the packet through the stack. This information will be copied from the source descriptor to the destination descriptor whenever a prefetch operation is performed or when transferring through an infrastructure DMA node.	Configurable

Table 10-16. Host Packet Descriptor Extended Packet Info Block Word 2 (Optional)

Bits	Name	Description	Rx Overwrite
31:0	Software Info 1	This field stores software centric information that needs to travel with the packet through the stack. This information will be copied from the source descriptor to the destination descriptor whenever a prefetch operation is performed or when transferring through an infrastructure DMA node.	Configurable

Table 10-17. Host Packet Descriptor Extended Packet Info Block Word 3 (Optional)

Bits	Name	Description	Rx Overwrite
31:0	Software Info 2	This field stores software centric information that needs to travel with the packet through the stack. This information will be copied from the source descriptor to the destination descriptor whenever a prefetch operation is performed or when transferring through an infrastructure DMA node.	Configurable

Table 10-18. Host Packet Descriptor Protocol Specific Word N (Optional)

Bits	Name	Description	Rx Overwrite
31:0	Protocol Specific Data N	This field stores information which varies depending on the block and packet type.	Configurable

10.1.2.2.2 Host Buffer Descriptor

The Host Buffer Descriptor is identical in size and organization to a Host Packet Descriptor but does not include valid information in the packet level fields and does not include a populated region for protocol specific information. Host Buffer Descriptors are designed to be linked onto a Host Packet Descriptor or another Host Buffer Descriptor to provide support for unlimited scatter/gather type operations. Host Buffer Descriptors provide information about a single corresponding data buffer.

Every Host buffer descriptor stores the following information:

Copyright © 2024 Texas Instruments Incorporated

1. Pointer to the first valid byte in the data buffer
2. Length of the data buffer
3. Pointer to the next buffer descriptor in the packet

Host Buffer Descriptors always contain 48 bytes of required information. Since it is a requirement that it is possible to convert a Host descriptor between a Buffer Descriptor and a Packet Descriptor (by filling in the appropriate fields) in practice, Host Buffer Descriptors will be allocated using the same sizes as Host Packet Descriptors.

The Host Buffer Descriptor layout is shown below.

Buffer Reclamation Info (16 bytes)
Linking Info (8 bytes)
Buffer Info (12 bytes)
Original Buffer Info (12 bytes)

A Host Packet Descriptor and zero or more Host Buffer Descriptors may be linked together using the Next Descriptor Pointer fields to form packets. The last descriptor in a packet has a zero Next Descriptor Pointer. Each Host Buffer descriptor also points to a single data buffer.

The contents of the Host Buffer Descriptor words are detailed in Table 10-19 through Table 10-30.

Table 10-19. Host Buffer Descriptor Reserved Word 0 (BD Word 0)

Bits	Name	Description	Rx Overwrite
31:0	RESERVED	Reserved	No

Table 10-20. Host Buffer Descriptor Reserved Word 1 (BD Word 1)

Bits	Name	Description	Rx Overwrite
31:0	RESERVED	Reserved	No

Table 10-21. Host Buffer Descriptor Buffer Reclamation Info (BD Word 2)

Bits	Name	Description	Rx Overwrite
31:18	RESERVED		No
17	Early Return	This field indicates that each buffer pointer should be immediately returned to the specified queue when data transfer is started from the packet instead of waiting for the entire buffer to be emptied. This flag is used to enable in-place reception of packets on a Receive Channel while the source packet is in the process of being transferred on a Transmit Channel. 0 = Buffer/Packet descriptor pointers should only be returned after all reads have been completed 1 = Buffer/Packet descriptor pointers should be returned immediately upon fetching the descriptor and beginning to transfer data.	No
16	Return Push Policy	This field indicates how a Transmit or Receive DMA should return the descriptor pointers to the free queues. This field is encoded as follows: 0 = Descriptor must be returned to tail of queue 1 = Descriptor must be returned to head of queue This bit is only used when the Return Policy bit is set to 1. The Rx DMA will only use this field when an error occurs during reception and the DMA must return descriptors back to the free queue from which they came. This field must be set to 0 for descriptors which will be placed on queues managed by the Ring Accelerator.	No
15:0	Packet Return Queue Num	This field indicates the ring number within the Ring Accelerator that the descriptor is to be returned to after transmission is complete.	No

Copyright © 2024 Texas Instruments Incorporated

Table 10-22. Host Buffer Descriptor Reserved Word 3 (BD Word 3)

Bits	Name	Description	Rx Overwrite
31:0	RESERVED	Reserved	No

Table 10-23. Host Buffer Descriptor Linking Word 0 (BD Word 4)

Bits	Name	Description	Rx Overwrite
31:0	Next Descriptor Pointer LSB	The 32 LSBs of the 48-bit, 16-byte aligned (min), memory address of the next buffer descriptor in the packet. If the value of this pointer is zero then the current buffer is the last buffer in the packet. The host sets the Next Descriptor Pointer.	Yes

Table 10-24. Host Buffer Descriptor Linking Word 1 (BD Word 5)

Bits	Name	Description	Rx Overwrite
31:20	RESERVED		Yes
19:16	Next Descriptor Pointer Address Space Select	Effectively bits 51:48 of the address. The value given in this field will be output by the DMA masters on the casel pin which is used by the infrastructure as an identifier for which address space this particular memory region is located within. Address space 0 is the default unified address space for a given device. Address spaces 1-15 are used for alternate address maps which may be external to the device (PCIe/Hyperlink) or in other 'tiles' on large devices.	Yes
15:0	Next Descriptor Pointer MSB	The 16 MSBs of the 48-bit next descriptor pointer	Yes

Table 10-25. Host Buffer Descriptor Buffer N Info Word 1 (BD Word 6)

Bits	Name	Description	Rx Overwrite
31:0	Buffer 0 Pointer LSB	The Buffer Pointer is the byte aligned memory address of the buffer associated with the buffer descriptor. This value will be written during reception. If the protocol specific words are placed at the beginning of the SOP buffer, this pointer will point to the PS words. The offset to the data in that case must be calculated by the consumer using the Protocol Specific Valid Word Count from Word 2. These are the 32 LSBs of the 48-bit buffer pointer.	Yes

Table 10-26. Host Buffer Descriptor Buffer N Info Word 1 (BD Word 7)

Bits	Name	Description	Rx Overwrite
31:20	RESERVED		Yes
19:16	Buffer 0 Pointer Address Space Select	Effectively bits 51:48 of the address. The value given in this field will be output by the DMA masters on the casel pin which is used by the infrastructure as an identifier for which address space this particular memory region is located within. Address space 0 is the default unified address space for a given device. Address spaces 1-15 are used for alternate address maps which may be external to the device (PCIe/Hyperlink) or in other 'tiles' on large devices.	Yes
15:0	Buffer 0 Pointer MSB	The MSBs of the 48-bit buffer pointer	Yes

Table 10-27. Host Buffer Descriptor Buffer N Info Word 2 (BD Word 8)

Bits	Name	Description	Rx Overwrite
31:22	RESERVED		Yes
21:0	Buffer N Length	The Buffer Length field indicates how many valid data bytes are in the buffer. Unused or protocol specific bytes at the beginning of the buffer are not counted in the Buffer Length field. This value will be overwritten during reception.	Yes

Table 10-28. Host Buffer Descriptor Original Buffer Info Word 0 (BD Word 9)

Bits	Name	Description	Rx Overwrite
31:28	RESERVED		No
27:0	Original Buffer 0 Length	The Buffer Length field indicates the original size of the buffer in bytes. Data bytes are in the buffer. This value will not be overwritten during reception. This value is read by the Rx DMA to determine the actual buffer size as allocated by the host at initialization. Since the buffer length in BD Word 8 is overwritten by the Rx port during reception, this field is necessary to permanently store the buffer size information.	No

Copyright © 2024 Texas Instruments Incorporated

Table 10-29. Host Buffer Descriptor Original Buffer Info Word 1 (BD Word 10)

Bits	Name	Description	Rx Overwrite
31:0	Original Buffer 0 Pointer LSB	The Buffer Pointer is the byte aligned memory address of the buffer associated with the buffer descriptor. This value will not be overwritten during reception. This value is read by the RX DMA to determine the actual buffer location as allocated by the host at initialization. Since the buffer pointer in BD Words 6/7 is overwritten by the Rx port during reception, this field is necessary to permanently store the buffer pointer information. This field contains the 32 LSBs of the 48-bit original buffer pointer	No

Table 10-30. Host Buffer Descriptor Original Buffer Info Word 2 (BD Word 11)

Bits	Name	Description	Rx Overwrite
31:20	RESERVED		Yes
19:16	Original Buffer 0 Pointer Address Space Select	Effectively bits 51:48 of the address – treated special by the infrastructure as an identifier for which address space this particular memory region is located within. Address space 0 is the default unified address space for a given device. Address spaces 1-15 are used for alternate address maps which may be external to the device (PCIe/Hyperlink) or in other 'tiles' on large devices.	Yes
15:0	Original Buffer 0 Pointer MSB	The 16 MSBs of the 48-bit original buffer pointer	No

10.1.2.2.3 Monolithic Packet Descriptor

The Monolithic Packet Descriptor contains the following information:

- Indicator which identifies the descriptor as a Monolithic Packet Descriptor
- Source and Destination Tags
- Packet Type
- Packet Length
- Packet Error Indicator
- Packet Return Information
- Protocol Specific Region Size
- Protocol Specific Region Offset
- Protocol Specific Control/Status Bits
- Packet Data

The maximum size of a Monolithic Packet Descriptor is 4 MB. Of this, Monolithic Packet Descriptors always contain 16 bytes of required information and may also contain 16 bytes of software specific tagging information and up to 128 bytes (indicated in 4 byte increments) of protocol specific information. How much protocol specific information (and therefore the allocated size of the descriptors) is application dependent.

The Monolithic Packet descriptor layout is shown below.

Packet Info (16 bytes)
Extended Packet Info Block (Optional) Includes Timestamp, and software Data Words (16 bytes)
Protocol Specific Data (Optional) (0 to M bytes where M is a multiple of 4)
Null Region (0 to 511 bytes)
Packet Data (0 to 4M – 1)
Other software Data (Optional and User Defined)

The 'Other software Data' portion of the descriptor exists after all of the defined words and is reserved for use by the host software to store private data. This region is not used in any way by the hardware components in a UDMA system and these modules will not modify any bytes within this region.

The contents of the Monolithic Packet Descriptor words are detailed in Table 10-31 through Table 10-40

Copyright © 2024 Texas Instruments Incorporated

Table 10-31. Monolithic Packet Descriptor Word 0

Bits	Name	Description	Rx Overwrite
31:30	2'd2	Monolithic Packet Descriptor type.	Yes
29	Extended Packet Info Block Present	This field indicates the presence of the Extended Packet Info Block in the descriptor. 0 = EPIB is not present 1 = 16 byte EPIB is present	Yes
28	RESERVED		Yes
27:22	Protocol Specific Valid Word Count	This field indicates the valid number of 32-bit words in the protocol specific region. This is encoded in increments of 4 bytes as follows: 0 = 0 bytes 1 = 4 bytes … 16 = 64 bytes ... 32 = 128 bytes 33-63 = RESERVED	Yes
21:0	Packet Length	The length of the packet in bytes. The valid range for the packet length is 0 to 4M-1 bytes. If the packet length is set to 0, the port will not actually transmit any information. Instead, the port will perform buffer/descriptor reclamation as instructed in the return information in word 2.	Yes

Table 10-32. Monolithic Packet Descriptor Word 1

Bits	Name	Description	Rx Overwrite
31:28	Error Flags	This field contains error flags that can be assigned based on the packet type	Yes
27:24	Protocol Specific Flags	This field contains protocol specific flags/information that can be assigned based on the packet type.	Yes
23:14	Packet ID	Unique Packet ID for packet within FlowID	Yes
13:0	Flow ID	Flow ID within which this packet is being transported. The FlowID is used by downstream blocks to make decisions about packet steering and resource allocations. FlowIDs are also used to allow specific packets to be received into specific sets of buffers.	Yes

Table 10-33. Monolithic Packet Descriptor Word 2

Bits	Name	Description	Rx Overwrite
31:27	Packet Type	This field indicates the type of this packet and is encoded as follows: 0-31 = To Be Assigned	Yes
26:18	Data Offset	This field indicates the byte offset from byte 0 of this descriptor to the location where the valid data begins. On Rx, this value is set equal to the value for the SOP offset given in the Rx DMA channel's Monolithic control register. When a monolithic packet is processed, this value may be modified in order to add or remove bytes to from the beginning of the packet. The value for this field can range from 0-511 bytes. Note that the value of this field must always be greater than or equal to 4 times the value given in the Protocol Specific Valid Word Count field.	Yes
17	Early Return	This field indicates that each buffer pointer should be immediately returned to the specified queue when data transfer is started from the packet instead of waiting for the entire buffer to be emptied. This flag is used to enable in-place reception of packets on a Receive Channel while the source packet is in the process of being transferred on a Transmit Channel. 0 = Buffer/Packet descriptor pointers should only be returned after all reads have been completed 1 = Buffer/Packet descriptor pointers should be returned immediately upon fetching the descriptor and beginning to transfer data.	No

Copyright © 2024 Texas Instruments Incorporated

Table 10-33. Monolithic Packet Descriptor Word 2 (continued)

Bits	Name	Description	Rx Overwrite
16	Return Push Policy	This field indicates how a Transmit DMA should return the descriptor pointers to the free queues. This field is encoded as follows: 0 = Descriptor must be returned to tail of queue 1 = Descriptor must be returned to head of queue This bit is only used when the Return Policy bit is set to 1. This field must be set to 0 for descriptors which will be placed on queues managed by the Ring Accelerator.	No
15:0	Packet Return Queue Num	This field indicates the queue number that the descriptor is to be returned to after transmission is complete.	No

Table 10-34. Monolithic Packet Descriptor Word 3

Bits	Name	Description	Rx Overwrite
31:24	Source Tag – Hi	This field is application specific. During Packet reception, the DMA controller in the port will overwrite this field as specified in the rx_src_tag_hi_sel field in the flow configuration table entry.	Configurable
23:16	Source Tag – Lo	This field is application specific. During Packet reception, the DMA controller in the port will overwrite this field as specified in the rx_src_tag_lo_sel field in the flow configuration table entry.	Configurable
15:8	Dest Tag – Hi	This field is application specific. During Packet reception, the DMA controller in the port will overwrite this field as specified in the rx_dest_tag_hi_sel field in the flow configuration table entry.	Configurable
15:0	Dest Tag – Lo	This field is application specific. During Packet reception, the DMA controller in the port will overwrite this field as specified in the rx_dest_tag_lo_sel field in the flow configuration table entry.	Configurable

Table 10-35. Monolithic Extended Packet Info Word 0 (Optional)

Bits	Name	Description	Rx Overwrite
31:0	Timestamp Info	This field contains an application specific timestamp which can be used for traffic shaping in a QoS enabled system.	Configurable

This word is only present if the Extended Packet Info Block present bit is set in Word 0.

Table 10-36. Monolithic Extended Packet Info Word 1 (Optional)

Bits	Name	Description	Rx Overwrite
31:0	Software Info 0	This field stores software centric information that needs to travel with the packet through the stack. This information will be copied from the source descriptor to the destination descriptor whenever a prefetch operation is performed or when transferring through an infrastructure DMA node.	Configurable

This word is only present if the Extended Packet Info Block present bit is set in Word 0.

Table 10-37. Monolithic Extended Packet Info Word 2 (Optional)

Bits	Name	Description	Rx Overwrite
31:0	Software Info 1	This field stores software centric information that needs to travel with the packet through the stack. This information will be copied from the source descriptor to the destination descriptor whenever a prefetch operation is performed or when transferring through an infrastructure DMA node.	Configurable

This word is only present if the Extended Packet Info Block present bit is set in Word 0.

Table 10-38. Monolithic Extended Packet Info Word 3 (Optional)

Bits	Name	Description	Rx Overwrite
31:0	Software Info 2	This field stores software centric information that needs to travel with the packet through the stack. This information will be copied from the source descriptor to the destination descriptor whenever a prefetch operation is performed or when transferring through an infrastructure DMA node.	Configurable

Copyright © 2024 Texas Instruments Incorporated

These words if present immediately follow the Software Data Block Information.

Table 10-39. Monolithic Packet Descriptor Protocol Specific Word M (Optional)

Bits	Name	Description	Rx Overwrite
31:0	Protocol Specific Data N	This field stores information which varies depending on the packet type.	Configurable

The payload data follows the protocol specific words at an offset specified in the data offset field of Word 0.

Table 10-40. Monolithic Packet Descriptor Payload Data Words 0-N

Bits	Name	Description	Rx Overwrite
31:0	Packet Data N	These words store the packet payload data.	Yes

This field is endian specific. In other words, this is the only field in the descriptor which changes based on the endianness of the system.

10.1.2.2.4 Transfer Request Descriptor

The Transfer Request Descriptor contains the following information:

- Indicator which identifies the descriptor as a TR Descriptor
- Source and Destination Tags
- Packet Error Indicator
- Packet Return Information
- Set of one or more Transfer Request Records
- Set of one or more Transfer Response Records

Transfer Request Packet Descriptors always contain 16 bytes of required information and a variable number of Transfer Request/Transfer Response Records (request and response counts match).

The Transfer Request descriptor layout is shown below.

Packet Info (16 bytes)
Null (0-102 bytes to bring start of TR request array into natural alignment for memory fetch efficiency)
Array of Transfer Request Records (M bytes)
Array of Transfer Response Records (4 bytes per)

Table 10-41. Transfer Request Packet Descriptor Word 0

Bits	Name	Description	Rx Overwrite
31:30	2'd3	TR Packet Descriptor type.	Yes
29	RESERVED		Yes
28:20	Reload Count	Specifies what to do when the last entry is processed in this packet. This field specifies how many times to return to the Reload Index upon reaching the Last Entry. When an internal count is incremented to this value and the TR indicated by the Last Entry has been processed, this packet will be considered complete and the descriptor will be placed back on the return queue specified in Word 2. A value of 0x1FF indicates that a perpetual loop is desired. In this case, the loop count is considered infinite and the internal count will not be incremented. A teardown operation on the channel will cause the loop to be broken at the nearest iteration boundary.	

Copyright © 2024 Texas Instruments Incorporated

Table 10-41. Transfer Request Packet Descriptor Word 0 (continued)

Bits	Name	Description	Rx Overwrite
19:14	Reload Index	Specifies the value to set the current processing index to when the last entry is processed and the Reload Enable is set to 1. This is basically an absolute index to jump to on the 2nd and following passes through the TR packet.	
13:0	Last Entry	Specifies the index of the last valid entry in this packet	Yes

Table 10-42. Transfer Request Packet Descriptor Word 1

Bits	Name	Description	Rx Overwrite
31:28	Error Flags	This field contains error flags that can be assigned based on the packet type	Yes
27	RESERVED		Yes
26:24	Transfer Request Nominal Element Size	Specifies the stride between TR entries. The value in this field must be set large enough that any TR in the buffer will fit within the given dimension. TRs are expected to be placed on boundaries as given in this dimension. This field is also used to calculate the location where the Transfer Responses will be written back. This field is encoded as follows: 0 = 16 byte Transfer Request record size 1 = 32-byte Transfer Request record size 2 = 64-byte Transfer Request record size 3 = 128-byte Transfer Request record size 4-7 = RESERVED	Yes
23:14	Packet ID	Unique Packet ID for packet within FlowID	Yes
13:0	Flow ID	Flow ID within which this packet is being transported. The FlowID is used by downstream blocks to make decisions about packet steering and resource allocations. FlowIDs are also used to allow specific packets to be received into specific sets of buffers.	Yes

Table 10-43. Transfer Request Packet Descriptor Word 2

Bits	Name	Description	Rx Overwrite
31:17	RESERVED		No
16	Return Push Policy	This field indicates how a Transmit DMA should return the descriptor pointers to the free queues. This field is encoded as follows: 0 = Descriptor must be returned to tail of queue 1 = Descriptor must be returned to head of queue This bit is only used when the Return Policy bit is set to 1. This field must be set to 0 for descriptors which will be placed on queues managed by the Ring Accelerator	No
15:0	Packet Return Queue Num	This field indicates the queue number that the descriptor is to be returned to after transmission is complete.	No

Table 10-44. Transfer Request Packet Descriptor Word 3

Bits	Name	Description	Rx Overwrite
31:24	Source Tag – Hi	This field is application specific. During Packet reception, the DMA controller in the port will overwrite this field as specified in the rx_src_tag_hi_sel field in the flow configuration table entry.	Configurable

Copyright © 2024 Texas Instruments Incorporated

Table 10-44. Transfer Request Packet Descriptor Word 3 (continued)

Bits	Name	Description	Rx Overwrite
23:16	Source Tag – Lo	This field is application specific. During Packet reception, the DMA controller in the port will overwrite this field as specified in the rx_src_tag_lo_sel field in the flow configuration table entry.	Configurable
15:8	Dest Tag – Hi	This field is application specific. During Packet reception, the DMA controller in the port will overwrite this field as specified in the rx_dest_tag_hi_sel field in the flow configuration table entry.	Configurable
15:0	Dest Tag – Lo	This field is application specific. During Packet reception, the DMA controller in the port will overwrite this field as specified in the rx_dest_tag_lo_sel field in the flow configuration table entry.	Configurable

Table 10-45. Transfer Request Packet Descriptor Payload Words 0-N

Bits	Name	Description	Rx Overwrite
31:0	Transfer Control Record Array	Transfer Control Records. This is an array of records which encapsulate the Transfer Request and Transfer Response messages which are used by the UTC	Yes (partial)

10.1.2.3 Transfer Request Record

10.1.2.3.1 Overview

This section describes the standard transfer request (TR) format to initiate a DMA transfer. The TRs can support both half and full duplex with multiple dimensions as well as cache warms. The UTC (Universal Transfer Controller) does not have to support all types of TRs. Each TR will also generate a TR response. A TR can be sent to the UTC either through a PSI-L interface or an optionally supported Direct TR submission (previously QDMA) which will have Memory Mapped TR submission registers that can be written. The Memory Mapped TR submission registers further be classified by submission type either a single burst (atomic write) or multiple burst (non-atomic). The non-atomic TR will be sent when the submission register with word 0 is written.

Table 10-46. Terms and Definitions

Term	Definition
TR	A transfer request to move data.
CR	A cache operation request. A request to send messages to a specified cache controller to prepare the cache for a future operation.
UTC	Universal Transfer Controller the module in the UDMA architecture that handled the actual movement of data requested in the TR.

10.1.2.3.2 Addressing Algorithm

For a basic TR request the same algorithm is used. The innermost loop always consumes physically contiguous elements from memory. Its implicit dimension is 1 byte, this can be changed if the TR contains the formatting flags field and it is supported by the UTC. The pointer itself moves from fetch to fetch on the maximum byte alignments specified for the UTC, in increasing order. In each level outside the inner loop, the loop moves the pointer to a new location based on the size of that loop level's dimension.

10.1.2.3.2.1 Linear Addressing (Forward)

The following code illustrates the basic algorithm for a 4-level loop nest block move. This model assumes that it transfers data 1 byte at a time and the input and output block are the same size in all dimensions.

```
// sptr is a byte pointer.
// dptr is a byte pointer.
// Source address is indirect
if (TR_ISA) {
```


Copyright © 2024 Texas Instruments Incorporated

```
  sptr = *sptr;
}
// Destination address is indirect
if (TR_IDA) {
  dptr = *dptr;
}
// TR_TRIGX can be selected to come from Global events, Local event, or none.
// Check for trigger of TYPE0
if (TR_TRIG0_TYPE == TYPE0)  while(!TR_TRIG0);
if (TR_TRIG1_TYPE == TYPE0)  while(!TR_TRIG1);
for (i3 = 0; i3 < ICNT3; i3++)
{
  sptr3 = sptr; // save current position before entering next level
  dptr3 = dptr; // save current position before entering next level
  // Check for trigger of TYPE1
  if (TR_TRIG0_TYPE == TYPE1)  while(!TR_TRIG0);
  If (TR_TRIG1_TYPE == TYPE1)  while(!TR_TRIG1);
  for (i2 = 0; i2 < ICNT2; i2++) {
    sptr2 = sptr; // save current position before entering next level
    dptr2 = dptr; // save current position before entering next level
    // Check for trigger of TYPE2
    if (TR_TRIG0_TYPE == TYPE2)  while(!TR_TRIG0);
    if (TR_TRIG1_TYPE == TYPE2)  while(!TR_TRIG1);
    for (i1 = 0; i1 < ICNT1; i1++) {
      sptr1 = sptr; // save current position before entering next level
      dptr1 = dptr; // save current position before entering next level
      // Check for trigger of TYPE3
      if(TR_TRIG0_TYPE == TYPE3)  while(!TR_TRIG0);
      if(TR_TRIG1_TYPE == TYPE3)  while(!TR_TRIG1);
      for (i0 = 0; i0 < ICNT0; i0++) {
        // UTC can combine these in optimized burst aligned accesses.
        *dptr = *sptr;
        sptr = sptr++;
        dptr = dptr++;
      }
      // Update based on saved pointer for this level
      sptr = sptr1 + SDIM1;
      dptr = dptr1 + DDIM1;
    }
    // Update based on saved pointer for this level
    sptr = sptr2 + SDIM2;
    dptr = dptr1 + DDIM2;
  }
  // Update based on saved pointer for this level
  sptr = sptr3 + SDIM3;
  dptr = dptr1 + DDIM3;
}
```

This form of addressing allows programs to specify regular paths through memory in a small number of parameters. Additionally, it allows for various stall points through the loop to allow for hardware or software induced pausing of the transfer. The following table defines these parameters more explicitly.

Table 10-47. Addressing Parameters for a Basic Stream

Parameter	Definition
ICNT0	Number of iterations for the innermost loop dimension, loop level 0. That is, DIM0 = BYTES.
ICNT1	Number of iterations for the first level above the innermost loop, loop level 1.
SDIM1	Number of bytes between the starting points for consecutive iterations of loop level 1 for the source.
DDIM1	Number of bytes between the starting points for consecutive iterations of loop level 1 for the destination.
ICNT2	Number of iterations for loop level 2.
SDIM2	Number of bytes between the starting points for consecutive iterations of loop level 2 for the source.
DDIM2	Number of bytes between the starting points for consecutive iterations of loop level 2 for the destination.
ICNT3	Number of iterations for loop level 3.
SDIM3	Number of bytes between starting points for consecutive iterations of loop level 3 for the source.

Copyright © 2024 Texas Instruments Incorporated

Table 10-47. Addressing Parameters for a Basic Stream (continued)

Parameter	Definition
DDIM3	Number of bytes between the starting points for consecutive iterations of loop level 3 for the destination.

10.1.2.3.3 Transfer Request Formats

The Transfer Request format builds on the same structure adding fields to the TR to give more configuration options. The structure in Table 10-48 shows the 512-bit data structure that makes up the maximum sized UTC Transfer request.

Table 10-48. Transfer Request Template

word 15		word 14		word 13		word 12
DICNT3	DICNT2	DICNT1	DICNT0	DDIM3		DDIM2
word 11		word 10		word 9		word 8
DADDR				DDIM1		FMTFLAGS
word 7		word 6		word 5		word 4
DIM3		DIM2		ICNT3	ICNT2	DIM1
word3		word 2		word 1		word0
ADDR				ICNT1	ICNT0	FLAGS

The UTC defines a four-level loop nest for addressing elements within the TR, using the algorithms described in the Addressing algorithm. Most of the fields in the TR template map directly to the parameters in those algorithms.

Table 10-49. Transfer Request Fields

Field Name	Description	Size (bits)
FLAGS	TR flags that specify type of TR and how the TR should be handled. It also supports the TR triggering and output events.	32
ICNT0	Total loop iteration count for level 0 (innermost)	16
ICNT1	Total loop iteration count for level 1	16
ADDR	Starting address for the source data or destination data if it is a half-duplex write	64
DIM1	Signed dimension for loop level 1 for the source data	32
ICNT2	Total loop iteration count for level 2	16
ICNT3	Total loop iteration count for level 3 (outermost)	16
DIM2	Signed dimension for loop level 2	32
DIM3	Signed dimension for loop level 3	32
FMTFLAGS	Flags that tell how the data is formatted either between the input and the output or if the data should use different addressing schemes or sizes. These flags are OPTIONAL and only specific features may be supported by a given DMA implementation. The UDMA/UTC capabilities register specifies which features are supported.	32
DDIM1	Signed dimension for loop level 1 for the destination data	32
DADDR	Starting address for the destination of the data	64
DDIM2	Signed dimension for loop level 2 for the destination data	32
DDIM3	Signed dimension for loop level 3 for the destination data	32
DICNT0	Total loop iteration count for level 0 (innermost) used for destination	16
DICNT1	Total loop iteration count for level 1 used for destination	16
DICNT2	Total loop iteration count for level 2 used for destination	16
DICNT3	Total loop iteration count for level 3 used for destination	16

The TR assumes all iteration counts as unsigned integers, and all dimensions as signed integers representing byte offsets.

Copyright © 2024 Texas Instruments Incorporated

The template above fully specifies the type of elements, length, and dimensions of the transfer.

Since all transfers will not require all the fields the TRs have the type field which allow the number of words required to be sent for the TR to be reduced.

10.1.2.3.4 Flags Field Definition

Table 10-50 expands the 32-bit FLAGS field. The numbers above each field here indicate bit numbers within the field.

Table 10-50. Transfer Request Template FLAGS Field

31 24				23 16			
CONFIGURATION SPECIFIC FLAGS				CMD ID			
15 14	13 12	11 10	9 8	7 6	5	4	3 0
TRIGGER1_TYPE	TRIGGER1	TRIGGER0_TYPE	TRIGGER0	EVENT_SIZE	WAIT	STATIC	TYPE

The flags field fills in the remaining details about the stream, as follows:

Table 10-51. FLAGS Field Descriptions

Bit	Field	Description
0-3	TYPE	The TYPE of TR that is being sent 0: One dimensional data move 1: Two dimensional data move 2: Three dimensional data move 3: Four dimensional data move 4: Four dimensional data move with data formatting 5: Four dimensional Cache Warm 6-7: Reserved 8: Four Dimensional Block Move 9: Four Dimensional Block Move with Repacking 10: Two Dimensional Block Move 11: Two Dimensional Block Move with Repacking 12-14: Reserved 15: Four Dimensional Block Move with Repacking and Indirection
4	STATIC	This field is only valid on Type 8 and type 9 TRs The TR is static and will continually reload into the channel upon completion of the TR until a channel tear down.
5	WAIT	This field is only valid on Type 15 TRs. This field indicates whether or not this TR should ensure it completes before allowing the next TR to start on this channel. The encoding is as follows: 0 = Allow next TR to start immediately 1 = Wait for this TR to complete before allowing next TR to start When set, the next TR will not be considered triggered (regardless of any other trigger conditions) until all write status responses for the current TR have landed.

Copyright © 2024 Texas Instruments Incorporated

Table 10-51. FLAGS Field Descriptions (continued)

Bit	Field	Description
6-7	EVENT_SIZE	This is how often the TR will generate an output event. 0: Event is only generated with the TR is complete 1: Event is generated when the second inner most loop (ICNT1) is decremented by 1. 2: Event is generated when the third inner most loop (ICNT2) is decremented by 1. 3: Event is generated when the outer most loop (ICNT3) is decremented by 1.
8-9	TRIGGER0	This is one of two selectable triggers. The receipt of this trigger will enable the TR to be active for enough data transfer as specified by the TRIGGER0_TYPE Field. 0: No Trigger 1: Global Trigger 0 for the channel 2: Global Trigger 1 for the channel 3: Local Event for the channel
10-11	TRIGGER0_TYPE	This is the type of data transfer that will be enabled by receiving a trigger. 0: The second inner most loop (ICNT1) will be decremented by 1. 1: The third inner most loop (ICNT2) will be decremented by 1. 2: The outer most loop (ICNT3) will be decremented by 1. 3: The entire TR will be allowed to complete.
12-13	TRIGGER1	This is one of two selectable triggers. The receipt of this trigger will enable the TR to be active for enough data transfer as specified by the TRIGGER1_TYPE Field. 0: No Trigger 1: Global Trigger 0 for the channel 2: Global Trigger 1 for the channel 3: Local Event for the channel This field is reserved for a Direct TR.
14-15	TRIGGER1_TYPE	This is the type of data transfer that will be enabled by receiving a trigger. 0: The second inner most loop (ICNT1) will be decremented by 1. 1: The third inner most loop (ICNT2) will be decremented by 1. 2: The outer most loop (ICNT3) will be decremented by 1. 3: The entire TR will be allowed to complete.
16-23	CMD ID	The Command ID for the TR. This will be sent back with the response for the Channel controller to identify the specific TR. For a Direct TR this is the event number that will be generated based on the event trigger.
24-31	Configuration Specific Flags	These are flag bits that can be specified by the specific UTC configuration. These bits will remain undefined in the global TR format to allow for customization requirements that might be required in the specific UTC implementation.

The following sections expand on each of these fields.

Copyright © 2024 Texas Instruments Incorporated

10.1.2.3.4.1 Type: TR Type Field

The TR Type field gives the size of the TR and which fields are expected in the TR. Based on the type the number of words required to be sent to complete the TR can be decreased. The tables below show the minimum size for each TR as well if any fields will be treated as reserved.

Table 10-52. Transfer Request Minimum Size Type 0 One Dimensional Transfer

w 0	w 1	w 2	w 3	w 4	w 5	w 6	w 7	w 8	w 9	w 10	w 11	w 12	w 13	w 14	w 15
FLAGS	ICNT0	ADDR		RESERVED/NOT REQUIRED											

Table 10-53. Transfer Request Minimum Size Type 1 Two Dimensional Transfer

w 0	w 1	w 2	w 3	w 4	w 5	w 6	w 7	w 8	w 9	w 10	w 11	w 12	w 13	w 14	w 15
FLAGS	ICNT0/1	ADDR		DIM1	RESERVED/NOT REQUIRED										

Table 10-54. Transfer Request Minimum Size Type 2 Three Dimensional Transfer

w 0	w 1	w 2	w 3	w 4	w 5	w 6	w 7	w 8	w 9	w 10	w 11	w 12	w 13	w 14	w 15
FLAGS	ICNT0/1	ADDR		DIM1	ICNT2	DIM2	RESERVED/NOT REQUIRED								

Table 10-55. Transfer Request Minimum Size Type 3 Four Dimensional Transfer

w 0	w 1	w 2	w 3	w 4	w 5	w 6	w 7	w 8	w 9	w 10	w 11	w 12	w 13	w 14	w 15
FLAGS	ICNT0/1	ADDR		DIM1	ICNT2/3	DIM2	DIM3	RESERVED/NOT REQUIRED							

Table 10-56. Transfer Request Minimum Size Type 4 Four Dimensional Transfer with Formatting

w 0	w 1	w 2	w 3	w 4	w 5	w 6	w 7	w 8	w 9	w 10	w 11	w 12	w 13	w 14	w 15
FLAGS	ICNT0/1	ADDR		DIM1	ICNT2/3	DIM2	DIM3	FMT FLAGS	RESERVED/NOT REQUIRED						

Table 10-57. Transfer Request Minimum Size Type 5 Cache Warm

w 0	w 1	w 2	w 3	w 4	w 5	w 6	w 7	w 8	w 9	w 10	w 11	w 12	w 13	w 14	w 15
FLAGS	ICNT0/1	ADDR		DIM1	ICNT2/3	DIM2	DIM3	CACHE FLAGS	RESERVED/NOT REQUIRED						

Table 10-58. Transfer Request Minimum Size Type 8 Four Dimensional Block Copy

w 0	w 1	w 2	w 3	w 4	w 5	w 6	w 7	w 8	w 9	w 10	w 11	w 12	w 13	w 14	w 15
FLAGS	ICNT0/1	ADDR		DIM1	ICNT2/3	DIM2	DIM3	FMT FLAGS	DDIM1	DADDR		DDIM2	DDIM3	RESERVED/NOT REQUIRED	

Table 10-59. Transfer Request Minimum Size Type 9 Four Dimensional Block Copy with Repacking

w 0	w 1	w 2	w 3	w 4	w 5	w 6	w 7	w 8	w 9	w 10	w 11	w 12	w 13	w 14	w 15
FLAGS	ICNT0/1	ADDR		DIM1	ICNT2/3	DIM2	DIM3	FMT FLAGS	DDIM1	DADDR		DDIM2	DDIM3	DICNT0/1	DICNT2/3

Table 10-60. Transfer Request Minimum Size Type 10 Two Dimensional Block Copy

w 0	w 1	w 2	w 3	w 4	w 5	w 6	w 7	w 8	w 9	w 10	w 11	w 12	w 13	w 14	w 15
FLAGS	ICNT0/1	ADDR		DIM1	RESERVED			FMT FLAGS	DDIM1	DADDR		RESERVED/NOT REQUIRED			

Table 10-61. Transfer Request Minimum Size Type 11 Two Dimensional Block Copy with Repacking

w 0	w 1	w 2	w 3	w 4	w 5	w 6	w 7	w 8	w 9	w 10	w 11	w 12	w 13	w 14	w 15
FLAGS	ICNT0/1	ADDR		DIM1	RESERVED			FMT FLAGS	DDIM1	DADDR		RESERVED		DICNT0/1	RESERVED/NOT REQUIRED

Copyright © 2024 Texas Instruments Incorporated

Table 10-62. Transfer Request Minimum Size Type 15 Four Dimensional Block Copy with Repacking and Indirection Support

w 0	w 1	w 2	w 3	w 4	w 5	w 6	w 7	w 8	w 9	w 10	w 11	w 12	w 13	w 14	w 15
FLAGS	ICNT0/1	ADDR	DIM1	ICNT2/3	DIM2	DIM3	RESERVED	DDIM1	DADDR	DDIM2	DDIM3	DICNT0/1	DICNT2/3		

10.1.2.3.4.2 STATIC: Static Field Definition

The static field is used to allow for a single TR to be reused repeatedly. If the static bit is set then the UTC upon completing the specified TR will restart the TR with the same values as were in the initial TR. This TR will stay active until the channel is torn down. The initiating of a tear down of the channel will clear the STATIC field in the active TR and it will be removed from the channel FIFO after the TR completion.

Note

If a TR is submitted to a channel FIFO with the STATIC field set then any TRs placed in the FIFO after that will NOT be run until the teardown has been initiated.

10.1.2.3.4.3 EVENT_SIZE: Event Generation Definition

The UTC allows for an event to be generated at specified intervals for each TR. As the TR passes through the various loops in the addressing algorithm it will generate an event that then can be routed to other event receivers, such as other channels or interrupts to processors.

Table 10-63. Event Size Encoding

Value	Event Type	When it Fires
0	Completion Event	When the TR is complete and all status for the TR has been received.
1	ICNT1 Decrement	Type 0: When the last data transaction is sent for the TR. Type 1-11: When ICNT1 is decremented
2	ICNT2 Decrement	Type 0 – 1,10-11: When the last transaction is sent for the TR. All Other Types: When ICNT2 is decremented
3	ICNT3 Decrement	Type 0 – 2,10-11: When the last transaction is sent for the TR. All Other Types: When ICNT3 is decremented

10.1.2.3.4.4 TRIGGER INFO: TR Triggers

The TR allows for two different triggers to be set to allow for the data transfer to be prevented or halted through the process of transferring the data. The triggers are specified by selecting a type of trigger and the size of the transfer that can be allowed for the receipt of a given trigger. The triggers themselves in the UTC are collected on a per channel basis and for each of three sources. Anytime the trigger event is received the internal counter is incremented. The trigger event will not be cleared until the specified block has started its transfer and it will **only** clear the triggers that are active.

CAUTION

This does allow for one TR in the channel to use global event 0 and the next TR to use global event 1. If while the first TR is running global event 0 can increment and will decrement each time the TR reaches the specified level. At the same time global event 1 will be incremented whenever it is received but it will not decrement until the next TR runs. If the global event 1 counter reaches overflows then an error event will NOT be generated but the event will be lost.

10.1.2.3.4.5 TRIGGERX_TYPE: Trigger Type

The trigger type sets the value of TR_TRIGX as shown in the addressing algorithm description. The TR_TRIGX value allows for the temporary halting of the TR until the expected event has been received. After the given loop has been entered the selected trigger will be decremented.

Copyright © 2024 Texas Instruments Incorporated

10.1.2.3.4.6 TRIGGERX: Trigger Selection

The TR is able to select up to 2 of 3 trigger sources for a given TR. The events can come from the dedicated local event on the UTC itself or from two assigned global events that come from the PSI-L interface on the UTC. The trigger counters are always enabled so that events can be received prior to the TR getting loaded. The trigger counters are only decremented when the trigger is active and the TR has scheduled the first transfer of the transaction.

10.1.2.3.4.7 CMD ID: Command ID Field Definition

The command ID is used as a unique identifier for the TR. The value is not used during the data transfer but is only used by the TR Completion Response to allow for hardware and software to identify the specific TR.

10.1.2.3.4.8 Configuration Specific Flags Definition

The configuration specific flags are specific to a given TR type. If a TR type does not require additional flags the field should be filled with 0's. Currently only Types 0-4 and Type 15 support the following configuration specific flags:

Table 10-64. Configuration Specific Flags

Bit	Field	Description
0	ISA	Indirect Source Address 0: Source address in TR is a directly usable pointer to the data 1: Source address in TR is a pointer to a 64-bit location that contains the actual pointer to the data
1	IDA	Indirect Destination Address 0: Destination address in TR is a directly usable pointer to the data 1: Destination address in TR is a pointer to a 64-bit location that contains the actual pointer to the data
2	SUPR_EVT	Suppress Event Output: 0 = Output events will be generated according to FLAGS.EVENT_SIZE field 1 = No output events will be generated for the duration of this TR execution This field is only valid on split and type 15 TRs (Type 0-3,15)
3	Reserved	Reserved for Future use.
6:4	EOL	This field is only valid on split TRs (Type 0-3). On source (Read) split TRs, this field specifies whether or not the EOL delimiters should be produced on the Tx PSI-L bus. The encodings of this field for source split TRs is as follows: 0 = SOL/EOL match SOP/EOP 1 = SOL/EOL boundaries are each ICNT0 bytes 2 = SOL/EOL boundaries are each ICNT0×ICNT1 bytes 3 = SOL/EOL boundaries are each ICNT0×ICNT1×ICNT2 bytes 4 = SOL/EOL boundaries are each ICNT0×ICNT1×ICNT2×ICNT3 bytes On destination (Write) split TRs, this field specifies how to handle EOL delimiters when they are encountered on the Rx (write) side of a TR. EOL delimiters can be produced from the Tx (read) side of a block copy operation or from a remotely paired peripheral when operating in split TR mode. The encodings of this field are as follows: 0: Ignore EOL 1: Line length is icnt0 bytes. Clear any remaining ICNT0 bytes and increment ICNT1 by 1 2: Line length is icnt0*icnt1 bytes. Clear any remaining ICNT0/1 bytes and increment ICNT2 by 1 3: Line length is icnt0*icnt1*icnt2 bytes. Clear any remaining ICNT0/1/2 bytes and increment ICNT3 by 1 5-7: RESERVED

Copyright © 2024 Texas Instruments Incorporated

Table 10-64. Configuration Specific Flags (continued)

Bit	Field	Description
7	EOP	This TR should generate an EOP on the streaming interface for any downstream packet oriented consumers to denote that a 'packet' of data is complete 0: No EOP flag will accompany the last PSI-L data phase associated with transfers from this TR 1: On egress the DMA will set the EOP flag coincident with transferring the last of the data for this TR. If the TR data does not complete an entire PSI-L data phase then the remaining bytes in the data phase will be skipped and the internal FIFO pointer will be updated to begin packing new data on a new data phase boundary.

10.1.2.3.5 TR Address and Size Attributes

The fields described below all describe the data transfer that needs to be made as described in the Linear Memory Addressing Block Move Algorithm.

10.1.2.3.5.1 ICNT0

The ICNT0 is the number of elements to transfer in the inner loop of the TR. If the TR type does not contain a FMTFLAGS field then this count is assumed to be the number of bytes to transfer. If the FMTFLAGS field is included in the types then the number of bytes to be transferred will be ICNT0 times the Element size rounded up to the nearest byte.

10.1.2.3.5.2 ICNT1

The ICNT1 field is the loop count for the second innermost loop count as defined in the Addressing algorithm.

10.1.2.3.5.3 ADDR

The address location is the initial address that will be accessed at the start of the transfer. All of the dimensions will also be based off of this value. This address can either physical or virtual based upon settings in the DMA channel Configuration register.

While the ADDR field is 64 bits wide, the usable extent of the address on a K3 system is 48 bits of absolute offset plus a 4-bit address space selector which indicates 1 of 16 different orthogonal address spaces that the pointer is referencing within. The format of the ADDR field is given in Table 10-65.

Table 10-65. ADDR Field Format

Bits	Subfield	Description
63:52	Reserved	Reserved
51:48	Address Space Select	Effectively bits 51:48 of the address. The value given in this field will be output by the DMA masters on the casel pin which is used by the infrastructure as an identifier for which address space this particular memory region is located within. Address space 0 is the default unified address space for a given device. Address spaces 1-15 are used for alternate address maps which may be external to the device (PCIe/Hyperlink) or in other 'tiles' on large devices.
47:0	Address	The 48-bit source or source/destination starting address for the transfer. This address will be interpreted to be either physical, virtual, or intermediate physical based on the tx_atype or rx_atype field in the DMA Tx/Rx channel's Configuration Register.

10.1.2.3.5.4 DIM1

This is the offset of the address from the initial address for the first access of the second loop. This will be added to the address from the loop before. This is a signed value so that the address can be both above and below the initial address.

10.1.2.3.5.5 ICNT2

This is the count for the third innermost loop in the addressing algorithm.

10.1.2.3.5.6 ICNT3

This is the count for the outermost loop in the addressing.

Copyright © 2024 Texas Instruments Incorporated

10.1.2.3.5.7 DIM2

This is the offset of the address from the initial address to the next address in the third innermost loop. This is a signed vales so that the address can be both above and below the previous address.

10.1.2.3.5.8 DIM3

This is the offset of the address from the initial address to the next address in the outermost loop. This is a signed values so that the address can be both above and below the previous address.

10.1.2.3.5.9 DDIM1

This is the offset of the destination address from the initial destination address for the first access of the second loop. This will be added to the destination address from the loop before. This is a signed value so that the destination address can be both above and below the initial destination address.

10.1.2.3.5.10 DADDR

The destination address location is the initial destination address that will be accessed at the start of the transfer. All of the dimensions will also be based off of this value. This address can either physical or virtual based upon settings in the DMA channel Configuration register

While the DADDR field is 64 bits wide, the usable extent of the address on a K3 system is 48 bits of absolute offset plus a 4 bit address space selector which indicates 1 of 16 different orthogonal address spaces that the pointer is referencing within. The format of the DADDR field is given as follows:

Table 10-66. DADDR Field Format

Bits	Subfield	Description
63:52	Reserved	Reserved
51:48	Address Space Select	Effectively bits 51:48 of the address. The value given in this field will be output by the DMA masters on the casel pin which is used by the infrastructure as an identifier for which address space this particular memory region is located within. Address space 0 is the default unified address space for a given device. Address spaces 1-15 are used for alternate address maps which may be external to the device (PCIe/Hyperlink) or in other 'tiles' on large devices.
47:0	Address	The 48-bit starting destination address for the transfer. This address will be interpreted to be either physical, virtual, or intermediate physical based on the tx_atype field in the DMA Tx channel's Configuration Register.

10.1.2.3.5.11 DDIM2

This is the offset of the destination address from the initial destination address for the first access of the third loop. This will be added to the destination address from the loop before. This is a signed value so that the destination address can be both above and below the initial destination address.

10.1.2.3.5.12 DDIM3

This is the offset of the destination address from the initial destination address for the first access of the outer loop. This will be added to the destination address from the loop before. This is a signed value so that the destination address can be both above and below the initial destination address.

10.1.2.3.5.13 DICNT0

This is the number of elements to use in the destination if the TR is repacking the data. This number reflects the number of elements to be transferred.

10.1.2.3.5.14 DICNT1

This is the number of times to execute the second loop to use in the destination transfer.

10.1.2.3.5.15 DICNT2

This is the number of times to execute the third loop to use in the destination transfer.

Copyright © 2024 Texas Instruments Incorporated

10.1.2.3.5.16 DICNT3

This is the number of times to execute the fourth loop to use in the destination transfer. The value of DICNT0 × DICNT1 × DICNT2 × DICNT3 must equal ICNT0 × ICNT1 × ICNT2 × ICNT3. If any of the values are zero they are NOT included in the multiplication.

10.1.2.3.6 FMTFLAGS

The format flags are used to specify special formatting of the transfer. Table 10-67 shows the definition of the sub-fields in the FMTFLAGS field.

Table 10-67. Transfer Request Template FMTFLAGS Field

31															16
AMODE SPECIFIC															
15	14	13	12	11				8	7			4	3	2	0
RESERVED		SECTR		DFMT					ELTYPE				DIR	AMODE	

The flags field fills in the remaining details about the stream, as follows:

Table 10-68. FMTFLAGS Field Descriptions

Bit	Field	Description
16-31	AMODE SPECIFIC	This 16-bit field is defined based on the addressing mode used as selected in the AMODE field.
14-15	Reserved	Reserved for Future use
12-13	SECTR	A secondary TR is required and the format of the secondary TR.
8-11	DFMT	The data reformatting function to apply in the middle of the TR
4-7	ELTYPE	The type of element and its size and potential output size for the innermost loop. If this is not specified in the TR then a byte is the assumed element.
3	DIR	For a block type transfer does the addressing mode apply for source or destination. The other one will use linear. If it is a non-block type TR then this field is ignored
0-2	AMODE	The Addressing Mode of TR that is being sent 0: Linear Addressing 1: Circular Addressing 2-7: Reserved for future use

10.1.2.3.6.1 AMODE: Addressing Mode Definition

The addressing mode field allows for different algorithms to be used other than the default linear addressing mode described in the Addressing algorithm.

10.1.2.3.6.1.1 Linear Addressing

This is the addressing that is used if there is no FMTFLAGS field in the TR.

10.1.2.3.6.1.2 Circular Addressing

Circular addressing modifies how the UTC performs address arithmetic so that addresses remain within a power-of-2 sized window. Circular addressing works by holding upper address bits constant during an address update, while allowing the lower address bits to vary. This contrasts with the default—linear addressing—which allows all of the address bits to vary.

The UTC TR provides address mode selection for each level of loop nest. Each level can select between linear addressing or circular addressing with one of two circular block sizes.

Conceptually, selectable addressing mode support replaces all address arithmetic with a function similar to the following.

```
enum addr_mode_t
{
LINEAR,// Linear addressing mode
```


Copyright © 2024 Texas Instruments Incorporated

```
CIRC0,// Circular addressing with block size 0
CIRC1// Circular addressing with block size 1
};
uint64_t circ_mask_0;// Circular addressing mask for block size 0
uint64_t circ_mask_1;// Circular addressing mask for block size 1
uint64_t address_add( uint64_t base, int32_t offset, addr_mode_t mode )
{
uint64_t new_addr = base + offset;
if ( mode == LINEAR )
return new_addr;
// Look up address mask based on selected circular buffer size.
// Mask contains 1s for bits that remain fixed, and 0s elsewhere.
uint64_t addr_mask = mode == CIRC0 ? circ_mask_0 : circ_mask_1;
return ( base & addr_mask ) | ( new_addr & ~addr_mask );
}
```

The address_add primitive then takes the place of plain addition for all TR address computations in this mode. The following example illustrates address_add.

```
for (i3 = 0; i3 < ICNT3; i3++)
{
ptr3 = ptr; // save current position before entering next level
for (i2 = 0; i2 < ICNT2; i2++)
{
ptr2 = ptr; // save current position before entering next level
for (i1 = 0; i1 < ICNT1; i1++)
{
ptr1 = ptr; // save current position before entering next level
for (i0 = 0; i0 < ICNT0; i0++)
{
fetch( ptr, ELEM_BYTES );
ptr = address_add( ptr, ELEM_BYTES, addr_mode_0 );
}
// Update based on saved pointer for this level
ptr = address_add( ptr1, DIM1, addr_mode_1 );
}
// Update based on saved pointer for this level
ptr = address_add( ptr2, DIM2, addr_mode_2 );
}
// Update based on saved pointer for this level
ptr = address_add( ptr3, DIM3, addr_mode_3 );
}
```

The values for the circular addressing can be selected by using the 16 bits of the AMODE specific Field as described in the AMODE SPECIFIC Addressing Mode Field.

10.1.2.3.6.2 DIR: Addressing Mode Direction Definition

This field is used if the TR TYPE specifies the TR is for a block move if the addressing mode specified in the AMODE field applies to the source addressing or the destination addressing. A value of 0 means the source addressing will use the method selected in the AMODE field and the destination address will use the default linear addressing. A value of 1 means that the destination addressing will use the method selected in the AMODE field and the source addressing will use the default linear addressing. If the TR TYPE is not a block transfer then this field will be ignored.

10.1.2.3.6.3 ELTYPE: Element Type Definition

The Element Type field allows the user to specify the basic unit that is used for the innermost loop. If the FMTFLAGS field is not present then the UTC will assume a value of 0 which is a byte size element. The element type also allows for the some expansion or contraction of the element size from the source to the destination.

Table 10-69. ELTYPE Encoding

Encoding	Definition
0	1 Byte per element
1	1.5 Bytes(12 bits) per element
2	2 Bytes per element
3	3 Bytes per element
4	4 Bytes per element

Copyright © 2024 Texas Instruments Incorporated

Table 10-69. ELTYPE Encoding (continued)

Encoding	Definition
5	8 Bytes per element
6	16 Bytes per element
7	32 Bytes per Element
8	1 Byte per Input Element 2 Bytes per Output Element
9	1.5 Bytes per Input Element 2 Bytes per Output Element
10	2 Bytes per Input Element 1 Byte per Output Element
11	2 Bytes per Input Element 1.5 Bytes per Output Element
12-15	Reserved for Future Use

10.1.2.3.6.4 DFMT: Data Formatting Algorithm Definition

The Data Formatting Algorithm is a method to specify a manipulation of the data between how it is read and how it is sent. If the TR Type is not a block operation TR then this field will be ignored. Otherwise Table 10-70 specifies what the encodings meaning.

Table 10-70. DFMT Encoding

Encoding	Definition
0	No Change. The input and output block will remain identical
1	Constant Copy. The input block is not an address but the address is up to a 64-bit constant
2	Transpose. The inner and second most inner loops are swapped so that rows become columns and columns become rows.
3	Reverse. The data in the row will be accessed in the reverse of the order that it is read. So the largest address read in a row will be the first address written.
4	Reverse Transpose. The data will be written in the reverse of the order that is read as well as transposed. So the largest address read in a row will be the address in the first line being written out.
5-15	Reserved for Future Use

10.1.2.3.6.5 SECTR: Secondary Transfer Request Definition

The Secondary Transfer Request field allows the user to specify if the TR requires an additional TR located in memory. If this is used then the address given in the TR is the pointer to the Secondary TR and the secondary TR will have a pointer to the actual final TR.

Table 10-71. Secondary Transfer Request Format

Encoding	Size	Definition
0	0	The TR does not require a secondary TR
1	64 bytes	The TR will fetch a 64-byte Secondary TR prior to the initial read.
2	128 bytes	The TR will fetch a 128-byte Secondary TR prior to the initial read.
3	TBD	Reserved for future use

10.1.2.3.6.5.1 Secondary TR Formats

The secondary TR format is a TR that is fetched by the UTC when it is executing a TR to fill in additional information that does not fit in the original TR. It is not included in the standard TR as the information required in it is not required until execution time and is kept separate so the Channel FIFO space can be minimized. The first four words of the Secondary TR are always the real start address of the transfer followed by a Secondary TR Flags field. The remaining words in the secondary TR are defined by the Secondary TR TYPE from the Secondary TR Flags field.

Table 10-72. Secondary Transfer Request Template for 64-Byte Secondary TR

word 15	word 14	word 13	word 12

Copyright © 2024 Texas Instruments Incorporated

Table 10-72. Secondary Transfer Request Template for 64-Byte Secondary TR (continued)

Type Specific			
word 11	word 10	word 9	word 8
Type Specific			
word 7	word 6	word 5	word 4
Type Specific			
word 3	word 2	word 1	word 0
Type Specific	SECONDARY TR FLAGS	ADDR	

10.1.2.3.6.5.2 Secondary TR FLAGS

The next diagram expands the generic 32-bit Secondary TR FLAGS field. The numbers above each field here indicate bit numbers within the field.

Table 10-73. SECONDARY TR FLAGS Field Definition

31			16
SEC_TR_TYPE_SPECIFIC			
15	4	3	0
SEC_TR_TYPE_SPECIFIC		SEC_TR_TYPE	

The flags field fills in the remaining details about the stream, as follows:

Table 10-74. FLAGS Field Descriptions

Bit	Field	Description
31-4	SEC_TR_TYPE_SPECIFIC	Reserved for the SEC_TR_TYPE to define based on its requirements
3-0	SEC_TR_TYPE	The TYPE of SEC_TR 0: Multiple Buffer Interleave 1-15: Reserved

The following sections expand on each of these fields.

10.1.2.3.6.5.2.1 SEC_TR_TYPE: Secondary TR Type Field

The Secondary TR Type field is used to state the purpose of the secondary TR. This can be defined

10.1.2.3.6.5.2.2 Multiple Buffer Interleave

The multiple buffer interleave is used to have a single TR plus Secondary TR that can define a fetch where a single line is selected from multiple buffers and then sent as a single buffer. This can be used along with transpose of the data to present data in slices optimal for processing the same algorithm on different data sets at the same time. An example of this can be seen in Figure 10-4.

Copyright © 2024 Texas Instruments Incorporated

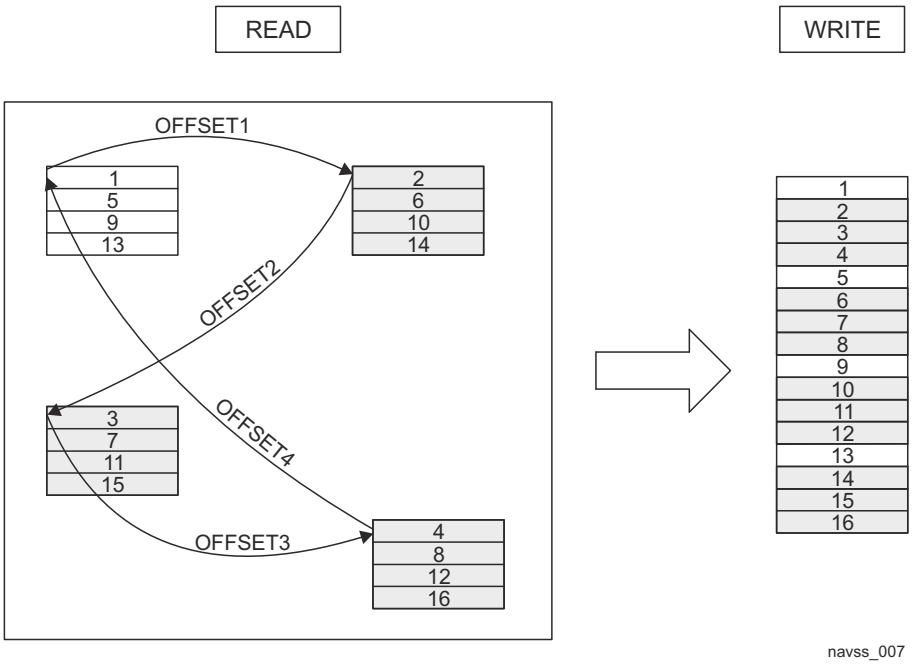


Figure 10-4. Example of Multiple Buffer Interleaving Read with 4 Offsets

The case above provides the ability to minimize the number of TRs submitted to make the transfer but the interleaved Read main advantage comes when used with the transpose mode enabled. Figure 10-5 shows the advantage of the interleaved mode along with the transpose mode to allow for maximum size transfer on both read and write transactions.

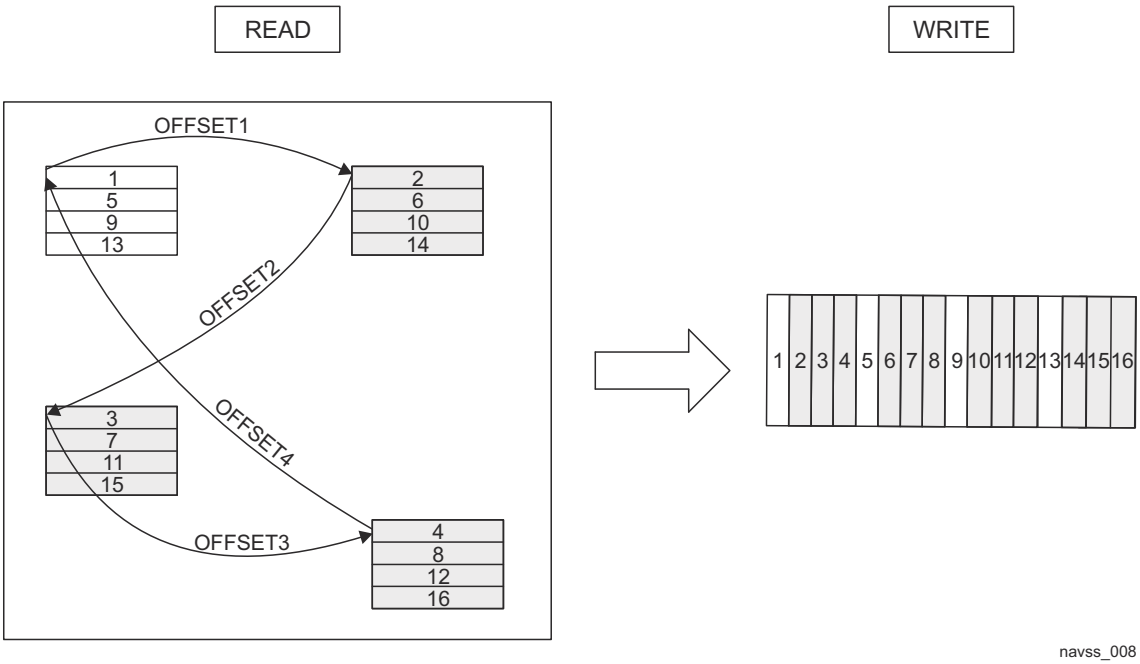


Figure 10-5. Example of Interleaved Read with 4 Offsets and Transpose Enabled

As the figure shows the data that is written out now contains one element from each block of memory and can be read in directly to processing units.

Table 10-75. SECONDARY TR FLAGS Field Definition

31	16
SEC_TR_TYPE_SPECIFIC	

Copyright © 2024 Texas Instruments Incorporated

Table 10-75. SECONDARY TR FLAGS Field Definition (continued)

15 6	5 4	3 0
SEC_TR_TYPE_SPECIFIC	NUMOFFSET	0 (Multiple Interleave Type)

The flags field fills in the remaining details about the stream, as follows:

Table 10-76. Multiple Buffer Interleave FLAGS Field Descriptions

Bit	Field	Description
6-31	Reserved	Reserved for Future Use
4-5	NUMOFFSET	The encoded value for the number of offsets. 0: 4, 1: 8, 2: 16
0-3	SEC_TR_TYPE	0: Multiple Buffer Interleave

Secondary TR TYPE from the Secondary TR Flags field.

Table 10-77. Multiple Buffer Interleave Secondary Transfer Request Template 128 Byte Secondary TR

word 31	word 30	word 29	word 28
RESERVED			
word 27	word 26	word 25	word 24
RESERVED			
word 23	word 22	word 21	word 20
RESERVED			
word 19	word 18	word 17	word 16
OFFSET15	OFFSET14	OFFSET13	OFFSET12
word 15	word 14	word 13	word 12
OFFSET11	OFFSET10	OFFSET9	OFFSET8
word 11	word 10	word 9	word 8
OFFSET7	OFFSET6	OFFSET5	OFFSET4
word 7	word 6	word 5	word 4
OFFSET3	OFFSET2	OFFSET1	OFFSET0
word3	word 2	word 1	word0
RESERVED	SECONDARY TR FLAGS	ADDR	

Multiple Buffer Interleave will work by setting the SECTR field in the TR presented to the channel to be either 64 or 128 bytes depending on the number of offsets required. An offset number of 4 or 8 will require 64 bytes while 16 offsets will require 128 bytes. When the channel becomes active it will see that the SECTR field is nonzero and fetch the SECTR of the size specified. The first access from the channel will be a read of the secondary TR as specified in the address field of the TR. Upon receiving the Secondary TR the channel will parse the type field and seeing that it is a Multiple Buffer interleave will load the offset index fields for the channel. Additionally it will load the ADDRESS field from the secondary TR into the Address of the actual TR to allow it to process as normal.

The first address fetch will then be to this SEC_TR_ADDRESS. After finishing the inner loop the next address will be SEC_TR_ADDRESS + OFFSET0.

Note

ICNT1 should be equal to the height of the transfer blocks times the number of offsets.

10.1.2.3.6.6 AMODE SPECIFIC: Addressing Mode Field

The AMODE Specific Fields are defined based upon the Addressing Mode being used as specified in AMODE: Addressing Mode Definition.

Copyright © 2024 Texas Instruments Incorporated

10.1.2.3.6.6.1 Circular Address Mode Specific Flags

If the circular addressing mode is selected in AMODE the definition of the upper 16 bits of the FMTFLAGS field are used to select the type of addressing for each entry in the loop and to define 2 unique masks that can be used for the circular buffering.

The following example code shows the method used to compute the masks referred to by the address_add function described in the‘ Address Arithmetic with Selectable Linear/Circular Addressing Modes section.

```
// Circular addressing masks.
// 1 bits in the mask indicate address bits that remain fixed.
// 0 bits in the mask indicate address bits that are allowed to vary.
uint64_t circ_mask_0;// mask for circular addressing block 0
uint64_t circ_mask_1;// mask for circular addressing block 1
void compute_circular_address_masks( int cbk0, int cbk1 )
{
int block_size_0 = cbk0 + 9;// power-of-2 of block size in bytes
int block_size_1 = cbk0 + cbk1 + 10;
if ( block_size_1 > 31 ) block_size_1 = 32;// clamp to 4GB maximum size
circ_mask_0 = (~0ULL) << block_size_0;
circ_mask_1 = (~0ULL) << block_size_1;
}
```

Example 3-4 Circular Address Mask Computation

Table 10-78. Transfer Request Template FMTFLAGS AMODE SPECIFC Circular Addressing Field

31 30	29 28	27 26	25 24	23 20	19 16
AM3	AM2	AM1	AM0	CBK1	CBK0

Table 10-79. Circular Address Mode Specific Flags Field Descriptions

Bit	Field	Description
16-19	CBK0	Specifies the circular block 0 value as used in Circular Address Mask Computation
20-23	CBK1	Specifies the circular block 1value as used in Circular Address Mask Computation
24-25	AM0	The address mode to use with ICNT0 loop
26-27	AM1	The address mode to use with ICNT1 loop
28-29	AM2	The address mode to use with ICNT2 loop
30-31	AM3	The address mode to use with ICNT3 loop

10.1.2.3.6.6.1.1 CBK0 and CBK1: Circular Block Size Selection

The CBK0 and CBK1 fields set the circular block sizes for circular addressing. CBK0 directly determines the circular block size for block 0. CBK1 combines with CBK0 to determine the circular block size for block 1. Specifically, the streaming engine sets block 1’s size according to CBK0 + CBK1 + 1. Therefore, circular block 1 is always larger than circular block 0.

The following table illustrates the resulting valid block sizes for circular block 0 and 1. For circular block 0, the use the value of CBK0 directly. Circular block 0 supports block sizes ranging from 512 bytes to 16M bytes. For circular block 1, use the value of CBK0 + CBK1 + 1. Circular block 1 supports sizes ranging from 1K bytes to 4G bytes.

Table 10-80. Circular Block Size Decoding

Encoded Block Size	Decoded Block Size	Encoded Block Size	Decoded Block Size	Encoded Block Size	Decoded Block Size	Encoded Block Size	Decoded Block Size
0	512	8	128K	16	32M	24	Reserved
1	1K	9	256K	17	64M	25	Reserved
2	2K	10	512K	18	128M	26	Reserved
3	4K	11	1M	19	1M	27	Reserved
4	8K	12	1M	20	1M	28	Reserved
5	16K	13	1M	21	1M	29	Reserved
6	32K	14	1M	22	1M	30	Reserved

Copyright © 2024 Texas Instruments Incorporated

Table 10-80. Circular Block Size Decoding (continued)

Encoded Block Size	Decoded Block Size	Encoded Block Size	Decoded Block Size	Encoded Block Size	Decoded Block Size	Encoded Block Size	Decoded Block Size
7	64K	15	1M	23	1M	31	Reserved

10.1.2.3.6.6.1.2 Amx: Addressing Mode Selection

The fields AM0 through AM3 control the addressing modes for each of the 4 dimensions. The UTC supports the following addressing modes:

Table 10-81. AMX Address Mode Selection Encoding

AMX	Meaning
00b	Linear addressing
01b	CKB0 (Circular Buffer 0 is used)
10b	CKB1 (Circular Buffer 1 is used)
11b	Reserved

Each dimension has its own addressing mode, independent of the other dimensions. This permits any mixture of linear and circular addressing among the 4dimensions.

When a given dimension selects linear addressing, the UTC uses unmodified two's complement arithmetic to update addresses for that loop level.

When a given dimension selects circular addressing via CBK0 or CBK1, the UTC uses the circular address arithmetic described above in the Address Arithmetic with Selectable Linear/Circular Addressing Modes section.

10.1.2.3.6.7 Cache Flags

If the TR type is a cache warm the format flags field is replaced with a Cache Flags field.

Table 10-82 shows the breakdown of the 32 bit Cache Flags field.

Table 10-82. Transfer Request Template CACHEFLAGS Field

31	24	23	16
CACHE OPERATION		RESERVED	
15	8	7	0
RESERVED		CACHE ID	

The flags field fills in the remaining details about the stream, as follows:

Table 10-83. FMTFLAGS Field Descriptions

Bit	Field	Description
24-31	Cache Operation	0: Prewarm the Cache 1: Prewarm MMU 2-255: Reserved for future use
8-23	Reserved	Reserved for future use
0-7	Cache ID	The Cache number to perform the operation on. 0-254: Cores. 255: L3 Cache

10.1.2.4 Transfer Response Record

Upon completing a TR the UTC will send back a TR Response to the UDMA-C. The UTC TR response format is a single 32-bit word as shown in Table 10-84.

Table 10-84. Transfer Request Response Template STATUS FLAGS Field

31	24	23		16
CONFIGURATION SPECIFIC STATUS		CMD ID		
15	8	7	4	3	0
RESERVED		STATUS_FIELD		STATUS_TYPE

Copyright © 2024 Texas Instruments Incorporated

Table 10-85. STATUS FLAGS Field Descriptions

Bit	Field	Description
31:24	Configuration Specific Flags	These are flag bits that can be specified by the specific UTC configuration. These bits will remain undefined in the global TR format to allow for customization requirements that might be required in the specific UTC implementation. For UDMA channels which are capable of interfacing to endpoint peripherals (or via a PDMA) the format of these flags are as follows: 31:28 error_flags – the error_flags field as is provided across PSI-L 27:24 ps_flags – the ps_flags field as is provided across PSI-L
23:16	CMD ID	The Command ID for the TR. This will be sent back with the response for the Channel controller to identify the specific TR.
15:8	Reserved	Reserved for Future use
7:4	STATUS_INFO	Information that is unique based on the STATUS_TYPE returned
3:0	STATUS_TYPE	The completion status of the TR.

10.1.2.4.1 STATUS Field Definition

The Status field is made up of two sub-fields STATUS_TYPE and STATUS_INFO. A value of 0 means it completed as requested any other value means an error has occurred. Table 10-86 states the legal STATUS_TYPEs and the use of the STATUS_INFO in each case.

10.1.2.4.1.1 STATUS_TYPE Definition

The STATUS_TYPE field is used to determine what type of status is being returned. Depending on the type of Status it might additionally have information in the STATUS_FIELD to give more details about the specific reason why the status was returned.

Table 10-86. STATUS_TYPE Encoded Values

Value	Error Type	Completion	STATUS FIELD Definition
0	None	Complete	None
1	Transfer Error	None to Partial	CBA Non-Complete Status Received
2	Aborted Error	None to Partial	None
3	Submission Error	None	Submission Error Type
4	Unsupported Feature	None	Feature Type
5	Transfer Exception	Partial	Exception Type
6	Teardown Flush	None	None
7-15	Reserved	Unknown	Reserved

10.1.2.4.1.1.1 Transfer Error

A transfer ERROR occurs anytime that one of the CBA transactions performed by the UTC returns a status other than complete. The UTC may continue the transfer or may abort the transfer and return back to an IDLE state. Once the transfer is finished (aborted or complete) the UTC will send back the TR Response with the STATUS TYPE of Transfer Error. The STATUS FIELD will contain the 3 bit CBA Status field the upper bit specifies if it was a read status (1) or a write status (0).

10.1.2.4.1.1.2 Aborted Error

An aborted error occurs if the PSI-L interface asserts the drop signal prior to the completion of the TR. The UTC will return back to an IDLE state and send back the TR Response with the STATUS TYPE. If the UTC receives a transfer error after receiving the abort the transfer error should be reported instead of the Abort Error.

10.1.2.4.1.1.3 Submission Error

A submission error is returned for a TR that is received that cannot be run. The STATUS_INFO field specifies the type of submission errors.

Copyright © 2024 Texas Instruments Incorporated

Table 10-87. Submission Error STATUS_INFO Encodings

Value	Submission Error Type
0	ICNT0 was 0
1	Channel FIFO was full when TR received
2	Channel is not owned by the submitter. Either CC submitting to Direct or Direct submitting to a CC channel
3-15	Reserved

10.1.2.4.1.1.4 Unsupported Feature

An unsupported feature error is returned for a TR that is received that cannot be run because it specifies a feature that is optional and not supported by the UTC that received the TR. The STATUS FIELD specifies the feature that was not supported. If it specifies multiple features that were not supported the UTC implementation can determine which type it wants to return

Table 10-88. Unsupported Feature STATUS_INFO Encodings

Value	Submission Error Type
0	TR Type not supported
1	STATIC not supported
2	EOL not supported
3	CONFIGURATION SPECIFIC not supported
4	AMODE not supported
5	ELTYPE not supported
6	DFMT not supported
7	SECTR not supported
8	AMODE SPECIFIC Field not supported
9-15	Reserved

10.1.2.4.1.1.5 Transfer Exception

A transfer exception is returned for a TR that completed but experienced a known exception during reception. The STATUS_INFO field specifies the type of transfer exception that was encountered.

Table 10-89. Transfer Exception STATUS_INFO Encodings

Value	Transfer Exception Type
0	EOP on incoming data stream was encountered prematurely (short packet)
1	EOP on incoming data stream was encountered late (long packet)
2-15	Reserved

10.1.2.4.1.1.6 Teardown Flush

Split mode UTC Rx channels (write channels) can respond with a teardown flush in the case that they have received a teardown message, all data has been transferred , and TRs have been prefetched (in anticipation of more data being received). In this case, the UTC will simply return the prefetched TR to the completion queue with the status type set to teardown flush.

10.1.2.5 Queues

Queues are used to hold either values or references that need to be passed between components in the system. These components could be software or hardware. Queues are implemented by the Ring Accelerator module.

10.1.2.5.1 Queue Types

10.1.2.5.1.1 Transmit Queues (Pass By Reference)

Tx channels use packet queues referred to as “transmit queues” to store the packets that are waiting to be transmitted. Tx channels require one or more packet queues dedicated to each transmit channel for this purpose. Multiple queues per channel may facilitate Quality of Service (QoS) in some applications. Pass by reference Tx Queues only pass a pointer to a descriptor (and optional information like the packet length). Pass

Copyright © 2024 Texas Instruments Incorporated

by reference Tx Queues can contain pointers to Host, Monolithic, or TR packet descriptors (depending on Tx channel mode configuration).

10.1.2.5.1.2 Transmit Queues (Pass By Value)

Pass by value Tx Queues are used for direct Transfer Request submission on Tx channels which are configured for 3rd party pass by value mode. Entries on a pass by value Tx Queue include the entire Transfer Request message and optional protocol specific information. Pass by value queues are only supported by the Ring Accelerator.

10.1.2.5.1.3 Transmit Completion Queues (Pass By Reference)

Tx channels also use packet queues referred to as “transmit completion queues” to return packets to the host after they are transmitted. The number of queues required by the channel or system for this purpose is driven by the requirements of the garbage collection software within the driver. Tx completion queues are only used when the packet descriptor indicates that the packet should be returned to a queue instead of directly recycling the descriptors to their queues of origin. Pass by reference Tx Completion Queues only pass a pointer to a descriptor.

10.1.2.5.1.4 Transmit Completion Queues (Pass By Value)

Pass by value Tx Completion Queues are used to pass the Transfer Response messages that are returned when a channel is configured in 3rd Party pass by value mode. Entries on a pass by value Tx Completion Queue include the Transfer Response message. Pass by value queues are only supported by the Ring Accelerator.

10.1.2.5.1.5 Receive Queues

Rx ports use packet queues referred to as “receive queues” to forward completed received packets to the host or another peer port entity. Rx ports may be configured in various ways to forward the received packets onto any number of receive queues. Rx ports may choose to queue their receive packets based strictly on Rx channel, protocol type, priority, direct forwarding requirements (i.e. to another ports Tx queue) or any combination of these and this is application specific. In many cases the receive queue is in fact also a transmit queue for another peer entity. It is just a matter of semantics with respect to which port is referenced in the system.

10.1.2.5.1.6 Free Descriptor Queues

Rx ports use queues referred to as “free descriptor queues” to allocate empty monolithic descriptors. The entries on the Free Descriptor Queues do not have any buffers attached as buffers are allocated as part of the descriptor for monolithic type packets.

10.1.2.5.1.7 Free Descriptor/Buffer Queues

Rx ports use queues referred to as “free descriptor/buffer queues” to allocated empty host type descriptors. The entries on the Free Descriptor/Buffer Queues have pre-attached empty buffers whose size and location are described in the “original buffer information” fields in the descriptor. The host is required to allocate both the descriptor and buffer and pre-link them prior to adding a descriptor to a free descriptor/buffer queue.

10.1.2.5.2 Ring Accelerator Queues Implementation

Work packets which are stored in a pass by reference queue managed by the Ring Accelerator are shown in Figure 10-6.

Copyright © 2024 Texas Instruments Incorporated

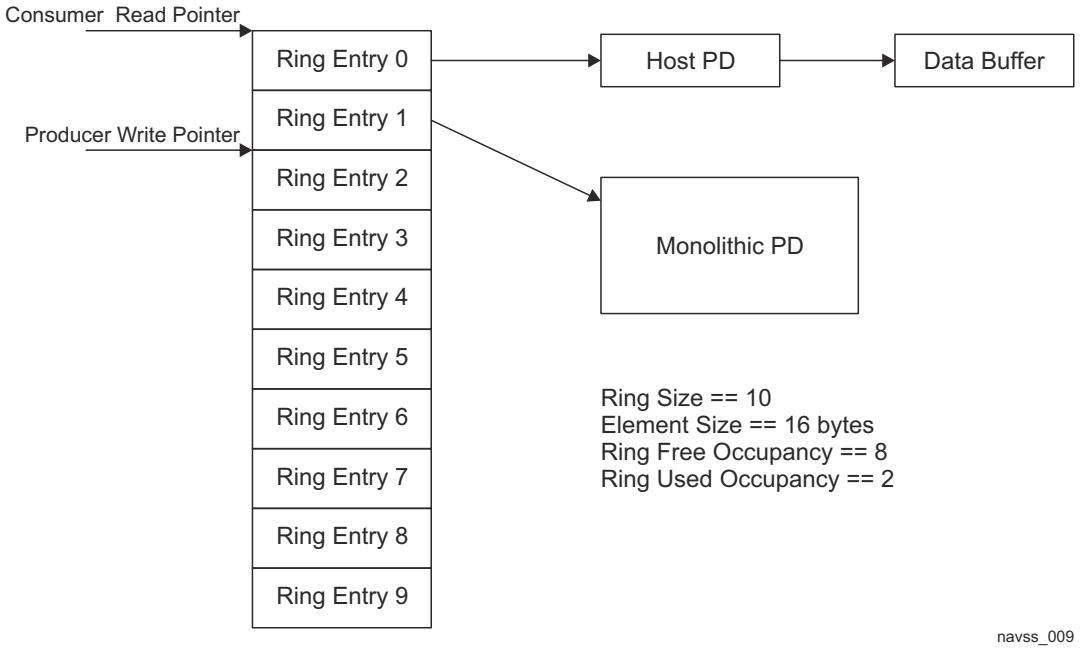


Figure 10-6. Ring Accelerator Based Queue Structure

Work packets stored in a pass by value queue managed by the Ring Accelerator are shown in Figure 10-7.

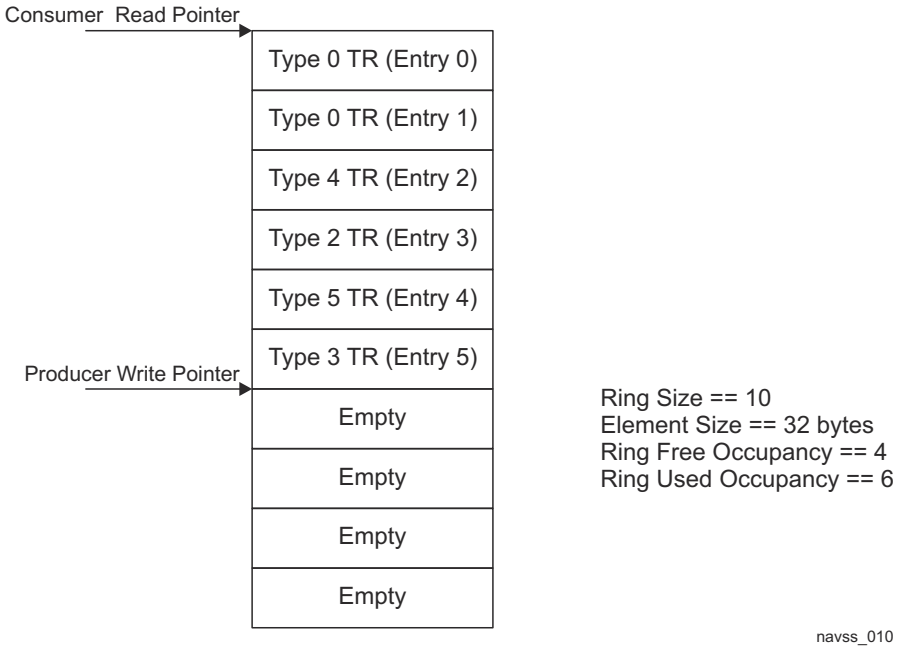


Figure 10-7. Ring Accelerator Based Direct Transfer Request Queue Structure

10.1.3 Operational Description

10.1.3.1 Resource Allocation

Queues, rings, memory, interrupts, DMA channels, and flow table entries are shared resources within the data movement architecture and their ownership and use must be managed in order to provide stable, safe, and secure operation. The K3 platform provides channelized firewalls which can be used to protect arrayed resources (like queues, rings, channel control registers, etc.) from unwanted/unauthorized use as well as restricted access to the firewall configurations themselves. It is outside the scope of this document to define how the resource management system functions but it is assumed that all of these resources are isolate-able and secure-able.

Copyright © 2024 Texas Instruments Incorporated

10.1.3.2 Ring Accelerator Operation

10.1.3.2.1 Queue Initialization

The base address, size and element size for each ring must be initialized via the provided registers prior to using the ring. Ring initialization is typically done only on channel setup as once initialized the rings can be used indefinitely. Optionally, an internal initialization machine may be provided to set default values for the ring address, size, and element sizes. This capability enables the rings to be used during boot without configuration. Not all rings will necessarily support this feature and if required must be configured at design time. Whether or not this machine is present, all rings are able to be re-configured at run time through the provided registers.

10.1.3.2.2 Queuing packets (Exposed Ring Mode)

Queuing of packets onto a packet queue using the Ring Accelerator is logically equivalent but requires a different set of transactions from queuing packets in queue mode. The producer (entity which is going to queue an entry onto the ring) maintains a current write pointer and a free entry occupancy count for each ring it controls. To queue an entry the producer will write the entry contents (typically a pointer to the Packet Descriptor along with optional sideband information) to the memory location pointed to by the current write pointer and will decrement its free entry occupancy. After the producer has confirmed that the data has actually landed it memory, it will then write to the doorbell register for the ring to increment the used occupancy count for the ring.

More than one entry can be queued with a single doorbell write as the entry count in the doorbell write indicates how many entries have been queued.

10.1.3.2.3 De-queuing packets (Exposed Ring Mode)

The consumer (entity which is going to de-queue an entry from the ring) maintains a current read pointer and a used entry occupancy count for each ring it controls. To de-queue an entry the consumer will read the entry contents from the memory location pointed to by the current read pointer and will decrement its used entry occupancy. The consumer will then write to the doorbell register for the ring to increment the free occupancy count for the ring.

10.1.3.2.4 Queuing packets (Queue Mode)

To queue an entry the producer will write the entry contents (typically a pointer to the Packet Descriptor along with optional sideband information) to the Proxy thread location allocated by the system, with the queue offset for the queue to access. The write within the queue offset to the completion offset (final byte of the data) will cause the entire data to be written to the RA queue.

10.1.3.2.5 De-queuing packets (Queue Mode)

To de-queue an entry the consumer will read the entry contents from the Proxy thread location allocated by the system, with the queue offset for the queue to access. The proxy will read the entry from the RA queue. The consumer can then read the entire contents of the entry. The completion offset (final byte of the data) will cause the proxy to flush that entry (so that it can fetch a new one from the RA upon the next read).

10.1.3.3 UDMA Internal Transmit Channel Setup (All Packet Types)

After a reset or a previous teardown operation but before queuing packets to a channel the host must initialize the channel's Tx Port DMA State. The host initializes the channel Tx Port DMA State by writing to the Tx Channel Configuration Register A. The Host may choose to write the enable bit in the Tx Channel Configuration Register A at the same time or after it has written all of the channel parameters but note that every write to the Tx Channel Configuration Registers will overwrite the channel state for all bytes that are enabled for the write transaction.

It should be noted that very little configuration information is required for a UDMA Tx channel compared to previous versions of UDMA. This is because UDMA has transitioned to a full modeless model for channel operation where each packet descriptor specifies the descriptor type, packet type, and the packet return parameters. Packet queues in UDMA may include combinations of host and monolithic descriptors in any order.

After a Transmit channel has been set up, packets can be added to the Queues for the channel to begin the transmit operation. The following sections describe how the transmit operations are performed for the various descriptor types.

Copyright © 2024 Texas Instruments Incorporated

10.1.3.4 UDMA Internal Transmit Channel Teardown (All Packet Types)

An Tx channel teardown is initiated by the host by writing the TX_TEARDOWN bit in the Tx Channel N Realtime Control Register. When the host initiates teardown, it can choose to perform either a graceful (no data loss) or forced teardown of the channel depending on the setting of the TX_FORCED_TEARDOWN bit in the Tx Channel N Realtime Control Register.

If the TX_FORCED_TEARDOWN bit is clear, a normal teardown has been initiated. In this case, the UDMA will do the following:

1. Stops performing any additional prefetches for the channel.
2. Completes any packets normally for which a pointer/descriptor/TR prefetch has already been performed
3. Sets the tdown bit on the EOP data phase of the last packet to be sent (if any traffic was pending when teardown was initiated) or sends a zero byte packet with the tdown, sop, and eop bits asserted. This signals the destination thread that all of the data has been sent.
4. Clears the channel enable in the Tx Channel N Realtime Control Register.
5. Resets the channel state (including scoreboards, FIFOs, counters, statistics, etc.) to their after reset values.
6. Sends a single, 4-byte, teardown complete message to the queue specified in the Tx Channel Completion Queue register for the channel. The format of this message is as follows:

Bits	Field	Value	Description
31	Forced	0	Indicates that the teardown was graceful and data was not lost.
30:14	-	0	RESERVED
13:4	Channel ID	channel number	Indicates which channel the teardown completed on
3:0	Teardown Indicator	0x1	Indicates that a teardown has completed – normal pointers must be 16 byte aligned so this value indicates this is a teardown

If the TX_FORCED_TEARDOWN bit is set, a destructive teardown has been initiated. In this case, the UDMA will do the following:

1. Suspends any triggers to allow the channel to operate even if the trigger source is no longer functioning.
2. Completes any packets which have been prefetched with or without transferring any data or generating events (the implementation has the option of doing whichever is simplest for that implementation).
3. Clears the channel enable in the Tx Channel N Realtime Control Register.
4. Resets the channel state (including scoreboards, FIFOs, counters, statistics, etc.) to their after reset values.
5. Sends a single, 4-byte, teardown complete message to the queue specified in the Tx Channel Completion Queue register for the channel. The format of this message is as follows:

Bits	Field	Value	Description
31	Forced	1	Indicates that the teardown was not graceful. Data was potentially lost.
30:14	-	0	RESERVED
13:4	Channel ID	channel number	Indicates which channel the teardown completed on
3:0	Teardown Indicator	0x1	Indicates that a teardown has completed – normal pointers must be 16 byte aligned so this value indicates this is a teardown

The host may issue a teardown on any channel at any time, regardless of whether the channel is actively receiving a packet or not.

The host determines that a teardown is complete by periodically polling the teardown and enable bits for the channel or by waiting to receive the teardown record on the queue specified in the Tx Channel Completion Queue register for the channel.

10.1.3.5 UDMA-P Transmit Channel Pause

Setting the TX_PAUSE bit in the Tx Channel N Realtime Control Register will suspend the channel from arbitration resulting in a halting of the flow of data. Clearing this bit will cause the channel to be added back into the arbitration list. Pausing a channel has no other destructive side effects (other than potential underrun/overrun conditions from a downstream data sink or upstream data source.

Copyright © 2024 Texas Instruments Incorporated

10.1.3.6 UDMA-P Transmit Operation (Host Packet Type)

After a channel has been set up it can begin to be used to transmit packets. Packet transmission in Host mode involves the following steps:

1. The Host is made aware of one or more chunks of data in memory that need to be transmitted as a packet. This may involve directly sourcing data from the Host or it may involve data which has been forwarded from another data source in the system.
2. The Host allocates and populates a host packet descriptor. The host will initialize the following fields within the packet descriptor:
   a. Descriptor Type (set to Host)
   b. Packet Length indicating the total number of bytes that are to be read from all of the buffers for this packet.
   c. Source Tag
   d. Destination Tag (application specific)
   e. Packet Type
   f. Any protocol specific flags for the given packet type
   g. Buffer Pointer with the byte aligned address of the first chunk of buffer data
   h. Buffer Length with the number of bytes in the first chunk of buffer data
   i. Descriptor reclamation policy fields indicating the mode and queue number for recycling the packet after transmission is complete.
   j. The Next Descriptor Pointer with the address of the next descriptor in this packet. If this is the last chunk of data in the packet, this field must be set to zero
   k. Any protocol specific descriptor sections that are required for the given packet type or system configuration
3. The Host allocates and populates host buffer descriptors as necessary to point to any remaining chunks of data that belong to this packet. The host will initialize the following fields within the host buffer descriptor:
   a. Buffer Descriptor reclamation policy fields (if the intention is to have the DMA automatically recycle the buffer descriptors as they are transmitted).
   b. Buffer Pointer with the byte aligned address of the given chunk of buffer data
   c. Buffer Length with the number of bytes in the given chunk of buffer data
   d. The Next Descriptor Pointer with the address of the next descriptor in this packet. If this is the last chunk of data in the packet, this field must be set to zero
4. The Host writes queues the packet onto one of the Transmit Queues for the desired DMA channel. Channels may provide more than one Tx Queue and may provide a particular prioritization policy between the queues. This behavior is application specific and is controlled by the DMA controller/scheduler implementation.
5. The Ring Accelerator provides a level sensitive status signal for the queue which indicates if any packets are currently pending. This level sensitive status line is sent to the hardware block which is responsible for scheduling DMA operations.
6. The DMA controller is eventually brought into context for the corresponding channel and begins to process the packet.
7. The DMA controller reads the packet descriptor pointer from the Ring Accelerator
8. The DMA controller reads the packet descriptor from memory
9. Step 9:
   - If the return policy bit (PD Word 2, bit 15) is cleared, the DMA controller empties each buffer in sequence by transmitting the contents in one or more block data moves. As each buffer is emptied, the DMA will read the next buffer descriptor to obtain the pointer and size of the data buffer as well as the pointer to the next descriptor in the chain.
   - If the return policy bit is set, the DMA controller empties each buffer in sequence by transmitting the contents in one or more block data moves.
10. Step 10:
    - If the return policy bit (PD Word 2, bit 15) is cleared, when all data for the packet has been transmitted as specified in the packet size field, the DMA will write the pointer to the packet descriptor to the queue specified in the return queue number fields of the packet descriptor.
    - If the return policy bit is set, as each buffer is emptied the DMA will write the buffer descriptor pointer to the queue specified in the return queue number field of the buffer descriptor. The DMA will also read the

Copyright © 2024 Texas Instruments Incorporated

next buffer descriptor to obtain the pointer and size of the data buffer, the return queue information for the descriptor, as well as the pointer to the next descriptor in the chain.

11. Step 11:
    - If the return policy bit (PD Word 2, bit 15) is cleared, after the Packet Descriptor pointer has been written, the Ring Accelerator will indicate the status of the Tx Completion Queue by sending an up event .
    - If the return policy bit is set, when all data for the packet has been transmitted as specified in the packet size field, if the current buffer is not marked as end of packet (via a null next descriptor pointer), the DMA will continue to walk the list returning buffer descriptors to the appropriate queues as described in step 10.
12. Step 12:
    - If the return policy bit (PD Word 2, bit 15) is cleared, the Interrupt Aggregator receives the up event and sets the corresponding bit in the interrupt status register as programmed in the interrupt mapping registers. This in turn causes an interrupt to the Host to be generated.
    - If the return policy bit is set, when all buffers have been returned, the DMA will write the pointer to the packet descriptor to the queue specified in the return queue number fields of the packet descriptor.
13. If the return policy bit (PD Word 2, bit 15) is cleared, host responds to status change from the Ring Accelerator (via the Interrupt Aggregator) and performs garbage collection as necessary for the packet.
14. If the return policy bit (PD Word 2, bit 15) is cleared, once the garbage collection causes the queue to become empty, the Ring Accelerator will send a down event to the Interrupt Aggregator which will clear the corresponding bit in the Interrupt Status Register and potentially de-assert the interrupt line.

If the return policy bit (PD Word 2, bit 15) is cleared, the complete process is illustrated on Figure 10-8.

If the return policy bit is set, the complete process is illustrated on Figure 10-9.

Figure 10-8 shows a diagram of the host packet transmit operation – complete packet return.

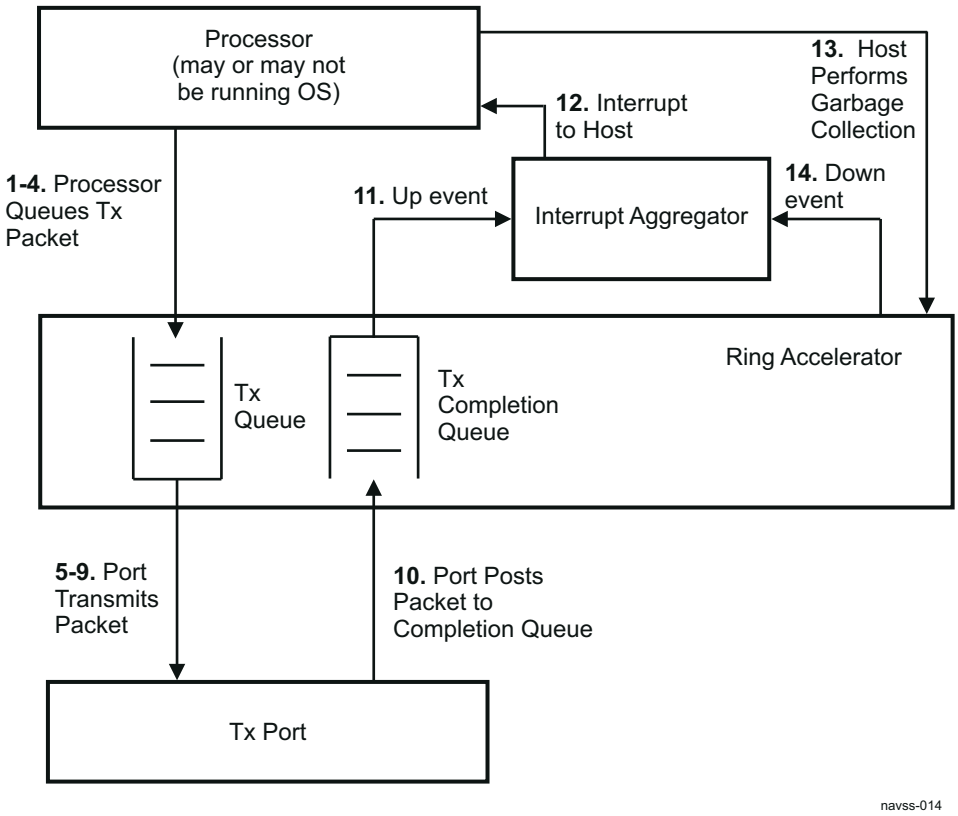


Figure 10-8. Host Packet Tx Operation – Complete Packet Return

Figure 10-9 shows a diagram of the host packet transmit operation – automatic buffer recycling packet return.

Copyright © 2024 Texas Instruments Incorporated

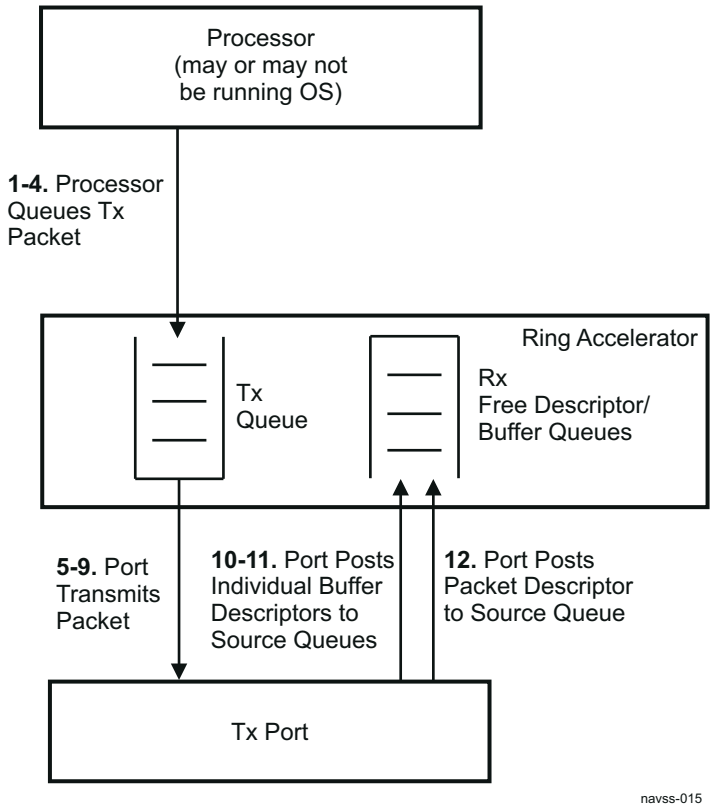


Figure 10-9. Host Packet Tx Operation – Automatic Buffer Recycling Packet Return

10.1.3.7 UDMA-P Transmit Operation (Monolithic Packet)

Packet transmission in Monolithic mode involves the following steps:

1. The Host allocates and populates a monolithic packet descriptor. The host will initialize the following fields within the packet descriptor:
   a. Descriptor Type (set to Monolithic)
   b. Packet Length indicating the total number of bytes in the packet
   c. Source Tag
   d. Destination Tag (application specific)
   e. Packet Type
   f. Any protocol specific flags for the given packet type
   g. Offset to payload data
   h. Descriptor reclamation policy fields indicating the mode and queue number for recycling the packet after transmission is complete.
   i. Networking Stack Information (application specific and optional)
   j. Any protocol specific words that are required for the given packet type
2. The Host queues the packet onto one of the Transmit Queues for the desired DMA channel. Channels may provide more than one Tx Queue and may provide a particular prioritization policy between the queues. This behavior is application specific and is controlled by the DMA controller/scheduler implementation.
3. The Ring Accelerator provides a level sensitive status signal for the queue which indicates if any packets are currently pending. This level sensitive status line is sent to the hardware block which is responsible for scheduling DMA operations.
4. The DMA controller is eventually brought into context for the corresponding channel and begins to process the packet.
5. The DMA controller reads the packet descriptor pointer from the Ring Accelerator.
6. The DMA controller reads the packet descriptor from memory
7. The DMA controller empties the data region in the descriptor by transmitting the contents in one or more block data moves. The size of these blocks is application specific. The DMA controller module specification is intended to document the block size used by a given implementation.

Copyright © 2024 Texas Instruments Incorporated

8. When all data for the packet has been transmitted as specified in the packet size field, the DMA will write the pointer to the packet descriptor to the queue specified in the return queue number fields of the packet descriptor.
9. After the Packet Descriptor pointer has been written, the Ring Accelerator indicates the status of the Tx Completion Queues to other ports/processors/prefetcher blocks using out-of-band level sensitive status lines. These status lines are set anytime a queue is non-empty.

   Whether or not any further processing is required on the packet at this point depends on the nature of the queue that the packet was returned to. The most common case is that the monolithic descriptor will be returned to the Rx Free Descriptor queue that it was allocated from originally. If this is the case, the Tx processing is complete at this point. Alternatively, the queue may also be a Tx Completion Queue in which case, the following optional steps may also be performed:
10. While most types of peer entities and embedded processors are able to directly and efficiently use these level sensitive status lines, cached processors may require a hardware block to convert the level status into pulsed interrupts and to perform some level of aggregation of the descriptor pointers from the completion queues into lists.
11. Host responds to status change from Ring Accelerator and performs garbage collection as necessary for packet.
12. Ring Accelerator sends down event when ring is empty

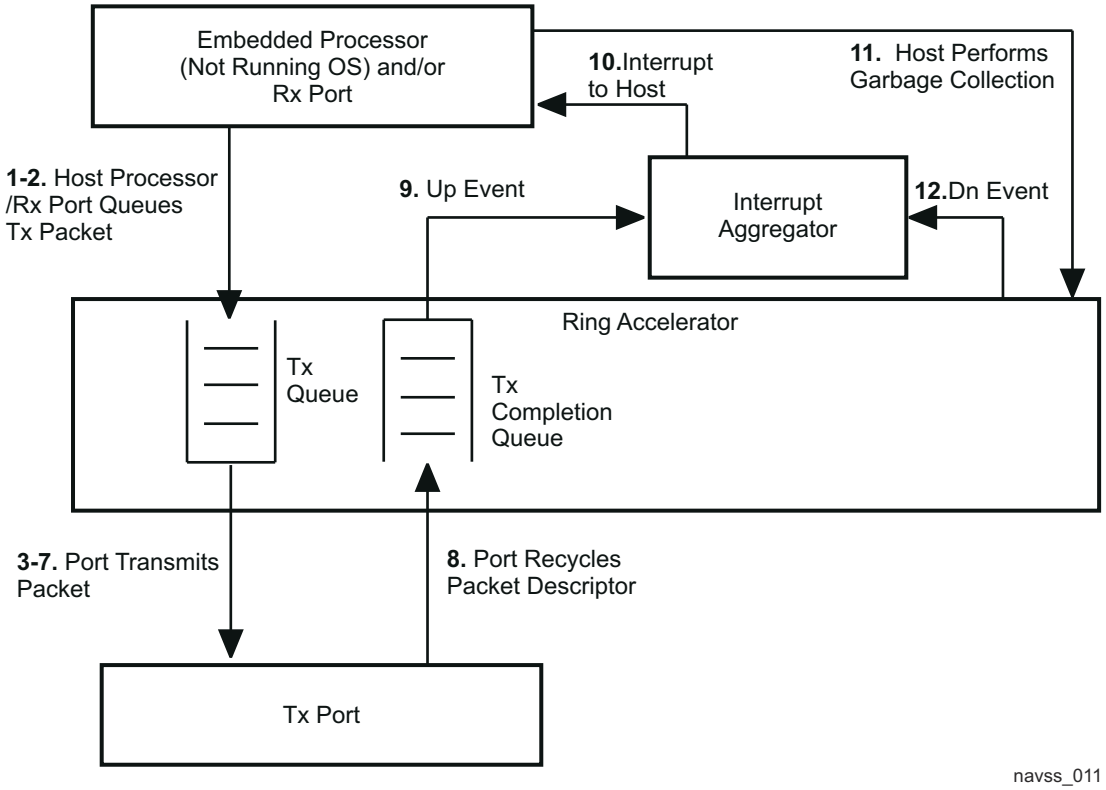


Figure 10-10. Monolithic Packet Tx Operation

10.1.3.8 UDMA Transmit Operation (TR Packet)

Packet transmission in TR Packet mode involves the following steps:

1. The Host allocates and populates a TR packet descriptor. The host will initialize the following fields within the packet descriptor:
   a. Descriptor Type (set to TR)
   b. Reload Enable to 1 if looping is required, otherwise 0
   c. Reload Index to an appropriate offset if Reload Enable set, otherwise 0
   d. Last Entry to TR count minus 1
   e. TR Nominal Element Size to a value that is as large as required for any given TR in the buffer

Copyright © 2024 Texas Instruments Incorporated

   f. Packet ID to a value that can be used to correlate the packet when it is placed back on the Tx completion queue
   g. Source Tag
   h. Destination Tag (application specific)
   i. Descriptor reclamation policy fields indicating the mode and queue number for recycling the packet after transmission is complete.
   j. A set of one or more valid Transfer Request records whose quantity matches the last index specified previously.
2. The Host queues the packet onto one of the Transmit Queues for the desired UDMA channel. Channels may provide more than one Tx Queue and may provide a particular prioritization policy between the queues. This behavior is application specific and is controlled by the DMA controller/scheduler implementation.
3. The Ring Accelerator provides a level sensitive status signal for the queue which indicates if any packets are currently pending. This level sensitive status line is sent to the hardware block which is responsible for scheduling DMA operations.
4. The DMA controller is eventually brought into context for the corresponding Tx channel and begins to process the packet.
5. The DMA controller reads the packet descriptor pointer from the RA.
6. The DMA controller reads the packet descriptor from memory
7. The DMA controller empties the data region in the descriptor by reading the contents in one or more nominal TR sized block data moves. These contents are then transmitted either to an internal UTC (in the case of a UDMA-P) or via PSI-L to a remote UTC (in the case of a UDMA-C).
8. If the UDMA includes an internal UTC, all of the data transfers specified in a TR will be completed as a series of reads (for single ended transfers) or reads and corresponding writes (for block copy transfers) and a Transfer Response will be returned indicating the completion and status of the transfer.
9. The UDMA will wait until Transfer Responses have been returned for each Transfer Request that it issued. As each Transfer Response is returned, the UDMA will write the response into the TR buffer in the Transfer Response records array. Each response is written into an array index which directly matches the index of the request record to which is corresponds.
10. When all Transfer Requests in the packet have been processed and all Transfer Responses have been written back and confirmed to have landed in memory, the UDMA will write the pointer to the packet descriptor to the queue specified in the return queue number fields of the packet descriptor.
11. After the Packet Descriptor pointer has been written, the Ring Accelerator indicates the status of the Tx Completion Queues to other ports/processors/prefetcher blocks using events sent to the Interrupt Aggregator. These events are then converted into standard K3 interrupts.

10.1.3.9 UDMA Transmit Operation (Direct TR)

If the rings are configured in pass by value mode individual Transfer Request and Response records can be passed between software and hardware without any higher level data structures. This mode allows for the UDMA to be prompted to do a transfer by just writing memory mapped locations with a source address, destination address and byte count (in the simplest case). Direct TR is accomplished as follows:

1. The Host writes a valid Transfer Request record to the Tx Queue of the desired UDMA channel.
2. The Ring Accelerator provides a level sensitive status signal for the queue which indicates if any packets are currently pending. This level sensitive status line is sent to the hardware block which is responsible for scheduling DMA operations.
3. The DMA controller is eventually brought into context for the corresponding channel and begins to process the packet.
4. The DMA controller reads the Transfer Request record from the ring via the RA in a single nominal TR sized block data move. These contents are then transmitted either to an internal UTC (in the case of a UDMA-P) or via PSI-L to a remote UTC (in the case of a UDMA-C).
5. If the UDMA includes an internal UTC, all of the data transfers specified in a TR will be completed as a series of reads (for single ended transfers) or reads and corresponding writes (for block copy transfers) and a Transfer Response will be returned indicating the completion and status of the transfer.
6. When a Transfer Response is returned for the Transfer Request record the UDMA will write the response into the next entry in the Transmit Completion Ring. Since the Transfer Responses come back from the UTC strongly ordered each response will be written into the ring strongly ordered as well.

Copyright © 2024 Texas Instruments Incorporated

7. After the Transfer Response has been written, the Ring Accelerator indicates the status of the Tx Completion Queues to other ports/processors/prefetcher blocks using events sent to the Interrupt Aggregator. These events are then converted into standard K3 interrupts.

10.1.3.10 UDMA Transmit Error/Exception Handling

There are 3 specific errors which may be encountered during Tx operation and which are trapped by the UDMA. The following table lists the errors and how the UDMA will process them:

Table 10-90. UDMA Tx Error/Exception Handling

Condition	Channel Mode	Pauses Channel	Error Flag Set in Channel	Data Transfer	TR Flush	Outputs Transfer Events
TR null icnt0	All UTC	Selectable [(1)]	Yes	No	Just that TR	No
Unsupported TR Type	All UTC	Selectable [(1)]	Yes	No	Just that TR	No
Bus Errors	All	Selectable [(1)]	Yes	Yes	No	Yes

(1) If PAUSE_ON_ERROR bit is set in channel configuration

10.1.3.10.1 Null Icnt0 Error

The icnt0 parameter in a TR is never allowed to be 0 as this would cause zero byte count transactions to be issued on the bus and is unproductive wrt completion of any useful work. The Tx UTC engine in the UDMA will trap any TR for which the icnt0 is zero and will immediately return a transfer response with the TR icnt0 error conditions set without performing any data transfer and without outputting any events (including end of TR event).

10.1.3.10.2 Unsupported TR Type

If a TR is provided to a channel which is of a type which is not supported by the channel, the Tx UTC engine in the UDMA-P will trap that TR and immediately return a transfer response with the Unsupported TR error conditions set without performing any data transfer and without outputting any events (including end of TR event).

10.1.3.10.3 Bus Errors

If a bus error is encountered during transmit the DMA will log the error by asserting the TX_ERROR bit for the channel (and optionally sending an error event if enabled) but the DMA will continue to attempt to complete the transfer using the returned (and potentially incorrect) data.

10.1.3.11 UDMA Receive Channel Setup (All Packet Types)

After a reset or a previous teardown operation but before receiving packets on a channel the host must initialize the channel's Rx Port DMA State. The host initializes the channel Rx Port DMA State by writing to the Rx Channel Configuration Registers. The Host may choose to write the enable bit in the Rx Global Channel Configuration Register at the same time or after it has written all of the channel parameters but note that every write to the Rx Channel Configuration Registers will overwrite the channel state for all bytes that are enabled for the write transaction.

10.1.3.12 UDMA Receive Channel Teardown

A Rx channel teardown is intended to be initiated at the data source (the source thread). Initiation is commanded by the host by writing the teardown bit in the PSI-L source thread register 0x408. This will cause the source of the data to gracefully terminate any packet that is in flight and send a PSI-L data phase with the tdown signal asserted. When the UDMA receives a data phase with the tdown attribute asserted it will immediately set the internal RX_TEARDOWN bit in the Rx Channel N Realtime Control Register.

Upon seeing the RX_TEARDOWN bit asserted the port can perform either a graceful (no data loss) or forced teardown of the channel depending on the setting of the RX_FORCED_TEARDOWN bit in the Rx Channel N Realtime Control Register.

If the RX_FORCED_TEARDOWN bit is clear, a normal teardown has been initiated. In this case, the UDMA will do the following:

1. Stops performing any additional prefetches for the channel (TR mode channels)

Copyright © 2024 Texas Instruments Incorporated

2. Completes any packets normally which may be pending in the ingress port buffers (what the port considers as pending packets is application specific).
3. Clears the channel enable in the Rx Channel Global Configuration Register.
4. Resets the channel state (including scoreboards, FIFOs, counters, statistics, etc.) to their after reset values.
5. Sends a single, 4-byte, teardown complete message to the queue specified in the Rx Channel Completion Queue register for the channel. The format of this message is as follows:

Bits	Field	Value	Description
31	Forced	0	Indicates that the teardown was graceful and data was not lost.
30:14	-	0	RESERVED
13:4	Channel ID	channel number	Indicates which channel the teardown completed on
3:0	Teardown Indicator	0x1	Indicates that a teardown has completed – normal pointers must be 16 byte aligned so this value indicates this is a teardown

If the RX_FORCED_TEARDOWN bit is set, a destructive teardown has been initiated. In this case, the UDMA will do the following:

1. Suspends any triggers to allow the channel to operate even if the trigger source is no longer functioning.
2. Completes any packets which may be pending in the ingress port buffers with or without transferring any data or generating events (the implementation has the option of doing whichever is simplest for that implementation) . Packets will be drained from the Rx FIFO either using 'null' write transfers or real transfers and the Packet Descriptors will be returned with an accurate accounting of actual bytes transferred.
3. Clears the channel enable in the Rx Channel N Realtime Control Register.
4. Resets the channel state (including scoreboards, FIFOs, counters, statistics, etc.) to their after reset values.
5. Sends a single, 4-byte, teardown complete message to the queue specified in the Rx Channel Completion Queue register for the channel. The format of this message is as follows:

Bits	Field	Value	Description
31	Forced	1	Indicates that the teardown was not graceful. Data was potentially lost.
30:14	-	0	RESERVED
13:4	Channel ID	channel number	Indicates which channel the teardown completed on
3:0	Teardown Indicator	0x1	Indicates that a teardown has completed – normal pointers must be 16 byte aligned so this value indicates this is a teardown

The host may issue a teardown on any channel at any time, regardless of whether the channel is actively receiving a packet or not. The normally intended method of issuing a teardown on a packet mode or single ended TR mode channel is to initiate the teardown at the remote PSI-L data source and allow it to flow into the UDMA. If that is not possible, the host may write the RX_TEARDOWN bit directly to a 1.

The Host determines that a teardown is complete by periodically polling the teardown and enable bits for the channel or by waiting to receive the teardown record on the queue specified in the Tx Channel Completion Queue register for the channel.

10.1.3.13 UDMA-P Receive Channel Pause

Setting the RX_PAUSE bit in the Rx Channel N Realtime Control Register will suspend the channel from arbitration resulting in a halting of the flow of data. Clearing this bit will cause the channel to be added back into the arbitration list. Pausing a channel has no other destructive side effects (other than potential underrun/overrun conditions from a downstream data sink or upstream data source.

10.1.3.14 UDMA-P Receive Free Descriptor/Buffer Queue Setup (Host Packets)

The Host is ultimately responsible for providing all of the free descriptors and buffers that the port uses as it receives packets. For Rx Free Descriptor/Buffer queues, descriptors are allocated in a large block by the Host and then initialized and linked with buffers before being given matched pairs where the Host initializes the

Copyright © 2024 Texas Instruments Incorporated

descriptor to point to its corresponding buffer before being pushed onto a Receive Free Descriptor/Buffer queue. Once the host has allocated both a descriptor and a buffer, it adds them to an Rx Free Descriptor/Buffer Queue by writing the pointer to the descriptor to the Rx Free Descriptor/Buffer Queue register D for the desired queue.

Before the Host adds each Rx buffer descriptor to the queue, it must first initialize the Rx buffer descriptor values as follows:

- Write the Original Buffer Pointer with the byte aligned address of the buffer data
- Write the Original Buffer Size
- Write the Return Queue Number field in the descriptor to appropriate values (required if auto-recycling is intended)
- Write the On Chip/ Off Chip indicator in the descriptor

Depending on the software architecture, the Host may also desire to write the return policy bit in the descriptor at this point as well. The DMA will not overwrite any of these listed fields (including return policy) during reception.

All other fields in the Descriptor do not need to be initialized as they will be overwritten by the Port on reception.

10.1.3.15 UDMA-P Receive FlowID Firewall Operation

Each incoming packet that enters the UDMA-P will either include an explicitly specified FlowID contained in the info words or it will have an implicitly specified default FlowID which is the same as the channel number. The Rx FlowID firewall checks the incoming FlowID on the received packet to verify that it is either the corresponding default FlowID for the channel or that the FlowID falls within the range specified in the Rx Channel Flow Range register. If the received flowID is not the channel number and it does not fall within the legal range as specified then the packet is dropped, the error is trapped in the Rx Flow ID Firewall Status register, and an optional output event is generated as specified in the Rx Flow ID Firewall Output Event Steering Register.

10.1.3.16 UDMA-P Receive Operation (Host Packet)

Figure 10-11 shows a diagram of the overall Receive operation for Host flow mode.

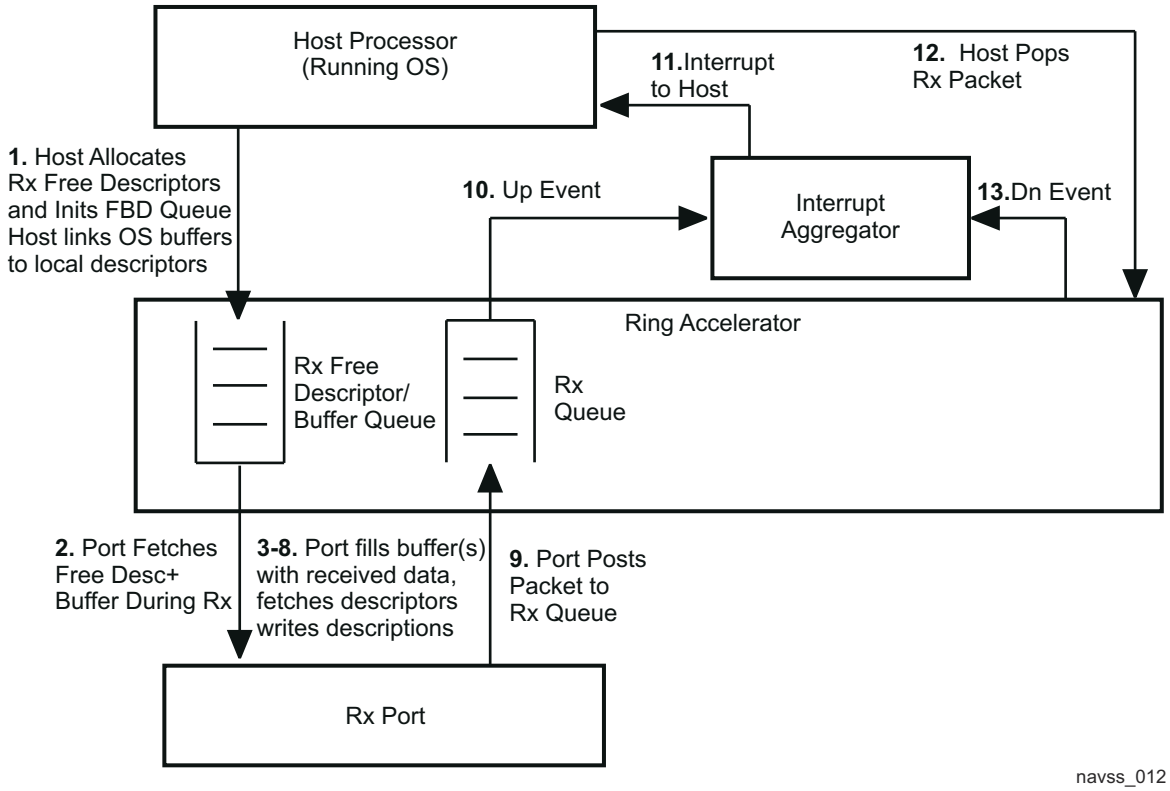


Figure 10-11. Host Packet Receive Operation

Copyright © 2024 Texas Instruments Incorporated

When packet reception begins on a given channel, the port will begin by fetching the first descriptor + buffer from the Ring Accelerator using the Free Descriptor/Buffer Queue 0 Index that was initialized in the Rx Flow Table for the flow which is being used by the channel. If the SOP Buffer Offset in the Rx Flow Table entry is nonzero, then the port will begin writing data after the offset number of bytes in the SOP buffer. The port will then continue filling that buffer and will fetch additional descriptors + buffers as needed using the Free Descriptor/Buffer Queue 1, 2, and 3 indexes for the 2nd, 3rd, and remaining buffers in the packet.

A detailed summary is as follows:

1. Host allocates, populates, and places pointers to free descriptor/buffer structures onto Rx Free Descriptor/ Buffer Queues
2. UDMA-P fetches packet descriptor pointer from RA
3. UDMA-P reads the packet descriptor to obtain the packet info, buffer pointer, and size
4. UDMA-P fills the buffer with received data
5. If packet is not complete, UDMA-P fetches next buffer descriptor pointer from RA
6. UDMA-P fills the buffer with received data

   If the RX_DESC_TYPE field in the Rx Flow Register A is set to 0 (Normal Rx Host Mode), Steps 5-7 continue until all packet data is received at which point the Rx engine proceeds to step 8. If the RX_DESC_TYPE is set to 1 (Single Buffer Packet Host Mode) the Rx engine will proceed to step 8 skipping step 7.
7. UDMA-P writes previous buffer descriptor to memory. This includes the following fields:
   a. Buffer information
   b. Buffer Pointer with the byte aligned address of the chunk of buffer data
   c. Buffer Length with the number of bytes in the chunk of buffer data
   d. Pointer to next buffer descriptor in packet or 0 if this is the EOP buffer.

   If the RX_DESC_TYPE field in the Rx Flow Register A is set to 0 (Normal Rx Host Mode), Steps 5-7 continue until all packet data is received at which point the Rx engine proceeds to step 8. If the RX_DESC_TYPE is set to 1 (Single Buffer Packet Host Mode) the Rx engine proceeds to step 8 as soon as the first buffer is full. Any remaining data in the packet from the line is then received into one or more additional packets each exactly one Host buffer long.

   The UDMA-P performs the following operations when the entire packet has been received:
8. UDMA-P writes the packet descriptor to memory. This includes the following fields:
   a. Descriptor Type (set to Host)
   b. Packet Length indicating the total number of bytes that are to be read from all of the buffers for this packet.
   c. Source Tag
   d. Destination Tag (application specific)
   e. Packet Type
   f. Any protocol specific flags for the given packet type
   g. SOP buffer information
      i. Buffer Pointer with the byte aligned address of the chunk of buffer data
      ii. Buffer Length with the number of bytes in the chunk of buffer data
   h. Any protocol specific words that are required for the given packet type
   i. Networking Stack Information (application specific and optional)
   j. Pointer to next buffer descriptor in packet or 0 if this is the EOP buffer.
9. UDMA-P writes the packet descriptor pointer to the appropriate Rx Queue. The absolute Queue that each packet to be forwarded to on completion of reception will either be the Queue which that was specified in the rx_dest_qnum field in the Rx Flow Table entry or will be a queue which was provided via the PSI-L destination queue number override field.
10. Once the packet pointer is written to the Ring Accelerator it will send an up event to the Interrupt Aggregator
11. The Interrupt Aggregator upon receiving the up event will set the appropriate bit in the Interrupt Status Register indicated by the interrupt mapping table which will cause an interrupt to be asserted to the Host
12. The Host will pop the packet pointer from the Rx Queue

Copyright © 2024 Texas Instruments Incorporated

13. If the pop causes the queue to become empty, the Ring Accelerator will send a down event to the Interrupt Aggregator thus causing the associated bit to be cleared and also potentially clearing the interrupt to the Host.

The Ring Accelerator is responsible for indicating the status of the Receive Queues to other ports/embedded processors using out-of-band level sensitive status lines. These status lines are set anytime a queue is non-empty.

10.1.3.17 UDMA-P Receive Operation (Monolithic Packet)

Figure 10-12 shows a diagram of the overall Receive operation for a monolithic descriptor type.

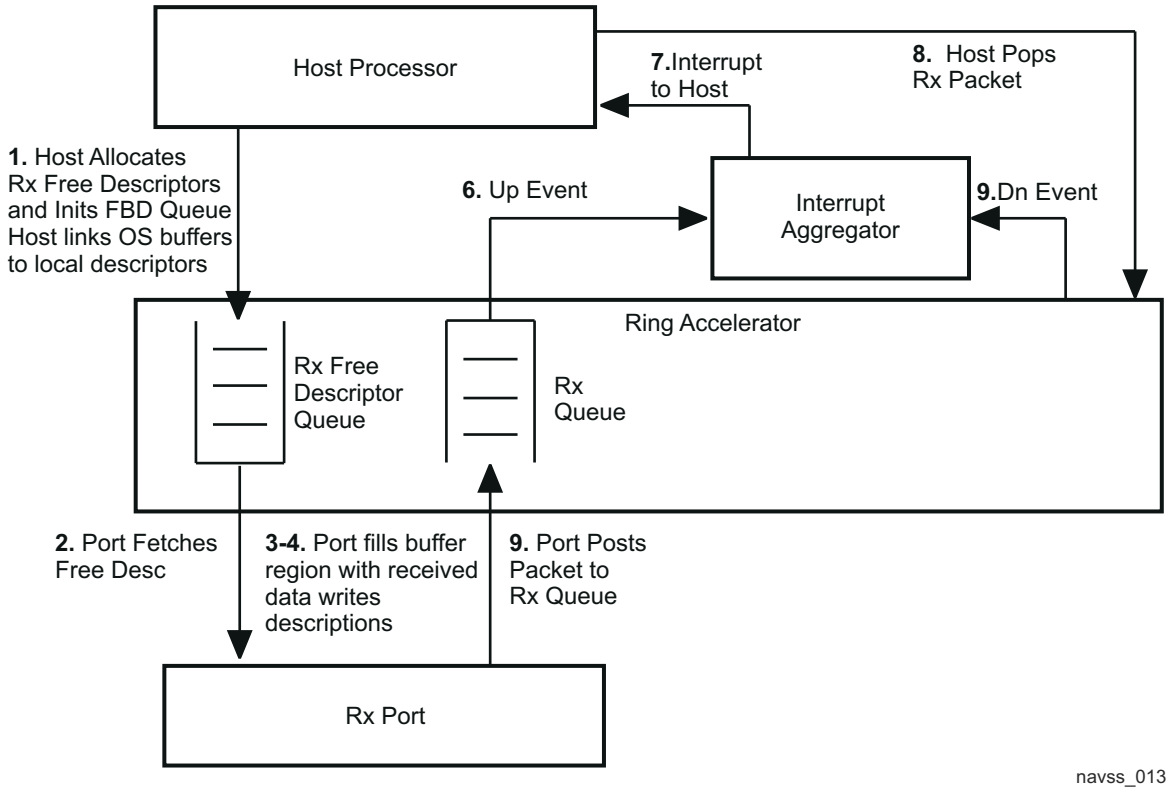


Figure 10-12. Monolithic Packet Receive Operation

The following sequence is a detailed summary of the Monolithic Packet Reception process:

1. The Host initializes free Descriptors and places them on an Rx Free Descriptor Queue
2. The UDMA-P fetches a single descriptor for the packet from the Ring Accelerator using the Monolithic Type Free Descriptor Queue Index that was initialized in the Rx Flow Table entry that is being used for the packet.
3. If the RX_SOP_OFFSET field in the Rx Flow Table entry is nonzero, then the port will begin writing data after the offset number of bytes in the payload portion of the monolithic descriptor. The port will then continue filling the payload portion of the descriptor until the entire packet has been received.

   The port performs the following operations when the entire packet has been received:
4. Writes the packet descriptor to memory. This includes the following fields:
   a. Descriptor Type (set to Monolithic)
   b. Packet Length indicating the total number of bytes in this packet.
   c. Source Tag
   d. Destination Tag (application specific)
   e. Packet Type
   f. Any protocol specific flags for the given packet type
   g. Networking Stack Information (application specific and optional)
   h. Any protocol specific words that are required for the given packet type

Copyright © 2024 Texas Instruments Incorporated

5. Writes the packet descriptor pointer to the appropriate Rx Queue. The absolute Queue that each packet to be forwarded to on completion of reception will either be the Queue which that was specified in the rx_dest_qnum fields in the Rx Flow Entry or can be overridden by the value in the PSI-L destination queue override field.
6. Once the packet pointer is written to the Ring Accelerator it will send an up event to the Interrupt Aggregator
7. The Interrupt Aggregator upon receiving the up event will set the appropriate bit in the Interrupt Status Register indicated by the interrupt mapping table which will cause an interrupt to be asserted to the Host
8. The Host will pop the packet pointer from the Rx Queue
9. If the pop causes the queue to become empty, the Ring Accelerator will send a down event to the Interrupt Aggregator thus causing the associated bit to be cleared and also potentially clearing the interrupt to the Host.

The Ring Accelerator is responsible for indicating the status of the Receive Queues to other ports/processors using out-of-band level sensitive status lines. These status lines are set anytime a queue is non-empty.

10.1.3.18 UDMA Receive Operation (TR Packet)

Packet reception in TR Packet mode involves the following steps:

1. The Host allocates and populates a TR packet descriptor. The host will initialize the following fields within the packet descriptor:
   a. Descriptor Type (set to TR)
   b. Reload Enable to 1 if looping is required, otherwise 0
   c. Reload Index to an appropriate offset if Reload Enable set, otherwise 0
   d. Last Entry to TR count minus 1
   e. TR Nominal Element Size to a value that is as large as required for any given TR in the buffer
   f. Packet ID to a value that can be used to correlate the packet when it is placed back on the Tx completion queue
   g. Source Tag
   h. Destination Tag (application specific)
   i. Descriptor reclamation policy fields indicating the mode and queue number for recycling the packet after transmission is complete.
   j. A set of one or more valid Transfer Request records whose quantity matches the last index specified previously.
2. The Host queues the packet onto one of the Rx TR Queues for the desired UDMA channel. Channels may provide more than one Rx TR Queue and may provide a particular prioritization policy between the queues. This behavior is application specific and is controlled by the DMA controller/scheduler implementation.
3. The Ring Accelerator provides a level sensitive status signal for the queue which indicates if any packets are currently pending. This level sensitive status line is sent to the hardware block which is responsible for scheduling DMA operations.
4. The DMA controller is eventually brought into context for the corresponding channel and begins to process the packet.
5. The DMA controller reads the packet descriptor pointer from the RA.
6. The DMA controller reads the packet descriptor from memory
7. The DMA controller empties the data region in the descriptor by reading the contents in one or more nominal TR sized block data moves. These contents are then transmitted to an internal UTC.
8. If the UDMA includes an internal UTC, all of the data transfers specified in a TR will be completed as a series of writes (for single ended transfers) and a Transfer Response will be returned indicating the completion and status of the transfer.
9. The UDMA will wait until Transfer Responses have been returned for each Transfer Request that it issued. As each Transfer Response is returned, the UDMA will write the response into the TR buffer in the Transfer Response records array. Each response is written into an array index which directly matches the index of the request record to which is corresponds.
10. When all Transfer Requests in the packet have been processed and all Transfer Responses have been written back and confirmed to have landed in memory, the UDMA will write the pointer to the packet descriptor to the queue specified in the return queue number fields of the packet descriptor.

Copyright © 2024 Texas Instruments Incorporated

11. After the Packet Descriptor pointer has been written, the Ring Accelerator indicates the status of the Rx Completion Queues to other ports/processors/prefetcher blocks using events sent to the Interrupt Aggregator. These events are then converted into standard K3 interrupts.

10.1.3.19 UDMA Receive Operation (Direct TR)

If the rings are configured in pass by value mode individual Transfer Request and Response records can be passed between software and hardware without any higher level data structures. This mode allows for the UDMA to be prompted to do a transfer by just writing memory mapped locations with a source address, destination address and byte count (in the simplest case). Direct TR is accomplished as follows:

1. The Host writes a valid Transfer Request record to the Rx TR Queue of the desired UDMA channel.
2. The Ring Accelerator provides a level sensitive status signal for the queue which indicates if any packets are currently pending. This level sensitive status line is sent to the hardware block which is responsible for scheduling DMA operations.
3. The DMA controller is eventually brought into context for the corresponding channel and begins to process the packet.
4. The DMA controller reads the Transfer Request record from the ring via the RA in a single nominal TR sized block data move. These contents are then transmitted to an internal UTC (in the case of a UDMA-P).
5. All of the data transfers specified in a TR will be completed as a series of writes (for single ended transfers) and a Transfer Response will be returned indicating the completion and status of the transfer.
6. When a Transfer Response is returned for the Transfer Request record the UDMA will write the response into the next entry in the Transmit Completion Ring. Since the Transfer Responses come back from the UTC strongly ordered each response will be written into the ring strongly ordered as well.
7. After the Transfer Response has been written, the Ring Accelerator indicates the status of the Rx Completion Queues to other ports/processors/prefetcher blocks using events sent to the Interrupt Aggregator. These events are then converted into standard K3 interrupts.

10.1.3.20 UDMA Receive Error/Exception Handling

Several types of errors/exceptions may be encountered during reception of data. The following table outlines the various errors and exceptions that can occur:

Table 10-91. UDMA Rx Error/Exception Handling

Condition	Channel Type	Severity	Pauses Channel	Error Flag Set	Data Transfer	Data Flush	TR Flush	Event Output
Descriptor Starvation	All Packet	Exception	No	No	Option	Option	-	No
Protocol Errors	All Packet	Info	No	No	Normal	No	-	No
Drop	Host – normal	Exception	No	No	Partial [2]	Until EOP	-	No
	Host – single buffer	Error	Selectable [1]	Yes	Partial [2]	Until EOP [3]	-	No
	Mono	Exception	No	No	Partial [2]	Until EOP	-	No
	UTC–single ended	Exception	No	No	Partial [2]	Until EOP	Until EOP [4]	Yes
	UTC- block copy	Illegal	-	-	-	-	-	-
EOL Asserted Prematurely	All Packet	Error	Selectable [1]	Yes	No	Yes	-	No
	UTC – single ended	Exception	No	No	Normal	No	No	Yes
	UTC – block copy	Illegal	-	-	-	-	-	-
EOP Asserted Prematurely (Short Packet)	UTC – single ended	Exception	No	No	Normal	No	Until EOP [4]	Yes
EOP Asserted Late (Long Packet)	UTC – single ended	Exception	No	No	Partial	Excess	No	Yes
TR null icnt0	All UTC	Error	Selectable [1]	Yes	No	No	Just that TR	No

Copyright © 2024 Texas Instruments Incorporated

Table 10-91. UDMA Rx Error/Exception Handling (continued)

Condition	Channel Type	Severity	Pauses Channel	Error Flag Set	Data Transfer	Data Flush	TR Flush	Event Output
Unsupported TR Type	All UTC	Error	Selectable [(1)]	Yes	No	No	Just that TR	No

(1) If PAUSE_ON_ERROR bit is set in channel configuration
(2) Data will not be transferred after drop is detected. All data after that point is guaranteed to be flushed and some data prior to the exact data phase on PSI-L where drop was asserted may also be flushed since data is constantly accumulated so it can be transferred in chunks.
(3) EOP will only occur when in single buffer mode when teardown is initiated. The data is a (never ending) stream. Drop is never supposed to be asserted during this mode of operation.
(4) EOP on TR is true for all PBV TRs, the final TR in a PBR, or if the user specified EOP in the TR flags

10.1.3.20.1 Error Conditions

The following sections describe items which fall under an error severity when encountered by the UDMA on Rx:

10.1.3.20.1.1 Bus Errors

If a bus error is encountered during receive the DMA will log the error by asserting the RX_ERROR bit for the channel (and optionally sending an error event if enabled) but the DMA will continue to attempt to complete the transfer using the returned (and potentially incorrect) data.

10.1.3.20.1.2 Null Icnt0 Error

The icnt0 parameter in a TR is never allowed to be 0 as this would cause zero byte count transactions to be issued on the bus and it unproductive towards completion of any useful work. The Rx UTC engine in the UDMA will trap any TR for which the icnt0 is zero and will immediately return a transfer response with the TR icnt0 error conditions set without performing any data transfer and without outputting any events (including end of TR event).

10.1.3.20.1.3 Unsupported TR Type

If a TR is provided to a channel which is of a type which is not supported by the channel, the Rx UTC engine in the UDMA-P will trap that TR and immediately return a transfer response with the Unsupported TR error conditions set without performing any data transfer and without outputting any events (including end of TR event).

10.1.3.20.2 Exception Conditions Exception Conditions

There are a few cases which can occur when processing incoming packets which are not considered errors but which will require handling in order to avoid locking up the UDMA packet or TR mode write engines.

Packet mode channel exceptions include:

- Descriptor Starvation
- Protocol Errors
- Dropped Packets

Since the TR mode engine performs transfer sequencing using a count based mechanism it is susceptible to becoming out of synchronization if the source of the data and the destination for the data do not exactly match in their expectations for how much data will be transferred. The following sections outline these scenarios.

TR mode channel exceptions include:

- Reception of EOL delimiter
- Short Packets
- Long Packets

10.1.3.20.2.1 Descriptor Starvation

Descriptor starvation occurs when the Port attempts to fetch a free descriptor from an Rx Free Descriptor Queue or Rx Free Descriptor/Buffer Queue and none are available. When no descriptor is available, the Ring Accelerator will return a value of 0x0 for the descriptor pointer (Optionally, some ports may contain internal counters for the free queues which indicate how many elements are currently queued. In this case, no access to the RA is required and starvation is directly detected via a zero element count). When the port detects that

Copyright © 2024 Texas Instruments Incorporated

descriptor starvation has occurred it will react based on the value of the RX_ERROR_HANDLING bit which was programmed into the Rx Flow N Configuration Register A.

- If the RX_ERROR_HANDLING bit is cleared:

The Port will assert a descriptor starvation indicator and will then return any descriptors that it has already used in the reception of the current packet back to the appropriate Free Descriptor or Free Descriptor/Buffer queue that they were fetched from. This is performed by writing the pointers for each descriptor to the Rx Free Descriptor Queue N Register in sequence using the queue indexes in the Rx DMA state. The port may choose to implement a counter that counts the number of packets that were dropped due to descriptor starvation and may also desire to distinguish between whether the starvation occurred on the packet descriptor (SOP) or on a buffer starvation.

- If the RX_ERROR_HANDLING bit is set:

The Port will assert a descriptor starvation indicator (which may cause an interrupt to the Host) and will then behave as if it had not performed the current data block transfer which caused the starvation. The Port is required to pause its current processing saving the state of the transfer. The next time that the channel comes into context it will again try to allocate a descriptor with the intention being that if buffers/descriptors have been added that the operation will complete successfully on a subsequent retry.

10.1.3.20.2.2 Protocol Errors

Protocol errors occur when the port logic detects that the received packet did not pass protocol specific criteria that was checked during reception of the packet. Examples of protocol errors include packet length errors, CRC errors, or alignment errors.

When protocol errors are detected, the port may choose whether or not it will drop the packet. The mechanism that is used to determine which packets to drop and which to forward is application specific. If the port determines that it needs to forward the packet to the Host, it will set the Packet Error bit in the Descriptor indicating that this is a packet which experienced a protocol related error. The type of protocol related error is generally indicated in either the Protocol Specific bits in the Host or Monolithic descriptor or in the Protocol Specific bytes region. The packet length information and certain portions of the Protocol Specific region may not be correct for packets which encounter errors.

10.1.3.20.2.3 Dropped Packets

Packets can be dropped within an Rx Port for any number for reasons which with the exception of the starvation case which was covered above are application specific.

If a split TR mode channel receives instruction to drop a packet on the PSI-L interface, the UDMA/UTC will continue executing the TRs but without data transfer up until a TR is completed for which the EOP condition was set. Events will be generated as normal although data may not be actually transferred.

10.1.3.20.2.4 Reception of EOL Delimiter

If an end of line delimiter is received by the Rx (write) side of the UTC the EOL_ADV field in the TR application specific flags field is used to advance the appropriate indices of the TR to the next level. Each EOL that is received will cause the TR to advance to the next specified loop iteration breaking as many intermediate loops as are required.

10.1.3.20.2.5 EOP Asserted Prematurely (Short Packet)

Split TR mode channels can experience an incoming packet whose length is shorter than what was expected based on one or more TRs which form the control description for the expected packet. If an EOP is received for a TR and the TR has not been completely executed, the port is required to receive as much data as is actually provided in the incoming data placing it in accordance with the instructions in the TR and the to continue executing any remaining TR(s) such that the required events are produced to keep downstream consumers in sync. In this case, no data will be transferred to the buffers described by the TR beyond what was provided in the incoming packet.

Copyright © 2024 Texas Instruments Incorporated

10.1.3.20.2.6 EOP Asserted Late (Long Packets)

Split TR mode channels can experience an incoming packet whose length is longer than what was expected based on one or more TRs which form the control description for the expected packet. If a TR completes and is marked as EOP but the incoming packet is not finished, then the port must throw away any additional received data until the incoming packet reaches an EOP condition. At this point, reception will start with a new TR and a new incoming packet.

10.1.3.21 UTC Operation

A UTC accepts Transfer Request records from the UDMA-C and performs a sequence of transactions as specified in the TR record in accordance with the address generation algorithm that is specified in the Transfer Request format specification. The UTC will wait until an optional triggering event has occurred to load the channel state into one of its read or write engines so that the data movement operations can be completed. The UTC will complete the specified transactions until it either has transferred a pre-set number of bytes or until it has reached a point in the sequence where the TR indicates that a trigger is required to continue execution. When either of these two conditions is met, the UTC will save off the current state of the channel and will wait until the channel is re-triggered at which point the TR execution will continue. Once the entire Transfer Request specified sequence of data transfers has completed, the UTC will wait to ensure that all outstanding writes have landed at their respective destination endpoints and will then return a Transfer Response message back to the UDMA-C indicating the success or failure of the Transfer Request. The UTC processes Transfer Request operations on a given channel with strong ordering such that all Transfer Response operations will be returned in an order corresponding to the exact order that Transfer Request operations were issued.

10.1.3.22 UTC Receive Error/Exception Handling

Several types of errors/exceptions may be encountered during reception of data. The following table outlines the various errors and exceptions that can occur:

Table 10-92. UTC Rx Error/Exception Handling

Condition	Channel Type	Severity	Pauses Channel	Error Flag Set	Data Transfer	Data Flush	TR Flush	Event Output
Drop	UTC–single ended	Exception	No	No	Partial[(2)]	Until EOP	Until EOP[(3)]	Yes
	UTC- block copy	Illegal	-	-	-	-	-	-
EOL Asserted Prematurely	UTC – single ended	Exception	No	No	Normal	No	No	Yes
	UTC – block copy	Illegal	-	-	-	-	-	-
EOP Asserted Prematurely (Short Packet)	UTC – single ended	Exception	No	No	Normal	No	Until EOP[(3)]	Yes
EOP Asserted Late (Long Packet)	UTC – single ended	Exception	No	No	Partial	Excess	No	Yes
TR null icnt0	All UTC	Error	Selectable[(1)]	Yes	No	No	Just that TR	No
Unsupported TR Type	All UTC	Error	Selectable[(1)]	Yes	No	No	Just that TR	No

(1) If pause_on_error bit is set in channel configuration
(2) Data will not be transferred after drop is detected. All data after that point is guaranteed to be flushed and some data prior to the exact data phase on PSI-L where drop was asserted may also be flushed since data is constantly accumulated so it can be transferred in chunks.
(3) EOP on TR is true for all PBV TRs, the final TR in a PBR, or if the user specified EOP in the TR flags

10.1.3.22.1 Error Handling

If an error occurs during the processing of a Transfer Request, the UTC will discontinue any further operations on the TR and will wait for all outstanding transactions to complete. At that point, the Transfer Response record will be returned to the UDMA-C with the appropriate codes indicating the type of failure as specified in the Transfer Response Format specification.

Copyright © 2024 Texas Instruments Incorporated

Depending on the value of the PAUSE_ON_ERROR bit in the UTC configuration register, the UTC will next do one of the following:

If the PAUSE_ON_ERROR bit is clear the channel will move on to the next Transfer Request record that it received from the UDMA-C.

If the PAUSE_ON_ERROR bit is set the channel will halt from processing any new Transfer Request records in order to allow the Host to examine the current channel state in order to determine the potential cause of the failure. The UTC is required to preserve as much of the channel state as possible as soon as a failure is determined. Note that the state will not necessarily provide exact details or a 'smoking gun' as the UTC implementation allows for large scale transaction pipelining and error status returns from the bus can occur well after the machine has moved on to other channel tasks.

10.1.3.22.1.1 Null Icnt0 Error

The icnt0 parameter in a TR is never allowed to be 0 as this would cause zero byte count transactions to be issued on the bus and it unproductive towards completion of any useful work. The Rx UTC engine in the UDMA will trap any TR for which the icnt0 is zero and will immediately return a transfer response with the TR icnt0 error conditions set without performing any data transfer and without outputting any events (including end of TR event).

10.1.3.22.1.2 Unsupported TR Type

If a TR is provided to a channel which is of a type which is not supported by the channel, the Rx UTC engine in the UDMA-P will trap that TR and immediately return a transfer response with the Unsupported TR error conditions set without performing any data transfer and without outputting any events (including end of TR event).

10.1.3.22.2 Exception Conditions

There are a few cases which can occur when processing TRs which are not considered errors but which will require handling in order to avoid locking up the UTC read or write engines. Since the UTC performs transfer sequencing using a count based mechanism it is susceptible to becoming out of synchronization if the source of the data and the destination for the data do not exactly match in their expectations for how much data will be transferred. The following outline these scenarios.

10.1.3.22.2.1 Reception of EOL Delimiter

If an end of line delimiter is received by the Rx (write) side of the UTC the EOL_ADV field in the TR application specific flags field is used to advance the appropriate indices of the TR to the next level. Each EOL that is received will cause the TR to advance to the next specified loop iteration breaking as many intermediate loops as are required.

10.1.3.22.2.2 EOP Asserted Prematurely (Short Packet)

Split TR mode channels can experience an incoming packet whose length is shorter than what was expected based on one or more TRs which form the control description for the expected packet. If an EOP is received for a TR and the TR has not been completely executed, the port is required to receive as much data as is actually provided in the incoming data placing it in accordance with the instructions in the TR and the to continue executing any remaining TR(s) such that the required events are produced to keep downstream consumers in sync. In this case, no data will be transferred to the buffers described by the TR beyond what was provided in the incoming packet.

10.1.3.22.2.3 EOP Asserted Late (Long Packets)

Split TR mode channels can experience an incoming packet whose length is longer than what was expected based on one or more TRs which form the control description for the expected packet. If a TR completes and is marked as EOP but the incoming packet is not finished, then the port must throw away any additional received data until the incoming packet reaches an EOP condition. At this point, reception will start with a new TR and a new incoming packet.

Copyright © 2024 Texas Instruments Incorporated

10.2 Navigator Subsystem (NAVSS)

This chapter describes the features and functions of the Navigator Subsystem (NAVSS) hardware modules in the device.

The SoC device contains two NAVSSes:

- Main NAVSS (NAVSS0) – See Section 10.2.1, *Main Navigator Subsystem (NAVSS)*
- MCU NAVSS (MCU_NAVSS0) – See Section 10.2.2, *MCU Navigator Subsystem (MCU NAVSS)*.

Copyright © 2024 Texas Instruments Incorporated

10.2.1 Main Navigator Subsystem (NAVSS)

This section describes the top-level integration of the main Navigator Subsystem (NAVSS) in the device.

10.2.1.1 NAVSS Overview

Main SoC Navigator Subsystem (NAVSS0) consists of DMA/Queue Management components – UDMA and Ring Accelerator (UDMASS), Peripherals (Module subsystem [MODSS]), Virtualization translation (VirtSS), and a North Bridge (NBSS).

UDMASS – UDMASS is the essential part of the *TI Data Movement Architecture* (see Section 10.1). UDMASS consists of:

- Unified DMA Controller – Section 10.2.3
- Ring Accelerator – Section 10.2.4
- Packet Streaming Interface (PSI-L) – Section 10.2.8
- Block Copy DMA Controller

MODSS – Module subsystem is a collection of peripherals with different system-level functions, for example, interprocessor communication and time sync, among others. NAVSS0 contains the following modules:

- Mailbox – Section 7.1
- Spinlock – Section 7.2
- Two Timer Managers (Timer banks) – Section 11.2
- Time Stamp Module (CPTS) – Section 11.1
- Infrastructure components such as:
  - CBASS – *System Interconnect*
  - Proxies – Section 10.2.5, Section 10.2.6
  - Interrupt aggregators – Section 10.2.7
  - Interrupt router – *NAVSS Interrupt Router Configuration*

NBSS – North bridge – Section 10.2.9

VirtSS – Virtualization sub-system provides all the translation components supported in K3 for virtualization:

- Peripheral Virtualization Units (PVU) – Section 8.3

ECC aggregators – Section 12.10.4

NAVSS0 hardware components and their integration are shows on Figure 10-13.

Copyright © 2024 Texas Instruments Incorporated

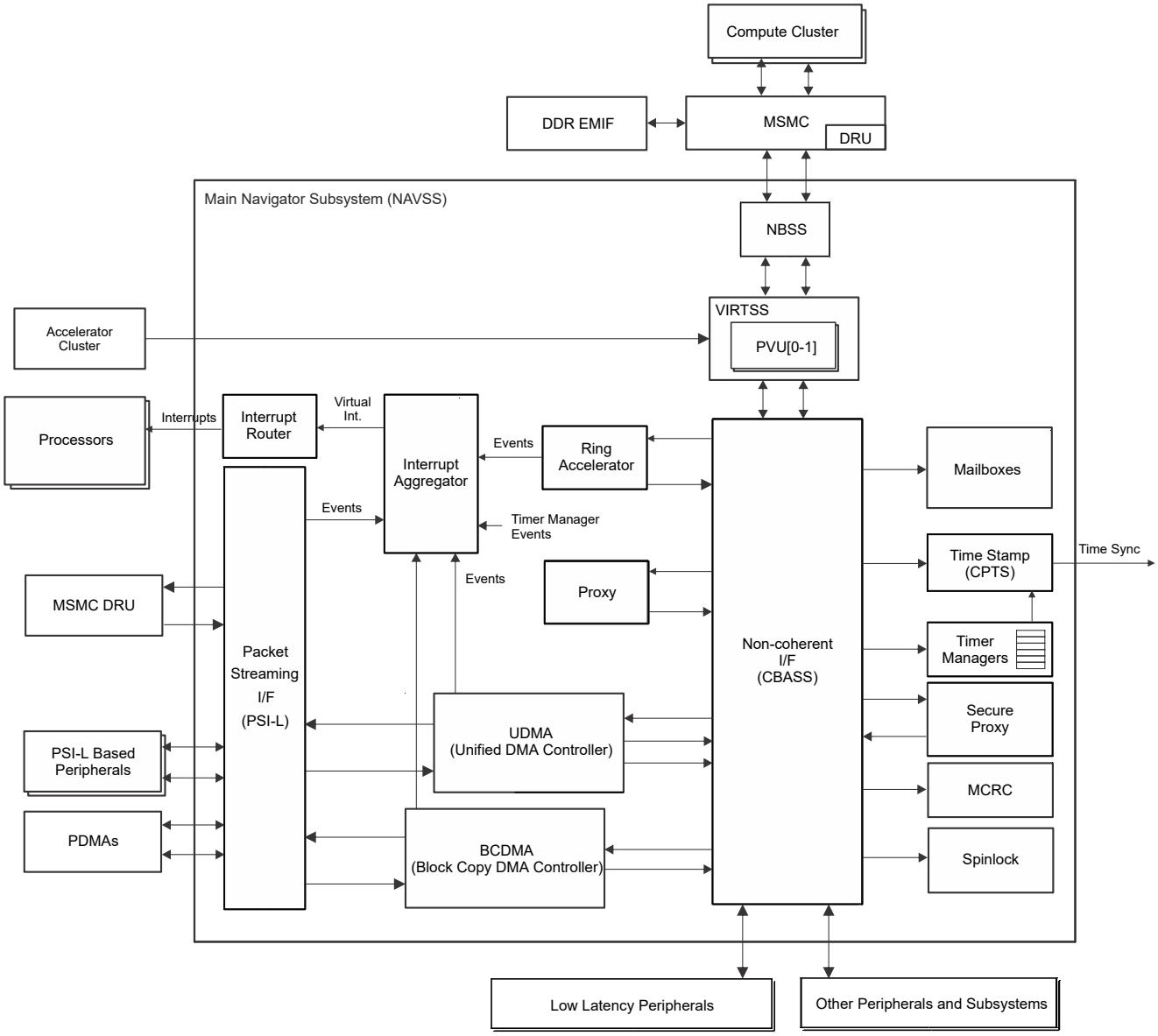


Figure 10-13. NAVSS0 Top-Level Block Diagram

Note

MCU NAVSS functionality is similar to that of main NAVSS, however it is composed of fewer peripherals and has reduced bandwidth.

10.2.1.1.1 Main Navigator Subsystem (NAVSS) Ports

Table 10-93. NAVSS Clocks and Resets

Clocks	
Module Clock Input	**Description**
MODSS_VBUS_D2_CLK	MODSS config interface clock. This clock is used for most of the NAVSS modules (MODSS).
UDMASS_VBUS_D2_CLK	UDMASS config interface clock. This clock is used for UDMASS modules.
MSMC0_CLK	MSMC0 PSI-L interface clock
PDMA_MAIN_MISC_CLK	PDMA_MAIN_MISC PSI-L interface clock

Copyright © 2024 Texas Instruments Incorporated

Table 10-93. NAVSS Clocks and Resets (continued)

PDMA_MAIN_USART_CLK	PDMA_MAIN_USART PSI-L interface clock
PDMA_MAIN_AASRC_CLK	PDMA_MAIN_AASRC PSI-L interface clock
PDMA_MAIN_DEBUG_CLK	PDMA_MAIN_DEBUG PSI-L interface clock
PDMA_MAIN_MCASP_G1_CLK	PDMA_MAIN_MCASP_G1 PSI-L interface clock
ICSS_G0_CLK	ICSS_G0 PSI-L interface clock
ICSS_G1_CLK	ICSS_G1 PSI-L interface clock
NAVSS_MCU_CLK	MCU_NAVSS PSI-L interface clock
NBSS_VBUS_D2_CLK	NBSS config interface clock. This clock is used for NBSS modules (NB0-1, ECC_AGGR0).
NBSS_VBUS_CLK	NBSS VBUSM. This clock is used for NB0
VIRTSS_VBUS_D2_CLK	VIRTSS config interface clock.
EXT[0-3]_DTI_CLK	DTI clocks
Resets	
Module Reset Input	**Description**
MODSS_RST	MODSS hardware reset
UDMASS_RST	UDMASS hardware reset
NBSS_RST	NBSS hardware reset
VIRTSS_RST	VIRTSS hardware reset

Table 10-94. NAVSS Hardware Requests

Interrupt Requests		
Module Interrupt Signal	**Description**	**Type**
INTR_ROUTER0_OUTL_INTR[63:0] INTR_ROUTER0_OUTL_INTR[127:64] INTR_ROUTER0_OUTL_INTR[191:128]	INTR_PEND[191:0] interrupts to GIC SPI	Level
INTR_ROUTER0_OUTL_INTR[223:192] INTR_ROUTER0_OUTL_INTR[255:224]	INTR_PEND[255:192] interrupts to main R5FSS0	Level
INTR_ROUTER0_OUTL_INTR[287:256] INTR_ROUTER0_OUTL_INTR[319:288]	INTR_PEND[319:256] interrupts to main R5FSS1	Level
INTR_ROUTER0_OUTL_INTR[351:344]	INTR_PEND[351:344] interrupts to C66SS0	Level
INTR_ROUTER0_OUTL_INTR[383:376]	INTR_PEND[383:376] interrupts to C66SS1	Level
INTR_ROUTER0_OUTL_INTR[391:348]	INTR_PEND[391:348] interrupts to PRU_ICSSG0 PR1	Level
INTR_ROUTER0_OUTL_INTR[399:392]	INTR_PEND[399:392] interrupts to PRU_ICSSG1 PR1	Level
INTR_ROUTER0_OUTL_INTR[407:400]	INTR_PEND[407:400] interrupts to MCU R5FSS0 CPU0 and CPU1	Level
MODSS_ECC_SEC_PEND	SEC interrupt from MODSS ECC_AGGR0	Level
MODSS_ECC_DED_PEND	DED interrupt from MODSS ECC_AGGR0	Level
UDMASS_ECC_SEC_PEND	SEC interrupt from UDMASS ECC_AGGR0	Level
UDMASS_ECC_DED_PEND	DED interrupt from UDMASS ECC_AGGR0	Level
NBSS_ECC_SEC_PEND	SEC interrupt from NBSS ECC_AGGR0	Level
NBSS_ECC_DED_PEND	DED interrupt from NBSS ECC_AGGR0	Level
VIRTSS_ECC_SEC_PEND	SEC interrupt from VIRTSS ECC_AGGR0	Level
VIRTSS_ECC_DED_PEND	DED interrupt from VIRTSS ECC_AGGR0	Level
L2G Interrupt Request Inputs		
Module Interrupt Signal	**Description**	**Type**
L2G_EVENT_PEND[7:0]	L2G interrupts from TIMESYNC_INTRTR0	Level
L2G_EVENT_PEND[15:8]	L2G interrupts from CMPEVT_INTRTR0	Level
L2G_EVENT_PEND[31:16]	L2G interrupts from GPIOMUX_INTRTR0	Level
DMA Events		

Copyright © 2024 Texas Instruments Incorporated

Table 10-94. NAVSS Hardware Requests (continued)

Module DMA Event	Description	Type
-	No PDMA channels to external DMA engines. See global event map.	-
Time Sync Event Inputs		
Module Sync Input	**Description**	**Type**
CPTS0_HW1_PUSH	CPTS Asynchronous hardware timestamp 1 push input	Pulse
CPTS0_HW2_PUSH	CPTS Asynchronous hardware timestamp 2 push input	Pulse
CPTS0_HW3_PUSH	CPTS Asynchronous hardware timestamp 3 push input	Pulse
CPTS0_HW4_PUSH	CPTS Asynchronous hardware timestamp 4 push input	Pulse
CPTS0_HW5_PUSH	CPTS Asynchronous hardware timestamp 5 push input	Pulse
CPTS0_HW6_PUSH	CPTS Asynchronous hardware timestamp 6 push input	Pulse
CPTS0_HW7_PUSH	CPTS Asynchronous hardware timestamp 7 push input	Pulse
CPTS0_HW8_PUSH	CPTS Asynchronous hardware timestamp 8 push input	Pulse
Time Sync Event Outputs		
Module Sync Output	**Description**	**Type**
CPTS0_TS_GENF0	CPTS Generation Function Output 0	Edge
CPTS0_TS_GENF1	CPTS Generation Function Output 1	Edge
CPTS0_TS_GENF2	CPTS Generation Function Output 2	Edge
CPTS0_TS_GENF3	CPTS Generation Function Output 3	Edge
CPTS0_TS_GENF4	CPTS Generation Function Output 4	Edge
CPTS0_TS_GENF5	CPTS Generation Function Output 5	Edge
CPTS0_TS_SYNC	CPTS Sync Output	Edge
Compare Event Outputs		
Module Compare Output	**Description**	**Type**
CPTS0_TS_COMP	CPTS Comparison Output	Edge

Copyright © 2024 Texas Instruments Incorporated

10.2.1.2 NAVSS Functional Description

See Section 10.1, *DMA Architecture*, for the *TI K3 Data Movement (DMA) Architecture* specification.

See the following sections for the respective module description:

UDMASS:
- Unified DMA Controller – Section 10.2.3
- Ring Accelerator – Section 10.2.4
- Packet Streaming Interface (PSI-L) – Section 10.2.8
- Block Copy DMA Controller

MODSS:
- Mailbox – Section 7.1
- Spinlock – Section 7.2
- Two Timer Managers (Timer banks) – Section 11.2
- Time Stamp Module (CPTS) – Section 11.1
- Infrastructure components:
  - CBASS – *System Interconnect*
  - Proxies – Section 10.2.5
  - Interrupt aggregators – Section 10.2.7
  - Interrupt router – *NAVSS Interrupt Router Configuration*

NBSS – North bridge – Section 10.2.9

ECC aggregators – Section 12.10.4

VirtSS – Virtualization sub-system provides all the translation components supported in K3 for virtualization:
- Peripheral Virtualization Units (PVU) – Section 8.3

Copyright © 2024 Texas Instruments Incorporated

10.2.1.3 NAVSS Interrupt Configuration

This section describes the actions needed to configure interrupts within the NAVSS. The information provided is applicable to either the NAVSS or MCU_NAVSS subsystems.

Table 10-95 are module-specific hardcoded parameters required to properly configure the NAVSS interrupt support.

Table 10-95. NAVSS Parameters

Parameter	Value NAVSS0	Value MCU_NAVSS0	Description
RA_RING_CNT	1024	286	Number of total rings supported
IA_SEVI	4608	1536	UDMA Interrupt Aggregator Source Event Input (SEVI) count
IA_VINTR	256	256	UDMA Interrupt Aggregator Virtual Interrupt (VINTR) count
IR_IBASE	See *NAVSS Interrupt Router Input Mapping*	See *Interrupt Router Input Mapping*	Interrupt Router input interrupt base number for given module (hardcoded values in interrupt router)
EO	See *Global Event Map*	See *Global Event Map*	Event offset (hardcoded offsets in NAVSS)

Table 10-96 lists software-configurable variables used in the examples and descriptions.

Table 10-96. NAVSS Software Variables

Variable	Valid Range	Description
R#	0 – RA_RING_CNT-1	Ring number
E#	0 – (destination module specific)	Event number
GE#	= EO + E#	Global event number
SB#	0 – IA_SEVI-1	Interrupt aggregator status bit number
VI#	0 – IA_VINTR-1	Virtual interrupt number
OI#	0 – 407	Interrupt router output CPU interrupt number. See *NAVSS Hardware Requests*

10.2.1.3.1 NAVSS Event and Interrupt Flow

Figure 10-14 illustrates the event and interrupt flow within the NAVSS and output interrupts to a CPU. Information that flows between NAVSS modules is also indicated (in this case, Global Event (GE) information flows between ring accelerator and the PSILSS).

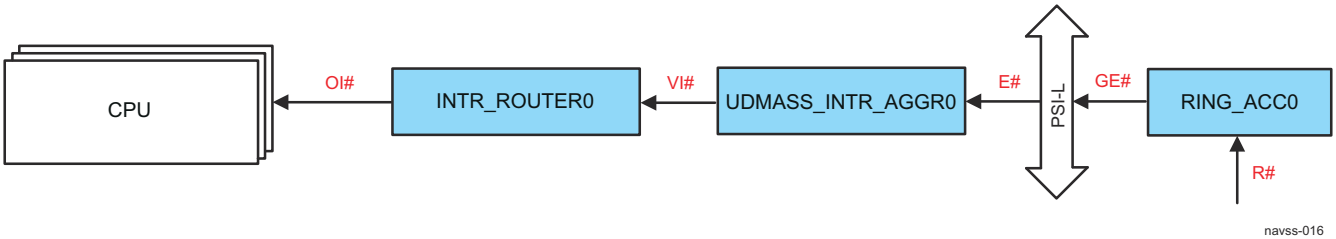


Figure 10-14. NAVSS Interrupt Flow

10.2.1.3.1.1 NAVSS Interrupts Description

This section describes the interrupt-related information that flows within NAVSS and how this information is configured or generated.

1. **Ring Number (R#)**
   - Data is read from and written to a specific Ring Accelerator ring using direct ring-mode, proxy, or secure proxy accesses
   - The Ring Accelerator has specific ring number ranges for specific purposes. The R# must match the intended purpose of the ring (see *RINGACC Ring Mapping*)
2. **Global Event and Event Numbers (GE# and E#)**
   - GE# = EO + E#

Copyright © 2024 Texas Instruments Incorporated

- EO determines the destination module for the event. The PSILSS will route the GE# to the destination module per the NAVSS event mapping, removing EO in the process. The destination module sees only E#. See *Global event Map* for details. Examples:
  - GE# in the range 0x0000 – 0x1FFF (EO = 0x0000, E# = 0x0000 – 0x1FFF) would go to NAVSS0_UDMASS_INTR_AGGR0. The INTR_AGGR sees E#.
  - GE# in the range 0x4000 – 0x47FF (EO = 0x4000, E# = 0x0000 – 0x07FF) would go to MCU_NAVSS0_UDMASS_INTR_AGGR0. The INTR_AGGR sees E#.
- For the INTR_AGGR module, E# ranges from 0 to (IA_SEVI - 1)

3. **Virtual Interrupt (VI#)**
   - The INTR_AGGR maps E# (via software configuration) to any SB#
   - SB# ranges from 0 to (IA_SEVI - 1)
   - Each VI# (0 – (IA_VINTR-1)) is driven by a group of 64 contiguous SB# numbers (VI# driven by SB#'s VI#×64 – (VI#×64)+63)
4. **Output Interrupt (OI#)**
   - The Interrupt Router has a hardwired set of interrupt inputs from various modules (see *NAVSS Interrupt Router Input Mapping*). Examples:
     - In the NAVSS, 256 interrupt inputs from the NAVSS0_UDMASS_INTR_AGGR0 are hardwired to interrupt inputs 0 – 255. The VI# is zero-relative to this range.
     - In the MCU_NAVSS, 4 event interrupt inputs from the MCRC0 are hardwired to interrupt inputs 256 – 259. The VI# is zero-relative to this range.
   - Software configures an interrupt input (based on VI# + IR_IBASE) to an OI#

10.2.1.3.1.2 Application Example

This section uses the following example to illustrate how to configure the NAVSS interrupt support for a specific use-case:

```
Using NAVSS0, use A72 MPU interrupt 7 for UDMAP0 receive ring 16 (assumptions: use up/down event
10, NAVSS0_UDMASS_INTR_AGGR0 for interrupt aggregation, and virtual interrupt 3).
```

Table 10-97 lists the modules' hardcoded parameters used for this example:

Table 10-97. NAVSS Parameters for the Application Example

Parameter	Value NAVSS0	From Table	Description
RA_UDMAP0_RX		*RINGACC Ring Mapping*	Starting ring number for UDMA0 receive channels
RA_RING_CNT	1024	*NAVSS0_RINGACC0 Configuration Parameters*	Number of total rings supported
IA_SEVI	4608	*Interrupt Aggregators Parameters*	NAVSS0_UDMASS_INTR_AGGR0 Steerable Event Input (SEVI) count
IA_VINTR	256	*Interrupt Aggregators Parameters*	NAVSS0_UDMASS_INTR_AGGR0 Virtual Interrupt (VINTR) count
IR_IBASE	0	*NAVSS Interrupt Router Input Mapping*	Interrupt-Router-input-interrupt base for NAVSS0_UDMASS_INTR_AGGR0
EO	0	*Global event Map*	Event offset. Destination is NAVSS0_UDMASS_INTR_AGGR0
CC_IBASE	0	*NAVSS0 Hardware Requests*	Interrupt-requests-to-CC base

Table 10-98 the variables and their respective calculations for this example.

Table 10-98. NAVSS Software Variables for the Application Example

Variable	Calculations/Value	Description
R#	= (RA_UDMAP0_RX + 16) + 16)	Receive ring 16 translates to ring number
E#	10	Zero-relative event number from PSILSS to NAVSS0_UDMASS_INTR_AGGR0
GE#	= (EO + E#) = (0 + 10) = 10	Global event number (event 10 to be sent to NAVSS0_UDMASS_INTR_AGGR0)

Copyright © 2024 Texas Instruments Incorporated

Table 10-98. NAVSS Software Variables for the Application Example (continued)

Variable	Calculations/Value	Description
VI#	= 3	VI# = 3 is per example assumptions. VI# could get any value 0 – IA_VINTR-1.
SB#	= 200	Must be in the range (VI#×64) – ((VI#×64)+63) = 192 – 255 in order to drive VI# 3. We will use 200 for this example.
OI#	= (A72_IBASE + 7) = (0 + 7) = 7	OI# = 7 is per example assumptions. OI# could get any value 0 – 119 to drive an A72 interrupt.

Table 10-99 shows software register writes that will configure the NAVSS to produce the desired interrupt per the example description.

Table 10-99. NAVSS Register Writes for the Application Example

Module	Register	Value to Write	Description
RINGACC0	RING[R#]_EVT (EVENT_j)	RING[]_EVT = GE# = 10	Ring event number must be specified as a global event number
INTR_AGGR0	ENTRY[E#]_INTMAP (IMAP_j)	ENTRY[10]_INTMAP = (((SB#/64)&0x1FF)<<8) | (SB#%64) = (3<<8) | 8 = 0x308	regnum = SB# / 64 bitnum = SB# % 64
INTR_AGGR0	VINT[SB#/ 64]_ENABLE_SET (ENABLE_SET_j)	VINT[3]_ENABLE_SET = 1<<(SB# % 64) = 1<<8 = 0x100	Enable interrupt
INTR_ROUTER0	INTERRUPT_CONTR OL[OI#] (MUXCNTL_y)	INTERRUPT_CONTROL[7] = IR_IBASE + VI# = 0 + 3 = 3	Route interrupt

Copyright © 2024 Texas Instruments Incorporated

10.2.2 MCU Navigator Subsystem (MCU NAVSS)

This chapter describes the integration of the MCU Navigator Subsystem (MCU NAVSS) in the device and internal module parameters.

10.2.2.1 MCU NAVSS Overview

MCU Navigator Subsystem (MCU NAVSS) has a subset of the modules of the main NAVSS and is instantiated in the MCU domain.

MCU Navigator Subsystem consists of DMA/Queue Management components – UDMA and Ring Accelerator (UDMASS), and Peripherals (Module subsystem [MODSS]).

UDMASS – UDMASS is the essential part of the TI Data Movement Architecture. UDMASS consists of:
- Unified DMA Controller
- Ring Accelerator
- Packet Streaming Interface (PSILSS)

MODSS – MODSS is a collection of peripherals with different system-level functions. NAVSS0 contains the following modules:
- Memory CRC module
- Infrastructure components such as CBASS, proxies, interrupt aggregators, and an interrupt router

ECC aggregators – for SEC/DED memory protection.

Figure 10-15 shows the MCU_NAVSS0 top-level block diagram.

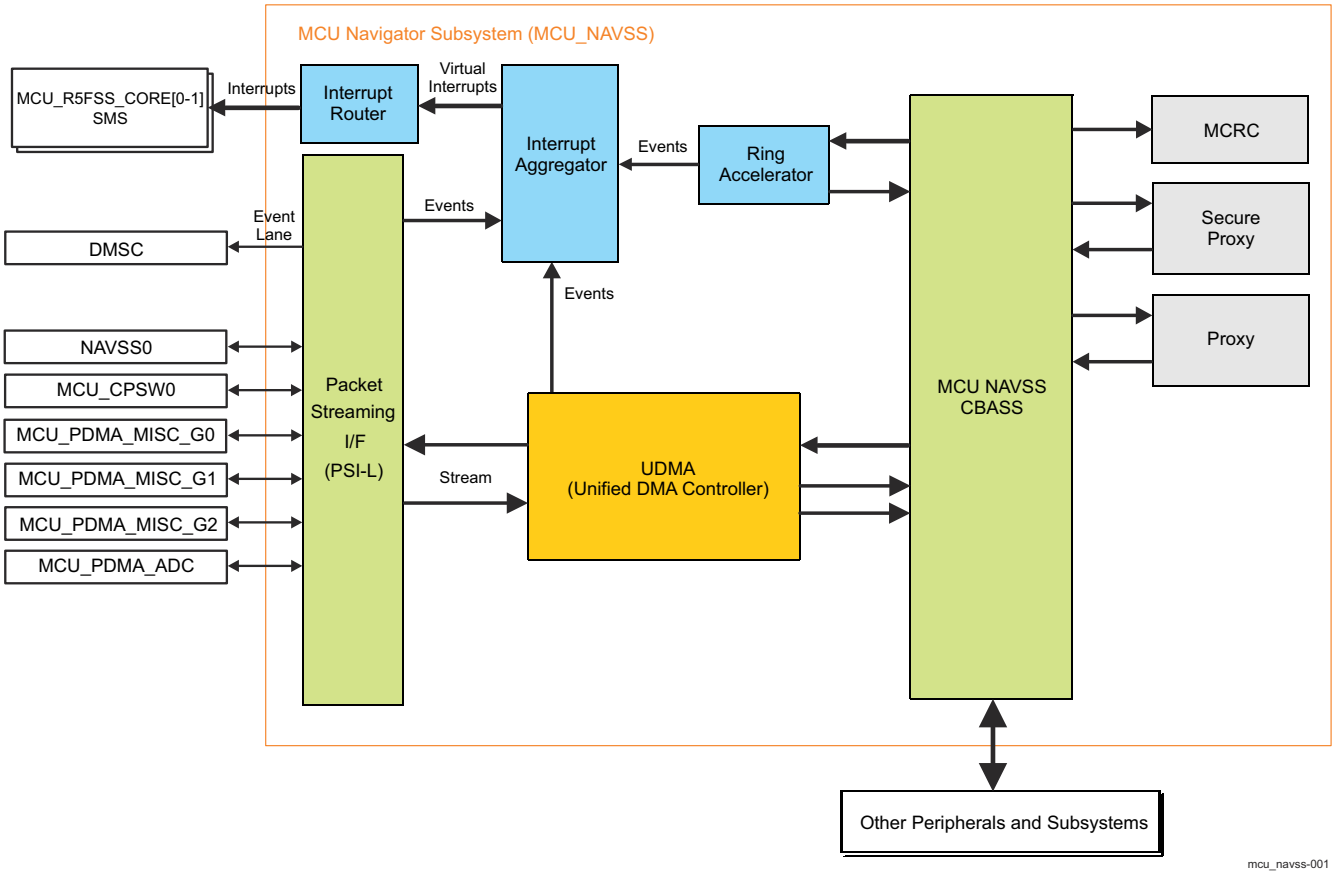


Figure 10-15. MCU NAVSS Top-Level Block Diagram

Copyright © 2024 Texas Instruments Incorporated

Note

MCU NAVSS functionality is similar to that of main NAVSS, however it is composed ot fewer peripherals and has reduced bandwidth. This chapter describes the integration and parameters specific for MCU NAVSS.

10.2.2.1.1 MCU NAVSS Ports

Table 10-100. MCU NAVSS Clocks and Resets

Clocks	
Module Clock Input	**Description**
MODSS_VBUS_D2_CLK	MODSS config interface clock. This clock is used for MCU_NAVSS modules (MODSS).
UDMASS_VBUS_D2_CLK	UDMASS config interface clock. This clock is used for UDMASS modules.
CPSW0_CLK	CPSW0 PSI-L interface clock
PDMA_MCU0_CLK	PDMA_MCU0 PSI-L interface clock
PDMA_MCU1_CLK	PDMA_MCU1 PSI-L interface clock
PDMA_MCU2_CLK	PDMA_MCU2 PSI-L interface clock
PDMA_ADC_CLK	PDMA_ADC PSI-L interface clock
Resets	
Module Reset Input	**Description**
MODSS_RST	MODSS hardware reset
UDMASS_RST	UDMASS hardware reset. Same as MODSS reset.

Table 10-101. MCU NAVSS Hardware Requests

Interrupt Requests		
Module Interrupt Signal	**Description**	**Type**
INTR_ROUTER0_OUTL_INT[31:0]	Interrupts to MCU R5FSS Core 0	Level
INTR_ROUTER0_OUTL_INT[64:32]	Interrupts to MCU R5FSS Core 1	Level
MODSS_ECC_SEC_PEND	SEC interrupt from MODSS ECC_AGGR0	Level
MODSS_ECC_DED_PEND	DED interrupt from MODSS ECC_AGGR0	Level
UDMASS_ECC_SEC_PEND	SEC interrupt from UDMASS ECC_AGGR0	Level
UDMASS_ECC_DED_PEND	DED interrupt from UDMASS ECC_AGGR0	Level
Inbound Events		
DMA Event	**Description**	**Type**
WKUP_GPIOMUX_OUTP[19:12] EVENT_PEND_INTR[3:0]	External L2G inputs into INTR_AGGR0. 4 MCRC events + 8 external.	Level

Copyright © 2024 Texas Instruments Incorporated

10.2.2.2 MCU NAVSS Functional Description

See *DMA Architecture*, for the *TI K3 Data Movement (DMA) Architecture* specification.

See the following sections for the respective module description:

UDMASS:

- Unified DMA Controller – Section 10.2.3
- Ring Accelerator – Section 10.2.4
- Packet Streaming Interface (PSI-L) – Section 10.2.8

MODSS:

- Infrastructure components:
  - CBASS – *System Interconnect*
  - Proxies – Section 10.2.5 and Section 10.2.6
  - Interrupt aggregator – Section 10.2.7

ECC aggregators – Section 12.10.4

Copyright © 2024 Texas Instruments Incorporated

10.2.3 Unified DMA Controller (UDMA)

This chapter describes the UDMA controller module, part of NAVSS.

10.2.3.1 UDMA Overview

The UDMA-P is intended to perform similar (but significantly upgraded) functions as the packet-oriented DMA used on previous SoC devices.

Table 10-102. UDMA Allocation Across Device Domains

Module Instance	Domain		
	WKUP	MCU	MAIN
NAVSS0_UDMAP0	-	-	✓ (NAVSS)
MCU_NAVSS0_UDMAP0	-	✓ (MCU NAVSS)	-

Figure 10-16 shows a top-level overview of the UDMA module.

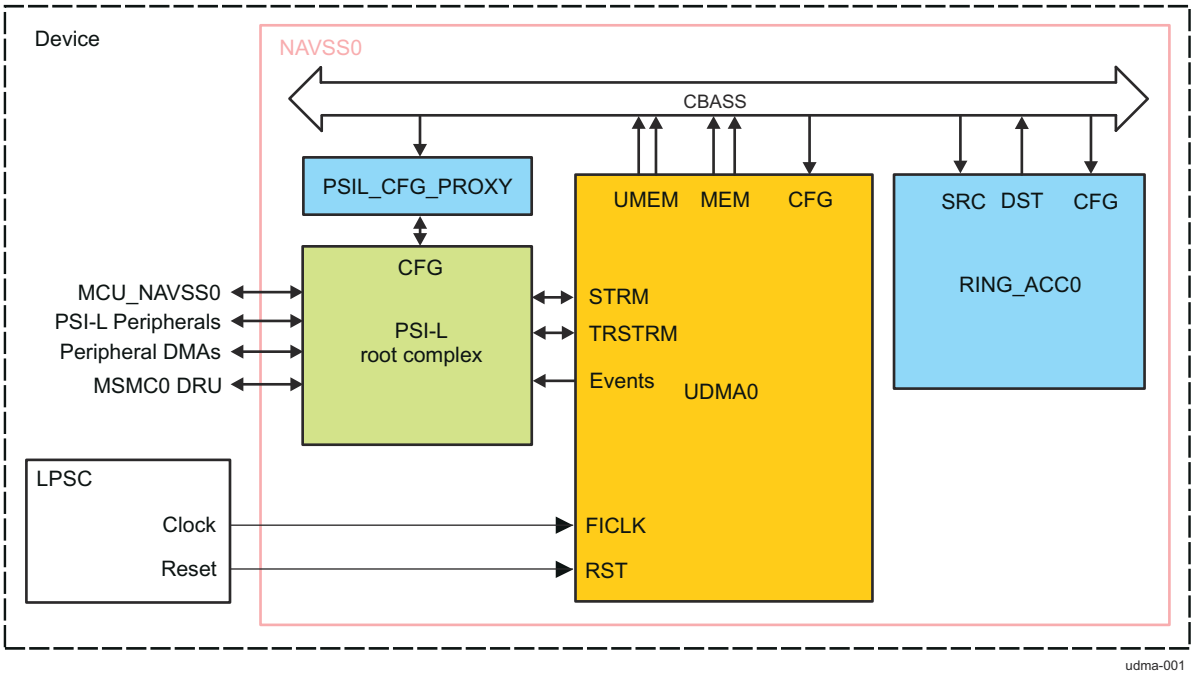


Figure 10-16. UDMA Overview

10.2.3.1.1 UDMA Features

NAVSS0_UDMAP0 controller module supports the following modes and features:

- *TI K3 DMA Architecture* compliant Tx/Rx port implementation (see Section 10.1, *DMA Architecture*)
- Implements *TI DMA Architecture*-compliant Packet-Oriented DMA Functionality (UDMA-P)
  - Provides internal Tx Packet Oriented DMA processing unit
  - Provides internal Rx Packet Oriented DMA processing unit
  - Supports *rflow_cnt* unique Rx flow table entries
  - Supports 1 transmit queue per data connection
  - Supports 1 receive queue per data connection
  - Supports physical separation of buffer control and payload information
  - Supports host and monolithic descriptor formats
  - Supports unlimited buffer scatter/gather for packets using host descriptor type
  - Supports data buffer sizes up to 4 Mbytes
  - Supports variable first valid byte offset within data buffers
  - Supports packet truncation on transmit
  - Provides per-channel buffering:

Copyright © 2024 Texas Instruments Incorporated

    - Provides 16 word deep × 128-bit Packet FIFO for each Tx channel
    - Provides 4 word deep Packet Info FIFO for each Rx channel
    - Provides 8 word deep × 128-bit Packet Data FIFO for each Rx channel
    - Supports up to 32 Protocol Specific words for Tx packets
    - Supports up to 32 Protocol Specific words for Rx packets
- Implements *TI DMA Architecture* compliant Third Party Channel Controller
  - Channel controller is built as a prefetcher engine
  - Provides the same functionality as a discrete UDMA-C/UTC combination
    - Provides a fully integrated *TI DMA Architecture* Third Party DMA compliant solution
  - Can also fetch TRs and writeback TR responses for up to *echan_cnt* external channels which are transported via the TR PSI-L interface
- Implements *TI DMA Architecture* compliant Unified Transfer Controller
  - Provides a memory read access unit
    - Supports read bursts up to 128 bytes (limited by Tx Per Channel FIFO depth for the channel)
  - Provides a memory write access unit
    - Supports write bursts up to 128 bytes (limited by Tx Per Channel FIFO depth for the channel)
  - Supports Type 0-4, and Type 15 Transfer Request types
- Provides a set of *TI DMA Architecture* compliant Unified DMA channels which all share the execution hardware using time division multiplexing
  - Supports *tchan_cnt* concurrent Tx channels
    - Each Tx channel can be configured at runtime to be:
      - Packet oriented Tx data channel
      - Third party combination source control and data channel (Third Party channel mode)
  - Supports *rchan_cnt* simultaneous Rx (destination) channels
    - Each Rx channel can be configured at runtime to be:
      - Packet oriented Rx data channel
      - Third party combination destination control and data channel (Third Party channel mode)
  - Provides support for three different groups of internal channels:
    - Ultra-high capacity (UHC) channels
      - Quantity specified by: *uchan_cnt*
      - Provide deeper Tx Per Channel FIFOs
      - Provide 16 deep descriptor/TR prefetch buffers
      - UHC channels are the first *uchan_cnt* channels within the Tx and Rx arrays (starting with channel 0 and extending up to channel *uchan_cnt*-1)
      - Tx and Rx UHC channels are specified as a matched pair
        - Still usable as split channels but design must have same number of Tx/Rx UHC channels
    - High-capacity (HC) channels:
      - Quantity specified by: *hchan_cnt* - *uchan_cnt*
      - Provide deeper Tx Per Channel FIFOs
      - Provide 16 deep descriptor/TR prefetch buffers
      - HC channels are the next *hchan_cnt* channels (after any UHC channels) within the Tx and Rx channel arrays (starting with channel *uchan_cnt* and extending up to *uchan_cnt* + *hchan_cnt*-1).
    - Normal capacity (NC) channels:
      - Quantity on Tx specified by: *tchan_cnt* - *uchan_cnt* - *hchan_cnt*
      - Quantity on Rx specified by: *rchan_cnt* - *uchan_cnt* - *hchan_cnt*
      - Provide standard depth Tx/Rx per channel FIFOs
      - Provide 1 deep descriptor/TR prefetch buffers

Note

User can still use ultra-high capacity and high-capacity channels as general normal-capacity channels though may not be taking advantage of the potential DMA throughput advantages.

  - Provides per-channel buffering:
    - Provides 16 word deep data FIFO for each NC Tx (source) channel
    - Provides 8 word deep data FIFO for each NC Rx (destination) channel

Copyright © 2024 Texas Instruments Incorporated

- Provides single 128-bit read/write VBUSM master interface for prefetcher accesses to RINGACC and memory.
  - Supports up to 16 outstanding reads.
  - Supports up to 16 outstanding writes.
- Provides single 128-bit read/write VBUSM master interface for packet DMA and coherency unit accesses to RINGACC and memory.
  - Supports up to 16 outstanding reads.
  - Supports up to 16 outstanding writes.
- Provides single 128-bit read only VBUSM master interface for payload transfers.
  - Supports up to 16 outstanding reads.
- Provides single 128-bit write only VBUSM master interface for payload transfers.
  - Supports up to 16 outstanding writes.
- Provides 128-bit wide PSI-L compliant source interface for sending data to remote UTCs and remote peripherals
  - Includes 1 outgoing event transport lane
- Provides 128-bit wide PSI-L compliant destination interface for receiving data from remote UTCs and remote peripherals
  - Includes 1 incoming event transport lane
- Provides PSI-L interface for sending Transfer Requests to remote UTC channels and receiving back Transfer Responses
  - Provides 32 entry deep unified Transfer Response FIFO for external channels
- Provides 0 local event input buses

Copyright © 2024 Texas Instruments Incorporated

10.2.3.1.2 UDMA Parameters

Table 10-103 shows the UDMA configuration parameters in this SoC.

Table 10-103. UDMA Configuration Parameters

Module Instance	Parameters[1]					
	tchan_cnt[2]	rchan_cnt[3]	echan_cnt [4]	hchan_cnt[6]	uchan_cnt[7]	rflow_cnt[5]
NAVSS0_UDMAP0	85	82	256	4	2	224
MCU_NAVSS0_UDMAP0	48	48	0	2	0	96

(1) Parameter values can be read from the capabilities registers. See UDMA_CAP2 and UDMA_CAP3 registers.
(2) Total internal Tx channel count (Normal- + High- + Ultra-high capacities)
(3) Total internal Rx channel count (Normal- + High- + Ultra-high capacities)
(4) External UTC channel count
(5) Rx flow table entry count
(6) High-capacity + Ultra-high capacity channel count
(7) Ultra-high capacity channel count

10.2.3.1.3 Unified DMA Controller (UDMA) Ports

Table 10-104. UDMA Clocks and Resets

Clocks	
Module Clock Input	**Description**
UDMAP0_FICLK	UDMA clock. This clock is used for all interface and functional operations.
Resets	
Module Reset Input	**Description**
UDMAP0_RST	UDMA hardware reset

Table 10-105. UDMA Hardware Requests

Interrupt Requests		
Module Interrupt Signal	**Description**	**Type**
-	The module does not generate traditional (local) interrupts	-
Outbound Events		
Module Event	**Description**	**Type**
TX Complete	To other UDMA/UTC/PDMA instances on the device for synchronization and triggering purposes.	ETL Push
RX Complete	To other UDMA/UTC/PDMA instances on the device for synchronization and triggering purposes.	ETL Push
Error		ETL Push
Inbound Events		
Module Event	**Description**	**Type**
TXCH0_TRG0	Tx Channel 0 Trigger 0 in third party mode	ETL Push
TXCH0_TRG1	Tx Channel 0 Trigger 1 in third party mode	ETL Push
...	...	ETL Push
TXCH139_TRG0	Tx Channel 139 Trigger 0 in third party mode	ETL Push
TXCH139_TRG1	Tx Channel 139 Trigger 1 in third party mode	ETL Push
RXCH0_TRG0	Rx Channel 0 Trigger 0 in third party mode	ETL Push
RXCH0_TRG1	Rx Channel 0 Trigger 1 in third party mode	ETL Push
...	...	ETL Push
RXCH139_TRG0	Rx Channel 139 Trigger 0 in third party mode	ETL Push
RXCH139_TRG1	Rx Channel 139 Trigger 1 in third party mode	ETL Push

Copyright © 2024 Texas Instruments Incorporated

10.2.3.2 UDMA Functional Description

The UDMA module supports the transmission and reception of various packet types. The UDMA is architected to facilitate the segmentation and reassembly of K3 DMA data structure compliant packets to/from smaller data blocks that are natively compatible with the specific requirements of each connected peripheral. Multiple Tx and Rx channels are provided within the DMA which allow multiple segmentation or reassembly operations to be ongoing. The DMA controller maintains state information for each of the channels which allows packet segmentation and reassembly operations to be time division multiplexed between channels in order to share the underlying DMA hardware.

An internal DMA scheduler is used to control the ordering and rate at which this multiplexing occurs for Transmit operations. The ordering and rate of Receive operations is indirectly controlled by the order in which blocks are pushed into the DMA on the Rx PSI-L interface.

The UDMA also supports acting as a Unified Channel Controller (UDMA-C)/Unified Transfer Controller (UTC) combined unit which accepts Transfer Request packets from Ring Accelerator and then performs the transfers which the Transfer Request (TR) specifies.

Channels in the UDMA can be configured to be either Packet-based or TR-based Third Party channels on a channel by channel basis.

10.2.3.2.1 Block Diagram

Figure 10-17 shows UDMA main internal components.

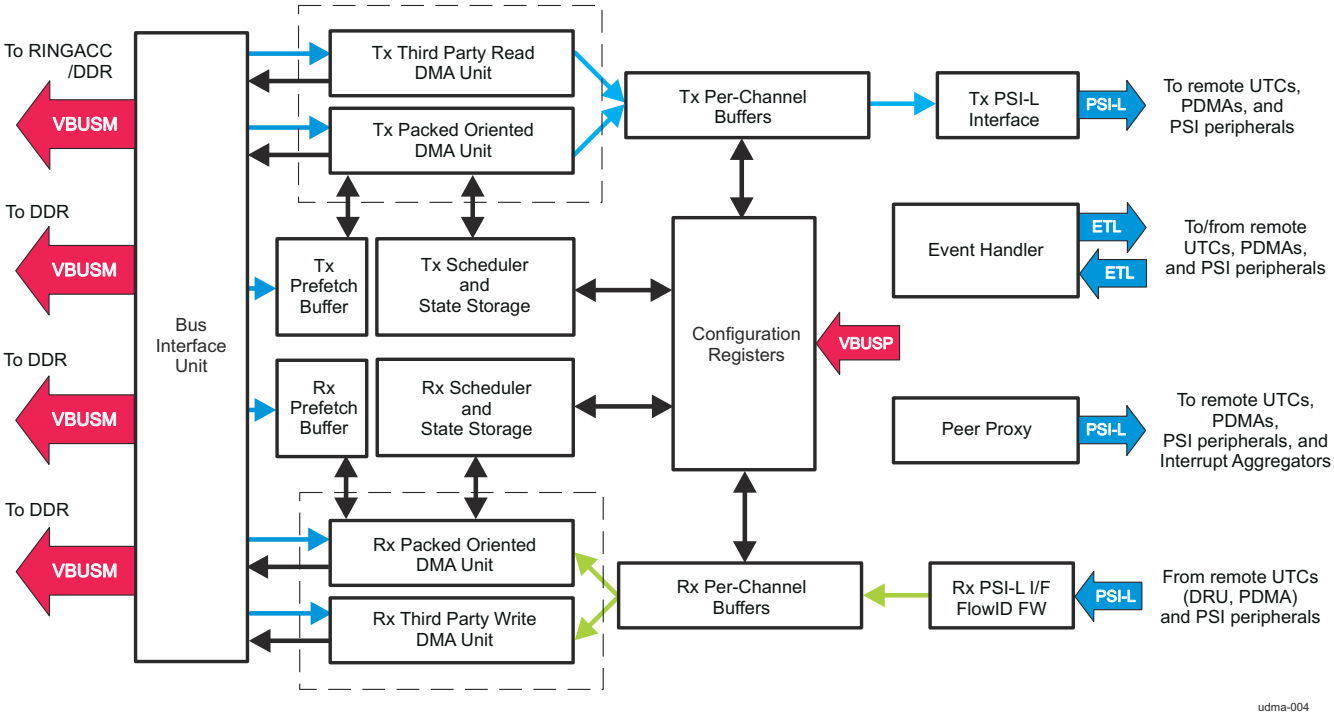


Figure 10-17. UDMA Block-Diagram

Bus Interface Unit

The Bus Interface Unit (BIU) is responsible for merging and buffering all of the transactions that originate from the various master blocks inside the DMA controller into the 4 separate VBUSM master interfaces. Arbitration between the blocks to a given VBUSM interface is round robin within a single priority level. A two word deep retiming buffer is provided on each sub interface of each provided VBUSM bus.

Tx Prefetcher Configuration Registers

The Tx Prefetcher Configuration Registers block is responsible for providing memory mapped registers for configuration of the Tx DMA function, monitoring the fullness level of the Tx prefetch buffers, maintaining Tx

Copyright © 2024 Texas Instruments Incorporated

prefetch state information, arbitrating which channel will be allowed to perform prefetch work next, issuing scheduler commands to the Tx Prefetch Units, and writing back the updated state returned from those same prefetch units.

Tx Prefetch Unit(s)

The Tx Prefetch Unit block is responsible for performing a fetch of either a Packet Descriptor (packet mode) or a TR to feed the downstream Tx DMA/Tx Read Units.

Tx Configuration Registers

The Tx Configuration Registers block (*UDMASS_UDMAP0_CFG_TCHAN Registers*) is responsible for monitoring the fullness level of the Tx Per Channel FIFOs, monitoring data transfer work which is pending, maintaining data movement thread state information, arbitrating which channel will be allowed to perform work next, issuing scheduler commands to the Tx Packet and TR based DMA core blocks, and writing back the updated state returned from those same DMA core blocks.

Tx Packet DMA Unit

The Tx Packet DMA Unit block implements all of the state machine functionality necessary to implement the K3 DMA Host and Monolithic Tx protocol. The Tx Packet DMA unit initiates VBUSM transactions in order to read and write descriptor pointers from the Ring Accelerator, read descriptors from memory, and read data from buffers in memory.

Tx Packet Coherency Unit

The Tx Packet Coherency Unit is responsible for ensuring that all control structures and data have been read (and updated if applicable) by the Tx DMA unit(s) prior to returning the packet descriptor pointer to the appropriate return queue in the Ring Accelerator. This unit ensures that the ordering of the Packet Descriptor pointer writes to the return queue directly matches the ordering in which those packets were fetched from the Tx queue. This unit is only used on channels which are in Pass-By-Reference mode

Tx TR Coherency Unit

The Tx TR Coherency Unit is responsible for ensuring that all control structures and data have been read prior to allowing the TR response to be written to either the Packet Descriptor or a Pass by Value queue in the Ring Accelerator. This unit ensures that the ordering of TR response writes also directly matches the ordering of Transfer Requests that were processed by the channel. This unit is only used on channels which are in TR mode.

Tx External Channel TR Coherency Unit

The Tx External Channel TR Coherency Unit is responsible for ensuring that TRs received back from remote UTCs are written in strong order to either the Packet Descriptor or a Pass-by-Value queue in the Ring Accelerator. This unit is only used on external UTC channels.

Tx Event Coherency Unit

The Tx Event Coherency Unit is responsible for ensuring that all data transfers have completed before a completion event is issued through the outgoing Event Transport Lane (ETL). This unit is only used on channels which are in TR mode.

Tx Per Channel Buffers

The Tx Per Channel Buffers implement a FIFO for each Tx DMA channel that is used for buffering packet control and payload data that has been fetched by the Tx Packet DMA units or Tx Third Party Read unit modules. The buffers are byte oriented on write so that the data from the DMA units which may not be full words can be packed properly. The buffers are block oriented on read in accordance with the transport mechanism outlined in the PSI-L Interface specification. Each Tx (source) channel in the UDMA controller maps directly onto a thread in the Tx PSI-L interface. The Tx Per Channel Buffer block outputs queue fullness information to the Tx Scheduler block which it then uses to determine when it must initiate DMA opportunities to backfill the buffers. The Tx Per Channel Buffer will initiate transfers to the remote paired thread whenever any data is available in each channel buffer and credits are available in the corresponding thread. The block will simultaneously monitor the status

Copyright © 2024 Texas Instruments Incorporated

of all of the threads and will perform a round robin arbitration between the different threads for the use of the Transmit PSI-L interface. Each thread for which the target is indicating it can accept data and which currently has data available in the channel buffer will be included in the arbitration.

Rx FlowID Firewall

The Rx FlowID firewall checks the incoming flowID on the received packet from the Rx PSI-L interface to verify that it is either the corresponding flowID for the channel (that is, the default flow ID for the channel which is the same as the channel number) or that the flowID falls within a programmed range that is considered legal for the channel. If the received flowID does not fall within the legal range then the packet is dropped and the error is trapped.

Rx Per Channel Buffers

The Rx Per Channel Buffers implement a FIFO for each Rx DMA channel that is used for buffering packet control and payload data that has been pushed into the DMA from the Rx PSI-L interface. The Rx Per Channel Buffers also includes an arbitration unit which determines which Rx DMA channel must be serviced next

Rx Configuration Registers

The Rx Configuration Registers block (*UDMASS_UDMAP0_CFG_RCHAN Registers*) is responsible for providing memory mapped registers for configuration of the Rx DMA functions including the default settings for the free descriptor and destination queues. For modularity and high speed pipelining reasons, the Rx traffic is looped through the Rx Configuration Registers block where the original stream information is merged with information from the configuration registers on its way to the Rx DMA unit module. This prevents the Rx DMA Core from having to spend cycles accessing the channel configuration information for the channel.

Rx Packet DMA Unit

The Rx Packet DMA Unit block implements all of the state machine functionality necessary to implement the K3 DMA Rx protocol for Host and Monolithic descriptor types. The Rx Packet DMA Unit initiates VBUSM transactions in order to read descriptor pointers from the Ring Accelerator, read buffer descriptor information, write descriptors to memory, and write data to buffers in memory.

Rx Packet Coherency Unit

The Rx Packet Coherency Unit is responsible for ensuring that all control structures and data have been written by the Rx DMA unit(s) prior to returning the packet descriptor pointer to the appropriate return queue in the Ring Accelerator. This unit ensures that the ordering of the Packet Descriptor pointer writes to the return queue directly matches the ordering in which those packets were fetched from the Rx free queues. Writes are not considered complete by this unit until the entire write status has been returned for all outstanding transactions for a given packet ID. This unit is only used on channels which are in Pass By Reference mode

Rx TR Coherency Unit

The Rx TR Coherency Unit is responsible for ensuring that all control structures have been read and all data has been written prior to allowing the TR response to be written to either the Packet Descriptor or a Pass by Value queue in the Ring Accelerator. This unit ensures that the ordering of TR response writes also directly matches the ordering of Transfer Requests that were processed by the channel. This unit is only used on channels which are in TR mode.

Rx Event Coherency Unit

The Rx Event Coherency Unit is responsible for ensuring that all data transfers have completed before a completion event is issued through the outgoing ETL. This unit is only used on channels which are in TR mode.

Third Party Read Unit

The Third Party Read Unit(s) are responsible for sequencing all of the Tx side channel control and data transfers. The Third Party Read Unit is responsible for performing the actual data read operations including sequential address generation and nested loop control. Like the Tx Packet DMA Unit, the Third Party Read Unit receives state information from the Tx Configuration Registers and returns that state when a transfer opportunity is complete. The Third Party Read Unit will perform as much data transfer as is specified in the Transfer Request

Copyright © 2024 Texas Instruments Incorporated

up until either a fixed number of bytes have been transferred, the transfer is completed, or the transfer has reached a point which requires an input trigger event to proceed.

Third Party Write Unit

The Third Party Write Unit(s) are responsible for sequencing all of the Rx side channel data transfers. The Third Party Write Unit is responsible for performing the actual data write operations including sequential address generation and nested loop control. Like the Rx Packet DMA Unit, the Third Party Write Unit receives state information from the Rx Configuration Registers and returns that state when a transfer opportunity is complete. The Third Party Write Unit will perform as much data transfer as is specified in the Transfer Request up until either a fixed number of bytes have been transferred or the transfer is completed or the transfer has reached a point which requires an input trigger event to proceed.

Event Handler

The Event Handler block is responsible for accepting and logging channel triggering events and for generating channel completion and error events.

PSI-L Real Time Proxy

The PSI-L Real Time Proxy block is responsible for allowing tunneled VBUSP transactions to the Real Time Remote Peer registers to generate PSI-L configuration transactions to the paired remote peer for each Tx and Rx channel.

10.2.3.2.2 General Functionality

This section is applicable to all UDMA modes.

10.2.3.2.2.1 Operational States

At any given time the UDMA can be in one of three different states as described in Table 10-106.

Table 10-106. Operational States

Operational State	Description
Init	This is the initial state of UDMA during and immediately after reset. During this state, all of the RAMs inside the UDMA will be initialized to known values including the ECC redundant parity bits. While in the Init state, the DMA will de-assert all ready signals on all applicable slave interfaces and will de-assert all request signals on all applicable master interfaces. The UDMA will automatically transition out of the Init state into the Idle state when all of the RAM initialization has been completed.
Idle	Once the UDMA leaves the Init state, it enters the Idle state whenever no outstanding transactions are pending on any of the UDMA interfaces (master or slave). The Idle state is generally a transient state and is used by the UDMA to determine when it is appropriate to allow the SoC power management complex to turn off the clock. As channels have work queued on them and transactions begin flowing into the system, the UDMA transitions to the Active state. When no more work is pending, or when the host pauses/disables active channels, or when the SoC power management desires to shut down the UDMA, the UDMA will account for any outstanding transactions and will re-enter the Idle state. The UDMA leaves the Idle state anytime it generates or receives a transactions that requires a return response as those protocols dictate that the clock must remain running to avoid faulting the handshaking protocol.
Active	The UDMA enters the Active state as soon as it issues a transaction or receives a transaction on any interface that uses a split protocol (expects a later response for a request). When all transactions have been accounted for (responses have all been either received or sent) the UDMA transitions to the Idle state.

10.2.3.2.2.2 Tx Channel Allocation

In NAVSS0, a total of 140 Packet Tx channels are provided within the DMA for concurrent data transfers between memory mapped space and the Tx Per Channel Buffers. Each of these channels can be configured to operate as a packet oriented channel (uses queues/rings/descriptors/buffer) or as a Third Party DMA source channel (uses Transfer Request packets to control read operations). Depending on which channel mode is selected, when a Tx channel comes into context the work for that channel will either be dispatched to a Tx Packet DMA Unit or a Third Party Read Unit respectively.

The Tx channels are allocated as shown in Table 10-107.

Copyright © 2024 Texas Instruments Incorporated

Table 10-107. Internal Tx Channel Allocation

DMA Channel	Function	Tx Queue (Ring)
0	Tx Channel	0 (Starting queue number in RINGACC for Tx channel 0)
...	...	...
139	Tx Channel	139

10.2.3.2.2.3 Rx Channel Allocation

In NAVSS0, a total of 140 Rx channels are provided within the DMA for concurrent transfers between the Rx Per Channel Buffers and memory mapped space. Each of these channels can be configured to operate as a packet oriented channel (uses rings/descriptors/buffer) or as a Third Party DMA destination channel (uses rings/ Transfer Request packets or TR standalone records to control read operations). Depending on which channel mode is selected, when a Tx channel comes into context the work for that channel will either be dispatched to an Rx Packet DMA Unit or a Third Party Write Unit respectively. The Rx channels are allocated as shown in Table 10-108.

Table 10-108. Rx Channel Allocation

DMA Channel	Function	Src Tag
0	Rx Channel	0
...	...	...
139	Rx Channel	139

10.2.3.2.2.4 Tx Teardown

As is specified in the *TI DMA Architecture* specification, the host initiates a Tx DMA channel teardown operation by writing to the TDOWN bit in the TCHANRT UDMA_TRT_CTL_j register. Once a channel teardown is requested, the appropriate Tx DMA unit will complete any packet/TRs that the channel has prefetched or is currently in the process of transferring and will then clear the EN bit, will send a Teardown packet out the outbound Payload Data PSI-L interface, and will push a Teardown Completion Record onto the Tx Default Queue for the channel. If the Tx DMA engine is not currently in a packet/TR for the channel, it will immediately clear the EN, send the Teardown packet, and push the Teardown Completion Record. Once the teardown is complete, no further packet processing will occur until the host software re-enables the channel.

10.2.3.2.2.5 Rx Teardown

As is specified in the *TI DMA Architecture* specification, the host initiates an optimal Rx DMA channel teardown operation by shutting down the incoming payload stream at its source (a remote peer peripheral or the Tx side of a block copy channel). Once teardown is initiated at the source, data will eventually drain from the incoming PSI-L stream and when all data is complete, the remote peer entity will send a Teardown packet (either a standalone packet or tdown asserted with eop on the last data phase of the last packet). When the UDMA receives a teardown message, the TDOWN bit will be set in RCHANRT UDMA_RRT_CTL_j. In the case where a remote peer cannot generate a teardown message, the host must ensure that all data has stopped flowing from the remote peer and will then directly set the TDOWN bit. Once the UDMA sees the TDOWN asserted it will wait for any existing packets which are currently held in the Rx Per Channel FIFO to be completed and will then de-assert the EN bit in RCHANRT UDMA_RRT_CTL_j. If the back end application continues to allow new packets to be pushed into the Rx PC FIFO, the DMA will continue to delay marking the channel as torn down as long as the Rx PC FIFO never reaches an empty state

10.2.3.2.2.6 Tx Clock Stop

The Tx clock stop interface allows the Tx portion of the UDMA to be gracefully commanded to shut down its operations and enter into an IDLE state so that the main clock (FICLK) can be stopped. When the tcs_clkstop_req input is asserted, the Tx portion of the UDMA will stop processing transmit packets/TRs for each channel at the next packet/TR boundary. Once all of the Tx channels have gracefully stopped transmission, the UDMA will assert the tcs_clkstop_ack output. Once the UDMA Tx engine has entered the IDLE state, it will remain there until the tx_clockstop_req is de-asserted.

Copyright © 2024 Texas Instruments Incorporated

10.2.3.2.2.7 Rx Clock Stop

The Rx clock stop interface allows the Rx portion of the UDMA to be gracefully commanded to shut down its operations and enter into an IDLE state so that the main clock (FICLK) can be stopped. When the rcs_clkstop_req input is asserted, the Rx portion of the UDMA will stop processing receive packets for each channel at the next packet boundary. Once all of the Rx channels have gracefully stopped reception, the UDMA will assert the rcs_clkstop_ack output. Once the UDMA Rx engine has entered the IDLE state, it will remain there until the rx_clockstop_req is de-asserted.

10.2.3.2.2.8 Rx Thread Enables

PSI-L threads must first be enabled in order to transfer commands or data. Threads are enabled or disabled by setting or clearing the enable bit in the PSI-L pairing registers for the thread. When a thread is disabled, it must drop any data phases which are sent but properly return the credits for the data phases which are dropped.

10.2.3.2.2.9 Events

Events are used in the *TI DMA Architecture* for triggering hardware or software (by being turned into interrupts) processes. Physical (or local) events in the *TI DMA Architecture* are implemented as active-high, single-cycle, long, synchronous pulses which are strobed to communicate that a particular condition has occurred. Virtual (or global) events are implemented as a value which is passed across a shared bus (ETL) - one value per cycle. Physical (local) events can be used by the UDMA to communicate with some local peripherals. Virtual (global) events are used by the UDMA to communicate with remote peripherals, the interrupt aggregator, and other DMA instances in the system.

10.2.3.2.2.9.1 Local Event Inputs

There are no local events routed to the UDMA local event inputs in the current NAVSS implementation.

10.2.3.2.2.9.2 Inbound Tx PSI-L Events

The Tx PSI-L interface includes a single inbound event transport lane (ETL) which allow events to be received from other UDMA/UTC/PDMA instances for synchronization and triggering purposes. Events passed in from this event transport lane will already have been decoded as destined for some event destination index for this particular UDMA. The appropriate LSBs of each event number are extracted and the specified event is asserted.

The event map for inbound events to the UDMA is shown in Table 10-109.

Table 10-109. Inbound Event Map

Event Number	Target
0	Tx Channel 0 Global Trigger 0
1	Tx Channel 0 Global Trigger 1
2	Tx Channel 1 Global Trigger 0
3	Tx Channel 1 Global Trigger 1
...	...
279	Tx Channel 139 Global Trigger 1
280	Rx Channel 0 Global Trigger 0
281	Rx Channel 0 Global Trigger 1
282	Rx Channel 1 Global Trigger 0
283	Rx Channel 1 Global Trigger 1
...	...
559	Rx Channel 139 Global Trigger 1

10.2.3.2.2.9.3 Outbound Rx PSI-L Events

The Rx PSI-L interface includes a single outbound event transport lane (ETL) which allows internal events to be sent to other UDMA/UTC/PDMA instances on the device for synchronization and triggering purposes. The destination event number is programmed for each event source within the UDMA. See registers:

- TCHAN UDMA_TOES_j
- TCHAN UDMA_TEOES_j

Copyright © 2024 Texas Instruments Incorporated

- RCHAN UDMA_ROES_j
- RCHAN UDMA_REOES_j

10.2.3.2.2.10 Emulation Control

The emulation control input and register bits (SOFT and FREE in the UDMA_EMU_CTRL register) allow DMA operation to be suspended. When the emulation suspend state is entered, the DMA will stop processing receive and transmit packets for each channel at the next packet boundary. Any packet currently in reception or transmission will be completed normally without suspension. Emulation control is implemented for compatibility with other peripherals. Table 10-110 shows the operations of the emulation control input and register bits.

Table 10-110. Emulation Behavior Control

Emulation Control Input	SOFT	FREE	Description
0	X	X	Normal Operation
1	0	0	Normal Operation
1	1	0	Emulation Suspend
1	X	1	Normal Operation

10.2.3.2.3 Packet Oriented Transmit Operation

Packet transmission is accomplished within the UDMA by moving data from structures that are located in memory via the VBUSM Memory Interfaces onto the Transmit Packet Data PSI-L Interface. This movement of data is performed in blocks whose size is specified by the Tx DMA scheduler.

10.2.3.2.3.1 Packet Mode VBUSM Master Interface Command ID Selection

The UDMA keeps a transaction command scoreboard for each specific direction for the Packet Mode VBUSM Master interface. The scoreboard tracks which command indexes are currently outstanding from the DMA on that interface. When a read or write is requested by one of the internal master agents in the UDMA, the scoreboard corresponding to the specified interface and for the specified direction (read/write) is queried starting at index 0 and extending up to the maximum index for that scoreboard. The lowest index that is found and which is not currently allocated is selected and is directly used as the command ID (*cid*). The index values for each scoreboard. Entries are freed for re-use in a write scoreboard when write status responses are received which account for all of the outstanding write data. Entries are freed for re-use in the read scoreboard only when all data has been returned for the read.

10.2.3.2.4 Packet Oriented Receive Operation

Packet reception is accomplished within the UDMA by moving data from the Receive PSI-L Interface to data structures that are located in memory accessible via the VBUSM Memory Interface(s).

10.2.3.2.4.1 Rx Packet Drop

The Rx PSI-L Interface master may assert the rstrmp_drop signal co-incident with the transfer of any data phase of a packet. When this signal is sampled asserted by the UDMA, the drop condition will be latched into a flag for that thread and the Rx Packet DMA unit will be notified that the packet is to be flushed and recycled rather than passed to the host. The drop signal is only required to be asserted with a single data phase and will only cause the current packet to be dropped from packet stream. The UDMA will drop all data given to it until the EOP is found on the thread and then the internal drop flag will be cleared. When the UDMA drops a packet on a channel configured for Packet mode there is no notification to the host.

10.2.3.2.4.2 Rx Starvation and the Starvation Timer

As described in the *TI DMA Architecture* specification, whenever the Rx DMA engine needs to read a Free Descriptor Pointer from the Ring Accelerator, it is possible that the queue may be empty. When this occurs it is referred to as buffer starvation. The Ring Accelerator provides a message interface to the UDMA which provides increment and decrement information so that the UDMA can keep a local occupancy counter for each ring that it can access from the RINGACC. When the Rx engine requires a free descriptor it will first check to see if the local occupancy for that ring is non-zero. If the occupancy is zero, starvation has occurred. When starvation occurs the Rx DMA will respond in one of two ways depending on the value of the rx_error_handling bit in Rx Flow Table entry that is currently being used for that particular packet reception opportunity. If the rx_error_handling bit is 0, the Rx DMA will drop the packet and will return any descriptors which it may have previously allocated

Copyright © 2024 Texas Instruments Incorporated

for that packet to the respective queues from which they were allocated. If the rx_error_handling bit is set, the Rx DMA is required to retry checking to see if a Free Descriptor is available at a later time.

A timer is provided in order to control the rate at which the retry attempts will occur. All channels share the same timer and the period of the timer is set using the TIMEOUT_CNT field in the UDMA_PERF_CTRL register. The TIMEOUT_CNT field specifies in clock cycles the minimum delay between subsequent attempts to check if a Free Descriptor is available on each individual channel.

Example: If the TIMEOUT_CNT is set to 256 and channel 1 finds its Free Descriptor queue empty on cycle 10, it will be required to wait until at least cycle 266 before it can make another attempt check the queue again. The actual number of cycles which will be waited by the DMA will typically be larger than the TIMEOUT_CNT value and this is not intended to be a highly precise operation. The critical requirement is that the DMA will not attempt to bring a channel into context to check if a buffer is available at a more frequent rate than is allowed.

10.2.3.2.5 Third Party Mode Operation

10.2.3.2.5.1 Events and Flow Control

The following sections describe the mechanisms that are provided for supporting flexible flow control between peripherals, the UDMA and other DMA instances.

10.2.3.2.5.1.1 Channel Triggering

Channels which are configured in Third Party mode must be triggered in order for them to perform work. Triggering can be configured to be immediate, one-shot, or periodic at different levels of TR completion and is specified in the TR itself. Each channel in the UDMA is capable of being triggered by two separate events which are mapped at specific offsets in a global event target map (just like an address map for memory mapped slaves). The UDMA maps its channel triggers with Tx side channels first followed by Rx side channels as described in Table 10-109, *Inbound Event Map.*

10.2.3.2.5.1.2 Internal TR Completion Events

Each TR which is processed, provides the ability to assert 1 of 4 possible output events at specific completion levels for the TR. Since the output event is specified directly in the TR and the TRs are written by potentially untrusted software processes, channels are only able to assert *virtual events* when they complete various portions of the TR. These virtual events are then mapped to actual destination events in the global event map by the TCHAN UDMA_TOES_j or RCHAN UDMA_ROES_j event steering registers. These registers are programmed statically by a trusted software process that has determined the channels that should have access to the specified event destination slots.

10.2.3.2.5.2 Transmit Operation

10.2.3.2.5.2.1 Transfer Request

After reset, all channels that are configured in Third Party transfer mode will be idle and waiting for work to be assigned to them. In order to initiate Third Party channel operation, the host software places work in the form of either a Transfer Request Descriptor or an individual Transfer Request record onto the Tx Queue of the channel. The exact mechanisms which are used and the contents of both the Transfer Request Descriptor and the Transfer Request record are given in the *TI DMA Architecture* specification.

10.2.3.2.5.2.2 Transfer Response

At the completion of each set of operations which fulfill a Transfer Request record, the UDMA will return a Transfer Response record to either the original Transfer Request Descriptor or the Tx Completion Queue, depending on the mechanism which was used to provide the Transfer Request record. If the Transfer Request record originated in a Transfer Request Descriptor then a corresponding slot in the Transfer Request Descriptor will be overwritten with the response status. If the Transfer Request record originated in a pass by value Tx Ring, the Transfer Response record will be placed in a pass by value Tx Completion Ring. The Transfer Response record will only contain a completion code which indicates if the transfer was a success or if a particular type of failure occurred. The exact mechanisms which are used and the contents of the Transfer Response Record are given in the *TI DMA Architecture* specification.

Copyright © 2024 Texas Instruments Incorporated

10.2.3.2.5.2.3 Data Transfer

Packet transmission is accomplished within the UDMA by moving data from structures that are located in memory via the VBUSM Memory Interface(s) onto the Transmit PSI-L Interface. On the Tx side of the UDMA, these transfers are always reads. Data is read from an attached memory mapped space and packed into the Rx Per Channel Buffer for that channel. At a later time, the data is moved from the Tx Per Channel FIFO to a remote peer DMA entity via the Tx PSI-L interface.

10.2.3.2.5.2.4 Memory Interface Transactions

The sequence of control transactions that are performed by the UDMA on the Read/Write VBUSM interface during transmit is to read a Transfer Request record from a logical work queue and to then return a corresponding Transfer Response record to a separate logical work queue when all the transfers in the request have been completed. This sequence repeats for as long as Transfer Request records are provided on the work queue. The exact details of the operations that are involved are covered in the *TI DMA Architecture* specification (see Section 10.1).

The sequences of data transactions that are performed by the UDMA on the Read Only VBUSM interface during transmit can be described as a sequence of piecewise linear read transfers whose starting addresses are each sequentially calculated based on loops and offsets given in the Transfer Request record. A TR can have up to 4 levels of nested transfer sets where each set can have a different element count and stride between elements. TR record formats are described in detail in the Transfer Request format specification.

10.2.3.2.5.2.5 Error Handling

In the case that an error occurs during any Tx operation, the UDMA will stop processing the TR at that point and will return an appropriate error code in the Transfer Response message as specified in the *TI DMA Architecture* specification. The PAUSE_ON_ERR bit in the TCHAN UDMA_TCFG_j registers controls whether or not the DMA will pause operation on that channel if an error or exception occurs. If the PAUSE_ON_ERR bit is set to 1 the channel will be paused so that the host can optionally read the channel state to determine where in the transfer the channel execution was. If the PAUSE_ON_ERR bit is 0 the DMA will flush the current Transfer Request record for channels in pass by value mode and will flush the current Transfer Request Descriptor for channels in pass by reference mode.

10.2.3.2.5.3 Receive Operation

10.2.3.2.5.3.1 Transfer Request

After reset, all channels that are configured in Third Party mode will be Idle and waiting for work to be assigned to them. In order to initiate Third Party channel operation, the Host places work in the form of either a Transfer Request Descriptor or an individual Transfer Request record onto the Rx TR Queue of the channel. The exact mechanisms which are used and the contents of both the Transfer Request Descriptor and the Transfer Request record are given in the *TI DMA Architecture* specification.

10.2.3.2.5.3.2 Transfer Response

At the completion of each set of operations which fulfill a Transfer Request, the UDMA will send a Transfer Response message back to either the Packet Descriptor (pass by reference channel mode) or to the completion queue in the Ring Accelerator (pass by value channel mode). The contents of this message are described in detail in the *TI DMA Architecture*.

10.2.3.2.5.3.3 Error Handling

In the case that an error occurs during any Rx operation, the UDMA will stop processing the TR at that point and will return an appropriate error code in the Transfer Response message as specified in the *TI DMA Architecture* specification. The PAUSE_ON_ERR bit in the RCHAN UDMA_RCFG_j registers controls whether or not the DMA will pause operation on that channel if an error or exception occurs. If the PAUSE_ON_ERR bit is set to 1 the channel will be paused so that the host can optionally read the channel state to determine where in the transfer the channel execution was. If the PAUSE_ON_ERR bit is 0 the DMA will flush the current Transfer Request record for channels in pass by value mode and will flush the current Transfer Request Descriptor for channels in pass by reference mode.

Copyright © 2024 Texas Instruments Incorporated

10.2.3.2.5.4 Data Transfer

Third Party mode packet reception is accomplished within the UDMA by unpacking and moving data from the Rx Per Channel FIFOs which were filled via the Rx PSI-L Interface to specified memory mapped address ranges via the write only VBUSM master interface. On the Rx side of the UDMA, these transfers are always writes. Each write transfer which is performed by the Third Party Write Unit is to a destination address and of a size as calculated from parameters specified in the Transfer Request packet which was previously received to control the channel behavior.

10.2.3.2.5.4.1 Memory Interface Transactions

The sequence of control transactions that are performed by the UDMA on the Read/Write VBUSM interface during transmit is to read a Transfer Request record from a logical work queue and to then return a corresponding Transfer Response record to a separate logical work queue when all the transfers in the request have been completed. This sequence repeats for as long as Transfer Request records are provided on the work queue. The exact details of the operations that are involved are covered in the *TI DMA Architecture* specification (see Section 10.1).

The sequences of data transactions that are performed by the UDMA on the Write Only Memory interface during receive can be described as a sequence of piecewise linear write transfers whose starting addresses are each sequentially calculated based on loops and offsets given in the Transfer Request record. A TR can have up to 4 levels of nested transfer sets where each set can have a different element count and stride between elements. TR record formats are described in detail in the Transfer Request format specification.

10.2.3.2.5.4.2 Rx Packet Drop

The Rx PSI-L Interface master may assert the rstrm_drop signal co-incident with the transfer of any data phase of a packet. When this signal is sampled asserted by the UDMA, the drop condition will be latched into a flag for that thread and the Third Party write unit will be notified that the packet is to be flushed and no further writes are to be performed. The drop signal is only required to be asserted with a single data phase and will only cause the current packet to be dropped from packet stream. The UDMA will drop all data given to it until the EOP is found on the thread and then the internal drop flag will be cleared. When the UDMA drops the packet on a channel that is in Third Party mode, a 'dropped by source' error code will be placed in the Transfer Response packet which is returned to the Packet Descriptor or completion queue.

Copyright © 2024 Texas Instruments Incorporated

10.2.4 Ring Accelerator (RINGACC)

This chapter describes the RINGACC module, part of the NAVSS.

10.2.4.1 RINGACC Overview

The Ring Accelerator (RINGACC or RA) provides hardware acceleration to enable straightforward passing of work between a producer and a consumer. There is one RINGACC module per NAVSS.

Table 10-111. RINGACC Allocation Across Device Domains

Instance	Domain		
	WKUP	MCU	MAIN
NAVSS0_RINGACC0	-	-	✓ (NAVSS)
MCU_NAVSS0_RINGACC0	-	✓ (MCU NAVSS)	-

Figure 10-18 shows a top-level overview of the RINGACC module.

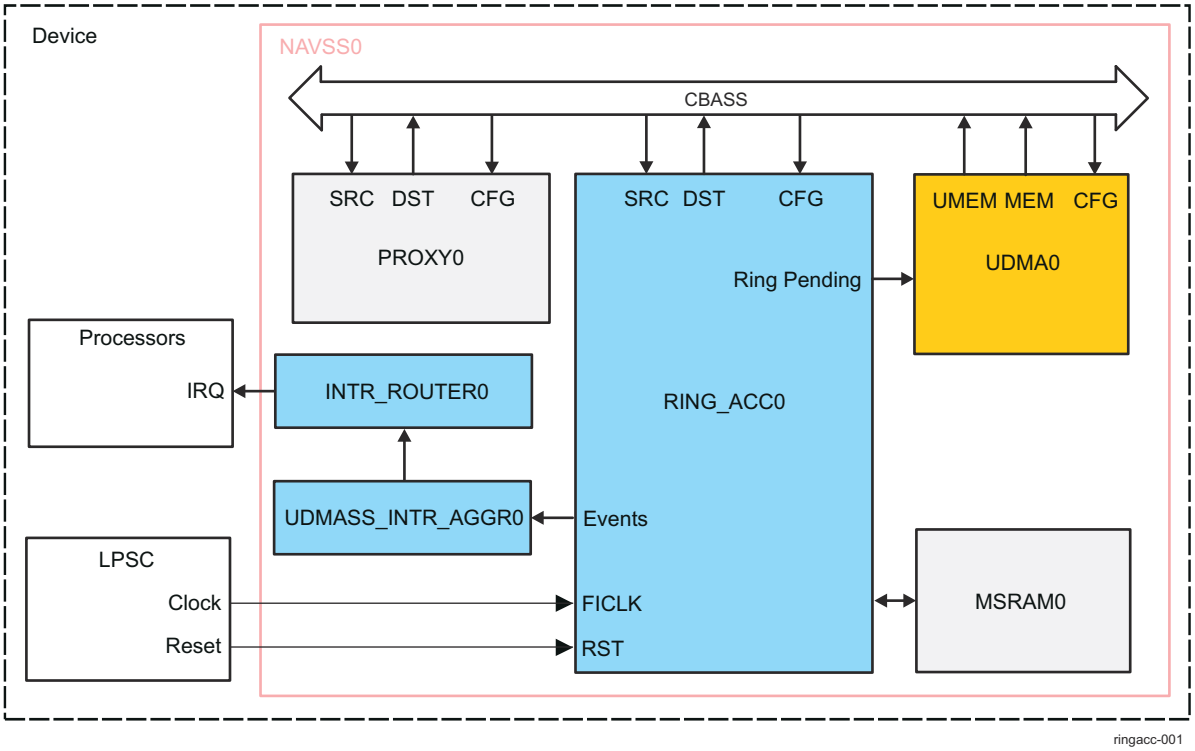


Figure 10-18. RINGACC Overview

10.2.4.1.1 RINGACC Features

NAVSS0_RINGACC0 supports the following features:

- Supports independent memory-mapped ring structures. See ring count in Table 10-112, *RINGACC Configuration Parameters*.
- Supports various modes for each ring based on usage and compatability
- Provides single-word deep shared incoming Transfer Response FIFO
- Provides bit-wide source VBUSM read/write slave interface for accesses from DMA controller entities.
  - Provides 2 word deep command FIFO
  - Provides 2 word deep write data FIFO
  - Provides 2 word deep read data FIFO
  - Provides 2 word deep write status FIFO
- Provides bit-wide destination VBUSM read/write master interface for accesses to ring structures in memory
- Supports up to 16 outstanding writes
- Supports up to 16 outstanding reads

Copyright © 2024 Texas Instruments Incorporated

- Source interface provides an array of 1024 × 512-byte long address windows (four for each ring) which are packed into a single contiguous address range
  - Read and write addresses which target a specific window are translated, in order to redirect the read or write transaction, to an effective address calculated from the base address for the ring plus current ring offset
  - Each read or write access presented on VBUSM slave interface is modified and bridged onto the VBUSM master interface

10.2.4.1.2 RINGACC Parameters

Table 10-112 shows the RINGACC configuration parameters in this SoC.

Table 10-112. RINGACC Configuration Parameters

Module Instance	Parameters		
	Ring Count	Number of Monitors	Proxy Target Base
NAVSS0_RINGACC0	1024	32	0x000038000000
MCU_NAVSS0_UDMASS_RINGACC0	286	32	0x00002B000000

Table 10-113 shows the MSRAM configuration parameters set during SoC design. MSRAM0 is accessible only from the ring accelerator (DST port).

Table 10-113. MSRAM Configuration Parameters

Module Instance	Parameters		
	Depth	Width	Base Address
NAVSS0_MSRAM0	4096	128	0x000030000000
MCU_NAVSS0_UDMASS_MSRAM0	3594	64	0x000028000000
MCU_NAVSS0_UDMASS_MSRAM1	4096	64	0x000028010000

10.2.4.1.3 Ring Accelerator (RINGACC) Ports

Table 10-114. RINGACC Clocks and Resets

Clocks	
Module Clock Input	**Description**
RINGACC_FICLK	RINGACC clock. This clock is used for all interface and functional operations.
Resets	
Module Reset Input	**Description**
RINGACC_RST	RINGACC hardware reset

Table 10-115. RINGACC Hardware Requests

Interrupt Requests		
Module Interrupt Signal	**Description**	**Type**
-	The module does not generate traditional interrupts	-
Outbound Events		
Module Event	**Description**	**Type**
RING[0:1023]_EVT	Ring events. Can be used by an external host for statistics or interrupts.	ETL Push
MON[0:31]_EVT	Monitor events. Can be used by an external host for statistics or interrupts.	ETL Push
RINGACC_ERROR_EVT	Bus error event for host interrupts	ETL Push

Copyright © 2024 Texas Instruments Incorporated

10.2.4.2 RINGACC Functional Description

The Ring Accelerator (RINGACC) converts constant-address read and write accesses to equivalent read or write accesses to a circular data structure in memory. The RINGACC eliminates the need for each DMA controller which needs to access ring elements from having to know the current state of the ring (base address, current offset). The DMA controller performs a read or write access to a specific address range (which maps to the source interface on the RINGACC) and the RINGACC replaces the address for the transaction with a new address which corresponds to the head or tail element of the ring (head for reads, tail for writes). Since the RINGACC maintains the state, multiple DMA controllers or channels are allowed to coherently share the same rings as applicable. The RINGACC serves a very similar function to the Queue Manager in the earlier SoCs. The RINGACC uses less memory and is able to place data which is destined towards software into cached memory directly.

10.2.4.2.1 Block Diagram

Figure 10-19 shows ring accelerator's main internal components.

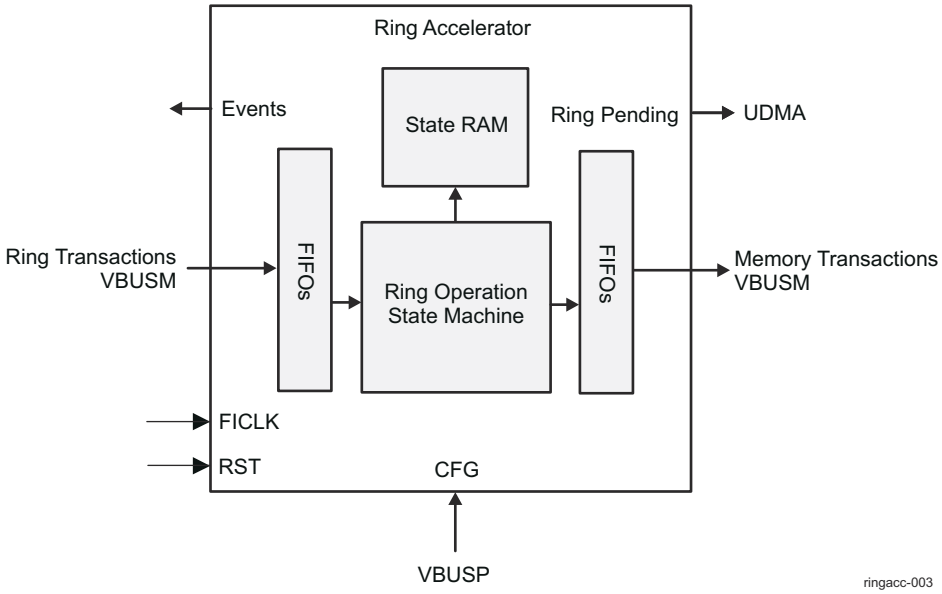


Figure 10-19. Ring Accelerator Block-Diagram

10.2.4.2.1.1 Configuration Registers

The Configuration Registers block is responsible for providing memory mapped registers for configuration of the RINGACC functions. The configuration registers are divided into:

- Static configuration fields (NAVSS0_UDMASS_RINGACC0_CFG Registers), which are typically initialized and then left at specified values for long time periods (or until a reset occurs)
- Real-time (RT) registers (NAVSS0_UDMASS_RINGACC0_CFG_RT Registers), which are modified frequently during runtime.

Configuration registers are listed and described in *RINGACC Register Manual*.

10.2.4.2.1.2 Source Command FIFO

The Source Command FIFO buffers VBUSM command phases which are accepted from the VBUSM source interface.

10.2.4.2.1.3 Source Write Data FIFO

The Source Write Data FIFO buffers VBUSM write data phases which are accepted from the VBUSM source interface.

Copyright © 2024 Texas Instruments Incorporated

10.2.4.2.1.4 Source Read Data FIFO

The Source Read Data FIFO buffers VBUSM read data phases which have been returned from the VBUSM destination interface (via the main state machine) and which are to be output on the VBUSM source interface.

10.2.4.2.1.5 Source Write Status FIFO

The Source Write Status FIFO buffers VBUSM write status data phases which have been returned from the VBUSM destination interface and which are to be output on the VBUSM source interface.

10.2.4.2.1.6 Main State Machine

The Main State Machine (MSM) block implements all of the control logic which is necessary to accept a command from the source interface, perform a lookup into the ring state RAM, determine the effective address for the destination interface transaction, modify the address (and byte count for reads) for the transaction, and place that modified transaction into the outgoing destination FIFOs. The MSM block is also responsible for updating the ring index and occupancy values and for producing output status to inidicate changes to the ring state. The MSM also modifies the ring state whenever the read data or write status for a previous ring transaction return and forwards those responses back to the originating master.

10.2.4.2.1.7 Destination Command FIFO

The Destination Command FIFO stores VBUSM transaction commands which are output from the MSM as modified versions of transactions that were popped fromt the Source Command FIFO. The Destination command FIFO allocates a new command ID for each read or write command that is pushed to the FIFO. Read IDs are allocated from a scoreboard maintained in the Destination Read Data FIFO. Write IDs are allocated from a scoreboard maintained in the Destination Write Status FIFO. When a new command is allocated, the Destination Command FIFO also pushes the original *cid* and *crouteid* values from the source side transaction into a scoreboard in either the Destination Read Data or Destination Write Status FIFOs depending on the applicable transaction direction. The read interface of the Destination Command FIFO directly drives commands onto the destination VBUSM command interface.

10.2.4.2.1.8 Destination Write Data FIFO

The Destination Write Data FIFO stores write data phases for write commands that are passing through the RINGACC towards memory. The write interface of the Destination Write Data FIFO is directly connected to the read interface of the Source Write Data FIFO and the read interface directly drives write data onto the destination VBUSM write data interface.

10.2.4.2.1.9 Destination Read Data FIFO

The Destination Read Data FIFO provides a read command translation scoreboard and also stores read data phases that are passing through the RINGACC towards the originating DMA controller. The write interface of the Destination Read Data FIFO is directly connected to the destination VBUSM read data interface. As read data passes through the Destination Read Data FIFO, the *rid* and *rrouteid* are changed back to their original values using information which was stored in read command translation scoreboard.

10.2.4.2.1.10 Destination Write Status FIFO

The Destination Write Status FIFO provides a write command translation scoreboard and also stores write status phases that are passing through the RINGACC towards the originating DMA controller. The write interface of the Destination Writes Status FIFO is directly connected to the destination VBUSM write status interface. As write status words pass through the Destination Write Status FIFO, the *sid* and *srouteid* are changed back to their original values using information which was stored in the write command translation scoreboard.

10.2.4.2.2 RINGACC Functional Operation

10.2.4.2.2.1 Queue Modes

Each ring or queue can be in one of four modes that determines how it must be accessed and the compatability it has with hardware and software.

10.2.4.2.2.1.1 Ring Mode

Ring mode is used when software owns one side of the ring, and hardware or another software owns the other side. This mode allows the software that owns a side of the ring to control that side including accessing

Copyright © 2024 Texas Instruments Incorporated

the memory directly rather than going through hardware, and software updates the hardware on any changes through the Doorbell registers. These doorbell accesses indicate when software has pushed or popped entries, allowing the entity on the other side of the ring to know when there are elements ready. The ring mode has limitations that the exposed side of the ring is completely under software control, and any atomicity needed must also be performed by software, which is why the exposed side of a ring is usually owned by a single thread of software. This limitation also means that any hardware configuration that would normally access the same ring for both pushes and pops cannot use ring mode as hardware cannot access the exposed side of the ring, such as free queues that are normally read by hardware but could also be pushed by hardware on an error.

When software is popping from the ring, it must guarantee that it never pops more elements through the Doorbell register (RINGACC_DB_j) than what is valid in the Ring Occupancy (RINGACC_OCC_j) register, which holds how many elements are ready for software consumption. There could be some elements which have data written to memory but have not received status back from memory and those are not considered ready to pop yet as the ring status has not fully updated. Therefore, just reading from memory to determine the number of elements to pop is not sufficient. If software attempts to pop more than the RINGACC_OCC_j register value, then the occupancies could go negative resulting in missed down events and spurious interrupts. An optimization for software is to read the RINGACC_OCC_j register less frequently and keep a local copy which guarantees the maximum number of allowed pops. When the software copy of RINGACC_OCC_j reaches 0, or periodically, the software can read the register to refresh the value. This must reduce the frequency of register reads instead of reading the register for every software pop.

10.2.4.2.2.1.2 Messaging Mode

Messaging mode requires that all accesses to the queue must go through RINGACC so that all accesses to the memory are controlled and ordered. RINGACC then controls the entire state of the queue, and software has no directly control, such as through doorbells and cannot access the storage memory directly. This is particularly useful when more than one software or hardware entity can be the producer and/or consumer at the same time. Software must access the data through bus accesses to tha RINGACC. As with the earlier example, if there are free queues that need the hardware to both push and pop, then they must be configured as at least messaging mode (or a later mode) and not ring mode.

10.2.4.2.2.1.3 Credentials Mode

Credentials mode adds per element credentials storage to messaging mode. This allows multiple producers with their own credentials to have those credentials stored with the element. This allows the consumer to inherit the credentials of each element rather than a single set for the entire queue, such as for DMA TR credential inheritance. This mode requires each operation to use two elements of storage since the credentials are stored in the second element, so the element count must be scaled appropriately. The credentials field is located in the same location as all modes, the word just before the first word of the message.

10.2.4.2.2.1.4 Queue Manager Mode

Queue manager mode makes the queue act like a traditional queue-manager (QM) queue in previous devices.

Note

Queue Manager mode is no longer supported in this device. This section was left for completeness only.

10.2.4.2.2.1.5 Peek Support

All modes except ring mode allow the peek operations so that software can view the next element without actually popping it from the queue. This is done by reading from a different offset from the normal pop operation. All the same fields are read but the element is not actually popped off of the queue. This is useful for algorithms that need to know about the overall queue and head element to make decisions but have not yet decided to pop from the queue.

10.2.4.2.2.1.6 Index Register Operation

In ring mode, the Index (RINGACC_INDX_j) register follows the software control of the ring through the doorbell registers, while the hardware index (RINGACC_HWINDX_j) register follows the hardware access of

Copyright © 2024 Texas Instruments Incorporated

the ring through bus transactions. It works for a ring in either direction as the doorbell register update indicates whether elements were produced or consumed. But in the other queue modes, there is no simple manner to determine which index is for software or hardware since they both use bus transactions which the module cannot differentiate. So for these other queue modes, the index register is always the read index, and the hardware index register is the write index.

10.2.4.2.2.2 VBUSM Slave Ring Operations

Each ring contains four memory spaces on the VBUSM slave bus for access, each of which is 512 bytes long, with a total of 4 kB per ring. Each ring space starts immediately after the preceding ring. The allocation per ring is as in Table 10-116.

Table 10-116. Ring Memory Partition

Offset	Operation	Supported Ring Modes
0x0 - 0x1FF	Reads pop from head. Writes push to head.	Read supported in all modes, write supported in all modes except Ring mode
0x200 - 0x3FF	Writes push to tail. Reads pop from tail.	Write supported in all modes, read supported in all modes except Ring mode
0x400 - 0x5FF	Reads peek from head. Writes ignored.	All modes except for Ring mode.
0x600 - 0x7FF	Reserved	Peeks from tail are not supported in this device
0x800 - 0xFFF	Reserved	Reserved

Within each 512-byte (0x200) space, the message data is right justified so that the last byte is always the 511-th byte. The element size of the ring defines which words are valid. The word just before the first valid message word is the credentials field. Access must not be made before the credentials register .

10.2.4.2.2.3 VBUSM Master Interface Command ID Selection

The RINGACC keeps a transaction command scoreboard for reads and writes. Each scoreboard tracks which command indexes are currently outstanding from the RINGACC and which are free to use. When a read or write is requested by the Main State Machine, the scoreboard corresponding to the specified direction (read/write) is queried starting at index 0 and extending up to the maximum index for that scoreboard. The lowest index that is found and which is not currently allocated is selected and is directly used as the command ID (*cid*) for the outgoing transaction on the destination interface. 15 is the maximum index value for each scoreboard. Entries are freed for re-use in a write scoreboard when write status responses are received which account for all of the outstanding write data. Entries are freed for re-use in the read scoreboard only when all data has been returned for the read.

10.2.4.2.2.4 Ring Push Operation (VBUSM Write to Source Interface)

The following sequence will occur for each VBUSM write which is sent to the source interface:

1. RINGACC extracts ring number from incoming write transaction address
2. RINGACC looks up ring state using ring number
3. RINGACC calculates effective write address using *ring base + ring_index*
4. RINGACC re-evaluates pending bit for ring
5. RINGACC increments hardware index and hardware occupancy for ring
6. RINGACC allocates *cid* from scoreboard and places original *crouteid*, *cid*, and *ring number* into scoreboard
7. RINGACC pushes altered *caddress*, *cid* and unaltered remainder of command attributes to output FIFO (note *routeid* is not included)

At a later time when write status returns:

1. RINGACC recovers original *cid*, *crouteid*, and *ring number* from scoreboard
2. RINGACC increments software index and occupancy for ring
3. RINGACC pushes restored *srouteid*, *sid*, and unaltered write status to output write status FIFO

10.2.4.2.2.5 Ring Pop Operation (VBUSM Read from Source Interface)

The following sequence will occur for each VBUSM read which is sent to the source interface:

Copyright © 2024 Texas Instruments Incorporated

1. RINGACC extracts ring number from incoming read transaction address
2. RINGACC looks up ring state using ring number
3. RINGACC calculates effective read address using ring base + ring_index
4. RINGACC increments hardware index and decrements hardware occupancy for ring
5. RINGACC re-evaluates pending bit for ring
6. RINGACC munges *cid* to add indicator in 4 MSBs to reconstitute *routeid* for returning read data
7. RINGACC pushes altered *caddress*, *cid* and unaltered remainder of command attributes to output FIFO (note *routeid* is not included)

At a later time when read data returns:

1. RINGACC recovers original *cid*, *crouteid*, and ring number from scoreboard
2. RINGACC increments software index and decrements software occupancy for ring
3. RINGACC pushes restored *routeid*, *rid*, and unaltered read data, status, and other control signals to output read FIFO

10.2.4.2.2.6 Host Doorbell Access

The following sequence will occur for each VBUSP write to the doorbell register (RINGACC_DB_j) for a ring:

1. RINGACC extracts ring number from config address
2. RINGACC looks up ring state using ring number
3. RINGACC adds doorbell ring value to both hardware and software occupancies for ring
4. RINGACC re-evaluates pending bit for ring

10.2.4.2.2.7 Queue Push Operation (VBUSM Write to Source Interface)

The following sequence will occur for each VBUSM write which is sent to the source interface:

1. RINGACC extracts ring numberfrom incoming write transaction address
2. RINGACC looks up ring state using ring number
3. RINGACC calculates effective write address using ring base + ring_index
4. RINGACC re-evaluates pending bit for ring
5. RINGACC increments WR index and RD- and WR-occupancy for ring
6. RINGACC allocates *cid* from scoreboard and places original *crouteid*, *cid*, and ring number into scoreboard
7. RINGACC pushes altered *caddress*, *cid* and unaltered remainder of command attributes to output FIFO (note *routeid* is not included)

At a later time when write status returns:

1. RINGACC recovers original *cid*, *crouteid*, and ring number from scoreboard
2. RINGACC pushes restored *srouteid*, *sid*, and unaltered write status to output write status FIFO

10.2.4.2.2.8 Queue Pop Operation (VBUSM Read from Source Interface)

The following sequence will occur for each VBUSM read which is sent to the source interface:

1. RINGACC extracts ring number from incoming read transaction address
2. RINGACC looks up ring state using ring number
3. RINGACC calculates effective read address using *ring base* + *ring_index*
4. RINGACC increments RD index and decrements RD- and WR-occupancy for ring
5. RINGACC re-evaluates pending bit for ring
6. RINGACC munges *cid* to add indicator in 4 MSBs to reconstitute *routeid* for returning read data
7. RINGACC pushes altered *caddress*, *cid* and unaltered remainder of command attributes to output FIFO (note *routeid* is not included)

At a later time when read data returns:

1. RINGACC recovers original *cid*, *crouteid*, and ring number from scoreboard
2. RINGACC pushes restored *rrouteid*, *rid*, and unaltered read data, status, and other control signals to output read FIFO

Copyright © 2024 Texas Instruments Incorporated

10.2.4.2.2.9 Mismatched Element Size Handling

When less data is written than defined by the element size for the ring for RING or MESSAGE modes, only the written data is stored in the memory and any remaining bytes maintain their old values. The index and *occ* are still incremented normally, and any special fields that are unwritten are assumed to be 0 .

Writing less data for CREDENTIALS mode is illegal, as those types need the full data to append the additional words to the correct offsets in memory.

When more data is written than defined by the element size for the ring, it will be an error and the write will be ignored and no index or *occ* increments occur. The only exception is for writes to an 8-byte element size ring, where the additional fields up to another 8 bytes are allowed but not written (for old Queue Manager compatability).

When less data is read than defined by the element size for the ring, only the read data is fetched from memory, and any remaining bytes are lost. The index and *occ* are still incremented normally.

When more data is read than defined by the element size for the ring, it will be an error and the read will not affect the ring, the read data will be 0s, and no index or *occ* decrements occur. Access beyond the credentials word must not be attempted and can result in unpredictable behavior.

All accesses must be for multiples of 32-bit words, and partial bytes are not supported for all ring modes.

10.2.4.2.3 Events

The module outputs events about queue levels that can be used by an external module for statistics or interrupts.

Each queue has

- an up event when the queue goes from 0 elements to 1 or more elements
- and a down event when the queue goes from 1 or more elements to 0 elements.

The event number is programmable per queue in the RINGACC_EVENT_j register. This event occurs when the state machine updates the final occupancy for the queue. In ring mode this is when the response from the memory access returns, while in other queue modes this is when the operation is processed.

An event number programmed to 0xFFFF indicates to not produce an event for that ring when an event is not necessary. The register event number fields reset to 0xFFFF, so they must be programmed to a valid value before events will be generated. This applies to the monitor event number fields as well.

10.2.4.2.4 Bus Error Handling

If a bus error is detected on a response to a ring memory transaction, an error event is triggered to alert software. Since the ring contents may be corrupted as the push or pop was not completed correctly, software should consider resetting the ring and any software or hardware elements using the ring. A log register (RINGACC_ERROR_LOG) captures the ring that had the error and whether it was from a push or pop, and an error up event is sent to the programmed event number. After an error is logged, another will not be captured until the first is read out. When read, if an error is pending an error down event is sent to clear any interrupt, and then the next bus error log can be captured. A read of the RINGACC_ERROR_LOG register when there is no error log pending will not produce any down events. The event that is triggered is also programmable. Only legal push and pop operations will capture this error, and not for any peek, emudbg read, flush command, or any operations that is illegal and causes the resulting memory access to be flushed.

10.2.4.2.5 Monitors

Monitors can be configured that allow various functions to be tracked of programmed queues. Each monitor is programmed to monitor a particular queue, the function to provide, and the data source to operate on. The supported functions are: to check the queue against programmed thresholds, to keep track of watermarks of the queue, to keep track of starvation occurrences (a read of an empty queue), or statistics capture. The supported data sources are: the queue element count, the head element size value, or the accumulated size value. Each monitor can also be programmed to produce an event should it have a triggering condition.

Copyright © 2024 Texas Instruments Incorporated

10.2.4.2.5.1 Threshold Monitor

A threshold monitor uses a programmed low threshold value and a programmed high threshold value to track the data source in the queue and cause an interrupt or status when a threshold is broken, either below the low value or above the high value. This is useful for software to replenish empty buffers when a queue has too few, or for a queue to accumulate enough elements so that software must start processing them, or for debug. The threshold being broken will cause the up or down event.

10.2.4.2.5.2 Watermark Monitor

A watermark monitor tracks the low and high values of the data source since the last read of the monitor. After initial setup or clearing, the watermark values will load the current value of the data source as the low and high. Then for successive operations if the resulting data source of the queue is below the low watermark, or above the high watermark, the associated watermark will be updated. When the monitor value registers are read, the value is cleared and starts tracking again using the current data source value. This can be useful for tracking low and high values of queue element counts or sizes for data tracking, debug or future optimizations, such as allocating items to a queue or buffer sizes based on counts.

10.2.4.2.5.3 Starvation Monitor

A starvation monitor tracks the number of starvation events (a read to an empty queue) that occurred on the queue. The data source is ignored, and the starvation count is always incremented whenever there is a read to the queue and the queue is empty. The count is cleared when the monitor value is read. This can be useful to trigger if any starvation event occurs, or to track how many of these events occur in a timeframe, and can be used for future optimizations or debug.

10.2.4.2.5.4 Statistics Monitor

The statistics monitor can track some simple statistics on the queue operations. The data source is ignored. The first value is the count of the number of writes to the queue, and the second value is the count of the number of reads from the queue. The counts are cleared when the monitor values are read. This can be useful to track the activity on a queue, and can be used for future optimizations or debug.

10.2.4.2.5.5 Overflow

RINGACC provides an overflow function where a push to a full ring will result in a push to a special overflow ring, rather than just losing the pushed data. The overflow ring is defined through the RINGACC_OVRFLOW register, and that ring must be defined with a large enough element size to cover the element size of any other ring, as well as enough elements to hold enough overflow data (as an overflow to the overflow ring will be lost). When there is a push to a full ring, there will be two elements written to the overflow ring, first the push data, and then the ring number that was full. A supervisor entity can own the overflow ring and perform what actions are needed from the overflow data. The size of the overflow ring must be an even number since two elements will be pushed for each overflow event. The overflow ring must be configured in *ring* or *message* modes.

10.2.4.2.5.6 Ring Update Port

To enhance performance for DMAs, the RINGACC provides a bus that indicates when ring updates occur, allowing the DMA to track their own rings and detect updates before the responses arrive. The bus will go valid whenever there is an update to the state of a ring, which can be from a push, pop, or doorbell write. It will not go valid for a peek, a emudbg pop, a push to a full ring, as those do not update the ring state. When the bus is valid, the other signals are valid and indicate whether the operation was a push or a pop, whether the resulting state of the ring is empty, the ring number, and the number of elements updated to the ring (the push determines whether it is subtracted or added to the current count). One special case is a pop from an empty ring, which indicates an update but the element count is 0 since no items were actually popped, but a DMA using the cnt to add or subtract must handle this correctly by default. The bus will go valid once the operation command has been completed, and not after the response has been recevied, even for ring mode rings, allowing the DMA to see the update as soon as it is known by the RINGACC even before it is known to software.

10.2.4.2.5.7 Tracing

This module provides a trace output of all operations so that the traffic can be viewed at a later time. The trace uses a simple write-only 32-bit VBUSP bus, with a defined packet header. The *msgtype* field defines the type of operation, and the message data is simply the data involved in the operation. It is likely the trace output will be

Copyright © 2024 Texas Instruments Incorporated

read slower than new operations could provide data, so there will be a trace buffer able to hold two complete messages plus trace headers. If a new operation starts while the trace buffer does not have enough space, then that trace message will not be sent. And RINGACC_TRACE_CTL register controls whether to enable the trace output, whether to trace a single queue or all queues, and whether to include the message data in the trace output.

Table 10-117 shows the trace message for a push.

Table 10-117. Trace Message for a Push

Word	Bits	Data	Comment
1	31:16	0x0	Time to be replaced by trace hardware
1	15:9	0x1	Msgtype for push
1	8:0	0x009	Standard Trace Header
2	31	push_to_head	Push to head flag, 0 = to tail, 1 = to head
2	30:16	0x0	Unused
2	15:0	queue	Queue for push
3-N	31:0	message data	Optional message data for push, length dependent on message size

Table 10-118 shows the trace message for a pop.

Table 10-118. Trace Message for a Pop

Word	Bits	Data	Comment
1	31:16	0x0	Time to be replaced by trace hardware
1	15:9	0x2	Msgtype for pop
1	8:0	0x009	Standard Trace Header
2	31	empty	empty pop, 0 = valid message, 1 = empty
2	30:16	0x0	unused
2	15:0	queue	queue for pop
3-N	31:0	message data	Optional message Data for pop, length dependent on message size

Table 10-119 shows the trace message for a peek.

Table 10-119. Trace Message for a Peek

Word	Bits	Data	Comment
1	31:16	0x0	Time to be replaced by trace hardware
1	15:9	0x3	Msgtype for peek
1	8:0	0x009	Standard Trace Header
2	31	empty	empty peek, 0 = valid message, 1 = empty
2	30:16	0x0	unused
2	15:0	queue	queue for peek
3-N	31:0	message data	Optional message Data for peek, length dependent on message size

Copyright © 2024 Texas Instruments Incorporated

10.2.5 Proxy

This chapter describes the Proxy module in a NAVSS.

10.2.5.1 Proxy Overview

The Proxy module provides proxy buffers for hosts that need to create large data bursts but can only perform small accesses.

Figure 10-20 shows a top-level overview of the Proxy module.

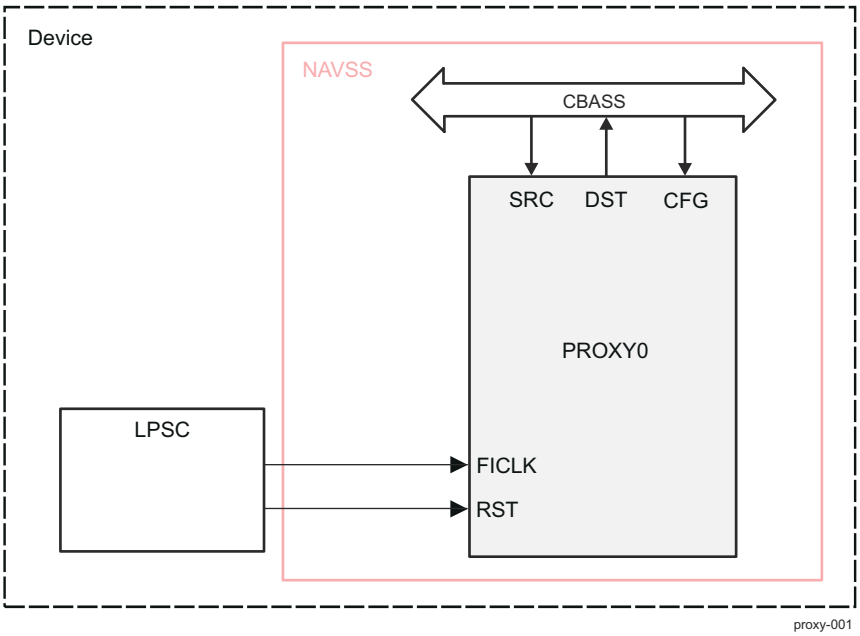


Figure 10-20. Proxy Overview

Table 10-120. Proxy Allocation Across Device Domains

Instance	Domain		
	WKUP	MCU	MAIN
NAVSS0_PROXY0	-	-	✓ (NAVSS)
MCU_NAVSS0_PROXY0	-	✓ (MCU NAVSS)	-

10.2.5.1.1 Proxy Features

Each Proxy supports the following features:

- Provides proxy buffers to store large data bursts that a host can only access in smaller amounts.
- Keeps the large data coherent until the complete data has been accessed.
- Allows for interleaved access between multiple hosts using multiple proxies.
- Each target has pre-configured number of channels, size of each channel, and address offset. See Table 10-121.

10.2.5.1.2 Proxy Parameters

Table 10-121 shows the Proxy configuration parameters set during SoC design.

Table 10-121. Proxy Configuration Parameters

Module Instance	Parameters				
	Proxies (1)	Buffer Size (2)	Target	Channels (3)	Sizes (4)
NAVSS0_PROXY0	64	256	NAVSS0_RINGACC0		4096
NAVSS0_PROXY0	64	128	NAVSS0_RINGACC0	1024	4096

Copyright © 2024 Texas Instruments Incorporated

Table 10-121. Proxy Configuration Parameters (continued)

MCU_NAVSS0_PROXY0	64	64	MCU_NAVSS0_RING ACC0	286	4096

(1) Number of proxy threads supported. Can be read off from the PROXY_CONFIG read-only register
(2) Number of bytes per proxy buffer
(3) Number of channels for the target
(4) Number of bytes per channel supported for the target

10.2.5.1.3 Proxy Ports

Table 10-122. Proxy Clocks and Resets

Clocks	
Module Clock Input	**Description**
PROXY_FICLK	Proxy clock. This clock is used for all interface and functional operations.
Resets	
Module Reset Input	**Description**
PROXY_RST	Proxy hardware reset

Table 10-123. Proxy Hardware Requests

Interrupt Requests		
Module Interrupt Signal	**Description**	**Type**
-	No interrupts to external interrupt controllers	-
DMA Events		
Module DMA Event	**Description**	**Type**
-	No PDMA channels to external DMA engines	-

10.2.5.2 Proxy Functional Description

This module provides proxy buffers for hosts that need to create large data bursts but can only perform small accesses. Examples of this need are for large Message Manager messages, Queue Manager data that is more than a pointer, or DMA Transfer Requests Messages. All of these require larger data sets, from 12 to 128 bytes, and the modules require the entire data on the bus in a single burst. Unfortunately, many CPUs cannot produce bursts to device memory, such as hardware registers, and the largest CPU access is typically 8 bytes. So the proxy provides the ability to access the module with a single burst, and at the same time allows the CPU to access the same data in smaller accesses. The proxy itself does not contain the resources, but it stays in front of other module (target module) that does contain the resources. The proxy will access the target module with the entire data in a single burst, once the host has completed the data.

10.2.5.2.1 Targets

10.2.5.2.1.1 Ring Accelerator

The RINGACC provides four distinct offsets to use per ring/queue shown in Table 10-124, totaling 4 kB space per ring/queue. Each channel has access modes (enabled via PROXY_CTL_j[17-16] MODE field):

- head access
- tail access
- peeks from head

Table 10-124. RINGACC Offsets per Ring/Queue

Offset	Operation
0x0 - 0x1FF	Reads pop from head. Writes push to head.
0x200 - 0x3FF	Writes push to tail. Reads pop from tail.
0x400 - 0x5FF	Reads peek from head. Writes ignored.
0x600 - 0x7FF	Reads peek from tail. Writes ignored. This mode is not supported in this device.
0x800 - 0xFFF	Reserved

Copyright © 2024 Texas Instruments Incorporated

10.2.5.2.2 Proxy Sizes

The number of proxies in the module is configured during SoC design. See Table 10-121. Each proxy has its own address space to the module and determining which proxy is being accessed is simply based on the address bits above the final size of each proxy space. The size of the proxy space is always 4 KB. This is enough to contain the data message, along with controls for which channel, access mode, and element size for the access.

10.2.5.2.3 Proxy Interleaving

Each proxy must have a single owner, as each proxy only has a single buffer to work on the current data. If there are multiple owners in the system, they must each use their own proxy. Then they will have their own buffer and each can access a different resource at the same time. Each proxy is completely independent and kept coherent regardless of accesses to other proxies or regardless of the order of accesses. So the proxies support interleaving of multiple hosts' accesses.

10.2.5.2.4 Proxy Host States

Each individual proxy per host uses a simple state machine to track the currently usage by that host. This state machine is used to track when a proxy is in use or not, but also for error detection. The proxy always stays in idle state, when there is no activity. There is a write state for when the proxy has been written and contains data that has yet to be written to a resource. There is a read state for when the proxy has read data from a resource and that entire data has not been fully read by the host. When a proxy access completes, final write or final read, then the state will go back to idle. The following sections show the utilization of these states.

10.2.5.2.5 Proxy Host Channel Selection

The host must program the PROXY_CTL_j (where j = 0h to 3Fh) prior to accessing the PROXY_DATA_j_y (where j = 0h to 3Fh, y = 0h to 7Fh) unless the previous values are to be reused. This sets the channel within the target, the access mode (head, tail, peek), and the element size. These values will be used when writing the full message to the target, or reading a new message from the target. The target address used will be based on the target base address, the programmed channel, the programmed access mode (for the offset within the 4 kB of the target channel), and the element size (for the offset within the 512 Bytes max size message).

target address = target base + (channel × 4KB) + (mode × 512B) + (512 - element size in bytes)

, for element size in bytes length.

The proxy will detect an error if these fields are changed while in the middle of a message.

10.2.5.2.6 Proxy Host Access

A proxy begins in an idle state, where there is no data in the buffer. In this state the proxy will accept any access to a new resource. If the proxy is in another state, representing a resource is in progress, and the host attempts to access another target or resource, or switches from read to write, then the proxy will detect that as an error and flush the buffer with no final target access and the data is lost and the proxy goes back to idle state.

10.2.5.2.6.1 Proxy Host Writes

The first write to an idle proxy puts the proxy into write mode for the selected target and channel. Further legal writes are stored in the buffer for the enabled bytes and position within the buffer. A write to a previously written location will overwrite the previous data. When there is a legal write to the last byte of the resource, then the entire data is written by the proxy to the target and channel as a single write burst. The proxy will wait for the write status from the target write before it sends the write status for this final byte host write. This allows the host to use the write status to guarantee the entire write data from the proxy has landed at the target. The proxy is then put back into idle state.

10.2.5.2.6.2 Proxy Host Reads

When in idle state, the first read to the proxy will put the proxy into read state, and the proxy will read the target and channel requested in a single burst of the target's channel size. When the data is returned it is stored in the buffer and the accessed part is returned to the host. Further legal reads are allowed and return the accessed data in the buffer. When there is a legal read to the last byte of the resource, the accessed data is returned, and the proxy is put back into idle state.

Copyright © 2024 Texas Instruments Incorporated

10.2.5.2.7 Permission Inheritance

The proxy will automatically inherit the permissions from the access and pass these to the target upon the target burst access. The exact transactions that are inherited from are the final byte write or the first read, and these exact permissions are used on the output bus transaction that accesses the target. This will copy the secure, debug, priv, privid, and virtid signals so that they can be firewalled during the target access for the resource.

10.2.5.2.8 Buffer Size

Buffer size is shown in Table 10-121. An attempt to access a larger message will result in the proxy going into error mode. Software must only create messages up to the supported target size.

10.2.5.2.9 Error Events

Each proxy can produce an event when it has detected an error. This error occurs when the host changes the target, resource, or access direction before completing the previous data. An error can also occur for an access to an illegal target, illegal offset within a target, such as beyond the target size or buffer size, or if the access spans multiple channels. This error does not include accessing reserved register locations within the proxy thread, as that results in normal reserved register behavior of ignoring write data and reading 0s. There is a separate status bit per proxy for these errors. The event number is programmed via PROXY_EVT_REG_j (where j = 0h to 3Fh) for each proxy. The event will not be sent if the event is programmed to 0xFFFF.

10.2.5.2.10 Debug Reads

Since debug reads with emudbg=1 must not cause any side effects or bus errors, any debug read will just read the buffer and never cause a target read. It will also not return any bus errors.

Copyright © 2024 Texas Instruments Incorporated

10.2.6 Secure Proxy

This chapter describes the Secure Proxy module in NAVSS.

10.2.6.1 Secure Proxy Overview

The Secure Proxy module provides proxy buffers for hosts that need to create large data bursts but can only perform small accesses. Secure Proxy is a modified version of the Proxy module in NAVSS.

Table 10-125. Secure Proxy Allocation Across Device Domains

Instance	Domain		
	WKUP	MCU	MAIN
NAVSS0_SEC_PROXY0	-	-	✓ (NAVSS)
MCU_NAVSS0_SEC_PROXY0	-	✓ (MCU NAVSS)	-

10.2.6.1.1 Secure Proxy Features

Secure Proxy supports the following features:

- Provides proxy function to store large data bursts that a host can only access in smaller amounts
- Supports a number of threads, where each has their own independent proxy function
- Keeps the large data burst coherent until the complete data has been accessed
- Allows for interleaved access between multiple hosts or tasks using multiple proxy threads
- Supports a programmable fixed queue for each proxy thread
- Supports multiple producers all writing to the same queue
- Supports a max message count for outbound proxy threads limiting the number of messages a thread can produce
- Supports programmable thresholds for when to generate events.

10.2.6.1.2 Secure Proxy Parameters

Table 10-126 shows the Secure Proxy configuration parameters set during SoC design.

Table 10-126. Secure Proxy Configuration Parameters

Module Instance	Parameters				
	Proxies (1)	Message Size (2)	Target	Channels (3)	Sizes (4)
NAVSS0_SEC_PROXY0	370	64	NAVSS0_RINGACC0	110	4096
MCU_NAVSS0_SEC_PROXY0	90	64	MCU_NAVSS0_RINGACC0	30	4096

(1) Number of proxy threads supported. Can be read off from the SEC_PROXY_CONFIG read-only register
(2) Number of bytes for message. Can be read off from the SEC_PROXY_CONFIG read-only register
(3) Number of channels for the target
(4) Number of bytes per channel supported for the target

10.2.6.1.3 Secure Proxy Ports

Table 10-127. Proxy Clocks and Resets

Clocks	
Module Clock Input	**Description**
SEC_PROXY_FICLK	Proxy clock. This clock is used for all interface and functional operations.
Resets	
Module Reset Input	**Description**
SEC_PROXY_RST	Proxy hardware reset

Table 10-128. Proxy Hardware Requests

Interrupt Requests		
Module Interrupt Signal	**Description**	**Type**
-	No interrupts to external interrupt controllers	-
DMA Events		

Copyright © 2024 Texas Instruments Incorporated

Table 10-128. Proxy Hardware Requests (continued)

Module DMA Event	Description	Type
-	No PDMA channels to external DMA engines	-

10.2.6.2 Secure Proxy Functional Description

This module provides proxy functions for hosts that need to create large data bursts but can only perform small accesses. Examples of this need are:

- large messages (for power control, security control, or IPC)
- data descriptors that are more than a pointer
- DMA Transfer Requests (TR)

All of these require larger data sets, from 12 to 128 bytes, and the storage hardware requires the entire data on the bus in a single burst. Unfortunately, many CPUs cannot produce bursts to device memory, such as registers, and the largest CPU access is typically 8 bytes. Therefore the proxy provides the ability to access the hardware with a single burst, and at the same time allows the CPU to access the same data in smaller accesses.

Figure 10-21 describes Secure Proxy major building blocks.

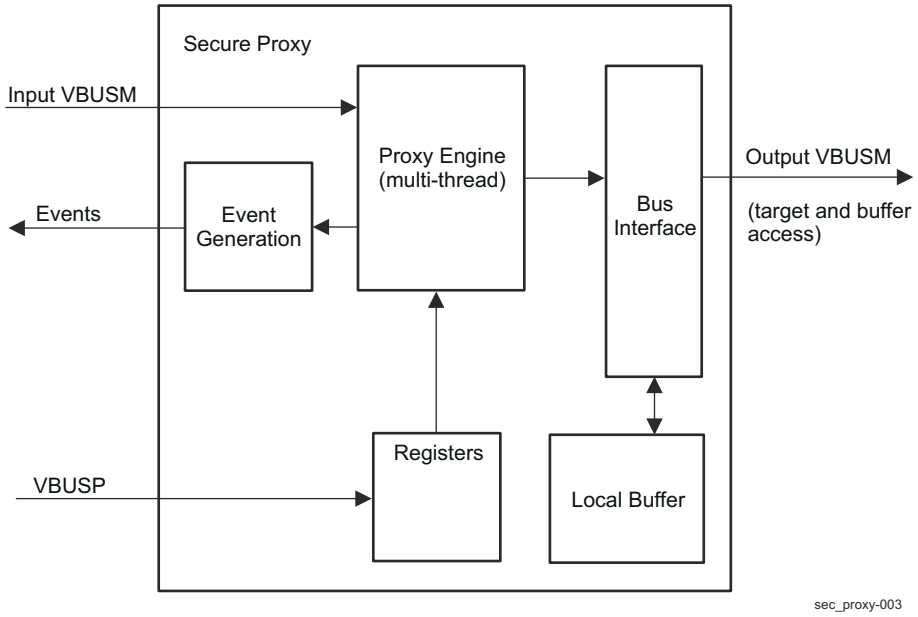


Figure 10-21. Secure Proxy Block Diagram

10.2.6.2.1 Targets

10.2.6.2.1.1 Ring Accelerator

The RING_ACC provides four distinct offsets to use per ring/queue shown in Table 10-129, totaling 4 KB space per ring/queue.

Secure Proxy always writes to tail, which is the offsets from 0x200 to 0x3FF. Secure Proxy always reads from the head, which is the offsets from 0x000 to 0x1FF. The latter two offsets in RING_ACC are not used by this proxy. And if the proxy message size is less than 512 bytes, then it always access the upper bytes of those regions, so that the message always ends on the last byte of each region, byte 511.

Table 10-129. RING_ACC Offsets per Ring/Queue

Offset	Operation
0x0 - 0x1FF	Reads pop from head. Writes push to head.
0x200 - 0x3FF	Writes push to tail. Reads pop from tail.
0x400 - 0x5FF	Reads peek from head. Writes ignored. Not used by Secure Proxy
0x600 - 0x7FF	Reads peek from tail. Writes ignored. Not used by Secure Proxy

Copyright © 2024 Texas Instruments Incorporated

Table 10-129. RING_ACC Offsets per Ring/Queue (continued)

Offset	Operation
0x800 - 0xFFF	Reserved

This means that target writes will use the following calculation:

$$\text{target base address} + (\text{queue} \times 4\text{KB}) + 0\text{x}400 - \text{msg_size} \quad (17)$$

,and burst until offset 0x3FF.

And target reads will use the following calculation:

$$\text{target base address} + (\text{queue} \times 4\text{KB}) + 0\text{x}200 - \text{msg_size} \quad (18)$$

,and burst until offset 0x1FF.

10.2.6.2.2 Buffers

The proxy does not store all the buffers internally. It is programmed with a buffer base address (SEC_PROXY_BUFFER_L, SEC_PROXY_BUFFER_H), and all accesses to the buffer are made through the output VBUSM port using that base. The external buffer must be enough to store one *msg_size* per proxy thread. The proxy only includes a small temporary buffer to save one entire message when accessing the target. Otherwise all host accesses are made to the external buffer. This buffer storage must be considered private for the proxy hardware, so any other access can cause unpredictable results.

10.2.6.2.2.1 Proxy Credits

To support outbound proxy limits, the proxy will use internal credits for tracking resource usage. SEC_PROXY_CTL_j register sets the max credits each outbound proxy thread contains. When a proxy thread sends a message, the current credit count (SEC_PROXY_STATUS_j) decrements by 1. If the current credit count is 0, then the proxy thread is not allowed to send any more messages, and they just remain inside the proxy (so that the host can come back later when a credit has become available and quickly send the same message). When a message that was send by this proxy thread is read by an inbound proxy thread then the credit is returned to this proxy thread and the current credit count increments by 1.

For inbound proxy threads, the credits are used to track pending messages. When an outbound proxy thread sends a message that is read by this proxy thread, then the current credit count of this proxy thread is incremented by 1. A current credit count that is non zero indicates a message is pending and can be read. When the proxy thread completes the read of a message the current credit count is decremented by 1. If the current credit count is zero, then the proxy thread will read an empty message where all the data is 0s. The credit count saturates at 255, so if there can be more messages in the system for that proxy thread than 255 then multiple proxy threads should be used, or the count can become mismatched after a saturate.

10.2.6.2.2.2 Proxy Private Word

To support tracking of credits from each proxy thread, the proxy will extract the first word from the total message size for private tracking usage. This will include saving the proxy thread ID used within the proxy. This is necessary so that when a proxy reads a new message from the target it can inspect this word and return the credit to the source proxy thread. This allows the outbound proxy threads to have a set number of message credits, which decrement when a new message is written, and increment when a message from that proxy thread is read, thus keeping the credits in the system coherent. Any data written to this first word of the message will be ignored, so software should not write to the first word of messages through the proxy. Consumer software can use the first word to identify the source proxy thread of the message (SEC_PROXY_DATA_j).

10.2.6.2.2.3 Completion Byte

The proxy completes a message when the completion byte is accessed. This completion byte is the final byte in the full size of the message. This byte can be written by any size access, whether byte, 32-bit, or 64-bit word. The write of a message is not complete until this final byte of the message is written, and only then will the message be sent to the target. The read of a message is not complete until this final byte of the message is read

Copyright © 2024 Texas Instruments Incorporated

by the host, and only then will reading a new message be allowed. Even if the final byte of the message is not needed, it must be accessed to complete the message inside the proxy hardware.

10.2.6.2.3 Proxy Thread Sizes

There are a configurable number of proxy threads in the module. Each thread has its own address space to the module and determining which proxy thread is being accessed is simply based on the address bits above the final size of each proxy thread space. Since each proxy thread will be programmed to a single resource, the only space requirement is that for using 4 KB to match typically used MMU page sizes. Only the first N bytes, matching the target channel size, is usable.

10.2.6.2.4 Proxy Thread Interleaving

Each proxy thread must have a single owner, as each proxy thread only has a single buffer to work on the current data. If there are multiple owners in the system, they must each use their own proxy thread. Then they will have their own buffer and each can access a different resource at the same time. Each proxy thread is completely independent and kept coherent regardless of accesses to other proxy threads or regardless of the order of accesses. So the proxy support interleaving of multiple hosts' accesses.

10.2.6.2.5 Proxy States

Each proxy thread uses a simple state machine to track the currently usage by that host. This state machine is used to track when a proxy thread is in use or not, but also for error detection. The proxy thread always starts in idle state, when there is no currently activity. There is a write state for when the proxy thread has been written and contains data that has yet to be written to a resource. There is a read state for when the proxy thread has read data from a resource and that entire data has not been fully read by the host. When a proxy thread access completes, final write or final read, then the state will go back to idle. The following sections show the utilization of these states.

10.2.6.2.6 Proxy Host Access

A proxy thread begins in an idle state, where there is no data in the buffer. In this state the proxy thread will accept an access to the resource.

10.2.6.2.6.1 Proxy Host Writes

The first write to an idle proxy thread puts the proxy thread into write mode for the selected channel. Further legal writes are stored in the buffer for the enabled bytes and offset within the buffer. A write to a previously written location will overwrite the previous data. The buffer is not cleared initially, so if a byte is not written, the message will include previous buffer data. When there is a legal write to the last byte of the resource, then the entire data is written by the proxy to the target and channel as a single write burst. The proxy will wait for the write status from the target write before it sends the write status for this final byte host write. This allows the host to use the write status to guarantee the entire write data from the proxy has landed at the target. The proxy thread is then put back into idle state. A write completion when there are no credits available results in the message not being written to the resource and the proxy thread remains in write mode (so that the message can be quickly completed when a credit becomes available).

10.2.6.2.6.2 Proxy Host Reads

When in idle state, the first read to the proxy thread will put the proxy into read state, and the proxy will read the target and channel requested in a single burst of the target's channel size. When the data is returned it is stored in the buffer and the accessed part is returned to the host. Further legal reads are allowed and return the accessed data in the buffer. When there is a legal read to the last byte of the resource, the accessed data is returned, and the proxy thread is put back into idle state. If an initial read is made and there are no pending messages, the proxy thread will return an empty message, where the data is all 0s.

10.2.6.2.6.3 Buffer Accesses

Each read or write will actually trigger a read or write to the external buffer space allocated for that proxy thread, where the real message data is stored. In addition, when a write message completes, the external buffer will be read for the entire message for that proxy thread so that it can then be written to the target. Similarly, after a read causes a read from the target to get a new message, the entire message is written to the external buffer to store the message to the buffer.

Copyright © 2024 Texas Instruments Incorporated

10.2.6.2.6.4 Target Access

After the write has completed and the external buffer read for the entire message, the entire message is then written to the target as a single burst. Similarly, for a read that needs a new message, the target is read for a full message as a single burst.

10.2.6.2.6.5 Error State

If an error is detected in a proxy thread, the proxy will put that proxy thread into error state, and the current buffer will be lost. A status bit will be set that indicates the proxy thread is in error. All accesses to that proxy thread will be ignored until the status bit is cleared. Once cleared the proxy thread is put back into idle state. Errors are: an access to an offset beyond the length of the message, or a mismatch in the direction of access against what was programmed such as reading an outbound thread or writing an inbound thread.

10.2.6.2.7 Permission Inheritance

The proxy will automatically inherit the permissions from the access and pass these to the target upon the target burst access. The exact transactions that are inherited from are the final byte write or the first read, and these exact permissions are used on the output bus transaction that accesses the target.

10.2.6.2.8 Resource Association

Each proxy thread has a register (SEC_PROXY_CTL_j) to define the resource within the target that the proxy thread is mapped to. The host has no ability to access any other resource in the target. This allows the setup of the proxy to determine the resources that all the hosts use.

10.2.6.2.9 Direction

Each proxy thread has a register (SEC_PROXY_CTL_j) to define the direction for access. It can either be an outbound proxy thread for writing messages to resources. Or it can be an inbound proxy thread for reading messages from resources. The host must match the direction programmed or it will result in a proxy thread error.

10.2.6.2.10 Threshold Events

Each proxy thread can produce events based on the state of the resource it uses. For outbound proxy threads, a threshold can be set so that an event is produced when there are now at least N messages available to write. For inbound proxy threads, a threshold can be set so that an event is produced when there are now at least N messages available to read. These threshold values are programmed in SEC_PROXY_EVT_MAP_j, allowing the host to determine how often it gets events. Programming the count values while the threshold is set to 0 will not trigger events, but if the threshold is not 0 then programming the counts will trigger events based on whether the threshold is reached or not. No event is generated if the event is programmed to 0xFFFF.

10.2.6.2.11 Error Events

Each proxy thread can produce an event when it has detected an error. This error occurs when the host violates the programmed setup of the proxy thread or accesses beyond the message size. There is a separate status bit per proxy thread for these errors. No event is generated if the event is programmed to 0xFFFF.

10.2.6.2.12 Bus Errors and Credits

If a target read for a message returns a bus error, this indicates the read was bad and the data is corrupt and usually 0s. Because the private word in contained in the message, if the private word is corrupted then the source proxy thread is not valid and the credit return will not be correct. Since the data is usually 0, this means the credit would be returned to proxy thread 0 incorrectly most of the time. And the credit will be lost for the actual source proxy thread of the message that was not read correctly. The hardware has no way to correct for this, as the message from the target is corrupt so it does not know the correct owner of the credit. If software detects that a read message is corrupt, it can attempt to correct the situation by resetting the affected proxy threads using that queue, to restore their credit counts back to the default. Software may also want to reset proxy thread 0 since it may be getting extra credits.

10.2.6.2.13 Debug

When the transaction has the *emudbg* set for a read, all side effects will not occur. This means that the read will not cause a target read since that would affect the target, and instead current buffer will be read even if the previous message has completed.

Copyright © 2024 Texas Instruments Incorporated

10.2.7 Interrupt Aggregator (INTR_AGGR)

This chapter describes the INTR_AGGR module, part of NAVSS.

10.2.7.1 INTR_AGGR Overview

The Interrupt Aggregator (INTR_AGGR or INTA) provides a centralized machine which handles the termination of system events to that they can be coherently processed by the host(s) in the system. The main function of the module is to convert between 'global events' (assertion and de-assertion events transmitted on the event-transport-lane (ETL) bus) and 'local events' (level sensitive signals on output, and level or edge sensitive signals on input).

Table 10-130. INTR_AGGR Allocation Across Device Domains

Instance	Domain		
	WKUP	MCU	MAIN
NAVSS0_INTR_AGGR0	-	-	✓ (NAVSS)
NAVSS0_INTR_AGGR1	-	-	✓ (NAVSS)
NAVSS0_UDMASS_INTR_AGGR0	-	-	✓ (NAVSS)
MCU_NAVSS0_UDMASS_INTR_AGGR0	-	✓ (MCU NAVSS)	-

10.2.7.1.1 INTR_AGGR Features

NAVSS0_INTR_AGGR0 supports the following features:

- 64-bit VBUSP slave using 64-bit registers
- Provides a set of *TI Interrupt Architecture* compliant interrupt status and mask registers which are used to pass specific event status to one or more host blocks.
  - Maps a collection of "DMA Messaged Events" which are input through an Event Transport Lane (ETL) into specific bit locations in one or more interrupt cause registers
  - Mapping is performed based on a programmable table which provides a single location for each ordinal input event number (0 through sevt_cnt-1)
    - Table specifies a specific bit in a specific cause register that the event should set or clear depending on the type of event (up vs down)
    - Tracks a single 'event up'/'event down' condition. There is no event counting. (The 'cnt' field of the ETL is ignored)
  - Tracked status interrupts are presented to the user through a standard interrupt register interface
    - Provides separate 'enable set' and 'enable clear' registers
    - Provides both masked and unmasked interrupt status
    - Interrupt status bits can be manually cleared using 'write 1 to clear'
- Provides a set of Global Event Input (GEVI) counters which can count events which were delivered via an ingress Event Transport Lane (ETL)
  - Each counted event provides an register by which the count can be read and an register by which a specified amount can be decremented by the host
  - Each counted event generates a egress Global event on a zero to non-zero and non-zero to zero transition, controllable through a mapping register
- Provides a set of Local Event Input (LEVI) to Global event registers which can be used to convert pulsed discrete interrupt inputs or clock synchronous rising edge events into Global events on an egress ETL
  - Each ingress event index provides a configurable 'pulse' or 'rising edge' bit, plus the configurable index of the generated Global event
- Provides a set of Global Event Input (GEVI) 'Multicast' registers which can take a Global event from an ingress ETL and generate two egress Global events on two egress ETL interfaces.

10.2.7.1.2 INTR_AGGR Parameters

Table 10-131 shows the MODSS Interrupt Aggregators 0 and 1, and UDMASS Interrupt Aggregator 0 parameters set during design time. 4 local (LEVI) interrupts are from the MCRC0 module.

Copyright © 2024 Texas Instruments Incorporated

Table 10-131. Interrupt Aggregators Parameters

Module Instance	Parameters				
	VINTR (1)	SEVI (2)	GEVI (3)	LEVI (4)	MEVI (5)
NAVSS0_INTR_AGGR0	64	1024	0	0	0
NAVSS0_INTR_AGGR1	64	1024	0	0	0
NAVSS0_UDMASS_INTR_AGGR0	256	4608	512	132	512
MCU_NAVSS0_UDMASS_INTR_AGGR0	256	1536	256	12	128

(1) VINTR – Number of Virtual Interrupt outputs. These are connected to the interrupt router inputs. Value can be read from INTA_INTCAP/UDMA_INTA_INTCAP register.
(2) SEVI – Number of Main (Steerable) Events. Value can be read from INTA_INTCAP/UDMA_INTA_INTCAP register.
(3) GEVI – Number of Countable Events. Value can be read from INTA_AUXCAP/UDMA_INTA_AUXCAP register.
(4) LEVI – Number of Local Event inputs. Value can be read from INTA_AUXCAP/UDMA_INTA_AUXCAP register.
(5) MEVI – Number of Multicast Events. Value can be read from INTA_AUXCAP/UDMA_INTA_AUXCAP register.

10.2.7.1.3 Interrupt Aggregator (INTR_AGGR) Ports

Table 10-132. INTR_AGGR Clocks and Resets

Clocks	
Module Clock Input	**Description**
INTR_AGGR_FICLK	INTR_AGGR clock. This clock is used for all interface and functional operations.
Resets	
Module Reset Input	**Description**
INTR_AGGR_RST	INTR_AGGR hardware reset

Table 10-133. INTR_AGGR Hardware Requests

Interrupt Requests		
Module Interrupt Signal	**Description**	**Type**
UDMASS_INTA0_VINTR_PEND[255:0]	Global/local events	Level
MODSS_INTA0_VINTR_PEND[63:0]	Global events from TIMER_MGR0	Level
MODSS_INTA1_VINTR_PEND[63:0]	Global events from TIMER_MGR1	Level
L2G Interrupt Request Inputs		
Module Interrupt Signal	**Description**	**Type**
L2G_EVENT_PEND[7:0]	Interrupts from TIMESYNC_INTRTR0	Level
L2G_EVENT_PEND[15:8]	Interrupts from CMPEVT_INTRTR0	Level
L2G_EVENT_PEND[31:16]	Interrupts from GPIOMUX_INTRTR0	Level
DMA Events		
Module DMA Event	**Description**	**Type**
-	No PDMA channels to external DMA engines	-

Copyright © 2024 Texas Instruments Incorporated

10.2.7.2 INTR_AGGR Functional Description

The Interrupt Aggregator provides a centralized machine which handles the termination of system events to that they can be coherently processed by the host(s) in the system. Both the signaling and content of K3 system events are incompatible with standard interrupt controllers. The INTR_AGGR provides mechanisms which convert the TI proprietary system events to standard level sensitive pending bits which can be used by the Interrupt Router and all downstream interrupt infrastructure. Events can be presented to the INTR_AGGR in two different forms:

- Events may be active high pulses, or clock synchronous rising edges, which are input on separate orthogonal pins. These are called 'Local Events'.
  - Local event signals that are used in pulse counting mode must be sourced by the same clock as the interrupt aggregator.
  - Local event signals that are used in edge counting mode must be sourced by pseudo-synchronous sources (and be a slower clock multiple) to the interrupt aggregator clock, and they must remain high for at least 1 'slow' clock cycle.
- Events may be messages which are input in time division multiplexed sequential order from an Event Transport Lane. These are called 'Global Events'.

In both cases, these events need to be conditioned to transition them from a transient indicator that something occurred to a persistent and reliable indication of the current state of the system. The INTR_AGGR is architected to perform this conversion in an efficient and cost effective manner.

10.2.7.2.1 Submodule Descriptions

10.2.7.2.1.1 Status/Mask Registers

The Status and Mask Registers block is responsible for providing persistent storage for the interrupt status and mask bits and for formatting those in a way that is compliant to the *TI Interrupt Architecture* requirements. These requirements include the ability to set and clear bits orthogonally and to provide a masked version of the status register that corresponds to the supplied bit mask for each register.

10.2.7.2.1.2 Interrupt Mapping Block

The Interrupt Mapping Block accepts ordinally numbered events (0-N) and converts those ordinal event numbers into a status register number and bit number pair that is then used to manipulate the specified bit in the Status Registers block.

10.2.7.2.1.3 Global Event Input (GEVI) Counters

The GEVI Counters block is responsible for accepting an Event Transport Lane events and summing the total message counts for each received event index. The module creates global egress events for zero to non-zero count up' events and non-zero to zero 'count down' events. The egress global event index is configured via GEVI UDMA_INTA_MCMAP_j register, so count status egress events can optionally be fed back into the INT_AGGR, via the ETL switch fabric.

10.2.7.2.1.4 Local Event Input (LEVI) to Global Event Conversion

The Local to Global event block is responsible for accepting an array of input pins and independently counting the number of cycles for which those pins are asserted high, or by counting individual clock synchronous rising edge events. The counting mode is configurable on a 'per pin' basis. The events are converted to egress Global events on an egress ETL. Global events can optionally be fed back into the INT_AGGR, via the ETL switch fabric.

10.2.7.2.1.5 Global Event Multicast

The Global event multicast block takes an ingress global event from an ingress ETL, and maps it into two egress Global events on two egress ETL interfaces. Global events can optionally be fed back into the INT_AGGR, via the ETL switch fabric.

Copyright © 2024 Texas Instruments Incorporated

10.2.7.2.2 General Functionality

10.2.7.2.2.1 Event to Interrupt Bit Steering

Each time an event message is received on the Main Event Transport Lane interface, the interrupt mapping block performs a direct lookup into an SRAM using the event number as the address. The SRAM stores a corresponding raw interrupt status register and bit number within that register which are to be manipulated anytime a message is received indicating something occurred on that specific event. When an *up* event is received, the specified bit in the j Status register will be set. When a *down* event is received, that same bit will be cleared. This block is what allows flexible aggregation of various system events into an array of bits in the interrupt status registers. It is intended that this mapping is essentially static - set up when a resource is allocated and left untouched until the resource is no longer needed. Note that the 'cnt' field of the ETL is ignored and no interrupt counting is performed here. When 'UpDn=1', the interrupt flag bit is set, and when 'UpDn=0', the flag bit is cleared.

10.2.7.2.2.2 Interrupt Status

The event messages which are generated from the event to interrupt bit steering logic are input to the Interrupt Status registers. Each time an event is received, the interrupt status registers machine will assert or de-assert the specified bit in the specified register. The assertion and de-assertion in the interrupt status register is unaffected by the interrupt enable state. When an up event is received, the corresponding bit is set and when a down event is received, the corresponding bit is cleared. Some sources of input events will not include the ability to send a down event. In these cases, the interrupt router provides the ability to clear the status bits through the Interrupt Source Clear register. The host will write a one to the specific bit in the register which is to be cleared and the interrupt status machine will clear the bit internally. It is not intended that the Host directly clear bits which are automatically cleared via down events from the source itself.

10.2.7.2.2.3 Interrupt Masked Status

Interrupt enable bits are used in conjunction with the interrupt status bits to create the interrupt masked status register values. The interrupt masked status register (INTA_STATUSM_j) contains the value of the interrupt status ANDed with the value of the interrupt enable register. Each time a new event message is received from the event to interrupt bit steering logic or the interrupt enable register is modified, the interrupt masked status register is re-evaluated.

10.2.7.2.2.4 Enabling/Disabling Individual Interrupt Source Bits

Separate (INTA_ENABLE_SET_j and INTA_ENABLE_CLEAR_j) registers are provided to allow individual enable bits to be enabled or disabled without the need for a read-modify-write operation. When the INTA_ENABLE_SET_j register is written, all bits within written bytes which are 1 will cause corresponding bits in the internal enable register to be set. When the INTA_ENABLE_CLEAR_j register is written, all bits within written bytes which are 1 will cause corresponding bits in the internal enable register to be cleared.

10.2.7.2.2.5 Interrupt Output Generation

Each interrupt masked status register (INTA_STATUSM_j) is accompanied by a single pending bit which indicates if any enabled interrupt source within the corresponding interrupt status register is currently asserted. These pending bits from the interrupt masked status registers are collectively output from the INTR_AGGR as the VINTR_PEND bus. The interrupt status outputs will always be updated to reflect the current values of the INTA_STATUSM_j register. This is accomplished by triggering output updates on write operations to the internal RAM that holds the current raw status and enable masks.

10.2.7.2.2.6 Global Event Counting

When enabled, the INTR_AGGR will provide a set of counters which will track how many outstanding messages have been observed for each ingress event index from the ingress Counted Global Event Transport Lane (ETL) interface. For each message received where the 'UpDn' flag is set, the corresponding counter will be incremented by value of the 'cnt' field of ingress message. Events where the 'UpDn' flag is cleared are ignored. The counter is made visible to software in the UDMA_INTA_COUNT_j register. When software has read the count, it can acknowledge that count has been seen and processed by writing back an integer value specifying the amount by which the counter should be decremented, and the counter will subtract that value from the current count (which may have updated since it was read). The count will saturate at a value of 0xFFFFFFFF. Writing a count 'ack' value higher than the one read is not supported and will produce non-deterministic results.

Copyright © 2024 Texas Instruments Incorporated

When a count transitions from zero to a non-zero value, a Global Up 'UpDn=1' event is sent out an egress ETL interface. When a count transitions from a non-zero value to zero, a down 'UpDn=0' event will be sent. The index of the Global event is mappable on a 'per-counter' basis, using the UDMA_INTA_MAP_j register. The event may be mapped such that it then re-enters INTR_AGGR's 'Event to Interrupt Bit Steering Block' to generate an interrupt to the host. The event routing is handled by an external event switch fabric. Note that writing a new egress event index via the UDMA_INTA_MAP_j register does not alter the stored event count for the ingress event register index.

10.2.7.2.2.7 Local Event to Global Event Conversion

When enabled, the INTR_AGGR will provide a set of Local to Global event conversion registers. Local events may be discrete clock synchronous pulses or clock synchronous rising edges. The counting mode is configurable on a 'per pin' basis. In pulsed mode, the module will track how many outstanding clock cycle long pulses have been observed on each of the provided levi interface pins. For each cycle in which a LEVI input pending bit is asserted the corresponding counter will be incremented by 1. In 'rising edge' mode, the number of rising edge transitions on the pin is counted.

The module will generate egress Global events on its egress ETL, with a different global index configured for each ingress LEVI pin. The LEVI pins numbers (1-N) are converted to arbitrary global event indices via the global event mapping UDMA_INTA_MAP_j registers. The egress Global events can themselves be mapped to re-enter the INTR_AGGR's Global Event Counting block, so that the counts can then be made visible to software in the GEVI count registers as described in Section 10.2.7.2.1.3. The event routing is handled by an external event switch fabric.

10.2.7.2.2.8 Global Event Multicast

In UDMASS_INTR_AGGR0, the INTR_AGGR will provide a set of Global event multicast (UDMA_INTA_MCMAP_j) registers. These registers allow an ingress Global event on its ingress ETL interface to be mapped into two egress Global events on two separate egress ETL interfaces. A set of registers map the ingress Global event index (1-N) into two arbitrary egress event indices.

Copyright © 2024 Texas Instruments Incorporated

10.2.8 Packet Streaming Interface Link (PSI-L)

This section describes the Packet Streaming Interface Link (PSI-L) for the device.

10.2.8.1 PSI-L Overview

PSI-L is a packet based protocol used to transfer data between several device modules. Each transaction is routed based on thread ID value instead of physical address. All PSI-L interfaces are point to point direct connections to NAVSS0 or MCU_NAVSS0. All switching functions among the PSI-L based interfaces are done inside NAVSS0 and MCU_NAVSS0.

The PSI-L also has an event interface referred to as ETL, which is purely used to transfer event information. Only MCU_NAVSS0 has such event interface. It is connected to WKUP_ and allows MCU_NAVSS0 to send event information to the interrupt aggregator inside WKUP_.

Figure 10-22 shows an overview of all PSI-L connections within the device.

Copyright © 2024 Texas Instruments Incorporated

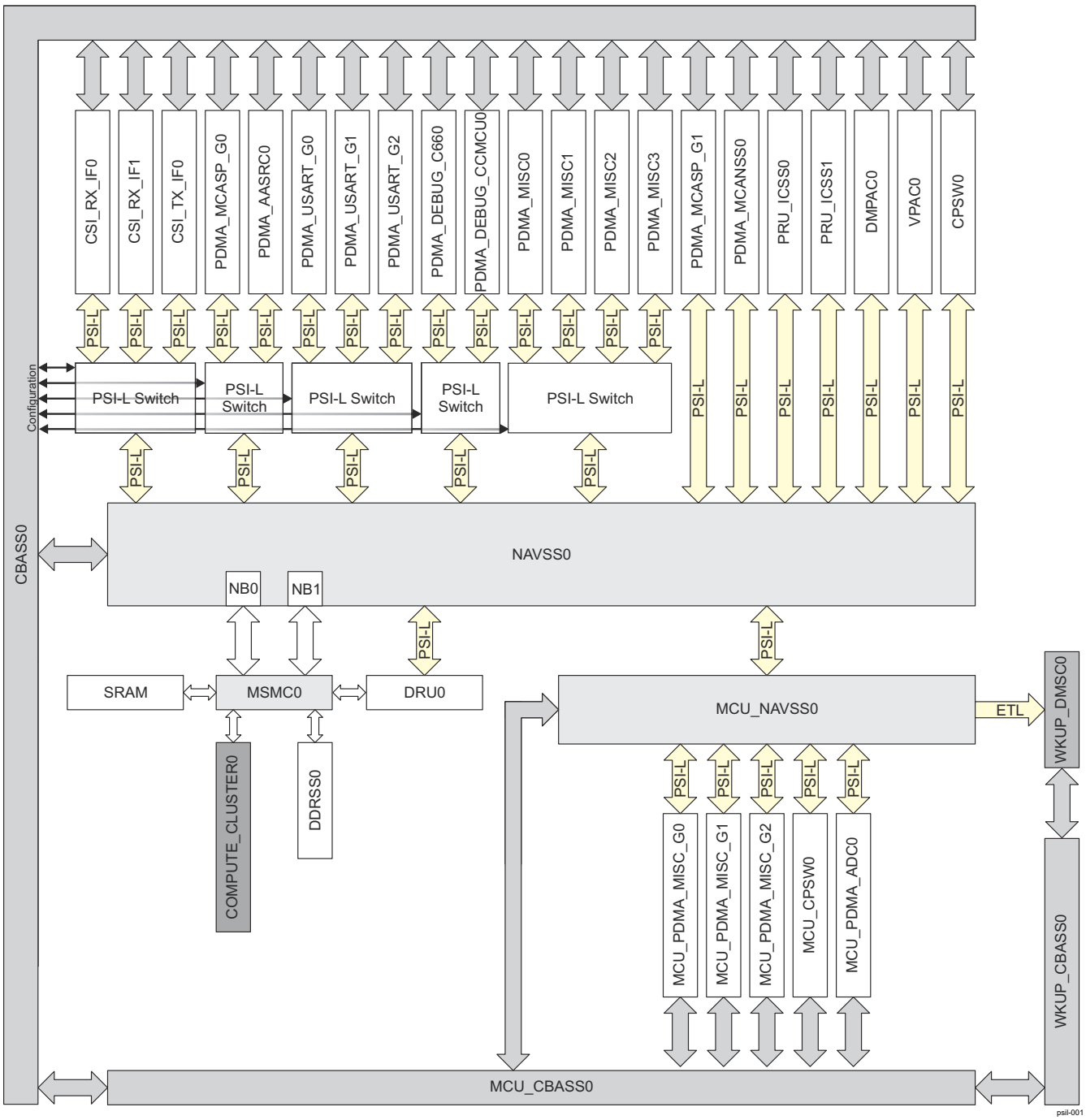


Figure 10-22. PSI-L Overview

Copyright © 2024 Texas Instruments Incorporated

10.2.8.2 PSI-L Functional Description

10.2.8.2.1 PSI-L Introduction

The PSI-L is used to transfer words of packet data and control information between two peer entities via direct connection. There are credits used for flow control to guarantee that blocking does not occur between threads. Multiple packet transfers can be ongoing simultaneously across a PSI-L interface each on a separate logical thread but all sharing a single datapath through time division multiplexing. Each packet which is transferred is accompanied by a destination thread ID which indicates the logical destination thread to which the packet is being sent.

There is low level hardware handshake between PSI-L master and slave used for transfering data. As it is blocking by nature, a higher level protocol is also used in which credits are maintained for each thread connection. Moments of non-readiness are allowed on the interface but there is no possibility that one thread can block another for an indefinite time since a thread must have credit for the destination before it is allowed to use the interface.

Figure 10-23 illustrates the point to point PSI-L connection.

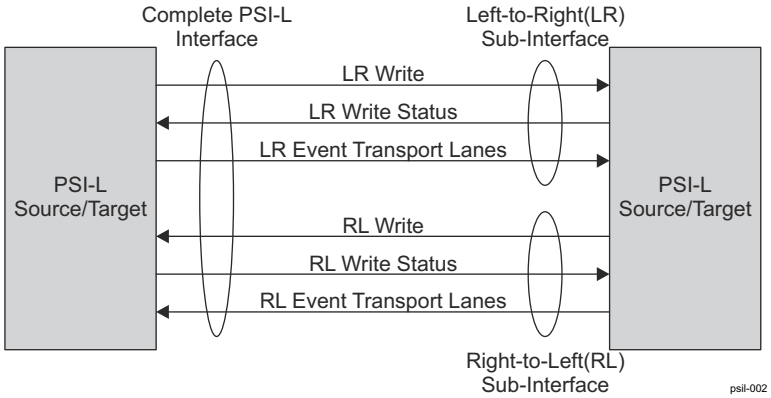


Figure 10-23. PSI-L Connection

PSI-L is intended to be a versatile interface which can carry various types of communications. Table 10-134 describes several types of threads which have been defined.

Table 10-134. Types of PSI-L Threads

Type	Description	Example Endpoints
Queue Management (QM) Messaging	Provides transport for queue push, pop, and divert messages to support distributed queue based work passing	Centralized queue manager or message manager and distributed DMA clients
Direct IO	Allows for tunneling CBASS compatible bus transactions through a PSI interconnect	Module which has no CBASS master to a remote proxy block
DMA Control Transport	Provides transport for DMA transfer descriptions from a source thread to a destination thread	DMA channel controller to DMA transfer controller
DMA Data Transport	Provides transport for packed data from a source thread to a destination thread	Packet oriented module like networking adapters to or from centralized DMA

Each thread on PSI-L is established between a single thread master and a single thread slave using a connection oriented transfer mechanism. A thread master and a thread slave are "paired" to create a logical connection between them referred to as a thread. Pairing is accomplished using a Pair Configuration Message which is valid in all thread types. Pairing can only occur between a thread master and a thread slave of the same type.

10.2.8.2.2 PSI-L Operation

The PSI-L allows multiple independent streams of packet data (threads) to share a single interface using data phase granularity time division multiplexing. This time division multiplexing is accomplished by dividing each stream into single data transfers. In this way a stream is comprised of a sequence of strongly ordered data

Copyright © 2024 Texas Instruments Incorporated

transfers but packet transfers between different streams are allowed to be interleaved. Each data phase is qualified with a type identifier which indicates whether the data is packet info, direct IO write or read info, timestamp, software info, control data, payload data, or status.

10.2.8.2.2.1 Event Transport

PSI-L provides a mechanism by which events can be transported within one or more optional sub-interfaces on a link. These sub-interfaces are referred to as event transport lanes (ETLs) and are independent of any PSI-L packet or message transfers which may be ongoing on the other interfaces within the PSI-L. A maximum of one event can be transferred on each event transport lane in each clock cycle. The event transport protocol is a simple request-ready type of hwardware handshake.

10.2.8.2.2.2 Threads

A thread is a complete flow-controlled stream of communication. The PSI-L thread space is divided into a 32K contiguous region representing all of the source threads (0x0000 - 0x7FFF) and a 32K contiguous region representing all of the destination threads (0x8000 - 0x8FFF).

Source threads are responsible for sending request transactions, accepting response transactions, and sending data transfer transactions. Destination threads are responsible for accepting request transactions, sending response transactions, and accepting data transfer transactions. A given thread generally performs only a single class of transactions (see Table 10-134). Both source and destination threads may act as masters for transfers on one of the PSI-L write sub-interfaces but in this case source threads are actually initiating data transfers while destination threads are only responding to a previous request issued by a source request. Source threads only transfer to destination threads and destination threads only transfer to source threads. Transfers between same thread types do not occur.

The following types of message transfers always originate from source threads and terminate in destination threads:

- Configuration write message
- Configuration read message
- QM push message
- QM pop message
- QM divert message
- Direct read operation message
- Direct write operation message
- DMA transfer request message
- DMA data message

The following types of message transfers always originate in destination threads and terminate in source threads:

- Configuration write response message
- Configuration read response message
- QM push response message
- QM pop response message
- QM divert response message
- DMA transfer response message

10.2.8.2.2.3 Arbitration Protocol

On each cycle, an arbiter built into the master side of each interface decides on which thread to transfer data in the next clock cycle. Both forward (from source threads) and reverse (from destination threads) direction transfers can occur across the same physical interface so the arbiter must ensure that sustained blocking of any thread can occur.

The decision about which thread to allow using the interface is based on the following factors:

1. Whether data is ready at the source to be sent
2. Whether credit exists for the thread such that it is guaranteed there is a place for the data to land in the destination endpoint. For reverse direction transfers (response messages), it is assumed that credit always exists.

Copyright © 2024 Texas Instruments Incorporated

3. The relative priority of the traffic that the thread is carrying for the given packet duration
4. Whether the transfer is for a forward or reverse direction thread (requests versus responses).

No thread can be considered for inclusion in arbitration unless both data is available and credit exists in the destination endpoint. For each thread which can be considered in the current arbitration cycle, the arbiter should then pick the thread with the highest priority. Reverse transfers should be considered higher priority than forward transfers.

10.2.8.2.2.4 Thread Configuration

Each source thread has programmable registers for configuring the pairing with a corresponding destination thread and for enabling data flow. Each destination thread has programmable registers for reporting the buffering capability of the thread and for enabling data flow. These registers are accessed using configuration read and write messages from a configuration host.

Each configuration write message sent from a configuration host is acknowledged with a configuration write response message from the addressed endpoint. Each configuration read message sent from a configuration host is acknowledged with a configuration read response message from the addressed endpoint.

10.2.8.2.2.4.1 Thread Pairing

Depending on the type of traffic that each source thread carries it may or may not have a fixed pairing with a destination thread.

10.2.8.2.2.4.1.1 Configuration Transaction Pairing

Configuration write and read transactions are only initiated by a special module called configuration proxy (CFG_PROXY). Each configuration proxy in the system uses a single source thread which can initiate configuration write and read transactions to the configuration registers of every other source and destination thread in the PSI-L system. The CFG_PROXY registers are described in *PSI-L CFG_PROXY Registers*.

Each source thread routes to a single target thread and this pairing is specified by programming a required pairing register set for each thread. A source has no ability to change what target thread it talks to. If a source must be able to talk to different targets (or target threads), then it can either have multiple threads or a streaming switch that can be programmed to route the transfers appropriately based on specific packet fields.

10.2.8.2.2.4.2 Configuration Registers Region

The configuration registers region contains static configuration information including the settings for linking a thread source to a thread destination. These registers are described in *PSI-L Configuration Registers*. The thread ID mapping is shown in *Navigator Subsystem (NAVSS)*.

Copyright © 2024 Texas Instruments Incorporated

10.2.9 NAVSS North Bridge (NB)

This section describes the North Bridge (NB) in the main Navigator Subsystem.

10.2.9.1 NB Overview

The North bridge bridges between CBA 4.0 compliant VBUSM interfaces and a three-threaded CBA 4.0 compliant VBUSM.C interface.

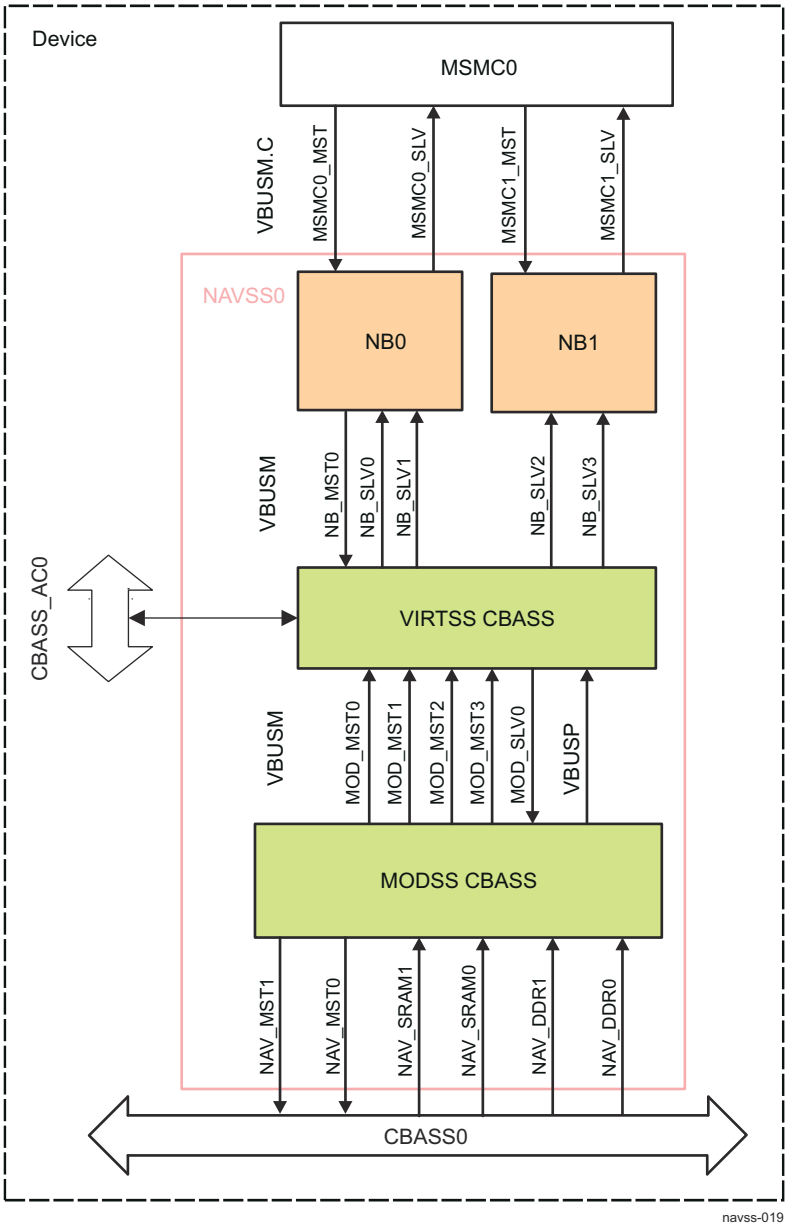


Figure 10-24. NB Overview

10.2.9.1.1 Features Supported

- Implements a CBA 4.0 VBUSM to/from VBUSM.C compliant bridge
- Implements CBA 4.0 VBUSM compliant slave interface(s) to receive traffic from a VBUSM source
- Implements a CBA 4.0 VBUSM compliant master interface to create traffic to a VBUSM destination
- Implements a CBA 4.0 VBUSM.C compliant interface to receive and create traffic to and from a VBUSM.C endpoint using 3 threads
- Can optionally support fragmentation of VBUSM write commands into defined burst aligned sizes

Copyright © 2024 Texas Instruments Incorporated

- Can optionally support fragmentation and reassembly of VBUSM read commands into defined burst aligned sizes
- Can optionally support synchronous or asynchronous clock conversion
- Can optionally support data width conversion
- Can optionally support using the CBA 4.0 VBUSM credit interface for the VBUSM interfaces
- Allows for priority escalation through the bridge, either for higher priority pending commands inside the bridge or requesting into the bridge
- Supports the K3 clkstop idle signal
- Can optionally support a maximum outstanding read byte count to limit the total read data
- Can optionally insert memory attributes for commands
- Can optionally maintain VBUSM order based on orderid
- Allows programming the mapping of the VBUSM source interfaces to VBUSM.C threads

10.2.9.1.2 NB Parameters

North Bridges are implemented in the SoC with the parameters shown in Table 10-135 and Table 10-136.

Table 10-135. NB0 Configuration

Parameter	Value	Description
Number of sources	2	The number of VBUSM source interfaces
Source 0	CBASS MSMC_SLV to MSMC0	Source 0
Source 1		Source 1
source_oidmap[0]	0-7	This defines the orderid mapping to the VBUSM sources. The index represents the source number. The values are integer.
source_oidmap[1]	8-15	
start64k	0x0060000000	This defines the starting address of the memory in VBUSM.C that uses the 64-KB table.
num64k	8192	This defines the number of entries in the 64-KB table.

Table 10-136. NB1 Configuration

Parameter	Value	Description
Number of sources	3	The number of VBUSM source interfaces
Source 0	CBASS MSMC_DDR_0 to MSMC1	Source 0
Source 1	CBASS MSMC_DDR_1 to MSMC1	Source 1
Source 2	CBASS MSMC_DDR_2 to MSMC1	Source 2
source_oidmap[0]	0-4	This defines the orderid mapping to the VBUSM sources. The index represents the source number. The values are integer.
source_oidmap[1]	5-9	
source_oidmap[2]	10-15	
start16m[0]	0x0080000000	This defines the starting address of the memory in VBUSM.C that uses the 16-MB table. The index is for each of the 16-MB regions.
start16m[1]	0x0800000000	
num16m[0]	128	This defines the number of entries in the 16-MB table. The index is for each 16-MB region.
num16m[1]	8192	

10.2.9.1.2.1 Compliance to Standards

The module is compliant to the CBA 4.0 VBUSM specification and VBUSM.C annex.

10.2.9.1.2.2 Features Not Supported

- Does not support fragmentation of VBUSM.C commands
- Does not support VBUSM.C coherence traffic

Copyright © 2024 Texas Instruments Incorporated

10.2.9.2 NB Functional Description

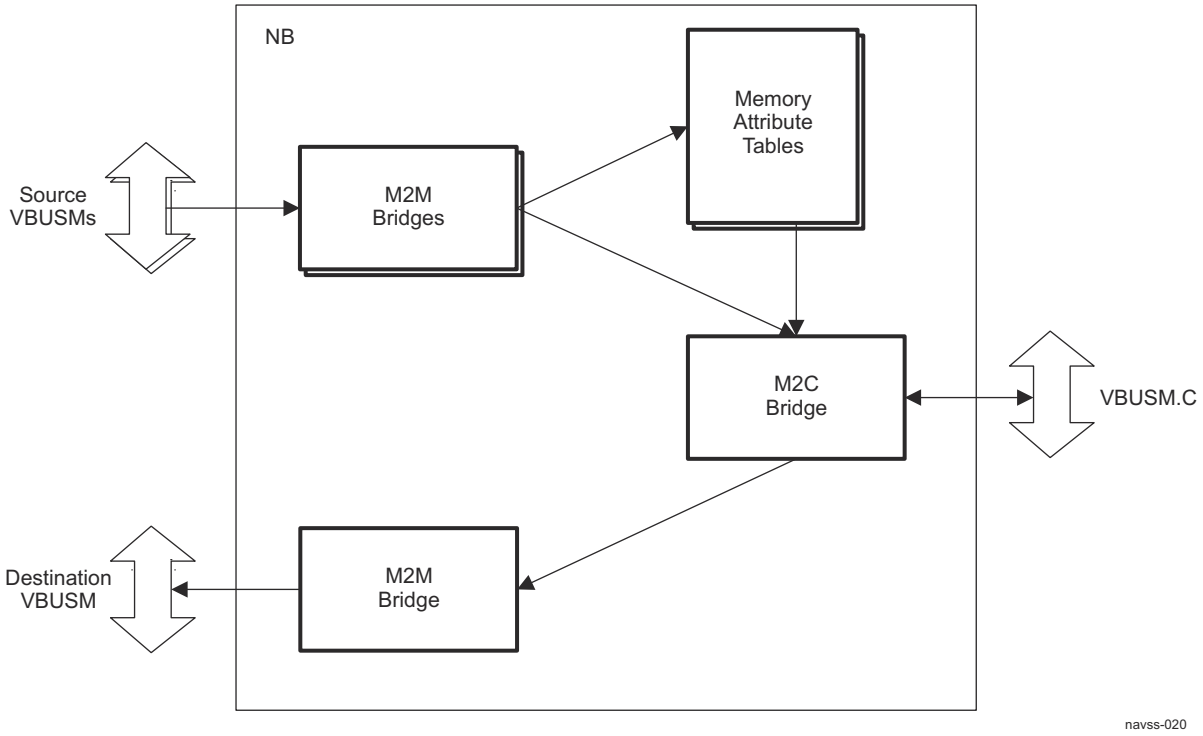


Figure 10-25. NB Block Diagram

The CBA 4.0 North bridge, NB, bridges between CBA 4.0 compliant VBUSM interfaces and a three threaded CBA 4.0 compliant VBUSM.C interface. This is achieved by using M2M bridges to support all the M2M features and then an M2C bridge to convert the VBUSM interfaces to a VBUSM.C interface, which support duplex traffic. The M2M bridges allow for clock frequency conversion, data width conversion and fragmentation for the Source M2M bridge(s). This allows the M2C bridge to remain simple and operate at the VBUSM.C clock frequency and data width, as well as assume all required burst limitations have already been handled. Some additional features needed for the north bridge are implemented, including: adding memory attributes to source VBUSM transactions, and forcing a maximum outstanding read data limit.

10.2.9.2.1 VBUSM Slave Interfaces

The slave interfaces on the bridge is the source VBUSM port to receive commands and return data and status. These ports have the defined prefix, smX (SoC Master, input to bridge), to identify them, where the X is the number to idenfity each port starting from zero. The number of these interfaces and bridges is shown in Table 10-135 and Table 10-136.

10.2.9.2.2 VBUSM Master Interface

The master interface on the bridge is the destination VBUSM port to send commands and receive data and status. These ports have the defined prefix, ss (SoC Slave, output of bridge), to identify them.

It is important to note that any feature not supported in the optional signals for this bus that is supported on the VBUSM.C interface will not be corrected inside the bridge. Examples are excluding the emudbg pin on this bus but expecting VBUSM.C transactions with emudbg set to not create bus status errors on this VBUSM bus outside the bridge (as the bridge will just pass the returned status which could be an error if it does not support the emudbg pin).

10.2.9.2.3 VBUSM.C Interfaces

The pair of interfaces on the bridge is the VBUSM.C port to send and receive commands and send and receive data and status. These ports will have the defined prefix, mm (Memory Master, output of bridge) and ms (Memory Slave, input to bridge), to identify them. These interfaces support three threads for sending and receiving traffic.

Copyright © 2024 Texas Instruments Incorporated

10.2.9.2.3.1 Multi-Threading

The VBUSM.C interfaces use multi-threading to support both input and output traffic on the same data signals. The threads are used to separate that traffic. Thread 0 is used for commands to VBUSM.C, thread 1 is used for commands from VBUSM.C, and thread 2 is used for realtime commands to VBUSM.C. Any responses due to commands must use the same thread when returned.

10.2.9.2.3.2 Write Command Crediting

VBUSM.C interfaces use credits for sending data phases. For writes and read responses to the MSMC (VBUSM.C consumer), there is a requirement that all credits for the write, including the command and all data phases, should be available before sending the command. Since the VBUSM.C data width is always half the max burst size on VBUSM.C, the write command must wait for a TAC credit available and 1 or 2 TDC credits available before the write can be sent.

10.2.9.2.3.3 Early Credit Response

The VBUSM.C interfaces use the credit mechanism available. They do not check for full burst available credits before sending any piece of the burst (command or data). To prevent deadlocks, the connected VBUSM.C device must support sending early credit responses, before the entire burst is received.

10.2.9.2.3.4 Priority Escalation

The VBUSM.C interface does not natively define a mechanism for priority escalation, since the cepriority signal is only valid when a command is sent, and can otherwise be ignored. To support this feature this bridge supports updating the cepriority output even if there is no valid command, so identify the highest priority of any pending commands when the bridge cannot sent a command, such as due to a lack of credits. The VBUSM.C consumer can assume the cepriority output from this bridge is always valid every cycle, and use the cepriority along with the priorities of all pending commands. When there are no pending commands, the cepriority output will be the lowest priority, 7.

10.2.9.2.4 Source M2M Bridges

The source M2M bridges perform all the usual bridging functions from the source VBUSM interface to an internal VBUSM interface running at the same clock frequency and data width of the VBUSM.C interface. It also performs the fragmentation to sizes the VBUSM.C interface requires. It additionally performs full read data reassembly, orderid based ordering, and read data cacheline conversion.

10.2.9.2.5 Destination M2M Bridge

The destination M2M bridge performs all the usual bridging functions to the destination VBUSM interface from an internal VBUSM interface running at the same clock frequency and data width of the VBUSM.C interface.

10.2.9.2.6 M2C Bridge

The M2C bridge performs the protocol conversion and multi-threading of the VBUSM.C interface from the internal VBUSM interfaces. It also handles the cacheline conversion of the write data.

A VBUSM.C write with a dtype of 1 (CPU instruction) will be converted to a VBUSM write with a dtype of 0 (CPU data), as this condition is allowed in VBUSM.C but not in VBUSM.

10.2.9.2.7 Memory Attribute Tables

The bridge can optionally include a memory attribute lookup table to add the coherence memory attributes required for VBUSM.C for VBUSM commands missing them. It is implemented using up to 4 tables, a table of 64-KB regions and 1 to 3 tables of 16-MB regions. The first table is meant to cover an internal memory with smaller granularity, while the second to fourth tables are meant to cover the external memory with a much larger size and larger granularity. The external memory has one to three regions as its address space might be fragmented in the SoC memory map. Starting address and number of table entries for each region are shown in Table 10-135 and Table 10-136.

When a command without memory attributes, an atype = 0, is input to the M2C bridge, this table performs the lookup in parallel and returns the memory attributes to the M2C bridge in the next cycle. If the atype is another value, it is assumed the memory attributes on the VBUSM interface are valid. The lookup is performed by matching the address against the starting addresses, assuming the successive regions always starts at later

Copyright © 2024 Texas Instruments Incorporated

address, including that the up to three 16-MB regions are given in increasing order. Then the offset from the starting address of the region and the region size, 64-KB or 16-MB, is used to calculate the entry of attributes for that address. Then a memory is read to retrieve the attributes and return them to the M2C bridge. An output FIFO is implemented to capture the attributes if they are not accepted immediately.

There is a memory attribute table module for each VBUSM slave interface. If the MAT is enabled, at least one of the 64-KB region and/or the first 16-MB region must be populated, or both. The second and third 16-MB regions are optional.

10.2.9.2.8 Outstanding Read Data Limiter

If a param defines a maximum outstanding read data byte count, then this function will track the requested read bytes against the limit. If a read command does not violate the total read bytes outstanding limit then the read is allowed to the VBUSM.C, otherwise it is stalled. When read data arrives from the VBUSM.C the outstanding count is reduced, allowing more read commands to be sent.

10.2.9.2.9 Ordering

Ordering becomes a concern due to the fact that the VBUSM commands are fragmented and not guaranteed to be returned in order by the VBUSM.C interface. This violates the CBA ordering rules for a single slave endpoint. So the bridge will force read data ordering based on the orderid signal across all masters, as a balance between cost and performance. This means that each read command received from the VBUSM interface with a particular orderid value will have their read data returned in exactly the same order. This means that even if the reads are from different masters, the data will still be returned in order, to reduce the amount of tracking the bridge has to perform. But, if the reads use different orderid values then that read data can be returned in any order, whichever is received first on the VBUSM.C interface.

10.2.9.2.10 Quality of Service

To support quality of service, QoS, the north bridge allows for configuring multiple VBUSM source interfaces. This not only allows defining particular sources to be realtime to achieve better qos, but also to load balance and effectively utilize any extra bandwidth provided by the VBUSM.C interface. There are registers (NB_THREADMAP) allowing programming of which source interfaces map to which of the two VBUSM.C threads, normal or realtime traffic. All traffic on the source will always map to the defined thread.

Any sources mapped to the realtime thread will be arbitrated first, before any sources mapped to the normal thread, to provide the QoS. If there are multiple sources mapped to the same thread then they are arbitrated based on priority and if they are the same priority then by round robin.

To route return traffic from the VBUSM.C back to the correct VBUSM source, the orderid is used. As such, each source mapped to each thread must use a different orderid value in the SoC. This would mean that if there are 8 sources total, with 4 mapped to each thread, then each group of 4 should not share the same orderid for any traffic routed to it in the SoC to keep their pathid values unique. The bridge is configured to identify which VBUSM source uses which orderid value(s).

10.2.9.2.11 IDLE Behavior

The north bridge IDLE output is a combination of all the IDLE outputs from the components. These are combined and registered as the NB IDLE on the Destination VBUSM bus clock.

10.2.9.2.12 Clock Power Management

The north bridge provides internal clock power management for all major components: bridges and register modules. Each component will track its own activity and shut off the clock tree when there has been no activity for 12 cycles. There is a 1 cycle latency for starting up the clock once off, so bus transactions may be initially stalled. The bus early wakeup signals can be used to wakeup the clocks before the transactions arrives to remove some of the latency impact.

Copyright © 2024 Texas Instruments Incorporated

10.3 Peripheral DMA (PDMA)

This chapter describes the PDMA architecture in the device.

10.3.1 PDMA Controller

10.3.1.1 PDMA Overview

The Peripheral DMA is a simple DMA which has been architected to specifically meet the data transfer needs of peripherals, which perform data transfers using memory mapped registers (MMRs) accessed via a standard non-coherent bus fabric. The PDMA module is located close to one or more peripherals which require an external DMA for data movement and is architected to reduce cost by using VBUSP interfaces and supporting only statically configured Transfer Request (TR) operations.

The PDMA is only responsible for performing the data movement transactions which interact with the peripherals themselves. Data which is read from a given peripheral is packed by a PDMA source channel into a PSI-L data stream which is then sent to a remote peer UDMA-P destination channel which then performs the movement of the data into memory. Likewise, a remote UDMA-P source channel fetches data from memory and transfers it to a peer PDMA destination channel over PSI-L which then performs the writes to the peripheral.

The PDMA architecture is intentionally heterogeneous (UDMA-P + PDMA) to right size the data transfer complexity at each point in the system to match the requirements of whatever is being transferred to or from. Peripherals are typically FIFO based and do not require multi-dimensional transfers beyond their FIFO dimensioning requirements, so the PDMA transfer engines are kept simple with only a few dimensions (typically for sample size and FIFO depth), hardcoded address maps, and simple triggering capabilities.

Multiple source and destination channels are provided within the PDMA which allow multiple simultaneous transfer operations to be ongoing. The DMA controller maintains state information for each of the channels and employs round-robin scheduling between channels in order to share the underlying DMA hardware.

The PDMA is linked to the UDMA-P (NAVSS) either through a direct PSI-L connection, or through a PSI-L switch (PSILSS).

10.3.1.1.1 PDMA Features

10.3.1.1.1.1 MCU_PDMA0 (MCU_PDMA_MISC_G0) Features

The MCU_PDMA0 module supports the following features:

- Implements CPPI 5.0 compliant third-party Unified Transfer Controller (UTC)
- Provides 1 memory write access units (write unit 0)
  - Provides a 32-bit wide VBUSP write only master interface for peripheral accesses
  - Supports write bursts up to 64 bytes (on selected channel types)
- Provides 1 memory read access unit (read unit 0):
  - Provides a 32-bit wide VBUSP read-only master interface for peripheral accesses
  - Supports read burst up to 64 bytes (on selected channel types)
  - Supports 1 outstanding read
- Supports up to 7 simultaneous destination (Tx) channels
- Supports up to 7 simultaneous source (Rx) channels
- Supports static transfer requests (TR) only
- Supports X-Y and MCAN transfer modes (does not support AASRC)
- Provides per-channel buffering:
  - Provides 8×128-bit word deep data FIFO for each destination channel
  - Provides 8×128-bit word deep data FIFO for each source channel
- Provides 128-bit wide PSI-L compliant data interface to remote UDMA-P and remote peripherals
- Provides 128-bit wide PSI-L compliant data interface from remote UDMA-P and remote peripherals

10.3.1.1.1.2 MCU_PDMA1 (MCU_PDMA_MISC_G1) Features

The MCU_PDMA1 module supports the following features:

- Implements CPPI 5.0 compliant third-party Unified Transfer Controller (UTC)
- Provides 1 memory write access unit (write unit 0):
  - Provides a 32-bit wide VBUSP write-only master interface for peripheral accesses.

Copyright © 2024 Texas Instruments Incorporated

  - Supports write bursts up to 64 bytes (on selected channel types)
- Provides 1 memory read access unit (read unit 0)
  - Provides a 32-bit wide VBUSP read-only master interface for peripheral accesses
  - Supports read burst up to 64 bytes (on selected channel types)
  - Supports 1 outstanding read
- Supports up to 8 simultaneous destination (Tx) channels
- Supports up to 8 simultaneous source (Rx) channels
- Supports static transfer requests (TR) only
- Supports X-Y transfer mode only (does not support MCAN and AASRC)
- Provides per-channel buffering:
  - Provides 8×128-bit word deep data FIFO for each destination channel
  - Provides 8×128-bit word deep data FIFO for each source channel
- Provides 128-bit wide PSI-L compliant data interface to remote UDMA-P and remote peripherals
- Provides 128-bit wide PSI-L compliant data interface from remote UDMA-P and remote peripherals

10.3.1.1.1.3 MCU_PDMA2 (MCU_PDMA_MISC_G2) Features

The MCU_PDMA2 module supports the following features:

- Implements CPPI 5.0 compliant third-party Unified Transfer Controller (UTC)
- Provides 1 memory write access unit (write unit 0):
  - Provides a 32-bit wide VBUSP write-only master interface for peripheral accesses.
  - Supports write bursts up to 64 bytes (on selected channel types)
- Provides 1 memory read access unit (read unit 0)
  - Provides a 32-bit wide VBUSP read-only master interface for peripheral accesses
  - Supports read burst up to 64 bytes (on selected channel types)
  - Supports 1 outstanding read
- Supports up to 4 simultaneous destination (Tx) channels
- Supports up to 4 simultaneous source (Rx) channels
- Supports static transfer requests (TR) only
- Supports X-Y and MCAN transfer modes (does not support AASRC)
- Provides per-channel buffering:
  - Provides 8×128-bit word deep data FIFO for each destination channel
  - Provides 8×128-bit word deep data FIFO for each source channel
- Provides 128-bit wide PSI-L compliant data interface to remote UDMA-P and remote peripherals
- Provides 128-bit wide PSI-L compliant data interface from remote UDMA-P and remote peripherals

10.3.1.1.1.4 MCU_PDMA3 (MCU_PDMA_ADC) Features

The MCU_PDMA3 module supports the following features:

- Implements CPPI 5.0 compliant third-party Unified Transfer Controller (UTC)
- Provides 0 memory write access units
- Provides 1 memory read access unit (read unit 0)
  - Provides a 32-bit wide VBUSP read-only master interface for peripheral accesses
  - Supports read burst up to 64 bytes (on selected channel types)
  - Supports 1 outstanding read
- Supports up to 0 simultaneous destination (Tx) channels
- Supports up to 4 simultaneous source (Rx) channels
- Supports static transfer requests (TR) only
- Supports X-Y transfer mode only (does not support MCAN and AASRC)
- Provides per-channel buffering:
  - Provides 8×128-bit word deep data FIFO for each destination channel
  - Provides 8×128-bit word deep data FIFO for each source channel
- Provides 128-bit wide PSI-L compliant data interface to remote UDMA-P and remote peripherals
- Provides 128-bit wide PSI-L compliant data interface from remote UDMA-P and remote peripherals

Copyright © 2024 Texas Instruments Incorporated

10.3.1.1.1.5 PDMA5 (PDMA_MCAN) Features

The PDMA5 module supports the following features:

- Implements CPPI 5.0 compliant third-party Unified Transfer Controller (UTC)
- Provides 1 memory write access unit (write unit 0):
  - Provides a 32-bit wide VBUSP write-only master interface for peripheral accesses.
  - Supports write bursts up to 64 bytes (on selected channel types)
- Provides 1 memory read access unit (read unit 0)
  - Provides a 32-bit wide VBUSP read-only master interface for peripheral accesses
  - Supports read burst up to 64 bytes (on selected channel types)
  - Supports 1 outstanding read
- Supports up to 54 simultaneous destination (Tx) channels
- Supports up to 54 simultaneous source (Rx) channels
- Supports static transfer requests (TR) only
- Supports MCAN transfer mode only (does not support X-Y and AASRC)
- Provides per-channel buffering:
  - Provides 8×128-bit word deep data FIFO for each destination channel
  - Provides 8×128-bit word deep data FIFO for each source channel
- Provides 128-bit wide PSI-L compliant data interface to remote UDMA-P and remote peripherals
- Provides 128-bit wide PSI-L compliant data interface from remote UDMA-P and remote peripherals
- Provides ECC support

10.3.1.1.1.6 PDMA6 (PDMA_MCASP_G0) Features

The PDMA6 module supports the following features:

- Implements CPPI 5.0 compliant third-party Unified Transfer Controller (UTC)
- Provides 1 memory write access unit (write unit 0):
  - Provides a 32-bit wide VBUSP write-only master interface for peripheral accesses.
  - Supports write bursts up to 64 bytes (on selected channel types)
- Provides 1 memory read access unit (read unit 0)
  - Provides a 32-bit wide VBUSP read-only master interface for peripheral accesses
  - Supports read burst up to 64 bytes (on selected channel types)
  - Supports 1 outstanding read
- Supports up to simultaneous destination (Tx) channels
- Supports up to simultaneous source (Rx) channels
- Supports static transfer requests (TR) only
- Supports X-Y transfer mode only (does not support MCAN and AASRC)
- Provides per-channel buffering:
  - Provides 8×128-bit word deep data FIFO for each destination channel
  - Provides 8×128-bit word deep data FIFO for each source channel
- Provides 128-bit wide PSI-L compliant data interface to remote UDMA-P and remote peripherals
- Provides 128-bit wide PSI-L compliant data interface from remote UDMA-P and remote peripherals

10.3.1.1.1.7 PDMA9 (PDMA_SPI_G0) Features

The PDMA9 module supports the following features:

- Implements CPPI 5.0 compliant third-party Unified Transfer Controller (UTC)
- Provides 1 memory write access unit (write unit 0):
  - Provides a 32-bit wide VBUSP write-only master interface for peripheral accesses.
  - Supports write bursts up to 64 bytes (on selected channel types)
- Provides 1 memory read access unit (read unit 0)
  - Provides a 32-bit wide VBUSP read-only master interface for peripheral accesses
  - Supports read burst up to 64 bytes (on selected channel types)
  - Supports 1 outstanding read
- Supports up to 16 simultaneous destination (Tx) channels
- Supports up to 16 simultaneous source (Rx) channels

Copyright © 2024 Texas Instruments Incorporated

- Supports static transfer requests (TR) only
- Supports X-Y transfer mode only (does not support MCAN and AASRC)
- Provides per-channel buffering:
  - Provides 8×128-bit word deep data FIFO for each destination channel
  - Provides 8×128-bit word deep data FIFO for each source channel
- Provides 128-bit wide PSI-L compliant data interface to remote UDMA-P and remote peripherals
- Provides 128-bit wide PSI-L compliant data interface from remote UDMA-P and remote peripherals
- Provides ECC support

10.3.1.1.1.8 PDMA10 (PDMA_SPI_G1) Features

The PDMA10 module supports the following features:

- Implements CPPI 5.0 compliant third-party Unified Transfer Controller (UTC)
- Provides 1 memory write access unit (write unit 0):
  - Provides a 32-bit wide VBUSP write-only master interface for peripheral accesses.
  - Supports write bursts up to 64 bytes (on selected channel types)
- Provides 1 memory read access unit (read unit 0)
  - Provides a 32-bit wide VBUSP read-only master interface for peripheral accesses
  - Supports read burst up to 64 bytes (on selected channel types)
  - Supports 1 outstanding read
- Supports up to 16 simultaneous destination (Tx) channels
- Supports up to 16 simultaneous source (Rx) channels
- Supports static transfer requests (TR) only
- Supports X-Y transfer mode only (does not support MCAN and AASRC)
- Provides per-channel buffering:
  - Provides 8×128-bit word deep data FIFO for each destination channel
  - Provides 8×128-bit word deep data FIFO for each source channel
- Provides 128-bit wide PSI-L compliant data interface to remote UDMA-P and remote peripherals
- Provides 128-bit wide PSI-L compliant data interface from remote UDMA-P and remote peripherals
- Provides ECC support

10.3.1.1.1.9 PDMA13 (PDMA_USART_G0) Features

The PDMA13 module supports the following features:

- Implements CPPI 5.0 compliant third-party Unified Transfer Controller (UTC)
- Provides 1 memory write access unit (write unit 0):
  - Provides a 32-bit wide VBUSP write-only master interface for peripheral accesses.
  - Supports write bursts up to 64 bytes (on selected channel types)
- Provides 1 memory read access unit (read unit 0)
  - Provides a 32-bit wide VBUSP read-only master interface for peripheral accesses
  - Supports read burst up to 64 bytes (on selected channel types)
  - Supports 1 outstanding read
- Supports up to 2 simultaneous destination (Tx) channels
- Supports up to 2 simultaneous source (Rx) channels
- Supports static transfer requests (TR) only
- Supports X-Y transfer mode only (does not support MCAN and AASRC)
- Provides per-channel buffering:
  - Provides 8×128-bit word deep data FIFO for each destination channel
  - Provides 8×128-bit word deep data FIFO for each source channel
- Provides 128-bit wide PSI-L compliant data interface to remote UDMA-P and remote peripherals
- Provides 128-bit wide PSI-L compliant data interface from remote UDMA-P and remote peripherals

10.3.1.1.1.10 PDMA14 (PDMA_USART_G1) Features

The PDMA14 module supports the following features:

- Implements CPPI 5.0 compliant third-party Unified Transfer Controller (UTC)
- Provides 1 memory write access unit (write unit 0):

Copyright © 2024 Texas Instruments Incorporated

  - Provides a 32-bit wide VBUSP write-only master interface for peripheral accesses.
  - Supports write bursts up to 64 bytes (on selected channel types)
- Provides 1 memory read access unit (read unit 0)
  - Provides a 32-bit wide VBUSP read-only master interface for peripheral accesses
  - Supports read burst up to 64 bytes (on selected channel types)
  - Supports 1 outstanding read
- Supports up to 2 simultaneous destination (Tx) channels
- Supports up to 2 simultaneous source (Rx) channels
- Supports static transfer requests (TR) only
- Supports X-Y transfer mode only (does not support MCAN and AASRC)
- Provides per-channel buffering:
  - Provides 8×128-bit word deep data FIFO for each destination channel
  - Provides 8×128-bit word deep data FIFO for each source channel
- Provides 128-bit wide PSI-L compliant data interface to remote UDMA-P and remote peripherals
- Provides 128-bit wide PSI-L compliant data interface from remote UDMA-P and remote peripherals

10.3.1.1.1.11 PDMA15 (PDMA_USART_G2) Features

The PDMA15 module supports the following features:

- Implements CPPI 5.0 compliant third-party Unified Transfer Controller (UTC)
- Provides 1 memory write access unit (write unit 0):
  - Provides a 32-bit wide VBUSP write-only master interface for peripheral accesses.
  - Supports write bursts up to 64 bytes (on selected channel types)
- Provides 1 memory read access unit (read unit 0)
  - Provides a 32-bit wide VBUSP read-only master interface for peripheral accesses
  - Supports read burst up to 64 bytes (on selected channel types)
  - Supports 1 outstanding read
- Supports up to 6 simultaneous destination (Tx) channels
- Supports up to 6 simultaneous source (Rx) channels
- Supports static transfer requests (TR) only
- Supports X-Y transfer mode only (does not support MCAN and AASRC)
- Provides per-channel buffering:
  - Provides 8×128-bit word deep data FIFO for each destination channel
  - Provides 8×128-bit word deep data FIFO for each source channel
- Provides 128-bit wide PSI-L compliant data interface to remote UDMA-P and remote peripherals
- Provides 128-bit wide PSI-L compliant data interface from remote UDMA-P and remote peripherals

10.3.1.1.2 Peripheral DMA (PDMA) Ports

Table 10-137. MCU Domain PDMA Clocks and Resets

Clocks	
Module Clock Input	**Description**
MCU_PDMA_FICLK	MCU_PDMA functional and interface clock
Resets	
Module Reset Input	**Description**
MCU_PDMA_RST	MCU_PDMA hardware reset

Table 10-138. MAIN Domain PDMA Clocks and Resets

Clocks	
Module Clock Input	**Description**
PDMA_FICLK	PDMA functional and interface clock
Resets	
Module Reset Input	**Description**
PDMA_RST	PDMA hardware reset

Copyright © 2024 Texas Instruments Incorporated

Table 10-139. MAIN Domain PDMA Hardware Requests

Interrupt Requests		
Module Interrupt Signal	**Description**	**Type**
PDMA5_ECC_SEC_PEND	PDMA5 SEC ECC error interrupt	Level
PDMA5_ECC_DED_PEND	PDMA5 DED ECC error interrupt	Level

Copyright © 2024 Texas Instruments Incorporated

10.3.1.2 PDMA Functional Description

10.3.1.2.1 PDMA Functional Blocks

Figure 10-26 shows the PDMA block diagram.

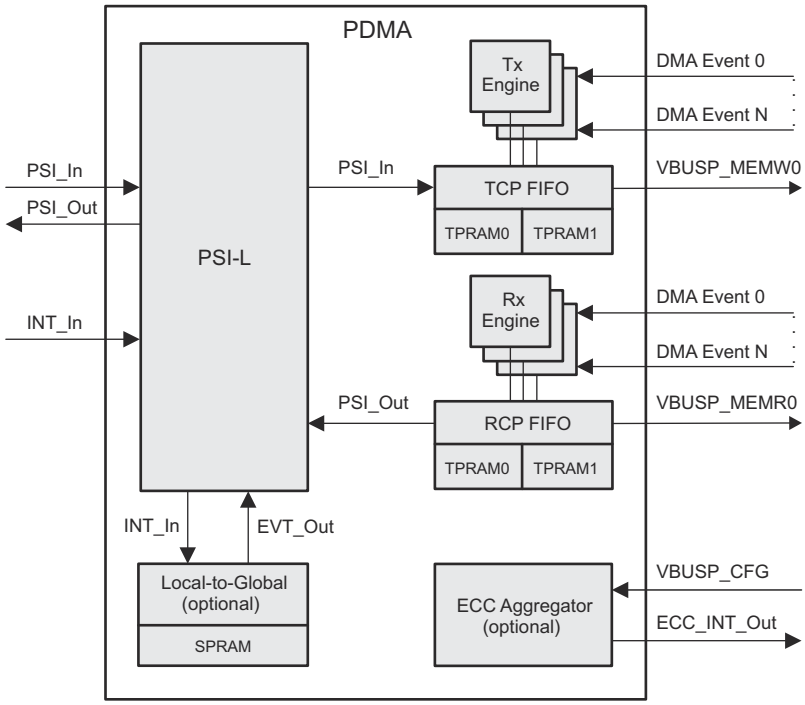


Figure 10-26. PDMA Block Diagram

Note

For detailed description of the PSI-L interface, see *Packet Streaming Interface Link (PSI-L)*.

10.3.1.2.1.1 Scheduler

The scheduler block is responsible for monitoring the fullness level of the various FIFOs, monitoring input DMA triggering events, maintaining channel state information, and issuing credits to the Tx and Rx DMA units when it is time to perform each low-level read or write operation.

10.3.1.2.1.2 Tx Per-Channel Buffers (TCP FIFO)

The Tx per-channel buffers implement channelized FIFOs for each DMA channel. A single logical data FIFO is provided for each destination DMA channel that is used for buffering payload data that has been pushed into the PDMA from the Tx PSI-L interface. The Tx per-channel buffers are always written with full Tx PSI-L wide words (except for EOL/EOP data phases) but are read on byte granularity from the Tx DMA units.

10.3.1.2.1.3 Tx DMA Unit (Tx Engine)

The Tx DMA unit block implements all of the state machine functionality necessary to implement static TR type UTC destination channels. The Tx DMA unit waits until it is triggered by an incoming DMA event and then writes data from the Tx per-channel data FIFO associated with the channel to an external memory mapped target via a VBUSP controller interface. The number and width of the writes that are performed is in accordance with the parameters, which have been programmed via PSI-L into the static TR for the channel.

10.3.1.2.1.4 Rx Per-Channel Buffers (RCP FIFO)

The Rx per-channel buffers implement a logical FIFO for each source DMA channel that is used for buffering payload data that has been fetched by the Rx DMA units. The buffers are byte-oriented on write so that the data from the Rx DMA units, which may not be full words, can be packed properly. The buffers are word-oriented on

Copyright © 2024 Texas Instruments Incorporated

read in accordance with the transport mechanism outlined in the PSI-L interface specification. Each channel in the Rx DMA controller maps directly onto a thread in the Rx PSI-L interface.

The Rx per-channel buffers block outputs queue fullness information to the scheduler block, which it then uses to determine when it should initiate DMA opportunities to backfill the buffers. The Rx per-channel buffers will initiate transfers to the remote paired thread whenever any data is available in each channel buffer and credits are available in the corresponding thread. The block will simultaneously monitor the status of all of the threads and will perform a round-robin arbitration between the different threads for the use of the Rx PSI-L interface. Each thread for which the target is indicating it can accept data and which currently has data available in the channel buffer will be included in the arbitration.

10.3.1.2.1.5 Rx DMA Unit (Rx Engine)

The Rx DMA unit block implements all of the state machine functionality necessary to implement static TR type UTC source channels. The Rx DMA unit waits until it is triggered by an incoming DMA event and then reads data from an external memory mapped source via a VBUSP controller read interface into the Rx per-channel data FIFO associated with the channel. The number and width of the writes that are performed is in accordance with the parameters, which have been programmed via PSI-L into the static TR for the channel.

10.3.1.2.2 PDMA General Functionality

10.3.1.2.2.1 Operational States

At any given time the PDMA can be in one of three different states as follows:

- INIT: This is the initial state of the machine during and immediately after reset. During this state, all of the RAMs inside the PDMA will be initialized to known values including the ECC redundant parity bits. While in the INIT state, the DMA will de-assert all 'ready' signals on all applicable target interfaces and will de-assert all 'request' signals on all applicable controller interfaces. The PDMA will automatically transition out of the INIT state into the IDLE state when all of the RAM initialization has been completed.
- IDLE: Once the PDMA leaves the INIT state, it enters the IDLE state whenever no outstanding transactions are pending on any of the PDMA interfaces (controller or target). The IDLE state is generally a transient state and is used by the PDMA to determine when it is appropriate to allow the SoC power management complex to turn off the clock. As channels have work queued on them and transactions begin flowing into the system, the PDMA transitions to the ACTIVE state. When no more work is pending, or when the host pauses/disables active channels, or when the power management complex on the SoC desires to shut down the PDMA and asserts 'clock stop request', the PDMA will account for any outstanding transactions and will re-enter the IDLE state. The PDMA leaves the IDLE state anytime it generates or receives a transactions that requires a return response as those protocols dictate that the clock must remain running to avoid faulting the handshaking protocol.
- ACTIVE: The PDMA enters the ACTIVE state as soon as it issues a transaction or receives a transaction on any interface that uses a split protocol (expects a later response for a request). When all transactions have been accounted for (responses have all been either received or sent), the PDMA transitions to the IDLE state.

10.3.1.2.2.2 Clock Stop

The clock stop interface is a request/acknowledge interface used to coordinate the handshaking of properly stopping the main clock to the IP. Before attempting to perform a clock stop operation, software is required to teardown all active channels (via 'real time' registers in the BCDMA/PKTDMA, or PSIL register 0x408 in PSIL based peripherals), and after this is complete, also clear the global enable bit for all channels (via PSIL register 0x2 in BCDMA/PKTDMA and PSIL based peripherals). Attempting a clock stop on the IP without first performing the channel teardowns or clearing of global enable bits will result in IP specific behavior that may be UNDEFINED.

The PDMA will not drive clock stop IDLE or clock stop ACK while the global enable (PSIL register 0x2) is set for any channel.

10.3.1.2.2.3 Emulation Control

The emulation control input (EMUSUSP) and the per-channel emulation control FREE bit allow PDMA operation to be suspended on a per-channel basis. When the emulation suspend state is entered, the PDMA will stop

Copyright © 2024 Texas Instruments Incorporated

processing receive and transmit TRs for each channel at the next TR boundary. Any TR currently in reception or transmission will be halted at its next FIFO boundary as configured through the 'X' and 'Y' parameters in the static TR.

Emulation control is implemented for compatibility with other peripherals. Each source and desintaion channel can be configured to either honor the suspend signal, or to 'free run' without regard to suspend. This is conrolled via the PDMA_PSILCFG_TX_RT_ENABLE / PDMA_PSILCFG_RX_RT_ENABLE[0] FREE bit.

10.3.1.2.3 PDMA Events and Flow Control

The following sections describe the simplified mechanism that is provided on the PDMA for triggering and completion event generation.

10.3.1.2.3.1 Channel Types

Each channel in the PDMA is configured at design time (and cannot be changed by software) to be either standard X-Y FIFO mode or MCAN channel or AASRC channel. Either of these two sections provide information about the channel type support in the device: *PDMA Features* and *PDMA Sources*.

When reading the sections below, ignore any information that concerns unsupported channel type.

10.3.1.2.3.1.1 X-Y FIFO Mode

Most peripherals which are serviced by the PDMA follow the pattern of transferring 'X' bytes from a fixed, non-changing address in a burst 'Y' times for each triggering event that is received. The 'X' parameter is typically 1-16 bytes and the 'Y' parameter can be any integer up to 2048.

10.3.1.2.3.1.2 MCAN Mode

The MCAN module is found to require a few minor differences than standard X-Y mode. First, buffer ownership on transmit is different in that the PDMA will start out owning the TX buffer space (instead of waiting for an initial DMA event from the MCAN). There is also a requirement to perform a write of a fixed value to a fixed address after each DMA event is serviced to pass ownership of a buffer back to the MCAN. Finally, there is an MCAN packet fragmentation requirement to place an 8-byte header on each 64-byte chunk that is transmitted and to remove the header from each 64-byte chunk (other than the first block) that is received.

10.3.1.2.3.1.3 AASRC Mode

The AASRC mode of the PDMA is similar to the X-Y mode in that X specifies the sample width, any Y specifies the number of FIFO samples to transfer per DMA request, but in addition to this, a programmable FIFO list is supplied that allows a single PDMA channel to transfer data to multiple FIFOs. The FIFO list specifies which FIFOS to access, and in which order to access them. Both AASRC 'stream' and 'group' modes are supported in terms of DMA signalling and the FIFO memory map. When in AASRC mode, the entire PDMA operates exclusively in this mode. AASRC channels to not share the same PDMA as do XY and MCAN channels.

The following addition details apply to the PDMA in AASRC mode:

- Main features:
  - Up to 16 independent RX channels and 16 TX channels
    - Each channel represents a different data stream to either the paired-DMA or a McASP PDMA
  - Each channel can be configured to read or write any of the AASRC FIFOs in any order
  - Packing support for 1, 2, 3, and 4 byte samples
    - Can directly talk to a McASP using any McASP data format
    - Packing sample size is fixed for a given channel
  - Supports both 'group' and 'stream' AASRC signaling and FIFO memory maps
    - There is a single stream/group mode setting for each PDMA channel
- Additional details:
  - Each channel has a 'mask' of which DMA requests must fire to activate the channel
    - In stream mode, the mask specifies the all the FIFO DMA requests that must fire
    - In group mode, the mask specifies the one group DMA request that must fire
  - Each channel specifies an ordered list of how FIFOs should be accessed

Copyright © 2024 Texas Instruments Incorporated

- All channels use a common list, specifying a range of entry indices within that list
  - This allow each channel to use a varying number of list elements
- The order list can be processed multiple times per DMA request
  - The number of samples to be transferred per request must match the configured threshold for the DMA request in the AASRC
- All FIFOs are specified by index only
  - In the TX ordering table, the indices are assumed to be TX FIFOs
  - In the RX ordering table, the indices are assumed to be RX FIFOs
  - If a DMA channel is in group mode, the PDMA assumes the indices referenced by that channel represent group mode FIFOs; otherwise, it assumes stream mode FIFOs
- The PDMA has configured base addressed and inter-FIFO spans such that is can always convert from a FIFO index to the proper Stream or Group mode FIFO address

10.3.1.2.3.2 Channel Triggering

Channels must be triggered in order for them to perform work. A local event input bus is provided on the PDMA and each bit in the input bus corresponds to the trigger for the channel with the same channel index as the bit index in the bus (bit 0 triggers channel 0, bit 1 triggers channel 1, etc.).

The PDMA provides a 2-bit counter per event input to accommodate startup latency in the channel.

10.3.1.2.3.3 Completion Events

All transfers on PDMA are split TRs in that they have a paired-DMA half and a (static) PDMA half. Completion events are designed to be triggered from the paired-DMA half of the split TR.

10.3.1.2.4 PDMA Transmit Operation

10.3.1.2.4.1 Destination (Tx) Channel Allocation

A total of *N* destination channels are provided within the PDMA for concurrent transfers from Tx per-channel buffers to the various attached peripherals, where *N* is a design-time configurable paramater. Each Tx channel requires a single PSI-L thread. See *PDMA Sources* for Tx channel allocation for each PDMA.

10.3.1.2.4.2 Destination (Tx) Channel Out-of-Band Signals

Table 10-140 shows how PSIL signals are handled for destination channels.

Table 10-140. Tx Channel Out-of-Band Signals

PSI-L Signal	XY/AASRC Mode	MCAN Mode
SOP	Ignored	Ignored
EOP	Ignored	Closes current MCAN packet
SOL	Ignored	Ignored
EOL	Ignored	Ignored
DROP	Ignored	Ignored
PKT_ERROR	Ignored	Ignored
PAUSE	Ignored	Ignored
TDOWN	Reflect in status; teardown when data marker reached	Reflect in status; teardown when data marker reached

10.3.1.2.4.3 Destination Channel Initialization

After reset, all threads/channels in the PDMA will be idle and waiting for work to be assigned to them. In order to initiate PDMA operation, the host will need to first pair the channel with a remote data source (normally a paired-DMA channel), setup the static TR for the channel, and enable the thread.

10.3.1.2.4.3.1 PSI-L Destination Thread Pairing

After reset, all threads on PSI-L are uninitialized and unpaired. The host will pair each destination PDMA channel with (typically) a corresponding source channel in the paired-DMA. The PDMA thread is configured to point to the paired-DMA thread and the paired DMA thread is configured to point to the PDMA thread. Once paired, the source channels in the paired-DMA will send all their data to the destination channel in the PDMA and

Copyright © 2024 Texas Instruments Incorporated

the destination channel in the PDMA will never see any data other than data from the source channel in the paired-DMA.

The following PSI-L registers are related to destination thread pairing:

- PDMA_PSILCFG_TX_ENABLE
- PDMA_PSILCFG_TX_CAPABILITIES
- PDMA_PSILCFG_TX_BYTE_COUNT
- PDMA_PSILCFG_TX_RT_ENABLE

Refer to their descriptions for further details.

10.3.1.2.4.3.2 Static Transfer Request Setup

After reset, all channels in the PDMA will be idle and waiting for work to be assigned to them. In order to initiate PDMA operation, the host will send configuration transactions across PSI-L and will setup the static TR for each destination channel. The format of the TR is identical for each DMA mode, consisting of 'X' and 'Y' parameters, but the field interpretation varies somewhat. The following PSI-L register bit fields are used to setup the static TR:

- PDMA_PSILCFG_TX_STATIC_TR[26-24] X
- PDMA_PSILCFG_TX_STATIC_TR[11-0] Y

Refer to their descriptions for further details.

10.3.1.2.4.3.3 PSI-L Destination Thread Enables

PSI-L destination threads must first be enabled in order to accept data. Threads are enabled or disabled by setting or clearing the ENABLE bit in the PSI-L pairing registers for the thread. When a thread is disabled, it must drop any data phases which are sent but properly return the credits for the data phases which are dropped.

Once a thread is paired, the TR has been set up and the enable is asserted, the channel is armed and ready to be triggered to perform the specified set of write transactions.

10.3.1.2.4.4 Data Transfer

Packet transmission is accomplished within the PDMA by unpacking and moving data from the Tx per-channel FIFOs which were filled via the transmit PSI-L interface to specified memory mapped address ranges via the VBUSP controller interfaces. On the Tx side of the PDMA, these transfers are always writes. Each write transfer which is performed by the Tx DMA unit is to a destination address that is hardcoded in the channel at design time and of a size as specified in the static transfer request.

The sequences of logical transactions that are performed by the PDMA on the memory interface during transmit is dependent on the channel type. The following sections describe what will happen for the two different channel types.

10.3.1.2.4.4.1 X-Y FIFO Mode Channel

The PDMA channel will remain idle until a pulse is detected on the associated input DMA request event pin. Once the pulse is detected, the DMA will sequentially issue a total of 'Y' parameter writes of 'X' parameter bytes to the data FIFO address specified for the channel. Each write that the DMA performs will be a single 'X' element in size (no large bursts). Once the total specified number of transactions has been completed the channel will return to an idle state and wait until it is triggered again. The write transfers that are performed will be accomplished as quickly as possible given availability of data in the Tx channelized FIFO and given the arbitration that may occur as a result of other channels also using the same write unit.

10.3.1.2.4.4.1.1 X-Y FIFO Burst Mode

The burst mode for XY is designed to allow the PDMA to burst across a FIFO region, while extracting out the 'sample size' from each data phase. So for example, it can read a "burst" of multiple 32-bit words from a McASP, and grab only 24-bits from each data phase.

It uses the same registers as standard XY mode, where X is the encoded sample size, and Y is the number of samples to read/write per DMA request. Setting the PDMA_PSILCFG_TX_STATIC_TR[31] BURST bit enables burst.

Copyright © 2024 Texas Instruments Incorporated

The following are the rules for enabling burst mode. Incompatible peripherals do not need to enable burst, or can enable it on RX without enabling it on TX.

- The VBUSP address will increment throughout the burst
- Bursts will be sub-divided to fit into a 64 byte transfer window
- One sample is transferred for every bus data phase
  - Cannot burst one byte samples from a 16-bit peripheral on a 32-bit bus
  - Cannot burst 64-bit samples on a 32-bit bus (configure for 2×32-bit samples instead)
- PDMA will 'gap' the byte enables on writes as needed

10.3.1.2.4.4.2 MCAN Mode Channel

The PDMA channel will remain idle until data is received from the paired-DMA over the PSIL interface. At this time, the PDMA will imediately start copying packet bytes into the MCAN TX buffer corresponding the PDMA/MCAN channel. The number of bytes copied is smaller of the remaining bytes in the packet or the value of the 'Y' parameter. Once a TX buffer has been filled, the PDMA will transfer ownership of the buffer to MCAN by performing an MCAN register write. The PDMA will then wait to regain ownership of the TX buffer by waiting for the corresponding MCAN TX event. At this time, the PDMA will continue with refilling the TX buffer for the next packet fragment. On subsequent fills (until the end of packet is reached), the PDMA will skip over the 8-byte MCAN header, leaving the original contents from the intial fragment copy in place.

10.3.1.2.4.4.2.1 MCAN Burst Mode

Since MCAN buffers are stored in linear memory, the burst mode for MCAN is a simple linear burst across the transfer window. The max burst size is set to the 72 byte size of the MCAN buffer. This will allow a full MCAN packet to be read out as a single burst.

10.3.1.2.4.4.3 AASRC Mode Channel

The AASRC channel is controlled primarily via the PDMA_PSILCFG_TX_AASRC_TX_FIFO_CONFIG register for the channel. This register holds three basic pieces of information:

- Whether the channel uses AASRC stream mode or group mode
- The slot range in the order table used (from designated first slot to designated last slot)
- The AASRC DMA request event(s) that must fire before the channel becomes active

The channel will remain idle until a pulse is detected on ALL associated input DMA request event pins required by the PDMA_PSILCFG_TX_AASRC_TX_FIFO_CONFIG setting. The pulses for the individual events are latched and held by the PDMA until they all arrive. Once the channel activates, it will start reading FIFO index values from TX order table (PDMA_PSILCFG_TX_AASRC_TX_ORDER_TABLE0, PDMA_PSILCFG_TX_AASRC_TX_ORDER_TABLE1), starting at the configured FIRSTSLOT and ending with the LASTSLOT. The actual FIFO indices used are obtained from the ordering table. For example, say FIRSTSLOT=3 and LASTSLOT=5. If the first 6 slots of the ordering table were: 0, 2, 4, 6, 8, 10, the FIFOs written for the event would be 6, 8, and 10, because 'slot 3' of the table contains 6, and it would proceed through 'slot 5' of the table which contains 10.

The X and Y registers are still used as they are in a normal X-Y mode. When accessing each FIFO, the setting of X determines the sample byte width written to the FIFO. The value of 'Y' determines how many times the entire FIFO list is processed for each activation of the channel. Once the total specified number of transactions has been completed the channel will return to an idle state and wait until it is triggered again. The write transfers that are performed will be accomplished as quickly as possible given availability of data in the TX channelized FIFO and given the arbitration that may occur as a result of other channels also using the same write unit.

10.3.1.2.4.5 Tx Pause

The host initiates a channel pause by setting the PDMA_PSILCFG_TX_RT_ENABLE[9] PAUSE bit. The paused channel can be resumed by clearing the register bit.

10.3.1.2.4.6 Tx Teardown

The host initiates a channel teardown by setting the TDOWN bit in the paired-DMA (BCDMA/PKTDMA) channel that is paired with the PDMA. The paired-DMA communicates the teardown state through the PSI-L data channel, to ensure that the teardown is not seen by the PDMA until all the previous paired-

Copyright © 2024 Texas Instruments Incorporated

DMA data for the channel has been flushed. At this time, the teardown state will be reflected in the PDMA_PSILCFG_TX_RT_ENABLE[30] TDOWN bit. Note that a non-synchronized teardown can also be initiated by directly clearing the PDMA_PSILCFG_TX_RT_ENABLE[31] ENABLE.

Once all data has been flushed from the PDMA to the peripheral, the enable state of the PDMA channel will be cleared in the PDMA_PSILCFG_TX_RT_ENABLE register, but the teardown bit will remain high. Upon completion, no further packet processing will occur until the host re-configures the channel. If the channel fails to teardown because the peripheral has stopped responding, or if the paired-DMA transmission stops on a data boundary that is not compatible with the static TR configuration, the PDMA_PSILCFG_TX_RT_ENABLE[28] FLUSH bit can be set to guarantee that all data can be properly flushed from the internal pipe.

10.3.1.2.4.7 Tx Channel Reset

In the unlikely event that channel synchronization is corrupted, a channel may fail to teardown gracefully, even with flush enabled. If this occurs, the channel may be reset by clearing the PDMA_PSILCFG_TX_ENABLE[31] ENABLE bit. This will cause a local reset of the entire channel, including TR and pairing registers. Note that it does not reset the paired-DMA peer. Resetting the paired-DMA peer is also required before re-initializing and re-pairing the channel.

10.3.1.2.4.8 Tx Debug/State Registers

The debug/state registers are supplied to give software applications additional information about the PDMA than they would need in regular operation, but which may be useful in debug situations. These registers appear on the PSI-L bus, near the static TR registers. For transmit, they are defined as follows:

- PDMA_PSILCFG_TX_DEBUG_1
- PDMA_PSILCFG_TX_DEBUG_2

Refer to their descriptions for further details.

10.3.1.2.5 PDMA Receive Operation

10.3.1.2.5.1 Source (Rx) Channel Allocation

A total of *M* destination channels are provided within the PDMA for concurrent transfers from the various attached peripherals into the Rx per-channel buffers and on to the PSI-L Rx Interface, where *M* is a design-time configurable paramater. Each Rx channel requires a single PSI-L thread. See *PDMA Sources* for Rx channel allocation for each PDMA.

10.3.1.2.5.2 Source Channel Initialization

After reset, all threads/channels in the PDMA will be idle and waiting for work to be assigned to them. In order to initiate PDMA operation, the host will need to first pair the channel with a remote data source (normally a paired-DMA channel), setup the static TR for the channel, and enable the thread.

10.3.1.2.5.2.1 PSI-L Source Thread Pairing

After reset, all threads on PSI-L are uninitialized and unpaired. The host will pair each source PDMA channel with (typically) a corresponding destination channel in the paired-DMA. The PDMA thread is configured to point to the paired-DMA thread and the paired-DMA thread is configured to point to the PDMA thread. Once paired, the source channels in the PDMA will send all their data to the source channel in the paired-DMA and the destination channel in the paired-DMA will never see any data other than data from the source channel in the PDMA.

Refer to their descriptions for further details.

10.3.1.2.5.2.2 Static Transfer Request Setup

After reset, all channels in the PDMA will be idle and waiting for work to be assigned to them. In order to initiate PDMA operation, the Host will send configuration transactions across PSI-L and will set up the static TR for each destination channel. The format of the TR is identical for each DMA mode, consisting of 'X', 'Y', and 'Z' parameters, but the field interpretation varies somewhat. The following PSI-L register bit fields are used to setup the static TR:

- PDMA_PSILCFG_RX_STATIC_TR[26-24] X
- PDMA_PSILCFG_RX_STATIC_TR[11-0] Y

Copyright © 2024 Texas Instruments Incorporated

- PDMA_PSILCFG_RX_STATIC_TR_Z[11-0] Z

Refer to their descriptions for further details.

10.3.1.2.5.2.3 PSI-L Source Thread Enables

PSI-L source threads must be enabled in order to process peripheral DMA requests and send data. When disabled, DMA requests are ignored.

Once a thread is paired, the TR has been set up and the enable is asserted, the channel is armed and ready to be triggered to perform the specified set of read transactions.

10.3.1.2.5.3 Data Transfer

Packet reception is accomplished within the PDMA by moving data from structures that are located in memory accessible via the VBUSP Memory Interface(s) onto the receive PSI-L interface. On the Rx side of the PDMA, these transfers are always reads. Data is read from an attached memory mapped space and packed into the Rx per-channel buffer for that channel. At a later time, the data is moved from the Rx per-channel FIFO to a remote peer DMA entity via the Rx PSI-L interface.

The sequences of logical transactions that are performed by the PDMA on the memory interface during receive is dependent on the channel type. The following sections describe what will happen for the two different channel types.

10.3.1.2.5.3.1 X-Y FIFO Mode Channel

The PDMA channel will remain idle until a pulse is detected on the associated input DMA request event pin. Once the pulse is detected, the DMA will sequentially issue a total of 'Y' parameter reads of 'X' parameter bytes from the data FIFO address specified for the channel. Each read that the DMA performs will be a single 'X' element in size (no large bursts). Once the total specified number of transactions has been completed the channel will return to an idle state and wait until it is triggered again. The read transfers that are performed will be accomplished as quickly as possible given availability of data in the Rx channelized FIFO and given the arbitration that may occur as a result of other channels also using the same read unit.

10.3.1.2.5.3.2 MCAN Mode Channel

The PDMA channel will remain idle until a pulse is detected on the associated input DMA request event pin. The DMA event pins are mapped from the CAN filter event bits. The event bits determine the channel and one of two CAN buffers to use for the RX operation. This allows the RX buffers on the CAN to be configured in a ping/pong fashion, preventing the loss of CAN packet data. When an event is received, the PDMA will start copying bytes out of the corresponding CAN buffer. The number of bytes copied is equal to the value of the 'Y' parameter in the channel configuration. Once all data has been copied from the buffer, the PDMA transfers ownership of the buffer back to the CAN by writing a CAN register. It then waits for another RX DMA event. The PDMA can copy multiple CAN RX buffers to the same CPPI packet. It will not close out the CPPI packet until it has copied out a number of RX buffers equal to the 'Z' parameter in the channel configuration. On subsequent RX buffer copies, the MCAN will skip the first 8 bytes of the RX buffer, which is the MCAN packet header.

The mapping of MCAN filter events to MCAN channels and buffer is important for properly configuring the DMA. The mapping is defined as shown in Table 10-141.

Table 10-141. Mapping of MCAN Filter Events to MCAN Channels

CAN FE2	CAN FE1	CAN FE0	Description
0	0	0	Not used
0	0	1	Not used
0	1	0	PDMA CAN channel offset 0, RX buffer 0
0	1	1	PDMA CAN channel offset 0, RX buffer 1
1	0	0	PDMA CAN channel offset 1, RX buffer 2
1	0	1	PDMA CAN channel offset 1, RX buffer 3
1	1	0	PDMA CAN channel offset 2, RX buffer 4
1	1	1	PDMA CAN channel offset 2, RX buffer 5

Copyright © 2024 Texas Instruments Incorporated

By using two filter entries in the CAN, software can setup a ping-pong buffer for a particular RX packet ID. For example, the following two filter entries will setup a ping-pong using RX buffers 0 and 1 for packet ID = 5 on CAN RX channel 0 (Table 10-142).

Table 10-142. Ping-Pong Buffer Example in MCAN Mode

Filter Entry	Packet Match ID	Filter Event Bits
0	5	0x2
1	5	0x3

10.3.1.2.5.3.2.1 MCAN Burst Mode

Since MCAN buffers are stored in linear memory, the burst mode for MCAN is a simple linear burst across the transfer window. The max burst size is set to the 72 byte size of the MCAN buffer. This will allow a full MCAN packet to be read out as a single burst.

10.3.1.2.5.3.3 AASRC Mode Channel

The AASRC channel is controlled primarily via the PDMA_PSILCFG_RX_AASRC_RX_FIFO_CONFIG register for the channel. This register holds three basic pieces of information:

- Whether the channel uses AASRC stream mode or group mode
- The slot range in the order table used (from designated first slot to designated last slot)
- The AASRC DMA request event(s) that must fire before the channel becomes active

The channel will remain idle until a pulse is detected on ALL associated input DMA request event pins required by the PDMA_PSILCFG_RX_AASRC_RX_FIFO_CONFIG setting. The pulses for the individual events are latched and held by the PDMA until they all arrive. Once the channel activates, it will start reading FIFO index values from RX order table (PDMA_PSILCFG_RX_AASRC_RX_ORDER_TABLE0, PDMA_PSILCFG_RX_AASRC_RX_ORDER_TABLE1), starting at the configured FIRSTSLOT and ending with the LASTSLOT. The actual FIFO indices used are obtained from the ordering table. For example, say FIRSTSLOT=3 and LASTSLOT=5. If the first 6 slots of the ordering table were: 0, 2, 4, 6, 8, 10, the FIFOs read for the event would be 6, 8, and 10, because 'slot 3' of the table contains 6, and it would proceed through 'slot 5' of the table which contains 10.

The X and Y registers are still used as they are in a normal X-Y mode. When accessing each FIFO, the setting of X determines the sample byte width read from the FIFO. The value of 'Y' determines how many times the entire FIFO list is processed for each activation of the channel. Once the total specified number of transactions has been completed the channel will return to an idle state and wait until it is triggered again. The read transfers that are performed will be accomplished as quickly as possible given availability of data in the TX channelized FIFO and given the arbitration that may occur as a result of other channels also using the same write unit.

10.3.1.2.5.4 Rx Pause

The host initiates a channel pause by setting the PDMA_PSILCFG_RX_RT_ENABLE[29] PAUSE bit. The paused channel can be resumed by clearing the register bit.

10.3.1.2.5.5 Rx Teardown

The host initiates a RX channel teardown by setting the PDMA_PSILCFG_RX_RT_ENABLE[30] TDOWN bit for the target RX channel. The PDMA communicates the teardown state to the paired-DMA through the PSI-L data channel, to ensure that the teardown is not seen by the paired-DMA until all the previous PDMA data for the channel has been flushed.

The PDMA will not stop reading peripheral data until it reaches a FIFO boundary as configured through the 'X' and 'Y' parameters in the static TR. It will always attempt to complete the 'Y' count for the current event being processed. Upon reaching a stopping point, the PDMA will then clear the ENABLE bit in the pairing register, however the TDOWN bit will remain set. No further packet processing will occur until the host re-configures the channel. The teardown process will propagate to the paired-DMA and its final status can be checked there.

Copyright © 2024 Texas Instruments Incorporated

10.3.1.2.5.6 Rx Channel Reset

In the unlikely event that channel synchronization is corrupted, a channel may fail to teardown gracefully. If this occurs, the channel may be reset by clearing the PDMA_PSILCFG_RX_ENABLE[31] ENABLE bit. This will cause a local reset of the entire channel, including TR and pairing registers. Note that it does not reset the paired-DMA peer. Resetting the paired-DMA peer is also required before re-initializing and re-pairing the channel.

10.3.1.2.5.7 Rx Debug/State Register

The debug/state registers are supplied to give software applications additional information about the PDMA than they would need in regular opteration, but which may be useful in debug situations. These registers appear on the PSI-L bus, near the static TR registers. For receive, they are defined as follows:

Refer to their descriptions for further details.

10.3.1.2.6 PDMA ECC Support

The PDMA optionally integrates an ECC aggregator module that consolidates the ECC configuration and status bits for all the ECC-supported memories within the PDMA. The ECC aggregator provides a single end-of-interrupt (EOI) handshake based interrupt to the host (for both single and double error detections), and a standard 32-bit VBUSP interface for configuring and querying the various ECC wrappers via their respective register set.

In this device, only PDMA5, PDMA9 and PDMA10 include an ECC aggregator.

For detailed description of the generic ECC aggregator functionality, see *ECC Aggregator*. For register descriptions of PDMA ECC aggregators, see *PDMA Registers*.

Copyright © 2024 Texas Instruments Incorporated

10.3.1.3 PDMA Registers

Copyright © 2024 Texas Instruments Incorporated

10.3.1.3.1 PDMA PSI-L TX Configuration Registers

Table 10-144 lists the PDMA PSI-L TX configuration registers (PDMA_PSILCFG_TX). These registers are not memory mapped and are accessed indirectly via CFG_PROXY modules in NAVSS.

Table 10-143. PDMA_PSILCFG_TX Instances

Instance	Base Address
PDMA_PSILCFG_TX	N/A

Table 10-144. PDMA_PSILCFG_TX Registers

Offset	Acronym	Register Name	PDMA_PSILCFG_TX Physical Address
2h	PDMA_PSILCFG_TX_ENABLE	TX enable register	N/A
40h	PDMA_PSILCFG_TX_CAPABILITIES	TX local capabilities register	N/A
400h	PDMA_PSILCFG_TX_STATIC_TR	TX static transfer request (X, Y) register	N/A
402h	PDMA_PSILCFG_TX_DEBUG_1	TX debug/state register 1	N/A
403h	PDMA_PSILCFG_TX_DEBUG_2	TX debug/state register 2	N/A
404h	PDMA_PSILCFG_TX_BYTE_COUNT	TX byte count register	N/A
405h	PDMA_PSILCFG_TX_AASRC_TX_FIFO_CONFIG	TX AASRC Tx FIFO configuration register	N/A
406h	PDMA_PSILCFG_TX_AASRC_TX_ORDER_TABLE0	TX AASRC Tx order table 0	N/A
407h	PDMA_PSILCFG_TX_AASRC_TX_ORDER_TABLE1	TX AASRC Tx order table 1	N/A
408h	PDMA_PSILCFG_TX_RT_ENABLE	TX real-time enable register	N/A

Copyright © 2024 Texas Instruments Incorporated

10.3.1.3.1.1 PDMA_PSILCFG_TX_ENABLE Register (Offset = 2h) [reset = 0h]

PDMA_PSILCFG_TX_ENABLE is shown in Figure 10-27 and described in Table 10-146.

Return to Summary Table.

Enable Register. This register contains enable control for the thread.

Table 10-145. PDMA_PSILCFG_TX_ENABLE Instances

Instance	Physical Address
PDMA_PSILCFG_TX	N/A

Figure 10-27. PDMA_PSILCFG_TX_ENABLE Register

31	30	29	28	27	26	25	24
ENABLE	RESERVED						
R/W-0h	R-0h						
23	22	21	20	19	18	17	16
RESERVED							
R-0h							
15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
RESERVED							
R-0h							

LEGEND: R = Read Only; R/W = Read/Write; *-n* = value after reset

Table 10-146. PDMA_PSILCFG_TX_ENABLE Register Field Descriptions

Bit	Field	Type	Reset	Description
31	ENABLE	R/W	0h	When set, the TX channel is enabled. When cleared, the TX channel is disabled. When disabled, the channel discards all ingress data. A one-to-zero transition on this bit fully resets the channel.
30-0	RESERVED	R	0h	Reserved

Copyright © 2024 Texas Instruments Incorporated

10.3.1.3.1.2 PDMA_PSILCFG_TX_CAPABILITIES Register (Offset = 40h) [reset = Xh]

PDMA_PSILCFG_TX_CAPABILITIES is shown in Figure 10-28 and described in Table 10-148.

Return to Summary Table.

Local Capabilities Register. This register provides the width and count of the credits for which buffering is available in the local thread. This register can be read by the system configuration software to determine what values to place in the following registers of the source paired thread:

- PDMA_PSILCFG_RX_PEER_CREDIT and PDMA_PSILCFG_RX_PEER_THREAD_ID

This register is only implemented for destination threads.

Table 10-147. PDMA_PSILCFG_TX_CAPABILITIES Instances

Instance	Physical Address
PDMA_PSILCFG_TX	N/A

Figure 10-28. PDMA_PSILCFG_TX_CAPABILITIES Register

31	30	29	28	27	26	25	24
RESERVED			LOCAL_THREAD_WIDTH				
R-0h			R-2h				
23	22	21	20	19	18	17	16
RESERVED							
R-0h							
15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
LOCAL_CREDIT_CNT							
R-8h							

LEGEND: R = Read Only; *-n* = value after reset

Table 10-148. PDMA_PSILCFG_TX_CAPABILITIES Register Field Descriptions

Bit	Field	Type	Reset	Description
31-29	RESERVED	R	0h	Reserved
28-24	LOCAL_THREAD_WIDTH	R	2h	Read-only element width for the local thread used for pairing purposes. 0h = 4 bytes 1h = 8 bytes 2h = 16 bytes 3h to 1Fh = Reserved
23-8	RESERVED	R	0h	Reserved
7-0	LOCAL_CREDIT_CNT	R	7h	Read-only local thread free entry count used for pairing purposes. This field is encoded as follows: 0h to 80h = Available count in elements 81h to FFh = Reserved

Copyright © 2024 Texas Instruments Incorporated

10.3.1.3.1.3 PDMA_PSILCFG_TX_STATIC_TR Register (Offset = 400h) [reset = 0h]

PDMA_PSILCFG_TX_STATIC_TR is shown in Figure 10-29 and described in Table 10-150.

Return to Summary Table.

Static Transfer Request (X, Y) Register. This register is used to define the 'X' and 'Y' parameters in a static TR.

Table 10-149. PDMA_PSILCFG_TX_STATIC_TR Instances

Instance	Physical Address
PDMA_PSILCFG_TX	N/A

Figure 10-29. PDMA_PSILCFG_TX_STATIC_TR Register

31	30	29	28	27	26	25	24
BURST	ACC32	RESERVED			X		
R/W-0h	R/W-0h	R-0h			R/W-0h		
23	22	21	20	19	18	17	16
RESERVED							
R-0h							
15	14	13	12	11	10	9	8
RESERVED				Y			
R-0h				R/W-0h			
7	6	5	4	3	2	1	0
Y							
R/W-0h							

LEGEND: R = Read Only; R/W = Read/Write; *-n* = value after reset

Table 10-150. PDMA_PSILCFG_TX_STATIC_TR Register Field Descriptions

Bit	Field	Type	Reset	Description
31	BURST	R/W	0h	**X-Y FIFO mode static TR:** When set, enables VBUSP burst mode on this channel. See the XY burst description for more information.
				MCAN mode static TR: When set, enables VBUSP burst mode on this channel. See the MCAN burst description for more information.
				AASRC mode static TR: Not used.
30	ACC32	R/W	0h	**X-Y FIFO mode static TR:** When set, enables 32-bit access mode. On a 32-bit PDMA, all accesses will have XCNT=4 to support legacy IP that is not fully VBUSP compliant. This bit is ignored if the PDMA VBUSP port is not 32 bits wide.
				MCAN mode static TR: Not used.
				AASRC mode static TR: When set, enables 32-bit access mode. On a 32-bit PDMA, all accesses will have XCNT=4 to support legacy IP that is not fully VBUSP compliant. This bit is ignored if the PDMA VBUSP port is not 32 bits wide.
29-27	RESERVED	R	0h	Reserved

Copyright © 2024 Texas Instruments Incorporated

Table 10-150. PDMA_PSILCFG_TX_STATIC_TR Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
26-24	X	R/W	0h	**X-Y FIFO mode static TR:** Element size. This field specifies how much data is transferred in each write which is performed by the DMA. This field is encoded as follows: 0h = 8 bits; 1h =16 bits; 2h = 24 bits; 3h = 32 bits; 4h = 64 bits; 5h-7h = RESERVED.
				MCAN mode static TR: Not used.
				AASRC mode static TR: Element size. This field specifies how much data is transferred in each write which is performed by the DMA. This field is encoded as follows: 0h = 8 bits; 1h =16 bits; 2h = 24 bits; 3h = 32 bits; 4h = 64 bits; 5h-7h = RESERVED.
23-12	RESERVED	R	0h	Reserved
11-0	Y	R/W	0h	**X-Y FIFO mode static TR:** Element count. This field specifies how many elements to transfer each time a trigger is received on the channel.
				MCAN mode static TR: Buffer Size. This field specifies how many bytes should be written to an MCAN TX buffer. This field includes the 8-byte MCAN header on the initial packet fragment. The PDMA will break up the source packet into fragments of this buffer size, copying the 8-byte MCAN header for the initial fragment, and then skipping it for each additional fragment and thus reusing the header from the first fragment. A buffer size less than 16 is treated as 16, and a buffer size greater than 72 is treated as 72.
				AASRC mode static TR: FIFO element count. In AASRC mode, a channel can service multiple FIFOs using a list supplied in its FIFO configuration. This field specifies how many times to service the FIFO list, hence how many elements to transfer from each FIFO, each time a trigger is received on the channel. Note for each loop specified by the value of Y, the entire list is processed. For example, if the FIFO list is 0, 1, 2, 3, and Y is set to 2 loops, the FIFOs are serviced in the order: 0, 1, 2, 3, 0, 1, 2, 3.

Copyright © 2024 Texas Instruments Incorporated

10.3.1.3.1.4 PDMA_PSILCFG_TX_DEBUG_1 Register (Offset = 402h) [reset = 0h]

PDMA_PSILCFG_TX_DEBUG_1 is shown in Figure 10-30 and described in Table 10-152.

Return to Summary Table.

Debug/State Register 1. The debug/state registers give software applications additional information about the PDMA than they would need in regular opteration, but which may be useful in debug situations.

Table 10-151. PDMA_PSILCFG_TX_DEBUG_1 Instances

Instance	Physical Address
PDMA_PSILCFG_TX	N/A

Figure 10-30. PDMA_PSILCFG_TX_DEBUG_1 Register

31	30	29	28	27	26	25	24
RESERVED							
R-0h							
23	22	21	20	19	18	17	16
RESEREVD							
R-0h							
15	14	13	12	11	10	9	8
Y							
R/W-0h							
7	6	5	4	3	2	1	0
Y							
R/W-0h							

LEGEND: R = Read Only; R/W = Read/Write; *-n* = value after reset

Table 10-152. PDMA_PSILCFG_TX_DEBUG_1 Register Field Descriptions

Bit	Field	Type	Reset	Description
31-16	RESERVED	R	0h	Reserved
15-0	Y	R/W	0h	This field holds the current Y count. In X-Y FIFO mode, this is the number of X sized samples yet to write to the peripheral for the DMA event being serviced. In MCAN mode, this field holds the next write offset to use when writing to the CAN TX buffer.

Copyright © 2024 Texas Instruments Incorporated

10.3.1.3.1.5 PDMA_PSILCFG_TX_DEBUG_2 Register (Offset = 403h) [reset = 0h]

PDMA_PSILCFG_TX_DEBUG_2 is shown in Figure 10-31 and described in Table 10-154.

Return to Summary Table.

Debug/State Register 2. The debug/state registers give software applications additional information about the PDMA than they would need in regular opteration, but which may be useful in debug situations.

Table 10-153. PDMA_PSILCFG_TX_DEBUG_2 Instances

Instance	Physical Address
PDMA_PSILCFG_TX	N/A

Figure 10-31. PDMA_PSILCFG_TX_DEBUG_2 Register

31	30	29	28	27	26	25	24
INEVENT	FLUSH	PAUSE	DATA	XDATA	RESERVED		
R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R-0h		
23	**22**	**21**	**20**	**19**	**18**	**17**	**16**
STATE				EVENTCNT			
R/W-0h				R/W-0h			
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**
RESERVED							
R-0h							
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
RESERVED							
R-0h							

LEGEND: R = Read Only; R/W = Read/Write; *-n* = value after reset

Table 10-154. PDMA_PSILCFG_TX_DEBUG_2 Register Field Descriptions

Bit	Field	Type	Reset	Description
31	INEVENT	R/W	0h	When set, the PDMA is in the middle of processing a FIFO event.
30	FLUSH	R/W	0h	When set, the PDMA is processing in a flushing state, where it runs without waiting for DMA requests and without writing data to the peripheral. It is only operating its internal state machine to allow internal data pipes to drain properly.
29	PAUSE	R/W	0h	When set, the PDMA waiting in a paused state. This bit will clear when data starts flowing again from the UDMA-P.
28	DATA	R/W	0h	When set, there is a non-zero amount of data still waiting to be written to the peripheral.
27	XDATA	R/W	0h	When set, there is enough data still waiting to be written to the peripheral to start servicing a peripheral DMA event.
26-24	RESERVED	R	0h	Reserved
23-20	STATE	R/W	0h	This code reflects the current state of the PDMA channel, and is specific to the current implementation.
19-16	EVENTCNT	R/W	0h	This field holds the number of backlogged DMA events yet to be serviced.
15-0	RESERVED	R	0h	Reserved

Copyright © 2024 Texas Instruments Incorporated

10.3.1.3.1.6 PDMA_PSILCFG_TX_BYTE_COUNT Register (Offset = 404h) [reset = 0h]

PDMA_PSILCFG_TX_BYTE_COUNT is shown in Figure 10-32 and described in Table 10-156.

Return to Summary Table.

Byte Count Register. This register contains the number of bytes that have been written to the VBUSP mapped peripheral.

Table 10-155. PDMA_PSILCFG_TX_BYTE_COUNT Instances

Instance	Physical Address
PDMA_PSILCFG_TX	N/A

Figure 10-32. PDMA_PSILCFG_TX_BYTE_COUNT Register

31	30	29	28	27	26	25	24
BYTES							
R/W-0h							
23	22	21	20	19	18	17	16
BYTES							
R/W-0h							
15	14	13	12	11	10	9	8
BYTES							
R/W-0h							
7	6	5	4	3	2	1	0
BYTES							
R/W-0h							

LEGEND: R = Read Only; R/W = Read/Write; *-n* = value after reset

Table 10-156. PDMA_PSILCFG_TX_BYTE_COUNT Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	BYTES	R/W	0h	This register contains the number of bytes that have been read from the VBUSP mapped peripheral. The register is write-to-decrement, so running counts can be tracked by reading register and then writing back the value that was read. It will wrap on overflow. It will reset to zero on a channel reset.

Copyright © 2024 Texas Instruments Incorporated

10.3.1.3.1.7 PDMA_PSILCFG_TX_AASRC_TX_FIFO_CONFIG Register (Offset = 405h) [reset = 0h]

PDMA_PSILCFG_TX_AASRC_TX_FIFO_CONFIG is shown in Figure 10-33 and described in Table 10-158.

Return to Summary Table.

AASRC Tx FIFO configuration register.

Table 10-157. PDMA_PSILCFG_TX_AASRC_TX_FIFO_CONFIG Instances

Instance	Physical Address
PDMA_PSILCFG_TX	N/A

Figure 10-33. PDMA_PSILCFG_TX_AASRC_TX_FIFO_CONFIG Register

31	30	29	28	27	26	25	24
GROUPMODE	DMAREQRESET	RESERVED					
R/W-0h	R/W-0h	R-0h					
23	22	21	20	19	18	17	16
LASTSLOT				FIRSTSLOT			
R/W-0h				R/W-0h			
15	14	13	12	11	10	9	8
DMAREQMASK							
R/W-0h							
7	6	5	4	3	2	1	0
DMAREQMASK							
R/W-0h							

LEGEND: R = Read Only; R/W = Read/Write; *-n* = value after reset

Table 10-158. PDMA_PSILCFG_TX_AASRC_TX_FIFO_CONFIG Register Field Descriptions

Bit	Field	Type	Reset	Description
31	GROUPMODE	R/W	0h	When set, the channel is in 'group mode'. It will look for group mode DMA requests, and access the group mode FIFOs. When clear, the channel is 'stream mode'. It will look for stream FIFO DMA requests, and access the stream mode FIFOs.
30	DMAREQRESET	R/W	0h	When set, resets any latched DMA request using the DMAREQMASK. This bit is self-clearing. It should be used to synchronize the AASRC event status with the PDMA in the event that the AASRC has been previously used and it is not known if the PDMA may have latched, and is holding, previous DMA requests.
29-24	RESERVED	R	0h	Reserved
23-20	LASTSLOT	R/W	0h	This is the index (0-15) of the last slot in the TX FIFO ordering table used by this channel. The ordering table is read to get the FIFO index to access for each slot, starting with the first and ending with the last.
19-16	FIRSTSLOT	R/W	0h	This is the index (0-15) of the first slot in the TX FIFO ordering table used by this channel. The ordering table is read to get the FIFO index to access for each slot, starting with the first and ending with the last.

Copyright © 2024 Texas Instruments Incorporated

Table 10-158. PDMA_PSILCFG_TX_AASRC_TX_FIFO_CONFIG Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
15-0	DMAREQMASK	R/W	0h	This field holds a set of flags indicating which AASRC DMA requests must fire in order for this channel to activate. In steam mode, these 16 flags correspond to the 16 DMA requests for each TX FIFO. The flags corresponding to all TX FIFOs involved with the channel should be set to 1. In group mode, these flags indicate which group mode DMA requests must fire. In this case, only bits [3:0] are relevant and only one bit should be set to 1 as a DMA channel only services a single group.

Copyright © 2024 Texas Instruments Incorporated

10.3.1.3.1.8 PDMA_PSILCFG_TX_AASRC_TX_ORDER_TABLE0 Register (Offset = 406h) [reset = 0h]

PDMA_PSILCFG_TX_AASRC_TX_ORDER_TABLE0 is shown in Figure 10-34 and described in Table 10-160.

Return to Summary Table.

AASRC TX order table 0 register.

Table 10-159. PDMA_PSILCFG_TX_AASRC_TX_ORDER_TABLE0 Instances

Instance	Physical Address
PDMA_PSILCFG_TX	N/A

Figure 10-34. PDMA_PSILCFG_TX_AASRC_TX_ORDER_TABLE0 Register

31	30	29	28	27	26	25	24
ENTRY7				ENTRY6			
R/W-0h				R/W-0h			
23	22	21	20	19	18	17	16
ENTRY5				ENTRY4			
R/W-0h				R/W-0h			
15	14	13	12	11	10	9	8
ENTRY3				ENTRY2			
R/W-0h				R/W-0h			
7	6	5	4	3	2	1	0
ENTRY1				ENTRY0			
R/W-0h				R/W-0h			

LEGEND: R/W = Read/Write; *-n* = value after reset

Table 10-160. PDMA_PSILCFG_TX_AASRC_TX_ORDER_TABLE0 Register Field Descriptions

Bit	Field	Type	Reset	Description
31-28	ENTRY7	R/W	7h	TX FIFO index for slot 7
27-24	ENTRY6	R/W	6h	TX FIFO index for slot 6
23-20	ENTRY5	R/W	5h	TX FIFO index for slot 5
19-16	ENTRY4	R/W	4h	TX FIFO index for slot 4
15-12	ENTRY3	R/W	3h	TX FIFO index for slot 3
11-8	ENTRY2	R/W	2h	TX FIFO index for slot 2
7-4	ENTRY1	R/W	1h	TX FIFO index for slot 1
3-0	ENTRY0	R/W	0h	TX FIFO index for slot 0

Copyright © 2024 Texas Instruments Incorporated

10.3.1.3.1.9 PDMA_PSILCFG_TX_AASRC_TX_ORDER_TABLE1 Register (Offset = 407h) [reset = 0h]

PDMA_PSILCFG_TX_AASRC_TX_ORDER_TABLE1 is shown in Figure 10-35 and described in Table 10-162.

Return to Summary Table.

AASRC Tx order table 1 register.

Table 10-161. PDMA_PSILCFG_TX_AASRC_TX_ORDER_TABLE1 Instances

Instance	Physical Address
PDMA_PSILCFG_TX	N/A

Figure 10-35. PDMA_PSILCFG_TX_AASRC_TX_ORDER_TABLE1 Register

31	30	29	28	27	26	25	24
ENTRY15				ENTRY14			
R/W-0h				R/W-0h			
23	22	21	20	19	18	17	16
ENTRY13				ENTRY12			
R/W-0h				R/W-0h			
15	14	13	12	11	10	9	8
ENTRY11				ENTRY10			
R/W-0h				R/W-0h			
7	6	5	4	3	2	1	0
ENTRY9				ENTRY8			
R/W-0h				R/W-0h			

LEGEND: R/W = Read/Write; *-n* = value after reset

Table 10-162. PDMA_PSILCFG_TX_AASRC_TX_ORDER_TABLE1 Register Field Descriptions

Bit	Field	Type	Reset	Description
31-28	ENTRY15	R/W	Fh	TX FIFO index for slot 15
27-24	ENTRY14	R/W	Eh	TX FIFO index for slot 14
23-20	ENTRY13	R/W	Dh	TX FIFO index for slot 13
19-16	ENTRY12	R/W	Ch	TX FIFO index for slot 12
15-12	ENTRY11	R/W	Bh	TX FIFO index for slot 11
11-8	ENTRY10	R/W	Ah	TX FIFO index for slot 10
7-4	ENTRY9	R/W	9h	TX FIFO index for slot 9
3-0	ENTRY8	R/W	8h	TX FIFO index for slot 8

Copyright © 2024 Texas Instruments Incorporated

10.3.1.3.1.10 PDMA_PSILCFG_TX_RT_ENABLE Register (Offset = 408h) [reset = 0h]

PDMA_PSILCFG_TX_RT_ENABLE is shown in Figure 10-36 and described in Figure 10-36.

Return to Summary Table.

Real Time Enable Register. This register allows enabling various channel settings in real time.

Table 10-163. PDMA_PSILCFG_TX_RT_ENABLE Instances

Instance	Physical Address
PDMA_PSILCFG_TX	N/A

Figure 10-36. PDMA_PSILCFG_TX_RT_ENABLE Register

31	30	29	28	27	26	25	24
ENABLE	TDOWN	PAUSE	FLUSH	RESERVED			
R/W-0h	R/W-0h	R/W-0h	R/W-0h	R-0h			
23	22	21	20	19	18	17	16
RESERVED							
R-0h							
15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
RESERVED					ERROR	IDLE	FREE
R-0h					R/W-0h	R-0h	R/W-0h

LEGEND: R = Read Only; R/W = Read/Write; *-n* = value after reset

Table 10-164. PDMA_PSILCFG_TX_RT_ENABLE Register Field Descriptions

Bit	Field	Type	Reset	Description
31	ENABLE	R/W	0h	When set, the TX channel is enabled. When cleared, the TX channel is disabled. When disabled, it discards all held data. It clears DMA event counts and ignores all future DMA events from the peripheral. It maintains data exchange with the UDMA-P so that credit handshake is not disrupted. A 'hard teardown' can be performed by directly clearing this bit. Note that this bit cannot be changed from 0 to 1 if the global enable bit in the peer enable register is 0.
30	TDOWN	R/W	0h	When set, the channel will commence a TX channel teardown procedure. To perform a TX teardown, the teardown bit should be set in the UDMA-P and it will automatically propagate to this register bit with the normal flow of peripheral data. Once the channel is fully stopped and ready to be reused (including returning all credits), the ENABLE bit is cleared.
29	PAUSE	R/W	0h	When set, the channel is in a paused state. It will stop on the next FIFO boundary. It continues to accept and count DMA events from the peripherals but will not act on them. The PAUSE bit can be cleared and data will resume.

Copyright © 2024 Texas Instruments Incorporated

Table 10-164. PDMA_PSILCFG_TX_RT_ENABLE Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
28	FLUSH	R/W	0h	When set, causes all TX channel data to be discarded instead of being written to the peripheral. It essentially allows the TX engine to 'free run' without DMA requests from the peripheral (it will also override 'pause'). This bit should be set only when a channel fails to complete its teardown procedure normally because a peripheral is no longer functioning or because data flow was halted on a boundary that is not compatible with the static TR configuration.
27-3	RESERVED	R	0h	Reserved
2	ERROR	R/W	0h	When set, the channel has encountered a PSI-L protocol violation. The ERROR bit can only be set by hardware and can only be cleared by software. Once this bit is set, the channel should be fully reset and re-initialized via the PSI-L pairing registers.
1	IDLE	R	0h	This is a read-only bit that signifies that the disabled thread is also idle. This bit is read only and can only become set if the ENABLE bit is cleared.
0	FREE	R/W	0h	When cleared, the channel honors the debug suspend signal. When set, the channel will 'free run', regardless of the value of debug suspend.

Copyright © 2024 Texas Instruments Incorporated

10.3.1.3.2 PDMA PSI-L RX Configuration Registers

Table 10-166 lists the PDMA PSI-L RX configuration registers (PDMA_PSILCFG_RX). These registers are not memory mapped and are accessed indirectly via CFG_PROXY modules in NAVSS.

Table 10-165. PDMA_PSILCFG_RX Instances

Instance	Base Address
PDMA_PSILCFG_RX	N/A

Table 10-166. PDMA_PSILCFG_RX Registers

Offset	Acronym	Register Name	PDMA_PSILCFG_RX Physical Address
0h	PDMA_PSILCFG_RX_PEER_THREAD_ID	RX peer thread ID register	N/A
1h	PDMA_PSILCFG_RX_PEER_CREDIT	RX peer credit register	N/A
2h	PDMA_PSILCFG_RX_ENABLE	RX enable register	N/A
400h	PDMA_PSILCFG_RX_STATIC_TR	RX static transfer request (X, Y) register	N/A
401h	PDMA_PSILCFG_RX_STATIC_TR_Z	RX static transfer request (Z) register	N/A
402h	PDMA_PSILCFG_RX_DEBUG_1	RX debug/state register 1	N/A
403h	PDMA_PSILCFG_RX_DEBUG_2	RX debug/state register 2	N/A
404h	PDMA_PSILCFG_RX_BYTE_COUNT	RX byte count register	N/A
405h	PDMA_PSILCFG_RX_AASRC_RX_FIFO_CONFIG	RX AASRC Rx FIFO configuration register	N/A
406h	PDMA_PSILCFG_RX_AASRC_RX_ORDER_TABLE0	RX AASRC Rx order table 0	N/A
407h	PDMA_PSILCFG_RX_AASRC_RX_ORDER_TABLE1	RX AASRC Rx order table 1	N/A
408h	PDMA_PSILCFG_RX_RT_ENABLE	RX real-time enable register	N/A
40Fh	PDMA_PSILCFG_RX_DEBUG_3	RX debug/state register 3	N/A

Copyright © 2024 Texas Instruments Incorporated

10.3.1.3.2.1 PDMA_PSILCFG_RX_PEER_THREAD_ID Register (Offset = 0h) [reset = 0h]

PDMA_PSILCFG_RX_PEER_THREAD_ID is shown in Figure 10-37 and described in Table 10-168.

Return to Summary Table.

Peer Thread ID Register. This register contains the thread ID of the thread destination which is used for routing the messages. This register is only implemented for source threads.

Table 10-167. PDMA_PSILCFG_RX_PEER_THREAD_ID Instances

Instance	Physical Address
PDMA_PSILCFG_RX	N/A

Figure 10-37. PDMA_PSILCFG_RX_PEER_THREAD_ID

31	30	29	28	27	26	25	24
THREAD_PRIORITY			PEER_THREAD_WIDTH				
R-0h			R/W-0h				
23	22	21	20	19	18	17	16
RESERVED							
R-0h							
15	14	13	12	11	10	9	8
PEER_THREAD_ID							
R/W-0h							
7	6	5	4	3	2	1	0
PEER_THREAD_ID							
R/W-0h							

LEGEND: R = Read Only; R/W = Read/Write; *-n* = value after reset

Table 10-168. PDMA_PSILCFG_RX_PEER_THREAD_ID Register Field Descriptions

Bit	Field	Type	Reset	Description
31-29	THREAD_PRIORITY	R	0h	Priority level to use for this source thread in arbitration. This field is hard coded to 0h and is not writable
28-24	PEER_THREAD_WIDTH	R/W	0h	Datapath width of paired peer destination thread. This field is encoded as follows: 0h = 32-bit 1h = 64-bit 2h = 128-bit 3h = 256-bit 4h = 512-bit
23-16	RESERVED	R	0h	Reserved
15-0	PEER_THREAD_ID	R/W	0h	Thread ID to which all nontransfer response, nonconfiguration messages from this thread are sent

Copyright © 2024 Texas Instruments Incorporated

10.3.1.3.2.2 PDMA_PSILCFG_RX_PEER_CREDIT Register (Offset = 1h) [reset = 0h]

PDMA_PSILCFG_RX_PEER_CREDIT is shown in Figure 10-38 and described in Table 10-170.

Return to Summary Table.

Peer Credit Register. This register contains a free credit count in completed destination data phases for the thread destination. This register is only implemented for source threads.

Table 10-169. PDMA_PSILCFG_RX_PEER_CREDIT Instances

Instance	Physical Address
PDMA_PSILCFG_RX	N/A

Figure 10-38. PDMA_PSILCFG_RX_PEER_CREDIT Register

31	30	29	28	27	26	25	24
RESERVED							
R-0h							
23	22	21	20	19	18	17	16
RESERVED							
R-0h							
15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
CREDIT_CNT							
R/W-0h							

LEGEND: R = Read Only; R/W = Read/Write; *-n* = value after reset

Table 10-170. PDMA_PSILCFG_RX_PEER_CREDIT Register Field Descriptions

Bit	Field	Type	Reset	Description
31-8	RESERVED	R	0h	Reserved
7-0	CREDIT_CNT	R/W	0h	Free entries in destination thread. This field is encoded as follows: 0h to 80h = Actual count in data phases 81h to FFh = Reserved

Copyright © 2024 Texas Instruments Incorporated

10.3.1.3.2.3 PDMA_PSILCFG_RX_ENABLE Register (Offset = 2h) [reset = 0h]

PDMA_PSILCFG_RX_ENABLE is shown in Figure 10-39 and described in Table 10-172.

Return to Summary Table.

Enable Register. This register contains enable control for the thread.

Table 10-171. PDMA_PSILCFG_RX_ENABLE Instances

Instance	Physical Address
PDMA_PSILCFG_RX	N/A

Figure 10-39. PDMA_PSILCFG_RX_ENABLE Register

31	30	29	28	27	26	25	24
ENABLE	RESERVED						
R/W-0h	R-0h						
23	22	21	20	19	18	17	16
RESERVED							
R-0h							
15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
RESERVED							
R-0h							

LEGEND: R = Read Only; R/W = Read/Write; *-n* = value after reset

Table 10-172. PDMA_PSILCFG_RX_ENABLE Register Field Descriptions

Bit	Field	Type	Reset	Description
31	ENABLE	R/W	0h	When set, the RX channel is enabled. When cleared, the RX channel is disabled. When disabled, the channel ignores all credit returns, and will not generate egress data. A one-to-zero transition on this bit fully resets the channel.
30-0	RESERVED	R	0h	Reserved

Copyright © 2024 Texas Instruments Incorporated

10.3.1.3.2.4 PDMA_PSILCFG_RX_STATIC_TR Register (Offset = 400h) [reset = 0h]

PDMA_PSILCFG_RX_STATIC_TR is shown in Figure 10-40 and described in Table 10-174.

Return to Summary Table.

Static Transfer Request (X, Y) Register. This register is used to define the 'X' and 'Y' parameters in a static TR.

Table 10-173. PDMA_PSILCFG_RX_STATIC_TR Instances

Instance	Physical Address
PDMA_PSILCFG_RX	N/A

Figure 10-40. PDMA_PSILCFG_RX_STATIC_TR Register

31	30	29	28	27	26	25	24
BURST	ACC32	RESERVED			X		
R/W-0h	R/W-0h	R-0h			R/W-0h		
23	22	21	20	19	18	17	16
RESERVED							
R-0h							
15	14	13	12	11	10	9	8
RESERVED				Y			
R-0h				R/W-0h			
7	6	5	4	3	2	1	0
Y							
R/W-0h							

LEGEND: R = Read Only; R/W = Read/Write; *-n* = value after reset

Table 10-174. PDMA_PSILCFG_RX_STATIC_TR Register Field Descriptions

Bit	Field	Type	Reset	Description
31	BURST	R/W	0h	**X-Y FIFO mode static TR:** When set, enables VBUSP burst mode on this channel. See the XY burst description for more information.
				MCAN mode static TR: When set, enables VBUSP burst mode on this channel. See the MCAN burst description for more information.
				AASRC mode static TR: Not used.
30	ACC32	R/W	0h	**X-Y FIFO mode static TR:** When set, enables 32-bit access mode. On a 32-bit PDMA, all accesses will have XCNT=4 to support legacy IP that is not fully VBUSP compliant. This bit is ignored if the PDMA VBUSP port is not 32 bits wide.
				MCAN mode static TR: Not used.
				AASRC mode static TR: When set, enables 32-bit access mode. On a 32-bit PDMA, all accesses will have XCNT=4 to support legacy IP that is not fully VBUSP compliant. This bit is ignored if the PDMA VBUSP port is not 32 bits wide.
29-27	RESERVED	R	0h	Reserved

Copyright © 2024 Texas Instruments Incorporated

Table 10-174. PDMA_PSILCFG_RX_STATIC_TR Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
26-24	X	R/W	0h	**X-Y FIFO mode static TR:** Element size. This field specifies how much data is transferred in each write which is performed by the DMA. This field is encoded as follows: 0h = 8 bits; 1h =16 bits; 2h = 24 bits; 3h = 32 bits; 4h = 64 bits; 5h-7h = RESERVED.
				MCAN mode static TR: Not used.
				AASRC mode static TR: Element size. This field specifies how much data is transferred in each write which is performed by the DMA. This field is encoded as follows: 0h = 8 bits; 1h =16 bits; 2h = 24 bits; 3h = 32 bits; 4h = 64 bits; 5h-7h = RESERVED.
23-12	RESERVED	R	0h	Reserved
11-0	Y	R/W	0h	**X-Y FIFO mode static TR:** Element count. This field specifies how many elements to transfer each time a trigger is received on the channel.
				MCAN mode static TR: Buffer Size. This field specifies how many bytes should be read from an MCAN RX buffer. This field includes the 8 byte MCAN header on the initial packet fragment. A buffer size less than 16 is treated as 16, and a buffer size greater than 72 is treated as 72.
				AASRC mode static TR: FIFO element count. In AASRC mode, a channel can service multiple FIFOs using a list supplied in its FIFO configuration. This field specifies how many times to service the FIFO list, hence how many elements to transfer from each FIFO, each time a trigger is received on the channel. Note for each loop specified by the value of Y, the entire list is processed. For example, if the FIFO list is 0, 1, 2, 3, and Y is set to 2 loops, the FIFOs are serviced in the order: 0, 1, 2, 3, 0, 1, 2, 3.

Copyright © 2024 Texas Instruments Incorporated

10.3.1.3.2.5 PDMA_PSILCFG_RX_STATIC_TR_Z Register (Offset = 401h) [reset = 0h]

PDMA_PSILCFG_RX_STATIC_TR_Z is shown in Figure 10-41 and described in Table 10-176.

Return to Summary Table.

Static Transfer Request (Z) Register. This register is used to define the 'Z' parameter in a static TR.

Table 10-175. PDMA_PSILCFG_RX_STATIC_TR_Z Instances

Instance	Physical Address
PDMA_PSILCFG_RX	N/A

Figure 10-41. PDMA_PSILCFG_RX_STATIC_TR_Z Register

31	30	29	28	27	26	25	24
EOL	RESERVED						
R/W-0h	R-0h						

23	22	21	20	19	18	17	16
RESERVED							
R-0h							

15	14	13	12	11	10	9	8
RESERVED				Z			
R-0h				R/W-0h			

7	6	5	4	3	2	1	0
Z							
R/W-0h							

LEGEND: R = Read Only; R/W = Read/Write; *-n* = value after reset

Table 10-176. PDMA_PSILCFG_RX_STATIC_TR_Z Register Field Descriptions

Bit	Field	Type	Reset	Description
31	EOL	R/W	0h	EOL mode. Normally, when the Z count of FIFO operations has been reached, the PDMA will close the packet with an 'EOP' indication. When this flag is set, the PDMA will instead trigger an EOL at the completion of Z.
30-12	RESERVED	R	0h	Reserved

Copyright © 2024 Texas Instruments Incorporated

Table 10-176. PDMA_PSILCFG_RX_STATIC_TR_Z Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
11-0	Z	R/W	0h	**X-Y FIFO mode static TR:** FIFO count. This field specifies how many full FIFO operations comprise a complete packet. When the count has been reached, the PDMA will close the packet with an EOP indication. If this parameter is set to NULL, then no packet delineation is supplied by the PDMA and all framing is controlled via the UDMA-P TR.
				MCAN mode static TR: Buffer Count. This field specifies how many MCAN RX buffers should be read before closing the CPPI packet with an EOP indication. When this count is greater than 1, multiple MCAN RX buffers will be read into a single CPPI packet buffer. The 8-byte MCAN header will be skipped on subsequent MCAN buffer reads. Setting this field to NULL will suppress all packet delineation, and should be avoided.
				AASRC mode static TR: FIFO count. This field specifies how many full FIFO operations comprise a complete packet. When the count has been reached, the PDMA will close the packet with an EOP indication. If this parameter is set to NULL, then no packet delineation is supplied by the PDMA and all framing is controlled via the UDMA-P TR.

Copyright © 2024 Texas Instruments Incorporated

10.3.1.3.2.6 PDMA_PSILCFG_RX_DEBUG_1 Register (Offset = 402h) [reset = 0h]

PDMA_PSILCFG_RX_DEBUG_1 is shown in Figure 10-42 and described in Table 10-178.

Return to Summary Table.

Debug/State Register 1. The debug/state registers give software applications additional information about the PDMA than they would need in regular opteration, but which may be useful in debug situations.

Table 10-177. PDMA_PSILCFG_RX_DEBUG_1 Instances

Instance	Physical Address
PDMA_PSILCFG_RX	N/A

Figure 10-42. PDMA_PSILCFG_RX_DEBUG_1 Register

31	30	29	28	27	26	25	24
Z							
R/W-0h							
23	22	21	20	19	18	17	16
Z							
R/W-0h							
15	14	13	12	11	10	9	8
Y							
R/W-0h							
7	6	5	4	3	2	1	0
Y							
R/W-0h							

LEGEND: R = Read Only; R/W = Read/Write; *-n* = value after reset

Table 10-178. PDMA_PSILCFG_RX_DEBUG_1 Register Field Descriptions

Bit	Field	Type	Reset	Description
31-16	Z	R/W	0h	This field holds the current Z count. In X-Y FIFO mode, this field holds the 1 based FIFO count of the FIFO being currently read, or the number of FIFO completions when the current operation completes. In MCAN mode, this field holds the zero based buffer index of the buffer currently being read, or the number of previously completed buffers.
15-0	Y	R/W	0h	This field holds the current Y count. In X-Y FIFO mode, this is the number of X sized samples yet to be read from the peripheral for the DMA event being serviced. In MCAN mode, this field holds the next read offset to use when read to the CAN RX buffer.

Copyright © 2024 Texas Instruments Incorporated

10.3.1.3.2.7 PDMA_PSILCFG_RX_DEBUG_2 Register (Offset = 403h) [reset = 0h]

PDMA_PSILCFG_RX_DEBUG_2 is shown in Figure 10-43 and described in Table 10-180.

Return to Summary Table.

Debug/State Register 2. The debug/state registers give software applications additional information about the PDMA than they would need in regular opteration, but which may be useful in debug situations.

Table 10-179. PDMA_PSILCFG_RX_DEBUG_2 Instances

Instance	Physical Address
PDMA_PSILCFG_RX	N/A

Figure 10-43. PDMA_PSILCFG_RX_DEBUG_2 Register

31	30	29	28	27	26	25	24
INEVENT	TDOWN	PAUSE	SPACE	XSPACE	BUFFER	RESERVED	
R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R-0h	
23	22	21	20	19	18	17	16
STATE				EVENTCNT			
R/W-0h				R/W-0h			
15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
RESERVED							
R-0h							

LEGEND: R = Read Only; R/W = Read/Write; *-n* = value after reset

Table 10-180. PDMA_PSILCFG_RX_DEBUG_2 Register Field Descriptions

Bit	Field	Type	Reset	Description
31	INEVENT	R/W	0h	When set, the PDMA is in the middle of processing a FIFO event.
30	TDOWN	R/W	0h	When set, the PDMA is processing a teardown operation. This bit is set simultaneously with the teardown bit in the source (RX) RT enable register. This bit will clear when the teardown is complete, regardless as to if the teardown bit in the pairing register is cleared or not. The teardown will propagate to the UDMA-P and its full completion status can be checked there.
29	PAUSE	R/W	0h	When set, the PDMA is stopped in a paused state. This bit will clear if the channel is unpaused or disabled.
28	SPACE	R/W	0h	When set, there is a non-zero amount of internal FIFO space available to hold new read data.
27	XSPACE	R/W	0h	When set, there is enough internal FIFO space available to start servicing a peripheral DMA event.
26	BUFFER	R/W	0h	This is the current RX buffer (0/1) for the current MCAN receive operation.
25-24	RESERVED	R	0h	Reserved
23-20	STATE	R/W	0h	This code reflects the current state of the PDMA channel, and is specific to the current implementation.
19-16	EVENTCNT	R/W	0h	This field holds the number of backlogged DMA events yet to be serviced.

Copyright © 2024 Texas Instruments Incorporated

Table 10-180. PDMA_PSILCFG_RX_DEBUG_2 Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
15-0	RESERVED	R	0h	Reserved

Copyright © 2024 Texas Instruments Incorporated

10.3.1.3.2.8 PDMA_PSILCFG_RX_BYTE_COUNT Register (Offset = 404h) [reset = 0h]

PDMA_PSILCFG_RX_BYTE_COUNT is shown in Figure 10-44 and described in Table 10-182.

Return to Summary Table.

Byte Count Register. This register contains the number of bytes that have been written to the VBUSP mapped peripheral.

Table 10-181. PDMA_PSILCFG_RX_BYTE_COUNT Instances

Instance	Physical Address
PDMA_PSILCFG_RX	N/A

Figure 10-44. PDMA_PSILCFG_RX_BYTE_COUNT Register

31	30	29	28	27	26	25	24
BYTES							
R/W-0h							
23	22	21	20	19	18	17	16
BYTES							
R/W-0h							
15	14	13	12	11	10	9	8
BYTES							
R/W-0h							
7	6	5	4	3	2	1	0
BYTES							
R/W-0h							

LEGEND: R = Read Only; R/W = Read/Write; *-n* = value after reset

Table 10-182. PDMA_PSILCFG_RX_BYTE_COUNT Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	BYTES	R/W	0h	This register contains the number of bytes that have been read from the VBUSP mapped peripheral. The register is write-to-decrement, so running counts can be tracked by reading register and then writing back the value that was read. It will wrap on overflow. It will reset to zero on a channel reset.

Copyright © 2024 Texas Instruments Incorporated

10.3.1.3.2.9 PDMA_PSILCFG_RX_AASRC_RX_FIFO_CONFIG Register (Offset = 405h) [reset = 0h]

PDMA_PSILCFG_RX_AASRC_RX_FIFO_CONFIG is shown in Figure 10-45 and described in Table 10-184.

Return to Summary Table.

AASRC Rx FIFO configuration register.

Table 10-183. PDMA_PSILCFG_RX_AASRC_RX_FIFO_CONFIG Instances

Instance	Physical Address
PDMA_PSILCFG_RX	N/A

Figure 10-45. PDMA_PSILCFG_RX_AASRC_RX_FIFO_CONFIG Register

31	30	29	28	27	26	25	24
GROUPMODE	DMAREQRESET	RESERVED					
R/W-0h	R/W-0h	R-0h					
23	22	21	20	19	18	17	16
LASTSLOT				FIRSTSLOT			
R/W-0h				R/W-0h			
15	14	13	12	11	10	9	8
DMAREQMASK							
R/W-0h							
7	6	5	4	3	2	1	0
DMAREQMASK							
R/W-0h							

LEGEND: R = Read Only; R/W = Read/Write; *-n* = value after reset

Table 10-184. PDMA_PSILCFG_RX_AASRC_RX_FIFO_CONFIG Register Field Descriptions

Bit	Field	Type	Reset	Description
31	GROUPMODE	R/W	0h	When set, the channel is in 'group mode'. It will look for group mode DMA requests, and access the group mode FIFOs. When clear, the channel is 'stream mode'. It will look for stream FIFO DMA requests, and access the stream mode FIFOs.
30	DMAREQRESET	R/W	0h	When set, resets any latched DMA request using the DMAREQMASK. This bit is self-clearing. It should be used to synchronize the AASRC event status with the PDMA in the event that the AASRC has been previously used and it is not known if the PDMA may have latched, and is holding, previous DMA requests.
29-24	RESERVED	R	0h	Reserved
23-20	LASTSLOT	R/W	0h	This is the index (0-15) of the last slot in the RX FIFO ordering table used by this channel. The ordering table is read to get the FIFO index to access for each slot, starting with the first and ending with the last.
19-16	FIRSTSLOT	R/W	0h	This is the index (0-15) of the first slot in the RX FIFO ordering table used by this channel. The ordering table is read to get the FIFO index to access for each slot, starting with the first and ending with the last.

Copyright © 2024 Texas Instruments Incorporated

Table 10-184. PDMA_PSILCFG_RX_AASRC_RX_FIFO_CONFIG Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
15-0	DMAREQMASK	R/W	0h	This field holds a set of flags indicating which AASRC DMA requests must fire in order for this channel to activate. In steam mode, these 16 flags correspond to the 16 DMA requests for each RX FIFO. The flags corresponding to all RX FIFOs involved with the channel should be set to 1. In group mode, these flags indicate which group mode DMA requests must fire. In this case, only bits [3:0] are relevant and only one bit should be set to 1 as a DMA channel only services a single group.

Copyright © 2024 Texas Instruments Incorporated

10.3.1.3.2.10 PDMA_PSILCFG_RX_AASRC_RX_ORDER_TABLE0 Register (Offset = 406h) [reset = 0h]

PDMA_PSILCFG_RX_AASRC_RX_ORDER_TABLE0 is shown in Figure 10-46 and described in Table 10-186.

Return to Summary Table.

AASRC RX order table 0 register.

Table 10-185. PDMA_PSILCFG_RX_AASRC_RX_ORDER_TABLE0 Instances

Instance	Physical Address
PDMA_PSILCFG_RX	N/A

Figure 10-46. PDMA_PSILCFG_RX_AASRC_RX_ORDER_TABLE0 Register

31	30	29	28	27	26	25	24
ENTRY7				ENTRY6			
R/W-0h				R/W-0h			
23	22	21	20	19	18	17	16
ENTRY5				ENTRY4			
R/W-0h				R/W-0h			
15	14	13	12	11	10	9	8
ENTRY3				ENTRY2			
R/W-0h				R/W-0h			
7	6	5	4	3	2	1	0
ENTRY1				ENTRY0			
R/W-0h				R/W-0h			

LEGEND: R/W = Read/Write; *-n* = value after reset

Table 10-186. PDMA_PSILCFG_RX_AASRC_RX_ORDER_TABLE0 Register Field Descriptions

Bit	Field	Type	Reset	Description
31-28	ENTRY7	R/W	7h	RX FIFO index for slot 7
27-24	ENTRY6	R/W	6h	RX FIFO index for slot 6
23-20	ENTRY5	R/W	5h	RX FIFO index for slot 5
19-16	ENTRY4	R/W	4h	RX FIFO index for slot 4
15-12	ENTRY3	R/W	3h	RX FIFO index for slot 3
11-8	ENTRY2	R/W	2h	RX FIFO index for slot 2
7-4	ENTRY1	R/W	1h	RX FIFO index for slot 1
3-0	ENTRY0	R/W	0h	RX FIFO index for slot 0

Copyright © 2024 Texas Instruments Incorporated

10.3.1.3.2.11 PDMA_PSILCFG_RX_AASRC_RX_ORDER_TABLE1 Register (Offset = 407h) [reset = 0h]

PDMA_PSILCFG_RX_AASRC_RX_ORDER_TABLE1 is shown in Figure 10-47 and described in Table 10-188.

Return to Summary Table.

AASRC RX order table 1 register.

Table 10-187. PDMA_PSILCFG_RX_AASRC_RX_ORDER_TABLE1 Instances

Instance	Physical Address
PDMA_PSILCFG_RX	N/A

Figure 10-47. PDMA_PSILCFG_RX_AASRC_RX_ORDER_TABLE1 Register

31	30	29	28	27	26	25	24
ENTRY15				ENTRY14			
R/W-0h				R/W-0h			
23	22	21	20	19	18	17	16
ENTRY13				ENTRY12			
R/W-0h				R/W-0h			
15	14	13	12	11	10	9	8
ENTRY11				ENTRY10			
R/W-0h				R/W-0h			
7	6	5	4	3	2	1	0
ENTRY9				ENTRY8			
R/W-0h				R/W-0h			

LEGEND: R/W = Read/Write; *-n* = value after reset

Table 10-188. PDMA_PSILCFG_RX_AASRC_RX_ORDER_TABLE1 Register Field Descriptions

Bit	Field	Type	Reset	Description
31-28	ENTRY15	R/W	Fh	RX FIFO index for slot 15
27-24	ENTRY14	R/W	Eh	RX FIFO index for slot 14
23-20	ENTRY13	R/W	Dh	RX FIFO index for slot 13
19-16	ENTRY12	R/W	Ch	RX FIFO index for slot 12
15-12	ENTRY11	R/W	Bh	RX FIFO index for slot 11
11-8	ENTRY10	R/W	Ah	RX FIFO index for slot 10
7-4	ENTRY9	R/W	9h	RX FIFO index for slot 9
3-0	ENTRY8	R/W	8h	RX FIFO index for slot 8

Copyright © 2024 Texas Instruments Incorporated

10.3.1.3.2.12 PDMA_PSILCFG_RX_RT_ENABLE Register (Offset = 408h) [reset = 0h]

PDMA_PSILCFG_RX_RT_ENABLE is shown in Figure 10-48 and described in Figure 10-48.

Return to Summary Table.

Real Time Enable Register. This register allows enabling various channel settings in real time.

Table 10-189. PDMA_PSILCFG_RX_RT_ENABLE Instances

Instance	Physical Address
PDMA_PSILCFG_RX	N/A

Figure 10-48. PDMA_PSILCFG_RX_RT_ENABLE Register

31	30	29	28	27	26	25	24
ENABLE	TDOWN	PAUSE	RESERVED				
R/W-0h	R/W-0h	R/W-0h	R-0h				
23	22	21	20	19	18	17	16
RESERVED							
R-0h							
15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
RESERVED						IDLE	FREE
R-0h						R-0h	R/W-0h

LEGEND: R = Read Only; R/W = Read/Write; *-n* = value after reset

Table 10-190. PDMA_PSILCFG_RX_RT_ENABLE Register Field Descriptions

Bit	Field	Type	Reset	Description
31	ENABLE	R/W	0h	When set, the RX channel is enabled. When cleared, the RX channel is disabled. When disabled, the channel ignores all DMA events from the peripheral. It maintains proper data transfers with the UDMA-P such that the credit handshake is not disrupted. However, unlike teardown, it does not close out the current open packet. Thus the TDOWN bit should always be used to disable an RX channel instead of manually clearing this bit. Failing to use teardown may result in stale data remaining in the internal RX FIFO, which would also prevent the channel from going idle without a channel reset. Note that this bit cannot be changed from 0 to 1 if the global enable bit in the peer enable register is 0.
30	TDOWN	R/W	0h	When set, the channel will commence a RX channel teardown procedure. It will stop on the next FIFO boundary. It then clears the DMA event count and ignores all future DMA events from the peripheral. After stopping peripheral reads, the PDMA sends a teardown message to the UDMA-P that also closes the current packet with an EOP. If no packet is open at the time of the teardown, the message also includes SOP. The EOP teardown message may or may not contain final packet data. Once the channel teardown is complete and ready to be reused, the ENABLE bit is cleared.

Copyright © 2024 Texas Instruments Incorporated

Table 10-190. PDMA_PSILCFG_RX_RT_ENABLE Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
29	PAUSE	R/W	0h	When set, the channel is in a paused state. It will stop on the next FIFO boundary. It continues to accept and count DMA events from the peripherals but will not act on them. The PAUSE bit can be cleared and data will resume. Pause will not stop teardown from completing.
28-2	RESERVED	R	0h	Reserved
1	IDLE	R	0h	This is a read-only bit that signifies that the paused or disabled channel is also idle. It can only become set if PAUSE is set or ENABLE is cleared.
0	FREE	R/W	0h	When cleared, the channel honors the debug suspend signal. When set, the channel will 'free run', regardless of the value of debug suspend.

Copyright © 2024 Texas Instruments Incorporated

10.3.1.3.2.13 PDMA_PSILCFG_RX_DEBUG_3 Register (Offset = 40Fh) [reset = 0h]

PDMA_PSILCFG_RX_DEBUG_3 is shown in Figure 10-49 and described in Table 10-192.

Return to Summary Table.

Debug/State Register 3. The debug/state registers give software applications additional information about the PDMA than they would need in regular opteration, but which may be useful in debug situations.

Table 10-191. PDMA_PSILCFG_RX_DEBUG_3 Instances

Instance	Physical Address
PDMA_PSILCFG_RX	N/A

Figure 10-49. PDMA_PSILCFG_RX_DEBUG_3 Register

31	30	29	28	27	26	25	24
Z							
R/W-0h							
23	22	21	20	19	18	17	16
Z							
R/W-0h							
15	14	13	12	11	10	9	8
Z							
R/W-0h							
7	6	5	4	3	2	1	0
Z							
R/W-0h							

LEGEND: R = Read Only; R/W = Read/Write; *-n* = value after reset

Table 10-192. PDMA_PSILCFG_RX_DEBUG_3 Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	Z	R/W	0h	This field holds the full width value of the current Z count. In X-Y FIFO mode, this field holds the 1 based FIFO count of the FIFO being currently read, or the number of FIFO completions when the current operation completes. In MCAN mode, this field holds the zero based buffer index of the buffer currently being read, or the number of previously completed buffers.

Copyright © 2024 Texas Instruments Incorporated

10.3.2 PDMA Sources

10.3.2.1 MCU Domain PDMA Event Maps

10.3.2.1.1 MCU_PDMA_MISC_G0 Event Map

Table 10-193. MCU_PDMA_MISC_G0 Tx Channel Allocation

Tx PDMA Channel	PDMA Input	Source Event	Channel Type	Trigger Type	Data FIFO Address
8000	SPI_MCU_0_TX_0	MCU_MCSPI0_DMA_WRITE_EVENT_0	XY	Edge	4030 0138h
8001	SPI_MCU_0_TX_1	MCU_MCSPI0_DMA_WRITE_EVENT_1	XY	Edge	4030 014Ch
8002	SPI_MCU_0_TX_2	MCU_MCSPI0_DMA_WRITE_EVENT_2	XY	Edge	4030 0160h
8003	SPI_MCU_0_TX_3	MCU_MCSPI0_DMA_WRITE_EVENT_3	XY	Edge	4030 0174h
8004	MCANSS_MCU_0_TX_0	MCU_MCAN0_MCANSS_TX_DMA_0	MCAN	Pulse	4050 0000h
8005	MCANSS_MCU_0_TX_1	MCU_MCAN0_MCANSS_TX_DMA_1	MCAN	Pulse	4050 0048h
8006	MCANSS_MCU_0_TX_2	MCU_MCAN0_MCANSS_TX_DMA_2	MCAN	Pulse	4050 0090h

Table 10-194. MCU_PDMA_MISC_G0 Rx Channel Allocation

Rx PDMA Channel	PDMA Input	Source Event	Channel Type	Trigger Type	Data FIFO Address
0	SPI_MCU_0_RX_0	MCU_MCSPI0_DMA_READ_EVENT_0	XY	Edge	4030 013Ch
1	SPI_MCU_0_RX_1	MCU_MCSPI0_DMA_READ_EVENT_1	XY	Edge	4030 0150h
2	SPI_MCU_0_RX_2	MCU_MCSPI0_DMA_READ_EVENT_2	XY	Edge	4030 0164h
3	SPI_MCU_0_RX_3	MCU_MCSPI0_DMA_READ_EVENT_3	XY	Edge	4030 0178h
4	MCANSS_MCU_0_FE_0	MCU_MCAN0_MCANSS_FE_0	MCAN	Pulse	4050 0900h
5	MCANSS_MCU_0_FE_1	MCU_MCAN0_MCANSS_FE_1	MCAN	Pulse	4050 0990h
6	MCANSS_MCU_0_FE_2	MCU_MCAN0_MCANSS_FE_2	MCAN	Pulse	4050 0A20h

Copyright © 2024 Texas Instruments Incorporated

10.3.2.1.2 MCU_PDMA_MISC_G1 Event Map

Table 10-195. MCU_PDMA_MISC_G1 Tx Channel Allocation

Tx PDMA Channel	PDMA Input	Source Event	Function	Trigger Type	Data FIFO Address
8000	SPI_MCU_1_TX_0	MCU_MCSPI1_DMA_WRITE_EVENT_0	XY	Edge	4031 0138h
8001	SPI_MCU_1_TX_1	MCU_MCSPI1_DMA_WRITE_EVENT_1	XY	Edge	4031 014Ch
8002	SPI_MCU_1_TX_2	MCU_MCSPI1_DMA_WRITE_EVENT_2	XY	Edge	4031 0160h
8003	SPI_MCU_1_TX_3	MCU_MCSPI1_DMA_WRITE_EVENT_3	XY	Edge	4031 0174h
8004	SPI_MCU_2_TX_0	MCU_MCSPI1_DMA_WRITE_EVENT_0	XY	Edge	4032 0138h
8005	SPI_MCU_2_TX_1	MCU_MCSPI1_DMA_WRITE_EVENT_1	XY	Edge	4032 014Ch
8006	SPI_MCU_2_TX_2	MCU_MCSPI1_DMA_WRITE_EVENT_2	XY	Edge	4032 0160h
8007	SPI_MCU_2_TX_3	MCU_MCSPI1_DMA_WRITE_EVENT_3	XY	Edge	4032 0174h

Table 10-196. MCU_PDMA_MISC_G1 Rx Channel Allocation

Rx PDMA Channel	PDMA Input	Source Event	Channel Type	Trigger Type	Data FIFO Address
0	SPI_MCU_1_RX_0	MCU_MCSPI1_DMA_READ_EVENT_0	XY	Edge	4031 013Ch
1	SPI_MCU_1_RX_1	MCU_MCSPI1_DMA_READ_EVENT_1	XY	Edge	4031 0150h
2	SPI_MCU_1_RX_2	MCU_MCSPI1_DMA_READ_EVENT_2	XY	Edge	4031 0164h
3	SPI_MCU_1_RX_3	MCU_MCSPI1_DMA_READ_EVENT_3	XY	Edge	4031 0178h
4	SPI_MCU_2_RX_0	MCU_MCSPI1_DMA_READ_EVENT_0	XY	Edge	4032 013Ch
5	SPI_MCU_2_RX_1	MCU_MCSPI1_DMA_READ_EVENT_1	XY	Edge	4032 0150h
6	SPI_MCU_2_RX_2	MCU_MCSPI1_DMA_READ_EVENT_2	XY	Edge	4032 0164h
7	SPI_MCU_2_RX_3	MCU_MCSPI1_DMA_READ_EVENT_3	XY	Edge	4032 0178h

Copyright © 2024 Texas Instruments Incorporated

10.3.2.1.3 MCU_PDMA_MISC_G2 Event Map

Table 10-197. MCU_PDMA_MISC_G2 Tx Channel Allocation

Tx PDMA Channel	PDMA Input	Source Event	Channel Type	Trigger Type	Data FIFO Address
8000	USART_MCU_0_TX_0	MCU_UART0_USART_DMA_0	XY	Edge	40A0 0000h
8001	MCANSS_MCU_1_TX_0	MCU_MCAN1_MCANSS_TX_DMA_0	MCAN	Pulse	4054 0000h
8002	MCANSS_MCU_1_TX_1	MCU_MCAN1_MCANSS_TX_DMA_1	MCAN	Pulse	4054 0048h
8003	MCANSS_MCU_1_TX_2	MCU_MCAN1_MCANSS_TX_DMA_2	MCAN	Pulse	4054 0090h

Table 10-198. MCU_PDMA_MISC_G2 Rx Channel Allocation

Rx PDMA Channel	PDMA Input	Source Event	Channel Type	Trigger Type	Data FIFO Address
0	USART_MCU_0_RX_0	MCU_UART0_USART_DMA_1	XY	Edge	40A0 0000h
1	MCANSS_MCU_1_FE_0	MCU_MCAN1_MCANSS_FE_0	MCAN	Pulse	4054 0900h
2	MCANSS_MCU_1_FE_1	MCU_MCAN1_MCANSS_FE_1	MCAN	Pulse	4054 0990h
3	MCANSS_MCU_1_FE_2	MCU_MCAN1_MCANSS_FE_2	MCAN	Pulse	4054 0A20h

Copyright © 2024 Texas Instruments Incorporated

10.3.2.1.4 MCU_PDMA_ADC Event Map

Table 10-199. MCU_PDMA_ADC Rx Channel Allocation

Rx PDMA Channel	PDMA Input	Source Event	Channel Type	Trigger Type	Data FIFO Address
0	ADC12_MCU_0_RX_0	MCU_ADC0_FIFO0_PULSE_0	XY	Pulse	4020 8100h
1	ADC12_MCU_0_RX_1	MCU_ADC0_FIFO1_PULSE_0	XY	Pulse	4020 8200h
2	ADC12_MCU_1_RX_0	MCU_ADC1_FIFO0_LEVEL_0	XY	Level	4021 8100h
3	ADC12_MCU_1_RX_1	MCU_ADC1_FIFO1_LEVEL_0	XY	Level	4021 8200h

Copyright © 2024 Texas Instruments Incorporated

10.3.2.2 MAIN Domain PDMA Event Maps

Copyright © 2024 Texas Instruments Incorporated

10.3.2.2.1 PDMA_MCAN Event Map

Table 10-200. PDMA_MCAN Tx Channel Allocation

Tx PDMA Channel	PDMA Input	Source Event	Channel Type	Trigger Type	Data FIFO Address
8000	MCANSS_MAIN_0_TX_0	MCAN0_MCANSS_TX_DMA_0	MCAN	Pulse	0270 8000h
8001	MCANSS_MAIN_0_TX_1	MCAN0_MCANSS_TX_DMA_1	MCAN	Pulse	0270 8048h
8002	MCANSS_MAIN_0_TX_2	MCAN0_MCANSS_TX_DMA_2	MCAN	Pulse	0270 8090h
8003	MCANSS_MAIN_1_TX_0	MCAN1_MCANSS_TX_DMA_0	MCAN	Pulse	0271 8000h
8004	MCANSS_MAIN_1_TX_1	MCAN1_MCANSS_TX_DMA_1	MCAN	Pulse	0271 8048h
8005	MCANSS_MAIN_1_TX_2	MCAN1_MCANSS_TX_DMA_2	MCAN	Pulse	0271 8090h
8006	MCANSS_MAIN_2_TX_0	MCAN2_MCANSS_TX_DMA_0	MCAN	Pulse	0272 8000h
8007	MCANSS_MAIN_2_TX_1	MCAN2_MCANSS_TX_DMA_1	MCAN	Pulse	0272 8048h
8008	MCANSS_MAIN_2_TX_2	MCAN2_MCANSS_TX_DMA_2	MCAN	Pulse	0272 8090h
8009	MCANSS_MAIN_3_TX_0	MCAN3_MCANSS_TX_DMA_0	MCAN	Pulse	0273 8000h
800A	MCANSS_MAIN_3_TX_1	MCAN3_MCANSS_TX_DMA_1	MCAN	Pulse	0273 8048h
800B	MCANSS_MAIN_3_TX_2	MCAN3_MCANSS_TX_DMA_2	MCAN	Pulse	0273 8090h
800C	MCANSS_MAIN_4_TX_0	MCAN4_MCANSS_TX_DMA_0	MCAN	Pulse	0274 8000h
800D	MCANSS_MAIN_4_TX_1	MCAN4_MCANSS_TX_DMA_1	MCAN	Pulse	0274 8048h
800E	MCANSS_MAIN_4_TX_2	MCAN4_MCANSS_TX_DMA_2	MCAN	Pulse	0274 8090h
800F	MCANSS_MAIN_5_TX_0	MCAN5_MCANSS_TX_DMA_0	MCAN	Pulse	0275 8000h
8010	MCANSS_MAIN_5_TX_1	MCAN5_MCANSS_TX_DMA_1	MCAN	Pulse	0275 8048h
8011	MCANSS_MAIN_5_TX_2	MCAN5_MCANSS_TX_DMA_2	MCAN	Pulse	0275 8090h
8012	MCANSS_MAIN_6_TX_0	MCAN6_MCANSS_TX_DMA_0	MCAN	Pulse	0276 8000h
8013	MCANSS_MAIN_6_TX_1	MCAN6_MCANSS_TX_DMA_1	MCAN	Pulse	0276 8048h
8014	MCANSS_MAIN_6_TX_2	MCAN6_MCANSS_TX_DMA_2	MCAN	Pulse	0276 8090h
8015	MCANSS_MAIN_7_TX_0	MCAN7_MCANSS_TX_DMA_0	MCAN	Pulse	0277 8000h
8016	MCANSS_MAIN_7_TX_1	MCAN7_MCANSS_TX_DMA_1	MCAN	Pulse	0277 8048h
8017	MCANSS_MAIN_7_TX_2	MCAN7_MCANSS_TX_DMA_2	MCAN	Pulse	0277 8090h
8018	MCANSS_MAIN_8_TX_0	MCAN8_MCANSS_TX_DMA_0	MCAN	Pulse	0278 8000h
8019	MCANSS_MAIN_8_TX_1	MCAN8_MCANSS_TX_DMA_1	MCAN	Pulse	0278 8048h
801A	MCANSS_MAIN_8_TX_2	MCAN8_MCANSS_TX_DMA_2	MCAN	Pulse	0278 8090h
801B	MCANSS_MAIN_9_TX_0	MCAN9_MCANSS_TX_DMA_0	MCAN	Pulse	0279 8000h
801C	MCANSS_MAIN_9_TX_1	MCAN9_MCANSS_TX_DMA_1	MCAN	Pulse	0279 8048h
801D	MCANSS_MAIN_9_TX_2	MCAN9_MCANSS_TX_DMA_2	MCAN	Pulse	0279 8090h
801E	MCANSS_MAIN_10_TX_0	MCAN10_MCANSS_TX_DMA_0	MCAN	Pulse	027A 8000h
801F	MCANSS_MAIN_10_TX_1	MCAN10_MCANSS_TX_DMA_1	MCAN	Pulse	027A 8048h
8020	MCANSS_MAIN_10_TX_2	MCAN10_MCANSS_TX_DMA_2	MCAN	Pulse	027A 8090h
8021	MCANSS_MAIN_11_TX_0	MCAN11_MCANSS_TX_DMA_0	MCAN	Pulse	027B 8000h
8022	MCANSS_MAIN_11_TX_1	MCAN11_MCANSS_TX_DMA_1	MCAN	Pulse	027B 8048h
8023	MCANSS_MAIN_11_TX_2	MCAN11_MCANSS_TX_DMA_2	MCAN	Pulse	027B 8090h
8024	MCANSS_MAIN_12_TX_0	MCAN12_MCANSS_TX_DMA_0	MCAN	Pulse	027C 8000h
8025	MCANSS_MAIN_12_TX_1	MCAN12_MCANSS_TX_DMA_1	MCAN	Pulse	027C 8048h
8026	MCANSS_MAIN_12_TX_2	MCAN12_MCANSS_TX_DMA_2	MCAN	Pulse	027C 8090h
8027	MCANSS_MAIN_13_TX_0	MCAN13_MCANSS_TX_DMA_0	MCAN	Pulse	027D 8000h
8028	MCANSS_MAIN_13_TX_1	MCAN13_MCANSS_TX_DMA_1	MCAN	Pulse	027D 8048h
8029	MCANSS_MAIN_13_TX_2	MCAN13_MCANSS_TX_DMA_2	MCAN	Pulse	027D 8090h
802A	MCANSS_MAIN_14_TX_0	MCAN14_MCANSS_TX_DMA_0	MCAN	Pulse	0268 8000h
802B	MCANSS_MAIN_14_TX_1	MCAN14_MCANSS_TX_DMA_1	MCAN	Pulse	0268 8048h

Copyright © 2024 Texas Instruments Incorporated

Table 10-200. PDMA_MCAN Tx Channel Allocation (continued)

Tx PDMA Channel	PDMA Input	Source Event	Channel Type	Trigger Type	Data FIFO Address
802C	MCANSS_MAIN_14_TX_2	MCAN14_MCANSS_TX_DMA_2	MCAN	Pulse	0268 8090h
802D	MCANSS_MAIN_15_TX_0	MCAN15_MCANSS_TX_DMA_0	MCAN	Pulse	0269 8000h
802E	MCANSS_MAIN_15_TX_1	MCAN15_MCANSS_TX_DMA_1	MCAN	Pulse	0269 8048h
802F	MCANSS_MAIN_15_TX_2	MCAN15_MCANSS_TX_DMA_2	MCAN	Pulse	0269 8090h
8030	MCANSS_MAIN_16_TX_0	MCAN16_MCANSS_TX_DMA_0	MCAN	Pulse	026A 8000h
8031	MCANSS_MAIN_16_TX_1	MCAN16_MCANSS_TX_DMA_1	MCAN	Pulse	026A 8048h
8032	MCANSS_MAIN_16_TX_2	MCAN16_MCANSS_TX_DMA_2	MCAN	Pulse	026A 8090h
8033	MCANSS_MAIN_17_TX_0	MCAN17_MCANSS_TX_DMA_0	MCAN	Pulse	026B 8000h
8034	MCANSS_MAIN_17_TX_1	MCAN17_MCANSS_TX_DMA_1	MCAN	Pulse	026B 8048h
8035	MCANSS_MAIN_17_TX_2	MCAN17_MCANSS_TX_DMA_2	MCAN	Pulse	026B 8090h

Table 10-201. PDMA_MCAN Rx Channel Allocation

Rx PDMA Channel	PDMA Input	Source Event	Channel Type	Trigger Type	Data FIFO Address
0	MCANSS_MAIN_0_FE_0	MCAN0_MCANSS_FE_0	MCAN	Pulse	0270 8900h
1	MCANSS_MAIN_0_FE_1	MCAN0_MCANSS_FE_1	MCAN	Pulse	0270 8990h
2	MCANSS_MAIN_0_FE_2	MCAN0_MCANSS_FE_2	MCAN	Pulse	0270 8A20h
3	MCANSS_MAIN_1_FE_0	MCAN1_MCANSS_FE_0	MCAN	Pulse	0271 8900h
4	MCANSS_MAIN_1_FE_1	MCAN1_MCANSS_FE_1	MCAN	Pulse	0271 8990h
5	MCANSS_MAIN_1_FE_2	MCAN1_MCANSS_FE_2	MCAN	Pulse	0271 8A20h
6	MCANSS_MAIN_2_FE_0	MCAN2_MCANSS_FE_0	MCAN	Pulse	0272 8900h
7	MCANSS_MAIN_2_FE_1	MCAN2_MCANSS_FE_1	MCAN	Pulse	0272 8990h
8	MCANSS_MAIN_2_FE_2	MCAN2_MCANSS_FE_2	MCAN	Pulse	0272 8A20h
9	MCANSS_MAIN_3_FE_0	MCAN3_MCANSS_FE_0	MCAN	Pulse	0273 8900h
10	MCANSS_MAIN_3_FE_1	MCAN3_MCANSS_FE_1	MCAN	Pulse	0273 8990h
11	MCANSS_MAIN_3_FE_2	MCAN3_MCANSS_FE_2	MCAN	Pulse	0273 8A20h
12	MCANSS_MAIN_4_FE_0	MCAN4_MCANSS_FE_0	MCAN	Pulse	0274 8900h
13	MCANSS_MAIN_4_FE_1	MCAN4_MCANSS_FE_1	MCAN	Pulse	0274 8990h
14	MCANSS_MAIN_4_FE_2	MCAN4_MCANSS_FE_2	MCAN	Pulse	0274 8A20h
15	MCANSS_MAIN_5_FE_0	MCAN5_MCANSS_FE_0	MCAN	Pulse	0275 8900h
16	MCANSS_MAIN_5_FE_1	MCAN5_MCANSS_FE_1	MCAN	Pulse	0275 8990h
17	MCANSS_MAIN_5_FE_2	MCAN5_MCANSS_FE_2	MCAN	Pulse	0275 8A20h
18	MCANSS_MAIN_6_FE_0	MCAN6_MCANSS_FE_0	MCAN	Pulse	0276 8900h
19	MCANSS_MAIN_6_FE_1	MCAN6_MCANSS_FE_1	MCAN	Pulse	0276 8990h
20	MCANSS_MAIN_6_FE_2	MCAN6_MCANSS_FE_2	MCAN	Pulse	0276 8A20h
21	MCANSS_MAIN_7_FE_0	MCAN7_MCANSS_FE_0	MCAN	Pulse	0277 8900h
22	MCANSS_MAIN_7_FE_1	MCAN7_MCANSS_FE_1	MCAN	Pulse	0277 8990h
23	MCANSS_MAIN_7_FE_2	MCAN7_MCANSS_FE_2	MCAN	Pulse	0277 8A20h
24	MCANSS_MAIN_8_FE_0	MCAN8_MCANSS_FE_0	MCAN	Pulse	0278 8900h
25	MCANSS_MAIN_8_FE_1	MCAN8_MCANSS_FE_1	MCAN	Pulse	0278 8990h
26	MCANSS_MAIN_8_FE_2	MCAN8_MCANSS_FE_2	MCAN	Pulse	0278 8A20h
27	MCANSS_MAIN_9_FE_0	MCAN9_MCANSS_FE_0	MCAN	Pulse	0279 8900h
28	MCANSS_MAIN_9_FE_1	MCAN9_MCANSS_FE_1	MCAN	Pulse	0279 8990h
29	MCANSS_MAIN_9_FE_2	MCAN9_MCANSS_FE_2	MCAN	Pulse	0279 8A20h
30	MCANSS_MAIN_10_FE_0	MCAN10_MCANSS_FE_0	MCAN	Pulse	027A 8900h
31	MCANSS_MAIN_10_FE_1	MCAN10_MCANSS_FE_1	MCAN	Pulse	027A 8990h
32	MCANSS_MAIN_10_FE_2	MCAN10_MCANSS_FE_2	MCAN	Pulse	027A 8A20h

Copyright © 2024 Texas Instruments Incorporated

Table 10-201. PDMA_MCAN Rx Channel Allocation (continued)

Rx PDMA Channel	PDMA Input	Source Event	Channel Type	Trigger Type	Data FIFO Address
33	MCANSS_MAIN_11_FE_0	MCAN11_MCANSS_FE_0	MCAN	Pulse	027B 8900h
34	MCANSS_MAIN_11_FE_1	MCAN11_MCANSS_FE_1	MCAN	Pulse	027B 8990h
35	MCANSS_MAIN_11_FE_2	MCAN11_MCANSS_FE_2	MCAN	Pulse	027B 8A20h
36	MCANSS_MAIN_12_FE_0	MCAN12_MCANSS_FE_0	MCAN	Pulse	027C 8900h
37	MCANSS_MAIN_12_FE_1	MCAN12_MCANSS_FE_1	MCAN	Pulse	027C 8990h
38	MCANSS_MAIN_12_FE_2	MCAN12_MCANSS_FE_2	MCAN	Pulse	027C 8A20h
39	MCANSS_MAIN_13_FE_0	MCAN13_MCANSS_FE_0	MCAN	Pulse	027D 8900h
40	MCANSS_MAIN_13_FE_1	MCAN13_MCANSS_FE_1	MCAN	Pulse	027D 8990h
41	MCANSS_MAIN_13_FE_2	MCAN13_MCANSS_FE_2	MCAN	Pulse	027D 8A20h
42	MCANSS_MAIN_14_FE_0	MCAN14_MCANSS_FE_0	MCAN	Pulse	0268 8900h
43	MCANSS_MAIN_14_FE_1	MCAN14_MCANSS_FE_1	MCAN	Pulse	0268 8990h
44	MCANSS_MAIN_14_FE_2	MCAN14_MCANSS_FE_2	MCAN	Pulse	0268 8A20h
45	MCANSS_MAIN_15_FE_0	MCAN15_MCANSS_FE_0	MCAN	Pulse	0269 8900h
46	MCANSS_MAIN_15_FE_1	MCAN15_MCANSS_FE_1	MCAN	Pulse	0269 8990h
47	MCANSS_MAIN_15_FE_2	MCAN15_MCANSS_FE_2	MCAN	Pulse	0269 8A20h
48	MCANSS_MAIN_16_FE_0	MCAN16_MCANSS_FE_0	MCAN	Pulse	026A 8900h
49	MCANSS_MAIN_16_FE_1	MCAN16_MCANSS_FE_1	MCAN	Pulse	026A 8990h
50	MCANSS_MAIN_16_FE_2	MCAN16_MCANSS_FE_2	MCAN	Pulse	026A 8A20h
51	MCANSS_MAIN_17_FE_0	MCAN17_MCANSS_FE_0	MCAN	Pulse	026B 8900h
52	MCANSS_MAIN_17_FE_1	MCAN17_MCANSS_FE_1	MCAN	Pulse	026B 8990h
53	MCANSS_MAIN_17_FE_2	MCAN17_MCANSS_FE_2	MCAN	Pulse	026B 8A20h

Copyright © 2024 Texas Instruments Incorporated

10.3.2.2.2 PDMA_MCASP_G0 Event Map

Table 10-202. PDMA_MCASP_G0 Tx Channel Allocation

Tx PDMA Channel	PDMA Input	Source Event	Channel Type	Trigger Type	Data FIFO Address
8000	MCASP_MAIN_0_TX_0	MCASP0_XMIT_DMA_EVENT_REQ_0	XY	Edge	02B0 8000h
8001	MCASP_MAIN_1_TX_0	MCASP1_XMIT_DMA_EVENT_REQ_0	XY	Edge	02B1 8000h
8002	MCASP_MAIN_2_TX_0	MCASP2_XMIT_DMA_EVENT_REQ_0	XY	Edge	02B2 8000h

Table 10-203. PDMA_MCASP_G0 Rx Channel Allocation

Rx PDMA Channel	PDMA Input	Source Event	Channel Type	Trigger Type	Data FIFO Address
0	MCASP_MAIN_0_RX_0	MCASP0_REC_DMA_EVENT_REQ_0	XY	Edge	02B0 8000h
1	MCASP_MAIN_1_RX_0	MCASP1_REC_DMA_EVENT_REQ_0	XY	Edge	02B1 8000h
2	MCASP_MAIN_2_RX_0	MCASP2_REC_DMA_EVENT_REQ_0	XY	Edge	02B2 8000h

Copyright © 2024 Texas Instruments Incorporated

10.3.2.2.3 PDMA_SPI_G0 Event Map

Table 10-204. PDMA_SPI_G0 Tx Channel Allocation

Tx PDMA Channel	PDMA Input	Source Event	Channel Type	Trigger Type	Data FIFO Address
8000	SPI_MAIN_0_TX_0	MCSPI0_DMA_WRITE_EVENT_0	XY	Edge	0210 0138h
8001	SPI_MAIN_0_TX_1	MCSPI0_DMA_WRITE_EVENT_1	XY	Edge	0210 014Ch
8002	SPI_MAIN_0_TX_2	MCSPI0_DMA_WRITE_EVENT_2	XY	Edge	0210 0160h
8003	SPI_MAIN_0_TX_3	MCSPI0_DMA_WRITE_EVENT_3	XY	Edge	0210 0174h
8004	SPI_MAIN_1_TX_0	MCSPI1_DMA_WRITE_EVENT_0	XY	Edge	0211 0138h
8005	SPI_MAIN_1_TX_1	MCSPI1_DMA_WRITE_EVENT_1	XY	Edge	0211 014Ch
8006	SPI_MAIN_1_TX_2	MCSPI1_DMA_WRITE_EVENT_2	XY	Edge	0211 0160h
8007	SPI_MAIN_1_TX_3	MCSPI1_DMA_WRITE_EVENT_3	XY	Edge	0211 0174h
8008	SPI_MAIN_2_TX_0	MCSPI2_DMA_WRITE_EVENT_0	XY	Edge	0212 0138h
8009	SPI_MAIN_2_TX_1	MCSPI2_DMA_WRITE_EVENT_1	XY	Edge	0212 014Ch
800A	SPI_MAIN_2_TX_2	MCSPI2_DMA_WRITE_EVENT_2	XY	Edge	0212 0160h
800B	SPI_MAIN_2_TX_3	MCSPI2_DMA_WRITE_EVENT_3	XY	Edge	0212 0174h
800C	SPI_MAIN_3_TX_0	MCSPI3_DMA_WRITE_EVENT_0	XY	Edge	0213 0138h
800D	SPI_MAIN_3_TX_1	MCSPI3_DMA_WRITE_EVENT_1	XY	Edge	0213 014Ch
800E	SPI_MAIN_3_TX_2	MCSPI3_DMA_WRITE_EVENT_2	XY	Edge	0213 0160h
800F	SPI_MAIN_3_TX_3	MCSPI3_DMA_WRITE_EVENT_3	XY	Edge	0213 0174h

Table 10-205. PDMA_SPI_G0 Rx Channel Allocation

Rx PDMA Channel	PDMA Input	Source Event	Channel Type	Trigger Type	Data FIFO Address
0	SPI_MAIN_0_RX_0	MCSPI0_DMA_READ_EVENT_0	XY	Edge	0210 013Ch
1	SPI_MAIN_0_RX_1	MCSPI0_DMA_READ_EVENT_1	XY	Edge	0210 0150h
2	SPI_MAIN_0_RX_2	MCSPI0_DMA_READ_EVENT_2	XY	Edge	0210 0164h
3	SPI_MAIN_0_RX_3	MCSPI0_DMA_READ_EVENT_3	XY	Edge	0210 0178h
4	SPI_MAIN_1_RX_0	MCSPI1_DMA_READ_EVENT_0	XY	Edge	0211 013Ch
5	SPI_MAIN_1_RX_1	MCSPI1_DMA_READ_EVENT_1	XY	Edge	0211 0150h
6	SPI_MAIN_1_RX_2	MCSPI1_DMA_READ_EVENT_2	XY	Edge	0211 0164h
7	SPI_MAIN_1_RX_3	MCSPI1_DMA_READ_EVENT_3	XY	Edge	0211 0178h
8	SPI_MAIN_2_RX_0	MCSPI2_DMA_READ_EVENT_0	XY	Edge	0212 013Ch
9	SPI_MAIN_2_RX_1	MCSPI2_DMA_READ_EVENT_1	XY	Edge	0212 0150h
10	SPI_MAIN_2_RX_2	MCSPI2_DMA_READ_EVENT_2	XY	Edge	0212 0164h
11	SPI_MAIN_2_RX_3	MCSPI2_DMA_READ_EVENT_3	XY	Edge	0212 0178h
12	SPI_MAIN_3_RX_0	MCSPI3_DMA_READ_EVENT_0	XY	Edge	0213 013Ch
13	SPI_MAIN_3_RX_1	MCSPI3_DMA_READ_EVENT_1	XY	Edge	0213 0150h
14	SPI_MAIN_3_RX_2	MCSPI3_DMA_READ_EVENT_2	XY	Edge	0213 0164h
15	SPI_MAIN_3_RX_3	MCSPI3_DMA_READ_EVENT_3	XY	Edge	0213 0178h

Copyright © 2024 Texas Instruments Incorporated

10.3.2.2.4 PDMA_SPI_G1 Event Map

Table 10-206. PDMA_SPI_G1 Tx Channel Allocation

Tx PDMA Channel	PDMA Input	Source Event	Channel Type	Trigger Type	Data FIFO Address
8000	SPI_MAIN_4_TX_0	MCSPI4_DMA_WRITE_EVENT_0	XY	Edge	0214 0138h
8001	SPI_MAIN_4_TX_1	MCSPI4_DMA_WRITE_EVENT_1	XY	Edge	0214 014Ch
8002	SPI_MAIN_4_TX_2	MCSPI4_DMA_WRITE_EVENT_2	XY	Edge	0214 0160h
8003	SPI_MAIN_4_TX_3	MCSPI4_DMA_WRITE_EVENT_3	XY	Edge	0214 0174h
8004	SPI_MAIN_5_TX_0	MCSPI5_DMA_WRITE_EVENT_0	XY	Edge	0215 0138h
8005	SPI_MAIN_5_TX_1	MCSPI5_DMA_WRITE_EVENT_1	XY	Edge	0215 014Ch
8006	SPI_MAIN_5_TX_2	MCSPI5_DMA_WRITE_EVENT_2	XY	Edge	0215 0160h
8007	SPI_MAIN_5_TX_3	MCSPI5_DMA_WRITE_EVENT_3	XY	Edge	0215 0174h
8008	SPI_MAIN_6_TX_0	MCSPI6_DMA_WRITE_EVENT_0	XY	Edge	0216 0138h
8009	SPI_MAIN_6_TX_1	MCSPI6_DMA_WRITE_EVENT_1	XY	Edge	0216 014Ch
800A	SPI_MAIN_6_TX_2	MCSPI6_DMA_WRITE_EVENT_2	XY	Edge	0216 0160h
800B	SPI_MAIN_6_TX_3	MCSPI6_DMA_WRITE_EVENT_3	XY	Edge	0216 0174h
800C	SPI_MAIN_7_TX_0	MCSPI7_DMA_WRITE_EVENT_0	XY	Edge	0217 0138h
800D	SPI_MAIN_7_TX_1	MCSPI7_DMA_WRITE_EVENT_1	XY	Edge	0217 014Ch
800E	SPI_MAIN_7_TX_2	MCSPI7_DMA_WRITE_EVENT_2	XY	Edge	0217 0160h
800F	SPI_MAIN_7_TX_3	MCSPI7_DMA_WRITE_EVENT_3	XY	Edge	0217 0174h

Table 10-207. PDMA_SPI_G1 Rx Channel Allocation

Rx PDMA Channel	PDMA Input	Source Event	Channel Type	Trigger Type	Data FIFO Address
0	SPI_MAIN_4_RX_0	MCSPI4_DMA_READ_EVENT_0	XY	Edge	0214 013Ch
1	SPI_MAIN_4_RX_1	MCSPI4_DMA_READ_EVENT_1	XY	Edge	0214 0150h
2	SPI_MAIN_4_RX_2	MCSPI4_DMA_READ_EVENT_2	XY	Edge	0214 0164h
3	SPI_MAIN_4_RX_3	MCSPI4_DMA_READ_EVENT_3	XY	Edge	0214 0178h
4	SPI_MAIN_5_RX_0	MCSPI5_DMA_READ_EVENT_0	XY	Edge	0215 013Ch
5	SPI_MAIN_5_RX_1	MCSPI5_DMA_READ_EVENT_1	XY	Edge	0215 0150h
6	SPI_MAIN_5_RX_2	MCSPI5_DMA_READ_EVENT_2	XY	Edge	0215 0164h
7	SPI_MAIN_5_RX_3	MCSPI5_DMA_READ_EVENT_3	XY	Edge	0215 0178h
8	SPI_MAIN_6_RX_0	MCSPI6_DMA_READ_EVENT_0	XY	Edge	0216 013Ch
9	SPI_MAIN_6_RX_1	MCSPI6_DMA_READ_EVENT_1	XY	Edge	0216 0150h
10	SPI_MAIN_6_RX_2	MCSPI6_DMA_READ_EVENT_2	XY	Edge	0216 0164h
11	SPI_MAIN_6_RX_3	MCSPI6_DMA_READ_EVENT_3	XY	Edge	0216 0178h
12	SPI_MAIN_7_RX_0	MCSPI7_DMA_READ_EVENT_0	XY	Edge	0217 013Ch
13	SPI_MAIN_7_RX_1	MCSPI7_DMA_READ_EVENT_1	XY	Edge	0217 0150h
14	SPI_MAIN_7_RX_2	MCSPI7_DMA_READ_EVENT_2	XY	Edge	0217 0164h
15	SPI_MAIN_7_RX_3	MCSPI7_DMA_READ_EVENT_3	XY	Edge	0217 0178h

Copyright © 2024 Texas Instruments Incorporated

10.3.2.2.5 PDMA_USART_G0 Event Map

Table 10-208. PDMA_USART_G0 Tx Channel Allocation

Tx PDMA Channel	PDMA Input	Source Event	Channel Type	Trigger Type	Data FIFO Address
8000	UART_MAIN_0_TX_0	UART0_USART_DMA_0	XY	Edge	0280 0000h
8001	UART_MAIN_1_TX_0	UART1_USART_DMA_0	XY	Edge	0281 0000h

Table 10-209. PDMA_USART_G0 Rx Channel Allocation

Rx PDMA Channel	PDMA Input	Source Event	Channel Type	Trigger Type	Data FIFO Address
0	UART_MAIN_0_RX_0	UART0_USART_DMA_1	XY	Edge	0280 0000h
1	UART_MAIN_1_RX_0	UART1_USART_DMA_1	XY	Edge	0281 0000h

Copyright © 2024 Texas Instruments Incorporated

10.3.2.2.6 PDMA_USART_G1 Event Map

Table 10-210. PDMA_USART_G1 Tx Channel Allocation

Tx PDMA Channel	PDMA Input	Source Event	Channel Type	Trigger Type	Data FIFO Address
8000	UART_MAIN_2_TX_0	UART2_USART_DMA_0	XY	Edge	0282 0000h
8001	UART_MAIN_3_TX_0	UART3_USART_DMA_0	XY	Edge	0283 0000h

Table 10-211. PDMA_USART_G1 Rx Channel Allocation

Rx PDMA Channel	PDMA Input	Source Event	Channel Type	Trigger Type	Data FIFO Address
0	UART_MAIN_2_RX_0	UART2_USART_DMA_1	XY	Edge	0282 0000h
1	UART_MAIN_3_RX_0	UART3_USART_DMA_1	XY	Edge	0283 0000h

Copyright © 2024 Texas Instruments Incorporated

10.3.2.2.7 PDMA_USART_G2 Event Map

Table 10-212. PDMA_USART_G2 Tx Channel Allocation

Tx PDMA Channel	PDMA Input	Source Event	Channel Type	Trigger Type	Data FIFO Address
8000	UART_MAIN_4_TX_0	UART4_USART_DMA_0	XY	Edge	0284 0000h
8001	UART_MAIN_5_TX_0	UART5_USART_DMA_0	XY	Edge	0285 0000h
8002	UART_MAIN_6_TX_0	UART6_USART_DMA_0	XY	Edge	0286 0000h
8003	UART_MAIN_7_TX_0	UART7_USART_DMA_0	XY	Edge	0287 0000h
8004	UART_MAIN_8_TX_0	UART8_USART_DMA_0	XY	Edge	0288 0000h
8005	UART_MAIN_9_TX_0	UART9_USART_DMA_0	XY	Edge	0289 0000h

Table 10-213. PDMA_USART_G2 Rx Channel Allocation

Rx PDMA Channel	PDMA Input	Source Event	Channel Type	Trigger Type	Data FIFO Address
0	UART_MAIN_4_RX_0	UART4_USART_DMA_1	XY	Edge	0284 0000h
1	UART_MAIN_5_RX_0	UART5_USART_DMA_1	XY	Edge	0285 0000h
2	UART_MAIN_6_RX_0	UART6_USART_DMA_1	XY	Edge	0286 0000h
3	UART_MAIN_7_RX_0	UART7_USART_DMA_1	XY	Edge	0287 0000h
4	UART_MAIN_8_RX_0	UART8_USART_DMA_1	XY	Edge	0288 0000h
5	UART_MAIN_9_RX_0	UART9_USART_DMA_1	XY	Edge	0289 0000h

Copyright © 2024 Texas Instruments Incorporated

10.4 Data Routing Unit (DRU)

This section describes the SoC level Data Routing Units (DRUs) for the device.

Note

There are also DRUs in VPAC0 and DMPAC0. Their functionality is almost same as described in this section. For the differences between all DRUs, see Section 10.4.3.8.

10.4.1 DRU Overview

The data routing unit (DRU) is a high bandwidth, flexible routing engine with programmable DMA transfer requests. It enables user to perform high speed data transfers between memory mapped slave endpoints, processor caches and shared caches. DRU behaves like a DMA transfer controller, moving data at CPU frequency.

The DRU supports the following features:

- Programmable configuration registers for direct transfer request submission
  - A dedicated atomic register set that requires entire TR sent in a single burst
  - Three nonatomic register sets that allow TR to be written in separate burst and a write to the SUBMIT_WORD0 register triggers the TR
- Read and write command queues
  - Five different queues with a split arbitration between fixed priority and round robin arbitration
  - Each channel can be configured to go to one of these queues
- Programmable priority for each queue
  - Allows setting priority for commands relative to other masters in the system
- One dedicated read port and one dedicated write port to generate independent read and write commands
  - Generates one read command every cycle
  - Generates one write data phase every cycle
  - Moves the data at CPU frequency
  - Optimizes transfers up to 128 byte boundaries
- Support for triggers to gate levels of the DMA loop
  - Supports one dedicated local trigger
  - Supports two dedicated external global triggers from the PSI-L
  - All triggers can be overwritten and controlled by software instead of hardware based event
- Support for region based and channelized firewall
  - Region based firewall intended to protect DRU configuration registers within a programmatically specified range
  - Channelized firewall that protects:
    - The real time configuration registers for each channel
    - The TR submission registers for each channel
    - The data transfers for each channel
- Data formatting networks
  - Circular addressing support for one side of the transfer
- Independent 48-bit address fields for source and destinations
- Up to four dimensional data transfers
  - Has independent source and destination index for up to four dimensional transfers
- Error Handling
  - Debug support through memory mapped registers
  - Data transfer error reports
- Error detection and Correction
  - Full SEC/DED support for the incoming data
  - Full SEC/DED protection each entry in the TR queues
  - Single bit parity support for each 32-bit word in the data buffer

Copyright © 2024 Texas Instruments Incorporated

  - Parity protection for the address
- PSI-L Interface
  - Receive requests for pre-warm of L2 or L3 Cache
  - Receive TR requests from the UDMA
  - Send response TRs
  - Receive data from a remote UTC
  - Send data to a remote UTC
  - Maintains 2D formatting across PSI-L if requested in TR
- Feature enhancements for DRU R30:
  - IO memory management unit and micro-TLB
    - Features integrated uTLB (Translation Look aside Buffer)
    - Supports virtual addresses and physical addresses in TR requests
    - A read and write uTLB with 2 entries per uTLB
    - Communicates through PSI-L to IOMMU
  - Compression and Decompression

Unsupported features:

- Transpose data between source and destination memory to memory transfers

10.4.1.1 Data Routing Unit (DRU) Ports

Table 10-214. DRU Clocks and Resets

Clocks	
Module Clock Input	**Description**
DRU_CLK	DRU clock
Resets	
Module Reset Input	**Description**
DRU_RST	DRU reset

Table 10-215. DRU Hardware Requests

Interrupt Requests		
Module Interrupt Signal	**Description**	**Type**
DRU_PROTOCOL_ERROR	DRU protocol violation event	Level
DRU_COMPLETE_EVENT[31-0]	DRU completion events	Level
DRU_ERROR_EVENT[31-0]	DRU error completion events	Level
DRU_LOCALOUT_EVENT[31-0]	DRU local output events	Level
DMA Events		
Module DMA Event	**Description**	**Type**
-	-	-

Copyright © 2024 Texas Instruments Incorporated

10.4.2 DRU Integration

This section describes the SoC level DRUs integration in the device, including information about clocks, resets, and hardware requests.

Note

There are also DRUs in VPAC0 and DMPAC0. Their functionality is almost same as the SoC level DRUs. For the differences between all DRUs, see Section 10.4.3.8. For integration details about the VPAC0 and DMPAC0 DRUs, see Section 6.7 *Vision Pre-processing Accelerator (VPAC)* and Section 6.8 *Depth and Motion Perception Accelerator (DMPAC)*.

Copyright © 2024 Texas Instruments Incorporated

10.4.3 DRU Functional Description

Note

This section describes the SoC level Data Routing Units (DRUs) for the device. There are also DRUs in VPAC0 and DMPAC0. Their functionality is almost same as described in this section and its subsections. For the differences between all DRUs, see Section 10.4.3.8.

10.4.3.1 DRU Basic Functionality

Figure 10-50 shows the DRU functional diagram.

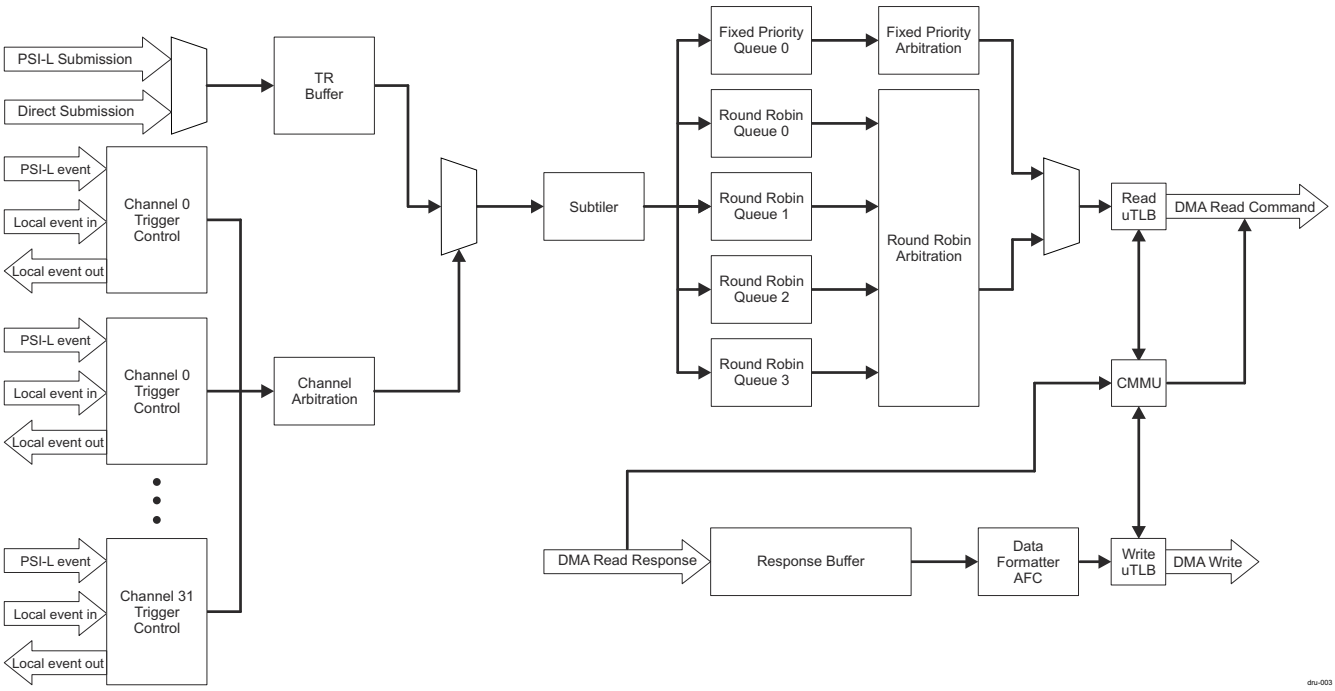


Figure 10-50. DRU Functional Diagram

The DRU receives commands to transfer data from one location to another called Transfer Requests (TRs) or to send messages to CPUs or MMUs to pre-warm logic called Cache Requests (CRs). The DRU can receive these commands through the following mechanisms:

- A direct write to the DRU submission registers
- TR submission over PSI-L from external UDMA-C

The previously described mechanisms of receiving TRs and CRs are also referred to as TR submission (see, Section 10.4.3.1.3). In other words, the DRU memory-to-memory transfers work by receiving TR or CR through TR submission. Then the corresponding request (TR or CR) is placed in a queue to be processed.

Once the TR has been received the Channel Trigger Control block detects if the triggers specified in the TR have been received. If the trigger is not specified (value of 0x0 in the TRIGGER0 or TRIGGER1 fields of the TR FLAGS field), the treatment is as it is always received. Once received the channel is considered active. All active channels are involved in arbitration process for using the subtiler. The channel arbitration is based on the channel queue. There are queues using a fixed priority arbitration - the lowest channel has the highest priority. There are also round robin queues that use the same algorithm as in the fixed priority queue arbitration (lowest channel - highest priority) but on a round robin basis.

The subtiler takes the winning channel (that is, channel with highest priority) and generates the next 1D or 2D transfer (called also sub-TR) based on the trigger size, transfer type and event size specified in the TR. Then the subtiler places that sub-TR into the queue specified for the channel. Once the sub-TR is generated an event control is notified and any active triggers are cleared if the sub-TR crosses the boundary of one of the triggers.

Copyright © 2024 Texas Instruments Incorporated

For example, if a TR has a trigger size for a 3D block and the ICNT2 value is 16, then the active triggers are cleared after 16 sub-TRs are sent to the queue. ICNT2 can be specified through one of the following:

- DRU_ATOMIC_SUBMIT_CURR_TR_WORD4_5_j[47-32] ICNT2 field
- DRU_SUBMIT_WORD4_5_j_k[47-32] ICNT2 field
- ICNT2 field of the UDMA-C TR received over PSI-L

Upon reaching the top of the queue the TR is processed. Based on the information in the TR a read of the specified location and size is performed followed by writing the data to the specified destination. Upon sending the last write of the entire TR loop a completion event is generated. The completion event output also places a TR response in the PSI-L response FIFO in case of UDMA-C TR. For information about the TR response format, see *Transfer Response Record* in *DMA Architecture*.

Note

The channel is an independent data flow that can be given requests for data movement. Each channel operates independently of the others but they all share common resources.

The queue is a shared unit between several channels. It holds the order of the channel data movement requests until the shared data engine can perform the actual data move. A channel is assigned a queue where its data movement requests are placed.

Note

The CHRT_SWTRIG register can only be accessed directly through the DRU configuration interface and is used if software wants to control the triggers for the TR.

Figure 10-51 shows the interaction between software and hardware to setup DRU, submit TRs and then release the channel to be used by a different resource. Section 10.4.3.1.1 through Section 10.4.3.1.5 explain in more detail what occurs in each process or decision.

Copyright © 2024 Texas Instruments Incorporated

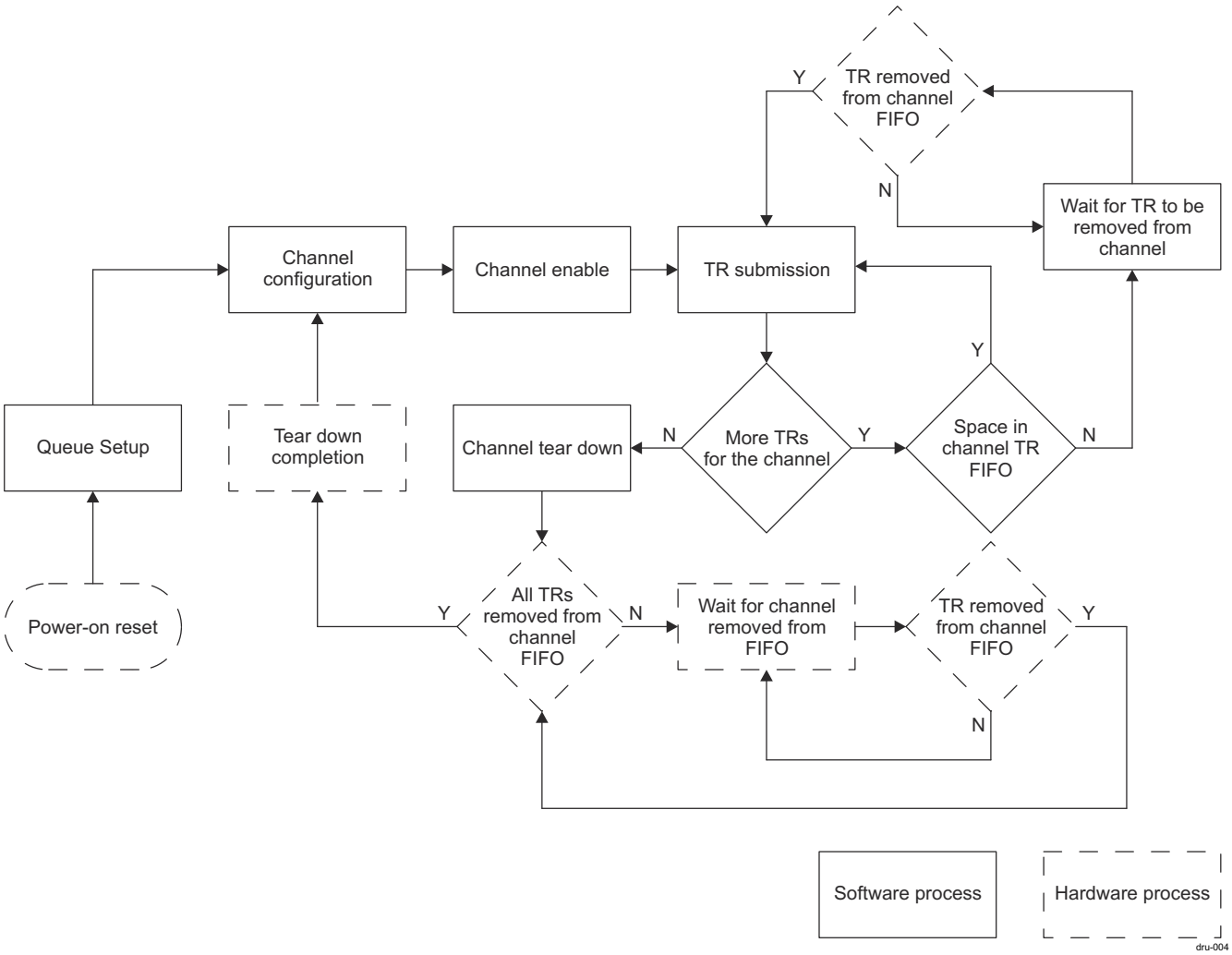


Figure 10-51. DRU Software Managed Flow Diagram

10.4.3.1.1 Queues

Every TR that is active is placed in one of the DRU queues. Each queue operates independently of the others. In a given queue the TRs complete in the order they are placed by the subtiler. Only one read and one write queue may exist at a time. The queues use same resources and need arbitration which is performed based on fixed priority or round robin scheme. In both cases the highest priority queue is the one with the lowest number. The queue priority is specified through the DRU_CFG_y[2-0] PRI field.

Whenever arbitration occurs if there is a TR in a fixed priority queue that TR always goes and the round robin queues are held off. If the fixed priority queues are empty then the round robin queues are arbitrated on a round robin basis, that is, each queue sends its specified number of transactions and then arbitration starts again. On the write side an arbitration ends if the read response has not been received for the next command when the current one completes.

The arbitration happens the last phase of a transfer chunk. A transfer chunk is defined as sending the number of read transactions specified by the DRU_CFG_y[23-16] CONSECUTIVE_TRANS field. After CONSECUTIVE_TRANS number of commands elapse the queue releases the bus and waits for the number of commands specified by the DRU_CFG_y[31-24] REARB_WAIT field before trying to request the bus and involve in arbitration again. Thus software can tune the data flow from each queue to prevent starvation.

If a queue is running and no other queues have a TR then the running one starts its next chunk without stopping. Each time arbitration is performed for round robin queues the queue with the next priority becomes the highest

Copyright © 2024 Texas Instruments Incorporated

priority and the current one the lowest priority queue. If a fixed priority queue is active then the current round robin queue remains the highest priority one until the CONSECUTIVE_TRANS count expires or queue is empty.

The DMA bus QoS and order ID attributes for each queue can also be configured through the DRU_CFG_y[10-8] QOS and DRU_CFG_y[7-4] ORDERID fields.

Note

Queue configuration must be done before setting up any channel as queue changes while channels are active can cause unknown behavior.

10.4.3.1.2 Channel Configuration

The DRU operates on having multiple channels that can run in parallel on the shared hardware but are independently triggered by events as needed. Each channel must be configured before it is used. The channel configuration has the following parts:

- Non-realtime configuration intended to be controlled by a single master in the system for all software processes and processors
- Real time configuration controlled by the software process that owns the channel

10.4.3.1.2.1 Non-realtime Channel Configuration

The non-realtime configuration consists of the following:

- Assigning the channel to a submission mechanism via the DRU_CFG_j[19] CHAN_TYPE_OWNER bit
- Assigning the channel via the DRU_CHST_SCHED_j[2-0] QUEUE field to a hardware queue to be used for channel transactions
- Assigning the channel via the DRU_CHOES0_j[15-0] EVT_NUM field to an event to be used for channel notifications

The DRU_CFG_j[31] PAUSE_ON_ERR bit can be used to control the channel behavior in case of error or exception.

The DRU_CFG_j registers must be written to before the channel is enabled via the DRU_CHRT_CTL_j[31] ENABLE bit as once a channel is enabled any write to the DRU_CFG_j, DRU_CHST_SCHED_j and DRU_CHOES0_j registers will result in write failure and an address error. If a channel does not exist an address error is generated upon write too.

10.4.3.1.2.2 Realtime Channel Configuration

Once the non-realtime channel configuration has been done the DRU_CHRT_CTL_j[31] ENABLE bit should be written. It must be written before any TRs can be written. The mechanism to write this bit is based on the ownership of the channel as specified via the DRU_CFG_j[19] CHAN_TYPE_OWNER bit. If the owner is a CPU, than this bit must be written directly through the DRU configuration interface. If the owner is UDMA, than the PSI-L interface must be used to write the ENABLE bit through the PSI-L register configuration write process as described in *PSI-L*.

Note

Once a channel has been enabled the channel configuration registers cannot be modified. The channel can only be disabled through writing 0x1 to the DRU_CHRT_CTL_j[30] TEARDOWN bit.

10.4.3.1.3 TR Submission

The DRU supports the following TR methods:

- A direct write to the DRU submission registers
- TR submission over PSI-L from external UDMA-C

The TR submission checks the following:

- If a legal TR has been submitted
- If the correct submission method has been used
- If there is space in the channel TR FIFO for the new TR

Copyright © 2024 Texas Instruments Incorporated

10.4.3.1.3.1 Direct TR Submission

The direct TR submission can be atomic or nonatomic. There is one atomic register set that allows for a TR to be submitted with a single burst write. There are also three nonatomic register sets for nonatomic direct TR submission. Each of these three sets allows direct TR submission without requiring submission in a single burst.

The direct atomic TR submission is handled through DRU register writes to the DRU_ATOMIC_SUBMIT_CURR_TR_WORD0_1_j to DRU_ATOMIC_SUBMIT_CURR_TR_WORD14_15_j registers which require write in a single 64-byte burst. Any write other than this size returns an address error. All burst values are checked if they are legal values and space in the channel FIFO is checked too. Then the TR is pushed into the FIFO and the write status is returned with success. If any of the legal TR format checks or FIFO space check fails or if the channel is not owned and enabled the result is an address error returned.

The direct nonatomic TR submission is handled through DRU register writes to the DRU_SUBMIT_WORD0_1_j_k to DRU_SUBMIT_WORD14_15_j_k registers which are intended to be used for cores not being able to perform a large burst write in a single cycle. There is a single set for each core regardless of the channel number. A write to the DRU_SUBMIT_WORD0_1_j_k register triggers the TR.

Note

The following should be taken into account:

- If a channel is full any write to the DRU_SUBMIT_WORD0_1_j_k register results in an address error.
- The DRU does not check anything to ensure that the same core writes to the same nonatomic register set. Software should perform the check procedure.

10.4.3.1.3.2 PSI-L TR Submission

The DRU PSI-L TR submission is used for TR submissions from an external UDMA-C. The DRU has a single submission thread for each channel. The channel must be configured as UDMA-C type channel before PSI-L pairing.

The PSI-L pairing, TR submission and response follow the mechanisms as described in *PSI-L*. For every TR a TR response is sent. If the TR fails a legal check or channel ownership or FIFO fullness check, an immediate TR response is sent back with the cause of the submission error provided in the response.

10.4.3.1.4 TR Removal from Channel

After all requested write responses are received from a TR the DRU sends completion event for the channel that completed the TR. Software can use this event to determine when a TR can be added to a channel FIFO. Each channel has a small FIFO allowing multiple TRs to be preloaded into a channel thus maximizing the channel throughput. The completion event signifies that an entry has become available in the FIFO.

10.4.3.1.5 Channel Tear Down

A channel must be torn down through setting the DRU_CHRT_CTL_j[30] TEARDOWN bit to 0x1. This allows for channel reconfiguration. Setting the TEARDOWN bit to 0x1 forces the channel to stop sending any new requests not already in the pipeline and when all outstanding requests have been received all other bits in the DRU_CHRT_CTL_j register are cleared. If the DRU_CFG_j[31] PAUSE_ON_ERR bit is set and an error occurs the tear down mechanism can be used to restart the channel. The TEARDOWN bit being set to 0x1 also makes the TR STATIC field in any future reads from the TR memory to be set to 0x0 no matter the value when the TR was submitted. This is the mechanism used to stop a static TR.

10.4.3.1.5.1 Tear Down Completion

The tear down completion process is when software has set the DRU_CHRT_CTL_j[30] TEARDOWN bit to 0x1 so no new TRs can be added to the channel but when previous TRs that had been added to the channel are still running. Once the TR FIFO for the channel is empty hardware clears the DRU_CHRT_CTL_j[30] TEARDOWN and DRU_CHRT_CTL_j[31] ENABLE bits. The channel is now back at the channel configuration process as described in Section 10.4.3.1.2.1 and can be reused as needed by the system.

Copyright © 2024 Texas Instruments Incorporated

10.4.3.2 DRU Virtualization

The DRU supports virtual addressing through integrated CMMU along with two uTLBs one for read commands and another for write commands. The CMMU is same as the one associated with C71SS.

The channel configuration adds a bit for ATYPE to choose if the addresses are virtual or physical. If a channel is configured to have virtual addresses the queue sends a request to the uTLB and waits for the status response before it sends the command. If the uTLB returns with a page fault then the TR will fail and all future commands for the TR will be flushed. If the pause on error option is set for the channel then no further TRs will be sent for the channel until it is torn down and setup again. All the attributes for the command are selected based on those returned from the uTLB.

10.4.3.3 DRU Compression and Decompression

The DRU supports compression and decompression of data in line by integrating the Accelerator Framework Compression (AFC) module inside the response buffer path.

The compression logic places the AFC module between the Read Response Buffer and a dedicated write engine for compression and a dedicated write engine of decompression. The read response buffer handles any out of order responses and then the data is pushed into FIFOs inside the AFC module. The AFC module pulls the data out of the FIFOs as 128-bit elements for compression or 128 CDBs for decompression.

Once the 128-bit entry is removed from the FIFO the symbol size specified for the compression is used to split the 128-bit element into the corresponding number of symbols. If the 128-bit does not divide into an even number of symbols then the remain bits are used with the next 128-bit element to create the first symbol. Once each symbol is determined then they are passed into the compression pipeline. The first stage of the compression pipeline is the difference phase. If it is a non-differential compression algorithm then a fixed bias value is subtracted from the symbol. If it is a differential compression algorithm then the symbol is subtracted from a symbol that is from a previous slot. If there was not data at the previous slot selected it will be zero until actual data is present.

The AFC module packs a byte stream into a set of 128-bit CDB that is defined as a super block. At the end of each super block the compression engine will generate a 8-byte write of the CDB information block. These CDB information blocks can then be read back in when decompressing the data to know the size location and decompression information to recreate the original data.

Table 10-216 through Table 10-220 show the templates used to setup the compression and decompression.

Table 10-216. Compression Secondary Transfer Request Template 64 Byte Secondary TR

word 15	word 14	word 13	word 12	
CDB3 Offset from ADDR	CDB3 Format	CDB2 Offset from ADDR	CDB2 Format	
word 11	word 10	word 9	word 8	
CDB1 Offset from ADDR	CDB1 Format	CDB0 Offset from ADDR	CDB0 Format	
word 7	word 6	word 5	word 4	
Max Offset/CDB Table Start Offset	DSTDIM0	SRCDIM0	SB ICNT1	SB ICNT0
word 3	word 2	word 1	word 0	
Compression Flags	SECONDARY TR FLAGS	ADDR of Compressed Data		

Table 10-217. Compression Secondary TR Field Descriptions

Word	Field	Description
0-1	ADDR of Compressed Data	The Location of where the compressed data starts.
2	Secondary TR FLAGS	The Secondary TR Flags that are common for all secondary TRs
3	Compression Flags	The Compression Flags used for the compression
4	SBICNT0 & SBICNT1	The size of the super block for the compression
5	DSTDIM0	The dimension to use for DICNT0 in the original TR after each super block
6	SRCDIM0	The dimension to use for ICNT0 in the original TR after each super block is complete

Copyright © 2024 Texas Instruments Incorporated

Table 10-217. Compression Secondary TR Field Descriptions (continued)

Word	Field	Description
7	Max Offset CDB Table Start Offset	For a compression TR this is the maximum offset value that can be written in the CDB table. This means that compressed data will not be written beyond ADDR + Max Offset. For a decompression TR this is the offset from the secondary TR that the first CDB entry should be read from in bytes. It is a signed value.
8	CDB0 Format	This is the format data for the first CDB table value written after the first super block. It will never be used in an initial fetch.
9	CDB0 Offset	This is where the CDB data for this entry begins relative to the ADDR of Compressed data and is always a positive number.

Table 10-218. Multiple Buffer Interleave FLAGS Field Descriptions

Bit	Field	Description
0-3	SEC_TR_TYPE	0: Multiple Buffer Interleave 1: CDB INFO Block and Decompression/Compression Configuration
4-31	SB_DIM0	The dimension to use between lines of the super block. Lower 4 bits are always 0 since minimum element size is 16 byte element.

Table 10-219. Compression FLAGS Field for Non-Differential Algorithms Descriptions

Bit	Field	Description
0-3	Compression Algorithm	0: Zero Based Algorithm 1: Unsigned Golumb 2: Signed Golumb 3-7: Reserved for Future Use 8-15: Differential Type see the Differential Compression Flags Definition
5-4	SB_AM0	The addressing mode to use for the super block during ICNT0 is decrementing. This is only used if the TR is using circular addressing for the direction of the compression engine.
6-7	SB_AM1	The addressing mode to use for the super block when SB_ICNT1 is decremented. This is only used if the TR is using circular addressing for the direction of the compression engine.
8-15	Bias	The constant value subtracted from the symbol before the compression algorithm is applied
16-31	RSVD	Reserved for future use

Table 10-220. Compression FLAGS Field for Differential Algorithms Descriptions

Bit	Field	Description
0-3	Compression Algorithm	0-7: Non-differential algorithms. See Table 10-219 8: Signed Exponential Golumb with differential logic 9-15: Reserved for Future use
5-4	SB_AM0	The addressing mode to use for the super block during ICNT0 is decrementing. This is only used if the TR is using circular addressing for the direction of the compression engine.
6-7	SB_AM1	The addressing mode to use for the super block when SB_ICNT1 is decremented. This is only used if the TR is using circular addressing for the direction of the compression engine.

Copyright © 2024 Texas Instruments Incorporated

Table 10-220. Compression FLAGS Field for Differential Algorithms Descriptions (continued)

Bit	Field	Description
8-9	Initial k value	The k value to be used for the initial super block. K value for later super blocks will be based upon the optimal k value calculation logic along with the Update K field. This is combined with the symbol size. 8-bit symbol 0: k = 0 1: k = 1 2: k = 2 3: k = 3 12-bit symbol 0: k = 4 1: k = 5 2: k = 6 3 k = 7 16-bit symbol 0: k = 8 1: k = 9 2; k = 10 3: k = 11
11-10	RSVD	Reserved for future use
12-13	Update K Value	Describes how the k value changes from one super block to the next. 0: K is constant and uses the Initial K value specified in the Compression Flags Field 1: K value moves by 1 towards the calculated optimal value. 2: K value moves instantly to the calculated optimal value
14-15	RSVD	Reserved for future use
16-18	Symbol Size	The size of a signal symbol used for compression 0: 8-bit symbol 1: 12-bit symbol 2: 16-bit symbol 3-7: Reserved for Future Use
19	RSVD	Reserved for future use
22-20	Number of Differences	The number of elements to use in the differential logic 0: No subtraction 1: all symbols use SubSel0 field to determine which symbols are used 2: SubSel0 is used for even symbols and SubSel1 is used for the odd symbols 3: repeating pattern of 3 using SubSel0, SubSel1,SubSel2 4: repeating pattern of 4 using SubSel0, SubSel1, SubSel2 and SubSel3 5-7: Reserved for future Use
23	RSVD	Reserved for future use
24-25	SubSel0	Selection for the subtraction module 0: Current symbol uses the adjacent symbol 1: Current symbol uses the symbol 2 spaces away 2: Current symbol uses the symbol 3 spaces away 3: Current symbol uses the symbol 4 spaces away
26-27	SubSel1	Selection for the subtraction module 0: Current symbol uses the adjacent symbol 1: Current symbol uses the symbol 2 spaces away 2: Current symbol uses the symbol 3 spaces away 3: Current symbol uses the symbol 4 spaces away

Copyright © 2024 Texas Instruments Incorporated

Table 10-220. Compression FLAGS Field for Differential Algorithms Descriptions (continued)

Bit	Field	Description
28-29	SubSel2	Selection for the subtraction module 0: Current symbol uses the adjacent symbol 1: Current symbol uses the symbol 2 spaces away 2: Current symbol uses the symbol 3 spaces away 3: Current symbol uses the symbol 4 spaces away
30-31	SubSel3	Selection for the subtraction module 0: Current symbol uses the adjacent symbol 1: Current symbol uses the symbol 2 spaces away 2: Current symbol uses the symbol 3 spaces away 3: Current symbol uses the symbol 4 spaces away

10.4.3.4 DRU Output Events

The DRU generates the following events:

- Local output events
- PSI-L global events
- Protocol violation event
- Completion events
- Error completion events

The EVENT_SIZE field of each TR specifies how often a TR generates an output event. That event is provided on the local output event interface as shown in Figure 10-50 and as global event on the PSI-L interface. There is one event per channel. The global event number used by a PSI-L event transport lane can be configured for each channel through the DRU_CHOES0_j[15-0] EVT_NUM field.

As shown in Table 10-221, if the EVENT_SIZE value is in range from 1 to 3, then the output event is generated when the write command for the given loop level (ICNT1, ICNT2 or ICNT3 decremented) has been sent. If EVENT_SIZE = 0, then the output event is generated when all responses have been received.

Table 10-221. EVENT_SIZE Encoding

EVENT_SIZE Value	Event Encoding	Description
0	TR complete	The TR has completed and all responses have been received
1	ICNT1 loop is decremented	The last write command of a row has been sent
2	ICNT2 loop is decremented	The last write command of an array has been sent
3	ICNT3 loop is decremented	The last write command of an array of an array has been sent

For more information about PSI-L interface and its event transport lane, see *PSI-L*.

For more information about the TR EVENT_SIZE field, see *Transfer Request Record* of *DMA Architecture*.

The protocol violation event is not channel associated.

The completion events are defined for each channel. The completion event fires whenever a channel completes a TR successfully. It is only generated when all responses are received from the write portion of the block copy. A single write completion event can occur in a given cycle since it is based on the write responses being received to generate the response.

The error completion events are defined for each channel. The error completion event fires whenever a channel completes a TR with an error and it has flushed out all data related to that TR in the response buffer. Upon determining that a TR is not able to complete successfully, the DRU enters a flush mode to remove all read data received after the error is determined as well as make sure all write responses are received. Once all of this criteria has been matched the write engine fires a completion event. If multiple error completions occur at the same time the DRU will still send them out one at a time using an arbitration scheme used for the original arbitration for the queue, either fixed priority or round robin based on the queue that has the error.

Copyright © 2024 Texas Instruments Incorporated

10.4.3.5 DRU Address Fetch Algorithm, TR and CR Formats

The address fetch algorithm, TR and CR formats are defined in *Transfer Request Record* in *DMA Architecture*. It documents all features that can be supported by a Universal Transfer Controller (UTC) like the DRU but it may not support all of them. The DRU_CAPABILITIES register must be read to check what features of the TR are supported by the DRU.

The TR format allows to perform some kinds of data reformatting or to alter from the standard addressing formats. The DRU supports the transpose data reformatting function and the circular buffering alternate addressing algorithm.

10.4.3.5.1 Transpose

Linear streams work for large classes of algorithms, but not all. In some cases it is desired to transpose rows to columns and columns to rows in the data structure. To help in these cases the DRU supports a transpose function.

The transpose type works on element sizes of 32, 64 or 128 bits by sending sub-TRs in the size of 32 elements by 32 lines for 32-bit, 16 elements by 16 lines for 64-bit and 8 elements by 8 lines for 128-bit element. The sub-TR that is being sent always results in the maximum transfer size. For the write requests the data in the response buffer always looks as a buffer aligned to the zeroth offset in the address.

10.4.3.5.2 Circular Buffering

Circular addressing modifies how the DRU performs address arithmetic so that addresses remain within a power-of-two sized window. Circular addressing works by holding upper address bits constant during an address update, while allowing the lower address bits to vary. This contrasts with the default linear addressing which allows all of the address bits to vary.

The TR provides address mode selection for each level of loop nest. Each level can select between linear addressing or circular addressing with one of two circular block sizes.

If circular addressing mode is selected in the AMODE field of the TR, the definition of the upper 16 bits of the FMTFLAGS field of the TR select the type of addressing for each entry in the loop and define two unique masks that can be used for the circular buffering. Additionally, the DIR field in the TR determines if the circular addressing flags apply for the source or the destination of the block move. The other direction always follows the standard linear addressing that is used if AMODE is not set.

For more information about circular buffering and the associated TR fields, see *Circular Address Mode Specific Flags* of *DMA Architecture*.

10.4.3.6 DRU Firewalls

Unlike most of the device modules whose associated firewalls reside on system interconnect (CBASS) level the DRU has its own embedded firewalls. They are the following:

- Region based firewall intended to protect DRU configuration registers within a programmatically specified range.
- Channelized firewall that protects:
  - The real time configuration registers for each channel:
    - DRU_CHRT_CTL_j
    - DRU_CHRT_SWTRIG_j
  - The TR submission registers for each channel:
    - DRU_ATOMIC_SUBMIT_CURR_TR_WORD0_1_j to DRU_ATOMIC_SUBMIT_CURR_TR_WORD14_15_j
    - DRU_SUBMIT_WORD0_1_j_k to DRU_SUBMIT_WORD14_15_j_k
  - The data transfers for each channel

The region based firewall should be configured to cover the range from 0x6D00 4000 to 0x6D0C FFFF. This allows for a single master to control the memory attribute channels and channel ownership. If the region based firewall is configured to overlap the channelized firewall then both firewall checks have to pass for the register update to occur. If the firewall check fails the DRU returns protection error status to the requestor.

Copyright © 2024 Texas Instruments Incorporated

Note

Protection error takes precedence over address error. If the address falls into the firewall protected range then a protection error is returned.

Note

Each SoC DRU (DRU0 and DRU1) has both the region based and channelized firewall.

The DRU region based and channelized firewalls are same as the other device firewalls. For more information about their functionality, see *Interconnect Firewalls* in *System Interconnect*. The DRU firewall associated registers are described in the *DRU Registers* sections.

10.4.3.7 DRU Errors

In all error cases the DRU reports the error to the system, ensures that all outstanding commands for the TR have completed and proceeds with the next TR in the queue as if the previous one has finished normally. In most error cases the read transaction completes as normal but the responses are flushed. This ensures that both the read and the write start fresh on the next TR. The error details are logged in the DRU_CHRT_STATUS_DET_j and DRU_CHRT_STATUS_CNT_j registers. The DRU_CAUSE_y registers can be read to check if an error for the corresponding channel has occurred.

10.4.3.8 DRU Configurations

Table 10-222 shows the differences between the SoC level DRUs and the rest DRUs in the device. The rest of the features and functionality is same as described in Section 10.4.1 and Section 10.4.3.

Table 10-222. Differences between DRU Configurations

Feature	SoC Level DRUs	DMPAC_UTC_DRU	VPAC_UTC0_RT_DRU, VPAC_UTC1_NRT_DRU
Cache warm support	Y	Y	N
Atomic direct TR submission	Y	N	N
Nonatomic TR submission register sets	3	0	0
Data reformat functions	Transpose	None	None
Minimum transpose size	32 bits	N/A	N/A
Embedded firewall	Y	Y	N
Event generation	Encoded	Bit vector	Bit vector

11 Time Sync

This chapter describes the time sync modules in the device.

11.1 Time Sync Module (CPTS) 981
11.2 Timer Manager 987
11.3 Time Sync and Compare Events 992

Copyright © 2024 Texas Instruments Incorporated

11.1 Time Sync Module (CPTS)

This chapter describes the Time Synchronization (CPTS) module.

11.1.1 CPTS Overview

The Common Platform Time Sync (CPTS) module is used to facilitate host control of time sync operations.

11.1.1.1 CPTS Features

Main features of CPTS module are:

- Supports the selection of multiple external clock sources
- Software control of time sync events via interrupt or polling
- Supports 8 hardware timestamp push inputs
- Supports timestamp counter compare output
- Supports timestamp counter bit output
- Supports 6 timestamp Generator function outputs
- 32-bit and 64-bit timestamp modes.

11.1.2 CPTS Functional Description

11.1.2.1 CPTS Architecture

Figure 11-1 shows the architecture of the CPTS module.

The CPTS module is used to facilitate host control of time sync operations. The CPTS collects time sync events and then presents them to the host for processing. The types of time sync events are as follows:

- Host Transmit Event
- Ethernet receive event
- Ethernet transmit event
- Time stamp push event
- Time stamp counter rollover event (32-bit mode only)
- Time stamp counter half-rollover event (32-bit mode only)
- Hardware Time Stamp Push Event
- Time Stamp Compare Event

The reference clock used for the time stamp (RCLK) can be derived from several sources.

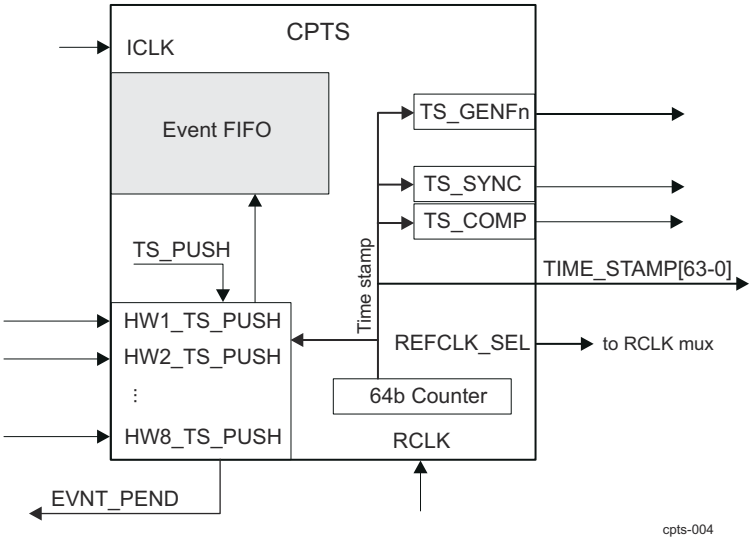


Figure 11-1. CPTS Block Diagram

11.1.2.2 CPTS Initialization

The CPTS module must be configured as follows:

1. Reset the CPTS module

Copyright © 2024 Texas Instruments Incorporated

2. Clear the CPTS_EN bit in the CPTS_CONTROL_REG
3. Write the RFTCLK_SEL value in the CPTS_RFTCLK_SEL_REG with the desired reference clock selection
4. Set the CPTS_EN bit in the CPTS_CONTROL_REG
5. If using interrupts and not polling, enable the interrupt by setting the TS_PEND_EN bit in the CPTS_INT_ENABLE_REG

11.1.2.3 32-bit Time Stamp Value

The time stamp value is a 32-bit value that increments on each RCLK rising edge when CPTS_EN is set to 1. When CPTS_EN is cleared to 0, the time stamp value is reset to 0.

If more than 32-bits of time stamp are required by the application, the host software must maintain the necessary number of upper bits. The upper time stamp value should be incremented by the host when the rollover event is detected.

For test purposes, the time stamp can be written via the time stamp load function (CPTS_TS_LOAD_VAL_REG and CPTS_TS_LOAD_EN_REG).

The TS_INC_VAL[2:0] value must be zero in 32-bit mode. Nudge and PPM adjustments are not supported in 32-bit mode.

11.1.2.4 64-bit Time Stamp Value

The time stamp value is a 64-bit value that increments on each RCLK rising edge when CPTS_EN is set to 1. When CPTS_EN is cleared to 0, the time stamp value is reset to 0.

64-bit mode is selected via CPTS_CONTROL_REG[5] MODE_64BIT bit set to 1.

For test purposes, the time stamp can be written via the time stamp load function (CPTS_TS_LOAD_VAL_REG, CPTS_TS_LOAD_HIGH_VAL_REG, and CPTS_TS_LOAD_EN_REG).

The TS_ADD_VAL feature is included to allow 1-ns timestamp operations with an RCLK rate less than 1 GHz. Table 11-1 shows the RCLK and TS_ADD_VAL values for 1-ns operations. The highest RCLK frequency possible should be used as allowed by the silicon technology.

Table 11-1. TS_ADD_VAL

RCLK (MHz)	TS_ADD_VAL
1000	0
500	1
333.33	2
250	3
200	4
166.66	5
142.85714	6
125	7

11.1.2.4.1 64-Bit Timestamp Nudge

The 64-bit TIME_STAMP value can be adjusted by writing the CPTS_TS_NUDGE_VAL_REG[7:0] register value which is a two's complement value. A value of 0xFF will subtract one RCLK clock from the next incremented TIME_STAMP[63:0] value. A nudge value of 0x01 will add one RCLK clock to the next incremented TIME_STAMP[63:0] value. For example, if the current TIME_STAMP value is 0x0F06, and TS_ADD_VAL[2:0]=3, the next incremented timestamp value would be 0x0F0A without a nudge and 0x0F0A +/- TS_NUDGE[7:0] with a nudge. The TS_NUDGE value is cleared to zero when the nudge has occurred.

11.1.2.4.2 64-bit Timestamp PPM

The 64-bit TIME_STAMP can be adjusted by parts per million (PPM) or by parts per hour (PPH). Writing a non-zero value to the TS_PPM[41:0] value enables PPM operations. The adjustment is up or down depending on the TS_PPM_DIR bit. The TIME_STAMP value is increased by the PPM value when TS_PPM_DIR is cleared and decreased by the PPM value when TS_PPM_DIR is set.

- To adjust for 100 parts per million the configured value for TS_PPM[41:0] is:

Copyright © 2024 Texas Instruments Incorporated

1 000 000/100 = 10 000 (DEC)

- To adjust for 1 part per hour at 1 GHz RCLK, the configured value for TS_PPM[41:0] is:
  (1 000 000 000Hz/1pph) × (3600 seconds/hour) = 346 30B8 A000 (HEX)

11.1.2.5 Event FIFO

All time sync events are push onto the Event FIFO. No overrun indication supported. Software must service the event FIFO in a timely manner to prevent FIFO overrun.

11.1.2.6 Timestamp Compare Output

CPTS features one Time Stamp Compare (TS_COMP) output.

Note

64-bit mode operation is identical to 32-bit mode except that all 64 bits of the TIME_STAMP[63-0] are used instead of only the lower 32 bits.

11.1.2.6.1 Non-Toggle Mode

The TS_COMP output is asserted for CPTS_TS_COMP_LEN_REG[31-0] RCLK periods when the TIME_STAMP[31-0] value compares with the CPTS_TS_COMP_VAL_REG[31-0] and the length value is non-zero. The TS_COMP rising edge occurs three RCLK periods after the values compare. A timestamp compare event is pushed into the event FIFO when TS_COMP is asserted. The polarity of the TS_COMP output is determined by the CPTS_CONTROL_REG[2] TS_POLARITY bit. The output is asserted low when the polarity bit is 0.

11.1.2.6.2 Toggle Mode

The TS_COMP output is asserted for CPTS_TS_COMP_LEN_REG RCLK periods when the TIME_STAMP[31-0] value compares with the CPTS_TS_COMP_VAL_REG[31-0] and the length value is non-zero. The TS_COMP toggles thereafter on CPTS_TS_COMP_LEN_REG[31-0] RCLK periods. The length high or low can be adjusted by writing the CPTS_TS_COMP_NUDGE_REG[7-0] register value which is a two's complement value. A value of 0xFF will subtract one RCLK from the CPTS_TS_COMP_LEN_REG value. A value of 0x01 will add one RCLK to the CPTS_TS_COMP_LEN_REG value. Only a single high or low time is adjusted (nudged) and the CPTS_TS_COMP_NUDGE_REG value is cleared to zero when the nudge has occurred. The TS_COMP output is asserted low when the CPTS_CONTROL_REG[2] TS_COMP_POLARITY bit is 0.

No compare events and no CPTS_EVNT interrupts are generated in toggle mode.

The CPTS_CONTROL_REG[6] TS_COMP_TOG bit must be set for toggle mode, and must be set before writing a non-zero value to CPTS_TS_COMP_LEN_REG.

11.1.2.7 Timestamp Sync Output

The TS_SYNC output is a selected bit of the TIME_STAMP counter value. One of the counter bits 17-31 can be selected in CPTS_CONTROL_REG[31-28] TS_SYNC_SEL. The TS_SYNC output is disabled when CPTS_CONTROL_REG[31-28] TS_SYNC_SEL is zero.

If the selected counter bit is 1 at the time when TS_SYNC_SEL value is written then a rising edge will not occur on the TS_SYNC output. A rising edge will occur on the TS_SYNC output upon the next transition-to-1 of the selected counter bit. The TS_SYNC_SEL value must be written to zero before changing to a different non-zero value. No events are generated due to the TS_SYNC operation. The TS_SYNC output is two RCLK periods after the actual count value.

11.1.2.8 Timestamp GENF Output

There are six TS Generate Function (TS_GENFn) outputs (n = 0 to 5). The TS_GENFn outputs have a programmable cycle (frequency) with a PPM feature and a software nudge feature. The TS_GENFn output cycle is CPTS_LENGTH_REG_j[31:0] RCLK periods (which is different than TS_COMP operation).

The TS_GENFn output cycle is CPTS_LENGTH_REG_j[31:0] RCLK periods beginning when the TIME_STAMP[63:0] value compares with the CPTS_COMP_LOW_REG_j[63:0] and the length value is non-

Copyright © 2024 Texas Instruments Incorporated

zero. The TS_GENFn output cycle repeats thereafter every CPTS_LENGTH_REG_j[31:0] RCLK periods. The upper 32-bit word should be written first for 64-bit values. The length should be zero while the comparison value and other configuration parameters are being configured. The length should be written non-zero to enable operations last. The first cycle after comparison is active high when the TS_GENFn_POLARITY bit is low. No compare events and no CPTS_EVNT interrupts are generated.

11.1.2.8.1 GENFn Nudge

The cycle length can be adjusted by writing the CPTS_NUDGE_REG_j[7:0] register value which is a two's complement value. A value of 0xFF will subtract one RCLK clock from the CPTS_LENGTH_REG_j[31:0] value. A value of 0x01 will add one RCLK clock to the CPTS_LENGTH_REG_j value. The CPTS_NUDGE_REG_j value is cleared to zero when the nudge has occurred.

11.1.2.8.2 GENFn PPM

The TS_GENFn output cycle can be adjusted by parts per million (PPM) or by parts per hour (PPH). Writing a non-zero CPTS_PPM_LOW_REG_j/CPTS_PPM_HIGH_REG_j value enables PPM operations. The PPM counter continually loads and decrements to zero and then loads again. A single RCLK adjustment is made when the PPM counter decrements to zero. The adjustment is up or down depending on the CPTS_CONTROL_REG_j[0] PPM_DIR bit. When PPM_DIR is set a single RCLK time is subtracted from the generate function counter which has the effect of increasing the generate function frequency by the PPM amount. When PPM_DIR is clear a single RCLK time is added to the generate function counter which has the effect of decreasing the generate function frequency by the PPM amount.

- To adjust for 100 parts per million the configured value for TS_GENFn_PPM[41:0] is:
  1 000 000/100 = 10 000 (DEC)
- To adjust for 1 part per hour at 1 GHz RCLK the configured value for TS_GENFn_PPM[41:0] is:
  (1 000 000 000Hz/1pph) × (3600 seconds/hour) = 346 30B8 A000 (HEX)

11.1.2.9 Time Sync Events

Time Sync events are 96-bit or 128-bit (for 32-bit and 64-bit time stamp respectively) values that are pushed onto the event FIFO and read by software in 32-bit reads. Four 32-bit registers, CPTS_EVENT_0_REG throught CPTS_EVENT_3_REG hold the data of a time sync event.

11.1.2.9.1 Time Stamp Push Event

Software can obtain the current time stamp value (at the time of the write) by initiating a time stamp push event. The push event is initiated by setting the TS_PUSH bit of the CPTS_TS_PUSH_REG. The time stamp value is returned in the event, along with a time stamp push event code.

11.1.2.9.2 Time Stamp Counter Rollover Event (32-bit mode only)

The CPTS module contains a 32-bit time stamp value. The counter upper bits are maintained by host software. The rollover event indicates to software that the time stamp counter has rolled over from 0xFFFF FFFF to 0x0000 0000 and the software-maintained upper count value should be incremented.

11.1.2.9.3 Time Stamp Counter Half-rollover Event (32-bit mode only)

The CPTS includes a time stamp counter half-rollover event. The half-rollover event indicates to software that the time stamp value has incremented from 0x7FFF FFFF to 0x8000 0000. The half-rollover event is included to enable software to correct a misaligned event condition. The half-rollover event is included to enable software to determine the correct time for each event that contains a valid time stamp value, such as an Ethernet event. If an Ethernet event occurs around a counter rollover (full rollover), the rollover event could possibly be loaded into the event FIFO before the Ethernet event, even though the Ethernet event time was actually taken before the rollover. Figure 11-2 shows a misalignment condition.

Host software must detect and correct for misaligned event conditions. For every event after a rollover and before a half-rollover, software must examine the time stamp most significant bit. If bit 31 of the time stamp value is low (0x0000 0000 through 0x7FFF FFFF), then the event time stamp was taken after the rollover and no correction is required. If the value is high (0x8000 0000 through 0xFFFF FFFF), the time stamp value was taken before the rollover and a misalignment is detected. The misaligned case indicates to software that it must subtract one from the upper count value stored in software to calculate the correct time for the misaligned event.

Copyright © 2024 Texas Instruments Incorporated

The misaligned event occurs only on the rollover boundary and not on the half-rollover boundary. Software only needs to check for misalignment from a rollover event to a half-rollover event.

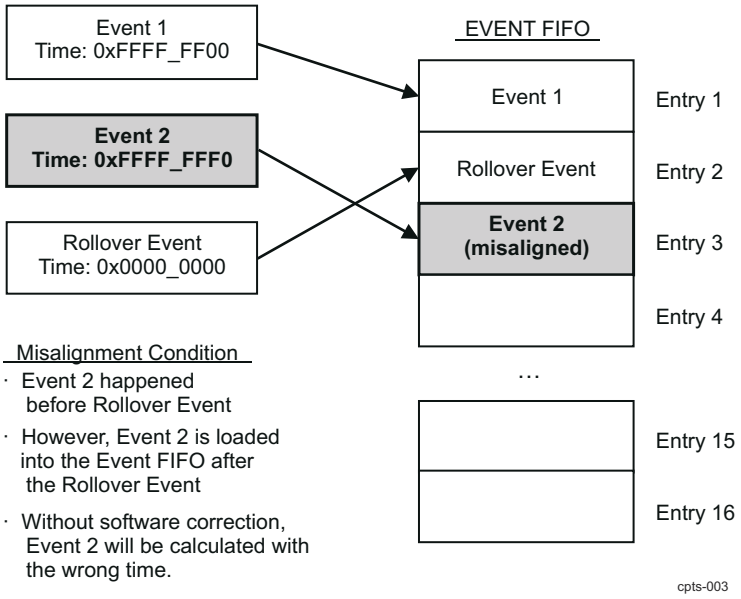


Figure 11-2. Event FIFO Misalignment Condition

11.1.2.9.4 Hardware Time Stamp Push Event

There are eight hardware time stamp inputs (HW[1-8]_TS_PUSH) that can cause hardware time stamp push events to be loaded into the Event FIFO. Each time stamp input is mapped in the device as shown in *CPTS Integration*.

The event is loaded into the event FIFO on the rising edge of the timer, and the PORT_NUMBER field in the CPTS_EVENT_1_REG register indicates the hardware time stamp input that caused the event.

Each hardware time stamp input must be asserted for at least 10 periods of the selected RCLK clock. Each input can be enabled or disabled by setting the respective bits in the CPTS_CONTROL_REG register.

Hardware time stamps are intended to be an extremely low frequency signals, such that the event FIFO does not overrun. Software must keep up with the event FIFO and ensure that there is no overrun, or events will be lost.

11.1.2.10 Timestamp Compare Event

Note

Timestamp compare events are generated for non-toggle mode only.

The CPTS can generate an event for a time stamp comparison in 32-bit or 64-bit mode. The TS_COMP output is also asserted when the event is generated. The event is generated when the TIME_STAMP value compares with the CPTS_TS_COMP_VAL_REG/CPTS_TS_COMP_HIGH_VAL_REG and the CPTS_TS_COMP_LEN_REG value is non-zero. The CPTS_TS_COMP_LEN_REG value should be written by software after the CPTS_TS_COMP_VAL_REG/CPTS_TS_COMP_HIGH_VAL_REG is written and should be zero when the comparison value is written.

11.1.2.11 CPTS Interrupt Handling

When an event is push onto the Event FIFO, an interrupt can be generated to indicate to software that a time sync event occurred. The following steps should be taken to process time sync events using interrupts:

1. Enable the TS_PEND interrupt by setting the TS_PEND_EN bit of the CPTS_INT_ENABLE_REG.
2. Upon interrupt, read the CPTS_EVENT_0_REG through CPTS_EVENT_3_REG register values.
3. Set the CPTS_EVENT_POP_REG[0] EVENT_POP bit to 1 to pop the previously read value off of the event FIFO.

Copyright © 2024 Texas Instruments Incorporated

4. Process the interrupt as required by the application software.

Software has the option of processing more than a single event from the event FIFO in the interrupt service routine in the following way:

1. Enable the TS_PEND interrupt by setting the TS_PEND_EN bit of the CPTS_INT_ENABLE_REG.
2. Upon interrupt, read the CPTS_EVENT_0_REG through CPTS_EVENT_3_REG register values.
3. Set the CPTS_EVENT_POP_REG[0] bit to 1 to pop the previously read value off of the event FIFO.
4. Wait for an amount of time greater than four RCLK periods plus four ICLK periods.
5. Read the TS_PEND_RAW bit in the CPTS_INTSTAT_RAW_REG register to determine if another valid event is in the event FIFO. If it is asserted, go to step 2; otherwise, proceed to step 6.
6. Process the interrupt(s) as required by the application software.

Software also has the option of disabling the interrupt and polling the TS_PEND_RAW bit of the CPTS_INTSTAT_RAW_REG to determine if a valid event is on the event FIFO.

Copyright © 2024 Texas Instruments Incorporated

11.2 Timer Manager

This chapter describes the Timer Manager module of the device.

11.2.1 Timer Manager Overview

The Timer Manager module provides timers for timing operations for the processes running on multiple processors in the device.

11.2.1.1 Timer Manager Features

Timer Manager has the following features:

- 1024 × 32-bit RAM-based independent timers
- Host access to determine which timer(s) expired
- 32 registers with individual timeout status (one bit per timer)
- A pair of quick-read registers with the number of timers that have expired and the ids of the first timers to expire
- A bank timeout register with the banks that have expired timers – to avoid having to read all timeout status registers when only a few timers have expired
- Groups of 16 timers separated into pages of 4-K address space
- Timer bits within each page to read expiration status for each timer when software only has access to that page
- 10 µs time to cycle through all of the timers
- Host access to reset individual timers
  - Reset values are preprogramed for each timer
  - Timers may all be programmed before the timer manager is enabled, or programmed after enabled.

11.2.2 Timer Manager Functional Description

11.2.2.1 Timer Manager Function Overview

Figure 11-3 shows the Timer Manager diagram.

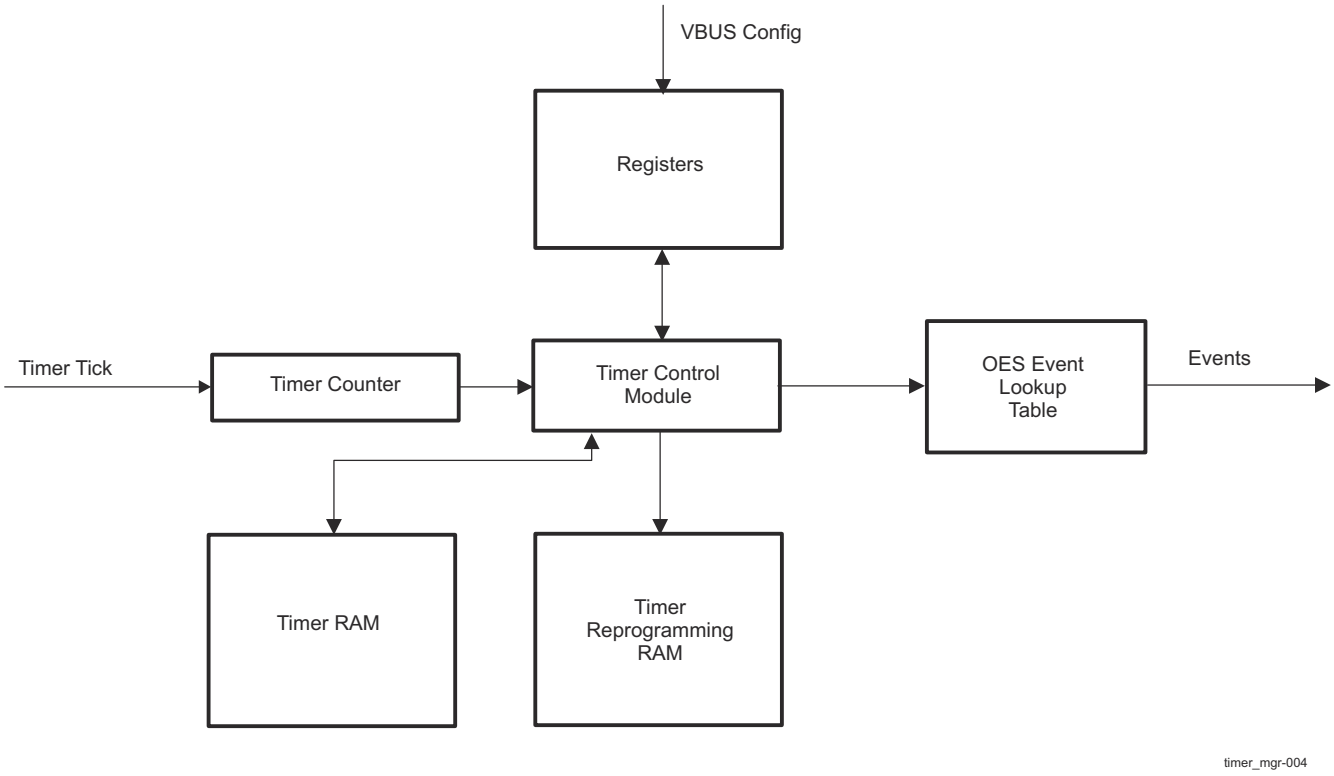


Figure 11-3. Timer Manager Block Diagram

Copyright © 2024 Texas Instruments Incorporated

The 1024 timers in the timer manager are organized as the Timer RAM in Figure 11-3. When the timer manager is enabled, the timer counter will increment with the input timer tick, and the control module continuously loops over the Timer RAM, comparing the values against this timer count. When the timer count is greater than a timer value, the timer control module will generate a timer expiration event.

Based on software writes, the timer control module will update the values in the Timer RAM to setup or cancel active timers. Whenever a new value is written to a TIMERMGR_SETUP_j_k memory address, the corresponding timer is touched/setup with the current sum of the timer count and the written value; written values are relative to the current timer count. These values are stored in the Timer Reprogramming RAM, so the software may also write to the TIMERMGR_CONTROL_j_k registers to flag a timer for reprogramming, using the previously stored reprogramming value. The latter method is preferred when the timer setup value is not changing, as it does not interrupt the RAM state machine.

11.2.2.2 Timer Counter

The Timer Manager takes an input timer tick to control the timer rate. All timers share a time base. On each rising edge of this tick event, the timer counter increments. The value of the timer counter may be read by software via the TIMERMGR_COUNTER register.

11.2.2.2.1 Timer Counter Rollover

A rollover bit is maintained for each timer, such that when reprogrammed, it is the carry bit from the sum of the current time (counter) and the relative reprogramming value for that timer. This has the effect that any timer whose rollover bit is currently set to a 1 will not expire in the current epoch.

When the counter transitions from 0xFFFF FFFF to 0x0000 0000, the timer control module changes the rollover bit for all timers from a 1 to a 0. All timers that previously did not have the rollover bit set, will expire when they are next visited by the state machine, as their timeout value was for the previous epoch.

The overall effect is that the timers continue to expire as expected, regardless of when the counter rollover occurs relative to the state machine.

11.2.2.3 Timer Control Module (FSM)

The timer control module controls the RAM accesses for reading and writing. It controls the loops over the timer RAM, checks the values in the RAM to observe expiring timers, flagging timeouts so that events may be triggered and reported. It also takes as input any software-written updates to setup, clear, and touch the timers.

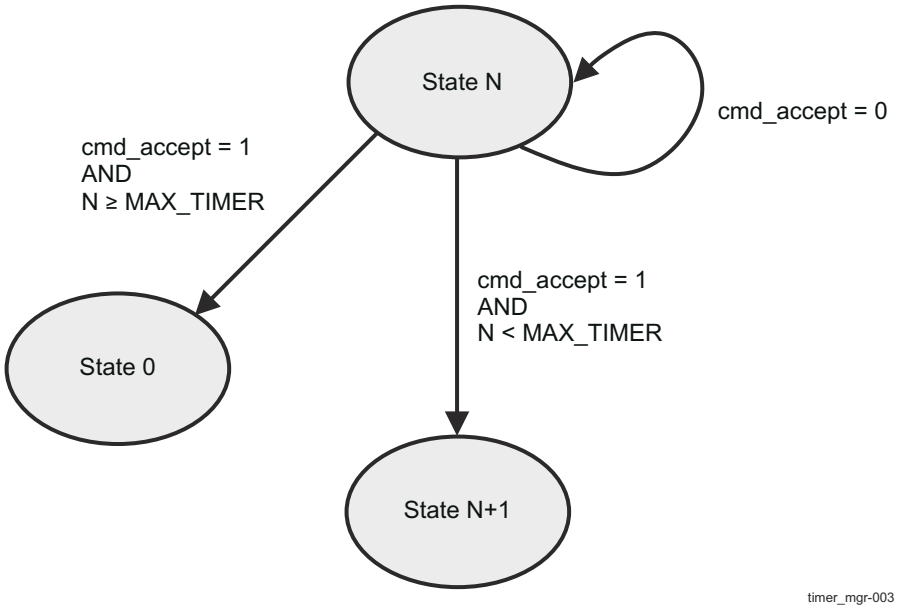


Figure 11-4. Timer State Machine

Copyright © 2024 Texas Instruments Incorporated

The timer states are sequential, that is, Timer 0 is checked, then Timer 1 is checked, then Timer 2, and so on. The Timer Control FSM will stall in a given state if the cmd_accept signal from the ksdw_spram_ecc module goes low, and will remain there until the ECC RAM will accept a command.

The timer state always transitions from N to N+1, unless N meets or exceeds the programmed MAX_TIMER value, in which case the system will start over at state 0.

11.2.2.4 Timer Reprogramming

There is a reprogramming module that runs in lock step with the FSM, always one state ahead – the FSM exports its next_state for use by the reprogramming RAM.

This RAM keeps track of the setup values for the timers as programmed by the TIMERMGR_SETUP_j_k memory mapped locations. There is a SET flag for each timer outside of the RAM – if this SET flag is true for the next_state timer, the VBUS ready signals are deasserted; if there is a pending timer setup, there cannot be a read or write to the memory, as this would interrupt the Timer FSM state machine and, with enough accesses, potentially cause the system to exceed the 10-µs window for timer accuracy.

11.2.2.5 Event FIFO

As the subsystem event aggregator requires more than one cycle to process events, there is an internal FIFO to store the generated events. Events may be generated by expiring timers or by activity on the CBA interface, as disabled timers or reset timers must send an event deactivation if the timer was previously expired.

This has an impact on performance, as if there are too many timer expirations in a given window, it potentially violates the 10-µs requirement (this is true without the event FIFO as well, given that the system cannot keep up with a timer expiring every cycle, in the worst case).

11.2.2.6 Output Event Lookup (OES RAM)

The expiring timers must be assigned corresponding event numbers for use in the instantiating system. These lookup values correspond to the 16-bit event number that is output on the event interface when a given timer expires.

The lookup table for timer to event number mapping defaults all timers to the invalid event number (0xFFFF), and any invalid event numbers observed upon lookup for an expiring timer will result in no events being generated by the timer manager – invalid (0xFFFF) events are suppressed and will not result in an event on the event interface.

It is expected that the output event assignments will be programmed once per module reset, and that they are static during operation of the timer manager.

Copyright © 2024 Texas Instruments Incorporated

11.2.3 Timer Manager Programming Guide

11.2.3.1 Timer Manager Low-level Programming Models

This section covers the low-level hardware programming sequences for configuration and usage of the module.

11.2.3.1.1 Initialization Sequence

Initialization of the Timer Manager must include the following steps:

1. Set the number of timers to be used. TIMERMGR_CNTL[10-1] MAX_TIMER
   - This controls the loop over the timer RAM and is the highest ID timer that will be used in the operation of the Timer Manager (MAX_TIMER + 1 is the total number of timers in use)
   - This number must not change during the course of operation; it can only be changed when TIMERMGR_CNTL[0] ENABLE = 0. It is assumed that the user will set the MAX_TIMER value as required.
   - This step can be skipped if using all timers. MAX_TIMER defaults to 1024 (that is, all timers in use).
2. Write output event mapping for all timers to the OES output event lookup table. This must be done before enabling timer manager and must not be done again once the timer manager has been enabled.
3. Write initial setup values to the TIMERMGR_SETUP_j_k registers. All timers that will be used initially must have a setup value before the Timer Manager is enabled.
4. Enable individual timers. There are two ways to do this.
   - To enable all timers, use the TIMERMGR_CNTL[12] MASS_ENABLE bit. This will enable all timers from 0 to MAX_TIMER.
     - Note that MAX_TIMER and MASS_ENABLE should be set in successive writes, or the previous value of MAX_TIMER will be used for MASS_ENABLE
     - MASS_ENABLE must only be used during initialization.
   - Alternately, individual timers can be enabled or disabled with TIMERMGR_CONTROL_j_k
     - It is therefore faster to set the individual TIMERMGR_CONTROL_j_k bits if more than half of the in-use timers are to be enabled initially. If more than half of the in-use timers are to be enabled initially, it is faster to use MASS_ENABLE and then disable specific timers with the TIMERMGR_CONTROL_j_k bits.
5. Enable the timer manager. Set TIMERMGR_CNTL[0] ENABLE to 1.

11.2.3.1.2 Real-time Operating Requirements

While running, a CPU can take the following actions to enable, disable, or touch the timers:

11.2.3.1.2.1 Timer Touch

When a timer needs to be updated – indicating that an interface or function is still alive and timeout should be delayed – there are two means of touching a timer:

- If the timer's respective TIMERMGR_SETUP_j_k has the appropriate value for its function, the timer can be set again by writing a 1 to the SET bit of the TIMERMGR_CONTROL_j_k register.
  - As the timer should remain enabled, this effectively means writing a 3 to the TIMERMGR_CONTROL_j_k register for the timer to be touched
- If the timer's respective TIMERMGR_SETUP_j_k needs a new value, writing this value has the side effect of also setting the timer – there is no need to also write to the SET bit of TIMERMGR_CONTROL_j_k. This should be relatively infrequent during operation, as the setup values for all timers must be written during initialization.

The preferred method for touching timers is using the SET bit of TIMERMGR_CONTROL_j_k. It is considerably more efficient. The reprogramming RAM is a single port RAM, and repeated writes to the TIMERMGR_SETUP_j_k registers will impact overall timer performance and may result in longer periods before a valid timeout is observed.

11.2.3.1.2.2 Timer Disable

When a timer is no longer in use, the timer should be disabled with the respective TIMERMGR_CONTROL_j_k enable bit.

Copyright © 2024 Texas Instruments Incorporated

A disabled timer does not stop the time for that timer relative to the counter, and its current value is considered invalid once disabled. A timer should not be disabled and then subsequently re-enabled without reprogramming/re-setting, as it may timeout unexpectedly.

11.2.3.1.2.3 Timer Enable

To enable a timer while the Timer Manager is running, the timer must first be setup again.

- If the current TIMERMGR_SETUP_j_k is appropriate for the timer's intended use, the timer can be re-enabled by writing a 3 to the TIMERMGR_CONTROL_j_k register. (ENABLE = 1, SET = 1)
- If the current TIMERMGR_SETUP_j_k needs to be changed, write to the TIMERMGR_SETUP_j_k before re-enabling the timer. After this value is changed, the timer can be re-enabled by writing 0x3 to the TIMERMGR_CONTROL_j_k register (ENABLE = 1, SET = 1)
  - While TIMERMGR_SETUP_j_k implies SET, it must still be set as the timer is enabled.

11.2.3.1.3 Power Up/Power Down Sequence

The timer manager must be disabled (clear TIMERMGR_CNTL[0] ENABLE to 0) before the timer manager is powered down. The timer manager will only be idle if it is not enabled and the event FIFO is empty – an enabled Timer Manager means the state machine is running and the timer manager is waiting for expired timers.

To power up the Timer Manager if it has been powered down but not reset, the timer values in the timer RAM must be assumed to be invalid. Any timers must either be reprogrammed (TIMER_SET) before the timer is re-enabled. If this is not done, the Timer Manager may report timeouts as soon as it is enabled.

Additionally, all timers that have expired must be cleared before disabling the Timer Manager, otherwise there will be no Event Down generated on the event interface to clear the expired timers from the interrupt aggregator. This effectively means that software must disable all timers before writing a 0 to TIMERMGR_CNTL[0] ENABLE.

The event FIFO will continue to drive pending events even if the Timer Manager is disabled, so the remaining Event Down actions will still occur – software only needs to disable the timers to ensure that the Event Down commands are pushed into the event FIFO but does not need to wait for the event FIFO to flush before disabling the timer manager.

Copyright © 2024 Texas Instruments Incorporated

11.3 Time Sync and Compare Events

11.3.1 Time Sync Architecture

This section provides a high-level overview of the SoC time synchronization architecture.

11.3.1.1 Time Sync Architecture Overview

Table 11-2 shows the network time synchronization functions supported by the device.

Table 11-2. Network Time Synchronization Functions in the SoC

Interface	Time Sync Functions	Supported by
1×PCIE	Precision time measurement (PTM)	PCIe controller hardware
2×CPSW	IEEE 1588-2008 (2-step), 802.1AS	CPTS in CPSW
External input reference clock	External time/clock reference	CPTS

Note

These time sync functions are described in detail in their respective chapters.

Any of these functions can be a time sync master in the system. A sync router (TIMESYNC_INTRTR0) provides flexibility for each time domain to choose its synch master independently. In addition, there is also a router (CMPEVT_INTRTR0) that provides selection of active counter compare events for routing as CPU or DMA events.

Copyright © 2024 Texas Instruments Incorporated

11.3.2 Time Sync Routers

11.3.2.1 Time Sync Routers Overview

The timesync event router (TIMESYNC_INTRTR0) and compare event router (CMPEVT_INTRTR0) are instantiations of the generic interrupt router module in the device.

The TIMESYNC_INTRTR0 implements a set of multiplexers to provide selection of active CPTS time sync events for routing to CPTS capable modules. There is one register per output that controls the selection (TIMESYNC_INTRTR0_MUXCNTL_y).

The CMPEVT_INTRTR0 implements a set of multiplexers to provide selection of active CPTS counter compare events for routing as CPU or DMA events. There is one register per output that controls the selection (CMPEVT_INTRTR0_MUXCNTL_y).

Table 11-3 summarizes the configuration details for TIMESYNC_INTRTR0 and CMPEVT_INTRTR0.

Table 11-3. TIMESYNC_INTRTR0 and CMPEVT_INTRTR0 Configuration

Module	Input Interrupt Type
TIMESYNC_INTRTR0	Level
CMPEVT_INTRTR0	Pulse

Refer to *INTRTR Overview* for the general programming sequence that is recommended to be followed for all interrupt routers in the device.

11.3.3 Time Sync Event Sources

See *Interrupt Sources* for Time Sync event mapping.

12 Peripherals

12.1 General Connectivity Peripherals 994
12.2 High-speed Serial Interfaces 1134
12.3 Memory Interfaces 1185
12.4 Industrial and Control Interfaces 1398
12.5 Audio Interfaces 1551
12.6 Display Subsystem (DSS) and Peripherals 1605
12.7 Camera Subsystem 1802
12.8 Shared MIPI D-PHY Transmitter (DPHY_TX) 1850
12.9 Timer Modules 1853
12.10 Internal Diagnostics Modules 1883

Copyright © 2024 Texas Instruments Incorporated

12.1 General Connectivity Peripherals

This section describes the general connectivity peripherals in the device.

12.1.1 Analog-to-Digital Converter (ADC)

This chapter describes the Analog-to-Digital Converter (ADC) in the device.

12.1.1.1 ADC Overview

The Analog-to-Digital Converter (ADC) module contains a single 12-bit ADC which can be multiplexed to any 1 of 8 analog inputs (channels).

Figure 12-1 presents an overview of the ADC modules.

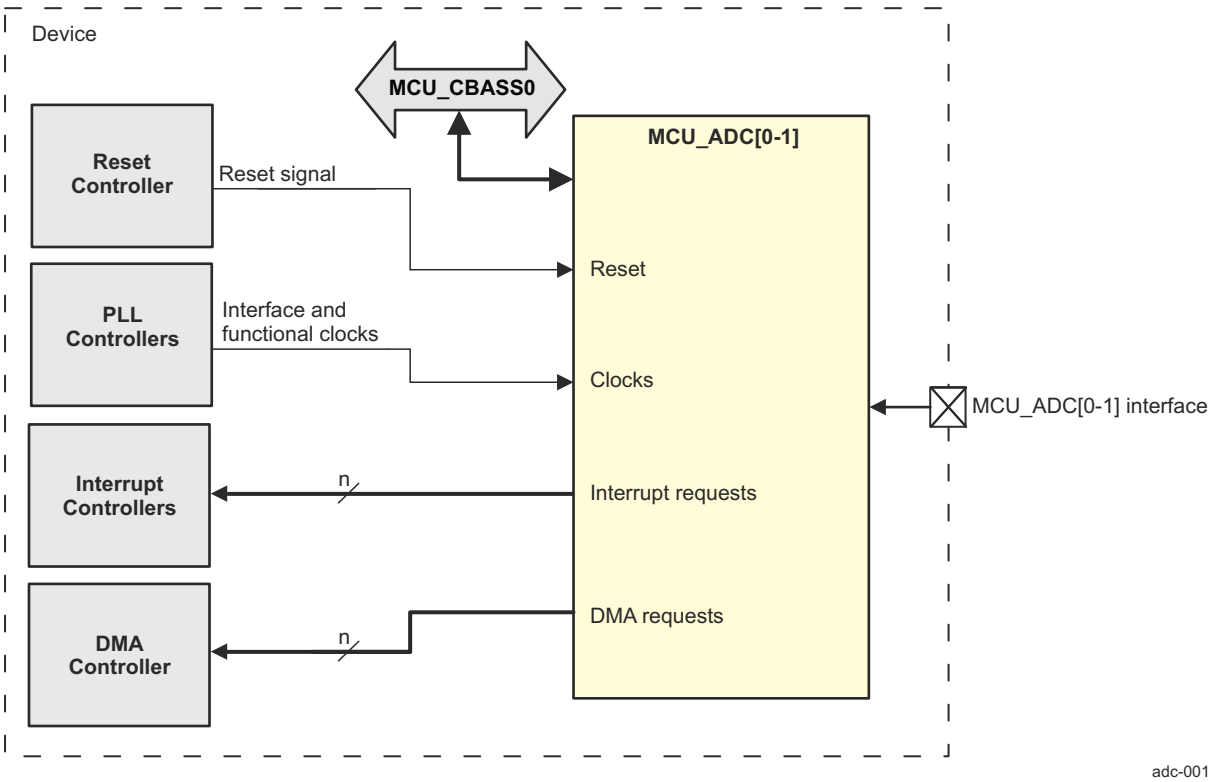


Figure 12-1. ADC Modules Overview

12.1.1.1.1 ADC Features

Each ADC module has the following features:

- 4 MSPS rate with a 60 MHz SMPL_CLK
- Single-ended or differential input options
- Programmable Finite State Machine (FSM) sequencer that supports the following:
  - Software initiated start of conversion
  - Optional hardware start of conversion (SOC), synchronized to external hardware event
  - Single conversion (one-shot mode)
  - Continuous conversions (continuous mode)
  - Sequence through all enabled steps based on a mask
  - Programmable open delay before executing each step
  - Programmable sampling delay for each step
  - Programmable averaging (16, 8, 4, 2, or 1) of input samples for each step
  - Store data in either of two FIFOs – 256-word × 16-bit
  - Option to encode input (channel) number with data

Copyright © 2024 Texas Instruments Incorporated

  - Support for servicing FIFOs via DMA or processor
  - Programmable DMA request event (for each FIFO)
- Support for the following interrupts and status, with masking:
  - Interrupt if AFE fails to return end of conversion (EOC)
  - Interrupt after a sequence of conversions (all non-masked steps)
  - Interrupt for FIFO threshold levels
  - Interrupt if sampled data is out of a programmable range
  - Interrupt for FIFO overflow and underflow conditions
  - Status bit to indicate if ADC is busy converting

Unsupported Features:

- See the Module Integration section of this document for a list of module features not supported by the integration on this Device.

12.1.1.1.2 ADC Ports

Table 12-1. MCU_ADC[0-1] Clocks and Resets

Clocks	
Module Clock Input	**Description**
MCU_ADC_VBUS_CLK	MCU_ADC interface clock
MCU_ADC_SYS_CLK	MCU_ADC system clock
MCU_ADC_CLK	MCU_ADC clock. Output of multiplexer, see *ADC Integration*. Multiplexer control is provided via CTRLMMR_MCU_ADC_CLKSEL[1-0] CLK_SEL bit field. Default state is HFOSC_CLKOUT.
Resets	
Module Reset Input	**Description**
MCU_ADC_RST	MCU_ADC reset

Table 12-2. MCU_ADC[0-1] Hardware Requests

Interrupt Requests		
Module Interrupt Signal	**Description**	**Type**
MCU_ADC_GEN_LEVEL_0	MCU_ADC interrupt request	Level
MCU_ADC_ECC_CORRECTED_ERR_LEVEL_0	MCU_ADC ECC corrected error interrupt request	Level
MCU_ADC_ECC_UNCORRECTED_ERR_LEVEL_0	MCU_ADC ECC uncorrected error interrupt request	Level
DMA Events		
Module DMA Event	**Description**	**Type**
MCU_ADC_FIFO0	MCU_ADC receive request line	Pulse
MCU_ADC_FIFO1	MCU_ADC receive request line	Pulse

12.1.1.2 ADC Environment

This section describes the ADC external connections (environment).

Table 12-3 describes the ADC I/O signals.

Table 12-3. ADC I/O Signals

Module Pin	I/O[(1)]	Description
AIN0	A/I	Analog input 0
AIN1	A/I	Analog input 1
AIN2	A/I	Analog input 2
AIN3	A/I	Analog input 3
AIN4	A/I	Analog input 4
AIN5	A/I	Analog input 5
AIN6	A/I	Analog input 6
AIN7	A/I	Analog input 7
REFN	A	ADC Reference (Negative)

Copyright © 2024 Texas Instruments Incorporated

Table 12-3. ADC I/O Signals (continued)

Module Pin	I/O[(1)]	Description
REFP	A	ADC Reference (Positive)

(1) I = Input; A = Analog

Copyright © 2024 Texas Instruments Incorporated

12.1.1.3 ADC Functional Description

12.1.1.3.1 ADC FSM Sequencer Functional Description

The FSM sequencer supports up to 16 steps that can be configured to perform analog-to-digital conversions. Each of the 16 steps can individually be enabled and configured to operate as software enabled steps or hardware enabled steps. Each step has its own programmable step configuration (ADC_CONFIG_j) and step delay (ADC_DELAY_j) registers that allow users to configure the ADC operation on a per-step basis.

The FSM sequencer enters Idle immediately after the ADC is enabled and remains in Idle while waiting for any of the 16 programmable steps to be enabled. When one or more step is enabled, the FSM sequencer will begin executing or skipping each step starting with the lowest and incrementing up through each step until the highest step has been executed or skipped. The FSM sequencer will execute enabled steps and skip disabled steps. Once all of the steps have been executed or skipped, the FSM sequencer will return to Idle or immediately begin repeating steps configured for continuous mode.

An ENDOFEQUENCE interrupt can be generated after the last enabled step has been executed. If any software enabled steps are configured for continuous operation and they are still enabled when the FSM sequencer has incremented through all the steps, the FSM sequencer will start the sequence again with the first enabled step. Hardware enabled steps are mapped to a hardware event and will need the hardware event to occur before being scheduled.

Figure 12-2 illustrates how the FSM sequencer works. Each shaded box represents a FSM state.

Note

Figure 12-2 does not represent the number of SMPL_CLK cycles required by each state since some states implement user programmable delays that may require more than one SMPL_CLK cycle to complete.

Copyright © 2024 Texas Instruments Incorporated

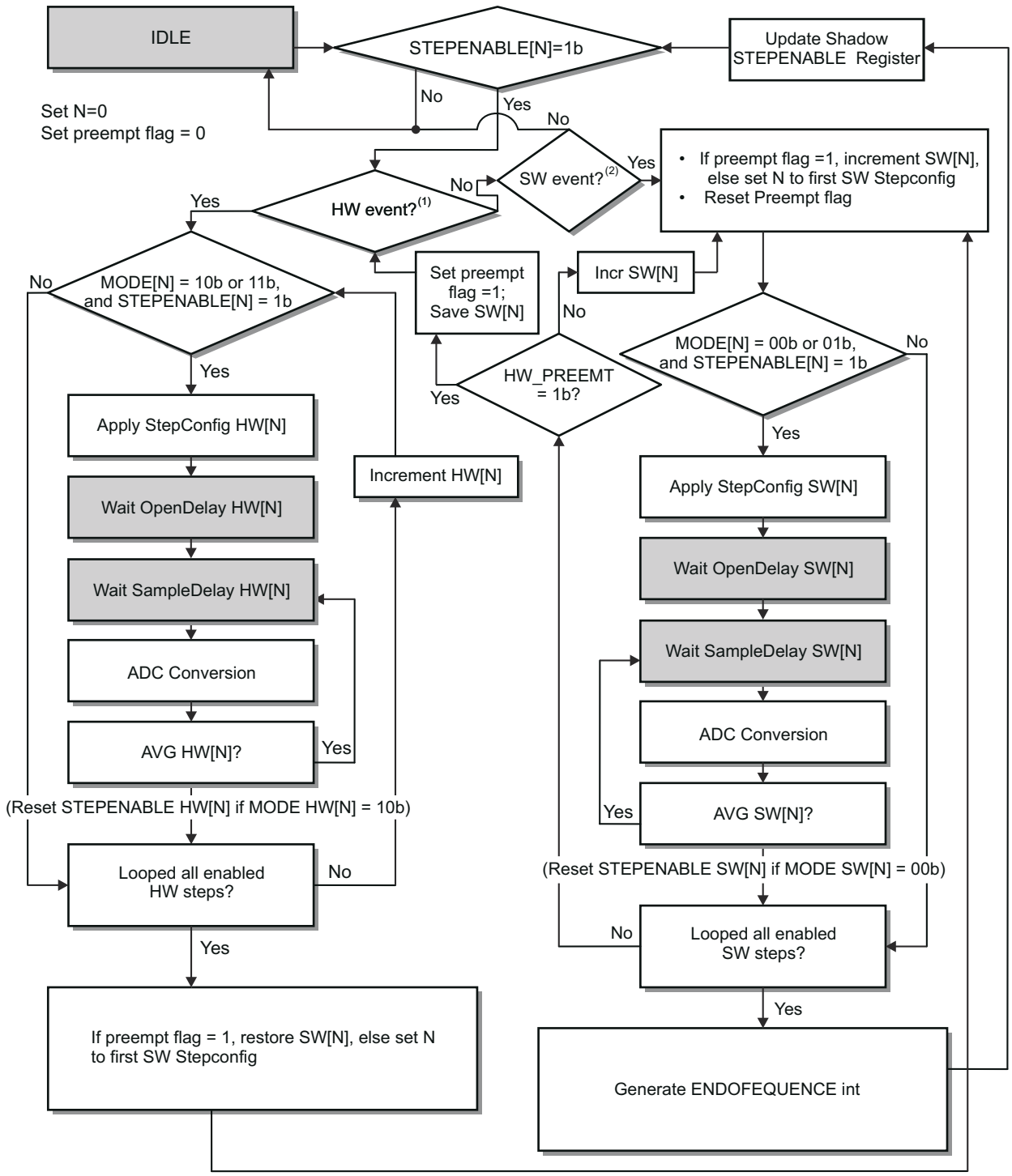


Figure 12-2. Sequencer FSM

12.1.1.3.1.1 Step Enable

A step is enabled or disabled via the ADC_STEPENABLE register. The system software should only write to the ADC_STEPENABLE register when the ADC is disabled or the FSM sequencer is idle. Therefore, it may be necessary to disable the ADC before updating the ADC_STEPENABLE register if the FSM sequencer is executing back-to-back steps where it never enters idle.

Copyright © 2024 Texas Instruments Incorporated

12.1.1.3.1.2 Step Configuration

12.1.1.3.1.2.1 One-Shot (Single) or Continuous Mode

Each step can be configured to operate in one-shot or continuous mode via the least significant bit of the MODE bit-field of the respective ADC_CONFIG_j register. Steps configured for one-shot mode will perform a single conversion before being disabled by the FSM sequencer. Steps configured for continuous mode will automatically be re-enabled by the FSM sequencer after each conversion until disabled by software.

12.1.1.3.1.2.2 Software- or Hardware-Enabled Steps

Each step can be configured to operate in SW Enabled mode or HW Enabled mode independently via the most significant bit of the MODE bit-field in the respective ADC_CONFIG_j register. However, the hardware event source is applied globally to all HW Enabled steps and configured via the ADC_CONTROL register.

The user can configure each step to begin immediately after the step is enabled by software (SW Enabled), or wait for a hardware event to occur (HW Enabled).

The EXT_TRIGGER input signal is used by HW Enabled steps. The trigger input has the option of being sourced from a device input pin or several on-device peripherals via a multiplexer controlled by TRIG_SEL of the respective CTRLMMR_MCU_ADC0_CTRL or CTRLMMR_MCU_ADC1_CTRL register. When this option is selected, the hardware event will be triggered after the EXT_TRIGGER input signal transitions from low to high and is held high for a period greater than two cycles of SYS_CLK. This hardware event will only schedule one complete sequence, even for continuous mode. If HW Enabled steps need to be executed again, a new hardware event must be generated after the previously triggered sequence has completed.

SW Enabled steps can be preempted by HW Enabled steps when the HW_PREEMPT bit in the ADC_CONTROL register is set.

12.1.1.3.1.2.3 Averaging of Samples

Each step can be configured, via the respective ADC_CONFIG_j register, to sample an input 1, 2, 4, 8, or 16 times and provide an average data value. If a step is configured to sample more than once, the additional samples are taken back-to-back immediately after the first sample. However, open delay is only applied to the first sample while sample delay is applied to all samples. Once the last sample has been taken the average data value will be stored in the FIFO.

12.1.1.3.1.2.4 Analog Multiplexer Input Select

The AFE contains two 9-to-1 analog multiplexers which are used to select the source connected to the positive input (INP) and negative input (INM) of the ADC. Each step can be configured, via the respective ADC_CONFIG_j register, to select the appropriate input sources. For more information, refer to Figure 12-4 and the ADC_CONFIG_j register field descriptions.

12.1.1.3.1.2.5 Differential Control

The ADC supports differential inputs and each step can be configured, via the respective ADC_CONFIG_j register, to select single-ended input mode or differential input mode.

Note

The ADC INM input must be sourced from REFN for any steps configured to operate in single-ended mode.

12.1.1.3.1.2.6 FIFO Select

The ADC contains two FIFOs and each step can be configured, via the respective ADC_CONFIG_j register, to store its ADC data into either FIFO.

12.1.1.3.1.2.7 Range Check Interrupt Enable

Each step can be configured, via the respective ADC_CONFIG_j register, to compare the value of ADC data to high and low limits programmed in the ADC_RANGE register. The out-of-range interrupt is generated if the value of ADC data is greater than the value of HIRANGE or less than the value of LOWRANGE. Refer to the

Copyright © 2024 Texas Instruments Incorporated

ADC_RANGE register field descriptions for information related to setting the high and low limits that trigger an out-of-range interrupt.

12.1.1.3.1.3 Open Delay and Sample Delay

The FSM sequencer provides two programmable delays for each step. The value of OPENDELAY is used to control when the acquisition begins after the step starts and the value of SAMPLEDELAY is used to control the acquisition period. Delays for each of the 16 steps can be configured independently via the respective ADC_DELAY_j register.

12.1.1.3.1.3.1 Open Delay

OPENDELAY defaults to a value of zero which causes the acquisition period to begin as soon as the step starts. The start of the acquisition period can be delayed one SMPL_CLK clock period for each incremental value of OPENDELAY.

12.1.1.3.1.3.2 Sample Delay

SAMPLEDELAY defaults to a value of zero which causes the acquisition period to be equal to two SMPL_CLK clock periods. The acquisition period can be extended one SMPL_CLK clock for each incremental value of SAMPLEDELAY.

The value of SAMPLEDELAY should be configured to provide enough time for the respective external voltage source to completely charge the AFE input capacitance during the acquisition period.

12.1.1.3.1.4 Interrupts

The ADC has 9 internal events that can trigger an interrupt to the processor. These events are logically or'ed together to produce a single interrupt to the processor. The user can individually enable or disable each interrupt source via the ADC_ENABLE_SET and ADC_ENABLE_CLR registers. Once an interrupt is generated, the ADC_STATUS register can be read to determine which interrupt source(s) interrupted the processor. The interrupt source(s) can be cleared by writing a ‘1’ to the corresponding bit of the ADC_STATUS register.

The following interrupts are supported by the device ADC:

- An ENDOFEQUENCE interrupt is generated after the FSM sequencer completes the last enabled step.
- FIFO0UNFL/FIFO1UNFL or FIFO0OVFL/FIFO1OVFL interrupts. Each FIFO also has the ability to generate overrun and underflow interrupts. The ADC does not recover from overrun or underflow conditions automatically. Therefore, software will need to disable and re-enable the ADC after this occurs. The FSM sequencer must be Idle before turning the ADC module back on. This can be confirmed by reading the ADC_SEQUENCER_STAT register, when FSM_BUSY = 0x0 and STEP_IDLE = 0x10000.
- FIFO0THRS and FIFO1THRS interrupts. Each FIFO has the ability to generate an interrupt when the FIFO word count has reached a programmable threshold value. The FIFO0THRS and FIFO1THRS interrupt is generated when the value of NUMWDS in the respective ADC_FIFO0WC or ADC_FIFO1WC register equals the word count threshold value programmed in THRESHOLD of the respective ADC_FIFO0THRESHOLD or ADC_FIFO1THRESHOLD register.
- OUTOFRANGE interrupt. This interrupt can be enabled or disabled on each step. If enabled, the ADC data value is compared to the low and high thresholds programmed in the global ADC_RANGE register. The OUTOFRANGE interrupt is generated if the ADC data value is greater than the value programmed in HIRANGE or less than the value programmed in LOWRANGE.
- AFE_EOC_MISSING interrupt. This interrupt is generated when the FSM sequencer does not receive an expected EOC from the AFE.

12.1.1.3.1.5 Power Management

The software also has the option to turn off the AFE power by writing to the ADC_CONTROL[4] PD bit. By default, after the initial power on the AFE is powered down and software is responsible for turning it on. Software must wait minimum 4 µs after a AFE power up and before starting a conversion (must be followed every time the AFE is powered up). It is the software’s responsibility to empty the FIFO before requesting the ADC to enter sleep state.

Copyright © 2024 Texas Instruments Incorporated

12.1.1.3.1.6 DMA Requests

Each FIFO can be serviced by the processor or DMA. The user can individually enable or disable each FIFO DMA request via the ADC_DMAENABLE_SET and ADC_DMAENABLE_CLR registers.

12.1.1.3.2 ADC AFE Functional Description

The AFE contains a single 12-bit successive approximation ADC which can be connected to one of eight user selectable analog inputs for each step executed by the FSM sequencer. There is also an option to configure the ADC to operate in differential mode where it samples the differential voltage applied to two of eight user selectable analog inputs.

The ADC sits idle until the FSM sequencer sends a Start of Conversion (SOC) pulse. Once the FSM sequencer sends a SOC pulse to the ADC, it begins sampling the analog input signal on the rising edge of SOC and continues sampling the analog input signal one cycle of SMPL_CLK after the falling edge of SOC. The SOC pulse width will be the value of SAMPLEDELAY plus one SMPL_CLK periods, so the total ADC acquisition time will be the value of SAMPLEDELAY plus two SMPL_CLK periods. The ADC captures the analog input signal at the end of the acquisition period and starts conversion, which requires thirteen SMPL_CLK periods to digitize the sampled input. An EOC signal is returned to the FSM sequencer to indicate the digital data is ready.

The ADC output is positive binary weighted data ranging from 0 to (2^{12} - 1). When configured for single-ended mode, ADC output data values 0 to (2^{12} - 1) represents a corresponding input voltage that ranges from the ADC negative reference (REFN) voltage to the ADC positive reference (REFP) voltage. When configured for differential mode, ADC output data values 0 to ((2^{12}/2) - 1) represents a corresponding negative differential input voltage that ranges from the REFP voltage to the REFN voltage and output data values (2^{12}/2) to (2^{12} - 1) represents a corresponding positive differential input voltage that ranges from the REFN voltage to the REFP voltage.

Using the minimum values of OPENDELAY and SAMPLEDELAY, the ADC can sample an analog input every 15 SMPL_CLK periods.

Figure 12-3 illustrates the operation of the FSM sequencer and ADC AFE, with indicators showing when the hardware configuration defined by ADC_CONFIG_j and ADC_DELAY_j registers are applied.

Note

Figure 12-3 assumes both steps shown are software enabled with averaging turned off, an OPENDELAY value of 1, and SAMPLEDELAY value of 0. The analog input is sampled for the duration of the SOC pulse plus one cycle of the SMPL_CLK. If the value of OPENDELAY were 0, it would appear as if the time associated with the Open portion of the FSM waveform would be removed from all the waveforms.

Copyright © 2024 Texas Instruments Incorporated

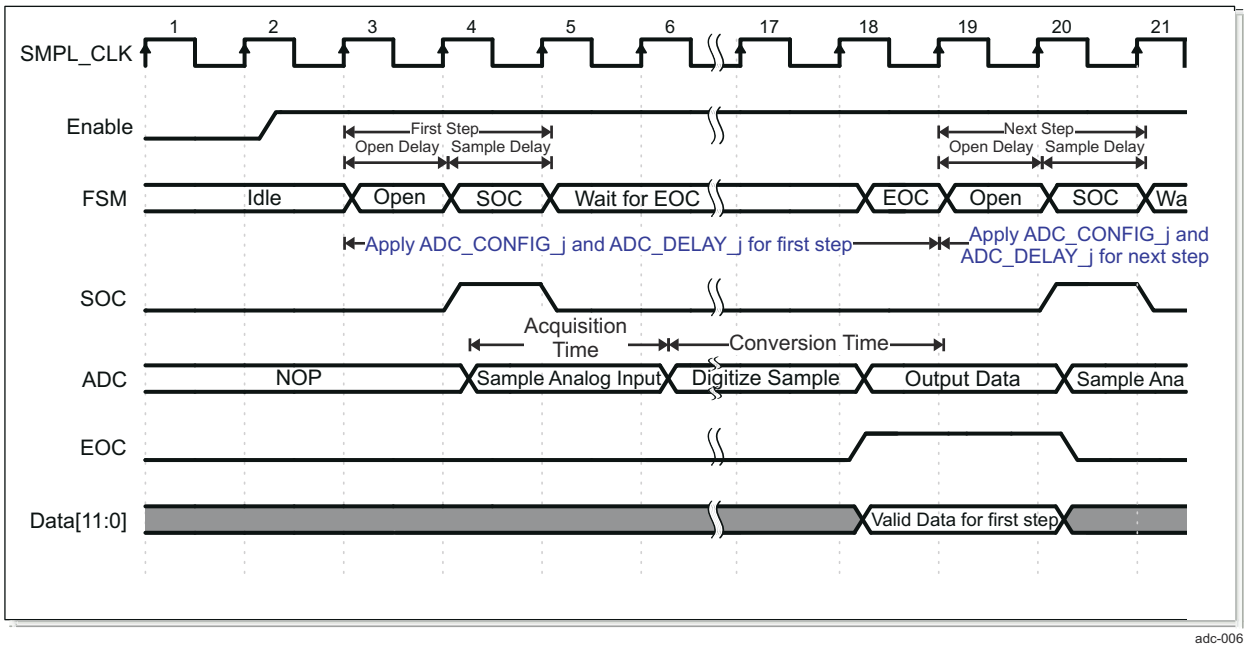


Figure 12-3. Example Timing Diagram for Sequencer

12.1.1.3.2.1 AFE Functional Block Diagram

The AFE has 8 analog inputs which can be configured as either singled-ended or differential inputs to the ADC. The user may configure a step to use any 1 of the 8 inputs when operating in single-ended mode or any 2 of the 8 inputs when operating in differential mode. For more information, refer to Figure 12-4 and the ADC_CONFIG_j register field descriptions.

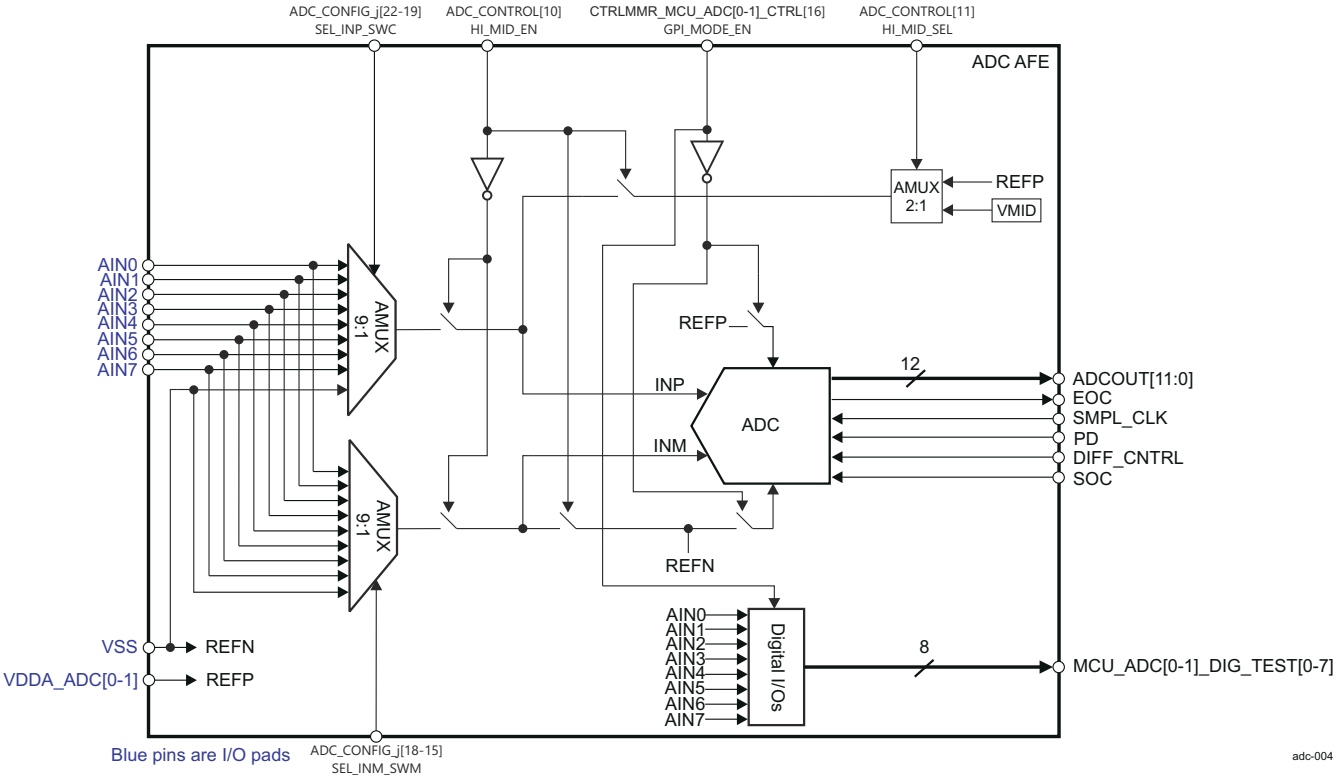


Figure 12-4. Functional Block Diagram

Copyright © 2024 Texas Instruments Incorporated

12.1.1.3.3 ADC FIFOs and DMA

12.1.1.3.3.1 FIFOs

There are two sample FIFOs (ADC_FIFO0 and ADC_FIFO1), which store sample data from the AFE. Can be configured, via the respective ADC_CONFIG_j register.

The processor can read ADC data directly from the FIFO by using the respective ADC_FIFO0DATA or ADC_FIFO1DATA register. The internal logic will pop the next data from the FIFO and increment the FIFO read pointers.

The FIFO data will no longer be accessible after the ADC is disabled because the FIFO pointers are reset. Therefore, it is important to read all FIFO data before disabling the ADC.

12.1.1.3.3.2 DMA

The ADC has a dedicated slave port for the DMA which allows it to perform continuous burst reads when accessing the FIFOs.

Each FIFO has its own DMA request which can be enabled via the ADC_DMAENABLE_SET register and disabled via the ADC_DMAENABLE_CLR register.

The first DMA request is generated after the FIFO leaves the EMPTY state and fills to the level programmed in DMAREQLEVEL of the respective ADC_FIFO0DMAREQ or ADC_FIFO1DMAREQ register.

Subsequently, a new DMA request is automatically generated on the next clock cycle after the current DMA access completes if the previous DMA access did not empty the FIFO.

12.1.1.3.4 ADC Error Correcting Code (ECC)

The Error Correcting Code (ECC) is a mechanism for providing increased system reliability (via reduction of memory soft errors) by allowing single bit errors to be detected and corrected and double bit errors to be detected.

The ECC Aggregator provides a single EOI-handshake based interrupt to the host (for both single and double error detections).

The ECC Aggregator will write 0's to all of memory after reset is released. The ADC_SEQUENCER_STAT[6] MEM_INIT_DONE bit will be 0 during this period and will be set to 1 upon completion.

For more information about the ECC Aggregator, see *ECC Aggregator* and *ADC ECC Registers*.

See also *Testing ECC Error Injection* and *ADC ECC Registers*.

12.1.1.3.4.1 Testing ECC Error Injection

The read enable (REN) input to the RAMs instantiated in this module are constantly tied to 1'b1. As a result, read is continuously in progress.

In order to test ECC single error injection on a particular RAM row, the following conditions have to be maintained before setting the ADC_ECC_CTRL[3] FORCE_SEC bit:

- ADC_ECC_CTRL[5] FORCE_N_ROW bit has to be clear. This is required in order to configure required row in
- ADC_ECC_CTRL[6] ERROR_ONCE bit has to be clear. If this bit is set, the ECC error occurs immediately after setting ADC_ECC_CTRL[3] FORCE_SEC bit since REN is always set
- ADC_ECC_ERR_CTRL1[31-0] ECC_ROW bit field has to be point to the required row address

12.1.1.3.5 ADC Functional Debug Mode

The ADC function can be validated by measuring known reference voltage sources. This feature provides register control that connects the AFE to known reference voltage sources. However, it does not automatically configure the FSM to perform conversions to measure these sources. Software will need to configure the FSM to perform a measurement after selecting this mode of operation.

ADC_CONTROL[10] HI_MID_EN is used to disconnect the AFE from all analog inputs and connect it to one of two known reference voltage sources. ADC_CONTROL[11] HI_MID_SEL is used to select REFP or VMID as the known reference voltage source.

Copyright © 2024 Texas Instruments Incorporated

The conversion results is expected to produce a full scale value when the AFE is connected to REFP and approximate half scale value when the AFE is connected to VMID.

The steps are:

- Disable the ADC if previously enabled and wait until the FSM is not busy.
- Put the AFE in a functional debug mode by setting the ADC_CONTROL[10] HI_MID_EN bit to 1.
- Select one of the two reference voltage sources by setting the ADC_CONTROL[11] HI_MID_SEL bit.
  - 0 selects VMID
  - 1 selects REFP
- Configure the FSM to measure the selected reference voltage source.
- Enable the ADC and evaluate the results.
- When debug is complete, disable the ADC and wait until the FSM is not busy, re-program the FSM for functional operation, then re-enable the ADC.

Copyright © 2024 Texas Instruments Incorporated

12.1.1.4 ADC Programming Guide

12.1.1.4.1 ADC Low-Level Programming Models

12.1.1.4.1.1 During Operation

Table 12-4. Turn-Off

Step	Register/Bit Field/Programming Model	Value
Turn off ADC module enable bit when finished. This will stop the ADC after the current step is complete.	ADC_CONTROL[0] MODULE_ENABLE	0

Table 12-5. Turn-On After Turn-Off

Step	Register/Bit Field/Programming Model	Value
Enable the ADC module enable bit	ADC_CONTROL[0] MODULE_ENABLE	1

Copyright © 2024 Texas Instruments Incorporated

12.1.2 General-Purpose Interface (GPIO)

This chapter describes the General-Purpose Input/Output (GPIO) for the device.

12.1.2.1 GPIO Overview

The General-Purpose Input/Output (GPIO) peripheral provides dedicated general-purpose pins that can be configured as either inputs or outputs. When configured as an output, the user can write to an internal register to control the state driven on the output pin. When configured as an input, user can obtain the state of the input by reading the state of an internal register.

In addition, the GPIO peripheral can produce host CPU interrupts and DMA synchronization events in different interrupt/event generation modes.

12.1.2.1.1 GPIO Features

Each channel in the GPIO modules has the following features:

- Supports 9 banks of 16 GPIO signals
- Supports up to 9 banks of interrupt capable GPIOs
- Interrupts:
  - Can enable interrupts for each bank of 16 GPIO signals
  - Interrupts can be triggered by rising and/or falling edge, specified for each interrupt capable GPIO signal
- Set/clear functionality:
  - Firmware writes 1 to corresponding bit position(s) to set or to clear GPIO signal(s). This allows multiple firmware processes to toggle GPIO output signals without critical section protection (disable interrupts, program GPIO, re-enable interrupts, to prevent context switching to another process during GPIO programming).
- Separate Input/Output registers:
  - If preferred by firmware, some GPIO output signals can be toggled by direct write to the output register(s) in addition to set/clear.
  - Output register, when read in, reflects output drive status. This, in addition to the input register reflecting pin status this allows wired logic to be implemented.

Unsupported Features:

- See the Module Integration section of this document for a list of module features not supported by the integration on this Device.

12.1.2.1.2 GPIO Ports

Table 12-6. GPIO Clocks and Resets

Clocks	
Module Clock Input	**Description**
GPIO_VBUS_CLK	GPIO functional and interface clock
Resets	
Module Reset Input	**Description**
GPIO_RST	GPIO reset

Table 12-7. GPIO[0-7] Hardware Requests

Interrupt Requests		
Module Interrupt Signal	**Description**	**Type**
GPIO_0_INT[0:127]	GPIO pins[0:127] interrupt request	Pulse
GPIO_GPIO_BANK_0	GPIO bank0 interrupt request	Pulse
GPIO_GPIO_BANK_1	GPIO bank1 interrupt request	Pulse
GPIO_GPIO_BANK_2	GPIO bank2 interrupt request	Pulse
GPIO_GPIO_BANK_3	GPIO bank3 interrupt request	Pulse
GPIO_GPIO_BANK_4	GPIO bank4 interrupt request	Pulse
GPIO_GPIO_BANK_5	GPIO bank5 interrupt request	Pulse
GPIO_GPIO_BANK_6	GPIO bank6 interrupt request	Pulse

Copyright © 2024 Texas Instruments Incorporated

Table 12-7. GPIO[0-7] Hardware Requests (continued)

GPIO_GPIO_BANK_7	GPIO bank7 interrupt request	Pulse

12.1.2.2 GPIO Environment

This section describes the GPIO external connections (environment).

Copyright © 2024 Texas Instruments Incorporated

Table 12-8 describes the GPIO I/O signals.

Table 12-8. GPIO I/O Signals

Module Pin	I/O[(1)]	Description
GPIO_BANK[0-8]_[0-15]	I/O	General-purpose input/output.[(2)]

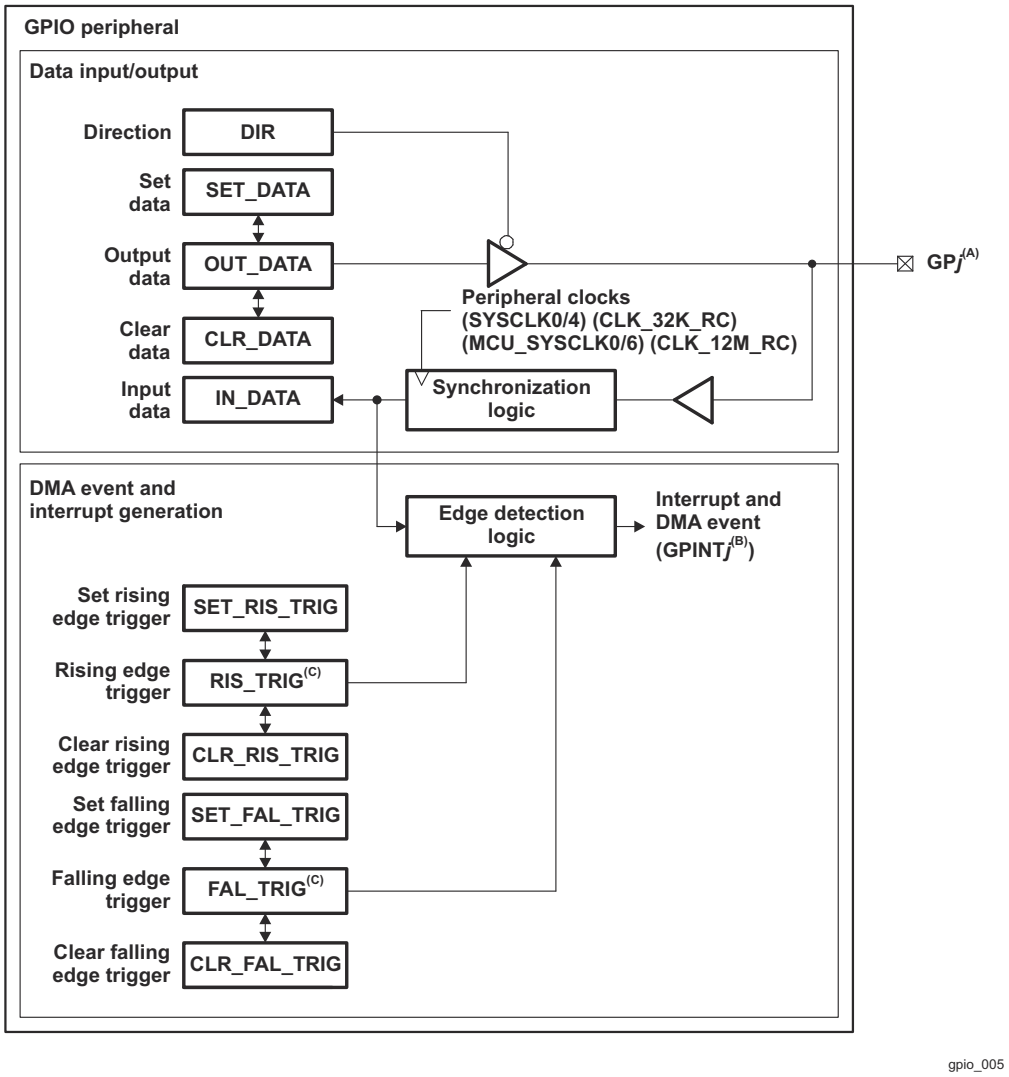
(1) I = Input; O = Output; I/O = Bidirectional
(2) Only GPIOs enabled through from the CTRLMMR_WKUP_PADCONFIG0 to CTRLMMR_WKUP_PADCONFIG93 or from the CTRLMMR_PADCONFIG0 to CTRLMMR_PADCONFIG172 registers for a particular instance can operate on I/O pins. For more information see *Pad Configuration Registers*.

Copyright © 2024 Texas Instruments Incorporated

12.1.2.3 GPIO Functional Description

12.1.2.3.1 GPIO Block Diagram

Figure 12-5 shows the general-purpose interface block diagram.



A. Some of the GPj pins are muxed with other device signals. For details, see the device-specific Datasheet.
B. All GPINTj can be used as host CPU interrupts and synchronization events to the DMA.
C. The RIS_TRIG and FAL_TRIG registers are internal to the GPIO module and are not visible to the host CPU.

Figure 12-5. GPIO Block Diagram

12.1.2.3.2 GPIO Function

Each GPIO pin (GPj) can be independently configured as either an input or an output using the GPIO direction registers. The GPIO direction register (DIR) specifies the direction of each GPIO signal. Logic 0 indicates the GPIO pin is configured as output, and logic 1 indicates input.

When configured as output, writing a 0x1 to a bit in the set data register drives the corresponding GPj to a logic-high state. Writing a 0x1 to a bit in the clear data register drives the corresponding GPj to a logic-low state. The output state of each GPj can also be directly controlled by writing to the output data register.

For example, to set GP8 to a logic-high state, the software can perform one of the following:

- Write 100h to the GPIO_SET_DATA01 register.
- Write 0h to the GPIO_DIR01 register to configure as output pin.

Copyright © 2024 Texas Instruments Incorporated

- Read in GPIO_OUT_DATA01 register, change bit 8 to 0x1, and write the new value back to GPIO_OUT_DATA01.

To set GP8 to a logic-low state, the software can perform one of the following:

- Write 100h to the GPIO_CLR_DATA01 register.
- Write 0h to the GPIO_DIR01 register to configure as output pin.
- Read in GPIO_OUT_DATA01 register, change bit 8 to 0x0, and write the new value back to GPIO_OUT_DATA01.

Note that writing a 0x0 to bits in the set data and clear data registers does not affect the GPIO pin state.

Also, for GPIO pins configured as input, writing to the set data, clear data, or output data registers does not affect the pin state.

For a GPIO pin configured as input, reading the input data register (IN_DATA) will return the pin state. Reading the SET_DATA register or the CLR_DATA data register will return the value in OUT_DATA, not the actual pin state. The pin state is available by reading the input data register. Note that when the direction is configured as input, the output state is determined by software's programming set/clear/output registers, and may not agree with the pin state, which is driven by an external device.

12.1.2.3.3 GPIO Interrupt and Event Generation

Each GPIO pin (GPj) can be configured to generate a host CPU interrupt (GPINTj) or a synchronization event to the DMA (GPINTj). Configuration is on per-bank basis. Each bit of the BINTEN parameter dictates YES/NO option for each bank. Bit 0 controls bank 0, bit 1 controls bank 1, and so on.

The interrupt can be generated on the rising-edge, falling-edge, or on both edges of the GPIO signal . The edge detection logic is synchronized to the GPIO peripheral clock.

The direction of the GPIO pin does not need to be input when using the pin to generate the interrupt or DMA event. When the GPIO pin is configured as input, transitions on the pin trigger interrupts or DMA events. When the GPIO pin is configured as output, software can toggle the GPIO output register to change the pin state and in turn trigger the interrupt or DMA event.

Note that the direction of the pin need not be input for interrupt generation to work. When the GPIO pin is configured as input, transitions on the pin trigger interrupts. When the GPIO pin is configured as output, firmware can toggle the GPIO output register to change the pin state, and in turn trigger interrupts.

Each interrupt output of GPIO signal are available at the module boundary. Each group of 16 GPIO_INTR_INTj signals also has their masked interrupt outputs ORed together to generate a per bank interrupt, available at the module boundary. The idea is to either connect individual interrupts or per bank interrupts to the system interrupt controller.

12.1.2.3.3.1 Interrupt Enable (per Bank)

The GPIO_BINTEN register provides interrupt enable/disable feature for each bank of 16 GPINT signals.

12.1.2.3.3.2 Trigger Configuration (per Bit)

Two internal registers, RIS_TRIG and FAL_TRIG, specify which edge of the GPj signal generates an interrupt or DMA event. Each bit in these two registers corresponds to a GPj pin. Table 12-9 describes the host CPU interrupt and DMA event generation of GPj pin based on the bit settings of the RIS_TRIG and FAL_TRIG registers.

Table 12-9. GPIO Interrupt and DMA Event Configuration Options

RIS_TRIG Bit n	FAL_TRIG Bit n	Host CPU Interrupt and DMA Event Generation
0	0	GPINTj interrupt and DMA event is disabled
0	1	GPINTj interrupt and DMA event is triggered on falling edge of GPj signal
1	0	GPINTj interrupt and DMA event is triggered on rising edge of GPj signal
1	1	GPINTj interrupt and DMA event is triggered on both rising and falling edge of GPj signal

Copyright © 2024 Texas Instruments Incorporated

The RIS_TRIG and FAL_TRIG registers are not directly accessible or visible to the host CPU. These registers are accessed indirectly through four registers: SET_RIS_TRIG, CLR_RIS_TRIG, SET_FAL_TRIG, and CLR_FAL_TRIG. Writing 1 to a bit on the SET_RIS_TRIG register sets the corresponding bit on the RIS_TRIG register. Writing 1 to a bit of the CLR_RIS_TRIG register clears the corresponding bit on the RIS_TRIG register. Writing to the SET_FAL_TRIG and CLR_FAL_TRIG registers works the same way on the FAL_TRIG register.

Reading the SET_RIS_TRIG or CLR_RIS_TRIG register returns the value of the RIS_TRIG register. Reading from the SET_FAL_TRIG and CLR_FAL_TRIG register returns the value of the FAL_TRIG register.

To use the GPIO pins as sources for host CPU interrupts and DMA events, the associated bank interrupt enable register bit in GPIO_BINTEN must also be set to 1. For example, to enable GPIO0_19 (which is in bank 1), GPIO_BINTEN[1] = 1 should be set to enable interrupts for bank 1.

12.1.2.3.3.3 Interrupt Status and Clear (per Bit)

The INTSTAT registers provide interrupt status upon reading, and interrupt clear feature upon writing 1 to the corresponding bit position(s). Upon receiving an interrupt, the ISR can examine the interrupt status and clear the processed interrupts.

Note

The GPIO module generates an interrupt pulse on the individual GPINT interrupt in response to each occurrence of the specified edge condition. Therefore, for GPINT signals having their interrupts routed directly to the interrupt controller, it is not necessary to clear the status bits in this module. The interrupt status and clear register is a facility for the per-bank interrupt connection.

12.1.2.3.4 GPIO Emulation Halt Operation

The GPIO peripheral is not affected by emulation halts.

Copyright © 2024 Texas Instruments Incorporated

12.1.2.4 GPIO Programming Guide

12.1.2.4.1 GPIO Low-Level Programming Models

12.1.2.4.1.1 GPIO Operational Modes Configuration

12.1.2.4.1.1.1 GPIO Read Input Register

Table 12-10. GPIO Read Input Register

Step	Register/Bit Field/Programming Model	Value
Read interrupt status of the corresponding bank	INTSTAT	-h
Read input register value	IN_DATA	-h
Clear interrupt status	INTSTAT	FFFF FFFFh

12.1.2.4.1.1.2 GPIO Set Bit Function

Table 12-11. GPIO Set Bit Function

Step	Register/Bit Field/Programming Model	Value
Write 1h to set desired bit(s) in SET_DATA register.	SET_DATA	-h

12.1.2.4.1.1.3 GPIO Clear Bit Function

Table 12-12. GPIO Clear Bit Function

Step	Register/Bit Field/Programming Model	Value
Write 1h to clear desired bit(s) in CLR_DATA register.	CLR_DATA	-h

Copyright © 2024 Texas Instruments Incorporated

12.1.3 Inter-Integrated Circuit (I2C) Interface

This section describes the Inter-Integrated Circuit (I2C) module in the device.

12.1.3.1 I2C Overview

The device contains Inter-Integrated Circuit (I2C) controllers each of which provides an interface between a local host (LH), such as an Arm or a Digital Signal Processor (DSP), and any I^2C-bus-compatible device that connects via the I^2C serial bus. External components attached to the I^2C bus can serially transmit and receive up to 8 bits of data to and from the LH device through the 2-wire I^2C interface.

Each multimaster I2C module can be configured to act like a slave or master I^2C-compatible device.

Some controllers have dedicated I^2C compliant open drain buffers and support high speed mode (up to 3.4 Mbps in 1.8 V mode and up to 400 Kbps in 3.3 V mode). Other controllers are multiplexed with standard LVCMOS I/O, connected to emulate open drain, and support fast mode (up to 400 Kbps in 1.8 V/3.3 V mode). The I^2C emulation is achieved by configuring the LVCMOS buffers to output Hi-Z instead of driving high when transmitting logic 1.

For the specific I/O buffer and timing characteristics of the different I2C instances, see the device-specific data sheet.

12.1.3.1.1 I2C Features

Each multicontroller I2C module has the following features:

- Compliant with Philips I^2C-bus specification version 2.1
- Supports a standard mode (up to 100 Kbps) and fast mode (up to 400 Kbps)
- Supports HS mode (up to 3.4 Mbps) on some instances. See the the device-specific Data Manual for details.
- 7-bit and 10-bit device addressing modes
- General call
- Start/Restart/Stop
- Multicontroller transmitter/target receiver mode
- Multicontroller receiver/target transmitter mode
- Combined controller transmit/receive and receive/transmit mode
- Built-in FIFO for buffered read
- Module enable/disable capability
- Programmable multitarget channel (responds to four separate addresses)
- Programmable clock generation
- 8-bit-wide data access
- Low power consumption
- Support Auto Idle mechanism
- Support Idle Request/Idle Acknowledge handshake mechanism
- Support for asynchronous wakeup mechanism
- Wide interrupt capability

Unsupported Features:

- See the Module Integration section of this document for a list of module features not supported by the integration on this Device.

Copyright © 2024 Texas Instruments Incorporated

12.1.3.1.2 I2C Ports

This section describes I2C ports related to clocks, resets, and hardware requests.

Table 12-13. I2C[0-6] Clocks and Resets

Clocks	
Module Clock Input	**Description**
I2C_OCP_CLK	I2C interface clock
I2C_SYS_CLK	I2C functional clock
Resets	
Module Reset Input	**Description**
I2C_RST	I2C reset

Table 12-14. I2C[0-6] Hardware Requests

Interrupt Requests		
Module Interrupt Signal	**Description**	**Type**
I2C_POINTRPEND_0	I2C Interrupt request	Level

Copyright © 2024 Texas Instruments Incorporated

12.1.3.2 I2C Environment

This section describes the I2C external connections (environment).

Table 12-15 describes the I2C I/O signals.

Table 12-15. I2C I/O Signals

Module Pin	I/O[(1)]	Description
SCL	I/O	I^2C serial clock line. Open-drain output buffer.
SDA	I/O	I^2C serial data line. Open-drain output buffer.

(1) I = Input; O = Output; I/O = Bidirectional

Copyright © 2024 Texas Instruments Incorporated

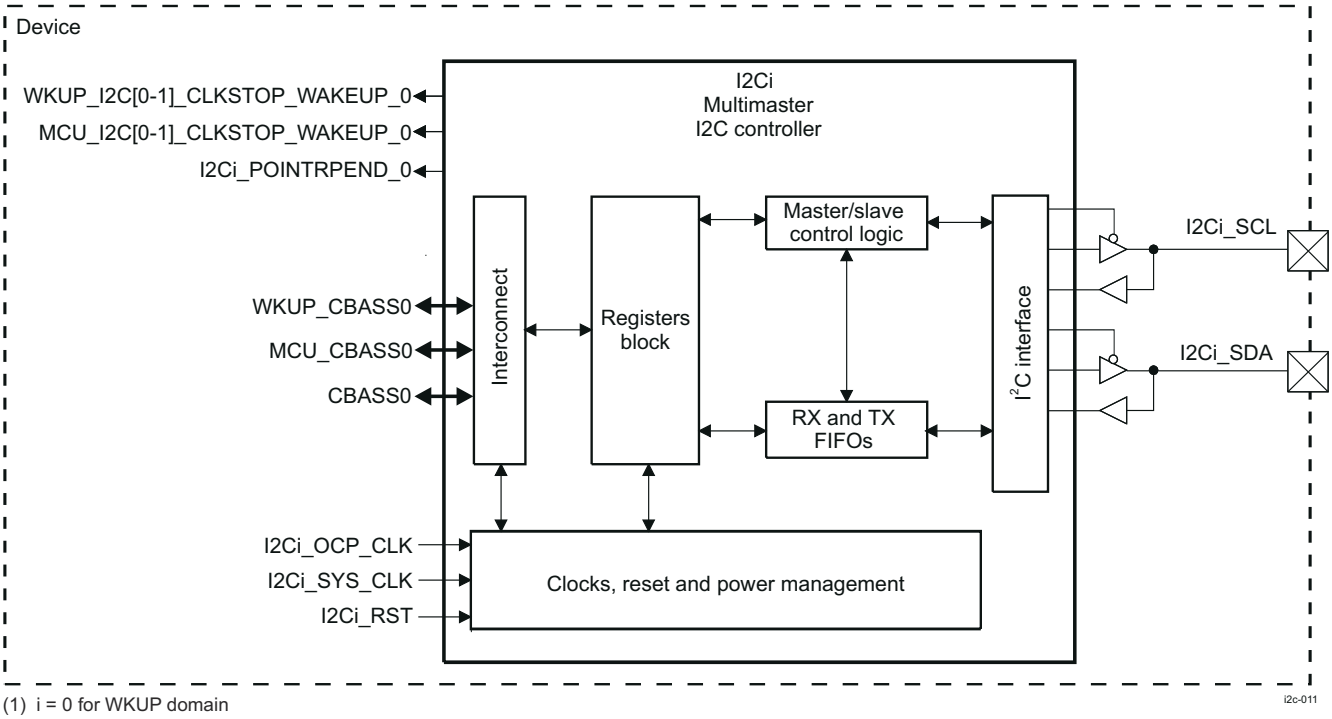
12.1.3.3 I2C Functional Description

12.1.3.3.1 I2C Block Diagram

Note

DMA mode and SCCB Protocol are not supported on this family of devices.

Figure 12-6 presents the multimaster I2C controller block diagram.



(1) i = 0 for WKUP domain
(2) i = 0 to 1 for MCU domain
(3) i = 0 to 6 for MAIN domain

Figure 12-6. I2C Block Diagram

The ten multimaster I2C controllers can be configured in F/S I^{2}C mode or HS I^{2}C mode. The operation mode is selected by configuring the I2C_CON[13-12] OPMODE bit field.

Table 12-16 lists the available operation modes.

Table 12-16. I^{2}C Operation Mode Selection

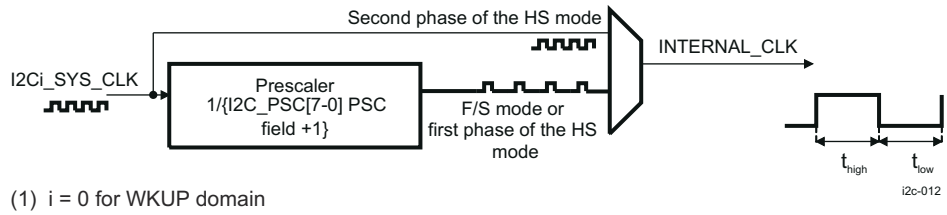
Operation Mode	Value of I2C_CON[13-12] OPMODE
F/S I^{2}C	0x0
HS I^{2}C	0x1
SCCB	0x2
Reserved (not used)	0x3

12.1.3.3.2 I2C Clocks

12.1.3.3.2.1 I2C Clocking

Figure 12-7 shows the I^{2}C clock generation of the I2C controllers.

Copyright © 2024 Texas Instruments Incorporated



(1) i = 0 for WKUP domain
(2) i = 0 to 1 for MCU domain
(3) i = 0 to 6 for MAIN domain

Figure 12-7. I2C Clock Generation

Each multimaster I2C controller uses the SYS_CLK functional clock in the Device Configuration section. The internal sampling clock INTERNAL_CLK is generated by dividing the functional clock by the I2C_PSC[7-0] PSC bit field value + 1 in F/S mode, or in the first phase of HS mode; or by directly using the functional clock in the second phase of HS mode (prescaler is bypassed).

The low time of the SCLL signal is determined by the I2C_SCLL[7-0] SCLL bit field in F/S mode and in the first phase of HS mode; or by the I2C_SCLL[15-8] HSSCLL bit field in the second phase of HS mode.

The high time of the SCLH signal is determined by the I2C_SCLH[7-0] SCLH bit field in F/S mode and in the first phase of HS mode; or by the I2C_SCLH[15-8] HSSCLH bit field in the second phase of HS mode.

Table 12-17 lists the t_{LOW} and t_{high} values in master mode only (in slave mode, the I2C controller does not generate the I^{2}C clock).

Table 12-17. I^{2}C t_{LOW} and t_{high}Values of the I^{2}C Clock

Mode	Clock	t_{LOW}	t_{high}
F/S or HS first phase	INTERNAL_CLK = SYS_CLK / (I2C_PSC[7-0] PSC bit field + 1)	(I2C_SCLL[7-0] SCLL bit field value + 7) × INTERNAL_CLK period	(I2C_SCLH [7-0] SCLH bit field value + 5) × I2Ci_INTERNAL_CLK period
HS second phase	SYS_CLK	(I2C_SCLL[15-8] HSSCLL bit field value + 7) × SYS_CLK period	(I2CSCLL [15-8] HSSCLH bit field value + 5) × I2Ci_SYS_CLK period

Note

For HS mode, the I2C_SCLL[15-8] HSSCLL and I2C_SCLL[7-0] SCLL bit fields must be programmed (the first phase of an HS transaction is performed at F/S speed).

For HS mode, the I2C_SCLH[15-8] HSSCLH and I2C_SCLH[7-0] SCLH bit fields must be programmed (the first phase of an HS transaction is performed at F/S speed).

Note

The equations in Table 12-17 give the SCLL timing values for SCLL/SCLH/HSSCLL/HSSCLH at I2C controller outputs. Actual t_{low} and thigh periods may vary depending on the board (the load capacitance on the SCLL signal). If necessary, any adjustments to the SCLL/SCLH/HSSCLL/HSSCLH values must be determined by measurements of actual SCL signal on the board.

CAUTION

During active mode (the I2C_CON[15] I2C_EN bit is set to 1), make no changes to the I2C_SCLL and I2C_SCLH registers. Changes may result in unpredictable behavior.

Table 12-18 lists the register values for obtaining the maximum I^{2}C bit rates and the maximum period of the filtered spikes in F/S mode and HS mode.

Copyright © 2024 Texas Instruments Incorporated

Table 12-18. I2C Register Values for Maximum I2C Bit Rates in I2C F/S, I2C HS Modes

	I²C Mode for			Description
	Standard Mode	**Fast Mode**	**High-Speed Mode**	
SYS_CLK frequency (MHz)	96			
OCP_CLK frequency (MHz)	133			
I2C_PSC[7-0] PSC	**23**	**9**	**1**	Prescaler value for F/S and HS modes
INTERNAL_CLK frequency (MHz)	4	9.6	96	
I2C_SCLL[7-0] SCLL	**13**	**7**	**115**	Value for F/S mode and first phase of HS mode
I2C_SCLH[7-0] SCLH	**15**	**5**	**113**	Value for F/S mode and first phase of HS mode
Maximum bit rate (Mbps)	0.1	0.4	0.4	F/S mode and first phase in HS mode maximum bit rate
Maximum filter period (ns)	250	104.2	10	
I2C_SCLL[15-8] HSSCLL			**12**	Values for second phase of HS mode
I2C_SCLH[15-8] HSSCLH			**5**	Values for second phase of HS mode
HS mode maximum bit rate (Mbps)			3.31	HS mode maximum bit rate
Maximum filter period (ns)			10	

Note

This table presents informative values only for the configuration parameters and the I²C bus performance obtained according to these values. The delays added by the analog pads are not considered in these figures.

Note

For WKUP_I2C0

For MCU_I2C[0-1]

For I2C[0-6]

I2Ci_INTERNAL_CLK freq = I2Ci_SYS_CLK / (PSC +1)

F/S filter period = 1 / I2Ci_INTERNAL_CLK

HS filter period = 1 / I2Ci_SYS_CLK freq

HS bit rate = I2Ci_SYS_CLK freq / (HSSCLL+ 7 + HSSCLH + 5)

FS bit rate = I2Ci_INTERNAL_CLK / (SCLL+ 7 + SCLH + 5)

12.1.3.3.2.2 I2C Automatic Blocking of the I2C Clock Feature

This feature offers the possibility for the LH to command the blocking of the I²C clock after the target addressing phase, when the I2C controller is addressed by an external controller device using a certain Own Address.

The release of the I²C clock can be performed independently for each Own Address (I2C_OA, and I2C_OAx registers, where x = 1, 2, 3) by deasserting the corresponding bit in the I2C_SBLOCK register.

12.1.3.3.3 I2C Software Reset

Each multicontroller I2C supports the software reset by accessing the I2C_SYSC[1] SRST bit (1: reset; 0: normal mode).

The software reset status can be checked by accessing the I2C_SYSS[0] RDONE bit (1: reset is done; 0: reset is ongoing).

Copyright © 2024 Texas Instruments Incorporated

To do a software reset, the following steps must be done:

1. Ensure that the module is disabled (clear the I2C_CON[15] I2C_EN bit to 0).
2. Set the I2C_SYSC[1] SRST bit to 1.
3. Enable the module by setting I2C_CON[15] I2C_EN bit to 1.
4. Check the I2C_SYSS[0] RDONE bit until it is set to 1 to indicate the software reset is complete.

Note

The I2C_CON[15] I2C_EN bit can hold the functional clock domain of the multicontroller I2C in reset after the device reset has been released. When the system bus reset is removed, this bit remains cleared. The functional part of the I2C controller is held in reset state while this bit is 0, and all configuration registers can be accessed.

The I2C_CON[15] I2C_EN bit must be set to 1 to enable the functional part of the I2C controller.

The I2C_SYSS[0] RDONE bit is asserted only after the module is enabled by setting the I2C_CON[15] I2C_EN bit to 1.

12.1.3.3.4 I2C Power Management

Table 12-19 describes power-management features available for the multimaster I2C controllers.

Note

For information about source clock gating, see *Power*, in the *Device Configuration*.

Note

Some of the I2C features described in this section may not be supported on this family of devices. For more information, see *I2C Not Supported Features*.

Table 12-19. I2C Local Power-Management Features

Feature	Registers	Description
Clock auto gating	I2C_SYSC[0] AUTOIDLE	This bit allows a local power optimization inside the module.
Slave idle modes	I2C_SYSC[4-3] IDLEMODE	Force-idle, no-idle, smart-idle, and smart-idle wakeup-capable modes are available.
Clock activity	I2C_SYSC[9-8] CLOCKACTIVITY	For configuration details, see Table 12-20.
Global wake-up enable	I2C_SYSC[2] ENAWAKEUP	This bit enables the wake-up feature at module level.

Table 12-20. I2C Clock Activity Settings

I2C_SYSC[9-8] CLOCKACTIVITY	Clock State When Module is in IDLE State		Features Available/Unavailable When Module is in IDLE State
	OCP_CLK	SYS_CLK	
00	OFF	OFF	Both clocks are disabled.
10	OFF	ON	Interface clock is disabled; Functional clock is enabled
01	ON	OFF	Functional clock is disabled; Interface clock is enabled
11	ON	ON	Both clocks are enabled.

12.1.3.3.5 I2C Interrupt Requests

The I2C controller must allocate a minimum of five registers for interrupts.

- Interrupt Raw Status (I2C_IRQSTATUS_RAW)
- Interrupt Enabled Status (I2C_IRQSTATUS)
- Interrupt Enable Set (I2C_IRQENABLE_SET)
- Interrupt enable Clear (I2C_IRQENABLE_CLR)
- End of interrupt (I2C_EOI)

End of Interrupt (I2C_EOI) is used by software to trigger an interrupt service completion.

Copyright © 2024 Texas Instruments Incorporated

Table 12-21 lists the event flags, and their mask, that can cause module interrupts.

Table 12-21. I2C Events

Event Flag	Event Unmask	Event Mask	Description
I2C_IRQSTATUS[0] AL	I2C_IRQENABLE_SET[0] AL_IE	I2C_IRQENABLE_CLR[0] AL_IE	Arbitration lost. This bit is automatically set by the hardware when it loses the arbitration in controller transmit mode, an interrupt is signaled to the host.
I2C_IRQSTATUS[1] NACK	I2C_IRQENABLE_SET[1] NACK_IE	I2C_IRQENABLE_CLR[1] NACK_IE	No acknowledgement. Bit is set when No Acknowledge is received, an interrupt is signaled to the host.
I2C_IRQSTATUS[2] ARDY	I2C_IRQENABLE_SET[2] ARDY_IE	I2C_IRQENABLE_CLR[2] ARDY_IE	Register access ready. When set to 1 it indicates that previous access has been performed and registers are ready to be accessed again. An interrupt is signaled to the host.
I2C_IRQSTATUS[3] RRDY	I2C_IRQENABLE_SET[3] RRDY_IE	I2C_IRQENABLE_CLR[3] RRDY_IE	Receive data ready. Set to 1 by core when in receiver mode, a new data can be read. An interrupt is signaled to the host.
I2C_IRQSTATUS[4] XRDY	I2C_IRQENABLE_SET[4] XRDY_IE	I2C_IRQENABLE_CLR[4] XRDY_IE	Transmit data ready. Set to 1 by core when transmitter is ready for new data. An interrupt is signaled to the host.
I2C_IRQSTATUS[5] GC	I2C_IRQENABLE_SET[5] GC_IE	I2C_IRQENABLE_CLR[5] GC_IE	General call. Set to 1 by core when General Call address was detected. An interrupt is signaled to the host.
I2C_IRQSTATUS[6] STC	I2C_IRQENABLE_SET[6] STC_IE	I2C_IRQENABLE_CLR[6] STC_IE	Start condition detected. An interrupt is signaled to the host.
I2C_IRQSTATUS[7] AERR	I2C_IRQENABLE_SET[7] AERR_IE	I2C_IRQENABLE_CLR[7] AERR_IE	Bus Access Error. An interrupt is signaled to the host.
I2C_IRQSTATUS[8] BF	I2C_IRQENABLE_SET[8] BF_IE	I2C_IRQENABLE_CLR[8] BF_IE	Bus free. An interrupt is signaled to the host.
I2C_IRQSTATUS[9] AAS	I2C_IRQENABLE_SET[9] AAS_IE	I2C_IRQENABLE_CLR[9] AAS_IE	Address recognized as target. An interrupt is signaled to the host.
I2C_IRQSTATUS[10] XUDF	I2C_IRQENABLE_SET [10] XUDF_IE	I2C_IRQENABLE_CLR[10] XUDF_IE	Transmit underflow. An interrupt is signaled to the host.
I2C_IRQSTATUS[11] ROVR	I2C_IRQENABLE_SET [11] ROVR_IE	I2C_IRQENABLE_CLR[11] ROVR_IE	Receive overrun. An interrupt is signaled to the host.
I2C_IRQSTATUS[12] BB	N/A	N/A	Bus busy indicator
I2C_IRQSTATUS[13] RDR	I2C_IRQENABLE_SET [13] RDR_IE	I2C_IRQENABLE_CLR[13] RDR_IE	Receive draining. An interrupt is signaled to the host.
I2C_IRQSTATUS[14] XDR	I2C_IRQENABLE_SET [14] XDR_IE	I2C_IRQENABLE_CLR[14] XDR_IE	Transmit draining. An interrupt is signaled to the host.

12.1.3.3.6 I2C Programmable Multitarget Channel Feature

This feature allows each multicontroller I2C to be addressed using four separate Own Addresses configured in the I2C_OA and I2C_OAx registers (where x = 1, 2, 3). An additional register (I2C_ACTOA) is used to indicate to the LH which address is used by the external controller to communicate with the I2C controller.

Each Own Address can be independently configured in 7-bit or 10-bit mode by setting the corresponding bit (I2C_CON[7] XOA0, I2C_CON[6] XOA1, I2C_CON[5] XOA2, or I2C_CON[4] XOA3).

12.1.3.3.7 I2C FIFO Management

Each multicontroller I2C implements two internal 8-bit FIFOs, the RX and TX FIFOs.

The depth of the RX and TX FIFOs can be checked by reading the I2C_BUFSTAT[15-14] FIFODEPTH bit field (0x0: 8 bytes, 0x1: 16 bytes, 0x2: 32 bytes, and 0x3: 64 bytes).

Copyright © 2024 Texas Instruments Incorporated

12.1.3.3.7.1 I2C FIFO Interrupt Mode

In FIFO interrupt mode (relevant interrupts enabled by the I2C_IRQENABLE_SET register), an interrupt signal informs the processor of the receiver and transmitter status. These interrupts are raised when the RX/TX FIFO thresholds (defined by the I2C_BUF[13-8] RXTRSH bit field value + 1 for the RX FIFO or the I2C_BUF[5-0] TXTRSH bit field value + 1 for the TX FIFO) are reached; the interrupt signals instruct the LH to transfer data to the destination (from the I2C controller in receive mode and/or from any source to the I2C controller FIFO in transmit mode).

Figure 12-8 and Figure 12-9 show receive and transmit operations, respectively, from a FIFO management point of view.

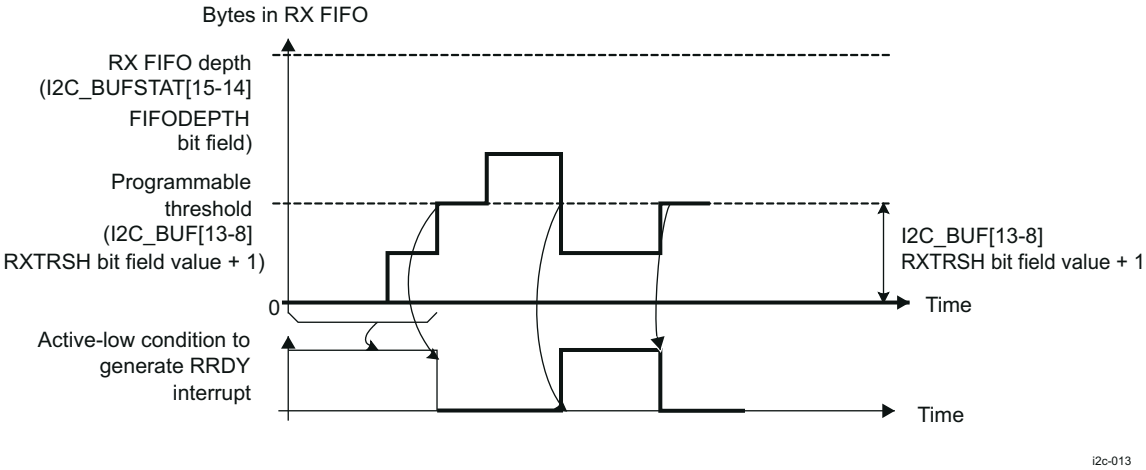


Figure 12-8. I2C Receive FIFO Interrupt Request Generation

In Figure 12-8 , the RRDY interrupt condition shows that the condition for generating an RRDY interrupt is achieved. The interrupt request is generated when this signal is active, and it can be cleared only by the LH by writing 1 in the corresponding bit. If the condition is still present after clearing the previous interrupt, another interrupt request is generated.

In receive mode, an RRDY interrupt is generated as soon as the FIFO reaches its receive threshold (I2C_BUF[13-8] RXTRSH bit field value + 1). The interrupt can be deasserted only when the LH has handled enough bytes to make the number of bytes in the RX FIFO lower than the programmed threshold. For each interrupt, the LH can be configured to read a number of bytes equal to the value of the RX FIFO threshold.

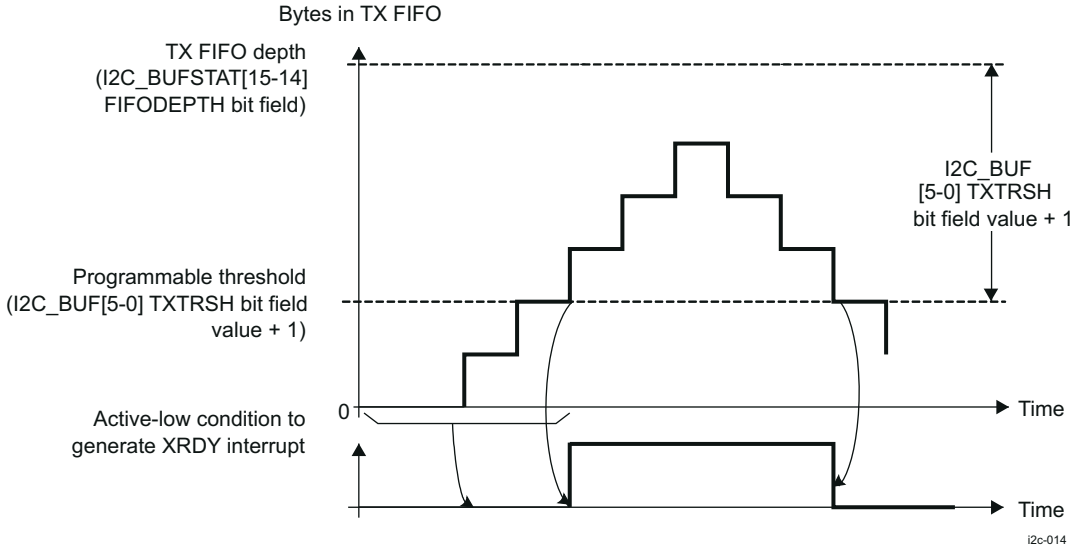


Figure 12-9. I2C Transmit FIFO Interrupt Request Generation

Copyright © 2024 Texas Instruments Incorporated

In Figure 12-9 , the XRDY interrupt condition shows that the condition for generating an XRDY interrupt is achieved. The interrupt request is generated when TX FIFO is empty or when the TX FIFO threshold is not reached, and the LH can clear the XRDY status bit by setting the I2C_IRQENABLE_CLR [4] XRDY_IE bit to 1 after transmitting the configured number of bytes. If the condition is still present after clearing the previous interrupt, another interrupt request is generated.

In interrupt mode, the module offers two options for the LH application to handle the interrupts:

- When detecting an interrupt request (XRDY or RRDY type), the LH can write/read 1 data byte to/from the TX/RX FIFO and then clear the interrupt. The module reasserts the interrupt until the interrupt condition is not met.
- When detecting an interrupt request (XRDY or RRDY type), the LH can be programmed to write/read the amount of data bytes specified by the corresponding FIFO threshold (I2C_BUF[5-0] TXTRSH + 1 or I2C_BUF[5-0] RXTRSH + 1). In this case, the interrupt condition is cleared and the next interrupt is asserted again when the XRDY or RRDY condition is met again.

If the second-interrupt-serving approach is used, an additional mechanism (draining feature) is implemented for cases where the transfer length is not a multiple of the FIFO threshold value (see Section 12.1.3.3.7.3, *Draining Feature [I2C Mode Only]*).

Note

In target transmit mode (the I2C_CON[10] MST bit is cleared and the I2C_CON[9] TRX bit is set to 1), the draining feature must not be used, because the transfer length is not known at configuration time, and the external controller can end the transfer at any point by not acknowledging 1 data byte. If the draining feature is used in target transmit mode, data can remain in the TX FIFO without being transmitted over the I^2C bus. In this case, the TX FIFO must be cleared by setting the I2C_BUF[6] TXFIFO_CLR bit.

12.1.3.3.7.2 I2C FIFO Polling Mode

In FIFO polling mode (the I2C_IRQENABLE_SET[4] XRDY_IE and I2C_IRQENABLE_SET[3] RRDY_IE bits are disabled), the status of the module (receiver or transmitter) can be checked by polling the I2C_IRQSTATUS_RAW[4] XRDY and the I2C_IRQSTATUS_RAW[3] RRDY bits (the I2C_IRQSTATUS_RAW[13] RDR and I2C_IRQSTATUS_RAW[14] XDR bits can also be polled if the draining feature is enabled). The I2C_IRQSTATUS_RAW[4] XRDY and I2C_IRQSTATUS_RAW[3] RRDY bits accurately reflect the interrupt conditions described in the discussion of FIFO interrupt mode.

12.1.3.3.7.3 I2C Draining Feature

Note

DMA mode and SCCB Protocol are not supported on this family of devices.

The draining feature is implemented to handle the end of a transfer whose length is not a multiple of the FIFO threshold values (the I2C_BUF[13-8] RXTRSH bit field value + 1 for the RX threshold and the I2C_BUF[5-0] TXTRSH field value + 1 for the TX threshold). It can also transfer the remaining number of bytes (because the threshold is not reached).

This feature prevents the LH or the DMA controller from trying more FIFO accesses than necessary (for example, to generate at the end of a transfer a DMA RX request having fewer bytes in the FIFO than the configured DMA transfer length). Otherwise, an AERR interrupt is generated by the I2C_IRQSTATUS_RAW[7] AERR bit.

The draining mechanism generates an interrupt using the I2C_IRQSTATUS_RAW[13] RDR or I2C_IRQSTATUS_RAW[14] XDR bit at the end of the transfer, informing the LH that it must check the amount of data left to be transferred (the I2C_BUFSTAT[13-8] RXSTAT or I2C_BUFSTAT[5-0] TXSTAT bit fields) and enable the draining feature of the DMA controller by reconfiguring the DMA transfer length according to this value (when the DMA mode is enabled) or perform only the required number of data accesses (when the DMA mode is disabled).

Copyright © 2024 Texas Instruments Incorporated

In receive mode (controller or target), if the RX FIFO threshold (the I2C_BUF[13-8] RXTRSH bit field value + 1) is not reached, but the transfer ends on the I^2C bus and data remains in the RX FIFO (less than the threshold), the receive draining interrupt (the I2C_IRQSTATUS_RAW[13] RDR bit) is asserted to inform the LH that it can read the amount of data in the RX FIFO (the I2C_BUFSTAT[13-8] RXSTAT bit field). The LH performs a number of data read accesses equal to the I2C_BUFSTAT[13-8] RXSTAT bit field (interrupt or polling mode), or reconfigures the DMA controller with the required value to drain the FIFO.

In controller transmit mode, if the TX FIFO threshold (the I2C_BUF[5-0] TXTRSH bit field value + 1) is not reached, but the amount of data remaining to be written in the TX FIFO is less than the threshold, the transmit draining interrupt (the I2C_IRQSTATUS_RAW[14] XDR bit) is asserted to inform the LH that it can read the amount of data remaining to be written in the TX FIFO (the I2C_BUFSTAT[5-0] TXSTAT bit field). The LH must write the required number of data bytes specified by the I2C_BUFSTAT[5-0] TXSTAT bit field value or reconfigure the DMA controller with the value required to transfer the last bytes to the FIFO.

In controller mode, the LH can alternately not check the values of the I2C_BUFSTAT[5-0] TXSTAT and I2C_BUFSTAT[13-8] RXSTAT bit fields, because it can obtain this information internally (by computing the I2C_CNT[15-0] DATACOUNT bit field value modulo I2C_BUF[13-8] RXTRSH or I2C_BUF[5-0] TXTRSH).

By default, the draining feature is disabled; it can be enabled using the I2C_IRQENABLE_SET[14] XDR_IE or I2C_IRQENABLE_SET[13] RDR_IE bits (default disabled) only for transfers with lengths not equal to the threshold values (I2C_BUF[5-0] TXTRSH bit field value + 1 for the TX threshold or the I2C_BUF[13-8] RXTRSH bit field value + 1 for the RX threshold).

12.1.3.3.8 I2C Noise Filter

The noise filter is used to suppress any noise that is 50 ns or less in case of F/S operation modes, and any noise that is 10 ns or less in case of HS mode operation. The noise filter is always one period of the INTERNAL_CLK clock. This way, for HS mode operation (prescaler bypassed), the filter suppresses spikes of less than 10.4 ns.

For standard mode (for example, the I2C_PSC[7-0] PSC bit field = 4), the maximum width of suppressed spikes is 46.1 ns.

To ensure correct filtering, the prescaler must be programmed accordingly by the I2C*i*.I2C_PSC[7-0] PSC bit field.

12.1.3.3.9 I2C System Test Mode

A system test mode is available for multicontroller I2C module testing. This mode is enabled by setting the I2C_SYSTEST[15] ST_EN bit to 1. When this bit is cleared to 0, the I2C controller is configured in normal operation mode.

In system test mode, the I2C_SYSTEST[13-12] TMODE bit field selects the type of test. Table 12-22 lists the tests available for the multicontroller HS I2C controllers.

Table 12-22. I2C List of Tests

I2C_SYSTEST[13-12] TMODE	Test	Description
00	Functional mode	Normal operation mode
01	Reserved (not used)	
10	Test of SCL serial clock line	The SCL line is driven with a permanent clock as if controlled with the parameters set in the I2C_PSC, I2C_SCLL, and I2C_SCLH registers.
11	Loop-back mode + SCL/SDA I/O	In controller transmit mode only, data transmitted out of the I2C_DATA register (write action) is received in the same I2C_DATA register through an internal path through the FIFO buffers. The interrupt request is normally generated if it is enabled. Moreover, the SCL and SDA are controlled with the I2C_SYSTEST[3-0] bits.

Copyright © 2024 Texas Instruments Incorporated

Note

When the I2C_SYSTEST[13-12] TMODE bit field is set to 11, the I2C controller must be configured in I^2C F/S (I2C_CON[13-12] OPMODE set to 00) or I^2C HS mode (I2C_CON[13-12] OPMODE set to 01).

Note

In normal operation mode (the I2C_SYSTEST[15] ST_EN bit cleared to 0), the I2C_SYSTEST[3-0] bits that control the SCL, SDA lines in system test mode are read-only bits.

In system test mode (the I2C_SYSTEST[15] ST_EN bit set to 1), the I2C_IRQSTATUS_RAW[4] XRDY, I2C_IRQSTATUS_RAW[3] RRDY, I2C_IRQSTATUS_RAW[10] XUDF, I2C_IRQSTATUS_RAW[11] ROVR, I2C_IRQSTATUS_RAW[2] ARDY and I2C_IRQSTATUS_RAW[1] NACK status bits can be set to 1 when the I2C_SYSTEST[11] SSB bit is set to 1. Clearing the I2C_SYSTEST[11] SSB bit to 0 does not clear the I2C_IRQSTATUS_RAW bits to 0. The I2C_IRQSTATUS_RAW bit field can be cleared to 0 only by writing 1 in the corresponding bits.

Copyright © 2024 Texas Instruments Incorporated

12.1.3.4 I2C Programming Guide

12.1.3.4.1 I2C Low-Level Programming Models

12.1.3.4.1.1 I2C Programming Model

This section describes the programming model of the multimaster I2C controllers configured in I^2C mode.

12.1.3.4.1.1.1 Main Program

12.1.3.4.1.1.1.1 Configure the Module Before Enabling the I2C Controller

Before enabling the I2C controller, perform the following steps:

1. Enable the functional and interface clocks (see *WKUP I2C Clocks and Resets*, *MCU_I2C Clocks and Resets*, and *I2C Clocks and Resets*).
2. Program the prescaler to obtain an approximately 12-MHz internal sampling clock by programming the corresponding value in the I2C_PSC[7-0] PSC bit field. This value depends on the frequency of the functional clock (SYS_CLK).
3. Program the I2C_SCLL[7-0] SCLL and I2C_SCLH[7-0] SCLH bit fields to obtain a bit rate of 100 Kbps or 400 Kbps. These values depend on the internal sampling clock frequency (see Table 12-17).
4. (Optional) Program the I2C_SCLL[15-8] HSSCLL and I2C_SCLH[15-8] HSSCLH bit fields to obtain a bit rate of 400 Kbps or 3.4 Mbps (for the second phase of HS mode). These values depend on the internal sampling clock frequency (see Table 12-17).
5. Configure the Own Address of the I2C controller by storing it in the I2C_OA register. Up to four Own Addresses can be programmed in the I2C_OA and I2C_OAx registers (where x = 1, 2, 3) for each I2C controller.

   Note

   For a 10-bit address, set the corresponding expand Own Address bit in the I2C_CON register.

6. Set the TX threshold (in transmitter mode) and the RX threshold (in receiver mode) by setting the I2C_BUF[5-0] TXTRSH bit field to (TX threshold – 1) and the I2C_BUF[13-8] RXTRSH bit field to (RX threshold – 1), where the TX and RX thresholds are greater than or equal to 1.
7. Take the I2C controller out of reset by setting the I2C_CON[15] I2C_EN bit to 1.

12.1.3.4.1.1.1.2 Initialize the I2C Controller

To initialize the I2C controller, perform the following steps:

1. Configure the I2C_CON register:
   - For controller or target mode, set the I2C_CON[10] MST bit (0: target; 1: controller).
   - For transmitter or receiver mode, set the I2C_CON[9] TRX bit (0: receiver; 1: transmitter).
2. If using an interrupt to transmit and receive data, set the corresponding bit in the I2C_IRQENABLE_SET register to 1 (the I2C_IRQENABLE_SET [4] XRDY_IE bit for the transmit interrupt, the I2C_IRQENABLE_SET [3] RRDY bit for the receive interrupt).

12.1.3.4.1.1.1.3 Configure Target Address and the Data Control Register

In controller mode, configure the target address register by programming the I2C_SA[9-0] SA bit field and the number of data bytes (I^2C data payload) associated with the transfer by programming the I2C_CNT[15-0] DCOUNT bit field.

Note

For a 10-bit address, set the I2C_CON[8] XSA bit to 1.

12.1.3.4.1.1.1.4 Initiate a Transfer

Poll the I2C_IRQSTATUS_RAW [12] BB bit. If it is cleared to 0 (bus not busy), configure the I2C_CON[0] STT and I2C_CON[1] STP bits. To initiate a transfer, the I2C_CON[0] STT bit must be set to 1, and it is not mandatory to set the I2C_CON[1] STP bit to 1.

Copyright © 2024 Texas Instruments Incorporated

12.1.3.4.1.1.1.5 Receive Data

Poll the I2C_IRQSTATUS_RAW [3] RRDY bit, or use the RRDY interrupt (the I2C_IRQENABLE_SET [3] RRDY_IE bit must be set to 1) to read the receive data in the I2C_DATA register.

If the transfer length does not equal the RX FIFO threshold (the I2C_BUF[13-8] RTRSH bit field + 1), use the draining feature (enable the RDR interrupt by setting the I2C_IRQENABLE_SET [13] RDR_IE bit to 1).

Note

In receive mode only, the I2C_IRQSTATUS_RAW [11] ROVR (receive overrun) bit indicates whether the receiver has experienced overrun. An overrun condition occurs when the shift register and the RX FIFO are full. An overrun condition does not result in data loss; the I2C controller simply holds SCL to low to prevent other bytes from being received.

The I2C_IRQSTATUS_RAW[7] AERR bit is set to 1 when a read access is performed in the I2C_DATA register while the RX FIFO is empty. The corresponding interrupt can be enabled by setting the I2C_IRQENABLE_SET [7] AERR_IE bit to 1.

12.1.3.4.1.1.1.6 Transmit Data

Poll the I2C_IRQSTATUS_RAW [4] XRDY bit, or use the XRDY interrupt (the I2C_IRQENABLE_SET [4] XRDY_IE bit must be set to 1) to write data to the I2C_DATA register.

If the transfer length does not equal the TX FIFO threshold (the I2C_BUF[5-0] TXTRSH bit field + 1), use the draining feature (enable the XDR interrupt by setting the I2C_IRQENABLE_SET [14] XDR_IE bit to 1).

Note

In transmit mode only, the I2C_IRQSTATUS_RAW [10] XUDF bit indicates whether the transmitter has experienced underflow.

In controller transmit mode, underflow occurs when the shift register and the TX FIFO are empty and there are still some bytes to transmit (the value of the I2C_CNT[15-0] DCOUNT bit field is not 0).

In target transmit mode, underflow occurs when the shift register and the TX FIFO are empty and the external I^2C controller device still requests data bytes to be read.

The I2C_IRQSTATUS_RAW [7] AERR bit is set to 1 when a write access is performed in the I2C_DATA register while the TX FIFO is full. The corresponding interrupt can be enabled by setting the I2C_IRQENABLE_SET [7] AERR_IE bit to 1.

12.1.3.4.1.1.2 Interrupt Subroutine Sequence

1. Test for arbitration lost (the I2C_IRQSTATUS_RAW [0] AL bit) and resolve accordingly.
2. Test for no acknowledgment (the I2C_IRQSTATUS_RAW [1] NACK bit) and resolve accordingly.
3. Test for register access ready (the I2C_IRQSTATUS_RAW [2] ARDY bit) and resolve accordingly.
4. Test for receive data ready (the I2C_IRQSTATUS_RAW [3] RRDY bit) and resolve accordingly.
5. Test for transmit data ready (the I2C_IRQSTATUS_RAW [4] XRDY bit) and resolve accordingly.
6. Test for general call (the I2C_IRQSTATUS_RAW [5] GC bit) and resolve accordingly.
7. Test for start (S) condition (the I2C_IRQSTATUS_RAW [6] STC bit) and resolve accordingly. For this test, the functional clock must be inactive.
8. Test for access error (the I2C_IRQSTATUS_RAW [7] AERR bit) and resolve accordingly.
9. Test for bus free (the I2C_IRQSTATUS_RAW [8] BF bit) and resolve accordingly.

12.1.3.4.1.1.3 Programming Flow-Diagrams

Figure 12-10 through Figure 12-16 are procedure flow charts for programming the F/S and HS I^2C modes.

Copyright © 2024 Texas Instruments Incorporated

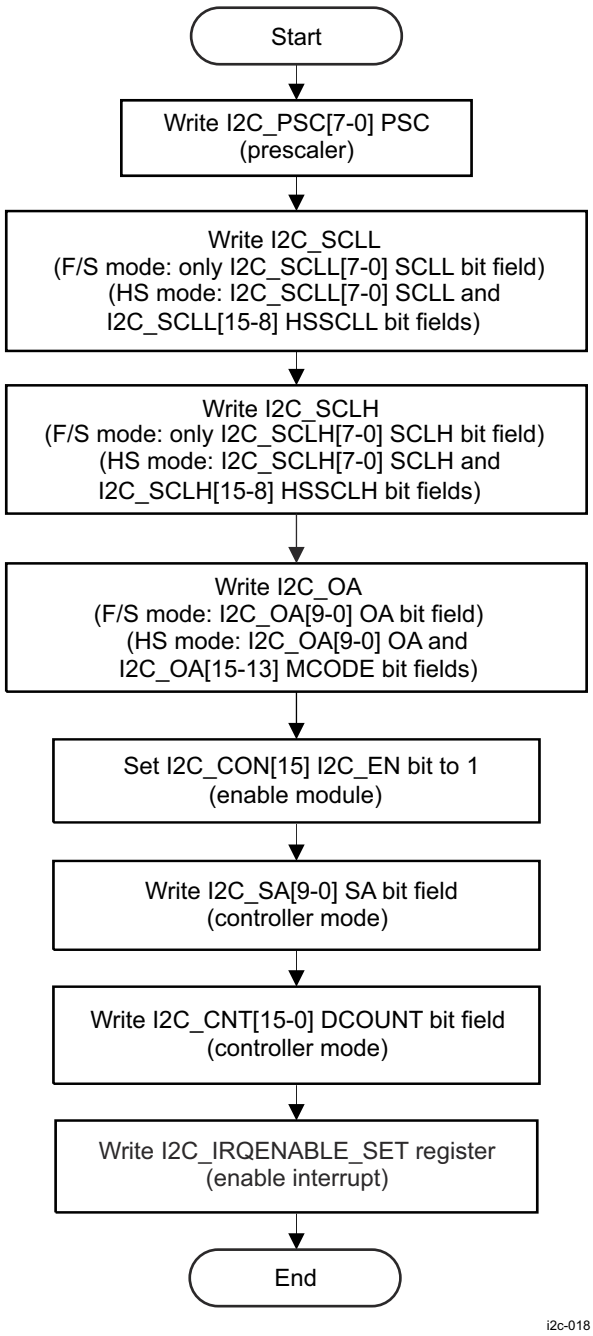
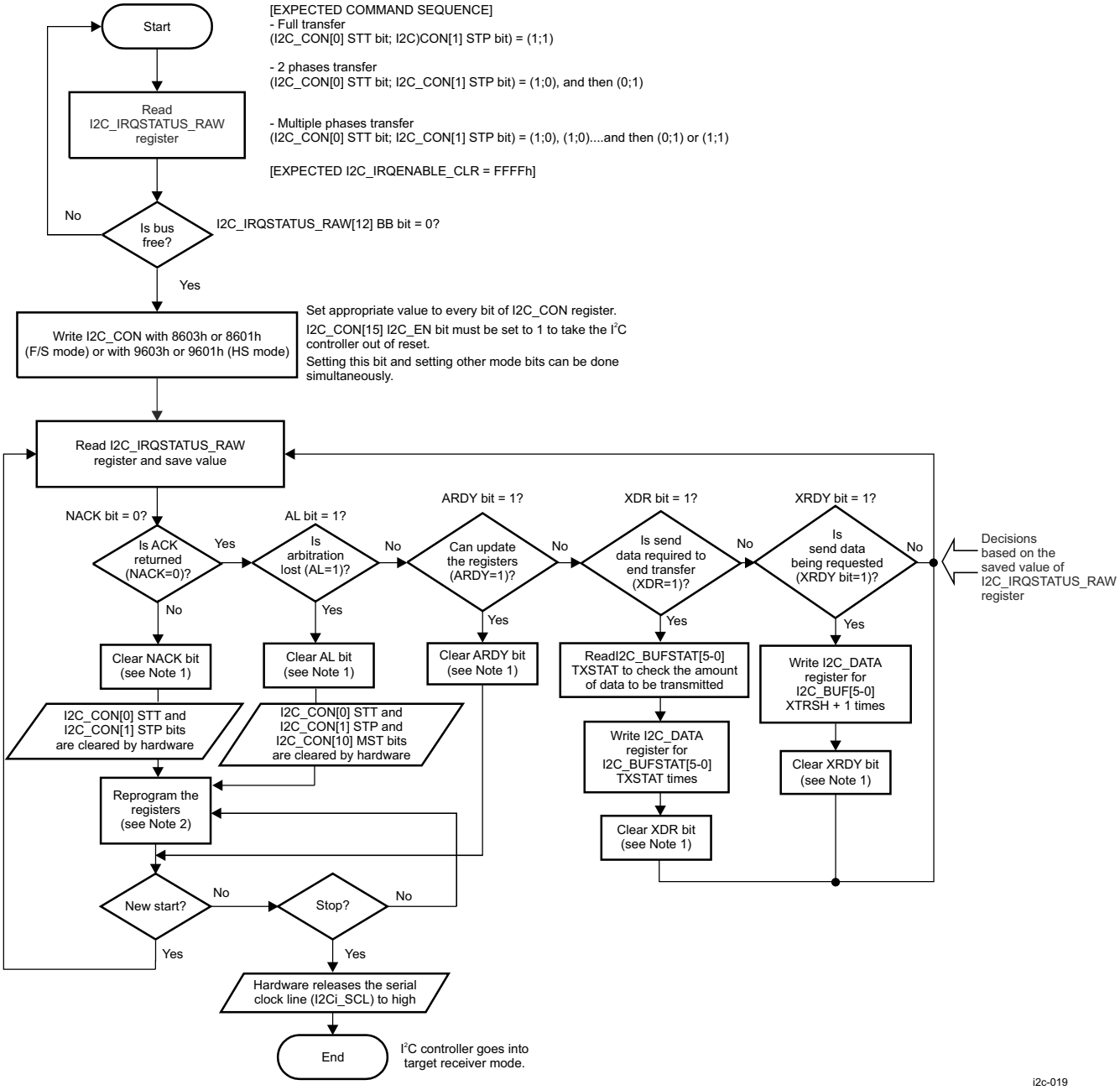


Figure 12-10. I2C Setup Procedure

Copyright © 2024 Texas Instruments Incorporated



A. The NACK, AL, ARDY, XDR, and XRDY bits are cleared by writing 1 to each corresponding bit in the I2C_IRQSTATUS register.

B. Reprogram the registers means: I2C_CON[11] STB and/or I2C_CON[10] MST bit and/or I2C_SA[9-0] SA register and/or I2C_CNT[15-0] DCOUNT register and/or I2C_CON[0] STT bit and/or I2C_CON[1] STP bit.

Figure 12-11. I2C Controller Transmitter Mode, Polling Method, in F/S and HS Modes

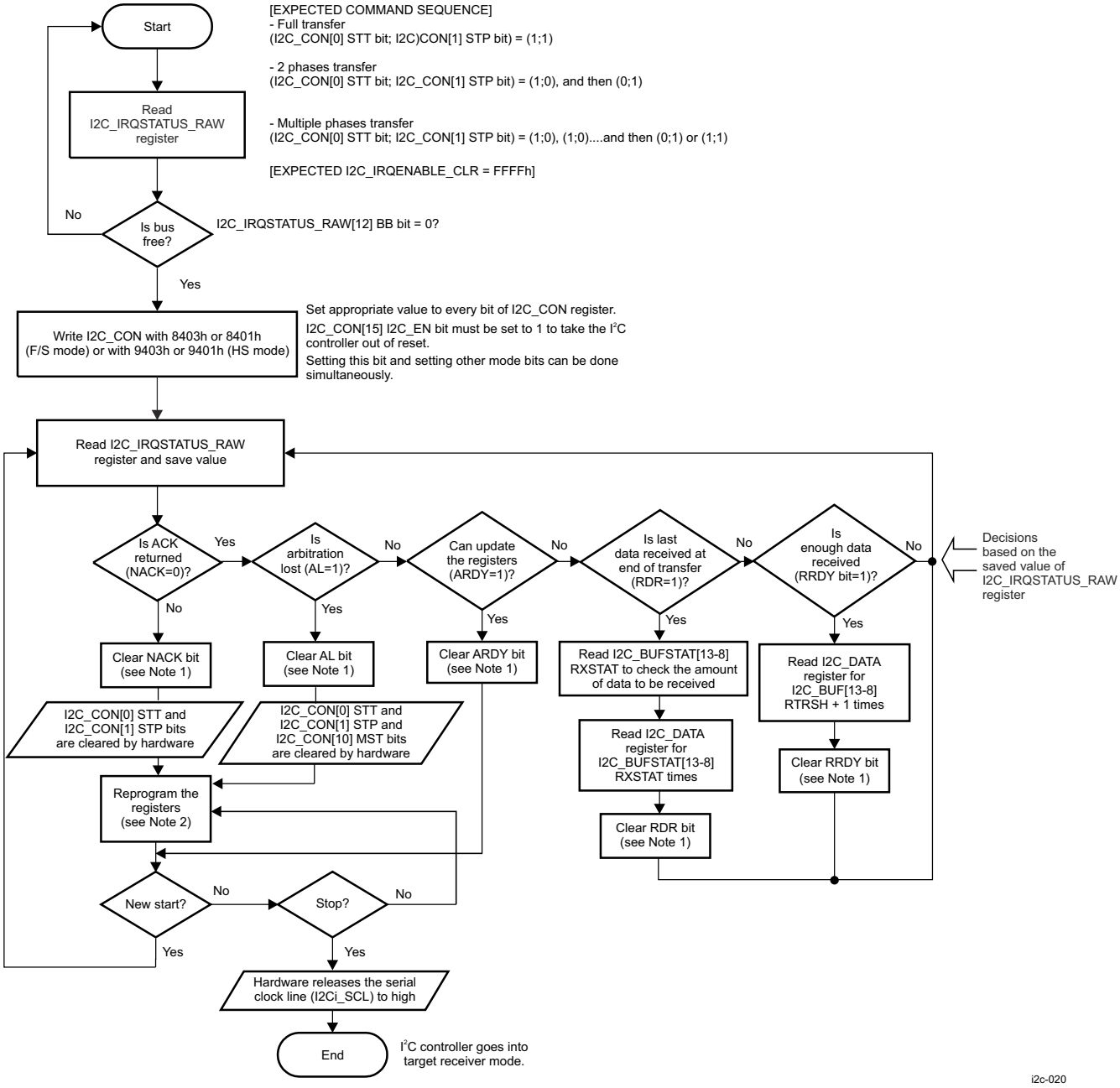
Note

The FIFO clearing can be made when the module is configured as transmitter, the receiver send a NACK in the middle of the transfer, and there is still data in the FIFO.

Note

In HS mode, the Sr condition and clock frequency switching are automatically generated by the multicontroller I2C.

Copyright © 2024 Texas Instruments Incorporated

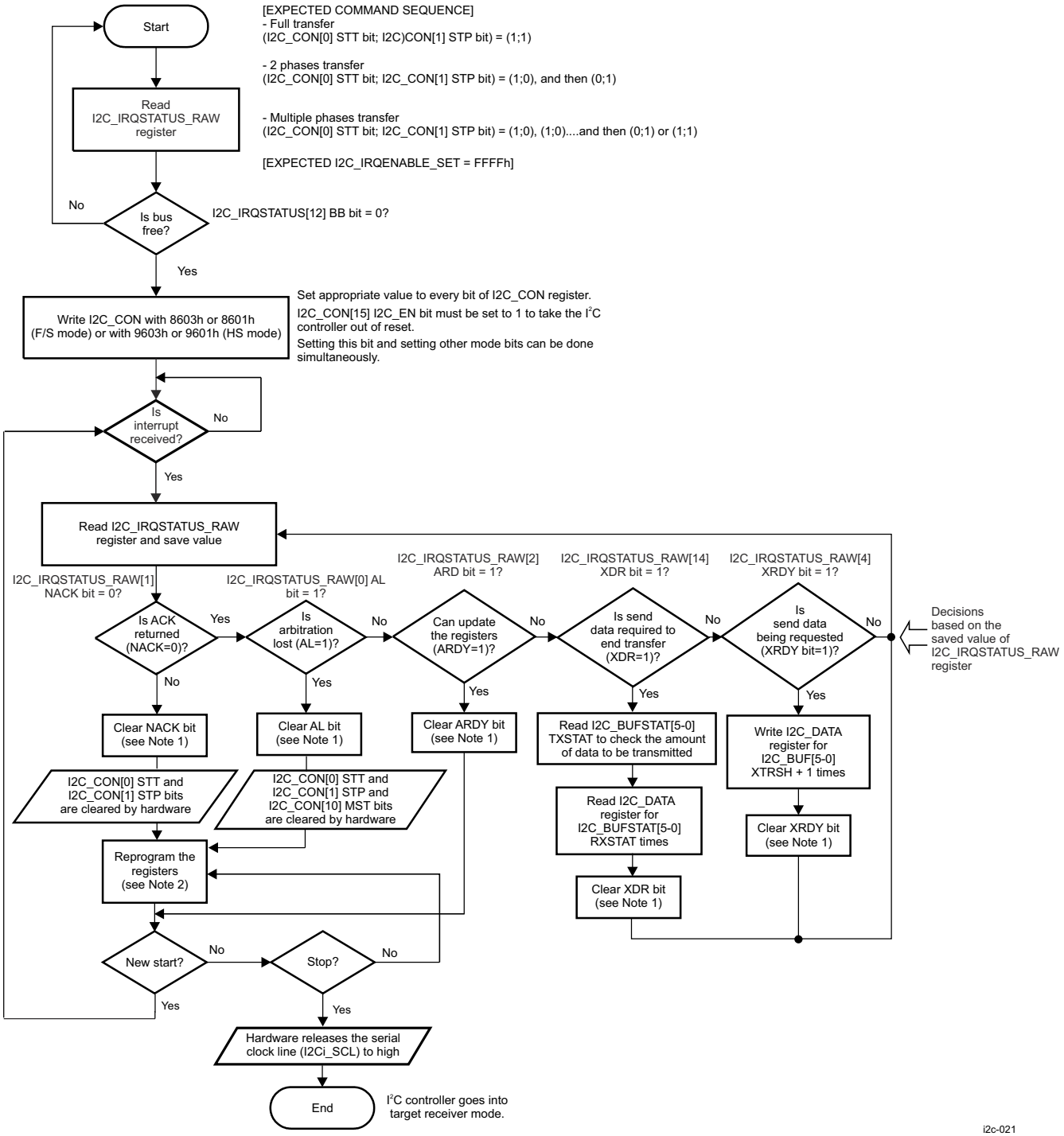


A. The NACK, AL, ARDY, RDR, and RRDY bits are cleared by writing 1 to each corresponding bit in the I2C_IRQSTATUS register.

B. Reprogram registers means: I2C_CON[11] STB and/or I2C_CON[10] MST bit and/or I2C_SA[9-0] SA register and/or I2C_CNT[15-0] DCOUNT register and/or I2C_CON[0] STT bit and/or I2C_CON[1] STP bit.

Figure 12-12. I2C Controller Receiver Mode, Polling Method, in F/S and HS Modes

Copyright © 2024 Texas Instruments Incorporated



A. The NACK, AL, ARDY, XDR, and XRDY bits are cleared by writing 1 to each corresponding bit in the I2C_IRQSTATUS register.

B. Reprogram registers means: I2C_CON[11] STB and/or I2C_CON[10] MST bit and/or I2C_SA[9-0] SA register and/or I2C_CNT[15-0] DCOUNT register and/or I2C_CON[0] STT bit and/or I2C_CON[1] STP bit.

Figure 12-13. I2C Controller Transmitter Mode, Interrupt Method, in F/S and HS Modes

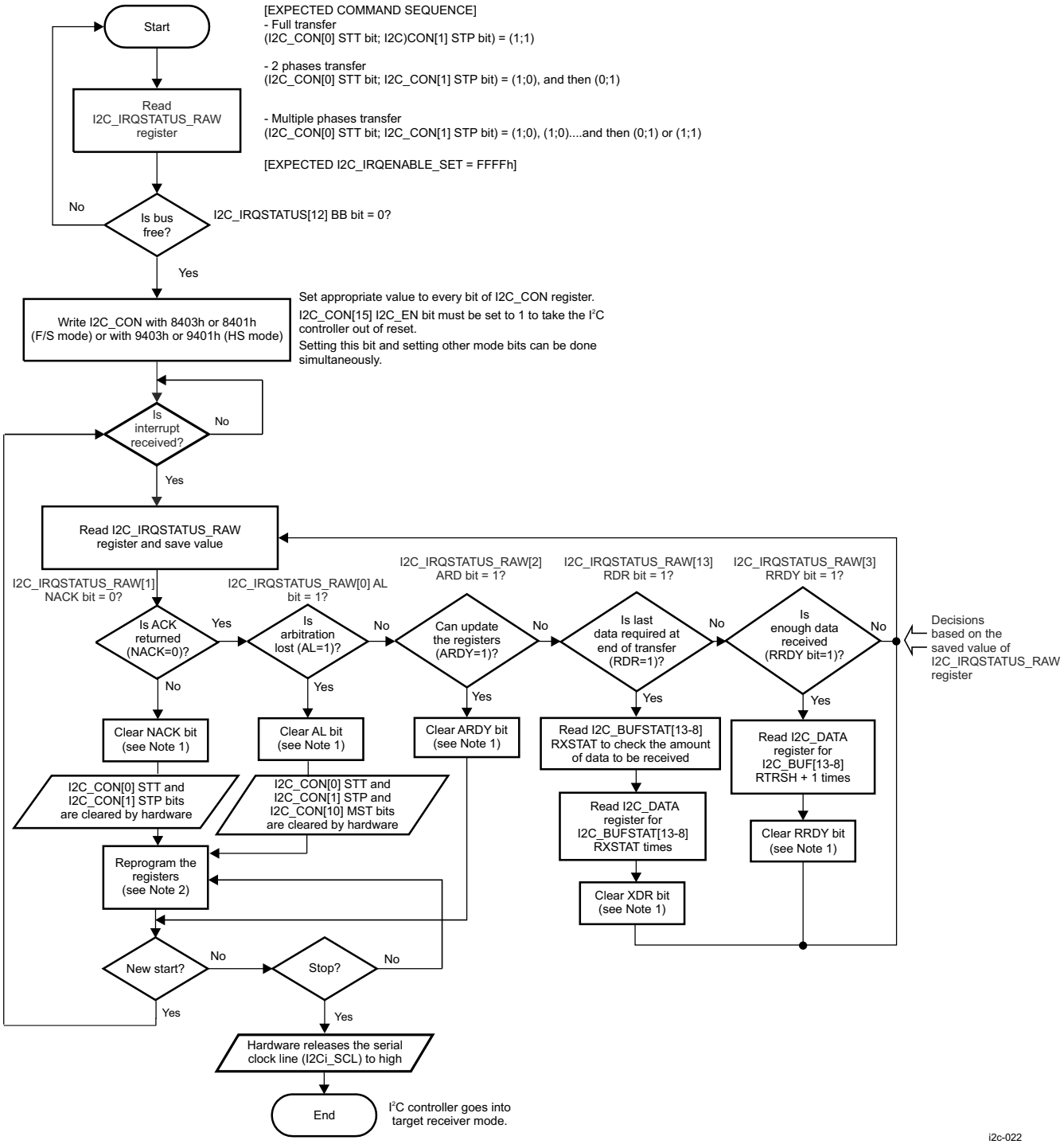
Note

The FIFO clearing can be made when the module is configured as transmitter, the receiver send a NACK in the middle of the transfer, and there is still data in the FIFO.

Copyright © 2024 Texas Instruments Incorporated

Note

In HS mode, the Sr condition and clock frequency switching are automatically generated by the multicontroller I2C.

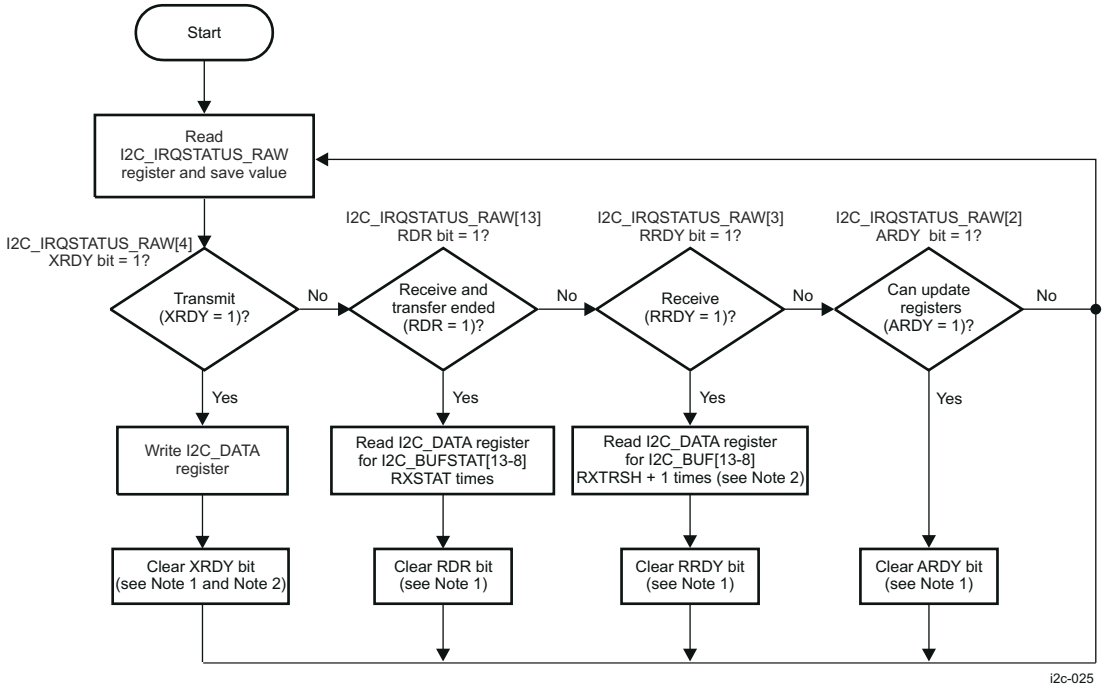


A. The NACK, AL, ARDY, RDR, and RRDY bits are cleared by writing 1 to each corresponding bit in the I2C_IRQSTATUS register.

B. Reprogram registers means: I2C_CON[11] STB and/or I2C_CON[10] MST bit and/or I2C_SA[9-0] SA register and/or I2C_CNT[15-0] DCOUNT register and/or I2C_CON[0] STT bit and/or I2C_CON[1] STP bit.

Figure 12-14. I2C Controller Receiver Mode, Interrupt Method, in F/S and HS Modes

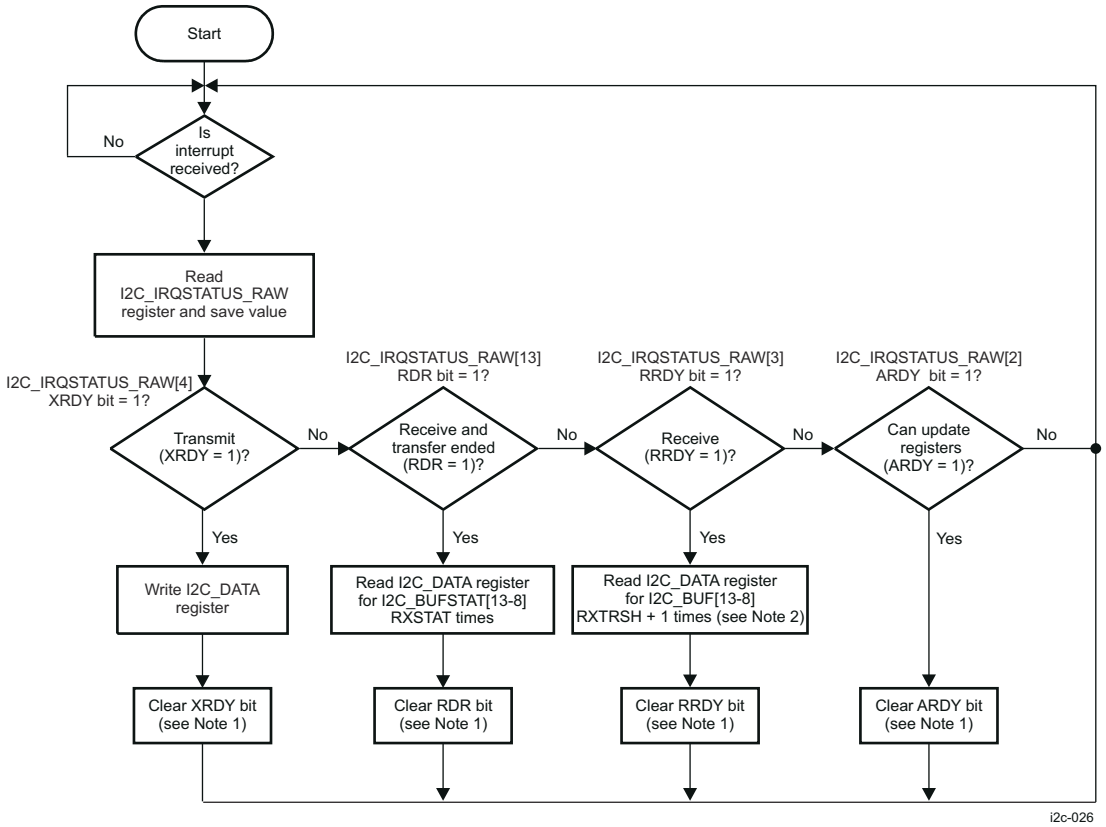
Copyright © 2024 Texas Instruments Incorporated



A. The XRDY, RDR, RRDY, and ARDY bits are cleared by writing 1 to each corresponding bit in the I2C_IRQSTATUS register.

B. In target transmitter mode, the amount of data requested by the external controller I^{2}C device is unknown; thus, the I2C_BUF[5-0] XTRSH bit field must be configured to 0x0 (TX threshold = 1).

Figure 12-15. I2C Target Transmitter/Receiver Mode, Polling



A. The XRDY, RDR, RRDY, and ARDY bits are cleared by writing 1 to each corresponding bit in the I2C_IRQSTATUS register.

Copyright © 2024 Texas Instruments Incorporated

B. In target transmitter mode, the amount of data requested by the external controller I^2C device is unknown; thus, the I2C_BUF[5-0] XTRSH bit field must be configured to 0x0 (TX threshold = 1).

Figure 12-16. I2C Target Transmitter/Receiver Mode, Interrupt

Copyright © 2024 Texas Instruments Incorporated

12.1.4 Improved Inter-Integrated Circuit (I3C) Interface

This section describes the Improved Inter-Integrated Circuit (I3C) module in the device.

12.1.4.1 I3C Overview

The device contains Improved Inter-Integrated Circuit (I3C) an interface between a local host (LH), such as an Arm, and any I3C-bus-compatible device that connects via the I3C serial bus.

For the specific I/O timing characteristics of the I3C , see the device-specific data sheet.

12.1.4.1.1 I3C Features

I3C module has the following features:

- Communication Modes:
  - Single Data Rate (SDR) mode
  - High Data Rate – Dual Data Rate (HD-DDR) mode
- Bus Modes:
  - Pure Bus Mode (supports only I3C devices). Maximum supported frequency is 10.375 MHz
- Master and Multi-Master modes
- Common Command Codes (CCC)
- Bus Enumeration and Discovery:
  - Slaves implement standardized characteristics registers that can be queried by the master after power up to enumerate the devices attached to the bus.
- Hot-Join capability:
  - A slave device can be powered on and join the bus after initial enumeration.
- In Band Interrupts (IBI):
  - Slave devices request interrupts through the I3C bus rather than with separate side-band signals. This reduces the total IO count of the SoC as well as PCB routing.
- Dynamic Address Assignment (DAA):
  - The master can assign each slave device an address during configuration. This supports IBI because interrupt priority among slaves is arbitrated with the same address arbitration scheme that is used for all communication.
- Static Addressing (SA)
- FIFO Buffers:
  - CMD Queueing out going commands:
    - CMD0_FIFO stores the least significant half-word of the command
    - CMD1_FIFO stores the most significant half-word of the command
  - TXFIFO stores data transmitted with each command
  - RXFIFO stores data received as command response
  - IBI_DATA_FIFO stores data received as part of the in-band-interrupt transfer (IBI can transfer 0, 1, or N bytes of data payload depending on the slave)
- Immediate high priority command queue
- Registers to store the parameters for the response to an IBI interrupt from a number of slaves:
  - Each register holds the slave address and appropriate response type and payload length information for two different slaves, so that the controller can process in band interrupts without CPU intervention.

Unsupported Features:

- See the Module Integration section of this document for a list of module features not supported by the integration on this Device.

12.1.4.1.2 I3C Ports

Table 12-23. I3C0 Clocks and Resets

Clocks	
Module Clock Input	**Description**
I3C_PCLK	I3C configuration clock
I3C_SCLK	I3C system clock
Resets	

Copyright © 2024 Texas Instruments Incorporated

Table 12-23. I3C0 Clocks and Resets (continued)

Module Reset Input	Description
I3C_RST	I3C reset

Table 12-24. I3C0 Hardware Requests

Interrupt Requests		
Module Interrupt Signal	**Description**	**Type**
I3C_I3C_INT_0	I3C interrupt request	Level
I3C_PCLK_ECC_UNCORR_LVL_0	I3C PCLK ECC interrupt request	Level
I3C_SCLK_ECC_UNCORR_LVL_0	I3C SCLK ECC interrupt request	Level
I3C_I3C_NONFATAL_INT_0	I3C non fatal interrupt request	Level
I3C_I3C_FATAL_INT_0	I3C fatal interrupt request	Level

12.1.4.2 I3C Environment

This section describes the I3C external connections (environment).

Table 12-25 describes the I3C I/O signals.

Table 12-25. I3C I/O Signals

Module Pin	I/O[(1)]	Description
SCL	I/O	I^3C serial clock line. Emulated open-drain output buffer.
SDA	I/O	I^3C serial data line. Emulated open-drain output buffer.
SDAPULLEN	O	I^3C data pull enable. [(2)]

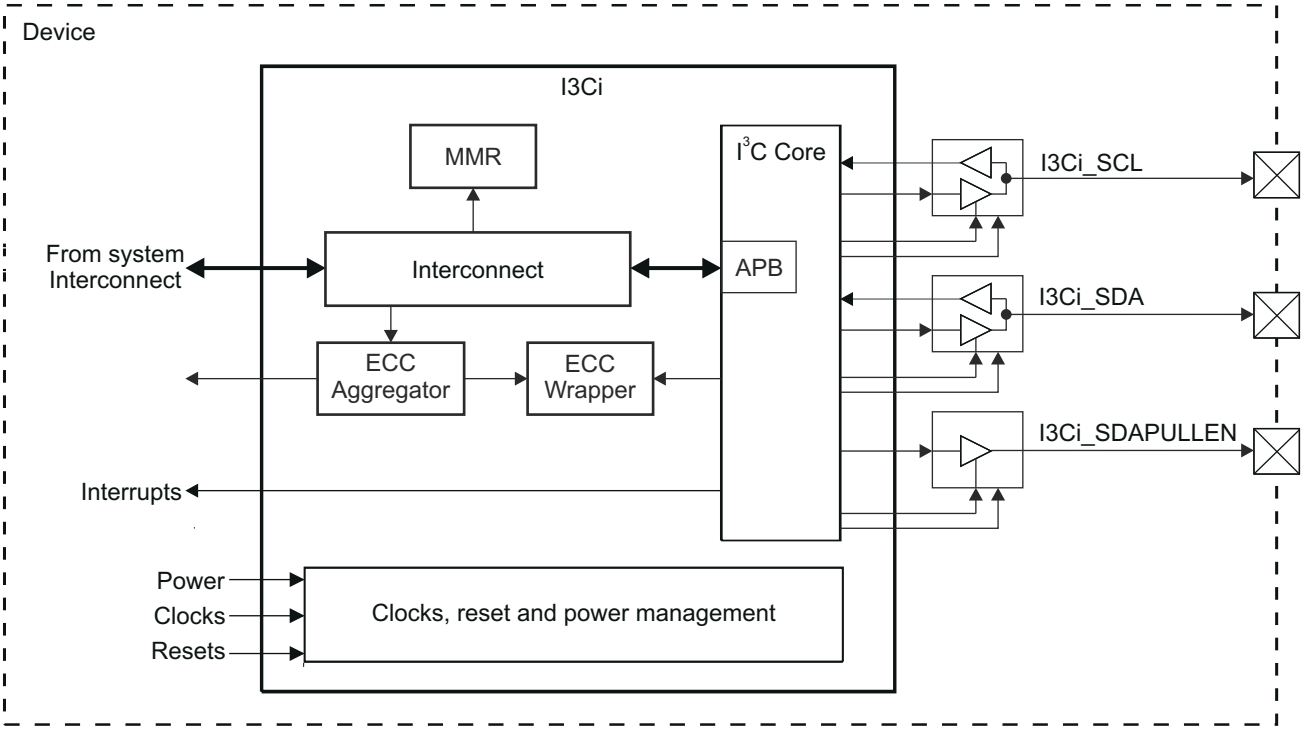
(1) I = Input; O = Output; I/O = Bidirectional
(2) Enable buffer/switch for open drain class pull-up, enabled during open drain mode.

Copyright © 2024 Texas Instruments Incorporated

12.1.4.3 I3C Functional Description

12.1.4.3.1 I3C Block Diagram

Figure 12-17 presents the I3C controller block diagram.



(1) i = 0 for MCU and MAIN domain

Figure 12-17. I3C Block Diagram

The two I3C controllers can be configured in SDR I^{3}C mode or HD-DDR I^{3}C mode. Default operation mode is SDR mode.

Table 12-26 lists the available operation modes.

Table 12-26. I^{3}C Operation Mode Selection

Operation Mode	Value of I3C_CMD0_FIFO[31] IS_DDR
SDR I^{3}C	0x0
HD-DDR I^{3}C	0x1

12.1.4.3.2 I3C Clock Configuration

The I3C controller allows to program time base for SCL clock generation. Programmable frequency dividers are provided for I^{3}C messaging modes independently. Based on type of messaging that is processed, one of the fields placed in I3C_PRESCL_CTRL0 register is used to establish constant ratio between the clocks and SCL clock for all messaging modes. I3C messaging uses the value of I3C field in I3C_PRESCL_CTRL0 register, and is combined with the frequency of inputs. In order to handle timing requirements specified for I^{3}C slaves I3C_PRESCL_CTRL1 register is provided.

Programming the prescalers should be performed prior to enabling I3C controller with I3C_CTRL[31] DEV_EN bit.

12.1.4.3.2.1 Setting Base Frequencies

To obtain desired base frequency of SCL signal for SDR/HDR I^{3}C messaging (I3C_FREQ) the I3C_PRESCL_CTRL0[9-0] I3C bit-field should be programmed with value defined by the following equation:

Figure 12-18. I3C Base Frequency

Copyright © 2024 Texas Instruments Incorporated

Programming the prescalers must be performed prior to enabling I3C Master controller with I3C_CTRL[31] DEV_EN bit in the control register.

In all timing schemes, the base frequency resulting from prescalers corresponds directly to the duration of the high level of the SCL line which takes always 50% of the base period. Depending on various conditions, the duration of low level is extended beyond the base period as necessary.

There is a specific relationship between SCL frequency and system clock frequency, which requires system clock to have a multiple of four cycles per each SCL period. Thus the selection of system clock frequency must be done with caution, since only specific frequencies will allow configuring SCL to fit into desired bandwidth.

Table 12-27 presents achievable SCL frequencies for exemplary set of system clock frequency values.

Table 12-27. Achievable SCL Frequencies

frequency	I3C_PRESCL_CTRL0[9-0] I3C	Actual frequencies
100 MHz	0x2	8.33 MHz
133 MHz	0x3	8.313 MHz
166 MHz	0x3	10.375 MHz
200 MHz	0x4	10 MHz

12.1.4.3.2.2 Asymmetric Push-Pull SCL Timing

In case of I^3C messaging, asymmetric generation pattern for driving SCL can be enabled. The main purpose for this feature is supporting I3C slaves that are not capable to use maximal SCL frequency during private read/write data transfers. Host can recognize limited speed device by examining value of I3C_DEV_IDn_RR2[15-8] bit-field for n = 0 to 11. Maximum frequency supported by slave can be retrieved by GETMSCL command. The low period of SCL is extended by the value programmed in I3C_PRESCL_CTRL1[15-8] PP_LOW bit-field, which can be calculated using the following equation:

Figure 12-19. SCL Asymmetric Push-Pull

The I3C_PRESCL_CTRL1[15-8] PP_LOW resolution is set to ¼ of SCL period. The high period of the SCL remains unchanged and is equal to half period of I^3C frequency.

The low period used in push-pull mode for slaves with limited SCL capability results from following equation:

$$SCL_LOW = I3C_PRESCL_CTRL1[15\text{-}8]\ PP_LOW \times \frac{T}{4}$$

i3c_014

Where:

SCL_LOW: SCL low period.
T: base SCL period.

Figure 12-20. SCL Asymmetric Push-Pull Actual Low Period

12.1.4.3.2.3 Open-Drain SCL Timing

In open-drain mode, due to protocol requirement, low period cannot be shorter than 160ns to support slowly rising SDA line. To fulfill this constrain in I^3C SDR mode, the low period must be always extended beyond the value resulting from the base I^3C frequency. The low period of SCL is extended by the value programmed in I3C_PRESCL_CTRL1[7-0] OD_LOW bit-field, which can be calculated using the following equation:

Figure 12-21. SCL Open-Drain Low Period

The I3C_PRESCL_CTRL1[7-0] OD_LOW bit-field resolution is set to ¼ of SCL period. The high period remains unchanged and is equal to half-period of I^3C frequency.

The actual SCL low period used in open drain mode results from the following equation:

Copyright © 2024 Texas Instruments Incorporated

$$SCL_LOW = I3C_PRESCL_CTRL1[7\text{-}0]\ OD_LOW \times \frac{T}{4}$$

i3c_015

Where:
SCL_LOW: SCL low period.
T: base SCL period.

Figure 12-22. SCL Open-Drain Actual Low Period

12.1.4.3.2.4 Changing Programmed Frequencies

Since SCL clock is derived, in order to change the SCL timing for any mode or to keep current I^3C bus timing, the following procedure should be used:

1. Disable the I3C controller by setting the I3C_CTRL[31] DEV_EN bit to 0.
2. Adjust to the new frequency if necessary, avoiding any glitches during change.
3. Reprogram the prescaler registers to achieve desired I^3C timing.
4. Enable the core by setting the I3C_CTRL[31] DEV_EN bit to 1.

12.1.4.3.3 I3C Interrupt Requests

The I3C controller has two sets of five interrupt registers:

- Interrupt Enable Register (I3C_MST_IER)
- Interrupt Disable Register (I3C_MST_IDR)
- Interrupt Mask Register (I3C_MST_IMR)
- Interrupt Clear Register (I3C_MST_ICR)
- Interrupt Status Register (I3C_MST_ISR)

After reset, status (I3C_MST_ISR) and mask (I3C_MST_IMR) registers both read zeroes, so all interrupts are disabled. CPU handles interrupts by several typical operations:

- To activate an interrupt, interrupt enable register (I3C_MST_IER) should be programmed with 0x1 on bit positions corresponding to interrupts sources which should be signaled to CPU. This interrupt mask is written to I3C_MST_IMR and defines which sources will be propagated.
- After receiving interrupt, I3C_MST_ISR should be read to identify the interrupt source. Since I3C_MST_ISR is always updated by all interrupt sources regardless of I3C_MST_IMR, the I3C_MST_ISR value should be compared to the programmed mask (stored in software or read from I3C_MST_IMR) as it can contain other flags which were not propagated. Interrupt handler should clear flag which triggered interrupt in I3C_MST_ISR by writing 0x1 to corresponding bits of I3C_MST_ICR. It is crucial for correct interrupt source identification to read status, apply mask and clear current interrupt flag before another (from the same or different source) interrupt can occur, so these actions should be executed at the beginning of the interrupt handler routine.
- If during operation additional interrupt sources need to be enabled, clearing their flags prior to enabling is necessary to avoid false signaling of events which occurred earlier.
- To disable interrupt source anytime during operation, corresponding bit in interrupt disable register (I3C_MST_IDR) should be programmed with 0x1, which in turn clears corresponding bit in I3C_MST_IMR, disabling interrupt source.

12.1.4.3.4 I3C Power Configuration

Before I3C controller is put into operation, several settings need to be adjusted after cycling the power on. The general settings collected in I3C_CTRL register. For most common usage scheme, two fields are the most important:

1. I3C_CTRL[31] DEV_EN bit allows enabling and disabling the I^3C bus controller. It must be enabled before any I^3C bus message is initiated, however following settings need to be programmed prior to enabling controller:
   a. Bus Mode (Only Pure Bus Mode is supported)
   b. SCL Divider prescalers (See *I3C Clock Configuration*)
   c. Devices Retaining Registers (See *I3C Retaining Registers Space*)

Copyright © 2024 Texas Instruments Incorporated

During normal operation the controller should stay enabled. Before disabling, the I3C_MST_STATUS0[18] IDLE should be examined to confirm that no bus operation is pending.

2. I3C_CTRL[1-0] BUS_MODE bit-field is set to Pure Bus Mode by default. The I3C controller supports only I^3C compatible devices on I^3C bus and if changed bit-field has no effect. For more information see *I3C Not Supported Features*.

12.1.4.3.5 I3C Dynamic Address Management

Enumeration of I3C devices involves Dynamic Address Assignment procedure. At power up of the I^3C bus, the I^3C Master should provide dynamic address for all I^3C devices under the following conditions:

- If I^3C Slave devices with Static Address are available (not supported for this device), firmware may choose to assign Dynamic Address using SETDASA CCC to shorten bus initialization time; such devices are called Static I^3C Devices.
- For the devices that are enumerated using SETDASA CCC command the Provisional ID, BCR and DCR information should be manually acquired, using GETPID / GETBCR / GETDCR CCC commands. This step is not mandatory if the values are known prior to bus initialization and programmed into I^3C Master by host application.
- Devices with no Static Address and these with Static Address not enumerated receive Dynamic Address using the DAA procedure initiated by issuing the ENTDAA CCC command.
- For the devices that are enumerated using ENTDAA CCC command the Provisional ID, BCR and DCR information are automatically received during DAA procedure.

During I^3C bus operation, current master can disable one or all I3C devices by sending Direct or Broadcast RSTDAA CCC command. In such case, corresponding *_CLR bits in I3C_DEVS_CTRL should be set by firmware to clean retaining registers. In order to re-enable these devices, DAA procedure must be performed (initiated by master or by Hot-Join requests sent by slave devices).

Depending on I^3C slave implementation, the slave that received RSTDAA CCC command may restore its static address. In order to perform private transfers to such slave or assign dynamic address using SETDASA CCC command rather than DAA procedure, the corresponding I3C_DEV_IDn_RR0[7-0] DEV_ADDR bit field (n = 0 to 11) should be updated with static address. Clearing Retaining Registers by setting corresponding I3C_DEVS_CTRL register is not mandatory in this scenario.

Current master can also assign new Dynamic Address to device which already has one by sending SETNEWDA command. After sending this command with new Dynamic Address in the data field, the I3C_DEV_IDn_RR0[7-0] DEV_ADDR bit field (n = 0 to 11) must be also updated with this address in order to perform subsequent transaction to this device.

12.1.4.3.6 I3C Retaining Registers Space

The information on configuration, Provisional ID, BCR and DCR of each device connected to the I^3C bus is stored by the I^3C master in a set of retaining registers divided into Device ID sections associated with each device. Application host take any action to guarantee maintaining this information anytime the bus topology changes.

The retaining register space is divided to up to 12 sections (depending on selected configuration), from I3C_DEV_ID0 to I3C_DEV_ID11. Each containing three 32-bit registers to hold whole necessary information on I3C devices connected to the bus (e.g. hot-join device is attached, some of DA are reset to new value, etc.).

The AMBA APB interface address space for retaining registers starts at 0x080 and is divided into 12 quadruplets.

The map organization is presented in Table 12-28 below.

Table 12-28. Retaining Registers Map Organization

Register	Address	Contents
I3C_DEV_ID0_RR0	0x080	Configuration register which stores settings for device that needs to be programmed by application host.
I3C_DEV_ID0_RR1	0x084	Register stores MSB of Provisional ID (1)
I3C_DEV_ID0_RR2	0x088	Register stores LSB of Provisional ID, DCR, BCR (1)

Copyright © 2024 Texas Instruments Incorporated

Table 12-28. Retaining Registers Map Organization (continued)

Register	Address	Contents
I3C_DEV_ID1_RR0	0x090	Configuration register which stores settings for device that needs to be programmed by application host.
I3C_DEV_ID1_RR1	0x094	Register stores MSB of Provisional ID (1)
I3C_DEV_ID1_RR2	0x098	Register stores LSB of Provisional ID, DCR, BCR (1)
I3C_DEV_ID2_RR0	0x0A0	Configuration register which stores settings for device that needs to be programmed by application host.
I3C_DEV_ID2_RR1	0x0A4	Register stores MSB of Provisional ID (1)
I3C_DEV_ID2_RR2	0x0A8	Register stores LSB of Provisional ID, DCR, BCR (1)
...	...	...
I3C_DEV_ID11_RR0	0x130	Configuration register which stores settings for device that needs to be programmed by application host.
I3C_DEV_ID11_RR1	0x134	Register stores MSB of Provisional ID (1)
I3C_DEV_ID11_RR2	0x138	Register stores LSB of Provisional ID, DCR, BCR (1)

(1) The information is relevant only for I^{3}C devices and is automatically filled by hardware during DAA procedure. Application host needs to fill the information for all devices with Dynamic Addresses Assignment with SETDASA CCC command.

A special device control register (I3C_DEVS_CTRL) is provided to store information on devices currently connected to the bus. Each DeviceID section contains configuration register and two registers holding Provisional ID, BCR and DCR of the corresponding device. The active state bit is set automatically by hardware after Dynamic Address is assigned to given device. Firmware shall set it for devices enumerated with SETDASA CCC command after their Retaining Registers are configured and before any transaction to these devices is initiated.

The clearing bits need to be set by firmware every time RSTDAA CCC is send to corresponding devices (firmware should clear all of them for broadcast RSTDAA) or controller constantly receives NACK response and application recognizes device as no longer connected to the bus.

12.1.4.3.7 I3C Dynamic Address Assignment Procedure

The I3C bus initialization procedure can be decomposed into three main stages, described below:

- **Stage 1:** On startup, the I3C master does not have any information about I^{3}C slave devices and all Retaining Registers are set to the default reset vector table;
- **Stage 2:** Is fully maintained by application host. The first register section (Device ID0) is reserved for I3C master self-description. The following registers starting from Device ID1 will be filled by DAA procedure in Stage 3. This implies rules firmware should follow:
  - I^{3}C devices with Static Address which firmware assume to enumerate with SETDASA CCC command shall be localized in Retaining Registers table (see Figure 12-23) after address space that reserved for I^{3}C devices intended to be enumerated with ENTDAA;
  - Corresponding I3C_DEVS_CTRL bits for active should be set;
- **Stage 3:** Before attempting DAA procedure, firmware should take following actions:
  - For Static I3C Devices intended to be enumerated with SETDASA CCC command:
    - Configuration registers should be programmed with Static Addresses and other settings relevant for I3C device;
    - Corresponding I3C_DEVS_CTRL bits for active should be set;
    - Dynamic Addresses should be assigned using SETDASA CCC command;
    - Configuration registers should be programmed with just assigned Dynamic Addresses;
    - Provisional ID / BCR / DCR Registers (I3C_DEV_IDn_RR1 and I3C_DEV_IDn_RR1, n = 0 to 11) should be programmed with data either predefined by application or obtained by reading PID / BCR / DCR with related CCC;
  - For I^{3}C Devices intended to be enumerated with ENTDAA CCC command:
    - Configuration registers should be programmed with Dynamic Addresses and other settings relevant for I3C Device;

Once all above conditions are satisfied the firmware can issue the ENTDAA CCC command, which is considered as start condition for Stage 3.

Copyright © 2024 Texas Instruments Incorporated

Figure 12-23 presents the I3C bus initialization for DAA procedure.

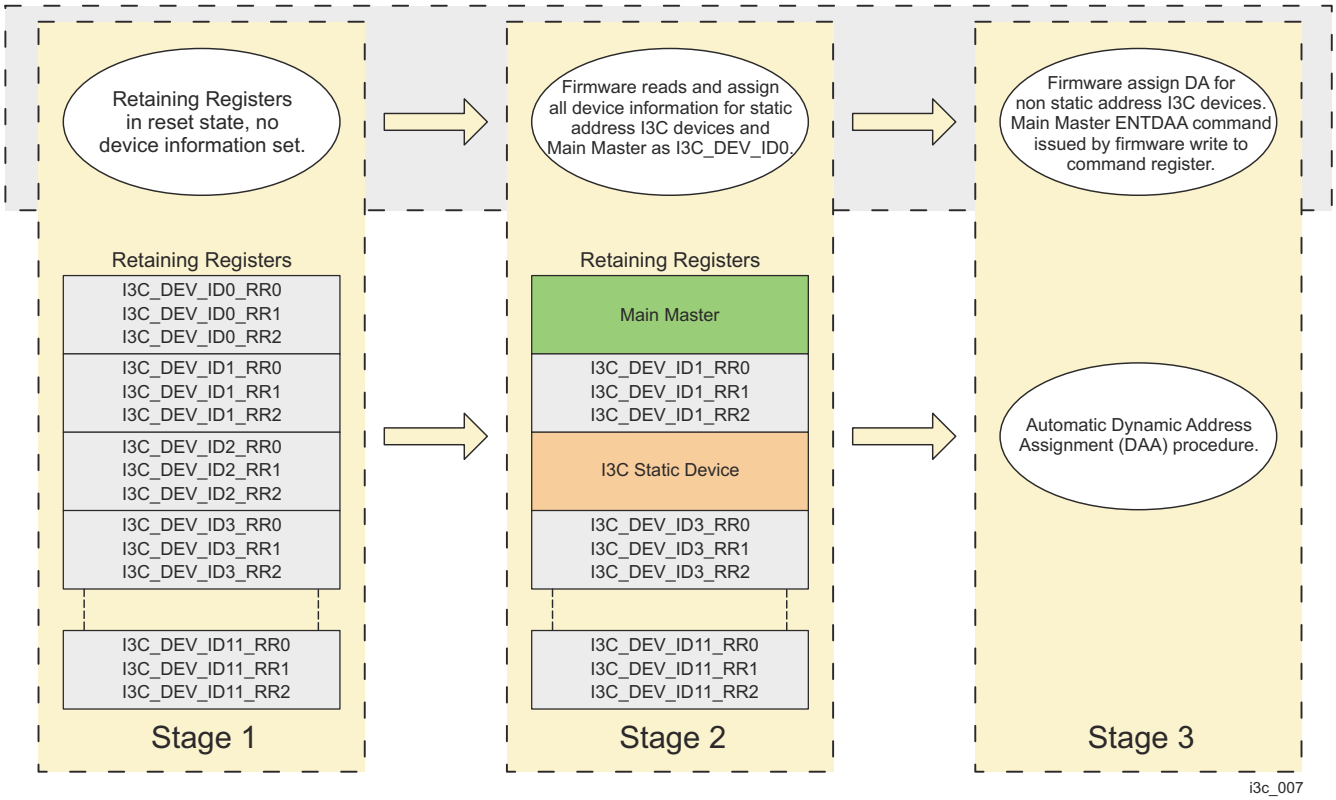


Figure 12-23. Power-up I3C Bus Initialization

During automatic Dynamic Address Assignment procedure, the I^{3}C devices send their Provisional ID, BCR and DCR using open-drain class driver. The Device ID sections are assigned to the I3C Devices based on the Provisional ID arbitration. As a result, the lower value of the Provisional ID the device has, the lower position in retaining registers map it gets. Once device gets Dynamic Address and its Retaining Registers are filled, corresponding I3C_DEVS_CTRL bit for active is set automatically.

Figure 12-24 presents the I^{3}C flow diagram for DAA procedure.

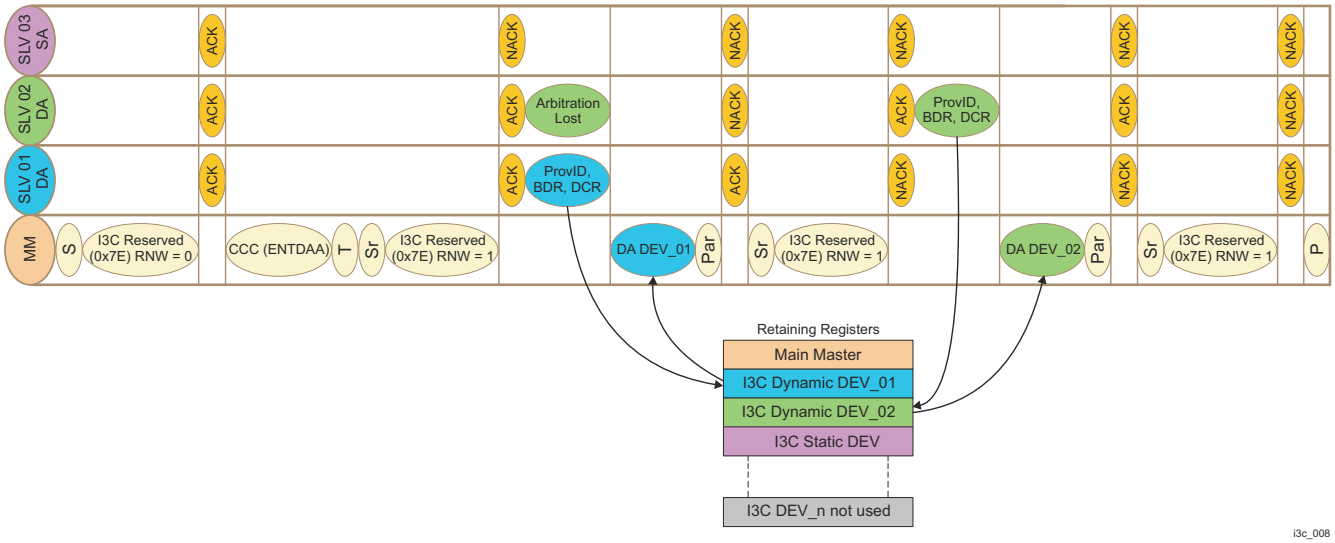


Figure 12-24. Flow Diagram for DAA Procedure

Copyright © 2024 Texas Instruments Incorporated

Note

Static I^{3}C devices have already assigned Dynamic Address with SETDASA CCC command and stay quiet during DAA procedure.

Note

Although arbitration continues over BCR and DCR values, it is not guaranteed to be resolved in this phase, if not resolved during Provisional ID period.

12.1.4.3.8 I3C Sending CCC Messages

To send a CCC message, I3C controller uses Host Command Format.

Following are the only relevant Command Word fields for CCC message:

- I3C_CMD0_FIFO[30] IS_CCC – should be set to 1;
- I3C_IMD_CMD1[7-0] CCC – specifies the Common Command Code to be sent;
- I3C_CMD0_FIFO[7-1] DEV_ADDR – should address slave device in case of Direct CCC command;
- I3C_CMD0_FIFO[23-12] PL_LEN – specifies the length of the Data field for commands (the number of bytes to be sent for particular CCC);
- I3C_CMD0_FIFO[0] RNW – defines the direction of the Data field: 0 for writing data to slave(s) and 1 for reading data from slaves;
- I3C_CMD0_FIFO[29] BCH – when two sequential directed CCC messages of the same type are sent; this bit determines whether the broadcast address and message code should be issued before the second address and command code;
- I3C_CMD0_FIFO[25] RSBC – use in conjunction with BCH when issuing short CCC messages;

The payload for CCC commands writing data to the slaves needs to be written to the TX FIFO prior to sending the two Command Words. The payload received by reading CCC commands can be read after receiving the COMP interrupt corresponding to the command. Note that ENTDAA command uses payload of 0, since the data it receives is not stored in RX FIFO; the amount of data depends on the number of slave responding to the command.

When sending a sequence of two identical directed CCC messages to different slaves, the I3C_CMD0_FIFO[29] BCH and I3C_CMD0_FIFO[25] RSBC Command Word fields can be used to send the second message as short CCC. I3C_CMD0_FIFO[29] BCH should be set to 0 for the second CCC message. Instead of sending the broadcast address, waiting for it to be acknowledged and then sending the message code, this will issue restart pattern and immediately transfer the address and payload afterwards. If I3C_CMD0_FIFO[29] BCH is set to 1 the second CCC message is sent with broadcast address and command code in the beginning. In all other CCC message cases the I3C_CMD0_FIFO[29] BCH field is disregarded. Because the short CCC is preceded by restart pattern in order for this to work the I3C_CMD0_FIFO[25] RSBC field for the first command should also be set to 1.

Table 12-29 below summarizes available CCC set along with their payload layout in data FIFOs and Command Word settings.

Table 12-29. CCC Commands

CCC [(1)]	Description	Payload Layout [(2)]	PL	RNW
ENEC_BC (0x00) ENEC_DC (0x80)	Enables slave event driven interrupts	TX FIFO: {28'b0, 1'b**HJ**, 1'b0, 1'b**MR**, 1'b**INT**} (**HJ** – Hot-Join, **MR** – Mastership Request IBI, **INT** – regular IBI)	1	0
DISEC_BC (0x01) DISEC_DC (0x81)	Disables slave event driven interrupts	TX FIFO: {28'b0, 1'b**HJ**, 1'b0, 1'b**MR**, 1'b**INT**} (**HJ** – Hot-Join, **MR** – Mastership Request IBI, **INT** – regular IBI)	1	0
ENTAS0_BC (0x02) ENTAS0_DC (0x82)	Set activity to State 0 (normal operation)	No payload	0	0
RSTDAA_BC (0x06) RSTDAA_DC (0x86)	Forget current Dynamic Address and wait for new one	No payload	0	0

Copyright © 2024 Texas Instruments Incorporated

Table 12-29. CCC Commands (continued)

CCC [1]	Description	Payload Layout [2]	PL	RNW
ENTDAA_BC (0x07)	Enter DAA procedure to assign Dynamic Addresses to unassigned devices	See Section 12.1.4.4.3, *Initiate DAA Procedure*	0	0
SETMWL_BC (0x09) SETMWL_DC (0x89)	Set maximum write length of a single private transfer	TX FIFO: {16'b0, **LSB**, **MSB**} (**LSB** and **MSB** of 16-bit Max Write Length value)	2	0
SETMRL_BC (0x0A) SETMRL_DC (0x8A)	Set maximum read length of a single private transfer	TX FIFO: {16'b0, **LSB**, **MSB**} (**LSB** and **MSB** of 16-bit Max Write Length value)	2	0
DEFSLVS_BC (0x08)	Broadcast triples of DA, DCR and SA (or 0) for each slave	No payload	0	0
ENTHDR_BC (0x20)	Switch to HD-DDR mode	No payload	0	0
SETDASA_DC (0x87)	Set Dynamic Address to slave with Static Address	TX FIFO: {24'b0, 7'b**DA**, 1'b**P**} (**DA** – new Dynamic Address, **P** – XNOR of DA bits)	1	0
SETNEWDA_DC (0x88)	Assign new Dynamic Address	TX FIFO: {24'b0, 7'b**DA**, 1'b**P**} (**DA** – new Dynamic Address, **P** – XNOR of DA bits)	1	0
GETMWL_DC (0x8B)	Get maximum write length of a single private transfer	RX FIFO: {16'b0, **LSB**, **MSB**} (**LSB** and **MSB** of 16-bit Max Write Length value)	2	1
GETMRL_DC (0x8C)	Get maximum read length of a single private transfer	RX FIFO: {16'b0, **LSB**, **MSB**} (**LSB** and **MSB** of 16-bit Max Read Length value)	2	1
GETPID_DC (0x8D)	Get Slave's Provisional ID value	RX FIFO[n+1]: {16'b0, **PID0**, **PID1**} RX FIFO[n]: {**PID2**, **PID3**, **PID4**, **PID5**}	6	1
GETBCR_DC (0x8E)	Get Device's Bus Characteristic Register value	RX FIFO: {24'b0, **BCR**}	1	1
GETDCR_DC (0x8F)	Get Device's Characteristic Register value	RX FIFO: {24'b0, **DCR**}	1	1
GETSTATUS_DC (0x90)	Read Device's operating status	RX FIFO: {16'b0, 2'b**AM**, 1'b**PE**, 1'b0, 4'b**INT**, 8'b0} (**AM** – Activity Mode, **PE** – Protocol Error, **INT** – Pending Interrupt number)		
GETACCMST_DC (0x91)	Get accept mastership	RX FIFO: {24'b0, **SlaveAddr**}	1	1
GETMXDS_DC (0x94)	Read Maximum SCL Frequency of the slave	RX FIFO: {16'b0, **MaxRd**, **MaxWr**} (**MaxRd** – Clock to Data Turnaround Time and Maximum Sustained Read Data Rate, **MaxWr** – Maximum Sustained Write Data Rate)	1	1
GETHDRCAP (0x96)	Ask slave for HDR modes it supports (can be sent only if BCR identifies HDR support)	RX FIFO: {24'b0, **HDRCAP**} (**HDRCAP** – HDR capability modes)	1	1

(1) CCC column displays:
- The name of the Common Command Code, suffixed with _BC for Broadcast CCC and _DC for Direct CCC;
- In parentheses is value of CCC field to be used in Command Word 1;

(2) Payload Layout column displays:
- {<...>,<field1>,<field0>} - concatenation of 8-bit CCC data fields within 32-bit FIFO cell;
- <width>'b<field_name> - field <field_name> takes <width> bits (8 bits if not specified);
- The payload fields are denoted in bold;

12.1.4.3.9 I3C In-Band Interrupt

In order to handle an incoming In-Band Interrupt from particular slave device, the address and configuration of IBI handling for this device needs to be stored in the I3C_SIR_MAP0 through I3C_SIR_MAP5 registers.

Copyright © 2024 Texas Instruments Incorporated

This register map holds information on device address, length of payload, device role and response. The map organization is presented in the Table 12-30 below.

Table 12-30. SIR Map Organization

Register	[31-16] bit fields	[15-0] bit fields
I3C_SIR_MAP0	Device 1	Device 0
I3C_SIR_MAP1	Device 3	Device 2
I3C_SIR_MAP2	Device 5	Device 4
I3C_SIR_MAP3	Device 7	Device 6
I3C_SIR_MAP4	Device 9	Device 8
I3C_SIR_MAP5	Reserved	Device 10

All devices requesting IBI, but not added to the SIR Map, will receive the NACK response from I3C master device.

Example for IBI Map configuration looks as follows:

1. Write 0x80470445 which is translated to the following configurations for first two devices:
   a. Regular slave with DA = 0x22 to be ACKed and receive 4 bytes of payload (Device 0)
   b. Secondary master with DA = 0x23 to be ACKed and receive no payload (Device 1)
2. Write 0x004A0749 which is translated to the following configurations for the third and fourth device:
   a. Regular slave with DA = 0x24 to be ACKed and receive 7 bytes of payload (Device 2)
   b. Regular slave with DA = 0x25 to be NACKed (Device 3)
3. Write 0x0000034D which configures the last fifth device as follows:
   a. Regular slave with DA = 0x26 to be ACKed and receive 3 bytes of payload (Device 4)

After receiving regular or mastership request IBI interrupt, firmware needs to read the I3C_IBIR register to determine which device requested IBI. The ordering of bits in the I3C_IBIR register corresponds to ordering of devices stored in I3C_SIR_MAP0 through I3C_SIR_MAP5 registers. Note that the IBI capable devices are enumerated and ordered independently from device ordering in the retaining registers used during DAA procedure.

12.1.4.3.9.1 Regular I3C Slave In-Band Interrupt

When regular IBI request appears on the I^3C bus, I^3C master hardware automatically handles whole IBI procedure based on settings in I3C_SIR_MAP0 through I3C_SIR_MAP5 registers. As the IBI procedure acknowledged by I3C master finishes (including payload reception), host is informed by the IBI interrupt that such event occurred and was successfully processed. At this point, firmware should:

1. Read I3C_IBIR register to determine requesting device
2. Read related payload from IBI RX FIFO
3. Execute related action (application/system dependent), if any designed for particular IBI/device.

12.1.4.3.9.2 Current Master Takeover In-Band Interrupt

When mastership takeover IBI request appears on the I^3C bus, I^3C master hardware automatically sends the response based on settings in I3C_SIR_MAP0 through I3C_SIR_MAP5 registers. After this host is informed by the I3C_MST_ISR interrupt that handoff procedure should be executed. At this point, firmware should:

1. Read I3C_IBIR register to determine requesting device (optionally, if such information is useful for application host)
2. Send the GETACCMST CCC to confirm mastership handover
3. Wait for the completion of pending I3C master command signaled by I3C_MST_ISR interrupt – at this moment I3C master switches to the slave mode
4. Write 0x001FXXXX to FLUSH_CTRL register to flush the FIFOs as the slave mode reuses TX and RX FIFOs (bits marked X hold RX_FIFO threshold so should be set/maintained as needed)
5. Depending on application/system, prepare for slave operation

Note that it is firmware and/or system responsibility to provide bus information to Secondary Masters before mastership handover is granted. It can be achieved in several ways:

Copyright © 2024 Texas Instruments Incorporated

- Secondary Masters listen to the bus during DAA and store necessary data
- Main Master sends DEFSLVS CCC after each ENTDAA / RSTDAA / SETNEWDA / SETDASA sequence that modifies bus

12.1.4.3.10 I3C Hot-Join Request

In order to accept any Hot-Join (HJ) request from slave device, application needs to set the I3C_CTRL[6] HJ_ACK and I3C_CTRL[8] HJ_DISEC control bits that define its response for Hot-Join request. Based on I3C_CTRL[6] HJ_ACK value the I3C master accepts or refuses the Hot-Join request by providing respectively ACK or NACK bit right after I3C_CMD0_FIFO[0] RNW bit. I3C_CTRL[8] HJ_DISEC bit controls whether master starts DAA procedure or sends broadcast DISEC command to disable HJ requests.

If Hot-Join request occurs, the application processor is notified by the I3C_MST_ISR interrupt. The application processor is aware how this particular Hot-Join request should be handled by checking current setting (can read the control bits or the stored previously used setting). In case of acknowledged request with subsequent DAA, the application processor is notified of automatically completed enumeration of hot-joined device by subsequent I3C_MST_ISR interrupt and simultaneously set I3C_MST_STATUS0 flag. In case of DISEC command, only I3C_MST_ISR interrupt will be set.

It is recommended to keep the I3C_CTRL[6] HJ_ACK bit disabled until the bus initialization procedure is completed in order to prevent slaves from indirect DAA enforcement.

12.1.4.3.11 I3C Immediate Commands

Immediate Command Register is implemented in order to allow the application processor to issue immediate command which has a higher priority over the ordinary command. The immediate commands are supposed to be used in exceptional cases and only CCC commands are applicable. The I3C supports only one immediate command at a time and it is accessed by the host through two APB registers: I3C_IMD_CMD0 and I3C_IMD_CMD1. If a normal command is currently being executed and new immediate command is loaded, then controller waits to finish the current command and after that starts executing the higher priority command regardless of the content of the ordinary command FIFO. Immediate command can be used only for CCC commands and the corresponding payload must not exceed 4 bytes. As a result the GETPID and DEVSLVS CCC commands cannot be executed using immediate commands. The format for the immediate command words is the same as for the ordinary commands, although some fields are reserved since not used or fixed for CCC commands (writing to these fields has no effect).

12.1.4.3.12 I3C Host Commands

The normal operation of the controller allows the firmware to build a message and then issue the transaction to the I^3C controller. The command queue is provided in order to allow the host to load a request for multiple messages. Single message consists of two 32-bit words (Command Word0 and Command Word1). In order to issue the transaction the following procedure should be followed:

- Write the payload to the TX FIFO (applicable for write transfers only);
- Write Command Word1;
- Write Command Word0;

If controller is enabled (I3C_CTRL[31] DEV_EN bit is set to 0x01), writing Command Word0 triggers the execution of the command, resulting in start condition on the I3C bus.

12.1.4.3.13 I3C Sending Private Data in SDR Messages

The I3C controller supports private data transfers in Single Data Rate with SCL frequencies up to 10.375 MHz. It supports all types of SDR transfers and provides hardware built-in features for transfers to I3C slave devices with limited maximum SCL frequency. Before any private transfer to the slave devices occurs, bus mode and SCL prescalers shall be programmed properly. Controller also provides vendor extension to the basic transfers which simplifies addressing internal registers of slave devices. To take advantage of this extension, the I3C controller must communicate with the slave devices supporting such transmission scheme.

The extension, called XMIT modes, combines standard private write and read frames to pass the slave register sub-address in addition to the actual payload. Four transmit modes are designed for specific usage:

Copyright © 2024 Texas Instruments Incorporated

- XMIT00 – burst (byte-by-byte) transfer with static register sub-address. Sends static sub-address followed by data bytes only. Expecting slave will make self-increment of the address for each data byte or will have payload buffer implemented.
- XMIT01 – single transfer with incremented register sub-address. Each data byte followed by repeated start condition and the slave's internal register address are incremented explicitly.
- XMIT10 – single transfer with static register sub-address. Each data byte followed by repeated start condition, register sub-address remains intact.
- XMIT11 – burst (byte-by-byte) transfer without register sub-address (first data byte after the slave address is a payload byte).

In addition to the multiple transmit modes, I3C controller has an option to provide 16-bit address of the internal register in slave device to support slaves with more than 256 internal addresses.

For arbitration priorities management, private SDR commands may use Broadcast Command Header or not. This is controlled by BCH field of command descriptor.

12.1.4.3.13.1 SDR Private Write Message

The Private Write message transmitted over I3C bus in all supported transmit modes is presented in Figure 12-25, *Master Write Transaction for Single and Multi Address Sequences*. The provided transmit modes use additional I3C write frame that utilizes the data field to pass slave register address ahead of payload byte(s). As writing command word triggers start condition immediately, it shall be guaranteed by firmware that the data payload is written to the Tx FIFO in advance before it is going to be transmitted.

The completion of the message frame on the I3C bus is signaled by the I3C_MST_ISR[16] IMM_COMP interrupt.

In case of transfers with data payload greater than the TX FIFO size, firmware can use I3C_MST_ISR[15] TX_THR interrupt to fill the FIFO on time. The system design should prevent the TX FIFO from underflow. During the time elapsed between reading the last four bytes from almost empty TX FIFO and reading the subsequent one, firmware is required to refill the TX FIFO.

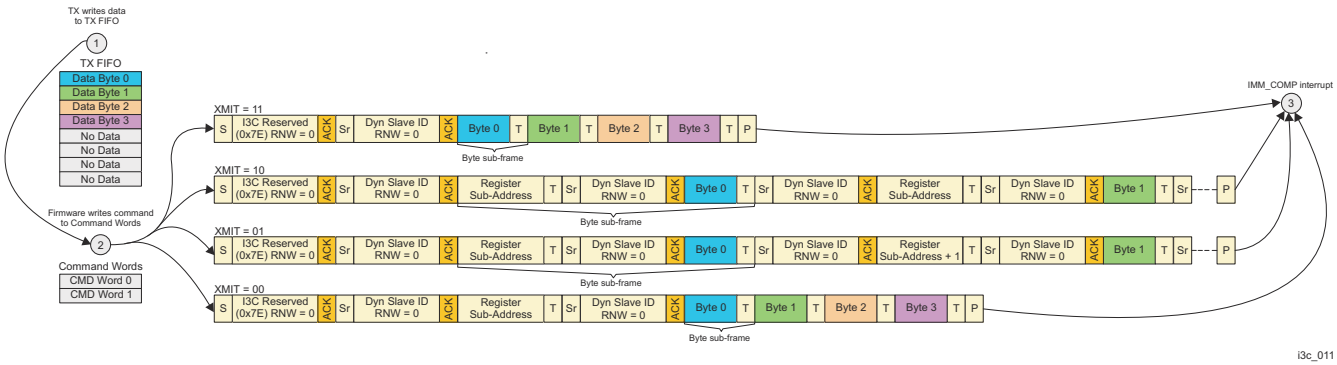


Figure 12-25. Master Write Transaction for Single and Multi Address Sequences

12.1.4.3.13.2 SDR Private Read Message

The Private Read message transmitted over I3C bus in all supported transmit modes is presented in Figure 12-26, *Master Read Transaction for Single and Multi Address Sequences*

Similarly to Private Write message, each I3C read frame is preceded by I3C write frame which sends slave register address to be read. After sending the command, firmware should wait for I3C_MST_ISR[16] IMM_COMP interrupt which signals the end of message and RX FIFO content ready to read.

In case of payload lengths exceeding RX FIFO size, firmware should utilize I3C_MST_ISR[7] TX_THR interrupt to read RX FIFO before it gets filled and prevent from data corruption. For easier management of long FIFO and/or long host read intervals, the amount of data available in FIFO that triggers the interrupt can be programmed according to actual conditions in given system.

Copyright © 2024 Texas Instruments Incorporated

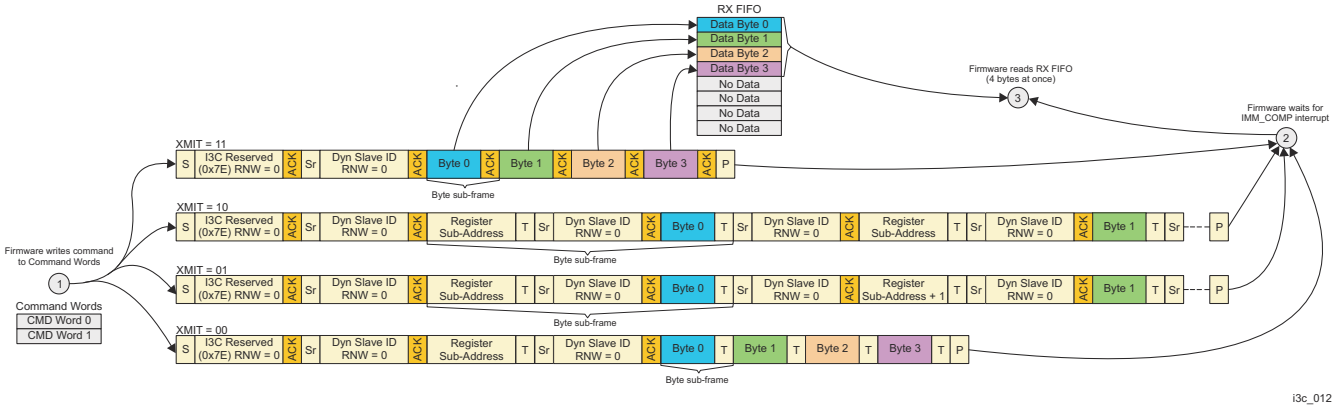


Figure 12-26. Master Read Transaction for Single and Multi Address Sequences

12.1.4.3.13.3 SDR Payload Length Adjustment

In case of Private Write SDR commands, the payload length that needs to be specified is always equal to the number of data bytes sent to slave in Private Write transmission. If the payload specified is greater than slave's capability, the data are simply ignored by slave.

In case of Private Read SDR commands slave is the device which normally decides how many bytes it is able to send and when the read transfer is terminated. This can also result from hardware architecture or from SETMRL CCC command sent earlier to the slave.

For simplified handling I3C design triggers the read abort procedure after receiving specified number of payload bytes. This matters in these transfer modes which send data bytes after one another. If one of these modes is used, the I3C_CMD0_FIFO[23-12] PL_LEN bit-field is interpreted as payload size that should not be reached during transfer rather than the actual number of bytes to be received. This applies only to read transfer from I3C devices.

Table 12-31. Payload Setting in SDR Mode

Transmission	I3C_CMD0_FIFO[28-27] XMIT_MODE	I3C_DEV_IDn_RR0[9] IS_I3C (n = 0 to 11)	I3C_CMD0_FIFO[23-12] PL_LEN
I3C, NCA	11	1	Abort limit(1)
I3C, Single	00	1	Abort limit(1)
I3C, Multi-Static	10	1	Number of bytes to receive
I3C, Multi-Incremental	01	1	Number of bytes to receive

(1) If payload size is known, it can be set to one more than the known size; otherwise it should be set to MRL+1 where MRL is value programmed with SETMRL CCC in particular slave or its hardware payload size limit, if known.

Copyright © 2024 Texas Instruments Incorporated

12.1.4.4 I3C Programming Guide

12.1.4.4.1 I3C Power-On Programming Model

Typical power-on programming sequence is as follows:

1. Program bus description into Retaining Registers according Section 12.1.4.3.6 and I3C_DEVS_CTRL.
2. If default values of SCL prescalers do not fit applied clock frequencies, program PRESCL_CTRL0 and PRESCL_CTRL1 according Section 12.1.4.3.2.
3. If any of following applies:
   a. Address Header Optimization is to be enabled.
   b. Halt mechanism should be enabled.

   then alter values of I3C_CTRL[1-0] BUS_MODE, I3C_CTRL[3] AHDR_OPT and I3C_CTRL[30] HALT_EN fields, respectively:
   a. Program I3C_CTRL register with value 0x80000000 to enable the controller.

12.1.4.4.2 I3C Static Devices Programming

I3C devices shipped with Static Address can be enumerated prior to DAA procedure using SETDASA CCC command. In order to enumerate I3C device firmware needs to follow procedure:

1. Prepare device's retaining register:
   a. Write corresponding I3C_DEV_IDn_RR0 (offset 0x080 + (n × 0x010)) for n = 0 to 11:
      i. I3C_DEV_IDn_RR0[9] IS_I3C (n = 0 to 11): 0x01
      ii. I3C_DEV_IDn_RR0[7-0] DEV_ADDR bit-field (n = 0 to 11): set to device's Static Address for [7-1] bits and set to Static Address parity for bit 0
2. Issue SETDASA CCC:
   a. Make sure that controller is enabled.
   b. Write new Dynamic Address to TX FIFO.
   c. Write Command Word1 with SETDASA CCC value:
      i. I3C_IMD_CMD1[7-0] CCC bit-field: 0x87
   d. Write Command Word0 with following field values:
      i. I3C_CMD0_FIFO[30] IS_CCC bit: 0x01
      ii. I3C_CMD0_FIFO[23-12] PL_LEN bit-field: 0x01
      iii. I3C_CMD0_FIFO[7-1] DEV_ADDR bit-field: Static Address
      iv. I3C_CMD0_FIFO[0] RNW bit: 0x00
3. Enable at least COMP, NACK and INVALID_DA interrupts by writing 0h to IER register.
4. Wait for COMP interrupt (read I3C_MST_ISR).
5. If COMP and no other interrupt occurs, continue. Otherwise, handle the error situation.
6. Fill Dynamic Address into device configuration register:
   a. Modify I3C_DEV_IDn_RR0 (offset 0x080 + (n × 0x010)) for n = 0 to 11 register by writing Dynamic Address to DEV_ADDR field and its parity to ADDR_PAR field (keep values of other fields)
7. If not already known to application host, retrieve BCR with GETBCR CCC:
   a. Write Command Word1 with GETBCR CCC value:
      i. I3C_IMD_CMD1[7-0] CCC bit-field: 0x8E
   b. Write Command Word0 with following field values:
      i. I3C_CMD0_FIFO[30] IS_CCC bit: 0x01
      ii. I3C_CMD0_FIFO[23-12] PL_LEN bit-field: 0x01
      iii. I3C_CMD0_FIFO[7-1] DEV_ADDR bit-field: Dynamic Address
      iv. I3C_CMD0_FIFO[0] RNW bit: 0x01
   c. Wait for COMP interrupt
   d. Read RX FIFO register - BCR is stored in bits [7:0]
8. Optionally retrieve Provisional ID and DCR with corresponding CCCs:
   a. Write Command Word1 with GETDCR CCC value:
      i. I3C_IMD_CMD1[7-0] CCC bit-field: 0x8F
   b. Write Command Word0 with following field values:

Copyright © 2024 Texas Instruments Incorporated

   i. I3C_CMD0_FIFO[30] IS_CCC bit: 0x01
   ii. I3C_CMD0_FIFO[23-12] PL_LEN bit-field: 0x01
   iii. I3C_CMD0_FIFO[7-1] DEV_ADDR bit-field: Dynamic Address
   iv. I3C_CMD0_FIFO[0] RNW bit: 0x01
c. Wait for COMP interrupt
d. Read RX FIFO register - DCR is stored in bits [7:0]
e. Write Command Word1 with GETPID CCC value:
   i. I3C_IMD_CMD1[7-0] CCC bit-field: 0x8D
f. Write Command Word0 with following field values:
   i. I3C_CMD0_FIFO[30] IS_CCC bit: 0x01
   ii. I3C_CMD0_FIFO[23-12] PL_LEN bit-field: 0x06
   iii. I3C_CMD0_FIFO[7-1] DEV_ADDR bit-field: Dynamic Address
   iv. I3C_CMD0_FIFO[0] RNW bit: 0x01
g. Wait for COMP interrupt
h. Read RX FIFO register twice – PID bytes are stored in two FIFO locations. See Table 12-29

9. Write retrieved device configuration data to retaining registers:
   a. Write four MSB of obtained PID value into I3C_DEV_IDn_RR1 (offset 0x084 + (n × 0x010)) for n = 0 to 11:
   b. Write two LSB of obtained PID value as well as BCR and DCR values into I3C_DEV_IDn_RR2 (offset 0x088 + (n × 0x010)) for n = 0 to 11:

Note

SETDASA CCC command has advantage over DAA in shorter time spent on device configuration as retrieving PID and DCR (6 + 1 bytes transmitted over I3C bus) is optional and can be skipped if not used.

12.1.4.4.3 I3C DAA Procedure Initiation

To initiate the DAA procedure I3C needs to issue ENTDAA CCC command following the below procedure:

1. Enable at least COMP and NACK interrupts.
2. Issue ENTDAA CCC command:
   a. Make sure controller is enabled.
   b. Write Command Word1 with ENTDAA CCC command value:
      i. I3C_IMD_CMD1[7-0] CCC bit-field: 0x07
   c. Write Command Word0 with following field values:
      i. I3C_CMD0_FIFO[30] IS_CCC bit: 0x01
      ii. I3C_CMD0_FIFO[23-12] PL_LEN bit-field: 0x00
      iii. I3C_CMD0_FIFO[7-1] DEV_ADDR bit-field: 0x00
      iv. I3C_CMD0_FIFO[0] RNW bit: 0x00
3. Wait for COMP interrupt (read I3C_MST_ISR)
4. If COMP and no other interrupt occurs, continue. Otherwise, handle the error situation. NACK interrupt indicates no slave devices available on bus.
5. Read I3C_MST_STATUS0[25] DAA_COMP bit. It is asserted simultaneously with I3C_MST_ISR interrupt flag.

Note

DAA procedure takes significant amount of time since there are always 9 bytes of data transmitted for each device (PID, BCR, DCR from device and DA to device), so for maximum bus size it will take 99 bytes.

12.1.4.4.4 I3C SDR Write Message Programming Model

To accomplish the SDR write transfer, the host must perform the following steps:

1. Write all payload data bytes into TX FIFO buffer:

Copyright © 2024 Texas Instruments Incorporated

   a. Write first four or less bytes to be sent, with first byte stored at LSB position.
   b. Repeat writes until whole payload is written.
2. Write Private Write command to command queue:
   a. Write the Command Word 1 (I3C_CMD1_FIFO) with slave CSR address:
      i. CSR (bits [15:0] or [7:0]): Slave register address
   b. Write the Command Word0 (I3C_CMD0_FIFO) with following data fields:
      i. I3C_CMD0_FIFO[31] IS_DDR: 0x0
      ii. I3C_CMD0_FIFO[30] IS_CCC: 0x0
      iii. I3C_CMD0_FIFO[28-27] XMIT_MODE: Transmit Mode
      iv. I3C_CMD0_FIFO[26] SBCA: 0x1 for 16-bit CSR address, 0x0 otherwise
      v. I3C_CMD0_FIFO[23-12] PL_LEN: Number of bytes to send
      vi. I3C_CMD0_FIFO[7-1] DEV_ADDR: Slave Address
      vii. I3C_CMD0_FIFO[0] RNW: 0x0
3. Wait for transfer completion notified by I3C_MST_ISR[16] IMM_COMP bit, which indicates that all data of particular command are successfully transmitted to the slave.

12.1.4.4.5 I3C SDR Read Message Programming Model

To accomplish the SDR read transfer, the host must perform the following steps:

1. Write Private Write command to command queue:
   a. Write the Command Word1 (I3C_CMD1_FIFO) with slave CSR address:
      i. CSR (bits [15:0] or [7:0]): Slave register address
   b. Write the Command Word0 (I3C_CMD0_FIFO) with following data fields:
      i. I3C_CMD0_FIFO[31] IS_DDR: 0x0
      ii. I3C_CMD0_FIFO[30] IS_CCC: 0x0
      iii. I3C_CMD0_FIFO[28-27] XMIT_MODE: Transmit Mode
      iv. I3C_CMD0_FIFO[23-12] PL_LEN: See Section 12.1.4.3.13.3
      v. I3C_CMD0_FIFO[7-1] DEV_ADDR: Slave Address
      vi. I3C_CMD0_FIFO[0] RNW: 0x1
2. Wait for transfer completion notified by I3C_MST_ISR[16] IMM_COMP bit, which indicates that all data of particular command are successfully sent by the slave device and received data are ready to be read by the host.
3. Read all payload data bytes from RX FIFO buffer:
   a. Read first four or less received bytes, with first received byte stored at LSB position.
   b. Repeat reads until whole payload is read.

12.1.4.4.6 I3C DDR Write Message Programming Model

To accomplish the DDR write transfer, the host must perform the following steps:

1. Prepare HDR-DDR message words (each 20-bit word is LSB aligned in TX FIFO cell; set bits unspecified below to 0):
   a. Write command word for write HDR-DDR command to TX FIFO using following field values:
      i. Preamble (bits [19:18]): 0x01
      ii. CMD (bits[17:10]): Write Command Code (from 0x00 to 0x7F)
      iii. DA (bits[9:3]): Slave Dynamic Address
      iv. Parity (bits[1:0]): Parity (XOR of odd and even bits of 16-bit payload)
   b. Write first data word to TX FIFO using following field values:
      i. Preamble (bits [19:18]): 0x10
      ii. Data (bits[17:2]): 16-bit data
      iii. Parity (bits[1:0]): Parity
   c. Write zero or more subsequent data words to TX FIFO using following field values:
      i. Preamble (bits [19:18]): 0x11
      ii. Data (bits[17:2]): 16-bit data
      iii. Parity (bits[1:0]): Parity
   d. Write CRC word to TX FIFO:

Copyright © 2024 Texas Instruments Incorporated

   i. Preamble (bits [19:18]): 0x01
   ii. Token (bits[17:14]): Ch
   iii. CRC (bits[13:9]): CRC5
   e. Repeat steps 1(a) to 1(d) as many write commands are to be sent.
2. Write ENTHDR CCC command to command queue:
   a. Write the Command Word1 (I3C_CMD1_FIFO) of ENTHDR CCC.
   b. Write the Command Word0 (I3C_CMD0_FIFO) of ENTHDR CCC with following data fields:
      i. I3C_CMD0_FIFO[31] IS_DDR: 0x0
      ii. I3C_CMD0_FIFO[30] IS_CCC: 0x1
      iii. I3C_CMD0_FIFO[0] RNW: 0x0
3. Write HDR-DDR command(s) to command queue:
   a. Write the Command Word1 (I3C_CMD1_FIFO).
   b. Write the Command Word0 (I3C_CMD0_FIFO) with following data fields:
      i. I3C_CMD0_FIFO[31] IS_DDR: 0x1
      ii. I3C_CMD0_FIFO[23-12] PL_LEN: Number of TX FIFO words
   c. Repeat steps 2(a) and 2(b) as many write commands are prepared in TX FIFO in step 1.
4. Wait for transfer completion notified by I3C_MST_ISR[16] IMM_COMP bit, which indicates that all data of particular command are successfully transmitted to the slave.

If I3C_MST_ISR[16] IMM_COMP interrupt is utilized, the controller allows the application processor to perform other tasks or sleep while message is being sent.

In case of multiple command issued to command queue, I3C_MST_ISR[5] CMDD_EMP interrupt may be utilized instead. It indicates completion of all commands in command queue and allows yet more efficient application processor resources management. It is issued simultaneously with the I3C_MST_ISR[5] CMDD_EMP interrupt corresponding to the last command in the queue.

12.1.4.4.7 I3C DDR Read Message Programming Model

To accomplish the DDR read transfer, the host must perform the following steps:
1. Prepare HDR-DDR message words (each 20-bit word is LSB-aligned in TX FIFO cell, set bits unspecified below to 0):
   a. Write Command Word for Read HDR-DDR command to TX FIFO using following field values:
      i. Preamble (bits [19:18]): 2’b01
      ii. CMD (bits[17:10]): Read Command Code (0x80..0xBF)
      iii. DA (bits[9:3]): Slave Dynamic Address
      iv. Parity (bits[1:0]): Parity (XOR of odd and even bits of 16-bit payload)
   b. Repeat step 1(a) as many read commands are to be sent.
2. Write ENTHDR CCC command to command queue:
   a. Write the Command Word1 (I3C_CMD1_FIFO) of ENTHDR CCC.
   b. Write the Command Word0 (I3C_CMD0_FIFO) of ENTHDR CCC with following data fields:
      i. I3C_CMD0_FIFO[31] IS_DDR: 0x0
      ii. I3C_CMD0_FIFO[30] IS_CCC: 0x1
      iii. I3C_CMD0_FIFO[0] RNW: 0x0
3. Write HDR-DDR command(s) to command queue:
   a. Write the Command Word1 (I3C_CMD1_FIFO)
   b. Write the Command Word0 (I3C_CMD0_FIFO) with following data fields:
      i. I3C_CMD0_FIFO[31] IS_DDR: 0x1
      ii. I3C_CMD0_FIFO[23-12] PL_LEN: 0x1
   c. Repeat steps 2(a) and 2(b) as many read commands are prepared in TX FIFO in step 1.
4. Wait for transfer completion notified by I3C_MST_ISR[16] IMM_COMP bit in interrupt status register, which indicates that all data of particular command are successfully sent by the slave device and received data are ready to be read by the host.
5. Read received payload:
   a. Read first data word from RX FIFO extracting field values:

Copyright © 2024 Texas Instruments Incorporated

   i. Preamble (bits [19:18]): 0x10
   ii. Data (bits[17:2]): 16-bit data
   iii. Parity (bits[1:0]): Parity
b. Read zero or more subsequent data words from RX FIFO extracting field values as long as the read preamble matches following value:
   i. Preamble (bits [19:18]): 0x11
   ii. Data (bits[17:2]): 16-bit data
   iii. Parity (bits[1:0]): Parity
c. As read preamble changes, read CRC Word:
   i. Preamble (bits [19:18]): 0x01
   ii. Token (bits[17:14]): Ch
   iii. CRC (bits[13:9]): CRC5
d. Confirm (by firmware) that CRC5 value matches data payload received.

If I3C_MST_ISR[16] IMM_COMP interrupt is utilized, the controller allows the application processor to perform other tasks or sleep while message is being sent.

In case of multiple command issued to command queue, I3C_MST_ISR[5] CMDD_EMP interrupt may be utilized instead of I3C_MST_ISR[16] IMM_COMP. It indicates completion of all commands in command queue and allows yet more efficient application processor resources management. It is issued simultaneously with the I3C_MST_ISR[16] IMM_COMP interrupt corresponding to the last command in the queue.

Copyright © 2024 Texas Instruments Incorporated

12.1.5 Multichannel Serial Peripheral Interface (MCSPI)

This section describes the Multichannel Serial Peripheral Interface (MCSPI) modules for the device.

12.1.5.1 MCSPI Overview

The MCSPI module is a multichannel transmit/receive, master/slave synchronous serial bus.

There are MCSPI modules in the device.

12.1.5.1.1 SPI Features

The SPI module includes the following main features:

- Serial clock with programmable frequency, polarity, and phase for each channel
- Wide selection of SPI word lengths, ranging from 4 to 32 bits
- Up to channels in controller mode, or single channel in receive mode
- Controller multichannel mode:
  - Full duplex/half duplex
  - Transmit-only/receive-only/transmit-and-receive modes
  - Flexible input/output (I/O) port controls per channel
  - Programmable clock granularity
  - Per channel configuration for clock definition, polarity enabling, and word width
- Single interrupt line for multiple interrupt source events
- Enable the addition of a programmable start-bit for MCSPI transfer per channel (start-bit mode)
- Supports start-bit write command
- Supports start-bit pause and break sequence
- Programmable shift operations (1-32 bits)
- Programmable timing control between chip select and external clock generation
- Built-in FIFO available for a single channel

Unsupported Features:

- See the Module Integration section of this document for a list of module features not supported by the integration on this Device.

Copyright © 2024 Texas Instruments Incorporated

12.1.5.1.2 MCSPI Ports

This section describes module ports related to clocks, resets, and hardware requests.

Table 12-32. MCSPI Clocks and Resets

Clocks	
Module Clock Input	**Description**
MCSPI_ICLK	MCSPI Interface Clock
MCSPI_FCLK	MCSPI Functional Clock
Resets	
Module Reset Input	**Description**
MCSPI_RST	MCSPI Asynchronous Reset
MCSPI_POR_RST	MCSPI Power-On Reset

Copyright © 2024 Texas Instruments Incorporated

Table 12-33. MCSPI Hardware Requests

Interrupt Requests		
Module Interrupt Signal	**Description**	**Type**
MCSPI_INTR_SPI_0	MCSPI Interrupt Request	Level
DMA Events		
Module DMA Event	**Description**	**Type**
MCSPI_DMA_WRITE_EVENT0	MCSPI Channel 0 Transmit (Write) Request Line	Pulse
MCSPI_DMA_READ_EVENT0	MCSPI Channel 0 Receive (Read) Request Line	Pulse
MCSPI_DMA_WRITE_EVENT1	MCSPI Channel 1 Transmit (Write) Request Line	Pulse
MCSPI_DMA_READ_EVENT1	MCSPI Channel 1 Receive (Read) Request Line	Pulse
MCSPI_DMA_WRITE_EVENT2	MCSPI Channel 2 Transmit (Write) Request Line	Pulse
MCSPI_DMA_READ_EVENT2	MCSPI Channel 2 Receive (Read) Request Line	Pulse
MCSPI_DMA_WRITE_EVENT3	MCSPI Channel 3 Transmit (Write) Request Line	Pulse
MCSPI_DMA_READ_EVENT3	MCSPI Channel 3 Receive (Read) Request Line	Pulse

12.1.5.2 MCSPI Environment

This section describes the MCSPI external connections (environment).

Table 12-34 describes the MCSPI I/O signals in master mode.

Table 12-34. MCSPI I/O Signals (Master Mode)

Module Pin	I/O[(1)]	Description
SPICLK	O	MCSPI Serial clock output for master mode.
SPIDAT[0]	O[(2)]	MCSPI Data I/O for master mode.
SPIDAT[1]	I[(3)]	MCSPI Data I/O for master mode.
SPIEN[i]	O	MCSPI Chip-select i output for master mode

(1) I = Input; O = Output
(2) Example configuration only. Can be configured either as input or as output depending on MCSPI_CHCONF_0/1/2/3[18] IS and MCSPI_CHCONF_0/1/2/3[16] DPE0.
(3) Example configuration only. Can be configured either as input or as output depending on MCSPI_CHCONF_0/1/2/3[18] IS and MCSPI_CHCONF_0/1/2/3[17] DPE1.

Table 12-35 describes the MCSPI I/O signals in slave mode.

Table 12-35. MCSPI I/O Signals (Slave Mode)

Module Pin	I/O[(1)]	Description
SPICLK	I	MCSPI serial clock input for slave mode.
SPIDAT[0]	I[(2)]	MCSPI Data I/O for slave mode.
SPIDAT[1]	O[(3)]	MCSPI Data I/O for slave mode.
SPIEN[i]	I[(4)]	MCSPI chip-select i input for slave mode.

(1) I = Input; O = Output
(2) Example configuration only. Can be configured either as input or as output depending on MCSPI_CHCONF_0/1/2/3[18] IS and MCSPI_CHCONF_0/1/2/3[16] DPE0.
(3) Example configuration only. Can be configured either as input or as output depending on MCSPI_CHCONF_0/1/2/3[18] IS and MCSPI_CHCONF_0/1/2/3[17] DPE1.
(4) The chip-select input in slave mode can be selected through the MCSPI_CHCONF_0/1/2/3[22-21] SPIENSLV bit field.

Copyright © 2024 Texas Instruments Incorporated

12.1.5.3 MCSPI Functional Description

12.1.5.3.1 SPI Block Diagram

Figure 12-27 shows the SPI module.

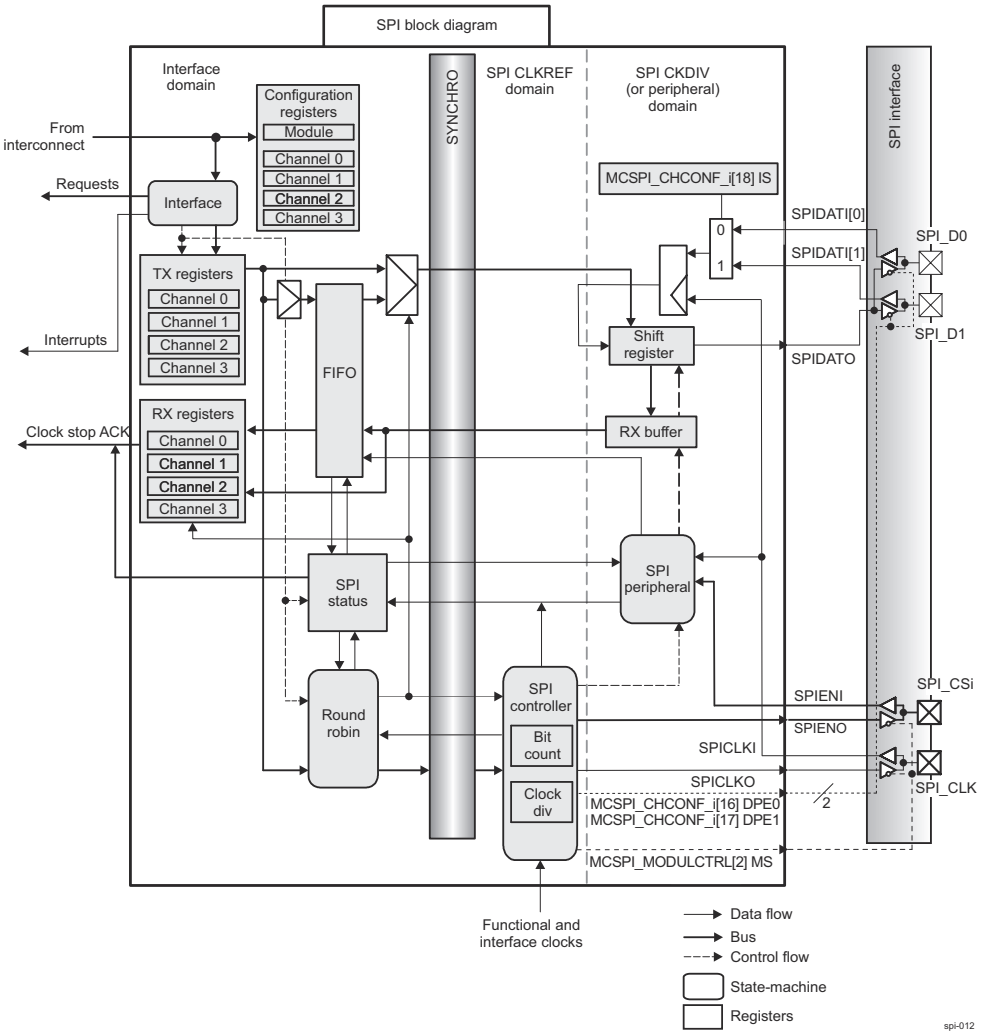


Figure 12-27. SPI Block Diagram

12.1.5.3.2 MCSPI Reset

The MCSPI module can be reset either by hardware or by software reset. All configuration registers and all state machines are reset by the hardware reset signal (MCSPI_RST). MCSPI can be reset by software through the MCSPI_SYSCONFIG[1] SOFTRESET bit. This bit has the same impact on the module as the hardware reset signal. The only exception is that the MCSPI_SYSCONFIG register is not affected by that software reset.

12.1.5.3.3 MCSPI Controller Mode

12.1.5.3.3.1 Controller Mode Features

The MCSPI controller mode supports multichannel communication with up to four independent MCSPI communication channel contexts. The MCSPI initiates a data transfer on the data lines (SPIDAT[0] and SPIDAT[1]) and generates clock (SPICLK) and control (SPIEN[i]) signals.

Connected to multiple external devices, the MCSPI exchanges data with one MCSPI device at a time through two main modes (available in peripheral mode):

- Two-data-pins interface mode (transmit-and-receive mode for full-duplex transmission)
- Single-data-pin interface mode (recommended for half-duplex transmission)

Copyright © 2024 Texas Instruments Incorporated

Note

There is a fixed chip select line allocation in multichannel controller mode. Channel i is mapped to SPIEN[i].

Two DMA request events (read and write) allow synchronized accesses of the DMA controller with the activity of MCSPI.

Three interrupt events can be used for data transmission and reception in controller mode (for more information about interrupts, see Section 13.1.3.4.7.1, *Interrupt Events in Controller Mode*).

12.1.5.3.3.2 Controller Transmit-and-Receive Mode (Full Duplex)

In full-duplex transmission, data is transmitted (shifted out serially on SPIDAT[0]) and received (shifted in serially on SPIDAT[1]) simultaneously on separate data lines.

The controller transmit-and-receive mode is programmable per channel (the MCSPI_CHCONF_0/1/2/3[13-12] TRM bit field).

Channel access to the shift registers for transmission/reception is based on the MCSPI_TX_0/1/2/3 transmitter register state, the MCSPI_RX_0/1/2/3 receiver register state, and round-robin arbitration.

Channels that meet the following rules are included in the round-robin list of active channels scheduled for transmission and/or reception. The arbiter skips channels that do not meet the rules and searches in the rotation for the next enabled channel.

- Rule 1: Only enabled channels (the MCSPI_CHCTRL_0/1/2/3[0] EN bit) can be scheduled for transmission and/or reception.
- Rule 2: If its MCSPI_TX_0/1/2/3 transmitter register is not empty (the MCSPI_CHSTAT_0/1/2/3[1] TXS bit), an enabled channel can be scheduled when the shift register is assigned. If the MCSPI_TX_0/1/2/3 register is empty when the shift register is assigned, the TXx_UNDERFLOW event is activated, and the next enabled channel with new data to transmit is scheduled (see also transmit-only mode).
- Rule 3: An enabled channel can be scheduled if its receive register is not full (the MCSPI_CHSTAT_0/1/2/3[0] RXS bit) when the shift register is assigned (see also receive-only mode). Therefore, the MCSPI_RX_0/1/2/3 register cannot be overwritten. The MCSPI_IRQSTATUS[3] RX0_OVERFLOW bit is never set to this mode.

When MCSPI word transfer completes (the MCSPI_CHSTAT_0/1/2/3[2] EOT bit is set), the updated MCSPI_TX_0/1/2/3 register of the next scheduled channel is loaded into the shift register. The serialization (transmit-and-receive) starts depending on the channel communication configuration. When serialization completes, the received data transfers to the channel receive register.

The serial clock (SPICLK) synchronizes shifting and sampling of the information on the two serial data lines (SPIDAT[0] and SPIDAT[1]). Each time a bit transfers out from the controller, 1 bit transfers in from the peripheral.

Figure 12-28 shows an example of a full-duplex system with a controller device on the left and a peripheral device on the right. After eight cycles of the serial clock SPICLK, WordA transfers from the controller to the peripheral. At the same time, WordB transfers from the peripheral to the controller.

Copyright © 2024 Texas Instruments Incorporated

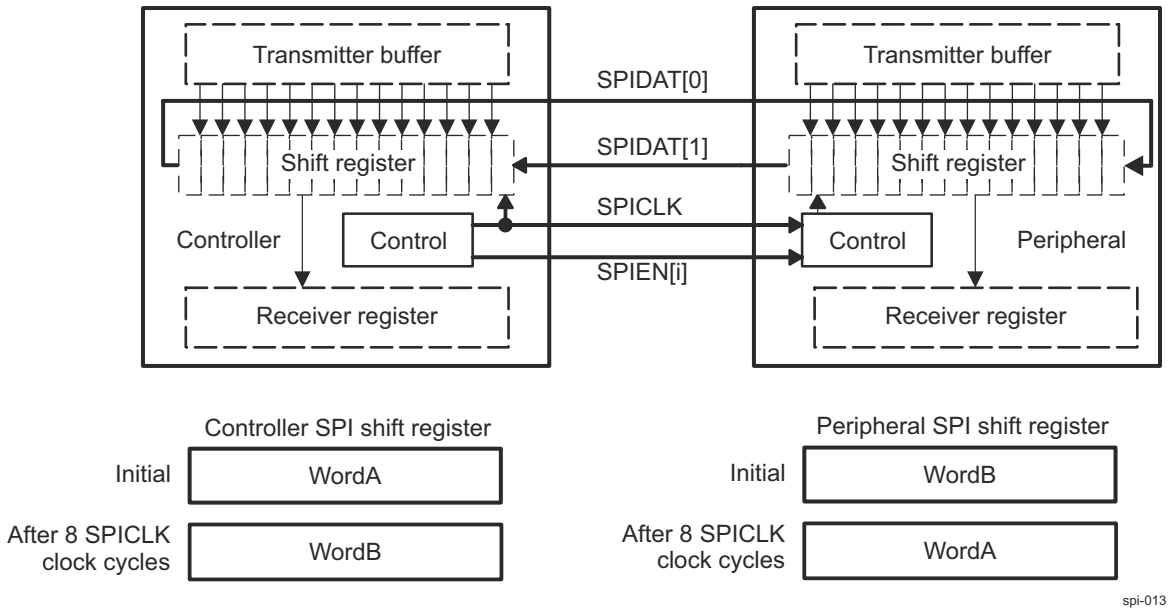


Figure 12-28. MCSPI Full-Duplex Transmission (Example)

12.1.5.3.3.3 Controller Transmit-Only Mode (Half Duplex)

The controller transmit-only mode prevents the processor from reading the MCSPI_RX_0/1/2/3 register (minimizing data movement) when only transmission is meaningful.

The controller transmit-only mode is programmable per channel (the MCSPI_CHCONF_0/1/2/3[13-12] TRM bit field). Transmission starts only after data is loaded into the MCSPI_TX_0/1/2/3 register.

Rule 1 and Rule 2, defined in Section 12.1.5.3.3.2, apply in this mode.

Rule 3, defined in Section 12.1.5.3.3.2, does not apply.

In controller transmit-only mode, the MCSPI_RX_0/1/2/3 register state FULL does not prevent transmission and the MCSPI_RX_0/1/2/3 register is always overwritten with the new MCSPI word. This event is not significant when only transmission is meaningful. Thus, the RX0_OVERFLOW bit in the MCSPI_IRQSTATUS register is never set in this mode.

The hardware automatically disables the RX_FULL interrupt and the DMA read requests.

The transfer status is given by the MCSPI_CHSTAT_0/1/2/3[2] EOT bit.

12.1.5.3.3.4 Controller Receive-Only Mode (Half Duplex)

The controller receive mode prevents the processor from refilling the MCSPI_TX_0/1/2/3 register (minimizing data movement) when only reception is meaningful.

The controller receive mode is programmable per channel (the MCSPI_CHCONF_0/1/2/3[13-12] TRM bit field).

The controller receive-only mode enables channel scheduling only on the empty state of the MCSPI_RX_0/1/2/3 register.

Rule 1 and Rule 3, defined in Section 12.1.5.3.3.2, apply in this mode.

Rule 2, defined in Section 12.1.5.3.3.2, does not apply.

In the controller receive-only mode, software must write dummy data to the MCSPI_TX_0/1/2/3 register. Only one dummy write is enough to receive any number of words from the peripheral. Software must ensure that the MCSPI_TX_0/1/2/3 register is always full (the TXx_EMPTY bits of MCSPI_IRQSTATUS) when receiving. The content of the MCSPI_TX_0/1/2/3 register is always loaded into the shift register when the shift register is assigned. After writing the dummy data to the MCSPI_TX_0/1/2/3 register, the TXx_EMPTY and TXx_UNDERFLOW bits in the MCSPI_IRQSTATUS register are never set in receive-only mode.

Copyright © 2024 Texas Instruments Incorporated

The MCSPI_CHSTAT_0/1/2/3[2] EOT bit gives the status of serialization. The RXx_FULL bits of the MCSPI_IRQSTATUS register are set when received data is loaded from the shift register to the corresponding MCSPI_RX_0/1/2/3 register. The MCSPI_IRQSTATUS[3] RX0_OVERFLOW bit is never set in this mode.

12.1.5.3.3.5 Single-Channel Controller Mode

When the MCSPI is configured as a controller device with a single enabled channel (MCSPI_MODULCTRL[2] MS = 0 and MCSPI_MODULCTRL[0] SINGLE = 1), the assertion of the SPIEN[i] signal is optional depending on device connected to the controller. In 3-pin mode (MCSPI_MODULCTRL[1] PIN34 = 1) the controller starts transmitting data when a write to the MCSPI_TX_0/1/2/3 register or the FIFO is performed. In 4-pin mode (MCSPI_MODULCTRL[1] PIN34 = 0) the assertion and de-assertion of SPIEN[i] is controlled by software using the MCSPI_CHCONF_0/1/2/3[20] FORCE bit.

12.1.5.3.3.5.1 Programming Tips When Switching to Another Channel

When a single channel is enabled and data transfer is ongoing:

- Wait for the MCSPI word transfer to complete (wait until the MCSPI_CHSTAT_0/1/2/3[2] EOT bit is set to 1) before disabling the current channel and enabling a different channel.
- Disable the current channel, and then enable the other channel.

12.1.5.3.3.5.2 Force SPIEN[i] Mode

Continuous transfers are allowed manually by keeping the SPIEN[i] signal active for successive MCSPI words transfer. Several sequences (configuration/enable/disable of the channel) can be run without deactivating the SPIEN[i] line. This mode is supported by all channels and any controller sequence can be used (transmit-receive, transmit-only, receive-only).

Keeping the SPIEN[i] active mode is supported when:

- A single channel is used (with the MCSPI_MODULCTRL[0] SINGLE bit set to 1).
- Transfer parameters are loaded in the configuration register of the appropriate channel (MCSPI_CHCONF_0/1/2/3).

  The state of the SPIEN[i] signal is programmable:
  - Writing 1 to the MCSPI_CHCONF_0/1/2/3[20] FORCE bit drives the SPIEN[i] line high when the MCSPI_CHCONF_0/1/2/3[6] EPOL bit is set to 0. SPIEN[i] is driven low when the MCSPI_CHCONF_0/1/2/3[6] EPOL bit is set to 1.
  - Writing 0 to the MCSPI_CHCONF_0/1/2/3[20] FORCE bit drives the SPIEN[i] line low when the MCSPI_CHCONF_0/1/2/3[6] EPOL bit is set to 0. SPIEN[i] is driven high when the MCSPI_CHCONF_0/1/2/3[6] EPOL bit is set to 1.
- A single channel is enabled (the MCSPI_CHCTRL_0/1/2/3[0] EN bit is set to 1). The first enabled channel activates the SPIEN[i] line.

When the channel is enabled, the SPIEN[i] signal activates with the programmed polarity. As in the multichannel controller mode, the transfer start depends on the status of the MCSPI_TX_0/1/2/3 register (the MCSPI_CHSTAT_0/1/2/3[1] TXS bit), the status of the MCSPI_RX_0/1/2/3 register (the MCSPI_CHSTAT_0/1/2/3[1] RXS bit), and the defined mode (the MCSPI_CHCONF_0/1/2/3[13-12] TRM bit field) of the channel enabled.

The MCSPI_CHSTAT_0/1/2/3[2] EOT bit gives the transfer status of each MCSPI word. The RXx_FULL bit in the MCSPI_IRQSTATUS register is set when received data is loaded from the shift register to the MCSPI_RX_0/1/2/3 register.

A change in the configuration parameters is propagated directly on the MCSPI interface. If the SPIEN[i] signal is activated, ensure that the configuration is changed only between MCSPI words to avoid corrupting the current transfer.

Note

To avoid data corruption, SPIEN[i] polarity and SPICLK phase and SPICLK polarity must not be modified when the SPIEN[i] signal is activated.

Copyright © 2024 Texas Instruments Incorporated

A delay between MCSPI words that requires the connected MCSPI peripheral device to switch from one configuration to another (for instance, from transmit-only to receive-only) must be handled by software.

At the end of the last MCSPI word, the channel must be deactivated (the MCSPI_CHCTRL_0/1/2/3[0] EN bit set to 0) and SPIEN[i] can be forced to its INACTIVE state using the MCSPI_CHCONF_0/1/2/3[20] FORCE bit.

Figure 12-29 and Figure 12-30 show successive transfers with SPIEN[i] maintained active low with a different configuration for each MCSPI word in single-data-pin and dual-data-pin interface modes, respectively.

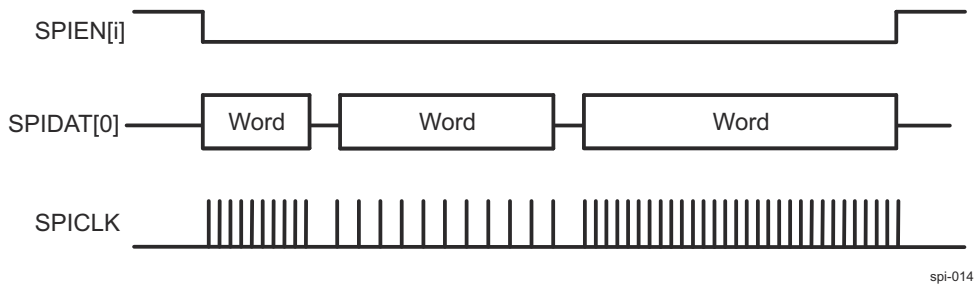


Figure 12-29. Continuous Transfers With SPIEN[i] Maintained Active (Single-Data-Pin Interface Mode)

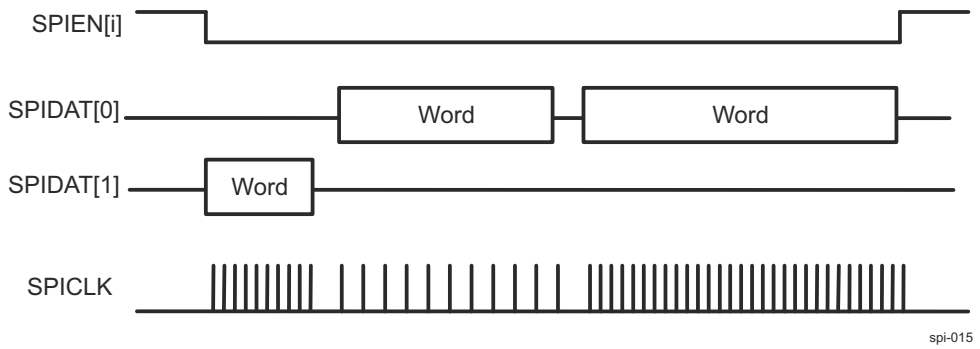


Figure 12-30. Continuous Transfers With SPIEN[i] Maintained Active (Dual-Data-Pin Interface Mode)

Note

The SPIEN[i] signal can be maintained active via software using the MCSPI_CHCONF_0/1/2/3[20] FORCE bit only when the MCSPI_MODULCTRL[0] SINGLE bit is set to 0x1.

12.1.5.3.3.5.3 Turbo Mode

Turbo mode improves the throughput of the MCSPI interface when a single channel is enabled by allowing transfers until the shift register and the MCSPI_RX_0/1/2/3 register are full. Turbo mode is time saving when a transfer exceeds two words. This mode is programmable per channel (through the MCSPI_CHCONF_0/1/2/3[9] TURBO bit).

When several channels are enabled, the TURBO bit has no effect and the channel access to the shift registers remains as previously described.

In turbo mode, Rule 1 and Rule 2 apply, but Rule 3 does not (see Section 12.1.5.3.3.2, *Controller Transmit-and-Receive Mode (Full Duplex)*). An enabled channel can be scheduled if its receive register is full (the MCSPI_CHSTAT_0/1/2/3[0] RXS bit) when the shift-register is assigned until the shift register is full.

The MCSPI_RX_0/1/2/3 register cannot be overwritten in turbo mode. Consequently, the MCSPI_IRQSTATUS[3] RX0_OVERFLOW bit is never set in this mode.

12.1.5.3.3.6 Start-Bit Mode

In start-bit mode, an extended bit is added before the MCSPI word to indicate whether the next MCSPI word must be handled as a command or as data. This feature is available only in controller mode. Start-bit mode

Copyright © 2024 Texas Instruments Incorporated

cannot be used at the same time as turbo mode and/or force SPIEN[i] mode. In this case, only one channel can be used; round-robin arbitration is not possible.

This mode is programmable per channel by setting the MCSPI_CHCONF_0/1/2/3[23] SBE bit to 1. The polarity of the extended bit is programmable per channel. When the MCSPI_CHCONF_0/1/2/3[24] SBPOL bit is set to 0, the MCSPI word must be handled as a command. When the MCSPI_CHCONF_0/1/2/3[24] SBPOL bit is set to 1, the MCSPI word must be handled as data. Moreover, start-bit polarity can be changed dynamically during start-bit transfer without disabling the channel for reconfiguration; in this case, users must configure the MCSPI_CHCONF_0/1/2/3[24] SBPOL bit before writing the MCSPI word to be transmitted to the TX register.

12.1.5.3.3.7 Chip-Select Timing Control

The chip-select (CS) timing control is available only in controller mode with automatic CS generation (the MCSPI_MODULCTRL[0] SINGLE bit set to 0) to add a programmable delay between CS assertion and first clock edge, or CS removal and last clock edge. This option is available only in 4-pin mode when MCSPI_MODULCTRL[1] PIN34 set to 0.

This mode is programmable per channel through the MCSPI_CHCONF_0/1/2/3[26-25] TCS0 bit field.

Figure 12-31 shows the CS SPIEN timing controls.

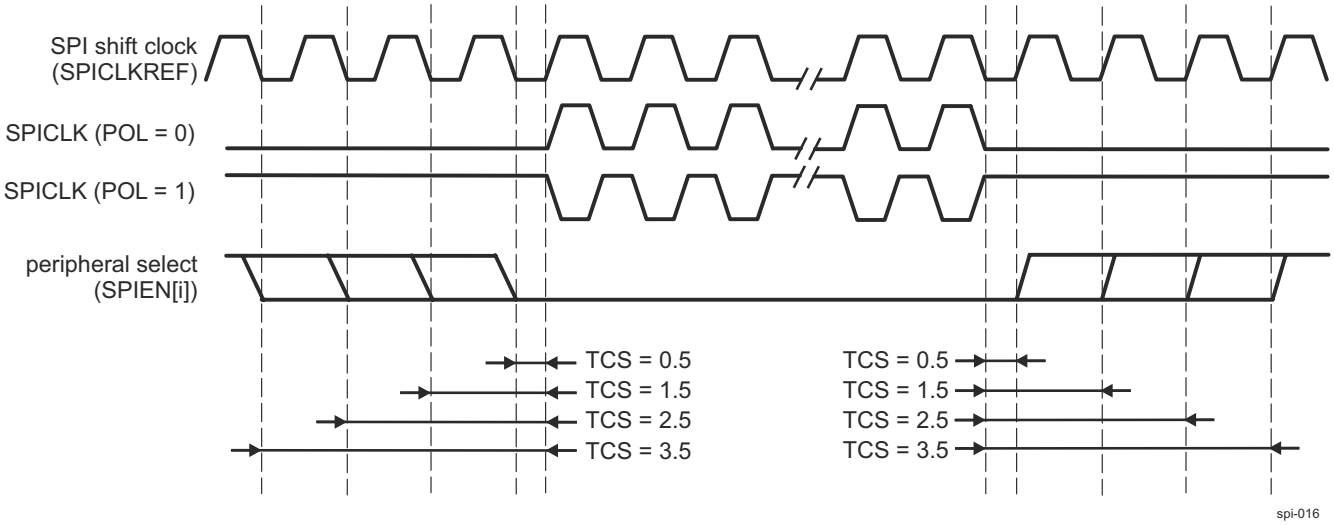


Figure 12-31. CS SPIEN Timing Controls

Note

Because of the design implementation for transfers using a clock divider ratio set to 1 (clock bypassed), a half cycle must be added to the value between CS assertion and the first clock edge with PHA = 1 or between CS removal and the last clock edge with PHA = 0.

12.1.5.3.3.8 Programmable MCSPI Clock (SPICLK)

In controller mode, the baud rate of the MCSPI serial clock is programmable.

An internal reference clock, SPICLKREF, is used as input of a programmable divider (the MCSPI_CHCONF_0/1/2/3[5-2] CLKD bit field) to generate the bit rate of the serial output clock SPICLK. Table 12-36 summarizes the supported divisor values.

Table 12-36. MCSPI Controller Clock Rates

Divider	Clock Rate
1	50 MHz[(1)]
2	25 MHz[(1)]
4	12.5 MHz
8	6.25 MHz

Copyright © 2024 Texas Instruments Incorporated

Table 12-36. MCSPI Controller Clock Rates (continued)

Divider	Clock Rate
16	3.125 MHz
32	1.5625 MHz
64	781.25 kHz
128	390 kHz(2)
256	195 kHz(2)
512	97.7 kHz(2)
1024	48.8 kHz(2)
2048	24.4 kHz(2)
4096	12.2 kHz(2)
8192 and higher: Division not supported	–

(1) These frequencies are not necessarily supported by all MCSPI modules. For more information, see the *Timing Requirements and Switching Characteristics* chapter in the device-specific Data sheet.
(2) Approximate Frequency

12.1.5.3.3.8.1 Clock Ratio Granularity

By default, the clock division ratio is defined by the MCSPI_CHCONF_0/1/2/3[5-2] CLKD bit field with power-of-2 granularity leading to a clock division in the range 1 to 4096; in this case, the duty cycle is always 50 percent. With the MCSPI_CHCONF_0/1/2/3[29] CLKG bit, clock division granularity can be changed to one clock cycle; in that case the MCSPI_CHCTRL_0/1/2/3[15-8] EXTCLK bit field is concatenated with the MCSPI_CHCONF_0/1/2/3[5-2] CLKD bit field to give a 12-bit-wide division ratio in the range 1 to 4096.

When granularity is one clock cycle (the CLKG bit set to 1), for the odd value of the clock ratio, the clock high level lasts one clock cycle more than the low level, depending on the MCSPI_CHCONF_0/1/2/3[1] POL and MCSPI_CHCONF_0/1/2/3[0] PHA bits (see Table 12-37).

Table 12-37. CLKSPIO High/Low Time Computation

Clock Ratio F_{RATIO}	CLKSPIO High Time	CLKSPIO Low Time
1	T_{HIGH_REF}	T_{LOW_REF}
Even >= 2	T_ref * (F_{RATIO}/2)	T_ref * (F_{RATIO}/2)
Odd >= (POL = PHA)	T_ref * (F_{RATIO}– 1)/2	T_ref * (F_{RATIO} + 1)/2
Odd >= (POL =! PHA)	T_ref * (F_{RATIO}+ 1)/2	T_ref * (F_{RATIO} – 1)/2

Note

F_{RATIO} = SPICLK frequency (F_{OUT}) division ratio
T_{HIGH} = SPICLK high time period
T_{LOW} = SPICLK low time period
T_ref = MCSPI_FCLK period
T_{HIGH_REF} = MCSPI_FCLK high time period
T_{LOW_REF} = MCSPI_FCLK low time period

If the CLKG bit is set to 1; F_{RATIO} = EXTCLK concatenated with CLKD + 1.

Copyright © 2024 Texas Instruments Incorporated

For odd ratio values, the duty cycle is calculated as follows:

Duty_cycle = $(1 – 1/F_{RATIO})/2$

Table 12-38 shows examples of clock granularity with a clock source frequency of 50 MHz.

Table 12-38. Clock Granularity Examples

EXTCLK	CLKD	CLKG	F_{RATIO}	PHA	POL	T_{HIGH} (ns)	T_{LOW} (ns)	T_{PERIOD} (ns)	Duty Cycle	F_{OUT} (MHz)
X	0	0	1	X	X	10.0	10.0	20.0	50–50	50
X	1	0	2	X	X	20.0	20.0	40.0	50–50	25
X	2	0	4	X	X	40.0	40.0	80.0	50–50	12.5
X	3	0	8	X	X	80.0	80.0	160.0	50–50	6.2
0	0	1	1	X	X	10.0	10.0	20.0	50–50	50
0	1	1	2	X	X	20.0	20.0	40.0	50–50	25
0	2	1	3	1	0	40.0	20.0	60.0	66–33	16.6
0	2	1	3	1	1	20.0	40.0	60.0	33–66	16.6
0	3	1	4	X	X	40.0	40.0	80.0	50–50	12.5
5	0	1	81	1	0	820.0	800.0	1620.0	50.6–49.4	0.617
5	7	1	88	X	X	880.0	880.0	1760.0	50–50	0.568

12.1.5.3.4 MCSPI Peripheral Mode

To select the MCSPI peripheral mode, set the MCSPI_MODULCTRL[2] MS bit.

A MCSPI peripheral device can be connected to up to four external MCSPI controller devices but handles transactions with one MCSPI controller device at a time.

In peripheral mode, the MCSPI initiates data transfer on the data lines (SPIDAT[0] and SPIDAT[1]) when it is selected by an active control signal (SPIEN[i]) and receives an MCSPI clock (SPICLK) from the external MCSPI controller device. Only channel 0 can be configured as a peripheral but through the MCSPI_CHCONF_0[22-21] SPIENSLV bit field any of the SPIEN[i] signals can be used to select the MCSPI module. In peripheral mode and when the MCSPI_MODULCTRL[1] PIN34 is set to 0x0 (default behaviour), the MCSPI uses the edge of SPIEN[i] to detect word length. For this reason, SPIEN[i] must become inactive between each word.

When the MCSPI_MODULCTRL[1] PIN34 is set to 0x0, the MCSPI does not support SPIEN[i] active between MCSPI words. In this case, the MCSPI uses the edge to detect word length.

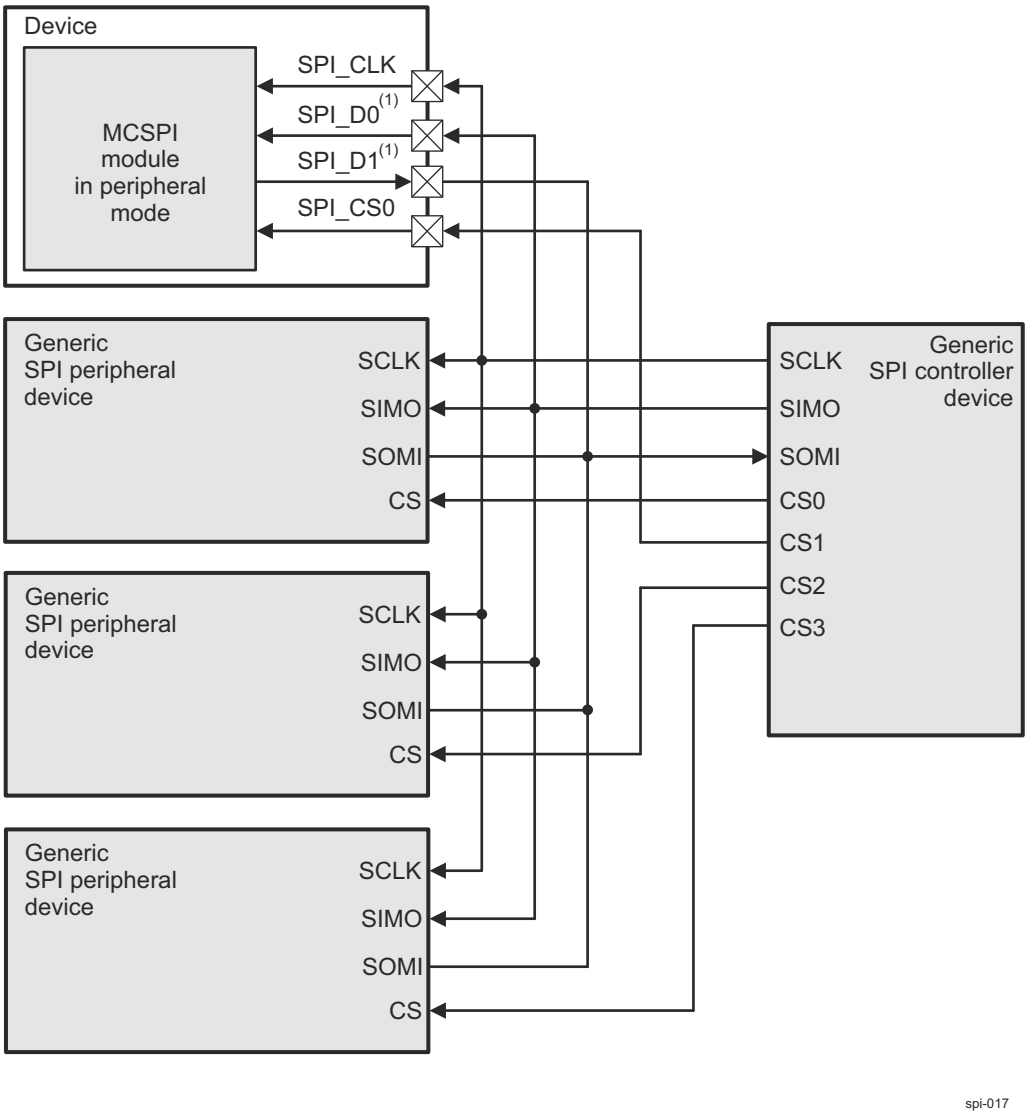
When the MCSPI_MODULCTRL[1] PIN34 is set to 0x1, a multiword transfer can be performed without needing the external MCSPI controller to deactivate SPIEN[i] between each word as in this case the MCSPI module works in 3-pin peripheral mode and SPIEN[i] is not needed.

12.1.5.3.4.1 Dedicated Resources

Only channel 0 can be enabled in peripheral mode.

Figure 12-32 shows an example of four peripherals wired on a single controller device.

Copyright © 2024 Texas Instruments Incorporated



A. Direction depends on MCSPI_CHCONF_0/1/2/3[16] DPE0, MCSPI_CHCONF_0/1/2/3[17] DPE1 and MCSPI_CHCONF_0/1/2/3[18] IS bits

Figure 12-32. Example of MCSPI Peripheral With One Controller and Multiple Peripheral Devices on Channel 0

Channel 0 in peripheral mode has the following resources:

- Its own channel enable, programmable with the MCSPI_CHCTRL_0[0] EN bit. This channel must be enabled before transmission and reception.
- For this mode, the peripheral-select signal can be detected on any of the SPIEN[i] ports. This is programmable with the MCSPI_CHCONF_0[22-21] SPIENSLV bit field.
- Its own transmitter register, MCSPI_TX_0, on top of the common transmit shift register. If the MCSPI_TX_0 register is empty, the MCSPI_CHSTAT_0[1] TXS bit is set. If MCSPI is selected by an external controller (the active signal on the SPIEN[i] port assigned to channel 0), the MCSPI_TX_0 register content of channel 0 is always loaded into the shift register, whether its content is updated or not. The MCSPI_TX_0 register must be loaded before MCSPI is selected by a controller.
- Its own receiver register, MCSPI_RX_0, on top of the common receive shift register. If the MCSPI_RX_0 register is full, the MCSPI_CHSTAT_0[0] RXS bit is set.

Copyright © 2024 Texas Instruments Incorporated

Note

The MCSPI_TX_1/2/3 and MCSPI_RX_1/2/3 registers are not used. Reading from or writing to a channel register other than channel 0 has no effect.

- Its own communication configuration with the following parameters through the MCSPI_CHCONF_0:
  - Transmit and receive modes, programmable with the TRM field
  - Interface mode (two data pins or single data pin) and data pins assignment, both programmable with the IS and DPE bits. (The MCSPI modules are in peripheral mode after reset and must be properly configured for the modules to act in controller mode.)
  - MCSPI word length, programmable with the WL bit
  - SPIEN[i] polarity, programmable with the EPOL bit
  - SPICLK polarity, programmable with the POL bit
  - SPICLK phase, programmable with the PHA bit

  The SPICLK frequency of a transfer is controlled by the external MCSPI controller connected to the MCSPI peripheral device. The MCSPI_CHCONF_0[5-2] CLKD bit field is not used in peripheral mode.

Note

The configuration of the channel can be loaded in the MCSPI_CHCONF_0 only when the channel is disabled.

- Two DMA request events, read and write, synchronize read/write accesses of the DMA controller with the activity of MCSPI. DMA requests are asserted using the MCSPI_CHCONF_0[15] DMAR bit for reading and the MCSPI_CHCONF_0[14] DMAW bit for writing.
- Four interrupt events (see Section 12.1.5.3.7.2, *Interrupt Events in Peripheral Mode*).

12.1.5.3.4.2 Peripheral Transmit-and-Receive Mode

The peripheral receive mode is programmable (set the MCSPI_CHCONF_0[13-12] TRM bit field to 0x0).

In peripheral transmit-and-receive mode, the MCSPI_TX_0 register must be loaded before MCSPI is selected by an external MCSPI controller device.

After a channel is enabled, transmission and reception proceed with interrupt and DMA request events.

The MCSPI_TX_0 register content is always loaded in the shift register whether it is updated or not. The event TX0_UNDERFLOW is activated accordingly and does not prevent transmission.

When the MCSPI word transfer completes (the MCSPI_CHSTAT_0[2] EOT bit is set to 1), the received data is transferred to the channel receive register.

To use MCSPI as a peripheral transmit-only device, the RX0_FULL and RX0_OVERFLOW interrupts and DMA read requests must be disabled due to the state of the MCSPI_RX_0 register (see Section 12.1.5.3.7.2, *Interrupt Events in Peripheral Mode*).

12.1.5.3.4.3 Peripheral Transmit-Only Mode

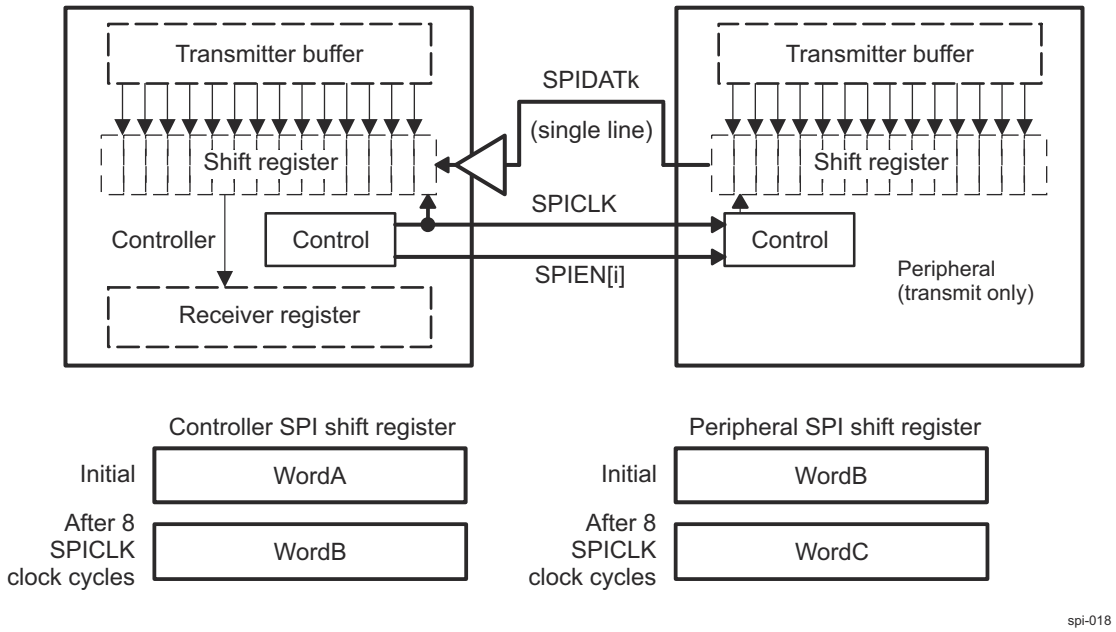
The peripheral transmit-only mode is programmable (set the MCSPI_CHCONF_0[13-12] TRM bit field to 0x2) and avoids the requirement for the processor to read the MCSPI_RX_0 register (minimizing data movement) only when transmission is meaningful.

To use the MCSPI as a peripheral transmit-only device, the RX0_FULL and RX0_OVERFLOW interrupts and DMA read requests must be disabled due to the state of the MCSPI_RX_0 register.

When the MCSPI word transfer completes, the MCSPI_CHSTAT_0[2] EOT bit is set.

Figure 12-33 shows a half-duplex system with a controller device on the left and a transmit-only peripheral device on the right. Each time a bit transfers out from the peripheral, 1 bit transfers in the controller. After eight cycles of the serial clock SPICLK, WordB transfers from the peripheral to the controller.

Copyright © 2024 Texas Instruments Incorporated



k = 0 or 1 depending on MCSPI_CHCONF_0/1/2/3[16] DPE0, MCSPI_CHCONF_0/1/2/3[17] DPE1 and MCSPI_CHCONF_0/1/2/3[18] IS bits

Figure 12-33. MCSPI Half-Duplex Transmission (Transmit-Only peripheral)

12.1.5.3.4.4 Peripheral Receive-Only Mode

The peripheral receive mode is programmable (set the MCSPI_CHCONF_0[13-12] TRM bit field to 0x1).

In receive-only mode, the MCSPI_TX_0 register must be loaded before the MCSPI is selected by an external MCSPI controller device. The MCSPI_TX_0 register content is always loaded into the shift register whether it is updated or not. The TX0_UNDERFLOW event is activated accordingly and does not prevent transmission.

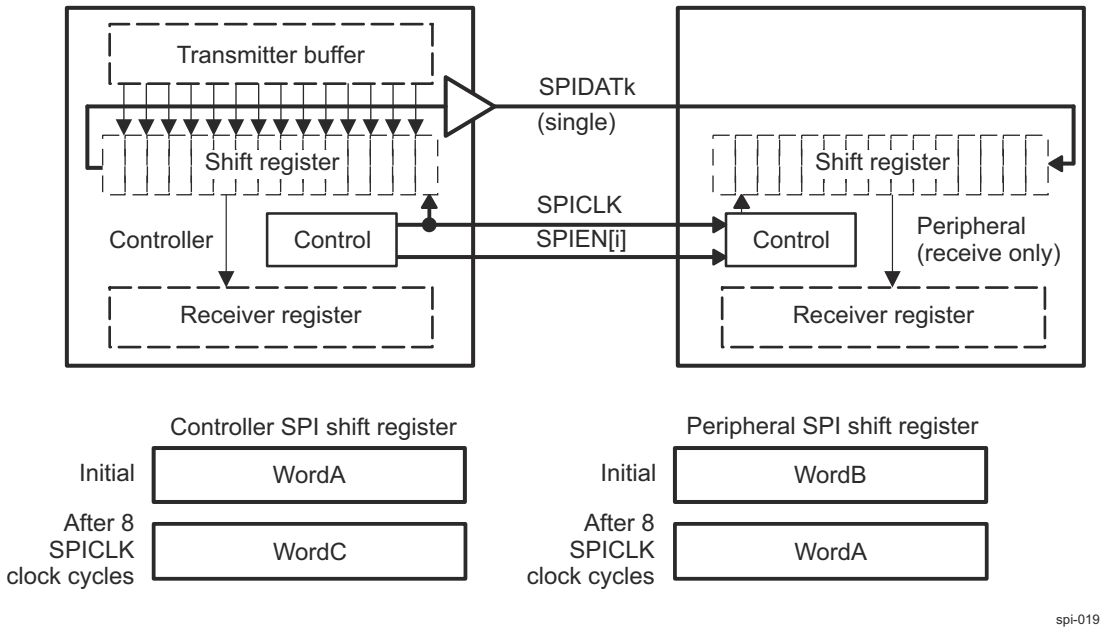
When the MCSPI word transfer completes (the MCSPI_CHSTAT_0[2] EOT bit is set to 1), the received data is transferred to the channel receive register.

To use the MCSPI as a peripheral receive-only device, the TX0_EMPTY and TX0_UNDERFLOW interrupts and the DMA write requests must be disabled due to the state of the MCSPI_TX_0 register.

For a full-duplex transmission, the serial clock (SPICLK) synchronizes shifting and sampling of the information on the single serial data line. For full duplex, two data lines are required. If SPICLK synchronizes on a single serial data line, the data line should be half-duplex.

Figure 12-34 shows a half-duplex system with a controller device on the left and a receive-only peripheral device on the right. Each time a bit transfers out from the controller, 1 bit transfers in from the peripheral. After eight cycles of the serial clock SPICLK, WordA transfers from the controller to the peripheral.

Copyright © 2024 Texas Instruments Incorporated



k = 0 or 1 depending on MCSPI_CHCONF_0/1/2/3[16] DPE0, MCSPI_CHCONF_0/1/2/3[17] DPE1 and MCSPI_CHCONF_0/1/2/3[18] IS bits

Figure 12-34. MCSPI Half-Duplex Transmission (Receive-Only Peripheral)

12.1.5.3.5 MCSPI 3-Pin or 4-Pin Mode

Depending on targeted application the MCSPI interface can be configured to use 3 or 4 pins through the MCSPI_MODULCTRL[1] PIN34 bit. If this bit is set to 0, MCSPI is in 4-pin mode using the SPICLK, SPIDAT[0], SPIDAT[1] and SPIEN[i] signals. If PIN34 is set to 1 the controller is in 3-pin mode and SPIEN[i] is not used. In this mode all options related to chip select management are useless (EPOL, FORCE and TCS0 bits of MCSPI_CHCONF_0/1/2/3). 3-pin and 4-pin operation applies to both controller and peripheral modes.

12.1.5.3.6 MCSPI FIFO Buffer Management

The MCSPI controller has a built-in 64-byte buffer to unload the DMA or interrupt handler and improve data throughput.

This buffer can be used by only one channel at a time and is selected by setting the MCSPI_CHCONF_0/1/2/3[28] FFER or MCSPI_CHCONF_0/1/2/3[27] FFEW bit to 1. If several channels are selected and several FIFO enable bit fields are set to 1, the controller forces the buffer not to be used; the driver must set only one FIFO enable bit field.

The buffer can be used in the following modes:

- Controller or peripheral mode
- Transmit-only, receive-only, or transmit-and-receive mode
- Single channel or turbo mode, or normal round-robin mode. In round-robin mode the buffer is used by only one channel.

Every word length (MCSPI_CHCONF_0/1/2/3[11-7] WL) is supported.

In transmit-and-receive mode, the buffer can be used in transmit (see Figure 12-35) or receive (see Figure 12-36) directions, or in both directions. If only one direction is chosen in transmit-and-receive mode, the full buffer is used for this direction. In both directions, the buffer is split into two halves, one for each direction (see Figure 12-37).

Copyright © 2024 Texas Instruments Incorporated

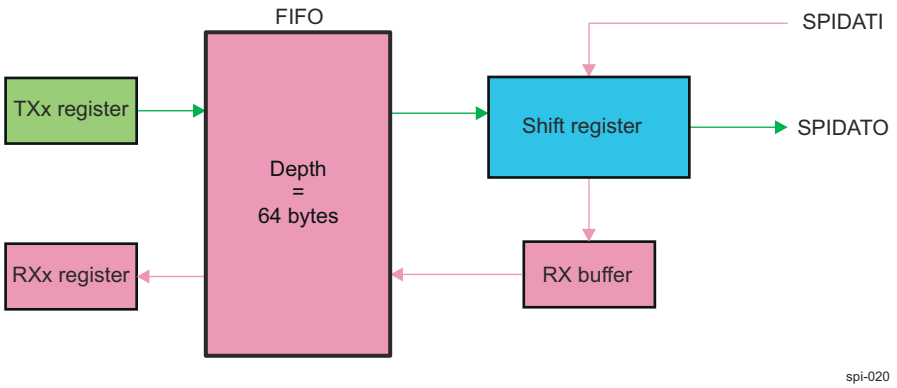


Figure 12-35. Buffer Used in Transmit Direction Only

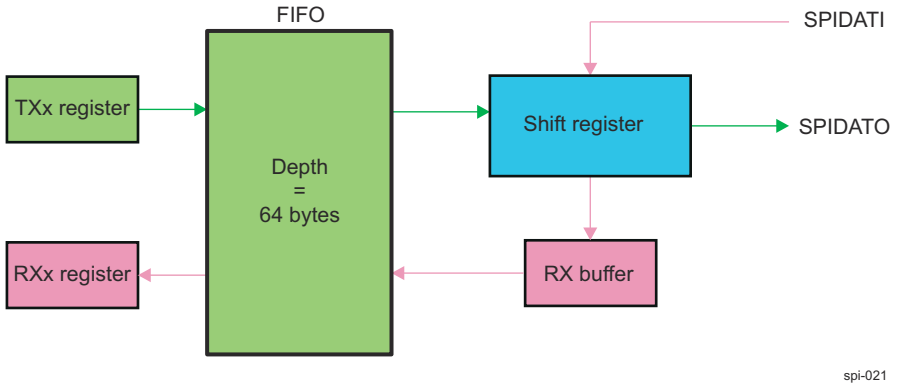


Figure 12-36. Buffer Used in Receive Direction Only

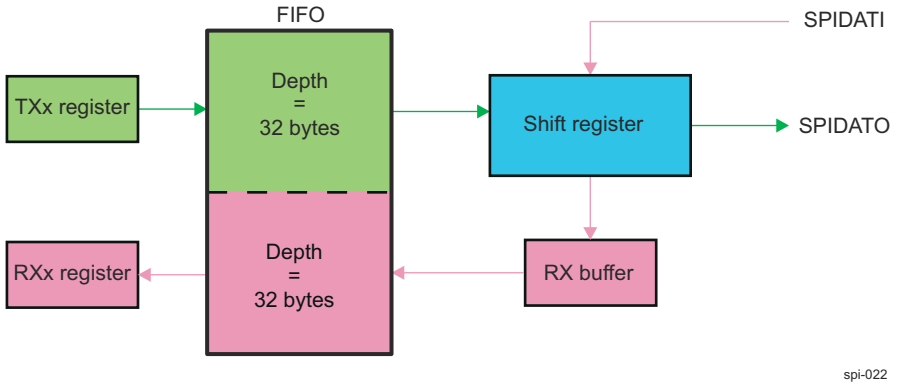


Figure 12-37. Buffer Used for Transmit and Receive Directions

Two levels (MCSPI_XFERLEVEL[5-0] AEL and MCSPI_XFERLEVEL[13-8] AFL) rule the buffer management. The granularity of these levels is 1 byte; it is not aligned with the MCSPI word length. The driver must set these values as a multiple of the MCSPI word length defined in WL. Table 12-39 lists the number of bytes written in the FIFO, depending on the word length.

Table 12-39. FIFO Writes, Word Length Relationship

	MCSPI Word Length (WL)		
	3 ≤ WL≤ 7	**8 ≤ WL ≤ 15**	**16 ≤ WL ≤ 31**
Number of bytes written in the FIFO	1 byte	2 bytes	4 bytes

The FIFO buffer pointers are reset when the corresponding channel is enabled or the FIFO configuration changes.

Copyright © 2024 Texas Instruments Incorporated

12.1.5.3.6.1 Buffer Almost Full

The MCSPI_XFERLEVEL[15-8] AFL bit field is needed when the buffer is used to receive an MCSPI word from a peripheral (the MCSPI_CHCONF_0/1/2/3[28] FFER bit must be set to 1). It defines the almost-full buffer status. See Figure 12-38.

When the FIFO pointer reaches this level, an interrupt or a DMA request is sent to the processor to enable the system to read AFL + 1 bytes from the receive register.

Note

AFL + 1 must correspond to a multiple value of the MCSPI_CHCONF_0/1/2/3[11-7] WL bit field.

When DMA is used, the request is de-asserted after the first receive register read.

No new request is asserted again as long as the system has not performed the correct number of read accesses.

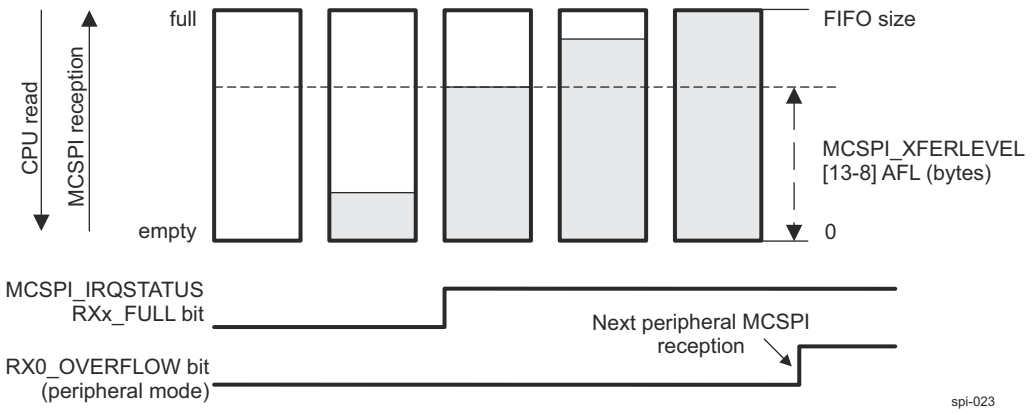


Figure 12-38. Buffer Almost Full Level (AFL)

Note

The MCSPI_IRQSTATUS register bits are not available in DMA mode. In DMA mode, the request is asserted on the same conditions as the MCSPI_IRQSTATUS RXx_FULL flag.

12.1.5.3.6.2 Buffer Almost Empty

The MCSPI_XFERLEVEL[7-0] AEL bit field is needed when the buffer is used to transmit an MCSPI word to a peripheral (the MCSPI_CHCONF_0/1/2/3[27] FFEW bit must be set to 1). It defines the almost-empty buffer status. See Figure 12-39.

When the FIFO pointer does not reach this level, an interrupt or a DMA request is sent to the processor to enable the system to write AEL + 1 bytes to the transmit register.

Note

AEL + 1 must correspond to a multiple value of the MCSPI_CHCONF_0/1/2/3[11-7] WL bit field.

When DMA is used, the request is de-asserted after the first transmit register write.

No new request is asserted again as long as the system has not performed the correct number of write accesses.

Copyright © 2024 Texas Instruments Incorporated

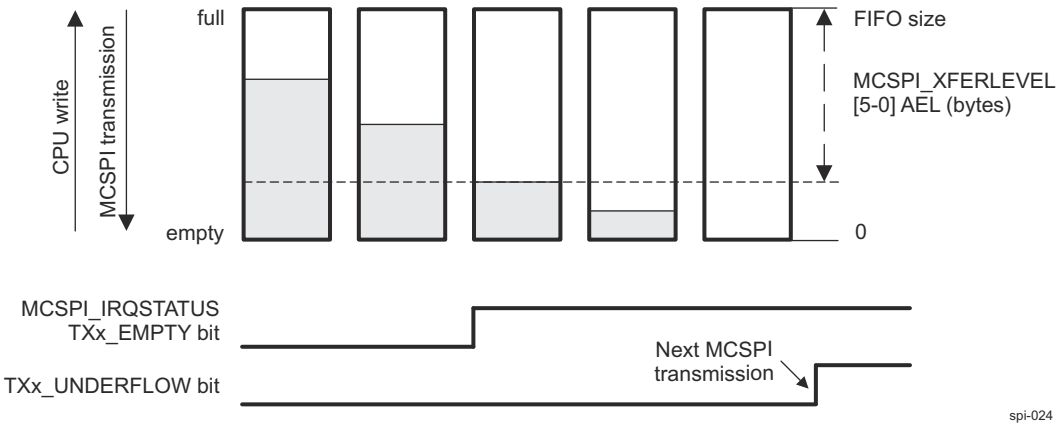


Figure 12-39. Buffer Almost Empty Level (AEL)

Note

The MCSPI_IRQSTATUS register bits are not available in DMA mode. In DMA mode, the request is asserted on the same conditions as the MCSPI_IRQSTATUS TXx_EMPTY flag.

12.1.5.3.6.3 End of Transfer Management

When the FIFO buffer is enabled for a channel, the user must previously configure in the MCSPI_XFERLEVEL register the AEL and AFL levels and especially the MCSPI_XFERLEVEL[31-16] WCNT bit field to define the number of MCSPI words to be transferred using the FIFO before enabling the channel.

This counter lets the controller stop the transfer correctly after a defined number of MCSPI word transfers. If WNCT is set to 0x0000, the counter is not used and the user must stop the transfer manually by disabling the channel; in this case, the user does not know how many MCSPI transfers have been done. For received words, software must poll the MCSPI_CHSTAT_i[5] RXFFE bit and read the MCSPI_RX_0/1/2/3 receive register to empty the FIFO buffer.

When the end-of-word count interrupt is generated (the MCSPI_IRQSTATUS[17] EOW bit is set), the user can disable the channel and poll the MCSPI_CHSTAT_0/1/2/3[5] RXFFE bit to know the last MCSPI words in the FIFO buffer and read them.

No new request is asserted as long as the system has not performed the correct number of write accesses.

12.1.5.3.6.4 Multiple MCSPI Word Access

The processor has the ability to perform multiple MCSPI word access to the receive or transmit registers within a single 32-bit interface access by setting the MCSPI_MODULCTRL[7] MOA to 1 under specific conditions:

- The channel selected has the FIFO enable.
- Only FIFO sense enabled support the kind of access.
- MCSPI_MODULCTRL[7] MOA is set to 1.
- Only 32-bit interface access and data width can be performed to receive or transmit registers, for other kind of access the processor must de-assert MCSPI_MODULCTRL[7] MOA bit.
- The level MCSPI_XFERLEVEL[7-0] AEL and MCSPI_XFERLEVEL[15-8] AFL must be 32-bit aligned, it means that AEL[0] = AEL[1] = 1 or AFL[0] = AFL[1] = 1.
- If MCSPI_XFERLEVEL[31-16] WCNT is used it must be configured according to MCSPI word length.
- The word length of MCSPI words allows to perform multiple MCSPI access, that means that MCSPI_CHCONF_0/1/2/3[11-7] WL is <16.

The number of MCSPI word access depends on MCSPI word length:

- 3 ≤ WL ≤ 7, MCSPI word length smaller or equal to byte length, 4 MCSPI words accessed per 32-bit interface read/write. If word count is used (MCSPI_XFERLEVEL[31-16] WCNT), set the bit field to WCNT[0] = WCNT[1] = 0.

Copyright © 2024 Texas Instruments Incorporated

- 8 ≤ WL ≤ 15, MCSPI word length greater than byte or equal to 16-bit length, 2 MCSPI words accessed per 32-bit interface read/write. If word count is used (MCSPI_XFERLEVEL[31-16] WCNT]), set the bit field to WCNT[0] = 0.
- 16 ≤ WL Multiple MCSPI word access is not applicable.

12.1.5.3.6.5 First MCSPI Word Delay

Figure 12-40 shows the MCSPI controller ability to delay the first MCSPI word transfer to give time for system to complete some parallel processes or fill the FIFO in order to improve transfer bandwidth. This delay is applied only on first MCSPI word after MCSPI channel enabled and first write in transmit register. It is based on output clock frequency.

This option is meaningful in controller mode and single channel mode asserted through MCSPI_MODULCTRL[0] SINGLE.

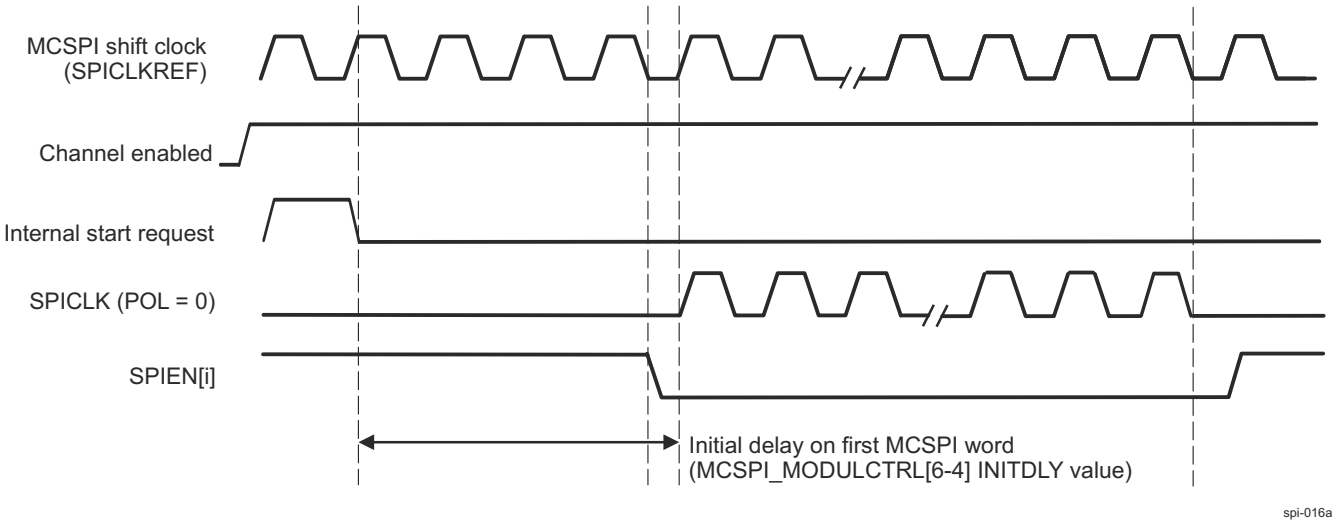


Figure 12-40. Controller Single Channel Initial Delay

Few delay values are available: No delay, 4/8/16/32 MCSPI cycles.

Its accuracy is half cycle in clock bypass mode and depends on clock polarity and phase.

12.1.5.3.7 MCSPI Interrupts

Each channel can issue interrupt events.

Each interrupt event has status bits in the MCSPI_IRQSTATUS register (RXx_FULL, TXx_UNDERFLOW, TXx_EMPTY, etc.) (where x = 0, 3) that indicate whether service is required. Each status bit has an interrupt enable bit (a mask) in the MCSPI_IRQENABLE register (RXx_FULL_ENABLE, TXx_UNDERFLOW_ENABLE, TXx_EMPTY_ENABLE, etc.).

When an interrupt occurs and a mask is later applied on it, the interrupt line is not asserted again, even if the interrupt source is not serviced.

The MCSPI supports interrupt-driven and polling operations.

12.1.5.3.7.1 Interrupt Events in Controller Mode

In controller mode, the interrupt events related to the state of the MCSPI_TX_0/1/2/3 register are TXx_EMPTY and TXx_UNDERFLOW. The interrupt event related to the state of the MCSPI_RX_0/1/2/3 register is RXx_FULL.

12.1.5.3.7.1.1 TXx_EMPTY

The TXx_EMPTY event is activated when a channel is enabled and its MCSPI_TX_0/1/2/3 register is empty (transient event). Enabling a channel automatically triggers this event, except in controller receive-only mode (see Section 12.1.5.3.3.4, *Controller Receive-Only Mode*). When the FIFO buffer is enabled (the

Copyright © 2024 Texas Instruments Incorporated

MCSPI_CHCONF_0/1/2/3[27] FFEW bit is set to 1), the MCSPI_IRQSTATUS TXx_EMPTY bit is set as soon as there is enough space in the buffer to write a number of bytes defined by the MCSPI_XFERLEVEL[5-0] AEL bit field.

The MCSPI_TX_0/1/2/3 register must be loaded with data to remove the source of the interrupt; the MCSPI_IRQSTATUS TXx_EMPTY interrupt status bit must be cleared for interrupt line deassertion (if the event is enabled as the interrupt source).

When FIFO is enabled, no new TXx_EMPTY event is asserted as long as the processor has not performed the number of writes into the MCSPI_TX_0/1/2/3 register defined by the MCSPI_XFERLEVEL[5-0] AEL bit field. The processor must perform the correct number of writes.

12.1.5.3.7.1.2 TXx_UNDERFLOW

The event TXx_UNDERFLOW is activated when the channel is enabled and if the MCSPI_TX_0/1/2/3 register or the FIFO is empty (not updated with new data) when an external controller device starts a data transfer with the MCSPI (transmit and receive).

The TXx_UNDERFLOW is a harmless warning in controller mode.

To avoid having a TXx_UNDERFLOW event at the beginning of a transmission, the TXx_UNDERFLOW event is not activated when no data has been loaded into the MCSPI_TX_0/1/2/3 register, because the channel is enabled. To avoid having a TXx_UNDERFLOW event, the MCSPI_TX_0/1/2/3 register must seldom be loaded.

The MCSPI_IRQSTATUS TXx_UNDERFLOW interrupt status bit must be cleared for interrupt line deassertion (if the event is enabled as the interrupt source).

12.1.5.3.7.1.3 RXx_ FULL

The RXx_FULL event is activated when a channel is enabled and the MCSPI_RX_0/1/2/3 register becomes filled (transient event). When the FIFO buffer is enabled (the MCSPI_CHCONF_0/1/2/3[28] FFER bit is set to 1), RXx_ FULL is asserted as soon as the number of bytes held in the FIFO to be read reaches the MCSPI_XFERLEVEL[13-8] AFL threshold.

The MCSPI_RX_0/1/2/3 register must be read to remove the source of the interrupt; the MCSPI_IRQSTATUS RXx_FULL interrupt status bit must be cleared for interrupt line deassertion (if the event is enabled as the interrupt source).

When FIFO is enabled, no new RXx_FULL event is asserted as long as the processor has not performed AFL + 1 reads into MCSPI_RX_0/1/2/3. The processor must perform the correct number of reads.

12.1.5.3.7.1.4 End Of Word Count

The MCSPI_IRQSTATUS[17] EOW event (end of word count) is activated when the channel is enabled and configured to use the built-in FIFO. This interrupt is raised when the controller performs the number of transfers defined in the MCSPI_XFERLEVEL[31-16] WCNT bit field. If WCNT is set to 0x0000, the counter is not enabled and this interrupt is not generated.

The end of word count interrupt also indicates that the MCSPI transfer is halted on the channel using the FIFO buffer as soon as MCSPI_XFERLEVEL[31-16] WCNT is not reloaded and the channel is not re-enabled.

The MCSPI_IRQSTATUS[17] EOW interrupt status bit must be cleared for interrupt line deassertion (if the event is enabled as the interrupt source).

12.1.5.3.7.2 Interrupt Events in Peripheral Mode

In peripheral mode, the interrupt events related to the state of the MCSPI_TX_0/1/2/3 register are TX0_EMPTY and TX0_UNDERFLOW. The interrupt events related to the state of the MCSPI_RX_0/1/2/3 are RX0_FULL and RX0_OVERFLOW (channels 1, 2, and 3 do not have a receiver overflow status bit). See the MCSPI_IRQSTATUS register.

12.1.5.3.7.2.1 TXx_EMPTY

The TXx_EMPTY event is activated when a channel is enabled and its MCSPI_TX_0/1/2/3 register is empty. Enabling the channel automatically raises this event. If the FIFO buffer is enabled (the

Copyright © 2024 Texas Instruments Incorporated

MCSPI_CHCONF_0/1/2/3[27] FFEW bit is set to 1), the TXx_EMPTY event is asserted as soon as there is enough space in buffer to write a number of bytes defined by the MCSPI_XFERLEVEL[5-0] AEL bit field.

The MCSPI_TX_0/1/2/3 register must be loaded with data to remove the source of the interrupt; the MCSPI_IRQSTATUS TXx_EMPTY interrupt status bit must be cleared for interrupt line deassertion (if the event is enabled as the interrupt source).

When FIFO is enabled, no new TXx_EMPTY event is asserted as long as the processor has not performed the number of writes into the MCSPI_TX_0/1/2/3 register defined by MCSPI_XFERLEVEL[5-0] AEL bit field. The processor must perform the correct number of writes.

12.1.5.3.7.2.2 TXx_UNDERFLOW

The TXx_UNDERFLOW event is activated when a channel is enabled and if the MCSPI_TX_0/1/2/3 register is empty (not updated with new data) when an external controller device starts a data transfer with the MCSPI (transmit and receive).

When FIFO is enabled, the data emitted while the underflow event is raised is not the last data written in the FIFO but the next data in the FIFO (an old transmitted value or a dummy data in the FIFO has been reset).

TXx_UNDERFLOW indicates an error (data loss) in peripheral mode.

To avoid having a TXx_UNDERFLOW event at the beginning of a transmission, the TXx_UNDERFLOW event is not activated when no data has been loaded into the MCSPI_TX_0/1/2/3 register because the channel is enabled.

The MCSPI_IRQSTATUS TXx_UNDERFLOW interrupt status bit must be cleared for interrupt line deassertion (if the event is enabled as the interrupt source).

12.1.5.3.7.2.3 RXx_FULL

The RXx_FULL event is activated when a channel is enabled and the MCSPI_RX_0/1/2/3 register is being filled (transient event). When the FIFO buffer is enabled (the MCSPI_CHCONF_0/1/2/3[28] FFER bit is set to 1), RXx_FULL is asserted as soon as the number of bytes held in the buffer to read defined by the MCSPI_XFERLEVEL[13-8] AFL bit field.

The MCSPI_RX_0/1/2/3 register must be read to remove the source of the interrupt; the MCSPI_IRQSTATUS RXx_FULL interrupt status bit must be cleared for interrupt line deassertion (if the event is enabled as the interrupt source).

When FIFO is enabled, no new RXx_FULL event is asserted as long as the processor has not performed AFL + 1 reads into MCSPI_RX_0/1/2/3. The processor must perform the correct number of reads.

12.1.5.3.7.2.4 RX0_OVERFLOW

The RX0_OVERFLOW event is activated in peripheral mode in transmit-and-receive mode or receive-only mode when a channel is enabled and the MCSPI_RX_0/1/2/3 register or FIFO is full when a new MCSPI word is received. The MCSPI_RX_0/1/2/3 register is always overwritten with the new MCSPI word. If the FIFO is enabled, data within the FIFO are overwritten; it must be considered as corrupted. The RX0_OVERFLOW event should not appear in peripheral mode using the FIFO.

The RX0_OVERFLOW event indicates an error (data loss) in peripheral mode.

The MCSPI_IRQSTATUS[3] RX0_OVERFLOW interrupt status bit must be cleared for interrupt line deassertion (if the event is enabled as the interrupt source).

12.1.5.3.7.2.5 End Of Word Count

The MCSPI_IRQSTATUS[17] EOW event (end of word count) is activated when the channel is enabled and configured to use the built-in FIFO. This interrupt is raised when the controller performs the number of transfers defined in the MCSPI_XFERLEVEL[31-16] WCNT bit field. If WCNT is set to 0x0000, the counter is not enabled and this interrupt is not generated.

The end of word count interrupt also indicates that the MCSPI transfer is halted on the channel using the FIFO buffer as soon as WCNT is not reloaded and the channel is not re-enabled.

Copyright © 2024 Texas Instruments Incorporated

The MCSPI_IRQSTATUS[17] EOW interrupt status bit must be cleared for interrupt line deassertion (if the event is enabled as the interrupt source).

12.1.5.3.7.3 Interrupt-Driven Operation

An interrupt enable bit in the MCSPI_IRQENABLE register can be set to enable each event to generate interrupt requests when the corresponding event occurs. Status bits are automatically set by hardware logic conditions.

When an event occurs (the single interrupt line is asserted), the processor must:

1. Read the MCSPI_IRQSTATUS register to identify which event occurred.
2. Read the MCSPI_RX_0/1/2/3 register that corresponds to the event to remove the source of an RXx_FULL event or write into the MCSPI_TX_0/1/2/3 register that corresponds to the event to remove the source of a TXx_EMPTY event. No action is required to remove the source of the TXx_ UNDERFLOW and RX0_OVERFLOW events.
3. Set the corresponding bit of the MCSPI_IRQSTATUS register to 1 to clear an interrupt status and then release the interrupt line.

The interrupt status bit must always be reset after channel enabling and before events are enabled as interrupt sources.

12.1.5.3.7.4 Polling

When the interrupt capability of an event is disabled in the MCSPI_IRQENABLE register, the interrupt line is not asserted, but the status bits in the MCSPI_IRQSTATUS register can be polled by software to detect when the corresponding event occurs.

Once the expected event occurs:

- RXx_FULL: To remove the source of the event, the processor must read the corresponding MCSPI_RX_0/1/2/3 register.
- TXx_EMPTY: To remove the source of the event, the processor must write into the corresponding MCSPI_TX_0/1/2/3 register.
- TXx_UNDERFLOW and RX0_OVERFLOW: No action is required to remove the source of the event.

To clear an interrupt, set the corresponding status bit of the MCSPI_IRQSTATUS register to 1. This does not affect the interrupt line state.

12.1.5.3.8 MCSPI DMA Requests

Each MCSPI channel, if enabled, can issue DMA requests. There are two DMA request lines per MCSPI channel (one for read and one for write).

The DMA read request line is asserted when the MCSPI channel is enabled and new data is available in the receive register of the MCSPI channel. A DMA read request can be individually masked with the MCSPI_CHCONF_0/1/2/3[15] DMAR bit. The DMA read request line is de-asserted when reading of the MCSPI_RX_0/1/2/3 register of the MCSPI channel completes.

The DMA write request line is asserted when the MCSPI channel is enabled and the MCSPI_TX_0/1/2/3 register of the MCSPI channel is empty. A DMA write request can be individually masked with the MCSPI_CHCONF_0/1/2/3[14] DMAW bit. The DMA write request line is de-asserted when loading of the MCSPI_TX_0/1/2/3 register of the channel completes.

12.1.5.3.9 MCSPI Power Saving Management

Power consumption can be optimized by switching off internal clocks (interface and functional clock) when there is no activity.

12.1.5.3.9.1 Normal Mode

In normal mode, internal MCSPI module clocks are automatically switched off (autogated) when there is no activity in peripheral or controller mode.

Autogating of the module interface clock and functional clock occurs when the following conditions are met:

- The MCSPI_SYSCONFIG[0] AUTOIDLE bit is set.

Copyright © 2024 Texas Instruments Incorporated

- In controller mode, there is no data to transmit or receive in all channels.
- In peripheral mode, the MCSPI is not selected by the external controller and there are no register accesses.

Autogating of the module interface clock and functional clock stops when the following conditions are met:

- In controller mode, an internal access occurs.
- In peripheral mode, an internal access occurs or the MCSPI is selected by the external controller.

12.1.5.3.9.2 Idle Mode

Note

Some of the MCSPI features described in this section may not be supported on this family of devices. For more information, see *MCSPI Not Supported Features*.

At the power management level, when all conditions to shut off the MCSPI_FCLK or MCSPI_ICLK clocks are met, the corresponding LPSC asserts a clock stop request to the MCSPI. Although this procedure is completely hardware-oriented and out of software control, the method in which the MCSPI module acknowledges the clock stop request can be configured through the MCSPI_SYSCONFIG[4-3] SIDLEMODE bit field.

The settings of the SIDLEMODE bit field and the related acknowledgement modes are:

- Force-idle mode (the MCSPI_SYSCONFIG[4-3] SIDLEMODE bit field is set to 0x0): The MCSPI module acknowledges unconditionally the clock stop request, regardless of its internal operations. This mode must be used carefully in this case because it does not prevent the loss of data when the clock is switched off.
- No-idle mode (the SIDLEMODE bit field is set to 0x1): The MCSPI never acknowledges the clock stop request and is safe from a module point of view because it ensures that the clocks remain active. However, it is not efficient to save power because it does not allow shut off of MCSPI_FCLK and MCSPI_ICLK.
- Smart-idle mode (the SIDLEMODE bit field is set to 0x2): The MCSPI acknowledges the clock stop request, depending on its internal activity. MCSPI acknowledges the shut off of MCSPI_FCLK and MCSPI_ICLK only when all pending transactions, IRQs, or DMA requests are treated. This is the best approach for efficient system power management.

When configured in smart-idle mode, the MCSPI also offers an additional feature to control gating of MCSPI_FCLK or MCSPI_ICLK. The MCSPI_SYSCONFIG[9-8] CLOCKACTIVITY bit field determines which clock shuts down (MCSPI_FCLK, MCSPI_ICLK, neither clock, or both clocks).

The setting of the CLOCKACTIVITY bit field is used internally to the MCSPI to determine on which part of the module the conditions to acknowledge the clock stop request are tested. For example, if MCSPI_FCLK is not shut off on clock stop request, the MCSPI considers only MCSPI_ICLK and the associated pending activities before acknowledging the request.

Some MCSPI features are associated with MCSPI_ICLK and others with MCSPI_FCLK. Using the CLOCKACTIVITY bit field with the smart-idle mode ensures that the features associated with the clock that remains active are always enabled, even if MCSPI acknowledges the clock stop request.

12.1.5.3.9.2.1 Force-Idle Mode

Force-idle mode is enabled and exited as follows:

- Force-idle mode is enabled when the MCSPI_SYSCONFIG[4-3] SIDLEMODE bit field is set to 0x0.
  - In force-idle mode, the MCSPI responds unconditionally to the clock stop request by de-asserting unconditionally the interrupt and DMA request lines, if asserted.
  - The transition from normal mode to idle mode does not affect the interrupt event bits of the MCSPI_IRQSTATUS register.
  - In force-idle mode, because the module must be disabled, the interrupt and DMA request lines are likely de-asserted. The interface clock and MCSPI clock provided to the MCSPI can be switched off.
  - A clock stop request during an MCSPI data transfer can lead to an unexpected and unpredictable result. Software must avoid such a request.

Copyright © 2024 Texas Instruments Incorporated

12.1.5.4 MCSPI Programming Guide

This section describes the low-level hardware programming sequences for the configuration and use of the MCSPI module.

12.1.5.4.1 MCSPI Operational Mode Configuration

12.1.5.4.1.1 MCSPI Operational Modes

The selection of the working mode is done with the MCSPI_CHCONF_0/1/2/3 register.

Table 12-40. MCSPI Receive Mode Initialization

Step	Register/Bit Field/Programming Model	Value
Set receive mode for the channel.	MCSPI_CHCONF_0/1/2/3[13-12] TRM	0x1
Configure SPI clock polarity/phase, clock divider, word length, and others for the channel.	MCSPI_CHCONF_0/1/2/3	0x-
Reset the status bits.	MCSPI_IRQSTATUS	0x0

Table 12-41. MCSPI Transmit Mode Initialization

Step	Register/Bit Field/Programming Model	Value
Set transmit mode for the channel.	MCSPI_CHCONF_0/1/2/3[13-12] TRM	0x2
Configure SPI clock polarity/phase, clock divider, word length, and others for the channel.	MCSPI_CHCONF_0/1/2/3	0x-
Reset the status bits.	MCSPI_IRQSTATUS	0x0

Table 12-42. MCSPI Transmit-and-Receive Mode Initialization

Step	Register/Bit Field/Programming Model	Value
Set transmit and receive mode for the channel.	MCSPI_CHCONF_0/1/2/3[13-12] TRM	0x0
Configure SPI clock polarity/phase, clock divider, word length, and others for the channel.	MCSPI_CHCONF_0/1/2/3	0x-
Reset the status bits.	MCSPI_IRQSTATUS	0x0

12.1.5.4.1.1.1 Common Transfer Sequence

MCSPI module allows the transfer of one or several words, according to different modes:

- CONTROLLER Normal, CONTROLLER Turbo, PERIPHERAL
- TRANSMIT–RECEIVE, TRANSMIT-ONLY, RECEIVE-ONLY
- Write and Read requests: Interrupts, DMA
- SPIEN[i] lines assertion/deassertion: automatic, manual

For all these sequences, the host process contains the main process and the interrupt routines.

The interrupt routines are called on the interrupt signals or by an internal call if the module is used in polling mode.

Table 12-43 represents the main sequence which is common to all transfers.

In multi-channel controller mode, the sequences of different channels can be run simultaneously.

Table 12-43. Common Transfer Sequence (Main Process)

Step	Register/Bit Field/Programming Model	Value
Write MCSPI_IRQSTATUS to reset channel status bits	MCSPI_IRQSTATUS[channel i bits]	0b1111
Write MCSPI_IRQENABLE to enable interrupts	MCSPI_IRQENABLE	0x-
Write MCSPI_CHCONF_0/1/2/3 to configure the channel	MCSPI_CHCONF_0/1/2/3	0x-
Start the channel	MCSPI_CHCTRL_0/1/2/3[0] EN	1
Wait for the first write request (TX empty or DMA write)		
Write the transmitter register with data	MCSPI_TX_0/1/2/3	0x-
Wait for the host event for end of transfer		

Copyright © 2024 Texas Instruments Incorporated

Table 12-43. Common Transfer Sequence (Main Process) (continued)

Step	Register/Bit Field/Programming Model	Value
Stop the channel	MCSPI_CHCTRL_0/1/2/3[0] EN	0

12.1.5.4.1.1.2 End of Transfer Sequences

The end of transfer depends on the transfer mode. Table 12-44 summarizes the type of end of transfer per transfer mode and gives a reference to the appropriate section for details.

Table 12-44. End of Transfer Sequences

		TRANSMIT-AND-RECEIVE		TRANSMIT-ONLY		RECEIVE-ONLY	
		INTERRUPT	DMA	INTERRUPT	DMA	INTERRUPT	DMA
CONTROLLER Normal	End of transfer sequence	See Section 12.1.5.4.1.1.3		See Section 12.1.5.4.1.1.4.1	See Section 12.1.5.4.1.1.4.2	See Section 12.1.5.4.1.1.5.1	See Section 12.1.5.4.1.1.5.2
	Minimum number of word	1	1	1	1	1	2
	DMA transfer size		N		N		N-1
CONTROLLER Turbo	End of transfer sequence	See Section 12.1.5.4.1.1.3		See Section 12.1.5.4.1.1.4.1	See Section 12.1.5.4.1.1.4.2	See Section 12.1.5.4.1.1.6.1	See Section 12.1.5.4.1.1.6.2
	Minimum number of word	1	1	1	1	2	3
	DMA transfer size		N		N		N-2
PERIPHERAL	End of transfer sequence	See Section 12.1.5.4.1.1.3		See Section 12.1.5.4.1.1.4.1	See Section 12.1.5.4.1.1.4.2	See Section 12.1.5.4.1.1.7	
	Minimum number of word	1	1	1	1	1	1
	DMA transfer size		N		N	N	N

The transfer to execute has a size of N words.

The different sequences can be merged in one process to manage transfers of several types. The end of transfer sequences are described from the start of the channel.

In these sequences, some soft variables are used:

- write_count = 0
- read_count = 0
- channel_enable = FALSE
- last_transfer = FALSE
- last_request = FALSE

They are initialized before starting the channel.

12.1.5.4.1.1.3 Transmit-and-Receive (Controller and Peripheral)

If the requests are configured in DMA, write_count and read_count are assigned with ‘N’ when the DMA handlers have completed their ‘N’ CBASS0 accesses.

Table 12-45. Transmit-and-Receive (Controller and Peripheral) (Main Process)

Step	Register/Bit Field/Programming Model	Value
Start the channel	MCSPI_CHCTRL_0/1/2/3[0] EN	1
Wait for write_count = N **AND** read_count = N		
Stop the channel	MCSPI_CHCTRL_0/1/2/3[0] EN	0

Table 12-46. Transmit-and-Receive (Controller and Peripheral) (Interrupt Routine)

Step	Register/Bit Field/Programming Model	Value
Read MCSPI_IRQSTATUS	MCSPI_IRQSTATUS	0x-
Write MCSPI_IRQSTATUS to reset channel status bits	MCSPI_IRQSTATUS[channel i bits]	0b1111

Copyright © 2024 Texas Instruments Incorporated

Table 12-46. Transmit-and-Receive (Controller and Peripheral) (Interrupt Routine) (continued)

Step	Register/Bit Field/Programming Model	Value
IF: TXx_EMPTY		
Write the transmitter register with data	MCSPI_TX_0/1/2/3	0x-
Increment write_count +1		
IF: RXx_FULL		
Read the receiver register	MCSPI_RX_0/1/2/3	
Increment read_count +1		
ENDIF		

12.1.5.4.1.1.4 Transmit-Only (Controller and Peripheral)

12.1.5.4.1.1.4.1 Based on Interrupt Requests

Table 12-47. Transmit-Only With Interrupts (Controller and Peripheral) (Main Process)

Step	Register/Bit Field/Programming Model	Value
Start the channel	MCSPI_CHCTRL_0/1/2/3[0] EN	1
Wait until last transfer = TRUE		
Wait for end of transfer	MCSPI_CHSTAT_0/1/2/3[2] EOT	=1
Stop the channel	MCSPI_CHCTRL_0/1/2/3[0] EN	0

Table 12-48. Transmit-Only With Interrupts (Controller and Peripheral) (Interrupt Routine)

Step	Register/Bit Field/Programming Model	Value
Read MCSPI_IRQSTATUS	MCSPI_IRQSTATUS	0x-
Write MCSPI_IRQSTATUS to reset channel status bits	MCSPI_IRQSTATUS[channel i bits]	0b1111
IF: TXx_EMPTY **AND** write_count < N		
Write the transmitter register with data	MCSPI_TX_0/1/2/3	0x-
Increment write_count +1		
ELSEIF: write_count ≥ N		
last_transfer = TRUE		
ENDIF		

12.1.5.4.1.1.4.2 Based on DMA Write Requests

When the DMA handler has completed its ‘N’ CBASS0 accesses, write_count is assigned with ‘N’.

Table 12-49. Transmit-Only With DMA (Controller and Peripheral) (Main Process)

Step	Register/Bit Field/Programming Model	Value
Start the channel	MCSPI_CHCTRL_0/1/2/3[0] EN	1
Wait until write_count = N		
Disable DMA write request	MCSPI_CHCONF_0/1/2/3[14] DMAW	0
Wait until last_transfer = TRUE		
Wait for end of transfer	MCSPI_CHSTAT_0/1/2/3[2] EOT	=1
Stop the channel	MCSPI_CHCTRL_0/1/2/3[0] EN	0

Table 12-50. Transmit-Only With DMA (Controller and Peripheral) (Interrupt Routine)

Step	Register/Bit Field/Programming Model	Value
Read MCSPI_IRQSTATUS	MCSPI_IRQSTATUS	0x-
Write MCSPI_IRQSTATUS to reset channel status bits	MCSPI_IRQSTATUS[channel i bits]	0b1111
IF: TXx_EMPTY **AND** write_count = N		
last_transfer = TRUE		
ENDIF		

Copyright © 2024 Texas Instruments Incorporated

12.1.5.4.1.1.5 Controller Normal Receive-Only

12.1.5.4.1.1.5.1 Based on Interrupt Requests

Table 12-51. Receive-Only With Interrupt (Controller Normal) (Main Process)

Step	Register/Bit Field/Programming Model	Value
Start the channel	MCSPI_CHCTRL_0/1/2/3[0] EN	1
Wait until last_request = TRUE		
Stop the channel	MCSPI_CHCTRL_0/1/2/3[0] EN	0
Read the receiver register	MCSPI_RX_0/1/2/3	0x-
Increment read_count +1		

Table 12-52. Receive-Only With Interrupt (Controller Normal) (Interrupt Routine)

Step	Register/Bit Field/Programming Model	Value
Read MCSPI_IRQSTATUS	MCSPI_IRQSTATUS	0x-
Write MCSPI_IRQSTATUS to reset channel status bits	MCSPI_IRQSTATUS[channel i bits]	0b1111
IF: RXx_FULL **AND** read_count = N - 1		
last_request = TRUE		
ELSEIF: read_count ≠ N - 1		
Read the receiver register	MCSPI_RX_0/1/2/3	0x-
Increment read_count +1		
ENDIF		

12.1.5.4.1.1.5.2 Based on DMA Read Requests

When the DMA handler has completed its ‘N-1’ CBASS0 accesses, read_count is assigned with ‘N-1’.

Table 12-53. Receive-Only With DMA (Controller Normal) (Main Process)

Step	Register/Bit Field/Programming Model	Value
Start the channel	MCSPI_CHCTRL_0/1/2/3[0] EN	1
Wait until read_count = N - 1		
Disable DMA read request	MCSPI_CHCONF_0/1/2/3[15] DMAR	0
Wait until last_transfer = TRUE		
Stop the channel	MCSPI_CHCTRL_0/1/2/3[0] EN	0
Read the receiver register	MCSPI_RX_0/1/2/3	0x-
Increment read_count +1		

Table 12-54. Receive-Only With DMA (Controller Normal) (Interrupt Routine)

Step	Register/Bit Field/Programming Model	Value
Read MCSPI_IRQSTATUS	MCSPI_IRQSTATUS	0x-
Write MCSPI_IRQSTATUS to reset channel status bits	MCSPI_IRQSTATUS[channel i bits]	0b1111
IF: RXx_FULL **AND** read_count = N		
last_transfer = TRUE		
ENDIF		

12.1.5.4.1.1.6 Controller Turbo Receive-Only

12.1.5.4.1.1.6.1 Based on Interrupt Requests

Table 12-55. Receive-Only With Interrupt (Controller Turbo) (Main Process)

Step	Register/Bit Field/Programming Model	Value
Start the channel	MCSPI_CHCTRL_0/1/2/3[0] EN	1
Wait until channel_enable = TRUE		
Wait until last_transfer = TRUE		
Wait for end of transfer	MCSPI_CHSTAT_0/1/2/3[2] EOT	=1

Copyright © 2024 Texas Instruments Incorporated

Table 12-55. Receive-Only With Interrupt (Controller Turbo) (Main Process) (continued)

Step	Register/Bit Field/Programming Model	Value
Stop the channel	MCSPI_CHCTRL_0/1/2/3[0] EN	0
Wait until channel_enable = FALSE		

Table 12-56. Receive-Only With Interrupt (Controller Turbo) (Interrupt Routine)

Step	Register/Bit Field/Programming Model	Value
Read MCSPI_IRQSTATUS	MCSPI_IRQSTATUS	0x-
Write MCSPI_IRQSTATUS to reset channel status bits	MCSPI_IRQSTATUS[channel i bits]	0b1111
IF: RXx_FULL		
IF: read_count = N - 2		
last_transfer = TRUE		
Wait until channel_enable = FALSE		
ENDIF		
IF: read_count < N		
Read the receiver register	MCSPI_RX_0/1/2/3	0x-
Increment read_count +1		
ENDIF		
ENDIF		

12.1.5.4.1.1.6.2 Based on DMA Read Requests

When the DMA handler has completed its ‘N-2’ CBASS0 accesses read_count is assigned with ‘N-2‘.

Table 12-57. Receive-Only With DMA (Controller Turbo) (Main Process)

Step	Register/Bit Field/Programming Model	Value
Start the channel	MCSPI_CHCTRL_0/1/2/3[0] EN	1
Wait until channel_enable = TRUE		
Wait until read_count = TRUE		
Disable DMA read request	MCSPI_CHCONF_0/1/2/3[15] DMAR	0
Wait until last_transfer = TRUE		
Wait for end of transfer	MCSPI_CHSTAT_0/1/2/3[2] EOT	=1
Stop the channel	MCSPI_CHCTRL_0/1/2/3[0] EN	0
Wait until channel_enable = FALSE		

Table 12-58. Receive-Only With DMA (Controller Turbo) (Interrupt Routine)

Step	Register/Bit Field/Programming Model	Value
Read MCSPI_IRQSTATUS	MCSPI_IRQSTATUS	0x-
Write MCSPI_IRQSTATUS to reset channel status bits	MCSPI_IRQSTATUS[channel i bits]	0b1111
IF: RXx_FULL		
IF: read_count = N - 2		
last_transfer = TRUE		
Wait until channel_enable = FALSE		
ENDIF		
IF: read_count < N		
Read the receiver register	MCSPI_RX_0/1/2/3	0x-
Increment read_count +1		
ENDIF		
ENDIF		

Copyright © 2024 Texas Instruments Incorporated

12.1.5.4.1.1.7 Peripheral Receive-Only

If the requests are configured in DMA, read_count is assigned with 'N' when the DMA handler has completed its 'N' CBASS0 accesses.

Table 12-59. Receive-Only (Peripheral) (Main Process)

Step	Register/Bit Field/Programming Model	Value
Start the channel	MCSPI_CHCTRL_0/1/2/3[0] EN	1
Wait until read_count = N		
Stop the channel	MCSPI_CHCTRL_0/1/2/3[0] EN	0

Table 12-60. Receive-Only (Peripheral) (Interrupt Routine)

Step	Register/Bit Field/Programming Model	Value
Read MCSPI_IRQSTATUS	MCSPI_IRQSTATUS	0x-
Write MCSPI_IRQSTATUS to reset channel status bits	MCSPI_IRQSTATUS[channel i bits]	0b1111
IF: RXx_FULL		
Read the receiver register	MCSPI_RX_0/1/2/3	0x-
Increment read_count +1		
ENDIF		

12.1.5.4.1.1.8 Transfer Procedures With FIFO

These flows describe the transfer with FIFO.

The MCSPI module allows the transfer of one or several words, according to different modes:

- CONTROLLER Normal, CONTROLLER Turbo, PERIPHERAL
- TRANSMIT–RECEIVE, TRANSMIT-ONLY, RECEIVE-ONLY
- Write and Read requests: IRQ, DMA

For all these flows, the host process contains the main process and the interrupt routine. This routine is called on the IRQ signals or by an internal call if the module is used in polling mode.

For more information, see Section 12.1.5.3.6, *MCSPI FIFO Buffer Management*.

Table 12-61. FIFO Mode Common Sequence (Controller) (Main Process)

Step	Register/Bit Field/Programming Model	Value
Write MCSPI_IRQSTATUS to reset channel status bits	MCSPI_IRQSTATUS	1
Write MCSPI_IRQENABLE to enable interrupts	MCSPI_IRQENABLE	1
Write MCSPI_CHCONF_0/1/2/3 to configure the channel	MCSPI_CHCONF_0/1/2/3	0x-
Write MCSPI_XFERLEVEL	MCSPI_XFERLEVEL	0x-
Start the channel	MCSPI_CHCTRL_0/1/2/3[0] EN	1
IF: Receive only		
Wait for the write request (TX empty or DMA write)		
Write for the transmitter register with data	MCSPI_TX_0/1/2/3	0x-
ENDIF		
Wait for the host event for end of transfer		
Stop the channel	MCSPI_CHCTRL_0/1/2/3[0] EN	0

12.1.5.4.1.1.8.1 Common Transfer Sequence in FIFO Mode

This flow describes the host sequence for a transfer of any type defined in Section 12.1.5.4.1.1.8, *Transfer Procedures With FIFO*.

In multi-channel, only one channel can use the FIFO.

Copyright © 2024 Texas Instruments Incorporated

Before enabling the FIFO for a channel (MCSPI_CHCONF_0/1/2/3[28] FFER and MCSPI_CHCONF_0/1/2/3[27] FFEW bits), the host must check that the FIFO is not enabled for another channel, even if these channels are not used.

In transmit-and-receive mode, the FIFO can be enabled for write or read request only, without FIFO for the other request.

In Peripheral mode, the channel 0 only can be activated. The correct SPIEN line is chosen in MCSPI_CHCONF_0[22-21] SPIENSLV bits.

The MCSPI module can start the transfer only when the first write request has been released by writing the MCSPI_TX_0/1/2/3 register, even in receive-only mode (only one write request occurs in this case).

12.1.5.4.1.1.8.2 End of Transfer Sequences in FIFO Mode

Table 12-62 summarizes the type of end of transfer per transfer mode and gives a reference to the appropriate section for details.

Table 12-62. End of Transfer Sequences in FIFO Mode

Word count	TRANSMIT AND RECEIVE	TRANSMIT-ONLY	RECEIVE-ONLY
Yes	See Figure 12-41	See Figure 12-43	See Figure 12-44
No	See Figure 12-42	See Figure 12-43	See Figure 12-45

The end of transfer sequences are described from the start of the channel.

In these sequences, some soft variables are used:

- write_count = N
- read_count = N
- last_request = FALSE

They are initialized before starting the channel.

12.1.5.4.1.1.8.3 Transmit-and-Receive With Word Count

Figure 12-41 shows the flow of a transfer in transmit-and-receive mode, with word count.

Copyright © 2024 Texas Instruments Incorporated

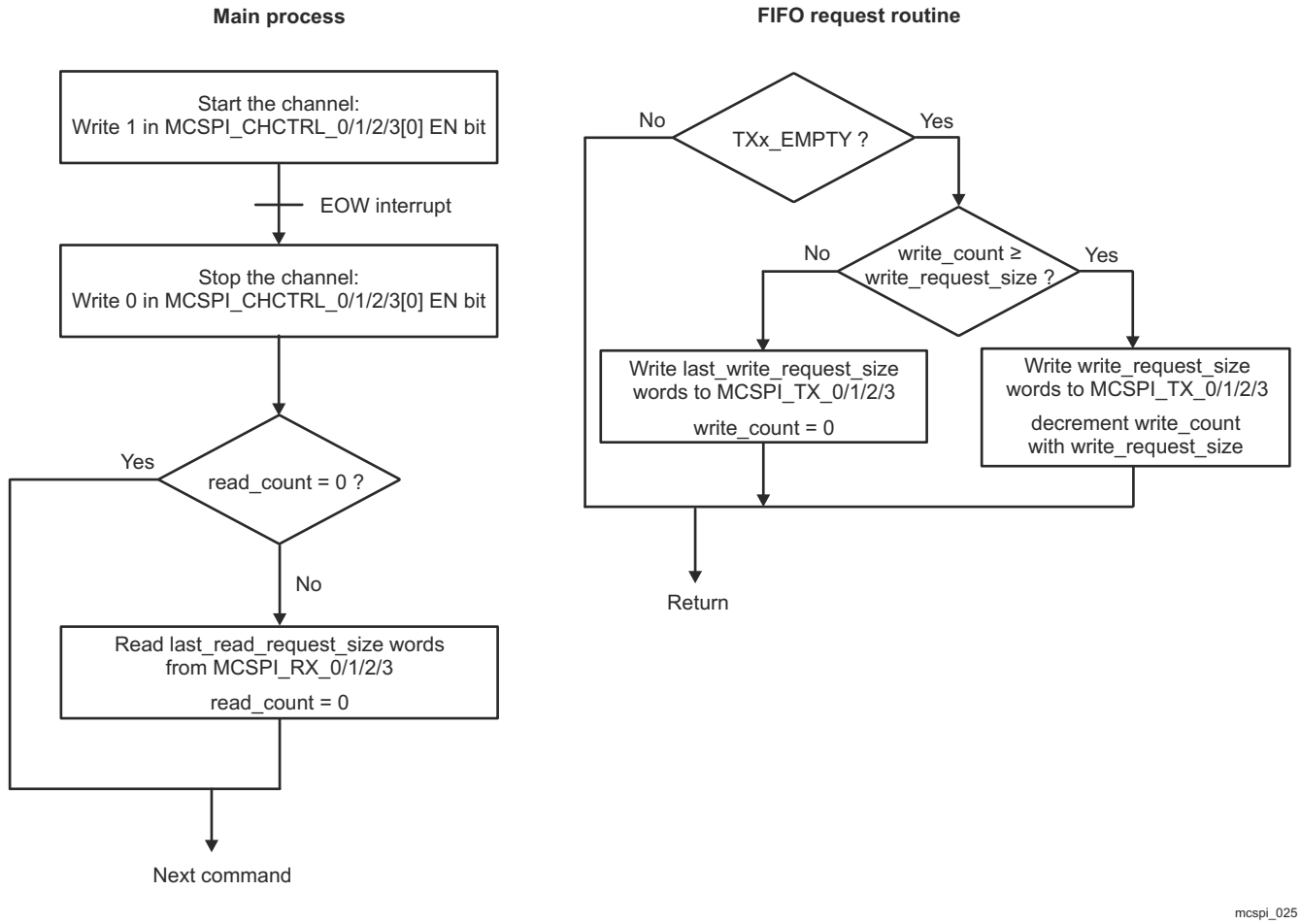


Figure 12-41. FIFO Mode Transmit-and-Receive With Word Count (Controller)

12.1.5.4.1.1.8.4 Transmit-and-Receive Without Word Count

Figure 12-42 shows the flow of a transfer in transmit-and-receive mode, without word count.

Copyright © 2024 Texas Instruments Incorporated

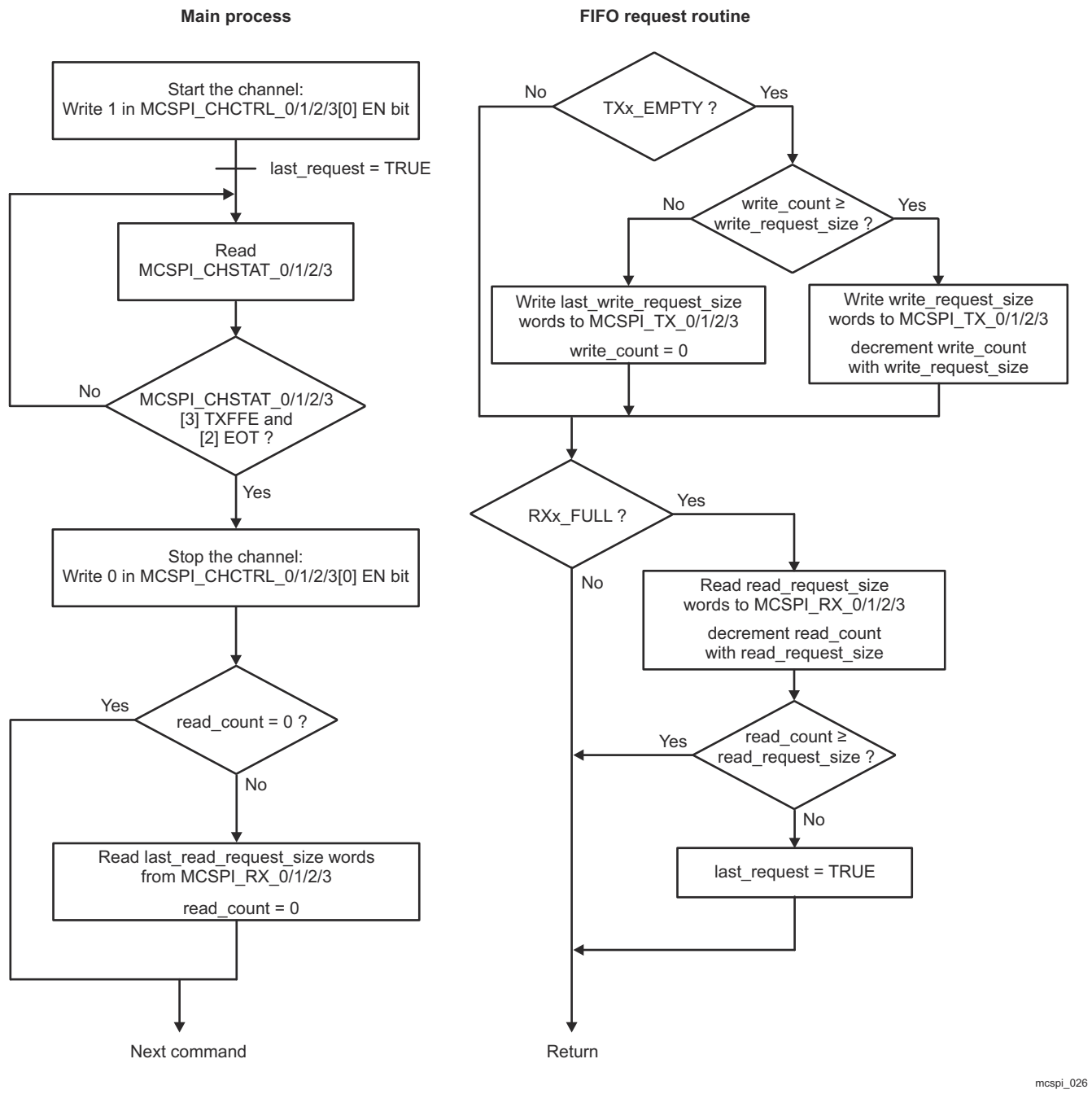


Figure 12-42. FIFO Mode Transmit-and-Receive Without Word Count (Controller)

12.1.5.4.1.1.8.5 Transmit-Only

Figure 12-43 shows the flow of a transfer in transmit-only mode, with or without word count. The difference between word count enabled or not is just on the condition after starting the channel:

- word count enable: wait for EOW interrupt
- word count disable: wait for write_count = 0

Copyright © 2024 Texas Instruments Incorporated

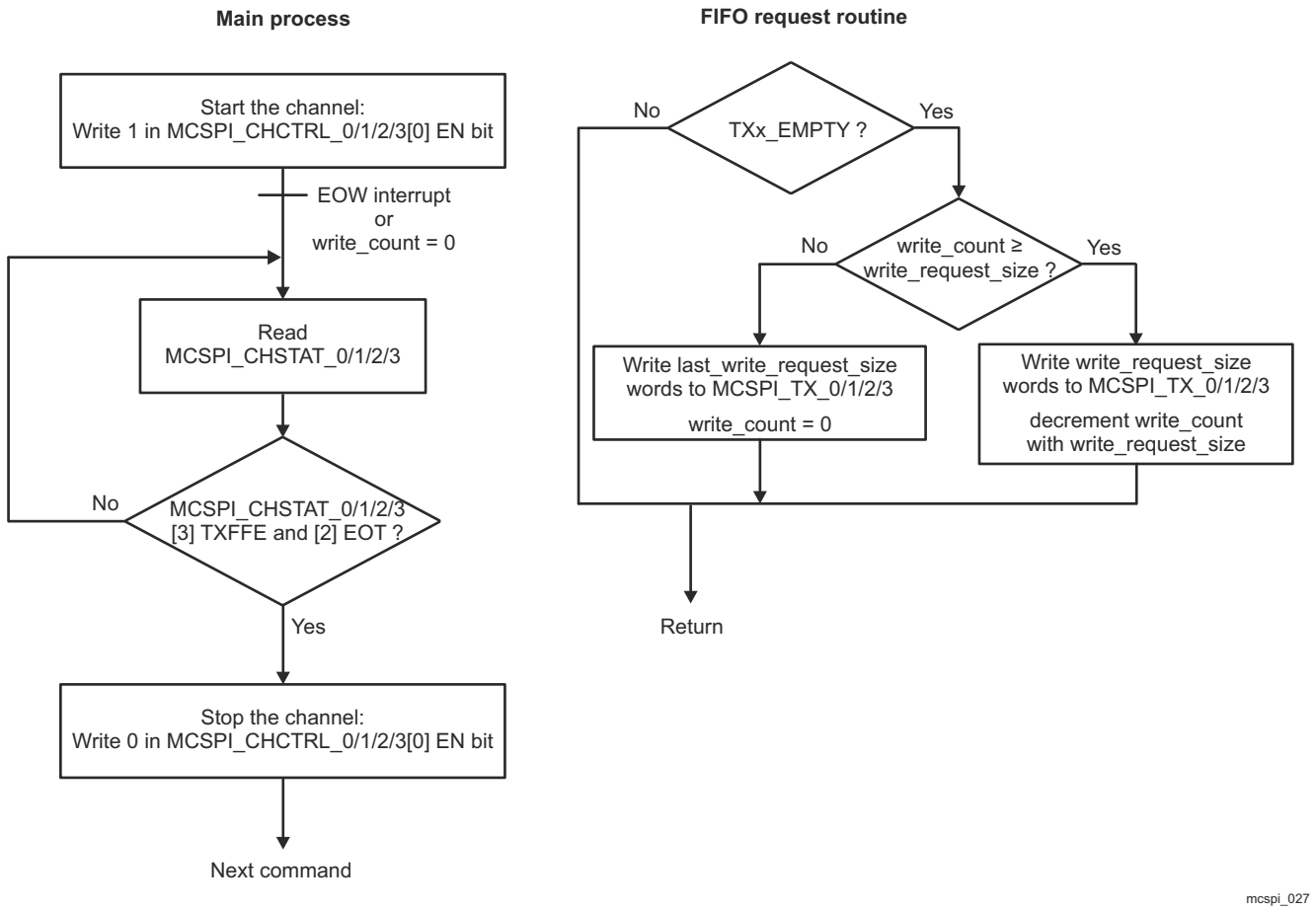


Figure 12-43. FIFO Mode Transmit-Only (Controller)

12.1.5.4.1.1.8.6 Receive-Only With Word Count

Figure 12-44 shows the flow of a transfer in receive-only mode, with word count.

Copyright © 2024 Texas Instruments Incorporated

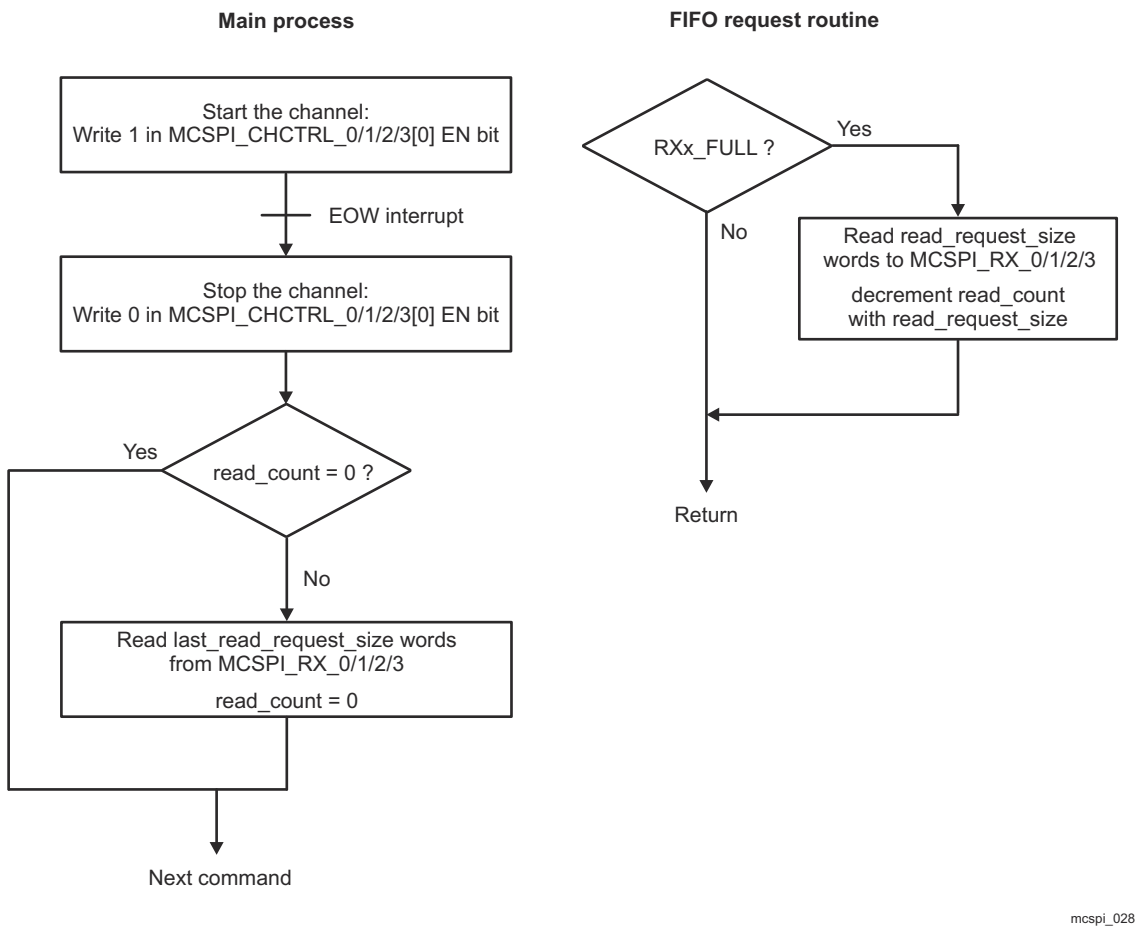


Figure 12-44. FIFO Mode Receive-Only With Word Count (Controller)

12.1.5.4.1.1.8.7 Receive-Only Without Word Count

Figure 12-45 shows the flow of a transfer in receive-only mode, with word count.

Copyright © 2024 Texas Instruments Incorporated

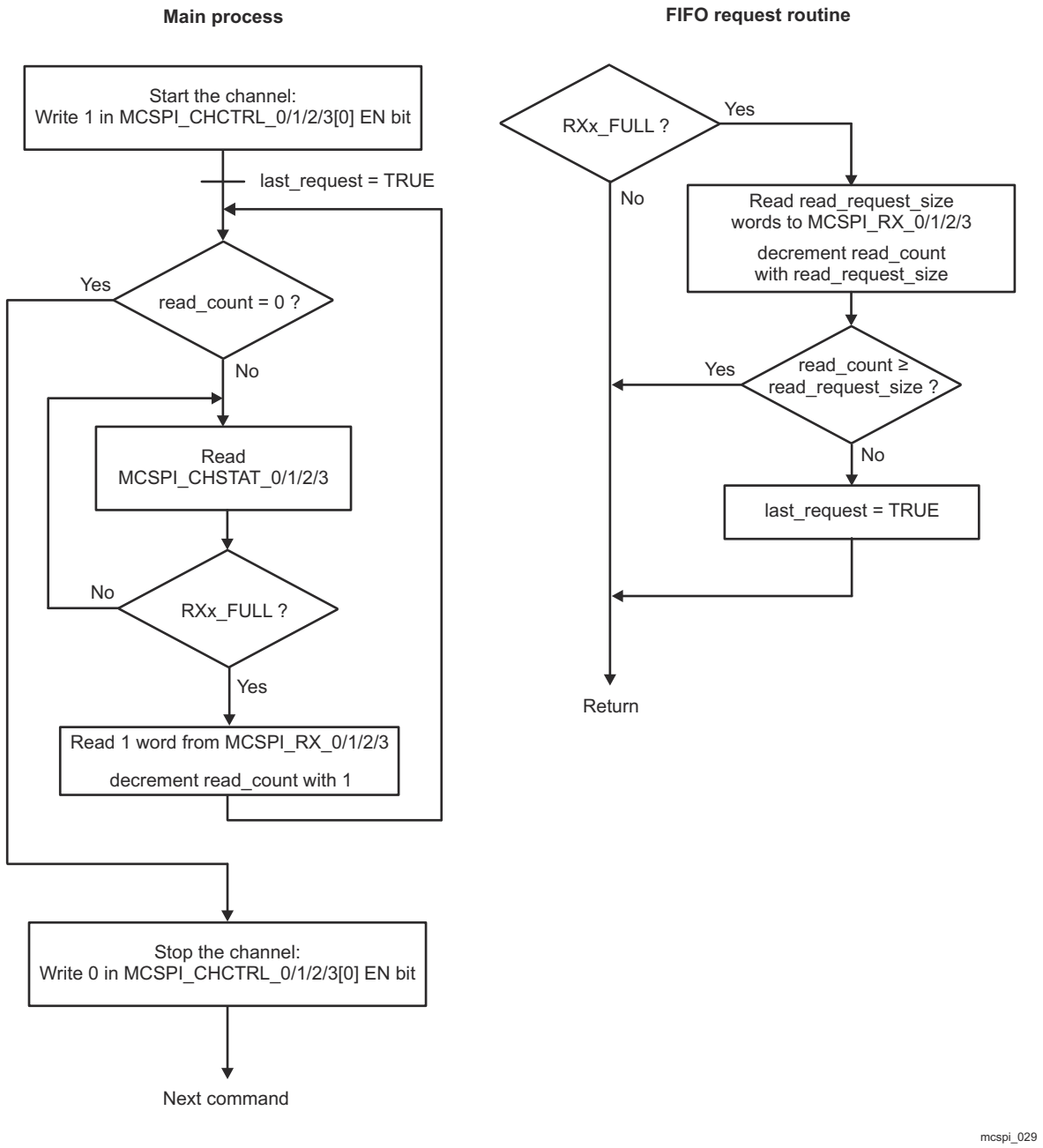


Figure 12-45. FIFO Mode Receive-Only Without Word Count (Controller)

12.1.5.4.1.1.9 Common Transfer Procedures Without FIFO – Polling Method

12.1.5.4.1.1.9.1 Receive-Only Procedure – Polling Method

Table 12-63 lists the receive-only procedure using the polling method.

Table 12-63. Receive-Only Procedure – Polling Method

Step	Register/Bit Field/Programming Model	Value
Configure the channel according to the mode.	See Table 12-40.	
Start the channel.	MCSPI_CHCTRL_0/1/2/3[0] EN	1
Wait for end-of-transfer.	MCSPI_CHSTAT_0/1/2/3[2] EOT	=1
Read the receiver register.	MCSPI_RX_0/1/2/3	0x-
Stop the channel if no more data is expected.	MCSPI_CHCTRL_0/1/2/3[0] EN	0

Copyright © 2024 Texas Instruments Incorporated

12.1.5.4.1.1.9.2 Receive-Only Procedure – Interrupt Method

Table 12-64 lists the receive-only procedure using the interrupt method.

Table 12-64. Receive-Only Procedure – Interrupt Method

Step	Register/Bit Field/Programming Model	Value
Configure the channel according to the mode.	See Table 12-40.	
Start the channel.	MCSPI_CHCTRL_0/1/2/3[0] EN	1
Enable the interrupt for the receiver register.	MCSPI_IRQENABLE[2] RX_FULL_ENABLE	1
Wait for interrupt.		
Read the status register.	MCSPI_IRQSTATUS[2] RX_FULL	1
Disable the interrupt if no more data is expected.	MCSPI_IRQENABLE[2] RX_FULL_ENABLE	0
Stop the channel if no more data is expected.	MCSPI_CHCTRL_0/1/2/3[0] EN	0
Read the receiver register.	MCSPI_RX_0/1/2/3	0x-

12.1.5.4.1.1.9.3 Transmit-Only Procedure – Polling Method

Table 12-65 lists the transmit-only procedure using the polling method.

Table 12-65. Transmit-Only Procedure – Polling Method

Step	Register/Bit Field/Programming Model	Value
Configure the channel according to the mode.	See Table 12-41.	
Start the channel.	MCSPI_CHCTRL_0/1/2/3[0] EN	1
Write the transmitter register with data.	MCSPI_TX_0/1/2/3	0x-
Wait until end of transfer?	MCSPI_CHSTAT_0/1/2/3[2] EOT	=1
Stop the channel.	MCSPI_CHCTRL_0/1/2/3[0] EN	0

12.1.5.4.1.1.9.4 Transmit-and-Receive Procedure – Polling Method

Table 12-66 lists the transmit-and-receive procedure using the polling method.

Table 12-66. Transmit-and-Receive Procedure – Polling Method

Step	Register/Bit Field/Programming Model	Value
Configure the channel according to the mode.	See Table 12-42.	
Start the channel.	MCSPI_CHCTRL_0/1/2/3[0] EN	1
Write the transmitter register with data.	MCSPI_TX_0/1/2/3	0x-
Wait until transmit/receive word?	MCSPI_CHSTAT_0/1/2/3[2] EOT	=1
Stop the channel.	MCSPI_CHCTRL_0/1/2/3[0] EN	0
Read the receiver register.	MCSPI_RX_0/1/2/3	0x-

Copyright © 2024 Texas Instruments Incorporated

12.1.6 Universal Asynchronous Receiver/Transmitter (UART)

This chapter describes the function, operation, and configuration of the Universal Asynchronous Receiver/Transmitter (UART)/RS-485/Infrared Data Association (IrDA)/Consumer Infrared (CIR)/ISO 7816 module in the device.

Note

UART and USART acronyms are used interchangeably in this section.

12.1.6.1 UART Overview

The UART is a slave peripheral that utilizes the DMA for data transfer or interrupt polling via host CPU. There are UART modules in the device. All UART modules support IrDA and CIR modes when 48 MHz function clock is used. Each UART can be used for configuration and data exchange with a number of external peripheral devices or interprocessor communication between devices.

12.1.6.1.1 UART Features

The UART includes the following features:

- 16C750-compatible
- RS-485 external transceiver auto flow control support
- 64-byte FIFO buffer for receiver and 64-byte FIFO buffer for transmitter
- Programmable interrupt trigger levels for FIFOs
- Programmable sleep mode
- The 48 MHz functional clock is default option and allows baud rates up to 3.6 Mbps
- Auto-baud between 1200 bits/s and 115.2 Kbits/s (only when 48 MHz function clock is used)
- Optional multi-drop transmission
- Configurable time-guard feature
- Configurable data format:
  - Parity bit: Even, Odd, None
  - Stop-bit: 1, 1.5, 2 bit
- Flow control: Hardware (RTS/CTS) or software (XON/XOFF)
- False start bit detection
- Line break generation and detection
- Fully prioritized interrupt system controls
- Internal test and loopback capabilities
- Modem control functions (CTS, RTS)
- Only module instance in MAIN domain has extended modem control signals (DCD, RI, DTR, DSR)

Unsupported Features:

- See the Module Integration section of this document for a list of module features not supported by the integration on this Device.

12.1.6.1.2 IrDA Features

The IrDA includes the following features:

- Support of IrDA 1.4 slow infrared (SIR), medium infrared (MIR), and fast infrared (FIR) communications:
  - Slow infrared (SIR 115.2 KBAUD), medium infrared (MIR 0.576 MBAUD) and fast infrared (FIR 4.0 MBAUD) operations (very fast infrared (VFIR) is not supported)
  - Frame formatting: addition of variable beginning-of-frame (xBOF) characters and end-of-frame (EOF) characters
  -
  - Asynchronous transparency (automatic insertion of break character)
  - Eight-entry status FIFO (with selectable trigger levels) to monitor frame length and frame errors
  - Framing error, CRC error, illegal symbol (FIR), and abort pattern (SIR, MIR) detection
- IrDA mode when 48 MHz function clock is used

Copyright © 2024 Texas Instruments Incorporated

12.1.6.1.3 CIR Features

The CIR mode uses a variable pulse-width modulation (PWM) technique (based on multiples of a programmable *t* period) to encompass the various formats of infrared encoding for remote-control applications. The CIR logic transmits data packets based on a user-definable frame structure and packet content.

The CIR includes the following features to provide CIR support for remote-control applications:

- Transmit and receive mode
- Free data format (supports any remote-control private standards)
- Selectable bit rate
- Configurable carrier frequency
- 1/2, 5/12, 1/3, or 1/4 carrier duty cycle
- CIR mode when 48 MHz function clock is used

12.1.6.1.4 UART Ports

Table 12-67. UART Clocks and Resets

Clocks	
Module Clock Input	**Description**
UART_ICLK	UART interface clock
UART_FCLK	UART functional clock
Resets	
Module Reset Input	**Description**
UART_RST	UART reset

Table 12-68. UART Hardware Requests

Interrupt Requests		
Module Interrupt Signal	**Description**	**Type**
USART_0_INT	UART interrupt request	Level
DMA Events		
Module DMA Event	**Description**	**Type**
USART_x_EVT0	UART transmit request line	Level
USART_x_EVT1	UART receive request line	Level

12.1.6.2 UART Environment

This section describes the UART/RS-485/IrDA/CIR external connections (environment).

Table 12-69 lists the UART interface input/output (I/O) signals.

Table 12-69. UART I/O Signals

Module Pin Name	I/O[(1)]	Description
RX	I	Serial data input
TX	O	Serial data output[(2)]
CTS	I	Clear to send[(3)]
RTS	O	Request to send[(4)]
DCD	I	Data Carrier Detect[(5)]
DSR	I	Data Set Ready[(6)]
DTR	O	Data Terminal Ready[(7)]
RIN	I	Ring Indicator[(8)]

(1) I = Input; O = Output
(2) Because this pin is active high in IrDA mode and the output is muxed, this pin is set to low on reset (when the UART_MDR1[2-0] bit field is set to 0x7) and takes the defined inactive level of that signal corresponding to when and how the UART_MDR1 register is programmed; that is, the output is 1 (inactive for UART modem modes) and 0 (inactive for IrDA modes).
(3) Active-low modem status signal. Reading the UART_MSR[4] NCTS_STS bit checks the condition of CTS. Reading the UART_MSR[0] CTS_STS bit checks a change of state of CTS since the last read of the modem status register. The auto-CTS mode uses CTS to control the transmitter.

Copyright © 2024 Texas Instruments Incorporated

(4) When active (low), the module is ready to receive data. Setting the UART_MCR[1] RTS bit activates RTS signal, which becomes inactive as the result of a module reset, loopback mode, or clearing the UART_MCR[1] RTS bit. In auto-RTS mode, RTS signal becomes inactive as a result of the receiver threshold logic.
(5) Active-low modem status signal. The condition of DCD can be checked by reading the UART_MSR[7] NCD_STS bit. Any change in its state can be detected by reading the UART_MSR[3] DCD_STS bit.
(6) Active-low modem status signal. Reading the UART_MSR[5] NDSR_STS bit checks the condition of DSR. Reading the UART_MSR[1] DSR_STS bit checks a change of state of DSR since the last read of the UART_MSR register.
(7) When active (low), this signal informs the modem that the module is ready to communicate. It is activated by setting the UART_MCR[0] DTR bit.
(8) Active-low modem status signal. The condition of RIN can be checked by reading the UART_MSR[6] NRI_STS bit. Any change in its state can be detected by reading the UART_MSR[2] RI_STS bit.

Table 12-70 lists the RS-485 interface input/output (I/O) signals.

Table 12-70. UART I/O Signals (RS-485 Mode)

Module Pin	I/O[(1)]	Description
RX	I	Serial data input
TX	O	Serial data output
DIR	O	RS-485 Direction

(1) I = Input; O = Output

Table 12-71 lists the IrDA interface I/O signals.

Table 12-71. UART I/O Signals (IrDA Mode)

Module Pin	I/O[(1)]	Description
RX	I	Serial data input
TX	O	Serial data output in CIR and IrDA modes (SIR, MIR, and FIR).[(2)]
SD	O	SD mode is used to configure the transceivers.[(3)]

(1) I = Input; O = Output
(2) In other modes, this pin is set to the reset value (inactive state).
(3) The SD pinout (see UART_ACREG[6] SD_MOD bit).

Copyright © 2024 Texas Instruments Incorporated

12.1.6.3 UART Functional Description

12.1.6.3.1 UART Block Diagram

The UART module can be divided into three main blocks:

- FIFO management
- Mode selection
- Protocol formatting

FIFO management is common to all functions and enables the transmission and reception of data from the host processor point of view.

There are two modes:

- Function mode: Routes the data to the chosen function (UART, RS-485, IrDA, or CIR) and enables the mechanism corresponding to the chosen function.
- Register mode: Enables conditional access to registers.

For more information about mode configuration, see *Mode Selection*.

Protocol formatting has three subcategories:

- Clock generation: The 48-MHz input clock generates all necessary clocks.
- Data formatting: Each function uses a dedicated state-machine that is responsible for the transition between FIFO data and the associated frame data.
- Interrupt management: Different interrupt types are generated depending on the chosen function. In each mode, when an interrupt is generated, the UART_IIR_UART register indicates the interrupt type.
  - UART mode interrupts: Seven interrupts prioritized in six different levels
  - IrDA mode interrupts: Eight interrupts. The interrupt line is activated when any interrupt is generated (there is no priority).
  - CIR mode interrupts: A subset of existing IrDA mode interrupts is used.

In parallel with these functional blocks, a power-saving strategy exists for each function.

The UART block diagram is shown below.

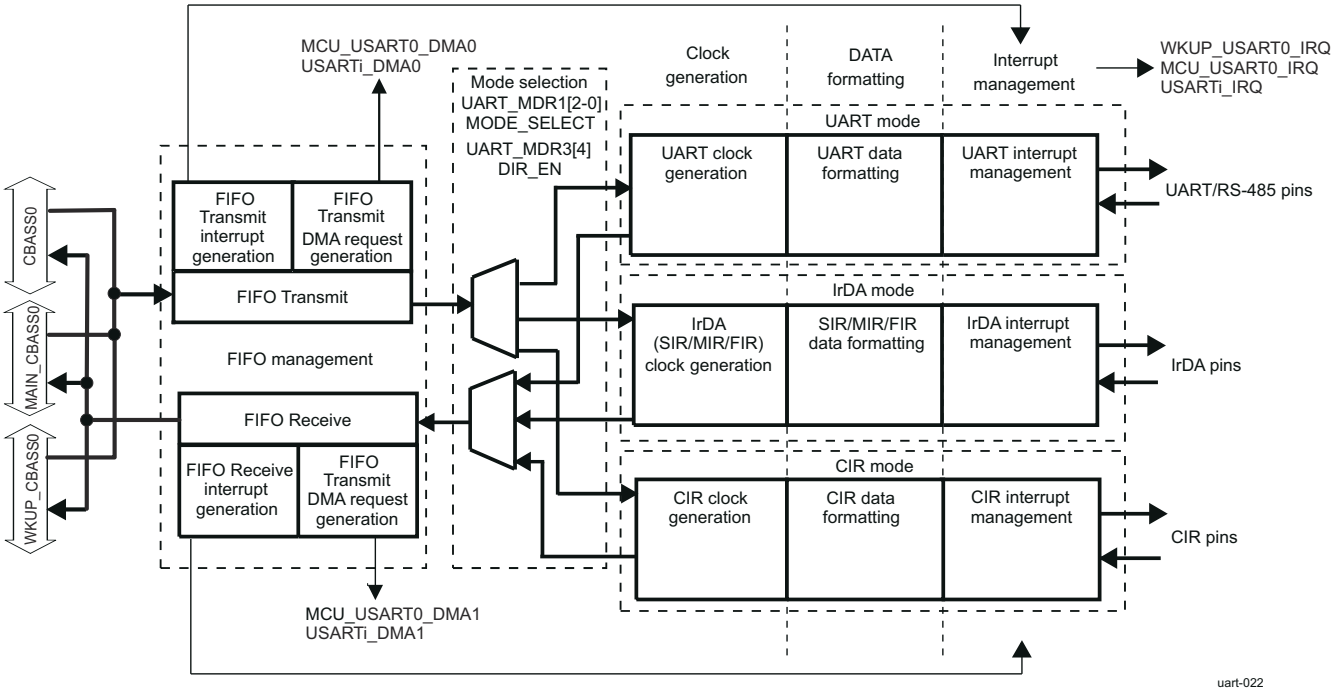


Figure 12-46. UART Functional Block Diagram

Copyright © 2024 Texas Instruments Incorporated

12.1.6.3.2 UART Clock Configuration

Each UART uses a 48-MHz functional clock for its logic and to generate external interface signals. Each UART uses an interface clock for register accesses.

12.1.6.3.3 UART Software Reset

The UART_SYSC[1] SOFTRESET bit controls the software reset; setting this bit to 1 triggers a software reset functionally equivalent to hardware reset.

12.1.6.3.3.1 Independent TX/RX

The receiver and transmitter are enabled by default after reset. Software can choose to disable, re-enable or to reset either the RX or the TX side independently of the other through the UART_ECR register.

12.1.6.3.4 UART Power Management

12.1.6.3.4.1 UART Mode Power Management

12.1.6.3.4.1.1 Module Power Saving

In UART modes, sleep mode is enabled by setting the UART_IER_UART[4] SLEEP_MODE bit to 1 (when the UART_EFR[4] ENHANCED_EN bit is set to 1).

Sleep mode is entered when all of the following conditions exist:

- The serial data input line, RX, is idle.
- The TX FIFO and TX shift register are empty.
- The RX FIFO is empty.
- The only pending interrupts are THR interrupts.

Sleep mode is a good way to lower UART power consumption, but this state can be achieved only when the UART is set to modem mode. Therefore, even if the UART has no key role functionally, it must be initialized in a functional mode to take advantage of sleep mode.

In sleep mode, the module clock and baud rate clock are stopped internally. Because most registers are clocked by these clocks, this greatly reduces power consumption. The module wakes up when a change is detected on the RX line, when data is written to the TX FIFO, and when there is a change in the state of the modem input pins.

An interrupt can be generated on a wake-up event by setting the UART_SCR[4] RX_CTS_WU_EN bit to 1. To understand how to manage the interrupt, see Section 12.1.6.3.5.1.2, *Wake-Up Interrupt*.

Note

There must be no writing to the divisor latches, UART_DLL and UART_DLH, to set the baud clock (BCLK) while in sleep mode. It is advisable to disable sleep mode using the UART_IER_UART[4] SLEEP_MODE bit before writing to the UART_DLL or UART_DLH register.

12.1.6.3.4.1.2 System Power Saving

Sleep and auto-idle modes are embedded power-saving features. Power-reduction techniques can be applied at the system level by shutting down certain internal clock and power domains of the device.

For more information, see *Power*, in the *Device Configuration*.

12.1.6.3.4.2 IrDA Mode Power Management

12.1.6.3.4.2.1 Module Power Saving

In IrDA modes, sleep mode is enabled by setting the UART_MDR1[3] IR_SLEEP bit to 1.

Sleep mode is entered when all of the following conditions exist:

- The serial data input line, RXD, is idle.
- The TX FIFO and TX shift register are empty.
- The RX FIFO is empty.
- No interrupts are pending except THR interrupts.

Copyright © 2024 Texas Instruments Incorporated

The module wakes up when a change is detected on the RXD line or when data is written to the TX FIFO.

12.1.6.3.4.2.2 System Power Saving

System power saving for the IrDA mode has the same function as for the UART mode (see Section 12.1.6.3.4.1.2, *System Power Saving*).

12.1.6.3.4.3 CIR Mode Power Management

12.1.6.3.4.3.1 Module Power Saving

Module power saving for the CIR mode has the same function as for the IrDA mode (see Section 12.1.6.3.4.2.1, *Module Power Saving*).

12.1.6.3.4.3.2 System Power Saving

System power saving for the CIR mode has the same function as for the UART mode (see Section 12.1.6.3.4.1.2, *System Power Saving*).

12.1.6.3.4.4 Local Power Management

Table 12-72 describes power-management features available for the UART.

Note

For information about source clock gating and the sleep/wake-up transitions description, see *Power*, in the *Device Configuration*.

Table 12-72. UART Local Power-Management Features

Feature	Registers	Description
Clock autogating		
Peripheral idle modes		
Clock activity	N/A	Feature not available
Controller standby modes	N/A	Feature not available
Global wake-up enable	UART_SYSC[2] ENAWAKEUP	This bit enables the wake-up feature at module level.
Wake-Up sources enable	N/A	Feature not available

12.1.6.3.5 UART Interrupt Requests

12.1.6.3.5.1 UART Mode Interrupt Management

12.1.6.3.5.1.1 UART Interrupts

The UART mode includes seven possible interrupts prioritized to six levels.

When an interrupt is generated, the interrupt identification register (UART_IIR_UART) sets the UART_IIR_UART[0] IT_PENDING bit to 0 to indicate that an interrupt is pending, and indicates the type of interrupt through the UART_IIR_UART[5-1] bit field. Table 12-73 summarizes the interrupt control functions.

Table 12-73. UART Mode Interrupts

IIR[5:0]	Priority Level	Interrupt Type	Interrupt Source	Interrupt Reset Method
000001	N/A	No Interrupt	N/A	N/A
000110	1	Receiver line status	OE, FE, PE, or BI errors occur in characters in the RX FIFO.	FE, PE, BI: Read the UART_RHR register. OE: Read the UART_LSR_UART register.
001100	2	RX time-out	Stale data in RX FIFO	Read the UART_RHR register if using the default timeout behavior: EFR2[6]=0 Cleared by reading its value (IIR) if using the periodic timeout behavior: EFR2[6]=1

Copyright © 2024 Texas Instruments Incorporated

Table 12-73. UART Mode Interrupts (continued)

IIR[5:0]	Priority Level	Interrupt Type	Interrupt Source	Interrupt Reset Method
000100	2	RHR interrupt	DRDY (data ready) (FIFO disabled) RX FIFO above trigger level (FIFO enabled)	Read the UART_RHR register until the interrupt condition disappears
000010	3	THR interrupt	TFE (THR empty) (FIFO disabled) TX FIFO below trigger level (FIFO enabled)	Write to the UART_THR until the interrupt condition disappears
000000	4	Modem status	See the UART_MSR register.	Read the MSR register
010000	5	XOFF interrupt/ special character interrupt	Receive XOFF characters/special character	Receive XON character(s), if XOFF interrupt/read of the UART_IIR_UART register, if special character interrupt
100000	6	CTS, RTS	RTS pin or CTS pin change state from active (low) to inactive (high)	Read the UART_IIR_UART register

For the receiver-line status interrupt, the UART_LSR_UART[7] RX_FIFO_STS bit generates the interrupt.

For the XOFF interrupt, if an XOFF flow character detection caused the interrupt, the interrupt is cleared by an XON flow character detection. If special character detection caused the interrupt, the interrupt is cleared by a read of the UART_IIR_UART register.

12.1.6.3.5.1.2 Wake-Up Interrupt

Wake-up interrupt is a special interrupt that works differently from other interrupts. This interrupt is enabled when the RX_CTS_DSR_WAKE_UP_ENABLE bit of the Supplementary Control Register (SCR[4]) is set to 1. The IIR register is not modified when it occurs, SSR[1] must be checked to detect a wake-up event. When a wake-up event occurs, the only way to clear it is to reset SCR[4] to 0. Wake-up can also occur if the WER[7] TX_WAKEUP_EN is set to 1 and one of the following events occurs:

1. THR interrupt is enabled and occurs (omitted if TX DMA request is enabled)
2. TX DMA request is enabled and occurs
3. TX_STATUS_IT is enabled and occurs (only IrDA and CIR modes). Cannot be used with THR Interrupt.

12.1.6.3.5.2 IrDA Mode Interrupt Management

12.1.6.3.5.2.1 IrDA Interrupts

The IrDA function generates interrupts. All interrupts can be enabled and disabled by writing to the appropriate bit in the interrupt enable register (UART_IER_IRDA). The interrupt status of the device can be checked by reading the interrupt identification register (UART_IIR_IRDA).

The UART, IrDA, and CIR modes have different interrupts in the UART module and, therefore, different UART_IER_IRDA and UART_IIR_IRDA mappings, depending on the selected mode.

The IrDA modes have eight possible interrupts (see Table 12-74). The interrupt line is activated when any interrupt is generated (there is no priority).

Table 12-74. IrDA Mode Interrupts

IIR_IRDA Bit	Interrupt Type	Interrupt Source	Interrupt Reset Method
0	RHR interrupt	DRDY (data ready) (FIFO disabled) RX FIFO above trigger level (FIFO enabled)	Read the UART_RHR register until the interrupt condition disappears.
1	THR interrupt	TFE (UART_THR empty) (FIFO disabled) TX FIFO below trigger level (FIFO enabled)	Write to the UART_THR until the interrupt condition disappears.
2	Last byte in RX FIFO	Last byte of frame in RX FIFO is available to be read at the UART_RHR port.	Read the UART_RHR register.
3	RX overrun	Write to the UART_RHR register when the RX FIFO is full.	Read UART_RESUME register.
4	Status FIFO interrupt	Status FIFO triggers level reached.	Read STATUS FIFO.

Copyright © 2024 Texas Instruments Incorporated

Table 12-74. IrDA Mode Interrupts (continued)

IIR_IRDA Bit	Interrupt Type	Interrupt Source	Interrupt Reset Method
5	TX status	UART_THR empty before EOF sent. Last bit of transmission of the IrDA frame occurred, but with an underrun error OR Transmission of the last bit of the IrDA frame completed successfully.	Read the UART_RESUME register OR Read the UART_IIR_IRDA register.
6	Receiver line status interrupt	CRC, ABORT, or frame-length error is written into the STATUS FIFO.	Read the STATUS FIFO (read until empty - maximum of eight reads required).
7	Received EOF	Received end-of-frame	Read the UART_IIR_IRDA register.

12.1.6.3.5.2.2 Wake-Up Interrupts

The wake-up interrupt for IrDA mode has the same function as that for UART mode (see Section 12.1.6.3.5.1.2, *Wake-Up Interrupt*).

12.1.6.3.5.3 CIR Mode Interrupt Management

12.1.6.3.5.3.1 CIR Interrupts

The CIR function generates interrupts that can be enabled and disabled by writing to the appropriate bit in the interrupt enable register (UART_IER_CIR). The interrupt status of the device can be checked by reading the interrupt identification register (UART_IIR_CIR).

The UART, IrDA, and CIR modes have different interrupts in the UART module and, therefore, different UART_IER_CIR and UART_IIR_CIR mappings, depending on the selected mode.

Table 12-75 lists the interrupt modes to be maintained. In CIR mode, the sole purpose of the UART_IIR_CIR[5] TX_STATUS_IT bit is to indicate that the last bit of infrared data was passed to the TX pin.

Table 12-75. CIR Mode Interrupts

IIR_CIR Bit Number	Interrupt Type	Interrupt Source	Interrupt Reset Method
0			
1	THR interrupt	TFE (THR empty) (FIFO disabled) TX FIFO below trigger level (FIFO enabled)	Write to the UART_THR register until the interrupt condition disappears
2			
3			
4	N/A for CIR mode	N/A for CIR mode	N/A for CIR mode
5	TX status	Transmission of the last bit of the frame is complete successfully	Read the UART_IIR_CIR register
6	N/A for CIR mode	N/A for CIR mode	N/A for CIR mode
7	N/A for CIR mode	N/A for CIR mode	N/A for CIR mode

12.1.6.3.5.3.2 Wake-Up Interrupts

The wake-up interrupt for CIR mode has the same function as that for UART mode (see Section 12.1.6.3.5.1.2, *Wake-Up Interrupt*).

12.1.6.3.6 UART FIFO Management

The FIFO is accessed by reading and writing the UART_RHR and UART_THR registers. Parameters are controlled using the FIFO control register (UART_FCR) and supplementary control register (UART_SCR). Reading the UART_SSR[0] TX_FIFO_FULL bit at 1 means the FIFO is full.

The UART_TLR register controls the FIFO trigger level, which enables DMA and interrupt generation. After reset, transmit (TX) and receive (RX) FIFOs are disabled; thus, the trigger level is the default value of 1 byte. Figure 12-47 shows the FIFO management registers.

Note

Data in the UART_RHR register is not overwritten when an overflow occurs.

Copyright © 2024 Texas Instruments Incorporated

Note

The UART_SFLSR, UART_SFREGL, and UART_SFREGH status registers are used in IrDA mode only. For information about their use, see Section 12.1.6.3.8.3.3, *IrDA Data Formatting*.

Note

Bits UART_FCR[2] TX_FIFO_CLEAR and UART_FCR[1] RX_FIFO_CLEAR are automatically cleared by hardware after 4 × + 5 × UARTi_FCLK clock cycles. This delay is needed to finish the resetting of the corresponding FIFO and DMA control registers.

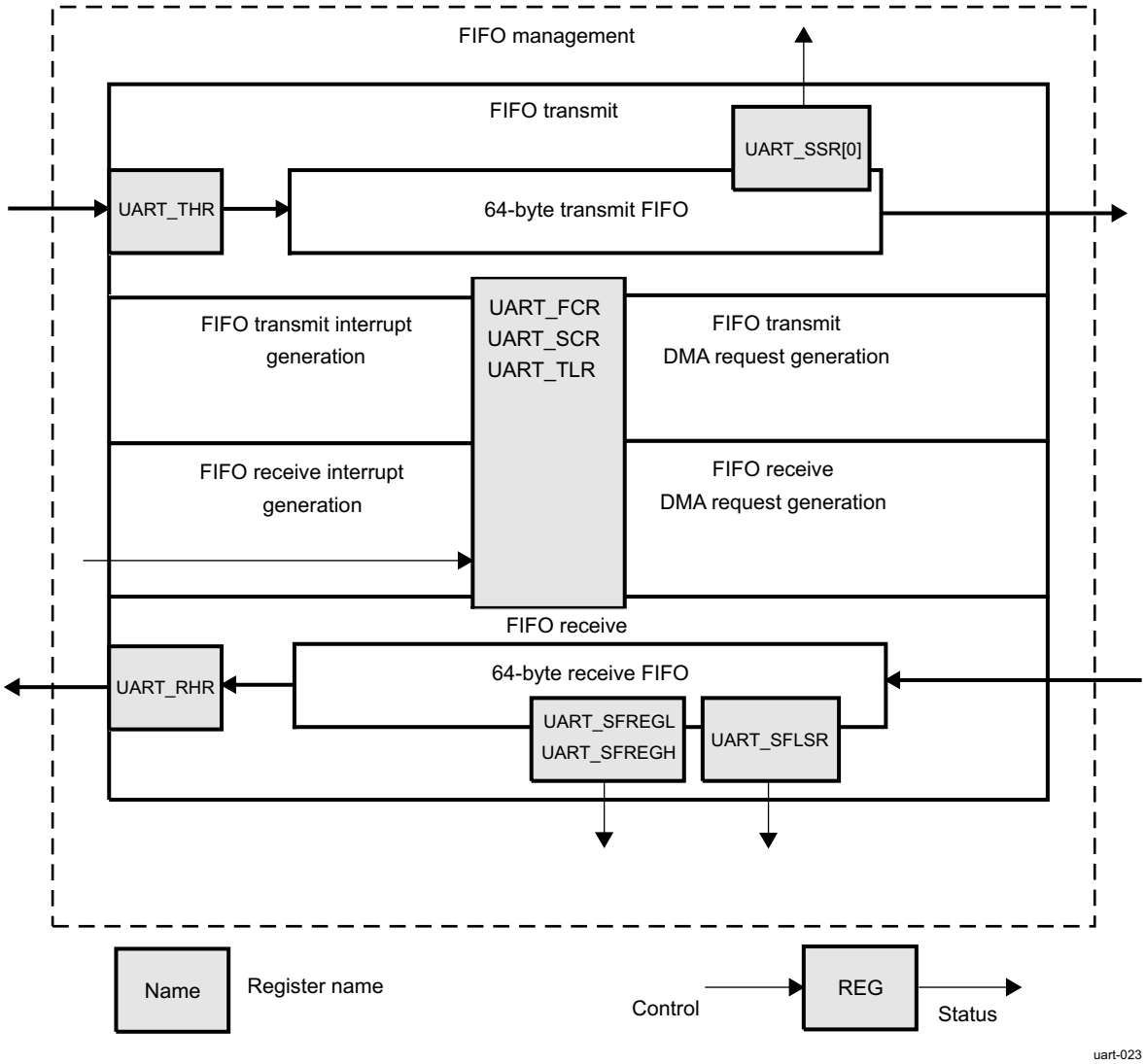


Figure 12-47. UART FIFO Management Registers

12.1.6.3.6.1 FIFO Trigger

12.1.6.3.6.1.1 Transmit FIFO Trigger

Table 12-76 lists the TX FIFO trigger level settings.

Table 12-76. UART TX FIFO Trigger Level Setting Summary

SCR[6]	TLR[3:0]	TX FIFO Trigger Level
0	= 0x0	Defined by the UART_FCR[5-4] TX_FIFO_TRIG bit field (8,16, 32, or 56 spaces)

Copyright © 2024 Texas Instruments Incorporated

Table 12-76. UART TX FIFO Trigger Level Setting Summary (continued)

SCR[6]	TLR[3:0]	TX FIFO Trigger Level
0	!= 0x0	Defined by the UART_TLR[3-0] TX_FIFO_TRIG_DMA bit field (from 4 to 60 spaces with a granularity of 4 spaces)
1	Value	Defined by the concatenated value of TLR[3:0] (higher bits) and FCR[5:4] (lower bits) from 1 to 63 spaces with a granularity of 1 space. **Note:** The combination of TLR[3:0]=0000 and FCR[5:4]=00 (all zeros) is not supported (min 1 space required). All zeros will result in unsupported behavior.

12.1.6.3.6.1.2 Receive FIFO Trigger

Table 12-77 lists the RX FIFO trigger-level settings.

Table 12-77. UART RX FIFO Trigger-Level Setting Summary

SCR[7]	TLR[7:4]	RX FIFO Trigger Level
0	= 0x0	Defined by the UART_FCR[7-6] RX_FIFO_TRIG bit field (8,16, 56, or 60 characters)
0	!= 0x0	Defined by the UART_TLR[7-4] RX_FIFO_TRIG_DMA bit field (from 4 to 60 characters with a granularity of 4 characters)
1	Value	Defined by the concatenated value of TLR[7:4] and FCR[7:6] from 1 to 63 characters with a granularity of one character. **Note:** The combination of TLR[7:4]=0000 and FCR[7:6]=00 (all zeros) is not supported (min 1 character required). All zeros will result in unsupported behavior.

The receive threshold is programmed using the UART_TCR[7-4] RX_FIFO_TRIG_START and UART_TCR[3-0] RX_FIFO_TRIG_HALT bit fields:

- Trigger levels from 0 to 60 bytes are available with a granularity of 4 (trigger level = 4 × [4-bit register value]).
- To ensure correct device operation, ensure that RX_FIFO_TRIG_HALT > RX_FIFO_TRIG when auto-RTS is enabled.

  Delay = [4 + 16 × (1 + CHAR_LENGTH + Parity + Stop – 0.5)] × Baud_rate + 4 × FCLK

  Note

  The RTS signal is deasserted after the UART module receives the data over RX_FIFO_TRIG_HALT. Delay means how long the UART module takes to deassert the RTS signal after reaching RX_FIFO_TRIG_HALT.

- In FIFO interrupt mode with flow control, ensure that the trigger level to HALT transmission is greater than or equal to the RX FIFO trigger level (the UART_TCR[7-4] RX_FIFO_TRIG_START bit field or the UART_FCR[7-6] RX_FIFO_TRIG bit field); otherwise, FIFO operation stalls. In FIFO DMA mode with flow control, this concept does not exist, because a DMA request is sent when a byte is received.

12.1.6.3.6.2 FIFO Interrupt Mode

In FIFO interrupt mode (the FIFO control register UART_FCR[0] FIFO_EN bit is set to 1 and relevant interrupts are enabled by the UART_IER_UART register), an interrupt signal informs the processor of the status of the receiver and transmitter. These interrupts are raised when the RX/TX FIFO threshold (the UART_TLR[7-4] RX_FIFO_TRIG_DMA and UART_TLR[3-0] TX_FIFO_TRIG_DMA bit fields or the UART_FCR[7-6] RX_FIFO_TRIG and UART_FCR[5-4] TX_FIFO_TRIG bit fields, respectively) is reached.

The interrupt signals instruct the Host CPU to transfer data to the destination (from the UART in receive mode and/or from any source to the UART FIFO in transmit mode).

When UART flow control is enabled with interrupt capabilities, the UART flow control FIFO threshold (the UART_TCR[3-0] RX_FIFO_TRIG_HALT bit field) must be greater than or equal to the RX FIFO threshold.

Figure 12-48 shows the generation of the RX FIFO interrupt request.

Copyright © 2024 Texas Instruments Incorporated

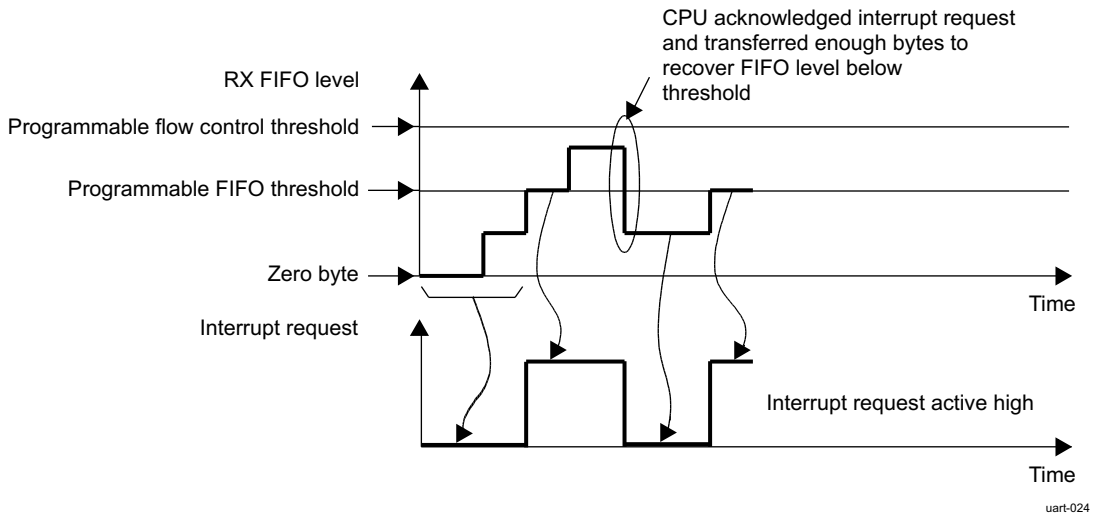


Figure 12-48. UART RX FIFO Interrupt Request Generation

In receive mode, no interrupt is generated until the RX FIFO reaches its threshold. Once low, the interrupt can be deasserted only when the Host CPU has handled enough bytes to put the FIFO level below threshold. The flow control threshold is set at a higher value than the FIFO threshold.

Figure 12-49 shows the generation of the TX FIFO interrupt request.

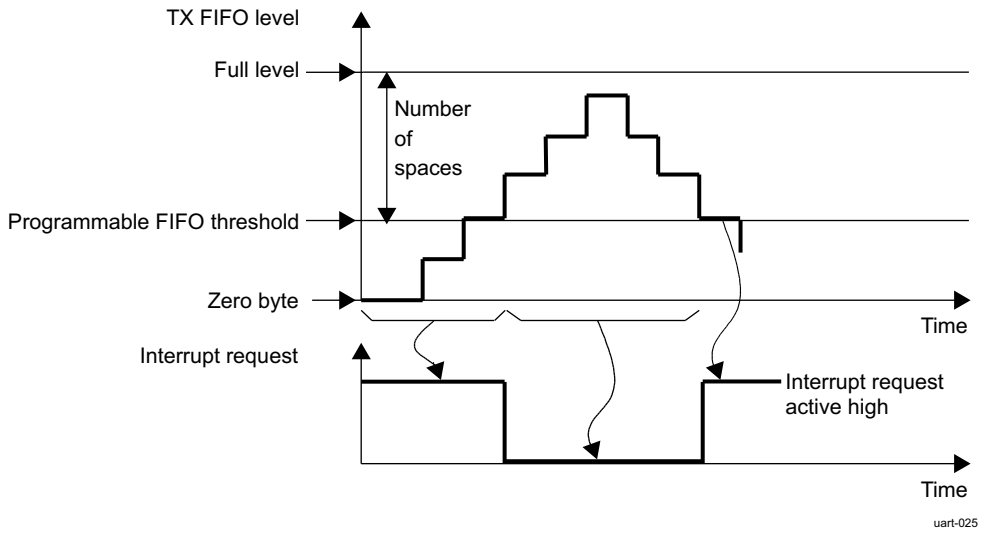


Figure 12-49. UART TX FIFO Interrupt Request Generation

In transmit mode, an interrupt request is automatically asserted when the TX FIFO is empty. This request is deasserted when the TX FIFO crosses the threshold level. The interrupt line is deasserted until a sufficient number of elements is transmitted to go below the TX FIFO threshold.

12.1.6.3.6.3 FIFO Polled Mode Operation

In FIFO polled mode (the UART_FCR[0] FIFO_EN bit is set to 0 and the relevant interrupts are disabled by the UART_IER_UART register), the status of the receiver and transmitter can be checked by polling the line status register (UART_LSR_UART).

This mode is an alternative to the FIFO interrupt mode of operation in which the status of the receiver and transmitter is automatically determined by sending interrupts to the Host CPU.

Copyright © 2024 Texas Instruments Incorporated

12.1.6.3.6.4 FIFO DMA Mode Operation

Although the DMA operation includes four modes (DMA modes 0 through 3), the information in *UART Hardware Requests*, assumes that mode 1 is used. (Mode 2 and mode 3 are legacy modes that use only one DMA request for each module.)

In mode 2, the remaining DMA request is used for RX. In mode 3, the remaining DMA request is used for TX.

DMA requests in mode 2 and mode 3 use the signals (where i = 0 to .

signals are not used by the module in mode 2 and mode 3:

The DMA mode and signals usage can be selected as follows:

- When SCR[0]=0:
  - Setting FCR[3] to 0 enables DMA mode 0
  - Setting FCR[3] to 1 enables DMA mode 1
- When SCR[0]=1:
  - SCR[2:1] determines DMA mode 0 to 3 according to the Supplementary Control Register (SCR) description.

For example:

- If no DMA operation is desired: set SCR[0] to 1 and SCR[2:1] to 00 (FCR[3] is discarded)
- If DMA mode 1 is desired: either set SCR[0] to 0 and FCR[3] to 1 or set SCR[0] to 1 and SCR[2:1] to 01 (FCR[3] is discarded)

If the FIFOs are disabled (FCR[0]=0), DMA operations occur in single character transfers.

Note that when DMA Mode 0 has been programmed, the signals associated with DMA operation are not active.

Depending on UART_MDR3[2] SET_DMA_TX_THRESHOLD, the threshold can be programmed different ways:

- SET_TX_DMA_THRESHOLD = 1:

  The threshold value will be the value of the UART_TX_DMA_THRESHOLD register. If SET_TX_DMA_THRESHOLD + TX trigger spaces 64, then the default method of threshold is used: threshold value = TX FIFO size.
- SET_TX_DMA_THRESHOLD = 0:

  The threshold value = TX FIFO size TX trigger space. The TX DMA line is asserted if the TX FIFO level is lower then the threshold. It remains asserted until TX trigger spaces number of bytes are written into the FIFO. The DMA line is then deasserted and the FIFO level is compared with the threshold value.

12.1.6.3.6.4.1 DMA sequence to disable TX DMA

In order to disable TX DMA if it is not needed anymore (e.g. all transfers are done and UART idle mode is desired), the following sequence must be use

1. DMA mode 1 is set (both TX/RX DMA) by registers UART_SCR[0] DMA_MODE_CTL = 0 and UART_FCR[3] DMA_MODE = 1:
   a. Set the UART_SCR[2-1] DMA_MODE_2 bit fields to 01 (DMA mode 1)
   b. Set the UART_SCR[0] DMA_MODE_CTL bit to1 (this setting of UART_SCR[0] DMA_MODE-CTL will ignores UART_FCR[3] DMA_MODE_CTL bit)

   Note

   It is strongly suggested to do steps ‘a’ and ‘b’ in two separate write in order to avoid malfunction of the device.

   c. Set the UART_FCR[3] DMA_MODE bit to 0. It is not necessary but suggested to avoid restore of DMA mode 1 during accidental reset of UART_SCR[0] DMA_MODE_CTL bit. Be sure that all data was read out from RX FIFO and if it possible disable the RX side. In UART mode the RTS/CTS or XOFF/XON protocol can be used. In IrDA modes RX can be forcibly disabled by setting UART_ACREG[5] DIS_IR_RX bit

Copyright © 2024 Texas Instruments Incorporated

Note

There can be RX DATA loss during the next steps if all DATA was not read out or there was an ongoing reception!

d. Set the UART_FCR[2-1] DMA_MODE bit field to 11 (clear TX and RX FIFO and resets its counter logic to 0. Returns to 0 after clearing FIFO).
e. Set the UART_SCR[2-1] DMA_MODE_2 bit field to 10 (DMA mode 2, RX only).
f. Set the UART_FCR[2-1] DMA_MODE bit field to 11 (clear TX and RX FIFO and the DMA request again).
g. Set the UART_SCR[2-1] DMA_MODE_2 bit field to 00 (no DMA) or keep 10 if RX DMA is needed.

2. DMA mode 1 is set (both TX/RX DMA) by registers UART_FCR[3] DMA_MODE = 0 and UART_SCR[0] DMA_MODE_CTL = 1, UART_SCR[2-1] DMA_MODE_2 = 01. It is almost the same as above, but steps ‘a’, and ‘b’ can be skipped:
   a. Set the UART_FCR[3] DMA_MODE bit to 0. It is not necessary but suggested to avoid restore of DMA mode 1 during accidental reset of UART_SCR[0] DMA_MODE_CTL bit. Be sure that all data was read out from RX FIFO and if it possible disable the RX side. In UART mode the RTS/CTS or XOFF/XON protocol can be used. In IrDA modes RX can be forcibly disabled by setting UART_ACREG[5] DIS_IR_RX bit

Note

There can be RX DATA loss during the next steps if all DATA was not read out or there was an ongoing reception!

   b. Set the UART_FCR[2-1] DMA_MODE bit field to 11 (clear TX and RX FIFO and resets its counter logic to 0. Returns to 0 after clearing FIFO).
   c. Set the UART_SCR[2-1] DMA_MODE_2 bit field to 10 (DMA mode 2, RX only).
   d. Set the UART_FCR[2-1] DMA_MODE bit field to 11 (clear TX and RX FIFO and the DMA request again).
   e. Set the UART_SCR[2-1] DMA_MODE_2 bit field to 00 (no DMA) or keep 10 if RX DMA is needed.

3. DMA mode 3 is set (TX DMA only) by registers UART_FCR[3] DMA_MODE = 0 and UART_SCR[0] DMA_MODE_CTL = 1, UART_SCR[2-1] DMA_MODE_2 = 11. It is the same as above:
   a. Set the UART_FCR[3] DMA_MODE bit to 0. It is not necessary but suggested to avoid restore of DMA mode 1 during accidental reset of UART_SCR[0] DMA_MODE_CTL bit. Be sure that all data was read out from RX FIFO and if it possible disable the RX side. In UART mode the RTS/CTS or XOFF/XON protocol can be used. In IrDA modes RX can be forcibly disabled by setting UART_ACREG[5] DIS_IR_RX bit

Note

There can be RX DATA loss during the next steps if all DATA was not read out or there was an ongoing reception!

   b. Set the UART_FCR[2-1] DMA_MODE bit field to 11 (clear TX and RX FIFO and resets its counter logic to 0. Returns to 0 after clearing FIFO).
   c. Set the UART_SCR[2-1] DMA_MODE_2 bit field to 10 (DMA mode 2, RX only).
   d. Set the UART_FCR[2-1] DMA_MODE bit field to 11 (clear TX and RX FIFO and the DMA request again).
   e. Set the UART_SCR[2-1] DMA_MODE_2 bit field to 00 (no DMA) or keep 10 if RX DMA is needed.

12.1.6.3.6.4.2 DMA Transfers (DMA Mode 1, 2, or 3)

Figure 12-50 through Figure 12-53 show the supported DMA operations.

Copyright © 2024 Texas Instruments Incorporated

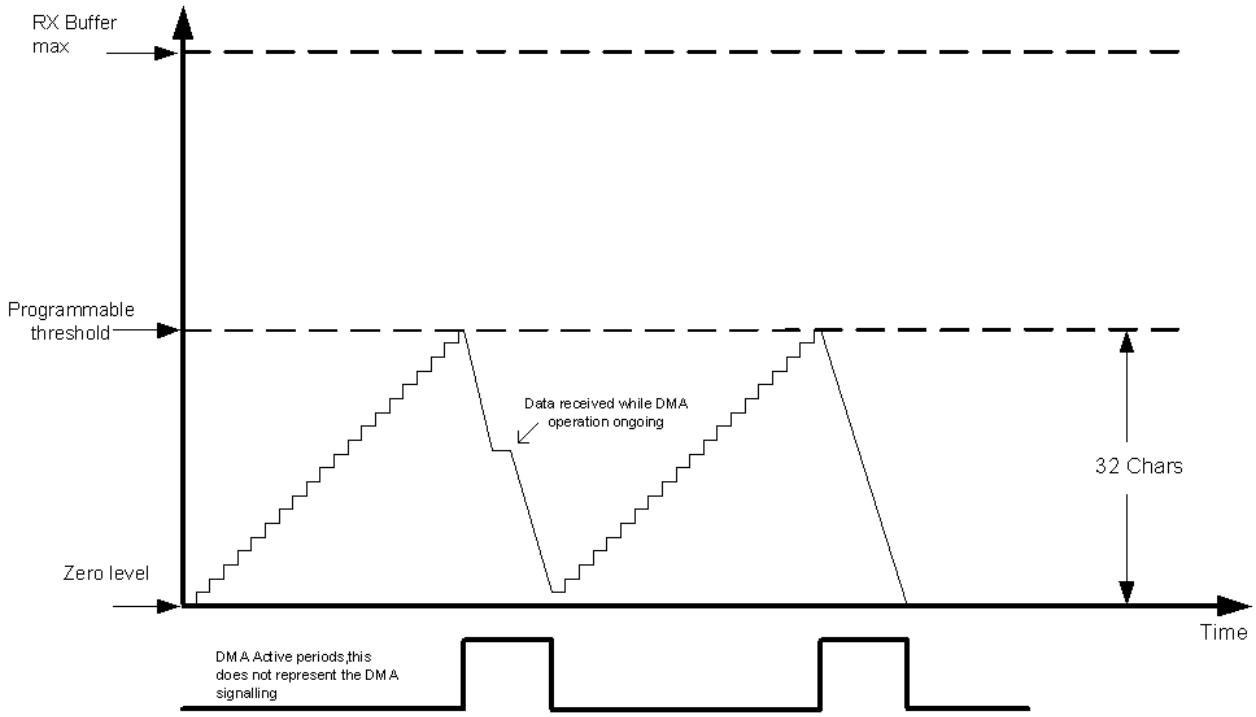


Figure 12-50. UART Receive FIFO DMA Request Generation (32 Characters)

In receive mode, a DMA request is generated when the RX FIFO reaches its threshold level defined in the trigger level register (UART_TLR). This request is deasserted when the number of bytes defined by the threshold level is read by the device DMA controllers.

In transmit mode, a DMA request is automatically asserted when the TX FIFO is empty. This request is deasserted when the number of bytes defined by the number of spaces in the UART_TLR register is written by the device DMA controllers. If an insufficient number of characters is written, the DMA request stays active.

Copyright © 2024 Texas Instruments Incorporated

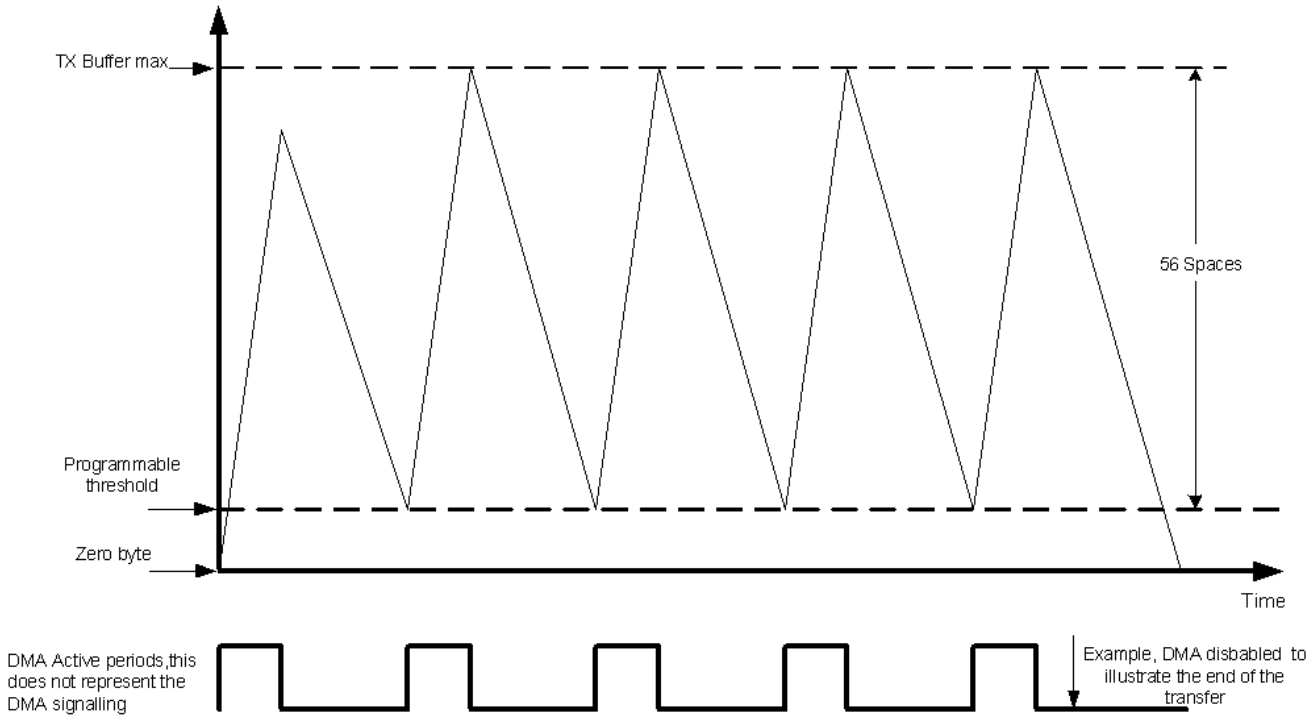


Figure 12-51. UART Transmit FIFO DMA Request Generation (56 Spaces)

The DMA request is again asserted if the FIFO can receive the number of bytes defined by the UART_TLR register.

The threshold can be programmed in a number of ways. Figure 12-51 shows a DMA transfer operating with a space setting of 56 that can arise from using the auto settings in the UART_FCR[5-4] TX_FIFO_TRIG bit field or the UART_TLR[3-0] TX_FIFO_TRIG_DMA bit field concatenated with the TX_FIFO_TRIG bit field.

The setting of 56 spaces in the UART module must correlate with the settings of the device DMA controllers, so that the buffer does not overflow (program the DMA request size of the LH controller to equal the number of spaces in the UART module).

Figure 12-52 shows an example with eight spaces to show the buffer level crossing the space threshold. The LH DMA controller settings must correspond to those of the UART module.

Copyright © 2024 Texas Instruments Incorporated

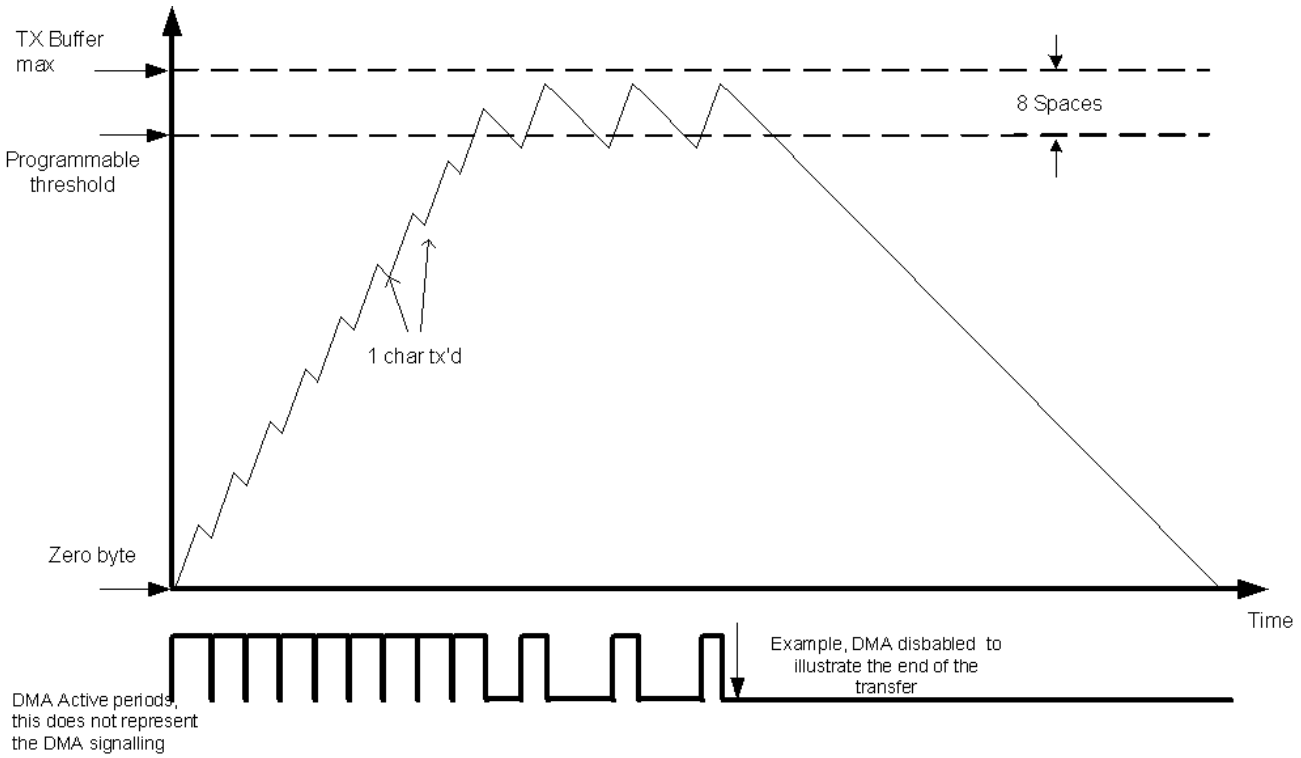


Figure 12-52. UART Transmit FIFO DMA Request Generation (8 Spaces)

The next example shows the setting of one space that uses the DMA for each transfer of one character to the transmit buffer (see Figure 12-53). The buffer is filled faster than the baud rate at which data is transmitted to the TX pin. Eventually, the buffer is completely full and the DMA operations stop transferring data to the transmit buffer.

On two occasions, the buffer holds the maximum amount of data words; shortly after this, the DMA is disabled to show the slower transmission of the data words to the TX pin. Eventually, the buffer is emptied at the rate specified by the baud rate settings of the UART_DLL and UART_DLH registers.

The DMA settings must correspond to the system LH DMA controller settings to ensure correct operation of this logic.

Copyright © 2024 Texas Instruments Incorporated

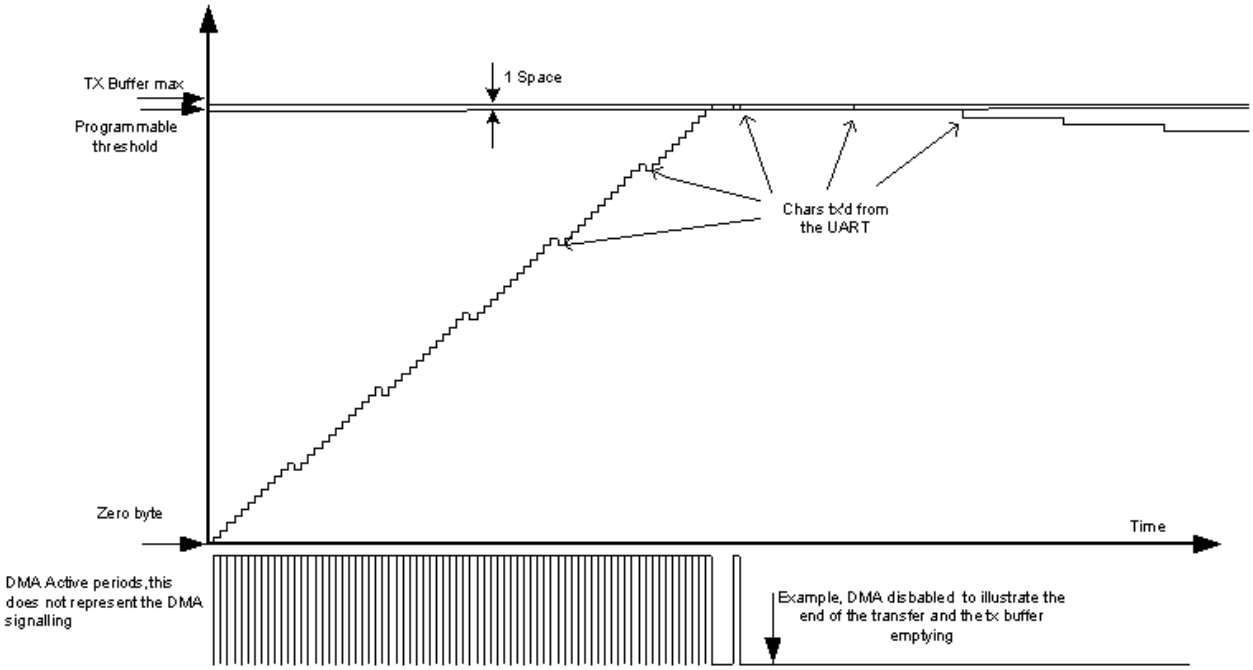


Figure 12-53. UART Transmit FIFO DMA Request Generation (1 Space)

The final example illustrates the setting of eight spaces but setting the TX DMA threshold directly by setting UART_MDR3[1] NONDEFAULT_FREQ bit and UART_TX_DMA_THRESHOLD register (see Figure 12-54). In the example, the UART_TX_DMA_THRESHOLD[5-0] TX_DMA_THRESHOLD = 3 and the trigger level is 8. The buffer is filled at a faster rate than the BAUD rate transmits data to the TX pin. The buffer is filled with 8 bytes and the DMA operations stop transferring data to the transmit buffer. When the buffer is emptied to the threshold level by transmission, the DMA operation activates again to fill the buffer with 8 bytes.

Eventually, the buffer will be emptied at the rate specified by the BAUD Rate settings of the UART_DLL and UART_DLH registers.

If the selected threshold level + trigger level exceeds max buffer size, then the original TX DMA threshold method is used to prevent TX overrun, regardless of the UART_MDR3[1] NONDEFAULT_FREQ value.

The DMA settings should correspond to the system Local Host DMA controller settings in order to ensure the correct operation of this logic.

Copyright © 2024 Texas Instruments Incorporated

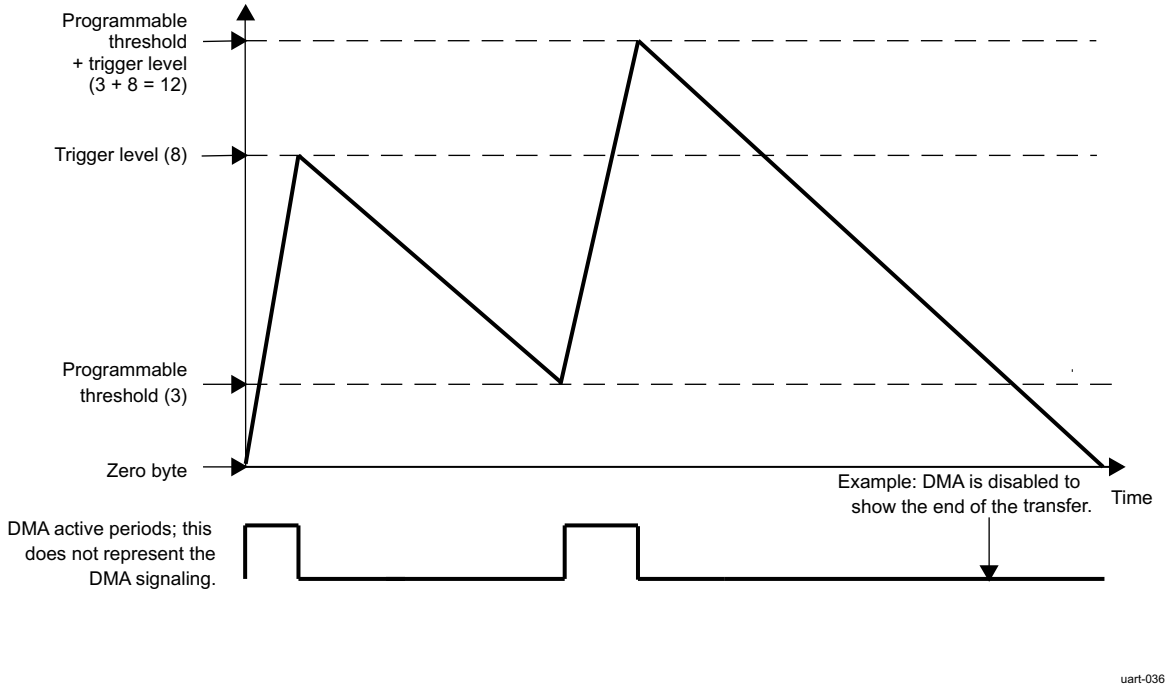


Figure 12-54. UART Transmit FIFO DMA Request Generation Using Direct TX DMA Threshold Programming. (Threshold = 3; Spaces = 8)

12.1.6.3.6.4.3 DMA Transmission

Figure 12-55 shows DMA transmission.

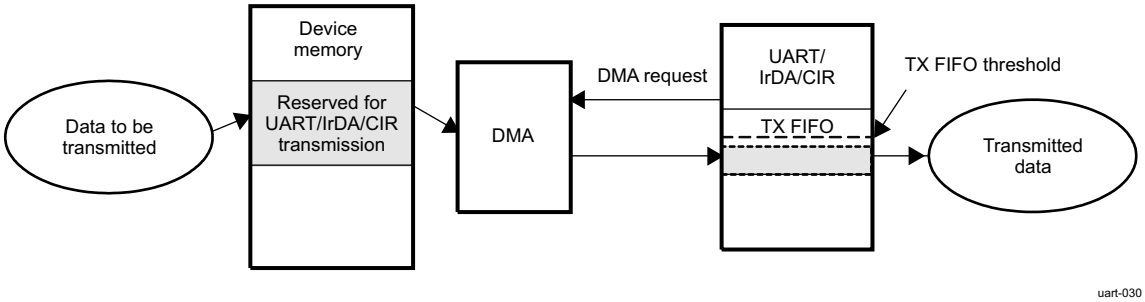


Figure 12-55. DMA Transmission

1. Data to be transmitted are put in the device memory reserved for UART transmission by the DMA:
   a. Until the TX FIFO trigger level is not reached, a DMA request is generated
   b. An element (1 byte) is transferred from the SDRAM to the TX FIFO at each DMA request (DMA element synchronization).
2. Data in the TX FIFO are automatically transmitted.
3. The end of the transmission is signaled by the UART_THR empty (TX FIFO empty).

Note

In IrDA mode, the transmission does not end immediately after the TX FIFO empties, at which point the last data byte, the CRC field, and the stop flag still must be transmitted; thus, the end of transmission occurs a few milliseconds after the UART_THR register empties.

12.1.6.3.6.4.4 DMA Reception

Figure 12-56 shows DMA reception.

Copyright © 2024 Texas Instruments Incorporated

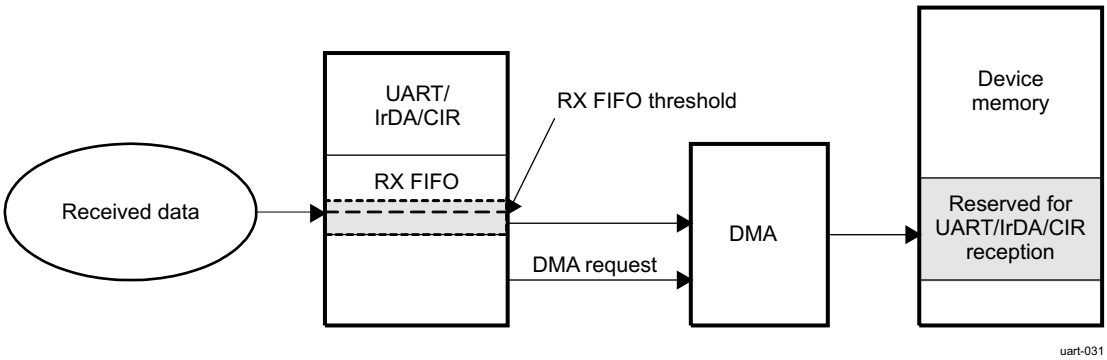


Figure 12-56. DMA Reception

1. Enable the reception.
2. Received data are put in the RX FIFO.
3. Data are transferred from the RX FIFO to the device memory by the DMA:
   a. At each received byte, the RX FIFO trigger level (one character) is reached and a DMA request is generated.
   b. An element (1 byte) is transferred from the RX FIFO to the SDRAM at each DMA request (DMA element synchronization).
4. The end of the reception is signaled by the EOF interrupt.

12.1.6.3.7 UART Mode Selection

12.1.6.3.7.1 Register Access Modes

12.1.6.3.7.1.1 Operational Mode and Configuration Modes

Register access depends on the register access mode, although register access modes are not correlated to functional mode selection. Three different modes are available:

- Operational mode
- Configuration mode A
- Configuration mode B

Operational mode is the selected mode when the function is active; serial data transfer can be performed in this mode.

Configuration mode A and configuration mode B are used during module initialization steps. These modes enable access to configuration registers, which are hidden in the operational mode. The modes are used when the module is inactive (no serial data transfer processed) and only for initialization or reconfiguration of the module.

The value of the UART_LCR register determines the register access mode (see Table 12-78).

Table 12-78. UART Register Access Mode Programming (Using UART_LCR)

Mode	Condition
Configuration mode A	UART_LCR[7] = 0x1 and UART_LCR[7-0] != 0xBF
Configuration mode B	UART_LCR[7] = 0x1 and UART_LCR[7-0] = 0xBF
Operational mode	UART_LCR[7] = 0x0

12.1.6.3.7.1.2 Register Access Submode

In each access register mode (operational mode or configuration mode A/B), some register accesses are conditional on the programming of a submode (MSR_SPR, TCR_TLR, and XOFF). These registers are identified in Table 12-100, *UART Load FIFO Triggers Defined by the Concatenated Value*.

Table 12-79 through Table 12-81 summarize the register access submodes.

Table 12-79. UART Subconfiguration Mode A Summary

Mode	Condition
MSR_SPR	(UART_EFR[4] = 0x0 or UART_MCR[6] = 0x0)

Copyright © 2024 Texas Instruments Incorporated

Table 12-79. UART Subconfiguration Mode A Summary (continued)

Mode	Condition
TCR_TLR	UART_EFR[4] = 0x1 and UART_MCR[6] = 0x1

Table 12-80. UART Subconfiguration Mode B Summary

Mode	Condition
TCR_TLR	UART_EFR[4] = 0x1 and UART_MCR[6] = 0x1
XOFF	(UART_EFR[4] = 0x0 or UART_MCR[6] = 0x0)

Table 12-81. UART Suboperational Mode Summary

Mode	Condition
MSR_SPR	UART_EFR[4] = 0x0 or UART_MCR[6] = 0x0
TCR_TLR	UART_EFR[4] = 0x1 and UART_MCR[6] = 0x1

12.1.6.3.7.1.3 Registers Available for the Register Access Modes

Table 12-82 lists the names of the register bits in each access register mode. Gray shading indicates that the register does not depend on the register access mode (available in all modes).

Table 12-82. UART Register Access Mode Overview

Address Offset	Registers					
	Configuration Mode A		Configuration Mode B		Operational Mode	
	Read	Write	Read	Write	Read	Write
0x000	UART_DLL	UART_DLL	UART_DLL	UART_DLL	UART_RHR	UART_THR
0x004	UART_DLH	UART_DLH	UART_DLH	UART_DLH	UART_IER_UART	UART_IER_UART
0x008	UART_IIR_UART	UART_FCR	UART_EFR	UART_EFR	UART_IIR_UART	UART_FCR
0x00C	UART_LCR	UART_LCR	UART_LCR	UART_LCR	UART_LCR	UART_LCR
0x010	UART_MCR	UART_MCR	UART_XON1_ADDR1	UART_XON1_ADDR1	UART_MCR	UART_MCR
0x014	UART_LSR_UART	–	UART_XON2_ADDR2	UART_XON2_ADDR2	UART_LSR_UART	–
0x018	UART_MSR / UART_TCR	UART_TCR	UART_TCR / UART_XOFF1	UART_TCR / UART_XOFF1	UART_MSR / UART_TCR	UART_TCR
0x01C	UART_SPR / UART_TLR	UART_SPR / UART_TLR	UART_TLR / UART_XOFF2	UART_TLR / UART_XOFF2	UART_SPR / UART_TLR	UART_SPR / UART_TLR
0x020	UART_MDR1	UART_MDR1	UART_MDR1	UART_MDR1	UART_MDR1	UART_MDR1
0x024	UART_MDR2	UART_MDR2	UART_MDR2	UART_MDR2	UART_MDR2	UART_MDR2
0x028	UART_SFLSR	UART_TXFLL	UART_SFLSR	UART_TXFLL	UART_SFLSR	UART_TXFLL
0x02C	UART_RESUME	UART_TXFLH	UART_RESUME	UART_TXFLH	UART_RESUME	UART_TXFLH
0x030	UART_SFREGL	UART_RXFLL	UART_SFREGL	UART_RXFLL	UART_SFREGL	UART_RXFLL
0x034	UART_SFREGH	UART_RXFLH	UART_SFREGH	UART_RXFLH	UART_SFREGH	UART_RXFLH
0x038	UART_UASR	–	UART_UASR	–	UART_BLR	UART_BLR
0x03C	–	–	–	–	UART_ACREG	UART_ACREG
0x040	UART_SCR	UART_SCR	UART_SCR	UART_SCR	UART_SCR	UART_SCR
0x044	UART_SSR	–	UART_SSR	–	UART_SSR	–
0x048	–	–	–	–	UART_EBLR	UART_EBLR
0x050	UART_MVR	–	UART_MVR	–	UART_MVR	–
0x054	UART_SYSC	UART_SYSC	UART_SYSC	UART_SYSC	UART_SYSC	UART_SYSC
0x058	UART_SYSS	–	UART_SYSS	–	UART_SYSS	–
0x05C	UART_WER	UART_WER	UART_WER	UART_WER	UART_WER	UART_WER
0x060	UART_CFPS	UART_CFPS	UART_CFPS	UART_CFPS	UART_CFPS	UART_CFPS
0x064	UART_RXFIFO_LVL	UART_RXFIFO_LVL	UART_RXFIFO_LVL	UART_RXFIFO_LVL	UART_RXFIFO_LVL	UART_RXFIFO_LVL

Copyright © 2024 Texas Instruments Incorporated

Table 12-82. UART Register Access Mode Overview (continued)

Address Offset	Registers					
	Configuration Mode A		Configuration Mode B		Operational Mode	
	Read	Write	Read	Write	Read	Write
0x068	UART_TXFIFO_LVL	UART_TXFIFO_LVL	UART_TXFIFO_LVL	UART_TXFIFO_LVL	UART_TXFIFO_LVL	UART_TXFIFO_LVL
0x06C	UART_IER2	UART_IER2	UART_IER2	UART_IER2	UART_IER2	UART_IER2
0x070	UART_ISR2	UART_ISR2	UART_ISR2	UART_ISR2	UART_ISR2	UART_ISR2
0x074	UART_FREQ_SEL	UART_FREQ_SEL	UART_FREQ_SEL	UART_FREQ_SEL	UART_FREQ_SEL	UART_FREQ_SEL
0x080	UART_MDR3	UART_MDR3	UART_MDR3	UART_MDR3	UART_MDR3	UART_MDR3
0x084	UART_TX_DMA_THRESHOLD	UART_TX_DMA_THRESHOLD	UART_TX_DMA_THRESHOLD	UART_TX_DMA_THRESHOLD	UART_TX_DMA_THRESHOLD	UART_TX_DMA_THRESHOLD

12.1.6.3.7.2 UART/RS-485/IrDA (SIR, MIR, FIR)/CIR Mode Selection

To select a mode, set the UART_MDR1[2:0] MODE_SELECT bit field (see Table 12-83).

Table 12-83. UART Mode Selection

Value	Mode
0x0:	UART 16× mode
0x1:	SIR mode
0x2:	UART 16× auto-baud
0x3:	UART 13× mode
0x4:	MIR mode
0x5:	FIR mode
0x6:	CIR mode
0x7:	Disable (default state)

MODE_SELECT is effective when the module is in operational mode (see Section 12.1.6.3.7.1, *Register Access Modes*).

To select a RS-485 mode, set the UART_MDR3[4] DIR_EN bit field to 0x1.

12.1.6.3.7.2.1 Registers Available for the UART Function

Only the registers listed in Table 12-84 are used for the UART function.

Table 12-84. UART Mode Register Overview

Address Offset	Registers[(1) (2)]					
	Configuration Mode A		Configuration Mode B		Operational Mode	
	Read	Write	Read	Write	Read	Write
0x000	UART_DLL	UART_DLL	UART_DLL	UART_DLL	UART_RHR	UART_THR
0x004	UART_DLH	UART_DLH	UART_DLH	UART_DLH	UART_IER_UART (UART)	UART_IER_UART (UART)
0x008	UART_IIR_UART	UART_FCR	UART_EFR [4]	UART_EFR [4]	UART_IIR_UART (UART)	UART_FCR (UART)
0x00C	UART_LCR	UART_LCR	UART_LCR	UART_LCR	UART_LCR	UART_LCR
0x010	UART_MCR	UART_MCR	UART_XON1_ADDR1	UART_XON1_ADDR1	UART_MCR	UART_MCR
0x014	UART_LSR_UART (UART)	–	UART_XON2_ADDR2	UART_XON2_ADDR2	UART_LSR_UART (UART)	–
0x018	UART_MSR/ UART_TCR	UART_TCR	UART_XOFF1/ UART_TCR	UART_XOFF1/ UART_TCR	UART_MSR/ UART_TCR	UART_TCR
0x01C	UART_TLR/ UART_SPR	UART_TLR/ UART_SPR	UART_TLR/ UART_XOFF2	UART_TLR/ UART_XOFF2	UART_TLR/ UART_SPR	UART_TLR/ UART_SPR
0x020	UART_MDR1	UART_MDR1 [2-0]	UART_MDR1 [2-0]	UART_MDR1 [2-0]	UART_MDR1 [2-0]	UART_MDR1 [2-0]

Copyright © 2024 Texas Instruments Incorporated

Table 12-84. UART Mode Register Overview (continued)

Address Offset	Registers[(1) (2)]					
	Configuration Mode A		Configuration Mode B		Operational Mode	
	Read	Write	Read	Write	Read	Write
0x024	UART_MDR2	UART_MDR2	UART_MDR2	UART_MDR2	UART_MDR2	UART_MDR2
0x028	–	–	–	–	–	–
0x02C	–	–	–	–	–	–
0x030	–	–	–	–	–	–
0x034	–	–	–	–	–	–
0x038	UART_UASR	–	UART_UASR	–	–	–
0x03C	–	–	–	–	–	–
0x040	UART_SCR	UART_SCR	UART_SCR	UART_SCR	UART_SCR	UART_SCR
0x044	UART_SSR	–	UART_SSR	–	UART_SSR	–
0x048	–	–	–	–	–	–
0x050	UART_MVR	–	UART_MVR	–	UART_MVR	–
0x054	UART_SYSC	UART_SYSC	UART_SYSC	UART_SYSC	UART_SYSC	UART_SYSC
0x058	UART_SYSS	–	UART_SYSS	–	UART_SYSS	–
0x05C	UART_WER	UART_WER	UART_WER	UART_WER	UART_WER	UART_WER
0x060	–	–	–	–	–	–
0x064	UART_RXFIFO_LVL	UART_RXFIFO_LVL	UART_RXFIFO_LVL	UART_RXFIFO_LVL	UART_RXFIFO_LVL	UART_RXFIFO_LVL
0x068	UART_TXFIFO_LVL	UART_TXFIFO_LVL	UART_TXFIFO_LVL	UART_TXFIFO_LVL	UART_TXFIFO_LVL	UART_TXFIFO_LVL
0x06C	UART_IER2	UART_IER2	UART_IER2	UART_IER2	UART_IER2	UART_IER2
0x070	UART_ISR2	UART_ISR2	UART_ISR2	UART_ISR2	UART_ISR2	UART_ISR2
0x074	UART_FREQ_SEL	UART_FREQ_SEL	UART_FREQ_SEL	UART_FREQ_SEL	UART_FREQ_SEL	UART_FREQ_SEL
0x080	UART_MDR3	UART_MDR3	UART_MDR3	UART_MDR3	UART_MDR3	UART_MDR3
0x084	UART_TX_DMA_THRESHOLD	UART_TX_DMA_THRESHOLD	UART_TX_DMA_THRESHOLD	UART_TX_DMA_THRESHOLD	UART_TX_DMA_THRESHOLD	UART_TX_DMA_THRESHOLD

(1) REGISTER_NAME(UART) notation indicates that the register exists for other functions (IrDA or CIR), but fields have different meanings for other functions (described separately in *UART Registers*).
(2) REGISTER_NAME[m:n] notation indicates that only register bits numbered m to n apply to the UART function.

12.1.6.3.7.2.2 Registers Available for the IrDA Function

Only the registers listed in Table 12-85 are used for the IrDA function.

Table 12-85. IrDA Mode Register Overview

Address Offset	Registers[(1) (2)]					
	Configuration Mode A		Configuration Mode B		Operational Mode	
	Read	Write	Read	Write	Read	Write
0x000	UART_DLL	UART_DLL	UART_DLL	UART_DLL	UART_RHR	UART_THR
0x004	UART_DLH	UART_DLH	UART_DLH	UART_DLH	UART_IER_UART (IrDA)	UART_IER_UART (IrDA)
0x008	UART_IIR_UART	UART_FCR	UART_EFR [4]	UART_EFR [4]	UART_IIR_UART (IrDA)	UART_FCR (IrDA)
0x00C	UART_LCR [7]	UART_LCR [7]	UART_LCR [7]	UART_LCR [7]	UART_LCR [7]	UART_LCR [7]
0x010	–	–	UART_XON1_ADDR1	UART_XON1_ADDR1	–	–
0x014	UART_LSR_UART (IrDA)	–	UART_XON2_ADDR2	UART_XON2_ADDR2	UART_LSR_UART (IrDA)	–
0x018	UART_MSR/ UART_TCR	UART_TCR	UART_TCR	UART_TCR	UART_MSR/ UART_TCR	UART_TCR

Copyright © 2024 Texas Instruments Incorporated

Table 12-85. IrDA Mode Register Overview (continued)

Address Offset	Registers[(1) (2)]					
	Configuration Mode A		Configuration Mode B		Operational Mode	
	Read	Write	Read	Write	Read	Write
0x01C	UART_TLR/ UART_SPR	UART_TLR/ UART_SPR	UART_TLR	UART_TLR	UART_TLR/ UART_SPR	UART_TLR/ UART_SPR
0x020	UART_MDR1	UART_MDR1	UART_MDR1	UART_MDR1	UART_MDR1	UART_MDR1
0x024	UART_MDR2	UART_MDR2	UART_MDR2	UART_MDR2	UART_MDR2	UART_MDR2
0x028	UART_SFLSR	UART_TXFLL	UART_SFLSR	UART_TXFLL	UART_SFLSR	UART_TXFLL
0x02C	UART_RESUME	UART_TXFLH	UART_RESUME	UART_TXFLH	UART_RESUME	UART_TXFLH
0x030	UART_SFREGL	UART_RXFLL	UART_SFREGL	UART_RXFLL	UART_SFREGL	UART_RXFLL
0x034	UART_SFREGH	UART_RXFLH	UART_SFREGH	UART_RXFLH	UART_SFREGH	UART_RXFLH
0x038	–	–	–	–	UART_BLR	UART_BLR
0x03C	–	–	–	–	UART_ACREG	UART_ACREG
0x040	UART_SCR	UART_SCR	UART_SCR	UART_SCR	UART_SCR	UART_SCR
0x044	UART_SSR	–	UART_SSR	–	UART_SSR	–
0x048	–	–	–	–	UART_EBLR	UART_EBLR
0x050	UART_MVR	–	UART_MVR	–	UART_MVR	–
0x054	UART_SYSC	UART_SYSC	UART_SYSC	UART_SYSC	UART_SYSC	UART_SYSC
0x058	UART_SYSS	–	UART_SYSS	–	UART_SYSS	–
0x05C	UART_WER [6-4]	UART_WER [6-4]	UART_WER [6-4]	UART_WER [6-4]	UART_WER [6-4]	UART_WER [6-4]
0x060	–	–	–	–	–	–
0x064	UART_RXFIFO_L VL	UART_RXFIFO_L VL	UART_RXFIFO_L VL	UART_RXFIFO_L VL	UART_RXFIFO_L VL	UART_RXFIFO_L VL
0x068	UART_TXFIFO_LV L	UART_TXFIFO_LV L	UART_TXFIFO_LV L	UART_TXFIFO_LV L	UART_TXFIFO_LV L	UART_TXFIFO_LV L
0x06C	UART_IER2	UART_IER2	UART_IER2	UART_IER2	UART_IER2	UART_IER2
0x070	UART_ISR2	UART_ISR2	UART_ISR2	UART_ISR2	UART_ISR2	UART_ISR2
0x074	UART_FREQ_SEL	UART_FREQ_SEL	UART_FREQ_SEL	UART_FREQ_SEL	UART_FREQ_SEL	UART_FREQ_SEL
0x080	UART_MDR3	UART_MDR3	UART_MDR3	UART_MDR3	UART_MDR3	UART_MDR3
0x084	UART_TX_DMA_T HRESHOLD	UART_TX_DMA_T HRESHOLD	UART_TX_DMA_T HRESHOLD	UART_TX_DMA_T HRESHOLD	UART_TX_DMA_T HRESHOLD	UART_TX_DMA_T HRESHOLD

(1) REGISTER_NAME(IrDA) notation indicates that the register exists for other functions (UART or CIR), but fields have different meanings for other functions (described separately in *UART Registers*).
(2) REGISTER_NAME[m:n] notation indicates that only register bits numbered m to n apply to the IrDA function.

12.1.6.3.7.2.3 Registers Available for the CIR Function

Only the registers listed in Table 12-86 are used for the CIR function.

Table 12-86. CIR Mode Register Overview

Address Offset	Registers [(1) (2)]					
	Configuration Mode A		Configuration Mode B		Operational Mode	
	Read	Write	Read	Write	Read	Write
0x000	UART_DLL	UART_DLL	UART_DLL	UART_DLL	–	UART_THR
0x004	UART_DLH	UART_DLH	UART_DLH	UART_DLH	UART_IER_UART (CIR)	UART_IER_UART (CIR)
0x008	UART_IIR_UART	UART_FCR	UART_EFR	UART_EFR	UART_IIR_UART (CIR)	UART_FCR (CIR)
0x00C	UART_LCR	UART_LCR [7]	UART_LCR [7]	UART_LCR [7]	UART_LCR [7]	UART_LCR [7]
0x010	–	–	–	–	–	–
0x014	UART_LSR_UART (CIR)	–	–	–	UART_LSR_UART (CIR)	–

Copyright © 2024 Texas Instruments Incorporated

Table 12-86. CIR Mode Register Overview (continued)

Address Offset	Registers (1) (2)					
	Configuration Mode A		Configuration Mode B		Operational Mode	
	Read	Write	Read	Write	Read	Write
0x018	UART_MSR/ UART_TCR	UART_TCR	UART_TCR	UART_TCR	UART_MSR/ UART_TCR	UART_TCR
0x01C	UART_TLR/ UART_SPR	UART_TLR/ UART_SPR	UART_TLR	UART_TLR	UART_TLR/ UART_SPR	UART_TLR/ UART_SPR
0x020	UART_MDR1 [3-0]	UART_MDR1 [3-0]	UART_MDR1 [3-0]	UART_MDR1 [3-0]	UART_MDR1 [3-0]	UART_MDR1 [3-0]
0x024	UART_MDR2	UART_MDR2	UART_MDR2	UART_MDR2	UART_MDR2	UART_MDR2
0x028	–	–	–	–	–	–
0x02C	UART_RESUME	–	UART_RESUME	–	UART_RESUME	–
0x030	–	–	–	–	–	–
0x034	–	–	–	–	–	–
0x038	–	–	–	–	–	–
0x03C	–	–	–	–	UART_ACREG	UART_ACREG
0x040	UART_SCR	UART_SCR	UART_SCR	UART_SCR	UART_SCR	UART_SCR
0x044	UART_SSR	–	UART_SSR	–	UART_SSR	–
0x048	–	–	–	–	UART_EBLR	UART_EBLR
0x050	UART_MVR	–	UART_MVR	–	UART_MVR	–
0x054	UART_SYSC	UART_SYSC	UART_SYSC	UART_SYSC	UART_SYSC	UART_SYSC
0x058	UART_SYSS	–	UART_SYSS	–	UART_SYSS	–
0x05C	UART_WER [6-4]	UART_WER [6-4]	UART_WER [6-4]	UART_WER [6-4]	UART_WER [6-4]	UART_WER [6-4]
0x060	UART_CFPS	UART_CFPS	UART_CFPS	UART_CFPS	UART_CFPS	UART_CFPS
0x064	UART_RXFIFO_L VL	UART_RXFIFO_L VL	UART_RXFIFO_L VL	UART_RXFIFO_L VL	UART_RXFIFO_L VL	UART_RXFIFO_L VL
0x068	UART_TXFIFO_LV L	UART_TXFIFO_LV L	UART_TXFIFO_LV L	UART_TXFIFO_LV L	UART_TXFIFO_LV L	UART_TXFIFO_LV L
0x06C	UART_IER2	UART_IER2	UART_IER2	UART_IER2	UART_IER2	UART_IER2
0x070	UART_ISR2	UART_ISR2	UART_ISR2	UART_ISR2	UART_ISR2	UART_ISR2
0x074	UART_FREQ_SEL	UART_FREQ_SEL	UART_FREQ_SEL	UART_FREQ_SEL	UART_FREQ_SEL	UART_FREQ_SEL
0x080	UART_MDR3	UART_MDR3	UART_MDR3	UART_MDR3	UART_MDR3	UART_MDR3
0x084	UART_TX_DMA_T HRESHOLD	UART_TX_DMA_T HRESHOLD	UART_TX_DMA_T HRESHOLD	UART_TX_DMA_T HRESHOLD	UART_TX_DMA_T HRESHOLD	UART_TX_DMA_T HRESHOLD

(1) REGISTER_NAME(CIR) notation indicates that the register exists for other functions (IrDA or UART), but fields have different meanings for other functions (described separately in *UART Registers*).
(2) REGISTER_NAME[m:n] notation indicates that only register bits numbered m to n apply to the CIR function.

12.1.6.3.8 UART Protocol Formatting

12.1.6.3.8.1 UART Mode

12.1.6.3.8.1.1 UART Clock Generation: Baud Rate Generation

The UART function contains a programmable baud generator and a set of fixed dividers that divide the 48-MHz clock input down to the expected baud rate.

Figure 12-57 shows the baud rate generator and associated controls.

Copyright © 2024 Texas Instruments Incorporated

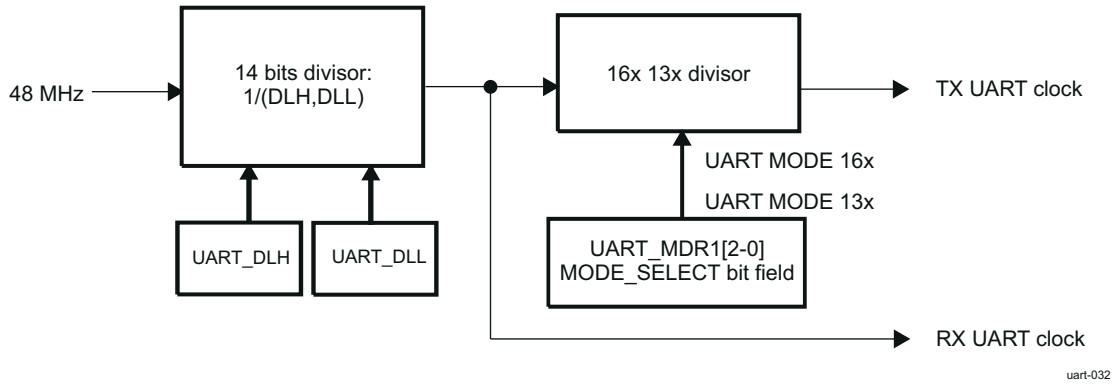


Figure 12-57. UART Baud Rate Generation

> **CAUTION**
>
> Before initializing or modifying clock parameter controls (UART_DLH, UART_DLL), UART_MDR1[2-0] MODE_SELECT = DISABLE must be set to 0x7. Failure to observe this rule can result in unpredictable module behavior.

12.1.6.3.8.1.2 Choosing the Appropriate Divisor Value

Two divisor values are:

- UART 16× mode: Divisor value = Operating frequency / (16× baud rate)
- UART 13× mode: Divisor value = Operating frequency / (13× baud rate)

Table 12-87. UART Baud Rate Settings (48-MHz Clock)

Baud Rate	Baud Multiple	DLH, DLL (Decimal)	DLH, DLL (Hex)	Actual Baud Rate	Error (%)
0.3 kbps	16x	10000	0x27, 0x10	0.3 kbps	0
0.6 kbps	16x	5000	0x13, 0x88	0.6 kbps	0
1.2 kbps	16x	2500	0x09, 0xC4	1.2 kbps	0
2.4 kbps	16x	1250	0x04, 0xE2	2.4 kbps	0
4.8 kbps	16x	625	0x02, 0x71	4.8 kbps	0
9.6 kbps	16x	312	0x01, 0x39	9.6153 kbps	+0.16
14.4 kbps	16x	208	0x00, 0xD0	14.423 kbps	+0.16
19.2 kbps	16x	156	0x00, 0x9C	19.231 kbps	+0.16
28.8 kbps	16x	104	0x00, 0x68	28.846 kbps	+0.16
38.4 kbps	16x	78	0x00, 0x4E	38.462 kbps	+0.16
57.6 kbps	16x	52	0x00, 0x34	57.692 kbps	+0.16
115.2 kbps	16x	26	0x00, 0x1A	115.38 kbps	+0.16
230.4 kbps	16x	13	0x00, 0x0D	230.77 kbps	+0.16
460.8 kbps	13x	8	0x00, 0x08	461.54 kbps	+0.16
921.6 kbps	13x	4	0x00, 0x04	923.08 kbps	+0.16
1.843 Mbps	13x	2	0x00, 0x02	1.846 Mbps	+0.16
3.6884 Mbps	13x	1	0x00, 0x01	3.6923 Mbps	+0.16

12.1.6.3.8.1.3 UART Data Formatting

The UART can use hardware flow control to manage transmission and reception. Hardware flow control significantly reduces software overhead and increases system efficiency by automatically controlling serial data flow using the RTS output and CTS input signals.

The UART is enhanced with the autobauding function. In control mode, autobauding lets the speed, the number of bits per character, and the parity selected be set automatically.

Copyright © 2024 Texas Instruments Incorporated

12.1.6.3.8.1.3.1 Frame Formatting

When autobauding is not used, frame format attributes must be defined in the UART_LCR register.

Character length is specified using the UART_LCR[1-0] CHAR_LENGTH bit field.

The number of stop-bits is specified using the UART_LCR[2] NB_STOP bit.

The parity bit is programmed using the UART_LCR[5-3] PARITY_EN, UART_LCR[5-3] PARITY_TYPE_1, and UART_LCR[5-3] PARITY_TYPE_2 bit fields (see Table 12-88).

Table 12-88. UART Parity Bit Encoding

PARITY_EN	PARITY_TYPE_1	PARITY_TYPE_2	Parity
0	N/A	N/A	No parity
1	0	0	Odd parity
1	1	0	Even parity
1	0	1	Forced 1
1	1	1	Forced 0

12.1.6.3.8.1.3.2 Hardware Flow Control

Hardware flow control is composed of auto-CTS and auto-RTS. Auto-CTS and auto-RTS can be enabled and disabled independently by programming the UART_EFR[7] AUTO_CTS_EN and UART_EFR[6] AUTO_RTS_EN bit fields, respectively.

With auto-CTS, CTS signal must be active before the module can transmit data.

Auto-RTS activates the RTS output only when there is enough room in the RX FIFO to receive data. It deactivates the RTS output when the RX FIFO is sufficiently full. The HALT and RESTORE trigger levels in the UART_TCR register determine the levels at which RTS is activated and deactivated.

If auto-CTS and auto-RTS are enabled, data transmission does not occur unless the RX FIFO has empty space. Thus, overrun errors are eliminated during hardware flow control. If auto-CTS and auto-RTS are not enabled, overrun errors occur if the transmit data rate exceeds the RX FIFO latency.

- Auto-RTS:

  Auto-RTS data flow control originates in the receiver block. The RX FIFO trigger levels used in auto-RTS are stored in the UART_TCR register. RTS is active if the RX FIFO level is below the HALT trigger level in the UART_TCR[3-0] RX_FIFO_TRIG_HALT bit field. When the RX FIFO HALT trigger level is reached, RTS is deasserted. The sending device (for example, another UART) can send an additional byte after the trigger level is reached because it may not recognize the deassertion of RTS until it begins sending the additional byte.

  RTS is automatically reasserted when the RX FIFO reaches the RESUME trigger level programmed by the UART_TCR[7-4] RX_FIFO_TRIG_START bit field. This reassertion requests the sending device to resume transmission.

  In this case, RTS is an active-low signal.
- Auto-CTS:

  The transmitter circuitry checks CTS before sending the next data byte. When CTS is active, the transmitter sends the next byte. To stop the transmitter from sending the next byte, CTS must be deasserted before the middle of the last stop-bit currently sent.

  The auto-CTS function reduces interrupts to the host system. When auto-CTS flow control is enabled, the CTS state changes do not have to trigger host interrupts because the device automatically controls its own transmitter. Without auto-CTS, the transmitter sends any data present in the transmit FIFO, and a receiver overrun error can result.

  In this case, CTS is an active-low signal.

Copyright © 2024 Texas Instruments Incorporated

12.1.6.3.8.1.3.3 Software Flow Control

Software flow control is enabled through the enhanced feature register (UART_EFR) and the modem control register (UART_MCR). Different combinations of software flow control can be enabled by setting different combinations of the UART_EFR[3-0] bit field (see Table 12-89).

Two other enhanced features relate to software flow control:

- XON-any function (UART_MCR[5] XON_EN): Operation resumes after receiving any character after the XOFF character is recognized. If special character detect is enabled and special character is received after XOFF1, it does not resume transmission. The special character is stored in the RX FIFO.

Note

The XON-any character is written into the RX FIFO even if it is a software flow character.

- Special character (UART_EFR[5] SPECIAL_CHAR_DETEC T): Incoming data is compared to XOFF2. When the special character is detected, the XOFF interrupt (UART_IIR_UART) is set, but it does not halt transmission. The XOFF interrupt is cleared by a read of UART_IIR_UART. The special character is transferred to the RX FIFO. Special character does not work with XON2, XOFF2, or sequential XOFFs.

Table 12-89. UART_EFR[3:0] Software Flow Control Options

Bit 3	Bit 2	Bit 1	Bit 0	TX, RX Software Flow Controls
0	0	X	X	No transmit flow control
1	0	X	X	Transmit XON1, XOFF1
0	1	X	X	Transmit XON2, XOFF2
1	1	X	X	Transmit XON1, XON2: XOFF1, XOFF2[(1)]
X	X	0	0	No receive flow control
X	X	1	0	Receiver compares XON1, XOFF1
X	X	0	1	Receiver compares XON2, XOFF2
X	X	1	1	Receiver compares XON1, XON2: XOFF1, XOFF2[(1)]

(1) In these cases, the XON1 and XON2 characters or the XOFF1 and XOFF2 characters must be transmitted/received sequentially with XON1/XOFF1 followed by XON2/XOFF2.
XON1 is defined in the UART_XON1_ADDR1[7-0] XON_WORD1 bit field. XON2 is defined in the UART_XON2_ADDR2[7-0] XON_WORD2 bit field.
XOFF1 is defined in the UART_XOFF1[7-0] XOFF_WORD1 bit field. XOFF2 is defined in the UART_XOFF2[7-0] XOFF_WORD2 bit field.

12.1.6.3.8.1.3.3.1 Receive (RX)

When software flow control operation is enabled, the UART compares incoming data with XOFF1/2 programmed characters (in certain cases, XOFF1 and XOFF2 must be received sequentially). When the correct XOFF characters are received, transmission stops after transmission of the current character completes. Detection of XOFF also sets the UART_IIR_UART[4] bit (if enabled by UART_IER_UART[5]) and causes the interrupt line to go low.

To resume transmission, an XON1/2 character must be received (in certain cases, XON1 and XON2 must be received sequentially). When the correct XON characters are received, the UART_IIR_UART[4] bit is cleared and the XOFF interrupt disappears.

Note

When a parity, framing, or break error occurs while receiving a software flow control character, this character is treated as normal data and is written to the RX FIFO.

When XON-any and special character detect are disabled and software flow control is enabled, no valid XON or XOFF characters are written to the RX FIFO. For example, when UART_EFR[1-0] = 0x2, if XON1 and XOFF1 characters are received, they are not written to the RX FIFO.

Copyright © 2024 Texas Instruments Incorporated

When pairs of software flow characters are programmed to be received sequentially (UART_EFR[1-0] = 0x3), the software flow characters are not written to the RX FIFO if they are received sequentially. However, received XON1/XOFF1 characters must be written to the RX FIFO if the subsequent character is not XON2/XOFF2.

12.1.6.3.8.1.3.3.2 Transmit (TX)

Two XOFF1 characters are transmitted when the RX FIFO passes the trigger level programmed by UART_TCR[3-0] RX_FIFO_TRIG_HALT. As soon as the RX FIFO reaches the trigger level programmed by UART_TCR[7-4] RX_FIFO_TRIG_START, two XON1 characters are sent, so the data transfer recovers.

Note

If software flow control is disabled after an XOFF character is sent, the module transmits XON characters automatically to enable normal transmission.

The transmission of XOFF(s)/XON(s) follows the same protocol as transmission of an ordinary byte from the TX FIFO. This means that even if the word length is 5, 6, or 7 characters, the 5, 6, or 7 LSBs of XOFF1/2 and XON1/2 are transmitted. The 5, 6, or 7 bits of a character are seldom transmitted, but this function is included to maintain compatibility with earlier designs.

It is assumed that software flow control and hardware flow control are never enabled simultaneously.

12.1.6.3.8.1.3.4 Autobauding Modes

In autobauding mode, the UART can extract transfer characteristics (speed, length, and parity) from an "at" (AT) command (ASCII code). These characteristics are used to receive data after an AT and to send data.

The following AT commands are valid:

AT	DATA	<CR>
at	DATA	<CR>
A/		
a/		

A line break during the acquisition of the sequence AT is not recognized, and an echo function is not implemented in hardware.

A/ and a/ are not used to extract characteristics, but they must be recognized because of their special meaning. A/ or a/ is used to instruct the software to repeat the last received AT command; therefore, an a/ always follows an AT, and transfer characteristics are not expected to change between an AT and an a/.

When a valid AT is received, AT and all subsequent data, including the final <CR> (0x0D), are saved to the RX FIFO. The autobaud state-machine waits for the next valid AT command. If an a/ (A/) is received, the a/ (A/) is saved in the RX FIFO and the state-machine waits for the next valid AT command.

On the first successful detection of the baud rate, the UART activates an interrupt to signify that the AT (upper or lower case) sequence is detected. The UART_UASR register reflects the correct settings for the baud rate detected. Interrupt activity can continue in this fashion when a subsequent character is received. Therefore, it is recommended that the software enable the RHR interrupt when using the autobaud mode.

The following settings are detected in autobaud mode with a module clock of 48 MHz:

- Speed:
  - 115.2K baud
  - 57.6K baud
  - 38.4K baud
  - 28.8K baud
  - 19.2K baud
  - 14.4K baud
  - 9.6K baud
  - 4.8K baud
  - 2.4K baud

Copyright © 2024 Texas Instruments Incorporated

  - 1.2K baud
- Length: 7 or 8 bits
- Parity: Odd, even, or space

Note

The combination of 7-bit character plus space parity is not supported.

Autobauding mode is selected when the UART_MDR1[2-0] MODE_SELECT bit field is set to 0x2. In UART autobauding mode, UART_DLL, UART_DLH, and UART_LCR[5-0] bit field settings are not used; instead, the UART_UASR register is updated with the configuration detected by the autobauding logic.

UASR Autobauding Status Register Use

This register is used to set up transmission according to the characteristics of the previous reception instead of the UART_LCR, UART_DLL, and UART_DLH registers when the UART is in autobauding mode.

To reset the autobauding hardware (to start a new AT detection) or to set the UART in standard mode (no autobaud), the UART_MDR1[2-0] MODE_SELECT bit field must be set to reset state (0x7) and then to the UART in autobauding mode (0x2) or to the UART in standard mode (0x0).

Use limitation:

- Only 7- and 8-bit characters (5- and 6-bit not supported)
- 7-bit character with space parity not supported
- Baud rate between 1200 and 115.2 bps (10 possibilities)

12.1.6.3.8.1.3.5 Error Detection

When the UART_LSR_UART register is read, the UART_LSR_UART[4:2] bit field reflects the error bits (BI: break condition, FE: framing error, PE: parity error) of the character at the top of the RX FIFO (the next character to be read). Therefore, reading the UART_LSR_UART register and then reading the UART_RHR register identifies errors in a character.

Reading the UART_RHR register updates the BI, FE, and PE bits (see Table 12-73 for the UART mode interrupts).

The UART_LSR_UART[7] RX_FIFO_STS bit is set when there is an error in the RX FIFO and is cleared only when no errors remain in the RX FIFO.

Note

Reading the UART_LSR_UART register does not cause an increment of the RX FIFO read pointer. The RX FIFO read pointer is incremented by reading the UART_RHR register.

Reading the UART_LSR_UART register clears the OE bit if it is set (see Table 12-73 for the UART mode interrupts).

12.1.6.3.8.1.3.6 Overrun During Receive

Overrun during receive occurs if the RX state-machine tries to write data into the RX FIFO when it is already full. When overrun occurs, the device interrupts the Host CPU with the UART_IIR_UART[5-1] IT_TYPE bit field set to 0x3 (receiver line status error) and discards the remaining portion of the frame.

Overrun also causes an internal flag to be set, which disables further reception. Before the next frame can be received, the Host CPU must:

- Reset the RX FIFO.
- Read the UART_RESUME register, which clears the internal flag.

Copyright © 2024 Texas Instruments Incorporated

12.1.6.3.8.1.3.7 Time-Out and Break Conditions

12.1.6.3.8.1.3.7.1 Time-Out Counter

An RX idle condition is detected when the receiver line (RX) is high for a time that equals 4x the programmed word length + 12 bits or manually configured amount of baud clocks, if a value other zero is set in the timeout register. RX is sampled midway through each bit.

For sleep mode, the counter is reset when there is activity on RX.

There are two modes of operation:

- In default operation on the UART_EFR2[6] TIMEOUT_BEHAVE is set to 0. For the time-out interrupt, the counter counts only when there is data in the RX FIFO, and the count is reset when there is activity on RX or when the UART_RHR register is read.
- Optionally, for choose to enable the timeout counter even if no character has been received by setting UART_EFR2[6] TIMEOUT_BEHAVE bit. This will generate periodic interrupts if the RX line remains idle. In this mode the counter will auto-reset when a timeout has been reached. Reading the UART_IIR_UART will clear the interrupt, but not the counter.

12.1.6.3.8.1.3.7.2 Break Condition

When a break condition occurs, TX is pulled low. A break condition is activated by setting the UART_LCR[6] BREAK_EN bit. The break condition is not aligned on word stream (a break condition can occur in the middle of a character). The only way to send a break condition on a full character is:

1. Reset the TX FIFO (if enabled).
2. Wait for the transmit shift register to empty (the UART_LSR_UART[6] TX_SR_E bit is set to 1).
3. Take a guard time according to stop-bit definition.
4. Set the BREAK_EN bit to 1.

The break condition is asserted while the BREAK_EN bit is set to 1.

The time-out counter and break condition apply only to UART modem operation and not to IrDA/CIR mode operation.

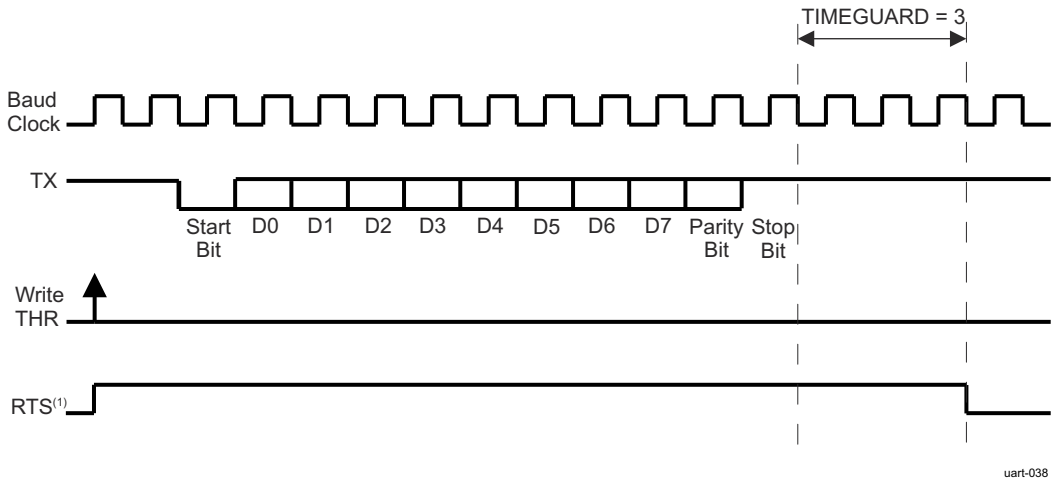
12.1.6.3.8.2 RS-485 Mode

12.1.6.3.8.2.1 RS-485 External Transceiver Direction Control

The UART_MDR3[4] DIR_EN bit enables hardware control over an external transceiver to support RS-485. The direction signal comes across the DIR port. The direction polarity is controlled by the UART_MDR3[3] DIR_POL bit. The direction is determined by the hardware monitoring the TX FIFO and the TX shift register. When both are empty the transceiver is set to RX. There is a guard band delay counter of 3 bit clock cycles after the TX shift register is going empty to allow time for the stop bit to transition through the transceiver before a direction change to receive might be applied.

Figure 12-58 shows the direction control.

Copyright © 2024 Texas Instruments Incorporated



(1) Assumes DIR_POL = 1

Figure 12-58. RS-485 External Transceiver Direction Control

12.1.6.3.8.3 IrDA Mode

12.1.6.3.8.3.1 IrDA Clock Generation: Baud Generator

The IrDA function contains a programmable baud generator and a set of fixed dividers that divide the 48-MHz clock input down to the expected baud rate.

Figure 12-59 shows the baud rate generator and associated controls.

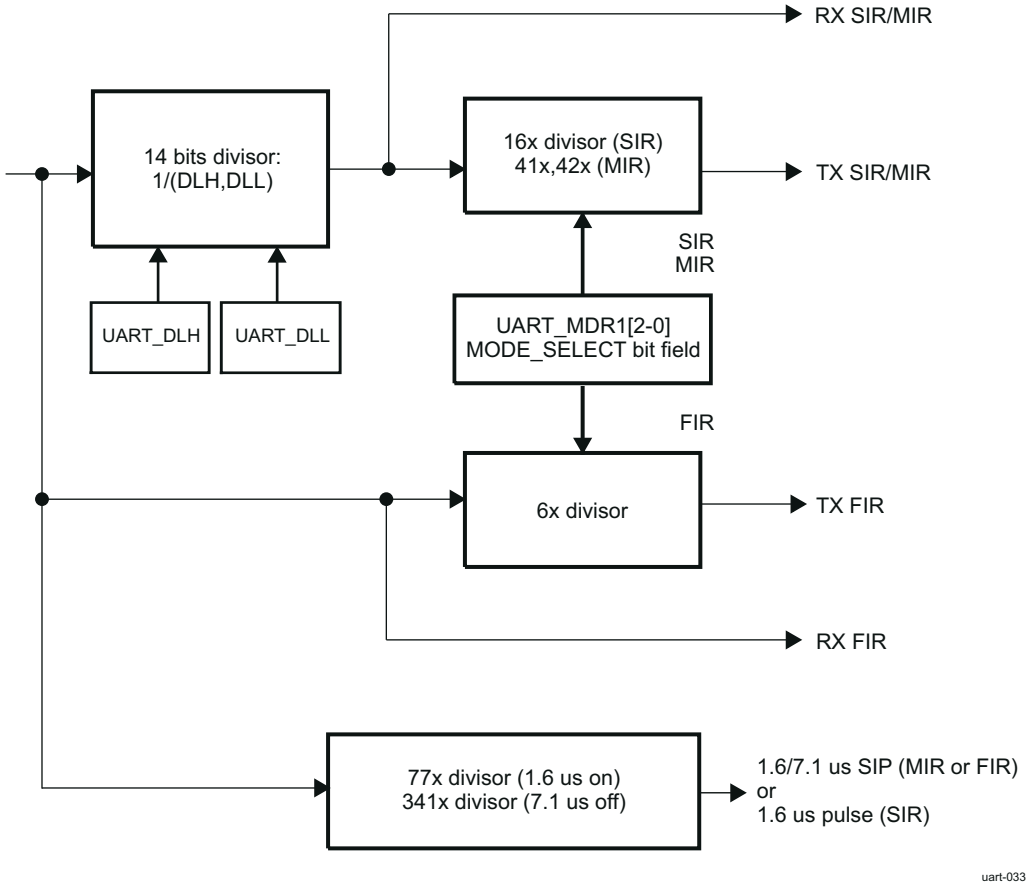


Figure 12-59. IrDA Baud Rate Generator

Copyright © 2024 Texas Instruments Incorporated

CAUTION

Before initializing or modifying clock parameter controls (UART_DLH, UART_DLL), MODE_SELECT=DISABLE (UART_MDR1[2-0] MODE_SELECT) must be set to 0x7). Failure to observe this rule can result in unpredictable module behavior.

12.1.6.3.8.3.2 Choosing the Appropriate Divisor Value

Three divisor values are:

- SIR mode: Divisor value = Operating frequency/(16× baud rate)
- MIR mode: Divisor value = Operating frequency/(41×/42× baud rate)
- FIR mode: Divisor value = None

Table 12-90 lists the IrDA baud rate settings.

Table 12-90. IrDA Baud Rate Settings

Baud Rate	IR Mode	Baud Multiple	Encoding	DLH, DLL (Decimal)	Actual Baud Rate	Error (%)	Source Jitter (%)	Pulse Duration
2.4 kbps	SIR	16x	3/16	1250	2.4 kbps	0	0	78.1 µs
9.6 kbps	SIR	16x	3/16	312	9.6153 kbps	+0.16	0	19.5 µs
19.2 kbps	SIR	16x	3/16	156	19.231 kbps	+0.16	0	9.75 µs
38.4 kbps	SIR	16x	3/16	78	38.462 kbps	+0.16	0	4.87 µs
57.6 kbps	SIR	16x	3/16	52	57.692 kbps	+0.16	0	3.25 µs
115.2 kbps	SIR	16x	3/16	26	115.38 kbps	+0.16	0	1.62 µs
0.576 Mbps	MIR	41×/42×	1/4	2	0.5756 Mbps(1)	0	+1.63/-0.80	416 ns
1.152 Mbps	MIR	41×/42×	1/4	1	1.1511 Mbps(1)	0	+1.63/-0.80	208 ns
4 Mbps	FIR	6×	4 PPM	–	4 Mbps	0	0	125 ns

(1) Average value

Note

Baud rate error and source jitter table values do not include 48-MHz reference clock error and jitter.

12.1.6.3.8.3.3 IrDA Data Formatting

The methods described in this section apply to all IrDA modes (SIR, MIR, and FIR).

12.1.6.3.8.3.3.1 IR RX Polarity Control

The UART_MDR2[6] IRRXINVERT bit provides the flexibility to invert the RX pin in the UART to ensure that the protocol at the output of the transceiver has the same polarity at module level. By default, the RX pin is inverted because most transceivers invert the IR receive pin.

12.1.6.3.8.3.3.2 IrDA Reception Control

The module can transmit and receive data, but when the device is transmitting, the IR RX circuitry is automatically disabled by hardware.

Operation of the RX input can be disabled by the UART_ACREG[5] DIS_IR_RX bit.

12.1.6.3.8.3.3.3 IR Address Checking

In all IR modes, when address checking is enabled, only frames intended for the device are written to the RX FIFO. This restriction avoids receiving frames not meant for this device in a multipoint infrared environment. It is possible to program two frame addresses that the UART IrDA receives, with the UART_XON1_ADDR1[7-0] XON_WORD1 and UART_XON2_ADDR2[7-0] XON_WORD2 bit fields.

Setting the UART_EFR[0] bit to 1 selects address1 checking. Setting the UART_EFR[1] bit to 1 selects address2 checking. Setting the UART_EFR[1-0] bit field to 0 disables all address checking operations. If both bits are set, the incoming frame is checked for private and public addresses.

Copyright © 2024 Texas Instruments Incorporated

If address checking is disabled, all received frames write to the RX FIFO.

12.1.6.3.8.3.3.4 Frame Closing

A transmission frame can be terminated in two ways:

- Frame-length method: Set the UART_MDR1[7] FRAME_END_MODE bit to 0. The Host CPU writes the value of the frame length to the UART_TXFLH and UART_TXFLL registers. The device automatically attaches end flags to the frame when the number of bytes transmitted equals the value of the frame length.
- Set-EOT bit method: Set the UART_MDR1[7] FRAME_END_MODE bit to 1. The Host CPU writes 1 to the UART_ACREG[0] EOT bit just before it writes the last byte to the TX FIFO. When the Host CPU writes the last byte to the TX FIFO, the device internally sets the tag bit for that character in the TX FIFO. As the TX state-machine reads data from the TX FIFO, it uses this tag-bit information to attach end flags and correctly terminate the frame.

12.1.6.3.8.3.3.5 Store and Controlled Transmission

In store and controlled transmission (SCT) mode, the Host CPU starts writing data to the TX FIFO. Then, after writing a part of a frame (for a bigger frame) or an entire frame (a small frame; that is, a supervisory frame), the Host CPU writes 1 to the UART_ACREG[2] SCTX_EN bit (deferred TX start) to start transmission.

SCT mode is enabled by setting the UART_MDR1[5] SCT bit to 1. This transmission method differs from normal mode, in which data transmission starts immediately after data is written to the TX FIFO. SCT mode is useful for sending short frames without TX underrun.

12.1.6.3.8.3.3.6 Error Detection

When the UART_LSR_UART register is read, the UART_LSR_UART[4-2] bit field reflects the error bits [FL, CRC, ABORT] of the frame at the top of the STATUS FIFO (the next frame status to be read).

The error is triggered by an interrupt (for IrDA mode interrupts, see Table 12-74). The STATUS FIFO must be read until empty (a maximum of eight reads is required).

12.1.6.3.8.3.3.7 Underrun During Transmission

Underrun during transmission occurs when the TX FIFO is empty before the end of the frame is transmitted. When underrun occurs, the device closes the frame with end flags but attaches an incorrect CRC value. The receiving device detects a CRC error and discards the frame; it can then ask for a retransmission.

Underrun also causes an internal flag to be set, which disables additional transmissions. Before the next frame can be transmitted, the Host CPU must:

- Reset the TX FIFO.
- Read the UART_RESUME register, which clears the internal flag.

This function can be disabled by the UART_ACREG[4] DIS_TX_UNDERRUN bit, compensated by the extension of the stop-bit in transmission if the TX FIFO is empty.

12.1.6.3.8.3.3.8 Overrun During Receive

Overrun during receive for the IrDA mode has the same function as that for the UART mode (see Section 12.1.6.3.8.1.3.6, *Overrun During Receive*).

12.1.6.3.8.3.3.9 Status FIFO

In IrDA modes, a status FIFO records the received frame status. When a complete frame is received, the length of the frame and the error bits associated with the frame are written to the status FIFO.

Reading the UART_SFREGH[3-0] MSB and UART_SFREGL[3-0] (LSB) bit fields obtains the frame length. The frame error status is read in the UART_SFLSR register. Reading the UART_SFLSR register increments the status FIFO read pointer. Because the status FIFO is eight entries deep, it can hold the status of eight frames.

The Host CPU uses the frame-length information to locate the frame boundary in the received frame data. The Host CPU can screen bad frames using the error status information and can later request the sender to resend only the bad frames.

Copyright © 2024 Texas Instruments Incorporated

This status FIFO can be used effectively in DMA mode because the Host CPU must be interrupted only when the programmed status FIFO trigger level is reached, not each time a frame is received.

12.1.6.3.8.3.3.10 Multi-drop Parity Mode with Address Match

Multi-drop mode is enabled in the UART_EFR2 register.

Address matching mode is only available with 8 bit character length setting. UART_LCR[1-0] CHAR_LENGTH bit fields should always be set to 0x11 (8 bits) prior to enabling the feature.

This mode allows the transmitter to send data on a line where multiple receivers are connected, when supported. In this mode, a set parity bit is used to mark an address, and a parity of 0 denotes data.

This setting affects how the parity is generated. Writing a 0x1 into the UART_ECR[0] A_MULTIDROP bit will set the parity bit for the next byte to be sent, which will then be considered an address, for sending a data frame, the UART_ECR[0] A_MULTIDROP bit has to be cleared.

On reception if the feature is enabled by setting the UART_EFR2[2] MULTIDROP bit to 0x1 incoming frames with parity set to 0x1 are treated as address frames and with parity set to 0x0 as data frames. The receiver will drop all data frames until a matching address frame was found.

The matching address is determined by the values set in UART_MAR, UART_MMR and UART_MBR registers and the value set in UART_EFR2[7] BROADCAST bit.

Table 12-91 summarizes the operation of address matching based on the mentioned values.

Table 12-91. Details of address matching

Received frame	Received parity	Frame type	UART_MAR	UART_MMR	UART_MBR	UART_EFR2[7] BROADCAST	Operation of receiver	Address matching
0xXX(2)	0	DATA	X(1)	X(1)	0xXX(2)	X(1)	Drops data until matching address found	N/A
0xXX(2)	1	ADDRESS	0xXX(2)	0x00	0xXX(2)	0	Matches any address	Yes
0xEF	1	ADDRESS	0xXX(2)	0xXX(2)	0xEF	1	Matches broadcast address	Yes
0x1A	1	ADDRESS	0x1A	0xFF	0xXX(2)	0	Single address match	Yes
0xF5	1	ADDRESS	0xF3	0xF9	0xXX(2)	0	Group address match	Yes

(1) X indicates a do not care bit value
(2) 0xXX indicates a do not care 8 bit hexadecimal value

The possible values for matching address can be calculated in the following way:

- Single and Group addresses can be formed by masking the UART_MAR registers value with the value set in the UART_MMR register, bits set to 0x0 in the UART_MMR register result in do not care values.
- Broadcast addresses can be set in the UART_MBR register if broadcast address is enabled in the UART_EFR2[7] BROADCAST bit, the module will match on received address frames containing the broadcast address.
- For more details, see example below:
  - UART_MAR: 0xF3, UART_MMR: 0xF9, UART_MBR: 0xFF
  - Single and Group addresses: 0xF1, 0xF3, 0xF5, 0xF7
  - Broadcast addresses: 0xFF

If an address match occurred the matching address value can be obtained from the UART_RHR register in the following way:

Copyright © 2024 Texas Instruments Incorporated

- If the FIFO is disabled or the threshold is set to 0x1, the matching address can be directly read from UART_RHR as the FIFO will not be overwritten.
- If the FIFO is enabled or the threshold is greater than 0x1, the matching address will be the latest frame in the FIFO with a parity error bit set.

For received data, the parity error bit in the UART_LSR_UART register is set when a bit with a parity of 0x1 is received indicating an address frame and the received address matches based on the values of UART_MAR, UART_MMR, UART_MBR and UART_EFR2[2] MULTIDROP bit.

In Multi-drop mode no parity is used, as the parity bit is used to differentiate address and data frames. The parity error bit is used for indicating an address match.

For enabling the interrupt generation for address matching UART_IER_UART[2] LINE_STS_IT bit has to be set to 0x1.

An interrupt for the matching address can be identified by reading the UART_IIR_UART[5-1] IT_TYPE bit fields, a value of 0x00011 indicates a receiver line status error. After the UART_LSR_UART[2] RX_PE bit has to be read, a value of 0x1 indicates that an address match occurred. The reception of a frame is indicated with a value of 0x1 in the UART_LSR_UART[0] RX_FIFO_E bit as the matching value is written into the FIFO regardless of the frame type (data or address). UART_LSR_UART[7] RX_FIFO_STS bit will also be set to 0x1 as the parity error bit is used to indicate a matching address.

Note that the operation of the UART_LSR_UART[2] RX_PE bit depends on the value set in UART_EFR2[2] MULTIDROP bit. If UART_EFR2[2] MULTIDROP bit is set to 0x0, UART_LSR_UART[2] RX_PE bit is used to indicate a received parity error. If UART_EFR2[2] MULTIDROP bit is set to 0x1, the receiver is in Multi-drop Address Match mode, thus the value in UART_LSR_UART[2] RX_PE bit is used to indicate an address match.

The interrupt is cleared the same way in both operation modes: reading the UART_LSR_UART register updates the values.

This feature is available in UART and synchronous modes. The ISO7816 has not defined Multidrop Parity Mode, so the feature should be left off.

12.1.6.3.8.3.3.11 Time-guard

The time-guard feature enables the UART interface to operate with slow remote devices.

When set, it will insert a number of idle states between transmitting two characters, the length of which can be set in the UART_TIMEGUARD register. The value in the register defines the number of baud clocks of idle period to insert.

This idle state essentially acts like a long stop bit. In UART and synchronous modes, a Timeguard is added in addition to the stop bit. In ISO7816 there is a waiting period rather than an actual stop bit. Software should set 1-2 or more Timeguard cycles according to the protocol used and the card requirements.

12.1.6.3.8.3.4 SIR Mode Data Formatting

This section provides specific instructions for SIR mode programming.

12.1.6.3.8.3.4.1 Abort Sequence

The transmitter can prematurely close a frame (abort) by sending the sequence 0x7DC1. The abort pattern closes the frame without a CRC field or an ending flag.

A transmission frame can be aborted by setting the UART_ACREG[1] ABORT_EN bit to 1. When this bit is set to 1, 0x7D and 0xC1 are transmitted and the frame is not terminated with CRC or stop flags.

When a 0x7D character followed immediately by a 0xC1 character is received without transparency, the receiver treats a frame as an aborted frame.

Copyright © 2024 Texas Instruments Incorporated

CAUTION

When the TX FIFO is not empty and the UART_MDR1[5] SCT bit is set to 1, the UART IrDA starts a new transfer with data of a previous frame when the aborted frame is sent. Therefore, the TX FIFO must be reset before sending an aborted frame.

12.1.6.3.8.3.4.2 Pulse Shaping

SIR mode supports the 3/16 or the 1.6-µs pulse duration methods. The UART_ACREG[7] PULSE_TYPE bit selects the pulse width method in the transmit mode.

12.1.6.3.8.3.4.3 SIR Free Format Programming

The SIR FF mode is selected by setting the module in the UART mode (UART_MDR1[2-0] MODE_SELECT = 0x0) and the UART_MDR2[3] PULSE bit to 1 to allow pulse shaping.

Because the bit format stays the same, some UART mode configuration registers must be set at specific values:

- UART_LCR[1-0] CHAR_LENGTH bit field = 0x3 (8 data bits)
- UART_LCR[2] NB_STOP bit = 0x0 (1 stop-bit)
- UART_LCR[3] PARITY_EN bit = 0x0 (no parity)

The UART mode interrupts are used for the SIR FF mode, but many are not relevant (XOFF, RTS, CTS, modem status register, etc.).

12.1.6.3.8.3.5 MIR and FIR Mode Data Formatting

This section describes common instructions for FIR and MIR mode programming.

At the end of a frame reception, the CPU reads the line status register (UART_LSR_UART) to detect errors in the received frame.

When the UART_MDR1[6] SIP_MODE bit is set to 1, the TX state-machine always sends one SIP at the end of a transmission frame. However, when the SIP_MODE bit is set to 0, SIP transmission depends on the UART_ACREG[3] SEND_SIP bit.

The CPU can set the SEND_SIP bit at least once every 500 ms. The advantage of this approach over the default approach is that the TX state-machine does not have to send the SIP at the end of each frame, thus reducing the overhead required.

12.1.6.3.8.4 CIR Mode

12.1.6.3.8.4.1 CIR Mode Clock Generation

Depending on the encoding method (variable pulse distance/biphase), the Host CPU must develop a data structure that combines 1 and 0 with a *t* period to encode the complete frame to transmit. This can then be transmitted to the infrared output with a modulation method, as shown in Figure 12-60.

Copyright © 2024 Texas Instruments Incorporated

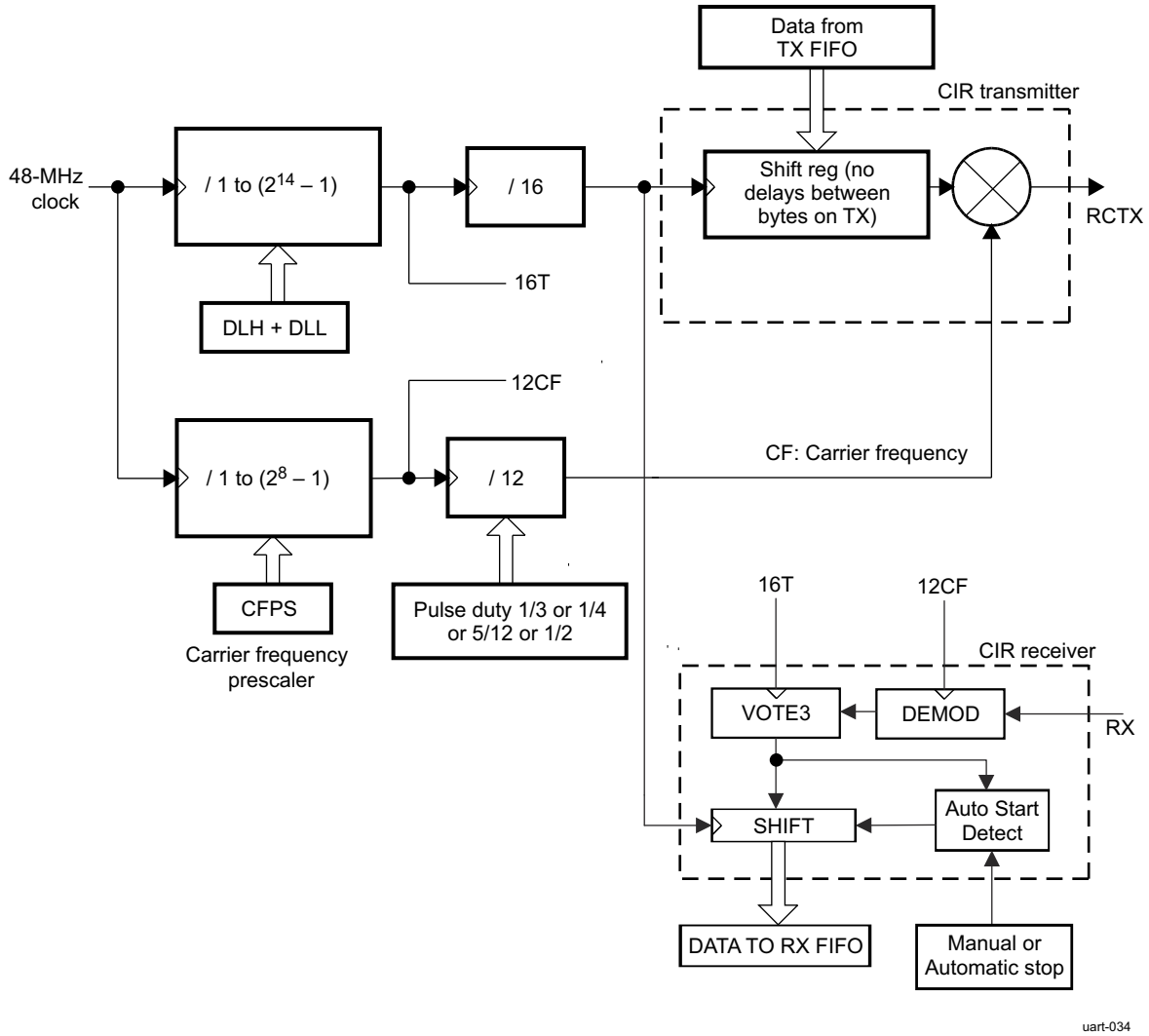


Figure 12-60. CIR Mode Block Components

Based on the requested modulation frequency, the UART_CFPS register must be set with the correct dividing value to provide an accurate pulse frequency:

Dividing value = (FCLK / 12) / MODfreq

Where:

FCLK = System clock frequency (48 MHz)

12 = Real value of baud multiple

MODfreq = Effective frequency of the modulation (MHz)

Example: For a targeted modulation frequency of 36 kHz, the value of CFPS must be set to 0x7 (decimal), which provides a modulation frequency of 36.04 kHz.

Note

The UART_CFPS register starts with a reset value of 105 (decimal), which translates to a frequency of 38.1 kHz.

The duty cycle of these pulses is user-defined by the pulse duty register bits in the UART_MDR2 register. Table 12-92 shows the duty cycle.

Copyright © 2024 Texas Instruments Incorporated

Table 12-92. CIR Duty Cycle

UART_MDR2[5-4] CIR_PULSE_MODE	Duty Cycle (High-Level)
00	1/4
01	1/3
10	5/12
11	1/2

12.1.6.3.8.4.2 CIR Data Formatting

The methods described in this section apply to all CIR modes.

12.1.6.3.8.4.2.1 IR RX Polarity Control

The IR RX polarity control for CIR mode has the same function as that for IrDA mode (see Section 12.1.6.3.8.3.3.1, *IR RX Polarity Control*).

12.1.6.3.8.4.2.2 CIR Transmission

In transmission, the Host CPU software must exercise an element of real-time control to transmit data packets, each of which must be emitted at a constant delay from the start-bits of each individual packet. Thus, when sending a series of packets, the packet-to-packet delay must respect a specific delay. Two methods can be used to control this delay:

- Filling the TX FIFO with a number of zero bits that are transmitted with a *t* period
- Using an external system timer to control the delay between each start-of-frame or between the end of a frame and the start of the next one. This can be performed by:
  - Controlling the start of the frame using the UART_MDR1[5] SCT bit and the UART_ACREG[2] SCTX_EN bit, depending on the timer status
  - Using the UART_IIR_UART[5] TX_STATUS_IT interrupt bit to preload the next frame in the TX FIFO and to control the start of the timer (in case of control delay between the end of a frame and the start of the next frame)

12.1.6.3.8.4.2.3 CIR Reception

There are 2 ways to stop a CIR reception:

- The Host CPU can disable the reception by setting the UART_ACREG[5] DIS_IR_RX bit to 1. When it considers that the reception is finished because a large number of 0 has been received. To receive a new frame, the UART_ACREG[5] DIS_IR_RX bit must be set to 0.
- An automatic stop mechanism can configured by setting a value in the BOF length register (UART_EBLR). If the value set in the UART_EBLR register is different than 0, this features is enabled and the number of bits received will begin counting from 0. When the counter reaches the value defined in the UART_EBLR register, reception is automatically disabled and UART_IIR_CIR[2] RX_STOP_IT bit is set. When a 1 is detected on the RX pin, reception is automatically re-enabled.

There is a limitation when receiving data in UART CIR mode. Certain IrDA transceivers on the market have a characteristic that causes shrinking of the received modulation pulse hold-time. The UART receive filtering schema is based on the same encoding mechanism used for transmission.

For the following scenario:

- Shift register period: 0.9µs
- Modulation frequency: 36kHz
- Duty cycle: 1/4 of a modulation frequency period

Data sent with these conditions would contains 7µs pulses within a 28µs period. The UART expects to receive similar incoming data on receive, but various transceiver timing characteristics typically only send 2µs modulated pulses. These 2µs pulses will be filtered out and RX FIFO will not receive data. This does not affect UART CIR mode in transmission.

CIR RX demodulation can be bypassed by setting the UART_MDR3[0] DISABLE_CIR_RX_DEMOD bit.

Copyright © 2024 Texas Instruments Incorporated

12.1.6.4 UART Programming Guide

This section describes the procedure for operating the UART with FIFO and DMA or interrupts. This three-part procedure ensures the quick start of the UART. It does not cover every UART feature.

The first programming model covers software reset of the UART. The second programming model describes FIFO and DMA configuration. The last programming model describes protocol, baud rate, and interrupt configuration.

Note

Each programming model can be used independently of the other two; for instance, reconfiguring the FIFOs and DMA settings only.

Each programming model can be executed starting from any UART register access mode (register modes, submodes, and other register dependencies). However, if the UART register access mode is known before executing the programming model, some steps that enable or restore register access are optional. For more information, see Section 12.1.6.3.7.1, *Register Access Modes*.

12.1.6.4.1 UART Mode selection

Table 12-93 describes how to set different register access mode.

Table 12-93. UART Configure Register Access Mode

Step	Register/Bit Field/Programming Model	Value
Set the register access mode A	UART_LCR[7] DIV_EN	1
	UART_LCR[7-0]	≠0xBF
Set the register access mode B	UART_LCR[7-0]	0xBF
Set the operational mode	UART_LCR[7] DIV_EN	0

12.1.6.4.2 UART Submode selection

This section describes how to set different register access submode.

Table 12-94. UART Configure Register Access Submode TCR_TLR

Step	Register/Bit Field/Programming Model	Value
Configure the submode TCR_TLR		
Configure mode B	see Table 12-93	
Enable writing to register bits UART_MCR[7-5]	UART_EFR[4] ENHANCED_EN	1
Configure mode A	see Table 12-93	0x1
Set the submode TCR_TLR	UART_MCR[6] TCR_TLR	1

Table 12-95. UART Configure Register Access Submode MSR_SPR

Step	Register/Bit Field/Programming Model	Value
First option: configure the submode MSR_SPR		
Configure mode B	see Table 12-93	
Set the submode MSR_SPR	UART_EFR[4] ENHANCED_EN	0
Second option: configure the submode MSR_SPR		
Configure mode B	see Table 12-93	
Enable writing to register bits UART_MCR[7-5]	UART_EFR[4] ENHANCED_EN	1
Set the submode MSR_SPR	UART_MCR[6] TCR_TLR	0

Table 12-96. UART Configure Register Access Submode XOFF

Step	Register/Bit Field/Programming Model	Value
Configure of the XOFF		
Configure B	see Table 12-93	

Copyright © 2024 Texas Instruments Incorporated

Table 12-96. UART Configure Register Access Submode XOFF (continued)

Step	Register/Bit Field/Programming Model	Value
Set the submode XOFF	UART_EFR[4] ENHANCED_EN	0

12.1.6.4.3 UART Load FIFO trigger and DMA mode settings

12.1.6.4.3.1 DMA mode Settings

To enable and configure program the DMA mode, perform the following steps:

Table 12-97. DMA Mode Settings

Step	Register/Bit Field/Programming Model	Value
Set the option of DMA mode configuration	UART_SCR[0] DMA_MODE_CTL	-
IF Configure DMA mode 0 and 1	UART_SCR[0] DMA_MODE_CTL	=0
Select the DMA mode, for more information see Section 12.1.6.3.6.4	UART_FCR[3] DMA_MODE	-
IF Configure DMA mode from 0 to 3	UART_SCR[0] DMA_MODE_CTL	=1
Select the DMA mode, for more information see Section 12.1.6.3.6.4	UART_SCR[2-1] DMA_MODE_2	-

12.1.6.4.3.2 FIFO Trigger Settings

In this section is described configuration and settings of FIFO trigger level, which enable DMA and interrupt generation.

Table 12-98. Load FIFO Triggers Defined by the FCR

Step	Register/Bit Field/Programming Model	Value
Configure register submode TCR_TLR	see Table 12-94	0x-
Set the desire RX FIFO trigger level	UART_FCR[5-4] TX_FIFO_TRIG	0x-
Set the desire TX FIFO trigger level	UART_FCR[7-6] RX_FIFO_TRIG	0x-

Table 12-99. Load FIFO Triggers Defined by the TLR

Step	Register/Bit Field/Programming Model	Value
Configure register submode TCR_TLR	see Table 12-94	0x-
Set the desire RX FIFO trigger level	UART_TLR[7-4] RX_FIFO_TRIG_DMA	0x-
Set the desire TX FIFO trigger level	UART_TLR[3-0] TX_FIFO_TRIG_DMA	0x-

Table 12-100. Load FIFO Triggers Defined by the Concatenated Value

Step	Register/Bit Field/Programming Model	Value
Configure register submode TCR_TLR	see Table 12-94	0x-
Set the register bit	UART_SCR[7] RX_TRIG_GRANU1	1
Set the desire RX FIFO trigger level	UART_TLR[7-4] RX_FIFO_TRIG_DMA UART_FCR[7-6] RX_FIFO_TRIG	0x-
Set the register bit	UART_SCR[6] TX_TRIG_GRANU1	1
Set the desire TX FIFO trigger level	UART_TLR[3-0] TX_FIFO_TRIG_DMA UART_FCR[5-4] TX_FIFO_TRIG	0x-

12.1.6.4.4 UART Protocol, Baud rate and interrupt settings

12.1.6.4.4.1 Baud rate settings

Table 12-101. UART Baud Rate Settings

Step	Register/Bit Field/Programming Model	Value
Disable UART mode	UART_MDR1[2-0] MODE_SELECT	0x7
Switch to register configuration mode B	see Table 12-93	
Enable access to UART_IER_UART[7-4]	UART_EFR[4] ENHANCED_EN	1
Switch register operational mode	see Table 12-93	

Copyright © 2024 Texas Instruments Incorporated

Table 12-101. UART Baud Rate Settings (continued)

Step	Register/Bit Field/Programming Model	Value
Disable sleep mode	UART_IER_UART[4] SLEEP_MODE	0
Switch to register configuration mode A or B	see Table 12-93	
Set the appropriate divisor value	UART_DLL[7-0] CLOCK_LSB UART_DLH[5-0] CLOCK_MSB	0x-

12.1.6.4.4.2 Interrupt settings

Table 12-102. UART Interrupt Settings

Step	Register/Bit Field/Programming Model	Value
Switch to register configuration mode B	see Table 12-93	0x7
Enable access to UART_IER_UART[7-4]	UART_EFR[4] ENHANCED_EN	1
Switch register operational mode	see Table 12-93	
Set the desired interrupt configuration (0: Disable the interrupt; 1: Enable the interrupt)	UART_IER_UART[7] CTS_IT UART_IER_UART[6] RTS_IT UART_IER_UART[5] XOFF_IT UART_IER_UART[4] SLEEP_MODE UART_IER_UART[3] MODEM_STS_IT UART_IER_UART[2] LINE_STS_IT UART_IER_UART[1] THR_IT UART_IER_UART[0] RHR_IT	0x-

12.1.6.4.4.3 Protocol settings

Load the desired protocol formatting (parity, stop-bit, character length) and switch to register operational mode.

Table 12-103. UART Protocol Settings

Step	Register/Bit Field/Programming Model	Value
Load desired protocol formatting, see Section 12.1.6.3.8.1.3.1 , *Frame Formatting*	UART_LCR[5] PARITY_TYPE_2 UART_LCR[4] PARITY_TYPE_1 UART_LCR[3] PARITY_EN UART_LCR[2] NB_STOP UART_LCR[1-0] PARITY_LENGTH	0x-
Switch to register operational mode	UART_LCR[7] DIV_EN UART_LCR[6] BREAK_EN	0

12.1.6.4.4.4 UART/RS-485/IrDA(SIR/MIR/FIR)/CIR

Table 12-104. UART Mode Selection

Step	Register/Bit Field/Programming Model	Value
Load the desired UART/IrDA (SIR, MIR, FIR)/CIR modes, see Section 12.1.6.3.7.2, *UART/RS-485/IrDA (SIR, MIR, FIR)/CIR Mode Selection*	UART_MDR1[2-0] MODE_SELECT	0x-
Load the desire RS-485 mode, see Section 12.1.6.3.7.2, *UART/RS-485/IrDA (SIR, MIR, FIR)/CIR Mode Selection*	UART_MDR3[4] DIR_EN	0x1

12.1.6.4.4.5 UART Multi-drop Parity Address Match Mode Configuration

Table 12-105. UART Multi-drop Parity Address Match Mode Configuration

Step	Register/Bit Field/Programming Model	Value
Disable receive mode	UART_ECR[3] RX_EN	0
Enable Multi-drop parity Address match mode	UART_EFR2[2] MULTIDROP	1
Set the matching device address	UART_MAR[7-0] ADDRESS	0x-
Set the address match masking	UART_MMR[7-0] MASK	0x-

Copyright © 2024 Texas Instruments Incorporated

Table 12-105. UART Multi-drop Parity Address Match Mode Configuration (continued)

Step	Register/Bit Field/Programming Model	Value
Set the broadcast address match	UART_MBR[7-0] BROADCAST_ADDRESS	0x-
Enable broadcast address matching if needed	UART_EFR2[7] BROADCAST	1
Enable receive mode	UART_ECR[3] RX_EN	1

12.1.6.4.5 UART Hardware and Software Flow Control Configuration

This section describes the programming steps to enable and configure hardware and software flow control. Hardware and software flow control cannot be used at the same time.

12.1.6.4.5.1 Hardware Flow Control Configuration

Table 12-106. UART Hardware Flow Control Configuration

Step	Register/Bit Field/Programming Model	Value
Configure register submode TCR_TLR	see Table 12-94	0x7
Load the start and halt trigger value.	UART_TCR[7-4] AUTO_RTS_START UART_TCR[3-0] AUTO_RTS_HALT	0x-
Enable or disable receive and transmit hardware flow control mode.	UART_EFR[7] AUTO_CTS_EN UART_EFR[6] AUTO_RTS_EN	0x-

12.1.6.4.5.2 Software Flow Control Configuration

Table 12-107. UART Software Flow Control Configuration

Step	Register/Bit Field/Programming Model	Value
Set the register access submode XOFF	see Table 12-96	
Load the software control characters	UART_XON1_ADDR1[7-0] XON_WORD1 UART_XON2_ADDR2[7-0] XON_WORD2 UART_XOFF1[7-0] XOFF_WORD1 UART_XOFF2[7-0] XOFF_WORD2	0x-
Set the register access submode TCR_TLR	see Table 12-94	
Enable or disable XON any function (0: Disable; 1: Enable).	UART_MCR[5] XON_EN	--
Load start and halt trigger value for software flow control	UART_TCR[7-4] AUTO_RTS_START UART_TCR[3-0] AUTO_RTS_HALT	0x-
Enable or disable special character function (0: Disable; 1: Enable)	UART_EFR[5] SPEC_CHAR	0x-
Set the software flow control mode	UART_EFR[3-0] SW_FLOW_CONTROL	0x-

12.1.6.4.6 IrDA Programming Model

12.1.6.4.6.1 SIR mode

12.1.6.4.6.1.1 Receive

The following programming model explains how to program the module to receive an IrDA frame with parity forced to 1, baud rate = 115.2 kbps, FIFOs disabled, 2 stop-bits, and 8-bit word length:

Table 12-108. SIR Mode Receive Settings

Step	Register/Bit Field/Programming Model	Value
Disable UART mode	UART_MDR1[2-0] MODE_SELECT	0x7
Grant access to the UART_DLL and UART_DLH registers	UART_LCR[7-0]	0x80
Load the baud rate(115.2 Kbps)	UART_DLL[7-0] CLOCK_LSB UART_DLH[5-0] CLOCK_MSB	0x1A 0x00
Set SIR mode	UART_MDR1[2-0] MODE_SELECT	0x1
Disable access to the UART_DLL and UART_DLH registers	UART_LCR[7-0]	0x00

Copyright © 2024 Texas Instruments Incorporated

Table 12-108. SIR Mode Receive Settings (continued)

Step	Register/Bit Field/Programming Model	Value
Enable the UART_RHR interrupt	UART_IER_IRDA[0] RHR_IT	1

12.1.6.4.6.1.2 Transmit

The following programming model explains how to program the module to transmit an IrDA 6-byte frame with no parity, baud rate = 115.2 kbps, FIFOs disabled, 3/16 encoding, 2 stop-bits, and 7-bit word length:

Table 12-109. SIR Mode Transmit Settings

Step	Register/Bit Field/Programming Model	Value
Disable UART mode	UART_MDR1[2-0] MODE_SELECT	0x7
Grant access to the UART_DLL and UART_DLH registers	UART_LCR[7-0]	0x80
Load the baud rate (115.2 Kbps)	UART_DLL[7-0] CLOCK_LSB	0x1A
	UART_DLH[5-0] CLOCK_MSB	0x00
Set SIR mode	UART_MDR1[2-0] MODE_SELECT	0x1
Disable access to the UART_DLL and UART_DLH registers	UART_LCR[7-0]	0x00
Force output DTR to active	UART_MCR[0] DTR	1
Enable the UART_THR interrupt	UART_IER_IRDA[1] THR_IT	0x1
Set transmit frame length to 6 bytes	UART_TXFLL[7-0] TXFLL	0x06
Set the seven starts of frame transmission	UART_EBLR[7-0] EBLR	0x08
Set SIR pulse width to be 1.6 µs	UART_ACREG[7] PULSE_TYPE	1

12.1.6.4.6.2 MIR mode

12.1.6.4.6.2.1 Receive

The following programming model explains how to program the module to receive an IrDA frame with no parity, baud rate = 1.152 Mpbs, and FIFOs disabled.

Table 12-110. MIR Mode Receive Settings

Step	Register/Bit Field/Programming Model	Value
Disable UART mode	UART_MDR1[2-0] MODE_SELECT	0x7
Grant access to the UART_DLL and UART_DLH registers	UART_LCR[7-0]	0x80
Load the baud rate (1.152 bps)	UART_DLL[7-0] CLOCK_LSB	0x01
	UART_DLH[5-0] CLOCK_MSB	0x00
Set MIR mode	UART_MDR1[2-0] MODE_SELECT	0x4
Disable access to the UART_DLL and UART_DLH registers	UART_LCR[7-0]	0x00
Force outputs DTR and RTS to active	UART_MCR[1-0]	0x3
Enable the UART_RHR interrupt	UART_IER_IRDA[0] RHR_IT	1

12.1.6.4.6.2.2 Transmit

The following programming model explains how to program the module to transmit an IrDA 60-byte frame with no parity, baud rate = 1.152 Mpbs, and FIFOs disabled.

Table 12-111. MIR Mode Transmit Settings

Step	Register/Bit Field/Programming Model	Value
Disable UART mode	UART_MDR1[2-0] MODE_SELECT	0x7
Grant access to the UART_DLL and UART_DLH registers	UART_LCR[7-0]	0x80
Load the baud rate (115.2 kbps)	UART_DLL[7-0] CLOCK_LSB	0x01
	UART_DLH[5-0] CLOCK_MSB	0x00
Set SIR mode	UART_MDR1[2-0] MODE_SELECT	0x4

Copyright © 2024 Texas Instruments Incorporated

Table 12-111. MIR Mode Transmit Settings (continued)

Step	Register/Bit Field/Programming Model	Value
Disable access to the UART_DLL and UART_DLH registers	UART_LCR[7-0]	0x00
Force output DTR to active	UART_MCR[0] DTR	1
Enable the UART_THR interrupt	UART_IER_IRDA[1] THR_IT	0x1
Set transmit frame length to 60 bytes	UART_TXFLL[7-0] TXFLL	0x3C
Set the eight additional starts of frame transmission	UART_EBLR[7-0] EBLR	0x08
SIP is sent at the end of transmission	UART_ACREG[3] SEND_SIP	1

12.1.6.4.6.3 FIR mode

12.1.6.4.6.3.1 Receive

The following programming model explains how to program the module to receive the IrDA frame with no parity, baud rate = 4 Mbps, FIFOs enabled, 8-bit word length.

Table 12-112. FIR Mode Receive Settings

Step	Register/Bit Field/Programming Model	Value
Disable UART mode	UART_MDR1[2-0] MODE_SELECT	0x7
Grant access to the UART_DLL and UART_DLH registers	UART_LCR[7-0]	0x80
Enable access to change UART_FCR[0]	UART_DLL[7-0] CLOCK_LSB UART_DLH[7-0] CLOCK_MSB	0x0
FIFO clear and enable	UART_FCR[2-0]	0x7
Set the FIFO trigger level	see Section 12.1.6.4.3, *Load FIFO trigger and DMA mode settings*	
Set FIR mode	UART_MDR1[2-0] MODE_SELECT	0x5
Set frame length	UART_RXFLL[7-0] RXFLL	0xA
Disable access to the UART_DLL and UART_DLH registers	UART_LCR[7-0]	0x00
Enable the UART_RHR interrupt	UART_IER_IRDA[0] RHR_IT	1

12.1.6.4.6.3.2 Transmit

The following programming model explains how to program the module to transmit an IrDA 4-byte frame with no parity, baud rate = 4 Mbps, FIFOs enabled, and 8-bit word length.

Table 12-113. FIR Mode Transmit Settings

Step	Register/Bit Field/Programming Model	Value
Disable UART mode	UART_MDR1[2-0] MODE_SELECT	0x7
Grant access to the UART_DLL and UART_DLH registers	UART_LCR[7-0]	0x80
Enable access to change UART_FCR[0]	UART_DLL[7-0] CLOCK_LSB UART_DLH[5-0] CLOCK_MSB	0x0
FIFO clear and enable	UART_FCR[2-0]	0x7
Set the FIFO trigger level	see Section 12.1.6.4.3, *Load FIFO trigger and DMA mode settings*	
Set FIR mode	UART_MDR1[2-0] MODE_SELECT	0x1
Disable access to the UART_DLL and UART_DLH registers	UART_LCR[7-0]	0x00
Set FIR mode and enable auto-SIP mode	UART_MDR1[7-0]	0x45
Set frame length	UART_TXFLL[7-0] TXFLL UART_TXFLH[7-0] TXFLH	0x4 0x0
Force output DTR to active	UART_MCR[0] DTR	1
Enable the UART_THR interrupt	UART_IER_IRDA[1] THR_IT	1
Set the eight additional starts of frame transmission	UART_EBLR[7-0] EBLR	0x08

Copyright © 2024 Texas Instruments Incorporated

Table 12-113. FIR Mode Transmit Settings (continued)

Step	Register/Bit Field/Programming Model	Value
SIP is sent at the end of transmission	UART_ACREG[3] SEND_SIP	1

Copyright © 2024 Texas Instruments Incorporated

12.2 High-speed Serial Interfaces

This section describes the high-speed serial interfaces in the device.

Copyright © 2024 Texas Instruments Incorporated

12.2.1 Gigabit Ethernet MAC (CPSW)

This chapter describes the Gigabit Ethernet MAC (Media Access Controller). For conceptual purposes the below documentation refers to this MAC as being a two port CPSW with port 0 being the CPPI DMA host port and port 1 being the Ethernet port.

12.2.1.1 CPSW Programming Guide

12.2.1.1.1 Initialization and Configuration of CPSW Subsystem

To configure the CPSW Ethernet Subsystem for operation, the host must perform the following:

1. Select the Interface (RMII, or RGMII) Mode. See the CTRLMMR_MCU_ENET_CTRL[1-0] MODE_SEL register.
2. Configure pads (pin muxing), as per the interface selected. Refer to *Pad Configuration Registers* and the device-specific Datasheet.
3. Enable the CPSW Ethernet Subsystem clocks. See *CPSW Integration*
4. Ensure that at least 2000 CPPI_ICLK periods are run after reset is de-asserted.
5. Configure the CPSW_CONTROL_REG register
6. Configure the Ethernet Port Source Address registers (CPSW_PN_SA_L_REG and CPSW_PN_SA_H_REG)
7. Configure the CPSW statistic port enable register CPSW_STAT_PORT_EN_REG
8. Configure the ALE (*Address Lookup Engine*)
9. Configure the MDIO (*Initializing the MDIO Module*)
10. Configure Ethernet port, as per the desired mode of operations

12.2.1.1.2 CPSW Reset

To reset the Ethernet port, the host must perform the following:

1. Set CMD_IDLE bit to 1h in the Ethernet port control register: CPSW_PN_MAC_CONTROL_REG.
2. Wait for IDLE bit to be set to 1h, which is indicated in the Ethernet port status register: CPSW_PN_MAC_STATUS_REG.
3. Set SOFT_RESET bit to 1h in the Ethernet port software reset register: CPSW_PN_MAC_SOFT_RESET_REG.
4. Wait for SOFT_RESET bit in the CPSW_PN_MAC_SOFT_RESET_REG registers to be cleared to confirm reset completion.
5. Configure the Ethernet ports.
6. Re-configure registers reset to default value by CPSW_PN_MAC_SOFT_RESET_REG.

12.2.1.1.3 MDIO Software Interface

12.2.1.1.3.1 Initializing the MDIO Module

The following steps are performed by the application software or device driver to initialize the MDIO device:

1. Configure the PREAMBLE and CLKDIV bits in the MDIO Control register (CPSW_MDIO_CONTROL_REG).
2. Enable the MDIO module by setting the ENABLE bit in CPSW_MDIO_CONTROL_REG.
3. The MDIO PHY alive status register (MDIO CPSW_MDIO_ALIVE_REG) can be read in polling fashion until a PHY connected to the system responded, and the MDIO PHY link status register (MDIO CPSW_MDIO_LINK_REG) can determine whether this PHY already has a link.
4. Set the appropriate PHY addresses in the MDIO user PHY select register (CPSW_MDIO_USER_PHY_SEL_REG_k, where k = 0 or 1), and set the LINKINT_ENABLE bit to enable a link change event interrupt if desirable.
5. Set the appropriate LINKSEL bit in the CPSW_MDIO_USER_PHY_SEL_REG_k register (where k = 0 or 1).
6. Set the appropriate USERINTMASKSET bit field in the CPSW_MDIO_USER_INT_MASK_SET_REG register.
7. If an interrupt on general MDIO register access is desired, set the corresponding bit in the MDIO user command complete interrupt mask set register (MDIO CPSW_MDIO_USER_INT_MASK_SET_REG) to use the MDIO user access register (MDIO CPSW_MDIO_USER_ACCESS_REG_k, where k = 0 or 1).

Copyright © 2024 Texas Instruments Incorporated

12.2.1.1.3.2 Writing Data To a PHY Register

The MDIO module includes a user access register (MDIO CPSW_MDIO_USER_ACCESS_REG_k, where k = 0 or 1) to directly access a specified PHY device.To write a PHY register, perform the following:

1. Check to ensure that the GO bit in the MDIO user access register (MDIO CPSW_MDIO_USER_ACCESS_REG_k) is cleared.
2. Write to the GO, WRITE, REGADR, PHYADR, and DATA bits in MDIO CPSW_MDIO_USER_ACCESS_REG_k corresponding to the PHY and PHY register SW wants to write.
3. The write operation to the PHY is scheduled and completed by the MDIO module. Completion of the write operation can be determined by polling the GO bit in MDIO CPSW_MDIO_USER_ACCESS_REG_k for a 0.
4. Completion of the operation sets the corresponding USERINTRAW bit (0 or 1) in the MDIO user command complete interrupt register (CPSW_MDIO_USER_INT_RAW_REG) corresponding to MDIO CPSW_MDIO_USER_ACCESS_REG_k used. If interrupts have been enabled on this bit using the MDIO user command complete interrupt mask set register (CPSW_MDIO_USER_INT_MASK_SET_REG), then the bit is also set in the MDIO user command complete interrupt register (CPSW_MDIO_USER_INT_MASKED_REG) and an interrupt is triggered on the host processor.

12.2.1.1.3.3 Reading Data From a PHY Register

The MDIO module includes a user access register (MDIO CPSW_MDIO_USER_ACCESS_REG_k, where k = 0 or 1) to directly access a specified PHY device. To read a PHY register, perform the following:

1. Check to ensure that the GO bit in the MDIO user access register (CPSW_MDIO_USER_ACCESS_REG_k, where k = 0 or 1) is cleared.
2. Write to the GO, REGADR, and PHYADR bits in the CPSW_MDIO_USER_ACCESS_REG_k register corresponding to the PHY and PHY register SW wants to read.
3. The read data value is available in the DATA bit field in MDIO CPSW_MDIO_USER_ACCESS_REG_k register after the module completes the read operation on the serial bus. Completion of the read operation can be determined by polling the GO and ACK bits in CPSW_MDIO_USER_ACCESS_REG_k register. After the GO bit has cleared, the ACK bit is set on a successful read.
4. Completion of the operation sets the corresponding USERINTRAW bit (0 or 1) in the MDIO user command complete interrupt register (CPSW_MDIO_USER_INT_RAW_REG) corresponding to MDIO CPSW_MDIO_USER_ACCESS_REG_k used. If interrupts have been enabled on this bit using the MDIO user command complete interrupt mask set register (CPSW_MDIO_USER_INT_MASK_SET_REG), then the bit is also set in the MDIO user command complete interrupt register (CPSW_MDIO_USER_INT_MASKED_REG) and an interrupt is triggered on the host processor.

Copyright © 2024 Texas Instruments Incorporated

12.2.2 Peripheral Component Interconnect Express (PCIe) Subsystem

This chapter describes the features and functions of the device Peripheral Component Interconnect Express (PCIe) subsystem.

12.2.2.1 PCIe Subsystem Overview

The Peripheral Component Interconnect Express (PCIe) subsystem is built around a multi-lane dual-mode PCIe controller that provides low pin-count, high reliability, and high-speed data transfers at rates of up to 8.0 Gbps per lane for serial links on backplanes and printed wiring boards.

Figure 12-61 provides PCIe subsystem overview.

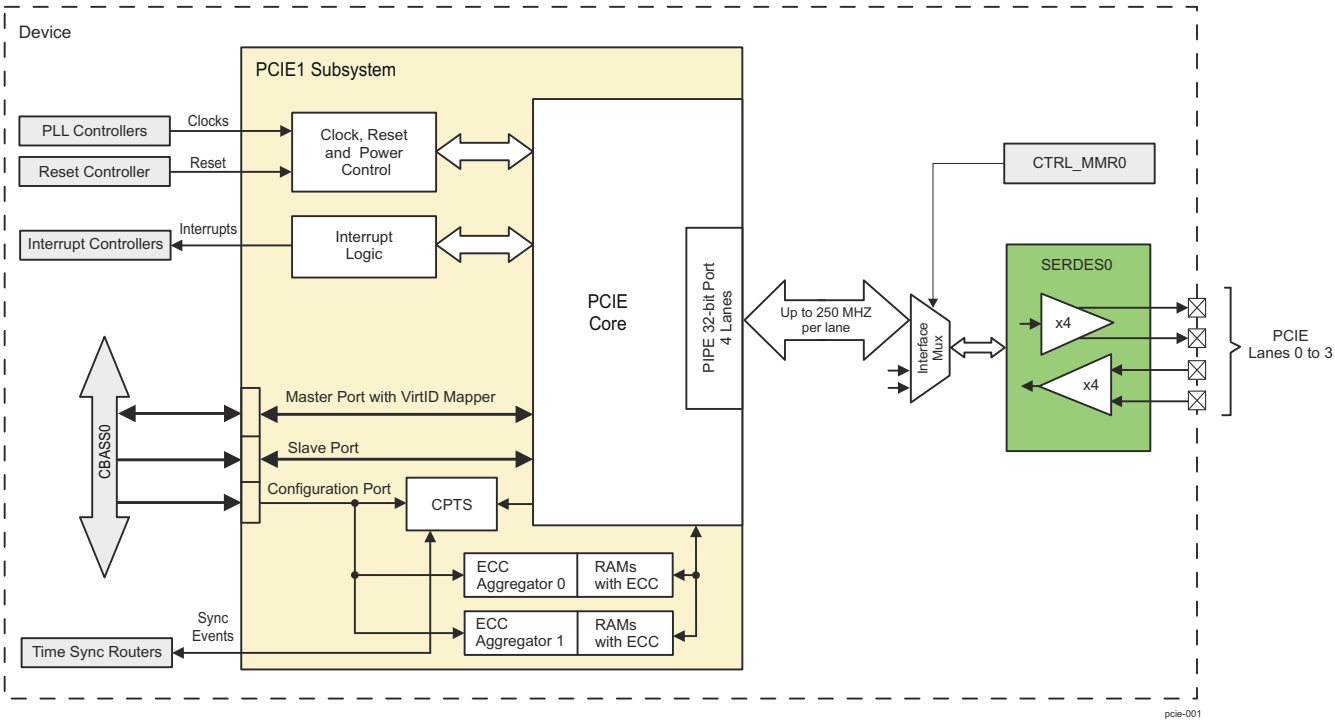


Figure 12-61. PCIe Subsystem Overview

12.2.2.1.1 PCIe Subsystem Features

Each PCIe subsystem supports the following main features:

- Compliance to PCIe® Base Specification, Revision 4.0 (Version 0.7)
- 4-lane configuration with up to 8.0 Gbps/lane (Gen3). Can be used in 1x4, 1x2, 1x1 modes.
- Gen3 (8 Gbps 128/130-bit encoding), Gen2 (5 Gbps 8/10-bit encoding), and Gen1 (2.5 Gbps 8/10-bit encoding) with auto-negotiation
- 62.5/125/250MHz operation on PIPE interface for Gen1/Gen2/Gen3, respectively
- Constant 32-bit PIPE width for Gen1/2/3 modes
- Dynamic PIPE width change when switching between Gen1/2/3 modes
- Dual mode: Root Port (RP) or End Point (EP) operation modes
- Maximum payload size of 256 bytes
- Maximum remote read request size of 4K bytes
- Four virtual channels (VC)
- Four traffic classes (TC)
- PCI Power Management states are:
  - L1 Active State Power Management
  - L1 Power Management substates support
  - D1 Device Power Management state
- Maximum number of non-posted outstanding transactions: 32
- Resizable BAR capability

Copyright © 2024 Texas Instruments Incorporated

- Separate Reference Clock with Independent Spread (SRIS)
- Legacy, MSI and MSI-X Interrupt Support
- 32 outbound address translation regions
- Precision time measurement (PTM)

Unsupported Features:

- See the Module Integration section of this document for a list of module features not supported by the integration on this Device.

Copyright © 2024 Texas Instruments Incorporated

12.2.2.1.2 PCIe Subsystem Ports

This section describes PCIe subsystem ports related to clocks, resets, and hardware requests.

Table 12-114. PCIe Subsystem Clocks and Resets

Clocks	
Module Clock Input	**Description**
PCIE_FICLK	PCIE interface bus clock (CBA_CLK).
PCIE_PM_CLK	PCIE core free-running clock used for low power state transitions and clock control generation.
PCIE_CPTS_RCLK	PCIE CPTS reference clock (RCLK). The CPTS RCLK clock frequency should be greater than or equal to the PCIE CBA_CLK clock frequency. Otherwise, the software will have to add some wait cycles before a correct event is generated by the CPTS module. Additionally, the CPTS RCLK should be set to same frequency as the PCIe core clock.
PCIE_LANE0_TXMCLK	PCIE core clock (CORE_CLK) driven by SERDES via lane 0. In multi-lane modes, only PCIE1_LANE0_TXMCLK is used as the core clock input for all lanes. The TXMCLKs for the rest of the lanes are left unconnected
PCIE_LANE0_RXCLK	PCIE PIPE interface clock driven by SERDES via lane 0. In multi-lane modes, only PCIE1_LANE0_RXCLK is used as the PIPE interface clock for all lanes. The RXCLKs for the rest of the lanes are left unconnected.
SERDES0_IP2_LN0_TXCLK	PCIE lane 0 PIPE/RAW TX return clock.
SERDES0_IP2_LN1_TXCLK	PCIE lane 1 PIPE/RAW TX return clock.
SERDES0_IP2_LN2_TXCLK	PCIE lane 2 PIPE/RAW TX return clock.
SERDES0_IP2_LN3_TXCLK	PCIE lane 3 PIPE/RAW TX return clock.
Resets	
Module Reset Input	**Description**
PCIE_RST	PCIE reset

Table 12-115. PCIe Subsystem Hardware Requests

Interrupt Requests		
Module Interrupt Signal	**Description**	**Type**
PCIE_DOWNSTREAM_PULSE_0	PCIE downstream interrupt	Pulse
PCIE_ERROR_PULSE_0	PCIE error interrupt	Pulse
PCIE_FLR_PULSE_0	PCIE function level interrupt	Pulse
PCIE_HOT_RESET_PULSE_0	PCIE hot reset interrupt	Pulse
PCIE_LEGACY_PULSE_0	PCIE legacy interrupt	Pulse
PCIE_LINK_STATE_PULSE_0	PCIE link state interrupt	Pulse
PCIE_LOCAL_LEVEL_0	PCIE local interrupt	Level
PCIE_PWR_STATE_PULSE_0	PCIE power state interrupt	Pulse
PCIE_PHY_LEVEL_0	PCIE PHY interrupt	Level
PCIE_PTM_VALID_PULSE_0	PCIE PTM valid interrupt	Pulse
PCIE_ECC0_CORR_LEVEL_0	PCIE ECC AGGR 0 correctable error interrupt	Level
PCIE_ECC0_UNCORR_LEVEL_0	PCIE ECC AGGR 0 uncorrectable error interrupt	Level
PCIE_ECC1_UNCORR_LEVEL_0	PCIE ECC AGGR 1 uncorrectable error interrupt	Level
PCIE_ASF_NONFATAL_LEVEL_0	PCIE active internal diagnostics interrupt	Level
PCIE_ASF_FATAL_LEVEL_0	PCIE active internal diagnostics interrupt	Level
PCIE_CPTS_PEND_0	PCIE timesync interrupt	Level
PCIE_DPA_PULSE_0	PCIE dynamic power allocation interrupt	Pulse
DMA Events		
Module DMA Event	**Description**	**Type**
-	The PCIe subsystem does not provide built-in DMA capabilities	-

Copyright © 2024 Texas Instruments Incorporated

Table 12-115. PCIe Subsystem Hardware Requests (continued)

Time Sync and Compare Events (Output)		
Module Event	**Description**	**Type**
PCIE_CPTS_HW1_PUSH_0	PCIE CPTS hardware push event (HW1_TS_PUSH)	Edge
PCIE_CPTS_COMP_0	PCIE CPTS compare output interrupt	Edge
PCIE_CPTS_SYNC_0	PCIE CPTS sync output interrupt	Edge
PCIE_CPTS_GENF_0	PCIE CPTS GENF0 output interrupt	Edge
Time Sync Events (Input)		
Module Event	**Description**	**Type**
PCIE_CPTS_HW2_PUSH_0	PCIE CPTS hardware time stamp push event (HW2_TS_PUSH)	Edge

12.2.2.2 PCIe Environment

This section describes the PCIe subsystem application fields from an environment point of view (external connections).

Table 12-116 describes the SERDES signal names at device level related to PCIe subsystem and specifies their functions.

Copyright © 2024 Texas Instruments Incorporated

Table 12-116. PCIe Subsystem I/O Signals

Device Level Signal	I/O [1]	Description
RXN0	I	PCIe Lane 0 Receive Differential Data (-)
RXP0	I	PCIe Lane 0 Receive Differential Data (+)
TXN0	O	PCIe Lane 0 Transmit Differential Data (-)
TXP0	O	PCIe Lane 0 Transmit Differential Data (+)
RXN1	I	PCIe Lane 1 Receive Differential Data (-)
RXP1	I	PCIe Lane 1 Receive Differential Data (+)
TXN1	O	PCIe Lane 1 Transmit Differential Data (-)
TXP1	O	PCIe Lane 1 Transmit Differential Data (+)
RXN2	I	PCIe Lane 2 Receive Differential Data (-)
RXP2	I	PCIe Lane 2 Receive Differential Data (+)
TXN2	O	PCIe Lane 2 Transmit Differential Data (-)
TXP2	O	PCIe Lane 2 Transmit Differential Data (+)
RXN3	I	PCIe Lane 3 Receive Differential Data (-)
RXP3	I	PCIe Lane 3 Receive Differential Data (+)
TXN3	O	PCIe Lane 3 Transmit Differential Data (-)
TXP3	O	PCIe Lane 3 Transmit Differential Data (+)
CLKREQn	I/O	PCIe sideband signal for negotiation of L1 Substate entry/exit. The PCIE1_CLKREQn pin operates as an active low open-drain bidirection reference clock request pin. 1 = no request for clock; 0 = request for clock

(1) I = Input; O = Output.

Copyright © 2024 Texas Instruments Incorporated

12.2.2.3 PCIe Subsystem Functional Description

12.2.2.3.1 PCIe Subsystem Block Diagram

The block diagram of PCIe subsystem is shown in Figure 12-62. The subsystem comprises of these major components – the PCIe Core with AXI interfaces, bridges to connect to the system CBASS0 interconnect master and slave interfaces, bridges to connect the system CBASS0 configuration interfaces, additional logic to implement the Precision Time Measurement (PTM), user configuration and interrupt, and RAMs to support the controller FIFOs.

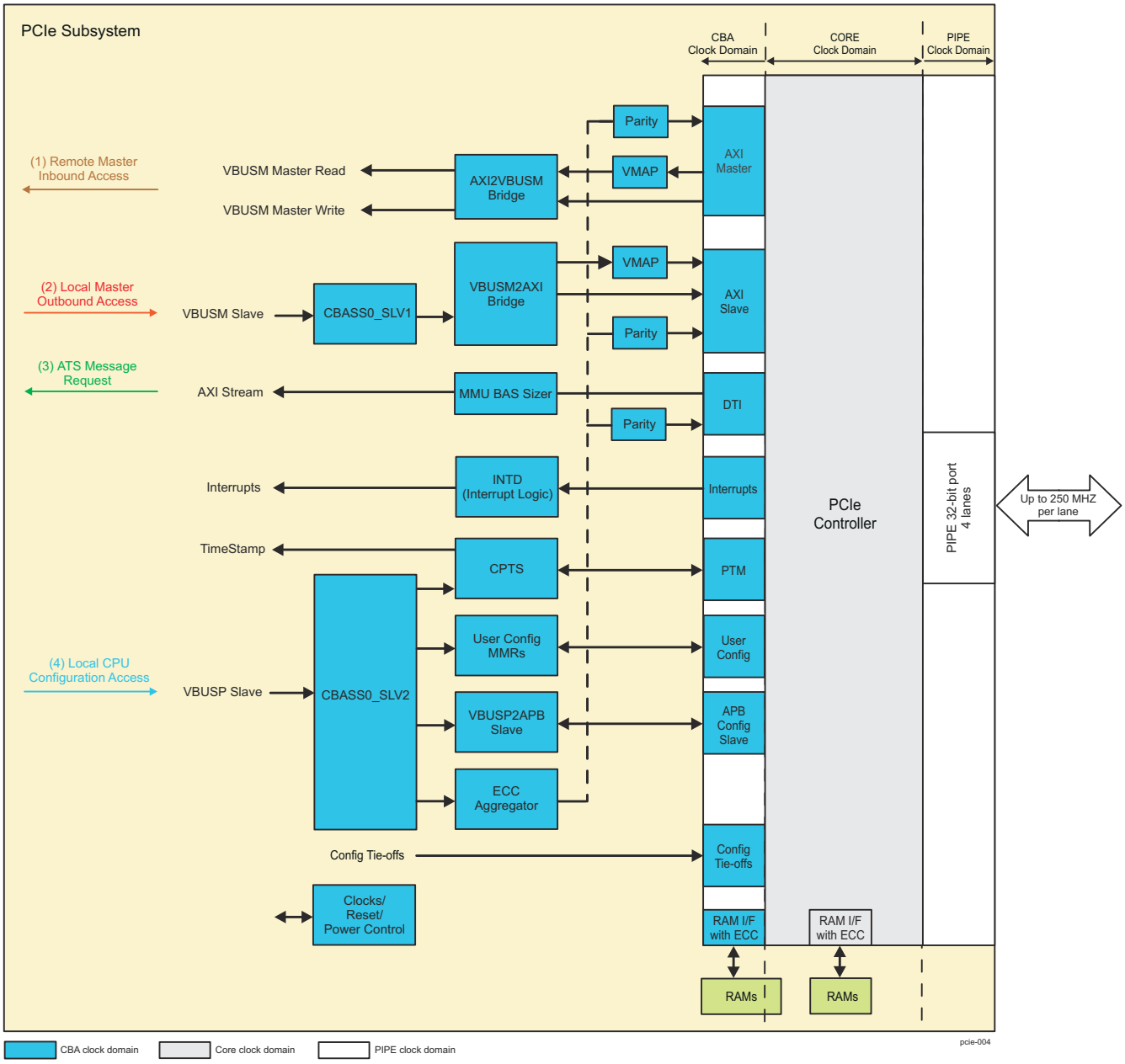


Figure 12-62. PCIe Subsystem Block Diagram

Figure 12-62 also shows some example data flows in the PCIe subsystem, such as:

1. A remote master issues a read or write access over PCIe to the local device. This will create a command to be issued on the PCIe VBUSM master read or write interface.
2. A master in the local device issues a read or write access over PCIe to the remote device. This will create a command to be issued on the PCIe VBUSM slave read or write interface.

Copyright © 2024 Texas Instruments Incorporated

3. A remote endpoint issues an Address Translation Request (ATS). This will create an inbound transaction on the AXI DTI interface.
4. A master in the local device wants to access the local PCIe configuration registers, the ECC aggregator registers or the CPTS registers. This will create a transaction over the PCIe VBUSP slave interface.

12.2.2.3.1.1 PCIe Core Module

The PCIe Core module supports dual mode of operation - it can be configured as an End Point (EP) and also as a Root Port (RP). The operational mode is selected with the CTRLMMR_PCIE1_CTRL[7] MODE_SEL register bit within the device Control Module (CTRL_MMR0). It is expected that the MODE_SEL bit is programmed during initial power up based on settings in the SoC boot configuration or a non-volatile storage such as eFuse or Flash memory.

It is not expected that the MODE_SEL setting would have to change during a full power cycle of the device. It is more likely that the operational mode of a SoC will stay as EP or RP for a particular end product's life cycle. It is not expected to switch back and forth during operation without a reset cycle.

The PCIe core module supports four virtual channels (VC) and four transfer classes (TC). The VCs can be used to implement Quality-of-Service (QoS) mechanism by enabling priority or round-robin arbitration. Typically, the highest numbered enabled VC is assigned the highest priority.

The PCIe core module is configured to support Single Root I/O virtualization (SR-IOV). It supports 6 Physical Functions (PF) and 16 Virtual Functions (VF).

There is one AXI master port and one AXI slave port in the PCIe core. All ingress data traffic regardless of the VC assigned will be delivered on the AXI master port. Similarly, all egress data that is presented on AXI slave port of the PCIe core will be assigned VCs through the outbound address translation registers.

12.2.2.3.1.2 PCIe PHY Interface

The PCIe subsystem incorporates a 4-lane PCIe compliant PHY (PIPE) interface to connect to a SERDES-based PHY. The PCIe PHY module consist of a SERDES module and a PCIe PCS (Physical Coding Sub-block) module. The SERDES module converts parallel data into PCIe serial signals and the PCIe PCS module provides an industry standard PIPE Interface to PCIe MAC. The frequency of the PIPE interface can be 62.5MHz, 125MHz or 250MHz depending on whether the system is operating in Gen1, Gen2 or Gen3 mode. The width of the PIPE interface remains constant at 32-bits for all modes of operation. The PCIe controller Lane 0 will always be the master lane for any lane width. For more information on the SERDES module, see *Serializer/Deserializer (SerDes)*.

12.2.2.3.1.3 CBA Infrastructure

The PCIe subsystem includes two CBASS modules (CBASS_SLV1 and CBASS_SLV2) to provide access to the various sub-components.

The CBASS_SLV1 is used to access the PCIe egress data region through the VBUSM2AXI bridge and the AXI slave port of the PCIe controller. There are three data regions implemented in the CBASS_SLV1. The PCIE_DAT0 region provides a region with 27 bits of address and PCIE_DAT1 region provides a region with 32 bits of address. These regions are provided to ensure separate memory spaces for 32-bit and 64-bit maps at the SoC level. The PCIE_DAT2 region provides a region with 48 bits of address. This is used to route all requests with *casel* not equal to zero to the PCIe controller slave port. The destination for all these three regions is the AXI slave port of the PCIe controller.

The CBASS_SLV2 provides multiple regions to access the VBUSP configuration ports of the sub-components of the PCIe sub-system like the PCIe controller configuration port, USER_CFG, VMAP, CPTS and the ECC aggregators. Having multiple regions on the VBUSP configuration port enables the SoC infrastructure to have hardware firewalls that can provide secure access to configuration registers.

The SERDES PLL should be initialized prior to accessing the PCIe core and ECC_AGGR1 configuration registers. Any accesses to these two regions prior to initializing the SERDES PLL will complete with a read/write status error in CBASS_SLV2.

Copyright © 2024 Texas Instruments Incorporated

12.2.2.3.1.4 VBUSM to AXI Bridges

A VBUSM2AXI protocol conversion bridge is attached to the each AXI slave port. This bridge translates all incoming data requests and sends them to the AXI slave port of the PCIe controller core. Wherever required, it also generates side band signals that the AXI bridge needs based upon the address and the address space of the transaction on the device's native bus.

12.2.2.3.1.5 AXI to VBUSM Bridges

The AXI master port of the PCIe core has an AXI2VBUSM bridge that is used for high speed data transfer during data read/write transactions originated by remote devices over the PCIe link. This bridge provides the protocol translation between the AXI master interface of the PCIe controller core and the VBUSM interface used in the device.

12.2.2.3.1.6 VBUSP to APB Bridge

A VBUSP2APB bridge is used to convert the device VBUSP configuration access to the APB protocol supported by the PCIe controller core.

12.2.2.3.1.7 Custom Logic

The PCIe subsystem includes custom logic to implement Precision Time Management (PTM), interrupts and user configuration registers.

12.2.2.3.2 PCIe Subsystem Reset Schemes

This section describes the reset schemes supported by PCIe Base Specification and the way these resets are supported by the PCIe Subsystem. PCIe Base Specification specifies two mechanisms for performing reset – Conventional Reset mechanism and Functional Level Reset mechanism.

12.2.2.3.2.1 PCIe Conventional Reset

There are three distinct types of Conventional Reset and each of these causes the hardware state machines, hardware logic, port states and configuration registers to be initialized to their default values.

1. Cold Reset – The reset occurring at power-up of the device is referred to as Cold Reset. Cold Reset is triggered by the PERSTn signal being asserted. PERSTn signal is an auxiliary signal in PCIe Specification and can be used at power on reset for the device that has PCIe as its primary bus interface to rest of the system. The device is allowed to generate its own power-on reset as long as the PCIe requirements for PERSTn are met. See PCIe Base Specifications for details.
2. Warm Reset – A reset can be triggered by the hardware without the removal of power from the device. This reset is referred to as Warm Reset. The hardware implementation is not specified by the PCIe Specification.
3. Hot Reset – The PCIe Specifications provides an in-band mechanism to propagate a Conventional Reset across a Link. This reset, called the Hot Reset, propagates via the transmission of TS1 Ordered Sets. In general, Hot Reset is software controlled procedure and can only be issued by Root Port in a PCIe network as the propagation is downstream only. PCIe subsystem translates the received in-band hot reset into an interrupt that can then be used by software to reset the PCIe subsystem.

After a conventional reset, the software must wait at least 100 ms before attempting any PCIe transaction on the device that has been reset. If the downstream device does not respond to transaction packets, it must not give up until 1 second plus an additional 50% (0.5 second) time is lapsed.

12.2.2.3.2.2 PCIe Function Level Reset

The Function Level Reset (FLR) is an optional in-band reset mechanism that is used to reset one particular function in a PCIe device. The PCIe core will initate the appropriate function level reset process when it receives the FLR message. The PCIe subsystem will raise an interrupt to the SoC, based on the FLR_IN_PROGRESS[5:0] and VF_FLR_IN_PROGRESS[15:0] status outputs from the PCIe core. Software can update the PCIE_USER_FLR_DONE[5-0] FLR_DONE and/or PCIE_USER_VF_FLR_DONE[15-0] VF_FLR_DONE register bits to signal to the PCIe core that the application layer FLR processing is complete.

12.2.2.3.2.3 PCIe Reset Isolation

It can be important to isolate reset of PCIe subsystem from reset of the rest of the device. The procedure to implement this depends upon what functionality the PCIe subsystem is providing – Root Port or End Point.

Copyright © 2024 Texas Instruments Incorporated

12.2.2.3.2.3.1 Root Port Reset with Device Not Reset

When PCIe subsystem is operating as a Root Port, it is the master controller on the PCIe subsystem. When PCIe subsystem is to be reset in this operating mode, the link to the downstream device will get disconnected. To accomplish this reset, the PCIe subsystem RP should issue a hot reset to End Point or Switch downstream. It should also stop any ongoing transactions. Then, a PCIe subsystem reset can be issued. The sequence of events is outlined below:

1. Stop transactions at system and application level. This is recommended for graceful suspension of activity at system level. It is not required by PCIe subsystem though. Not choosing to gracefully stop transaction would cause some of the outstanding transactions to complete in error and it may be harmful from software stability standpoint.
2. Disable “Bus Master Enable” for End Points (see Section 12.2.2.3.6.2, *PCIe Transaction Limitations*). Note that a hand shake with EP Software may be required as not all End Point application software will automatically become aware of this disable action from RP.
3. Optionally, issue Hot Reset to End Point devices via PCIE_USER_RSTCMD[0] INIT_HOT_RESET bit . Note that the link will be getting disconnected. So, a reset may be happening regardless of the Hot Reset command from RP. It depends on the architecture of the other (external) device. In devices with PCIe subsystem, link disconnection automatically results in a reset request interrupt. A PCIe subsystem reset is required to get the memory buffers out of internal flush modes.
4. Initiate Clock Stop sequence and wait for Acknowledgement. For more information, see Section 12.2.2.3.3.1, *CBA Power Management.*
5. Issue reset to PCIe subsystem (local reset).
6. Reinitialize PCIe subsystem and downstream devices. PCIe bus enumeration must be performed again.

Note that it is possible for Root Port to occasionally see the downstream device going down for some reason and disconnecting. Such events typically occur due to an internal error condition in the End Point. When such an event occurs, the sequence of events for the Root Port will be as follows:

1. The PCIe subsystem in RP mode will issue the “Reset Request” interrupt when it detects link getting disconnected unexpectedly.
2. The PCIe subsystem will flush all master transactions and any completion data from pending transactions. It will also start completing slave transactions with error completions.
3. After servicing the interrupt and disabling further interrupts, the system software must reset PCIe subsystem Root Port (see steps #4 and later specified previously in this section).

12.2.2.3.2.3.2 Device Reset with Root Port Not Reset

As a Root Port, PCIe core is the master of PCIe subsystem. If the device hardware and the software is reset and re-initialized, the system does not get additional benefit of keeping the PCIe subsystem alive as all context information is lost. So, this mode is not supported.

12.2.2.3.2.3.3 End Point Device Reset with Root Port Not Reset

It may be desirable to isolate the reset only to PCIe subsystem when it is operating as an End Point. Such resets may be implemented when a Hot Reset command is received from upstream device (Root Port or Switch). The sequence to be followed is as outlined below.

1. Hot Reset is received. The “Request Reset” interrupt is triggered.
2. The PCIe subsystem will automatically enter Flush Mode. The PCIe Link Training and Status State Machine (LTSSM) will be disabled automatically.
3. PCIe subsystem will complete all master transactions and any completion data from pending transactions will be accepted and discarded. It will also start issuing error response for new slave transactions.
4. Upon receipt of the reset interrupt from PCIe subsystem, the system software must immediately start preparing for shut down of PCIe subsystem. The DMA or software process accessing PCIe subsystem should gracefully suspend operations. Otherwise, step #3 may continue for unduly long time and Root Port may assume that the EP is non-responsive to its link re-training attempts.
5. Read the Reset Command register (PCIE_USER_RSTCMD) to check if the bridge activity flush bit is zero. This indicates that it is safe to issue warm reset to PCIe subsystem.
6. Initiate Clock Stop sequence. This will ensure no outstanding transactions exist.
7. Issue a PCIe subsystem Reset.

Copyright © 2024 Texas Instruments Incorporated

8. Resume initialization sequence.

12.2.2.3.2.3.4 Device Reset with End Point Device Not Reset

As an End Point, the PCIe subsystem does not need to stay operational when the device reset is happening. Upon detecting the link disconnection, the upstream port will re-train the link when Root Port issues a link retrain command to itself or to a switch port next to the EP.

If it is required that the PCIe subsystem be operational (without any transactions though), the End Point must negotiate with the other devices (primarily Root Port) to stop activity. Otherwise, if the End Point goes into force idle/standby or clock stop modes without properly managing the process, there will be timeouts or errors on incoming read transactions and writes will be completely lost. In such situation, the Root Port will encounter excessive errors and may issue a hot reset to the End Point anyway.

12.2.2.3.2.4 PCIe Reset Limitations

Once a reset situation occurs, the PCIe subsystem prepares for reset assertion from the device level reset controller. In this mode, all outbound write transactions are discarded and all outbound reads are returned with error. Similarly, all inbound reads/writes are discarded and any pending reads are completed but data is discarded. Additionally, any register reads on local or remote registers regions are returned in error even through data may be correct. Register writes on local registers are executed correctly.

Whenever the reset interrupt is asserted by PCIe subsystem, the host controller should clear the interrupt, perform a clock stop request/ack sequence and issue a local reset to PCIe subsystem.

12.2.2.3.2.5 PCIe Reset Requirements

Whenever link goes from connected to disconnected state, the PCIe subsystem requires re-initialization based on the events occurring in PCIe core. A request for reset is typically going to be issued and will manifest as a link down interrupt. Since loopback requires transitions through link connect and disconnect, it will generate a link down interrupts. Typically, issuing the warm reset (module reset for CBA) would be sufficient.

12.2.2.3.3 PCIe Subsystem Power Management

PCIe has multiple power management protocols. Some of them are invoked by the hardware, such as Active State Power Management (ASPM), while others are activated at higher levels via software.

The PCIe core supports D0, D1 and D3-Hot power states.

Link power states L0, L0s, L1, L1s are supported.

L0s entry and exit is managed by the PCIe core if ASPM L0s is enabled.

L1 entry and exit is also managed by the PCIe core if ASPM L1 is enabled. Software can force the exit from L1 by writing to the PCIE_USER_PMCMD[0] CLIENT_REQ_EXIT_L1 bit in the PCIe subsystem. Entry into L1 can also be blocked by setting the PCIE_USER_PMCMD[0] CLIENT_REQ_EXIT_L1 bit.

L1s support requires that the CLKREQ pin be connected to the remote peer. L1s power state is entered when the link is in L1 and CLKREQ pin is de-asserted. Software can force the exit from L1s by writing to either the PCIE_USER_PMCMD[0] CLIENT_REQ_EXIT_L1 or PCIE_USER_PMCMD[1] CLIENT_REQ_EXIT_L1_SUBSTATE bits in the PCIe subsystem.

12.2.2.3.3.1 CBA Power Management

The power management for PCIe subsystem with CBA interface is primarily accomplished through the clock stop protocol. Any time the clock stop protocol is initiated, the PCIe subsystem will acknowledge after making sure that there are no outstanding transactions. If there are any pending transactions in the subsystem, then the clock stop acknowledgement procedure cannot be completed. Therefore, it is necessary that the system software first suspend activity on the serial link as well as on the CBA interface. To do so, it is expected that the devices communicating with the device on which clock stop procedure is to be performed agree to stop transactions targeted to the device in question. Similarly, it is required that the outgoing transactions also be stopped to successfully complete the clock stop sequence. The PCIe subsystem will guarantee that in the event there are pending transactions that are still in process of draining will prevent the clock-stop acknowledgement to be issued to CBA subsystem power management logic.

Copyright © 2024 Texas Instruments Incorporated

Note

The PCIe subsystem does not have ability to terminate the clock stop state. Therefore, any wakeup sequence can only be initiated through other means such as software driven timers or software detected events occurring outside of PCIe subsystem.

12.2.2.3.4 PCIe Subsystem Interrupts

The PCIe subsystem provides a mix of interrupts from the PCIe controller interrupt, CPTS interrupts, and interrupts generated in the subsystem using the internal Interrupt Distributor (CP_INTD) module.

The CP_INTD module is used to generate and aggregate interrupts from various status signals of the PCIe core. Interrupts from the PCIe core and CPTS module are ported out directly to the PCIe subsystem boundary without any aggregation. Table 12-117 shows the details on the PCIe subsystem interrupt outputs.

Table 12-117. PCIe Subsystem Interrupt Events

Interrupt Name	Clock Dependency	Description
PCIE_LEGACY_PULSE	CBA_CLK	PCIe legacy interrupt. INTA_OUT, INTB_OUT, INTC_OUT and INTD_OUT status outputs from the PCIe core are aggregated. Note: Valid in RP mode only.
PCIE_ERROR_PULSE	CBA_CLK	PCIe error interrupt. FATAL_ERROR_OUT, NON_FATAL_ERROR_OUT and CORRECTABLE_ERROR_OUT status outputs from the PCIe core are aggregated. Note: Valid in both RP and EP modes.
PCIE_ASF_FATAL_LEVEL	CBA_CLK	PCIe internal diagnostics fatal interrupt. The ASF_INT_FATAL from the PCIe controller is brought out directly without any aggregation. Note: Valid in both RP and EP modes.
PCIE_ASF_NONFATAL_LEVEL	CBA_CLK	PCIe internal diagnostics nonfatal interrupt. The ASF_INT_NONFATAL from the PCIe controller is brought out directly without any aggregation. Note: Valid in both RP and EP modes.
PCIE_FLR_PULSE	CBA_CLK	PCIe Function Level interrupt. FLR_IN_PROGRESS[5:0] and VF_FLR_IN_PROGRESS[15:0] status outputs from the PCIe core are aggregated. Note: Valid in EP mode only.
PCIE_DOWNSTREAM_PULSE	CBA_CLK	PCIe downstream interrupt. • F0_VSEC_INTERRUPT_OUT, • F1_VSEC_INTERRUPT_OUT, • F2_VSEC_INTERRUPT_OUT, • F3_VSEC_INTERRUPT_OUT, • F4_VSEC_INTERRUPT_OUT and • F5_VSEC_INTERRUPT_OUT status outputs from the PCIe controller are aggregated. Note: Valid in EP mode only
PCIE_HOT_RESET_PULSE	CBA_CLK	PCIe hot reset interrupt. HOT_RESET_OUT status output from the PCIe controller is aggregated. Note: Valid in EP mode only.
PCIE_LINK_STATE_PULSE	CBA_CLK	PCIe link state interrupt. LINK_DOWN_RESET_OUT status from the PCIe core is aggregated. Note: Valid in both RP and EP modes.
PCIE_PWR_STATE_PULSE	CBA_CLK	PCIe power state interrupt. POWER_STATE_CHANGE status output from the PCIe core are aggregated. Note: Valid in both EP and RP modes.
PCIE_DPA_PULSE	CBA_CLK	PCIe dynamic power allocation interrupt. DPA_INTERRUPT[5:0] status output from the PCIe core are aggregated. Note: Valid in both EP and RP modes.

Copyright © 2024 Texas Instruments Incorporated

Table 12-117. PCIe Subsystem Interrupt Events (continued)

Interrupt Name	Clock Dependency	Description
PCIE_LOCAL_LEVEL	CBA_CLK	PCIe local interrupt. The LOCAL_INTERRUPT from the PCIe controller is brought out directly without any aggregation. Note: Valid in both RP and EP modes.
PCIE_PHY_LEVEL	CBA_CLK	PCIe PHY interrupt. The PHY_INTERRUPT_OUT from the PCIe controller is brought out directly without any aggregation. Note: Valid in both RP and EP modes.
PCIE_PTM_VALID_PULSE	CBA_CLK	PCIe PTM valid interrupt. PTM_LOCAL_TIMER_OUT_VALID status output from the PCIe controller is aggregated. Note: Valid only in EP mode.
PCIE_ECC0_UNCORR_PULSE	CBA_CLK	PCIe ECC Aggregator0 uncorrected pulse interrupt.
PCIE_ECC0_UNCORR_LEVEL	CBA_CLK	PCIe ECC Aggregator0 uncorrected level interrupt.
PCIE_ECC0_CORR_PULSE	CBA_CLK	PCIe ECC Aggregator0 corrected pulse interrupt.
PCIE_ECC0_CORR_LEVEL	CBA_CLK	PCIe ECC Aggregator0 corrected level interrupt.
PCIE_ECC1_UNCORR_PULSE	LANE0_TXMCLK	PCIe ECC Aggregator1 uncorrected pulse interrupt.
PCIE_ECC1_UNCORR_LEVEL	LANE0_TXMCLK	PCIe ECC Aggregator1 uncorrected level interrupt.
PCIE_CPTS_PEND_INTR	CBA_CLK	PCIe CPTS level interrupt.

Copyright © 2024 Texas Instruments Incorporated

12.2.2.3.4.1 Interrupts Aggregation

The PCIe subsystem includes the CP_INTD module to aggregate some of the PCIe controller signals into the subsystem interrupts indicated in Table 12-117. The interrupt aggregator takes both level and pulse signals from the PCIe core and produces the aggregated pulse interrupt outputs.

The aggregator also supports the End of Interrupt (EOI) feature. EOI can be used to re-trigger a pulse interrupt output, if a PCIe controller level signal is still asserted at the end of interrupt processing. The re-triggering of the the pulse interrupt can be achieved by writing the specified EOI vector value to the PCIE_USER_EOI_VECTOR register.

Table 12-118. PCIe Controller Interrupts Aggregation - 1

Aggregated Interrupt	CP_INTD Registers and Bits Mapping		EOI_VECTOR	PCIe Controller Signal	Description
	PCIE_INTD_ENABLE_REG_SYS_0 PCIE_INTD_ENABLE_CLR_REG_SYS_0 PCIE_INTD_STATUS_REG_SYS_0[(1)]				
PCIE_DOWNSTREAM_PULSE	0	pcie_downstream_0	0	down_pf0_intr	Downstream PF0 interrupt (EP mode only)
	1	pcie_downstream_1		down_pf1_intr	Downstream PF1 interrupt (EP mode only)
	2	pcie_downstream_2		down_pf2_intr	Downstream PF2 interrupt (EP mode only)
	3	pcie_downstream_3		down_pf3_intr	Downstream PF3 interrupt (EP mode only)
	4	pcie_downstream_4		down_pf4_intr	Downstream PF4 interrupt (EP mode only)
	5	pcie_downstream_5		down_pf5_intr	Downstream PF5 interrupt (EP mode only)
-	31-6	Reserved	-	-	-

(1) Register is not used since all PCIe controller signals being aggregated are of type 'level'.

Copyright © 2024 Texas Instruments Incorporated

Table 12-119. PCIe Controller Interrupts Aggregation - 2

Aggregated Interrupt	CP_INTD Registers and Bits Mapping		EOI_VECTOR	PCIe Controller Signal	Description
	PCIE_INTD_ENABLE_REG_SYS_1 PCIE_INTD_ENABLE_CLR_REG_SYS_1 PCIE_INTD_STATUS_REG_SYS_1				
PCIE_FLR_PULSE	0	pcie_flr_0	1	flr0_in_progress	PF0 function-level reset (EP mode only)
	1	pcie_flr_1		flr1_in_progress	PF1 function-level reset (EP mode only)
	2	pcie_flr_2		flr2_in_progress	PF2 function-level reset (EP mode only)
	3	pcie_flr_3		flr3_in_progress	PF3 function-level reset (EP mode only)
	4	pcie_flr_4		flr4_in_progress	PF4 function-level reset (EP mode only)
	5	pcie_flr_5		flr5_in_progress	PF5 function-level reset (EP mode only)
	6	pcie_flr_6		vf0_flr_in_progress	VF0 function-level reset (EP mode only)
	7	pcie_flr_7		vf1_flr_in_progress	VF1 function-level reset (EP mode only)
	8	pcie_flr_8		vf2_flr_in_progress	VF2 function-level reset (EP mode only)
	9	pcie_flr_9		vf3_flr_in_progress	VF3 function-level reset (EP mode only)
	10	pcie_flr_10		vf4_flr_in_progress	VF4 function-level reset (EP mode only)
	11	pcie_flr_11		vf5_flr_in_progress	VF5 function-level reset (EP mode only)
	12	pcie_flr_12		vf6_flr_in_progress	VF6 function-level reset (EP mode only)
	13	pcie_flr_13		vf7_flr_in_progress	VF7 function-level reset (EP mode only)
	14	pcie_flr_14		vf8_flr_in_progress	VF8 function-level reset (EP mode only)

Copyright © 2024 Texas Instruments Incorporated

Table 12-119. PCIe Controller Interrupts Aggregation - 2 (continued)

Aggregated Interrupt	CP_INTD Registers and Bits Mapping		EOI_VECTOR	PCIe Controller Signal	Description
	PCIE_INTD_ENABLE_REG_SYS_1 PCIE_INTD_ENABLE_CLR_REG_SYS_1 PCIE_INTD_STATUS_REG_SYS_1				
	15	pcie_flr_15		vf9_flr_in_progress	VF9 function-level reset (EP mode only)
	16	pcie_flr_16		vf10_flr_in_progress	VF10 function-level reset (EP mode only)
	17	pcie_flr_17		vf11_flr_in_progress	VF11 function-level reset (EP mode only)
	18	pcie_flr_18		vf12_flr_in_progress	VF12 function-level reset (EP mode only)
	19	pcie_flr_19		vf13_flr_in_progress	VF13 function-level reset (EP mode only)
	20	pcie_flr_20		vf14_flr_in_progress	VF14 function-level reset (EP mode only)
	21	pcie_flr_21		vf15_flr_in_progress	VF15 function-level reset (EP mode only)
PCIE_LEGACY_PULSE	22	pcie_legacy_0	2	int0	Legacy interrupt A (RP mode only)
	23	pcie_legacy_1		int1	Legacy interrupt B (RP mode only)
	24	pcie_legacy_2		int2	Legacy interrupt C (RP mode only)
	25	pcie_legacy_3		int3	Legacy interrupt D (RP mode only)
PCIE_PWR_STATE_PULSE	26	pcie_pwr_state	3	power_state_change_interrupt	Power state change to D1 or D3
-	31-27	Reserved	-	-	-

Copyright © 2024 Texas Instruments Incorporated

Table 12-120. PCIe Controller Interrupts Aggregation - 3

Aggregated Interrupt	CP_INTD Registers and Bits Mapping		EOI_VECTOR	PCIe Controller Signal	Description
	PCIE_INTD_ENABLE_REG_SYS_2 PCIE_INTD_ENABLE_CLR_REG_SYS_2 PCIE_INTD_STATUS_REG_SYS_2 PCIE_INTD_STATUS_CLR_REG_SYS_2				
PCIE_DPA_PULSE	0	pcie_dpa_0	N/A(1)	dpa_intr0	PF0 DPA power state change (EP mode only)
	1	pcie_dpa_1		dpa_intr1	PF1 DPA power state change (EP mode only)
	2	pcie_dpa_2		dpa_intr2	PF2 DPA power state change (EP mode only)
	3	pcie_dpa_3		dpa_intr3	PF3 DPA power state change (EP mode only)
	4	pcie_dpa_4		dpa_intr4	PF4 DPA power state change (EP mode only)
	5	pcie_dpa_5		dpa_intr5	PF5 DPA power state change (EP mode only)
PCIE_ERROR_PULSE	6	pcie_error_0	N/A(1)	correctable_error	Correctable error
	7	pcie_error_1		non_fatal_error	Non-Fatal error
	8	pcie_error_2		fatal_error	Fatal error
PCIE_HOT_RESET_PULSE	9	pcie_hot_reset	N/A(1)	hot_reset	Hot reset (EP mode only)
PCIE_LINK_STATE_PULSE	10	pcie_link_state	N/A(1)	link_state	Link down reset
PCIE_PTM_VALID_PULSE	11	pcie_ptm	N/A(1)	ptm_local_timer_valid	PTM local timer valid (EP mode only)
-	31-12	Reserved	-	-	-

(1) EOI is not used since the input signal is of type 'pulse'.

Copyright © 2024 Texas Instruments Incorporated

12.2.2.3.4.2 Interrupt Generation in EP Mode

When PCIe subsystem is operating as an End Point (EP), either the legacy interrupts, MSI or MSI-X interrupts can be triggered to the upstream ports (eventually leading to an interrupt in RP device). As per PCIe Specifications, each PCIe function may generate only one of the Legacy or MSI interrupt types as decided during configuration period.

12.2.2.3.4.2.1 Legacy Interrupt Generation in EP Mode

The End Point (EP) can trigger generation of a PCI Legacy Interrupt at the Root Port via an in-band Assert_INTx / Deassert_INTx messages (where x = A, B, C, or D). Software can write to the PCIE_USER_LEGACY_INTR_SET register to trigger the required INTx (where x = A, B, C, or D) assert and de-assert message from the PCIe core. Once an assert message has been generated, it cannot be generated again until a deassert message is generated. Thus, only one interrupt can be pending at a time.

There is no hardware input port provided that will allow generation of legacy interrupts on the EP port.

Note

The interrupt messaging mechanism makes it unfeasible to guarantee a time of delivery of the interrupt unlike in conventional designs where the interrupt line is often electrically connected to the final destination.

12.2.2.3.4.2.2 MSI and MSI-X Interrupt Generation

In the case of MSI interrupts signaling, the interrupt conditions are communicated from the EP to the RP via messages. Thus, upon the occurrence of an interrupt condition, a message is sent by the EP with information that identifies the origin of the interrupt. Each message is in the form of a memory write request, containing an address and a data value to be written. Each PCI function supported by a device can be assigned a separate memory address, thus providing separate virtual channels for signaling interrupts generated by each function. In addition, the MSI mechanism allows a maximum of 32 distinct data patterns in the messages generated by each PCI function, and each pattern can be assigned to an interrupt condition within the function.

In the case of MSI-X interrupts signaling, the operation is similar to the MSI mode, except that the mechanism allows a much larger number of distinct interrupt conditions as many as 2048 per function to be communicated, and enables a distinct address to be defined for each of these conditions. This mechanism requires two tables to be stored in the EP memory. The MSI-X table contains the address/data patterns to be used for each interrupt condition (as many as 2048 per function) as well as individual enable/mask bits, and the Pending Bit Array (PBA) stores the status of each interrupt condition. Interrupt conditions are communicated from the EP to the RP via messages (write requests), as in the case of the MSI mode.

12.2.2.3.4.3 Interrupt Reception in EP Mode

The PCIe specification does not have provision for End Points to receive legacy interrupts. As a result, only events other than these are used to map to interrupts.

12.2.2.3.4.3.1 PCIe Core Downstream Interrupts

The Vendor Specific Capability signals of the PCIe core
- F0_VSEC_INTERRUPT_OUT, F1_VSEC_INTERRUPT_OUT, F2_VSEC_INTERRUPT_OUT, F3_VSEC_INTERRUPT_OUT, F4_VSEC_INTERRUPT_OUT and F5_VSEC_INTERRUPT_OUT, are used to generate interrupts to the EP from the RP. The F0_VSEC_INTERRUPT_OUT represents the interrupt for the EP Physical Function 0, the F1_VSEC_INTERRUPT_OUT is the interrupt output for EP Physical Function 1 and so on. These signals are aggregated into the PCIE_DOWNSTREAM_PULSE interrupt to the local host.

The RP can write to the Vendor Specific Control registers (PCIE_CORE_PFn_I_VENDOR_SPECIFIC_CONTROL_REG) to assert these signals at the EP and this will trigger the PCIE_DOWNSTREAM_PULSE interrupt to the EP host.

12.2.2.3.4.3.2 PCIe Core Function Level Reset Interrupts

The PCIE_FLR_PULSE interrupt is asserted to indicate to the host that the PCIe controller has received a function-level reset request from the remote side. The PCIE_FLR_PULSE interrupt is an

Copyright © 2024 Texas Instruments Incorporated

aggregation of the FLR_IN_PROGRESS[5:0] and VF_FLR_IN_PROGRESS[15:0] signals from the PCIe core. Each bit of FLR_IN_PROGRESS[5:0] represents Physical Function0 through 5 and each bit of the VF_FLR_IN_PROGRESS[15:0] represents Virtual Function0 through 15.

Upon assertion of the function level reset interrupt, software will need to write to the PCIE_USER_FLR_DONE[5-0] FLR_DONE bit field within 100ms to acknowledge to the PCIe core that all the application level processing related to the function level reset is complete. Similarly, software will need to acknowledge virtual function level reset completion by writing to the PCIE_USER_VF_FLR_DONE[15-0] VF_FLR_DONE bit field within 100ms of the virtual function level reset being asserted.

12.2.2.3.4.3.3 PCIe Core Power Management Event Interrupts

The PCIE_PWR_STATE_PULSE interrupt is generated to let the software know of the power management events. The PCIE_PWR_STATE_PULSE interrupt is generated by the POWER_STATE_CHANGE_INTERRUPT output of the PCIe core. This interrupt is asserted when the power state of a physical or virtual function is being changed to D1 or D3 state by writing into their Power Management Control register (PCIE_CORE_PFn_I_PWR_MGMT_CTRL_STAT_REP or PCIE_CORE_VFm_I_PWR_MGMT_CTRL_STAT_REP, respectively).

Software can check the PCIE_USER_LINKSTATUS[23-16] POWER_STATE_CHANGE_FUNCTION_NUM register field to determine the physical function for which power state change occurred. The PCIE_USER_PMCMD[2] POWER_STATE_CHANGE_ACK register bit can be used to acknowledge the POWER_STATE_CHANGE_INTERRUPT.

The PCIE_DPA_PULSE interrupt is generated by aggregating the PCIe controller DPA_INTRs interrupt status outputs. This interrupt is asserted in EP mode when there is a configuration write to the dynamic power allocation control register (PCIE_CORE_PFn_I_DPA_CTRL_STATUS_REG) to modify the DPA power state of the device. The DPA_INTR0 is asserted for such an event for PF0, the DPA_INTR1 for PF1 and so on.

12.2.2.3.4.3.4 PCIe Core Hot Reset Request Interrupt

When the link is down, the Root Port may request reset of the End Point. This request is terminated as an PCIE_HOT_RESET_PULSE interrupt to the End Point host software. All outstanding transactions are completed in error on slave port and further transactions are not generated on the master port. Once the transactions are completely stopped, the software should issue a local reset to PCIe subsystem. The re-initialization process may then be started.

The PCIE_HOT_RESET_PULSE interrupt is generated from the HOT_RESET_OUT output of the PCIe core.

12.2.2.3.4.3.5 PTM Valid Interrupt

In EP mode, the PCIe subsystem will generate the PCIE_PTM_VALID_PULSE interrupt to the local CPU after a PTM dialog between the PTM requester (EP) and the PTM responser (RP). The PTM valid interrupt indicates to the CPU in the EP that the local timers have been updated with the timestamp data obtained from the RP. The CPU can read the timer registers inside the EP to determine the current timebase.

The PCIE_PTM_VALID_PULSE interrupt is generated from the PTM_LOCAL_TIMER_OUT_VALID signal from the PCIe core.

12.2.2.3.4.4 Interrupt Generation in RP Mode

As per PCIe Base Specifications, Root Port ports only receive interrupts. There is no mechanism to generate interrupts from RP port to EP mode as per PCIe specification.

However, PCIe subsystem does support generation of interrupts from RP to EP. The downstream interrupts described in Section 12.2.2.3.4.3.1, *PCIe Core Downstream Interrupts* provides a mechanism for the RP to generate software triggered interrupts to the EP.

12.2.2.3.4.5 Interrupt Reception in RP Mode

12.2.2.3.4.5.1 PCIe Legacy Interrupt Reception in RP Mode

The RP can receive any of the four legacy INTx interrupts from the EP. These will trigger the PCIE_LEGACY_PULSE interrupt to the host CPU.

Copyright © 2024 Texas Instruments Incorporated

The PCIE_LEGACY_PULSE interrupt is an aggregation INTA_OUT, INTB_OUT, INTC_OUT and INTD_OUT signals from the PCIe core.

12.2.2.3.4.5.2 MSI/MSI-X Interrupt Reception in RP Mode

The PCIe core decodes all MSI and MSI-X messages received from the link and forwards them on the low-priority AXI master interface. These messages must then be routed to the system interrupt controller by the SoC interconnect (CBASS0).

12.2.2.3.4.5.3 Advanced Error Reporting Interrupt

If enabled by software at the time of enumeration, PCIe subsystem will generate this interrupt to indicate occurrence of errors of various levels of severity. The software can choose not to enable Advanced Error Reporting but if it enables, it must process the interrupt as per PCIe Specifications.

The advanced error reporting interrupt is signaled by the PCIE_ERROR_PULSE interrupt. The PCIE_ERROR_PULSE is an aggregation of the FATAL_ERROR_OUT, NON_FATAL_ERROR_OUT and CORRECTABLE_ERROR_OUT from the PCIe core.

12.2.2.3.4.6 Common Interrupt Reception in RP and EP Modes

These interrupts are common to both RP and EP modes of operation.

12.2.2.3.4.6.1 PCIe Local Interrupt

The LOCAL_INTERRUPT from the PCIe controller is ported out directly to the PCIe subsystem boundary. For more details, see the PCIe specification

12.2.2.3.4.6.2 PHY Interrupt

The PHY_INTERRUPT_OUT from the PCIe controller is ported out directly to the PCIe subsystem boundary. For more details, see the PCIe specification

12.2.2.3.4.6.3 Link down Interrupt

If the PHY link is disconnected, the PCIE_LINK_STATE_PULSE interrupt will be generated. The expected course of action is to reset the entire PCIe subsystem and restart. All application states must also be initialized so that the operations can resume following the reset and renegotiation of the link.

The PCIE_LINK_STATE_PULSE interrupt is generated from the LINK_DOWN_RESET_OUT output of the PCIe core.

12.2.2.3.4.6.4 Transaction Error Interrupts

If there is a timeout on PCIe or an abort, PCIE_PHY_LOCAL_LEVEL interrupt will be issued. The PCIE_PHY_LOCAL_LEVEL interrupt is generated by the aggregation of the LOCAL_INTERRUPT and PHY_INTERRUPT_OUT signals from the PCIe core. For more details, see the PCIe specification.

12.2.2.3.4.6.5 Power Management Event Interrupt

This PCIE_PWR_STATE_PULSE interrupt is generated to let the software know of the power management events.The PCIE_PWR_STATE_PULSE interrupt is generated by aggregating the POWER_STATE_CHANGE_INTERRUPT and DPA_INTERRUPT outputs of the PCIe core.

In EP mode, software can check the PCIE_USER_LINKSTATUS[23-16] POWER_STATE_CHANGE_FUNCTION_NUM bit field to determine the physical function for which power state change occurred.

The PCIE_USER_PMCMD[2] POWER_STATE_CHANGE_ACK bit can be used to acknowledge the POWER_STATE_CHANGE_INTERRUPT.

12.2.2.3.4.6.6 Active Internal Diagnostics Interrupts

The active internal diagnostics interrupt is signaled by the PCIE_ASF_PULSE interrupt. This is an aggregation of the ASF_CSR_ERR, ASF_DAP_ERR, ASF_INTEGRITY_ERR, ASF_INT_FATAL, ASF_INT_NONFATAL, ASF_PROTOCOL_ERR, ASF_SRAM_CORR_ERR, ASF_SRAM_UNCORR_ERR, and ASF_TRANS_TO_ERR signals from the PCIe core.

Copyright © 2024 Texas Instruments Incorporated

12.2.2.3.4.7 ECC Aggregator Interrupts

The PCIE_ECC0_UNCORR_PULSE/PCIE_ECC0_UNCORR_LEVEL and PCIE_ECC0_CORR_PULSE/PCIE_ECC0_CORR_LEVEL interrupts are asserted by the CBA clock domain ECC Aggregator. The PCIE_ECC1_UNCORR_PULSE/PCIE_ECC1_UNCORR_LEVEL interrupts are asserted by the Core clock domain ECC Aggregator. The Core clock domain ECC Aggregator correctable interrupts are not exported since this aggregator is only connected to the parity injection logic and the correctable interrupts for this module will never fire.

12.2.2.3.4.8 CPTS Interrupt

The PCIE CPTS_PEND_INT interrupt is asserted by the CPTS module in the PCIe subsystem to signal a TimeStamp event.

12.2.2.3.5 PCIe Subsystem DMA Support

The PCIe Subsystem has different requirements based upon whether it is operated in Root Port (RP) or End Point (EP) mode. The PCIe subsystem does not have DMA capabilities built into it. It has internal slave and master ports connected to the device-level interconnect.

An external DMA engine can make burst data read/writes on the slave port and the master port on PCIe subsystem can initiate reads/writes to memory on behalf of a remote PCIe device.

The PCIe subsystem does not specify the DMA protocol that is used for data transfers. The software implementations on the two ends of the PCIe link implement a data transfer protocol that is compatible with each other.

As a result, there will be several software drivers required – one driver that will manage the PCIe subsystem in RP mode and another set of drivers that will run on the RP side and will manage each of the End Points connected to the PCIe subsystem RP.

Similarly, when PCIe subsystem is operating as an End Point, there are two drivers – one to manage the PCIe subsystem from the device side and another driver that will run on the device operating as the Root Port.

12.2.2.3.5.1 PCIe DMA Support in RP Mode

When operating in Root Port mode, a DMA controller internal to the device (outside PCIe subsystem) can perform DMA to and from any remote device located on the PCIe subsystem. The memory address of such devices is available to the software via the PCIe bus enumeration procedure. In addition, the PCIe subsystem has a provision to perform memory address translation on outbound requests. Thus, the software is able to map different memory regions in its memory map to correspond to different addresses (and different access types) on the PCIe side.

12.2.2.3.5.2 PCIe DMA Support in EP Mode

When operating as a PCIe End Point, the device will be located in PCIe memory map at location programmed in the Base Address Registers by the PCIe Root device. Any PCIe transactions destined for the device from the upstream ports will get transferred to the master port on the PCIe subsystem. Similarly, any transactions originating from the software will be sent over to PCIe link.

In End Point mode, the PCIe subsystem provides address translation functionality. It is possible to map memory accesses originating on PCIe side to memory accesses with different address on the CBA bus side. These address ranges are configurable through application registers.

12.2.2.3.6 PCIe Subsystem Transactions

12.2.2.3.6.1 PCIe Supported Transactions

PCIe subsystem supports the following transactions:

1. Memory Read/Write in inbound/outbound direction in Root Port (RP) and End Point (EP) modes.
2. I/O Read/Write in outbound direction in RP mode.
3. Configuration Read/Write in outbound direction in RP mode.
4. Configuration Read/Write in inbound direction in EP mode.
5. Message Transactions for interrupts and power management in RP and EP modes.

Copyright © 2024 Texas Instruments Incorporated

The following transactions are not supported:

1. Locked Read transactions and associated messages.
2. Inbound I/O Transaction Layer Packets (TLPs).
3. User defined messages.

12.2.2.3.6.2 PCIe Transaction Limitations

Added

There are some limitations that must be adhered to while issuing transactions to PCIe subsystem internal bus interface.

- System Initialization

> **CAUTION**
>
> The PCIe subsystem operation is completely dependent upon the availability of the SERDES module for PCIe transactions. The SERDES module must be configured and corresponding PLLs must be locked before the PCIe subsystem is able to process any data transactions. Any data accesses to the PCIe subsystem prior to this initialization may result in an internal bus hang condition.

- Remote Configuration and I/O Requests

Since the remote configuration and I/O transaction windows are directly mapped to internal bus space, the software must care to not access these spaces when there is no operational PCIe link. No response may be generated for such transactions. It is recommended that checks be built into software to avoid remote accesses in the absence of an operational link.

- Byte Strobe Limitations

For any type of write transactions, the byte enables can only have a single unbroken string of 1s. In other words, in a transaction, if a byte's write strobe is set, then all following bytes must have write strobe set until the last byte with write enabled. "Holes" or "Zeros" in between the byte enables are not allowed.

Since the internal bus width is greater that 32-bit, the TLP (Transaction Layer Packets) size will not be 1 (PCIe counts in 32-bit units) and therefore, it is through the FBE/LBE (First/Last Byte Enable) that the actual data transfer size is controlled.

- Burst Type Limitations

The PCIe core and respectively the PCIe subsystem does not support 'fixed' or 'wrap' burst types on its Slave or Master port. Transactions that require any burst type other than incremental burst type will result in unspecified behavior, possibly bus lock up.

- Transaction Address Alignment

The PCIe subsystem imposes a limitation of a maximum of 128-byte outbound read/write command. However, if the starting address is not aligned to an 8-byte boundary, then the maximum transaction size is reduced to 120 bytes. This limitation is placed to avoid arithmetic overflow in computing transaction length from CBA to AXI. Unspecified behavior will occur, if misaligned transactions in outbound direction are not limited to a maximum of 120 bytes.

- Read Interleaving

Read interleaving refers to the process of returning split read responses from multiple transactions. This implies that read data is not guaranteed to be sent in sequential order (data for one transaction to be sent completely before the next). The PCIe core will not interleave read responses, if the outbound read command/transaction size does not exceed the maximum transaction size configured in the PCIe core.

12.2.2.3.7 PCIe Subsystem Address Translation

The PCIe subsystem uses the address translation registers in the PCIe core to translate outbound system addresses to PCIe space and inbound PCIe address to a valid system address. Note that internal address translation is different from the Address Translation Services (ATS).

Copyright © 2024 Texas Instruments Incorporated

12.2.2.3.7.1 PCIe Inbound Address Translation

12.2.2.3.7.1.1 Root Port Inbound PCIe to AXI Address Translation

The Root Port inbound PCIe to AXI address translation is performed on memory and IO TLPs. The selection of which address translation registers to use in the translation process is dependent on the BAR match of the incoming TLP. In Root Port mode there are 2 bars, so BAR 0 and BAR1 registers are implemented. There is a BAR7 register which is used as a no match BAR address translation register. In Root Port mode there are 2 bars but a BAR value of 7 will be indicated by HAL2AXI when BAR matching is disabled. Any address that does not match the Root Port BARs will be sent out as a BAR7 TLP. Each BAR register is implemented as two 32-bit registers which are named addr0 and addr1. The "Root Port Inbound PCIe to AXI Address Translation Logic" takes the upper bits from the "Root Port Inbound PCIe to AXI Address Translation Registers" and the lower bits are taken from the inbound PCIe address to form the AXI address. An addr0 [5:0] + 1 number of lower bits are passed from the inbound PCIe address to AXI address. In other words, the number of bits taken from anbound PCIe address is given by the addr0[5:0] + 1 value.

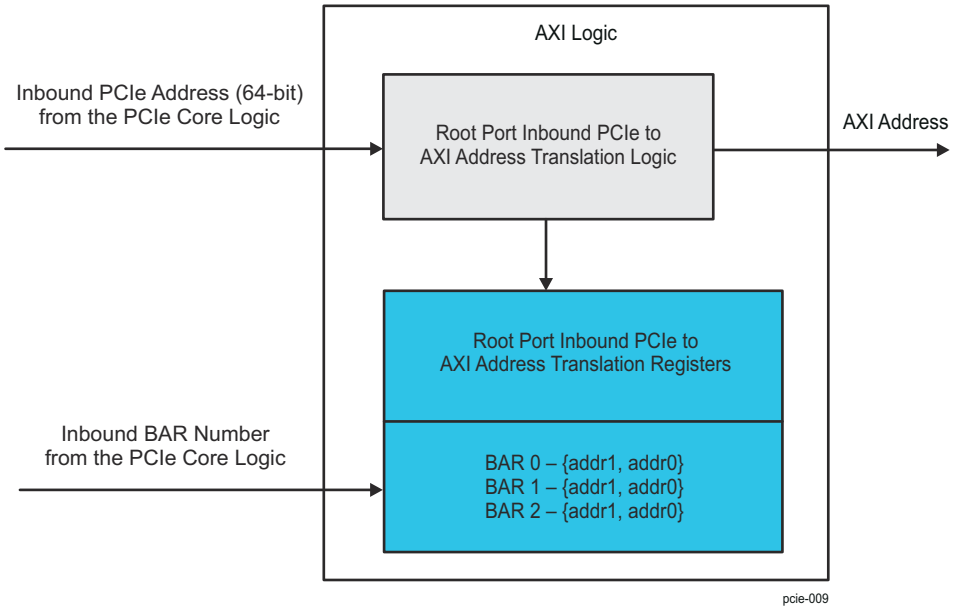


Figure 12-63. PCIE Root Port Inbound PCIe to AXI Address Translation

A set of registers corresponding to one Root Port BAR is shown in Table 12-121.

Table 12-121. PCIE Root Port Inbound PCIe to AXI Address Translation Registers for one BAR

Register Name	Bits	Description	Default Value
addr1 (where BAR can be bar0, bar1 or bar7)	31:0	Upper [63:32] bits of the AXI address.	32'd0
addr0 (where BAR can be bar0, bar1 or bar7)	31:8	Lower [31:8] bits of the AXI address.	24'd0
	7:6	Reserved	2'd0
	5:0	Number of address bits passed through from PCIe to AXI. The PCIe controller passes the programmed value + 1 bits from PCIe to AXI. Minimum value to be programmed into this field is 7 as the lower 8 bits of the base address programmed in these registers (AXI) are replaced by zeros by the Root Port Inbound PCIe to AXI Address Translation Logic.	6'd0

12.2.2.3.7.1.2 End Point Inbound PCIe to AXI Address Translation

The End Point Inbound PCIe to AXI address translation is performed on memory and IO TLPs. The selection of which address translation registers to use in the translation process is dependent on the function number and BAR match of the incoming TLP. In End Point mode there are 7 bars per function, so 7 sets of registers are

Copyright © 2024 Texas Instruments Incorporated

implemented per function, each BAR having two 32-bit registers (addr0 and addr1). The "End Point Inbound PCIe to AXI Address Translation Logic" takes the upper bits from the "End Point Inbound PCIe to AXI Address Translation Registers" and the lower bits are taken from the Inbound PCIe Address to form the AXI address. The number of bits to pass from Inbound PCIe Address to AXI is decided by the Inbound BAR aperture.

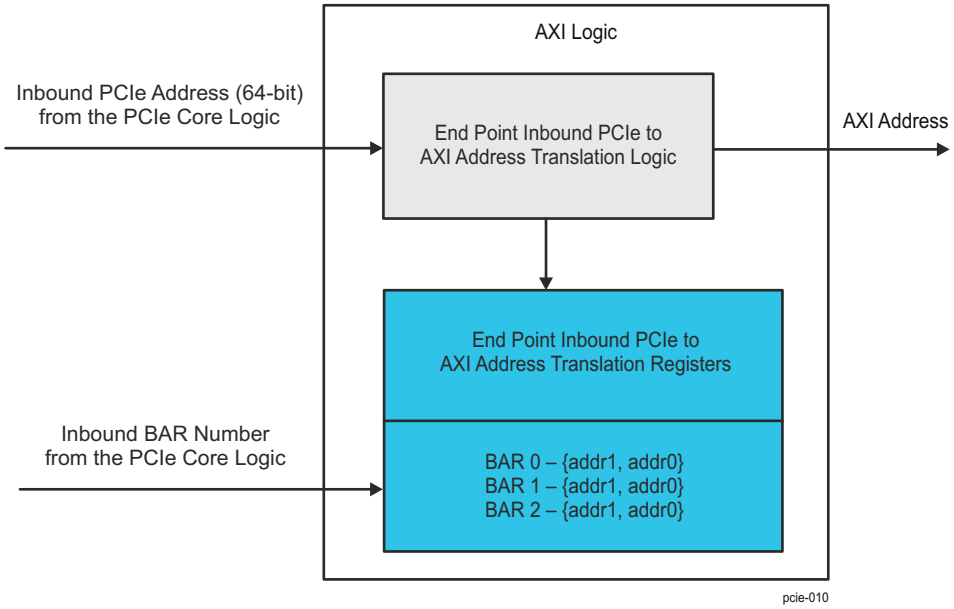


Figure 12-64. PCIE End Point Inbound PCIe to AXI Address Translation

A set of registers corresponding to one End Point BAR is shown in Table 12-122.

Table 12-122. PCIE End Point Inbound PCIe to AXI Address Translation Registers for one BAR

Register Name	Bits	Description	Default Value
addr1 (where BAR can be bar0, bar1, bar2,...bar7, and PF can be pf0, pf1, pf2,... pf21)	31:0	Upper [63:32] bits of the AXI address.	32'd0
addr0 (where BAR can be bar0, bar1, bar2,...bar7, and PF can be pf0, pf1, pf2,... pf21)	31:0	Lower [31:0] bits of the AXI address.	32'd0

12.2.2.3.7.2 PCIe Outbound Address Translation

The PCIe Subsystem allows mapping of PCIe addresses to/from the internal bus addresses of the device. This is accomplished by using internal address translation unit (iATU) in the PCIe core. For each outbound read/write request, the address translation module within PCIe subsystem can convert a VBUSM address to a PCIe address of memory Read/Write type.

If a transaction is large enough that it goes past the address translation region, unspecified behavior may occur. The address translation only works at the time a command is issued. So, a memory write, for example, will not automatically go to next translation region, if it starts in the previous one and is bigger than the remaining size in the starting translation region.

The “Dynamic Method: Sideband Descriptor Based” outbound address translation mechanism is used to bypass the outbound address translation unit under certain conditions. For more details on the bypass operation, see Section 12.2.2.3.7.2.1, *PCIe Outbound Address Translation Bypass*.

12.2.2.3.7.2.1 PCIe Outbound Address Translation Bypass

The PCIe subsystem supports bypassing the outbound address translation in the PCIe controller when the *casel* value on the VBUSM slave interface is non-zero.

Copyright © 2024 Texas Instruments Incorporated

The device UDMA has the ability to send transactions to alternate address maps which are external to the local device. This is achieved by using casel identifier in the DMA descriptors. There are three requirements that need to be met in the PCIe subsystem when using this mode in the DMA, namely:

- Block transactions based on initiator credentials to support FFI (Freedom from Interference)
- Bypass the Address Translation Unit (ATU) inside the PCIe controller since the transaction address is external to the local device
- Assign the correct Bus, Device, Function (BDF) and Traffic Class (TC) values to the transaction

To meet these requirements, the PCIe subsystem uses the *cvirtid* values on the VBUSM slave interface to filter transactions based on initiator credentials. The VMAP block in the PCIe subsystem has a *virtid* match register (PCIE_VMAP_OB_VIRTID_MATCH) and a set of 32 registers that can be used to program the descriptor fields when the ATU logic in the PCIe controller is bypassed.

The PCIe subsystem will filter outbound VBUSM slave transactions that have a non-zero casel value. If the *cvirtid[11:5]* attribute of the VBUSM slave transaction matches the value programmed in the PCIE_VMAP_OB_VIRTID_MATCH register, the credentials of the transaction initiator are verified. The *cvirtid[11:5]* value should also be non-zero for a successful match.

When the credentials match, the ATU of the PCIe controller is bypassed . The *cvirtid[4:0]* is used to select one of the 32 PCIE_VMAP_DESC_j descriptor registers. These registers can be programmed to assign the BDF and TC values to the outbound PCIe transaction. If the PCIE_VMAP_DESC_j[16] BD_EN0 register bit is set, then the bus, device and function numbers are all set from the external descriptor registers. If the BD_EN bit is not set, then the bus and device numbers are assigned by the PCIe controller using the values captured during enumeration. If ARI mode is enabled (in the PCIE_CORE_PFn_I_SRIOV_CTRL_STATUS_REG register), the function number uses the PCIE_VMAP_DESC_j[7:0] DEV_FUNC_NUM register field. If ARI mode is not enabled, the device number uses PCIE_VMAP_DESC_j[7:4] DEV_FUNC_NUM field and the function number uses the [3:0] DEV_FUNC_NUM field. The ARI mode is a PCIe controller hardware configuration option that is enabled for PCIe endpoints that support SR-IOV.

If the credentials of the initiator fail to match the programmed value in the PCIE_VMAP_OB_VIRTID_MATCH register, the VBUSM slave transaction is then flushed by the bridge and is not submitted to the PCIe controller. The VBUSM2AXI2 bridge will return a protection error status for both read and write transactions. In addition, the write data is discarded and the correct number of read data phases with data value of zero are returned.

12.2.2.3.8 PCIe Subsystem Virtualization Support

The PCIe subsystem supports Address Translation Request (ATS) messages in Root Port mode. All PCIe ATS requests are forwarded to the SMMU. The ATS response sent back by the SMMU is forwarded to the remote PCIe requestor.

In End Point mode, the PCIe subsystem supports SR-IOV with 6 physical functions (PF), and 4 virtual functions (VF) per physical function for PF0-PF3 only (6PF/16VF).

12.2.2.3.8.1 End Point SR-IOV Support

In End Point (EP) mode, the PCIe subsystem supports Single Root I/O Virtualization (SR-IOV) with four physical functions (PF) and four virtual functions (VF) per physical function for PF0-PF3 only.

12.2.2.3.8.2 Root Port ATS Support

In Root Port (RP) mode, the PCIe subsystem has an AXI streaming interface (DTI) to connect to the system level SMMU. The Address Translation Messages (ATS) from the End Point is routed to the SMMU over the DTI interface. The translated address and any invalidation requests from the SMMU, sent to the PCIe over the DTI interface, is converted to ATS messages and sent to the End Point.

12.2.2.3.8.3 VirtID Mapping

The PCIe subsystem includes the VMAP module that can be programed to map (convert) the incoming 16-bit PCIe requestID + 2-bit PCIe AT bits to the 16-bit CBA virtID + 2-bit CBA atype attributes.

Figure 12-65 shows the block diagram of the VMAP module. The blocks shown in the diagram are replicated for the read and write AXI channels. However, there is only one set of configuration registers in the VMAP module and these are shared by both the read and write channel logic.

Copyright © 2024 Texas Instruments Incorporated

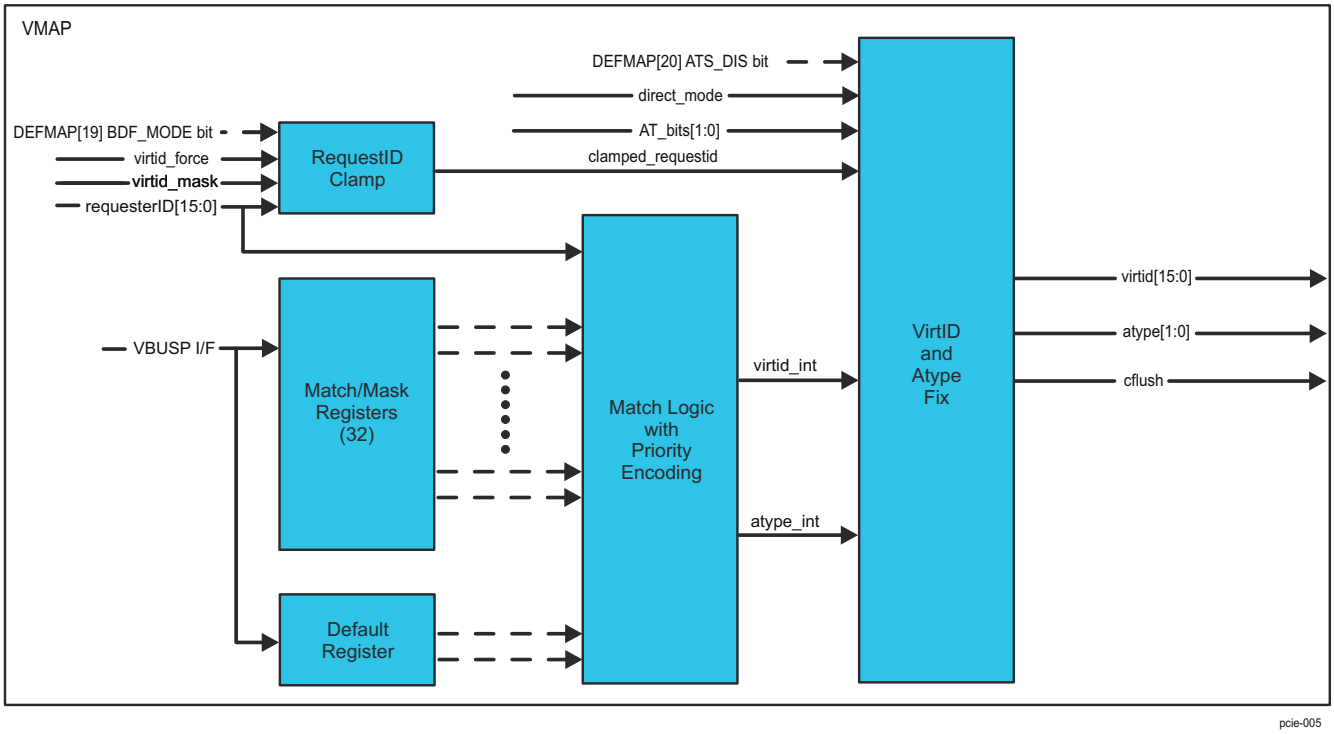


Figure 12-65. PCIe Subsystem VirtID Mapper

The RequestID Clamp block is used to clamp the incoming requestID to a maximum value. This block compares the incoming PCIe requestID[15:12] masked with the virtid_mask[3:0] to an expected value. Bit [19] BDF_MODE in PCIE_VMAP_DEFMAP is used to set the value to virtid_force[3:0] or 4'h0. The clamped_requestid is set to the maximum value of FFFFh if the comparison is false. The clamped_requestid value is later used downstream by the PVU or SMMU in the SoC.

Each of the VMAP modules includes a set of 32 match and mask registers. Software can enable the registers to be used for the mapping by setting the [0] EN bit in PCIE_VMAP_CTRL_j registers. Software can also program the [15-0] RID and [31-16] MASK bit fields in PCIE_VMAP_REQID_j registers to indicate the incoming PCIe requestid that needs to be matched. The [11-0] VID and [17-16] ATYPE bit fields in PCIE_VMAP_VIRTID_j registers are used to program the required CBA virtID and Atype fields in case of a match. In order to prevent a spurious match, the [15-0] RID and [31-16] MASK bit fields in PCIE_VMAP_REQID_j registers should be set up before the [0] EN bit in PCIE_VMAP_CTRL_j registers is set. The same sets of registers are used for both read and write transactions.

The Match logic with Priority Encoding block compares the incoming PCIe requestID with the values programmed in the VMAP control registers and sets the value of the virtid_int and atype_int variables. For every incoming PCIe transaction, the incoming PCIe requestID[11:0] bits are masked with the PCIE_VMAP_REQID_j[31-16] MASK bit field and compared against each of the thirty two values indicated by PCIE_VMAP_REQID_j[15-0] RID that are enabled. The location of the first match of this comparison operation is used to select the [11-0] VID and [17-16] ATYPE bit fields in PCIE_VMAP_VIRTID_j registers. This is used as the virtid_int and atype_int values for the downstream logic. In case none of the enabled [15-0] RID and [31-16] MASK bit fields in PCIE_VMAP_REQID_j registers pairs match the incoming requestID, the value in [11-0] DEF_VID and [17-16] DEF_ATYPE bit fields in PCIE_VMAP_DEFMAP registers is used.

The VirtID and Atype Fix block uses the incoming PCIe AT_bits along with the matched and clamped values of the requestID from the downstream blocks to set the final values of the virtID, catype and cflush attributes for the particular CBA transaction. The values of the virtID, catype and cflush are determined based on the following criteria:

1. If the incoming PCIe TLP has a translated address, as indicated by the PCIe AT_bits equal to 2 and if the match operation results in an expected atype value of 2 and Address Translation Services (ATS) is enabled

Copyright © 2024 Texas Instruments Incorporated

in the system, as indicated by the PCIE_VMAP_DEFMAP[20] ATS_DIS register bit, the virtID is set to the *clamped_requestid* and the *catype* value is set to 2. This enables the PCIe subsystem to pass along the full incoming requestID[15:0] to the upstream SMMU/PVU. The incoming transaction is flushed if it fails this criteria by setting the cflush attribute to 1. The PCIe EP initiating this transaction will be sent a Completion Abort (CA) response for a read request, if the transaction is flushed.

2. If the incoming PCIe TLP does not have a translated address, as indicated by the PCIe AT_bits equal to 0, the virtID can be set to the output of the mask/match registers or the *clamped_requestid*. The *catype* attribute for this transaction can be set to the value from the mask/match registers.

Note

The PCIE implementation in the device uses the *direct_mode* workaround. This forces the *cvirtID* and *catype* to 0 when the incoming transaction matches the criteria outlined in (1) above.

The system Interconnect only supports 12-bits of *virtid*. As a result, only *virtid[11:0]* from the PCIe VMAP module will be used in the system and *virtid[15:12]* is not connected.

The pseudo-code for the virtID mapping algorithm implemented is described below.

```
// Requestid Clamp
// Note: only the low 4 bits of virtid_force & virtid_mask are used in the current PCIe wrapper
// virtid_mask[7:4] should always be 1
// virtid_mask[n] of 1 means use the virtid_force value for this bit
// virtid_mask[n] of 0 means use the requestID value for this bit, virtid_force[n] should be 0
// Bit [19] BDF_MODE in PCIE_VMAP_DEFMAP specifies if we are using
// 0 based bus numbers (0) or offset bus numbers (1)
expected_value[3:0] = PCIE_VMAP_DEFMAP[19] BDF_MODE ? virtid_force[3:0] : 4'h0;
if (requestID[15:12] & virtid_mask[3:0] == expected_value[3:0] )
{
clamped_requestID[15:0] = requestID[15:0];
}
else
{
clamped_requestID[15:0] = FFFFh];
}
// Match Logic with Priority Encoding
for n in 0 to 31 do
if (PCIE_VMAP_CTRL_j[0] EN == 1h &&
(requestID & PCIE_VMAP_REQID_j[31-16] MASK) == PCIE_VMAP_REQID_j[15-0] RID )
{
found[n] = true;
}
else
{
```


Copyright © 2024 Texas Instruments Incorporated

```
found[n] = false;
}
done
// priority encoder for 32 results from above
// uses lowest numbered match
// also outputs a bool specifying if any match was found
all_miss = true;
for n in 0 to 31 do
if (found[n])
{
virtid_int[11:0] = PCIE_VMAP_VIRTID_j[11-0] VID;
atype_int[1:0] = PCIE_VMAP_VIRTID_j[17-16] ATYPE;
all_miss = false;
break;
}
done
// Use the default if no match is found
if (all_miss)
{
virtid_int[11:0] = PCIE_VMAP_DEFMAP[11-0] DEF_VID;
atype_int[1:0] = PCIE_VMAP_DEFMAP[17-16] DEF_ATYPE;
}
// VirtID and Atype Fix
if (AT_bits == 0x2) // Is this a pre-translated address?
{
// Is this EP using SMMU and is ATS enabled?
if (atype_int == 2h && PCIE_VMAP_DEFMAP[20] == 0h )
{
if (direct_mode == 1h) //
{
// yes, send directly to destination address
atype[1:0] = 2'h0; // Direct
virtid[15:0] = 16'h0; // should not matter
flush = 1'b0; // don't turn on the flush bit
at_cba = 1'b0; // Not used
}
else
```


Copyright © 2024 Texas Instruments Incorporated

```
{
// no, send to SMMU TBU for processing
atype[1:0] = 2'h2; // //SMMU
virtid[15:0] = clamped_requestID[15:0]; //Use the correct StreamID
flush = 1'b0; // don't turn on the flush bit
at_cba = 1'b1; // Indicate a pre-translated address
}
}
else
{
// All other cases should error the transaction for pre-translated transaction
// PVU & Direct should not see any pre-translated requests
// SMMU with ATS disables also errors
atype[1:0] = 2'h2; // use VirtSS to force the error
virtid[15:0] = 16'h0; // should not matter
flush = 1'b1; // force the error
at_cba = 1'b1; // should nott matter
}
}
else
{
// This transaction is not pre-translated
atype[1:0] = atype_int //use the atype from the register
flush = 1'b0; // don't force an n error
at_cba = 1'b0; // this is not pre-e- translated
if (atype_int == 2'h2)
{
// this is SMMU, use the clamped RequestID
virtid[15:0] = clamped_requestID[15:0];
}
else
{
// this is not SMMU, use the value from the register
virtid[11:0] = virtid_int[11:0];
virtid[15:12] = 4'h0;
}
}
```


Copyright © 2024 Texas Instruments Incorporated

12.2.2.3.9 PCIe Subsystem Quality-of-Service (QoS)

The Virtual Channel/Transfer Class (VC/TC) feature in the PCIe core, in conjunction with the device system CBASS Quality-of-Service (QOS) capabilities, provide a mechanism for supporting differentiated QOS within the PCIe subsystem. The policy for traffic differentiation is determined by the Transfer Class (TC) and Virtual Channel (VC) mapping and VC-based arbitration mechanism.

A Virtual Channel is established when one or more TCs are associated with a physical resource designated by a VC ID. Every Traffic Class that is supported on a given path within the subsystem must be mapped to one of the enabled Virtual Channels. Every Port must support the default TC0/VC0 pair – this is “hardwired.” Any additional TC mapping or additional VC resource enablement is optional. The number of VC resources provisioned within a component or enabled within a given subsystem may vary due to implementation and usage model requirements.

The PCIe core in the PCIe subsystem is configured to support four virtual channels (4VC/4TC). For both ingress and egress traffic, VC3, the highest numbered virtual channel, has the highest priority.

For ingress traffic, the TC information from each transaction on the AXI master information is mapped to the CCHANID signal of the VBUSM master interface. The system level interconnect can use the CCHANID along with the ORDERID for QoS purposes. Table 12-123 shows the TC mapping to the 12-bit CCHANID of each VBUSM master interface.

Table 12-123. PCIe Subsystem QoS Ingress CCHANID Mapping

TC Value	CCHANID[11:0]
0	{9’d0, 3’b000}
1	{9’d0, 3’b001}
2	{9’d0, 3’b010}
3	{9’d0, 3’b011}
4	{9’d0, 3’b110}
5	{9’d0, 3’b101}
6	{9’d0, 3’b110}
7	{9’d0, 3’b111}

For egress traffic requiring the highest priority it should be mapped to VC=3 in the PCIe controller. The VC mapping is done using the AXI outbound descriptor registers in the PCIe controller.

12.2.2.3.10 PCIe Subsystem Precision Time Measurement (PTM)

The Precision Time Measurement (PTM) enables precise coordination of events across multiple components with independent local time clocks. Ordinarily, such precise coordination would be difficult given that individual time clocks have differing notions of the value and rate of change of time. To work around this limitation, PTM enables components to calculate the relationship between their local times and a shared PTM Master Time: an independent time domain associated with a PTM Root.

The PTM defines the following components:

- PTM Requester - A Function capable of using PTM as a consumer associated with an Endpoint.
- PTM Responder - A Function capable of using PTM to supply PTM Master Time associated with a RP.
- Time Source - A local clock associated with a PTM Responder.
- PTM Root – The source of PTM Master Time for a PTM hierarchy. A PTM Root must also be a Time Source and is typically also a PTM Responder.

When using PTM between two components on a Link, the EP sends PTM requests to the RP on the same link. During each dialog, the RP populates the PTM Response message based on timestamps stored during previous PTM dialogs. Once each component has historical timestamps from the preceding dialog, the EP can combine its timestamps with those passed in the PTM Response message to calculate the PTM Master Time.

The PCIe core implements all of the features required to handle the PTM conversation between the requestor and responder in hardware. In addition, an internal TimeStamp module (CPTS) is connected to the timestamp interface of the PCIe core so that events can be logged.

Copyright © 2024 Texas Instruments Incorporated

The Precision Time Measurement (PTM) support in the PCIe subsystem is handled by the combination of the PCIe core and the CPTS module.

The PCIe core controller handles the PCIe conversation to exchange timestamps between the RP and EP. It also includes logic to calculate the timestamp differences as specified in the PCIe standard. A 64-bit timer that is the master clock is included in the PCIe core. The PCIe controller can be programmed to initiate the PTM conversation automatically by setting the [0] PTMRQM bit in the PCIE_CORE_LM_I_PTM_LOCAL_CONTROL_REG register.

In EP mode, there are two methods to register a CPTS HW1 push timestamp that is needed to transfer the PCIe PTM timebase to the local system time base.

1. The first method involves the PTM_LOCAL_TIMER_OUT_VALID and PTM_EP_TIMER. The PTM_LOCAL_TIMER_OUT_VALID output from the PCIe controller will be asserted when the PTM conversation is complete and the EP local timer has been updated. The adjusted value of the EP 64-bit timer is available in the PTM_LOCAL_TIMER_OUT output of the controller. A second 64-bit free running timer, PTM_EP_TIMER, is used to shadow the value of the EP timebase. The PTM_EP_TIMER is loaded with the value of the PTM_LOCAL_TIMER_OUT[63:0] when PTM_LOCAL_TIMER_OUT_VALID is high. When PTM_LOCAL_TIMER_OUT_VALID is low, PTM_EP_TIMER will be free running based on the same PCIE CORE_CLK that drives the PTM_LOCAL_TIMER_OUT. The increment value of PTM_EP_TIMER can be adjusted by writing to the PCIE_USER_PTM_CFG[10-8] PTM_EP_TIMER_ADJ register field. The increment value is one by default. The value of the PTM_EP_TIMER will not be updated to the RP timebase until the first PTM dialog has occurred. Software can set the PCIE_USER_PTM_CFG[6-0] PTM_CLK_SEL register field value between 0 and 63 to choose a particular bit of the PTM_EP_TIMER as the source for the CPTS HW1 time stamp push event.
2. The second method to register the CPTS HW1 push timestamp involves only the PTM_LOCAL_TIMER_OUT_VALID output from the PCIe controller. Software can set the PCIE_USER_PTM_CFG[6-0] PTM_CLK_SEL register field value to 64 to select the PTM_LOCAL_TIMER_OUT_VALID as the source of the CPTS HW1 time stamp push event. This will create a CPTS HW1 timestamp when there is a PTM dialog between the PCIe End Point and Root Port. The actual value of the EP PTM local timer can be read from the PCIE_USER_PTM_TIMER_LOW and PCIE_USER_PTM_TIMER_HIGH registers.

In RP mode, the CPTS can take control of the PTM master time. The 64-bit CTPS timer is continuously loaded into the PCIe PTM master clock by tying high the PTM_TIMER_IN_VALID input of the PCIe controller. This allows for software to adjust the PCIe PTM master clock as necessary using the CTPS module and maintain the PTM master time by monotonically increasing it as required by the PCIe standard. The CPTS HW1_PUSH timestamp input is driven by the particular bit of the PTM_LOCAL_TIMER_IN counter, as selected by the value in the PCIE_USER_PTM_CFG[6-0] PTM_CLK_SEL register field.

The reference clock (RCLK) for the CPTS can be controlled via the CTRLMMR_PCIE1_CLKSEL[3-0] CPTS_CLKSEL register field in the device Control Module. This enables the SoC flexibility in choosing the CPTS reference clock input. In RP mode, the RCLK needs to be connected to the SERDES PIPE clock since the CPTS will be driving the PTM master clock. For more information on the RCLK source clock selection, see *PCIe Subsystem Integration*.

Figure 12-66 shows the PTM support logic in the PCIe subsystem.

Copyright © 2024 Texas Instruments Incorporated

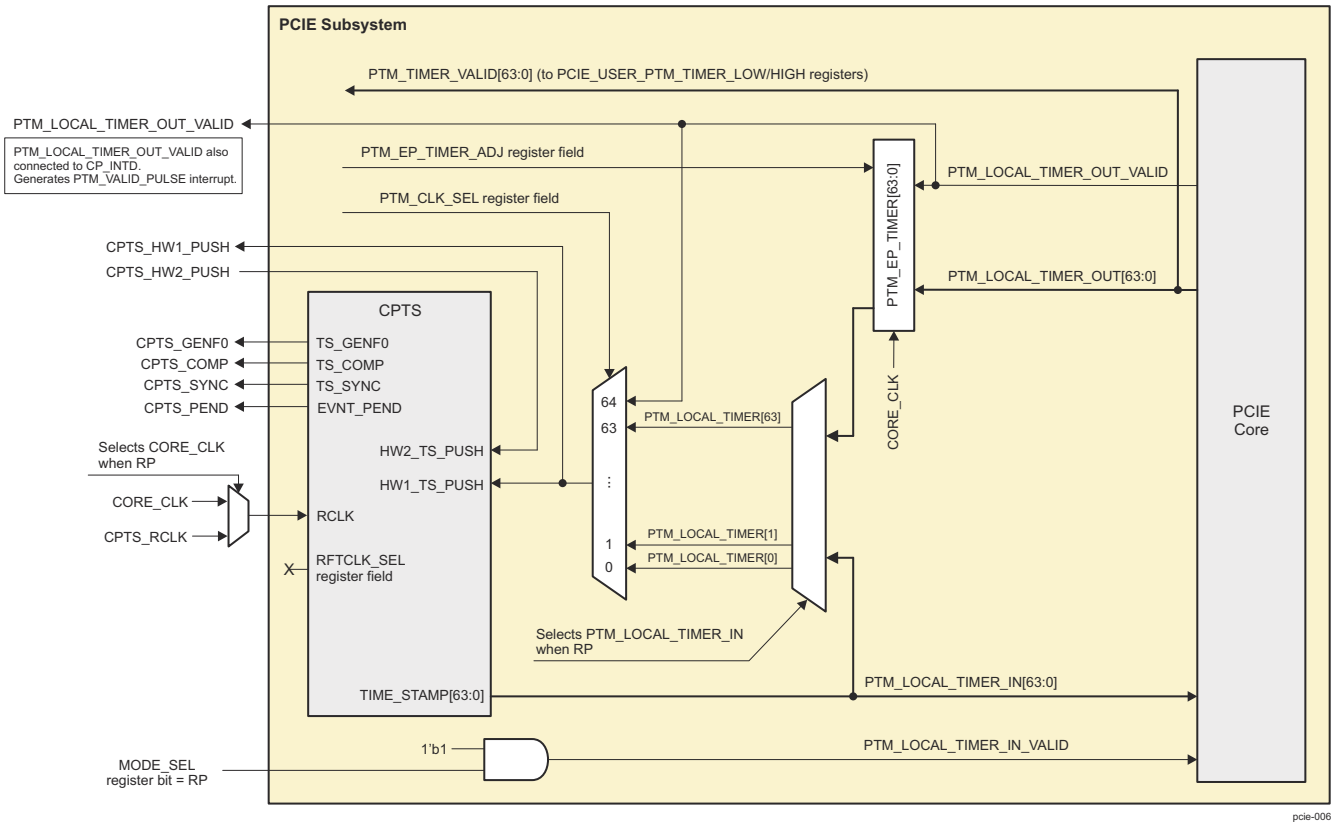


Figure 12-66. PCIe Subsystem PTM Implementation

12.2.2.3.11 PCIe Subsystem Loopback

The PCIe subsystem provides loopback support through PIPE interface (Link Layer).

12.2.2.3.11.1 PCIe PIPE Loopback

The procedure depends upon whether the device is operating in RP or EP Mode. In either case, the PCIe subsystem can be loopback master or loopback slave as outlined in PCIe specifications. A loopback master is the component requesting loopback and transmitting the data. A loopback slave is the component looping back the data.

Note

The PIPE loopback mode cannot be used for looping back transactions.

12.2.2.3.11.1.1 PIPE Loopback Master Mode

The loopback path when PCIe subsystem is loopback master is:

PCIe subsystem (loopback master) -> PIPE (TX) -> PCIe Link -> Loopback -> PCIe Link -> PIPE (RX) -> PCIe subsystem

12.2.2.3.11.1.2 PIPE Loopback Slave Mode

The loopback path when PCIe subsystem is loopback slave is:

Remote device -> PCIe Link -> PIPE (RX) -> Loopback -> PIPE (TX) -> PCIe Link -> Remote device

If the PCIe subsystem is a loopback slave, then the incoming serial data is routed back to the originating device from the PIPE interface as per PCIe loopback requirements. Typically, PCIe test equipment will be used as loopback master and it will transition PCIe subsystem into loopback slave state following which the inbound transactions will be loopback to the test equipment. There is no programming required on PCIe subsystem to enter loopback in slave mode. PHY support is not required to use this loopback mode.

Copyright © 2024 Texas Instruments Incorporated

12.2.2.3.12 PCIe Subsystem Error Handling

As an End Point, PCIe subsystem reports errors to root complex so that corrective action may be taken. These messages become error interrupts on the Root Port side. In addition, the errors that are detected by the PCIe subsystem locally are also reported to software via interrupts.

Note that for several errors of the same type that are detected in close succession, the PCIe subsystem will not necessarily send as many error messages to Root Port. It is guaranteed to send at least one such error message.

12.2.2.3.12.1 PCIe AXI to/from VBUSM Bus Error Mapping

Table 12-124 and Table 12-125 show the bus error mapping to/from the AXI interface on the PCIe controller to the system VBUSM interfaces.

The AXI2VBUSM bridge maps the bus errors on the ingress AXI interface to the VBUSM master interface as shown in the table below.

Table 12-124. AXI to VBUSM Bus Error Mapping

VBUSM		AXI	
Code	Error	Code	Error
0 (decimal)	Success	0h	OKAY
1 (decimal)	N/A	3h	N/A
2-3 (decimal) 7 (decimal)	All Others	2h	SLVERR

The VBUSM2AXI bridge maps the bus errors on the eggress VBUSM slave interface to the AXI interface as shown in the table below.

Table 12-125. VBUSM to AXI Bus Error Mapping

AXI		VBUSM	
Code	Error	Code	Error
N/A	N/A	2 (decimal)	Protection Error
0h	OKAY	0 (decimal)	Success
3h	DECERR	1 (decimal)	Addressing Error
2h	SLVERR	1 (decimal)	Addressing Error

12.2.2.3.13 PCIe Subsystem Internal Diagnostics Features

12.2.2.3.13.1 PCIe Parity

All the parity inputs on the PCIe controller core are driven by the EDC_INV module. This allows the parity input to the PCIe controller to be inverted and thus inject an error. Figure 12-67 shows the parity logic on the AXI AWADDR port as an example. The parity of the AWADDR inputs to the PCIe controller are calculated as an XOR and this is fed into the PCIe controller using the EDC_INV block. The input AWADDR_par can be inverted using commands from the ECC Aggregator to enable error injection. Similar logic is present on all other parity inputs to the PCIe controller.

Checking input parity and output parity are handled inside the PCIe controller. Any errors in parity are signaled using the PCIe active internal diagnostics interrupt (PCIE_ASF_PEND).

Copyright © 2024 Texas Instruments Incorporated

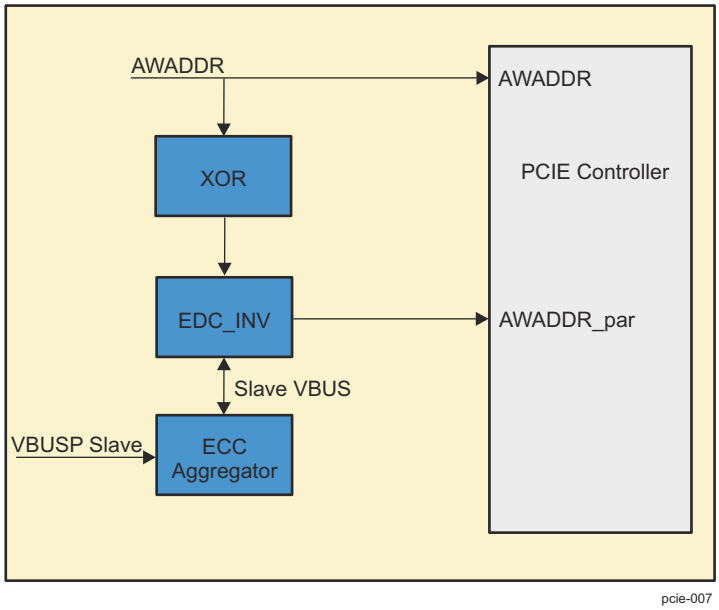


Figure 12-67. Parity Logic on AXI AWADDR Input

12.2.2.3.13.2 ECC Aggregators

The PCIe subsystem instantiates two ECC Aggregators: AXI_ECC_AGGR and CORE_ECC_AGGR.

AXI_ECC_AGGR is connected to the RAMs in PCIe CBA Clock Domain and CORE_ECC_AGGR is connected to the RAMs in the PIPE Clock Domain.

The ECC Aggregator modules integrated within the PCIe subsystem provide a mechanism to control and monitor the ECC RAMs via Single Error Correction (SEC) and Double Error Detection (DED) functions. Each ECC Aggregator module gathers level pending status from the ECC RAMs into two interrupts to system level. One interrupt is for correactable errors (SEC) and the other one for uncorrectable errors (DED). The ECC Aggregator supports software readable status of ECC single/double-bit errors and associated information such as RAM address and data bit(s) that are in error. Write back correction for async read RAMs is not supported.

For more information on the ECC Aggregator operation, see Section 12.10.4, *ECC Aggregator*.

12.2.2.3.13.3 RAM ECC Inversion

The RAM ECC logic inside the PCIe controller can be tested using the ECC inversion logic.

12.2.2.3.14 LTSSM State Encoding

Table 12-126 is used when the PCIe/M-PCIe core is configured to be operating in PCIe mode.

Table 12-126 provides the encoding of the LTSSM states on the LTSSM_STATE output of the core, as well the state read from the Physical Layer Configuration Register 0.

Table 12-126. LTSSM State Encoding

LTSSM State Name	Value (hex)
Detect.Quiet	00
Detect.Active	01
Polling.Active	02
Polling.Compliance	03
Polling.Configuration	04
Configuration.Linkwidth.Start	05
Configuration.Linkwidth.Accept	06
Configuration.Lanenum.Accept	07
Configuration.Lanenum.Wait	08

Copyright © 2024 Texas Instruments Incorporated

Table 12-126. LTSSM State Encoding (continued)

LTSSM State Name	Value (hex)
Configuration.Complete	09
Configuration.Idle	0A
Recovery.RcvrLock	0B
Recovery.Speed	0C
Recovery.RcvrCfg	0D
Recovery.Idle	0E
L0	10
Rx_L0s.Entry	11
Rx_L0s.Idle	12
Rx_L0s.FTS	13
Tx_L0s.Entry	14
Tx_L0s.Idle	15
Tx_L0s.FTS	16
L1.Entry	17
L1.Idle	18
L2.Idle	19
L2.TransmitWake	1A
Disabled	20
Loopback.Entry (Master)	21
Loopback.Active (Master)	22
Loopback.Exit (Master)	23
Loopback.Entry (Slave)	24
Loopback.Active (Slave)	25
Loopback.Exit (Slave)	26
Hot Reset	27
Recovery.Equalization, Phase 0	28
Recovery.Equalization, Phase 1	29
Recovery.Equalization, Phase 2	2A
Recovery.Equalization, Phase 3	2B

12.2.3 Universal Serial Bus (USB) Subsystem

This section describes the USB 3.0 Dual-Role-Device (DRD) interface subsystem of the device.

Note

This chapter serves to describe the integration of the third-party USB controller and should not be considered sufficient for those wishing to modify the existing Linux or RTOS USB driver(s) or create a new driver to support this controller implementation. For those who do wish to substantially modify the existing Linux or RTOS USB driver(s), or create new drivers, contact TI for more information on how to obtain the third-party documentation under NDA.

12.2.3.1 USB Overview

Similar to earlier versions of USB bus, USB 3.0 is a general-purpose cable bus, supporting data exchange between a host device and a wide range of simultaneously accessible peripherals.

The device supports these USB subsystems:

- USB3SS0 is SuperSpeed (SS) USB 3.0 Dual-Role-Device (DRD) subsystem with on-chip SS (USB3.0) PHY and HS/FS/LS [1](USB2.0) PHY

Copyright © 2024 Texas Instruments Incorporated

Figure 12-68 shows the USB subsystem highlights.

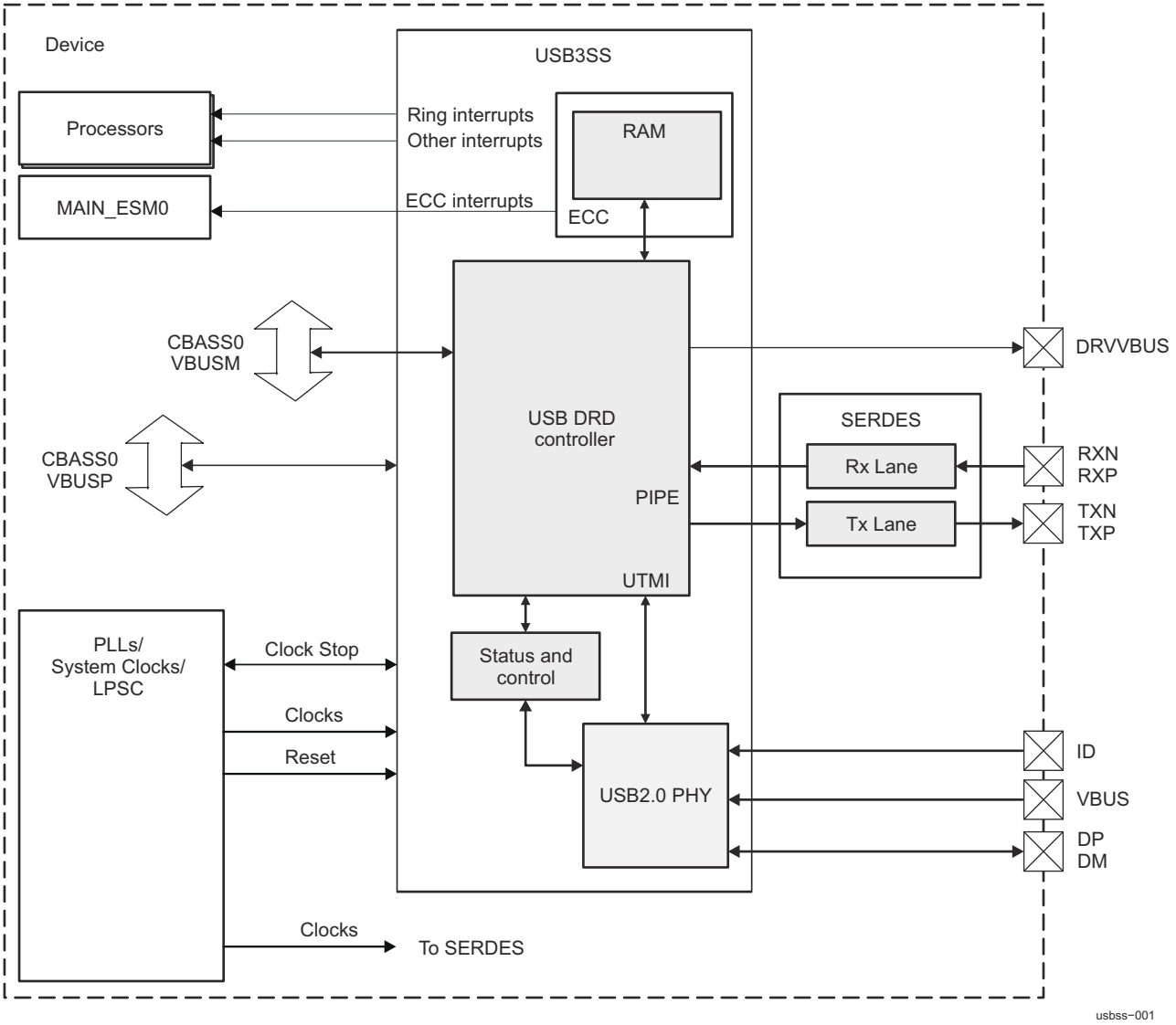


Figure 12-68. USB Subsystem Overview

12.2.3.1.1 USB Features

The USB Dual-Role-Device (DRD) subsystem, supports the following USB features:

- General features:
  - Supports Peripheral (aka Device) mode at Superspeed (5 Gbps), Highspeed (480 Mbps), and Fullspeed (12 Mbps)
  - Supports Host mode at Superspeed (5 Gbps), Highspeed (480 Mbps), Fullspeed (12 Mbps), and Lowspeed (1.5 Mbps)
  - Static peripheral operation
  - Static host operation
  - Compliant with USB 3.1 Gen1 Specification
  - Limited OTG 2.0 functionality
  - Supports VBUS and ID detection
  - Host Negotiation Protocol (HNP) support

[1] LowSpeed (LS) is supported only in host mode.

Copyright © 2024 Texas Instruments Incorporated

  - Charger Downstream Port (CDP) as per Battery Charging Specification, Revision 1.2
- Each controller instance contains single xHCI with the following features:
  - Compatible to the xHCI specification (revision 1.0)
  - Supports 15 Transmit (TX), 15 Receive (RX) endpoints (EPs), and one EP0 endpoint which is bidirectional
  - Internal scatter-gather DMA controller
  - Dynamic data buffering
  - USB3 power saving states (U1, U2, and U3) and USB2 L1/L2
  - 64 slots supported with 32 endpoints per slot, for host operation
- Operation flexibility:
  - Uniform programming model for SS, HS, FS, and LS operation
  - Multiple interrupt lines:
    - 8 interrupts associated with 8 programmable Event Rings for multi-core support
    - Interrupt for OTG events
- Functional safety:
  - Internal RAM with ECC
  - Hardware transaction timeout monitor
- Supports VBUS and ID detection in the USB2.0 PHY
- External requirements and I/O features:
  - Requires an external charge pump or power switch for VBUS 5-V generation
  - Requires an external circuitry or PMIC to start the battery charging upon Battery Charger (BC) detected event
  - Requires external high-precision resistors for USB2.0 PHY and SERDES termination calibration
  - Supports internal data lane swapping for Type C connector. All other Type C features, like cable detection and CC configuration, have to be implemented using external circuitry
  - Supports short circuit protection to GND and short-term short circuit protection to VBUS on data pins

Unsupported Features:

- See the Module Integration section of this document for a list of module features not supported by the integration on this Device.

12.2.3.1.2 USB Ports

This section describes USB module ports related to clocks, resets, and hardware requests.

Table 12-127. USB Clocks and Resets

Clocks	
Module Clock Input	**Description**
ACLK	VBUSM interface clock
PCLK	VBUSP interface clock
BUFCLK	VBUSM bridge clock
CLK_LPM	Low power clock (24 MHz). This clock has to be free running when USB is enabled.
USB2_REFCLOCK	USB2PHY reference clock. HFOSC selected via CTRLMMR_USB0_CLKSEL. Frequency value must be indicated to PHY via USB3P0SS_STATIC_CONFIG.
USB2_APB_PCLK	USB2PHY VBUSP interface clock
Resets	
Module Reset Input	**Description**
USB_RST	USB3SS0 hardware reset

Table 12-128. USB Hardware Requests

Interrupt Requests		
Module Interrupt	**Description**	**Type**
IRQ_[0:7]	8 event ring interrupts in host mode. In device mode, IRQ[6] is device USB interrupt and IRQ[7] is device wakeup request.	Level
HOST_SYSTEM_ERROR	USB host system error	Level
OTGIRQ	USB OTG events	Level

Copyright © 2024 Texas Instruments Incorporated

Table 12-128. USB Hardware Requests (continued)

ASF_INT_NONFATAL	USB safety features nonfatal interrupt	Level
ASF_INT_FATAL	USB safety features fatal interrupt	Level
A_ECC_AGGR_CORRECTED_ERR_LEVEL	ECC aggregator interrupt	Level
A_ECC_AGGR_UNCORRECTED_ERR_LEVEL	ECC aggregator interrupt	Level
DMA Events		
Module DMA Event	**Description**	**Type**
-	No PDMA channels to external DMA engines. USB has an internal DMA controller.	-

12.2.3.1.3 USB Terminology

The following acronyms and abbreviations are related to USB.

Term	Definition
ADP	Attach Detection Protocol: detects USB OTG attach/detach events.
DRD	Dual Role Device: USB Host and Peripheral capable.
DS	Down Stream (A USB port facing from a Host or Hub to a Device/Peripheral).
EP	Endpoint: USB communication channel between the USB link partners carrying a single transfer type (BULK, ISOCH, INT, or CONTROL) and bus sharing arbitration scheme.
FS	Full-Speed USB data rate (12 Mbps).
HNP	Host Negotiation Protocol, OTG extension to swap USB host and peripheral roles.
HS	High-Speed USB data rate (480 Mbps).
ITP	Isochronous Timestamp Packet: USB SS micro-frame boundary packets.
LMP	Link Management Packet.
LS	Low-Speed USB data rate (1.5 Mbps).
OTG	On-The-Go extension to USB protocol.
PHY	Physical Layer Device.
SS	Super-Speed USB data rate. 5 Gbps (USB3.0 or USB3.1 Gen1)
US	Upstream facing from a device (or hub) to the host (or a hub).
USB IF	USB Implementers Forum. The governing organization that develops and maintains the USB specifications and compliance standard. http://www.usb.org
xHC	Host Controller, designates the USB hardware that implements the xHCI specification.
xHCI	eXtensible Host Controller Interface. The specification that defines the register level interface for host controller.

12.2.3.2 USB Environment

This chapter details the USB subsystem environment.

Table 12-129 describes the external signals of the USB3SS0 subsystem.

Table 12-129. USB3SS Input/Output Description

Device Pin	I/O[(1)]	Description
DP	I/O	USB2.0 data pins. These are HS/FS/LS bidirectional differential data lane (D+/D-)
DM	I/O	
SSRX1P (SSRX2P)	I	USB3.0 differential data receive lane (RX+ pin)
SSRX1N (SSRX2N)	I	USB3.0 differential data receive lane (RX- pin)
SSTX1P (SSTX2P)	O	USB3.0 differential data transmit lane (TX+ pin)
SSTX1N (SSTX2N)	O	USB3.0 differential data transmit lane (TX- pin)
REXT	A/I	USB3.0 SerDes external calibration resistor. Requires a 3.01 kOhm ±1% accurate off-chip resistor connected from this pin to ground.
ID	A/I	USB cable identifier (A/B-device). Analog ID pin sense with internal pull-up
VBUS	A/I	An 3.3-V analog input for monitoring the voltage on VBUS (VBUS sense). 5-V VBUS must be applied via an external resistive divider /3. For example, R1 = 20 kOhm and R2 = 10 kOhm

Copyright © 2024 Texas Instruments Incorporated

Table 12-129. USB3SS Input/Output Description (continued)

Device Pin	I/O[(1)]	Description
DRVVBUS	O	A digital output signal for VBUS Power Supply Enabling. Used to enable an external charge pump or power switch to supply +5V power to the VBUS port, when appropriate
RCALIB	A/I	External resistor for USB2.0 PHY calibration. Requires a 500 Ohm ±1% off-chip resistor connected from this pin to ground

(1) I = Input; O = Output; A = Analog

Copyright © 2024 Texas Instruments Incorporated

12.2.3.3 USB Functional Description

Note

This chapter serves to describe the integration of the third-party USB controller and should not be considered sufficient for those wishing to modify the existing Linux or RTOS USB driver(s) or create a new driver to support this controller implementation. For those who do wish to substantially modify the existing Linux or RTOS USB driver(s), or create new drivers, contact TI for more information on how to obtain the third-party documentation under NDA.

The USB3SS subsystem contains the USB3.0 Dual Role Device (DRD) controller module and a USB2.0 PHY module. A wrapper module is controlling some top-level functions like Host and Device role switch and reset release.

The USB controller uses USB2PHY for USB2.0 operation and one of the device SERDESes for USB3.0 speeds. The SERDES also contains lane swap feature for USB Type-C plug flipping support.

12.2.3.3.1 USB Type-C Connector Support

Lane swapping for Type-C connector support is performed within the SERDES wrapper module. Type-C CC detection and configuration has to be performed by external ICs. Type-C device attachment/detachment and cable orientation have to be communicated using I2C, GPIO, or a similar method.

For USB Type-C support, the SERDES wrapper and SERDES must be programmed for USB protocol on lane 0, lane 1, and/or lane2, and lane 3. Software must hold PHY in reset and then write to the LN10_SWAP and/or LN23_SWAP bit of the to mux the PIPE interface to the other lane. Also the SuperSpeed port in the controller has to be in disabled state when programming LN10_SWAP or LN23_SWAP.

12.2.3.3.2 USB Controller Reset

USB controller has three resets going in, called preset_n, pwrup_rst_n, and aresetn.

- preset_n comes from the mod_g_rst_n input coming from LPSC
- pwrup_rst_n is PWRUP_RST_N register bit in the USB wrapper module (USB3P0SS_W1). This reset has to be deasserted before accessing controller registers. The default value for pwrup_rst_n is 0 (reset asserted). This reset goes to the default asserted state when mod_g_rst_n reset is asserted.
- areset_n comes from PWRUP_RST_N register bit, but the deassertion is synchronized to ACLK.

12.2.3.3.3 Overcurrent Detection

The overcurrent detection circuit is external to the device. Software must specify an overcurrent condition to the controller whenever the overcurrent signal is received from the detection circuit. Software must set the OVERCURRENT_N bit to 0 in the USB3P0SS_W1 register when overcurrent was detected.

12.2.3.3.4 Top-Level Initialization Sequence

The following initialization sequence has to be followed before configuring the controller for USB operation:

1. Software must select and configure the SERDES for USB operation. Some of the registers are shown in . Refer to *Serializer/Deserializer (SerDes)* for more details. USB2.0 PHY contains default configuration that normally does not need change.
2. Software must configure the pseudo-static settings in USB3P0SS_W1 register
3. Software must set PWRUP_RST_N to 1 in order to deassert controller reset

After the above sequence, software can access controller registers.

Copyright © 2024 Texas Instruments Incorporated

12.2.4 Serializer/Deserializer (SerDes)

This section describes the Serializer/Deserializer (SerDes) in the device.

12.2.4.1 SerDes Overview

SerDes' goal is to convert device (SoC) parallel data into serialized data that can be output over a high-speed electrical interface. In the opposite direction, SerDes converts high-speed serial data into parallel data that can be processed by the device. To this end, the SerDes contains a variety of functional blocks to handle both the external analog interface as well as the internal digital logic.

Most important building blocks of SerDes are:

- Physical Media Attachment (PMA):
  - Lanes: The lanes handle all inputs and outputs from the serial interface, and contain the Tx/Rx I/Os, serializer/deserializer, and Clock and Data Recovery (CDR) unit.
  - Common module (CMN): The CMN handles peripheral and Tx clocking of the SerDes. It consists of internal PLLs and external reference clock input buffer, reset, and startup circuitry.
- Physical Coding Sub-block (PCS): The PCS is responsible for translating data from/to the parallel interface, as well as data encoding/decoding and symbol alignment.
- WIZ: The WIZ acts as a wrapper for the SerDes, and can both send control signals to and report status signals from the SerDes, and muxes SerDes to peripherals.

12.2.4.1.1 SerDes Features

The SERDES module features include:

- Quad lane PHY containing:
  - Transmit and Receive I/Os
  - Serializer
  - Deserializer
  - Clock and data recovery (CDR) unit
- Common Module (CMN)
  - PLLs
  - Master bias
  - Automatic calibration of pin termination resistors
  - Reference clock input buffers
  - Reset and startup management
- Physical Coding Sub-block (PCS)
  - USB3.1 Gen 1 (5 Gbps)
  - PCIe Gen 1 (2.5 Gbps), Gen 2 (5 Gbps), Gen 3 (8 Gbps)
  - Hyperlink (10 Gbps)
  - Display Port / Embedded Display port v1.4
  - Symbol alignment
  - Selectable serial pin polarity reversal for both transmit and receive paths
  - Bit stream reordering
- Physical Media Attachment (PMA) layer
  - Transmit equalization
  - Receive equalization
  - Data path BIST with programmable pattern generation and error detection
  - Serial bit stream and parallel word loopback for both line and parallel side
  - 8-bit ADC provides digitized ATB measurement results
  - Supports DC and AC JTAG (boundary scan) per IEEE 1149.6

The SERDES mux (WIZ) module supports the following features:

- Multiplexes device interfaces onto a single SERDES lane (Tx and Rx)
- Provides registers to implement SERDES control and status functions and alignment delays
- Clock generator block for providing MAC transmit clock
- Rx comma align block
  - Performs de-stuffing the Rx data stream in the event that the Rx rate is different from the Tx rate

Copyright © 2024 Texas Instruments Incorporated

– Supports comma detection that is not sensitive to false commas using all 8B/10B character combinations

Copyright © 2024 Texas Instruments Incorporated

12.2.4.1.2 SerDes Ports

This section describes SerDes module ports related to clocks, resets, and hardware requests.

Table 12-130. SerDes Clocks and Resets

Clocks	
Module Clock Input	**Description**
SERDES_ICLK	VBUS interface clock
CMN_REFCLK_INT	Internal reference clock from device sources. Software selectable.
Resets	
Module Reset Input	**Description**
SERDES_RST	Serdes LPSC reset

Table 12-131. SerDes Hardware Requests

Interrupt Requests		
Module Interrupt Signal	**Description**	**Type**
PHY_PWR_TIMEOUT_LVL_0	Lane power timeout interrupt	Level
DMA Events		
Module DMA Event	**Description**	**Type**
-	No PDMA channels to external DMA engines	-

12.2.4.1.3 Industry Standards Compatibility

All SerDes interfaces are configured as point-to-point connections. It is assumed that the connection is made between the SoC and another device that is compliant to the appropriate industry standard. The list of supported standards is given below.

This chapter deals with the physical layer and, therefore, it is the electrical specifications in these standards that are relevant. For more information regarding protocol compliance [2], see the device-specific Datasheet.

- PCIe: This is electrically compliant with Base specification revision 4.0.
- USB: This is electrically compliant with USB 3.1.
- Hyperlink: Electrically compliant to OIF-CEI-11G-SR.
- Display Port/ Embedded Display Port VESA Standard v1.4

Table 12-132. SerDes Supported Standards

Standard	Bit Rate (Gbps)	Reference Clock Frequency (MHz)	Bus Width (bits)
PCI Express 4.0 Gen1	2.5	19.2, 24, 25, 26, 100	32
PCI Express 4.0 Gen2	5.0	19.2, 24, 25, 26, 100	32
PCI Express 4.0 Gen3	8.0	19.2, 24, 25, 26, 100	32
Hyperlink	10.0	19.2, 24, 25, 26, 100, 125	20
DP/eDP	1.62-8.1	19.2, 24, 25, 26	10
USB 3.1 Gen1 SuperSpeed	5.0	19.2, 24, 25, 26, 100	32

[2] Electrical compatibility does not guarantee interoperability with devices

Copyright © 2024 Texas Instruments Incorporated

12.2.4.2 SerDes Environment

12.2.4.2.1 SerDes I/Os

Figure 12-69 shows the I/O interface signals of SERDES.

Copyright © 2024 Texas Instruments Incorporated

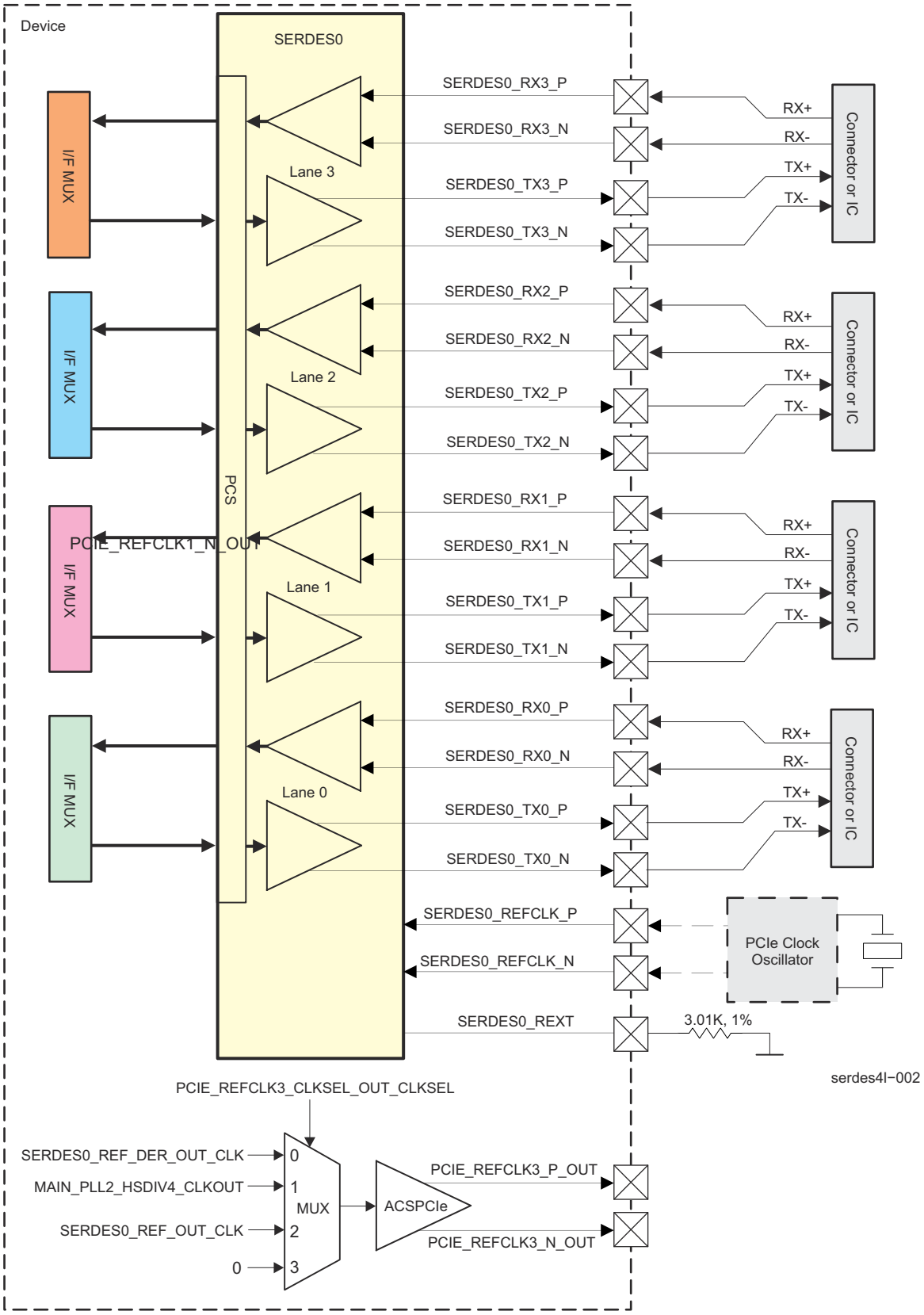


Figure 12-69. SerDes Environment

Copyright © 2024 Texas Instruments Incorporated

Note

Although containing some of the basic external components, Figure 12-69 must not be considered as an exhaustive guide for the PCB designer. TI provides additional documents for those who are willing to design PCBs and/or fine tune the SerDes.

Table 12-133 describes the external signals of SERDES0 module.

Table 12-133. SerDes Input/Output Description

Device Pin	I/O[(1)]	Description
SERDES0_RX0_P	I	SerDes differential data receive pins. Lane 0
SERDES0_RX0_N	I	
SERDES0_TX0_P	O	SerDes differential data transmit pins. Lane 0
SERDES0_TX0_N	O	
SERDES0_RX1_P	I	SerDes differential data receive pins. Lane 1
SERDES0_RX1_N	I	
SERDES0_TX1_P	O	SerDes differential data transmit pins. Lane 1
SERDES0_TX1_N	O	
SERDES0_RX2_P	I	SerDes differential data receive pins. Lane 2
SERDES0_RX2_N	I	
SERDES0_TX2_P	O	SerDes differential data transmit pins. Lane 2
SERDES0_TX2_N	O	
SERDES0_RX3_P	I	SerDes differential data receive pins. Lane 3
SERDES0_RX3_N	I	
SERDES0_TX3_P	O	SerDes differential data transmit pins. Lane 3
SERDES0_TX3_N	O	
PCIE_REFCLK1_P_OUT	O	Serdes Internal reference clock Output
PCIE_REFCLK1_N_OUT	O	
SERDES0_REFCLK_P	I	SerDes external system reference clock
SERDES0_REFCLK_N	I	
SERDES0_REXT	A/I	PMA external calibration resistor. Requires a 3.01 kOhm ±1% accurate off-chip resistor connected from this pin to ground.

(1) I = Input; O = Output; A = Analog

Copyright © 2024 Texas Instruments Incorporated

12.2.4.3 SerDes Integration

This section describes SerDes module integration in the device, including information about clocks, resets, and hardware requests.

Table 12-134 summarize the integration of SerDes in device MAIN domain.

Table 12-134. SerDes Hardware Requests

Interrupt Requests			
Module Instance	**Module Interrupt Signal**	**Description**	**Type**
SERDES0	PHY_PWR_TIMEOUT_LVL_0	Lane power timeout interrupt	Level
DMA Events			
Module Instance	**Module DMA Event**	**Description**	**Type**
SERDES0	-	No PDMA channels to external DMA engines	-

Note

For more information on the interconnects, see *System Interconnect*.

For more information on the power, reset and clock management, see the corresponding sections within *Device Configuration*.

Copyright © 2024 Texas Instruments Incorporated

12.2.4.3.1 WIZ Settings

This section describes the quasi-static settings that software must perform after global resets. These settings are made via registers in the CTRL_MMR0 module. For CTRL_MMR0 description, please refer to *Control Module (CTRL_MMR)*.

12.2.4.3.1.1 Interface Selection

SerDeses provide PHY functions for the following high-speed interfaces:

- DP/eDP
- PCIe1 (one- to four-lane)
- Hyperlink
- USB3.0 on lane 1 or 3. Lane 0 or 2 used for Type-C plug reversal feature.

Table 12-135 describes the interface combinations supported by SERDES0.

Table 12-135. SERDES0 Supported Configurations

Interface Alias	CTRLMMR_SERDES0_LN0_CTRL [1:0]		CTRLMMR_SERDES0_LN1_CTRL [1:0]		CTRLMMR_SERDES0_LN2_CTRL [1:0]		CTRLMMR_SERDES0_LN3_CTRL [1:0]	
	LANE_FUNC_SEL	Interface on Lane 0	LANE_FUNC_SEL	Interface on Lane 1	LANE_FUNC_SEL	Interface on Lane 2	LANE_FUNC_SEL	Interface on Lane 3
IP1	0x0	eDP0 Lane0	0x0	eDP0 Lane 1	0x0	eDP0 Lane 2/0	0x0	eDP0 Lane3/1
IP2	0x1	PCIe Lane0	0x1	PCIe Lane 1	0x1	PCIe Lane 2	0x1	PCIe Lane 3
IP3	0x2	-(1)	0x2	USB3_0	0x2	-(1)	0x2	USB3_0
IP4	0x3	Hyperlink Lane0	0x3	Hyperlink Lane 1	0x3	Hyperlink Lane 2	0x3	Hyperlink Lane 3

(1) Reserved for USB0 for type-C connector lane swap. That is, select this function if Type C was implemented in the design. See Section 12.2.3, *USB*.

As seen in Table 12-135, USB0 can be routed to two different lanes. To avoid routing to two lanes at the same time, an additional muxing exists.

Table 12-136 describes the additional muxing for USB0 to lane 1 or to lane 3.

Note

Settings in Table 12-136 must be aligned with the settings made in Table 12-135.

Table 12-136. USB0 Muxing to Serdes Lanes

CTRLMMR_USB0_CTRL	
[27] SERDES_SEL	Lane Selected
0	Lane 1 (Lane 0 for Type-C)
1	Lane 3 (Lane 2 for Type-C)

12.2.4.3.1.2 ACSPCIe Reference Clock Selection

Table 12-137 describes the SERDES ACSPCIe reference clock selection. ACSPCIe buffer clk selection: Needs PCIE_REFCLK1_CLKSEL_OUT_CLK_EN to be enabled for driving the clk out onto the ref clk pins.

Table 12-137. SerDes ACSPCIe Reference Clock Selection

[1:0] CLK_SEL	ACSPCIe Clk Source
0x0	SERDES_REF_DER_OUT_CLK
0x1	MAIN_PLL2_HSDIV4_CLKOUT
0x2	SERDES0_REF_OUT_CLK
0x3	0

Copyright © 2024 Texas Instruments Incorporated

12.2.4.4 SerDes Functional Description

12.2.4.4.1 SerDes Block Diagram

Figure 12-70 is a top-level, non-exhaustive diagram of SerDes and WIZ wrapper. Note that only one (Tx + Rx) lane is shown.

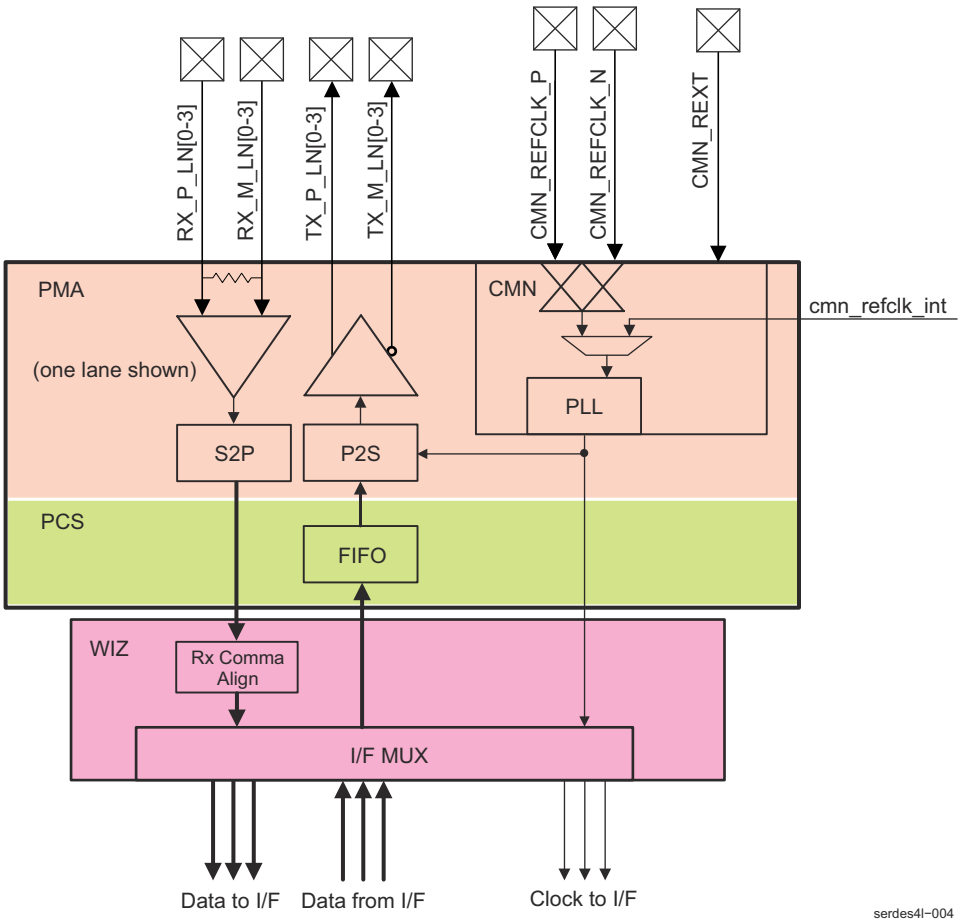


Figure 12-70. SerDes and WIZ Block Diagram

Building blocks of SerDes include:

- Lanes: The lanes handle all inputs and outputs from the serial interface, and contain the Tx/Rx I/Os, serializer/deserializer (P2S/S2P), and Clock and Data Recovery (CDR) unit. Each SerDes contains one Tx and Rx lane.
- Common module (CMN): The CMN handles peripheral and Tx clocking of the SerDes. It consists of internal PLLs and external reference clock input buffer, reset, and startup circuitry.
- Lanes and CMN are parts of the Physical Media Attachment (PMA) layer.
- Physical Coding Sub-block (PCS): The PCS is responsible for translating data from/to the parallel interface, as well as data encoding/decoding and symbol alignment.
- WIZ: The WIZ acts as a wrapper for the SerDes, and can both send control signals to and report status signals from the SerDes (register interface), and muxes SerDes to peripherals (USB, PCIe, and others).

Copyright © 2024 Texas Instruments Incorporated

12.3 Memory Interfaces

This section describes the memory interfaces in the device.

Copyright © 2024 Texas Instruments Incorporated

12.3.1 Flash Subsystem (FSS)

This section describes the Flash Subsystem (FSS) in the device.

12.3.1.1 FSS Overview

The Flash Subsystem (FSS) provides access to external flash devices via Octal SPI (OSPI) andHyperBus™ interface along with encryption/decryption, authentication, and in-line ECC protection.

The FSS includes two OSPI and one HyperBus interface. For more information, see *Octal Serial Peripheral Interface (OSPI)* and *HyperBus Interface*.

Table 12-138. FSS Allocation Across Device Domains

Instance	Domain		
	WKUP	MCU	MAIN
MCU_FSS0	-	✓	-

The first FSS path (FSS0) has access to either OSPI0 or HyperBus interface and can be configured for ECC, authentication, or both. The second FSS path (FSS1) has access only to OSPI1 and can be configured with or without ECC.

Figure 12-71. FSS Overview

12.3.1.1.1 FSS Features

The FSS has the following features:

- Two simultaneous flash interfaces:
  - Two OSPIs (OSPI0 and OSPI1) OR
  - One HyperBus interface and one OSPI (OSPI1)
- Primary OSPI0/HyperBus interface supports:
  - Execute in place (XIP) operation
  - 32-byte block copy (BC) operation
  - Prefetch for two XIP flows to improve performance from external flash device
- Secondary OSPI1 interface supports:
  - 32-byte block copy (BC) operation
  - ECC
- OSPIs support single, dual, quad, or octal SPI devices
- OSPIs support up to 4 devices
- HyperBus interface supports up to 2 devices
- The OSPIs and HyperBus interface have independent power management for low power operations

Unsupported Features:

- See the Module Integration section of this document for a list of module features not supported by the integration on this Device.

12.3.1.1.2 Flash Ports

Table 12-139. MCU_FSS0 Clocks and Resets

Clocks	
Module Clock Input	**Description**
MCU_FSS_ICLK	MCU_FSS Interface Clock
Resets	
Module Reset Input	**Description**
MCU_FSS_RST	MCU_FSS System Reset

Table 12-140. MCU_FSS0 Hardware Requests

Interrupt Requests		
Module Interrupt Signal	**Description**	**Type**

Copyright © 2024 Texas Instruments Incorporated

Table 12-140. MCU_FSS0 Hardware Requests (continued)

MCU_FSS_ECC_ERR_INT	MCU_FSS ECC Error Interrupt	Level

12.3.1.2 FSS Environment

Copyright © 2024 Texas Instruments Incorporated

Table 12-141 describes the MCU_FSS0 I/O signals.

Table 12-141. MCU_FSS0 I/O Signals

FSS Interface	I/O Signals
OSPIs (OSPI0 and OSPI1)	For more information about OSPIs I/O signals, see *OSPI Environment*.
HyperBus interface	For more information about HyperBus interface I/O signals, see *HyperBus Environment*.

Copyright © 2024 Texas Instruments Incorporated

12.3.1.3 FSS Functional Description

12.3.1.3.1 FSS Block Diagram

The FSS provides access to external Flash and RAM devices. It supports XIP (Execute-in-Place) and BC (Block Copy) operations.

The FSS consists of two OSPIs and one HyperBus interface. There are two FSS paths - FSS0 and FSS1 (see). The first path (FSS0) includes OSPI0 and HyperBus interface. The second path (FSS1) includes OSPI1. These two FSS paths can be used simultaneously (see *FSS Typical Application*).

Table 12-142 shows the FSS allowed interface combinations.

Table 12-142. FSS Allowed Interface Combinations

Combination	Primary Interface (FSS0)	Secondary Interface (FSS1)
1.	OSPI0 (MCU_FSS0_OSPI0)	Not Used
2.	HyperBus Interface (MCU_FSS0_HPB0)	Not Used
3.	OSPI0 (MCU_FSS0_OSPI0)	OSPI1 (MCU_FSS0_OSPI1)
4.	HyperBus Interface (MCU_FSS0_HPB0)	OSPI1 (MCU_FSS0_OSPI1)
5.	Not Used	OSPI1 (MCU_FSS0_OSPI1)

Figure 12-72 shows the FSS block diagram.

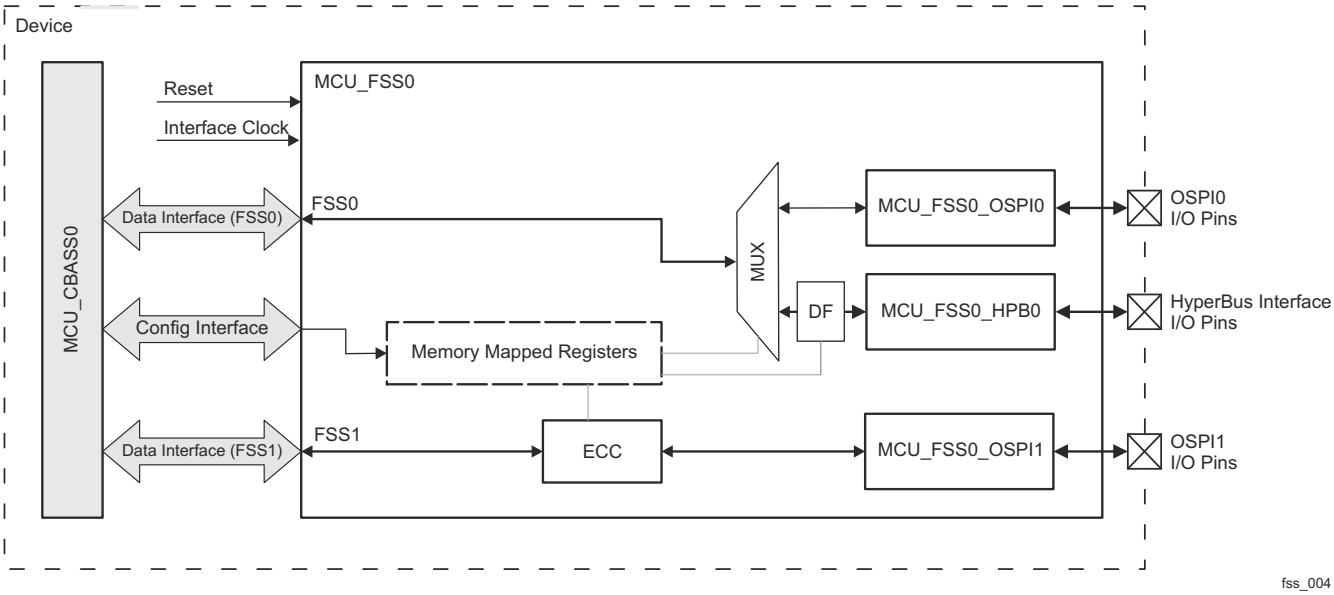


Figure 12-72. FSS Block Diagram

FSS Blocks:

- MCU_CBASS0: The MCU_CBASS0 interconnect allows FSS to communicate with the device modules and subsystems.
- Data Interface (FSS0): It is 64-bit data/32-bit address multi issue data interface with coherent in-band bypass. It provides accessibility to either the OSPI0 or HyperBus interface.
- Data Interface (FSS1): It is 64-bit data/32-bit address multi issue data interface with coherent in-band bypass. It provides accessibility to the OSPI1.
- Config Interface: It is used for configuration of the memory mapped registers within the FSS.
- Interface Clock and Reset:
  - For more information, see *MCU_FSS0 Clocks and Resets*.
  - For more information, see *MCU_FSS0_OSPI Clocks and Resets*.
  - For more information, see *MCU_FSS0_HPB0 Clocks and Resets*.

Copyright © 2024 Texas Instruments Incorporated

- Memory Mapped Registers: This block conditionally includes registers from the following blocks: FSS, ECC, MUX, and DF. The configuration of these registers defines which combination of FSS interfaces is selected (see Table 12-142) and also which FSS features and operation modes are used. For more information, see MCU_FSS0_SYSCONFIG.
- MUX: The Muxing (MUX) block is used as a software controlled switch in the primary FSS (FSS0) interface. It defines which interface to be used (OSPI0 or HyperBus interface). The MUX block can switch only when the traffic is idle. The software has responsibility to cause the traffic to be stopped.
- DF: The Dynamic Fragmenter (DF) module is responsible for fragmenting write data to the flash region so that all writes to the flash region are done in 16-bits chunks (a requirement for HyperFlash). It passes all other transaction through unaffected.
- FSS Interfaces:
  - MCU_FSS0_OSPI0: The OSPI0 is part of the primary FSS interface (FSS0). It can be configured to use ECC and/or OTFE module.
  - MCU_FSS0_OSPI1: The OSPI1 is part of the secondary FSS interface (FSS1). It can be configured with or without ECC.
  - MCU_FSS0_HPB0: The HyperBus interface is part of the primary FSS interface (FSS0). It can be configured to use ECC and/or OTFE module.
- FSS I/O Pins:
  - OSPI1 I/O Pins: All used MCU_FSS0_OSPI1 interface pins (for more information, see *OSPI I/O Signals*).
  - HyperBus I/O Pins: All used MCU_FSS0_HPB0 interface pins (for more information, see *HyperBus I/O Signals*).

12.3.1.3.2 FSS ECC Support

The Error Correcting Code (ECC) is a mechanism for providing increased system reliability via reduction of memory software errors by allowing single bit errors to be detected and corrected (SEC) and double bit errors to be detected (DED).

For more information about ECC, refer to *ECC Aggregator*.

For FSS1 path when the stand-alone ECC module is used:

- The input for the ECC module is 32-byte data block.
- The ECC module generates ECC codes for the 32-byte data block.
- The ECC module packs data + ECC, reformats the address and sends this to the flash controller (OSPI1 flash controller).

For more information, see *ECC Calculation*.

12.3.1.3.3 FSS Modes of Operation

Table 12-142 presents the possible combinations of FSS interfaces.

Both paths (FSS0 and FSS1) support XIP and BC modes.

12.3.1.3.4 FSS Memory Regions

Table 12-143 shows the FSS memory regions.

Table 12-143. FSS Memory Regions

Address Range		Size	Description
FSS0 (HyperBus interface or OSPI0)	FSS1 (OSPI1)		
0x04 0000 0000 to 0x04 FFFF FFFF	0x06 0000 0000 to 0x06 FFFF FFFF	4 GB	External Memory Space (Region 0)
0x00 5000 0000 to 0x00 57FF FFFF	0x00 5800 0000 to 0x00 5FFF FFFF	128 MB	Boot Space (Region 1)
0x05 0000 0000 to 0x05 FFFF FFFF	0x07 0000 0000 to 0x07 FFFF FFFF	4 GB	External Memory Space (Region 3)

12.3.1.4 FSS Programming Guide

12.3.1.4.1 FSS Initialization Sequence

Initialization steps:

- Configure the main boot parameters for FSS0 or FSS1):

Copyright © 2024 Texas Instruments Incorporated

  - Select the boot block to be used.
  - Select the size of the boot block to be used.
- Enable FSS in PSC.
- Configure the FSS interface which will be used:
  - For FSS0 - OSPI0 or HyperBus interface.
  - For FSS1 - OSPI1.
- Enable the selected FSS interface in PSC.
- Configure the FSS interface:
  - For more information about OSPI configuration, please see *Octal Serial Peripheral Interface (OSPI)*.
  - For more information about HyperBus interface configuration, please see *HyperBus Interface*.
- Configure the ECC region start address and size:
  - MCU_FSS0_ECC_RGSTRT_j
  - MCU_FSS0_ECC_RGSIZ_j
- Enable the ECC option if the target flash device has ECC encoded within.
- If the target device is HyperFlash/SRAM configure fragmentation address boundary:
  - Flash device requires fragmentation.
  - SRAM device does not require fragmentation.
- Enable error interrupts (see MCU_FSS0_ENABLE_SET).

12.3.1.4.2 FSS Real-Time Operation

In the event of an ECC single error detect, the 32-byte block address and associated error bits are stored and an interrupt is generated (if enabled). The CPU can then service the interrupt and determine the error type. If a single error occurs, the CPU can scrub the flash block to determine if the error is permanent and requires reprogramming.

In the event of an ECC double error detect, the 32-byte block address and associated error bits are stored and an interrupt is generated (if enabled). The CPU can then service the interrupt and determine the error type. If the double error detect is within the flash, the region is corrupt and have to be treated as unused.

12.3.1.4.3 FSS Power Up/Down Sequence

There are four PSC controls for the FSS: the FSS itself, OSPI0, HyperBus interface, and OSPI1. The CPU enables the appropriate interfaces before using FSS0 or FSS1 of the FSS.

The FSS0 can be used for HyperBus interface or OSPI0 and only the interface in use has to be enabled. Software should ensure the selected interface is enabled prior to FSS0 transactions.

Copyright © 2024 Texas Instruments Incorporated

The FSS1 can only be used to access OSPI1. The FSS1 transaction may be blocked depending on the power state of the OSPI1.

Normal Power Down Sequence:

- Block any new transaction to the particular interface - FSS0 or FSS1.
- Power down the target FSS interface.
- In the event when all targets are powered down, the FSS can then be powered down.

Copyright © 2024 Texas Instruments Incorporated

12.3.2 Octal Serial Peripheral Interface (OSPI)

This section describes the Octal Serial Peripheral Interface (OSPI) module for the device.

12.3.2.1 OSPI Overview

The Octal Serial Peripheral Interface (OSPI) module is a kind of Serial Peripheral Interface (SPI) module which allows single, dual, quad or octal read and write access to external flash devices.

The OSPI module is used to transfer data, either in a memory mapped direct mode (for example a processor wishing to execute code directly from external flash memory), or in an indirect mode where the module is set-up to silently perform some requested operation, signaling its completion via interrupts or status registers. For indirect operations, data is transferred between system memory and external flash memory via an internal SRAM which is loaded for writes and unloaded for reads by a device master at low latency system speeds. Interrupts or status registers are used to identify the specific times at which this SRAM should be accessed using user programmable configuration registers.

12.3.2.1.1 OSPI Features

The OSPI modules have the following features:

- Support for single, dual, quad (QSPI mode) or octal (on MCU_FSS0_OSPI0 only) I/O instructions.
- Support dual Quad-SPI mode for fast boot applications.
- Memory mapped ‘direct’ mode of operation for performing flash data transfers and executing code from flash memory.
- Software triggered 'indirect' mode of operation for performing low latency and non-processor intensive flash data transfers.
- Local SRAM of configurable size to reduce advanced high-performance bus overhead and buffer flash data during indirect transfers.
- Set of software advanced peripheral bus accessible flash control registers to perform any flash command, including data transfers up to 8-bytes at a time.
- Additional addressable memory bank to accommodate more than 8-bytes at a time.
- Support for XIP, sometimes referred to as continuous mode.
- Support for DDR Mode and DTR protocol (including Octal DDR protocol with DQS for Octal-SPI devices)
- Programmable device sizes.
- Programmable write protected regions to block system writes from taking effect.
- Programmable delays between transactions.
- Legacy mode allowing software direct access to low level transmit and receive FIFOs, bypassing the higher layer processes.
- An independent reference clock to decouple bus clock from SPI clock – allows slow system clocks.
- Programmable baud rate generator to generate OSPI clocks.
- Features included to improve high speed read data capture mechanism.
- Option to use adapted clocks or DQS to further improve read data capturing.
- Programmable interrupt generation.
- Up to four external device selects - OSPI and QSPI devices can be mixed:
  - MCU_FSS0_OSPI0 has four chip-selects
  - MCU_FSS0_OSPI1 has two chip-selects.
- Programmable data decoder, enables continuous addressing mode for each of the connected devices and auto-detection of boundaries between devices.
- Support BOOT mode.
- Bidirectional CRC on Multiple-SPI interface.
- Handling ECC errors for flash devices with embedded correction engine.
- Full integration with PHY module dedicated to more flexible and power efficient transfers.
- MCU_FSS0_OSPI0 supports RESET_OUT[1-0] and ECC_FAIL pins for external flash devices where ECC is checked on the flash.
- Automatic Flash device status polling for programming operation (Auto HW Polling)

12.3.2.1.2 OSPI Ports

Table 12-144. MCU_FSS0_OSPI Clocks and Resets

Clocks

Copyright © 2024 Texas Instruments Incorporated

Table 12-144. MCU_FSS0_OSPI Clocks and Resets (continued)

Module Clock Input	Description
OSPI_HCLK	MCU_FSS0_OSPI data transfer clock
OSPI_PCLK	MCU_FSS0_OSPI configuration clock
OSPI_RCLK	MCU_FSS0_OSPI Reference clock. Mux controlled by CTRLMMR_MCU_OSPI0_CLKSEL[0] CLK_SEL in *Control Module (CTRL_MMR)*
Resets	
Module Reset Input	**Description**
MCU_FSS0_OSPI_RST	MCU_FSS0_OSPI reset

Table 12-145. MCU_FSS0_OSPI Hardware Requests

Interrupt Requests		
Module Interrupt Signal	**Description**	**Type**
MCU_FSS0_OSPI_LVL_INTR_0	MCU_FSS0_OSPI interrupt	Level
MCU_FSS0_OSPI_0_OSPI_ECC_CORR_LVL_INTR_0	MCU_FSS0_OSPI ECC correctable error interrupt	Level
MCU_FSS0_OSPI_0_OSPI_ECC_UNCORR_LVL_INTR_0	MCU_FSS0_OSPI ECC uncorrectable error interrupt	Level

12.3.2.2 OSPI Environment

This section describes the OSPI external connections (environment).

Table 12-146 lists and describes the MCU_FSS0_OSPI I/O signals.

Table 12-146. OSPI I/O Signals

Module Pin	I/O[(1)]	Description
DQ0	IO	MCU_FSS0_OSPI0 data input/output 0
DQ1	IO	MCU_FSS0_OSPI0 data input/output 1
DQ2	IO	MCU_FSS0_OSPI0 data input/output 2
DQ3	IO	MCU_FSS0_OSPI0 data input/output 3
DQ4	IO	MCU_FSS0_OSPI0 data input/output 4
DQ5	IO	MCU_FSS0_OSPI0 data input/output 5
DQ6	IO	MCU_FSS0_OSPI0 data input/output 6
DQ7	IO	MCU_FSS0_OSPI0 data input/output 7
N_SS_OUT0	O	MCU_FSS0_OSPI0 external flash device chip select 0
N_SS_OUT1	O	MCU_FSS0_OSPI0 external flash device chip select 1
N_SS_OUT2	O	MCU_FSS0_OSPI0 external flash device chip select 2
N_SS_OUT3	O	MCU_FSS0_OSPI0 external flash device chip select 3
OCLK	O	MCU_FSS0_OSPI0 clock output for the external flash device
	O	MCU_FSS0_OSPI0 external loopback output
DQS	I[(2)]	MCU_FSS0_OSPI0 data strobe / external loopback input
RESET_OUT0	O	MCU_FSS0_OSPI0 reset output 0 for the external flash device
RESET_OUT1	O	MCU_FSS0_OSPI0 reset output 1 for the external flash device
ECC_FAIL	I	MCU_FSS0_OSPI0 ECC status from the external flash device
DQ0	IO	MCU_FSS0_OSPI1 data input/output 0
DQ1	IO	MCU_FSS0_OSPI1 data input/output 1
DQ2	IO	MCU_FSS0_OSPI1 data input/output 2
DQ3	IO	MCU_FSS0_OSPI1 data input/output 3
N_SS_OUT0	O	MCU_FSS0_OSPI1 external flash device chip select 0
N_SS_OUT1	O	MCU_FSS0_OSPI1 external flash device chip select 1
OCLK	O	MCU_FSS0_OSPI1 clock output for the external flash device
	O	MCU_FSS0_OSPI1 external loopback output

Copyright © 2024 Texas Instruments Incorporated

Table 12-146. OSPI I/O Signals (continued)

Module Pin	I/O[(1)]	Description
DQS	I[(2)]	MCU_FSS0_OSPI1 data strobe / external loopback input

(1) I = Input; O = Output
(2) When used as an external loopback input, the DQS signal can alternatively be referred to as LBCLKI. The LBCLKI clock input signal is a looped back version of the LBCLKO clock output signal and facilitates easier timing closure at higher speeds. The loopback has to be at board level in order to support higher OSPI speeds. The source of the loopback clock is defined by CTRLMMR_MCU_OSPIn_CLKSEL[4] LOOPCLK_SEL bit in *Control Module (CTRL_MMR)*.

Copyright © 2024 Texas Instruments Incorporated

12.3.2.3 OSPI Functional Description

12.3.2.3.1 OSPI Block Diagram

Figure 12-73 shows the OSPI module block diagram.

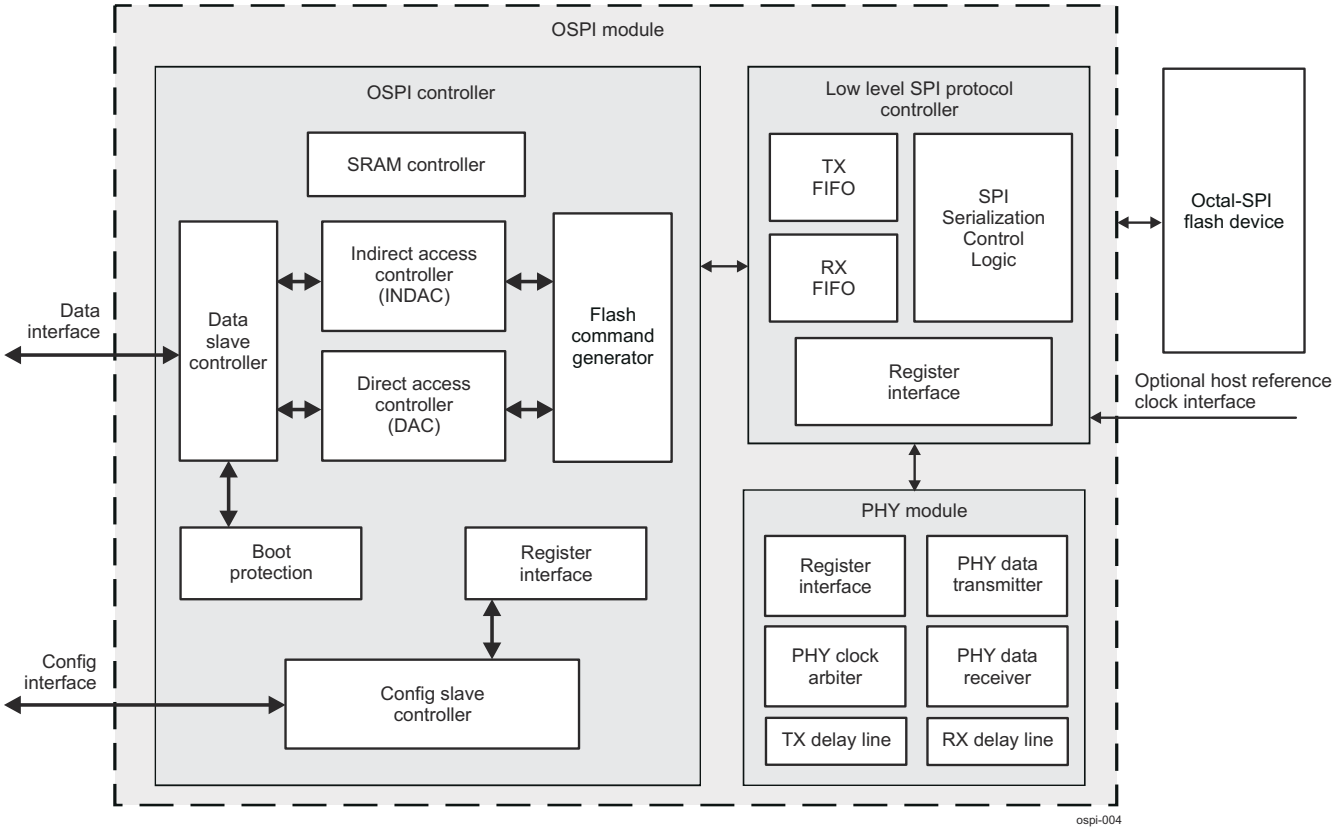


Figure 12-73. OSPI Block Diagram

The OSPI module is composed of three main blocks. The first one is the OSPI controller, the second one is the low level SPI protocol controller, and the third one is the integrated PHY.

The OSPI module has the following two slave interfaces:

- Data slave interface intended for data transfer.
- Configuration slave interface intended for accessing the programmable set of registers.

12.3.2.3.1.1 Data Slave Interface

The data interface is used for data transfer to external flash devices in direct and indirect mode of operation. The data slave controller validates incoming data accesses, responds to invalid requests, performs any required byte and halfword reordering, blocks writes that violate the programmed write protection rules (only for direct access) and forwards the transfer request to either the direct access controller (DAC) or the indirect access controller (INDAC).

The data interface bus is 32-bits wide. Therefore only byte, halfword and word accesses are permitted. When the controller is configured to work in SPI Octal DDR Mode or Octal DDR Protocol (where 2 bytes are collected within single SPI clock cycle what exceeds the size of 1 byte transfer request), 8 bit transfer size is not allowed.

Copyright © 2024 Texas Instruments Incorporated

Note

Cache line wrap accesses over the data slave port should be word aligned.

Data slave port doesn't support cache line wrap bursts of 128 bytes.

12.3.2.3.1.2 Configuration Slave Interface

The configuration interface is used to configure the OSPI module and perform software controlled flash accesses using the OSPI_FLASH_CMD_CTRL_REG register (for more information refer to Section 12.3.2.3.11, *Software Triggered Instruction Generator (STIG)*). Depending on the address it routes the incoming interconnect transfer to the Low level SPI protocol controller or to the ECC aggregator. The configuration port is also used to interact with the OSPI configuration and SRAM ECC registers.

Note

The configuration interface supports only 32-bit accesses. For single byte or halfword manipulations software should perform read-modify-write operations.

12.3.2.3.1.3 OSPI Clock Domains

The OSPI module has two main clock sources for the Octal-SPI controller.

- For interface clocks
- For reference clock

The source for the interface clocks corresponds to the configuration and data buses. The data bus clock (OSPI_HCLK) is the main system clock used to transfer data over the data bus between a master on the system interconnect and the OSPI module. The data bus clock also drives the internal OSPI SRAM. The configuration bus clock (OSPI_PCLK) is used to access the OSPI configuration register and perform basic configuration and for interrupt handling. The OSPI reference clock (OSPI_RCLK) drives the SPI transmit and receive logic in the OSPI module. It is also used to generate the output SPI protocol clock (OSPI_OCLK) and for oversampling of the input data. Using the reference clock (OSPI_RCLK) allows the OSPI module to decouple the frequency of the SPI flash device from the device system clocks, thereby providing more flexible clocking solution.

Note

There is no particular clock ratio requirement between configuration (OSPI_PCLK) and data bus (OSPI_HCLK) clocks.

12.3.2.3.2 OSPI Modes

Note

Some of the OSPI features described in this section may not be supported on this family of devices. For more information, see *OSPI Not Supported Features*.

The OSPI module supports four SPI modes. These modes are defined through the OSPI_CONFIG_REG[1] SEL_CLK_POL_FLD and OSPI_CONFIG_REG[2] SEL_CLK_PHASE_FLD bits. The SEL_CLK_POL_FLD bit defines the clock polarity and the SEL_CLK_PHASE_FLD bit defines the data launch and data capture relation to the OSPI clock edges. Table 12-147 gives a brief description of these modes.

Table 12-147. OSPI Modes

SPI Mode	SEL_CLK_POL_FLD	SEL_CLK_PHASE_FLD	Description
0	0	0	Clock inactive state: low
			Data launch edge: clock falling edge
			Data capture edge: clock rising edge
1	0	1	Clock inactive state: low
			Data launch edge: clock rising edge
			Data capture edge: clock falling edge

Copyright © 2024 Texas Instruments Incorporated

Table 12-147. OSPI Modes (continued)

SPI Mode	SEL_CLK_POL_FLD	SEL_CLK_PHASE_FLD	Description
2	1	0	Clock inactive state: high
			Data launch edge: clock rising edge
			Data capture edge: clock falling edge
3	1	1	Clock inactive state: high
			Data launch edge: clock falling edge
			Data capture edge: clock rising edge

Octal flash devices provide DQS signal which allows source synchronous capture, but for Quad flash devices the OSPI module has a loopback mode. In this loopback mode the clock, looped back at board level, is used for registering the input data, and the edge used is same as the launch edge, thus giving a full cycle path (for more information, see *Read Data Capture*).

12.3.2.3.2.1 Read Data Capture

Figure 12-74 shows the Read Data Capture Logic in the OSPI module.

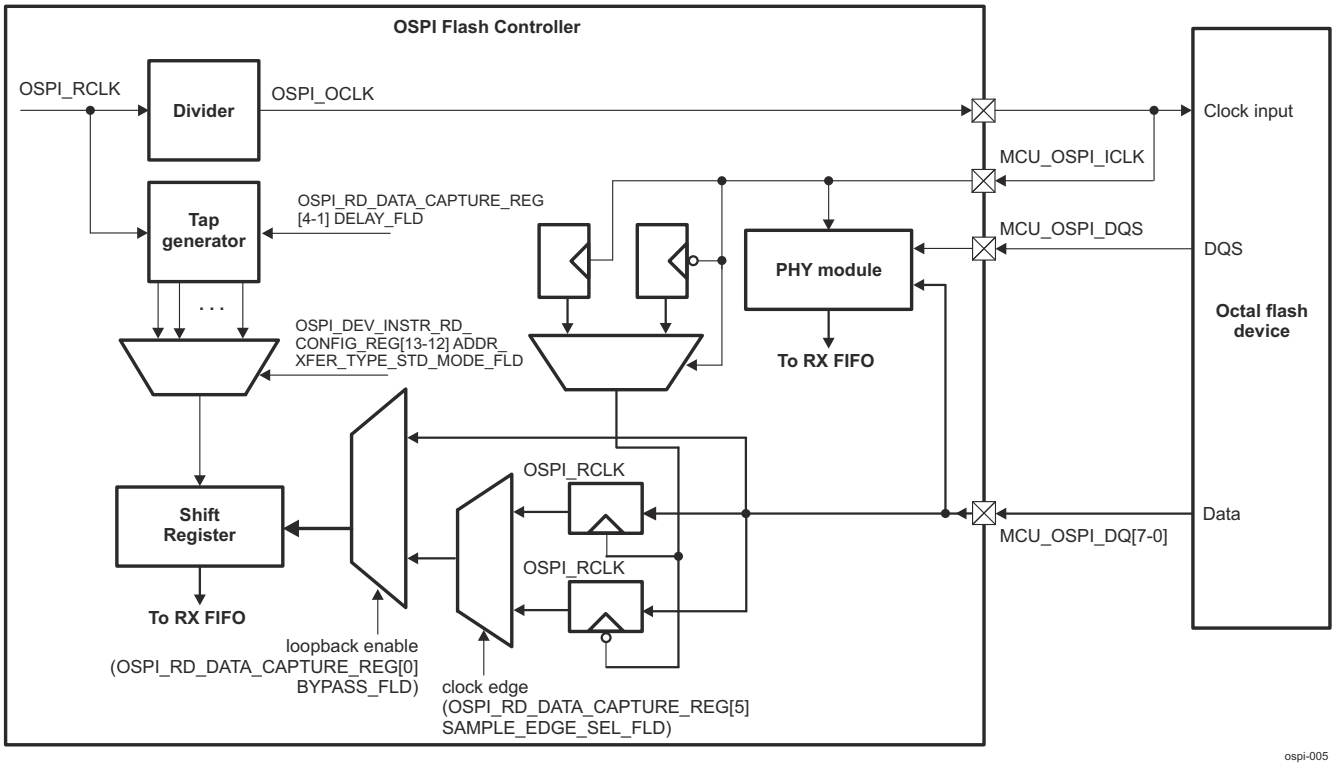


Figure 12-74. Read Data Capture Logic

The PHY module includes a DLL which allows adjustment of the sampling edge with respect to the incoming data to achieve maximum frequency. There are three sources for the sampling signal:

- The reference clock
- Output SPI clock external loopback
- The DQS (only available in Octal Flash devices)

The loopback mode (only for Quad flash devices) can work in two cases. The first one is when OSPI_CONFIG_REG[2] SEL_CLK_PHASE_FLD=0. When SEL_CLK_PHASE_FLD=1 there aren't enough clock falling edges for the register pipeline to catch the last data driven, thus causing a functional failure. Additionally, since the capture edge is falling edge, it gives a full cycle input path only in SPI mode 0, that is when SEL_CLK_POL_FLD=0 and SEL_CLK_PHASE_FLD=0. Thus SPI mode 0 is the first of two modes that support high MHz operation (greater than 50 MHz). The second mode is when SEL_CLK_PHASE_FLD=1 and

Copyright © 2024 Texas Instruments Incorporated

SEL_CLK_PHASE_FLD=1 (SPI mode 3). In this case the missing clock falling edge is compensated inside the OSPI controller when using the incorporated PHY module by inverting the loopback clock.

The loopback mode is enabled by writing 0x0 to OSPI_RD_DATA_CAPTURE_REG[0] BYPASS_FLD. The taps are selected by programming OSPI_RD_DATA_CAPTURE_REG[4-1] DELAY_FLD field. The taps delay the read data capturing logic by the programmed number of OSPI_RCLK cycles.

12.3.2.3.2.1.1 Mechanisms of Data Capturing

There are two mechanisms of data capturing in the OSPI module. They can be combined in some parts to ensure reliable sampling solution independent on the system requirements and the controller configuration. The mechanisms are as follows:

- Data capturing mechanism using taps
- Data capturing mechanism using PHY module.

12.3.2.3.2.1.2 Data Capturing Mechanism Using Taps

This section describes the data capturing mechanism where sampling point is adjusted for one of the reference clock edges inside divided OSPI clock.

After POR, the adapted loopback clock circuit and the OSPI_RCLK delay register line both wake in a disabled state. The OSPI_RD_DATA_CAPTURE_REG register provides the control for the mechanism using taps.

OSPI_RD_DATA_CAPTURE_REG[5] SAMPLE_EDGE_SEL_FLD bit selects the edge of the reference clock, on which data outputs from flash memory are sampled.

OSPI_RD_DATA_CAPTURE_REG[4-1] DELAY_FLD bit field controls the additional number of read data capture cycles (this is the fast reference clock, running at least x4 of the device clock) that should be applied to the internal read data capture circuit. The large clock-to-out delay of the flash memory together with trace delays as well as other device delays may impose a maximum flash clock frequency which is less than the flash memory device itself can operate at. To compensate, software shall set this register to a value that guarantees robust data captures.

12.3.2.3.2.1.3 Data Capturing Mechanism Using PHY Module

PHY module is responsible for data capturing. More detailed description of all internal PHY sampling mechanisms is included in Section 12.3.2.3.16.2, *Read Data Capturing by the PHY Module*.

12.3.2.3.2.2 External Pull Down on DQS

Per the OSPI protocol, the FLASH device drives DQS while CS is asserted. When CS is not asserted the FLASH device presents HiZ on DQS. When configured to use DQS, the controller uses the DQS as a clock, which samples the incoming data into a FIFO. Noise on the DQS when it is HiZ can cause spurious false triggering of the FIFO and filling it with invalid data. There is no way to clear this data except to reset the OSPI module.

To avoid this issue, it is recommended to add a pull down on the DQS line.

During device wakeup, before the IO ring is configured properly, the CS to the FLASH device is HiZ. Depending on the actual level of the CS line the FLASH device might drive the DQS High, Low or HiZ. A pull down on DQS forces the DQS input to Low, but the DQS might still be High or in the presence of noise there might be transitions between Low and High. This again can cause the same issue of capturing garbage data in the Controller FIFO.

To avoid this issue it is recommended to release the OSPI from reset only after the IO ring is configured properly.

12.3.2.3.3 OSPI Power Management

Note

The OSPI module does not provide any hardware signal for busy or idle status. Software need to ensure that the OSPI module is idle before clocks can be shut off by reading the OSPI_CONFIG_REG[31] IDLE_FLD bit.

Copyright © 2024 Texas Instruments Incorporated

OSPI_PCLK and OSPI_HCLK share the same clock stop request/acknowledge and clock enable/acknowledge interface.

12.3.2.3.4 Auto HW Polling

The OSPI controller is capable of automatically testing the Flash device busy bit to guarantee no reads or writes are ignored by the flash when it is busy burning in programmed data.

At the end of a programming transaction, the Flash device goes into a burn-in state and becomes busy.

When Auto HW Polling is enabled, the OSPI controller keeps track of programming transactions and will initiate a Flash status read polling transactions automatically, until Flash indicates it is not busy, before any additional data read or programming operations are sent to the flash device. See OSPI_WRITE_COMPLETION_CTRL_REG register and the associated registers.

The OSPI controller requires that the OSPI_WRITE_COMPLETION_CTRL_REG[23-16] POLL_COUNT_FLD field should always be set with values greater or equal to 3 (>=3).

12.3.2.3.5 Flash Reset

OSPI provides Flash reset out ports. These ports are active low and controlled thru OSPI_CONFIG_REG register.

12.3.2.3.6 OSPI Memory Regions

Table 12-148 shows the OSPI memory map in MCU domain.

Table 12-148. OSPI Memory Map

Address Range		Size	Description
MCU_FSS0_OSPI0	MCU_FSS0_OSPI1		
0x00 4704 0000 to 0x00 4704 0100	0x00 4705 0000 to 0x00 4705 0100	256 B	Configuration registers space
0x00 4704 4000 to 0x00 4704 4200	0x00 4705 4000 to 0x00 4705 4200	512 B	Global control registers space
0x00 4706 8000 to 0x00 4706 8400	0x00 4706 4000 to 0x00 4706 4400	1 KB	OSPI_ECC_AGGR registers space

Note

For more information about the memory space, see Section 12.3.1.3.4, *FSS Memory Regions*.

12.3.2.3.7 OSPI Interrupt Requests

The OSPI module generates three interrupts per instance. The ECC interrupts (MCU_FSS0_OSPI_0_OSPI_ECC_CORR_LVL_INTR_0 and MCU_FSS0_OSPI_0_OSPI_ECC_UNCORR_LVL_INTR_0 for MCU_FSS0_OSPI0; MCU_FSS0_OSPI_1_OSPI_ECC_CORR_LVL_INTR_0 and MCU_FSS0_OSPI_1_OSPI_ECC_UNCORR_LVL_INTR_0 for MCU_FSS0_OSPI1) are generated by the ECC aggregator.

The other interrupts (MCU_FSS0_OSPI0_LVL_INTR_0 for MCU_FSS0_OSPI0 and MCU_FSS0_OSPI1_LVL_INTR_0 for MCU_FSS0_OSPI1) are generated by the OSPI module.

Table 12-149 lists the event flags and the corresponding mask bits of the sources which can cause interrupts.

Copyright © 2024 Texas Instruments Incorporated

Table 12-149. OSPI Events

Event Flag	Event Mask	Description
OSPI_IRQ_STATUS_REG[0] MODE_M_FAIL_FLD	OSPI_IRQ_MASK_REG[0] MODE_M_FAIL_MASK_FLD	Event Flag and Event Mask for the OSPI Interrupts.
OSPI_IRQ_STATUS_REG[1] UNDERFLOW_DET_FLD	OSPI_IRQ_MASK_REG[1] UNDERFLOW_DET_MASK_FLD	
OSPI_IRQ_STATUS_REG[2] INDIRECT_OP_DONE_FLD	OSPI_IRQ_MASK_REG[2] INDIRECT_OP_DONE_MASK_FLD	
OSPI_IRQ_STATUS_REG[3] INDIRECT_READ_REJECT_FLD	OSPI_IRQ_MASK_REG[3] INDIRECT_READ_REJECT_MASK_FLD	
OSPI_IRQ_STATUS_REG[4] PROT_WR_ATTEMPT_FLD	OSPI_IRQ_MASK_REG[4] PROT_WR_ATTEMPT_MASK_FLD	
OSPI_IRQ_STATUS_REG[5] ILLEGAL_ACCESS_DET_FLD	OSPI_IRQ_MASK_REG[5] ILLEGAL_ACCESS_DET_MASK_FLD	
OSPI_IRQ_STATUS_REG[6] INDIRECT_XFER_LEVEL_BREACH_FLD	OSPI_IRQ_MASK_REG[6] INDIRECT_XFER_LEVEL_BREACH_MASK_FLD	
OSPI_IRQ_STATUS_REG[7] RECV_OVERFLOW_FLD	OSPI_IRQ_MASK_REG[7] RECV_OVERFLOW_MASK_FLD	
OSPI_IRQ_STATUS_REG[8] TX_FIFO_NOT_FULL_FLD	OSPI_IRQ_MASK_REG[8] TX_FIFO_NOT_FULL_MASK_FLD	
OSPI_IRQ_STATUS_REG[9] TX_FIFO_FULL_FLD	OSPI_IRQ_MASK_REG[9] TX_FIFO_FULL_MASK_FLD	
OSPI_IRQ_STATUS_REG[10] RX_FIFO_NOT_EMPTY_FLD	OSPI_IRQ_MASK_REG[10] RX_FIFO_NOT_EMPTY_MASK_FLD	
OSPI_IRQ_STATUS_REG[11] RX_FIFO_FULL_FLD	OSPI_IRQ_MASK_REG[11] RX_FIFO_FULL_MASK_FLD	
OSPI_IRQ_STATUS_REG[12] INDRD_SRAM_FULL_FLD	OSPI_IRQ_MASK_REG[12] INDRD_SRAM_FULL_MASK_FLD	
OSPI_IRQ_STATUS_REG[13] POLL_EXP_INT_FLD	OSPI_IRQ_MASK_REG[13] POLL_EXP_INT_MASK_FLD	
OSPI_IRQ_STATUS_REG[14] STIG_REQ_INT_FLD	OSPI_IRQ_MASK_REG[14] STIG_REQ_MASK_FLD	
OSPI_IRQ_STATUS_REG[16] RX_CRC_DATA_ERR_FLD	OSPI_IRQ_MASK_REG[16] RX_CRC_DATA_ERR_MASK_FLD	
OSPI_IRQ_STATUS_REG[17] RX_CRC_DATA_VAL_FLD	OSPI_IRQ_MASK_REG[17] RX_CRC_DATA_VAL_MASK_FLD	
OSPI_IRQ_STATUS_REG[18] TX_CRC_CHUNK_BRK_FLD	OSPI_IRQ_MASK_REG[18] TX_CRC_CHUNK_BRK_MASK_FLD	
OSPI_IRQ_STATUS_REG[19] ECC_FAIL_FLD	OSPI_IRQ_MASK_REG[19] ECC_FAIL_MASK_FLD	

Copyright © 2024 Texas Instruments Incorporated

Table 12-149. OSPI Events (continued)

Event Flag	Event Mask	Description
OSPI_ECC_SEC_STATUS_REG0[0] SRAM_PEND	OSPI_ECC_SEC_ENABLE_SET_REG0[0] SRAM_ENABLE_SET OSPI_ECC_SEC_ENABLE_CLR_REG0[0] SRAM_ENABLE_CLR	Event Flag and Event Mask for the ECC Interrupts.
OSPI_ECC_DED_STATUS_REG0[0] SRAM_PEND	OSPI_ECC_DED_ENABLE_SET_REG0[0] SRAM_ENABLE_SET OSPI_ECC_DED_ENABLE_CLR_REG0[0] SRAM_ENABLE_CLR	
OSPI_ECC_AGGR_STATUS_SET[1-0] PARITY	OSPI_ECC_AGGR_ENABLE_SET[0] PARITY	
OSPI_ECC_AGGR_STATUS_SET[3-2] TIMEOUT	OSPI_ECC_AGGR_ENABLE_SET[1] TIMEOUT	
OSPI_ECC_AGGR_STATUS_CLR[1-0] PARITY	OSPI_ECC_AGGR_ENABLE_CLR[0] PARITY	
OSPI_ECC_AGGR_STATUS_CLR[3-2] TIMEOUT	OSPI_ECC_AGGR_ENABLE_CLR[1] TIMEOUT	

12.3.2.3.8 OSPI Data Interface

12.3.2.3.8.1 Data Interface Address Remapping

The incoming data interface address, by default, maps directly to the address sent serially to the FLASH device. If the FLASH device has a 24-bit address, then the 24 LSB's of the data address is forwarded. A remap feature is available to remap all incoming data addresses to ADDRESS + N, where N is the value stored in the OSPI_REMAP_ADDR_REG[31-0] VALUE_FLD bit field. It is enabled via the OSPI_CONFIG_REG[16] ENB_AHB_ADDR_REMAP_FLD bit. This feature could be used when software needs to move boot code to another FLASH region.

12.3.2.3.8.2 Write Protection

In order to protect the FLASH device, a software controlled write protection feature is supported. Any data write detected (by using DAC), pointing to an area of the FLASH that is protected, is not permitted.

A programmable region of the FLASH device, defined as a number of FLASH 'blocks' starting from a particular block number can be protected. Three programmable registers are provided. The first OSPI_LOWER_WR_PROT_REG register defines the FLASH block that is located at the bottom of the region to be protected. The second OSPI_UPPER_WR_PROT_REG register defines the FLASH block that is located at the top of the region to be protected. The third OSPI_WR_PROT_CTRL_REG register is a control register consisting of 2 bits. The OSPI_WR_PROT_CTRL_REG[0] INV_FLD bit allows software to invert the region that is being protected, causing the programmed region to become the only areas of FLASH memory that is not protected from writes. The OSPI_WR_PROT_CTRL_REG[1] ENB_FLD bit is the write protection enable bit. When this bit is set to 0, the FLASH device is unprotected.

For implementation, the data interface must map the incoming address into its associated FLASH block. A block can be between 1 and 65 KB, programmed via the OSPI_DEV_SIZE_CONFIG_REG register.

12.3.2.3.8.3 Access Forwarding

For legal accesses, the data interface will forward all accesses to one of two access controllers - the direct access and the indirect access controllers. Assuming DAC has been enabled via the OSPI_CONFIG_REG[7] ENB_DIR_ACC_CTRL_FLD bit, then by default all accesses will be forwarded to this controller. Before any accesses can be forwarded to INDAC, it must first be configured by software. This process is fully explained in Section 12.3.2.3.10, *Indirect Controller (INDAC)*. If DAC is disabled, any incoming access that cannot be forwarded to INDAC will be completed immediately with an error. If DAC is enabled, the same access will be forwarded and serviced by DAC.

Copyright © 2024 Texas Instruments Incorporated

12.3.2.3.9 OSPI Direct Access Controller (DAC)

Direct access refers to the operation where data interface accesses directly trigger a read or write to FLASH memory. It is memory mapped and can be used to both access and directly execute code from external FLASH memory. Any incoming access that is not recognized as being within the programmable indirect trigger region is assumed to be a direct access and will be serviced by the DAC. Note that accesses that use DAC do not use the embedded SRAM. The data transfer stops when read or write burst is carried out. The amount of wait states applied will be dependent on the latency through the controller. Latency is kept to a minimum when the use of XIP read instructions are enabled (see OSPI_CONFIG_REG[18] ENTER_XIP_MODE_IMM_FLD and OSPI_CONFIG_REG[17] ENTER_XIP_MODE_FLD bits).

12.3.2.3.10 OSPI Indirect Access Controller (INDAC)

12.3.2.3.10.1 Indirect Read Controller

The aim of the indirect mode of operation is to read significant numbers of bytes from FLASH memory without requiring a data interface access to trigger it. Instead indirect operations are controlled and triggered by software via specific control/configuration Indirect Read Transfer registers (OSPI_INDIRECT_READ_XFER_CTRL_REG, OSPI_INDIRECT_READ_XFER_WATERMARK_REG, OSPI_INDIRECT_READ_XFER_START_REG, and OSPI_INDIRECT_READ_XFER_NUM_BYTES_REG). This block will communicate with an embedded low level SPI protocol state machine module to perform an efficient and optimized FLASH read burst, placing the read data into the local SRAM module ready for fast and low latency delivery to any external master.

By default, the Indirect Read controller is disabled. Before enabling it, software must configure how much data is required and the start address. The start address and total number of bytes to be fetched is defined in OSPI_INDIRECT_READ_XFER_START_REG and OSPI_INDIRECT_READ_XFER_NUM_BYTES_REG registers, respectively. Up to two indirect operations can be programmed at any one time. The second operation can be triggered while the first is in progress. Supporting two indirect operations allows a short turnaround time between the completion of one indirect operation and the start of the second. For more information refer to Section 12.3.2.3.10.3, *Indirect Access Queuing*.

The total number of bytes to read in an indirect operation is not limited by the size of the SRAM. The size of SRAM will only limit the size of requests. In the case of SRAM overrun, the controller will back pressure FLASH reads until space becomes available in the SRAM. Back pressuring the reads on the SPI interface is handled by completing any current read burst, waiting until space in the SRAM becomes available and then issuing a new read burst at the address where the previous terminated burst ended.

An external master will be able to fetch the data that the controller has read from external FLASH memory by issuing data interface reads to the OSPI module. The address of the incoming read access must be in the range of indirect trigger address programmed via the OSPI_IND_AHB_ADDR_TRIGGER_REG register to indirect trigger address + 2**(indirect trigger address range) - 1. Default value of the range is equal to 16 locations. This allows a 16-beat burst to be applied starting from the indirect trigger address. The smaller bursts are possible to handle effectively as well with this approach. Furthermore it is not strict requirement to push consecutive address sequence. Actual address just has to be in the indirect range to grant SRAM as source. Each valid Indirect Read will cause the internal SRAM to be popped, thereby decoupling the incoming read access address from the FLASH address – that is not direct mapped. Therefore the indirect trigger address does not have any relationship with the FLASH address. It is just to indicate that data should take the SRAM as source instead of the FLASH memory array after triggering of any valid Indirect Read. The FLASH address for Indirect Read is taken from the OSPI_INDIRECT_READ_XFER_START_REG register. Assuming the requested data is present in the SRAM at the point the data interface access is received by the OSPI module, then the data will be fetched from the SRAM and the response to the read burst will be achieved with minimum latency. Once the data has been read from the SRAM, the OSPI module will free up the associated resource in the SRAM.

If a read access is received whose address is not within the range described above then that access will not be completed using the indirect controller. It will instead be serviced by the direct access controller.

If a read access is received whose address is within the range described above but the requested data is not immediately present in the SRAM then wait states will be applied until the data has been read from FLASH and pushed to the SRAM.

Copyright © 2024 Texas Instruments Incorporated

If a read burst is received whose access elements traverse the Indirect trigger range, then the accesses within the Indirect trigger range will be processed by the indirect controller and the rest will be taken by the direct access controller. This is likely to be a software configuration error.

The external master is only permitted to issue 32-bit data interface reads until the last word of an indirect transfer. This helps keep the SRAM control logic less complex. On the final read, the external master may issue a 16-bit (Halfword) or byte access to complete the transfer. It is also permitted for the external master to always issue a 32-bit Word read on the last indirect access. The controller will pad the upper bits of the response with zero. The current expectation is that the SRAM will be kept fairly full while the read operation is carried out. The fill level of the SRAM is directly readable by software reading the OSPI_SRAM_FILL_REG register.

An indirect operation may be cancelled at any time by setting 1 to OSPI_INDIRECT_READ_XFER_CTRL_REG[1] CANCEL_FLD bit.

Any bus master should be allowed to initiate an indirect access. The OSPI module provide software access mechanism to the SRAM fill-level directly via configuration registers and then decide for itself when the data should be fetched from the local SRAM. The fill level watermark register (see OSPI_INDIRECT_READ_XFER_WATERMARK_REG register) is provided. When the SRAM fill level passes this watermark, an interrupt is generated. If the watermark value is > 0, the watermark interrupt is also generated when the final byte of data has been read by the OSPI module and placed in the SRAM, even if the actual SRAM fill level has not risen above the watermark. This last feature is useful to avoid software tracking how much data has been read and resetting the watermark value for the last few bytes of an indirect read transfer.

Two further interrupt sources are provided to help understand the status of an indirect operation. Firstly, an interrupt is generated when an indirect operation has completed. Secondly, an interrupt is generated if an Indirect Read operation was requested but could not be accepted due to the fact 2 indirect operations have already been buffered by the OSPI module.

Setting the OSPI_INDIRECT_READ_XFER_CTRL_REG[0] START_FLD bit starts an indirect read operation. OSPI_INDIRECT_READ_XFER_CTRL_REG[2] RD_STATUS_FLD bit is available to check the status.

12.3.2.3.10.1.1 Indirect Read Transfer Process

The following sequence can be followed:

1. Setup OSPI_CONFIG_REG register.
2. Setup the indirect transfer's FLASH start address in the OSPI_INDIRECT_READ_XFER_START_REG register.
3. Setup the number of bytes to be transferred in the OSPI_INDIRECT_READ_XFER_NUM_BYTES_REG register.
4. Setup the indirect transfer's trigger address in the OSPI_IND_AHB_ADDR_TRIGGER_REG register.
5. Setup the indirect transfer's trigger address range in the OSPI_INDIRECT_TRIGGER_ADDR_RANGE_REG register.
6. If the watermark interrupt feature is to be used, set the OSPI_INDIRECT_READ_XFER_WATERMARK_REG register which will cause an interrupt to be generated when the fill level increases beyond the watermark level. Setting the watermark can be useful indication to software when to read the next part of the indirect read transfer. Note that if the watermark is set to a value other than zero, the watermark interrupt will always trigger once the final byte of indirect transfer has been fetched and placed in the embedded SRAM, even if the watermark value is higher than the actual completed fill level.
7. Trigger Indirect Read access by setting the OSPI_INDIRECT_READ_XFER_CTRL_REG[0] START_FLD bit to 1.
8. If the watermark interrupt feature is to be used, wait for watermark interrupt. Else poll the SRAM fill level via the OSPI_SRAM_FILL_REG register to decide when sufficient data is in the SRAM to trigger data fetches.
9. Read the expected amount of data from SRAM. If there is still more data to fetch in order to complete the indirect read transfer, then loop back to step 8. Otherwise continue to step 10.
10. The completion status of the Indirect Read operation can be polled via the OSPI_INDIRECT_READ_XFER_CTRL_REG[5] IND_OPS_DONE_STATUS_FLD bit.
11. An Indirect Complete interrupt will be generated when the Indirect read operation has completed.

Copyright © 2024 Texas Instruments Incorporated

12.3.2.3.10.2 Indirect Write Controller

The aim of the indirect mode of operation is to perform bulk transfer of data from the processor into a FLASH memory in the most efficient manner. The fewest possible write cycles inside the FLASH device will be carried out for the indirect transfer, thus maximizing the life of the device. Indirect write operation can be thought of from a software perspective as the inverse of the indirect read. It is controlled and triggered by software via specific control/configuration Indirect Write Transfer registers (for more information see the following registers: OSPI_INDIRECT_WRITE_XFER_CTRL_REG, OSPI_INDIRECT_WRITE_XFER_WATERMARK_REG, OSPI_INDIRECT_WRITE_XFER_START_REG, and OSPI_INDIRECT_WRITE_XFER_NUM_BYTES_REG). This block will await delivery of the write data via the external data interface master, placing it in the local SRAM before communicating with the existing legacy SPI core to perform an efficient and optimized FLASH write burst.

By default, the indirect write controller is disabled. Before enabling it, the software must configure how much data is required and the start address. The start address and total number of bytes to be written is defined in OSPI_INDIRECT_WRITE_XFER_START_REG and OSPI_INDIRECT_WRITE_XFER_NUM_BYTES_REG registers, respectively. Up to two indirect operations can be programmed at any one time. The second operation can be triggered while the first is in progress. Supporting two indirect operations allows a short turnaround time between the completion of one indirect operation and the start of the second. The Indirect write queuing is very similar to indirect read queuing. For more information refer to Section 12.3.2.3.10.3, *Indirect Access Queuing*.

The total number of bytes to write in an indirect operation is not limited by the size of the SRAM. The size of SRAM will only limit the amount of data that can be accepted from the external master. In the case of an SRAM overrun, the controller will back pressure the data interface with wait states. Note the fill level of the SRAM is readable via programmable OSPI_SRAM_FILL_REG register and this can be used to avoid this situation.

An external master will provide the write data and will transfer this to the OSPI module by issuing data interface writes. The address of the incoming write access must be in the range of Indirect trigger address programmed via the OSPI_IND_AHB_ADDR_TRIGGER_REG register to Indirect trigger address + 2**(Indirect trigger address range) - 1. Default value of the range is equal to 16 locations. This allows a 16-beat burst to be applied starting from the Indirect trigger address. The smaller bursts are possible to handle effectively as well with this approach. Furthermore it is not strict requirement to push consecutive address sequence. Actual address just has to be in the Indirect Range to grant SRAM as source. Each write will cause the internal SRAM to be pushed, thereby decoupling the incoming write access address from the FLASH address – that is not direct mapped. Therefore Indirect trigger address does not have any relationship with FLASH address. It is just to indicate that data should take SRAM as source instead of FLASH Memory array after triggering of any valid Indirect Write. The FLASH address for Indirect Write is taken from the OSPI_INDIRECT_WRITE_XFER_START_REG register. Assuming the SRAM is not full at the point the data interface access is received by the OSPI module, then the data will be pushed to the SRAM with minimum latency.

If a write access is received whose address is not within the range described above then that access will not be completed using the indirect controller. It will instead be serviced by the direct access controller.

If a write access is received whose address is within the range described above but the SRAM is full then wait states will be applied until some or all of the data has been pushed from the SRAM to the FLASH.

If a write burst is received whose access elements traverse the Indirect trigger range, then the accesses within the Indirect trigger range will be processed by the indirect controller, and the rest will be taken by the direct access controller. This is likely to be a software configuration error.

The external master is only permitted to issue 32-bit data interface writes until the last word of an indirect transfer. This helps keep the SRAM control logic less complex. On the final write, the external master may issue a 32-bit word, 16-bit (halfword) or a byte access to complete the transfer. If the number of bytes to write is less than 4 on the last transfer, the master is still permitted to issue a 32-bit transfer. In these cases, the extra bytes are discarded by the controller.

When the SRAM holds a number of bytes equal to or greater than the size of a FLASH page (which itself is programmed into the OSPI module, with a default of 256 bytes) or when the SRAM holds all remaining bytes of the currently executing indirect transfer, the OSPI module will initiate a write burst to the flash command generator.

Copyright © 2024 Texas Instruments Incorporated

An indirect operation may be cancelled at any time by setting 1 to the OSPI_INDIRECT_WRITE_XFER_CTRL_REG[1] CANCEL_FLD bit.

Any bus master should be allowed to initiate an indirect access. The OSPI module provide software access mechanism to the SRAM fill-level directly via the configuration registers and then decide for itself when the data should be written to the local SRAM. The fill level watermark register (see OSPI_INDIRECT_WRITE_XFER_WATERMARK_REG register) is provided. When the SRAM fill level falls below this watermark, an interrupt is generated.

Two further interrupt sources are provided to help understand the status of an indirect operation. Firstly, an interrupt is generated when an indirect operation has completed. Secondly, an interrupt is generated if an indirect write operation was requested but could not be accepted due to the fact 2 indirect operations have already been buffered by the OSPI module.

Setting the OSPI_INDIRECT_WRITE_XFER_CTRL_REG[0] START_FLD bit starts an indirect write operation. The OSPI_INDIRECT_WRITE_XFER_CTRL_REG[2] WR_STATUS_FLD bit is available to check the status.

12.3.2.3.10.2.1 Indirect Write Transfer Process

The following sequence can be followed:

1. Setup OSPI_CONFIG_REG register.
2. Setup the indirect transfer's FLASH start address in the OSPI_INDIRECT_WRITE_XFER_START_REG register.
3. Setup the number of bytes to be transferred in the OSPI_INDIRECT_WRITE_XFER_NUM_BYTES_REG register.
4. Setup the indirect transfer's trigger address in the OSPI_IND_AHB_ADDR_TRIGGER_REG register.
5. Setup the indirect transfer's trigger address range in the OSPI_INDIRECT_TRIGGER_ADDR_RANGE_REG register.
6. It is functionally valid for software to simply write all the data to the SRAM in one block transfer. However, if the total number of bytes to write is greater than the size of the partitioned SRAM, then it is quite likely the SRAM will become full causing the OSPI to back-pressure the system data bus for a considerable time. This time is based on the FLASH data-rate and the page-write time of the device. To avoid sending all the write data in one block transfer, software can make use of the watermark interrupt to identify a convenient time to send data a page at a time to the SRAM module. Alternatively, software can poll the SRAM fill level register directly to identify how empty the SRAM is at any one time in order to make a judgment as to when the most practical time to send the next part of the transfer.
7. If the watermark interrupt feature is to be used, set the OSPI_INDIRECT_WRITE_XFER_WATERMARK_REG register which will cause an interrupt to be generated when the fill level falls below the watermark. The watermark should be set to a number between zero and a page size. That is if the page size is 256 bytes, then setting the watermark to a value between 10 and 250 is reasonable and will cause the interrupt to trigger when the fill level drops below the programmed number. Setting the watermark can be useful to provide an indication to software when to write the next page of data to the SRAM.
8. Trigger Indirect Write access by setting OSPI_INDIRECT_WRITE_XFER_CTRL_REG[0] START_FLD bit.
9. If the remaining number of bytes still to be transferred into the SRAM for the current indirect transfer is greater than a FLASH page, then write 1 FLASH page worth of data to the SRAM. Otherwise send the remaining data from the indirect transfer to SRAM.
10. If all the data in the indirect transfer has now been sent to the SRAM, then go to 12 and await indirect complete status. Otherwise if there is more data still to be transferred then either:
    - If the watermark interrupt feature is being used, then wait for watermark interrupt.
    - Alternatively the SRAM fill level can be interrogated to identify a convenient time to send more data.
11. Loop back to 9.
12. Optional: The completion status of the Indirect write operation can be polled via OSPI_INDIRECT_WRITE_XFER_CTRL_REG[5] IND_OPS_DONE_STATUS_FLD.
13. An Indirect Complete interrupt will be generated when the Indirect write operation has completed.

Copyright © 2024 Texas Instruments Incorporated

12.3.2.3.10.3 Indirect Access Queuing

Software is permitted to queue up to two indirect transfers for both the indirect write controller and the indirect read controller. Supporting two indirect operations allows a short turnaround time between the completion of one indirect operation and the start of the second. Any attempt to queue more than two operations will cause an interrupt to be generated. To take advantage of this feature, software should attempt to keep both indirect programming slots full at all times.

From the software perspective, indirect access queuing is achieved by triggering bit 0 of the indirect transfer control register (OSPI_INDIRECT_READ_XFER_CTRL_REG[0] START_FLD bit or OSPI_INDIRECT_WRITE_XFER_CTRL_REG[0] START_FLD bit) twice in short succession. The indirect number of bytes register (OSPI_INDIRECT_READ_XFER_NUM_BYTES_REG or OSPI_INDIRECT_WRITE_XFER_NUM_BYTES_REG register) and the indirect FLASH start address register (OSPI_INDIRECT_READ_XFER_START_REG or OSPI_INDIRECT_WRITE_XFER_START_REG register) must be setup with the relevant transfer data before START_FLD bit can be triggered for each transfer. Since these registers will change regularly, the hardware must keep sampled versions of these registers for the duration of the indirect transfer.

The internal register block will only issue an indirect start trigger to the key underlying datapath blocks one at a time. There are 2 independent datapath blocks in the indirect access controller that will receive and independently sample this information. The first is the datapath block on the data bus side of the SRAM. For indirect reads, this is a read interface, for indirect writes, it is a write interface. The second is the datapath block on the FLASH side of the SRAM. For indirect reads, this is a write interface, for indirect writes, it is a read interface. Both blocks will process the indirect transfers at different times. For example, for an indirect read operation, the datapath block on the FLASH side of the SRAM will be able to start processing the second queued transfer as soon as the last byte of the first transfer has been written to the SRAM. Before commencing the second transfer, this block must resample the OSPI_INDIRECT_READ_XFER_NUM_BYTES_REG and OSPI_INDIRECT_READ_XFER_START_REG registers. Similarly, the datapath block on the bus side will resample the same registers locally when it has forwarded all the FLASH data associated with the first indirect transfer from the SRAM onto the data bus.

12.3.2.3.10.4 Consecutive Writes and Reads Using Indirect Transfers

It is permitted for software to trigger an indirect read operation while an indirect write operation is in progress. Similarly it is permitted to trigger an indirect write while an indirect read operation is in progress. Indirect write operations will take overall precedence.

12.3.2.3.10.5 Accessing the SRAM

The SRAM depth is separated in two segmets. The lower segment is reserved for indirect read use. The upper segment is for indirect write use only. The size of each segment is programmable via the OSPI_SRAM_PARTITION_CFG_REG register. This feature allows to allocate how many bits of the SRAM address bus are allocated to indirect read. By default, this is set so that exactly half of the SRAM is portioned for use by the indirect read controller. To ensure the read data bus is not directly fed by the SRAM read data through combinatorial logic, an extra bank of holding registers is included in the indirect read data path. These registers act as an extra location to be added to the allocated number of SRAM locations for indirect read.

To illustrate how the SRAM (and the extra bank of holding registers) can be allocated between indirect read and write, the following example is provided. The depth of the SRAM in this example is configured to be 8 bits. This is equal to 256 locations.

- If the OSPI_SRAM_PARTITION_CFG_REG[7-0] ADDR_FLD field is set to 0x00, then 256 locations are allocated to indirect writes and 1 location to indirect reads.
- If the OSPI_SRAM_PARTITION_CFG_REG[7-0] ADDR_FLD field is set to 0x01, then 255 locations are allocated to indirect writes and 2 locations to indirect reads.
- If the OSPI_SRAM_PARTITION_CFG_REG[7-0] ADDR_FLD field is set to 0x02, then 254 locations are allocated to indirect writes and 3 locations to indirect reads.
- And so on until.
- If the OSPI_SRAM_PARTITION_CFG_REG[7-0] ADDR_FLD field is set to 0xFD, then 3 locations are allocated to indirect writes and 254 locations to indirect reads.

Copyright © 2024 Texas Instruments Incorporated

- If the OSPI_SRAM_PARTITION_CFG_REG[7-0] ADDR_FLD field is set to 0xFE, then 2 locations are allocated to indirect writes and 255 locations to indirect reads.
- If the OSPI_SRAM_PARTITION_CFG_REG[7-0] ADDR_FLD field is set to 0xFF, then 1 location is allocated to indirect writes and 256 locations to indirect reads.

Note

A value of 0xFF or 0x00 in the OSPI_SRAM_PARTITION_CFG_REG register should be avoided by software, as only the bottom 8 bits of the SRAM fill level are accessible through software (up to 255 limit) via the OSPI_SRAM_FILL_REG register. If the fill level reaches 256 on either the indirect read or write side, it will appear when reading the Fill Level to be 0.

There are four SRAM sources that are arbitrated and muxed onto the single SRAM port. Up to three sources can access this port at any one time. The sources are described as follows:

- Indirect Write, Write source. This is located on the data bus side of the SRAM.
- Indirect Write, Read source. This is located on the FLASH side of the SRAM.
- Indirect Read, Write source. This is located on the FLASH side of the SRAM.
- Indirect Read, Read source. This is located on the data bus side of the SRAM.

A fixed priority arbitration scheme is implemented. Table 12-150 shows priority allocated to these sources.

Table 12-150. SRAM Access Priority

SRAM Access Priority		
Indirect Write	Write to SRAM (from System Data Bus)	3rd (exclusive with Data Bus Read Request)
	Read from SRAM (from OSPI Module)	2nd
Indirect Read	Write to SRAM (from OSPI Module)	1st
	Read from SRAM (from System Data Bus)	3rd (exclusive with Data Bus Write Request)

Note

With the exception of the write port during an Indirect Read operation (on the FLASH side of the SRAM), the logic driving all four sources must not assume single cycle completion. Writes to the SRAM during an indirect read must be allowed to complete immediately to avoid data loss. Therefore this port is given maximum priority.

12.3.2.3.11 OSPI Software-Triggered Instruction Generator (STIG)

The DAC and INDAC are used to transfer data. In order to access the volatile and non-volatile configuration registers, the legacy SPI Status register, other status/protection registers as well as to perform ERASE functions, a separate software controller is required. The software triggered instruction generator (STIG) is controlled using the OSPI_FLASH_CMD_CTRL_REG register by setting up the command to issue to the FLASH device. This is a generic controller and can be used to perform any instruction that the FLASH device supports from the extended SPI protocol. Configuring of instructions which are not compliant with the specification of the FLASH devices could cause unpredicted behavior of the controller. OSPI_FLASH_CMD_CTRL_REG[31-24] CMD_OPCODE_FLD bits should be set different than OSPI_DEV_INSTR_RD_CONFIG_REG[7-0] RD_OPCODE_NON_XIP_FLD and OSPI_DEV_INSTR_WR_CONFIG_REG[7-0] WR_OPCODE_FLD. The OSPI_FLASH_CMD_CTRL_REG[0] CMD_EXEC_FLD bit is used to trigger the command. The OSPI_FLASH_CMD_CTRL_REG[1] CMD_EXEC_STATUS_FLD bit is used by software to poll the status of the command execution. For reads, when the command has been serviced (OSPI_FLASH_CMD_CTRL_REG[1] CMD_EXEC_STATUS_FLD bit toggles from '1' to '0'), up to 8 bytes of read data will be placed in the OSPI_FLASH_RD_DATA_LOWER_REG and OSPI_FLASH_RD_DATA_UPPER_REG registers. For writes, the write data should be placed in the OSPI_FLASH_WR_DATA_LOWER_REG and OSPI_FLASH_WR_DATA_UPPER_REG registers.

The completion of the STIG request could be also checked by the corresponding interrupt. The occurrence of the interrupt indicates that the controller is ready for accepting a new STIG request. It is important to notice that completion of the STIG request is not equivalent to completion it on SPI side. For example, if STIG is configured

Copyright © 2024 Texas Instruments Incorporated

to the command composed of data to transmit only, the data is taken from the corresponding STIG register fields and put into TX FIFO. Since all bytes to write are known, another STIG can be queued before serialization of the current one is completed.

There are some commands which require more data to read than 8 bytes (for example READ ID command). The additional STIG Memory Bank is implemented in order to accommodate these data if needed. The STIG Memory Bank (internal component of the controller) is controlled by the OSPI_FLASH_CMD_CTRL_REG[2] STIG_MEM_BANK_EN_FLD bit. If enabled, the number of bytes to read in the STIG is extended to the value ranging from 16 to 512 as defined in OSPI_FLASH_COMMAND_CTRL_MEM_REG[18-16] NB_OF_STIG_READ_BYTES_FLD bit field. It should be noticed that there are very few commands (excluding Read Array ones which are not intended to handle effectively in STIG Mode but in Direct/Indirect Modes) which return more than 8 bytes to the controller. If the maximum number of bytes to Read using STIG in target application is less than 512, the depth of the STIG Memory Bank can be set smaller what will result in saving noticeable part of the area.

If number of bytes to Read in the STIG as defined in OSPI_FLASH_COMMAND_CTRL_MEM_REG[18-16] NB_OF_STIG_READ_BYTES_FLD bit field exceeds the Memory Bank Depth, remaining data will overwrite the STIG Memory Bank locations starting from its first address. OSPI_FLASH_RD_DATA_LOWER_REG and OSPI_FLASH_RD_DATA_UPPER_REG keep the last 8 bytes read from the Flash Device by STIG when Memory Bank is enabled. Therefore, for example if the user wants to get just a single byte from the last eight bytes from long continuous read SPI data chain, there is no need to access the STIG Memory Bank since data can be taken from suitable Flash Command Read Data register. In order to access more data, STIG Memory Bank data request should be triggered. It is controlled by the OSPI_FLASH_COMMAND_CTRL_MEM_REG and works analogously for triggering STIG from the functional standpoint.

OSPI_FLASH_COMMAND_CTRL_MEM_REG[0] TRIGGER_MEM_BANK_REQ_FLD bit is used to trigger the command, bit OSPI_FLASH_COMMAND_CTRL_MEM_REG[1] MEM_BANK_REQ_IN_PROGRESS_FLD is used by software to poll the status of the command execution. When MEM_BANK_REQ_IN_PROGRESS_FLD bit toggles from "1" to "0", the byte of data (OSPI_FLASH_COMMAND_CTRL_MEM_REG[15-8] MEM_BANK_READ_DATA_FLD) from corresponding address (OSPI_FLASH_COMMAND_CTRL_MEM_REG[28-20] MEM_BANK_ADDR_FLD bit field) is valid. The address should be set before triggering the STIG Memory Bank access. Each consecutive STIG access overwrites the previous one so that the data in the Bank always fit into byte index fetched by the last STIG access configured to use the Memory Bank (first incoming byte equals first address of the Memory Bank, second one equals the second address and so on).

12.3.2.3.11.1 Servicing a STIG Request

A STIG request will cause the OSPI Flash controller to interrogate the OSPI_FLASH_CMD_CTRL_REG register to determine what and how many bytes it should send to the FLASH device. The OSPI_FLASH_CMD_CTRL_REG[31-24] CMD_OPCODE_FLD field of this register indicate the instruction to be sent and is always pushed first. If there is an address to send, then the address (the size of which is also programmed in the same register) is sent next. The address itself is stored in the OSPI_FLASH_CMD_ADDR_REG register. If Mode bits are enabled by OSPI_FLASH_CMD_CTRL_REG[18] ENB_MODE_BIT_FLD bit. OSPI_MODE_BIT_CONFIG_REG[7-0] MODE_FLD bit field are being sent right after address. If OSPI_FLASH_CMD_CTRL_REG[18] ENB_MODE_BIT_FLD and OSPI_CONFIG_REG[29] CRC_ENABLE_FLD are both enabled, STIG will replace XIP Mode bits (not applicable for CRC aware SPI interface) for automatically calculated address CRC byte. Therefore, to execute CRC aware STIGs (meaning the commands requiring sending address CRC byte), ENB_MODE_BIT_FLD bit should always be set. If there are any dummy cycles to send (the size of which is also programmed in OSPI_FLASH_CMD_CTRL_REG register) then those are sent next. If there is data to write or read (the size of which is also programmed in OSPI_FLASH_CMD_CTRL_REG register) then for the case of writes, up to 8 bytes can be sent (as stored in the Flash Command Write Data registers, OSPI_FLASH_WR_DATA_LOWER_REG and OSPI_FLASH_WR_DATA_UPPER_REG registers) next. In the read case, when the read data has been collected from the FLASH device, the OSPI Flash Controller stores that in the Flash Command Read Data Registers (OSPI_FLASH_RD_DATA_LOWER_REG and OSPI_FLASH_RD_DATA_UPPER_REG registers). Up to 8 bytes can be get if OSPI_FLASH_CMD_CTRL_REG[2] STIG_MEM_BANK_EN_FLD bit is disabled or up to 512 when enabled. When the OSPI Flash controller starts to service a STIG request, it sets the

Copyright © 2024 Texas Instruments Incorporated

OSPI_FLASH_CMD_CTRL_REG[1] CMD_EXEC_STATUS_FLD bit to indicate a command execution is in progress. When the OSPI Flash controller is in the auto-polling state, servicing a STIG request is slightly different. Most of devices are largely inaccessible after a program operation until the device has completed that write. Some group of them has a possibility to suspend programming page. It can be controlled by the OSPI_POLLING_FLASH_STATUS_REG[8] DEVICE_STATUS_VALID_FLD bit, which indicate active auto-polling phase. After requesting a STIG, the OSPI Flash Controller immediately issues appropriate OPCODE to Memory. During servicing a STIG (in auto-polling phase) the status bit of command execution remains steady and other parts of transfer such as ADDRESS or DUMMY BITS, and so forth, are disabled (to issued Program Suspend Command is needed OPCODE only). There is a programmable option to add delay between every repetitive poll operation (delay is defined by OSPI_WRITE_COMPLETION_CTRL_REG[31-24] POLL_REP_DELAY_FLD bit field). This feature is implemented to free up SPI bandwidth if needed.

Note

The OSPI data is sent LSB first, while address is sent MSB first.

Note

The STIG complete status bit gets cleared before the actual flash access completes. Software should wait for about 700 ns if there is any dependency on actual access completion.

12.3.2.3.11.2

Note

The OSPI data is sent LSB first, while address is sent MSB first.

Note

The STIG complete status bit gets cleared before the actual flash access completes. Software should wait for about 700 ns if there is any dependency on actual access completion.

12.3.2.3.12 OSPI Arbitration Between Direct / Indirect Access Controller and STIG

When multiple controllers are active simultaneously, a simple fixed-priority arbitration scheme is used to arbitrate between each interface and access the external FLASH. The fixed priority is defined as follows, highest priority first.

- The Indirect Access Write
- The Direct Access Write
- The STIG
- The Direct Access Read
- The Indirect Access Read

12.3.2.3.13 OSPI Command Translation

Requests issued by the direct access controller, the indirect access controller or the STIG will be translated into a sequence of byte transfers to send downstream (before serialization to the FLASH device). These sequences depend on the requested transfer but an example of a typical 1-byte non sequential READ is shown below:

INSTRUCTION OPCODE -> ADDRESS -> Mode Byte -> Dummy Bytes -> 1 byte of don't care

For sequential accesses, an extra byte of data per read is pushed to the FLASH device on the back of the above sequence assuming it can be done so with no gap between each transferred byte.

When PHY mode is enabled and consequently no clock divider is configured, latency caused by multi domain synchronization may make an extra byte insufficient to avoid the transfer gap. To ensure the sequential access non-interrupted and keep the maximum performance of the controller, PHY Pipeline Mode is implemented. When enabled, number of don't care bytes is calculated based on the configuration.

Copyright © 2024 Texas Instruments Incorporated

The actual sequence sent to the FLASH device depends on the requested transfers, whether the transfer is non-sequential or sequential, whether the device has been configured in XIP mode and the state of the main Device Instruction Type programmable registers (OSPI_DEV_INSTR_RD_CONFIG_REG and OSPI_DEV_INSTR_WR_CONFIG_REG).

For writes, the write enable latch (or WEL) within the FLASH device itself must be high before a write sequence can be issued. The OSPI Flash Controller will automatically issue the write enable latch command before triggering a write command via the direct or indirect access controllers (DAC/INDAC) – that is the user does not need to perform this operation. For increasing flexibility and performance user can turn off this feature by setting the OSPI_DEV_INSTR_WR_CONFIG_REG[8] WEL_DIS_FLD bit. The opcode for WREN is typically 0x06 and is common between devices.

When write requests from the direct or indirect access controllers are no longer being received and all outstanding requests have been sent, the FLASH device will automatically start the page program write cycle. Any incoming request at this time will be held in wait states until the cycle has completed. The OSPI Flash Controller will automatically poll the FLASH device legacy SPI status register to identify when the write cycle has completed. This is achieved by sending the RDSR opcode to the FLASH device and waiting until the device itself has indicated the write cycle has completed (until the Write in Progress bit has cleared to zero and the write enable latch bit has also cleared to zero or device is ready bit has set to one). The WREN and the RDSR device instructions are the only ones that are sent by the controller under the hood. For any other specific instruction that the user determines should be sent to the device (for example if the device needs to be unprotected before a write command is issued), these should be handled separately by issuing FLASH commands via the STIG.

There is an option to trigger HOLD or RESET feature on I/Os of the Flash Device. The HOLD one is generally common across the devices and takes an alternative function of DQ3 pin (applicable when device operates neither in Quad SPI mode nor DDR). The transfer can be hold and then resumed by dedicated software trigger field (OSPI_CONFIG_REG[4] HOLD_PIN_FLD). The devices which have the HOLD feature on DQ3 usually need another dedicated pin for hardware reset and ones without HOLD feature usually have alternative reset on DQ3 what makes the additional reset pin being redundant. The controller supports both variants and reset selection register field (OSPI_CONFIG_REG[6] RESET_CFG_FLD) allows the user to configure which hardware reset solution is implemented in the device under usage.

After configuration is done, it is possible to trigger HOLD or RESET features using I/Os (OSPI_CONFIG_REG[4] HOLD_PIN_FLD or OSPI_CONFIG_REG[5] RESET_PIN_FLD bits). After HOLD activation the controller is introduced into waiting state and any other operations should not be requested before de-asserting of HOLD configuration bit. The HOLD feature is useful when any SPI transaction needs to be prolonged in order to adjust it into specific point in time. Note that any HOLD trigger issued during active SPI transaction may be synchronized into reference clock domain at the time the SPI transfer turns to be finished. In this case, there is nothing to hold so the low level SPI logic will not activate HOLD on DQ3. To check if HOLD request suspended the transfer OSPI_CONFIG_REG[31] IDLE_FLD bit can be polled for. If SPI is not in the IDLE state, the transfer was successfully suspended. It is important for the software to take care of resetting OSPI_CONFIG_REG[4] HOLD_PIN_FLD bit before newly triggered SPI transaction. In case HOLD request is set before the beginning of the transfer it will be HOLD right after it starts what may not always be a goal. The hardware RESET needs to be activated when CS is high (no valid transaction is present on SPI bus). It can be checked by polling of OSPI_CONFIG_REG[31]. If the controller is in the IDLE state and no other transfer requests are queued to perform, the hardware RESET can be triggered. The RESET feature is useful when any write, program or erase operation needs to be cancelled. No transfer request is permitted before driving the reset back to being inactive. Triggering HOLD or RESET on DQ3 at the time the device is configured to work in Quad SPI mode or DDR will overwrite transfer data on DQ3 with '0'. This behavior is considered as a software error so it is advisable for the system to make sure that the flash device was introduced to suitable SPI mode (that is by polling its configuration register) before triggering alternative DQ3 function. There are four independent reset outputs implemented to separate between multiple devices connected to the controller (up to 4 are supported). The decision which reset output is to be activated after triggering OSPI_CONFIG_REG[5] RESET_PIN_FLD bit is made based on OSPI_CONFIG_REG[9] PERIPH_SEL_DEC_FLD and OSPI_CONFIG_REG[13-10] PERIPH_CS_LINES_FLD bits. Reset output OSPI_ECC_VECTOR is to be directly driven into corresponding dedicated RESET pins of the devices with separated RESET pin and alternatively, Reset output OSPI_ECC_VECTOR is to be

Copyright © 2024 Texas Instruments Incorporated

control OSPI_ECC_VECTOR of the DQ3 RESET devices enabling separating of DQ3 Master Outputs on SoC integration level.

The controller supports all combinations of CPHA and CPOL for Serial Clock. It allows the controller to support any SPI slave devices not limited to Flash Memories. Multiple-SPI flash devices use just a subset of these combinations depending on the Transfer Mode as defined in Table 12-151.

Table 12-151. Flash SPI Modes

(SEL_CLK_POL_FLD, SEL_CLK_PHASE_FLD)	Edge Mode	Support
0x00 (SPI MODE 0)	SDR	Yes
0x01 (SPI MODE 1)	SDR	No
0x10 (SPI MODE 2)	SDR	No
0x11 (SPI MODE 3)	SDR	No
0x00 (SPI MODE 0)	DDR	Yes
0x01 (SPI MODE 1)	DDR	No
0x10 (SPI MODE 2)	DDR	No
0x11 (SPI MODE 3)	DDR	No

12.3.2.3.14 Selecting the Flash Instruction Type

In order to send the correct READ and WRITE opcodes, software should initialize the OSPI_DEV_INSTR_RD_CONFIG_REG and the OSPI_DEV_INSTR_WR_CONFIG_REG registers. These registers include fields to setup the required instruction opcodes that is intended to be used to access the FLASH (default is basic READ and basic page program) as well as the instruction type, edge mode (DDR or SDR) and whether the instruction uses single, dual, quad or octal pins for address and data transfer. Providing this level of control to the user provides a future proofed generic solution. To ensure the controller can operate from a reset state, the registers will be reset to an opcode compatible with SIO devices what can be modified using BOOT feature.

Despite being applicable for both READs and WRITEs, the OSPI_DEV_INSTR_RD_CONFIG_REG[9-8] INSTR_TYPE_FLD field only appears once – it is not included in the OSPI_DEV_INSTR_WR_CONFIG_REG register. If software sets this to anything other than '0', then the address transfer type and the data transfer type bits of both OSPI_DEV_INSTR_RD_CONFIG_REG and OSPI_DEV_INSTR_WR_CONFIG_REG registers become don't care. It is made available to allow software to support the less common FLASH instructions where the opcode, address and data are sent on 2 or 4 lanes (the opcode from most instructions are sent serially to the FLASH device, even for dual/quad instructions).

There are devices capable to handling Read Operations in Dual Data Rate Mode (DDR) (it is also called Dual Transfer Rate Mode (DTR)). That means they can issue and capture the data on both rising and falling edges during working with dedicated command type. This enables the controller to maintain throughput at twice lower frequency of OSPI clock. The Device Read Instruction Register has DDR enable bit which informs Octal-SPI Flash Controller that opcode written into Read Opcode field is capable with DDR command type. The other field defined in OSPI_RD_DATA_CAPTURE_REG[19-16] DDR_READ_DELAY_FLD which enables the controller to shift the transmitted data in DDR mode. By default, data are shifted by 1 clock cycle to ensure hold timing greater than 0 during DDR transactions. It may not be sufficient for high reference clock frequency in accordance with the high dividers.

Table 12-152 shows how software should configure the OSPI module for selected specific READ and WRITE instruction supported by the abovementioned device.

Table 12-152. READ and WRITE Instruction Configuration

READ

Copyright © 2024 Texas Instruments Incorporated

Table 12-152. READ and WRITE Instruction Configuration (continued)

OPCODE	OPCODE sent over how many lanes / edge mode?	ADDRESS / DUMMY / MODE sent over how many lanes / edge mode?	DATA bytes sent over how many lanes / edge mode?	Instruction Type (OSPI_DEV_INSTR_RD_CONFIG_REG[9-8] INSTR_TYPE_FLD)	Address transfer type (OSPI_DEV_INSTR_RD_CONFIG_REG[13-12] ADDR_XFER_TYPE_STD_MODE_FLD)	Data transfer type (OSPI_DEV_INSTR_RD_CONFIG_REG[17-16] DATA_XFER_TYPE_EXT_MODE_FLD)	DDR bit enable (OSPI_DEV_INSTR_RD_CONFIG_REG[10] DDR_EN_FLD)
READ	1/SDR	1/SDR	1/SDR	0	0	0	0
FAST_READ	1/SDR	1/SDR	1/SDR	0	0	0	0
DTR_FAST_READ	1/SDR	1/DDR	1/DDR	0	0	0	1
DOFR (Dual O/p Fast Read)	1/SDR	1/SDR	2/SDR	0	0	1	0
DIOFR (Dual I/O Fast Read)	1/SDR	2/SDR	2/SDR	0	1	1	0
DDIOFR (DTR Dual I/ O Fast Read)	1/SDR	2/DDR	2/DDR	0	1	1	1
QOFR (Quad O/p Fast Read)	1/SDR	1/SDR	4/SDR	0	0	2	0
QIOFR (Quad I/O Fast Read)	1/SDR	4/SRD	4/SDR	0	2	2	0
DQIOFR (DTR Quad I/ O Fast Read)	1/SDR	4/DDR	4/DDR	0	2	2	1
OOFR (Octal O/p Fast Read)	1/SDR	1/SDR	8/SDR	0	0	3	0
OIOFR (Octal I/O Fast Read)	1/SDR	8/SDR	8/SDR	0	3	3	0
DOIOFR (DTR Octal O/p Fast Read)	1/SDR	1/DDR	8/DDR	0	0	3	1
4DOIOFR (4-byte DTR Octal I/O Fast Read)	1/SDR	8/DDR	8/DDR	0	3	3	1
DCFR (Dual Command Fast Read)	2/SDR	2/SDR	2/SDR	1	Don't care	Don't care	0
DDCFR (DTR Dual Command Fast Read)	2/SDR	2/DDR	2/DDR	1	Don't care	Don't care	1
QCFR (Quad Command Fast Read)	4/SDR	4/SRD	4/SDR	2	Don't care	Don't care	0
DQCFR (DTR Quad Command Fast Read)	4/SDR	4/DDR	4/DDR	2	Don't care	Don't care	1
OCFR (Octal Command Fast Read)	8/SDR	8/SDR	8/SDR	3	Don't care	Don't care	0
4DOCFR (4-byte DTR Octal Command Fast Read)	8/SDR	8/DDR	8/DDR	3	Don't care	Don't care	1
WRITE							

Copyright © 2024 Texas Instruments Incorporated

Table 12-152. READ and WRITE Instruction Configuration (continued)

OPCODE	OPCODE sent over how many lanes?	ADDRESS / DUMMY / MODE sent over how many lanes?	DATA bytes sent over how many lanes?	Instruction Type (OSPI_DEV_INSTR_RD_CONFIG_REG[9-8] INSTR_TYPE_FLD)	Address transfer type (OSPI_DEV_INSTR_WR_CONFIG_REG[13-12] ADDR_XFER_TYPE_STD_MODE_FLD)	Data transfer type (OSPI_DEV_INSTR_WR_CONFIG_REG[17-16] DATA_XFER_TYPE_EXT_MODE_FLD)
PP	1	1	1	0	0	0
DIFP (Dual Input Fast Program)	1	1	2	0	0	1
DIEFP (Dual Input Extended Fast Program)	1	2	2	0	1	1
QIFP (Quad Input Fast Program)	1	1	4	0	0	2
QIEFP (Quad Input Extended Fast Program)	1	4	4	0	2	2
OIFP (Octal Input Fast Program)	1	1	8	0	0	3
OIEFP (Octal Input Extended Fast Program)	1	8	8	0	3	3
DCPP (Dual Command Fast Program)	2	2	2	1	Don't care	Don't care
QCPP (Quad Command Fast Program)	4	4	4	2	Don't care	Don't care
OCPP (Octal Command Fast Program)	8	8	8	3	Don't care	Don't care

Note

This data are applicable for both 3-byte or 4-byte address variants of the commands if did not indicate otherwise.

Note

In DTR protocol all transfer phases (including opcode) take DDR edge mode independently on the command under execution. DTR protocol is to be enabled by OSPI_CONFIG_REG[24] ENABLE_DTR_PROTOCOL_FLD bit. It has higher priority than DDR Mode enable bit from OSPI_DEV_INSTR_RD_CONFIG_REG[10] DDR_EN_FLD.

12.3.2.3.15 OSPI Data Integrity

The CRC aware SPI transfer can be performed when both controller and device are configured to work in the Octal DDR Protocol.

For write transactions (the controller transmits data throughout all transfer), the controller is responsible for sending address CRC byte (XOR of all address bytes) following address bytes and TX data CRC byte (XOR of all data bytes to write) following data chunk with size as defined in OSPI_MODE_BIT_CONFIG_REG[10-8] CHUNK_SIZE_FLD bit field. All CRC data are being calculated and sent automatically by the controller and the external device is responsible for reacting accordingly on any possible interpolation on the Flash interface. For read transactions, the controller is also responsible for sending address CRC byte (like

Copyright © 2024 Texas Instruments Incorporated

for write) and for getting and progressing RX data CRC byte returning by the Flash Device after each chunk with size as defined in OSPI_MODE_BIT_CONFIG_REG[10-8] CHUNK_SIZE_FLD bit field. At the time when the Flash Device is returning data back to the controller, the controller dynamically calculates checksum byte by byte. Once the chunk is completed, CRC returned from Flash Device should fit to dynamically calculated CRC by the controller. In case of any deviation, controller reports CRC error to the system by corresponding interrupt (CRC error interrupt). The controller also provides the last captured CRC data in RX data chunk (defined in OSPI_MODE_BIT_CONFIG_REG[31-24] RX_CRC_DATA_LOW_FLD and OSPI_MODE_BIT_CONFIG_REG[23-16] RX_CRC_DATA_UP_FLD bit fields) to give the software driver the opportunity to further detecting any data corruption on system interfaces. The CRC data valid interrupt informs the system about the accessibility of the new RX CRC data in the registers. Once the system gets the full data word, it can calculate CRC by itself. At the time it collects all data words in chunk and then gets the CRC data valid interrupt, it can compare these data and react accordingly.

Some devices also have embedded ECC mechanism allowing them to report data abnormal conditions on their ECC Correction Signal output. At the time this output turns low, the device expect the OSPI controller to read status register of the device in order to get more details about the source of detected abnormal situation. The OSPI controller investigates ECC status on its ECC_FAIL input and generates an interrupt when detecting this signal being low.

12.3.2.3.16 OSPI PHY Module

OSPI module fully integrates PHY module dedicated to more flexible and power efficient transfers.

The PHY module communicates with the OSPI Flash controller via the aforementioned PHY Interface and handles data transfer on low-level stage of design hierarchy. However, when the OSPI_RCLK is configured to be equal to the SPI clock instead of alternative approach using clock divider, there is just one OSPI_RCLK cycle (not 4 or more) within single SPI period or half period for DDR Mode (SPI Control Module works on reference clock). Given that OSPI_RCLK is the input clock for RX FIFO and the output one for TX FIFO, the PHY solution incurs more restrictive requirement for value of system clock in order to synchronize data without SPI transfer interruption. For example, when the controller operates in DDR 1× octal Mode, 2 bytes of data (equivalent to one RX FIFO location) is gathered within just single OSPI_RCLK cycle. The controller cannot predict next data access while operating in the Direct Mode (meaning its size or whether it is sequential to the previous one or not). As a result, if the OSPI_HCLK is not significantly greater than OSPI_RCLK, the SPI transfer has to be suspended until the Flash Command Generator forwards new data to TX FIFO.

An optional PHY Pipeline Mode is implemented to avoid the necessity of stable clocking of the system clock for the Direct Mode when the PHY mode is enabled and to keep maximum performance while ensuring correct operation of the OSPI controller with the PHY using low frequencies from all its domains. This mode is a trade-off between large software overhead when operating in the Indirect Mode and the described limitations present in the Direct Mode. For more information about PHY Pipeline Mode, see Section 12.3.2.3.16.1, *PHY Pipeline Mode*.

When DDR 2× Mode is granted based on configuration – SPI transfer is automatically performed using the PHY module even if the OSPI_CONFIG_REG[3] PHY_MODE_ENABLE_FLD is de-asserted. SDR 2× commands are handled with PHY module paths being bypassed. Nevertheless, dividers of 2, 4 or 6 for DDR and divider of 2 for SDR should not be configured based on controller requirements and these configurations are perceived as a software error.

The following steps are an example of software algorithm of adapting the OSPI controller with the PHY module incorporated to work in octal 1× clock DDR Protocol. Note that all necessary configuration steps described in Section 12.3.2.4.2, *Configuring the OSPI Controller for Optimal Use* shall be completed before the algorithm.

1. Set PHY mode enable (OSPI_CONFIG_REG[3] PHY_MODE_ENABLE_FLD bit) and DDR protocol (OSPI_CONFIG_REG[24] ENABLE_DTR_PROTOCOL_FLD bit). It is assumed that device is configured to work in DDR Protocol.
2. Before setting the DLL parameters, software calibration could be needed. OSPI_PHY_MASTER_CONTROL_REG[23] PHY_MASTER_BYPASS_MODE_FLD bit controls the bypass mode of the master and slave DLLs. If this bit is set, the DLL bypass mode is enabled. This mode is

Copyright © 2024 Texas Instruments Incorporated

intended to be used only for debug. When set to 0, a Master operational mode is selected, when set to 1 the Bypass mode is selected.
DLL works in normal mode of operation where the slave delay line settings are used as fractional delay of the master delay line encoder reading of the number of delays in one cycle.
Master DLL is disabled with only 1 delay element in its delay line. The slave delay lines decode delays in absolute delay elements rather than as fractional delays.

- DLL Bypass Mode (follow only if operating in this mode):
  - Depending on frequency of reference clock, calculate how many delay elements should be used to shift this clock by 25% of its period (best case for DDR transfers from setup/hold timings standpoint). Note that delay could be slightly different in a real design. TX Delay is configured in OSPI_PHY_CONFIGURATION_REG[22-16] PHY_CONFIG_TX_DLL_DELAY_FLD bit field.
  - Re-synchronize DLLs by asserting OSPI_PHY_CONFIGURATION_REG[31] PHY_CONFIG_RESYNC_FLD bit (If this bit is already set by previous re-synchronization, toggle sequence from "0" to "1" must be generated in order to trigger re-synchronization DLL logic) and set PHY bypass mode enable through OSPI_PHY_MASTER_CONTROL_REG[23] PHY_MASTER_BYPASS_MODE_FLD bit.
- DLL Master Mode (follow only if operating in this mode):
  - Drive DLL reset bit OSPI_PHY_CONFIGURATION_REG[30] PHY_CONFIG_RESET_FLD into low.
  - Calculate initial delay value for the Master DLL according to the OSPI_PHY_MASTER_CONTROL_REG[6-0] PHY_MASTER_INITIAL_DELAY_FLD bit field.
  - Depending on frequency of reference clock, calculate how many delay elements should be used to shift this clock by 25% of its period (best case for DDR transfers from setup/hold timings standpoint). Note that delay could be slightly different in a real design. TX Delay is configured in OSPI_PHY_CONFIGURATION_REG[22-16] PHY_CONFIG_TX_DLL_DELAY_FLD bit field.
  - Re-synchronize DLLs by asserting OSPI_PHY_CONFIGURATION_REG[31] PHY_CONFIG_RESYNC_FLD (If this bit is already set by previous re-synchronization, toggle sequence from "0" to "1" must be generated in order to trigger re-synchronization DLL logic) and set DLL reset bit back to high (since both bits are within the same register, it is acceptable to set both bits simultaneously).
  - Poll OSPI_DLL_OBSERVABLE_LOWER_REG[15] DLL_OBSERVABLE_LOWER_LOOPBACK_LOCK_FLD bit. When set – lock is done.
  - Re-synchronize Slave DLLs by asserting OSPI_PHY_CONFIGURATION_REG[31] PHY_CONFIG_RESYNC_FLD bit (If this bit is already set by previous re-synchronization, toggle sequence from "0" to "1" must be generated in order to trigger re-synchronization DLL logic) and set TX DLL Delay (OSPI_PHY_CONFIGURATION_REG[22-16] PHY_CONFIG_TX_DLL_DELAY_FLD) and RX DLL Delay (OSPI_PHY_CONFIGURATION_REG[6-0] PHY_CONFIG_RX_DLL_DELAY_FLD) fields which are equivalent to percentage clock offsets now. It is recommended to wait for the new configuration being propagated by 20 reference clock cycles before triggering the next SPI transfer.
- Consider Read Data from location where its value is predictable. This step can be performed in different ways, depending on the device. Parameter Page, ID, Status, Data from OTP region or Data from location of Flash Array the value of which is known can act as the pattern.
- Trigger Read request chosen from above options.
- Check correctness of data and store that information:
  - Increment value of RX clock delay – it is configurable in the OSPI_PHY_CONFIGURATION_REG[6-0] PHY_CONFIG_RX_DLL_DELAY_FLD bit field.
  - Re-synchronize DLLs.
  - Trigger valid Read request.
  - Check correctness of data and store information.
  - If range boundary of RX clock delay is achieved, go to step 3. Otherwise go back to step "Increment value of RX clock delay".

3. Set RX clock delay value for one from the middle of valid range based on information in storage.
4. Re-synchronize DLLs.
5. Set OSPI_DEV_INSTR_RD_CONFIG_REG for Octal Read DDR Configuration (each transfer phase should be configured to work in Octal mode, Number of Dummy cycles should be set as specified in the

Copyright © 2024 Texas Instruments Incorporated

documentation of the device or more when because of additional read paths delays of actual systems data is predicted to be flopped by PHY module with delay excesses actual cycle of SPI clock generated by the controller).

6. Enable Pipeline mode in the OSPI_CONFIG_REG[25] PIPELINE_PHY_FLD bit.
7. Perform Sequential Read of Data consistent with conditions indicated within Section 12.3.2.3.16.1, *PHY Pipeline Mode*.
8. After de-asserting the data slave select signal by software – poll OSPI_CONFIG_REG[31] IDLE_FLD bit.
9. When it is asserted to high – next transfer request can be triggered.

12.3.2.3.16.1 PHY Pipeline Mode

This mode is used for Direct Read Mode of operation. If any other operations are intended to be executed, it is recommended to disable PHY Pipeline Mode and re-enable for subsequent Direct Reads in PHY mode. Since there is comprehensive software mechanism controlling Read data transfers in Indirect Mode, pipeline of data interface accesses is not effective for this mode. Enable PHY Pipeline feature when at least four 4-byte-sized data words are predicted to be read in sequentially. The Flash Command Generator pipelines and puts them into TX FIFO which causes CS to remain active because low level SPI protocol controller controls TX FIFO fill level. In order to correctly trigger Direct Read in Pipeline Mode TX FIFO must be empty. Therefore first polling of OSPI_CONFIG_REG[31] IDLE_FLD bit needs to be done. The sequential data transfer will be interrupted when the data slave select signal of the data interface is asserted to low. This information is also detected by Data Slave Module which informs the Flash Command Generator that the next access is invalid and a TX FIFO locations can be flushed transparently for the system.

In PHY Pipeline Mode it is recommended for Data Master not to introduce wait states in between consecutive occurrences of the data interface signal that indicates transfer has finished. It will ensure regular transfer rate on data side. Introducing wait states gradually slows data transfer rate down and may finally cause SPI transfer interruption because of TX FIFO data starvation. The system, however, may need to introduce some number of wait states after completion of sequential transfer (composed of 4-byte sized data words) for progressing the data. The dedicated buffer is implemented in Data Slave Controller which collects all incoming data during wait states injection. In order to keep SPI transfer uninterrupted, number of wait states should be as little as possible. The higher the OSPI_HCLK/ OSPI_RCLK ratio the more wait states can be introduced without SPI transfer interruption. In case the system is able to launch a new transfer before wait states overflow, buffered data transfer to the host will continue. It compensates slowed down transfer by introducing wait states. In case the system is not able to launch a new transfer before wait states overflow, next incoming transfer is considered non-sequential and is executed after all pipelined data is flushed.

This mode can be enabled when following conditions are met:

- OSPI_HCLK > OSPI_RCLK (Comparing the slow data clock with the fast reference one makes Pipeline Mode ineffective – Suspend of SPI Transfer would be possible. Consequently, this condition has to be met to operate in this mode.)
- Only 4-byte sized Data Words are permitted (This ensures more data clock cycles for synchronization of FIFOs between consecutive pulses of the signal indicating transfer has finished.)
- The transfer with introduced wait states or non-sequential transfers can only be triggered in between at least four 4-byte sized Data Bursts sequential accesses (16 Bytes) to be sure that Data Master can trust buffered incoming data during wait states injection.
- Do not use Pipeline Mode along with Continuous Mode (XIP). Benefit of XIP is limited for bulk data transfers intended to execute in Pipeline Mode.

12.3.2.3.16.2 Read Data Capturing by the PHY Module

Read Data Capturing by the PHY module is useful, as the user is not responsible for the design dedicated DLL being compatible with the Octal-SPI Flash Controller. Another benefit is an option to adjust both SPI clock and sampling clock in a very wide range to fit them into individual requirements of any system. If loopback clock (OSPI_RD_DATA_CAPTURE_REG[0] BYPASS_FLD) and PHY mode (OSPI_CONFIG_REG[3] PHY_MODE_ENABLE_FLD) are both enabled, the loopback clock is driven into RX DLL instead of gated reference clock. Because of the architecture of DLL, loopback clock needs to be provided in SPI Mode 0.
If DQS (OSPI_RD_DATA_CAPTURE_REG[8] DQS_ENABLE_FLD) and PHY mode (OSPI_CONFIG_REG[3] PHY_MODE_ENABLE_FLD) are both enabled, the DQS is driven into RX DLL instead of gated reference clock.

Copyright © 2024 Texas Instruments Incorporated

12.3.2.4 OSPI Programming Guide

12.3.2.4.1 Configuring the OSPI Controller for Use After Reset

The OSPI controller has been designed to wake up in a state that is suitable for performing basic reads and writes using the direct access controller. The BASIC read (opcode 0x03) and BASIC write (opcode 0x02) instructions are operations supported by all target devices. The controller also wakes up with a baud rate divider setting of divide-by-32. Assuming the reference clock is operating at 400 MHz after reset, then this means the effective SPI clock is just 12.5 MHz. This should be slow enough to meet all timing requirements of all target devices without any further device programming.

If the target device does not use 3 address bytes, the device size configuration register must be modified to the appropriate size.

If software plans to write to the device, and the number of bytes per device page is not equal to 256, then the device size configuration register must also be modified.

While not a requirement, it is prudent for software to enable the write protect feature prior to enabling the OSPI controller. This will block any data writes from taking effect. To do so, the protection registers (OSPI_LOWER_WR_PROT_REG, OSPI_UPPER_WR_PROT_REG and OSPI_WR_PROT_CTRL_REG) should be setup and the number of bytes per device block in the device size configuration register should also be setup.

After Power-on Reset (POR), software can read from and write to the FLASH device (albeit slowly). Enabling/Disabling the controller and DAC is achieved with just one write to corresponding fields of the OSPI_CONFIG_REG register. User shall take note to maintain the default values of the baud rate divisor and the default state of SEL_CLK_POL_FLD/ SEL_CLK_PHASE_FLD bits of this register. A write data value of 0x00780081 is recommended.

12.3.2.4.2 Configuring the OSPI Controller for Optimal Use

Note

When using the OSPI Controller, the opcodes in OSPI_DEV_INSTR_RD_CONFIG_REG[7-0] RD_OPCODE_NON_XIP_FLD, OSPI_DEV_INSTR_WR_CONFIG_REG[7-0] WR_OPCODE_FLD and OSPI_WRITE_COMPLETION_CTRL_REG[7-0] OPCODE_FLD bit fields shall not match the opcode in the OSPI_FLASH_CMD_CTRL_REG[31-24] CMD_OPCODE_FLD bit field.

For high speed transfers PHY mode can be enabled and for optimal configuration PHY Pipeline mode is recommended. For more information, see Section 12.3.2.3.16.1, *PHY Pipeline Mode*.

To access the flash optimally, software must configure the controller accurately:

1. Wait until any pending STIG or INDAC operation has completed or poll OSPI_CONFIG_REG[31] IDLE_FLD bit.
2. Disable the DAC through OSPI_CONFIG_REG[7] ENB_DIR_ACC_CTLR_FLD bit. It is permitted, but not necessary to also disable the OSPI controller completely via OSPI_CONFIG_REG[0] ENB_SPI_FLD bit.
3. Update the OSPI_DEV_INSTR_RD_CONFIG_REG and OSPI_DEV_INSTR_WR_CONFIG_REG registers for the instruction type you wish to use for indirect and direct writes and reads.
4. Update the OSPI_MODE_BIT_CONFIG_REG[7-0] MODE_FLD bit field if mode bits have been enabled in the OSPI_DEV_INSTR_RD_CONFIG_REG[20] MODE_BIT_ENABLE_FLD bit.
5. Update the OSPI_DEV_SIZE_CONFIG_REG if the contents are incorrect. Note parts or all of this register may have been updated after initialization. The number of address bytes is a key configuration setting required for performing reads and writes. The number of bytes per page is required for performing any write. The number of bytes per device block is only required if the write protect feature is used. If the default values are correct for the target device, or if some of the values (not including the number address bytes) were incorrect but device writes were not permitted.
6. Update the OSPI_DEV_DELAY_REG. This register allows the user to tweak how the chip select is driven after each FLASH access. This is required as each device may have different timing requirements. As the serial clock frequency is increased, these timing requirements become more important. Note the numbers programmed in this register are based on the period of reference clock. Example: A device

Copyright © 2024 Texas Instruments Incorporated

needs 50ns minimum time before CS can be re-asserted after it has been de-asserted. By default, the controller will only provide a minimum of 1 SCLK period. When the device is operating at 100 MHz, the SCLK period is only 10ns, so 40ns extra is required. Since the register defines the number of reference clock cycles to add, and reference clock is running at 400 MHz (2.5ns period), then the user should program a value of at least 16 to the OSPI_DEV_DELAY_REG[31-24] D_NSS_FLD. This delay can be extended during auto-polling phase. There is possibility to define the polling repetition delay in the OSPI_WRITE_COMPLETION_CTRL_REG[31-24] POLL_REP_DELAY_FLD bit field.

7. Update the OSPI_REMAP_ADDR_REG register, if required. Affects DAC path only.
8. Setup and enable write protection registers (OSPI_LOWER_WR_PROT_REG, OSPI_UPPER_WR_PROT_REG and OSPI_WR_PROT_CTRL_REG) if they are required and if they have not already been setup from post initialization.
9. Enable required interrupts via the OSPI_IRQ_MASK_REG register.
10. Setup the baud rate divisor in the OSPI_CONFIG_REG[22-19] MSTR_BAUD_DIV_FLD to define the required clock frequency of the target device.
11. Update the OSPI_RD_DATA_CAPTURE_REG register. This register will delay when the read data is captured and can help when the read data path from the device to the controller is long and the device clock frequency is high. An update to this register may not be necessary.
12. Enable the OSPI controller and the DAC via the OSPI_CONFIG_REG.

12.3.2.4.3 Using the Flash Command Control Register (STIG Operation)

The OSPI_FLASH_CMD_CTRL_REG register provides software means to access the FLASH device in a flexible and programmable manner. This is known as a STIG operation (Software Triggered Instruction Generator). The instruction opcode, number of address bytes (if any), the address itself, number of dummy cycles (if any), number of write data bytes (if any), the write data itself and the number of read data bytes (if any) can be programmed. Once these have been programmed, software can trigger the command via OSPI_FLASH_CMD_CTRL_REG[0] CMD_EXEC_FLD bit and wait for its acceptance by polling OSPI_FLASH_CMD_CTRL_REG[1] CMD_EXEC_STATUS_FLD bit. When CMD_EXEC_STATUS_FLD bit turns de-asserted, another STIG can be triggered. This method of accessing the FLASH is the typical mechanism that software would use to access the FLASH device's registers, as well as for performing ERASE operations. It can also be used to access the FLASH array itself, although the maximum of 8 data bytes may be read or written at any one time, defined in the Flash Command Write and Read Data registers (OSPI_FLASH_RD_DATA_LOWER_REG, OSPI_FLASH_RD_DATA_UPPER_REG, OSPI_FLASH_WR_DATA_LOWER_REG and OSPI_FLASH_WR_DATA_UPPER_REG). This number of bytes can be extended for Read Data commands using additional STIG Memory Bank controlled by OSPI_FLASH_CMD_CTRL_REG[2] STIG_MEM_BANK_EN_FLD and OSPI_FLASH_COMMAND_CTRL_MEM_REG.

Commands issued using this interface have a higher priority than all other READ accesses coming from data interface, and will therefore interrupt any READ commands being requested by the indirect or direct controllers.

12.3.2.4.4 Using SPI Legacy Mode

SPI legacy mode allows software to access the internal TX-FIFO and RX-FIFO directly, thus bypassing the direct, indirect and STIG controllers.

Legacy mode allows the user to issue any FLASH instruction to the device, but does place a heavy software overhead in order to manage the fill levels of the FIFO's effectively. This is because the legacy SPI core is bi-directional in nature, with data continuously being transferred in either direction while the chip select is enabled. Even if the driver only wishes to read data from the FLASH device, dummy data must be written out to ensure the chip select stays active, and vice versa for write transactions.

Since the TX-FIFO and RX-FIFO are of limited depth, software has a responsibility to maintain the FIFO levels to ensure the TX-FIFO does not become exhausted during the instruction execution and the RX-FIFO doesn't overflow. This can place a lot of overhead on software. Interrupts are provided to indicate when the fill levels pass programmable watermarks, which are themselves programmable registers OSPI_TX_THRESH_REG and OSPI_RX_THRESH_REG.

The limited depth may impose the limitation over execution of some specific SPI commands in legacy mode. Note that the controller interprets all transmitted bytes as valid. For example, if the Flash Device was configured

Copyright © 2024 Texas Instruments Incorporated

to return valid data after many dummy cycles, the TX FIFO could become full before the controller sends all of dummy data.

12.3.2.4.5 Entering XIP Mode from POR

XIP is a mode that can be entered in a non-volatile way if the device has XIP enabled as a non-volatile configuration setting. Software will not be able to discover the state of XIP from POR via FLASH status register reads as the only operation a FLASH device will recognize when XIP mode is enabled is an XIP read operation.

If it is already known that the device will enter XIP from POR, then the OSPI_MODE_BIT_CONFIG_REG[7-0] MODE_FLD and OSPI_CONFIG_REG[18] ENTER_XIP_MODE_IMM_FLD bits should be set in initial boot.

If it is not already known that the device will enter XIP from POR, and XIP from POR may be supported by the attached FLASH device, then software can attempt to exit XIP mode by issuing an XIP exit command using a STIG command (via the OSPI_FLASH_CMD_CTRL_REG register). To do this, software must be aware of the mode bit requirements of that device, as XIP entry and exit changes per device.

12.3.2.4.6 Entering XIP Mode Otherwise

XIP mode is supported in most FLASH devices. Some of them use signature bits that are sent to the device immediately following the address bytes, other use signature bits and also require a FLASH device configuration register write to enable XIP. For the FLASH devices that must be compliant to the OSPI controller, the following steps can be taken by software to enter XIP mode:

1. Disable the DAC and INDAC (OSPI_CONFIG_REG[7] ENB_DIR_ACC_CTLR_FLD) to ensure no new data read accesses will be sent to the FLASH device.
2. (Optional) Configure the OSPI_FLASH_CMD_CTRL_REG to issue a VCR write to FLASH memory, because XIP mode must first be enabled for some devices.
3. Configure the XIP mode bits in the OSPI_MODE_BIT_CONFIG_REG[7-0] MODE_FLD bit field.
4. Enable the local controllers XIP mode by setting OSPI_CONFIG_REG[17] ENTER_XIP_MODE_FLD bit.
5. Re-enable the DAC and, if required, the INDAC.

12.3.2.4.7 Exiting XIP Mode

To exit XIP mode, software should first disable the DAC and INDAC to ensure no new data read accesses will be sent to the FLASH device. It should then set mode bits to other than the established in the corresponding Flash Device specifications. These are dependent on the FLASH device and manufacturer. Software should then reset OSPI_CONFIG_REG[17] ENTER_XIP_MODE_FLD.

Note the FLASH device must see a READ instruction before it can disable its internal XIP mode state, so this means XIP mode will internally stay active until the next READ instruction is serviced. User must take care to ensure that XIP mode is disabled before the end of any READ sequence.

Copyright © 2024 Texas Instruments Incorporated

12.3.3 HyperBus Interface

This section describes the HyperBus module in the device.

12.3.3.1 HyperBus Overview

The HyperBus module is a part of the device Flash Subsystem (FSS). For more information about FSS, see *Flash Subsystem (FSS)*.

The HyperBus module is a low pin count memory interface that provides high read/write performance. The HyperBus module connects to HyperBus memory (HyperFlash or HyperRAM) and uses simple HyperBus protocol for read and write transactions.

There is one HyperBus™ module inside the device. The HyperBus module includes one HyperBus Memory Controller (HBMC).

12.3.3.1.1 HyperBus Features

- Support for Cypress® HyperFlash and HyperRAM
- Up to 166 MHz maximum memory bus operation for reads
  - Supports up to 166 MHz dual data rate (333 MBps) flash devices for system requiring rapid boot or instant-on displays
  - Supports up to 333 MBps external pseudo-RAM (HyperRAM) for systems

  Note: For more information, see section Timing and Switching Characteristics in the device-specific Datasheet.
- Low pin count interface with LVCMOS I/O pins (can be muxed with other FSS interfaces (OSPIs))
- Two memory chip selects
- Linear incrementing mode for reads and writes
- Up to 16 outstanding read transactions (see Section 12.3.3.3.5, *HyperBus True Continuous Read (TCR) Mode*)
- Asynchronous bus clock

12.3.3.1.2 Hyperbus Ports

Table 12-153. MCU_FSS0_HPB0 Clocks and Resets

Clocks	
Module Clock Input	**Description**
MCU_FSS0_HPB0_ICLK	Interface Clock
MCU_FSS0_HPB0_CLKX1	Functional Clock 1 (main functional clock)
MCU_FSS0_HPB0_CLKX1_INV	Functional Clock 2 (inverted Functional Clock 1 to create 180 degree phase shift)
MCU_FSS0_HPB0_CLKX2	Functional Clock 3 (2 × Functional Clock 1 for DDR data and command)
MCU_FSS0_HPB0_CLKX2_INV	Functional Clock 4 (inverted Functional Clock 3 to create 180 degree phase shift)
Resets	
Module Reset Input	**Description**
MCU_FSS0_HPB0_RST	HyperBus Reset

Table 12-154. MCU_FSS0_HPB0 Hardware Requests

Interrupt Requests		
Module Interrupt Signal	**Description**	**Type**
MCU_HYPFLSH_INT	HyperBus Interrupt Request	Level
MCU_HYPFLSH_SEC_INT	HyperBus ECC SEC Error Interrupt Request	Level
MCU_HYPFLSH_DED_INT	HyperBus ECC DED Error Interrupt Request	Level

12.3.3.2 HyperBus Environment

This section describes the HyperBus external connections (environment).

Table 12-155 describes the HyperBus I/O signals.

Copyright © 2024 Texas Instruments Incorporated

Table 12-155. HyperBus I/O Signals

Module Pin	I/O[1]	Description
CK	O	HyperBus Clock Output (differential)
CK**n**	O	HyperBus Inverted Clock Output (differential)
RWDS	I/O/HiZ	HyperBus Read/Write Data Strobe
DQ[7:0]	I/O/HiZ	HyperBus Data
CS**n**0	O	HyperBus Chip Select 0
CS**n**1	O	HyperBus Chip Select 1
RESET**n**	O	HyperBus Reset Output
RESETO**n**	I	Reset State Indicator (output from HyperBus memory)
INT**n**	I	Memory Interrupt (output from HyperBus memory)
WP**n** [2]	O	HyperBus Write Protect (output to HyperBus memory)

(1) I = Input; O = Output; HiZ = High Impedance
(2) WP**n** pin is not used on Cypress flash devices.

Copyright © 2024 Texas Instruments Incorporated

12.3.3.3 HyperBus Functional Description

The HyperBus module is a high throughput memory interface. It is a part of the device FSS and provides execute in place (XIP) operation and block copy access to HyperBus memory devices (HyperRAM and HyperFlash). The FSS enables in-line ECC protection and authentication features for the HyperBus module.

For more information about FSS, see *Flash Subsystem (FSS)*.

The HyperBus module is compliant to the *Cypress HyperBus™ Specification v1.2*.

Figure 12-75 shows the HyperBus module block diagram.

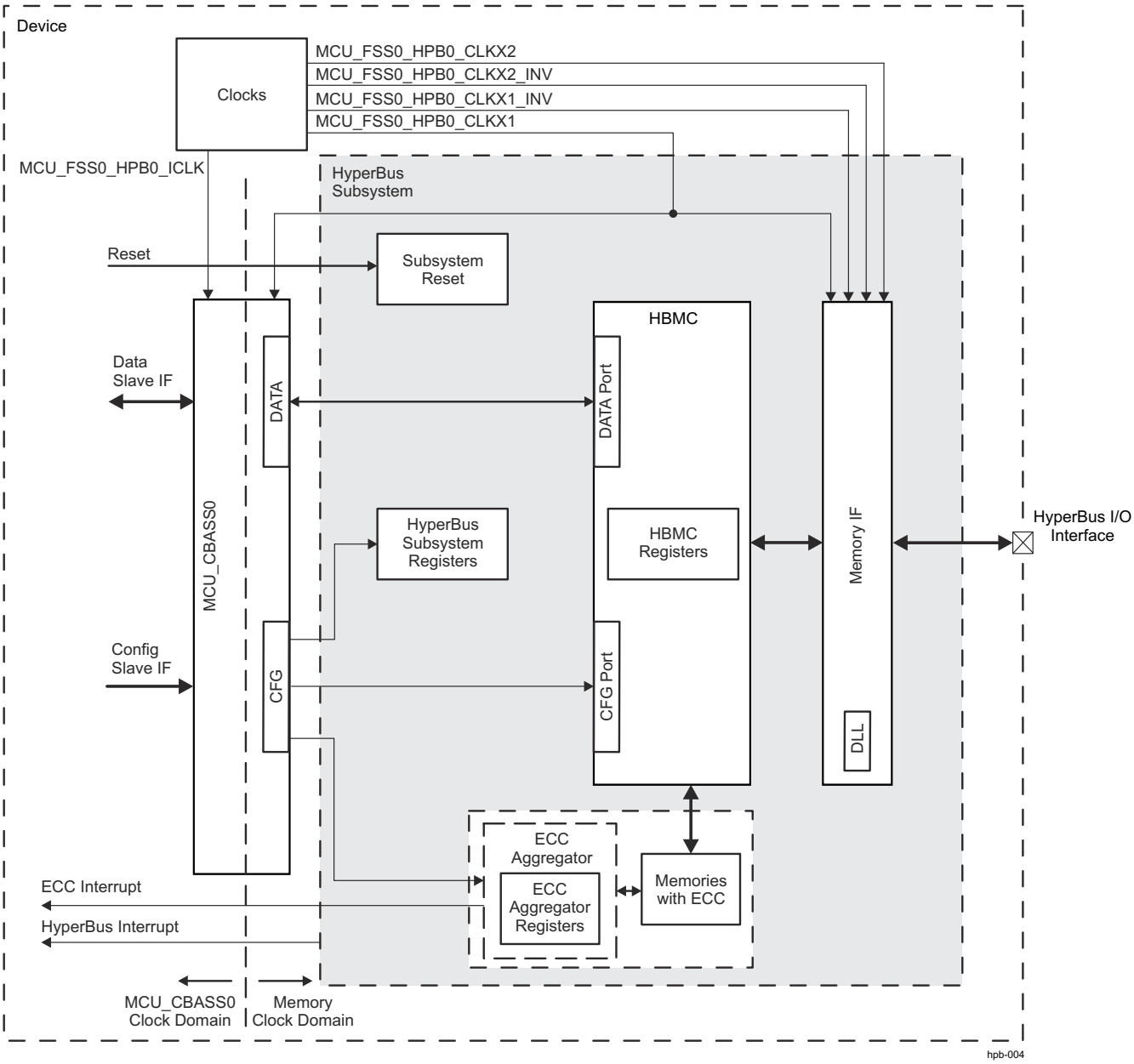


Figure 12-75. HyperBus Module Block Diagram

Basic Blocks:

- HBMC: The HyperBus Memory Controller (HBMC) is the main part (the core) of the HyperBus Subsystem and provides accessibility to external HyperBus memory (HyperFlash or HyperRAM) using simple HyperBus protocol for data read and write transfers.

Copyright © 2024 Texas Instruments Incorporated

  - The Data Port on the HBMC is used for CBASS data slave interface to access the external HyperBus memory.
  - The CFG Port on the HBMC is used for configuration of the HBMC registers.
- HyperBus I/O Interface: The HyperBus I/O Interface includes all used interface pins (for more information, see *HyperBus I/O Signals*).
- Memory IF: The Memory IF block implements the following functionality:
  - Convert single-rate command and data output from the HBMC to double-rate (DDR)
  - Use DLL to delay the incoming read data strobe from the memory
- FIFO Memories with ECC: There are internal HyperBus FIFOs implemented with memories that are used during read and write transactions. These FIFO memories are ECC protected (for more information about the FIFOs, see Section 12.3.3.3.3, *HyperBus Internal FIFOs*).
- ECC Aggregator: The ECC Aggregator facilitates aggregating and reporting internal HyperBus FIFO memory errors (for more information, see Section 12.3.3.3.2, *HyperBus ECC Support*).
- HBMC Registers: This block includes set of all used HBMC registers.
- HyperBus Subsystem Registers: This block implements memory-mapped registers at the HyperBus Subsystem level.
- ECC Aggregator Registers: This block includes all used ECC Aggregator registers.

  For more information about all HyperBus registers, see *HyperBus Registers*.
- MCU_CBASS0: The MCU_CBASS0 interconnect implements the clock domain crossing bridges needed to separate the two main clock domains: MCU_CBASS0 clock domain and Memory clock domain.
- Data Slave IF: The Data Slave IF on the interconnect is a 32-bit wide interface and is used to access an external HyperBus memory via the Data Port on the HBMC. The data interface supports linear incrementing addresses only. Cache-line wrap and fixed addressing modes are not supported.
- Config Slave IF: The Config Slave IF on the interconnect is a 32-bit wide interface and is used to access the HBMC registers (via the CFG Port on the HBMC), HyperBus Subsystem registers, and ECC Aggregator registers. The config interface also supports linear incrementing address mode only.

  For more information about supported and not supported HyperBus features, see *HyperBus Overview*.
- Clocks: For more information about HyperBus Clocks, see *MCU_FSS0_HBP0 Clocks and Resets*.
- Subsystem Reset: The reset signal to the Subsystem Reset block provides reset to the all HyperBus Subsystem parts (for more information about HyperBus Resets, see *MCU_FSS0_HBP0 Clocks and Resets*).
- Interrupts: For more information, see *MCU_FSS0_HBP0 Clocks and Resets* and *HyperBus Interrupts*.

12.3.3.3.1 HyperBus Interrupts

The HyperBus module sources one active high level HyperBus interrupt and two active high level ECC Aggregator interrupts (see *MCU_FSS0_HPB0 Hardware Requests*).

The HyperBus interrupt is generated based on the HyperBus memory's active low level interrupt. In the HyperBus Subsystem this active low level interrupt is converted to active high level HyperBus interrupt (see RESETO**n** pin in *Hyperbus I/O Signals*).

The ECC Aggregator interrupts are generated based on the ECC errors (single (correctable) and double (uncorrectable) bit errors) in the embedded HyperBus memories (for more information, see *HyperBus ECC Support*).

12.3.3.3.2 HyperBus ECC Support

The Error Correcting Code (ECC) is a mechanism for providing increased system reliability via reduction of memory software errors by allowing single bit errors to be detected and corrected (SEC) and double bit errors to be detected (DED).

The SEC logic detects and corrects a single bit error (single bit error per ECC word or per ECC data segment). The DED logic only detects (does not correct) double errors (double bit errors per ECC word or per ECC data segment).

The embedded HyperBus memories are ECC protected.

Copyright © 2024 Texas Instruments Incorporated

12.3.3.3.2.1 ECC Aggregator

Note

For more information about ECC Aggregator, refer to Section 12.10.4, *ECC Aggregator*.

For more information about ECC Aggregator Registers, refer to *HyperBus Registers*.

12.3.3.3.3 HyperBus Internal FIFOs

There are 10 FIFO buffers in the module that help improve performance. Table 12-156 shows the HyperBus FIFOs details.

Table 12-156. HyperBus FIFOs

Buffer Name	Size (Width x Depth)
Address (ADR) FIFO	Two port RAM – 16 x 46-bit
Write data (WDAT) FIFO	Two port RAM – 256 x 40-bit
Write status (BDAT) FIFO	Two port RAM – 16 x 2-bit
Read data (RDAT) FIFO	Two port RAM – 128 x 40-bit
Receive (RX) FIFO	Two port RAM – 256 x 20-bit
Address write (AW) FIFO	Two port RAM – 16 x 44-bit
Write ID (WID) FIFO	Two port RAM – 16 x 2-bit
Address write ID (AWID) FIFO	Two port RAM – 16 x 14-bit
Address read (AR) FIFO	Two port RAM – 16 x 44-bit
Address read ID (ARID) FIFO	Two port RAM – 16 x 18-bit

12.3.3.3.4 HyperBus Data Regions

For more information about HyperBus data regions, refer to Section 12.3.1.3.4, *FSS Memory Regions*.

12.3.3.3.5 HyperBus True Continuous Read (TCR) Mode

The HBMC supports True Continuous Read (TCR) operation for HyperFlash. This feature is able to improve HyperFlash read performance.

When this feature is enabled, the HBMC can merge multiple up to 16 max burst read commands that are present in its command FIFO into a single memory read transaction to the HyperFlash memory. As a result, the command input and initial latency for the latter burst read commands are saved, which leads to improved throughput.

Copyright © 2024 Texas Instruments Incorporated

12.3.3.4 HyperBus Programming Guide

12.3.3.4.1 HyperBus Initialization Sequence

The HBMC provides two sets of Memory Base Address, Memory Configuration and Memory Timing registers to access devices connected to the two external chip selects. The following programming guidelines need to use the correct set of registers depending on which chip select is accessed.

The HBMC compares the input address bits [31-24] to the value programmed in the MCU_FSS0_HPB0_MC_MBAR_y registers to determine the external chip select that is accessed.

12.3.3.4.1.1 HyperFlash Access

Table 12-157. HyperFlash Access Sequence

Step	Description
1.	Ensure that the FIFO RAM auto-initialization is complete by reading the MCU_FSS0_HPB0_SS_RAM_STAT_REG[0] INIT_DONE bit .
2.	Lock MDLL:
	- To ensure that the MDLL in the HyperBus is locked, the reset to the HyperBus module should be de-asserted only after all the clock inputs are stable at the desired operating frequency (see *MCU_FSS0_HPB0 Clocks and Resets*). The MDLL can lose the lock if the frequencies of the clock inputs to the HyperBus module are changed during operation.
	- To ensure that the MDLL is stabilized, the following sequence is required. Boot code should attempt to read 64 bytes of Flash data, for 16 iterations, and if the data is the same in 4 successive iterations, the DLL can be considered to be stabilized and the software proceed with normal Flash access .
	- The MCU_FSS0_HPB0_SS_DLL_STAT_REG[0] MDLL_LOCK and MCU_FSS0_HPB0_SS_DLL_STAT_REG[1] SDL_LOCK bits can be used to determine if the master delay line and slave delay line have locked, respectively .
3.	Initialize Memory Configuration register (MCU_FSS0_HPB0_MC_MCR_y):
	- MCU_FSS0_HPB0_MC_MCR_y[31] MAXEN bit and MCU_FSS0_HPB0_MC_MCR_y[26-18] MAXLEN bit field based on burst transaction length to memory.
	- MCU_FSS0_HPB0_MC_MCR_y[17] TCMO = 1h, to enable HBMC to merge multiple accesses with sequential addresses into a single memory access. This will improve memory throughput.
	- MCU_FSS0_HPB0_MC_MCR_y[16] ACS = 0h, to set 'No asymmetry cache system support'.
	- MCU_FSS0_HPB0_MC_MCR_y[5] CRT = 0h, to set 'Memory space' instead 'CR space'.
	- MCU_FSS0_HPB0_MC_MCR_y[4] DEVTYPE = 0h, to set 'HyperFlash' instead 'HyperRAM'.
	- MCU_FSS0_HPB0_MC_MCR_y[1-0] WRAPSIZE = 00h, since the HBMC does not support wrap bursts.
4.	Initialize Memory Timing register (MCU_FSS0_HPB0_MC_MTR_y) based on timing of the memory device being used.
5.	Initialize Memory Base Address register (MCU_FSS0_HPB0_MC_MBAR_y):
	- MCU_FSS0_HPB0_MC_MBAR_y[31-24] A_MSB = 8 MSB bits of memory address space. This will define the start of the 16 MB address region in the system memory where the HyperFlash can be accesed. The HBMC will initiate HyperFlash access to any memory mapped access in this range.
6.	Check the MCU_FSS0_HPB0_MC_CSR[10] RRSTOERR bit in Controller Status register to ensure that the HyperFlash is out of reset.
7.	Normal HyperFlash access can be performed after this.
8.	The HyperFlash device registers and CFI (Common Flash Interface) region can be accessed by read/write transactions to the offset from the base address as specified in the device command summary table.
9.	The HyperFlash memory data array can be read as memory mapped access.
10.	The HyperFlash write sequence needs to be followed to update contents of the memory array.

Copyright © 2024 Texas Instruments Incorporated

12.3.3.4.1.2 HyperRAM Access

Table 12-158. HyperRAM Access Sequence

Step	Description
1.	Wait 150 µs for RAM to power up after reset. The HyperRAM does not provide any feedback on reset/ready state to the HBMC .
2.	Ensure that the FIFO RAM auto-initialization is complete by reading the MCU_FSS0_HPB0_SS_RAM_STAT_REG[0] INIT_DONE bit .
3.	Lock MDLL:
	- To ensure that the MDLL in the HyperBus is locked, the reset to the HyperBus module should be de-asserted only after all the clock inputs are stable at the desired operating frequency (see *MCU_FSS0_HPB0 Clocks and Resets*). The MDLL can lose the lock if the frequencies of the clock inputs to the HyperBus module are changed during operation.
	- To ensure that the MDLL is stabilized, the following sequence is required. Boot code should attempt to read 64 bytes of RAM data, for 16 iterations, and if the data is the same in 4 successive iterations, the DLL can be considered to be stabilized and the software proceed with normal RAM access .
	- The MCU_FSS0_HPB0_SS_DLL_STAT_REG[0] MDLL_LOCK and MCU_FSS0_HPB0_SS_DLL_STAT_REG[1] SDL_LOCK bits can be used to determine if the master delay line and slave delay line have locked, respectively .
4.	Initialize Memory Configuration register (MCU_FSS0_HPB0_MC_MCR_y):
	- MCU_FSS0_HPB0_MC_MCR_y[31] MAXEN bit and MCU_FSS0_HPB0_MC_MCR_y[26-18] MAXLEN bit field based on burst transaction length to memory.
	- MCU_FSS0_HPB0_MC_MCR_y[17] TCMO = 0h, since HBMC does not support command merging for HyperRAM accesses.
	- MCU_FSS0_HPB0_MC_MCR_y[16] ACS = 0h, to set 'No asymmetry cache system support'.
	- MCU_FSS0_HPB0_MC_MCR_y[5] CRT = 0h or 1h, depending on what needs to be accessed (memory or register space).
	- MCU_FSS0_HPB0_MC_MCR_y[4] DEVTYPE = 1h, to set 'HyperRAM' instead 'HyperFlash'.
	- MCU_FSS0_HPB0_MC_MCR_y[1-0] WRAPSIZE = 00h, since the HBMC does not support wrap bursts.
5.	Initialize MCU_FSS0_HPB0_MC_MTR_y register based on timing of the memory device being used.
6.	Initialize MCU_FSS0_HPB0_MC_MCR_y register:
	- MCU_FSS0_HPB0_MC_MCR_y[31-24] A_MSB = 8 MSB bits of memory address space. This will define the start of the 16 MB address region in the system memory where the HyperRAM can be accesed. The controller will initiate HyperRAM access to any memory mapped access in this range.
7.	Check the MCU_FSS0_HPB0_SS_DLL_STAT_REG[0] MDLL_LOCK bit to ensure the Master DLL is locked.
8.	Normal HyperRAM access can be performed after this.
9.	The HyperRAM device registers can be accessed by read/write transactions to the offset from the base address as specified in the device command summary table with MCU_FSS0_HPB0_MC_MCR_y[5] CRT bit set to register space.
10.	The HyperRAM memory data array can be read/written to as memory mapped access with MCU_FSS0_HPB0_MC_MCR_y[5] CRT bit set to memory space.

Note

HyperRAM devices have a 150 µs startup time. Software needs to wait this duration after reset to initiate transactions to a HyperRAM device.

12.3.3.4.2 HyperBus Real-time Operating Requirements

The HBMC may assert the memory interrupt based on the status of the memory transaction. Software needs to handle this interrupt using the MCU_FSS0_HPB0_MC_ISR register. The MCU_FSS0_HPB0_MC_ISR register can provide the status information to determine the cause of the interrupt.

12.3.3.4.3 HyperBus Power Up/Down Sequence

Software needs to ensure that there are no pending transactions before stopping clock and/or power to the HBMC.

Copyright © 2024 Texas Instruments Incorporated

12.3.4 General-Purpose Memory Controller (GPMC)

This section describes the General-Purpose Memory Controller (GPMC) for the device.

12.3.4.1 GPMC Overview

The General-Purpose Memory Controller is a unified memory controller dedicated for interfacing with external memory devices like:

- Asynchronous SRAM-like memories and application-specific integrated circuit (ASIC) devices
- Asynchronous, synchronous, and page mode (available only in non-multiplexed mode) burst NOR flash devices
- NAND flash
- Pseudo-SRAM devices

12.3.4.1.1 GPMC Features

The main features of the GPMC are:

- 8- or 16-bit-wide data path to external memory device
- Supports up to 4 chip select regions of programmable size and programmable base addresses in a total address space of 1GB
- Supports on-the-fly error code detection using the Bose-ChaudhurI-Hocquenghem (BCH) (t = 4, 8, or 16) or Hamming code to improve the reliability of NAND with a minimum effect on software (NAND flash with 512-byte page size or greater)
- Fully pipelined operation for optimal memory bandwidth usage
- The clock to the external memory is provided from GPMC_FCLK divided by 1, 2, 3, or 4
- Supports programmable autoclock gating when no access is detected
- Independent and programmable control signal timing parameters for setup and hold time on a per-chip basis. Parameters are set according to the memory device timing parameters with a timing granularity of one GPMC_FCLK clock cycle.
- Flexible internal access time control (wait state) and flexible handshake mode using external WAIT pin monitoring
- Support bus keeping
- Support bus turnaround
- Prefetch and write-posting engine associated with DMA controller at system level to achieve full performance from the NAND device with minimum effect on NOR/SRAM concurrent access
- 32-bit interconnect slave interface which supports non-wrapping and wrapping burst of up to 16x32 bits.

The GPMC supports the following various access types:

- Asynchronous read/write access
- Asynchronous read page access (4, 8, and 16 Word16)
- Synchronous read/write access
- Synchronous read/write burst access without wrap capability (4, 8 and 16 Word16)
- Synchronous read/write burst access with wrap capability (4, 8 and 16 Word16)
- Address-data-multiplexed (AD) access
- Address-address-data (AAD) multiplexed access
- Little-endian access only

The GPMC can communicate with a wide range of external devices:

- External asynchronous or synchronous 8-bit wide memory or device (non burst device)
- External asynchronous or synchronous 16-bit wide memory or device
- External 16-bit non-multiplexed NOR flash device
- External 16-bit address and data multiplexed NOR Flash device
- External 8-bit and 16-bit NAND flash device
- External 16-bit pseudo-SRAM (pSRAM) device

Note

Page mode is available only in non-multiplexed mode.

Copyright © 2024 Texas Instruments Incorporated

Unsupported Features:

- See the Module Integration section of this document for a list of module features not supported by the integration on this Device.

12.3.4.1.2 GPMC Ports

Table 12-159. GPMC0 Clocks and Resets

Clocks	
Module Clock Input	**Description**
GPMC_FCLK	GPMC Functional Clock. For more information about clock multiplexing, see CTRLMMR_GPMC_CLKSEL[1-0] CLK_SEL in *Control Module (CTRL_MMR)*.
GPMC_ICLK	GPMC Interface Clock
Resets	
Module Reset Input	**Description**
GPMC_RST	GPMC Asynchronous Reset

Table 12-160. GPMC0 Hardware Requests

Interrupt Requests		
Module Interrupt Signal	**Description**	**Type**
GPMC_GPMC_SINTERRUPT_0	GPMC Interrupt Request	Level

12.3.4.2 GPMC Environment

This section describes the GPMC0 external connections (environment).

Table 12-161 lists the GPMC subsystem input/output (I/O) pins.

Table 12-161. GPMC I/O Signals (Master Mode)

Module Pin	I/O[(1)]	Description
A[27-0]	O	28-bit output address bus
A[16-1]/D[15-0]	I/O	Multiplexed address/data
nCS[3-0]	O	Chip-selects (active low)
CLK	O	Clock generated for the external memory or device. For more information, see *GPMC Integration*.
RET_CLK		
N/A	O	Free running clock. GPMC functional clock (GPMC0_FCLK) propagated on a device pad. For more information on the GPMC0_FCLK_MUX integration, see *GPMC Integration*.
nADV/ALE	O	Address valid (active low). Also used as address latch enable (active high) for NAND protocol memories.
nOE/nRE	O	Output enable (active low). Also used as read enable (active low) for NAND protocol memories.
nWE	O	Write enable (active low)
nBE0/CLE	O	Lower-byte enable (active low). Also used as command latch enable for NAND protocol memories.
nBE1	O	Upper-byte enable (active low)
WAIT[3-0]	I	External wait signal for NOR and NAND protocol memories. Can be mapped on any of the chip-selects.
nWP	O	Write protect (active low)
DIR	O	This signal can be used to control an external buffer direction. Also controls the signal direction of D[15-0]. Low during transmit (for write access: data OUT from GPMC0 to memory). High during receive (for read access: data IN from memory to GPMC0).

(1) I = Input; O = Output; I/O - Bidirectional

Copyright © 2024 Texas Instruments Incorporated

12.3.4.3 GPMC Functional Description

The GPMC basic programming model offers maximum flexibility to support various access protocols for each of the four configurable chip-selects. Use optimal chip-select settings, based on the characteristics of the external device:

- Different protocols can be selected to support generic asynchronous or synchronous random-access devices (NOR flash, SRAM) or to support specific NAND devices.
- The address and data bus can be multiplexed on the same external bus.
- Read and write access can be independently defined as asynchronous or synchronous.
- System requests (byte, 16-bit word, burst) are performed through single or multiple accesses. External access profiles (single, multiple with optimized burst length, native- or emulated-wrap) are based on external device characteristics (supported protocol, bus width, data buffer size, native-wrap support).
- System burst read or write requests are synchronous-burst (multiple-read or multiple-write). When neither burst nor page mode is supported by external memory or ASIC devices, system burst read or write requests are translated to successive single synchronous or asynchronous accesses (single reads or single writes). 8-bit wide devices are supported only in single synchronous or single asynchronous read or write mode.
- To simulate a programmable internal-wait-state, an external WAIT pin can be monitored to dynamically control external access at the beginning (initial access time) of and during a burst access.

Each control signal is controlled independently for each chip-select. The internal functional clock of the GPMC (GPMC_FCLK) is used as a time reference to specify the following:

- Read- and write-access duration
- Most GPMC external interface control-signal assertion and deassertion times
- Data-capture time during read access
- External wait-pin monitoring time
- Duration of idle time between accesses, when required

12.3.4.3.1 GPMC Block Diagram

Figure 12-76 shows the GPMC functional block diagram. The GPMC consists of six blocks:

- Interconnect port interface
- Address decoder, GPMC configuration, and chip-select configuration register file
- Access engine
- Prefetch and write-posting engine
- Error correction code engine (ECC)
- External device/memory port interface

Copyright © 2024 Texas Instruments Incorporated

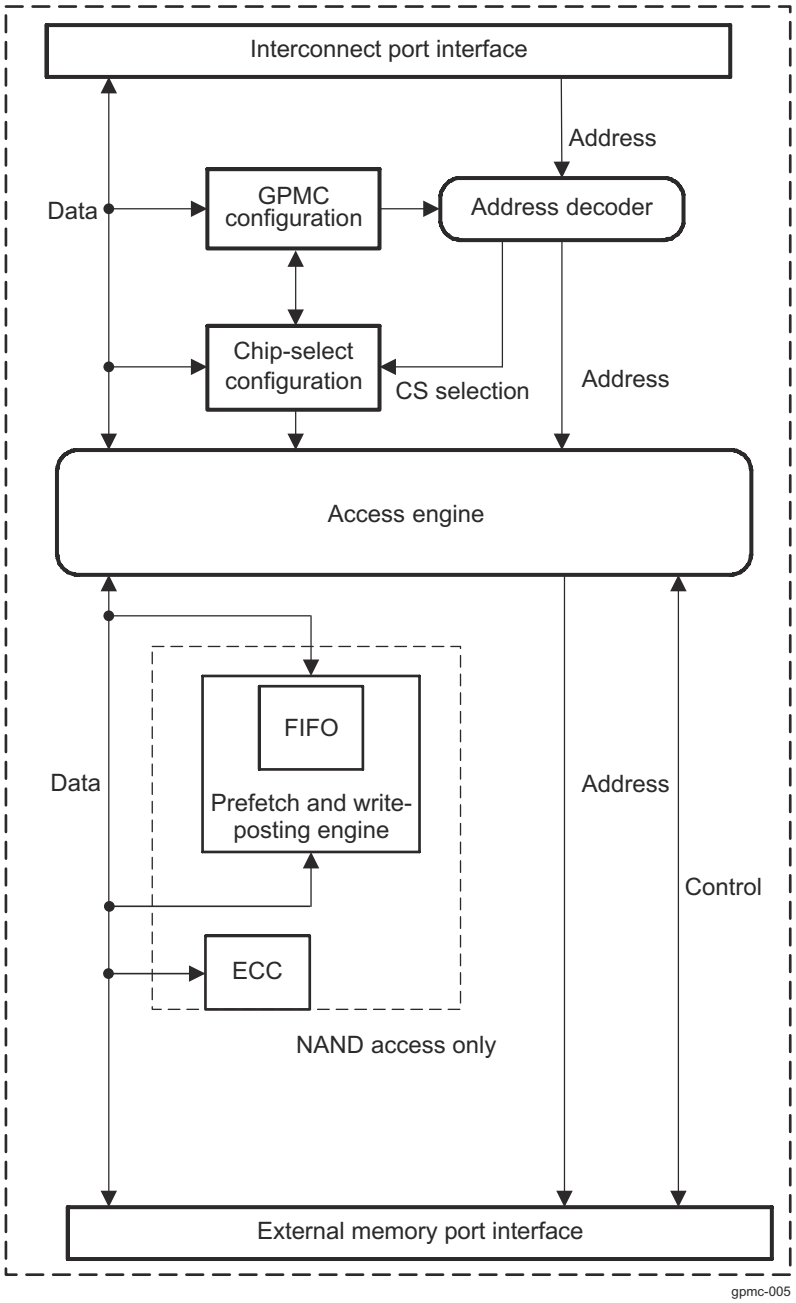


Figure 12-76. GPMC Block Diagram

The GPMC can access various external devices. The flexible programming model allows a wide range of attached device types and access schemes.

Based on the programmed configuration bit fields stored in the GPMC registers, the GPMC can generate the timing of all control signals depending on the attached device and access type.

Given the chip-select decoding and its associated configuration registers, the GPMC selects the appropriate control signal timing for the device type.

12.3.4.3.2 GPMC Clock Configuration

Table 12-162 describes the GPMC clocks.

Table 12-162. GPMC Clocks

Signal	I/O[(1)]	Description
GPMC_FCLK	I	Functional clock

Copyright © 2024 Texas Instruments Incorporated

Table 12-162. GPMC Clocks (continued)

Signal	I/O[(1)]	Description
GPMC_ICLK	I	Interface clock
CLK (GPMC_CLKOUT pin)	O	External clock provided to synchronous external memory devices.

(1) I = Input; O = Output; I/O - Bidirectional

The GPMC output clock (CLK) is generated by the GPMC from the internal GPMC_FCLK clock. The source of the GPMC_FCLK is described in *GPMC0 Clocks and Resets*. The GPMC output clock is configured using the GPMC_CONFIG1_i[1-0] GPMCFCLKDIVIDER bit field (where i = 0 to 3), as shown in Table 12-163.

Table 12-163. GPMC Output Clock Configuration

Source Clock	GPMC_CONFIG1_i[1-0] GPMCFCLKDIVIDER	GPMC Output Clock Provided to External Memory Device
GPMC_FCLK	00	GPMC_FCLK
	01	GPMC_FCLK/2
	10	GPMC_FCLK/3
	11	GPMC_FCLK/4

When using synchronous interface protocols, the GPMC output clock (CLK), toggles only during the read or write access cycle. In some applications, it may be desirable to have a continuous clock running at the GPMC interface clock frequency for clocking attached devices. This option is enabled by an optional clock path from the GPMC functional clock input (GPMC_FCLK) to the GPMC_FCLK_MUX pin. This output clock, to GPMC_FCLK_MUX pin, can be selected through the standard MUXMODE selection of the GPMC_FCLK_MUX pin PADCONFIG control register.

Note that when using such synchronous interface protocols with the continuous clock option, user should ensure that the GPMC outputs are timed to the same frequency (GPMC_CONFIG1_i[1-0] GPMCFCLKDIVIDER = 0).

12.3.4.3.3 GPMC Power Management

Table 12-164 describes the local power-management features available for the GPMC module.

Table 12-164. GPMC Local Power-Management Features

Feature	Registers	Description
Clock autogating	GPMC_SYSCONFIG[0] AUTOIDLE	This bit allows a local power optimization inside the module, by gating the GPMC_ICLK clock upon the internal activity.
Slave idle modes	GPMC_SYSCONFIG[4-3] SIDLEMODE	Force-idle, no-idle and smart-idle modes are available.

12.3.4.3.4 GPMC Interrupt Requests

The GPMC generates one interrupt request (see *GPMC0 Hardware Requests*).

Table 12-165 lists the event flags, and their mask, that can cause module interrupts.

Table 12-165. GPMC Interrupt Events

Event Flag	Event Mask	Sensitivity	Description
GPMC_IRQSTATUS[11] WAIT3EDGE DETECTIONSTATUS	GPMC_IRQENABLE[11] WAIT3EDGE DETECTIONENABLE	Edge	Wait3 edge detection interrupt: Triggered if a rising or falling edge is detected on the GPMC_WAIT3 signal. The rising or falling edge detection of Wait3 is selected through the GPMC_CONFIG[11] WAIT3PINPOLARITY bit.
GPMC_IRQSTATUS[10] WAIT2EDGE DETECTIONSTATUS	GPMC_IRQENABLE[10] WAIT2EDGE DETECTIONENABLE	Edge	Wait2 edge detection interrupt: Triggered if a rising or falling edge is detected on the GPMC_WAIT2 signal. The rising or falling edge detection of Wait2 is selected through the GPMC_CONFIG[10] WAIT2PINPOLARITY bit.

Copyright © 2024 Texas Instruments Incorporated

Table 12-165. GPMC Interrupt Events (continued)

Event Flag	Event Mask	Sensitivity	Description
GPMC_IRQSTATUS[9] WAIT1EDGE DETECTIONSTATUS	GPMC_IRQENABLE[9] WAIT1EDGE DETECTIONENABLE	Edge	Wait1 edge detection interrupt: Triggered if a rising or falling edge is detected on the GPMC_WAIT1 signal. The rising or falling edge detection of Wait1 is selected through the GPMC_CONFIG[9] WAIT1PINPOLARITY bit.
GPMC_IRQSTATUS[8] WAIT0EDGE DETECTIONSTATUS	GPMC_IRQENABLE[8] WAIT0EDGE DETECTIONENABLE	Edge	Wait0 edge detection interrupt: Triggered if a rising or falling edge is detected on the GPMC_WAIT0 signal. The rising or falling edge detection of Wait0 is selected through the GPMC_CONFIG[8] WAIT0PINPOLARITY bit.
GPMC_IRQSTATUS[1] TERMINAL COUNTSTATUS	GPMC_IRQENABLE[1] TERMINAL COUNTENABLE	Level	Terminal count event: Triggered on prefetch process completion; that is, when the number of currently remaining data to be requested reaches 0
GPMC_IRQSTATUS[0] FIFOEVENTSTATUS	GPMC_IRQENABLE[0] FIFOEVENTENABLE	Level	FIFO event interrupt: Indicates available FIFO levels for write-posting mode and prefetch mode. GPMC_PREFETCH_CONFIG1[2] DMAMODE must be set to 0.

12.3.4.3.5 GPMC Interconnect Port Interface

Note

Some of the GPMC features described in this section may not be supported on this family of devices. For more information, see *GPMC Not Supported Features*.

The GPMC interconnect interface is a pipelined interface including a 16 × 32-bit word write buffer.

Any system host can issue external access requests through the GPMC.

The device system can issue the following requests through this interface:

- One 8-, 16-, or 32-bit interconnect access (read/write)
- Two incrementing 32-bit interconnect accesses (read/write)
- Two wrapped 32-bit interconnect accesses (read/write)
- Four incrementing 32-bit interconnect accesses (read/write)
- Four wrapped 32-bit interconnect accesses (read/write)
- Eight incrementing 32-bit interconnect accesses (read/write)
- Eight wrapped 32-bit interconnect accesses (read/write)

Only linear burst transactions are supported; interleaved burst transactions are not supported. Only power-of-two-length precise bursts 2 × 32, 4 × 32, 8 × 32, and 16 × 32, with the burst base address aligned on the total burst size, are supported (this limitation applies to incrementing bursts only).

This interface also provides one interrupt and one DMA request line for specific event control.

It is recommended to program the GPMC_CONFIG1_i[24-23] ATTACHEDDEVICEPAGELENGTH bit field according to the page length of the effective attached device and to enable the GPMC_CONFIG1_i[31] WRAPBURST bit if the attached device supports wrapping burst.

It is possible, however, to emulate wrapping burst on a nonwrapping memory by providing relevant addresses within the page or by splitting transactions. Bursts larger than the memory page length are chopped into multiple burst transactions. Because of the alignment requirements, a page boundary is never crossed.

12.3.4.3.6 GPMC Address and Data Bus

The current application supports GPMC connection to NAND devices and to address/data-multiplexed memories or devices. Connection to address/data-non-multiplexed memories or devices is supported with an address range of 256MB.

Copyright © 2024 Texas Instruments Incorporated

Depending on the GPMC configuration of each chip-select, address and data bus lines that are not required for a particular access protocol are not updated (changed from current value) and are not sampled when input (input data bus).

- For address/data-multiplexed and AAD-multiplexed NOR devices, the address is multiplexed on the data bus.
- 8-bit-wide NOR devices do not use GPMC I/O: GPMC_AD[15-8] for data (they are used for address if needed).
- 16-bit-wide NAND devices do not use GPMC I/O: .
- 8-bit-wide NAND devices do not use GPMC I/O: and GPMC I/O: GPMC_AD[15-8].

12.3.4.3.6.1 GPMC I/O Configuration Setting

Note

In this section and the following sections, the i in GPMC_CONFIGx_i stands for the GPMC chip-select i, where i = 0 to 3.

To select a NAND device, program the following register fields:

- GPMC_CONFIG1_i[11-10] DEVICETYPE = 0b10
- GPMC_CONFIG1_i[9-8] MUXADDDATA = 0b00

To select an address/data-multiplexed device, program the following register fields:

- GPMC_CONFIG1_i[11-10] DEVICETYPE = 0b00
- GPMC_CONFIG1_i[9-8] MUXADDDATA = 0b10

To select an address/address/data-multiplexed device, program the following register fields:

- GPMC_CONFIG1_i[11-10] DEVICETYPE = 0b00
- GPMC_CONFIG1_i[9-8] MUXADDDATA = 0b01

To select an address/data-non-multiplexed device, program the following register fields:

- GPMC_CONFIG1_i[11-10] DEVICETYPE = 0b00
- GPMC_CONFIG1_i[9-8] MUXADDDATA = 0b00

12.3.4.3.7 GPMC Address Decoder and Chip-Select Configuration

Addresses are decoded accordingly with the address request of the chip-select and the content of the chip-select base address register file, which includes a set of global GPMC configuration registers and four sets of chip-select configuration registers.

The GPMC configuration register file is memory-mapped and can be read or written with byte, 16-bit word, or 32-bit word accesses. The register file must be configured as a noncacheable, nonbufferable region to prevent any desynchronization between host execution (write request) and the completion of register configuration (write completed with register updated).

After the chip-select is configured, the access engine accesses the external device, drives the external interface control signals, and applies the interface protocol based on user-defined timing parameters and settings.

12.3.4.3.7.1 Chip-Select Base Address and Region Size

Any external memory or ASIC device attached to the GPMC external interface can be accessed by any device system host within the GPMC 128MB address space. For more information, see *Memory Map*.

Note

Even though GPMC supports total address space of 1GB, only 128MB are physically available in this device.

The GPMC 128MB address space can be divided into a maximum of four chip-select regions with programmable base address and programmable chip-select size. The chip-select size is programmable from 16MB to 256MB

Copyright © 2024 Texas Instruments Incorporated

(must be a power-of-two) and is defined by the mask field. Attached memory smaller than the programmed chip-select region size is accessed through the entire chip-select region (aliasing).

Each chip-select has a 6-bit base address encoding and 4-bit decoding mask, which must be programmed according to the following rules:

- The programmed chip-select region base address must be aligned on the chip-select region size address boundary and is limited to a power-of-two address value. During access decoding, the value of the register base address is used to compare the address with the address bit line mapping, as shown in Figure 12-77 (with A0 as the device system byte-address line). The base address is programmed through the GPMC_CONFIG7_i[5-0] BASEADDRESS bit field.
- The register mask is used to exclude some address lines from the decoding. A register mask bit field set to 0 suppresses the associated address line from the address comparison (incoming address bit line is don't care). The value of the register mask must be limited to the subsequent value, based on the desired chip-select region size. Any other value has an undefined result. When multiple chip-select regions with overlapping addresses are enabled concurrently, access to these chip-select regions is cancelled and a GPMC access error is posted. The mask field is programmed through the GPMC_CONFIG7_i[11-8] MASKADDRESS bit field.

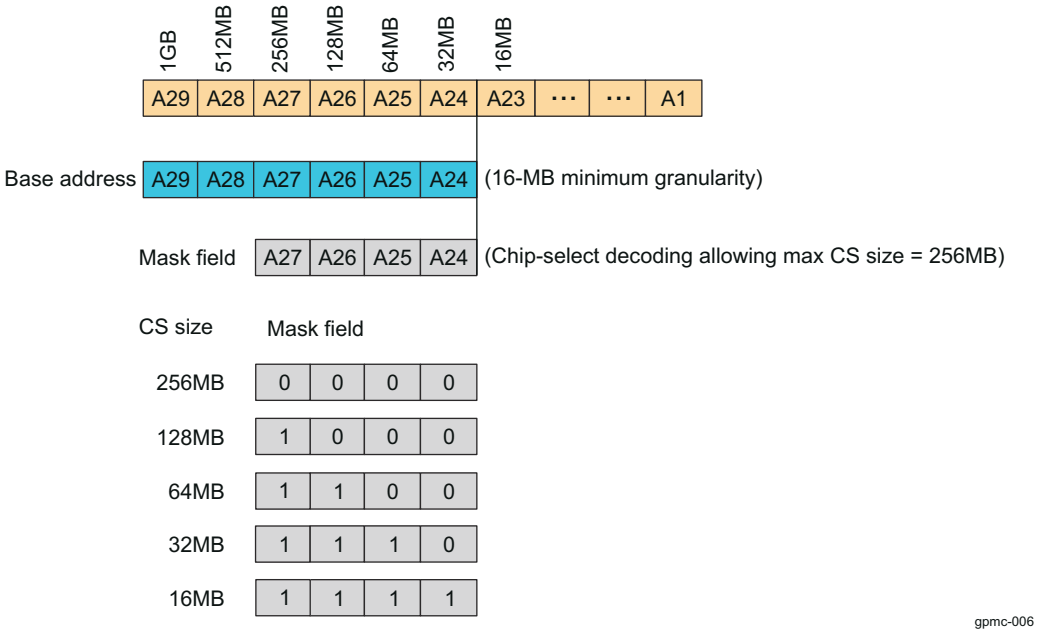


Figure 12-77. Chip-Select Address Mapping and Decoding Mask

For example, to map the 128MB address space (from 2000 0000h to 27FF FFFFh), the GPMC_CONFIG7_i[5-0] BASEADDRESS bit field should be set to 20h.

Chip-select configuration (base and mask address or any protocol and timing settings) must be performed while the associated chip-select is disabled through the GPMC_CONFIG7_i[6] CSVALID bit (where i stands for the GPMC chip-select i, where i = 0 to 3). In addition, a chip-select configuration can be disabled only if there is no ongoing access to that chip-select. This requires monitoring the activity of the prefetch or write-posting engine if the engine is active on the chip-select. Also, the write buffer state must be monitored to wait for any posted write completion to the chip-select.

Any access attempted to a nonvalid GPMC address region (CSVALID disabled or address decoding outside a valid chip-select region) is not propagated to the external interface and a GPMC access error is posted. In case of overlapping chip-selects, an error is generated and no access occurs on either chip-select.

CS0 is the only chip-select region enabled after a power up or GPMC reset.

Copyright © 2024 Texas Instruments Incorporated

CAUTION

Although the GPMC interface can drive up to four chip-selects, the frequency specified for this interface is for a specific load. If this load is exceeded, the maximum frequency cannot be reached. One solution is to implement a board with buffers to allow the slowest device to maintain the total load on the lines at the value specified in the device-specific Datasheet.

12.3.4.3.7.2 Access Protocol

12.3.4.3.7.2.1 Supported Devices

The access protocol of each chip-select can be independently specified through the GPMC_CONFIG1_i[11-10] DEVICETYPE parameter (where i = 0 to 3) for:

- Random-access synchronous or asynchronous memory, such as NOR flash and SRAM
- NAND flash asynchronous devices

Note

For more information about the NAND flash GPMC basic programming model and NAND support, see Section 12.3.4.3.11, *GPMC NAND Device Basic Programming Model*, and Section 12.3.4.3.11.1, *NAND Memory Device in Byte or Word16 Stream Mode*.

12.3.4.3.7.2.2 Access Size Adaptation and Device Width

Each chip-select can be independently configured through the GPMC_CONFIG1_i[13-12] DEVICESIZE bit field (where i = 0 to 3) to interface with a 16- or 8-bit-wide device. System requests with data width greater than the external device data bus width are split into successive accesses according to the external device data-bus width and little-endian data organization.

12.3.4.3.7.2.3 Address/Data-Multiplexing Interface

For random synchronous or asynchronous memory interfacing (DEVICETYPE = 0b00), an address- and data-multiplexing protocol can be selected through the GPMC_CONFIG1_i[9-8] MUXADDDATA bit field (where i = 0 to 3). The nADV signal must be used as the external device address latch control signal. For the associated chip-select configuration, nADV assertion and deassertion time and nOE assertion time must be set to the appropriate value to meet the address latch setup/hold time requirements of the external device. See *GPMC Integration*.

Note

This address/data-multiplexing interface is not applicable to NAND device interfacing. NAND devices require a specific address, command, and data-multiplexing protocol. See Section 12.3.4.3.11, *NAND Device Basic Programming Model*.

12.3.4.3.7.3 External Signals

12.3.4.3.7.3.1 WAIT Pin Monitoring Control

GPMC access time can be dynamically controlled using an external GPMC_WAIT pin when the external device access time is not deterministic and cannot be defined and controlled using only the GPMC internal RDACCESSTIME, WRACCESSTIME, and PAGEBURSTACCESSTIME wait-state generator.

The GPMC features four input WAIT pins: GPMC_WAIT0 through GPMC_WAIT3. These pins allow control of external devices with different WAIT pin polarity. They also allow the overlap of WAIT pin assertion from different devices without affecting access to devices for which the WAIT pin is not asserted.

- The GPMC_CONFIG1_i[17-16] WAITPINSELECT bit field (where i = 0 to 3) selects which input GPMC_WAIT pin is used for the device attached to the corresponding chip-select.
- The polarity of the WAIT pin is defined through the WAITxPINPOLARITY bit of the GPMC_CONFIG register. A WAIT pin configured to be active low means that low level on the WAIT signal indicates that the data is not ready and that the data bus is invalid. When a WAIT pin is inactive, data is valid.

Copyright © 2024 Texas Instruments Incorporated

The GPMC access engine can be configured per chip-select to monitor or not the WAIT pin of the external memory device, based on the access type: read or write.

- The GPMC_CONFIG1_i[22] WAITREADMONITORING bit defines whether or not the WAIT pin must be monitored during read accesses.
- The GPMC_CONFIG1_i[21] WAITWRITEMONITORING bit defines whether or not the WAIT pin must be monitored during write accesses.

The GPMC access engine can be configured to monitor the WAIT pin of the external memory device asynchronously or synchronously with the GPMC output clock, depending on the access type: synchronous or asynchronous (the GPMC_CONFIG1_i[29] READTYPE and GPMC_CONFIG1_i[27] WRITETYPE bits).

12.3.4.3.7.3.1.1 Wait Monitoring During Asynchronous Read Access

When WAIT pin monitoring is enabled for read accesses (WAITREADMONITORING), the effective access time is a logical AND combination of the RDACCESSTIME timing completion and the wait-deasserted state.

During asynchronous read accesses with WAIT pin monitoring enabled, the WAIT pin must be at a valid level (asserted or deasserted) for at least two GPMC clock cycles before RDACCESSTIME completes, to ensure correct dynamic access-time control through WAIT pin monitoring. The advance pipelining of the two GPMC clock cycles is the result of the internal synchronization requirements for the WAIT signal.

In this context, RDACCESSTIME is used as a wait invalid timing window and is set to such a value that the WAIT pin is at a valid state two GPMC clock cycles before RDACCESSTIME completes.

Similarly, during a multiple-access cycle (for example, asynchronous read page mode), the effective access time is a logical AND combination of PAGEBURSTACCESSTIME timing completion and the wait-deasserted state. Wait monitoring pipelining is also applicable to multiple accesses (access within a page).

- Wait monitored as active freezes the CYCLETIME counter. For an access within a page, when the CYCLETIME counter is by definition in a lock state, wait monitored as asserted extends the current access time in the page. Control signals are kept in their current state. The data bus is considered invalid, and no data are captured during this clock cycle.
- Wait monitored as inactive unfreezes the CYCLETIME counter. For an access within a page, when the CYCLETIME counter is by definition in a lock state, wait monitored as inactive completes the current access time and starts the next access phase in the page. The data bus is considered valid, and data are captured during this clock cycle. In case of a single access or if this was the last access in a multiple-access cycle, all signals are controlled according to their related control timing value and according to the CYCLETIME counter status.

When a delay larger than two GPMC clocks must be observed between wait-pin deactivation time and data valid time (including the required GPMC and the device data setup time), an extra delay can be added between wait-pin deassertion time detection and effective data-capture time and the effective unlock of the CYCLETIME counter. This extra delay can be programmed in the GPMC_CONFIG1_i[19-18] WAITMONITORINGTIME bit field (where i = 0 to 3).

Note

- The WAITMONITORINGTIME parameter does not delay the WAIT pin active or inactive detection, nor does it modify the two GPMC clocks pipelined detection delay.
- This extra delay is expressed as a number of GPMC output clock cycles, even though the access is defined as asynchronous, and no GPMC output clock is provided to the external device. Still, because GPMC_CONFIG1_i[1-0] GPMCFCLKDIVIDER is used as a divider for the GPMC clock, it must be programmed to define the correct WAITMONITORINGTIME delay.

Figure 12-78 shows wait behavior during an asynchronous single read access.

Figure 12-78. Wait Behavior During an Asynchronous Single Read Access (GPMCFCLKDivider = 1)

Copyright © 2024 Texas Instruments Incorporated

Note

The WAIT signal is active low. GPMC_CONFIG1_i[19-18] WAITMONITORINGTIME = 0b00, or 0b01.

12.3.4.3.7.3.1.2 Wait Monitoring During Asynchronous Write Access

When WAIT pin monitoring is enabled for write accesses (GPMC_CONFIG1_i[21] WAITWRITEMONITORING bit = 0x1), the wait invalid timing window is defined by the WRACCESSTIME field. WRACCESSTIME must be set so that the WAIT pin is at a valid state two GPMC clock cycles before WRACCESSTIME completes. The advance pipelining of the two GPMC clock cycles is the result of the internal synchronization requirements for the WAIT signal.

- Wait monitored as active freezes the CYCLETIME counter. This informs the GPMC that the data bus is not captured by the external device. The control signals are kept in their current state. The data bus still drives the data.
- Wait monitored as inactive unfreezes the CYCLETIME counter. This informs that the data bus is correctly captured by the external device. All signals, including the data bus, are controlled according to their related control timing value and to the CYCLETIME counter status.

When a delay larger than two GPMC clock cycles must be observed between wait-pin deassertion time and the effective data write into the external device (including the required GPMC data setup time and the device data setup time), an extra delay can be added between wait-pin deassertion time detection and effective data write time into the external device and the effective unfreezing of the CYCLETIME counter. This extra delay can be programmed in the GPMC_CONFIG1_i[19-18] WAITMONITORINGTIME bit field (where i = 0 to 3).

Note

- The WAITMONITORINGTIME parameter does not delay the WAIT pin assertion or deassertion detection, nor does it modify the two GPMC clock cycles pipelined detection delay.
- This extra delay is expressed as a number of GPMC output clock cycles, even though the access is defined as asynchronous, and even though no clock is provided to the external device. Still, because the GPMC_CONFIG1_i[1-0] GPMCFCLKDIVIDER bit field is used as a divider for the GPMC clock, it must be programmed to define the correct WAITMONITORINGTIME delay.

12.3.4.3.7.3.1.3 Wait Monitoring During Synchronous Read Access

During synchronous accesses with WAIT pin monitoring enabled, the WAIT pin is captured synchronously with GPMC output clock, using the rising edge of this clock.

The WAIT signal can be programmed to apply to the same clock cycle in which it is captured. Alternatively, it can be sampled one or two GPMC output clock cycles ahead of the clock cycle to which it applies. This pipelining is applicable to the entire burst access and to all data phases in the burst access. This wait pipelining depth is programmed in the GPMC_CONFIG1_i[19-18] WAITMONITORINGTIME bit field (where i = 0 to 3), and is expressed as a number of GPMC output clock cycles.

In synchronous mode, when WAIT pin monitoring is enabled (the GPMC_CONFIG1_i[22] WAITREADMONITORING bit), the effective access time is a logical AND combination of the RDACCESSTIME timing completion and the wait-deasserted state detection.

Depending on the programmed value of WAITMONITORINGTIME, the WAIT pin must be at a valid level, either asserted or deasserted:

- In the same clock cycle the data is valid if WAITMONITORINGTIME = 0 (at RDACCESSTIME completion)
- In the WAITMONITORINGTIME x (GPMCFCLKDIVIDER + 1) GPMC_FCLK clock cycles before RDACCESSTIME completion if WAITMONITORINGTIME is not equal to 0

Similarly, during a multiple-access cycle (burst mode), the effective access time is a logical AND combination of PAGEBURSTACCESSTIME timing completion and the WAIT-INACTIVE state. The wait pipelining-depth programming applies to the whole burst access.

- Wait monitored as active freezes the CYCLETIME counter. For an access within a burst (when the CYCLETIME counter is by definition in a lock state), wait monitored as active extends the current access

Copyright © 2024 Texas Instruments Incorporated

time in the burst. Control signals are kept in their current state. The data bus is considered invalid, and no data are captured during this clock cycle.

- Wait monitored as inactive unfreezes the CYCLETIME counter. For an access within a burst (when the CYCLETIME counter is by definition in lock state), wait monitored as inactive completes the current access time and starts the next access phase in the burst. The data bus is considered valid, and data are captured during this clock cycle. In a single access or if this was the last access in a multiple-access cycle, all signals are controlled according to their relative control timing value and the CYCLETIME counter status.

Figure 12-79 shows wait behavior during a synchronous read burst access.

Figure 12-79. Wait Behavior During a Synchronous Read Burst Access

Note

The WAIT signal is active low. WAITMONITORINGTIME = 0b00, 0b01.

12.3.4.3.7.3.1.4 Wait Monitoring During Synchronous Write Access

During synchronous accesses with WAIT pin monitoring enabled (the GPMC_CONFIG1_i[21] WAITWRITEMONITORING bit), the WAIT pin is captured synchronously with GPMC output clock, using the rising edge of this clock.

If enabled, external WAIT pin monitoring can be used in combination with WRACCESSTIME to delay the GPMC output clock capture edge of the effective memory device.

WAIT-monitoring pipelining depth is similar to synchronous read access:

- At WRACCESSTIME completion if WAITMONITORINGTIME = 0
- In the WAITMONITORINGTIME x (GPMCFCLKDIVIDER + 1) GPMC_FCLK cycles before WRACCESSTIME completion if WAITMONITORINGTIME is not equal to 0

Wait-monitoring pipelining definition applies to whole burst accesses:

- Wait monitored as active freezes the CYCLETIME counter. For accesses within a burst, when the CYCLETIME counter is by definition in a lock state, wait monitored as active indicates that the data bus is not being captured by the external device. Control signals are kept in their current state. The data bus is kept in its current state.
- Wait monitored as inactive unfreezes the CYCLETIME counter. For accesses within a burst, when the CYCLETIME counter is by definition in a lock state, wait monitored as inactive indicates the effective data capture of the bus by the external device and starts the next access of the burst. In case of a single access or if this was the last access in a multiple access cycle, all signals, including the data bus, are controlled according to their related control timing value and the CYCLETIME counter status.

Note

WAIT monitoring is supported for all configurations except GPMC_CONFIG1_i[19-18] WAITMONITORINGTIME = 0x0 (where i = 0 to 3) for write bursts with a clock divider of 1 or 2 (the GPMC_CONFIG1_i[1-0] GPMCFCLKDIVIDER bit field is equal to 0x0 or 0x1, respectively).

12.3.4.3.7.3.1.5 Wait With NAND Device

For information about the use of the WAIT pin for communication with a NAND flash external device, see Section 12.3.4.3.11.2, *NAND Device-Ready Pin*.

12.3.4.3.7.3.1.6 Idle Cycle Control Between Successive Accesses

12.3.4.3.7.3.1.6.1 Bus Turnaround (BUSTURNAROUND)

To prevent data-bus contention, an access that follows a read access to a slow memory/device must be delayed (in other words, control the nCS/nOE deassertion to data bus in high-impedance delay).

The bus turnaround is a time-out counter starting after nCS or nOE deassertion time, whichever occurs first, and delays the next access start-cycle time. The counter is programmed through the GPMC_CONFIG6_i[3-0] BUSTURNAROUND bit field (where i = 0 to 3).

Copyright © 2024 Texas Instruments Incorporated

After a read access to a chip-select with a nonzero BUSTURNAROUND, the next access is delayed until the BUSTURNAROUND delay completes, if the next access is one of the following:

- A write access to any chip-select (the same or different chip-select from which the data was read)
- A read access to a different chip-select than the chip-select from which the data was read access
- A read or write access to a chip-select associated with an address/data-multiplexed device

Note

Bus keeping starts after bus turnaround completion so that DIR changes from IN to OUT after bus turnaround. The bus does not have enough time to go into high-impedance even though it can be driven with the same value before bus turnaround timing.

BUSTURNAROUND delay runs in parallel with GPMC_CONFIG6_i[3-0] CYCLE2CYCLEDELAY bit field delays. BUSTURNAROUND is a timing parameter for the ending chip-select access, while CYCLE2CYCLEDELAY is a timing parameter for the following chip-select access. The effective minimum delay between successive accesses is driven by these delay timing parameters and by the access type of the following access (see the following figures).

Another way to prevent bus contention is to define an earlier nCS or nOE deassertion time for slow devices or to extend the value of RDCYCLETIME. Doing this prevents bus contention, but it also affects all accesses of this specific chip-select.

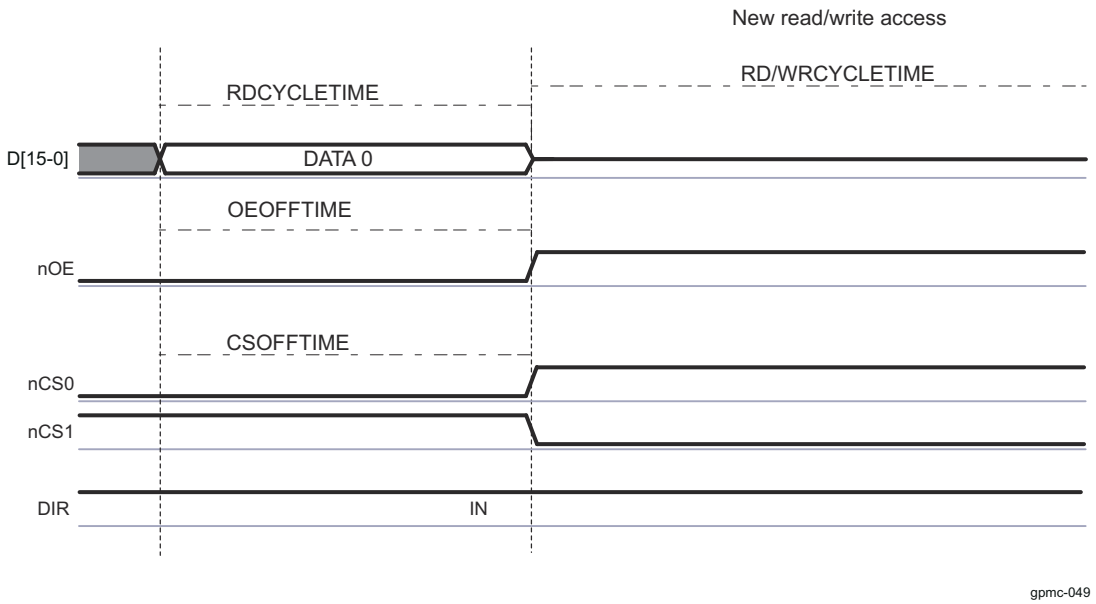


Figure 12-80. Read-to-Read for an Address-Data Multiplexed Device, on Different Chip-Select, Without Bus Turnaround (nCS Attached to a Fast Device)

12.3.4.3.7.3.1.6.2 Idle Cycles Between Accesses to Same Chip-Select (CYCLE2CYCLESAMECSEN, CYCLE2CYCLEDELAY)

Some devices require a minimum chip-select signal inactive time between accesses. The GPMC_CONFIG6_i[7] CYCLE2CYCLESAMECSEN bit (where i = 0 to 3) enables insertion of a minimum number of GPMC_FCLK cycles, defined by the GPMC_CONFIG6_i[11-8] CYCLE2CYCLEDELAY bit field, between successive accesses of any type (read or write) to the same chip-select.

If CYCLE2CYCLESAMECSEN is enabled, any subsequent access to the same chip-select is delayed until its CYCLE2CYCLEDELAY completes. The CYCLE2CYCLEDELAY counter starts when CSRDOFFTIME/ CSWROFFTIME completes.

Copyright © 2024 Texas Instruments Incorporated

The same applies to successive accesses occurring during 32-bit word or burst accesses split into successive single accesses when the single-access mode is used (GPMC_CONFIG1_i[30] READMULTIPLE = 0 or GPMC_CONFIG1_i[28] WRITEMULTIPLE = 0).

All control signals (CS, ADV#/ALE, BE0#/CLE, WE#, and GPMC output clock (CLK)) are kept inactive (ADV#/ALE, BE0#/CLE, and GPMC output clock at low level; and CS, OE#/RE, and WE at high level) during the idle GPMC_FCLK cycles. This prevents back-to-back accesses to the same chip-select without idle cycles between accesses.

12.3.4.3.7.3.1.6.3 Idle Cycles Between Accesses to Different Chip-Select (CYCLE2CYCLEDIFFCSEN, CYCLE2CYCLEDELAY)

Because of the pipelined behavior of the system, successive accesses to different chip-selects can occur back-to-back with no idle cycles between accesses. Depending on the control signals (nCS, nADV/ALE, nBE0/CLE, nOE/RE, nWE) assertion and deassertion timing parameters and on the device timing parameters, the assertion times of some control signals may overlap between the successive accesses to a different chip-select. Similarly, some control signals (WE, OE/RE) may not respect required transition times.

To work around overlapping and to observe the required control-signal transitions, a minimum of CYCLE2CYCLEDELAY inactive cycles is inserted between the access being initiated to this chip-select and the previous access ending for a different chip-select. This applies to any type of access (read or write).

If the GPMC_CONFIG6_i[6] CYCLE2CYCLEDIFFCSEN bit is enabled, the chip-select access is delayed until CYCLE2CYCLEDELAY cycles have expired since the end of a previous access to a different chip-select. CYCLE2CYCLEDELAY count starts at CSRDOFFTIME/CSWROFFTIME completion. All control signals are kept inactive during the idle GPMC_FCLK cycles.

Note

CYCLE2CYCLESAMECSEN and CYCLE2CYCLEDIFFCSEN must be set in the GPMC_CONFIG6_i registers to get idle cycles inserted between accesses on this chip-select and after accesses to a different chip-select, respectively.

The CYCLE2CYCLEDELAY delay runs in parallel with the BUSTURNAROUND delay. The BUSTURNAROUND is a timing parameter defined for the ending chip-select access, whereas CYCLE2CYCLEDELAY is a timing parameter defined for the starting chip-select access. The effective minimum delay between successive accesses is based on the larger delay timing parameter and on access type combination, because bus turnaround does not apply to all access types. For more information about bus turnaround, see Section 3.4.3.7.3.1.6.1, *Bus Turnaround (BUSTURNAROUND)*.

Table 12-166 describes the configuration required for idle cycle insertion.

Table 12-166. Idle Cycle Insertion Configuration

1st Access Type	BUSTURN AROUND Timing Parameter	Second Access Type	Chip-Select	Add/Data Multiplexed	CYCLE2 CYCLE SAMECSEN Parameter	CYCLE2 CYCLE DIFFCSEN Parameter	Idle Cycle Insertion Between the Two Accesses
R/W	= 0	R/W	Any	Any	0	x	No idle cycles are inserted if the two accesses are well pipelined.
R	> 0	R	Same	Nonmuxed	x	0	No idle cycles are inserted if the two accesses are well pipelined.
R	> 0	R	Different	Nonmuxed	0	0	BUSTURNAROUND cycles are inserted.
R	> 0	R/W	Any	Muxed	0	0	BUSTURNAROUND cycles are inserted.
R	> 0	W	Any	Any	0	0	BUSTURNAROUND cycles are inserted.

Copyright © 2024 Texas Instruments Incorporated

Table 12-166. Idle Cycle Insertion Configuration (continued)

1st Access Type	BUSTURN AROUND Timing Parameter	Second Access Type	Chip-Select	Add/Data Multiplexed	CYCLE2 CYCLE SAMECSEN Parameter	CYCLE2 CYCLE DIFFCSEN Parameter	Idle Cycle Insertion Between the Two Accesses
W	> 0	R/W	Any	Any	0	0	No idle cycles are inserted if the two accesses are well pipelined.
R/W	= 0	R/W	Same	Any	1	x	CYCLE2CYCLEDELAY cycles are inserted.
R/W	= 0	R/W	Different	Any	x	1	CYCLE2CYCLEDELAY cycles are inserted.
R/W	> 0	R/W	Same	Any	1	x	CYCLE2CYCLEDELAY cycles are inserted. If BTA idle cycles already apply on these two back-to-back accesses, the effective delay is max (BUSTURNAROUND, CYCLE2CYCLEDELAY).
R/W	> 0	R/W	Different	Any	x	1	CYCLE2CYCLEDELAY cycles are inserted. If BTA idle cycles already apply on these two back-to-back accesses, the effective delay is maximum (BUSTURNAROUND, CYCLE2CYCLEDELAY).

12.3.4.3.7.3.1.7 Slow Device Support (TIMEPARAGRANULARITY Parameter)

All access-timing parameters can be multiplied by 2 by setting the GPMC_CONFIG1_i[4] TIMEPARAGRANULARITY bit (where i stands for the GPMC chip-select i, where i = 0 to 3). Increasing all access timing parameters allows support of slow devices.

12.3.4.3.7.3.2 DIR Pin

The DIR pin is used to control I/O direction on the GPMC data bus GPMC_D[15-0]. Depending on pad multiplexing, this signal can be output and used externally to the device, if required. The DIR pin is low during transmit (OUT) and high during receive (IN).

For write accesses, the DIR pin stays OUT from start-cycle time to end-cycle time.

For read accesses, the DIR pin goes from OUT to IN at nOE assertion time and stays IN until:

- BUSTURNAROUND is enabled
  - The DIR pin goes from IN to OUT at end-cycle time plus programmable bus turnaround time.
- BUSTURNAROUND is disabled
  - After an asynchronous read access, the DIR pin goes from IN to OUT at RDACCESSTIME + 1 GPMC_FCLK cycle or when RDCYCLETIME completes, whichever occurs last.
  - After a synchronous read access, the DIR pin goes from IN to OUT at RDACCESSTIME + 2 GPMC_FCLK cycles or when RDCYCLETIME completes, whichever occurs last.

Because of the bus-keeping feature of the GPMC, after a read or write access and with no other accesses pending, the default value of the DIR pin is OUT (see Section 12.3.4.3.8.10, *Bus Keeping Support*). In non-multiplexed devices, the DIR pin stays IN between two successive read accesses to prevent unnecessary toggling.

12.3.4.3.7.3.3 Reset

No reset signal is sent to the external memory device by the GPMC.

GPMC_RST is the reset signal for the GPMC module. It is connected and controlled by LPSC8 in VD_CORE. That reset signal initializes the internal state-machine and the internal configuration registers.

Copyright © 2024 Texas Instruments Incorporated

12.3.4.3.7.3.4 Write Protect Signal (nWP)

When connected to the attached memory device, the write protect signal can enable or disable the lockdown function of the attached memory. The nWP output pin value is controlled through the GPMC_CONFIG[4] WRITEPROTECT bit which is common for all chip selects.

12.3.4.3.7.3.5 Byte Enable (nBE1/nBE0)

Byte enable signals (nBE1/nBE0) are:

- Valid (asserted or nonasserted according to the incoming system request) from access start to access completion for asynchronous and synchronous single accesses
- Asserted low from access start to access completion for asynchronous and synchronous multiple read accesses
- Valid (asserted or nonasserted, according to the incoming system request) synchronously to each written data for synchronous multiple write accesses

12.3.4.3.7.4 Error Handling

When an error occurs in the GPMC, the error information is stored in the GPMC_ERR_TYPE register and the address of the illegal access is stored in the GPMC_ERR_ADDRESS register. The GPMC keeps only the first error abort information until the GPMC_ERR_TYPE register is reset. Subsequent accesses that cause errors are not logged until the error is cleared by hardware with the GPMC_ERR_TYPE[0] ERRORVALID bit.

- ERRORNOTSUPPADD occurs when an incoming system request address decoding does not match any valid chip-select region, or if two chip-select regions are defined as overlapped, or if a register file access is tried outside the valid address range of 1KB.
- ERRORNOTSUPPMCMD occurs when an unsupported command request is decoded at the interconnect interface.
- ERRORTIMEOUT: A time-out mechanism prevents the system from hanging. The start value of the 9-bit time-out counter is defined in the GPMC_TIMEOUT_CONTROL register and enabled with the GPMC_TIMEOUT_CONTROL[0] TIMEOUTENABLE bit. When enabled, the counter starts at start-cycle time until it reaches 0 and data is not responded to from memory, and then a time-out error occurs. When data are sent from memory, this counter is reset to its start value. With multiple accesses (asynchronous page mode or synchronous burst mode), the counter is reset to its start value for each data access within the burst.

The GPMC does not generate interrupts on these errors. An interrupt generation is handled at interconnect level.

12.3.4.3.8 GPMC Timing Setting

The GPMC offers maximum flexibility to support various access protocols. Most of the timing parameters of the protocol access used by the GPMC to communicate with attached memories or devices are programmable on a chip-select basis. Assertion and deassertion times of control signals are defined to match the attached memory or device timing specifications and to get maximum performance during accesses. For more information about GPMC_CLKOUT and GPMC_FCLK, see *GPMC_CLKOUT*.

Note

In the following sections, the start access time refers to the time at which the access begins.

12.3.4.3.8.1 Read Cycle Time and Write Cycle Time (RDCYCLETIME / WRCYCLETIME)

The GPMC_CONFIG5_i[4-0] RDCYCLETIME and GPMC_CONFIG5_i[12-8] WRCYCLETIME bit fields (where i = 0 to 3) define the address bus and byte-enable valid times for read and write accesses. To ensure a correct duty cycle of GPMC output clock between accesses, RDCYCLETIME and WRCYCLETIME are expressed in GPMC_FCLK cycles and must be multiples of the GPMC output clock cycle. The RDCYCLETIME and WRCYCLETIME bit fields can be set with a granularity of 1 or 2 through the GPMC_CONFIG1_i[4] TIMEPARAGRANULARITY bit.

When RDCYCLETIME or WRCYCLETIME completes, if they are not already deasserted, all control signals (nCS, nADV/ALE, nOE/RE, nWE, and BE0/CLE) are deasserted to their reset values, regardless of their deassertion time parameters.

Copyright © 2024 Texas Instruments Incorporated

An exception to this forced deassertion occurs when a pipelined request to the same chip-select or to a different chip-select is pending. In such a case, it is not necessary to deassert a control signal with deassertion time parameters equal to the cycle-time parameter. This exception to forced deassertion prevents any unnecessary glitches. This requirement also applies to BE signals, thus avoiding an unnecessary BE glitch transition when pipelining requests.

Note

All control signals (CS, ADV#/ALE, BE0#/CLE, WE#, and GPMC output clock) are kept inactive (ADV#/ALE, BE0#/CLE, and GPMC output clock at low level; and CS, OE#/RE, and WE at high level) during the idle GPMC_FCLK cycles.

If no inactive cycles are required between successive accesses to the same chip-select or a different chip-select (GPMC_CONFIG6_i[7] CYCLE2CYCLESAMECSEN = 0 or GPMC_CONFIG6_i[6] CYCLE2CYCLEDIFFCSEN = 0, where i = 0 to 3), and if assertion-time parameters associated with the pipelined access are equal to 0, asserted control signals (nCS, nADV/ALE, nBE0/CLE, nWE, and nOE/RE) are kept asserted. This applies to any read/write to read/write access combination.

If inactive cycles are inserted between successive accesses (that is, CYCLE2CYCLESAMECSEN = 1 or CYCLE2CYCLEDIFFCSEN = 1), the control signals are forced to their respective default reset values for the number of GPMC_FCLK cycles defined in CYCLE2CYCLEDELAY.

12.3.4.3.8.2 nCS: Chip-Select Signal Control Assertion/Deassertion Time (CSONTIME / CSRDOFFTIME / CSWROFFTIME / CSEXTRADELAY)

The GPMC_CONFIG2_i[3-0] CSONTIME bit field (where i = 0 to 3) defines the nCS signal-assertion time relative to the start access time. It is common for read and write accesses.

The GPMC_CONFIG2_i[12-8] CSRDOFFTIME (read access) and GPMC_CONFIG2_i[20-16] CSWROFFTIME (write access) bit fields define the nCS signal deassertion time relative to start access time.

The CSONTIME, CSRDOFFTIME, and CSWROFFTIME parameters apply to synchronous and asynchronous modes. CSONTIME can be used to control an address and byte-enable setup time before chip-select assertion. CSRDOFFTIME and CSWROFFTIME can be used to control an address and byte-enable hold time after chip-select deassertion.

nCS signal transitions, as controlled through CSONTIME, CSRDOFFTIME, and CSWROFFTIME, can be delayed by a half-GPMC_FCLK period by enabling the GPMC_CONFIG2_i[7] CSEXTRADELAY bit. This half-GPMC_FCLK period provides more granularity on the nCS assertion and deassertion time to ensure proper setup and hold time relative to GPMC output clock. CSEXTRADELAY is especially useful in configurations where GPMC output clock and GPMC_FCLK have the same frequency, but it can also be used for all GPMC configurations. If enabled, CSEXTRADELAY applies to all parameters that control nCS transitions.

The CSEXTRADELAY bit must be used carefully to avoid control signal overlap between successive accesses to different chip-selects. This implies the need to program the RDCYCLETIME and WRCYCLETIME bit fields to be greater than the nCS signal-deassertion time, including the extra half-GPMC_FCLK-period delay.

12.3.4.3.8.3 nADV/ALE: Address Valid/Address Latch Enable Signal Control Assertion/ Deassertion Time (ADVONTIME / ADVRDOFFTIME / ADVWROFFTIME / ADVEXTRADELAY/ADVAADMUXONTIME/ ADVAADMUXRDOFFTIME/ADVAADMUXWROFFTIME)

The GPMC_CONFIG3_i[3-0] ADVONTIME field (where i = 0 to 3) defines the nADV/ALE signal-assertion time relative to start access time. It is common to read and write accesses.

The GPMC_CONFIG3_i[12-8] ADVRDOFFTIME (read access) and GPMC_CONFIG3_i[20-16] ADVWROFFTIME (write access) bit fields define the nADV/ALE signal-deassertion time relative to start access time.

ADVONTIME can be used to control an address and byte-enable valid setup time control before nADV/ALE assertion. ADVRDOFFTIME and ADVWROFFTIME can be used to control an address and byte-enable valid hold time control after nADV/ALE deassertion. ADVRDOFFTIME and ADVWROFFTIME apply to synchronous and asynchronous modes.

Copyright © 2024 Texas Instruments Incorporated

The nADV/ALE signal transitions as controlled through ADVONTIME, ADVRDOFFTIME, and ADVWROFFTIME can be delayed by a half-GPMC_FCLK period by enabling the GPMC_CONFIG3_i[7] ADVEXTRADELAY bit. This half-GPMC_FCLK period provides more granularity on nADV/ALE assertion and deassertion time to ensure proper setup and hold time relative to GPMC output clock. The ADVEXTRADELAY configuration parameter is especially useful in configurations where GPMC output clock and GPMC_FCLK have the same frequency, but can be used for all GPMC configurations. If enabled, ADVEXTRADELAY applies to all parameters controlling nADV/ALE transitions.

ADVEXTRADELAY must be used carefully to avoid control-signal overlap between successive accesses to different chip-selects. This implies the need to program the RDCYCLETIME and WRCYCLETIME bit fields to be greater than nADV/ALE signal-deassertion time, including the extra half-GPMC_FCLK-period delay.

GPMC_CONFIG3_i[6-4] ADVAADMUXONTIME, GPMC_CONFIG3_i[26-24] ADVAADMUXRDOFFTIME, and GPMC_CONFIG3_i[30-28] ADVAADMUXWROFFTIME parameters have the same functions as ADVONTIME, ADVRDOFFTIME, and ADVWROFFTIME, but apply to the first address phase in the AAD-multiplexed protocol. The user must ensure that ADVAADMUXxxOFFTIME is programmed to a value less than or equal to ADVxxOFFTIME. Functionality in AAD-multiplexed mode is undefined if the settings do not comply with this requirement. ADVAADMUXxxOFFTIME can be programmed to the same value as ADVONTIME if no high nADV pulse is needed between the two AAD-multiplexed address phases, which is the typical case in synchronous mode. In this configuration, nADV is kept low until it reaches the correct ADVxxOFFTIME.

For more information about the use of ADVONTIME, ADVRDOFFTIME, ADVWROFFTIME, and ADVAADMUXRDOFFTIME and ADVAADMUXWROFFTIME for command latch enable (CLE) and address latch enable (ALE) use for a NAND flash interface, see Section 12.3.4.3.11, *GPMC NAND Access Description*.

12.3.4.3.8.4 nOE/nRE: Output Enable/Read Enable Signal Control Assertion/Deassertion Time (OEONTIME / OEOFFTIME / OEEXTRADELAY / OEAADMUXONTIME / OEAADMUXOFFTIME)

The GPMC_CONFIG4_i[3-0] OEONTIME bit field (where i = 0 to 3) defines the nOE/nRE signal assertion time relative to start access time. It applies only to read accesses.

The GPMC_CONFIG4_i[12-8] OEOFFTIME bit field defines the nOE/nRE signal deassertion time relative to start access time. It applies only to read accesses. nOE/nRE is not asserted during a write cycle.

The OEONTIME, OEOFFTIME, OEAADMUXONTIME, and OEAADMUXOFFTIME parameters apply to synchronous and asynchronous modes. OEONTIME can be used to control an address and byte enable valid setup time control before nOE/nRE assertion. OEOFFTIME can be used to control an address and byte-enable valid hold time control after nOE/nRE assertion.

The OEAADMUXONTIME and OEAADMUXOFFTIME parameters have the same functions as OEONTIME and OEOFFTIME, but apply to the first OE assertion in the AAD-multiplexed protocol for a read phase, or to the only OE assertion for a write phase. The user mut ensure that OEAADMUXOFFTIME is programmed to a value less than OEONTIME. Functionality in AAD-multiplexed mode is undefined if the settings do not comply with this requirement. OEAADMUXOFFTIME must never be equal to OEONTIME because the AAD-multiplexed protocol requires a second address phase with the nOE signal deasserted before nOE can be asserted again to define a read command.

The nOE/RE signal transitions as controlled through OEONTIME, OEOFFTIME, OEAADMUXONTIME, and OEAADMUXOFFTIME can be delayed by a half-GPMC_FCLK period by enabling the GPMC_CONFIG4_i[7] OEEXTRADELAY bit. This half-GPMC_FCLK period provides more granularity on the nOE/RE assertion and deassertion time to ensure proper setup and hold time relative to GPMC output clock. If enabled, OEEXTRADELAY applies to all parameters controlling nOE/nRE transitions.

OEEXTRADELAY must be used carefully, to avoid control-signal overlap between successive accesses to different chip-selects. This implies the need to program RDCYCLETIME and WRCYCLETIME to be greater than the nOE/RE signal-deassertion time, including the extra half-GPMC_FCLK-period delay.

Note

When the GPMC generates a read access to an address-/data-multiplexed device, it drives the address bus until nOE assertion time.

Copyright © 2024 Texas Instruments Incorporated

12.3.4.3.8.5 nWE: Write Enable Signal Control Assertion/Deassertion Time (WEONTIME / WEOFFTIME / WEEXTRADELAY)

The GPMC_CONFIG4_i[19-16] WEONTIME bit field (where i = 0 to 3) defines the nWE signal-assertion time relative to start access time. The GPMC_CONFIG4_i[28-24] WEOFFTIME bit field defines the nWE signal-deassertion time relative to start access time. These bit fields apply only to write accesses. nWE is not asserted during a read cycle.

WEONTIME can be used to control an address and byte-enable valid setup time control before nWE assertion. WEOFFTIME can be used to control an address and byte-enable valid hold time control after nWE assertion.

nWE signal transitions as controlled through WEONTIME, and WEOFFTIME can be delayed by a half-GPMC_FCLK period by enabling the GPMC_CONFIG4_i[23] WEEXTRADELAY bit. This half-GPMC_FCLK period provides more granularity on nWE assertion and deassertion time to ensure proper setup and hold time relative to GPMC output clock. If enabled, WEEXTRADELAY applies to all parameters controlling nWE transitions.

The WEEXTRADELAY bit must be used carefully to avoid control-signal overlap between successive accesses to different chip-selects. This implies the need to program the WRCYCLETIME bit field to be greater than the nWE signal-deassertion time, including the extra half-GPMC_FCLK-period delay.

12.3.4.3.8.6 GPMC_CLKOUT

The GPMC output clock generated for external synchronous memory or device is GPMC_CLKOUT.

- The GPMC_CLKOUT clock frequency is the GPMC_FCLK functional clock frequency divided by 1, 2, 3, or 4, depending on the GPMC_CONFIG1_i[1-0] GPMCFCLKDIVIDER bit field (where i = 0 to 3), with a guaranteed 50-percent duty cycle. For information about the duty cycle error, see the device-specific Datasheet.
- The GPMC_CLKOUT clock is activated only when the access in progress is defined as synchronous (read or write access).
- The GPMC_CONFIG1_i[26-25] CLKACTIVATIONTIME bit field (where i = 0 to 3) defines the number of GPMC_FCLK cycles from start access time to GPMC_CLKOUT activation.
- The GPMC_CLKOUT clock is stopped when cycle time completes and is asserted low between accesses.
- The GPMC_CLKOUT clock is kept low when access is defined as asynchronous.

CAUTION

When the cycle time completes, the GPMC_CLKOUT may be high because of the GPMC_CONFIG1_i[1-0] GPMCFCLKDIVIDER bit field. To ensure correct stoppage of the GPMC_CLKOUT clock within the required 50-percent duty cycle, the user must extend the RDCYCLETIME or WRCYCLETIME value.

Note

To ensure a correct external clock cycle, the following rules must be applied:

- (RDCYCLETIME CLKACTIVATIONTIME) must be a multiple of (GPMCFCLKDIVIDER + 1).
- The PAGEBURSTACCESSTIME value must be a multiple of (GPMCFCLKDIVIDER + 1).

12.3.4.3.8.7 GPMC Output Clock and Control Signals Setup and Hold

Control-signal transition (assertion and deassertion) setup and hold values with respect to the GPMC output clock edge can be controlled in the following ways:

- For the GPMC output clock signal, the GPMC_CONFIG1_i[26-25] CLKACTIVATIONTIME bit field (where i = 0 to 3) allows setup and hold control of control-signal assertion time.
- The use of a divided GPMC output clock allows setup and hold control of the control-signal assertion and deassertion times.
- When the GPMC output clock runs at the GPMC_FCLK frequency so that GPMC output clock edge and control-signal transitions refer to the same GPMC_FCLK edge, the control-signal transitions can be delayed by a half-GPMC_FCLK period to provide minimum setup and hold times. This half-GPMC_FCLK delay is enabled with the CSEXTRADELAY, ADVEXTRADELAY, OEEXTRADELAY, or WEEXTRADELAY parameter.

Copyright © 2024 Texas Instruments Incorporated

This delay must be used carefully to prevent control-signal overlap between successive accesses to different chip-selects. This implies that the RDCYCLETIME and WRCYCLETIME are greater than the last control-signal deassertion time, including the extra half-GPMC_FCLK cycle.

12.3.4.3.8.8 Access Time (RDACCESSTIME / WRACCESSTIME)

The read/write access time durations can be programmed independently through the GPMC_CONFIG5_i[20-16] RDACCESSTIME and GPMC_CONFIG6_i[28-24] WRACCESSTIME bit fields (where i = 0 to 3). This allows nOE and GPMC data-capture timing parameters to be independent of nWE and memory device data capture timing parameters. The RDACCESSTIME and WRACCESSTIME bit fields can be set with a granularity of 1 or 2 through the GPMC_CONFIG1_i[4] TIMEPARAGRANULARITY bit.

12.3.4.3.8.8.1 Access Time on Read Access

In asynchronous read mode, for single and paged accesses, the GPMC_CONFIG5_i[20-16] RDACCESSTIME bit field (where i = 0 to 3) defines the number of GPMC_FCLK cycles from start access time to the GPMC_FCLK rising edge used for the first data capture. RDACCESSTIME must be programmed to the rounded greater value (in GPMC_FCLK cycles) of the read access time of the attached memory device.

In synchronous read mode, for single or burst accesses, RDACCESSTIME defines the number of GPMC_FCLK cycles from the start access time to the GPMC_FCLK rising edge corresponding to the GPMC output clock rising edge used for the first data capture.

GPMC output clock, which is sent to the memory device for synchronization with the GPMC controller, is internally retimed to correctly latch the returned data. The GPMC_CONFIG5_i[4-0] RDCYCLETIME bit field must be greater than RDACCESSTIME to let the GPMC latch the last return data using the internally retimed GPMC output clock.

The external WAIT signal can be used in conjunction with RDACCESSTIME to control the effective GPMC data-capture GPMC_FCLK edge on read access in asynchronous and synchronous modes. For more information about wait monitoring, see Section 12.3.4.3.7.3.1, *WAIT Pin Monitoring Control*.

12.3.4.3.8.8.2 Access Time on Write Access

In asynchronous write mode, the GPMC_CONFIG6_i[28-24] WRACCESSTIME timing parameter is not used to define the effective write access time. Instead, it is used as a wait invalid timing window and must be set to a correct value so that the GPMC_WAIT pin is at a valid state two GPMC output clock cycles before WRACCESSTIME completes. For more information about wait monitoring, see Section 12.3.4.3.7.3.1, *WAIT Pin Monitoring Control*.

In synchronous write mode, for single or burst accesses, WRACCESSTIME defines the number of GPMC_FCLK cycles from the start access time to the GPMC output clock rising edge used by the memory device for the first data capture.

The external WAIT signal can be used in conjunction with WRACCESSTIME to control the effective memory device data-capture GPMC output clock edge for a synchronous write access. For more information about wait monitoring, see Section 12.3.4.3.7.3.1, *WAIT Pin Monitoring Control*.

12.3.4.3.8.9 Page Burst Access Time (PAGEBURSTACCESSTIME)

The GPMC_CONFIG5_i[27-24] PAGEBURSTACCESSTIME bit field (where i = 0 to 3) can be set with a granularity of 1 or 2 through the GPMC_CONFIG1_i[4] TIMEPARAGRANULARITY bit.

12.3.4.3.8.9.1 Page Burst Access Time on Read Access

In asynchronous page read mode, the delay between successive word captures in a page is controlled through the PAGEBURSTACCESSTIME bit field. The PAGEBURSTACCESSTIME parameter must be programmed to the rounded greater value (in GPMC_FCLK cycles) of the read access time of the attached device.

In synchronous burst read mode, the delay between successive word captures in a burst is controlled through the PAGEBURSTACCESSTIME bit field.

The external WAIT signal can be used in conjunction with PAGEBURSTACCESSTIME to control the effective GPMC data-capture GPMC_FCLK edge on read access. For more information about wait monitoring, see Section 12.3.4.3.7.3.1, *WAIT Pin Monitoring Control*.

Copyright © 2024 Texas Instruments Incorporated

12.3.4.3.8.9.2 Page Burst Access Time on Write Access

Asynchronous page write mode is not supported. PAGEBURSTACCESSTIME is irrelevant in this case.

In synchronous burst write mode, PAGEBURSTACCESSTIME controls the delay between successive memory device word captures in a burst.

The external WAIT signal can be used in conjunction with PAGEBURSTACCESSTIME to control the effective memory device data capture GPMC output clock edge in synchronous write mode. For more information about wait monitoring, see Section 12.3.4.3.7.3.1, *WAIT Pin Monitoring Control*.

12.3.4.3.8.10 Bus Keeping Support

At the end cycle time of a read access, if no other access is pending, the GPMC drives the bus with the last data read after RDCYCLETIME completes to prevent bus floating and reduce power consumption.

After a write access, if no other access is pending, the GPMC keeps driving the data bus after WRCYCLETIME completes with the same data to prevent bus floating and power consumption.

12.3.4.3.9 GPMC NOR Access Description

For each chip-select configuration, the read access can be specified as asynchronous or synchronous access through the GPMC_CONFIG1_i[29] READTYPE bit (where i = 0 to 3). For each chip-select configuration, the write access can be specified as synchronous or asynchronous access through the GPMC_CONFIG1_i[27] WRITETYPE bit where (i = 0 to 3).

Asynchronous and synchronous read and write access time and related control signals are controlled through timing parameters that refer to GPMC_FCLK. The primary difference of synchronous mode is the availability of a configurable clock interface to control the external device. Synchronous mode also affects data-capture and wait-pin monitoring schemes in read access.

For more information about asynchronous and synchronous access, see the descriptions of GPMC output clock (CLK), RdAccessTime, WrAccessTime, and WAIT pin monitoring.

For more information about timing-parameter settings, see the sample timing diagrams in this chapter.

Note

The address bus and nBE[1-0] are fixed for the duration of a synchronous burst read access, but they are updated for each beat of an asynchronous page-read access.

12.3.4.3.9.1 Asynchronous Access Description

This section describes:

- Asynchronous single-read operation on an address/data multiplexed device
- Asynchronous single write operation on an address/data-multiplexed device
- Asynchronous single read operation on an AAD-multiplexed device
- Asynchronous single write operation on an AAD-multiplexed device
- Asynchronous multiple (page) read operation on a non-multiplexed device

In asynchronous operations GPMC output clock is not provided outside the GPMC and is kept low.

12.3.4.3.9.1.1 Access on Address/Data Multiplexed Devices

12.3.4.3.9.1.1.1 Asynchronous Single-Read Operation on an Address/Data Multiplexed Device

For formulas to calculate timing parameters, see Section 12.3.4.4.6.1, *GPMC Timing Parameters Formulas*.

Table 12-206 lists the timing bit fields to set up to configure the GPMC in asynchronous single-read mode.

When the GPMC generates a read access to an address/data-multiplexed device, it drives the address bus until nOE assertion time. For more information, see Section 12.3.4.3.7.2.3, *Address/Data-Multiplexing Interface*.

Address bits (A[16-1] from a GPMC perspective, A[15-0] from an external device perspective) are placed on the address/data bus, and the remaining address bits are placed on the address bus. The address phase ends at nOE assertion, when the DIR signal goes from OUT to IN.

Copyright © 2024 Texas Instruments Incorporated

- Chip-select signal nCS:
  - nCS assertion time is controlled by the GPMC_CONFIG2_i[3-0] CSONTIME bit field. It controls the address setup time to nCS assertion.
  - nCS deassertion time is controlled by the GPMC_CONFIG2_i[12-8] CSRDOFFTIME bit field. It controls the address hold time from nCS deassertion.
- Address valid signal nADV:
  - nADV assertion time is controlled by the GPMC_CONFIG3_i[3-0] ADVONTIME bit field.
  - nADV deassertion time is controlled by the GPMC_CONFIG3_i[12-8] ADVRDOFFTIME bit field.
- Output enable signal nOE:
  - nOE assertion indicates a read cycle.
  - nOE assertion time is controlled by the GPMC_CONFIG4_i[3-0] OEONTIME bit field.
  - nOE deassertion time is controlled by the GPMC_CONFIG4_i[12-8] OEOFFTIME bit field.
- Read data is latched when RDACCESSTIME completes. Access time is defined in the GPMC_CONFIG5_i[20-16] RDACCESSTIME bit field.
- Direction signal DIR: DIR goes from OUT to IN at the same time that nOE is asserted.
- The end of the access is defined by the GPMC_CONFIG5_i[4-0] RDCYCLETIME parameter.

In the GPMC, when a 16-bit wide device is attached to the controller, a 32-bit word write access is split into two 16-bit word write accesses. For more information about GPMC access size and type adaptation, see Section 12.3.4.3.9.5, *System Burst Versus External Device Burst Support*.

Between two successive accesses, if an nCS pulse is needed:

- The GPMC_CONFIG6_i[11-8] CYCLE2CYCLEDELAY bit field can be programmed with the GPMC_CONFIG6_i[7] CYCLE2CYCLESAMECSEN bit enabled.
- The CSWROFFTIME and CSONTIME parameters also allow a chip-select pulse, but this affects all other types of access.

12.3.4.3.9.1.1.2 Asynchronous Single-Write Operation on an Address/Data-Multiplexed Device

For formulas to calculate timing parameters, see Section 12.3.4.4.6.1, *GPMC Timing Parameters Formulas*.

Table 12-206 lists the timing bit fields to set up to configure the GPMC in asynchronous single-write mode.

When the GPMC generates a write access to an address/data-multiplexed device, it drives the address bus until nWE assertion time. For more information, see Section 12.3.4.3.7.2.3, *Address/Data-Multiplexing Interface*.

The nCS and nADV signals are controlled in the same way as for a asynchronous single-read operation on an address/data-multiplexed device.

- Write enable signal nWE:
  - nWE assertion indicates a write cycle.
  - nWE assertion time is controlled by the GPMC_CONFIG4_i[19-16] WEONTIME bit field.
  - nWE deassertion time is controlled by the GPMC_CONFIG4_i[28-24] WEOFFTIME bit field.
- Direction signal DIR: DIR signal is OUT during the entire access.
- The end of the access is defined by the GPMC_CONFIG5_i[12-8] WRCYCLETIME parameter.

Address bits A[16-1] (GPMC point of view) are placed on the address/data bus at the start of cycle time, and the remaining address bits are placed on the address bus.

Data is driven on the address/data bus at a GPMC_CONFIG6_i[19-16] WRDATAONADMUXBUS time.

Note

Multiple write access in asynchronous mode is not supported. If WRITEMULTIPLE is enabled with WRITETYPE as asynchronous, the GPMC processes single asynchronous accesses.

After a write operation, if no other access (read or write) is pending, the data bus keeps its previous value. See Section 12.3.4.3.8.10, *Bus Keeping Support*.

12.3.4.3.9.1.1.3 Asynchronous Multiple (Page) Write Operation on an Address/Data-Multiplexed Device

Write multiple (page) access in asynchronous mode is not supported for address/data-multiplexed devices.

Copyright © 2024 Texas Instruments Incorporated

If the GPMC_CONFIG1_i[28] WRITEMULTIPLE bit is enabled (0x1) with the GPMC_CONFIG1_i[27] WRITETYPE bit as asynchronous (0x0), the GPMC processes single asynchronous accesses.

For accesses on non-multiplexed devices, see Section 12.3.4.3.9.3, *Asynchronous and Synchronous Accesses in non-multiplexed Mode*.

12.3.4.3.9.1.2 Access on Address/Address/Data-Multiplexed Devices

12.3.4.3.9.1.2.1 Asynchronous Single Read Operation on an AAD-Multiplexed Device

For formulas to calculate timing parameters, see Section 12.3.4.4.6.1, *GPMC Timing Parameters Formulas*.

Table 12-206 lists the timing bit fields to set up to configure the GPMC in asynchronous single write mode.

When the GPMC generates a read access to an AAD-multiplexed device, all address bits are driven onto the address/data bus in two separate phases. The first phase is used for the MSB address and is qualified with nOE driven low. The first address phase ends at the first nOE deassertion time. The second phase for LSB address is qualified with nOE driven high. The second address phase ends at the second nOE assertion time, when the DIR signal goes from OUT to IN.

The nCS and DIR signals are controlled in the same way as for an asynchronous single-read operation on an address/data-multiplexed device.

- Address valid signal nADV. nADV is asserted and deasserted twice during a read transaction:
  - nADV first assertion time is controlled by the GPMC_CONFIG3_i[6-4] ADVAADMUXONTIME bit field.
  - nADV first deassertion time is controlled by the GPMC_CONFIG3_i[26-24] ADVAADMUXRDOFFTIME bit field.
  - nADV second assertion time is controlled by the GPMC_CONFIG3_i[3-0] ADVONTIME bit field.
  - nADV second deassertion time is controlled by the GPMC_CONFIG3_i[12-8] ADVRDOFFTIME bit field.
- Output Enable signal nOE. nOE is asserted and deasserted twice during a read transaction (nOE second assertion indicates a read cycle):
  - nOE first assertion time is controlled by the GPMC_CONFIG4_i[6-4] OEAADMUXONTIME bit field.
  - nOE first deassertion time is controlled by the GPMC_CONFIG3_i[15-13] OEAADMUXOFFTIME bit field.
  - nOE second assertion time is controlled by the GPMC_CONFIG4_i[3-0] OEONTIME bit field.
  - nOE second deassertion time is controlled by the GPMC_CONFIG4_i[12-8] OEOFFTIME bit field.

12.3.4.3.9.1.2.2 Asynchronous Single-Write Operation on an AAD-Multiplexed Device

For formulas to calculate timing parameters, see Section 12.3.4.4.6.1, *GPMC Timing Parameters Formulas*.

Table 12-206 lists the timing bit fields to set up to configure the GPMC in asynchronous single-write mode.

When the GPMC generates a write access to an AAD-multiplexed device, all address bits are driven onto the address/data bus in two separate phases. The first phase is used for the MSB address and is qualified with nOE driven low. The second phase for LSB address is qualified with nOE driven high. The address phase ends at nWE assertion time.

The nCS, nWE, and DIR signals are controlled in the same way as for an asynchronous single-write operation on an address/data-multiplexed device. See Table 12-197, *NAND Memory Type*.

- Address valid signal nADV is asserted and deasserted twice during a write transaction:
  - nADV first assertion time is controlled by the GPMC_CONFIG3_i[6-4] ADVAADMUXONTIME bit field.
  - nADV first deassertion time is controlled by the GPMC_CONFIG3_i[30-28] ADVAADMUXWROFFTIME bit field.
  - nADV second assertion time is controlled by the GPMC_CONFIG3_i[3-0] ADVONTIME bit field.
  - nADV second deassertion time is controlled by the GPMC_CONFIG3_i[20-16] ADVWROFFTIME bit field.
- Output enable signal nOE is asserted during the address phase of a write transaction:
  - nOE assertion time is controlled by the GPMC_CONFIG4_i[6-4] OEAADMUXONTIME bit field.
  - nOE deassertion time is controlled by the GPMC_CONFIG3_i[15-13] OEAADMUXOFFTIME bit field.

The address bits for the first address phase are driven onto the data bus until nOE deassertion. Data is driven onto the address/data bus at the clock edge defined by the GPMC_CONFIG6_i[19-16] WRDATAONADMUXBUS parameter.

Copyright © 2024 Texas Instruments Incorporated

12.3.4.3.9.1.2.3 Asynchronous Multiple (Page) Read Operation on an AAD-Multiplexed Device

Write multiple (page) access in asynchronous mode is not supported for AAD-multiplexed devices.

If the GPMC_CONFIG1_i[28] WRITEMULTIPLE bit is enabled (0x1) with the GPMC_CONFIG1_i[27] WRITETYPE bit as asynchronous (0x0), the GPMC processes single asynchronous accesses.

For accesses on non-multiplexed devices, see Section 12.3.4.3.9.3, *Asynchronous and Synchronous Accessed in non-multiplexed Mode*.

12.3.4.3.9.2 Synchronous Access Description

This section describes read and write synchronous accesses on address/data-multiplexed devices. All information in this section can be applied to any type of memory (non-multiplexed, address and data-multiplexed, or AAD-multiplexed) with the difference limited to the address phase. For accesses on non-multiplexed devices, see Section 12.3.4.3.9.3, *Asynchronous and Synchronous Accessed in non-multiplexed Mode*.

In synchronous operations:

- The GPMC_CLKOUT clock is provided outside the GPMC when accessing the memory device.
- The GPMC_CLKOUT clock is derived from the GPMC_FCLK clock using the GPMC_CONFIG1_i[1-0] GPMCFCLKDIVIDER bit field. In the following section i stands for the chip-select number, i = 0 to 3.
- The GPMC_CONFIG1_i[26-25] CLKACTIVATIONTIME bit field specifies that the GPMC_CLKOUT is provided outside the GPMC for 0 to 2 GPMC_FCLK cycles after start access time until RDCYCLETIME or WRCYCLETIME completes.

12.3.4.3.9.2.1 Synchronous Single Read

For formulas to calculate timing parameters, see Section 12.3.4.4.6.1, *GPMC Timing Parameters Formulas*.

Table 12-206 lists the timing bit fields to set up to configure the GPMC in asynchronous single-read mode.

When the GPMC generates a read access to an address/data-multiplexed device, it drives the address bus until nOE assertion time. For more information, see Section 12.3.4.3.7.2.3, *Address/Data-Multiplexing Interface*.

- Chip-select signal nCS:
  - nCS assertion time is controlled by the GPMC_CONFIG2_i[3-0] CSONTIME bit field and ensures address setup time to nCS assertion.
  - nCS deassertion time is controlled by the GPMC_CONFIG2_i[12-8] CSRDOFFTIME bit field and ensures address hold time to nCS deassertion.
- Address valid signal nADV:
  - nADV assertion time is controlled by the GPMC_CONFIG3_i[3-0] ADVONTIME bit field.
  - nADV deassertion time is controlled by the GPMC_CONFIG3_i[12-8] ADVRDOFFTIME bit field.
- Output enable signal nOE:
  - nOE assertion indicates a read cycle.
  - nOE assertion time is controlled by the GPMC_CONFIG4_i[3-0] OEONTIME bit field.
  - nOE deassertion time is controlled by the GPMC_CONFIG4_i[12-8] OEOFFTIME bit field.
- Initial latency for the first read data is controlled by GPMC_CONFIG5_i[20-16] RDACCESSTIME bit field or by monitoring the WAIT signal.
- Total access time (the GPMC_CONFIG5_i[4-0] RDCYCLETIME bit field) corresponds to RDACCESSTIME plus the address hold time from nCS deassertion, plus time from RDACCESSTIME to CSRDOFFTIME.
- Direction signal DIR: DIR goes from OUT to IN at the same time as nOE assertion.

When the GPMC generates a write access to an AAD-multiplexed device, all address bits are driven onto the address/data bus in two separate phases. The first phase is used for the MSB address and is qualified with nOE driven low. The second phase for LSB address is qualified with nOE driven high. The address phase ends at nWE assertion time.

The nCS and DIR signals are controlled in the same way as for a synchronous single-read operation on an address/data-multiplexed device.

- Address valid signal nADV is asserted and deasserted twice during a read transaction:
  - nADV first assertion time is controlled by the GPMC_CONFIG3_i[6-4] ADVAADMUXONTIME bit field.

Copyright © 2024 Texas Instruments Incorporated

  - nADV first deassertion time is controlled by the GPMC_CONFIG3_i[26-24] ADVAADMUXRDOFFTIME bit field.
  - nADV second assertion time is controlled by the GPMC_CONFIG3_i[3-0] ADVONTIME bit field.
  - nADV second deassertion time is controlled by the GPMC_CONFIG3_i[12-8] ADVRDOFFTIME bit field.
- Output Enable signal nOE is asserted and deasserted twice during a read transaction (nOE second assertion indicates a read cycle):
  - nOE first assertion time is controlled by the GPMC_CONFIG4_i[6-4] OEAADMUXONTIME bit field.
  - nOE first deassertion time is controlled by the GPMC_CONFIG3_i[15-13] OEAADMUXOFFTIME bit field.
  - nOE second assertion time is controlled by the GPMC_CONFIG4_i[3-0] OEONTIME bit field.
  - nOE second deassertion time is controlled by the GPMC_CONFIG4_i[12-8] OEOFFTIME bit field.

After a read operation, if no other access (read or write) is pending, the data bus is driven with the previous read value. See Section 12.3.4.3.8.10, *Bus Keeping Support*.

12.3.4.3.9.2.2 Synchronous Multiple (Burst) Read (4-, 8-, 16-Word16 Burst With Wraparound Capability)

Figure 12-81 and Figure 12-82 show a synchronous multiple-read operation with GPMCFCLKDivider equal to 0 and 1, respectively.

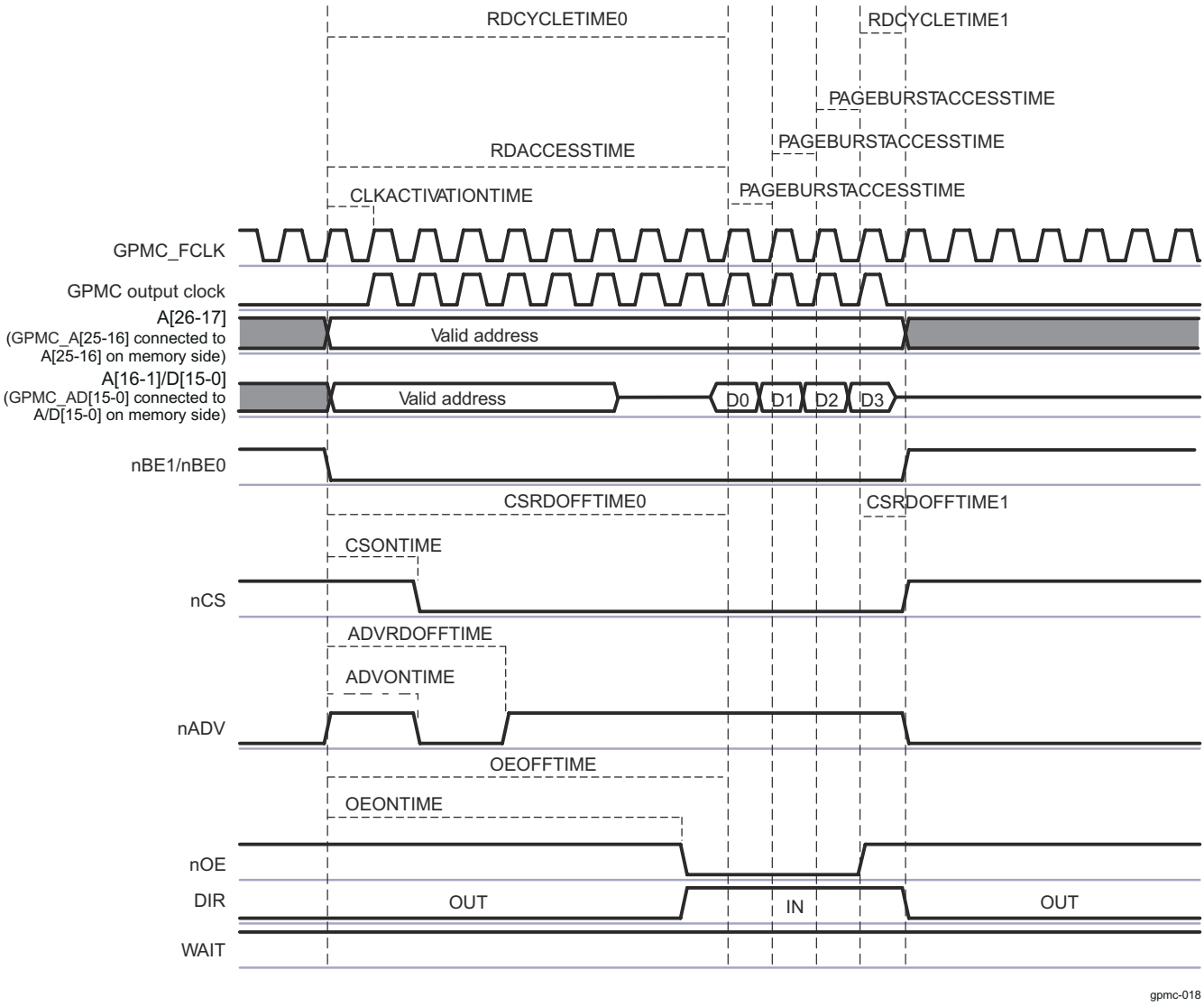


Figure 12-81. Synchronous Multiple (Burst) Read (GPMCFCLKDIVIDER = 0)

Copyright © 2024 Texas Instruments Incorporated

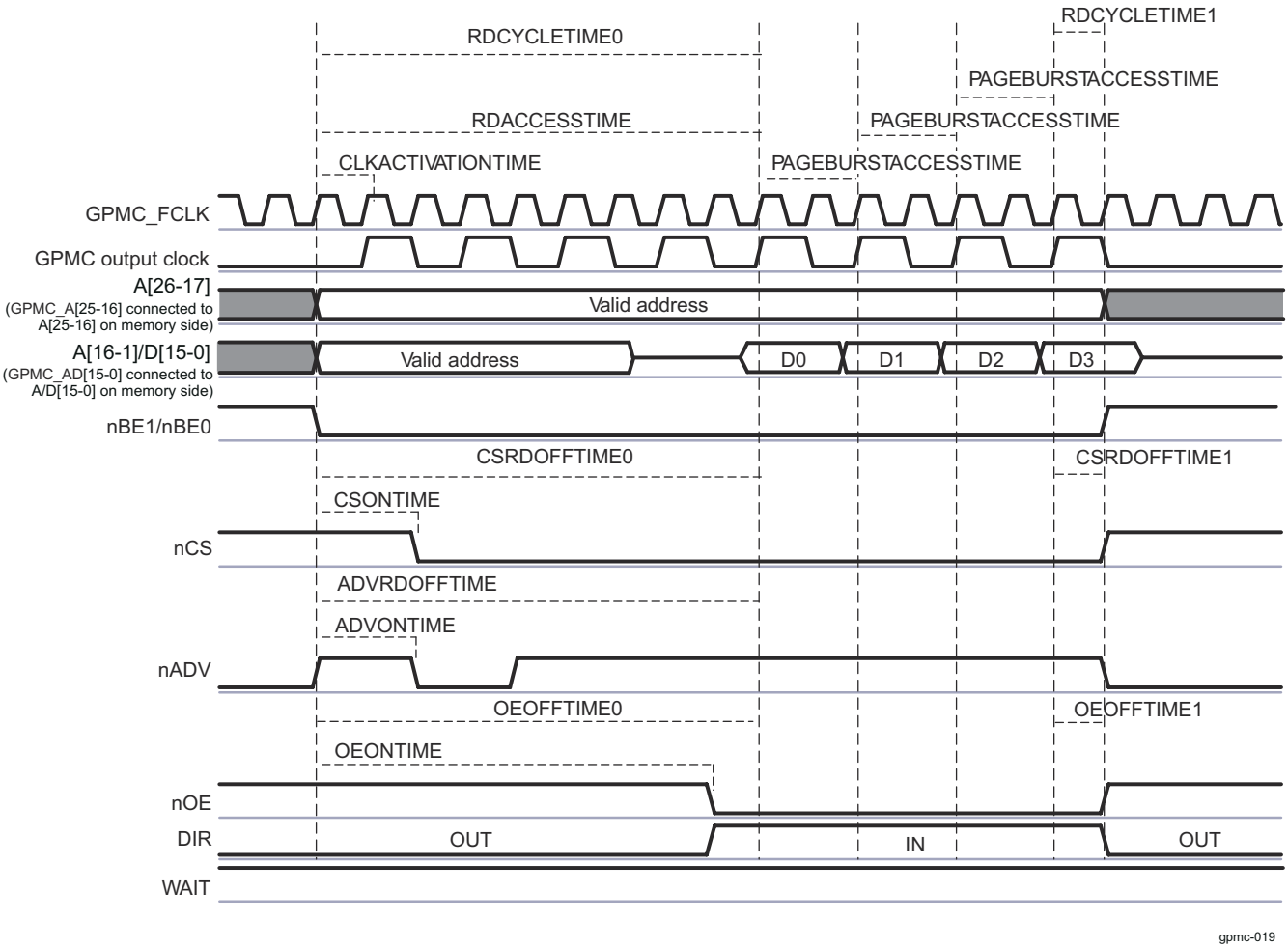


Figure 12-82. Synchronous Multiple (Burst) Read (GPMCFCLKDIVIDER = 1)

When the GPMC_CONFIG5_i[20-16] RDACCESSTIME bit field completes, control-signal timings are frozen during the multiple data transactions, corresponding to the GPMC_CONFIG5_i[27-24] PAGEBURSTACCESSTIME bit field multiplied by the number of remaining data transactions.

The nCS, nADV, nOE, and DIR signals are controlled in the same way as for a synchronous single-read operation. See Table 12-192, *NOR Memory Type*.

Initial latency for the first read data is controlled by RDACCESSTIME or by monitoring the WAIT signal. Successive read data are provided by the memory device every one or two GPMC_CLKOUT cycles. The PAGEBURSTACCESSTIME parameter must be set accordingly with the GPMC_CONFIG1_i[1-0] GPMCFCLKDIVIDER bit field and the memory-device internal configuration. Depending on the device page length, the GPMC checks the device page crossing during a new burst request and purposely inserts initial latency (of RDACCESSTIME) when required.

Total access time GPMC_CONFIG5_i[4-0] RDCYCLETIME corresponds to RDACCESSTIME plus the address hold time from nCS deassertion. In Figure 12-82, the programmed value of RDCYCLETIME equals RDCYCLETIME0 + RDCYCLETIME1.

After a read operation, if no other access (read or write) is pending, the data bus is driven with the previous read value. See Section 12.3.4.3.8.10, *Bus Keeping Support*.

Burst wraparound is enabled through the GPMC_CONFIG1_i[31] WRAPBURST bit and allows a 4-, 8-, or 16-Word16 linear burst access to wrap within its burst-length boundary through the GPMC_CONFIG1_i[24-23] ATTACHEDDEVICEPAGELENGTH bit field.

Copyright © 2024 Texas Instruments Incorporated

12.3.4.3.9.2.3 Synchronous Single Write

Burst write mode is used for synchronous single or burst accesses.

When the GPMC generates a write access to an address/data-multiplexed device, it drives the data bus (with address bits A[16-1]) until the GPMC_CONFIG6_i[19-16] WRDATAONADMUXBUS bit field time. The first data of the burst is driven on the address/data bus at WRDATAONADMUXBUS time.

12.3.4.3.9.2.4 Synchronous Multiple (Burst) Write

Synchronous burst write mode provides synchronous single or consecutive accesses.

Figure 12-83 shows a synchronous burst write access when the chip-select is configured in address/data-multiplexed mode.

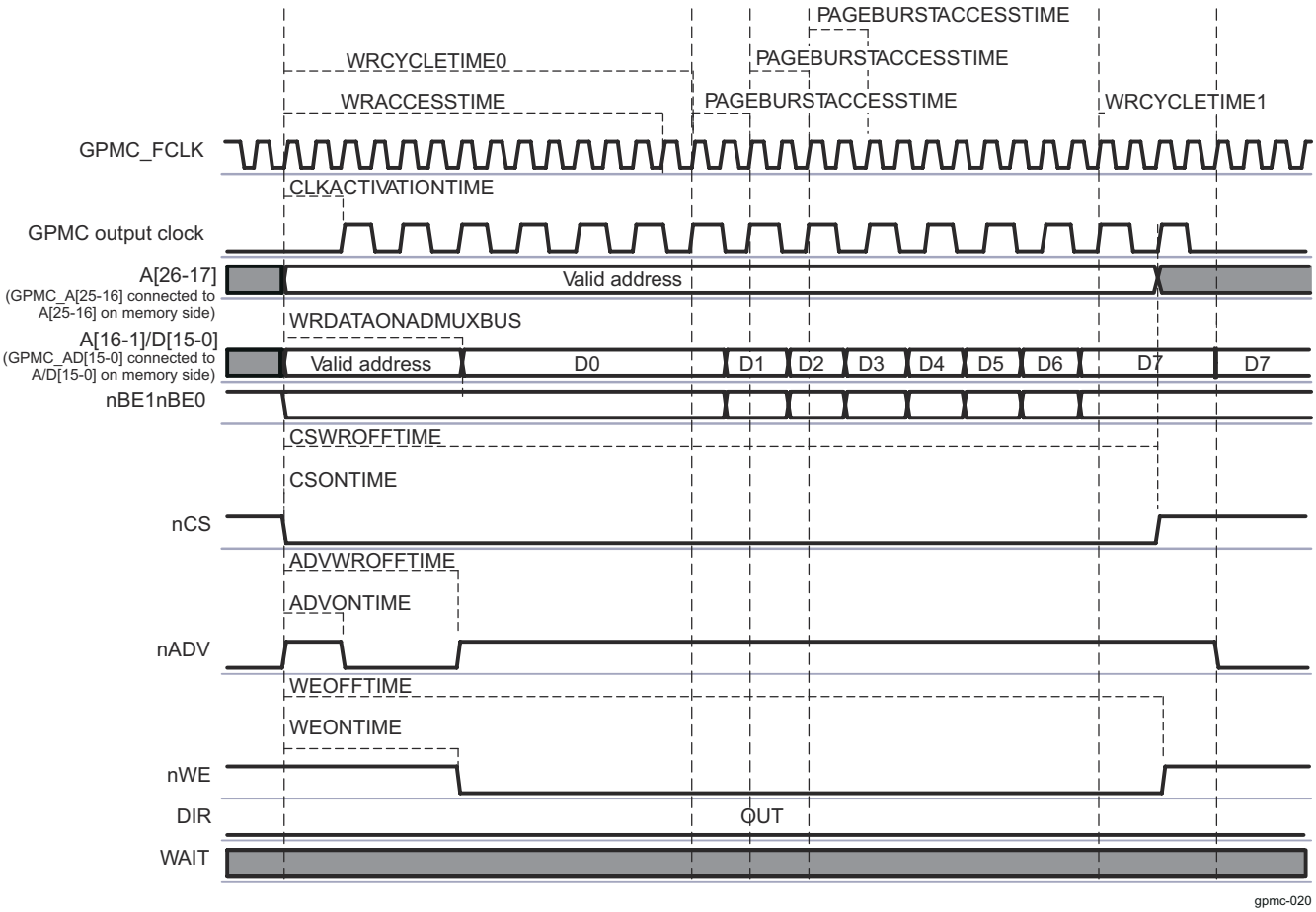


Figure 12-83. Synchronous Multiple Write (Burst Write) in Address/Data-Multiplexed Mode

Figure 12-84 shows the same synchronous burst write access when the chip-select is configured in address/address/data-multiplexed (AAD-multiplexed) mode.

Copyright © 2024 Texas Instruments Incorporated

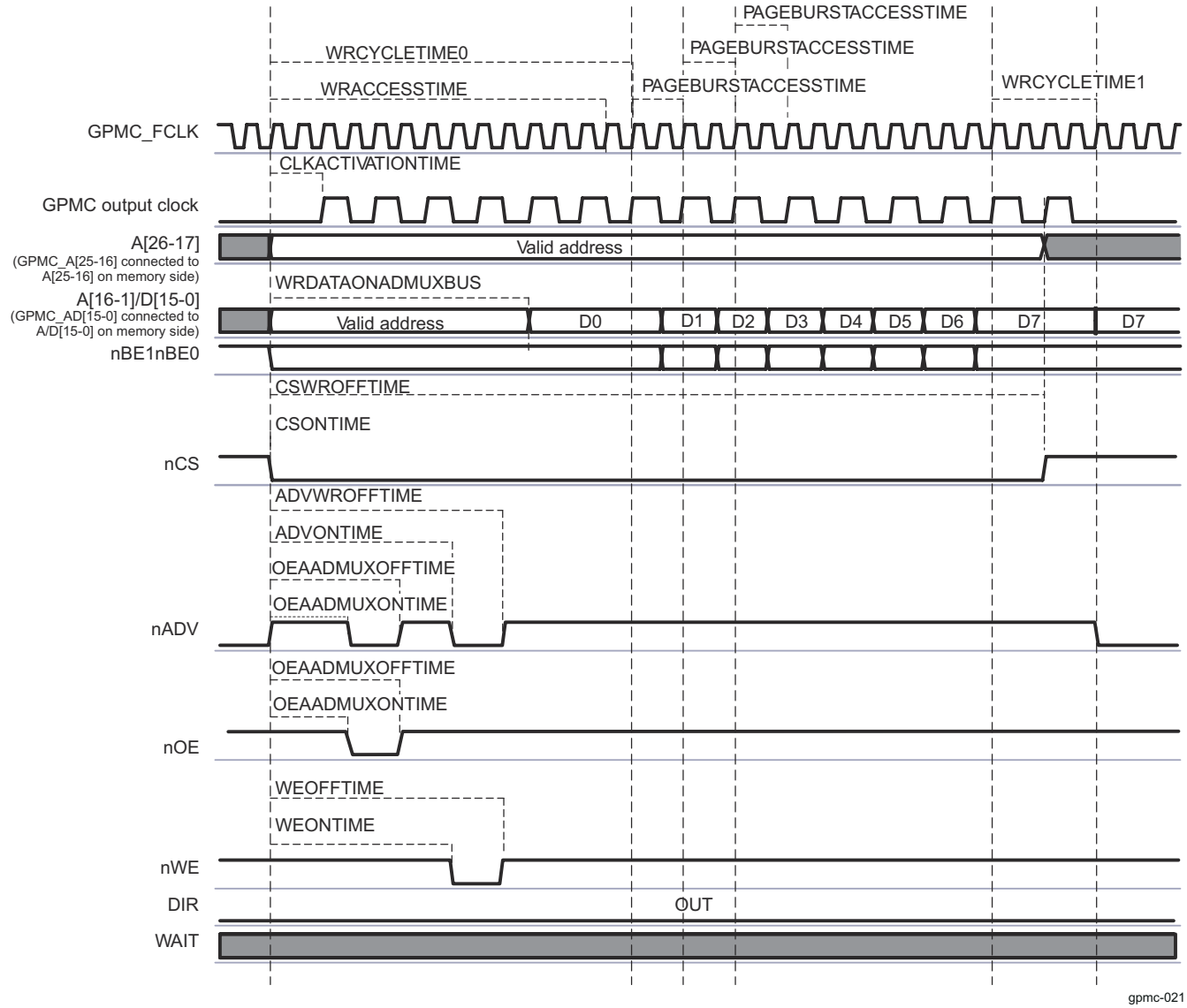


Figure 12-84. Synchronous Multiple Write (Burst Write) in Address/Address/Data-Multiplexed Mode

The first data of the burst is driven on the A/D bus at the GPMC_CONFIG6_i[19-16] WRDATAONADMUXBUS bit field.

When WRACCESSTIME completes, control-signal timings are frozen during the multiple data transactions, corresponding to the GPMC_CONFIG5_i[27-24] PAGEBURSTACCESSTIME bit field multiplied by the number of remaining data transactions.

When the GPMC generates a read access to an address/data-multiplexed device, it drives the address bus until nOE assertion time. For more information, see Section 12.3.4.3.7.2.3, *Address/Data-Multiplexing Interface*.

- Chip-select signal nCS:
  - nCS assertion time is controlled by the GPMC_CONFIG2_i[3-0] CSONTIME bit field (where i = 0 to 3) and ensures address setup time to nCS assertion.
  - nCS deassertion time controlled by the GPMC_CONFIG2_i[20-16] CSWROFFTIME bit field and ensures address hold time to nCS deassertion.
- Address valid signal nADV:
  - nADV assertion time is controlled by the GPMC_CONFIG3_i[3-0] ADVONTIME bit field.
  - nADV deassertion time is controlled by the GPMC_CONFIG3_i[20-16] ADVWROFFTIME bit field.
- Write enable signal nWE:

Copyright © 2024 Texas Instruments Incorporated

  - nWE assertion indicates a read cycle.
  - nWE assertion time is controlled by the GPMC_CONFIG4_i[19-16] WEONTIME bit field.
  - nWE deassertion time is controlled by the GPMC_CONFIG4_i[28-24] WEOFFTIME bit field.

Note

The nWE falling edge must not be used to control the time when the burst first data is driven in the address/data bus, because some new devices require the nWE signal to be low during the address phase.

- Direction signal DIR is OUT during the entire access.

When the GPMC generates a write access to an AAD-multiplexed device, all address bits are driven onto the address/data bus in two separate phases. The first phase is used for the MSB address and is qualified with nOE driven low. The second phase for LSB address is qualified with nOE driven high. The address phase ends at nWE assertion time.

The nCS, and DIR signals are controlled as previously described.

- Address valid signal nADV is asserted and deasserted twice during a read transaction:
  - nADV first assertion time is controlled by the GPMC_CONFIG3_i[6-4] ADVAADMUXONTIME bit field.
  - nADV first deassertion time is controlled by the GPMC_CONFIG3_i[26-24] ADVAADMUXRDOFFTIME bit field.
  - nADV second assertion time is controlled by the GPMC_CONFIG3_i[3-0] ADVONTIME bit field.
  - nADV second deassertion time is controlled by the GPMC_CONFIG3_i[12-8] ADVRDOFFTIME bit field.
- Output Enable signal nOE is asserted and deasserted twice during a read transaction (nOE second assertion indicates a read cycle):
  - nOE first assertion time is controlled by the GPMC_CONFIG4_i[6-4] OEAADMUXONTIME bit field.
  - nOE first deassertion time is controlled by the GPMC_CONFIG4_i[15-13] OEAADMUXOFFTIME bit field.
  - nOE second assertion time is controlled by the GPMC_CONFIG4_i[3-0] OEONTIME bit field.
  - nOE second deassertion time is controlled by the GPMC_CONFIG4_i[12-8] OEOFFTIME bit field.

First write data is driven by the GPMC at GPMC_CONFIG6_i[19-16] WRDATAONADMUXBUS, when in address/data-multiplexed configuration. The next write data of the burst is driven on the bus at WRACCESSTIME + 1 during GPMC_CONFIG5_i[27-24] PAGEBURSTACCESSTIME GPMC_FCLK cycles. The last data of the synchronous burst write is driven until GPMC_CONFIG5_i[12-8] WRCYCLETIME completes.

- WRACCESSTIME is defined in the GPMC_CONFIG6_i[28-24] bit field.
- The PAGEBURSTACCESSTIME parameter must be set accordingly with GPMCFCLKDIVIDER and the memory-device internal configuration.

Total access time GPMC_CONFIG5_i[12-8] WRCYCLETIME corresponds to WRACCESSTIME plus the address hold time from nCS deassertion. In Figure 12-83, the programmed value of WRCYCLETIME equals WRCYCLETIME0 + WRCYCLETIME1. WRCYCLETIME0 and WRCYCLETIME1 delays are not actual parameters and are only a graphical representation of the full WRCYCLETIME value.

After a write operation, if no other access (read or write) is pending, the data bus keeps the previous value. See Section 12.3.4.3.8.10, *Bus Keeping Support*.

12.3.4.3.9.3 Asynchronous and Synchronous Accesses in non-multiplexed Mode

Page mode is available only in non-multiplexed mode.

- Asynchronous single-read operation on a non-multiplexed device
- Asynchronous single-write operation on a non-multiplexed device
- Asynchronous multiple- (page mode) read operation on a non-multiplexed device
- Synchronous operations on a non-multiplexed device

12.3.4.3.9.3.1 Asynchronous Single-Read Operation on non-multiplexed Device

The address (For a 16-bit data memory device, hence GPMC A[0] is not necessary to be output) is driven onto the address bus and the 16-bit data is driven onto the data bus D[15-0].

Copyright © 2024 Texas Instruments Incorporated

Read data is latched at GPMC_CONFIG1_5[20-16] RDACCESSTIME completion time. The end of the access is defined by the GPMC_CONFIG1_5[4-0] RDCYCLETIME parameter.

The nCS, nADV, nOE, and DIR signals are controlled in the same way as address/data-multiplexed accesses (see Table 12-197, *NAND Memory Type*).

12.3.4.3.9.3.2 Asynchronous Single-Write Operation on non-multiplexed Device

The nCS, nADV, nWE, and DIR signals are controlled in the same way as address/data-multiplexed accesses (see Table 12-197).

12.3.4.3.9.3.3 Asynchronous Multiple (Page Mode) Read Operation on non-multiplexed Device

Figure 12-85 shows an asynchronous multiple-read operation on a non-multiplexed device in which two word32 host read accesses to the GPMC are split into one multiple- (page mode of 4 word16) read access to the attached device.

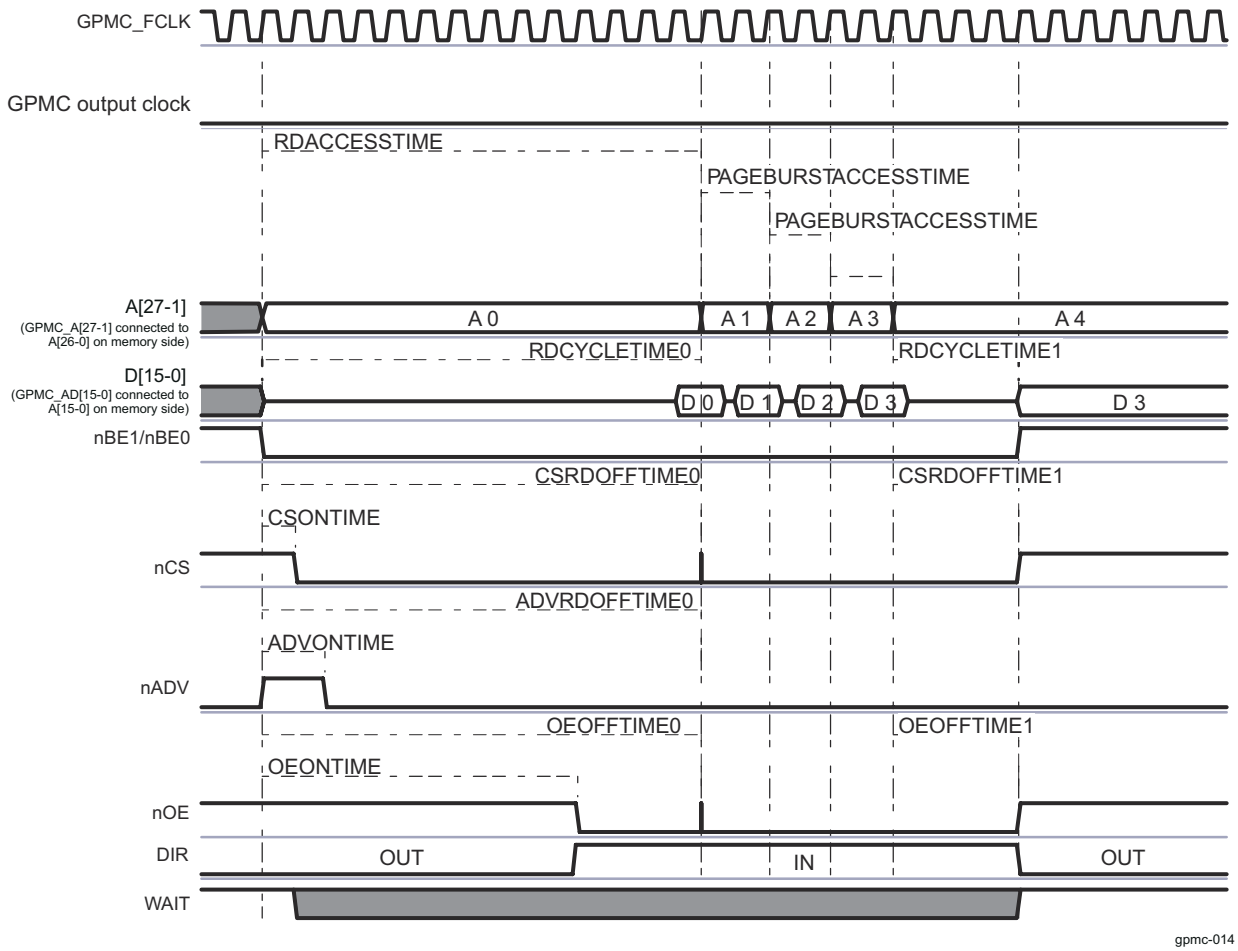


Figure 12-85. Asynchronous Multiple (Page Mode) Read

Note

The WAIT signal is active low.

The nCS, nADV, nOE, and DIR signals are controlled in the same way as address/data-multiplexed accesses (see Table 12-197).

When RDACCESSTIME completes, control signal timings are frozen during the multiple data transactions, corresponding to PAGEBURSTACCESSTIME multiplied by the number of remaining data transactions.

Copyright © 2024 Texas Instruments Incorporated

Read data is latched at GPMC_CONFIG5_i[20-16] RDACCESSTIME completion time (where i = 0 to 3). The end of the access is defined by the GPMC_CONFIG5_i[4-0] RDCYCLETIME parameter.

During consecutive accesses, the GPMC increments the address after each data read completes.

Delay between successive read data in the page is controlled by the GPMC_CONFIG5_i[27-24] PAGEBURSTACCESSTIME parameter. Depending on the device page length, the GPMC can control device page crossing during a burst request and insert initial RDACCESSTIME latency. Page crossing is possible only with a new burst access, meaning a new initial access phase is initiated.

Total access time RDCYCLETIME corresponds to RDACCESSTIME, plus the address hold time, starting from the nCS deassertion.

- The read cycle time is defined in the GPMC_CONFIG5_i[4-0] RDCYCLETIME bit field.
- In Figure 12-85, the programmed value of RDCYCLETIME equals RDCYCLETIME0 (before paged accesses) + RDCYCLETIME1 (after paged accesses).

12.3.4.3.9.3.4 Synchronous Operations on a non-multiplexed Device

All information for this section is equivalent to similar operations for address/data-multiplexed or AAD-multiplexed accesses. The only difference resides in the address phase. See Section 12.3.4.4.3, *GPMC Configuration in NOR Mode*.

12.3.4.3.9.4 Page and Burst Support

Each chip-select can be configured to process system single or burst requests into successive single accesses or asynchronous page/synchronous burst accesses, with appropriate access size adaptation.

Depending on the external device page or burst capability, read and write accesses can be independently configured through the GPMC. The GPMC_CONFIG1_i[30] READMULTIPLE and GPMC_CONFIG1_i[28] WRITMULTIPLE bits (where i = 0 to 3) are associated with the READTYPE and WRITETYPE parameters.

Note

- Asynchronous write page mode is not supported.
- 8-bit-wide device support is limited to nonburstable devices (READMULTIPLE and WRITEMULTIPLE are ignored).
- Not applicable to NAND device interfacing.

12.3.4.3.9.5 System Burst vs External Device Burst Support

The device system can issue the following requests to the GPMC:

- Byte, 16-bit word, 32-bit word requests (byte-enable-controlled). This is always a single request from the interconnect point of view.
- Incrementing fixed-length bursts of two, four, and eight words
- Wrapped (critical word access first) fixed-length burst of two, four, or eight words

To process a system request with the optimal protocol, the READMULTIPLE (and READTYPE) and WRITEMULTIPLE (and WRITETYPE) parameters must be set according to the burstable capability (synchronous or asynchronous) of the attached device.

The GPMC access engine issues only fixed-length bursts. The maximum length that can be issued is defined per chip-select by the GPMC_CONFIG1_i[24-23] ATTACHEDDEVICEPAGELENGTH bit field (where i = 0 to 3). When the value of ATTACHEDDEVICEPAGELENGTH is less than the length of the system burst request (including the appropriate access size adaptation according to the device width), the GPMC splits the system burst request into multiple bursts. Within the specified 4-, 8-, or 16-word value, the value of the ATTACHEDDEVICEPAGELENGTH bit field must correspond to the maximum length burst supported by the memory device configured in fixed-length burst mode (as opposed to continuous burst mode).

To get optimal performance from memory devices that natively support 16 Word16-length-wrapping burst capability (critical word access first), the ATTACHEDDEVICEPAGELENGTH parameter must be set to 16 words and the GPMC_CONFIG1_i[31] WRAPBURST bit (where i = 0 to 3) must be set to 1. Similarly

Copyright © 2024 Texas Instruments Incorporated

DEVICEPAGELENGTH is set to 4 and 8 for memories supporting 4 and 8 Word16-length-wrapping burst, respectively.

When the memory device does not offer (or is not configured to offer) native 16 Word16-length-wrapping burst, the WRAPBURST parameter must be cleared, and the GPMC access engine emulates the wrapping burst by issuing the appropriate burst sequences according to the value of ATTACHEDDEVICEPAGELENGTH.

When the memory device does not support native-wrapping burst, there is usually no difference in behavior between a fixed-burst length mode and a continuous-burst mode configuration (except for a potential power increase from a memory-speculative data prefetch in a continuous burst read). However, even though continuous burst mode is compatible with GPMC behavior, because the GPMC access engine issues only fixed-length burst and does not benefit from continuous burst mode, it is best to configure the memory device in fixed-length burst mode.

The memory device maximum-length burst (configured in fixed-length burst wrap or nonwrap mode) usually corresponds to the memory device data buffer size. Memory devices with a minimum of 16 half-word buffers are the most appropriate (especially with wrap support), but memory devices with smaller buffer size (4 or 8) are also supported, assuming that the GPMC_CONFIG1_i[24-23] ATTACHEDDEVICEPAGELENGTH bit field is set accordingly to 4 or 8 words.

The device system issues only requests with addresses or starting addresses for nonwrapping burst requests; that is, the request size boundary is aligned. In case of an eight-word-wrapping burst, the wrapping address always occurs on the eight-word boundary. As a consequence, all words requested must be available from the memory data buffer when the buffer size is equal to or greater than the value of ATTACHEDDEVICEPAGELENGTH. This usually means that data can be read from or written to the buffer at a constant rate (number of cycles between data) without wait-states between data accesses. If the memory does not behave this way (nonzero wait-state burstable memory), WAIT pin monitoring must be enabled to dynamically control data access completion within the burst.

Note

When the system burst request length is less than the value of ATTACHEDDEVICEPAGELENGTH, the GPMC proceeds with the required accesses.

12.3.4.3.10 GPMC pSRAM Access Specificities

pSRAM devices are SRAM-pin-compatible low-power memories that contain a self-refreshed DRAM memory array. The GPMC_CONFIG1_i[11-10] DEVICETYPE bit field (where i = 0 to 3) must be set to 0b00.

The pSRAM device uses the NOR protocol. It supports the following operations:

- Asynchronous single read
- Asynchronous page read
- Asynchronous single write
- Synchronous single read and write
- Synchronous burst read
- Synchronous burst write (not supported by NOR flash memory)

pSRAM devices must be powered up and initialized in a predefined manner according to the specifications of the attached device.

pSRAM devices can be programmed to use either mode: fixed or variable latency. pSRAM devices can automatically schedule autorefresh operations, which force the GPMC to use its WAIT signal capability when read or write operations occur during an internal self-refresh operation, or they can automatically include the autorefresh operation in the access time. These devices do not require additional WAIT signal capability or a minimum nCS high pulse width between consecutive accesses to ensure that the correct internal refresh operation is scheduled.

12.3.4.3.11 GPMC NAND Access Description

NAND (8-bit and 16-bit) memory devices using a standard NAND asynchronous address/data-multiplexing scheme can be supported on any chip-select with the appropriate asynchronous configuration settings.

Copyright © 2024 Texas Instruments Incorporated

As for any other type of memory compatible with the GPMC interface, accesses to a chip-select allocated to a NAND device can be interleaved with accesses to chip-selects allocated to other external devices. This interleaved capability limits the system to *chip enable don't care* NAND devices, because the chip-select allocated to the NAND device must be deasserted if accesses to other chip-selects are requested.

12.3.4.3.11.1 NAND Memory Device in Byte or 16-bit Word Stream Mode

NAND devices require correct command and address programming before data array read or write accesses. The GPMC does not include specific hardware to translate a random address system request into a NAND-specific multiphase access. In that sense, GPMC NAND support, as opposed to random memory-map device support, is data stream-oriented (byte or 16-bit word).

The GPMC NAND programming model depends on a software driver for address and command formatting with the correct data address pointer value according to the block and page structure. Because of NAND structure and protocol interface diversity, the GPMC does not support automatic command and address phase programming, and software drivers must access the NAND device ID to ensure that correct command and address formatting are used for the identified device.

NAND device data read and write accesses are achieved through an asynchronous read or write access. The associated chip-select signal timing control must be programmed according to the NAND device timing specification.

Any chip-select region can be qualified as a NAND region to constrain the nADV/ALE signal as ALE (ALE active high, default state value at low) during address program access, and the nBE0/CLE signal as CLE (CLE active high, default state value at low) during command program access. GPMC address lines are not used (the previous value is not changed) during NAND access.

12.3.4.3.11.1.1 Chip-Select Configuration for NAND Interfacing in Byte or Word Stream Mode

The GPMC_CONFIG7_i register (where i = 0 to 3) associated with a NAND device region interfaced in byte or word stream mode can be initialized with a minimum size of 16MB, because any address location in the chip-select memory region can be used to access a NAND data array. The NAND flash protocol specifies an address sequence where address bits are passed through the data bus in a series of write accesses with the ALE pin asserted. After this address phase, all operations are streamed and the system requests address is irrelevant.

CAUTION

To allow correct command, address, and data-access controls, the GPMC_CONFIG1_i register associated with a NAND device region must be initialized in asynchronous read and write modes with the parameters listed in Table 12-167. Failure to comply with these settings corrupts the NAND interface protocol.

Table 12-167. Chip-Select Configuration for NAND Interfacing

Bit Field	Register	Value	Comments
WRAPBURST	GPMC_CONFIG1_i[31] (1)	0	No wrap
READMULTIPLE	GPMC_CONFIG1_i[30]	0	Single access
READTYPE	GPMC_CONFIG1_i[29]	0	Asynchronous mode
WRITEMULTIPLE	GPMC_CONFIG1_i[28]	0	Single access
WRITETYPE	GPMC_CONFIG1_i[27]	0	Asynchronous mode
CLKACTIVATIONTIME	GPMC_CONFIG1_i[26-25]	0b00	
ATTACHEDDEVICEPAGELENGTH	GPMC_CONFIG1_i[24-23]	Don't care	Single-access mode
WAITREADMONITORING	GPMC_CONFIG1_i[22]	0	Wait not monitored by GPMC access engine
WAITWRITEMONITORING	GPMC_CONFIG1_i[21]	0	Wait not monitored by GPMC access engine
WAITMONITORINGTIME	GPMC_CONFIG1_i[19-18]	Don't care	Wait not monitored by GPMC access engine

Copyright © 2024 Texas Instruments Incorporated

Table 12-167. Chip-Select Configuration for NAND Interfacing (continued)

Bit Field	Register	Value	Comments
WAITPINSELECT	GPMC_CONFIG1_i[17-16]		Select which wait is monitored by edge detectors
DEVICESIZE	GPMC_CONFIG1_i[13-12]	0b00 or 0b01	8- or 16-bit interface
DEVICETYPE	GPMC_CONFIG1_i[11-10]	0b10	NAND device in stream mode
MUXADDDATA	GPMC_CONFIG1_i[9-8]	0b00	non-multiplexed mode
TIMEPARAGRANULARITY	GPMC_CONFIG1_i[4]	0	Timing achieved with best GPMC clock granularity
GPMCFCLKDIVIDER	GPMC_CONFIG1_i[1-0]	Don't care	Asynchronous mode

(1) i = 0 to 3

The GPMC_CONFIG1_i to GPMC_CONFIG4_i registers (where i = 0 to 3) associated with a NAND device region must be initialized with the correct control-signal timing value according to the NAND device timing parameters.

12.3.4.3.11.1.2 NAND Device Command and Address Phase Control

NAND devices require multiple address programming phases. The software driver must issue the correct number of command and address program accesses, according to the device command set and the device address-mapping scheme.

NAND device-command and address-phase programming is achieved through write requests to the GPMC_NAND_COMMAND_i and GPMC_NAND_ADDRESS_i register locations (where i = 0 to 3) with the correct command and address values. These locations are mapped in the associated chip-select register region. The associated chip-select signal timing control must be programmed according to the NAND device timing specification.

Command and address values are not latched during the access and cannot be read back at the register location.

- Only write accesses must be issued to these locations, but the GPMC does not discard any read access. Accessing a NAND device with nOE and CLE or ALE asserted (read access) can produce undefined results.
- Write accesses to the GPMC_NAND_COMMAND_i and GPMC_NAND_ADDRESS_i register locations must be posted for faster operations (where i = 0 to 3). The GPMC_CONFIG[0] NANDFORCEPOSTEDWRITE bit enables write accesses to these locations as posted, even if they are defined as nonposted.

A write buffer is used to store write transaction information before the external device is accessed:

- Up to eight consecutive posted write accesses can be accepted and stored in the write buffer.
- For nonposted write, the pipeline is one deep.
- An GPMC_STATUS[0] EMPTYWRITEBUFFERSTATUS bit stores the empty status of the write buffer.

The GPMC_NAND_COMMAND_i and GPMC_NAND_ADDRESS_i registers (where i = 0 to 3) are 32-bit word locations, which means any 32-or 16-bit word access is split into 4- or 2-byte accesses if an 8-bit-wide NAND device is attached. For multiple-command phase or multiple-address phase, the software driver can use 32- or 16-bit word access to these registers, but it must consider the splitting and little-endian ordering scheme. When only one byte command or address phase is required, only byte write access to the GPMC_NAND_COMMAND_i and GPMC_NAND_ADDRESS_i registers can be used, and any of the four byte locations of the registers is valid.

The same applies to a GPMC_NAND_COMMAND_i and a GPMC_NAND_ADDRESS_i (where i = 0 to 3) 32-bit word write access to a 16-bit-wide NAND device (split into two 16-bit word accesses). In the case of a 16-bit word write access, the MSByte of the 16-bit word value must be set according to the NAND device requirement (usually 0). Either 16-bit word location or any one of the four byte locations of the registers is valid.

Copyright © 2024 Texas Instruments Incorporated

12.3.4.3.11.1.3 Command Latch Cycle

Writing data at the GPMC_NAND_COMMAND_i location (where i = 0 to 3) places the data as the NAND command value on the bus, using a regular asynchronous write access.

- nCE is controlled by the CSONTIME and CSWROFFTIME timing parameters.
- CLE is controlled by the ADVONTIME and ADVWROFFTIME timing parameters.
- nWE is controlled by the WEONTIME and WEOFFTIME timing parameters.
- ALE and nRE (nOE) are maintained inactive.

Figure 12-86 shows the NAND command latch cycle.

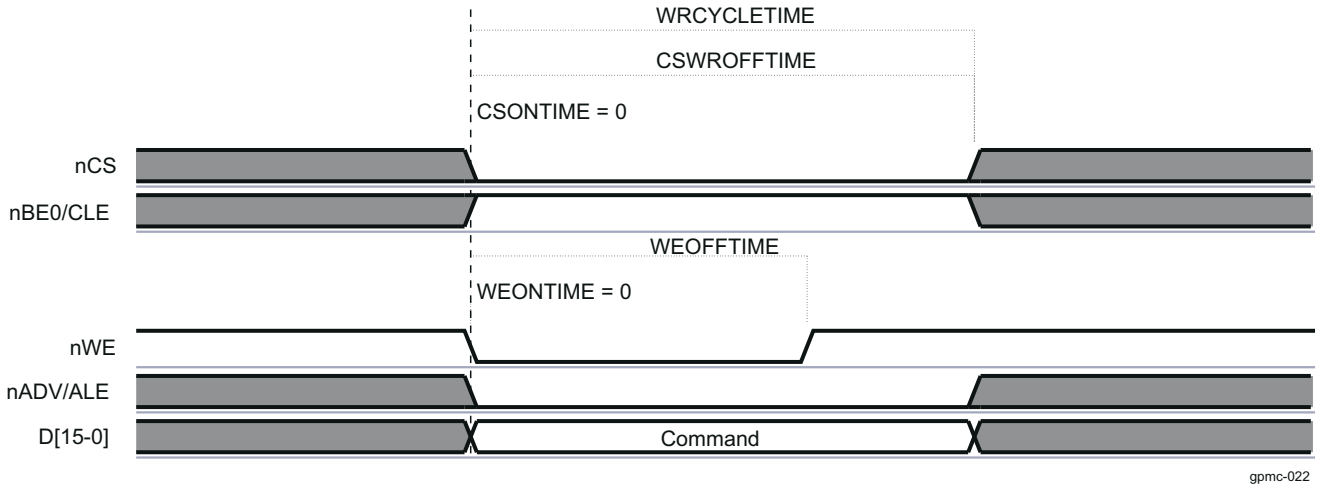


Figure 12-86. NAND Command Latch Cycle

Note

CLE is shared with the nBE0 output signal and has an inverted polarity from BE0. The NAND qualifier deals with this. During the asynchronous NAND data access cycle, nBE0 (also nBE1) must not toggle, because it is shared with CLE.

NAND flash memories do not use byte-enable signals.

12.3.4.3.11.1.4 Address Latch Cycle

Writing data at the GPMC_NAND_ADDRESS_i location (where i = 0 to 3) places the data as the NAND partial address value on the bus, using a regular asynchronous write access.

- nCS is controlled by the CSONTIME and CSWROFFTIME timing parameters.
- ALE is controlled by the ADVONTIME and ADVWROFFTIME timing parameters.
- nWE is controlled by the WEONTIME and WEOFFTIME timing parameters.
- CLE and nRE (nOE) are maintained inactive.

Figure 12-87 shows the NAND address latch cycle.

Copyright © 2024 Texas Instruments Incorporated

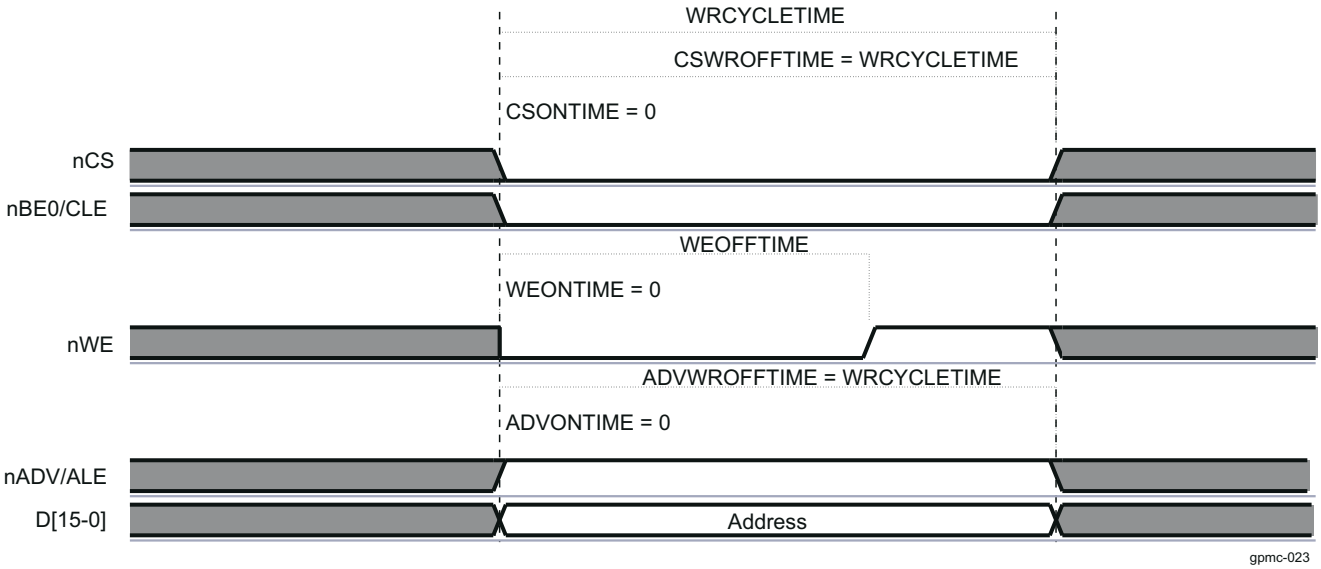


Figure 12-87. NAND Address Latch Cycle

Note

ALE is shared with the nADV output signal and has an inverted polarity from ADV. The NAND qualifier deals with this. During the asynchronous NAND data access cycle, ALE is kept stable.

12.3.4.3.11.1.5 NAND Device Data Read and Write Phase Control in Stream Mode

NAND device data read and write accesses are achieved through a read or write request to the chip-select-associated memory region at any address location in the region or through a read or write request to the GPMC_NAND_DATA_i location (where i = 0 to 3) mapped in the chip-select-associated control register region. GPMC_NAND_DATA_i is not a true register, but an address location to enable nRE or nWE signal control. The associated chip-select signal timing control must be programmed according to the NAND device timing specification.

Reading data from the GPMC_NAND_DATA_i location or from any location in the associated chip-select memory region activates an asynchronous read access.

- nCS is controlled by the CSONTIME and CSRDOFFTIME timing parameters.
- nRE is controlled by the OEONTIME and OEOFFTIME timing parameters.
- To take advantage of nRE high-to-data invalid minimum timing value, RDACCESSTIME can be set so that data are effectively captured after nRE deassertion. This allows optimization of NAND read access cycle time completion. For optimal timing parameter settings, see the NAND device and the device timing parameters.

ALE, CLE, and nWE are maintained inactive.

Figure 12-88 shows the NAND data read cycle.

Copyright © 2024 Texas Instruments Incorporated

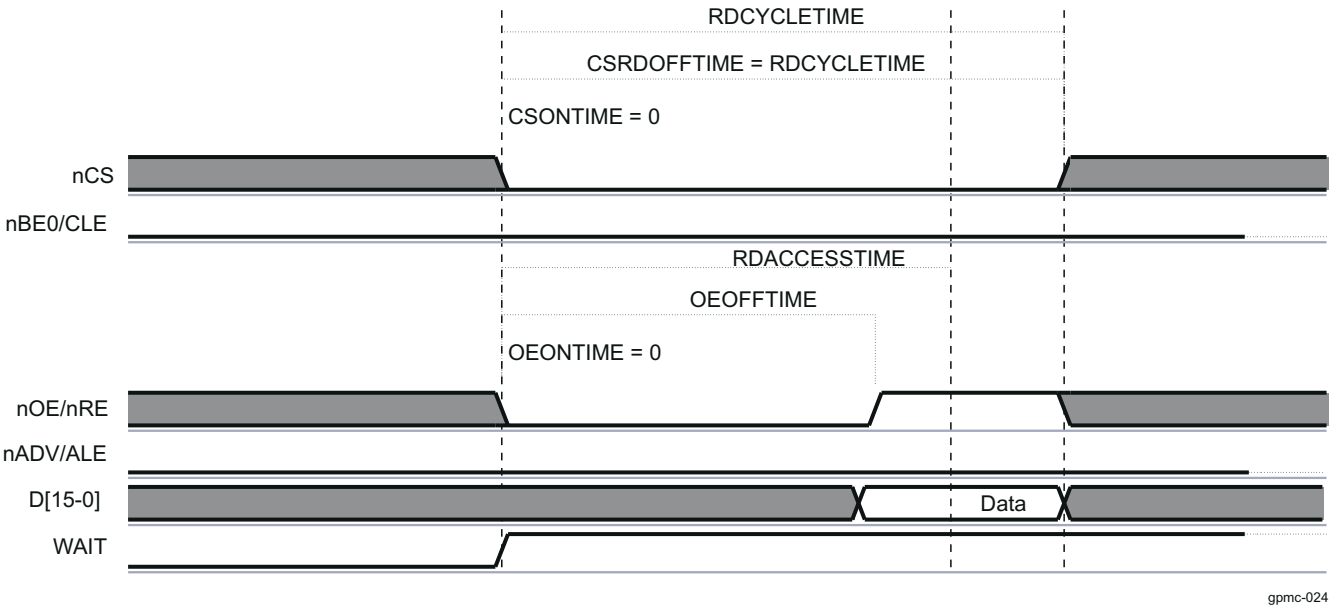


Figure 12-88. NAND Data Read Cycle

Writing data to the GPMC_NAND_DATA_i location or to any location in the associated chip-select memory region activates an asynchronous write access.

- nCS is controlled by the CSONTIME and CSWROFFTIME timing parameters.
- nWE is controlled by the WEONTIME and WEOFFTIME timing parameters.
- ALE, CLE, and nRE (nOE) are maintained inactive.

Figure 12-89 shows the NAND data write cycle.

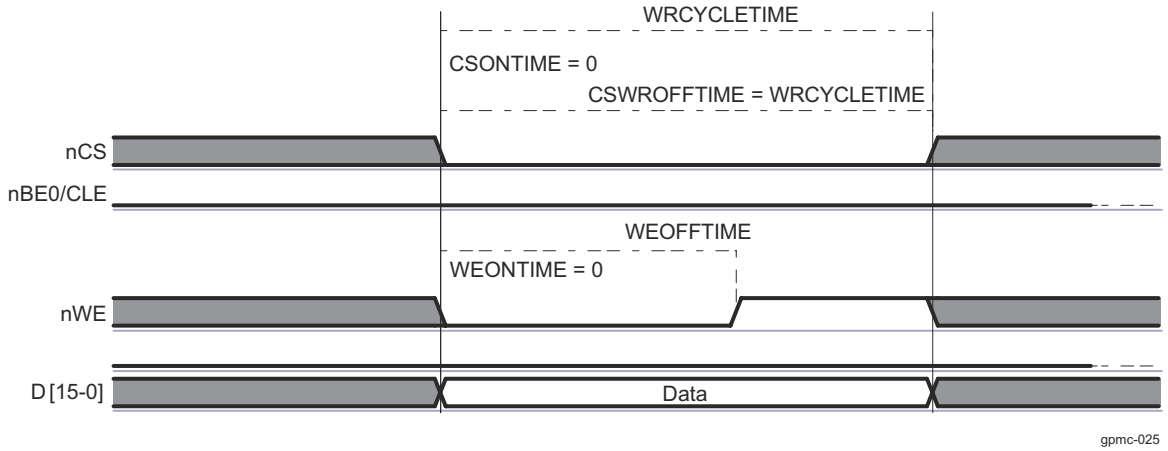


Figure 12-89. NAND Data Write Cycle

12.3.4.3.11.1.6 NAND Device General Chip-Select Timing Control Requirement

For most NAND devices, read data access time is dominated by nCS-to-data-valid timing and has faster nRE-to-data-valid timing. Successive accesses with nCS deassertions between accesses are affected by this timing constraint. Because accesses to a NAND device can be interleaved with other chip-select accesses, there is no certainty that nCS always stays low between two accesses to the same chip-select. Moreover, an nCS deassertion time between the same chip-select NAND accesses is likely to be required as follows: the nCS deassertion requires programming CYCLETIME and RDACCESSTIME according to the nCS-to-data-valid critical timing.

Copyright © 2024 Texas Instruments Incorporated

To get full performance from NAND read and write accesses, the prefetch engine can dynamically reduce the following on back-to-back NAND accesses (to the same memory) and suppress the minimum nCS high pulse width between accesses:

- RDCYCLETIME
- WRCYCLETIME
- RDACCESSTIME
- WRACCESSTIME
- CSRDOFFTIME
- CSWROFFTIME
- ADVRDOFFTIME
- ADVWROFFTIME
- OEOFFTIME
- WEOFFTIME

For more information about optimal prefetch engine access, see Section 12.3.4.3.11.4, *Prefetch and Write-Posting Engine*.

Some NAND devices require minimum write-to-read idle time, especially for device-status read accesses following status-read command programming (write access). If such write-to-read transactions are used, a minimum nCS high pulse width must be set. For this, CYCLE2CYCLESAMECSEN and CYCLE2CYCLEDELAY must be set according to the appropriate timing requirement to prevent any timing violation.

NAND devices usually have an important nRE high-to-data bus in three-state mode. This requires a bus turnaround setting (BUSTURNAROUND = 1) so that the next access to a different chip-select is delayed until the BUSTURNAROUND delay completes. Back-to-back NAND read accesses to the same NAND flash are not affected by the programmed bus turnaround delay.

12.3.4.3.11.1.7 Read and Write Access Size Adaptation

12.3.4.3.11.1.7.1 8-Bit-Wide NAND Device

Host 16- and 32-bit word read and write access requests to a chip-select associated with an 8-bit-wide NAND device are split into successive read and write byte accesses to the NAND memory device. Byte access is ordered according to little-endian organization. A NAND 8-bit-wide device must be interfaced on the D0D7 interface bus lane. GPMC data accesses are justified on this bus lane when the cs is associated with an 8-bit-wide NAND device.

12.3.4.3.11.1.7.2 16-Bit-Wide NAND Device

Host 32-bit word read and write access requests to a chip-select associated with a 16-bit-wide NAND device are split into successive read and write 16-bit word accesses to the NAND memory device. 16-bit word access is ordered according to little-endian organization.

Host byte read and write access requests to a 16-bit-wide NAND device are completed as 16-bit accesses on the device itself, because there is no byte-addressing capability on 16-bit-wide NAND devices. This means that the NAND device address pointer is incremented on a 16-bit word basis and not on a byte basis. For a read access, only the requested byte is given back to the host, but the remaining byte is not stored or saved by the GPMC, and the next byte or 16-bit word read access gets the next 16-bit word NAND location. For a write access, the invalid byte part of the 16-bit word is driven to FF, and the next byte or 16-bit word write access programs the next 16-bit word NAND location.

Generally, byte access to a 16-bit-wide NAND device must be avoided, especially when ECC calculation is enabled. 8- or 16-bit ECC-based computations are corrupted by a byte read to a 16-bit-wide NAND device, because the nonrequested byte is considered invalid on a read access (not captured on the external data bus; FF is fed to the ECC engine) and is set to FF on a write access.

Host requests (read/write) issued in the chip-select memory region are translated in successive single or split accesses (read/write) to the attached device. Therefore, incrementing 32-bit burst requests are translated in multiple 32-bit sequential accesses following the access adaptation of the 32-bit to 8- or 16-bit device.

Copyright © 2024 Texas Instruments Incorporated

12.3.4.3.11.2 NAND Device-Ready Pin

The NAND memory device provides a ready pin to indicate data availability after a block/page opening and to indicate that data programming is complete. The ready pin can be connected to one of the wait GPMC input pins; data read accesses must not be tried when the ready pin is sampled inactive (device is not ready) even if the associated chip-select WAITREADMONITORING bit field is set. The duration of the NAND device busy state after the block/page opening is so long (up to 50 micro second) that accesses occurring when the ready pin is sampled inactive can stall GPMC access and eventually cause a system time-out.

Note

If a read access to a NAND flash is done using WAIT monitoring mode, the device is blocked during a page opening, and so is the GPMC. If the correct settings are used, other chip-selects can be used while the memory processes the page opening command.

To avoid a time-out caused by a block/page opening delay in NAND flash, disable the WAIT pin monitoring for read and write accesses (that is, set the GPMC_CONFIG1_i[21] WAITWRITEMONITORING and GPMC_CONFIG1_i[22] WAITREADMONITORING bits to 0, where i = 0 to 3), and use one of the following methods instead:

- Use software to poll the WAITxSTATUS bit (where x = 0 to 3) of the GPMC_STATUS.
- Configure an interrupt that is generated on the WAIT signal change (through the GPMC_IRQENABLE register bits[11-8]).

Even if the READWAITMONITORING bit is not set, the external memory nR/B pin status is captured in the programmed wait bit in the GPMC_STATUS register.

The READWAITMONITORING bit method must be used for other memories than NAND flash, if they require the use of a WAIT signal.

12.3.4.3.11.2.1 Ready Pin Monitored by Software Polling

The ready signal state can be monitored through the GPMC_STATUS WAITxSTATUS bit (where x = 0 to 3). Software must monitor the ready pin only when the signal is declared valid. Refer to the NAND device timing parameters to set the correct software temporization to monitor ready only after the invalid window is complete from the last read command written to the NAND device.

12.3.4.3.11.2.2 Ready Pin Monitored by Hardware Interrupt

Each GPMC_WAIT input pin can generate an interrupt when a wait-to-no-wait transition is detected. Depending on whether the GPMC_CONFIG WAITxPINPOLARITY bits (where x = 0 to 3) is active low or active high, the wait-to-no-wait transition is a low-to-high external WAIT signal transition or a high-to-low external WAIT signal transition, respectively.

The wait transition pin detector must be cleared before any transition detection. This is done by writing 1 to the WAITxEDGEDETECTIONSTATUS bit (where x = 0 to 3) of the GPMC_IRQSTATUS register according to the GPMC_WAIT pin used for the NAND device-ready signal monitoring. To detect a wait-to-no-wait transition, the transition detector requires a wait active time detection of a minimum of two GPMC_FCLK cycles. Software must incorporate precautions to clear the wait transition pin detector before wait (busy) time completes.

A wait-to-no-wait transition detection can issue a GPMC interrupt if the WAITxEDGEDETECTIONENABLE bit in the GPMC_IRQENABLE register is set and if the WAITxEDGEDETECTIONSTATUS bit field in the GPMC_IRQSTATUS register is set.

The WAITMONITORINGTIME bit field does not affect wait-to-no-wait transition time detection.

It is also possible to poll the WAITxEDGEDETECTIONSTATUS bit field in the GPMC_IRQSTATUS register according to the GPMC_WAIT pin used for NAND device ready signal monitoring.

12.3.4.3.11.3 ECC Calculator

The GPMC includes an error code correction (ECC) calculator circuitry that enables ECC calculation on the fly during data read or data program (that is, write) operations. The page size supported by the ECC calculator in one calculation/context is 512 bytes.

Copyright © 2024 Texas Instruments Incorporated

The user can choose from two different algorithms with different error correction capabilities through the GPMC_ECC_CONFIG[16] ECCALGORITHM bit:

1. Hamming code for 1-bit error code correction on 8- or 16-bit NAND flash organized with page size greater than 512 bytes
2. Bose-Chaudhurl-Hocquenghem (BCH) code for 4- to 16-bit error correction

The GPMC does not handle the error code correction directly. During writes, the GPMC computes parity bits. During reads, the GPMC provides enough information for the processor to correct errors without reading the data buffer all over again.

The Hamming code ECC is based on a 2-dimensional (2D) (row and column) bit parity accumulation. This parity accumulation is accomplished on the programmed number of bytes or 16-bit words read from the memory device, or is written to the memory device in stream mode.

Because the ECC engine includes only one accumulation context, it can be allocated to only one chip-select at a time through the GPMC_ECC_CONFIG[3-1] ECCCS bit field. Even if two chip-selects use different ECC algorithms, one the Hamming code and the other a BCH code, they must define separate ECC contexts because some of the ECC registers are common to all types of algorithms.

12.3.4.3.11.3.1 Hamming Code

All references to ECC in this subsection refer to the 1-bit error correction Hamming code.

The ECC is based on a 2D (row and column) bit parity accumulation known as the Hamming code. The parity accumulation is done for a programmed number of bytes or 16-bit word read from the memory device or written to the memory device in stream mode.

There is no automatic error detection or correction, and the software NAND driver must read the multiple ECC calculation results, compare them to the expected code value, and take the appropriate corrective actions according to the error handling strategy (ECC storage in spare byte, error correction on read, block invalidation).

The ECC engine includes a single accumulation context. It can be allocated to a single designated chip-select at a time, and parallel computations on different chip-selects are not possible. Because it is allocated to a single chip-select, the ECC computation is not affected by interleaved GPMC accesses to other chip-selects and devices. The ECC accumulation is sequentially processed in the order of data read from or written to the memory on the designated chip-select. The ECC engine does not differentiate read accesses from write accesses and does not differentiate data from command or status information. Software must ensure that only relevant data are passed to the NAND flash memory while the ECC computation engine is active.

The starting NAND page location must be programmed first, followed by an ECC accumulation context reset with an ECC enabling, if required. The NAND device accesses discussed in the following sections must be limited to data read or write until the specified number of ECC calculations is complete.

12.3.4.3.11.3.1.1 ECC Result Register and ECC Computation Accumulation Size

The GPMC includes up to nine ECC result registers (GPMC_ECCj_RESULT, where j = 1 to 9) to store ECC computation results when the specified number of bytes or 16-bit words has been computed.

The ECC result registers are used sequentially: one ECC result is stored in one ECC result register on the list, the next ECC result is stored in the next ECC result register on the list, and so forth, until the last ECC computation. The value of the GPMC_ECCj_RESULT register is valid only when the programmed number of bytes or 16-bit words has been accumulated, which means that the same number of bytes or 16-bit words has been read from or written to the NAND device in sequence.

The GPMC_ECC_CONTROL[3-0] ECCPOINTER bit field must be set to the correct value to select the ECC result register to be used first in the list for the incoming ECC computation process. The ECCPointer can be read to determine which ECC register is used in the next ECC result storage for the ongoing ECC computation. The value of the GPMC_ECCj_RESULT register (where j = 1 to 9) can be considered valid when ECCPOINTER equals j + 1. When the GPMC_ECCj_RESULT register (where j = 9) is updated, ECCPOINTER is frozen at 10, and ECC computing is stopped (ECCENABLE = 0).

Copyright © 2024 Texas Instruments Incorporated

The ECC accumulator must be reset before any ECC computation accumulation process. The GPMC_ECC_CONTROL[8] ECCCLEAR bit must be set to 1 (nonpersistent bit) to clear the accumulator and all ECC result registers.

For each ECC result (each GPMC_ECCj_RESULT register, where j = 1 to 9), the number of bytes or 16-bit words used for ECC computing accumulation can be selected from between two programmable values.

The ECCjRESULTSIZE bits (where j = 1 to 9) in the GPMC_ECC_SIZE_CONFIG register select which programmable size value (ECCSIZE0 or ECCSIZE1) must be used for this ECC result (stored in the GPMC_ECCj_RESULT register).

The ECCSIZE0 and ECCSIZE1 bit fields allow selection of the number of bytes or 16-bit words used for ECC computation accumulation. Any even values from 2 to 512 are allowed.

Flexibility in the number of ECCs computed and the number of bytes or 16-bit words used in the successive ECC computations enables different NAND page error-correction strategies. Usually based on 256 or 512 bytes and on 128 or 256 16-bit word, the number of ECC results required is a function of the NAND device page size. Specific ECC accumulation size can be used when computing the ECC on the NAND spare byte.

For example, with a 2-KB data page, 8-bit-wide NAND device, eight ECCs accumulated on 256 bytes can be computed and added to one extra ECC computed on the 24 spare bytes area where the eight ECC results used for comparison and correction with the computed data page ECC are stored. The GPMC then provides nine GPMC_ECCj_RESULT registers (j = 1 to 9) to store the results. In this case, ECCSIZE0 is set to 256, and ECCSIZE1 is set to 24; the ECC[1-8]RESULTSIZE bits are set to 0, and the ECC9RESULTSIZE bit is set to 1.

12.3.4.3.11.3.1.2 ECC Enabling

The GPMC_ECC_CONFIG[3-1] ECCCS bit field selects the allocated chip-select. The GPMC_ECC_CONFIG[0] ECCENABLE bit enables ECC computation on the next detected read or write access to the selected chip-select.

The following fields must not be changed or cleared while an ECC computation is in progress:

- CCPOINTER
- ECCCLEAR
- ECCSIZE
- ECCjRESULTSIZE (where j = 1 to 9)
- ECC16B
- ECCCS

The ECC accumulator and ECC result register must not be changed or cleared while an ECC computation is in progress.

Table 12-168 describes the ECC enable settings.

Table 12-168. ECC Enable Settings

Bit Field	Register	Value	Comments
ECCCS	GPMC_ECC_CONFIG	0–3	Selects the chip-select where ECC is computed
ECC16B	GPMC_ECC_CONFIG	0/1	Selects column number for ECC calculation
ECCCLEAR	GPMC_ECC_CONTROL	0–7	Clears all ECC result registers
ECCPOINTER	GPMC_ECC_CONTROL	0–7	A write to this bit field selects the ECC result register where the first ECC computation is stored. Set to 1 by default.
ECCSIZE1	GPMC_ECC_SIZE_CONFIG	0x00–0xFF	Defines ECCSIZE1
ECCSIZE0	GPMC_ECC_SIZE_CONFIG	0x00–0xFF	Defines ECCSIZE0
ECCjRESULTSIZE (j from 1 to 9)	GPMC_ECC_SIZE_CONFIG	0/1	Selects the size of ECCn result register
ECCENABLE	GPMC_ECC_CONFIG	1	Enables the ECC computation

Copyright © 2024 Texas Instruments Incorporated

12.3.4.3.11.3.1.3 ECC Computation

The ECC algorithm is a multiple parity bit accumulation computed on the odd and even bit streams extracted from the byte or Word16 streams. The parity accumulation is split into row and column accumulations, as shown in Figure 12-90 and Figure 12-91. The intermediate row and column parities are used to compute the upper level row and column parities. Only the final computation of each parity bit is used for ECC comparison and correction.

P1o = bit7 XOR bit5 XOR bit3 XOR bit1 on each byte of the data stream
P1e = bit6 XOR bit4 XOR bit2 XOR bit0 on each byte of the data stream

P2o = bit7 XOR bit6 XOR bit3 XOR bit2 on each byte of the data stream
P2e = bit5 XOR bit4 XOR bit1 XOR bit0 on each byte of the data stream

P4o = bit7 XOR bit6 XOR bit5 XOR bit4 on each byte of the data stream
P4e = bit3 XOR bit2 XOR bit1 XOR bit0 on each byte of the data stream

Each column parity bit is XORed with the previous accumulated value.

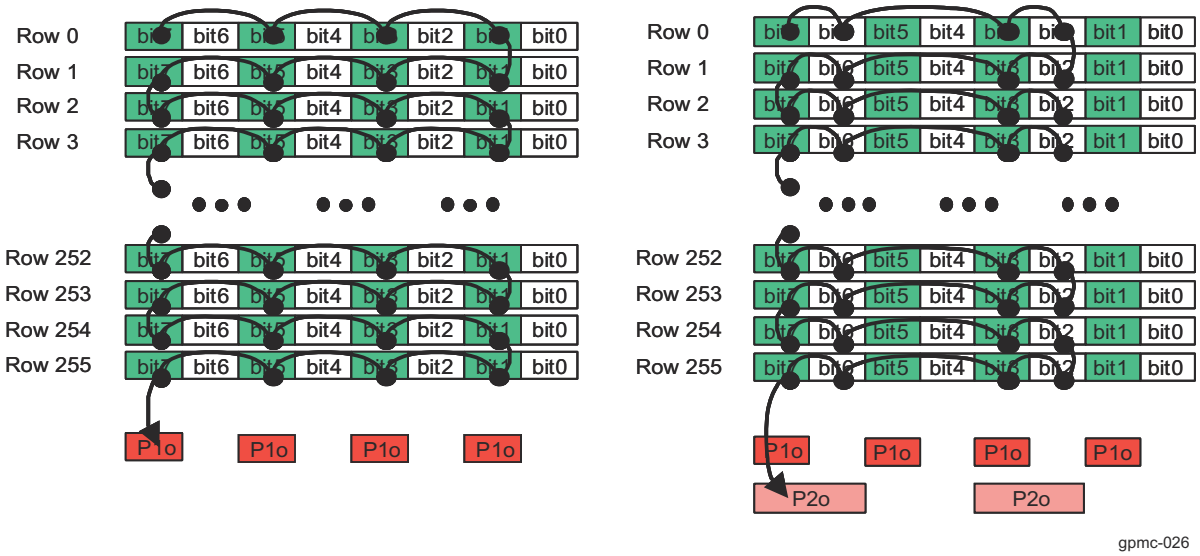


Figure 12-90. Hamming Code Accumulation Algorithm (1/2)

For line parities, the bits of each new data are XORed together, and line parity bits are computed as described below:

P8e = row0 XOR row2 XOR row4 XOR ... XOR row254
P8o = row1 XOR row3 XOR row5 XOR ... XOR row255

P16e = row0 XOR row1 XOR row4 XOR row5 XOR ... XOR row252 XOR row 253
P16o = row2 XOR row3 XOR row6 XOR row7 XOR ... XOR row254 XOR row 255

Copyright © 2024 Texas Instruments Incorporated

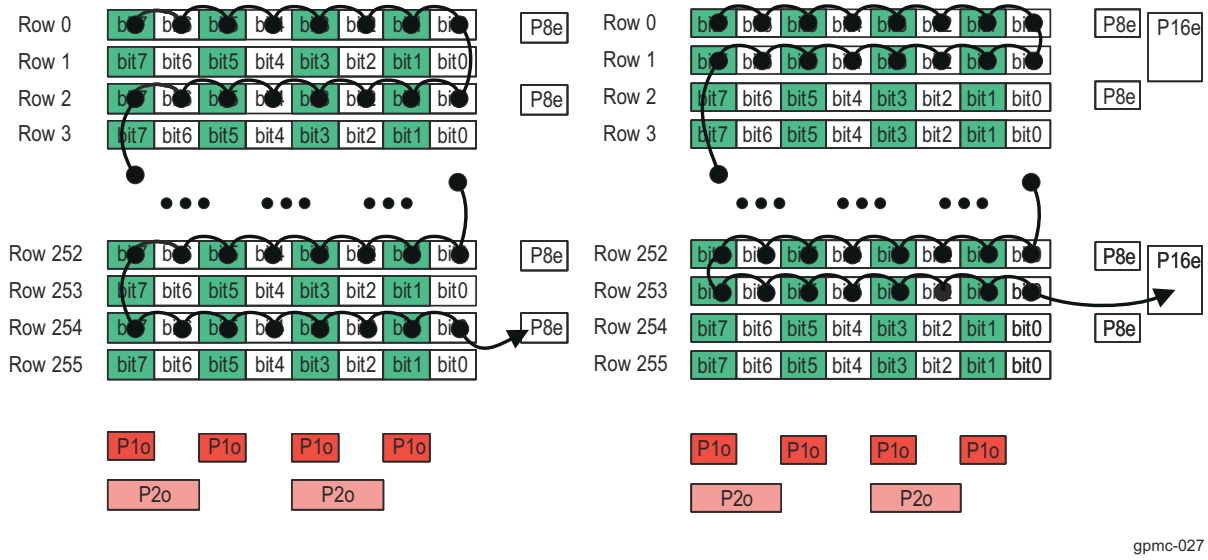


Figure 12-91. Hamming Code Accumulation Algorithm (2/2)

Unused parity bits in the result registers are set to 0.

Figure 12-92 shows ECC computation for a 256-byte data stream (read or write). The result includes six column parity bits (P1o-P2o-P4o for odd parities, and P1e-P2e-P4e for even parities) and sixteen row parity bits (P8o-P16o-P32o--P1024o for odd parities, and P8e-P16e-P32e--P1024e for even parities).

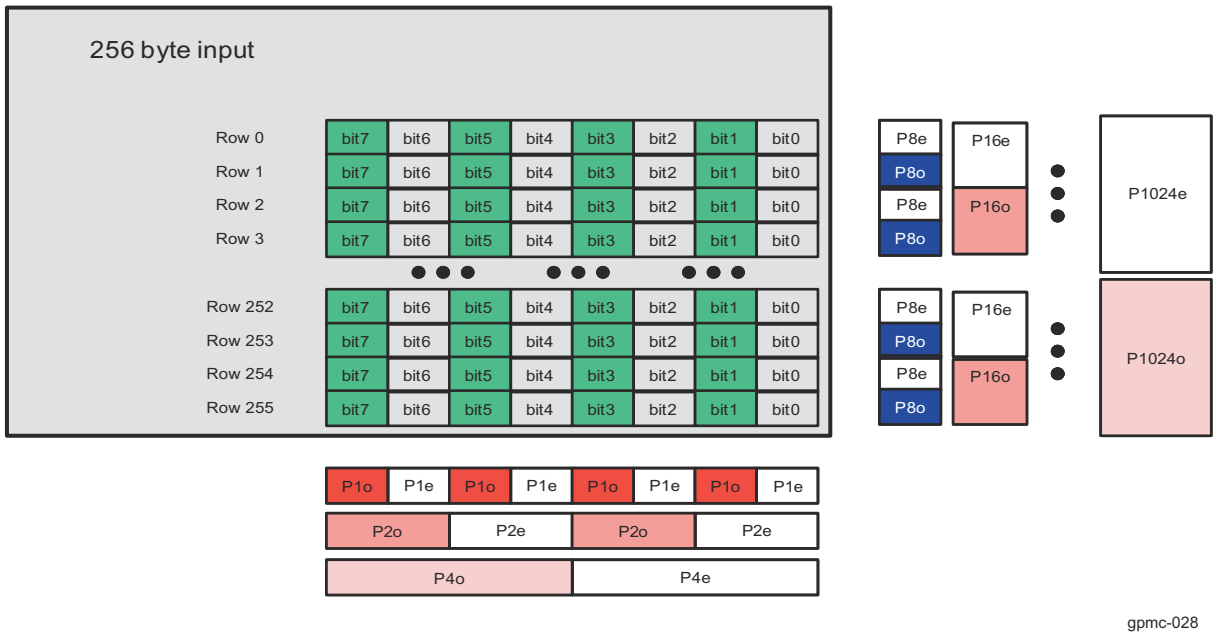


Figure 12-92. ECC Computation for a 256-Byte Data Stream (Read or Write)

Figure 12-93 shows ECC computation for a 512-byte data stream (read or write). The result includes six column parity bits (P1o-P2o-P4o for odd parities, and P1e-P2e-P4e for even parities) and eighteen row parity bits (P8o-P16o-P32o--P1024o- - P2048o for odd parities, and P8e-P16e-P32e--P1024e- P2048e for even parities).

Copyright © 2024 Texas Instruments Incorporated

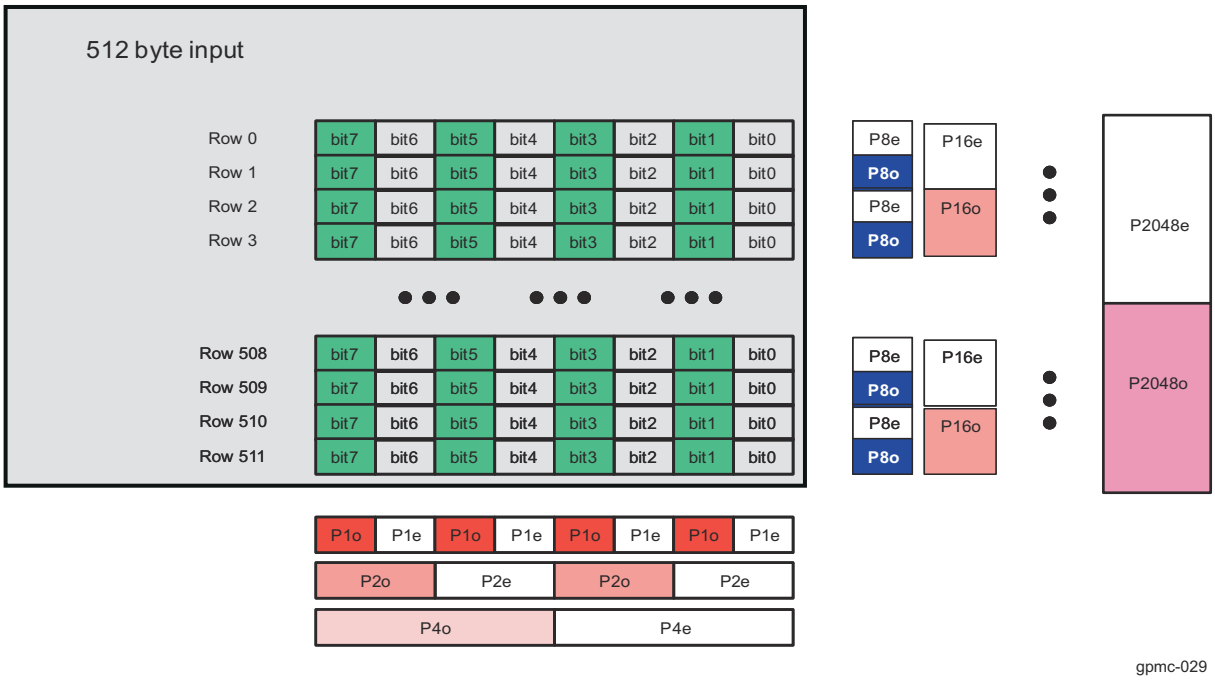


Figure 12-93. ECC Computation for a 512-Byte Data Stream (Read or Write)

For a 2-KB page, four 512 bytes ECC calculations plus 1 for the spare area are required. Results are stored in the GPMC_ECCj_RESULT registers (where j = 1 to 9).

12.3.4.3.11.3.1.4 ECC Comparison and Correction

To detect an error, the computed ECC result must be XORed with the parity value stored in the spare area of the accessed page.

- If the result of this logical XOR is all 0s, no error is detected and the read data is correct.
- If every second bit in the parity result is a 1, 1 bit is corrupted and is located at bit address (P2048o, P1024o, P512o, P256o, P128o, P64o, P32o, P16o, P8o, P4o, P2o, P1o). Software must correct the corresponding bit.
- If only 1 bit in the parity result is 1, it is an ECC error and the read data is correct.

12.3.4.3.11.3.1.5 ECC Calculation Based on 8-Bit Word

The 8-bit-based ECC computation is used for 8-bit-wide NAND device interfacing.

The 8-bit-based ECC computation can be used for 16-bit-wide NAND device interfacing to get backward compatibility on the error-handling strategy used with 8-bit-wide NAND devices. In this case, the 16-bit-wide data read from or written to the NAND device is fragmented into 2 bytes. According to little-endian access, the LSB of the 16-bit-wide data is ordered first in the byte stream used for 8-bit-based ECC computation.

12.3.4.3.11.3.1.6 ECC Calculation Based on 16-Bit Word

ECC computation based on an 16-bit word is used for 16-bit-wide NAND device interfacing. This ECC computation is not supported when interfacing an 8-bit-wide NAND device, and the GPMC_ECC_CONFIG[7] ECC16B bit must be set to 0 when interfacing an 8-bit-wide NAND device.

The parity computation based on 16-bit words affects the row and column parity mapping. The main difference is that the odd and even parity bits P8o and P8e are computed on rows for an 8-bit-based ECC and on columns for a 16-bit based ECC. Figure 12-94 and Figure 12-95 show a 128 Word16 ECC computation scheme and a 256 16-bit word ECC computation scheme.

Copyright © 2024 Texas Instruments Incorporated

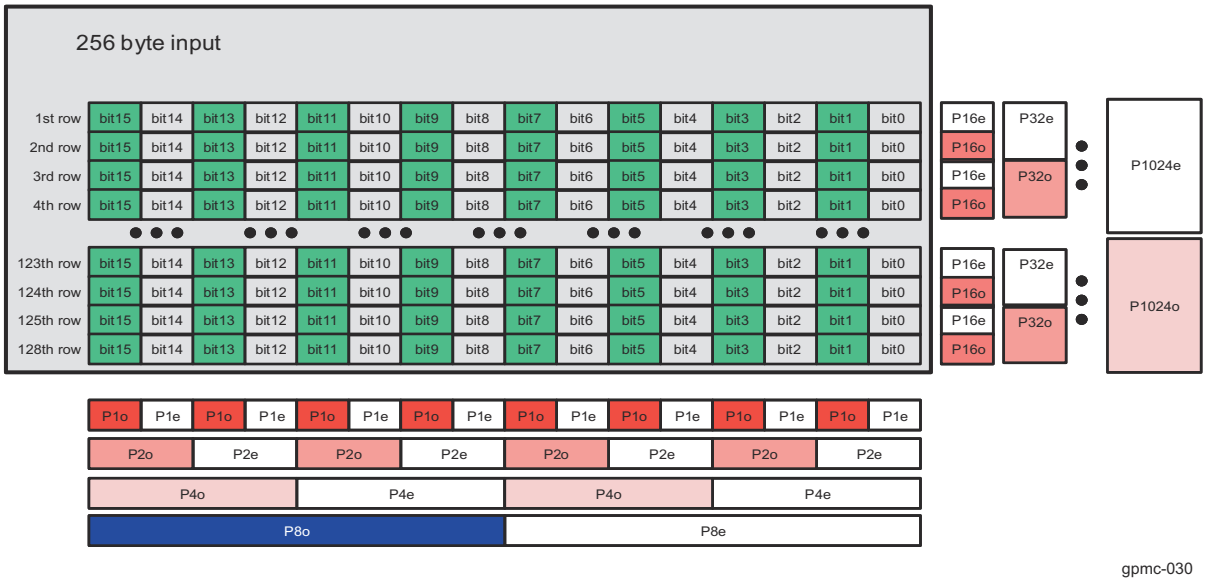


Figure 12-94. 128 Word16 ECC Computation

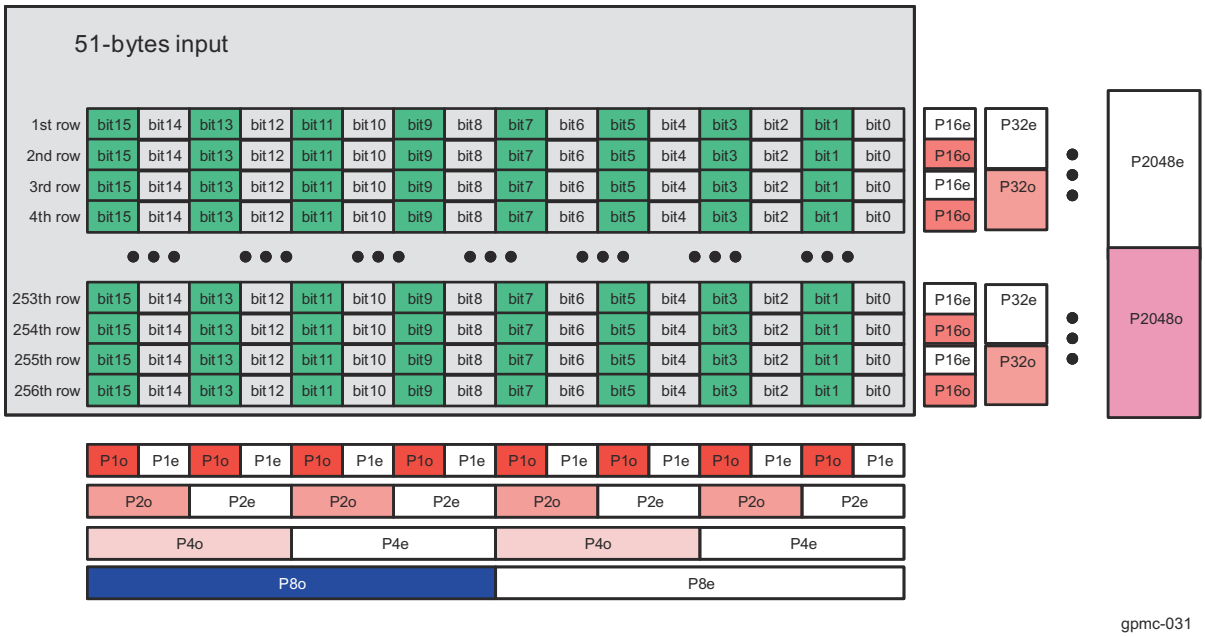


Figure 12-95. 256 Word16 ECC Computation

12.3.4.3.11.3.2 BCH Code

All references to ECC in this subsection refer to the 4- to 16-bit error correction BCH code.

12.3.4.3.11.3.2.1 Requirements

1. Read and write accesses to a NAND flash take place by whole pages, in a predetermined sequence: first the data byte page itself, and then some spare bytes, including the BCH ECC (and other information). The NAND device can cache a full page, including spares, for read and write accesses.
   - Typical page write sequence:
     - Sequential write to NAND cache of main data plus spare data, for a page. ECC is calculated on the fly. Calculated ECC can be inserted on the fly in the spares or replaced by dummy accesses.
     - When the calculated ECC is replaced by dummy accesses, it must be written to the cache in a second, separate phase. The ECC module is disabled during that time.
     - NAND writes its cache line (page) to the array.

Copyright © 2024 Texas Instruments Incorporated

- Typical page read sequence:
  - Sequential read of a page. ECC is calculated on the fly.
  - The status of the ECC module buffers determines the presence of errors.

2. Accesses to several memories can be interleaved by the GPMC, but only one of those memories at a time can be a NAND using the BCH engine; in other words, only one BCH calculation (for example, for a single page) can be ongoing at any time. The sequential nature of NAND accesses ensures that the data is always written or read out in the same order. BCH-relevant accesses are selected by the chip-selects of the GPMC.
3. Each page can hold up to 4KB of data, spare bytes not included. This means up to 8 × 512-byte BCH messages. Because all the data is written or read out first, followed by the BCH ECC, the BCH engine must be able to hold eight 104-bit remainders or syndromes (or smaller, 52-bit ones) at the same time.

   The BCH module can store all remainders internally. After the page start, an internal counter is used to detect the 512-byte sector boundaries. On those boundaries, the current remainder is stored and the divider reset for the next calculation. At the end of the page, the BCH module contains all remainders.
4. NAND access cycles hold 8 or 16 bits of data each (1 or 2 bytes); Each NAND cycle takes at least four cycles of the GPMC internal clock. This means the NAND flash timing parameters must define a RDCYCLETIME and a WRCYCLETIME of at least four clock cycles after optimization when using the BCH calculator.
5. The spare area is assumed to be large enough to hold the BCH ECC; that is, to have a message of at least 13 bytes available per 512-byte sector of data. The zone of unused spare area by the ECC may or may not be protected by the same ECC scheme, by extending the BCH message beyond 512 bytes (the maximum codeword is 1023 bytes long, ECC included, which leaves much space to cover spare bytes).

12.3.4.3.11.3.2.2 Memory Mapping of BCH Codeword

BCH encoding considers a block of data to protect as a polynomial message M(x). In a standard case, 512 bytes of data (that is, 2^{12} bits = 4096 bits) are seen as a polynomial of degree $2^{12} – 1 = 4095$, with parameters ranging from M0 to M4095. For 512 bytes of data, 52 bits are required for 4-bit error correction, 104 bits are required for 8-bit error correction, and 207 bits are required for 16-bit error correction. The ECC is a remainder polynomial R(x) of degree 103 (or 51, depending on the selected mode). The complete codeword C(x) is the concatenation of M(x) and R(x), as described in Table 12-169.

Table 12-169. Flattened BCH Codeword Mapping (512 Bytes + 104 Bits)

	Message M(x)			ECC R(x)		
Bit Number	M4095	...	M0	R103	...	R0

If the message is extended by the addition of spare bytes to be protected by the same ECC, the principle is still valid. For example, a 3-byte extension of the message gives a polynomial message M(x) of degree ((512 + 3) × 8) – 1 = 4119, for a total of 3 + 13 = 16 spare bytes of spare, all protected as part of the same codeword.

The message and the ECC bits are manipulated and mapped in the GPMC byte-oriented system. The ECC bits are stored in the following registers (where i = 0 to 3):

- GPMC_BCH_RESULT0_i
- GPMC_BCH_RESULT1_i
- GPMC_BCH_RESULT2_i
- GPMC_BCH_RESULT3_i

12.3.4.3.11.3.2.2.1 Memory Mapping of Data Message

The data message mapping must follow the following rules:

- Bit endianness within a byte is little-endian; that is, the bytes LSB is also the lowest-degree polynomial parameter: a byte b7-b0 (with b0 the LSB) represents a segment of polynomial $b7 * x^{(7+i)} + b6 * x^{(6+i)} + ... + b0 * x^{i}$
- The message is mapped in the NAND starting with the highest-order parameters, that is, in the lowest addresses of a NAND page.
- Byte endianness within the 16-bit words in the NAND is big-endian (that is, the same message mapped in 8- and 16-bit memories has the same content at the same byte address).

Copyright © 2024 Texas Instruments Incorporated

Note

The BCH module has no visibility over actual addresses. The most important point is the sequence of data words the BCH sees. However, the NAND page is always scanned incrementally in read and write accesses, which produces the mapping patterns described in the following.

Table 12-170 and Table 12-171 describe the mapping of the same 512-byte vector (typically, a BCH message) in the NAND memory space. The byte address is only an offset module 512 (0x200), because the same page may contain several contiguous 512-byte sectors (BCH blocks). The LSB and MSB are, respectively, the bits M0 and M(2^{12}–1) of the codeword mapping discussed previously. In both cases the data vectors are aligned; that is, their boundaries coincide with the RAM data word boundaries.

Table 12-170. Aligned Message Byte Mapping in 8-Bit NAND

Byte Offset	8-Bit Word
0x000	(MSB) Byte 511 (0x1FF)
0x001	Byte 510 (0x1FE)
...	...
0x1FF	Byte 0 (0x0) (LSB)

Table 12-171. Aligned Message Byte Mapping in 16-Bit NAND

Byte Offset	16-Bit Word MSB	16-Bit Word LSB
0x000	Byte 510 (0x1FE)	(MSB) Byte 511 (0x1FF)
0x002	Byte 508 (0x1FC)	Byte 509 (0x1FD)
...	...	...
0x1FE	Byte 0 (0x0)	(LSB) Byte 1 (0x1)

Table 12-172 through Table 12-177 list the mapping in memory of arbitrarily-sized messages, starting on access (byte or 16-bit word) boundaries for more clarity. Note that message may actually start and stop on arbitrary nibbles. A nibble is a 4-bit entity. The unused nibbles are not discarded, and they can still be used by the BCH module, but as part of the next message section (for example, on the ECC of another sector).

Table 12-172. Aligned Nibble Mapping of Message in 8-Bit NAND

Byte Offset	8-Bit Word	
	4-Bit Most Significant Nibble	4-Bit Least Significant Nibble
1	(MSB) Nibble S-1	Nibble S-2
2	Nibble S-3	Nibble S-4
...	...	...
S/2 – 2	Nibble 3	Nibble 2
S/2 – 1	Nibble 1	Nibble 0 (LSB)

Table 12-173. Misaligned Nibble Mapping of Message in 8-Bit NAND

Byte Offset	8-Bit Word	
	4-Bit Most Significant Nibble	4-Bit Least Significant Nibble
1	(MSB) Nibble S-1	Nibble S-2
2	Nibble S-3	Nibble S-4
...	...	...
(S + 1) / 2 – 2	Nibble 2	Nibble 1
(S + 1) / 2 – 1	Nibble 0 (LSB)	

Table 12-174. Aligned Nibble Mapping of Message in 16-Bit NAND

Byte Offset	16-Bit Word			
	4-Bit Most Significant Nibble		4-Bit Least Significant Nibble	
0	Nibble S-3	Nibble S-4	(MSB) Nibble S-1	Nibble S-2
2	Nibble S-7	Nibble S-8	Nibble S-5	Nibble S-6

Copyright © 2024 Texas Instruments Incorporated

Table 12-174. Aligned Nibble Mapping of Message in 16-Bit NAND (continued)

Byte Offset	16-Bit Word			
...	...		...	
S/2 – 4	Nibble 5	Nibble 4	Nibble 7	Nibble 6
S/2 – 2	Nibble 1	Nibble 0 (LSB)	Nibble 3	Nibble 2

Table 12-175. Misaligned Nibble Mapping of Message in 16-Bit NAND (1 Unused Nibble)

Byte Offset	16-Bit Word			
	4-Bit Most Significant Nibble		4-Bit Least Significant Nibble	
0	Nibble S-3	Nibble S-4	(MSB) Nibble S-1	Nibble S-2
2	Nibble S-7	Nibble S-8	Nibble S-5	Nibble S-6
...	...		...	
(S + 1) / 2 – 4	Nibble 4	Nibble 3	Nibble 6	Nibble 5
(S + 1) / 2 – 2	Nibble 0 (LSB)		Nibble 2	Nibble 1

Table 12-176. Misaligned Nibble Mapping of Message in 16-Bit NAND (2 Unused Nibbles)

Byte Offset	16-Bit Word			
	4-Bit Most Significant Nibble		4-Bit Least Significant Nibble	
0	Nibble S-3	Nibble S-4	(MSB) Nibble S-1	Nibble S-2
2	Nibble S-7	Nibble S-8	Nibble S-5	Nibble S-6
...	...		...	
(S + 2) / 2 – 4	Nibble 3	Nibble 2	Nibble 5	Nibble 4
(S + 2) / 2 – 2			Nibble 1	Nibble 0 (LSB)

Table 12-177. Misaligned Nibble Mapping of Message in 16-Bit NAND (3 Unused Nibbles)

Byte Offset	16-Bit Word			
	4-Bit Most Significant Nibble		4-Bit Least Significant Nibble	
0	Nibble S-3	Nibble S-4	(MSB) Nibble S-1	Nibble S-2
2	Nibble S-7	Nibble S-8	Nibble S-5	Nibble S-6
...	...		...	
(S + 3) / 2 – 4	Nibble 2	Nibble 1	Nibble 4	Nibble 3
(S + 3) / 2 – 2			Nibble 0 (LSB)	

Many other cases exist than those given in the previous tables; for example, where the message does not start on a word boundary.

12.3.4.3.11.3.2.2.2 Memory-Mapping of the ECC

The ECC (or remainder) is presented by the BCH module as a single 104-bit (or 52-bit), little-endian vector. Software must fetch those 13 bytes (or 6 bytes) from the module interface and then store them to the spare area (page write) in the NAND or to an intermediate buffer for comparison with the stored ECC (page read). There are no constraints on the ECC mapping inside the spare area: it is software-controlled.

It is advised, however, to maintain a coherence in the respective formats of the message or the ECC remainder once they have been read out of the NAND. The error correction algorithm works from the complete codeword (concatenated message and remainder) once an error is detected. The creation of this codeword must be made as straightforward as possible.

There are cases in which the same NAND access contains both data and the ECC protecting that data. This is the case when the data/ECC boundary (which can be on any nibble) does not coincide with an access boundary. The ECC is calculated on the fly following the write. In that case, the write must also contain part of the ECC because it is impossible to insert the ECC on the fly. Instead:

- During the initial page write (BCH encoding), the ECC is replaced by dummy bits. The BCH encoder is by definition turned OFF during the ECC section, so the BCH result is unmodified.
- During a second phase, the ECC is written to the correct location, next to the actual data.

Copyright © 2024 Texas Instruments Incorporated

- The completed line buffer is then written to the NAND array.

12.3.4.3.11.3.2.2.3 Wrapping Modes

For a given wrapping mode, the module automatically goes through a specific number of sections as data is being fed into the module. For each section, the BCH core can be enabled (in which case the data is fed to the BCH divider) or not (in which case the BCH simply counts to the end of the section). When enabled, the data is added to the ongoing calculation for a given sector number (for example, number 0).

Wrapping modes are described as follows. To better understand and see the real-life read and write sequences implemented with each mode, see Section 12.3.4.3.11.3.2.3, *Supported NAND Page Mappings and ECC Schemes*.

For each mode:

- A sequence describes the mode in pseudo language, with, for each section, the size and the buffer used for ECC processing (if ON). The programmable lengths are size, size0, and size1.
- A checksum condition is given. If the checksum condition is not respected for a given mode, the module behavior is unpredictable. S is the number of sectors in the page; size0 and size1 are the section sizes programmed for the mode, in nibbles.

Wrapping modes 8 through 11 insert a 1-nibble padding where the BCH processing is OFF. This is intended for t = 4 ECC, where ECC is 6 bytes long and the ECC area is expected to include (at least) 1 unused nibble to remain byte-aligned.

12.3.4.3.11.3.2.2.3.1 Manual Mode (0x0)

This mode is intended for short sequences, added manually to a given buffer through the software data port input. A complete page may be built out of several such sequences.

To process an arbitrary sequence of 4-bit nibbles, accesses to the software data port, containing the appropriate data, must be made. If the sequence end does not coincide with an access boundary (for example, to process 5 nibbles = 20 bits in 16-bit access mode) and those nibbles must be skipped, a number of unused nibbles must be programmed in GPMC_ECC_SIZE_CONFIG[31-22] ECCSIZE1 In the same example, 5 nibbles to process + 3 to discard = 8 nibbles = 2 × 16-bit accesses. Software must set:

- GPMC_ECC_SIZE_CONFIG[21-12] ECCSIZE0 = 0x5
- GPMC_ECC_SIZE_CONFIG[31-22] ECCSIZE1 = 0x3

Note

In the following figures, size and size0 are the same parameter.

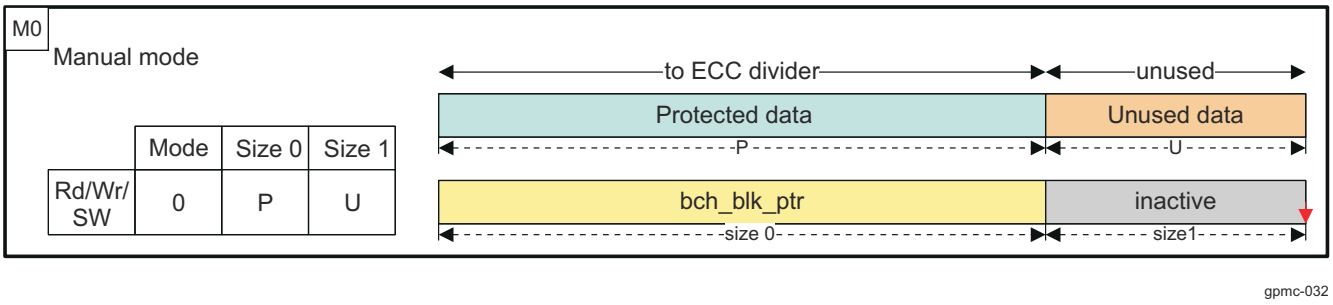


Figure 12-96. Manual Mode Sequence and Mapping

Section processing sequence:

- One time with buffer
  - size0 nibbles of data, processing ON
  - size1 nibbles of unused data, processing OFF

Checksum: size0 + size1 nibbles must fit in a whole number of accesses.

Copyright © 2024 Texas Instruments Incorporated

In the following sections, S is the number of sectors in the page.

12.3.4.3.11.3.2.2.3.2 Mode 0x1

Page processing sequence:

- Repeat with buffer 0 to S-1
  - 512-byte data, processing ON
- Repeat with buffer 0 to S-1
  - size0 nibbles spare, processing ON
  - size1 nibbles spare, processing OFF

Checksum: Spare area size (nibbles) = S – (size0 + size1)

12.3.4.3.11.3.2.2.3.3 Mode 0xA (10)

Page processing sequence:

- Repeat with buffer 0 to S-1
  - 512-byte data, processing ON
- Repeat with buffer 0 to S-1
  - size0 nibbles spare, processing ON
  - 1 nibble pad spare, processing OFF
  - size1 nibbles spare, processing OFF

Checksum: Spare area size (nibbles) = S – (size0 + 1 + size1)

12.3.4.3.11.3.2.2.3.4 Mode 0x2

Page processing sequence:

- Repeat with buffer 0 to S-1
  - 512-byte data, processing ON
- Repeat with buffer 0 to S-1
  - size0 nibbles spare, processing OFF
  - size1 nibbles spare, processing ON

Checksum: Spare area size (nibbles) = S – (size0 + size1)

12.3.4.3.11.3.2.2.3.5 Mode 0x3

Page processing sequence:

- Repeat with buffer 0 to S-1
  - 512-byte data, processing ON
- One time with buffer 0
  - size0 nibbles spare, processing ON
- Repeat with buffer 0 to S-1
  - size1 nibbles spare, processing ON

Checksum: Spare area size (nibbles) = size0 + (S – size1)

12.3.4.3.11.3.2.2.3.6 Mode 0x7

Page processing sequence:

- Repeat with buffer 0 to S-1
  - 512-byte data, processing ON
- One time with buffer 0
  - size0 nibbles spare, processing ON
- Repeat S times (no buffer used)
  - size1 nibbles spare, processing OFF

Checksum: Spare area size (nibbles) = size0 + (S – size1)

Copyright © 2024 Texas Instruments Incorporated

12.3.4.3.11.3.2.2.3.7 Mode 0x8

Page processing sequence:

- Repeat with buffer 0 to S-1
  - 512-byte data, processing ON
- One time with buffer 0
  - size0 nibbles spare, processing ON
- Repeat with buffer 0 to S-1
  - 1 nibble padding spare, processing OFF
  - size1 nibbles spare, processing ON

Checksum: Spare area size (nibbles) = size0 + (S – (1 + size1))

12.3.4.3.11.3.2.2.3.8 Mode 0x4

Page processing sequence:

- Repeat with buffer 0 to S-1
  - 512-byte data, processing ON
- One time (no buffer used)
  - size0 nibbles spare, processing OFF
- Repeat with buffer 0 to S-1
  - size1 nibbles spare, processing ON

Checksum: Spare area size (nibbles) = size0 + (S – size1)

12.3.4.3.11.3.2.2.3.9 Mode 0x9

Page processing sequence:

- Repeat with buffer 0 to S-1
  - 512-byte data, processing ON
- One time (no buffer used)
  - size0 nibbles spare, processing OFF
- Repeat with buffer 0 to S-1
  - 1 nibble padding spare, processing OFF
  - size1 nibbles spare, processing ON

Checksum: Spare area size (nibbles) = size0 + (S – (1 + size1))

12.3.4.3.11.3.2.2.3.10 Mode 0x5

Page processing sequence:

- Repeat with buffer 0 to S-1
  - 512-byte data, processing ON
- Repeat with buffer 0 to S-1
  - size0 nibbles spare, processing ON
- Repeat with buffer 0 to S-1
  - size1 nibbles spare, processing ON

Checksum: Spare area size (nibbles) = S – (size0 + size1)

12.3.4.3.11.3.2.2.3.11 Mode 0xB (11)

Page processing sequence:

- Repeat with buffer 0 to S-1
  - 512-byte data, processing ON
- Repeat with buffer 0 to S-1
  - size0 nibbles spare, processing ON
- Repeat with buffer 0 to S-1
  - 1 nibble padding spare, processing OFF

Copyright © 2024 Texas Instruments Incorporated

  - size1 nibbles spare, processing ON

Checksum: Spare area size (nibbles) = S – (size0 + 1 + size1)

12.3.4.3.11.3.2.2.3.12 Mode 0x6

Page processing sequence:

- Repeat with buffer 0 to S-1
  - 512-byte data, processing ON
- Repeat with buffer 0 to S-1
  - size0 nibbles spare, processing ON
- Repeat S times (no buffer used)
  - size1 nibbles spare, processing OFF

Checksum: Spare area size (nibbles) = S – (size0 + size1)

12.3.4.3.11.3.2.3 Supported NAND Page Mappings and ECC Schemes

The following rules apply to the entire mapping description:

- Main data area (sectors) size is hardcoded to 512 bytes.
- Spare area size is programmable.
- All page sections (of main area data bytes, protected spare bytes, unprotected spare bytes, and ECC) are defined as explained in Section 3.4.3.11.3.2.2.1, *Memory Mapping of Data Message*.

Each of the following sections gives a NAND page mapping example (per-sector spare mappings, pooled spare mapping, per-sector spare mapping, with ECC separated at the end of the page).

In the mapping diagrams, sections that belong to the same BCH codeword have the same color (blue or green); unprotected sections are not covered (orange) by the BCH scheme.

Below each mapping diagram, a write (encoding) and read (decoding: syndrome generation) sequence is given, with the number of the active buffers at each point in time (yellow). In the inactive zones (grey), no computing is taking place but the data counter is still active.

In Figure 12-97 through Figure 12-99, the tables on the left summarize the mode, size0, and size1 parameters to program for, respectively, write and read processing of a page, with the given mapping, where:

- P is the size of spare byte section Protected by the ECC (in nibbles)
- U is the size of spare byte section Unprotected by the ECC (in nibbles)
- E is the size of the ECC (in nibbles)
- S is the number of Sectors per page (two in the current diagrams)

Each time the processing of a BCH block is complete (ECC calculation for write/encoding, syndrome generation for read/decoding, indicated by red arrows), the update pointer is pulsed. The processing for block 0 can be the first or the last to complete, depending on the NAND page mapping and operation (read or write). All examples show a page size of 1 KB + spares; that is, S = 2 sectors of 512 bytes. The same principles can be extended to larger pages by adding more sectors.

The actual BCH codeword size is used during the error location work to restrict the search range: by definition, errors can happen only in the codeword that was actually written to the NAND, not in the mathematical codeword of $n = 2^{13} – 1 = 8191$ bits; that codeword (higher-order bits) is all-zero and implicit during computations.

The actual BCH codeword size depends on the mode, the programmed sizes, and the sector number (all sizes in nibbles):

- Spares mapped and protected per sector (below: see M1-M2-M3-M9-M10):
  - All sectors: (512) + P + E
- Spares pooled and protected by sector 0 (below: see M5-M6):
  - Sector 0 codeword: (512) + P + E
  - Other sectors: (512) + E
- Unprotected spares (below: see M4-M7-M8-M11-M12):
  - All codewords (512) + E

Copyright © 2024 Texas Instruments Incorporated

12.3.4.3.11.3.2.3.1 Per-Sector Spare Mappings

In these schemes, each 512-byte sector of the main area has its own dedicated section of the spare area. The spare area of each sector is composed of:

- ECC, which must be located after the data it protects
- Other data, which may or may not be protected by the ECC for its sector.

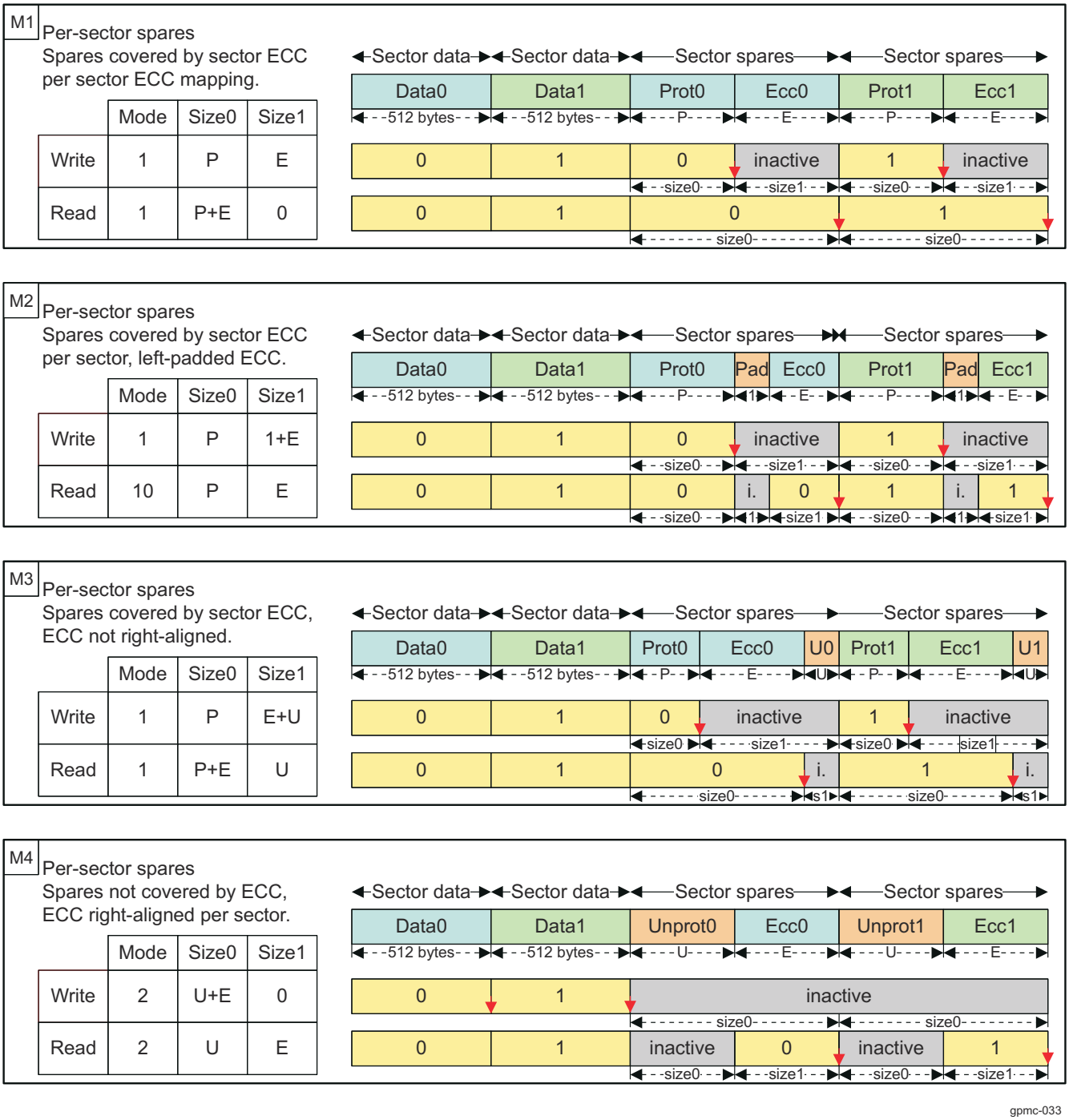


Figure 12-97. NAND Page Mapping and ECC: Per-Sector Schemes

12.3.4.3.11.3.2.3.2 Pooled Spare Mapping

In the following schemes, the spare area is pooled for the page.

- The ECC of each sector is aligned at the end of the spare area.

Copyright © 2024 Texas Instruments Incorporated

- The non-ECC spare data may or may not be covered by the ECC of sector 0.

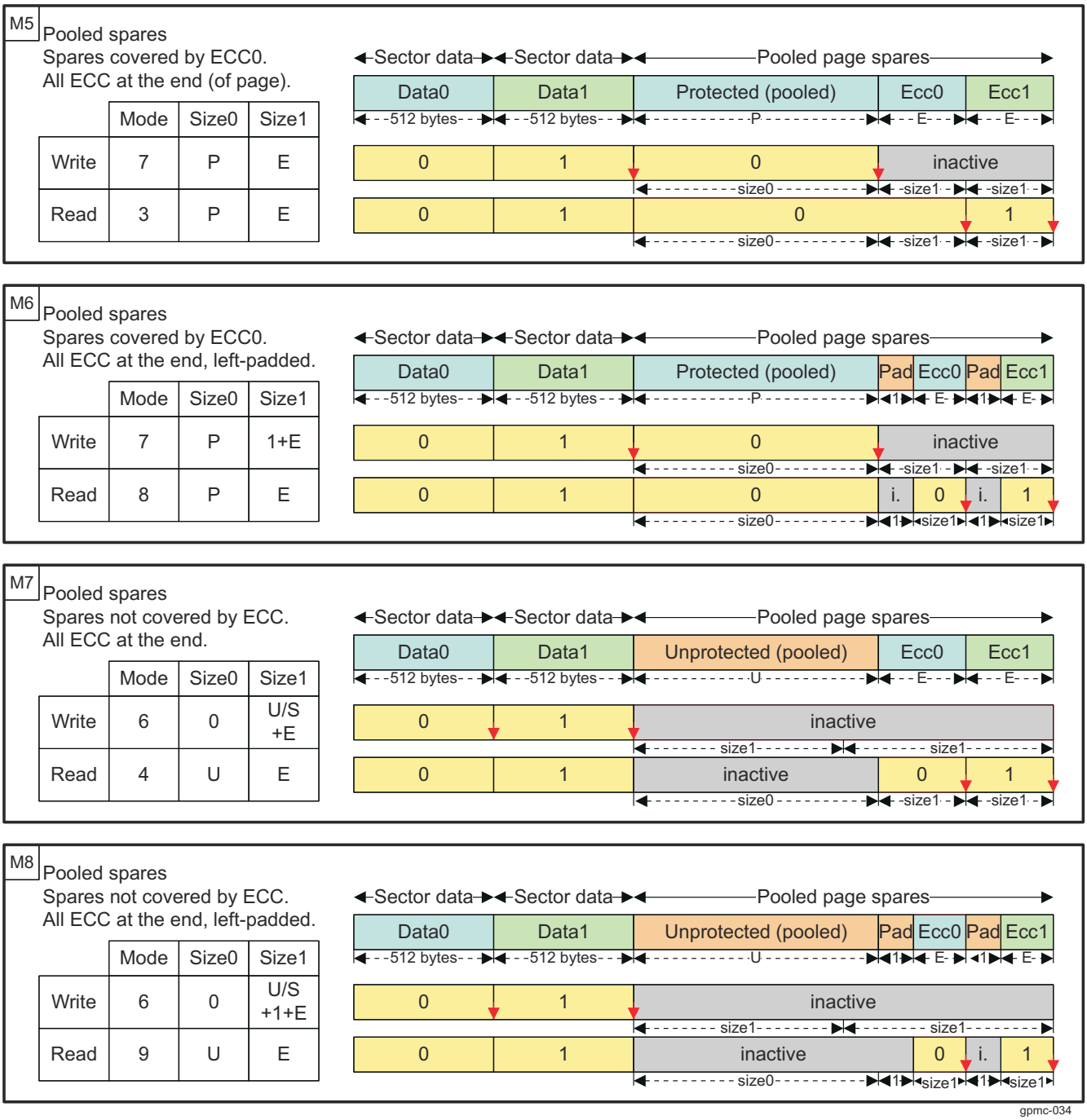


Figure 12-98. NAND Page Mapping and ECC: Pooled Spare Schemes

12.3.4.3.11.3.2.3.3 Per-Sector Spare Mapping, with ECC Separated at the End of the Page

In these schemes, each 512-byte sector of the main area is associated with two sections of the spare area.

- ECC section, all aligned at the end of the page
- Other data section, aligned before the ECCs, each of which may or may not be protected by the ECC for its sector.

Copyright © 2024 Texas Instruments Incorporated

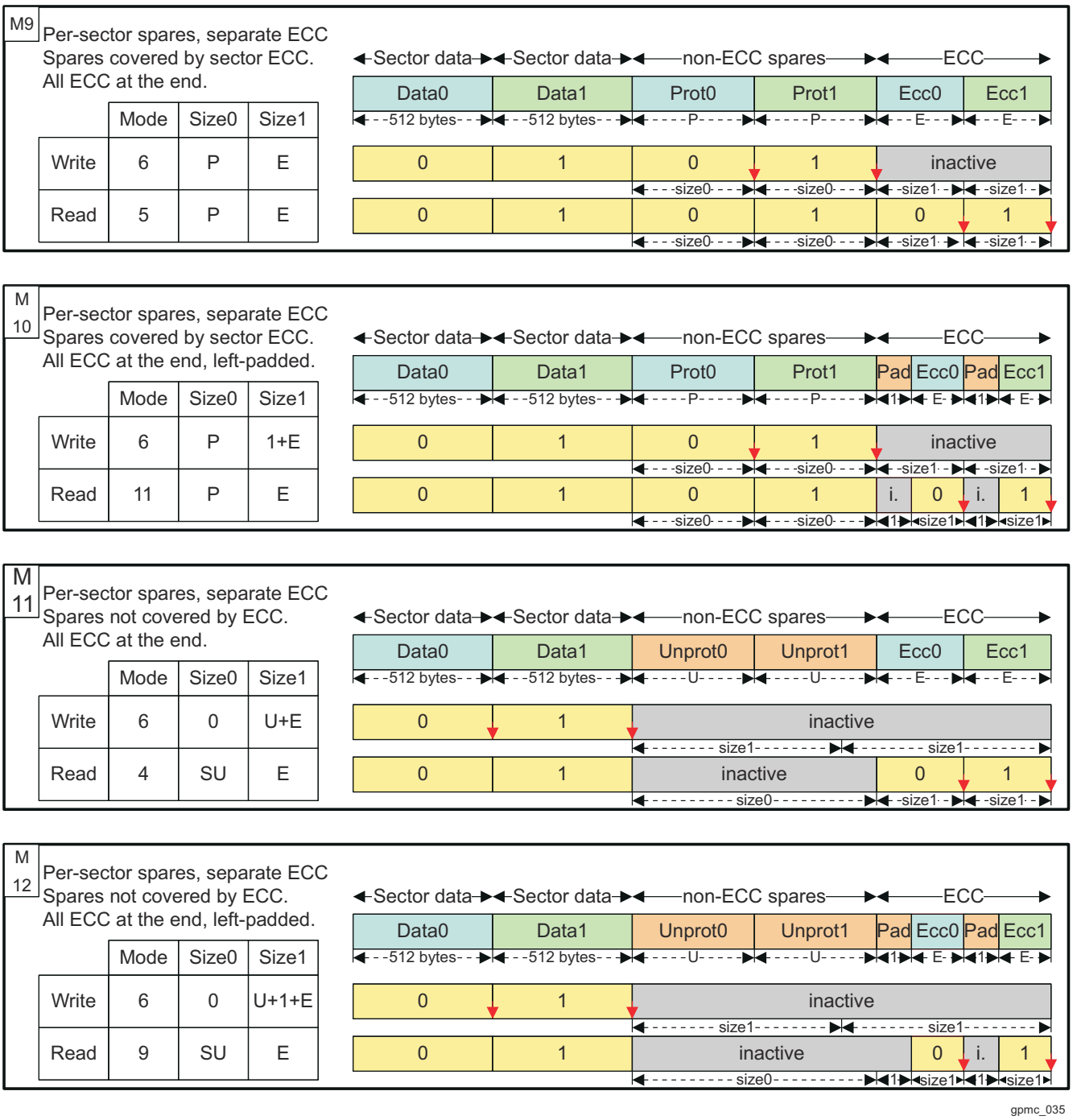


Figure 12-99. NAND Page Mapping and ECC: Per-Sector Schemes, With Separate ECC

12.3.4.3.11.4 Prefetch and Write-Posting Engine

Note

Some of the GPMC features described in this section may not be supported on this family of devices. For more information, see *GPMC Not Supported Features*.

NAND device data access cycles are usually much slower than the CPU system frequency; such NAND read or write accesses issued by the processor affect the overall system performance, especially considering long read or write sequences required for NAND page loading or programming. To minimize this effect on system

Copyright © 2024 Texas Instruments Incorporated

performance, the GPMC includes a prefetch and write-posting engine, which can be used to read from or write to any chip-select location in a buffered manner.

The prefetch and write-posting engine is a simplified embedded-access requester that presents requests to the access engine on a user-defined chip-select target. The access engine interleaves these requests with any request coming from the interconnect interface; as a default, the prefetch and write-posting engine has the lowest priority.

The prefetch and write-posting engine is dedicated to data-stream access (as opposed to random data access); thus, it is primarily dedicated to NAND support. The engine does not include an address generator; the request is limited to chip-select target identification. It includes a 64-byte FIFO associated with a DMA request synchronization line, for optimal DMA-based use.

The prefetch and write-posting engine uses an embedded 64-byte (32 16-bit word) FIFO to prefetch data from the NAND device in read mode (prefetch mode) or to store host data to be programmed into the NAND device in write mode (write-posting mode). The FIFO draining and filling (read and write) can be controlled by a device host processor through interrupt synchronization (an interrupt is triggered whenever a programmable threshold is reached) or by a device DMA module through DMA request synchronization, with a programmable request byte size in prefetch or posting mode.

The prefetch and write-posting engine includes a single memory pool. Therefore, only one mode, read or write, can be used at any given time. In other words, the prefetch and write-posting engine is a single-context engine that can be allocated to only one chip-select at a time for a read prefetch or a write-posting process.

The engine does not support atomic command and address phase programming and is limited to linear memory read or write access. As a consequence, it is limited to NAND data-stream access. The engine depends on the NAND software driver to control block and page opening with the correct data address pointer initialization, before the engine can read from or write to the NAND memory device.

Once started, engine data read and write sequencing is based solely on FIFO location availability and until the total programmed number of bytes is read or written.

Any host-concurrent accesses to a different chip-select are correctly interleaved with ongoing engine accesses. The engine has the lowest priority access so that host accesses to a different chip-select do not suffer a large latency.

A round-robin arbitration scheme can be enabled to ensure minimum bandwidth to the prefetch and write-posting engine in the case of back-to-back direct memory requests to a different chip-select. If the GPMC_PREFETCH_CONFIG1[23] PFPWENROUNDROBIN bit is enabled, the arbitration grants the prefetch and write-posting engine access to the GPMC bus for a number of requests programmed in the GPMC_PREFETCH_CONFIG1[19-16] PFPWWEIGHTEDPRIO bit field.

The prefetch/write-posting engine read or write request is routed to the access engine with the chip-select destination ID. After the required arbitration phase, the access engine processes the request as a single access with the data access size equal to the device size specified in the corresponding chip-select configuration.

Note

The destination chip-select configuration must be set to the NAND protocol-compatible configuration for which address lines are not used (the address bus is not changed from its current value). Selecting a different chip-select configuration can produce undefined behavior.

12.3.4.3.11.4.1 General Facts About the Engine Configuration

The engine can be configured only if the GPMC_PREFETCH_CONTROL[0] STARTENGINE bit is deasserted.

The engine must be correctly configured in prefetch or write-posting mode and must be linked to a NAND chip-select before it can be started. The chip-select is linked using the GPMC_PREFETCH_CONFIG1[26-24] ENGINECSSELECTOR bit field.

Copyright © 2024 Texas Instruments Incorporated

In prefetch and write-posting modes, the engine uses byte or 16-bit word access requests, respectively, for an 8- or 16-bit-wide NAND device attached to the linked chip-select. The FIFOTHRESHOLD and TRANSFERCOUNT bit fields must be programmed accordingly as a number of bytes.

When the GPMC_PREFETCH_CONFIG1[7] ENABLEENGINE bit is set, the FIFO entry on the interconnect port side is accessible at any address in the associated chip-select memory region. When the ENABLEENGINE bit is set, any host access to this chip-select is rerouted to the FIFO input. Directly accessing the NAND device linked to this chip-select from the host is still possible through the following registers (where i = 0 to 3):

- GPMC_NAND_COMMAND_i
- GPMC_NAND_ADDRESS_i
- GPMC_NAND_DATA_i

The FIFO entry on the interconnect port can be accessed with byte, 16-bit word, or 32-bit word access size, according to little-endian format, even though the FIFO input is 32 bits wide.

The FIFO control is made easier through the use of interrupts or DMA requests associated with the FIFOTHRESHOLD bit field. The GPMC_PREFETCH_STATUS[30-24] FIFOPOINTER bit field monitors the number of available bytes to be read in prefetch mode or the number of free empty slots that can be written in write-posting mode. The GPMC_PREFETCH_STATUS[13-0] COUNTVALUE bit field monitors the number of remaining bytes to be read or written by the engine according to the value of the TRANSFERCOUNT bit field. The FIFOPOINTER and COUNTVALUE bit fields are always expressed as a number of bytes even if a 16-bit-wide NAND device is attached to the linked chip-select.

In prefetch mode, when the FIFOPOINTER equals 0 (that is, the FIFO is empty), a host read access receives the byte last read from the FIFO as its response. In case of 32-bit word or 16-bit word read accesses, the last byte read from the FIFO is copied the required number of times to fit the requested word size. In write-posting mode, when the FIFOPOINTER equals 0 (that is, the FIFO is full), a host write overwrites the last FIFO byte location. There is no underflow or overflow error reporting in the GPMC.

12.3.4.3.11.4.2 Prefetch Mode

The prefetch mode is selected when the GPMC_PREFETCH_CONFIG1[0] ACCESSMODE bit is cleared.

The NAND software driver must issue the block and page opening (READ) command with the correct data address pointer initialization before the engine can be started to read from the NAND memory device. The engine is started by asserting the GPMC_PREFETCH_CONTROL[0] STARTENGINE bit. The STARTENGINE bit automatically clears when the prefetch process completes.

If required, the ECC calculator engine must be initialized (that is, reset, configured, and enabled) before the prefetch engine is started so that the ECC is computed correctly on all data read by the prefetch engine.

When the GPMC_PREFETCH_CONFIG1[3] SYNCHROMODE bit is cleared, the prefetch engine starts requesting data as soon as the STARTENGINE bit is set. If using this configuration, the host must monitor the NAND device-ready pin so that it sets the STARTENGINE bit only when the NAND device is in a ready state (that is, data is valid for prefetching).

When the SYNCHROMODE bit is set, the prefetch engine starts requesting data when an active-to-inactive WAIT signal transition is detected. The transition detector must be cleared before any transition detection (see Section 12.3.4.3.11.2.2, *Ready Pin Monitored by Hardware Interrupt*). The GPMC_PREFETCH_CONFIG1[5-4] WAITPINSELECTOR bit field selects which GPMC_WAIT pin edge detector triggers the prefetch engine in this synchronized mode.

If the STARTENGINE bit is set after the NAND address phase (page opening command), the engine is effectively started only after the actual NAND address phase completion. To prevent GPMC stall during this NAND address phase, set the STARTENGINE bit before NAND address phase completion when in synchronized mode. The prefetch engine starts when an active-to-inactive WAIT signal transition is detected. The STARTENGINE bit is automatically cleared on prefetch process completion.

The prefetch engine issues a read request to fill the FIFO with the amount of data specified by the GPMC_PREFETCH_CONFIG2[13-0] TRANSFERCOUNT bit field.

Table 12-178 describes the prefetch mode configuration.

Copyright © 2024 Texas Instruments Incorporated

Table 12-178. Prefetch Mode Configuration

Bit Field	Register	Value	Comments
STARTENGINE	GPMC_PREFETCH_CONTROL[0]	0	Prefetch engine can be configured only if STARTENGINE is set to 0.
ENGINECSSELECTOR	GPMC_PREFETCH_CONFIG1[26-24]	0 to 3	Selects the chip-select associated with a NAND device where the prefetch engine is active.
ACCESSMODE	GPMC_PREFETCH_CONFIG1[0]	0	Selects prefetch mode
FIFOTHRESHOLD	GPMC_PREFETCH_CONFIG1[14-8]		Selects the maximum number of bytes read or written by the host on DMA or interrupt request
TRANSFERCOUNT	GPMC_PREFETCH_CONFIG2[13-0]		Selects the number of bytes to be read or written by the engine to the selected chip-select
SYNCHROMODE	GPMC_PREFETCH_CONFIG1[3]	0/1	Selects when the engine starts the access to the chip-select
WAITPINSELECT	GPMC_PREFETCH_CONFIG1[17-16]	0 to 1	Selects WAIT pin edge detector (if GPMC_PREFETCH_CONFIG1[3] SYNCHROMODE = 0x1)
ENABLEOPTIMIZEDACCESS	GPMC_PREFETCH_CONFIG1[27]	0/1	See Section 12.3.4.3.11.4.6, *Optimizing NAND Access Using the Prefetch and Write-Posting Engine*.
CYCLEOPTIMIZATION	GPMC_PREFETCH_CONFIG1[30-28]		Number of clock cycle removed to timing parameters
ENABLEENGINE	GPMC_PREFETCH_CONFIG1[7]	1	Engine enabled
STARTENGINE	GPMC_PREFETCH_CONTROL[0]	1	Starts the prefetch engine

12.3.4.3.11.4.3 FIFO Control in Prefetch Mode

Note

Some of the GPMC features described in this section may not be supported on this family of devices. For more information, see *GPMC Not Supported Features*.

The FIFO can be drained directly by a device host processor or a DMA module channel.

In draining mode, the FIFO status can be monitored through the GPMC_PREFETCH_STATUS[30-24] FIFOPOINTER bit field or through the GPMC_PREFETCH_STATUS[16] FIFOTHRESHOLDSTATUS bit. The FIFOPOINTER indicates the current number of available data to be read; FIFOTHRESHOLDSTATUS set to 1 indicates that at least FIFOTHRESHOLD bytes are available from the FIFO.

An interrupt can be triggered by the GPMC if the GPMC_IRQENABLE[0] FIFOEVENTENABLE bit is set. The FIFO interrupt event is logged, and the GPMC_IRQSTATUS[0] FIFOEVENTSTATUS bit is set. To clear the interrupt, all the available bytes must be read, or at least enough bytes to get below the programmed FIFO threshold, and the FIFOEVENTSTATUS bit must be cleared to enable further interrupt events. The FIFOEVENTSTATUS bit must always be reset before asserting the FIFOEVENTENABLE bit to clear any out-of-date logged interrupt event. This interrupt generation must be enabled after enabling the STARTENGINE bit.

Prefetch completion can be monitored through the GPMC_PREFETCH_STATUS[13-0] COUNTVALUE bit field. COUNTVALUE indicates the number of currently remaining data to be requested according to the TRANSFERCOUNT value. An interrupt can be triggered by the GPMC when the prefetch process is complete (that is, COUNTVALUE equals 0) if the GPMC_IRQENABLE[1] TERMINALCOUNTEVENTENABLE bit is set. At prefetch completion, the TERMINALCOUNT interrupt event is also logged, and the GPMC_IRQSTATUS[1] TERMINALCOUNTSTATUS bit is set. To clear the interrupt, the TERMINALCOUNTSTATUS bit must be cleared. The TERMINALCOUNTSTATUS bit must always be cleared prior to asserting the TERMINALCOUNTEVENTENABLE bit to clear any out-of-date logged interrupt event.

Copyright © 2024 Texas Instruments Incorporated

Note

The COUNTVALUE value is valid only when the prefetch engine is active (started), and an interrupt is only triggered when COUNTVALUE reaches 0, that is, when the prefetch engine automatically goes from an active to an inactive state.

The number of bytes to be prefetched (programmed in TRANSFERCOUNT) must be a multiple of the programmed FIFOTHRESHOLD to trigger the correct number of interrupts allowing a deterministic and transparent FIFO control. If this guideline is respected, the number of ISR accesses is always required and the FIFO is always empty after the last interrupt is triggered. In other cases, the TERMINALCOUNT interrupt must be used to read the remaining bytes in the FIFO (the number of remaining bytes being lower than the FIFOTHRESHOLD value).

In DMA draining mode, the GPMC_PREFETCH_CONFIG1[2] DMAMODE bit must be set so that the GPMC issues a DMA hardware request when at least FIFOTHRESHOLD bytes are ready to be read from the FIFO. The DMA channel that owns this DMA request must be programmed so that the number of bytes programmed in FIFOTHRESHOLD is read from the FIFO during the DMA request process. The DMA request is kept active until this number of bytes has effectively been read from the FIFO, and no other DMA request can be issued until the ongoing active request is complete.

In prefetch mode, the TERMINALCOUNT event is also a source of DMA requests if the number of bytes to be prefetched is not a multiple of FIFOTHRESHOLD, the remaining bytes in the FIFO can be read by the DMA channel using the last DMA request. This assumes that the number of remaining bytes to be read is known and controlled through the DMA channel programming model.

Any potentially active DMA request is cleared when the prefetch engine goes from inactive-to-active prefetch (the STARTENGINE bit is set to 1). The associated DMA channel must always be enabled after setting the STARTENGINE bit so that the out-of-date active DMA request does not trigger spurious DMA transfers.

12.3.4.3.11.4.4 Write-Posting Mode

The write-posting mode is selected when the GPMC_PREFETCH_CONFIG1[0] ACCESSMODE bit is set.

The NAND software driver must issue the correct address pointer initialization command (page program) before the engine can start writing data into the NAND memory device. The engine starts when the GPMC_PREFETCH_CONTROL[0] STARTENGINE bit is set to 1. The STARTENGINE bit clears automatically when posting completes. When all data have been written to the NAND memory device, the NAND software driver must issue the second cycle program command and monitor the status for programming process completion (adding ECC handling, if required).

If used, the ECC calculator engine must be started (configured, reset, and enabled) before the posting engine is started so that the ECC parities are calculated properly on all data written by the prefetch engine to the associated chip-select.

In write-posting mode, the GPMC_PREFETCH_CONFIG1[3] SYNCHROMODE bit must be cleared so that posting starts as soon as the STARTENGINE bit is set and the FIFO is not empty.

If the STARTENGINE bit is set after the NAND address phase (page program command), the STARTENGINE setting is effective only after the actual NAND command completion. To prevent GPMC stall during this NAND command phase, set the STARTENGINE bit field before the NAND address completion and ensure that the associated DMA channel is enabled after the NAND address phase.

The posting engine issues a write request when valid data are available from the FIFO and until the programmed GPMC_PREFETCH_CONFIG2[13-0] TRANSFERCOUNT accesses are complete.

The STARTENGINE bit clears automatically when posting completes. When all data have been written to the NAND memory device, the NAND software driver must issue the second cycle program command and monitor the status for programming process completion. The closing program command phase must be issued only when the full NAND page has been written into the NAND flash write buffer, including the spare area data and the ECC parities, if used.

Table 12-179 describes the write-posting configuration.

Copyright © 2024 Texas Instruments Incorporated

Table 12-179. Write-Posting Mode Configuration

Bit Field	Register	Value	Comments
STARTENGINE	GPMC_PREFETCH_CONTROL[0]	0	Write-posting engine can be configured only if STARTENGINE is set to 0.
ENGINECSSELECTOR	GPMC_PREFETCH_CONFIG1[26-24]	0 to 3	Selects the chip-select associated with a NAND device where the prefetch engine is active
ACCESSMODE	GPMC_PREFETCH_CONFIG1[0]	1	Selects write-posting mode
FIFOTHRESHOLD	GPMC_PREFETCH_CONFIG1[14-8]		Selects the maximum number of bytes read or written by the host on DMA or interrupt request
TRANSFERCOUNT	GPMC_PREFETCH_CONFIG2[13-0]		Selects the number of bytes to be read or written by the engine from/to the selected chip-select
SYNCHROMODE	GPMC_PREFETCH_CONFIG1[3]	0	Engine starts the access to chip-select as soon as STARTENGINE is set.
ENABLEOPTIMIZEDACCESS	GPMC_PREFETCH_CONFIG1[27]	0/1	See Section 12.3.4.3.11.4.6, *Optimizing NAND Access Using the Prefetch and Write-Posting Engine*.
CYCLEOPTIMIZATION	GPMC_PREFETCH_CONFIG1[30-28]		
ENABLEENGINE	GPMC_PREFETCH_CONFIG1[7]	1	Engine enabled
STARTENGINE	GPMC_PREFETCH_CONTROL[0]	1	Starts the prefetch engine

12.3.4.3.11.4.5 FIFO Control in Write-Posting Mode

Note

Some of the GPMC features described in this section may not be supported on this family of devices. For more information, see *GPMC Not Supported Features*.

The FIFO can be filled directly by a device host processor or a DMA module channel.

In filling mode, the FIFO status can be monitored through the FIFOPOINTER or through the GPMC_PREFETCH_STATUS[16] FIFOTHRESHOLDSTATUS bit. FIFOPOINTER indicates the current number of available free byte places in the FIFO, and the FIFOTHRESHOLDSTATUS bit, when set, indicates that at least FIFOTHRESHOLD free byte places are available in the FIFO.

An interrupt can be issued by the GPMC if the GPMC_IRQENABLE[0] FIFOEVENTENABLE bit is set. When the interrupt is fired, the GPMC_IRQSTATUS[0] FIFOEVENTSTATUS bit is set. To clear the interrupt, enough bytes must be written to fill the FIFO or to get below the programmed threshold, and the FIFOEVENTSTATUS bit must be cleared to get further interrupt events. The FIFOEVENTSTATUS bit must always be cleared before asserting the FIFOEVENTENABLE bit to clear any out-of-date logged interrupt event. This interrupt must be enabled after enabling the STARTENGINE bit.

The posting completion can be monitored through the GPMC_PREFETCH_STATUS[13-0] COUNTVALUE bit field. COUNTVALUE indicates the current number of remaining data to be written based on the value of the TRANSFERCOUNT bit field. An interrupt is issued by the GPMC when the write-posting process completes (that is, COUNTVALUE equal to 0) if the GPMC_IRQENABLE[1] TERMINALCOUNTEVENTENABLE bit is set. When the interrupt is fired, the GPMC_IRQSTATUS[1] TERMINALCOUNTSTATUS bit is set. To clear the interrupt, the TERMINALCOUNTSTATUS bit must be cleared. The TERMINALCOUNTSTATUS bit must always be cleared before asserting the TERMINALCOUNTEVENTENABLE bit to clear any out-of-date logged interrupt event.

Note

The value of the COUNTVALUE bit field is valid only if the write-posting engine is active and started, and an interrupt is issued only when COUNTVALUE reaches 0; that is, when the posting engine automatically goes from active to inactive.

Copyright © 2024 Texas Instruments Incorporated

In DMA filling mode, the DMAMode bit field in the GPMC_PREFETCH_CONFIG1[2] DMAMODE bit must be set so that the GPMC issues a DMA hardware request when at least FIFOTHRESHOLD bytes-free places are available in the FIFO. The DMA channel that owns this DMA request must be programmed so that a number of bytes equal to the value programmed in the FIFOTHRESHOLD bit field are written into the FIFO during the DMA access. The DMA request remains active until the associated number of bytes has effectively been written into the FIFO, and no other DMA request can be issued until the ongoing active request completes.

Any potentially active DMA request is cleared when the prefetch engine goes from inactive-to-active prefetch (STARTENGINE set to 1). The associated DMA channel must always be enabled after setting the STARTENGINE bit so that an out-of-date active DMA request does not trigger spurious DMA transfers.

In write-posting mode, DMA or CPU fills the FIFO with no consideration for the associated byte enables. Any byte stored in the FIFO is written into the memory device.

12.3.4.3.11.4.6 Optimizing NAND Access Using the Prefetch and Write-Posting Engine

Access time to a NAND memory device can be optimized for back-to-back accesses if the associated nCS signal is not deasserted between accesses. The GPMC access engine can track prefetch engine accesses to optimize the access timing parameter programmed for the allocated chip-select, if no accesses to other chip-selects (that is, interleaved accesses) occur. Similarly, the access engine also eliminates CYCLE2CYCLEDELAY even if CYCLE2CYCLESAMECSEN is set. This capability is limited to the prefetch and write-posting engine accesses, and accesses to a NAND memory device (through the defined chip-select memory region or through the GPMC_NAND_DATA_i location, where i = 0 to 3) are never optimized.

The GPMC_PREFETCH_CONFIG1[27] ENABLEOPTIMIZEDACCESS bit must be set to enable optimized accesses. To optimize access time, the GPMC_PREFETCH_CONFIG1[30-28] CYCLEOPTIMIZATION bit field defines the number of GPMC_FCLK cycles to be suppressed from the following timing parameters:

- RDCYCLETIME
- WRCYCLETIME
- RDACCESSTIME
- WRACCESSTIME
- CSOFFTIME
- ADVOFFTIME
- OEOFFTIME
- WEOFFTIME

Figure 12-100 shows that in the case of back-to-back accesses to the NAND flash through the prefetch engine, CYCLE2CYCLESAMECSEN is forced to 0 when using optimized accesses. The first access uses the regular timing settings for this chip-select. All accesses after this one use settings reduced by x clock cycles, x being defined by the GPMC_PREFETCH_CONFIG1[30-28] CYCLEOPTIMIZATION bit field.

Copyright © 2024 Texas Instruments Incorporated

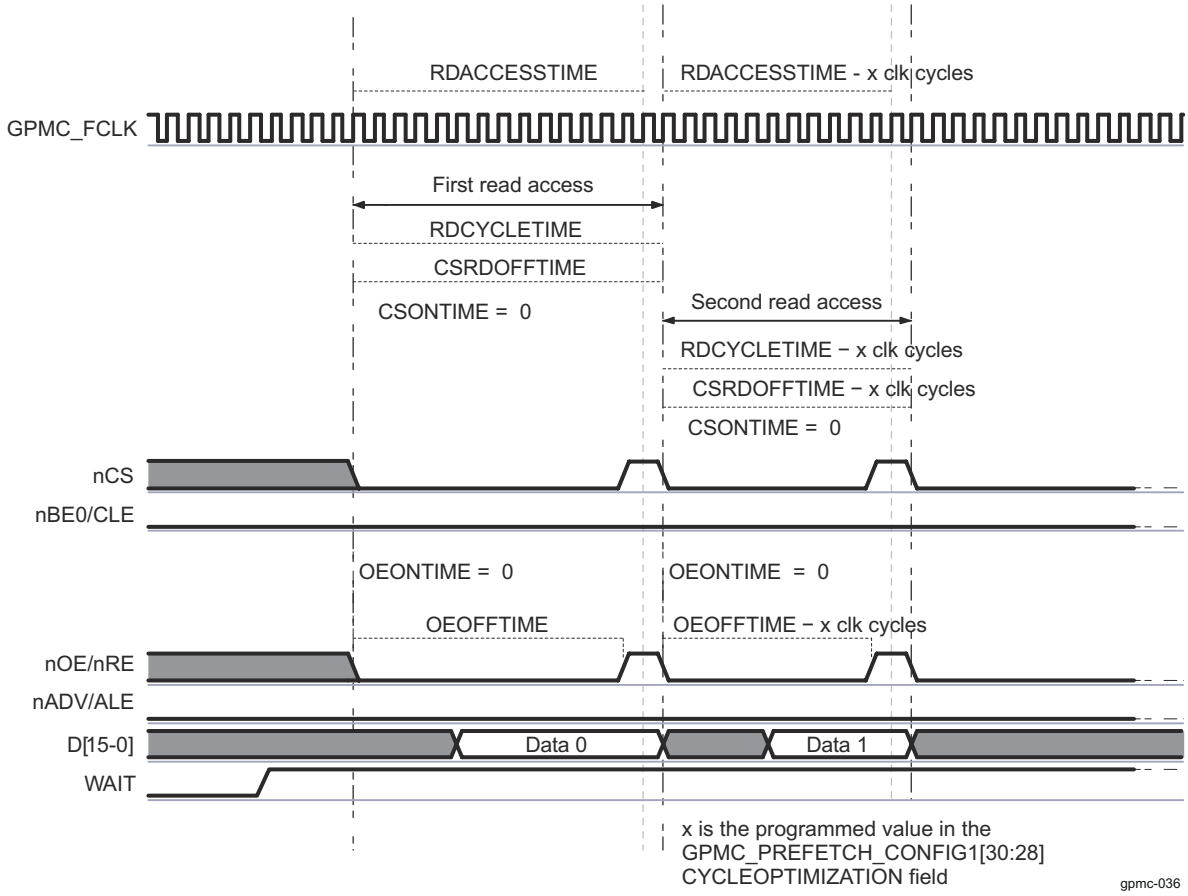


Figure 12-100. NAND Read Cycle Optimization Timing Description

12.3.4.3.11.4.7 Interleaved Accesses Between Prefetch and Write-Posting Engine and Other Chip-Selects

Any on-going read or write access from the prefetch and write-posting engine is completed before an access to any other chip-select can be initiated. As a default, the arbiter uses a fixed-priority algorithm, and the prefetch and write-posting engine has the lowest priority. The maximum latency added to access starting time in this case equals the RDCYCLETIME or WRCYCLETIME (optimized or not) plus the requested BUSTURNAROUND delay for bus turnaround completion programmed for the chip-select to which the NAND device is connected.

Alternatively, a round-robin arbitration can be used to prioritize accesses to the external bus. This arbitration scheme is enabled by setting the GPMC_PREFETCH_CONFIG1[23] PFPWENROUNDROBIN bit. When a request to another chip-select is received while the prefetch and write-posting engine is active, priority is given to the new request. The request processed thereafter is the prefetch and write-posting engine request, even if another interconnect request is passed in the mean time. The engine keeps control of the bus for an additional number of requests programmed in the GPMC_PREFETCH_CONFIG1[19-16] PFPWWEIGHTEDPRIO bit field. Control is then passed to the direct interconnect request.

As an example, the round-robin arbitration scheme is selected with PFPWWEIGHTEDPRIO set to 0x2. Considering that the prefetch and write-posting engine and the interconnect interface are always requesting access to the external interface, the GPMC grants priority to the direct interconnect access for one request. The GPMC then grants priority to the engine for three requests, and finally back to the direct interconnect access, until the arbiter is reset when one of the two initiators stops initiating requests.

12.3.4.3.12 GPMC Use Cases and Tips

12.3.4.3.12.1 How to Set GPMC Timing Parameters for Typical Accesses

12.3.4.3.12.1.1 External Memory Attached to the GPMC Module

As discussed in the introduction to this chapter, the GPMC module supports the following external memory types:

Copyright © 2024 Texas Instruments Incorporated

- Asynchronous or synchronous, 8- or 16-bit-wide memory or device
- 16-bit address/data-multiplexed or non-multiplexed NOR flash device
- 8- or 16-bit NAND flash device

The following examples describe how to calculate GPMC timing parameters by showing a typical parameter setup for the access to be performed.

The example is based on a 512-Mb multiplexed NOR flash memory with the following characteristics:

- Type: NOR flash (address/data-multiplexed mode)
- Size: 512M bits
- Data Bus: 16 bits wide
- Speed: 104-MHz clock frequency
- Read access time: 80 ns

12.3.4.3.12.1.2 Typical GPMC Setup

Table 12-180 lists some of the I/Os of the GPMC module.

Table 12-180. Typical GPMC Setup Signals

Module Pin	I/O[(1)]	Description
GPMC0_FCLK	Internal	Functional clock. Acts as the time reference.
GPMC0_ICLK	Internal	Interface clock. Acts as the time reference.
GPMC0_CLKOUT	O	External clock provided to the external device for synchronous operations
	O	Address
GPMC0_AD[15-0]	I/O	Data-multiplexed with addresses A[16-1] on memory side
GPMC0_CSn[3-0]	O	Chip-selects
GPMC0_ADVn_ALE	O	Address valid enable
GPMC0_OEn_REn	O	Output enable (read access only)
GPMC0_WEn	O	Write enable (write access only)
GPMC0_WAIT[3-0]	I	Ready signal from memory device. Indicates when valid burst data is ready to be read

(1) I = Input; O = Output

The following sections demonstrate how to calculate GPMC parameters for three access types:

- Synchronous burst read
- Asynchronous read
- Asynchronous single write

12.3.4.3.12.1.2.1 GPMC Configuration for Synchronous Burst Read Access

Note

The examples in Section 12.3.4.3.12.1.2.1 through Section 12.3.4.3.12.1.2.3 are based on a clock rate of 104 MHz. See the device-specific Datasheet for the maximum frequency appropriate for this device and the memory datasheet for the maximum frequency for the particular memory device.

The clock runs at 104 MHz (f = 104 MHz; T = 9.615 ns).

Table 12-181 shows the timing parameters (on the memory side) that determine the parameters on the GPMC side.

Table 12-182 shows how to calculate timings for the GPMC using the memory parameters.

Figure 12-101 shows the synchronous burst read access.

Table 12-181. Useful Timing Parameters on the Memory Side

AC Read Characteristics on the Memory Side	Description	Duration (ns)
tCES	nCS setup time to clock	0
tACS	Address setup time to clock	3

Copyright © 2024 Texas Instruments Incorporated

Table 12-181. Useful Timing Parameters on the Memory Side (continued)

AC Read Characteristics on the Memory Side	Description	Duration (ns)
tIACC	Synchronous access time	80
tBACC	Burst access time valid clock to output delay	5.2
tCEZ	Chip-select to High-Z	7
tOEZ	Output enable to High-Z	7
tAVC	nADV setup time	6
tAVD	nAVD pulse	6
tACH	Address hold time from clock	3

The following terms, which describe the timing interface between the controller and its attached device, are used to calculate the timing parameters on the GPMC side:

- Read access time (GPMC side): Time required to activate the clock + read access time requested on the memory side + data setup time required for optimal capture of a burst of data
- Data setup time (GPMC side): Ensures a good capture of a burst of data (as opposed to taking a burst of data out). One word of data is processed in one clock cycle (T = 9.615 ns). The read access time between two bursts of data is tBACC = 5.2 ns. Therefore, data setup time is a clock period – tBACC = 4.415 ns of data setup.
- Access completion (GPMC side): (Different from page burst access time) Time required between the last burst access and access completion: nCS/nOE hold time (nCS and nOE must be released at the end of an access. These signals are held to allow the access to complete).
- Read cycle time (GPMC side): Read access time + access completion
- Write cycle time for burst access: Not supported for NOR flash memory

Table 12-182. Calculating GPMC Timing Parameters

Parameter Name on GPMC Side	Formula	Duration (ns)	Number of Clock Cycles (F = 104 MHz)	GPMC Register Configurations
GPMC FCLK Divider	–	–	–	GPMCFCLKDIVIDER = 0x0
ClkActivation Time	min (tCES, tACS)	3	1	CLKACTIVATIONTIME = 0x1
RdAccessTime	roundmax (ClkActivationTime + tIACC + DataSetupTime)	94.03: (9.615 + 80 + 4.415)	10: roundmax (94.03 / 9.615)	RDACCESSTIME = 0xA
PageBurst RdAccessTime	roundmax (tBACC)	roundmax (5.2)	1	PAGEBURSTACCESSTIME = 0x1
RdCycleTime	RdAccess time + max (tCEZ, tOEZ)	101.03: (94.03 + 7)	11	RDCYCLETIME = 0xB
CsOnTime	tCES	0	0	CSONTIME = 0x0
CsReadOffTime	RdCycleTime	-	11	CSRDOFFTIME = 0xB
AdvOnTime	tAVC (1)	0	0	ADVONTIME = 0x0
AdvRdOffTime	tAVD + tAVC (2)	12	2	ADVRDOFFTIME = 0x2
OeOnTime (3)	(ClkActivationTime + tACH) < OeOnTime (ClkActivationTime + tIACC)	–	3, for instance	OEONTIME = 0x3
OeOffTime	RdCycleTime	–	11	OEOFFTIME = 0xB

(1) The external clock provided to the NOR flash is not yet available.
(2) AdvRdOffTime – AdvOnTime = tAVD; thus, AdvRdOffTime = tAVD + AdvOnTime = tAVD + tAVC.
(3) OeOnTime must ensure that addresses are available. It must not exceed the availability of the first burst of data.

Copyright © 2024 Texas Instruments Incorporated

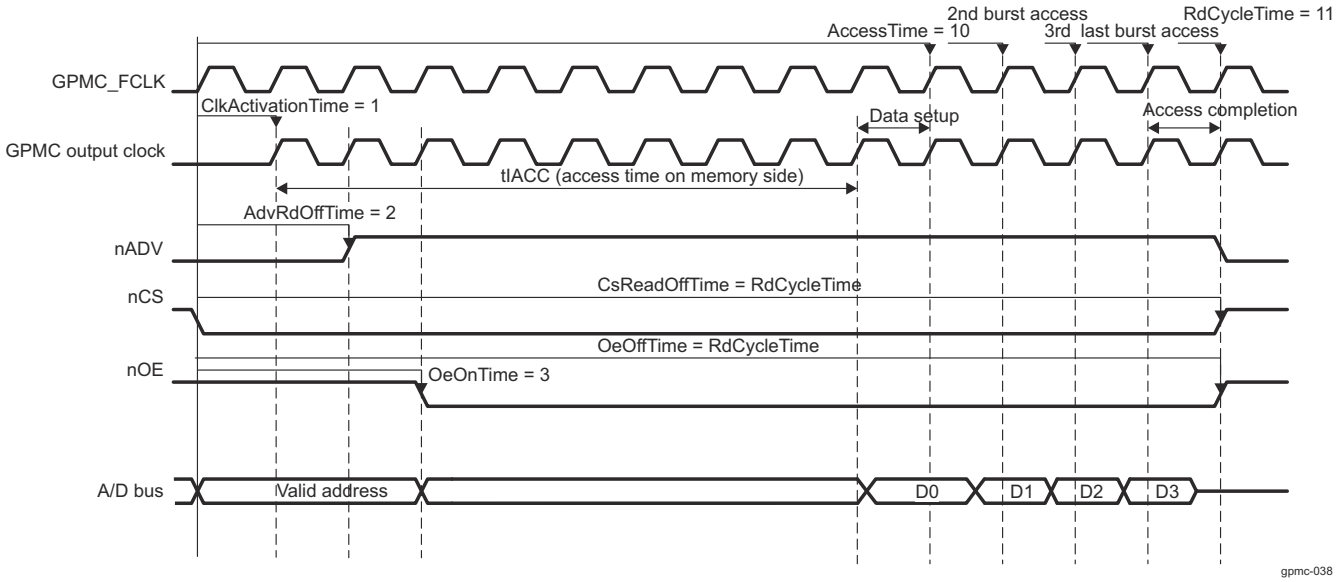


Figure 12-101. Synchronous Burst Read Access (Timing Parameters in Clock Cycles)

12.3.4.3.12.1.2.2 GPMC Configuration for Asynchronous Read Access

The clock runs at 104 MHz (f = 104 MHz; T = 9.615 ns).

Table 12-183 shows the timing parameters (on the memory side) that determine the parameters on the GPMC side.

Table 12-184 shows how to calculate timings for the GPMC using the memory parameters.

Figure 12-102 shows the asynchronous read access.

Table 12-183. AC Characteristics for Asynchronous Read Access

AC Read Characteristics on the Memory Side	Description	Duration (ns)
tCE	Read Access time from nCS low	80
tAAVDS	Address setup time to rising edge of nADV	3
tAVDP	nADV low time	6
tCAS	nCS setup time to nADV	0
tOE	Output enable to output valid	6
tOEZ	Output enable to High-Z	7

Use the following formula to calculate the RdCycleTime parameter for this typical access:

RdCycleTime = RdAccessTime + AccessCompletion = RdAccessTime + 1 clock cycle + tOEZ:

1. On the memory side, the external memory makes the data available to the output bus. This is the memory-side read access time defined in Table 12-183: the number of clock cycles between the address capture (nADV rising edge) and the data valid on the output bus.

   The GPMC requires some hold time to allow the data to be captured correctly and the access to be finished.
2. To read the data correctly, the GPMC must be configured to meet the data setup time requirement of the memory; the GPMC module captures the data on the next rising edge. This is access time on the GPMC side.
3. There must also be a data hold time for correctly reading the data (checking that there is no nOE/nCS deassertion while reading the data). This data hold time is one clock cycle (that is, RdAccessTime + 1).
4. To complete the access, nOE/nCS signals are driven to High-Z. RdAccessTime + 1 + tOEZ is the read cycle time.
5. Addresses can now be relatched and a new read cycle begun.

Copyright © 2024 Texas Instruments Incorporated

Table 12-184. GPMC Timing Parameters for Asynchronous Read Access

Parameter Name on GPMC Side	Formula	Duration (ns)	Number of Clock Cycles (F = 104 MHz)	GPMC Register Configurations
ClkActivationTime	n/a (asynchronous mode)			
RdAccessTime	round max (tCE)	80	10	RDACCESSTIME = 0xA
PageBurstAccessTime	N/A (single access)			
RdCycleTime	RdAccessTime + 1cycle + tOEZ	96.615	12	RDCYCLETIME = 0xC
CsOnTime	tCAS	0	0	CSONTIME = 0x0
CsReadOffTime	RdAccessTime + 1 cycle	89.615	11	CSRDOFFTIME = 0xB
AdvOnTime	tAAVDS	3	1	ADVONTIME = 0x1
AdvRdOffTime	tAAVDS + tAVDP	9	1	ADVRDOFFTIME = 0x1
OeOnTime	OeOnTime >= AdvRdOffTime (multiplexed mode)	-	3, for instance	OEONTIME = 0x3
OeOffTime	RdAccessTime + 1cycle	89.615	11	OEOFFTIME = 0xB

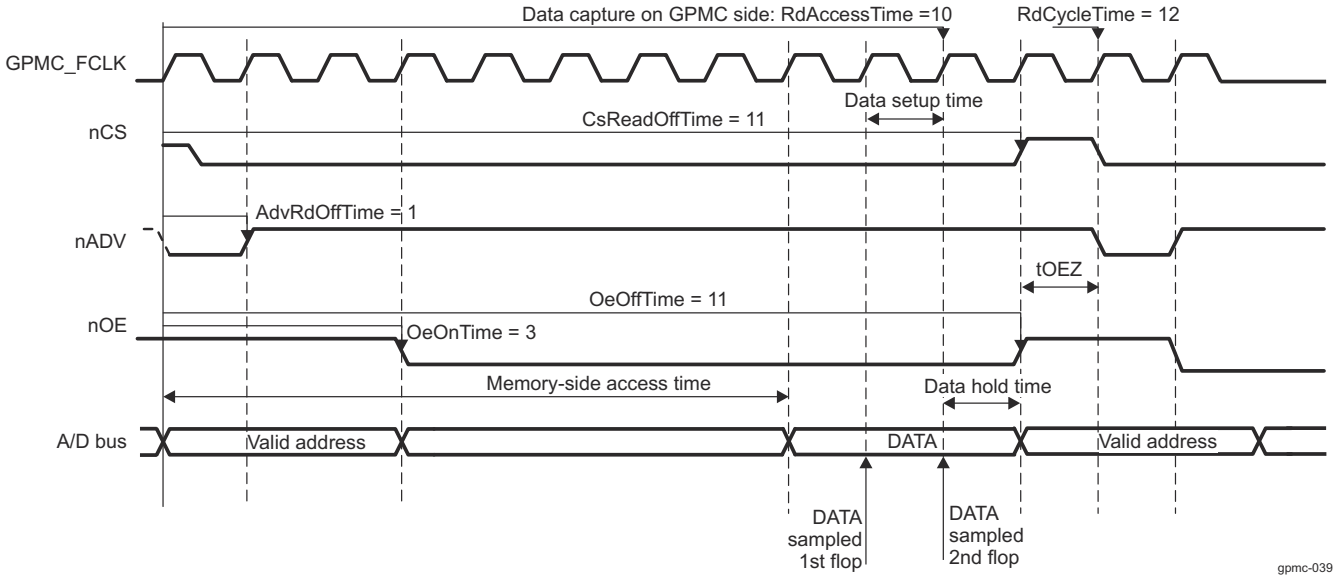


Figure 12-102. Asynchronous Single Read Access (Timing Parameters in Clock Cycles)

12.3.4.3.12.1.2.3 GPMC Configuration for Asynchronous Single Write Access

The clock runs at 104 MHz: (f = 104 MHz; T = 9.615 ns).

Table 12-186 shows how to calculate timings for the GPMC using the memory parameters.

Table 12-185 shows the timing parameters (on the memory side) that determine the parameters on the GPMC side.

Figure 12-103 shows the synchronous burst write access.

Table 12-185. AC Characteristics for Asynchronous Single Write (Memory Side)

AC Characteristics on the Memory Side	Description	Duration (ns)
tWC	Write cycle time	60
tAVDP	nADV low time	6
tWP	Write pulse width	25
tWPH	Write pulse width high	20
tCS	nCS setup time to nWE	3
tCAS	nCS setup time to nADV	0

Copyright © 2024 Texas Instruments Incorporated

Table 12-185. AC Characteristics for Asynchronous Single Write (Memory Side) (continued)

AC Characteristics on the Memory Side	Description	Duration (ns)
tAVSC	nADV setup time	3

For asynchronous single write access, write cycle time is WrCycleTime = WeOffTime + AccessCompletion = WeOffTime + 1. For the AccesCompletion, the GPMC requires one cycle of data hold time (nCS deassertion). For more information, see the device-specific Datasheet.

Table 12-186. GPMC Timing Parameters for Asynchronous Single Write

Parameter Name on GPMC Side	Formula	Duration (ns)	Number of Clock Cycles (F = 104 MHz)	GPMC Registers Configuration
ClkActivationTime	N/A (asynchronous mode)			
WdAccessTime	Applicable only to WAITMONITORING (the value is the same as for read access)			
PageBurstAccessTime	N/A (single access)			
WrCycleTime	WeOffTime + AccessCompletion	57.615	6	WRCYCLETIME = 0x6
CsOnTime	tCAS	0	0	CSONTIME = 0x0
CsWrOffTime	WeOffTime + 1	57.615	6	CSWROFFTIME = 0x6
AdvOnTime	tAVSC	3	1	ADVONTIME = 0x1
AdvWrOffTime	tAVSC + tAVDP	9	1	ADVWROFFTIME = 0x1
WeOnTime	tCS	3	1	WEONTIME = 0x1
WeOffTime	tCS + tWP + tWPH	48	5	WEOFFTIME = 0x5

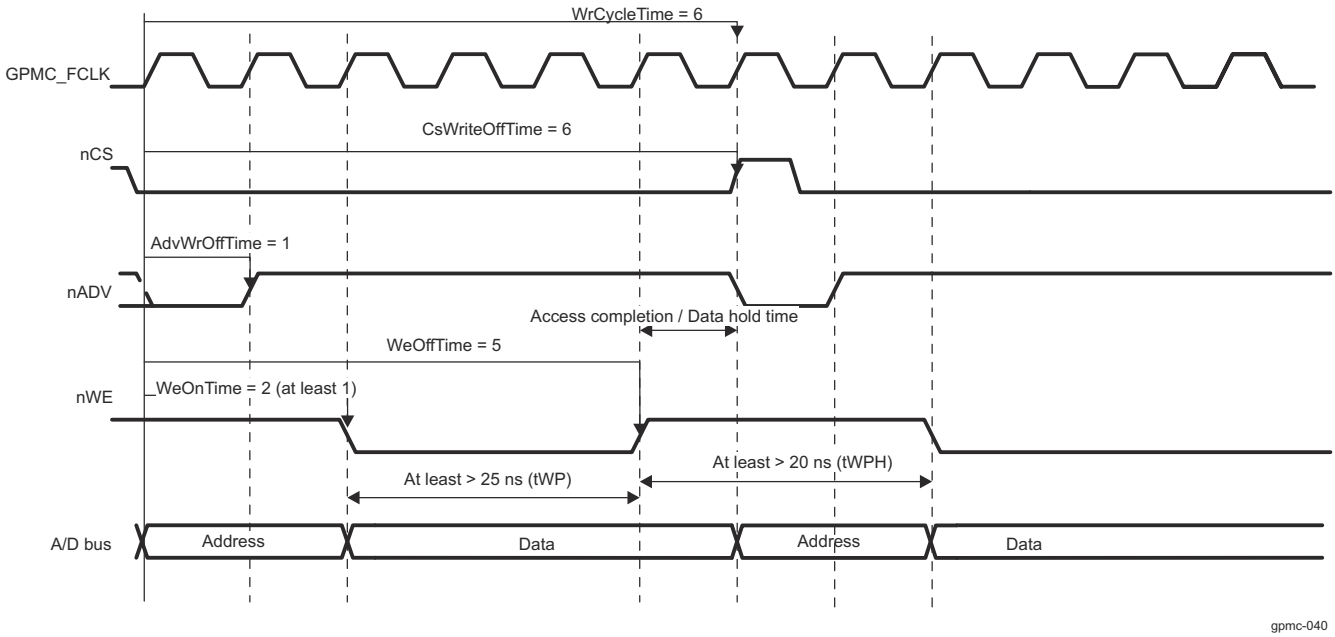


Figure 12-103. Asynchronous Single Write Access (Timing Parameters in Clock Cycles)

12.3.4.3.12.2 How to Choose a Suitable Memory to Use With the GPMC

This section is intended to help the user select a suitable memory device to interface with the GPMC controller.

12.3.4.3.12.2.1 Supported Memories or Devices

NAND flash and NOR flash architectures are the two flash technologies. The GPMC supports various types of external memory or devices, basically any one that supports NAND or NOR protocols:

- 8- and 16-bit-wide asynchronous or synchronous memory or device (only 8-bit: nonburst device)
- 16-bit address and data-multiplexed NOR flash devices (pSRAM, and so on)
- 8- and 16-bit NAND flash devices

Copyright © 2024 Texas Instruments Incorporated

12.3.4.3.12.2.1.1 Memory Pin Multiplexing

This section describes the interfacing differences of the GPMC supported memories.

Table 12-187. Supported Memory Interfaces

Function	16-Bit Address/ Data-Multiplexed pSRAM or NOR Flash [1]	16-Bit NAND	8-Bit NAND
GPMC_A[24]			
GPMC_A[23]			
GPMC_A[22]			
GPMC_A[21]			
GPMC_A[20]			
GPMC_A[19]			
GPMC_A[18]			
GPMC_A[17]			
GPMC_A[16]			
GPMC_A[15]			
GPMC_A[14]			
GPMC_A[13]			
GPMC_A[12]			
GPMC_A[11]			
GPMC_A[10]	A26		
GPMC_A[9]	A25		
GPMC_A[8]	A24		
GPMC_A[7]	A23		
GPMC_A[6]	A22		
GPMC_A[5]	A21		
GPMC_A[4]	A20		
GPMC_A[3]	A19		
GPMC_A[2]	A18		
GPMC_A[1]	A17		
GPMC_A[0]	A16		
GPMC_AD[15]	D15 or A16	IO15	
GPMC_AD[14]	D14 or A15	IO14	
GPMC_AD[13]	D13 or A14	IO13	
GPMC_AD[12]	D12 or A13	IO12	
GPMC_AD[11]	D11 or A12	IO11	
GPMC_AD[10]	D10 or A11	IO10	
GPMC_AD[9]	D9 or A10	IO9	
GPMC_AD[8]	D8 or A9	IO8	
GPMC_AD[7]	D7 or A8	IO7	
GPMC_AD[6]	D6 or A7	IO6	
GPMC_AD[5]	D5 or A6	IO5	
GPMC_AD[4]	D4 or A5	IO4	
GPMC_AD[3]	D3 or A4	IO3	
GPMC_AD[2]	D2 or A3	IO2	
GPMC_AD[1]	D1 or A2	IO1	
GPMC_AD[0]	D0 or A1	IO0	
GPMC_CLKOUT	CLK		

Copyright © 2024 Texas Instruments Incorporated

Table 12-187. Supported Memory Interfaces (continued)

Function	16-Bit Address/ Data-Multiplexed pSRAM or NOR Flash [1]	16-Bit NAND	8-Bit NAND
GPMC_CSn0	nCS0 (chip-select)	nCE0 (chip-enable)	
GPMC_CSn1	nCS1	nCE1	
GPMC_CSn2	nCS2	nCE2	
GPMC_CSn3	nCS3	nCE3	
GPMC_ADVn_ALE	nADV (address valid)	ALE (address latch enable)	
GPMC_OEn_REn	nOE (output enable)	nRE (read enable)	
GPMC_WEn	nWE (Write enable)	nWE (write enable)	
GPMC_BE0n_CLE	nBE0 (byte enable)	CLE (command latch enable)	
GPMC_BE1n	nBE1		
GPMC_WAIT0	WAIT0	R/nB0 (ready/busy)	
GPMC_WAIT1	WAIT1	R/nB1	
GPMC_WAIT2	WAIT2	R/nB2	
GPMC_WAIT3	WAIT3	R/nB3	
GPMC_WPn	nWP (Write Protect)	nWP (Write Protect)	

(1) Addresses seen from the device side. When interfacing to the external device, A1 is connected to the memory A0, A2 to the memory A1, and so on.

12.3.4.3.12.2.1.2 NAND Interface Protocol

NAND flash architecture, introduced in 1989, is a flash technology. NAND is a page-oriented memory device; that is, read and write accesses are done by pages. NAND achieves great density by sharing common areas of the storage transistor, which creates strings of serially connected transistors (in NOR devices, each transistor stands alone). Because of its high density NAND is best suited to devices that require high capacity data storage, such as pictures, music, and data files. NAND nonvolatility makes of it a good storage solution for many applications where mobility, low power, and speed are key factors. Low pin count and simple interface are other advantages of NAND.

Table 12-188 summarizes the NAND interface signals level applied to external device or memories.

Table 12-188. NAND Interface Bus Operations Summary

Bus Operation	CLE	ALE	nCE	nWE [1]	nRE [1]	nWP
Read (cmd input)	H	L	L	RE	H	x
Read (add input)	L	H	L	RE	H	x
Write (cmd input)	H	L	L	RE	H	H
Write (add input)	L	H	L	RE	H	H
Data input	L	L	L	RE	H	H
Data output	L	L	L	H	FE	x
Busy (during read)	x	x	H [2]	H [2]	H [2]	x
Busy (during program)	x	x	x	x	x	H
Busy (during erase)	x	x	x	x	x	H
Write protect	x	x	x	x	x	L
Standby	x	x	H	x	x	H/L

(1) RE stands for rising edge; FE stands for falling edge
(2) Can be nCE high, or WE and nRE high.

12.3.4.3.12.2.1.3 NOR Interface Protocol

NOR flash architecture, introduced in 1988, is a flash technology. Unlike NAND, which is a sequential access device, NOR is directly addressable; that is, it is designed to be a random access device. NOR is best suited to devices used to store and run code or firmware, usually in small capacities. While NOR has fast read capabilities, it also has slow write and erase functions when compared to the NAND architecture.

Copyright © 2024 Texas Instruments Incorporated

Table 12-189 summarizes the level of the NOR interface signals applied to external devices or memories.

Table 12-189. NOR Interface Bus Operations Summary

Bus Operation	CLK	nADV	nCS	nOE	nWE	WAIT	DQ[15-0]
Read (asynchronous)	x	L	L	L	H	Asserted	Output
Read (synchronous)	Running	L	L	L	H	Driven	Output
Read (burst suspend)	Halted	x	L	H	H	Active	Output
Write	x	L	L	H	L	Asserted	Input
Output disable	x	x	L	H	H	Asserted	High-Z
Standby	x	x	H	x	x	High-Z	High-Z

12.3.4.3.12.2.1.4 Other Technologies

Other supported device types interact with the GPMC through the NOR interface protocol.

FPGA (Field-Programmable Gate Array) is a low-power integrated circuit based on an array of programmable logic blocks.

pSRAM (pseudo-static random access memory) is a low-power memory device. pSRAM is based on the DRAM cell with internal refresh and address control features, and interfaces as a synchronous NOR flash. It has synchronous write capability.

Copyright © 2024 Texas Instruments Incorporated

12.3.4.4 GPMC Basic Programming Model

12.3.4.4.1 GPMC High-Level Programming Model Overview

The goal of the basic high-level programming model is to introduce a top-down approach to users that need to configure the GPMC module.

Figure 12-104 and Table 12-190 through Table 12-191 show a programming model top-level diagram for the GPMC, and a description of each step. Each block of the diagram is described in one of the following sections through a set of registers to configure.

Copyright © 2024 Texas Instruments Incorporated

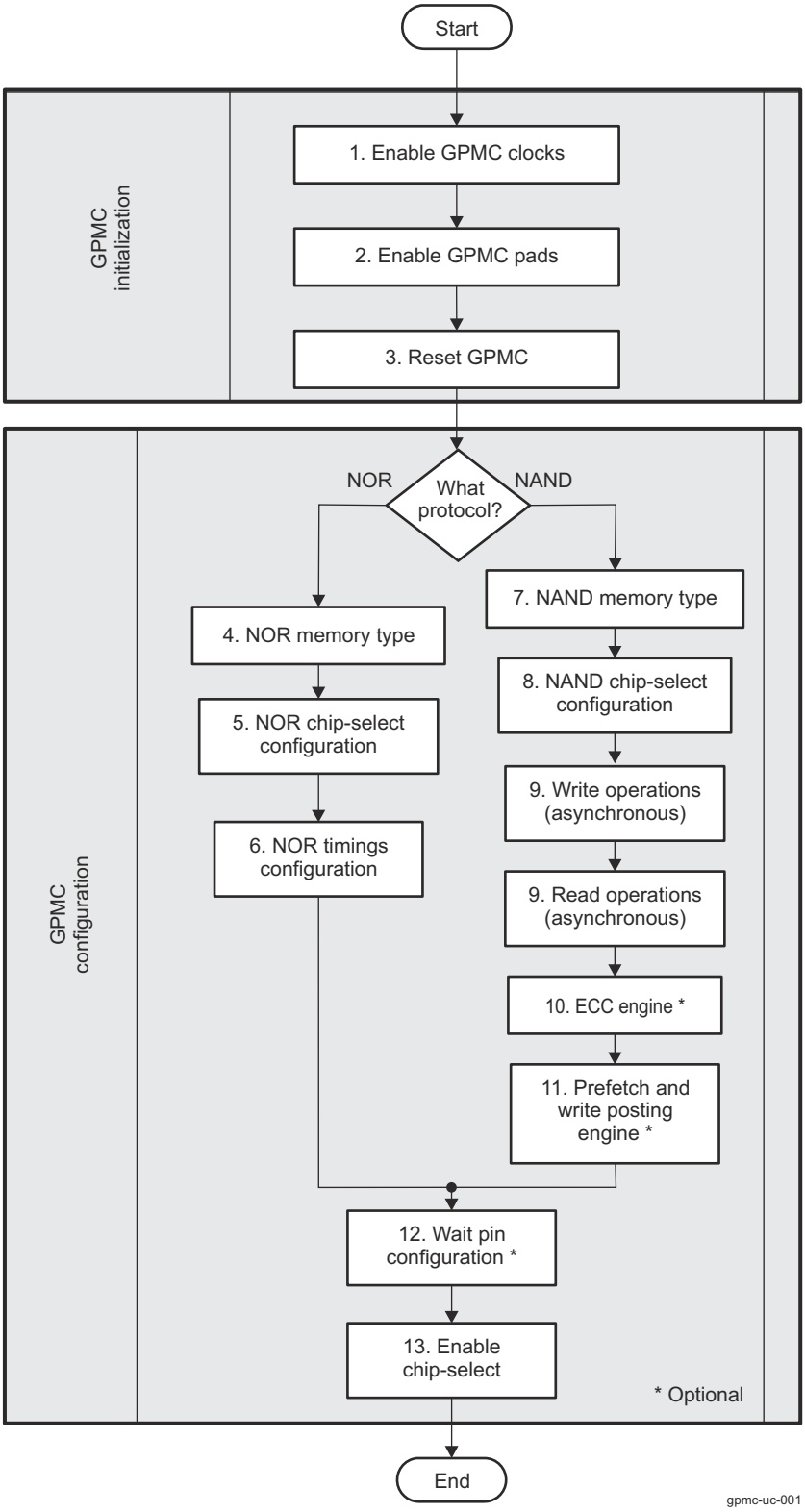


Figure 12-104. Programming Model Top-Level Diagram

Table 12-190. GPMC Configuration in NOR Mode

Step	Description
NOR Memory Type	See Table 12-192.
NOR Chip-Select Configuration	See Table 12-193.

Copyright © 2024 Texas Instruments Incorporated

Table 12-190. GPMC Configuration in NOR Mode (continued)

Step	Description
NOR Timings Configuration	See Table 12-194.
WAIT Pin Configuration	See Table 12-202.
Enable Chip-Select	See Table 12-203.

Table 12-191. GPMC Configuration in NAND Mode

Step	Description
NAND Memory Type	See Table 12-197.
NAND Chip-Select Configuration	See Table 12-198.
Write Operations (Asynchronous)	See Table 12-199.
Read Operations (Asynchronous)	See Table 12-199.
ECC Engine	See Table 12-200.
Prefetch and Write-Posting Engine	See Table 12-201.
WAIT Pin Configuration	See Table 12-202.
Enable Chip-Select	See Table 12-203.

12.3.4.4.2 GPMC Initialization

GPMC can be reset via software bit in LPSC. For more information, see Section 5.3, *Reset*.

12.3.4.4.3 GPMC Configuration in NOR Mode

This section gives a generic configuration for parameters related to the NOR memory connected to the GPMC.

Table 12-192. NOR Memory Type

Subprocess Name	Register / Bit Field	Value
Set the NOR protocol.	GPMC_CONFIG1_i[11-10] DEVICETYPE	0x0
Set a device size.	GPMC_CONFIG1_i[13-12] DEVICESIZE	x
Select an address and data multiplexing protocol.	GPMC_CONFIG1_i[9] MUXADDDATA	x
Set the attached device page length.	GPMC_CONFIG1_i[24-23] ATTACHEDDEVICEPAGELENGTH	x
Set the wrapping burst capabilities.	GPMC_CONFIG1_i[31] WRAPBURST	x
Select a timing signals latencies factor.	GPMC_CONFIG1_i[4] TIMEPARAGRANULARITY	x
Select an output clock frequency (1).	GPMC_CONFIG1_i[1-0] GPMCFCLKDIVIDER	x
Choose an output clock activation time (1).	GPMC_CONFIG1_i[26-25] CLKACTIVATIONTIME	x
Set a single or multiple access for read operations (1).	GPMC_CONFIG1_i[30] READMULTIPLE	x
Set a synchronous or asynchronous mode for read operations.	GPMC_CONFIG1_i[29] READTYPE	x
Set a single or multiple access for write operations.	GPMC_CONFIG1_i[28] WRITEMULTIPLE	x
Set a synchronous or asynchronous mode for write operations.	GPMC_CONFIG1_i[27] WRITETYPE	x

(1) Applies only to synchronous configurations (or non-multiplexed asynchronous for multiple access one)

Table 12-193. NOR Chip-Select Configuration

Subprocess Name	Register/Bit Field	Value
Select the chip-select base address.	GPMC_CONFIG7_i[5-0] BASEADDRESS	x
Select the chip-select mask address.	GPMC_CONFIG7_i[11-8] MASKADDRESS	x

Table 12-194. NOR Timings Configuration

Subprocess Name	Register/Bit Field	Value
Configure adequate timing parameters in various memory modes.	See Section 12.3.4.4.6, *GPMC Timing Parameters*	

Copyright © 2024 Texas Instruments Incorporated

Table 12-195. WAIT Pin Configuration

Subprocess Name	Register/Bit Field	Value
Enable or disable WAIT pin monitoring for read operations.	GPMC_CONFIG1_i[22] WAITREADMONITORING	x
Enable or disable WAIT pin monitoring for write operations.	GPMC_CONFIG1_i[21] WAITWRITEMONITORING	x
Select a WAIT pin monitoring time.	GPMC_CONFIG1_i[19-18] WAITMONITORINGTIME	x
Choose the input WAIT pin for the chip-select.	GPMC_CONFIG1_i[17-16] WAITPINSELECT	x

Table 12-196. Enable Chip-Select

Subprocess Name	Register/Bit Field	Value
When all parameters are configured, enable the chip-select.	GPMC_CONFIG7_i[6] CSVALID	x

12.3.4.4.4 GPMC Configuration in NAND Mode

This section gives a generic configuration for parameters related to the NAND memory connected to the GPMC.

Note

Some of the GPMC features described in this section may not be supported on this family of devices. For more information, see *GPMC Not Supported Features*.

Table 12-197. NAND Memory Type

Subprocess Name	Register/Bit Field	Value
Set the NAND protocol.	GPMC_CONFIG1_i[11-10] DEVICETYPE	0x2
Set a device size.	GPMC_CONFIG1_i[13-12] DEVICESIZE	x
Set the address and data multiplexing protocol to non-multiplexed attached device.	GPMC_CONFIG1_i[9] MUXADDDATA	0x0
Select a timing signals latencies factor.	GPMC_CONFIG1_i[4] TIMEPARAGRANULARITY	x
Set a synchronous or asynchronous mode and a single or multiple access for read and write operations.	See Section 12.3.4.4.5, *Set Memory Access*.	x

Table 12-198. NAND Chip-Select Configuration

Subprocess Name	Register/Bit Field	Value
Select the chip-select base address.	GPMC_CONFIG7_i[5-0] BASEADDRESS	x
Select the chip-select minimum granularity (16MB).	GPMC_CONFIG7_i[11-8] MASKADDRESS	x

Table 12-199. Asynchronous Read and Write Operations

Subprocess Name	Register/Bit Field	Value
Configure adequate timing parameters in asynchronous modes	See Section 12.3.4.4.6, *GPMC Timing Parameters*.	

Table 12-200. ECC Engine

Subprocess Name	Register/Bit Field	Value
Select the ECC result register where the first ECC computation is stored (applies only to Hamming).	GPMC_ECC_CONTROL[3-0] ECCPOINTER	x [(2)]
Clear all ECC result registers.	GPMC_ECC_CONTROL[8] ECCCLEAR	Write 1 to clear.
Define ECCSIZE0 and ECCSIZE1.	GPMC_ECC_SIZE_CONFIG[21-12] ECCSIZE0 and [31-22] ECCSIZE1	x [(1)]
Select the size of each of the 9 result registers (size specified by ECCSIZE0 or ECCSIZE1).	GPMC_ECC_SIZE_CONFIG[j-1] ECCjRESULTSIZE where j = 1 to 9	x
Select the chip-select where ECC is computed.	GPMC_ECC_CONFIG[3-1] ECCCS	x
Select the Hamming code or BCH code ECC algorithm in use.	GPMC_ECC_CONFIG[16] ECCALGORITHM	x
Select word size for ECC calculation.	GPMC_ECC_CONFIG[7] ECC16B	x
If the BCH code is used, Set an error correction capability and Select a number of sectors to process.	GPMC_ECC_CONFIG[13-12] ECCBCHTSEL and GPMC_ECC_CONFIG[6-4] ECCTOPSECTOR	x

Copyright © 2024 Texas Instruments Incorporated

Table 12-200. ECC Engine (continued)

Subprocess Name	Register/Bit Field	Value
Enable the ECC computation.	GPMC_ECC_CONFIG[0] ECCENABLE	0x1

(1) Depends on the size of each sector in the NAND page
(2) This parameter depends on the numbers of sectors in a page.

Table 12-201. Prefetch and Write-Posting Engine

Subprocess Name	Register/Bit Field	Value
Disable the engine before configuration.	GPMC_PREFETCH_CONTROL[0] STARTENGINE	0x0
Select the chip-select associated with a NAND device where the prefetch engine is active.	GPMC_PREFETCH_CONFIG1[26-24] ENGINECSSELECTOR	x
Select access direction through prefetch engine, read or write.	GPMC_PREFETCH_CONFIG1[0] ACCESSMODE	x
Select the threshold used to issue an interrupt request.	GPMC_PREFETCH_CONFIG1[14-8] FIFOTHRESHOLD	x
Select interrupt synchronization mode.	GPMC_PREFETCH_CONFIG1[2] DMAMODE	x
Select if the engine immediately starts accessing the memory upon STARTENGINE assertion or if hardware synchronization based on a WAIT signal is used.	GPMC_PREFETCH_CONFIG1[3] SYNCHROMODE	x
Select which WAIT pin edge detector should start the engine in synchronized mode.	GPMC_PREFETCH_CONFIG1[5-4] WAITPINSELECTOR	x
Enter a number of clock cycles removed to timing parameters (for all back-to-back accesses to the NAND flash except the first one).	GPMC_PREFETCH_CONFIG1[30-28] CYCLEOPTIMIZATION	x
Enable the prefetch postwrite engine.	GPMC_PREFETCH_CONFIG1[7] ENABLEENGINE	0x1
Select the number of bytes to be read or written by the engine to the selected chip-select.	GPMC_PREFETCH_CONFIG2[13-0] TRANSFERCOUNT	x
Start the prefetch engine.	GPMC_PREFETCH_CONTROL[0] STARTENGINE	0x1

Table 12-202. WAIT Pin Configuration

Subprocess Name	Register/Bit Field	Value
Selects when the engine starts the access to chip-select.	GPMC_PREFETCH_CONFIG1[3] SYNCHROMODE	x
Select which WAIT pin edge detector should start the engine in synchronized mode.	GPMC_PREFETCH_CONFIG1[5-4] WAITPINSELECTOR	x

Table 12-203. Enable Chip-Select

Subprocess Name	Register/Bit Field	Value
When all parameters are configured, enable the chip-select.	GPMC_CONFIG7_i[6] CSVALID	x

12.3.4.4.5 Set Memory Access

Note

Some of the GMPC features described in this section may not be supported on this family of devices. For more information, see *GPMC Not Supported Features*.

This section describes the bit field to configure to set the GPMC in various memory modes. Table 12-204 and Table 12-205 provide check lists for mode parameters and access type parameters, respectively.

Table 12-204. Mode Parameters Check List

Register	Bit	Name	Asynchronous				Synchronous			
			Single Read Access	Single Write Access	Multiple Read (Page) Access	Multiple Write (Page) Access	Single Read Access	Single Write Access	Multiple Read (Burst) Access	Multiple Write (Burst) Access
GPMC_CONFIG1_i	30	READMULTIPLE	0x0	Don't care	0x1 (1)	Not Supported	0x0	Don't care	0x1	Don't care
GPMC_CONFIG1_i	29	READTYPE	0x0	Don't care	0x0 (1)	Not Supported	0x1	Don't care	0x1	Don't care

Copyright © 2024 Texas Instruments Incorporated

Table 12-204. Mode Parameters Check List (continued)

Register	Bit	Name	Asynchronous				Synchronous			
GPMC_CONFIG1_i	28	WRITEMULTIPLE	Don't care	0x0	- (1)	Not Supported	Don't care	0x0	Don't care	0x1
GPMC_CONFIG1_i	27	WRITETYPE	Don't care	0x0	- (1)	Not Supported	Don't care	0x1	Don't care	0x1

(1) Multiple read is not supported in address/data-multiplexed and AAD-multiplexed modes. Multiple read is supported in non-multiplexed mode.

Table 12-205. Access Type Parameters Check List

Register	Bit	Name	Access Type		
			non-multiplexed	Address/ Data-Multiplexed	AAD-Multiplexed
GPMC_CONFIG1_i	9-8	MUXADDDATA	0x0	0x2	0x1

12.3.4.4.6 GPMC Timing Parameters

Figure 12-105 shows a programming model diagram for the NOR interfacing timing parameters.

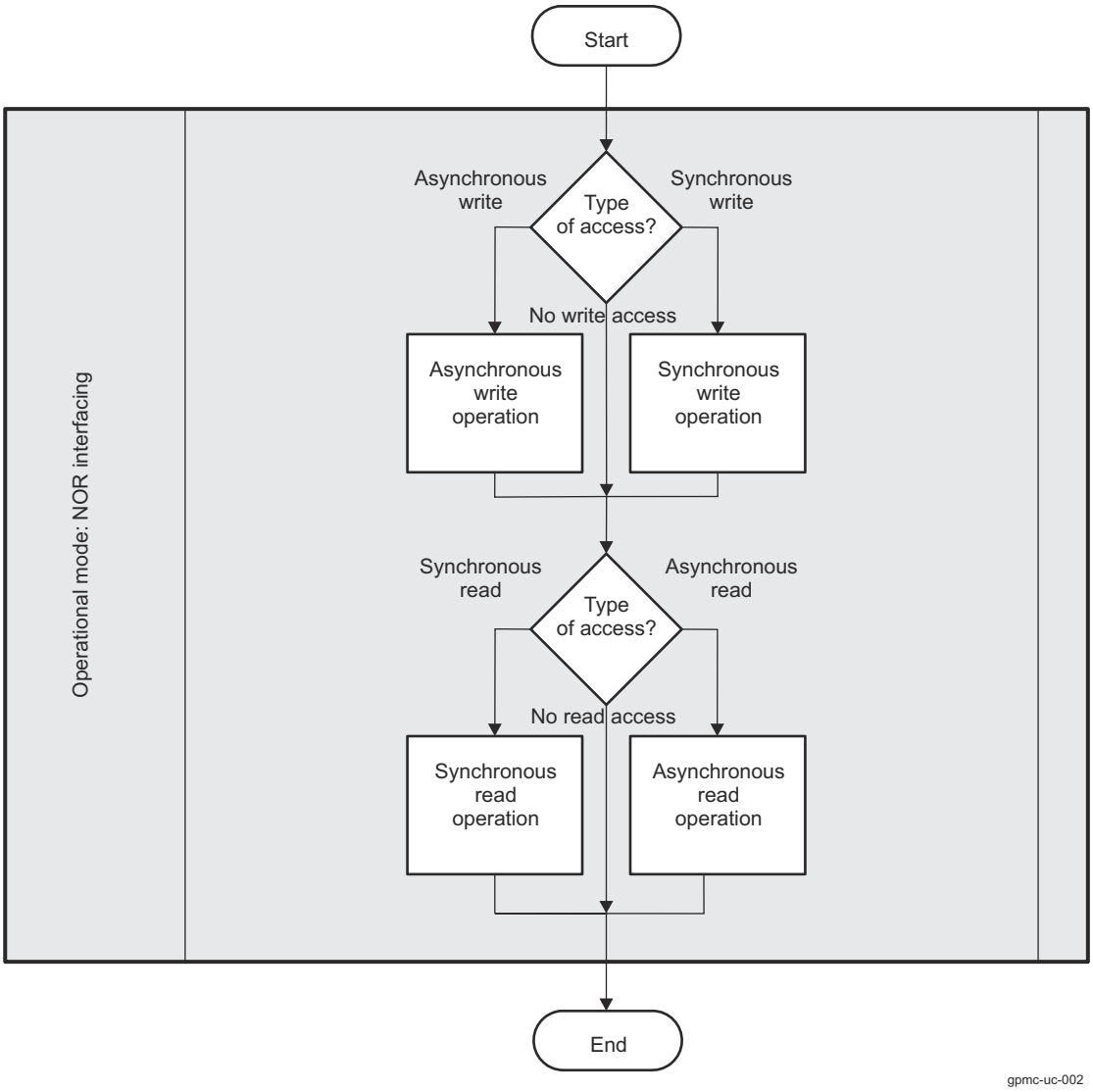


Figure 12-105. NOR Interfacing Timing Parameters Diagram

Table 12-206 lists the bit fields to configure adequate timing parameters in various memory modes.

Copyright © 2024 Texas Instruments Incorporated

Table 12-206. Timing Parameters

			Asynchronous			Synchronous				Access Type		
Register	Bit	Name	Single Read Access	Single Write Access	Multiple Read (Page) access	Single Read Access	Single Write Access	Multiple Read (Burst) Access	Multiple Write (Burst) Access	Non-multiplexed	Address/ Data-Multiplexed	AAD Multiplexed
GPMC_CONFIG1_i	9	MUXADDDATA	y	y	y	y	y	y	y	y	y	y
GPMC_CONFIG1_i	29	READTYPE	y		y	y		y		y	y	y
GPMC_CONFIG1_i	30	READMULTIPLE	y		y	y		y		y	y	y
GPMC_CONFIG1_i	27	WRITETYPE		y			y		y	y	y	y
GPMC_CONFIG1_i	28	WRITEMULTIPLE		y			y		y	y	y	y
GPMC_CONFIG1_i	31	WRAPBURST						y	y	y	y	y
GPMC_CONFIG1_i	26-25	CLKACTIVATIONTIME				y	y	y	y	y	y	y
GPMC_CONFIG1_i	19-18	WAITMONITORINGTIME	y	y	y	y	y	y	y	y	y	y
GPMC_CONFIG1_i	4	TIMEPARAGRANULARITY	y	y	y	y	y	y	y	y	y	y
GPMC_CONFIG2_i	20-16	CSWROFFTIME		y			y		y	y	y	y
GPMC_CONFIG2_i	12-8	CSRDOFFTIME	y		y	y		y		y	y	y
GPMC_CONFIG2_i	7	CSEXTRADELAY	y	y	y	y	y	y	y	y	y	y
GPMC_CONFIG2_i	3-0	CSONTIME	y	y	y	y	y	y	y	y	y	y
GPMC_CONFIG3_i	30-28	ADVAADMUXWROFFTIME		y			y		y			y
GPMC_CONFIG3_i	30-29	ADVAADMUXRDOFFTIME	y		y	y		y				y
GPMC_CONFIG3_i	6-4	ADVAADMUXONTIME	y	y	y	y	y	y	y			y
GPMC_CONFIG3_i	20-16	ADVWROFFTIME		y			y		y	y	y	y

Copyright © 2024 Texas Instruments Incorporated

Table 12-206. Timing Parameters (continued)

			Asynchronous				Synchronous			Access Type		
GPMC_CONFIG3_i	12-8	ADVRDOFFTIME	y		y	y		y		y	y	y
GPMC_CONFIG3_i	7	ADVEXTRADELAY	y	y	y	y	y	y	y	y	y	y
GPMC_CONFIG3_i	3-0	ADVONTIME	y	y	y	y	y	y	y	y	y	y
GPMC_CONFIG4_i	15-13	OEAADMUXOFFTIME	y	y	y	y	y	y	y			y
GPMC_CONFIG4_i	6-4	OEAADMUXONTIME	y	y	y	y	y	y	y			y
GPMC_CONFIG4_i	28-24	WEOFFTIME		y			y		y	y	y	y
GPMC_CONFIG4_i	23	WEEXTRADELAY		y			y		y	y	y	y
GPMC_CONFIG4_i	19-16	WEONTIME		y			y		y	y	y	y
GPMC_CONFIG4_i	12-8	OEOFFTIME	y		y	y		y		y	y	y
GPMC_CONFIG4_i	7	OEEXTRADELAY	y		y	y		y		y	y	y
GPMC_CONFIG4_i	3-0	OEONTIME	y		y	y		y		y	y	y
GPMC_CONFIG5_i	27-24	PAGEBURSTACCESSTIME			y			y	y	y	y	y
GPMC_CONFIG5_i	20-16	RDACCESSTIME	y		y	y		y		y	y	y
GPMC_CONFIG5_i	12-8	WRCYCLETIME		y			y		y	y	y	y
GPMC_CONFIG5_i	4-0	RDCYCLETIME	y		y	y		y		y	y	y
GPMC_CONFIG6_i	28-24	WRACCESSTIME		y			y		y	y	y	y
GPMC_CONFIG6_i	19-16	WRDATAONADMUXBUS		y			y		y		y	y
GPMC_CONFIG6_i	11-8	CYCLE2CYCLEDELAY	y	y	y	y	y	y	y	y	y	y
GPMC_CONFIG6_i	7	CYCLE2CYCLESAMECSEN	y	y	y	y	y	y	y	y	y	y

Copyright © 2024 Texas Instruments Incorporated

Table 12-206. Timing Parameters (continued)

			Asynchronous				Synchronous			Access Type		
GPMC_CONFIG6_i	6	CYCLE2CYCLEDIFFCSEN	y	y	y	y	y	y	y	y	y	y
GPMC_CONFIG6_i	3-0	BUSTURNAROUND	y	y	y	y	y	y	y	y	y	y
GPMC_CONFIG7_i	6	CSVALID	y	y	y	y	y	y	y	y	y	y

12.3.4.4.6.1 GPMC Timing Parameters Formulas

This section is intended to help the user calculate the GPMC timing bit field values. Formulas are not listed exhaustively.

The section describes:

- NAND flash interface timing parameters formulas
- Synchronous NOR flash timing parameters formulas
- Asynchronous NOR flash timing parameters formulas

For complete information, such as OPP and board effects on timings, see the device-specific Datasheet.

12.3.4.4.6.1.1 NAND Flash Interface Timing Parameters Formulas

This section lists formulas to calculate NAND timing parameters. This is the case when GPMC_CONFIG1_i[11-10] DEVICETYPE = 0x2. Table 12-207 describes the NAND timing parameters.

Table 12-207. NAND Formulas Description

Configuration Parameter	Unit	Description
A	ns	Pulse duration – GPMC_WEn valid time
B	ns	Delay time – GPMC_CS valid to GPMC_WEn valid
C	ns	Delay time – GPMC_BE0n_CLE/GPMC_ADVn_ALE high to GPMC_WEn valid
D	ns	Delay time – GPMC_AD[15-0] valid to GPMC_WEn valid
E	ns	Delay time – GPMC_WEn invalid to GPMC_AD[15-0] invalid
F	ns	Delay time – GPMC_WEn invalid to GPMC_BE0n_CLE/GPMC_ADVn_ALE invalid
G	ns	Delay time – GPMC_WEn invalid to GPMC_CS invalid
H	ns	Cycle time – Write cycle time
I	ns	Delay time – GPMC_CS valid to GPMC_OEn_REn valid
J	ns	Setup time – GPMC_AD[15-0] valid to GPMC_OEn_REn invalid
K	ns	Pulse duration – GPMC_OEn_REn valid time
L	ns	Cycle time – Read cycle time
M	ns	Delay time – GPMC_OEn_REn invalid to GPMC_CS invalid

The configuration parameters are calculated through the following formulas. For more information, see the device-specific Datasheet.

A = (WEOffTime – WEOnTime) * (TimeParaGranularity + 1) * GPMC_FCLK period

B = ((WEOnTime – CSOnTime) * (TimeParaGranularity + 1) + 0.5 * (WEExtraDelay – CSExtraDelay)) * GPMC_FCLK period

C = ((WEOnTime – ADVOnTime) * (TimeParaGranularity + 1) + 0.5 * (WEExtraDelay - ADVExtraDelay)) * GPMC_FCLK period

D = (WEOnTime * (TimeParaGranularity + 1) + 0.5 * WEExtraDelay) * GPMC_FCLK period

E = (WrCycleTime – WEOffTime * (TimeParaGranularity + 1) – 0.5 * WEExtraDelay) * GPMC_FCLK period

F = (ADVWrOffTime – WEOffTime * (TimeParaGranularity + 1) + 0.5 * (ADVExtraDelay – WEExtraDelay) * GPMC_FCLK period

G = (CSWrOffTime – WEOffTime * (TimeParaGranularity + 1) + 0.5 * (CSExtraDelay – WEExtraDelay) * GPMC_FCLK period

Copyright © 2024 Texas Instruments Incorporated

H = WrCycleTime * (1 + TimeParaGranularity) * GPMC_FCLK period

I = ((OEOnTime – CSOnTime) * (TimeParaGranularity + 1) + 0.5 * (OEExtraDelay – CSExtraDelay)) * GPMC_FCLK period

J = ((RdAccessTime – OEOffTime) * (TimeParaGranularity + 1) – 0.5 * OEExtraDelay)) * GPMC_FCLK period

K = (OEOffTime – OEOnTime) * (1 + TimeParaGranularity) * GPMC_FCLK period

L = RdCycleTime * (1 + TimeParaGranularity) * GPMC_FCLK period

M = (CSRdOffTime – OEOffTime * (TimeParaGranularity + 1) + 0.5 * (CSExtraDelay - OEExtraDelay) * GPMC_FCLK period

Figure 12-106 shows a simplified example of command latch cycle timing where formulas are associated with signal waves.

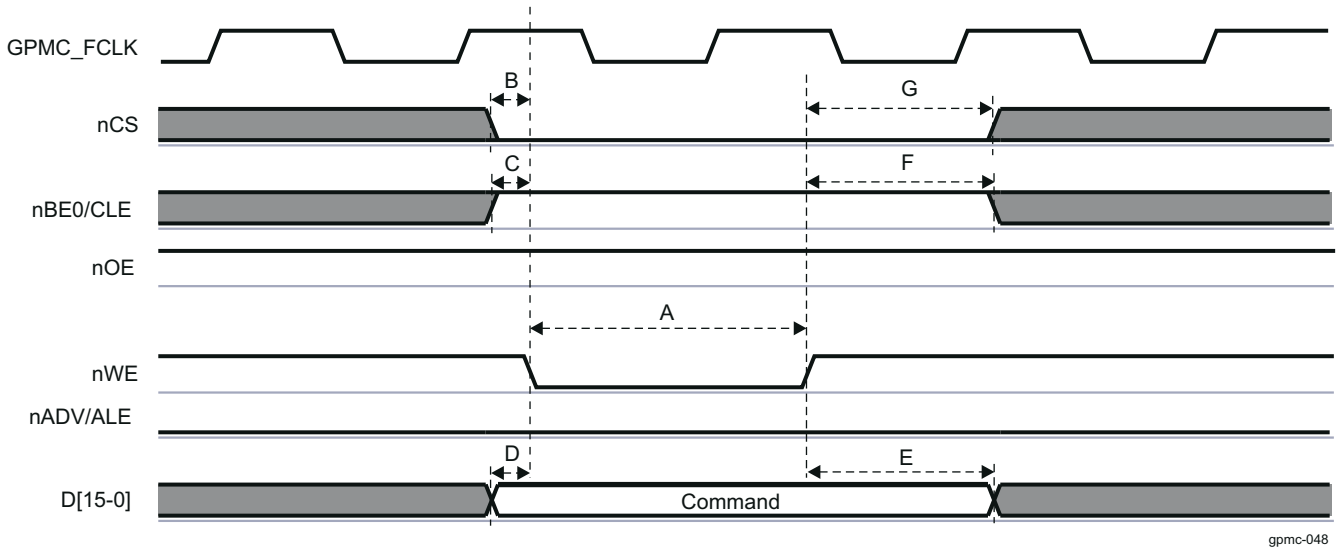


Figure 12-106. NAND Command Latch Cycle Timing Simplified Example

12.3.4.4.6.1.2 Synchronous NOR Flash Timing Parameters Formulas

This section lists all formulas to calculate synchronous NOR timing parameters. This is the case when GPMC_CONFIG1_i[11-10] DEVICETYPE = 0x0 and when READTYPE or WRITETYPE are set to synchronous mode. Table 12-208 describes the synchronous NOR formulas.

Table 12-208. Synchronous NOR Formulas Description

Configuration Parameter	Unit	Description
A	ns	Pulse duration – nCS low
B	ns	Delay time – address bus valid to CLK first edge
		Delay time – nBE0 / nBE1 valid to CLK first edge
C	ns	Pulse duration – nBE0 / nBE1 low
D	ns	Delay time – CLK rising edge to nBE0 / nBE1 invalid
		Delay time – CLK rising edge to nADV/ALE invalid
E	ns	Delay time – CLK rising edge to nCS invalid
		Delay time – CLK rising edge to nOE/nRE invalid
F	ns	Delay time – CLK rising edge to nCS transition
G	ns	Delay time – CLK rising edge to nADV/ALE transition
H	ns	Delay time – CLK rising edge to nOE/nRE transition
I	ns	Delay time – CLK rising edge to nWE transition
J	ns	Delay time – CLK rising edge to A[16-1/]D[15-0] data bus transition
		Delay time – CLK rising edge to nBE0 / nBE1 transition
K	ns	Pulse duration – nADV/ALE low
L	ns	Delay time – WAIT invalid to first data latching CLK edge

Copyright © 2024 Texas Instruments Incorporated

The configuration parameters are calculated through the following formulas. For more information, see the device-specific Datasheet.

1. For single read accesses:
   A = (CSRDOFFTIME – CSONTIME) * (TIMEPARAGRANULARITY + 1) * GPMC_FCLK period
   C = RDCYCLETIME * (TIMEPARAGRANULARITY + 1) * GPMC_FCLK period
   D = (RDCYCLETIME – RDACCESSTIME) * GPMC_FCLK period
   E = (CSRDOFFTIME – RDACCESSTIME) * GPMC_FCLK period
2. For burst read accesses (where n is the page burst access number):
   A = (CSRDOFFTIME – CSONTIME + (n – 1) * PAGEBURSTACCESSTIME) * (TIMEPARAGRANULARITY + 1) * GPMC_FCLK period
   C = (RDCYCLETIME + (n – 1) * PAGEBURSTACCESSTIME) * (TIMEPARAGRANULARITY + 1) * GPMC_FCLK period
   D = (RDCYCLETIME – (RDACCESSTIME + (n – 1) * PAGEBURSTACCESSTIME)) * GPMC_FCLK period
   E = (CSRDOFFTIME – (RDACCESSTIME + (n – 1) * PAGEBURSTACCESSTIME)) * GPMC_FCLK period
3. For burst write accesses (where n is the page burst access number):
   A = (CSWROFFTIME – CSONTIME + (n – 1) * PAGEBURSTACCESSTIME) * (TIMEPARAGRANULARITY + 1) * GPMC_FCLK period
   C = (WRCYCLETIME + (n – 1) * PAGEBURSTACCESSTIME) * (TIMEPARAGRANULARITY + 1) * GPMC_FCLK period
   D = (WRCYCLETIME – (RDACCESSTIME + (n – 1) * PAGEBURSTACCESSTIME)) * GPMC_FCLK period
   E = (CSWROFFTIME – (RDACCESSTIME + (n – 1) * PAGEBURSTACCESSTIME)) * GPMC_FCLK period
4. For all accesses:
   For nCS falling edge (chip-select activated):
   - Case where GPMC_CONFIG1_i[1-0] GPMCFCLKDIVIDER = 0x0:
     F = 0.5 * CSEXTRADELAY * GPMC_FCLK period
   - Case where GPMCFCLKDIVIDER = 0x1:
     F = 0.5 * CSEXTRADELAY * GPMC_FCLK period, when (CLKACTIVATIONTIME and CSONTIME are odd) or (CLKACTIVATIONTIME and CSONTIME are even)
     F = (1 + 0.5 * CSEXTRADELAY) * GPMC_FCLK period otherwise.
   - Case where GPMCFCLKDIVIDER = 0x2:
     F = 0.5 * CSEXTRADELAY * GPMC_FCLK period, when (CSONTIME – CLKACTIVATIONTIME) is a multiple of 3
     F = (1 + 0.5 * CSEXTRADELAY) * GPMC_FCLK period, when (CSONTIME – CLKACTIVATIONTIME – 1) is a multiple of 3
     F = (2 + 0.5 * CSEXTRADELAY) * GPMC_FCLK period, when (CSONTIME – CLKACTIVATIONTIME – 2) is a multiple of 3

   For nCS rising edge (chip-select deactivated) in reading mode:
   - Case where GPMC_CONFIG1_i[1-0] GPMCFCLKDIVIDER = 0x0:
     F = 0.5 * CSEXTRADELAY * GPMC_FCLK period
   - Case where GPMCFCLKDIVIDER = 0x1:
     F = 0.5 * CSEXTRADELAY * GPMC_FCLK period, when (CLKACTIVATIONTIME and CSRDOFFTIME are odd) or (CLKACTIVATIONTIME and CSRDOFFTIME are even)
     F = (1 + 0.5 * CSEXTRADELAY) * GPMC_FCLK period otherwise.
   - Case where GPMCFCLKDIVIDER = 0x2:
     F = 0.5 * CSEXTRADELAY * GPMC_FCLK period, when (CSRDOFFTIME – CLKACTIVATIONTIME) is a multiple of 3
     F = (1 + 0.5 * CSEXTRADELAY) * GPMC_FCLK period, when (CSRDOFFTIME – CLKACTIVATIONTIME – 1) is a multiple of 3
     F = (2 + 0.5 * CSEXTRADELAY) * GPMC_FCLK period, when (CSRDOFFTIME – CLKACTIVATIONTIME – 2) is a multiple of 3

   For nCS rising edge (chip-select deactivated) in writing mode:
   - Case where GPMC_CONFIG1_i[1-0] GPMCFCLKDIVIDER = 0x0:
     F = 0.5 * CSEXTRADELAY * GPMC_FCLK period
   - Case where GPMCFCLKDIVIDER = 0x1:

Copyright © 2024 Texas Instruments Incorporated

F = 0.5 * CSEXTRADELAY * GPMC_FCLK period, when (CLKACTIVATIONTIME and CSWROFFTIME are odd) or (CLKACTIVATIONTIME and CSWROFFTIME are even)
F = (1 + 0.5 * CSEXTRADELAY) * GPMC_FCLK period otherwise.

- Case where GPMCFCLKDIVIDER = 0x2:
  F = 0.5 * CSEXTRADELAY * GPMC_FCLK period, when (CSWROFFTIME – CLKACTIVATIONTIME) is a multiple of 3
  F = (1 + 0.5 * CSEXTRADELAY) * GPMC_FCLK period, when (CSWROFFTIME – CLKACTIVATIONTIME – 1) is a multiple of 3
  F = (2 + 0.5 * CSEXTRADELAY) * GPMC_FCLK period, when (CSWROFFTIME – CLKACTIVATIONTIME – 2) is a multiple of 3

For nADV falling edge (nADV activated):

- Case where GPMC_CONFIG1_i[1-0] GPMCFCLKDIVIDER = 0x0:
  G = 0.5 * ADVEXTRADELAY * GPMC_FCLK period
- Case where GPMCFCLKDIVIDER = 0x1:
  G = 0.5 * ADVEXTRADELAY * GPMC_FCLK period, when (CLKACTIVATIONTIME and ADVONTIME are odd) or (CLKACTIVATIONTIME and ADVONTIME are even)
  G = (1 + 0.5 * ADVEXTRADELAY) * GPMC_FCLK period otherwise.
- Case where GPMCFCLKDIVIDER = 0x2:
  G = 0.5 * ADVEXTRADELAY * GPMC_FCLK period, when (ADVONTIME – CLKACTIVATIONTIME) is a multiple of 3
  G = (1 + 0.5 * ADVEXTRADELAY) * GPMC_FCLK period, when (ADVONTIME – CLKACTIVATIONTIME – 1) is a multiple of 3
  G = (2 + 0.5 * ADVEXTRADELAY) * GPMC_FCLK period, when (ADVONTIME – CLKACTIVATIONTIME – 2) is a multiple of 3

For nADV rising edge (nADV deactivated) in reading mode:

- Case where GPMC_CONFIG1_i[1-0] GPMCFCLKDIVIDER = 0x0:
  G = 0.5 * ADVEXTRADELAY * GPMC_FCLK period
- Case where GPMCFCLKDIVIDER = 0x1:
  G = 0.5 * ADVEXTRADELAY * GPMC_FCLK period, when (CLKACTIVATIONTIME and ADVRDOFFTIME are odd) or (CLKACTIVATIONTIME and ADVRDOFFTIME are even)
  G = (1 + 0.5 * ADVEXTRADELAY) * GPMC_FCLK period otherwise.
- Case where GPMCFCLKDIVIDER = 0x2:
  G = 0.5 * ADVEXTRADELAY * GPMC_FCLK period, when (ADVRDOFFTIME - CLKACTIVATIONTIME) is a multiple of 3
  G = (1 + 0.5 * ADVEXTRADELAY) * GPMC_FCLK period, when (ADVRDOFFTIME - CLKACTIVATIONTIME - 1) is a multiple of 3
  G = (2 + 0.5 * ADVEXTRADELAY) * GPMC_FCLK period, when (ADVRDOFFTIME - CLKACTIVATIONTIME - 2) is a multiple of 3

For nADV rising edge (nADV deactivated) in writing mode:

- Case where GPMC_CONFIG1_i[1-0] GPMCFCLKDIVIDER = 0x0:
  G = 0.5 * ADVEXTRADELAY * GPMC_FCLK period
- Case where GPMCFCLKDIVIDER = 0x1:
  G = 0.5 * ADVEXTRADELAY * GPMC_FCLK period, when (CLKACTIVATIONTIME and ADVWROFFTIME are odd) or (CLKACTIVATIONTIME and ADVWROFFTIME are even)
  G = (1 + 0.5 * ADVEXTRADELAY) * GPMC_FCLK period otherwise.
- Case where GPMCFCLKDIVIDER = 0x2:
  G = 0.5 * ADVEXTRADELAY * GPMC_FCLK period, when (ADVWROFFTIME – CLKACTIVATIONTIME) is a multiple of 3
  G = (1 + 0.5 * ADVEXTRADELAY) * GPMC_FCLK period, when (ADVWROFFTIME – CLKACTIVATIONTIME – 1) is a multiple of 3
  G = (2 + 0.5 * ADVEXTRADELAY) * GPMC_FCLK period, when (ADVWROFFTIME – CLKACTIVATIONTIME – 2) is a multiple of 3

For nOE falling edge (nOE activated):

- Case where GPMC_CONFIG1_i[1-0] GPMCFCLKDIVIDER = 0x0:

Copyright © 2024 Texas Instruments Incorporated

H = 0.5 * OEEXTRADELAY * GPMC_FCLK period

- Case where GPMCFCLKDIVIDER = 0x1:
  H = 0.5 * OEEXTRADELAY * GPMC_FCLK period, when (CLKACTIVATIONTIME and OEONTIME are odd) or (CLKACTIVATIONTIME and OEONTIME are even)
  H = (1 + 0.5 * OEEXTRADELAY) * GPMC_FCLK period otherwise.
- Case where GPMCFCLKDIVIDER = 0x2:
  H = 0.5 * OEEXTRADELAY * GPMC_FCLK period, when (OEONTIME – CLKACTIVATIONTIME) is a multiple of 3
  H = (1 + 0.5 * OEEXTRADELAY) * GPMC_FCLK period, when (OEONTIME – CLKACTIVATIONTIME – 1) is a multiple of 3
  H = (2 + 0.5 * OEEXTRADELAY) * GPMC_FCLK period, when (OEONTIME – CLKACTIVATIONTIME – 2) is a multiple of 3

For nOE rising edge (nOE deactivated):

- Case where GPMC_CONFIG1_i[1-0] GPMCFCLKDIVIDER = 0x0:
  H = 0.5 * OEEXTRADELAY * GPMC_FCLK period
- Case where GPMCFCLKDIVIDER = 0x1:
  H = 0.5 * OEEXTRADELAY * GPMC_FCLK period, when (CLKACTIVATIONTIME and OEOFFTIME are odd) or (CLKACTIVATIONTIME and OEOFFTIME are even)
  H = (1 + 0.5 * OEEXTRADELAY) * GPMC_FCLK period otherwise.
- Case where GPMCFCLKDIVIDER = 0x2:
  H = 0.5 * OEEXTRADELAY * GPMC_FCLK period, when (OEOFFTIME – CLKACTIVATIONTIME) is a multiple of 3
  H = (1 + 0.5 * OEEXTRADELAY) * GPMC_FCLK period, when (OEOFFTIME – CLKACTIVATIONTIME – 1) is a multiple of 3
  H = (2 + 0.5 * OEEXTRADELAY) * GPMC_FCLK period, when (OEOFFTIME – CLKACTIVATIONTIME – 2) is a multiple of 3

For nWE falling edge (nWE activated):

- Case where GPMC_CONFIG1_i[1-0] GPMCFCLKDIVIDER = 0x0:
  I = 0.5 * WEEXTRADELAY * GPMC_FCLK period
- Case where GPMCFCLKDIVIDER = 0x1:
  I = 0.5 * WEEXTRADELAY * GPMC_FCLK period, when (CLKACTIVATIONTIME and WEONTIME are odd) or (CLKACTIVATIONTIME and WEONTIME are even)
  I = (1 + 0.5 * WEEXTRADELAY) * GPMC_FCLK period otherwise.
- Case where GPMCFCLKDIVIDER = 0x2:
  I = 0.5 * WEEXTRADELAY * GPMC_FCLK period, when (WEONTIME – CLKACTIVATIONTIME) is a multiple of 3
  I = (1 + 0.5 * WEEXTRADELAY) * GPMC_FCLK period, when (WEONTIME – CLKACTIVATIONTIME – 1) is a multiple of 3
  I = (2 + 0.5 * WEEXTRADELAY) * GPMC_FCLK period, when (WEONTIME – CLKACTIVATIONTIME – 2) is a multiple of 3

For nWE rising edge (nWE deactivated):

- Case where GPMC_CONFIG1_i[1-0] GPMCFCLKDIVIDER = 0x0:
  I = 0.5 * WEEXTRADELAY * GPMC_FCLK period
- Case where GPMCFCLKDIVIDER = 0x1:
  I = 0.5 * WEEXTRADELAY * GPMC_FCLK period, when (CLKACTIVATIONTIME and WEOFFTIME are odd) or (CLKACTIVATIONTIME and WEOFFTIME are even)
  I = (1 + 0.5 * WEEXTRADELAY) * GPMC_FCLK period otherwise.
- Case where GPMCFCLKDIVIDER = 0x2:
  I = 0.5 * WEEXTRADELAY * GPMC_FCLK period, when (WEOFFTIME – CLKACTIVATIONTIME) is a multiple of 3
  I = (1 + 0.5 * WEEXTRADELAY) * GPMC_FCLK period, when (WEOFFTIME – CLKACTIVATIONTIME – 1) is a multiple of 3
  I = (2 + 0.5 * WEEXTRADELAY) * GPMC_FCLK period, when (WEOFFTIME – CLKACTIVATIONTIME – 2) is a multiple of 3

Copyright © 2024 Texas Instruments Incorporated

For nADV low pulse duration:

- Read operation:
  K = (ADVRDOFFTIME – ADVONTIME) * (TIMEPARAGRANULARITY + 1) * GPMC_FCLK period
- Write operation:
  K = (ADVWROFFTIME – ADVONTIME) * (TIMEPARAGRANULARITY + 1) * GPMC_FCLK period

For WAIT invalid to first data latching GPMC output clock edge:

- L = WAITMONITORINGTIME * (GPMCFCLKDIVIDER + 1) * GPMC_FCLK period + GPMC output clock period

Figure 12-107 shows a simplified example of a synchronous NOR single read where formulas are associated with signal waves.

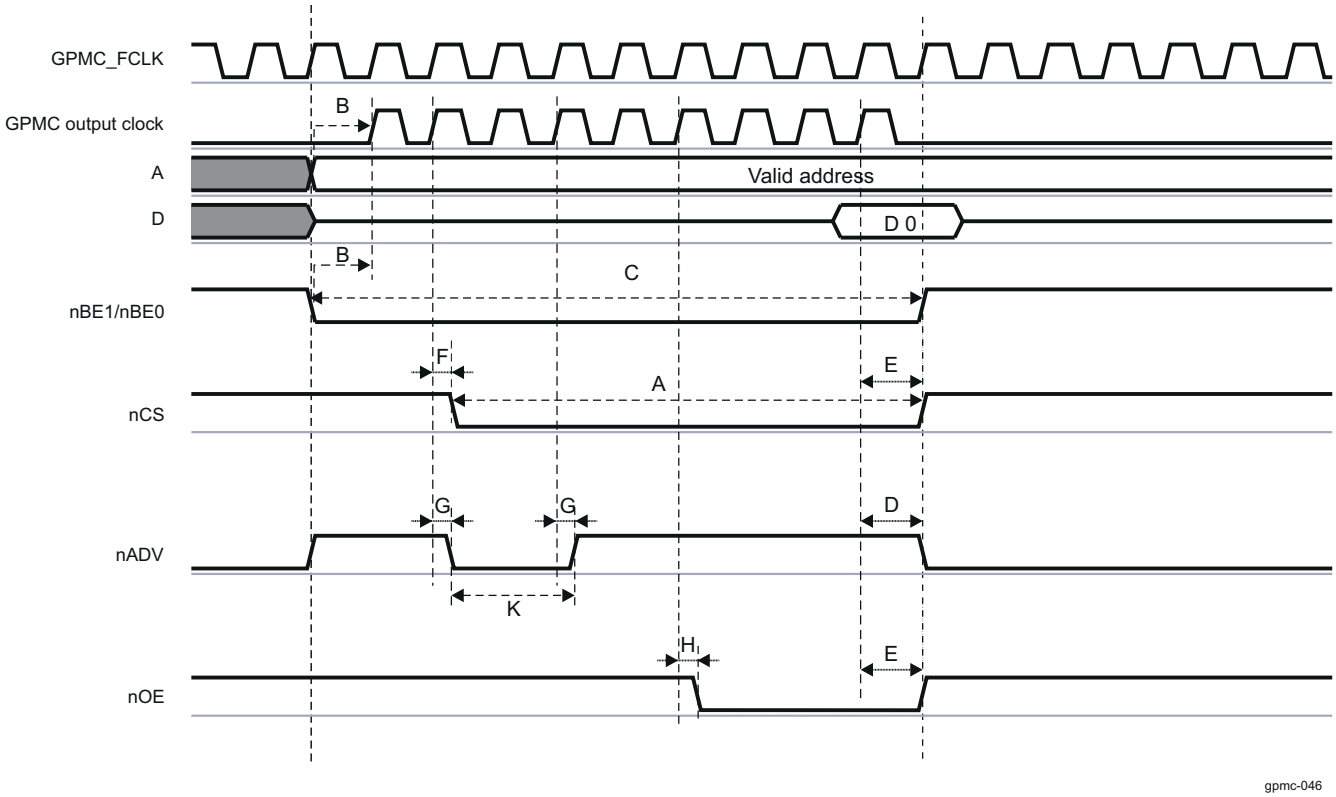


Figure 12-107. Synchronous NOR Single Read Simplified Example

12.3.4.4.6.1.3 Asynchronous NOR Flash Timing Parameters Formulas

This section lists all the formulas to calculate asynchronous NOR timing parameters. This is the case when GPMC_CONFIG1_i[11-10] DEVICETYPE = 0x0 and when READTYPE or WRITETYPE are set to asynchronous mode. Table 12-209 describes the asynchronous NOR formulas.

Table 12-209. Asynchronous NOR Formulas Description

Configuration Parameter	Unit	Description
A	ns	Pulse duration – nCS low
B	ns	Delay time – nCS valid to nADV/ALE invalid
C	ns	Delay time – nCS valid to nOE/nRE invalid (single read)
D	ns	Pulse duration – address bus valid - 2nd, 3rd and 4th accesses
E	ns	Delay time – nCS valid to nWE valid
F	ns	Delay time – nCS valid to nWE invalid
G	ns	Address invalid duration between two successive R/W accesses
H	ns	Setup time – read data valid before nOE/nRE high

Copyright © 2024 Texas Instruments Incorporated

Table 12-209. Asynchronous NOR Formulas Description (continued)

Configuration Parameter	Unit	Description
I	ns	Delay time – nCS valid to nOE/nRE invalid (burst read)
J	ns	Delay time – address bus valid to nCS valid
		Delay time – data bus valid to nCS valid
		Delay time – nBE0 / nBE1 valid to nCS valid
K	ns	Delay time – nCS valid to nADV/ALE valid
L	ns	Delay time – nCS valid to nOE/nRE valid
M	ns	Delay time – nCS valid to first data latching edge
N	ns	Pulse duration – nBE0 / nBE1 valid time
O	ns	Delay time – nCS valid to nADV/ALE valid

The configuration parameters are calculated through the following formulas. These formulas are not exhaustive. For more information, see the device-specific Datasheet.

- nCS low pulse:
  For single read: A = (CSRDOFFTIME – CSONTIME) * (TIMEPARAGRANULARITY + 1) * GPMC_FCLK period
  For burst read: A = (CSRDOFFTIME – CSONTIME + (N – 1) * PAGEBURSTACCESSTIME) * (TIMEPARAGRANULARITY + 1) * GPMC_FCLK period, where N = page burst access number
  For single write: A = (CSWROFFTIME – CSONTIME) * (TIMEPARAGRANULARITY + 1) * GPMC_FCLK period
  For burst write: A = (CSWROFFTIME – CSONTIME + (N – 1) * PAGEBURSTACCESSTIME) * (TIMEPARAGRANULARITY + 1) * GPMC_FCLK period, where N = page burst access number
- nCS valid to nADV/ALE invalid delay:
  For reading: B = ((ADVRDOFFTIME – CSONTIME) * (TIMEPARAGRANULARITY + 1) + 0.5 * (ADVEXTRADELAY – CSEXTRADELAY)) * GPMC_FCLK period
  For writing: B = ((ADVWROFFTIME – CSONTIME) * (TIMEPARAGRANULARITY + 1) + 0.5 * (ADVEXTRADELAY – CSEXTRADELAY)) * GPMC_FCLK period
- C = ((OEOFFTIME – CSONTIME) * (TIMEPARAGRANULARITY + 1) + 0.5 * (OEEXTRADELAY – CSEXTRADELAY)) * GPMC_FCLK period
- D = PAGEBURSTACCESSTIME * (TIMEPARAGRANULARITY + 1) * GPMC_FCLK period
- E = ((WEONTIME – CSONTIME) * (TIMEPARAGRANULARITY + 1) + 0.5 * (WEEXTRADELAY – CSEXTRADELAY)) * GPMC_FCLK period
- F = ((WEOFFTIME – CSONTIME) * (TIMEPARAGRANULARITY + 1) + 0.5 * (WEEXTRADELAY – CSEXTRADELAY)) * GPMC_FCLK period
- G = CYCLE2CYCLEDELAY * GPMC_FCLK period
- H = ((OEOFFTIME – RDACCESSTIME) * (TIMEPARAGRANULARITY + 1) + 0.5 * OEEXTRADELAY) * GPMC_FCLK period
- I = ((OEOFFTIME + (N – 1) * PAGEBURSTACCESSTIME – CSONTIME) * (TIMEPARAGRANULARITY + 1) + 0.5 * (OEEXTRADELAY – CSEXTRADELAY)) * GPMC_FCLK period, where N = page burst access number
- J = (CSONTIME * (TIMEPARAGRANULARITY + 1) + 0.5 * CSEXTRADELAY) * GPMC_FCLK period
- K = ((ADVONTIME – CSONTIME) * (TIMEPARAGRANULARITY + 1) + 0.5 * (ADVEXTRADELAY – CSEXTRADELAY)) * GPMC_FCLK period
- L = ((OEONTIME – CSONTIME) * (TIMEPARAGRANULARITY + 1) + 0.5 * (OEEXTRADELAY – CSEXTRADELAY)) * GPMC_FCLK period
- M = ((RDACCESSTIME - CSONTIME) * (TIMEPARAGRANULARITY + 1) – 0.5 * CSEXTRADELAY) * GPMC_FCLK period
- nBE0 /nBE1 pulse:
  For single read: N = RDCYCLETIME * (TIMEPARAGRANULARITY + 1) * GPMC_FCLK period
  For burst read: N = (RDCYCLETIME + (N – 1) * PAGEBURSTACCESSTIME) * (TIMEPARAGRANULARITY + 1) * GPMC_FCLK period, where N = page burst access number

Copyright © 2024 Texas Instruments Incorporated

For burst write: N = (WRCYCLETIME + (N – 1) * PAGEBURSTACCESSTIME) * (TIMEPARAGRANULARITY + 1) * GPMC_FCLK period, where N = page burst access number

- O = ((WRCYCLETIME + (N – 1) * PAGEBURSTACCESSTIME – CSONTIME) * (TIMEPARAGRANULARITY + 1) + 0.5 * (ADVEXTRADELAY – CSEXTRADELAY)) * GPMC_FCLK period

Figure 12-108 shows a simplified example of an asynchronous NOR single write where formulas are associated with signal waves.

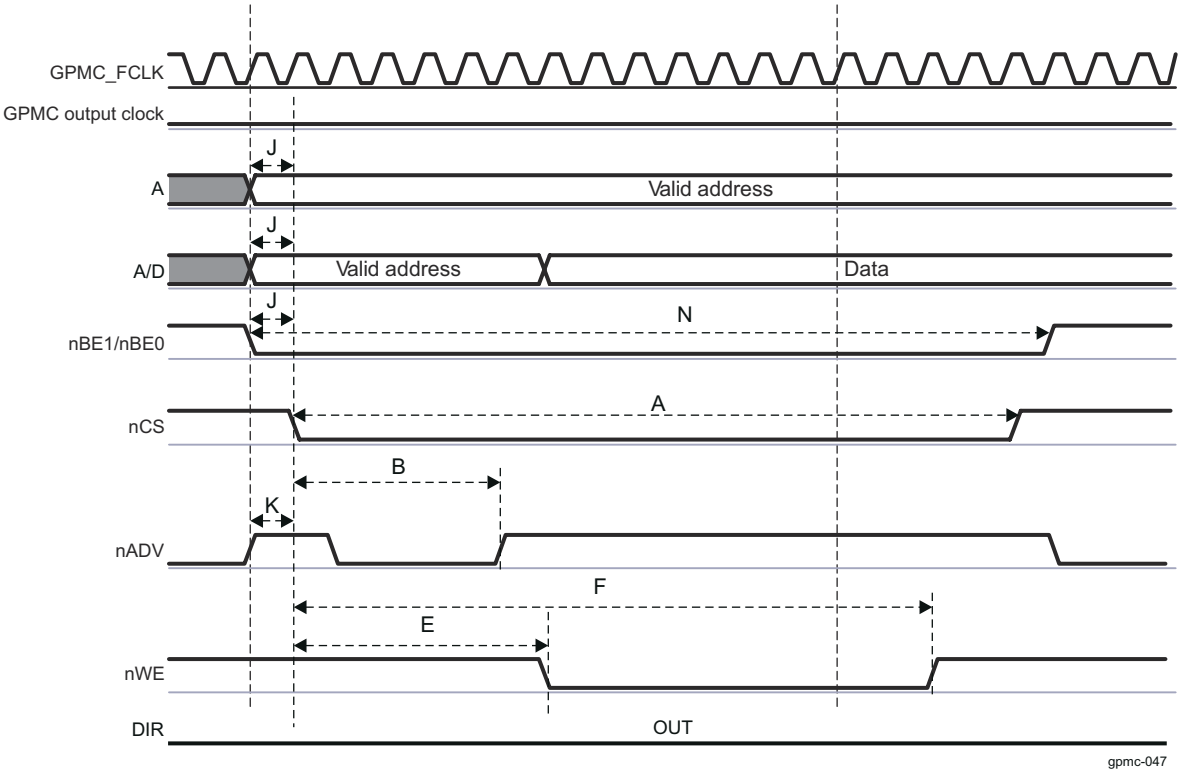


Figure 12-108. Asynchronous NOR Single Write Simplified Example

Note

Write multiple access is not supported in asynchronous mode. If WRITEMULTIPLE is enabled with WRITETYPE as asynchronous, the GPMC processes single asynchronous accesses.

Copyright © 2024 Texas Instruments Incorporated

12.3.5 Error Location Module (ELM)

This section describes the Error Location Module (ELM) for the device.

12.3.5.1 ELM Overview

The ELM extracts error addresses from generated syndrome polynomials.

The ELM is used with the GPMC. Syndrome polynomials generated on-the-fly when reading a NAND flash page and stored in GPMC registers are passed to the ELM. A host processor can then correct the data block by flipping the bits to which the ELM error-location outputs point.

When reading from NAND flash memories, some level of error-correction is required. In the case of NAND modules with no internal correction capability, sometimes referred to as *bare NANDs*, the correction process is delegated to the memory controller. ELM can be also used to support parallel NOR flash or NAND flash.

The General-Purpose Memory Controller (GPMC) probes data read from an external NAND flash and uses this to compute checksum-like information, called syndrome polynomials, on a per-block basis. Each syndrome polynomial gives a status of the read operations for a full block, including 512 bytes of data, parity bits, and an optional spare-area data field, with a maximum block size of 1023 bytes. Computation is based on a Bose-Chaudhuri-Hocquenghem (BCH) algorithm. The ELM extracts error addresses from these syndrome polynomials.

Based on the syndrome polynomial value, the ELM can detect errors, compute the number of errors, and give the location of each error bit. The actual data is not required to complete the error-correction algorithm. Errors can be reported anywhere in the NAND flash block, including in the parity bits.

The maximum acceptable number of errors that can be corrected depends on a programmable configuration parameter. 4-, 8-, and 16-bit error-correction levels are supported. The ELM depends on a static and fixed definition of the generator polynomial for each error-correction level that corresponds to the generator polynomials defined in the GPMC (there are three fixed polynomial for the three correction error levels). A larger number of errors than the programmed error-correction level may be detected, but the ELM cannot correct them all. The offending block is then tagged as *uncorrectable* in the associated computation exit status register. If the computation is successful, that is, if the number of errors detected does not exceed the maximum value authorized for the chosen correction capability, the exit status register contains the information on the number of detected errors.

When the error-location process completes, an interrupt is triggered to inform the software that its status can be checked. The number of detected errors and their locations in the NAND block can be retrieved from the module through register accesses.

12.3.5.1.1 ELM Features

The ELM has the following features:

- 4, 8, and 16 bits per 512-byte block error-location, based on BCH algorithms
- Eight simultaneous processing contexts
- Page-based and continuous modes
- Interrupt generation on error-location process completion:
  - When the full page has been processed in page mode
  - For each syndrome polynomial in continuous mode.

12.3.5.1.2 ELM Ports

Table 12-210. ELM0 Clocks and Resets

Clocks	
Module Clock Input	**Description**
ELM_FICLK	ELM functional and interface clock
Resets	
Module Reset Input	**Description**
ELM_RST	ELM hardware reset

Copyright © 2024 Texas Instruments Incorporated

Table 12-211. ELM0 Hardware Requests

Interrupt Requests		
Module Interrupt Signal	**Description**	**Type**
ELM_ELM_POROCPSINTERRUPT_LVL_0	Error-location process complete interrupt	Level

Copyright © 2024 Texas Instruments Incorporated

12.3.5.2 ELM Functional Description

The ELM0 module is hereinafter referred to as ELM.

The ELM is designed around the error-location engine, which handles the computation based on the input syndrome polynomials.

The ELM maps the error-location engine to a standard interconnect interface by using a set of registers to control inputs and outputs.

12.3.5.2.1 ELM Software Reset

To perform a software reset, set the ELM_SYSCONFIG[1] SOFTRESET bit to 1. The ELM_SYSSTATUS[0] RESETDONE bit indicates that the software reset is complete when its value is 1. When the software reset completes, the ELM_SYSCONFIG[1] SOFTRESET bit is automatically reset.

12.3.5.2.2 ELM Power Management

Note

Some of the ELM features described in this section may not be supported on this family of devices. For more information, see *ELM Not Supported Features*.

Table 12-212 describes the power-management features available to the ELM.

Table 12-212. Local Power-Management Features

Feature	Registers	Description
Clock autogating	ELM_SYSCONFIG[0] AUTOGATING	This bit allows a local power optimization inside the module by gating the ELM_FICLK clock upon the interface activity.
Idle modes	ELM_SYSCONFIG[4-3] SIDLEMODE	Force-idle, no-idle, and smart-idle modes are available.
Clock activity	ELM_SYSCONFIG[8] CLOCKACTIVITY	The clock can be switched off or maintained.

12.3.5.2.3 ELM Interrupt Requests

Table 12-213 lists the event flags, and their masks, that can cause module interrupts asserting the signal.

Table 12-213. ELM Events

Event Flag	Event Mask	Description
ELM_IRQSTATUS[8] PAGE_VALID	ELM_IRQENABLE[8] PAGE_MASK	Page interrupt
ELM_IRQSTATUS[7] LOC_VALID_7	ELM_IRQENABLE[7] LOCATION_MASK_7	Error-location interrupt for syndrome polynomial 7
ELM_IRQSTATUS[6] LOC_VALID_6	ELM_IRQENABLE[6] LOCATION_MASK_6	Error-location interrupt for syndrome polynomial 6
ELM_IRQSTATUS[5] LOC_VALID_5	ELM_IRQENABLE[5] LOCATION_MASK_5	Error-location interrupt for syndrome polynomial 5
ELM_IRQSTATUS[4] LOC_VALID_4	ELM_IRQENABLE[4] LOCATION_MASK_4	Error-location interrupt for syndrome polynomial 4
ELM_IRQSTATUS[3] LOC_VALID_3	ELM_IRQENABLE[3] LOCATION_MASK_3	Error-location interrupt for syndrome polynomial 3
ELM_IRQSTATUS[2] LOC_VALID_2	ELM_IRQENABLE[2] LOCATION_MASK_2	Error-location interrupt for syndrome polynomial 2
ELM_IRQSTATUS[1] LOC_VALID_1	ELM_IRQENABLE[1] LOCATION_MASK_1	Error-location interrupt for syndrome polynomial 1
ELM_IRQSTATUS[0] LOC_VALID_0	ELM_IRQENABLE[0] LOCATION_MASK_0	Error-location interrupt for syndrome polynomial 0

12.3.5.2.4 ELM Processing Initialization

ELM_LOCATION_CONFIG global setting parameters must be set before using the error-location engine. The ELM_LOCATION_CONFIG[1-0] ECC_BCH_LEVEL bit field defines the error-correction level used

Copyright © 2024 Texas Instruments Incorporated

(4-, 8-, or 16-bit error correction). The ELM_LOCATION_CONFIG[26-16] ECC_SIZE bit field defines the maximum buffer length beyond which the engine processing no longer looks for errors.

Software can choose to use the ELM in continuous mode or page mode. If all ELM_PAGE_CTRL[i] SECTOR_i bits (i is the syndrome polynomial number, where i = 0 to 7) are reset, continuous mode is used. In any other case, page mode is implicitly selected.

- Continuous mode: Each syndrome polynomial is processed independently. Results for a syndrome can be retrieved and acknowledged at any time, regardless of the status of the other seven processing contexts.
- Page mode: Syndrome polynomials are grouped into atomic entities: only one page can be processed at any given time, even if all eight contexts are not used for this page. Unused contexts are lost and cannot be affected to any other processing. The full page must be acknowledged and cleared before moving to the next page.

For completion interrupts to be generated correctly, all ELM_IRQENABLE[i] LOCATION_MASK_i bits (where i = 0 to 7) must be forced to 0 when in page mode, and set to 1 in continuous mode. Additionally, the ELM_IRQENABLE[8] PAGE_MASK bit must be set to 1 when in page mode.

Software initiates error-location processing by writing a syndrome polynomial into one of the eight possible register sets. Each of these register sets includes seven registers: ELM_SYNDROME_FRAGMENT_0_i to ELM_SYNDROME_FRAGMENT_6_i. The first six registers can be written in any order, but ELM_SYNDROME_FRAGMENT_6_i must be written last because it includes the validity bit, which instructs the ELM that this syndrome polynomial must be processed (the ELM_SYNDROME_FRAGMENT_6_i[16] SYNDROME_VALID bit).

As soon as one validity bit is asserted (ELM_SYNDROME_FRAGMENT_6_i[16] SYNDROME_VALID = 0x1, where i = 0 to 7), error-location processing can start for the corresponding syndrome polynomial. The associated ELM_LOCATION_STATUS_i and ELM_ERROR_LOCATION_0_i to ELM_ERROR_LOCATION_15_i registers (where i = 0 to 7) are not reset. Software must not consider them until the corresponding ELM_IRQSTATUS[i] LOC_VALID_i bit is set.

12.3.5.2.5 ELM Processing Sequence

While the error-location engine is busy processing one syndrome polynomial, further syndrome polynomials can be written. They are processed when the current processing completes.

The engine completes early when:

- No error is detected; that is, when the ELM_LOCATION_STATUS_i[8] ECC_CORRECTABLE bit is set to 1 and the ELM_LOCATION_STATUS_i[4-0] ECC_NB_ERRORS bit field is set to 0x0.
- Too many errors are detected; that is, when the ELM_LOCATION_STATUS_i[8] ECC_CORRECTABLE bit is set to 0 while the ELM_LOCATION_STATUS_i[4-0] ECC_NB_ERRORS bit field is set with the value output by the error-location engine. The reported number of errors is not ensured if ECC_CORRECTABLE is 0.

If the engine completes early, the associated error-location registers ELM_ERROR_LOCATION_0_i to ELM_ERROR_LOCATION_15_i (where i = 0 to 7) are not updated.

In all other cases, the engine goes through the entire error-location process. Each time an error location is found, it is logged in the associated ECC_ERROR_LOCATION bit field. The first error detected is logged in the ELM_ERROR_LOCATION_0_i[12-0] ECC_ERROR_LOCATION bit field; the second is logged in the ELM_ERROR_LOCATION_1_i[12-0] ECC_ERROR_LOCATION bit field, and so on.

Table 12-214 describes the ELM_LOCATION_STATUS_i value decoding.

Table 12-214. ELM_LOCATION_STATUS_i Value Decoding

ECC_CORRECTABLE Value	ECC_NB_ERRORS Value	Status	Number of Errors Detected	Action Required
1	0	OK	0	None
1	≠ 0	OK	ECC_NB_ERRORS	Correct the data buffer read based on the ELM_ERROR_LOCATION_0_i to ELM_ERROR_LOCATION_15_i results.
0	Any	Failed	Unknown	Software-dependent

Copyright © 2024 Texas Instruments Incorporated

12.3.5.2.6 ELM Processing Completion

When the processing for a given syndrome polynomial completes, its ELM_SYNDROME_FRAGMENT_6_i[16] SYNDROME_VALID bit is reset. It must not be set again until the exit status registers, ELM_LOCATION_STATUS_i (where i = 0 to 7) for this processing are checked. Failure to comply with this rule leads to potential loss of the first polynomial process data output.

The error-location engine signals the process completion to the ELM. When this event is detected, the corresponding ELM_IRQSTATUS[i] LOC_VALID_i bit (where i = 0 to 7) is set. The processing exit status is available from the associated ELM_LOCATION_STATUS_i register, and error locations are stored in order in the ECC_ERROR_LOCATION bit fields. Software must read only valid error-location registers based on the number of errors detected and located.

Immediately after the error-location engine completes, a new syndrome polynomial can be processed, if any is available, as reported by the ELM_SYNDROME_FRAGMENT_6_i[16] SYNDROME_VALID bit, depending on the configured error-correction level. If several syndrome polynomials are available, a round-robin arbitration is used to select one for processing.

In continuous mode (that is, all bits in ELM_PAGE_CTRL are reset), an interrupt is triggered whenever a ELM_IRQSTATUS[i] LOC_VALID_i bit is asserted. Software must read the ELM_IRQSTATUS register to determine which polynomial is processed and retrieve the exit status and error locations (ELM_LOCATION_STATUS_i and ELM_ERROR_LOCATION_0_i to ELM_ERROR_LOCATION_15_i). When done, software must clear the corresponding ELM_IRQSTATUS[i] LOC_VALID_i bit by setting it to 1. Other status bits must be set to 0 so that other interrupts are not unintentionally cleared. When using this mode, the ELM_IRQSTATUS[8] PAGE_VALID interrupt is never triggered.

In page mode, the module does not trigger interrupts for the processing completion of each polynomial because the ELM_IRQENABLE[i] LOCATION_MASK_i bits are cleared. A page is defined using the ELM_PAGE_CTRL register. Each SECTOR_i bit set means the corresponding polynomial i is part of the page processing. A page is fully processed when all tagged polynomials have been processed, as logged in the ELM_IRQSTATUS[i] LOC_VALID_i bits. The module triggers an ELM_IRQSTATUS[8] PAGE_VALID interrupt whenever it detects that the full page has been processed. To make sure the next page can be correctly processed, all status bits in the ELM_IRQSTATUS register must be cleared by using a single atomic-write access.

Note

Do not modify page setting parameters in the ELM_PAGE_CTRL register unless the engine is idle, no polynomial input is valid, and all interrupts have been cleared.

Because no polynomial-level interrupt is triggered in page mode, polynomials cleared in the ELM_PAGE_CTRL[i] SECTOR_i bits (where i = 0 to 7) are processed as usual, but are essentially ignored. Software must manually poll the ELM_IRQSTATUS bits to check for their status.

Copyright © 2024 Texas Instruments Incorporated

12.3.5.3 ELM Basic Programming Model

12.3.5.3.1 ELM Low-Level Programming Model

12.3.5.3.1.1 Processing Initialization

Table 12-215. ELM Processing Initialization

Step	Register/Bit Field/Programming Model	Value
Resets the module.	ELM_SYSCONFIG[1] SOFTRESET	0x1
Wait until reset is done.	ELM_SYSSTATUS[0] RESETDONE	0x1
Configure the target interface power management.	ELM_SYSCONFIG[4-3] SIDLEMODE	Set value
Defines the error-correction level used.	ELM_LOCATION_CONFIG[1-0] ECC_BCH_LEVEL	Set value
Defines the maximum buffer length.	ELM_LOCATION_CONFIG[26-16] ECC_SIZE	Set value
Sets the ELM in continuous mode or page mode.	ELM_PAGE_CTRL	Set value
IF continuous mode is used:	All ELM_PAGE_CTRL[i] SECTOR_i (where i = 0 to 7)	0x0
Enable interrupt for syndrome polynomial i.	ELM_IRQENABLE[i] LOCATION_MASK_i	0x1
ELSE (page mode is used):	One syndrome polynomial i is set ELM_PAGE_CTRL[i] SECTOR_i (where i = 0 to 7)	0x1
Disable all interrupts for syndrome polynomial and enable PAGE_MASK interrupt.	All ELM_IRQENABLE[i] LOCATION_MASK_i = 0x0 and ELM_IRQENABLE[8] PAGE_MASK = 0x1	Set value
ENDIF		Set value
Set the input syndrome polynomial i.	ELM_SYNDROME_FRAGMENT_0_i	Set value
	ELM_SYNDROME_FRAGMENT_1_i	Set value
	ELM_SYNDROME_FRAGMENT_5_i	Set value
	ELM_SYNDROME_FRAGMENT_6_i	Set value
Initiates the computation process.	ELM_SYNDROME_FRAGMENT_6_i[16] SYNDROME_VALID	0x1

12.3.5.3.1.2 Read Results

The engine goes through the entire error-location process and results can be read. Table 12-216 and Table 12-217 describe the processing completion for continuous and page modes, respectively.

Table 12-216. ELM Processing Completion for Continuous Mode

Step	Register/Bit Field/Programming Model	Value
Wait until process is complete for syndrome polynomial i: Wait until the interrupt is generated, or poll the status register.		
Read for which i the error-location process is complete.	ELM_IRQSTATUS[i] LOC_VALID_i	0x1
IF the process fails (too many errors):	ELM_LOCATION_STATUS_i[8] ECC_CORRECTABLE	0x0
It is software dependant.		
ELSE (process successful, the engine completes):	ELM_LOCATION_STATUS_i[8] ECC_CORRECTABLE	0x1
Read the number of errors.	ELM_LOCATION_STATUS_i[4-0] ECC_NB_ERRORS	
Read the error-location bit addresses for syndrome polynomial i of the ECC_NB_ERRORS first registers. Software must correct errors in the data buffer.	ELM_ERROR_LOCATION_0_i[12-0] ECC_ERROR_LOCATION	
	ELM_ERROR_LOCATION_1_i[12-0] ECC_ERROR_LOCATION	
	...	
	ELM_ERROR_LOCATION_15_i[12-0] ECC_ERROR_LOCATION	
ENDIF		
Clear the corresponding i interrupt.	ELM_IRQSTATUS[i] LOC_VALID_i	0x1

A new syndrome polynomial can be processed after the end of processing (ELM_SYNDROME_FRAGMENT_6_i[16] SYNDROME_VALID = 0x0) and after the exit status register check (ELM_LOCATION_STATUS_i).

Copyright © 2024 Texas Instruments Incorporated

Table 12-217. ELM Processing Completion for Page Mode

Step	Register/Bit Field/Programming Model	Value
Wait until process is complete for syndrome polynomial i: Wait until the interrupt is generated, or poll the status register.		
Wait for page completed interrupt: All error locations are valid.	ELM_IRQSTATUS[8] PAGE_VALID	0x1
Repeat the following actions the necessary number of times. That is, once for each valid defined block in the page.		
Read the process exit status.	ELM_LOCATION_STATUS_i[8] ECC_CORRECTABLE	
IF the process fails (too many errors):	ELM_LOCATION_STATUS_i[8] ECC_CORRECTABLE	0x0
It is software dependent.		
ELSE (process successful, the engine completes):	ELM_LOCATION_STATUS_i[8] ECC_CORRECTABLE	0x1
Read the number of errors.	ELM_LOCATION_STATUS_i[4-0] ECC_NB_ERRORS	
Read the error-location bit addresses for syndrome polynomial i of the ECC_NB_ERRORS first registers.	ELM_ERROR_LOCATION_0_i[12-0] ECC_ERROR_LOCATION	
	ELM_ERROR_LOCATION_1_i[12-0] ECC_ERROR_LOCATION	
	...	
	ELM_ERROR_LOCATION_15_i[12-0] ECC_ERROR_LOCATION	
ENDIF		
End Repeat		
Clear the ELM_IRQSTATUS register.	ELM_IRQSTATUS	0x1FF

Next page can be correctly processed after a page is fully processed, when all tagged polynomials have been processed (ELM_IRQSTATUS[i] LOC_VALID_i = 0x1 for all syndrome polynomials i used in the page).

12.3.5.3.1.3

Next page can be correctly processed after a page is fully processed, when all tagged polynomials have been processed (ELM_IRQSTATUS[i] LOC_VALID_i = 0x1 for all syndrome polynomials i used in the page).

12.3.5.3.2 Use Case: ELM Used in Continuous Mode

In this example, the ELM is programmed for an 8-bit error-correction capability in continuous mode (see Table 12-218). After reading a 528-byte NAND flash sector (512B data plus 16B spare area) with a 16-bit interface, a nonzero polynomial syndrome is reported from the GPMC (polynomial syndrome 0 is used in the ELM):

- P = 0x0A16ABE115E44F767BFB0D0980.

Table 12-218. Use Case: Continuous Mode

Step	Register/Bit Field/Programming Model	Value
Reset the module.	ELM_SYSCONFIG[1] SOFTRESET	0x1
Wait until reset is done.	ELM_SYSSTATUS[0] RESETDONE	0x1
Configure the target interface power management: Smart idle is used.	ELM_SYSCONFIG[4-3] SIDLEMODE	0x2
Define the error-correction level used: 8 bits.	ELM_LOCATION_CONFIG[1-0] ECC_BCH_LEVEL	0x1
Define the maximum buffer length: 528 bytes (2 × 528 = 1056).	ELM_LOCATION_CONFIG[26-16] ECC_SIZE	0x420
Set the ELM in continuous mode.	ELM_PAGE_CTRL	0
Enable interrupt for syndrome polynomial 0.	ELM_IRQENABLE[0] LOCATION_MASK_0	0x1
Set the input syndrome polynomial 0.	ELM_SYNDROME_FRAGMENT_0_i (where i = 0)	0xFB0D0980
	ELM_SYNDROME_FRAGMENT_1_i (where i = 0)	0xE44F767B
	ELM_SYNDROME_FRAGMENT_2_i (where i = 0)	0x16ABE115
	ELM_SYNDROME_FRAGMENT_3_i (where i = 0)	0x0000000A
Initiate the computation process.	ELM_SYNDROME_FRAGMENT_6_i[16] SYNDROME_VALID (where i = 0)	0x1

Copyright © 2024 Texas Instruments Incorporated

Table 12-218. Use Case: Continuous Mode (continued)

Step	Register/Bit Field/Programming Model	Value
Wait until process is complete for syndrome polynomial 0: is generated or poll the status register.		
Read that error-location process is complete for syndrome polynomial 0.	ELM_IRQSTATUS[0] LOC_VALID_0	0x1
Read the process exit status: All errors were successfully located.	ELM_LOCATION_STATUS_i[8] ECC_CORRECTABLE (where i = 0)	0x1
Read the number of errors: Four errors detected.	ELM_LOCATION_STATUS_i[4-0] ECC_NB_ERRORS (where i = 0)	0x4
Read the error-location bit addresses for syndrome polynomial 0 of the first four registers: Errors are located in the data buffer at decimal addresses 431, 1062, 1909, 3452.	ELM_ERROR_LOCATION_0_i (where i = 0)	0x1AF
	ELM_ERROR_LOCATION_1_i (where i = 0)	0x426
	ELM_ERROR_LOCATION_2_i (where i = 0)	0x775
	ELM_ERROR_LOCATION_3_i (where i = 0)	0xD7C
Clear the corresponding interrupt for polynomial 0.	ELM_IRQSTATUS[0] LOC_VALID_0	0x1

The NAND flash data in the sector are seen as a polynomial of degree 4223 (number of bits in a 528 byte buffer minus 1), with each data bit being a coefficient in the polynomial. When reading from a NAND flash using the GPMC module, computation of the polynomial syndrome assumes that the first NAND word read at address 0x0 contains the highest-order coefficient in the message. Furthermore, in the 16-bit NAND word, bits are ordered from bit 7 to bit 0, and then from bit 15 to bit 8. Based on this convention, an address table of the data buffer can be built. NAND memory addresses in Table 12-219 are given in decimal format.

Table 12-219. 16-Bit NAND Sector Buffer Address Map

NAND Memory Address	Message Bit Addresses in Memory Word															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	4215	4214	4213	4212	4211	4210	4209	4208	4223	4222	4221	4220	4219	4218	4217	4216
1	4175	4174	4173	4172	4171	4170	4169	4168	4183	4182	4181	4180	4179	4178	4177	4176
...																
47	3463	3462	3461	3460	3459	3458	3457	3456	3471	3470	3469	3468	3467	3466	3465	3464
48	3447	3446	3445	3444	3443	3442	3441	3440	3455	3454	3453	3452	3451	3450	3449	3448
49	3431	3430	3429	3428	3427	3426	3425	3424	3439	3438	3437	3436	3435	3434	3433	3432
50	3415	3414	3413	3412	3411	3410	3409	3408	3423	3422	3421	3420	3419	3418	3417	3416
...																
255	135	134	133	132	131	130	129	128	143	142	141	140	139	138	137	136
256	119	118	117	116	115	114	113	112	127	126	125	124	123	122	121	120
257	103	102	101	100	99	98	97	96	111	110	109	108	107	106	105	104
258	87	86	85	84	83	82	81	80	95	94	93	92	91	90	89	88
259	71	70	69	68	67	66	65	64	79	78	77	76	75	74	73	72
260	55	54	53	52	51	50	49	48	63	62	61	60	59	58	57	56
261	39	38	37	36	35	34	33	32	47	46	45	44	43	42	41	40
262	23	22	21	20	19	18	17	16	31	30	29	28	27	26	25	24
263	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8

The table can now be used to determine which bits in the buffer were incorrect and must be flipped. In this example, the first bit to be flipped is bit 4 from the 49th byte read from memory. It is up to the processor to correctly map this word to the copied buffer and flip this bit. The same process must be repeated for all detected errors.

Copyright © 2024 Texas Instruments Incorporated

12.3.5.3.3 Use Case: ELM Used in Page Mode

In this example, the ELM module is programmed for a 16-bit error-correction capability in page mode (see Table 12-220). After reading a 528-byte NAND flash sector (512B data plus 16B spare area) with a 16-bit interface, four non-zero polynomial syndromes are reported from the GPMC (polynomial syndrome 0, 1, 2, and 3 are used in the ELM):

- P0 = 0xE8B0 12ADDB5A318E05BE B0693DB28330B5CC A329AA05E0B718EF
- P1 = 0xBAD0 49A0D932C22E6669 0948DF08BE093336 79C6BA10E5F935EB
- P2 = 0x69D9 B86ABCD5EC3697FA A6498FEE54556EA0 1579EF7D60BA3189
- P3 = 0x0

Table 12-220. Use Case: Page Mode

Step	Register/Bit Field/Programming Model	Value
Reset the module	ELM_SYSCONFIG[1] SOFTRESET	0x1
Wait until reset is done.	ELM_SYSSTATUS[0] RESETDONE	0x1
Configure the target interface power management: Smart idle is used.	ELM_SYSCONFIG[4-3] SIDLEMODE	0x2
Define the error-correction level used: 16 bits	ELM_LOCATION_CONFIG[1-0] ECC_BCH_LEVEL	0x2
Define the maximum buffer length: 528 bytes	ELM_LOCATION_CONFIG[26-16] ECC_SIZE	0x420
Set the ELM in page mode (four blocks in a page)	ELM_PAGE_CTRL[0] SECTOR_0	0x1
	ELM_PAGE_CTRL[1] SECTOR_1	0x1
	ELM_PAGE_CTRL[2] SECTOR_2	0x1
	ELM_PAGE_CTRL[3] SECTOR_3	0x1
Disable all interrupts for syndrome polynomial and enable PAGE_MASK interrupt.	ELM_IRQENABLE	0x100
Set the input syndrome polynomial 0.	ELM_SYNDROME_FRAGMENT_0_i (where i = 0)	0xE0B718EF
	ELM_SYNDROME_FRAGMENT_1_i (where i = 0)	0xA329AA05
	ELM_SYNDROME_FRAGMENT_2_i (where i = 0)	0x8330B5CC
	ELM_SYNDROME_FRAGMENT_3_i (where i = 0)	0xB0693DB2
	ELM_SYNDROME_FRAGMENT_4_i (where i = 0)	0x318E05BE
	ELM_SYNDROME_FRAGMENT_5_i (where i = 0)	0x12ADDB5A
	ELM_SYNDROME_FRAGMENT_6_i (where i = 0)	0xE8B0
Set the input syndrome polynomial 1.	ELM_SYNDROME_FRAGMENT_0_i (where i = 1)	0xE5F935EB
	ELM_SYNDROME_FRAGMENT_1_i (where i = 1)	0x79C6BA10
	ELM_SYNDROME_FRAGMENT_2_i ((where i = 1)	0xBE093336
	ELM_SYNDROME_FRAGMENT_3_i (where i = 1)	0x0948DF08
	ELM_SYNDROME_FRAGMENT_4_i (where i = 1)	0xC22E6669
	ELM_SYNDROME_FRAGMENT_5_i (where i = 1)	0x49A0D932
	ELM_SYNDROME_FRAGMENT_6_i (where i = 1)	0xBAD0
Set the input syndrome polynomial 2.	ELM_SYNDROME_FRAGMENT_0_i (where i = 2)	0x60BA3189
	ELM_SYNDROME_FRAGMENT_1_i (where i = 2)	0x1579EF7D
	ELM_SYNDROME_FRAGMENT_2_i (where i = 2)	0x54556EA0
	ELM_SYNDROME_FRAGMENT_3_i (where i = 2)	0xA6498FEE
	ELM_SYNDROME_FRAGMENT_4_i (where i = 2)	0xEC3697FA
	ELM_SYNDROME_FRAGMENT_5_i (where i = 2)	0xB86ABCD5
	ELM_SYNDROME_FRAGMENT_6_i (where i = 2)	0x69D9

Copyright © 2024 Texas Instruments Incorporated

Table 12-220. Use Case: Page Mode (continued)

Step	Register/Bit Field/Programming Model	Value
Set the input syndrome polynomial 3.	ELM_SYNDROME_FRAGMENT_0_i (where i = 3)	0x0
	ELM_SYNDROME_FRAGMENT_1_i (where i = 3)	0x0
	ELM_SYNDROME_FRAGMENT_2_i (where i = 3)	0x0
	ELM_SYNDROME_FRAGMENT_3_i (where i = 3)	0x0
	ELM_SYNDROME_FRAGMENT_4_i (where i = 3)	0x0
	ELM_SYNDROME_FRAGMENT_5_i (where i = 3)	0x0
	ELM_SYNDROME_FRAGMENT_6_i (where i = 3)	0x0
Initiate the computation process for syndrome polynomial 0	ELM_SYNDROME_FRAGMENT_6_i[16] SYNDROME_VALID (where i = 0)	0x1
Initiate the computation process for syndrome polynomial 1	ELM_SYNDROME_FRAGMENT_6_i[16] SYNDROME_VALID (where i = 1)	0x1
Initiate the computation process for syndrome polynomial 2	ELM_SYNDROME_FRAGMENT_6_i[16] SYNDROME_VALID (where i = 2)	0x1
Initiate the computation process for syndrome polynomial 3	ELM_SYNDROME_FRAGMENT_6_i[16] SYNDROME_VALID (where i = 3)	0x1
Wait until process is complete for syndrome polynomial 0, 1, 2, and 3: Wait until the interrupt is generated or poll the status register.		
Wait for page completed interrupt: All error locations are valid.	ELM_IRQSTATUS[8] PAGE_VALID	0x1
Read the process exit status for syndrome polynomial 0: All errors were successfully located.	ELM_LOCATION_STATUS_i[8] ECC_CORRECTABLE (where i = 0)	0x1
Read the process exit status for syndrome polynomial 1: All errors were successfully located.	ELM_LOCATION_STATUS_i[8] ECC_CORRECTABLE (where i = 1)	0x1
Read the process exit status for syndrome polynomial 2: All errors were successfully located.	ELM_LOCATION_STATUS_i[8] ECC_CORRECTABLE (where i = 2)	0x1
Read the process exit status for syndrome polynomial 3: All errors were successfully located.	ELM_LOCATION_STATUS_i[8] ECC_CORRECTABLE (where i = 3)	0x1
Read the number of errors for syndrome polynomial 0: four errors detected.	ELM_LOCATION_STATUS_i[4-0] ECC_NB_ERRORS (where i = 0)	0x4
Read the number of errors for syndrome polynomial 1: two errors detected.	ELM_LOCATION_STATUS_i[4-0] ECC_NB_ERRORS (where i = 1)	0x2
Read the number of errors for syndrome polynomial 2: one error detected.	ELM_LOCATION_STATUS_i[4-0] ECC_NB_ERRORS (where i = 2)	0x1
Read the number of errors for syndrome polynomial 3: no errors detected.	ELM_LOCATION_STATUS_i[4-0] ECC_NB_ERRORS (where i = 3)	0x0
Read the error-location bit addresses for syndrome polynomial 0 of the first four registers:	ELM_ERROR_LOCATION_0_i (where i = 0)	0x1FE
	ELM_ERROR_LOCATION_1_i (where i = 0)	0x617
	ELM_ERROR_LOCATION_2_i (where i = 0)	0x650
	ELM_ERROR_LOCATION_3_i (where i = 0)	0xA83
Read the error-location bit addresses for syndrome polynomial 1 of the first two registers:	ELM_ERROR_LOCATION_0_i (where i = 1)	0x4
	ELM_ERROR_LOCATION_1_i (where i = 1)	0x1036
Read the errors location bit addresses for syndrome polynomial 2 of the first registers:	ELM_ERROR_LOCATION_0_i (where i = 1)	0x3E8
Clear the ELM_IRQSTATUS register.	ELM_IRQSTATUS	0x1FF

Copyright © 2024 Texas Instruments Incorporated

12.3.6 Multi-Media Card Secure Digital (MMCSD) Interface

This section describes the MMCSD modules in the device.

12.3.6.1 MMCSD Overview

There are two MMCSD modules inside the device - MMCSD0 and MMCSD1. Each MMCSD module includes one MMCSD Host Controller.

Each controller has the following data bus width:

- MMCSD0 - 8-bit wide data bus
- MMCSD1 - 4-bit wide data bus

The MMCSD Host Controller provides an interface to eMMC 5.1 (embedded Multi-Media Card), SD 4.10 (Secure Digital), and SDIO 4.0 (Secure Digital IO) devices. The MMCSD Host Controller deals with MMC/SD/SDIO protocol at transmission level, data packing, adding cyclic redundancy checks (CRCs), start/end bit insertion, and checking for syntactical correctness.

The MMCSD Host Controller provides accessibility to external MMC/SD/SDIO devices using a Programmed IO method or DMA data transfer method. In programmed IO method, the device CPU transfers data using the Buffer Data Port register (MMCSD0_DATA_PORT / MMCSD1_DATA_PORT). In DMA data transfer method, the MMCSD Host Controller can read or write memory without device CPU intervention.

Copyright © 2024 Texas Instruments Incorporated

12.3.6.1.1 MMCSD Features

Each MMCSD Host Controller supports:

- Integrated DMA controller supporting SD Advanced DMA - ADMA2 and ADMA3 (for more information about ADMA support, see Section 12.3.6.3.5, *Advanced DMA*)
- System Bus Interface:
  - 64-bit data width (master interface)
  - 64-bit address
  - Clock asynchronous to MMCSD clock (MMCi_CLK)
  - Little endian only
- Configuration Bus Interface:
  - 32-bit data width (slave interface)
  - Linear incrementing addressing mode
  - 32-bit aligned accesses only
  - Little endian only
- Muxing of other LVCMOS interfaces onto the MMCSD interface at the SoC level (MMCSD1 only)

MMCSD0 Host Controller (eMMC interface):

- MultiMedia Card Support:
  - eMMC Electrical Standard 5.1 (JESD84-B51)
  - Backward compatible with earlier eMMC standards
  - Legacy MMC SDR:
    - 1.8 V, 8-bit bus width, 0-25 MHz, 25 MBps
    - 1.8 V, 4-bit bus width, 0-25 MHz, 12.5 MBps
    - 1.8 V, 1-bit bus width, 0-25 MHz, 3.125 MBps
  - High Speed SDR:
    - 1.8 V, 8-bit bus width, 0-50 MHz, 50 MBps
    - 1.8 V, 4-bit bus width, 0-50 MHz, 25 MBps
    - 1.8 V, 1-bit bus width, 0-50 MHz, 6.25 MBps
  - High Speed DDR:
    - 1.8 V, 8-bit bus width, 0-50 MHz, 100 MBps
    - 1.8 V, 4-bit bus width, 0-50 MHz, 50 MBps
  - HS200 SDR:
    - 1.8 V, 0-200 MHz, 8-bit bus width, 200 MBps
    - 1.8 V, 0-200 MHz, 4-bit bus width, 100 MBps
  - HS400 DDR:
    - 1.8 V, 0-200 MHz, 8-bit bus width, 400 MBps

MMCSD1 Host Controller (SD/SDIO interface):

-
- Secure Digital Card Support:
  - Backward compatible with earlier SD card specifications
  - SD Host Controller Standard Specification 4.10 and SD Physical Layer Specification v3.01
  - SDIO Specification v3.00
  - Default Speed mode: 3.3 V signaling, frequency up to 25 MHz, up to 12.5 MBps
  - High Speed mode: 3.3 V signaling, frequency up to 50 MHz, up to 25 MBps
  - SDR12: UHS-I 1.8 V signaling, frequency up to 25 MHz, up to 12.5 MBps
  - SDR25: UHS-I 1.8 V signaling, frequency up to 50 MHz, up to 25 MBps
  - SDR50: UHS-I 1.8 V signaling, frequency up to 100 MHz, up to 50 MBps
  - DDR50: UHS-I 1.8 V signaling, frequency up to 50 MHz, up to 50 MBps
  - SDR104: UHS-I 1.8 V signaling, frequency up to 200 MHz, up to 100 MBps

Unsupported Features:

- See the Module Integration section of this document for a list of module features not supported by the integration on this Device.

Copyright © 2024 Texas Instruments Incorporated

12.3.6.1.2 MMCSD Ports

Table 12-221. MMCSD Clocks and Resets

Clocks	
Module Clock Input	**Description**
MMCSD_ICLK	MMCSD Interface Clock
MMCSD_FCLK	MMCSD Functional Clock
Resets	
Module Reset Input	**Description**
MMCSD_RST	MMCSD Asynchronous Reset

Table 12-222. MMCSD Hardware Requests

Interrupt Requests		
Module Interrupt Signal	**Description**	**Type**
MMCSD_EMMCSS_INTR_0	MMCSD Interrupt Request	Level
MMCSD_EMMCSS_RXMEM_CORR_ERR_LVL_0	MMCSD Receive ECC Correctable Error Interrupt Request	Level
MMCSD_EMMCSS_RXMEM_UNCORR_ERR_LVL_0	MMCSD Receive ECC Uncorrectable Error Interrupt Request	Level
MMCSD_EMMCSS_TXMEM_CORR_ERR_LVL_0	MMCSD Transmit ECC Correctable Error Interrupt Request	Level
MMCSD_EMMCSS_TXMEM_UNCORR_ERR_LVL_0	MMCSD Transmit ECC Uncorrectable Error Interrupt Request	Level

12.3.6.2 MMCSD Environment

This section describes the MMCSDi external connections (environment).

Copyright © 2024 Texas Instruments Incorporated

Table 12-223 describes the MMCSDi I/O signals.

Table 12-223. MMCSDi I/O Signals

Module Pin	I/O[(1)]	Description
DAT[7:0]	I/O	Data Signals
DS	I	Data Strobe
CALPAD	A	PHY Calibration Resistor
CLK	O	External Clock
CMD	I/O	Command Line
DAT[3:0]	I/O	Data Signals
WP	I	SD Card Write Protect
CD	I	SD Card Detect

(1) I = Input; O = Output; HiZ = High Impedance; A = Analog

Copyright © 2024 Texas Instruments Incorporated

12.3.6.3 MMCSD Functional Description

12.3.6.3.1 Block Diagram

Figure 12-109 shows the MMCSDi module block diagram (where i = 0 to 1).

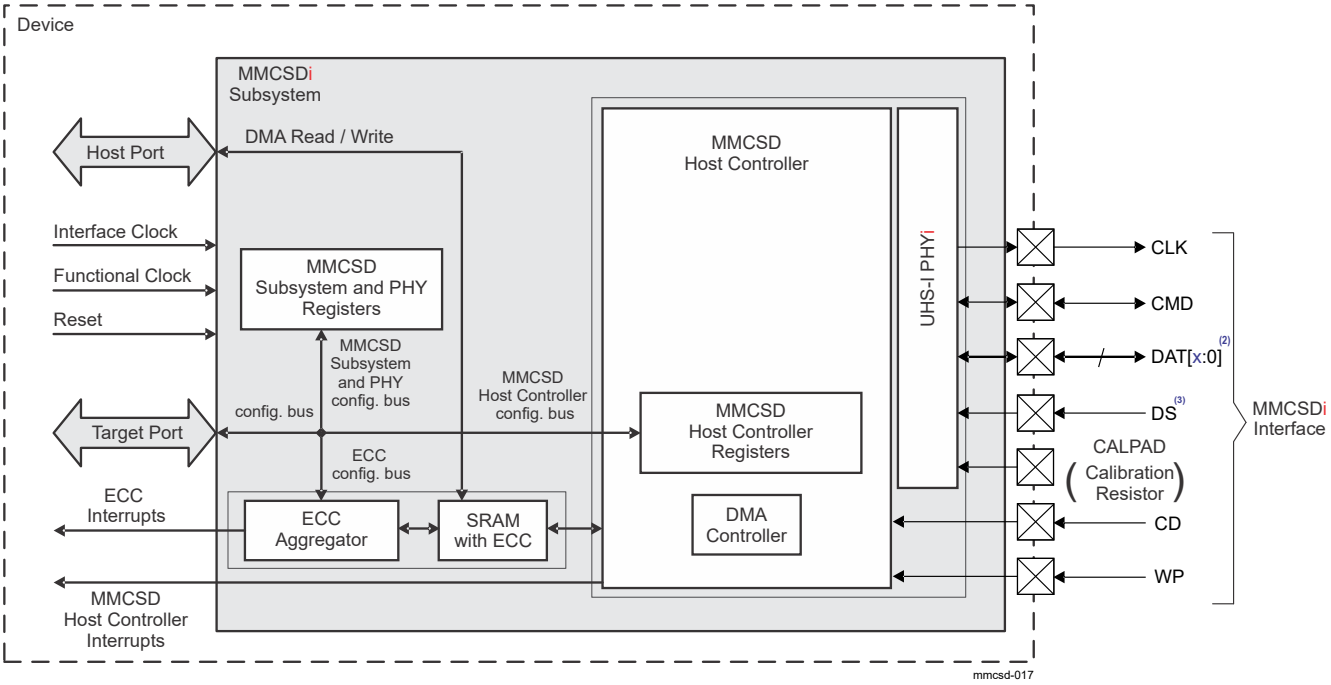


Figure 12-109. MMCSDi Module Block Diagram

Basic Blocks:

- MMCSD Host Controller: The MMCSD Host Controller is situated in the MMCSD Subsystem and provides accessibility to external MMC/SD/SDIO devices using a Programmed IO method or DMA data transfer method.
- UHS-I PHY: The integrated UHS-I PHY provides an interface between the MMCSD Host Controller and external MMC/SD/SDIO devices.
- MMCSDi Interface: The MMCSDi Interface includes all used interface pins (for more information, see *MMCSDi I/O Signals*).
- Host Port: Provides a 64-bit wide read/write interface between the device and the MMCSD Subsystem internal SRAM.
- Target Port: Provides a 32-bit wide interface between the device and the MMCSD Subsystem and MMCSD Host Controller parts.
- MMCSD Host Controller Registers: This block includes set of all MMCSD Host Controller registers.
- MMCSD Subsystem and PHY Registers: This block implements memory-mapped registers at the MMCSD Subsystem level and memory-mapped registers to control and program the MMCSD PHY.
- ECC Aggregator: The ECC Aggregator block facilitates aggregating and reporting internal SRAM ECC errors (for more information, see Section 12.3.6.3.4, *ECC Support*).
- Internal SRAM with ECC: The internal SRAM block is used for data storage during read/write transactions.
- DMA Controller: Manages the data transfer between the device memory and the MMCSD Subsystem internal SRAM.
- System Clocks: There are asynchronous relationship between the interface (system) clock and functional clock (for more information, see *MMCSD Clocks*).
- System Reset: The reset to the MMCSD Subsystem provides reset to the all MMCSD Subsystem parts (for more information, see *MMCSD Clocks*).
- Interrupts: The MMCSD Subsystem sources one MMCSD Host Controller interrupt and four ECC Aggregator interrupts (for more information, see *MMCSD Hardware Requests* and *Interrupt Requests*).

For more information about all MMCSD registers, see *MMCSD Registers*.

Copyright © 2024 Texas Instruments Incorporated

The MMCSD Host Controller can use a Programmed IO method (PIO) or DMA data transfer method to access external MMC/SD/SDIO devices.

The target port is used for device CPU access to the MMCSD Host Controller register set. The MMCSD Host Controller register set provides the communication between the device CPU and the MMCSD Host Controller. In PIO method the device CPU transfers data using the MMCSD0_DATA_PORT / MMCSD1_DATA_PORT register. The data flow from the device memory to the external MMC/SD/SDIO device (and vice versa) is through the target port, the MMCSD0_DATA_PORT / MMCSD1_DATA_PORT register in the MMCSD Host Controller, and the PHY.

The host port is used for connection to the DMA controller (for more information about ADMA support, see Section 12.3.6.3.5, *Advanced DMA* and MMCSD0_CAPABILITIES / MMCSD1_CAPABILITIES register). In DMA data transfer method the DMA controller uses the host port to transfer data between the device memory and the MMCSD Subsystem internal SRAM. The other side of the internal SRAM is connected to the MMCSD Host Controller. The data flow from the device memory to the external MMC/SD/SDIO device (and vice versa) is through the host port, internal SRAM, MMCSD Host Controller, and the PHY. The DMA controller manages the read/write operations from/to the internal SRAM without device CPU intervention.

12.3.6.3.2 Memory Regions

shows MMCSDi module memory map.

12.3.6.3.3 Interrupt Requests

Each MMCSD instance sources one active high level MMCSD Host Controller interrupt and four active high level ECC Aggregator interrupts (see *MMCSD Hardware Requests*). The MMCSD Host Controller interrupt is generated based on the bit values in the MMCSD0_NORMAL_INTR_STS / MMCSD1_NORMAL_INTR_STS and MMCSD0_NORMAL_INTR_STS_ENA / MMCSD1_NORMAL_INTR_STS_ENA registers. The ECC Aggregator interrupts are generated based on the ECC errors - single and double bit errors (for more information about ECC Aggregator interrupts, see Section 12.3.6.3.4, *ECC Support*).

12.3.6.3.4 ECC Support

The Error Correcting Code (ECC) is a mechanism for providing increased system reliability via reduction of memory software errors by allowing single bit errors to be detected and corrected (SEC) and double bit errors to be detected (DED).

There are two SRAM memories (transmit SRAM and receive SRAM) within each MMCSD instance. These two memories are ECC protected.

The SEC logic detects and corrects a single bit error (single bit error per ECC word or per ECC data segment). The DED logic only detects (does not correct) double errors (double bit errors per ECC word or per ECC data segment).

12.3.6.3.4.1 ECC Aggregator

There are two SRAMs (each with size 4 KB - 512 words × 64-bit) in each MMCSD Subsystem. One SRAM dedicated for transmit and one SRAM dedicated for receive operations.

Figure 12-110 shows the ECC Aggregator block diagram.

Copyright © 2024 Texas Instruments Incorporated

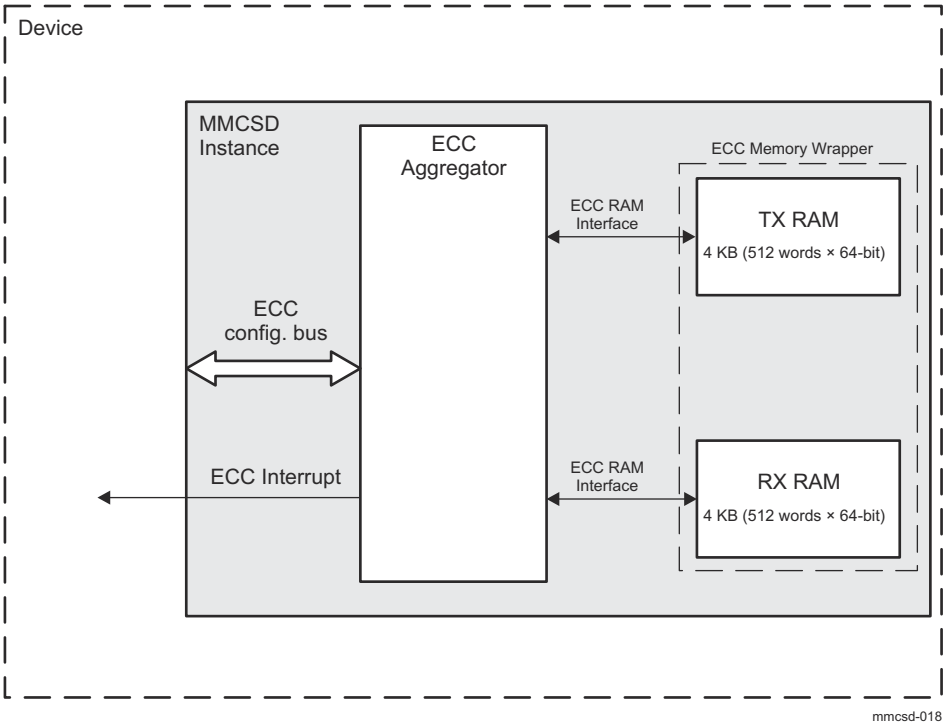


Figure 12-110. ECC Aggregator Block Diagram

Note

For more information about ECC Aggregator Registers, refer to *MMCSD Registers*.

12.3.6.3.5 Advanced DMA

The MMCSD modules support SD Advanced DMA – ADMA2 and ADMA3.

Note

For more information, refer to the SD Association specification titled *"SD Host Controller Simplified Specification, Part A2, Version 4.10"*.

12.3.6.3.6 eMMC PHY BIST

12.3.6.3.6.1 BIST Overview

To verify the eMMC5.1 PHY functionality, the BIST function is built on top of the eMMC5.1 PHY that can either be controlled from SOC registers or from special Vendor Registers defined in the TI's eMMC5.1 Host Controller. The functionality of the BIST is to be able to test various functions of the PHY (without using the eMMC protocol/ software stack) and provide the results of the testing to the diagnostic software.

The BIST logic drives a known pattern on the transmit path (into DFE) and is received on the receive path (as the IOs are bidirectional in nature). The received data is captured in the BIST logic (after the DFE) and are compared. The result of the pattern comparision is reported into status register for software to read out.

The fixed pattern is the well-known tuning pattern used in HS200 mode as defined by the JEDEC. This pattern is 128-bytes in length, is preceded by eMMC START Condition on all lanes, and is terminated with END Condition on all lanes. Unlike the normal Tuning command, there is no CRC appended to this pattern.

The 128-byte Test Pattern is as follows:

Copyright © 2024 Texas Instruments Incorporated

F	F	0	F	F	F	0	0	F	F	C	C	C	3	C	C	C	3	3	C	C	C	F	F	F	E	F	F	F	E	E	F
F	F	0	F	F	F	0	0	F	F	C	C	C	3	C	C	C	3	3	C	C	C	F	F	F	E	F	F	F	E	E	F
Byte #1	Byte #2	Byte #3	Byte #4	Byte #5	Byte #6	Byte #7	Byte #8	Byte #9	Byte #10	Byte #11	Byte #12	Byte #13	Byte #14	Byte #15	Byte #16	Byte #17	Byte #18	Byte #19	Byte #20	Byte #21	Byte #22	Byte #23	Byte #24	Byte #25	Byte #26	Byte #27	Byte #28	Byte #29	Byte #30	Byte #31	Byte #32

F	F	D	F	F	F	D	D	F	F	F	B	F	F	F	B	B	F	F	F	7	F	F	F	7	7	F	7	B	D	E	F
F	F	D	F	F	F	D	D	F	F	F	B	F	F	F	B	B	F	F	F	7	F	F	F	7	7	F	7	B	D	E	F
Byte #33	Byte #34	Byte #35	Byte #36	Byte #37	Byte #38	Byte #39	Byte #40	Byte #41	Byte #42	Byte #43	Byte #44	Byte #45	Byte #46	Byte #47	Byte #48	Byte #49	Byte #50	Byte #51	Byte #52	Byte #53	Byte #54	Byte #55	Byte #56	Byte #57	Byte #58	Byte #59	Byte #60	Byte #61	Byte #62	Byte #63	Byte #64

F	F	F	0	F	F	F	0	0	F	F	C	C	C	3	C	C	C	3	3	C	C	C	F	F	F	E	F	F	F	E	E
F	F	F	0	F	F	F	0	0	F	F	C	C	C	3	C	C	C	3	3	C	C	C	F	F	F	E	F	F	F	E	E
Byte #65	Byte #66	Byte #67	Byte #68	Byte #69	Byte #70	Byte #71	Byte #72	Byte #73	Byte #74	Byte #75	Byte #76	Byte #77	Byte #78	Byte #79	Byte #80	Byte #81	Byte #82	Byte #83	Byte #84	Byte #85	Byte #86	Byte #87	Byte #88	Byte #89	Byte #90	Byte #91	Byte #92	Byte #93	Byte #94	Byte #95	Byte #96

F	F	F	D	F	F	F	D	D	F	F	F	B	F	F	F	B	B	F	F	F	7	F	F	F	7	7	F	7	B	D	E
F	F	F	D	F	F	F	D	D	F	F	F	B	F	F	F	B	B	F	F	F	7	F	F	F	7	7	F	7	B	D	E
Byte #97	Byte #98	Byte #99	Byte #100	Byte #101	Byte #102	Byte #103	Byte #104	Byte #105	Byte #106	Byte #107	Byte #108	Byte #109	Byte #110	Byte #111	Byte #112	Byte #113	Byte #114	Byte #115	Byte #116	Byte #117	Byte #118	Byte #119	Byte #120	Byte #121	Byte #122	Byte #123	Byte #124	Byte #125	Byte #126	Byte #127	Byte #128

Figure 12-111. 128-byte Test Pattern

In non-HS200 Modes, the testing is performed for one time with programmed Tx Phase and Rx Phase values. In HS200 Mode, the testing is performed 32-times and it is expected that some of the iterations will have data mismatch because of the RxPhase Timing. The BIST engine iterates through all 32-phases of Rx Clock and provides the result of data/cmd line compare for each of the phases in a 32-bit status vector.

12.3.6.3.6.2 BIST Modes

The following are the various modes supported by the eMMC5.1 PHY BIST Logic. These modes test both the CMD line and all DATA Lines.

12.3.6.3.6.2.1 DS Mode

This is the Default Speed (DS) mode where the eMMC CLK is set to 25 MHz and the timing of the interface is from negative edge launch to positive edge capture. The DLL and the DLY lines are disabled in this phase.

12.3.6.3.6.2.2 HS Mode with TXDLY using DLL

This is the High Speed (HS) mode where the eMMC CLK is set to 50 MHz. To emulate the Interface timing, a Hold time is inserted on Transmit data lines by using the Phase shifted Tx Clock. The amount of phase shift can be from 1 to 16 Taps. In this mode, the TX Clock Phase shift is being performed by using one of the first 16-taps of the DLL TXCLK Phases. The Phase shifting on the RX path is disabled in this mode.

12.3.6.3.6.2.3 HS Mode with TXDLY using Delay Chain

This is the High Speed (HS) mode where the eMMC CLK is set to 50 MHz. To emulate the Interface timing, a Hold time is inserted on Transmit data lines by using the Phase shifted Tx Clock. The amount of phase shift can be from 1 to 16 Taps. In this mode, the TX Clock Phase shift is being performed by using one of the first 16-taps of the TX Delay Chain Phases. The Phase shifting on the RX path is disabled in this mode.

The DLL is disabled in this mode.

12.3.6.3.6.2.4 DDR50 Mode with TXDLY using DLL

This is the DDR50 mode where the eMMC CLK is set to 50 MHz. The data is driven on both the edges of the clock. To emulate the Interface timing, a Hold time is inserted on Transmit data lines by using the Phase shifted TX Clock. The amount of phase shift can be from 1 to 16 Taps. In this mode, the TX Clock Phase shift is being

Copyright © 2024 Texas Instruments Incorporated

performed by using one of the first 16-taps of the DLL TXCLK Phases. The Phase shifting on the RX path is disabled in this mode.

12.3.6.3.6.2.5 DDR50 Mode with TXDLY using Delay Chain

This is the DDR50 mode where the eMMC CLK is set to 50 MHz. The data is driven on both the edges of the clock. To emulate the Interface timing, a Hold time is inserted on Transmit data lines by using the Phase shifted Tx Clock. The amount of phase shift can be from 1 to 16 Taps. In this mode, the Tx Clock Phase shift is being performed by using one of the first 16-taps of the Tx Delay Chain Phases. The Phase shifting on the RX path is disabled in this mode.The DLL is disabled in this mode.

12.3.6.3.6.2.6 HS200 Mode with TX/RXDLY using DLL

This is the HS200 mode where the eMMC CLK is set to 200 MHz. To emulate the Interface timing, a small Hold time is inserted on Transmit data lines by using the Phase shifted TX Clock. The amount of phase shift can be from 1 to 16 Taps. The RxClock Phase is adjusted from 1 to 32 Phases to check the data at different phases. In this mode, the TX Phase shift is being performed by using one of the first 16-taps of the DLL TXCLK Phases and the RX Phase shift is being performed by using one of the 32 taps of the DLL RXCLK Phases.

12.3.6.3.6.2.7 HS200 Mode with TX/RXDLY using Delay Chain

This is the HS200 mode where the eMMC CLK is set to 200 MHz. To emulate the Interface timing, a small Hold time is inserted on Transmit data lines by using the Phase shifted Tx Clock. The amount of phase shift can be from 1 to 16 Taps. The Rx Clock Phase is adjusted from 1 to 32 Phases to check the data at different phases. In this mode, the TX Phase shift is being performed by using one of the first 16-taps of the Delay Chain Phases and the RX Phase shift is being performed by using one of the 32 taps of the Delay Chain Phases.

The DLL is disabled in this mode.

12.3.6.3.6.2.8 HS400 Mode

In this mode, the Receive data is captured using the STRB line. To support this mode, the AFE is configured such that the STRB is toggled during the time when the fixed pattern transmission (including the START/END). The Receive STRB line is phase shifted by 90 and 180 degrees by the DLL. As the data is captured using the STRB, there is no need to enable the RX Clock Phases, and these can be disabled. To maintain the timing, the TX Phase can be adjusted accordingly.

12.3.6.3.6.3 BIST Functionality

The BIST engine inside the eMMC PHY leverages the Tuning Pattern to verify the functionality of the PHY (including the IOs and the DLL). The BIST engine is controlled from SOC and the eMMC Host Controller should be disabled during the BIST Mode. The BIST can be run at different modes (Clock Speeds) and the result of the BIST operation is presented to the SOC.

In normal mode, the PHY Transmit Data path operates on the MMCSD0_CLOCK_CONTROL[15:8].SDCLK_FRQSEL. This clock is provided by the eMMC Host Controller and the frequency of this clock is varied based on the operating mode. The following are the various frequencies used in various modes.

- Default Speed: 25 MHz
- High Speed: 25 MHz
- DDR52: 50 MHz
- HS200: 200 MHz
- HS400: 200 MHz

This clock is generated by the eMMC Host Controller and under direct control of the software through the Clock Control Register @0x2C of the Host Controller Register Set. The eMMC Host Controller uses the XIN_CLK (the reference clock to the Host Controller, which is usually 200 MHz) and uses the Clock Divider Value and the Clock Enable Control.

The BIST Engine runs on the same MMCSD0_CLOCK_CONTROL[15:8].SDCLK_FRQSEL. Though the Host Controller is disabled during the BIST operation, the software should program the Clock Control Register so that the MMCSD0_CLOCK_CONTROL[15:8].SDCLK_FRQSEL is generated for the BIST engine.

Copyright © 2024 Texas Instruments Incorporated

12.3.6.3.6.4 Signal Interface

The BIST logic is controlled by a set of signals which can be from a set of registers that are software accessible. The following are the signals for BIST Functions.

Table 12-224. BIST Logic Signals

Signal	Direction	Description
MMCSD0_SS_PHY_CTRL_6_REG[31].BISTENABLE	Input	**BIST Enable** When set the BIST Controller controls the DFE and overwrites the data path signals from the Host Controller Interface. When this is cleared, the Host Controller signals are connected to the DFE and BIST is disabled.
MMCSD0_SS_PHY_CTRL_6_REG[27:24].BISTMODE	Input	**BIST Mode** This selects one of the specified BIST Modes for running the BIST sequence. The following are the encoding for the mode: 4'b0000: HS200 Mode with DLL 4'b0001: HS200 Mode with DLY_CHAIN 4'b0010: HS400 Mode (STROBE Mode) 4'b0011: DDR52 Mode with DLL 4'b0100: DDR52 Mode with DLY_CHAIN 4'b0101: HS Mode with DLL 4'b0110: HS Mode with DLY_CHAIN 4'b0111: DS Mode (with DLY_CHAIN)
MMCSD0_SS_PHY_CTRL_6_REG[30].BISTSTART	Input	**BIST Start** The rising edge of this signal is used to start the BIST function on the specified BIST Mode. When the BIST is complete the MMCSD0_SS_PHY_STAT_1_REG[3].BISTDONE output is asserted. Before setting this signal, the SOC must make sure that the MMCSD0_SS_PHY_STAT_1_REG[3].BISTDONE is low (deasserted).
MMCSD0_SS_PHY_STAT_2_REG[31:0].BISTSTATUS	Output	**BIST Status** This is the status of the BIST Testing. In non HS200 Mode, the Bit[0] indicates whether the BIST Passed or not. In HS200 modes, the 32-bit indicates the Pattern Match at each of the 32-taps on RxClk Path.
MMCSD0_SS_PHY_STAT_1_REG[3].BISTDONE	Output	**BIST Done** This is asserted when the BIST completes its sequence. The result of the test is presented in the MMCSD0_SS_PHY_STAT_2_REG[31:0].BISTSTATUS output. This is asserted until the MMCSD0_SS_PHY_CTRL_6_REG[30].BISTSTART is deasserted at which point the MMCSD0_SS_PHY_STAT_1_REG[3].BISTDONE is also cleared.

12.3.6.3.6.5 Programming Flow

For the BIST operation, the OD_EN/REN/PU/OD controls to the CMD/STRB/DAT lines should be programmed as follows:

- OD_EN: 1'b0
- REN: 1'b0
- PU: 1'b1
- OD: 1'b0

Copyright © 2024 Texas Instruments Incorporated

12.3.6.3.6.5.1 DS Mode

The following are the various settings required for performing the BIST in Default Speed timing.

12.3.6.3.6.5.1.1 Configuration

- MMCSD0_CLOCK_CONTROL[15:8].SDCLK_FRQSEL should be set to 25 MHz.
- PHY's DLL should be disabled.
- MMCSD0_SS_PHY_CTRL_6_REG[27:24].BISTMODE should be set to 4'b0111.
- MMCSD0_SS_PHY_CTRL_6_REG[31].BISTENABLE should be set to 1'b1.
- MMCSD0_SS_PHY_CTRL_4_REG[20].OTAPDLYENA to 1'b0.
- MMCSD0_SS_PHY_CTRL_4_REG[15:12].OTAPDLYSEL to 4'b0000.
- MMCSD0_SS_PHY_CTRL_4_REG[8].ITAPDLYENA to 1'b0
- MMCSD0_SS_PHY_CTRL_4_REG[4:0].ITAPDLYSEL to 5'b00000
- MMCSD0_SS_PHY_CTRL_4_REG[9].ITAPCHGWIN to 1'b0

12.3.6.3.6.5.1.2 BIST Programming

1. Set the MMCSD0_SS_PHY_CTRL_6_REG[30].BISTSTART to 1'b1.
2. Wait until the MMCSD0_SS_PHY_STAT_1_REG[3].BISTDONE is set to 1'b1.
3. Read the MMCSD0_SS_PHY_STAT_2_REG[31:0].BISTSTATUS[0] Bit and check the value.

   If (MMCSD0_SS_PHY_STAT_2_REG[31:0].BISTSTATUS*0+==1'b1) then

   BIST PASS

   else

   BIST FAIL;

   endif
4. Clear the MMCSD0_SS_PHY_CTRL_6_REG[30].BISTSTART to 1'b0.

12.3.6.3.6.5.2 HS Mode with DLY_CHAIN

The following are the various settings required for performing the BIST in HS Mode with DLY Chain.

12.3.6.3.6.5.2.1 Configuration

- MMCSD0_CLOCK_CONTROL[15:8].SDCLK_FRQSEL should be set to 50 MHz.
- PHY's DLL should be disabled.
- MMCSD0_SS_PHY_CTRL_6_REG[27:24].BISTMODE should be set to 4'b0110.
- MMCSD0_SS_PHY_CTRL_6_REG[31].BISTENABLE should be set to 1'b1.
- MMCSD0_SS_PHY_CTRL_4_REG[20].OTAPDLYENA to 1'b1.
- MMCSD0_SS_PHY_CTRL_4_REG[15:12].OTAPDLYSEL to on of the tap Value based on Timing Closure.
- MMCSD0_SS_PHY_CTRL_4_REG[8].ITAPDLYENA* to 1'b0.
- MMCSD0_SS_PHY_CTRL_4_REG[4:0].ITAPDLYSEL* to 5'b00000
- MMCSD0_SS_PHY_CTRL_4_REG[9].ITAPCHGWIN to 1'b0

* If Timing closure requires usage of Input Tap Enable and Tap Select should be set appropriately.

12.3.6.3.6.5.2.2 BIST Programming

1. Set the MMCSD0_SS_PHY_CTRL_6_REG[30].BISTSTART to 1'b1.
2. Wait until the MMCSD0_SS_PHY_STAT_1_REG[3].BISTDONE is set to 1'b1.
3. Read the MMCSD0_SS_PHY_STAT_2_REG[31:0].BISTSTATUS[0] Bit and check the value.

   If (MMCSD0_SS_PHY_STAT_2_REG[31:0].BISTSTATUS*0+==1'b1) then

   BIST PASS

   else

   BIST FAIL;

   endif
4. Clear the MMCSD0_SS_PHY_CTRL_6_REG[30].BISTSTART to 1'b0.

Copyright © 2024 Texas Instruments Incorporated

12.3.6.3.6.5.3 HS Mode with DLL

The following are the various settings required for performing the BIST in HS Mode with DLL.

12.3.6.3.6.5.3.1 Configuration

- MMCSD0_CLOCK_CONTROL[15:8].SDCLK_FRQSEL should be set to 50 MHz.
- PHY's DLL should be Enabled.
- MMCSD0_SS_PHY_CTRL_6_REG[27:24].BISTMODE should be set to 4'b0101.
- MMCSD0_SS_PHY_CTRL_6_REG[31].BISTENABLE should be set to 1'b1.
- MMCSD0_SS_PHY_CTRL_4_REG[20].OTAPDLYENA to 1'b1.
- MMCSD0_SS_PHY_CTRL_4_REG[15:12].OTAPDLYSEL to on of the tap Value based on Timing Closure.
- MMCSD0_SS_PHY_CTRL_4_REG[8].ITAPDLYENA* to 1'b0.
- MMCSD0_SS_PHY_CTRL_4_REG[4:0].ITAPDLYSEL* to 5'b00000
- MMCSD0_SS_PHY_CTRL_4_REG[9].ITAPCHGWIN to 1'b0
- MMCSD0_SS_PHY_CTRL_5_REG[10:8].FRQSEL should be set for 50 MHz Mode.

* If Timing closure requires usage of Input Tap Enable and Tap Select should be set appropriately.

12.3.6.3.6.5.3.2 BIST Programming

1. Wait for MMCSD0_SS_PHY_STAT_1_REG[0].DLLRDY to be 1'b1 (DLL is Ready)
2. Set the MMCSD0_SS_PHY_CTRL_6_REG[30].BISTSTART to 1'b1.
3. Wait until the MMCSD0_SS_PHY_STAT_1_REG[3].BISTDONE is set to 1'b1.
4. Read the MMCSD0_SS_PHY_STAT_2_REG[31:0].BISTSTATUS[0] Bit and check the value.

   If (MMCSD0_SS_PHY_STAT_2_REG[31:0].BISTSTATUS*0+==1'b1) then

   BIST PASS

   else

   BIST FAIL;

   endif
5. Clear the MMCSD0_SS_PHY_CTRL_6_REG[30].BISTSTART to 1'b0.

12.3.6.3.6.5.4 DDR52 Mode with DLY_CHAIN

The following are the various settings required for performing the BIST in DDR52 Mode with DLY Chain.

12.3.6.3.6.5.4.1 Configuration

- MMCSD0_CLOCK_CONTROL[15:8].SDCLK_FRQSEL should be set to 50 MHz.
- PHY's DLL should be disabled.
- MMCSD0_SS_PHY_CTRL_6_REG[27:24].BISTMODE should be set to 4'b0100.
- MMCSD0_SS_PHY_CTRL_6_REG[31].BISTENABLE should be set to 1'b1.
- MMCSD0_SS_PHY_CTRL_4_REG[20].OTAPDLYENA to 1'b1.
- MMCSD0_SS_PHY_CTRL_4_REG[15:12].OTAPDLYSEL to on of the tap Value based on Timing Closure.
- MMCSD0_SS_PHY_CTRL_4_REG[8].ITAPDLYENA* to 1'b0.
- MMCSD0_SS_PHY_CTRL_4_REG[4:0].ITAPDLYSEL* to 5'b00000
- MMCSD0_SS_PHY_CTRL_4_REG[9].ITAPCHGWIN to 1'b0

If Timing closure requires usage of Input Tap Enable and Tap Select should be set appropriately.

12.3.6.3.6.5.4.2 BIST Programming

1. Set the MMCSD0_SS_PHY_CTRL_6_REG[30].BISTSTART to 1'b1.
2. Wait until the MMCSD0_SS_PHY_STAT_1_REG[3].BISTDONE is set to 1'b1.
3. Read the MMCSD0_SS_PHY_STAT_2_REG[31:0].BISTSTATUS[0] Bit and check the value.

   If (MMCSD0_SS_PHY_STAT_2_REG[31:0].BISTSTATUS*0+==1'b1) then

   BIST PASS

   else

Copyright © 2024 Texas Instruments Incorporated

BIST FAIL;

endif

4. Clear the MMCSD0_SS_PHY_CTRL_6_REG[30].BISTSTART to 1'b0.

12.3.6.3.6.5.5 DDR52 Mode with DLL

The following are the various settings required for performing the BIST in DDR52 Mode with DLL.

12.3.6.3.6.5.5.1 Configuration

- MMCSD0_CLOCK_CONTROL[15:8].SDCLK_FRQSEL should be set to 50 MHz
- PHY's DLL should be Enabled.
- MMCSD0_SS_PHY_CTRL_6_REG[27:24].BISTMODE should be set to 4'b0011.
- MMCSD0_SS_PHY_CTRL_6_REG[31].BISTENABLE should be set to 1'b1.
- MMCSD0_SS_PHY_CTRL_4_REG[20].OTAPDLYENA to 1'b1.
- MMCSD0_SS_PHY_CTRL_4_REG[15:12].OTAPDLYSEL to one of the tap values based on Timing Closure.
- MMCSD0_SS_PHY_CTRL_4_REG[8].ITAPDLYENA* to 1'b0.
- MMCSD0_SS_PHY_CTRL_4_REG[4:0].ITAPDLYSEL* to 5'b00000
- MMCSD0_SS_PHY_CTRL_4_REG[9].ITAPCHGWIN to 1'b0
- MMCSD0_SS_PHY_CTRL_5_REG[10:8].FRQSEL should be set for 50MHz Mode.

If Timing closure requires usage of Input Tap Enable and Tap Select should be set appropriately.

12.3.6.3.6.5.5.2 BIST Programming

1. Wait for MMCSD0_SS_PHY_STAT_1_REG[0].DLLRDY to be 1'b1 (DLL is Ready)
2. Set the MMCSD0_SS_PHY_CTRL_6_REG[30].BISTSTART to 1'b1.
3. Wait until the MMCSD0_SS_PHY_STAT_1_REG[3].BISTDONE is set to 1'b1.
4. Read the MMCSD0_SS_PHY_STAT_2_REG[31:0].BISTSTATUS[0] Bit and check the value.

   If (MMCSD0_SS_PHY_STAT_2_REG[31:0].BISTSTATUS*0+==1'b1) then

   BIST PASS

   else

   BIST FAIL;

   endif
5. Clear the MMCSD0_SS_PHY_CTRL_6_REG[30].BISTSTART to 1'b0.

12.3.6.3.6.5.6 HS200 Mode with DLY_CHAIN

The following are the various settings required for performing the BIST in HS200 Mode with DLY Chain.

12.3.6.3.6.5.6.1 Configuration

- MMCSD0_CLOCK_CONTROL[15:8].SDCLK_FRQSEL should be set to 200 MHz.
- PHY's DLL should be disabled.
- MMCSD0_SS_PHY_CTRL_6_REG[27:24].BISTMODE should be set to 4'b0001.
- MMCSD0_SS_PHY_CTRL_6_REG[31].BISTENABLE should be set to 1'b1.
- MMCSD0_SS_PHY_CTRL_4_REG[20].OTAPDLYENA to 1'b1.
- MMCSD0_SS_PHY_CTRL_4_REG[15:12].OTAPDLYSEL to one of the tap values based on Timing Closure.
- MMCSD0_SS_PHY_CTRL_4_REG[8].ITAPDLYENA to 1'b0
- MMCSD0_SS_PHY_CTRL_4_REG[4:0].ITAPDLYSEL to 5'b00000
- MMCSD0_SS_PHY_CTRL_4_REG[9].ITAPCHGWIN to 1'b0

*In this mode, the BIST Engine cycles thru all the 32 taps, so there is no need of controlling the Input Tap Delay.

12.3.6.3.6.5.6.2 BIST Programming

1. Set the MMCSD0_SS_PHY_CTRL_6_REG[30].BISTSTART to 1'b1.
2. Wait until the MMCSD0_SS_PHY_STAT_1_REG[3].BISTDONE is set to 1'b1.
3. Read the MMCSD0_SS_PHY_STAT_2_REG[31:0].BISTSTATUS*31:0+ Bit and run the "HS200 Bist Result Check Procedure"

Copyright © 2024 Texas Instruments Incorporated

4. Clear the MMCSD0_SS_PHY_CTRL_6_REG[30].BISTSTART to 1'b0.

12.3.6.3.6.5.7 HS200 Mode with DLL

The following are the various settings required for performing the BIST in HS200 Mode with DLL.

12.3.6.3.6.5.7.1 Configuration

- MMCSD0_CLOCK_CONTROL[15:8].SDCLK_FRQSEL should be set to 200 MHz.
- PHY's DLL should be disabled.
- MMCSD0_SS_PHY_CTRL_6_REG[27:24].BISTMODE should be set to 4'b0000.
- MMCSD0_SS_PHY_CTRL_6_REG[31].BISTENABLE should be set to 1'b1.
- MMCSD0_SS_PHY_CTRL_4_REG[20].OTAPDLYENA to 1'b1.
- MMCSD0_SS_PHY_CTRL_4_REG[15:12].OTAPDLYSEL to one of the tap values based on Timing Closure.
- MMCSD0_SS_PHY_CTRL_4_REG[8].ITAPDLYENA* to 1'b0.
- MMCSD0_SS_PHY_CTRL_4_REG[4:0].ITAPDLYSEL* to 5'b00000
- MMCSD0_SS_PHY_CTRL_4_REG[9].ITAPCHGWIN to 1'b0
- MMCSD0_SS_PHY_CTRL_5_REG[10:8].FRQSEL should be set for 200 MHz Mode.

* If Timing closure requires usage of Input Tap Enable and Tap Select should be set appropriately.

12.3.6.3.6.5.7.2 BIST Programming

1. Wait for MMCSD0_SS_PHY_STAT_1_REG[0].DLLRDY to be 1'b1 (DLL is Ready)
2. Set the MMCSD0_SS_PHY_CTRL_6_REG[30].BISTSTART to 1'b1.
3. Wait until the MMCSD0_SS_PHY_STAT_1_REG[3].BISTDONE is set to 1'b1.
4. Read the MMCSD0_SS_PHY_STAT_2_REG[31:0].BISTSTATUS*31:0+ Bit and run the "HS200 Bist Result Check Procedure".
5. Clear the MMCSD0_SS_PHY_CTRL_6_REG[30].BISTSTART to 1'b0.

12.3.6.3.6.5.8 HS400 Mode with DLL

The following are the various settings required for performing the BIST in HS400 Mode with DLL.

12.3.6.3.6.5.8.1 Configuration

- MMCSD0_CLOCK_CONTROL[15:8].SDCLK_FRQSEL should be set to 200 MHz.
- PHY's DLL should be enabled.
- MMCSD0_SS_PHY_CTRL_6_REG[27:24].BISTMODE should be set to 4'b0010.
- MMCSD0_SS_PHY_CTRL_6_REG[31].BISTENABLE should be set to 1'b1.
- MMCSD0_SS_PHY_CTRL_4_REG[20].OTAPDLYENA to 1'b1.
- MMCSD0_SS_PHY_CTRL_4_REG[15:12].OTAPDLYSEL to one of the tap values based on Timing Closure.
- MMCSD0_SS_PHY_CTRL_4_REG[8].ITAPDLYENA to 1'b0
- MMCSD0_SS_PHY_CTRL_4_REG[4:0].ITAPDLYSEL to 5'b00000
- MMCSD0_SS_PHY_CTRL_4_REG[9].ITAPCHGWIN to 1'b0
- MMCSD0_SS_PHY_CTRL_5_REG[10:8].FRQSEL should be set for 200 MHz Mode.
- MMCSD0_SS_PHY_CTRL_4_REG[31:24].STRBSEL should be set to appropriate values based on based on STRB timing closure.

* If Timing closure requires usage of Input Tap Enable and Tap Select should be set appropriately.

12.3.6.3.6.5.8.2 BIST Programming

1. Wait for MMCSD0_SS_PHY_STAT_1_REG[0].DLLRDY to be 1'b1 (DLL is Ready)
2. Set the MMCSD0_SS_PHY_CTRL_6_REG[30].BISTSTART to 1'b1.
3. Wait until the MMCSD0_SS_PHY_STAT_1_REG[3].BISTDONE is set to 1'b1.
4. Read the MMCSD0_SS_PHY_STAT_2_REG[31:0].BISTSTATUS[0] Bit and check the value.

   If (MMCSD0_SS_PHY_STAT_2_REG[31:0].BISTSTATUS*0+==1'b1) then

   BIST PASS

   else

   BIST FAIL;

Copyright © 2024 Texas Instruments Incorporated

endif

5. Clear the MMCSD0_SS_PHY_CTRL_6_REG[30].BISTSTART to 1'b0.

12.3.6.3.6.6 HS200 BIST Result Check Procedure

In HS200 Mode, the BIST engine in the PHY runs Tuning Pattern test 32 times, with different TAP values selected, for each iteration. The result from the 32 iterations is presented on the MMCSD0_SS_PHY_STAT_2_REG[31:0].BISTSTATUS output. It is expected that some of the TAP values will have timing issues and results in pattern mismatch when compared with the received pattern for those taps. The PASS/FAIL for the HS200 Mode is based on the following theory.

Of the 32 taps, the Data Capture EYE for certain taps will result in mismatches, but these will be sequential tap values. Thus, in a full 32 iterations, a sequence of taps results in data mismatches. Also, based on the timing closure, the number of taps with pattern mismatch should be approximately 1/4th of the number of taps with pattern match. This 1/4th count is not absolute, as this involves various factors such as the timing closure, PVT variation, and so forth. Anywhere from 4 to 16 invalid sequential taps is normal.

For example, if 8 of the 32 sequential taps have the data mismatches, the following would be the resulting MMCSD0_SS_PHY_STAT_2_REG[31:0].BISTSTATUS output based which of the 8 sequential taps the mismatches are. The following is the notation used:

- 1: Pattern matched at this tap
- 0: Pattern match failed at this tap

Copyright © 2024 Texas Instruments Incorporated

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0

Copyright © 2024 Texas Instruments Incorporated

12.3.6.4 MMCSD Programming Guide

12.3.6.4.1 Sequences

This section defines basic sequence flow chart divided into several sub sequences.

Note

This section is applicable for MMCSD0 and MMCSD1 but MMCSD0 registers are used in the content.

Note

UHSII is not supported. For more information, see *Not Supported Features*.

"Wait for interrupts" is used in the flow chart. This means the Host Driver waits until specified interrupts are asserted. If already asserted, then fall through that step in the flow chart. Timeout checking shall be always required to detect no interrupt generated but this is not described in the flow chart.

This specification uses the double box like Figure 12-112, (the step (1) in Figure 12-116), it means that the other flows, which already are shown, shall be performed. Therefore, the interrupt may be included in the other flows.

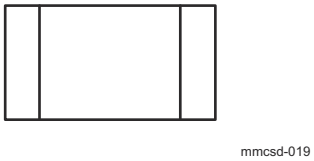


Figure 12-112. Double Box Notation

12.3.6.4.1.1 SD Card Detection

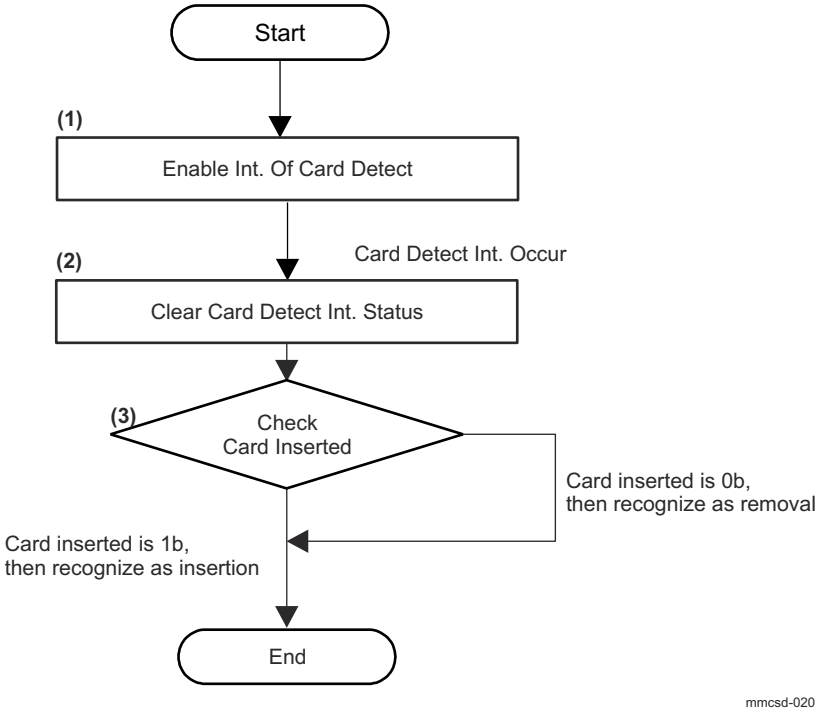


Figure 12-113. SD Card Detect Sequence

The flow chart for detecting a SD card is shown in Figure 12-113.

Each step is executed as follows:

To enable interrupt for card detection, write 1 to the following bits:

Copyright © 2024 Texas Instruments Incorporated

- MMCSD0_NORMAL_INTR_STS_ENA[6] CARD_INSERTION
- MMCSD0_NORMAL_INTR_SIG_ENA[6] CARD_INSERTION
- MMCSD0_NORMAL_INTR_STS_ENA[7] CARD_REMOVAL
- MMCSD0_NORMAL_INTR_SIG_ENA[7] CARD_REMOVAL

(1) When the Host Driver detects the card insertion or removal, clear its interrupt statuses. If Card Insertion interrupt is generated, write 1 to MMCSD0_NORMAL_INTR_STS_ENA[6] CARD_INSERTION bit. If Card Removal interrupt is generated, write 1 to MMCSD0_NORMAL_INTR_STS_ENA[7] CARD_REMOVAL bit.

(2) Check MMCSD0_PRESENTSTATE[16] CARD_INSERTED bit. In the case where MMCSD0_PRESENTSTATE[16] CARD_INSERTED bit is 1, the Host Driver can supply the power and the clock to the SD card. In the case where MMCSD0_PRESENTSTATE[16] CARD_INSERTED bit is 0, the other executing processes of the Host Driver shall be immediately closed.

If miniSD adaptor is used for standard SD slot and miniSD card is inserted or extracted from the adaptor, card detect interrupt may not be generated. When host does not receive response to any commands, miniSD card is extracted or in idle state, the host should try to re-initialize the card. In this case, all card information shall be re-loaded.

Copyright © 2024 Texas Instruments Incorporated

12.3.6.4.1.2 SD Clock Control

12.3.6.4.1.2.1 Internal Clock Setup Sequence

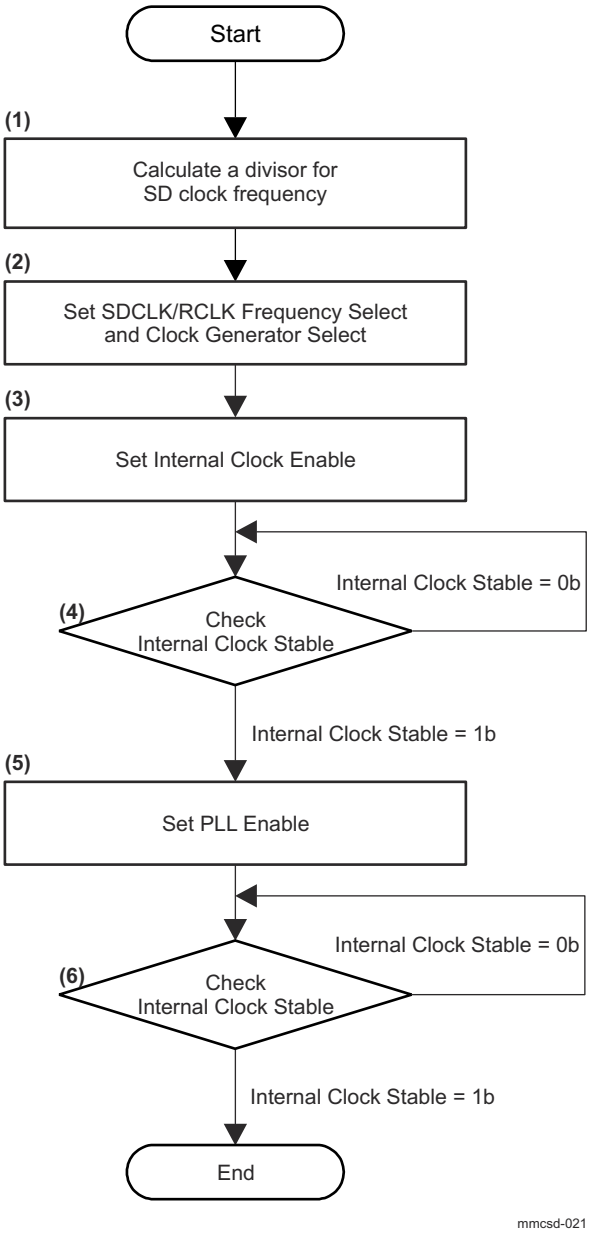


Figure 12-114. Internal Clock Setup Sequence

The sequence for supplying SD Clock to a SD card is described in Figure 12-114. From Version 4.10, MMCSD0_CLOCK_CONTROL[3] PLL_ENA bit is added. This sequence is also applicable to prior versions which do not support MMCSD0_CLOCK_CONTROL[3] PLL_ENA bit.

(1) Calculate a divisor to determine SD Clock frequency for legacy IF or RCLK frequency for UHS-II IF by reading MMCSD0_CAPABILITIES[15-8] BASE_CLK_FREQ and MMCSD0_CAPABILITIES[55-48] CLOCK_MULTIPLIER bit fields. If non-zero value is set to MMCSD0_CAPABILITIES[55-48] CLOCK_MULTIPLIER bit field, Programmable Clock Mode can be used (for more information, see MMCSD0_CLOCK_CONTROL[5] CLKGEN_SEL bit). If MMCSD0_CAPABILITIES[15-8] BASE_CLK_FREQ bit field is 00 0000b, the Host System shall provide this information to the Host Driver by another method.

(2) Set MMCSD0_CLOCK_CONTROL[15-8] SDCLK_FRQSEL bit field and MMCSD0_CLOCK_CONTROL[5] CLKGEN_SEL bit in accordance with the calculated result of step (1).

Copyright © 2024 Texas Instruments Incorporated

If MMCSD0_HOST_CONTROL2[15] PRESET_VALUE_ENA bit is set, these bits are set by Host Controller automatically as specified in the Preset Value register (MMCSD0_PRESET_VALUE0 - MMCSD0_PRESET_VALUE10).

(3) Set MMCSD0_CLOCK_CONTROL[0] INT_CLK_ENA bit.

(4) Check MMCSD0_CLOCK_CONTROL[1] INT_CLK_STABLE bit. Repeat this step until this status is 1. Clock will be stable in shorter time but timeout of this loop is defined as 150 ms.

(5) Set MMCSD0_CLOCK_CONTROL[3] PLL_ENA bit. This step does not affect Host Controllers which do not support MMCSD0_CLOCK_CONTROL[3] PLL_ENA bit.

(6) If MMCSD0_CLOCK_CONTROL[3] PLL_ENA bit is supported, PLL locked may be checked by this status (if MMCSD0_CLOCK_CONTROL[3] PLL_ENA bit is not supported, this status is supposed to indicate 1 by step (3)). Clock will be stable in shorter time but timeout of this loop is defined as 150 ms.

12.3.6.4.1.2.2 SD Clock Supply and Stop Sequence

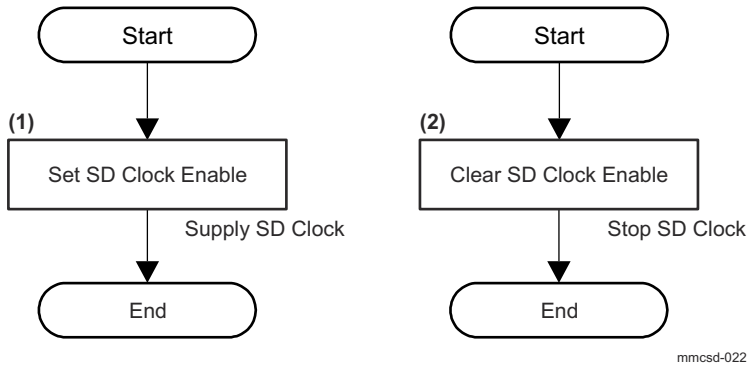


Figure 12-115. SD Clock Supply and Stop Sequence

The flow chart for stopping the SD Clock is shown in Figure 12-115. The Host Driver shall not stop the SD Clock when a SD transaction is occurring on the SD Bus -- namely, when either MMCSD0_PRESENTSTATE[1] INHIBIT_DAT or MMCSD0_PRESENTSTATE[0] INHIBIT_CMD bit is set to 1.

(1) Setup Internal Clock (see Figure 12-114). Then set MMCSD0_CLOCK_CONTROL[2] SD_CLK_ENA bit to 1. Then, the Host Controller starts supplying the SD Clock.

(2) Set MMCSD0_CLOCK_CONTROL[2] SD_CLK_ENA bit to 0. Then, the Host Controller stops supplying the SD Clock. Internal Clock is still oscillating.

Copyright © 2024 Texas Instruments Incorporated

12.3.6.4.1.2.3 SD Clock Frequency Change Sequence

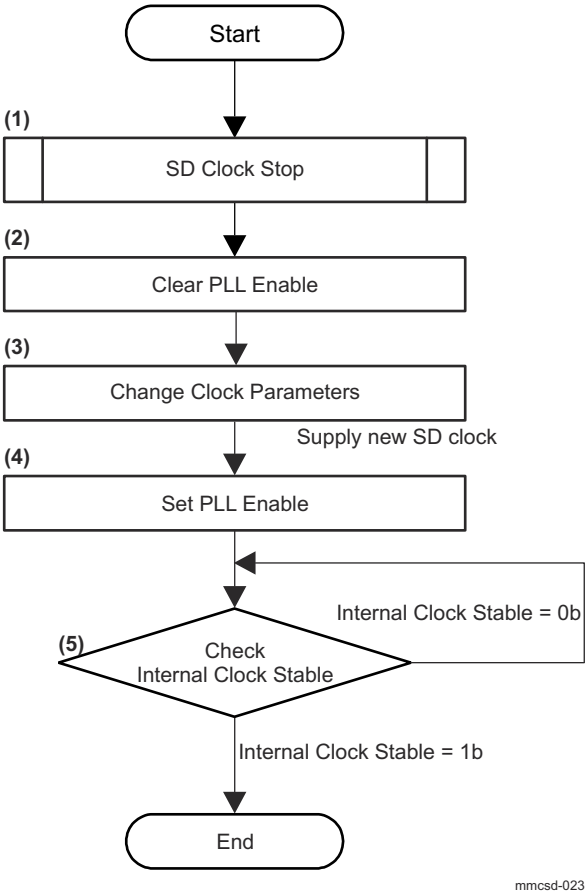


Figure 12-116. SD Clock Change Sequence

The sequence for changing SD Clock frequency is shown in Figure 12-116.

(1) Execute the SD Clock Stop Sequence (see Figure 12-115). If SD Clock supply to card is already stopped, skip this step.

(2) Clear MMCSD0_CLOCK_CONTROL[3] PLL_ENA bit to 0. If MMCSD0_CLOCK_CONTROL[3] PLL_ENA bit is not supported, this step has no effect and may be skipped.

(3) When MMCSD0_HOST_CONTROL2[15] PRESET_VALUE_ENA bit is set to 0, change clock parameters in the MMCSD0_CLOCK_CONTROL register. When MMCSD0_HOST_CONTROL2[15] PRESET_VALUE_ENA bit is set to 1, select Bus Speed Mode as described.

(4) Set MMCSD0_CLOCK_CONTROL[3] PLL_ENA bit to 1. If MMCSD0_CLOCK_CONTROL[3] PLL_ENA bit is not supported, this step this step has no effect and may be skipped.

(5) Wait until MMCSD0_CLOCK_CONTROL[1] INT_CLK_STABLE bit is set to 1. Clock will be stable in shorter time but timeout of this loop is defined as 150 ms. SD Clock Supply Sequence is required to provide clock to device (see Figure 12-115).

12.3.6.4.1.3 SD Bus Power Control

The sequence for controlling the SD Bus Power is described in Figure 12-117.

Copyright © 2024 Texas Instruments Incorporated

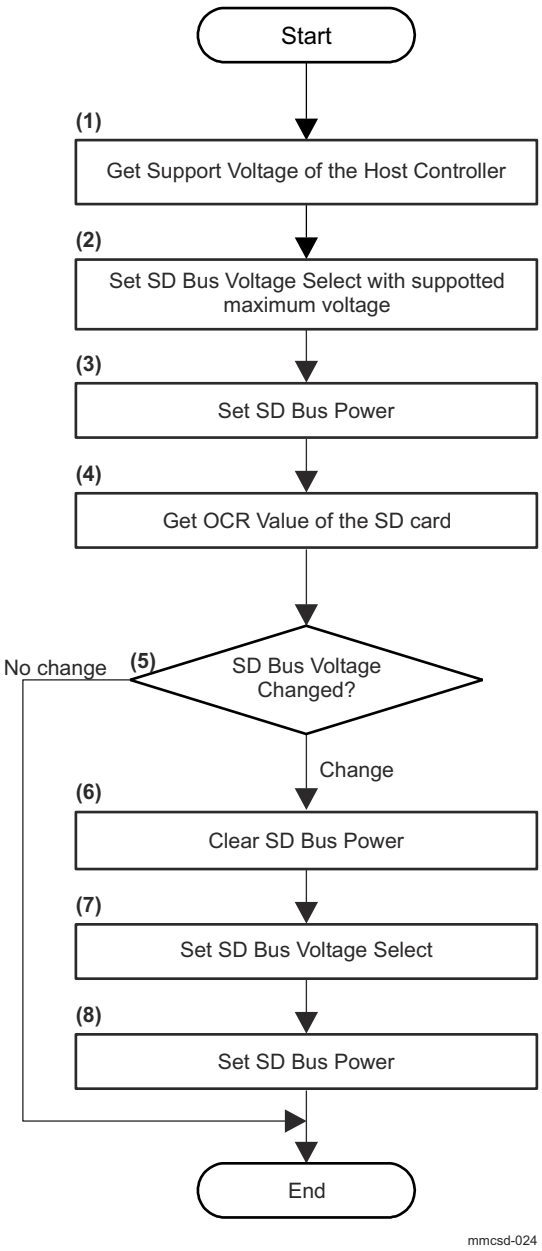


Figure 12-117. SD Bus Power Control Sequence

(1) By reading the MMCSD0_CAPABILITIES register, get the support voltage of the Host Controller.

(2) Set MMCSD0_POWER_CONTROL[3-1] SD_BUS_VOLTAGE bit field with maximum voltage that the Host Controller supports.

(3) Set MMCSD0_POWER_CONTROL[0] SD_BUS_POWER bit to 1.

(4) Get the OCR value of all function internal of SD card.

(5) Judge whether SD Bus voltage needs to be changed or not. In case where SD Bus voltage needs to be changed, go to step (6). In case where SD Bus voltage does not need to be changed, go to "End".

(6) Set MMCSD0_POWER_CONTROL[0] SD_BUS_POWER bit to 0 for clearing this bit. The card requires voltage rising from 0 volt to detect it correctly. The Host Driver shall clear MMCSD0_POWER_CONTROL[0] SD_BUS_POWER bit before changing voltage by setting MMCSD0_POWER_CONTROL[3-1] SD_BUS_VOLTAGE bit field.

Copyright © 2024 Texas Instruments Incorporated

(7) Set MMCSD0_POWER_CONTROL[3-1] SD_BUS_VOLTAGE bit field.

(8) Set MMCSD0_POWER_CONTROL[0] SD_BUS_POWER to 1.

Note: Step (2) and step (3) can be executed at same time. And also, step (7) and step (8) can be executed at same time.

12.3.6.4.1.4 Changing Bus Width

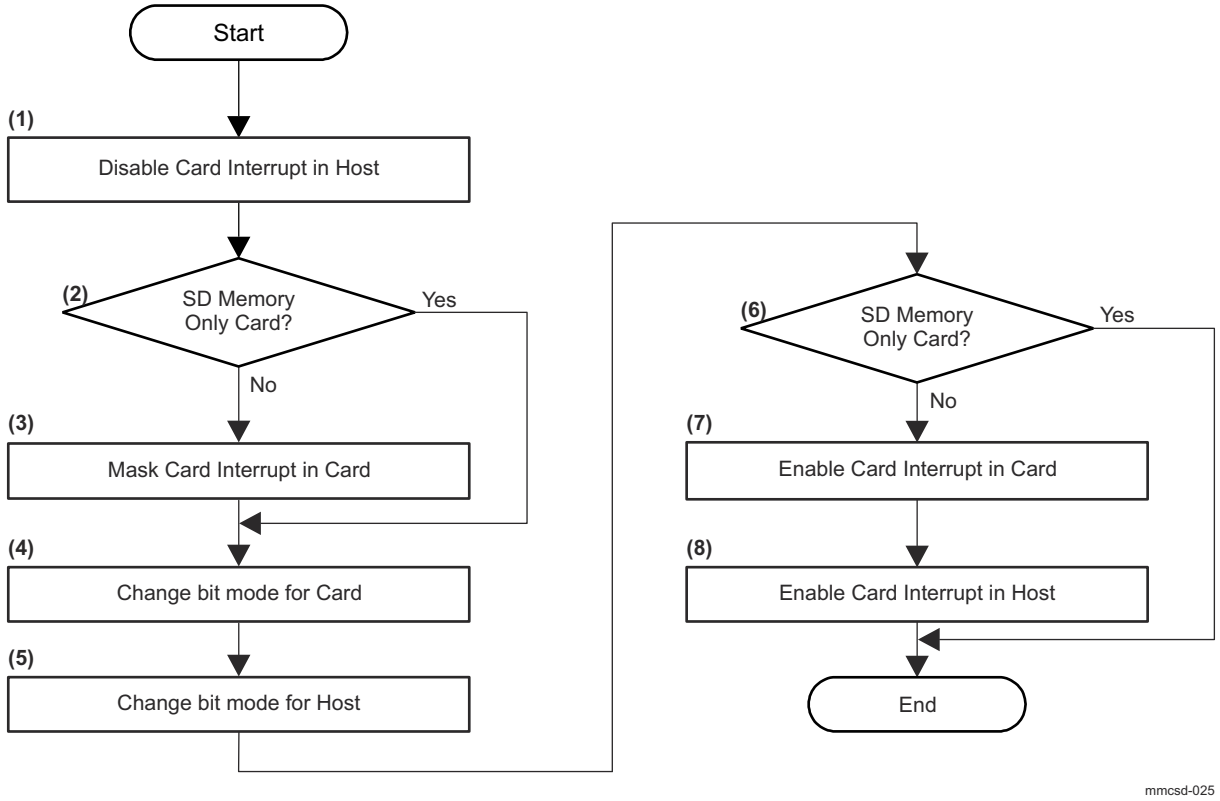


Figure 12-118. Change Bus Width Sequence

The sequence for changing bit mode on SD Bus is shown in Figure 12-118.

(1) Set MMCSD0_NORMAL_INTR_STS_ENA[8] CARD_INTERRUPT bit to 0 for masking incorrect interrupts that may occur while changing the bus width.

(2) In case of SD memory only card, go to step (4). In case of other card, go to step (3).

(3) Set "IENM" of the CCCR in a SDIO or SD combo card to 0 by CMD52.

(4) Change the bus width mode for a SD card. SD Memory Card bus width is changed by ACMD6 and SDIO card bus width is changed by setting Bus Width of Bus Interface Control register in CCCR.

(5) In case of changing to 4-bit mode, set MMCSD0_HOST_CONTROL1[1] DATA_WIDTH bit to 1. In another case (1-bit mode), set this bit to 0.

(6) In case of SD memory only card, go to the "End". In case of other card, go to step (7).

(7) Set "IENM" of the CCCR in a SDIO or SD combo card to 1 by CMD52.

(8) Set MMCSD0_NORMAL_INTR_STS_ENA[8] CARD_INTERRUPT bit to 1.

Note that if the card is locked, bus width cannot be changed. Unlock the card is required before changing bus width.

Copyright © 2024 Texas Instruments Incorporated

12.3.6.4.1.5 Timeout Setting on DAT Line

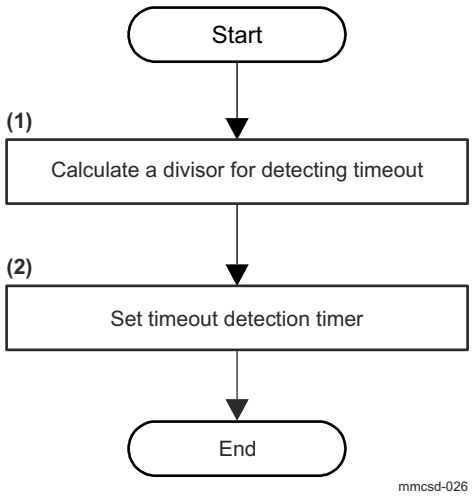


Figure 12-119. Timeout Setting Sequence

In order to detect timeout errors on DAT line, the Host Driver shall execute the following two steps before any SD transaction. For more information regarding SD transactions:

(1) Calculate a divisor to detect timeout errors by reading MMCSD0_CAPABILITIES[5-0] TIMEOUT_CLK_FREQ bit field and MMCSD0_CAPABILITIES[7] TIMEOUT_CLK_UNIT bit. If MMCSD0_CAPABILITIES[5-0] TIMEOUT_CLK_FREQ bit field is 00 0000b, the Host System shall provide this information to the Host Driver by another method.

(2) Set MMCSD0_TIMEOUT_CONTROL[3-0] COUNTER_VALUE bit field in accordance with the value from step (1) above.

12.3.6.4.1.6 Card Initialization and Identification (for SD I/F)

Figure 12-120 and Figure 12-121 shows initialization and card identification sequence for the Standard Capacity SD Memory Card (SDSC), the High Capacity SD Memory Card (SDHC) and the Extended Capacity SD Memory Card (SDXC) that was based on the Physical Layer Version 3.01. Refer to the latest sequence.

Copyright © 2024 Texas Instruments Incorporated

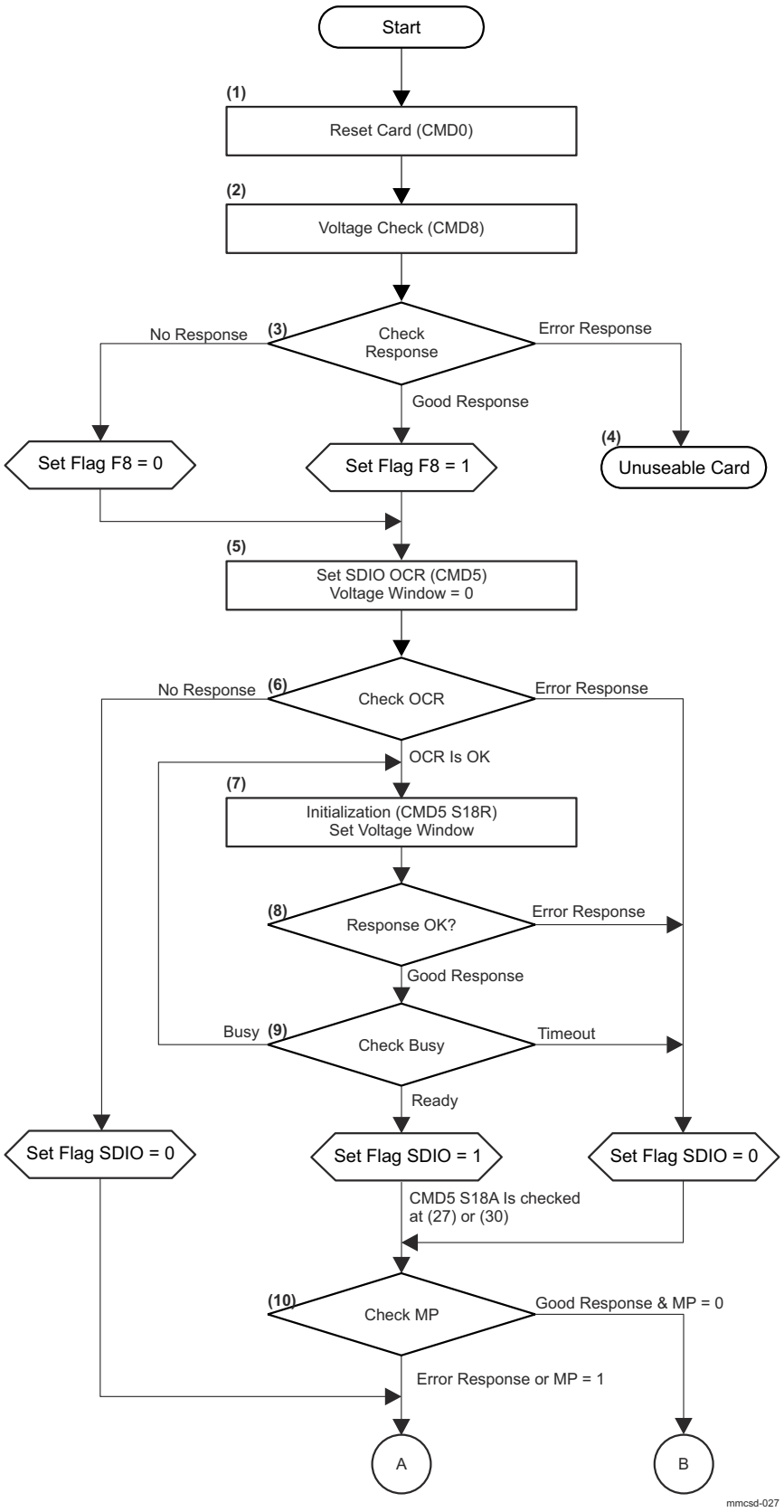


Figure 12-120. Card Initialization and Identification (1)

Copyright © 2024 Texas Instruments Incorporated

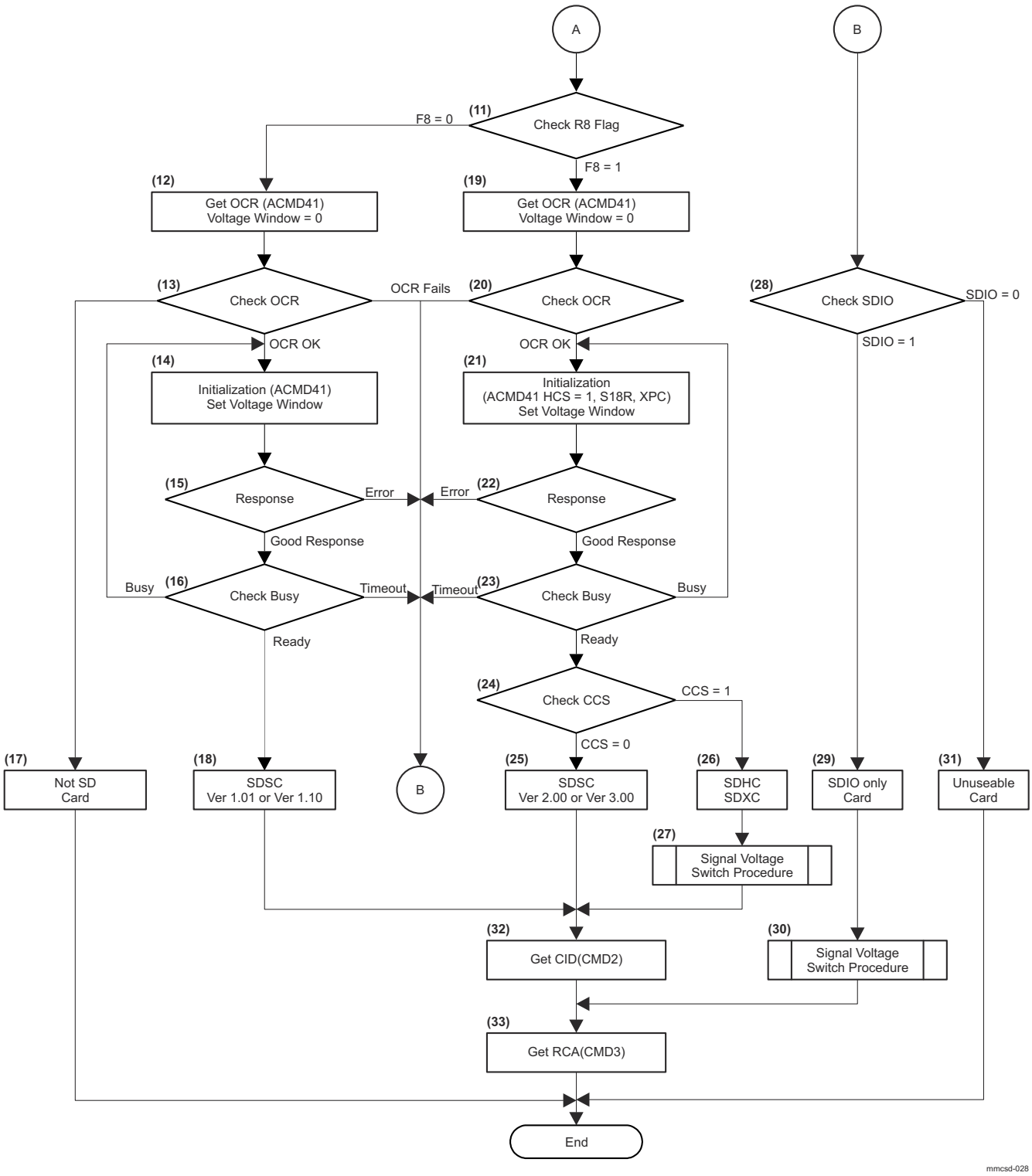


Figure 12-121. Card Initialization and Identification (2)

(1) SD Bus mode is selected by CMD0 (Keep Pin 1 to high during CMD0 execution).

(2) New CMD8 shall be issued after CMD0 to support High Capacity SD Memory Card.

(3) Voltage check command enables the Hosts to support future low voltage specification. However, at this time, only one voltage is defined. Legacy cards and Not SD cards do not respond to CMD8. In this case, set F8 to 0 (F8 is CMD8 valid flag used in step (11)) and go to Step (5). Only Version 2.00 or higher cards can respond to

Copyright © 2024 Texas Instruments Incorporated

CMD8. The host needs to check whether CRC of the response is valid and whether VHS and check pattern in the argument are equal to VCA and check pattern in the response. Passing all these checks results in CMD8 response OK. In this case, set F8 to 1 and go to step (5). If one of the checks is failed, go to step (4).

(4) Initialization is stopped by CMD8 fails. The Host Driver should retry step (1) to (3) one more time (this is not described in the figure).

(5) SDIO OCR is available by issuing CMD5 with setting voltage window (bit 23 to 0) in the argument to 0. SDIO initialization is not started.

(6) No response means the card does not have SDIO function. Set SDIO flag to 0 and go to step (11). If the card responds to CMD5 and the response is OK, go to step (7). If the response is error, set SDIO flag to 0 and go to step (10). SDIO flag indicates whether SDIO functions are initialized or not.

(7) The SDIO portion starts initialization by CMD5 with setting the supply voltage to the voltage window. UHS-I supported host sets S18R to 1. If the supplied voltage is not matched with voltage window of card, the card goes into inactive state and does not return the response.

(8) If no response or error response is receive, set SDIO flag to 0 and go to step (10). If good response is received, go to step (9).

(9) Check busy status in the response. If busy is released, set SDIO flag to 1 and go to step (10). Repeat from step (7) while busy is indicated. Detecting timeout of 1 second exits the loop. In this case, set SDIO flag to 0 and go to step (10).

(10) Good response in this step means that all responses received at (6) and (8) are valid. When response is good, MP (memory present) flag in the response can be checked. If the response valid and MP = 0, go to step (28). Otherwise, go to step (11).

(11) Check F8 flag set in step (3). If CMD8 is executed correctly (F8 = 1), go to step (19). Otherwise, go to step (12).

(12) OCR is available by issuing ACMD41 with the voltage window (bit 23 to 0) in the argument is set to 0. Memory initialization is not started. The response of CMD55 (ACMD41) may indicate illegal command error due to some SD cards do not recognized CMD8. The Host Driver should ignore this error or issue CMD0 before ACMD41 to clear this error status.

(13) If response of CMD55 is not received, the card is not SD cards and goes to (17). If the card responds to CMD55, it may also respond to CMD41. If the responses of ACMD41 are OK, go to Step (14). Otherwise, go to step (28). Locked card can be detected by the card status in the response of CMD55.

(14) The memory portion starts initialization by Issuing ACMD41 with setting the supply voltage to the voltage window. If the supplied voltage is not matched with voltage window of card, the card goes into inactive state and does not return the response.

(15) If no response or error response is received, go to step (28). If good response is received, go to step (16).

(16) Check busy status in the response. If busy is released, go to step (18). Repeat from step (14) while busy is indicated. The interval of ACMD41 shall be less than 50 ms. Detecting timeout of 1 second exits the loop and go to step (28).

(17) The host recognizes that the card is not SD memory card and quits SD card initialization.

(18) The host recognizes that the card is Version 1.xx Standard Capacity SD Memory Card. Go to Step (30).

(19) OCR is available by issuing ACMD41 with setting the voltage window (bit 23 to 0) in the argument is set to 0. Memory initialization is not started. Setting of HCS does not affect this operation.

(20) If the card responds to CMD55, it may also respond to CMD41. If the responses of ACMD41 are OK, go to Step (21). Otherwise, go to step (28). Locked card can be detected by the card status in the response of CMD55.

(21) The memory portion starts initialization by Issuing ACMD41 with setting the supply voltage to the voltage window. UHS-I supported host sets S18R to1. If the host can supply more than 150mA, XPC is set to 1. HCS in

Copyright © 2024 Texas Instruments Incorporated

the argument is set to 1, which indicates supporting High Capacity Memory Card. If the supplied voltage is not matched with voltage window of card, the card goes into inactive state and does not return the response.

(22) If no response or error response is received, go to step (28). If good response is received, go to step (23).

(23) Check busy status in the response. If busy is released, go to step (24). Repeat from step (21) while busy is indicated. The interval of ACMD41 shall be less than 50 ms. Detecting timeout of 1 second exits the loop and go to step (28).

(24) CCS in the response is valid after busy is released. If CCS = 0, it indicates the Standard Capacity SD Memory Card and go to step (25). If CCS = 1, it indicates the High Capacity SD Memory Card or Extended Capacity Memory Card and go to Step (26).

(25) The host recognizes that the card is Ver2.00 or Ver3.00 Standard Capacity SD Memory Card. Optimal functions defined in Version 2.00 or higher are available. Go to Step (32).

(26) The host recognizes that the card is the High Capacity SD Memory Card or Extended Capacity Memory Card.

(27) Perform the signal voltage switch procedure and go to step (32).

(28) Check SDIO flag. If SDIO = 1, go to step (28). Otherwise, go to step (31).

(29) The host recognizes that the card is SDIO only card and go to step (30).

(30) Perform the signal voltage switch procedure and go to step (33).

(31) The host recognizes that the card is unusable.

(32) In case of memory card, CMD2 is issued to get CID and Go to Step (31).

(33) CMD3 is issued to get RCA. If the RCA number is 0, the Host should issue CMD3 again.

Copyright © 2024 Texas Instruments Incorporated

12.3.6.4.1.6.1 Signal Voltage Switch Procedure (for UHS-I)

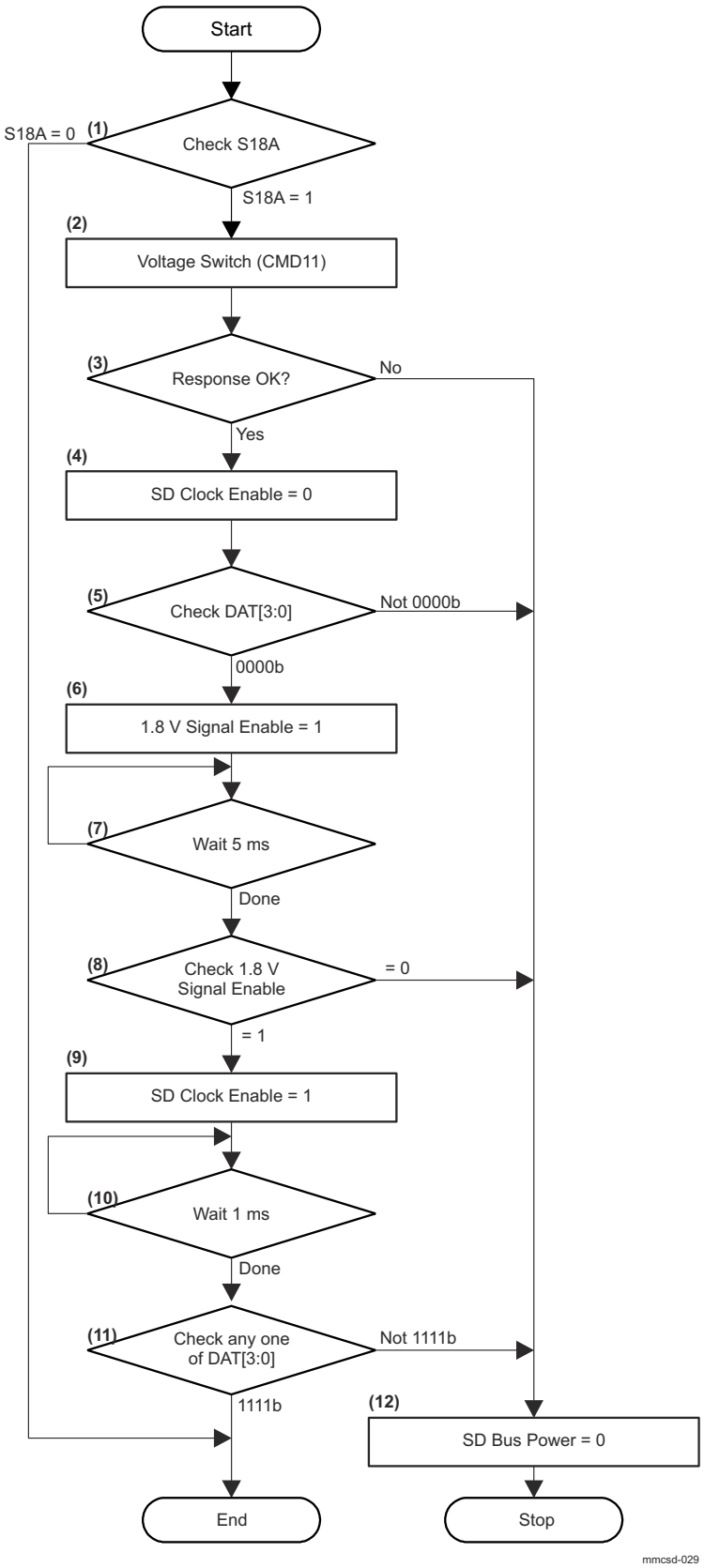


Figure 12-122. Signal Voltage Switch Procedure

Copyright © 2024 Texas Instruments Incorporated

(1) If S18A of CMD5 or S18A of ACMD41 is set to 1, signal voltage switch is performed according to the following steps. Otherwise, exits from this procedure.

(2) Issue CMD11.

(3) Check response and if an error is detected, go to step (12).

(4) Stop providing SD clock to the card.

(5) Check DAT[3:0] level. If the level is 0000b, the card is ready to start voltage switch sequence. Otherwise, go to (12) to quit the sequence.

(6) Set MMCSD0_HOST_CONTROL2[3] V1P8_SIGNAL_ENA bit.

(7) Wait 5 ms. 1.8 V voltage regulator shall be stable within this period.

(8) If MMCSD0_HOST_CONTROL2[3] V1P8_SIGNAL_ENA bit is cleared by Host Controller, go to step (12).

(9) Provide SD Clock to the card again.

(10) Wait 1 ms.

(11) Check DAT[3:0] level. If the level is 1111b, switch to 1.8 V signal level is completed successfully. Otherwise, go to (12).

(12) If an error occurs during voltage switch procedure, stop providing the power to the card. In this case, Host Driver should retry initialization procedure by setting S18R to 0 at step (7) and (21) in Figure 12-120 and Figure 12-121.

12.3.6.4.1.7 SD Transaction Generation

This section describes the sequences how to generate and control various kinds of SD transactions. SD transactions are classified into three cases:

(1) Transactions that do not use the DAT line.

(2) Transactions that use the DAT line only for the busy signal.

(3) Transactions that use the DAT line for transferring data.

In this specification the first and the second case's transactions are classified as "Transaction Control without Data Transfer using DAT Line", the third case's transaction is classified as "Transaction Control with Data Transfer using DAT Line". Refer to the latest SD Physical Layer Specification and SDIO Specification for more detail about SD commands specification.

12.3.6.4.1.7.1 Transaction Control without Data Transfer Using DAT Line

In this section, the sequence for how to issue SD Command and how to complete SD Command is explained. Figure 12-123 shows the sequence to issue a SD Command and Figure 12-124 shows the sequence to finalize a SD Command.

12.3.6.4.1.7.1.1 The Sequence to Issue a SD Command

The sequence to issue the SD Command is detailed in Figure 12-123.

Copyright © 2024 Texas Instruments Incorporated

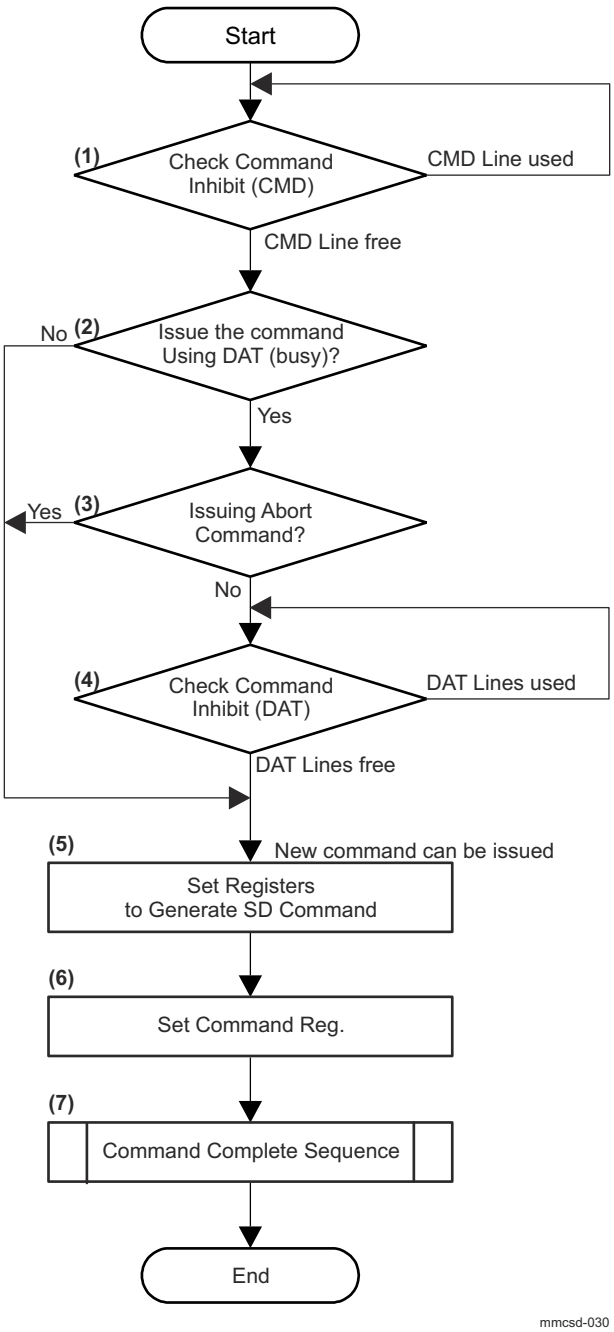


Figure 12-123. SD Command Issue Sequence

(1) Check MMCSD0_PRESENTSTATE[0] INHIBIT_CMD bit. Repeat this step until MMCSD0_PRESENTSTATE[0] INHIBIT_CMD bit is 0. That is, when MMCSD0_PRESENTSTATE[0] INHIBIT_CMD bit is 1, the Host Driver shall not issue a SD Command.

(2) If the Host Driver issues a SD Command using DAT lines including busy signal, go to step (3). If without using DAT lines including busy signal, go to step (5).

(3) If the Host Driver is issuing an abort command, go to step (5). In the case of nonabort command, go to step (4).

(4) Check MMCSD0_PRESENTSTATE[1] INHIBIT_DAT bit. Repeat this step until MMCSD0_PRESENTSTATE[1] INHIBIT_DAT bit is set to 0.

(5) Set registers except the MMCSD0_COMMAND register.

Copyright © 2024 Texas Instruments Incorporated

(6) Set the MMCSD0_COMMAND register.

Note: Writing the upper byte [3] in the MMCSD0_COMMAND register causes the Host Controller to issue a SD command to the SD card.

(7) Perform Command Completion Sequence in accordance.

12.3.6.4.1.7.1.2 The Sequence to Finalize a Command

Figure 12-124 shows the sequence to finalize a SD Command when response check is disabled. There is a possibility that some errors (Command Index/End bit/CRC/Timeout Error) occur during this sequence. If response check is enabled, error is indicated by Response Error Interrupt.

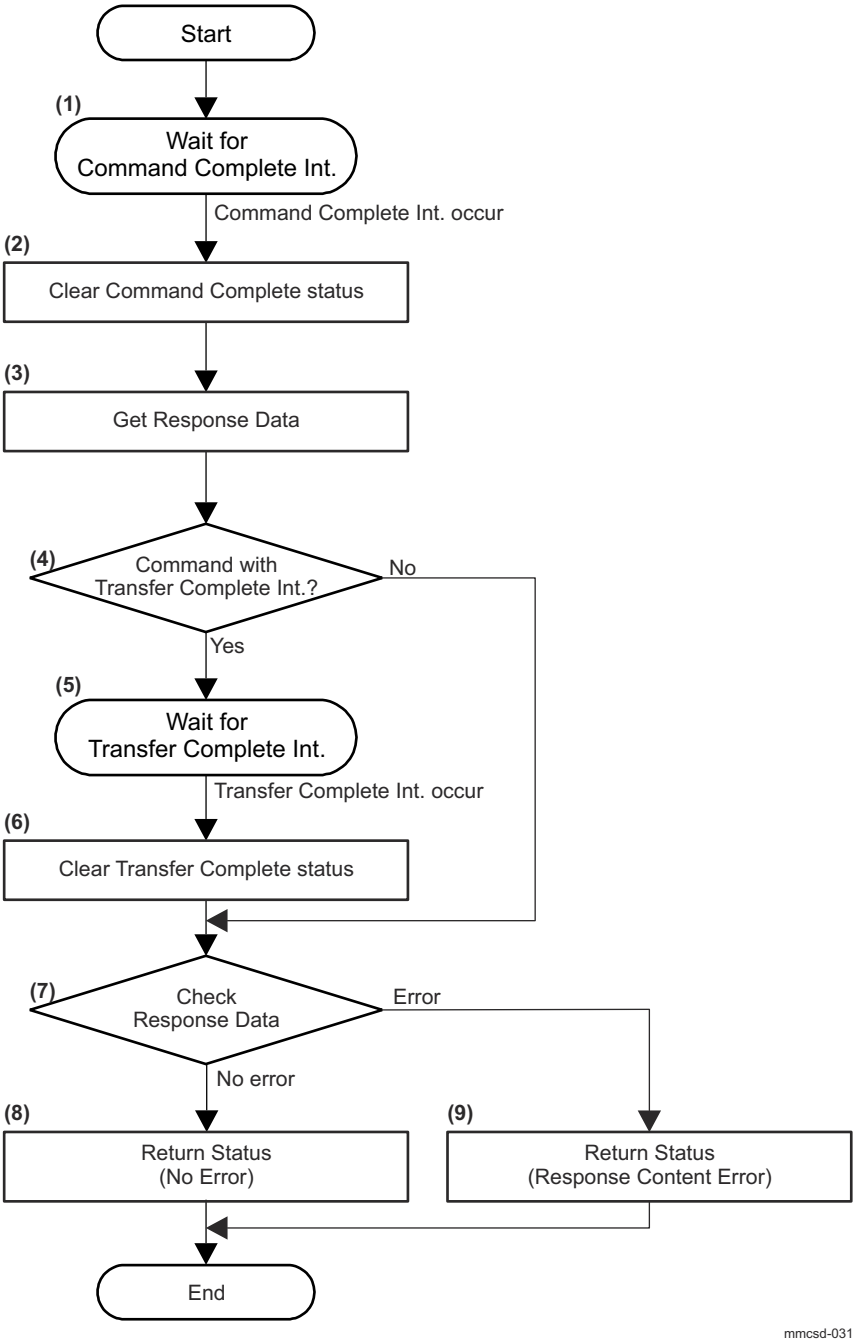


Figure 12-124. Command Complete Sequence

Copyright © 2024 Texas Instruments Incorporated

(1) If MMCSD0_TRANSFER_MODE[8] RESP_INTR_DIS bit is set to 1 (response check is enabled), go to stop (4) else wait for the Command Complete Interrupt (MMCSD0_NORMAL_INTR_STS[0] CMD_COMPLETE). If the Command Complete Interrupt has occurred, go to step (2).

(2) Write 1 to MMCSD0_NORMAL_INTR_STS[0] CMD_COMPLETE bit to clear this bit.

(3) Read the Response register (see MMCSD0_RESPONSE_0 - MMCSD0_RESPONSE_7) and get necessary information of the issued command.

(4) Judge whether the command uses the Transfer Complete Interrupt (MMCSD0_NORMAL_INTR_STS_ENA[1] XFER_COMPLETE) or not. If it uses Transfer Complete, go to step (5). If not, go to step (7).

(5) Wait for the Transfer Complete Interrupt. If the Transfer Complete Interrupt has occurred, go to step (6).

(6) Write 1 to MMCSD0_NORMAL_INTR_STS_ENA[1] XFER_COMPLETE bit to clear this bit.

(7) Check for errors in Response Data (MMCSD0_RESPONSE_0 - MMCSD0_RESPONSE_7). If there is no error, go to step (8). If there is an error, go to step (9).

(8) Return Status of "No Error".

(9) Return Status of "Response Contents Error".

Note1: While waiting for the Transfer Complete interrupt, the Host Driver shall only issue commands that do not use the busy signal.

Note2: The Host Driver shall judge the Auto CMD12 complete by monitoring Transfer Complete.

Note3: When the last block of un-protected area is read using memory multiple block read command (CMD18), OUT_OF_RANGE error may occur even if the sequence is correct. The Host Driver should ignore it. This error will appear in the response of Auto CMD12 or in the response of the next memory command.

12.3.6.4.1.7.2 Transaction Control with Data Transfer Using DAT Line

Depending on whether DMA (optional) is used or not, there are two execution methods.

In addition, the sequences for SD transfers are classified into following three kinds according to how the number of blocks is specified:

(1) Single Block Transfer: The number of blocks is specified to the Host Controller before the transfer. The number of blocks specified is always one.

(2) Multiple Block Transfer: The number of blocks is specified to the Host Controller before the transfer. The number of blocks specified shall be one or more.

(3) Infinite Block Transfer: The number of blocks is not specified to the Host Controller before the transfer. This transfer is continued until an abort transaction is executed. This abort transaction is performed by CMD12 in the case of a SD memory card and by CMD52 in the case of a SDIO card.

12.3.6.4.1.7.2.1 Not using DMA

The sequence for not using DMA is shown in Figure 12-125.

Copyright © 2024 Texas Instruments Incorporated

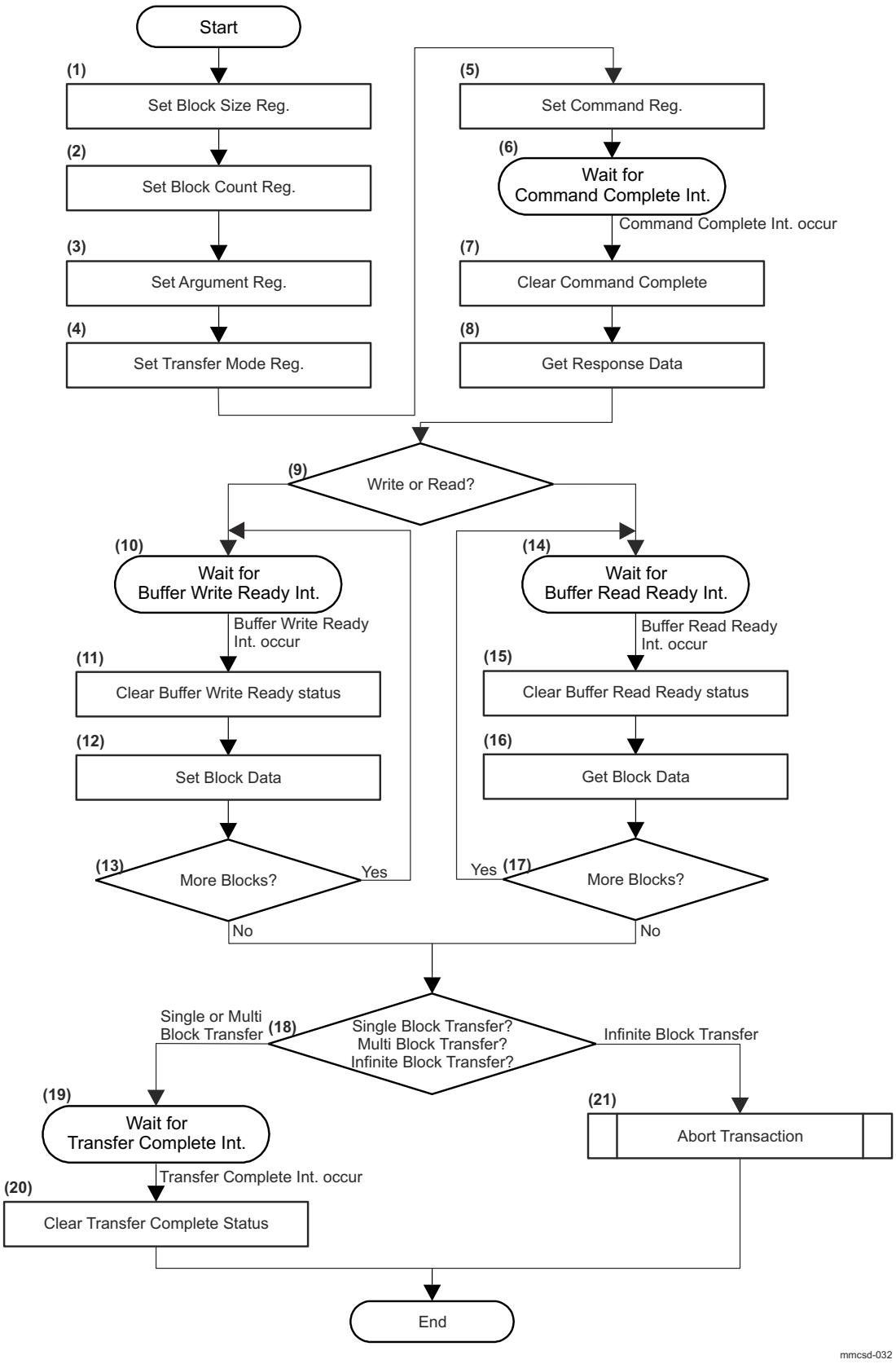


Figure 12-125. Transaction Control with Data Transfer Using DAT Line Sequence (Not using DMA)

Copyright © 2024 Texas Instruments Incorporated

(1) Set the value corresponding to the executed data byte length of one block to the MMCSD0_BLOCK_SIZE register.

(2) Set the value corresponding to the executed data block count to the MMCSD0_BLOCK_COUNT register in accordance with *Determination of Transfer Type*.

(3) Set the argument value to Argument register (MMCSD0_ARGUMENT1_LO and MMCSD0_ARGUMENT1_HI).

(4) Set the value to the MMCSD0_TRANSFER_MODE register. The Host Driver determines Multi/Single Block Select, Block Count Enable, Data Transfer Direction, Auto CMD12 Enable and DMA Enable in the MMCSD0_TRANSFER_MODE register. Multi/Single Block Select and Block Count Enable are determined according to *Determination of Transfer Type*.

If response check is enabled (MMCSD0_TRANSFER_MODE[7] RESP_ERR_CHK_ENA = 1), set MMCSD0_TRANSFER_MODE[8] RESP_INTR_DIS bit to 1 and select Response Type R1/R5 (MMCSD0_TRANSFER_MODE[6] RESP_TYPE).

(5) Set the value to MMCSD0_COMMAND register.

Note: When writing the upper byte [3] of MMCSD0_COMMAND register, SD command is issued.

(6) If response check is enabled, go to stop (9) else wait for the Command Complete Interrupt (MMCSD0_NORMAL_INTR_STS[0] CMD_COMPLETE bit).

(7) Write 1 to the MMCSD0_NORMAL_INTR_STS[0] CMD_COMPLETE bit for clearing this bit.

(8) Read Response register (MMCSD0_RESPONSE_0 - MMCSD0_RESPONSE_7) and get necessary information of the issued command.

(9) In the case where this sequence is for write to a card, go to step (10). In case of read from a card, go to step (14).

(10) Then wait for Buffer Write Ready Interrupt (MMCSD0_NORMAL_INTR_STS_ENA[4] BUF_WR_READY).

(11) Write 1 to the MMCSD0_NORMAL_INTR_STS_ENA[4] BUF_WR_READY bit for clearing this bit.

(12) Write block data (in according to the number of bytes specified at the step (1)) to MMCSD0_DATA_PORT register.

(13) Repeat until all blocks are sent and then go to step (18).

(14) Then wait for the Buffer Read Ready Interrupt (MMCSD0_NORMAL_INTR_STS_ENA[5] BUF_RD_READY).

(15) Write 1 to the MMCSD0_NORMAL_INTR_STS_ENA[5] BUF_RD_READY bit for clearing this bit.

(16) Read block data (in according to the number of bytes specified at the step (1)) from the MMCSD0_DATA_PORT register.

(17) Repeat until all blocks are received and then go to step (18).

(18) If this sequence is for Single or Multiple Block Transfer, go to step (19). In case of Infinite Block Transfer, go to step (21).

(19) Wait for Transfer Complete Interrupt (MMCSD0_NORMAL_INTR_STS_ENA[1] XFER_COMPLETE).

(20) Write 1 to the MMCSD0_NORMAL_INTR_STS_ENA[1] XFER_COMPLETE bit for clearing this bit.

(21) Perform the sequence for Abort Transaction in accordance with Section 12.3.6.4.1.8, *Abort Transaction*.

Note: Step (1) and Step (2) can be executed at same time. Step (4) and Step (5) can be executed at same time.

12.3.6.4.1.7.2.2 Using SDMA

The sequence for using SDMA is shown in Figure 12-126.

Copyright © 2024 Texas Instruments Incorporated

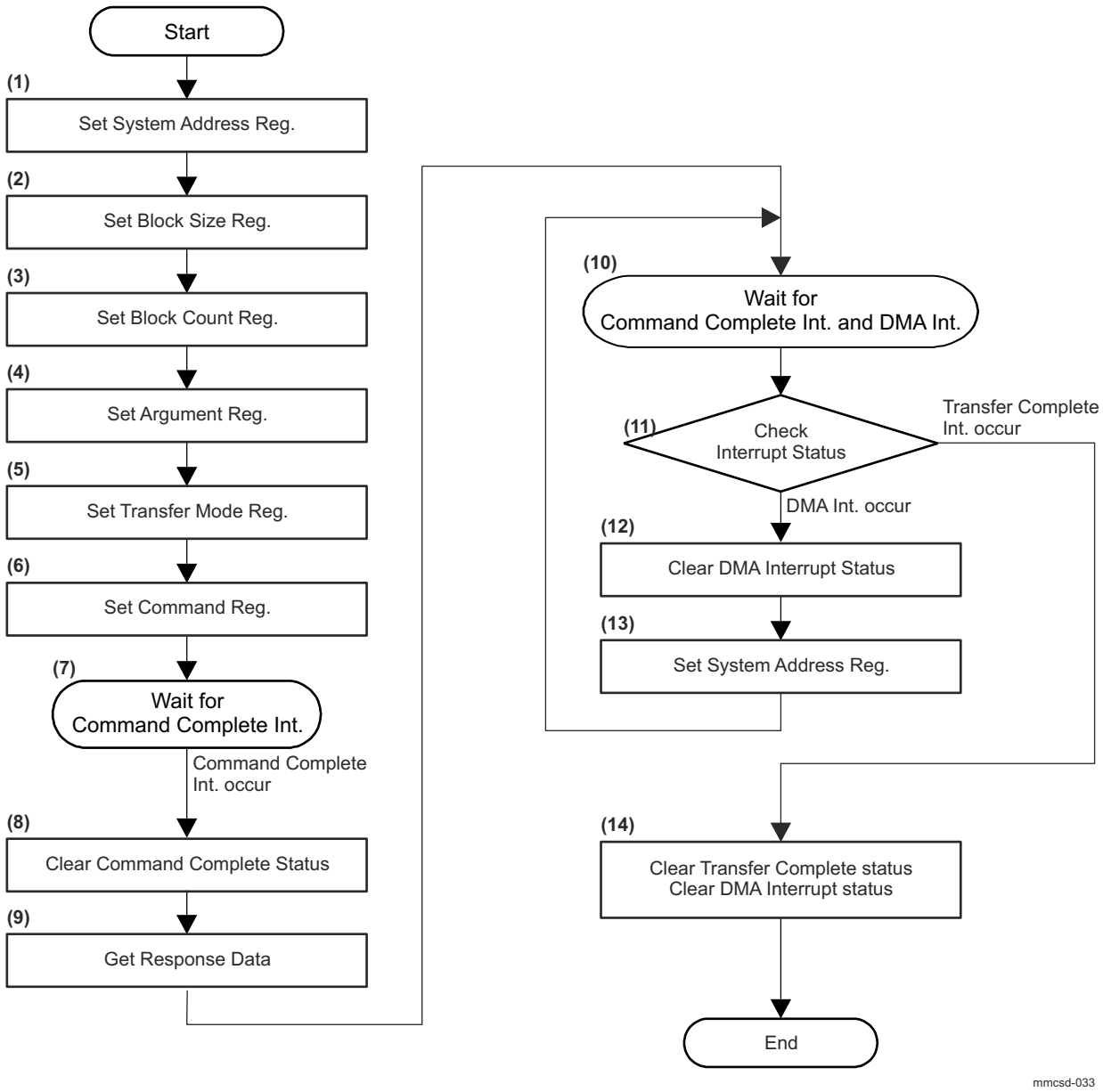


Figure 12-126. Transaction Control with Data Transfer Using DAT Line Sequence (Using SDMA)

(1) Data location of system memory is set to the SDMA System Address register if MMCSD0_HOST_CONTROL2[12] HOST_VER40_ENA = 0 or set to the MMCSD0_ADMA_SYS_ADDRESS register if MMCSD0_HOST_CONTROL2[12] HOST_VER40_ENA = 1.

(2) Set the value corresponding to the executed data byte length of one block in the MMCSD0_BLOCK_SIZE register.

(3) Set the value corresponding to the executed data block count in the MMCSD0_BLOCK_COUNT register in accordance with *Determination of Transfer Type*.

(4) Set the argument value to the Argument register (MMCSD0_ARGUMENT1_LO and MMCSD0_ARGUMENT1_HI).

(5) Set the value to the MMCSD0_TRANSFER_MODE register. The Host Driver determines Multi/Single Block Select, Block Count Enable, Data Transfer Direction, Auto CMD12 Enable and DMA Enable in the MMCSD0_TRANSFER_MODE register. Multi/Single Block Select and Block Count Enable are determined according to *Determination of Transfer Type*. If response check is enabled (MMCSD0_TRANSFER_MODE[7]

Copyright © 2024 Texas Instruments Incorporated

RESP_ERR_CHK_ENA = 1), set MMCSD0_TRANSFER_MODE[8] RESP_INTR_DIS bit to 1 and select Response Type R1/R5.

(6) Set the value to the MMCSD0_COMMAND register.

Note: When writing to the upper byte [3] of the MMCSD0_COMMAND register, the SD command is issued and SDMA is started.

(7) If response check is enabled, go to stop (10) else wait for the Command Complete Interrupt (MMCSD0_NORMAL_INTR_STS[0] CMD_COMPLETE bit).

(8) Write 1 to the MMCSD0_NORMAL_INTR_STS[0] CMD_COMPLETE bit to clear this bit.

(9) Read Response register (MMCSD0_RESPONSE_0 - MMCSD0_RESPONSE_7) and get necessary information of the issued command.

(10) Wait for the Transfer Complete Interrupt (MMCSD0_NORMAL_INTR_STS_ENA[1] XFER_COMPLETE) and DMA Interrupt (MMCSD0_NORMAL_INTR_STS_ENA[3] DMA_INTERRUPT).

(11) If MMCSD0_NORMAL_INTR_STS_ENA[1] XFER_COMPLETE bit is set to 1, go to Step (14) else if MMCSD0_NORMAL_INTR_STS_ENA[3] DMA_INTERRUPT bit is set to 1, go to Step (12). The MMCSD0_NORMAL_INTR_STS_ENA[1] XFER_COMPLETE bit is higher priority than the MMCSD0_NORMAL_INTR_STS_ENA[3] DMA_INTERRUPT bit.

(12) Write 1 to the MMCSD0_NORMAL_INTR_STS_ENA[3] DMA_INTERRUPT bit to clear this bit.

(13) Set the next system address of the next data position to the System Address register (MMCSD0_ADMA_SYS_ADDRESS) and go to Step (10).

(14) Write 1 to the MMCSD0_NORMAL_INTR_STS_ENA[1] XFER_COMPLETE bit and MMCSD0_NORMAL_INTR_STS_ENA[3] DMA_INTERRUPT bit to clear this bit.

Note: Step (2) and Step (3) can be executed simultaneously. Step (5) and Step (6) can also be executed simultaneously.

12.3.6.4.1.7.2.3 Using ADMA

The sequence for using ADMA is shown in Figure 12-127.

Copyright © 2024 Texas Instruments Incorporated

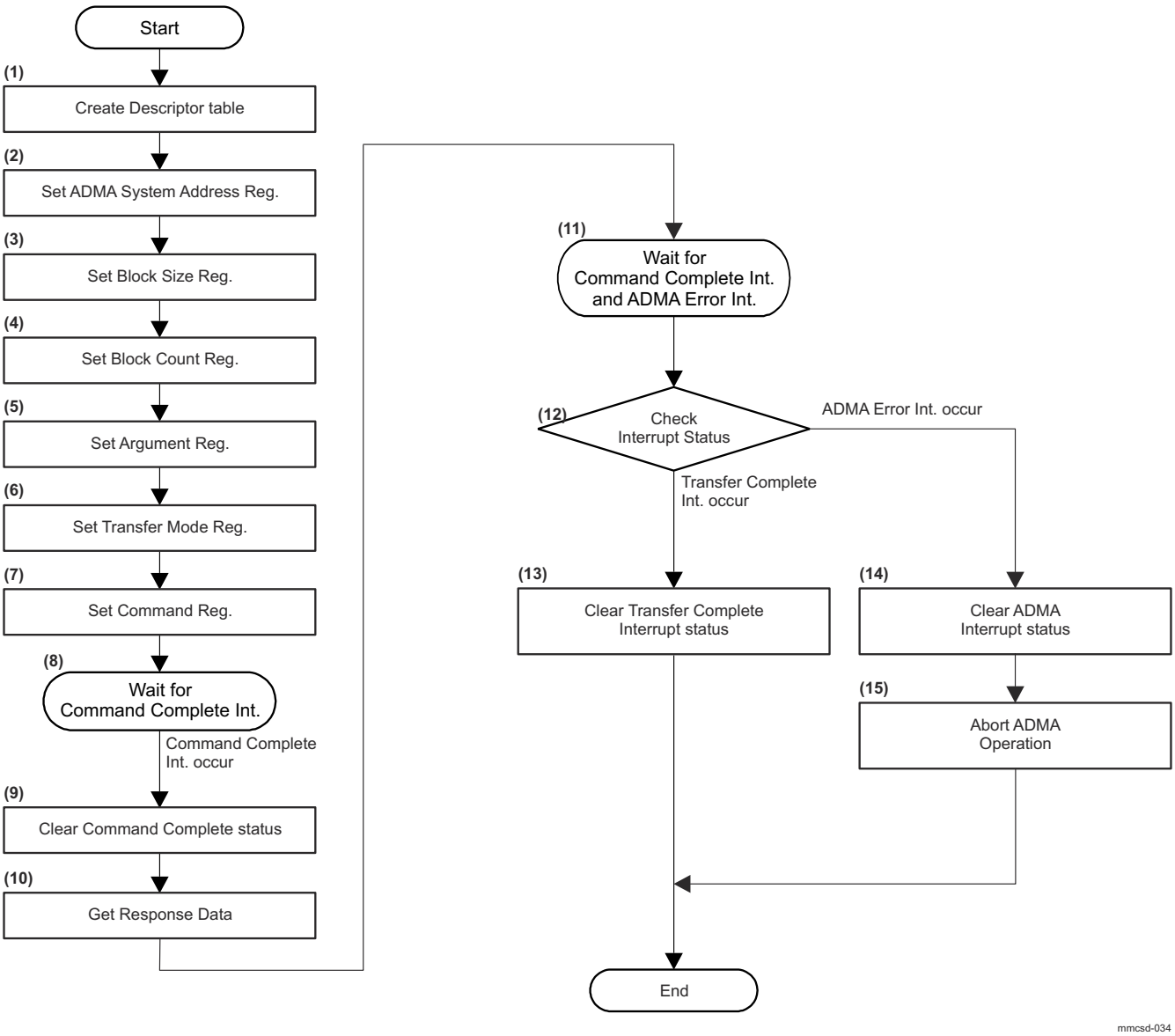


Figure 12-127. Transaction Control with Data Transfer Using DAT Line Sequence (Using ADMA)

(1) Create Descriptor table for ADMA in the system memory.

(2) Set the Descriptor address for ADMA in the MMCSD0_ADMA_SYS_ADDRESS register.

(3) Set the value corresponding to the executed data byte length of one block in the MMCSD0_BLOCK_SIZE register.

(4) Set the value corresponding to the executed data block count in the MMCSD0_BLOCK_COUNT register in accordance with *Determination of Transfer Type*.

(5) Set the argument value to the Argument register (MMCSD0_ARGUMENT1_LO and MMCSD0_ARGUMENT1_HI).

(6) Set the value to the MMCSD0_TRANSFER_MODE register. The Host Driver determines Multi/Single Block Select, Block Count Enable, Data Transfer Direction, Auto CMD12 Enable and DMA Enable in the MMCSD0_TRANSFER_MODE register. Multi/Single Block Select and Block Count Enable are determined according to *Determination of Transfer Type*. If response check is enabled (MMCSD0_TRANSFER_MODE[7] RESP_ERR_CHK_ENA = 1), set MMCSD0_TRANSFER_MODE[8] RESP_INTR_DIS bit to 1 and select Response Type R1/R5 (MMCSD0_TRANSFER_MODE[6] RESP_TYPE).

Copyright © 2024 Texas Instruments Incorporated

(7) Set the value to the MMCSD0_COMMAND register.

Note: When writing to the upper byte [3] of the MMCSD0_COMMAND register, the SD command is issued and DMA is started.

(8) If response check is enabled, go to stop (11) else wait for the Command Complete Interrupt (MMCSD0_NORMAL_INTR_STS[0] CMD_COMPLETE bit).

(9) Write 1 to the MMCSD0_NORMAL_INTR_STS[0] CMD_COMPLETE bit to clear this bit.

(10) Read Response register (MMCSD0_RESPONSE_0 - MMCSD0_RESPONSE_7) and get necessary information of the issued command.

(11) Wait for the Transfer Complete Interrupt (MMCSD0_NORMAL_INTR_STS_ENA[1] XFER_COMPLETE) and ADMA Error Interrupt (MMCSD0_ADMA_ERR_STATUS[1-0] ADMA_ERR_STATE).

(12) If MMCSD0_NORMAL_INTR_STS_ENA[1] XFER_COMPLETE bit is set to 1, go to Step (13) else if MMCSD0_ADMA_ERR_STATUS[1-0] ADMA_ERR_STATE bit field is set to 1, go to Step (14).

(13) Write 1 to the MMCSD0_NORMAL_INTR_STS_ENA[1] XFER_COMPLETE bit to clear this bit.

(14) Write 1 to the MMCSD0_ADMA_ERR_STATUS[1-0] ADMA_ERR_STATE bit field to clear this bit.

(15) Abort ADMA operation. SD card operation should be stopped by issuing abort command. If necessary, the Host Driver checks MMCSD0_ADMA_ERR_STATUS register to detect why ADMA error is generated.

Note: Step (3) and Step (4) can be executed simultaneously. Step (6) and Step (7) can also be executed simultaneously.

12.3.6.4.1.8 Abort Transaction

An abort transaction is performed by issuing CMD12 for a SD memory card and by issuing CMD52 for a SDIO card. There are two cases where the Host Driver needs to do an Abort Transaction. The first case is when the Host Driver stops Infinite Block Transfers. The second case is when the Host Driver stops transfers while a Multiple Block Transfer is executing.

There are two ways to issue an Abort Command. The first is an asynchronous abort. The second is a synchronous abort. In an asynchronous abort sequence, the Host Driver can issue an Abort Command at any time unless MMCSD0_PRESENTSTATE[0] INHIBIT_CMD bit is set to 1. In a synchronous abort, the Host Driver shall issue an Abort Command after the data transfer stopped by using MMCSD0_BLOCK_GAP_CONTROL[0] STOP_AT_BLK_GAP bit.

12.3.6.4.1.8.1 Asynchronous Abort

The sequence for Asynchronous Abort is shown in Figure 12-128.

Copyright © 2024 Texas Instruments Incorporated

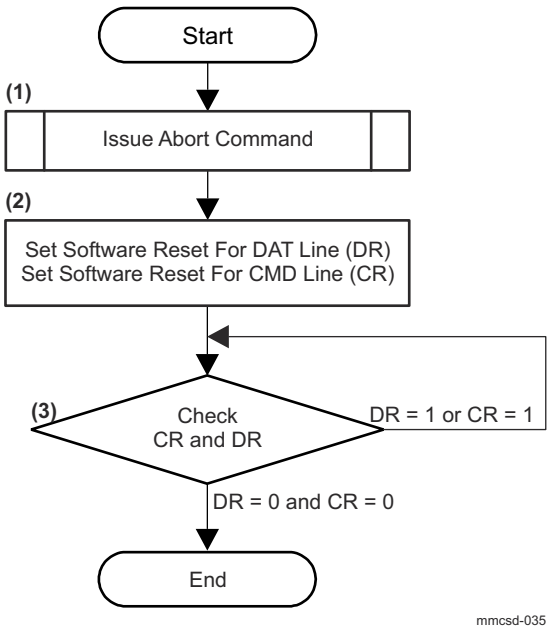


Figure 12-128. Asynchronous Abort Sequence

(1) Issue Abort Command in accordance.

(2) Set both MMCSD0_SOFTWARE_RESET[2] SWRST_FOR_DAT bit and MMCSD0_SOFTWARE_RESET[1] SWRST_FOR_CMD bit to 1 to do software reset.

(3) Check MMCSD0_SOFTWARE_RESET[2] SWRST_FOR_DAT bit and MMCSD0_SOFTWARE_RESET[1] SWRST_FOR_CMD bit. If both MMCSD0_SOFTWARE_RESET[2] SWRST_FOR_DAT bit and MMCSD0_SOFTWARE_RESET[1] SWRST_FOR_CMD bit are 0, go to "End". If either MMCSD0_SOFTWARE_RESET[2] SWRST_FOR_DAT bit or MMCSD0_SOFTWARE_RESET[1] SWRST_FOR_CMD bit is 1, go to step (3).

12.3.6.4.1.8.2 Synchronous Abort

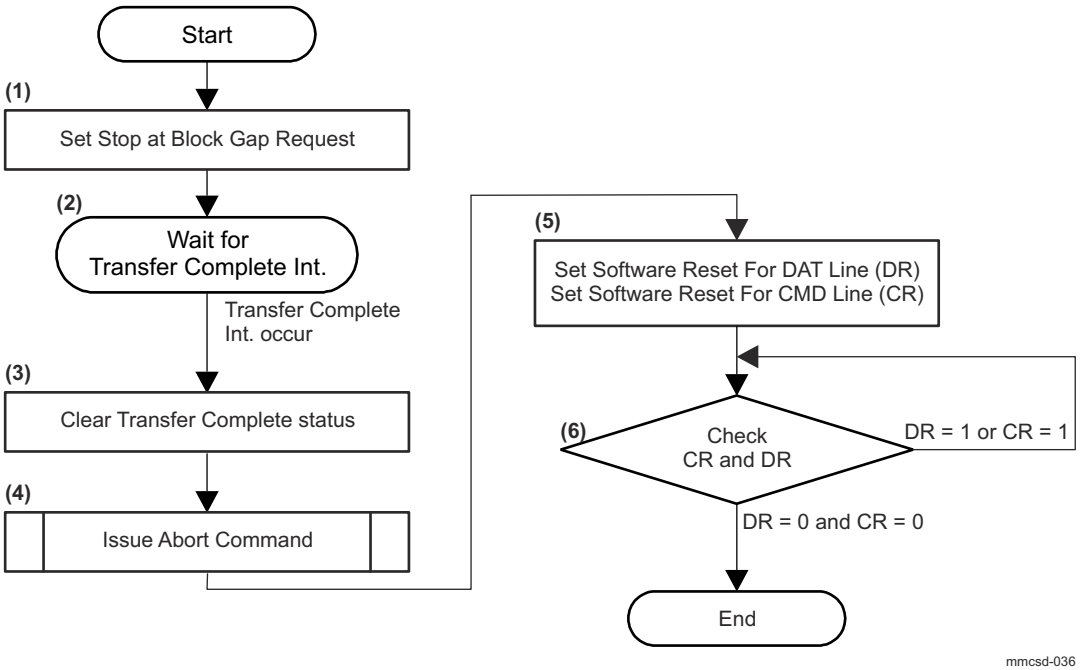


Figure 12-129. Synchronous Abort Sequence

Copyright © 2024 Texas Instruments Incorporated

(1) Set the MMCSD0_BLOCK_GAP_CONTROL[0] STOP_AT_BLK_GAP bit to 1 to stop SD transactions.

(2) Wait for the Transfer Complete Interrupt (MMCSD0_NORMAL_INTR_STS_ENA[1] XFER_COMPLETE).

(3) Set the MMCSD0_NORMAL_INTR_STS_ENA[1] XFER_COMPLETE bit to 1 to clear this bit.

(4) Issue the Abort Command

(5) Set both MMCSD0_SOFTWARE_RESET[2] SWRST_FOR_DAT bit and MMCSD0_SOFTWARE_RESET[1] SWRST_FOR_CMD bit to 1 to do software reset.

(6) Check both MMCSD0_SOFTWARE_RESET[2] SWRST_FOR_DAT bit and MMCSD0_SOFTWARE_RESET[1] SWRST_FOR_CMD bit. If both MMCSD0_SOFTWARE_RESET[2] SWRST_FOR_DAT bit and MMCSD0_SOFTWARE_RESET[1] SWRST_FOR_CMD bit are 0, go to "End". If either MMCSD0_SOFTWARE_RESET[2] SWRST_FOR_DAT bit or MMCSD0_SOFTWARE_RESET[1] SWRST_FOR_CMD bit is 1, go to step (6).

12.3.6.4.1.9 Changing Bus Speed Mode

This section describes the sequence for switching the bus speed mode: Default Speed, High Speed mode and UHS-I mode. The switch command (CMD6) is used to change memory card bus speed mode. The EHS bit (SDIO Version 2.00) or BSS[2-0] bits (SDIO Version 3.00) in the CCCR register is used to change the bus speed mode for SDIO card. In case of Combo card, either of the switch method changes both memory and IO bus speed mode. This means the first switch is effective. Refer to the Physical Layer Specification Version 3.0x and the SDIO Specification Version 3.00 for more information about switching bus speed. Figure 12-130 shows the sequence for switching bus speed mode for Combo Card. Note that if the card is locked, bus width cannot be changed. Unlock the card is required before changing bus width.

Copyright © 2024 Texas Instruments Incorporated

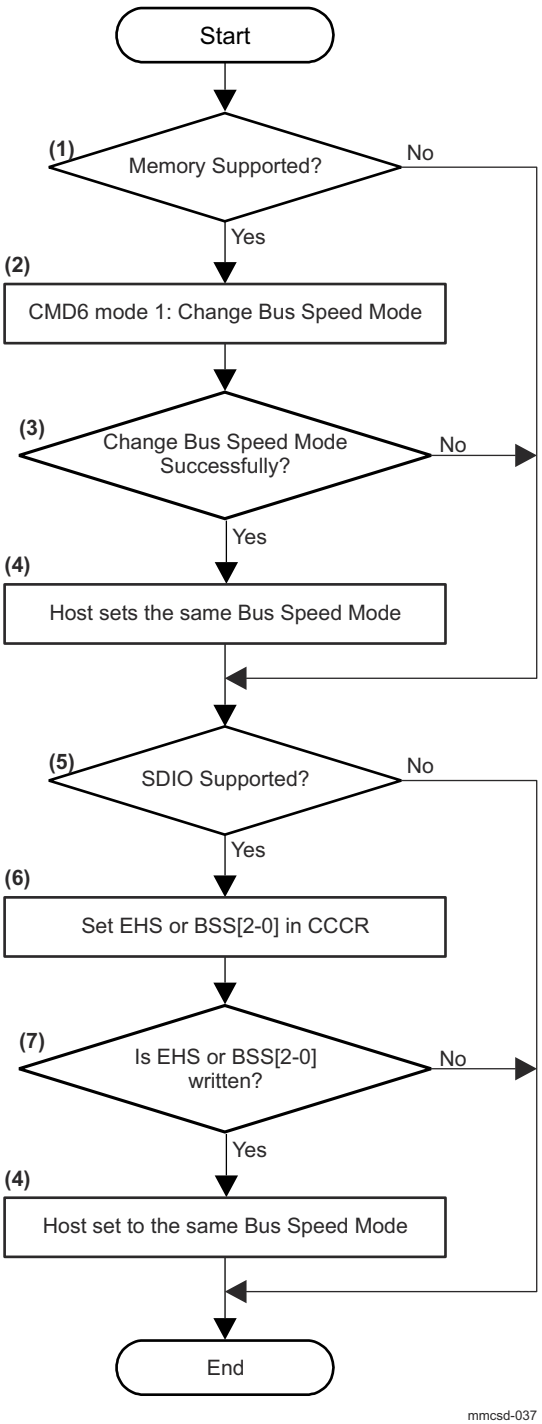


Figure 12-130. Changing Bus Speed Mode

(1) The Host Driver checks if the card supports memory. If not supported, go to (5).

(2) Issue CMD6 with mode 1 to change bus speed mode (Default, High Speed mode or UHS-I mode).

(3) Check the response of CMD6. If the card does not supports CMD6 (no response) or bus speed is not changed successfully, go to step (5). In this case, the card is in Default Speed mode.

(4) The Host Driver changes the Host Controller bus speed mode to the same mode.

(5) The Host Driver checks if the card supports SDIO. If not supported, go to the end.

Copyright © 2024 Texas Instruments Incorporated

(6) Issue CMD52 to write EHS bit or BSS[2-0] bits in CCCR to change bus speed mode (the same bus speed mode of (2) shall be set).

(7) If EHS or BSS[2-0] are not changed successfully, go to the end.

(8) The Host Driver changes the Host Controller bus speed to the same mode. In case of Combo card, bus speed is already changed at step (4) and this step does not affect changing bus speed.

12.3.6.4.1.10 Error Recovery

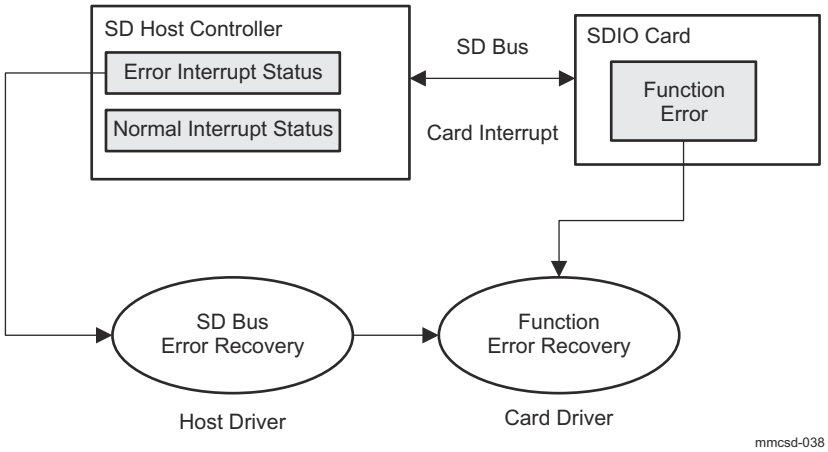


Figure 12-131. Error Report and Recovery

Figure 12-131 shows concept of error report and its recovery. The Host Controller has 2 interrupt status registers. If an error occurs in the SD Bus transaction, one of the bits is set in the MMCSD0_ERROR_INTR_STS register. If the function errors occur in the SDIO card, the card interrupt informs these function errors and the Card interrupt is set in the MMCSD0_NORMAL_INTR_STS_ENA register (the Card Interrupt is used to inform not only error statuses but also normal information. For example, to inform function ready). The Card Driver shall do function error recovery because the Host Driver does not know how to control the function. In the case that function error occurs due to SD Bus error, SD Bus error recovery is required before function error recovery. Abort command is used to recover SD Bus, and then the Host Driver should save error statuses related to SD Bus errors before issuing abort command and transfer these statuses to the Card Driver. These statuses may be used to recover function error. Following explanations are related to SD Bus error recovery. This specification does not specify the function error recovery.

When an error occurs during data transfer in 2L-HD UHS-II mode, there will be the case that Host Controller cannot drive D0 lane in input mode due to DIR LSS for retrieving lane direction is not detected. In this case, Host Driver cannot issue abort command for recovery. Then if DIR LSS is not detected, Host Controller sets MMCSD0_UHS2_ERR_INTR_STS[17] DEADLOCK_TIMEOUT bit. Host Driver should execute power cycle if MMCSD0_UHS2_ERR_INTR_STS[17] DEADLOCK_TIMEOUT bit is detected to recover from this error. Furthermore, if this type of error is detected several times in 2L-HD, Host Driver should use FD mode rather than using 2L-HD.

Implementation Note:

If the Card Driver cannot recover the function errors, the Host Driver should try following methods.

(9) Using IOEx for SDIO card

IOEx may be used as the reset per function basis. Sequence is as follows:

Clear IOEx = 0 and wait until IORx = 0 and then set IOEx = 1 again. SDIO may be recovered when IORx = 1.

(10) Using reset command for memory and SDIO card Re-initialization sequence is required.

(11) Off and on power supply for the SD Bus.

The card may be recovered by the power on reset. Re-initialization sequence is required.

Copyright © 2024 Texas Instruments Incorporated

The two cases where the Host Driver needs the "Error Recovery" sequence are classified as follows:

(1) Error Interrupt Recovery:

If error interrupt is indicated by the MMCSD0_ERROR_INTR_STS register, the Host Driver shall apply this sequence.

(2) Auto CMD12 Error Recovery:

If there are errors in Auto CMD12, the Host Driver shall apply this sequence. In terms of Return Status, Auto CMD12 Error Recovery is classified into 4 cases. It is shown in Figure 12-132. If error occurs during memory write transfer, strongly recommend using ACMD22 and then in the following recovery sequence, retry to send remaining blocks not written.

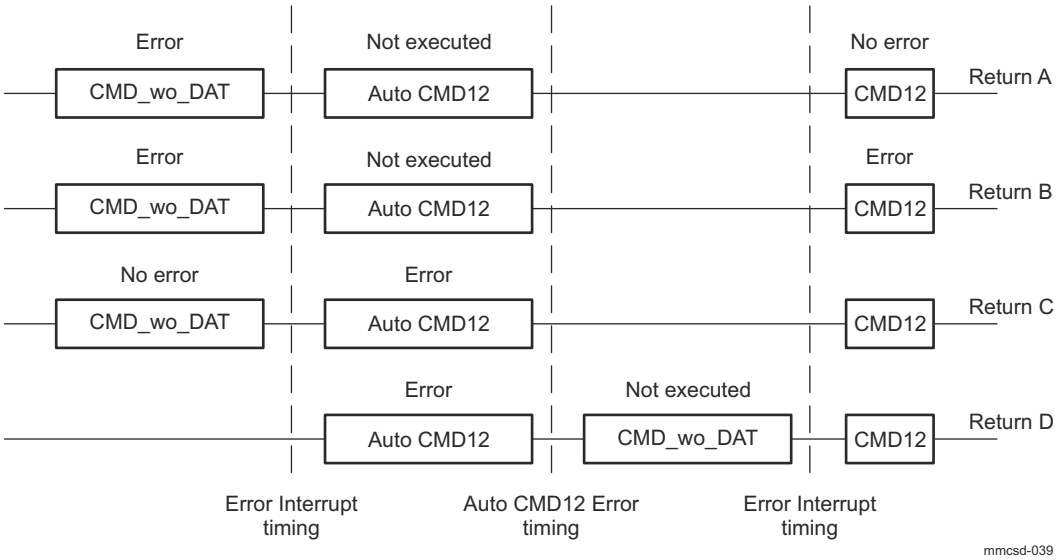


Figure 12-132. Return Status of Auto CMD12 Error Recovery

Implementation Note:

Abort command is used to recover from SD Bus error. SDIO transaction abort using CMD52 returns response but in the case of memory transaction abort using CMD12, response returns depending on the memory card state. If no response returns after issue CMD12, the Host Driver should check card state using CMD13. If the state is "tran" in the CURRENT_STATE, consider CMD12 is successful.

Implementation Note:

The following sequence is one possible error recovery flow. There may be another methods, sometimes using interrupts or polling. It can be possible to use another flows, based on Host System requirements.

In these error recovery sequences, return statuses for the next sequence. When the Host Controller cannot issue the next command due to SD Bus error, the error recovery sequences return "Non-recoverable" status. In this case, the Host System may cut off power to the SD Bus and then power on SD Bus and initialize both the Host Controller and the SD card again.

Copyright © 2024 Texas Instruments Incorporated

12.3.6.4.1.10.1 Error Interrupt Recovery

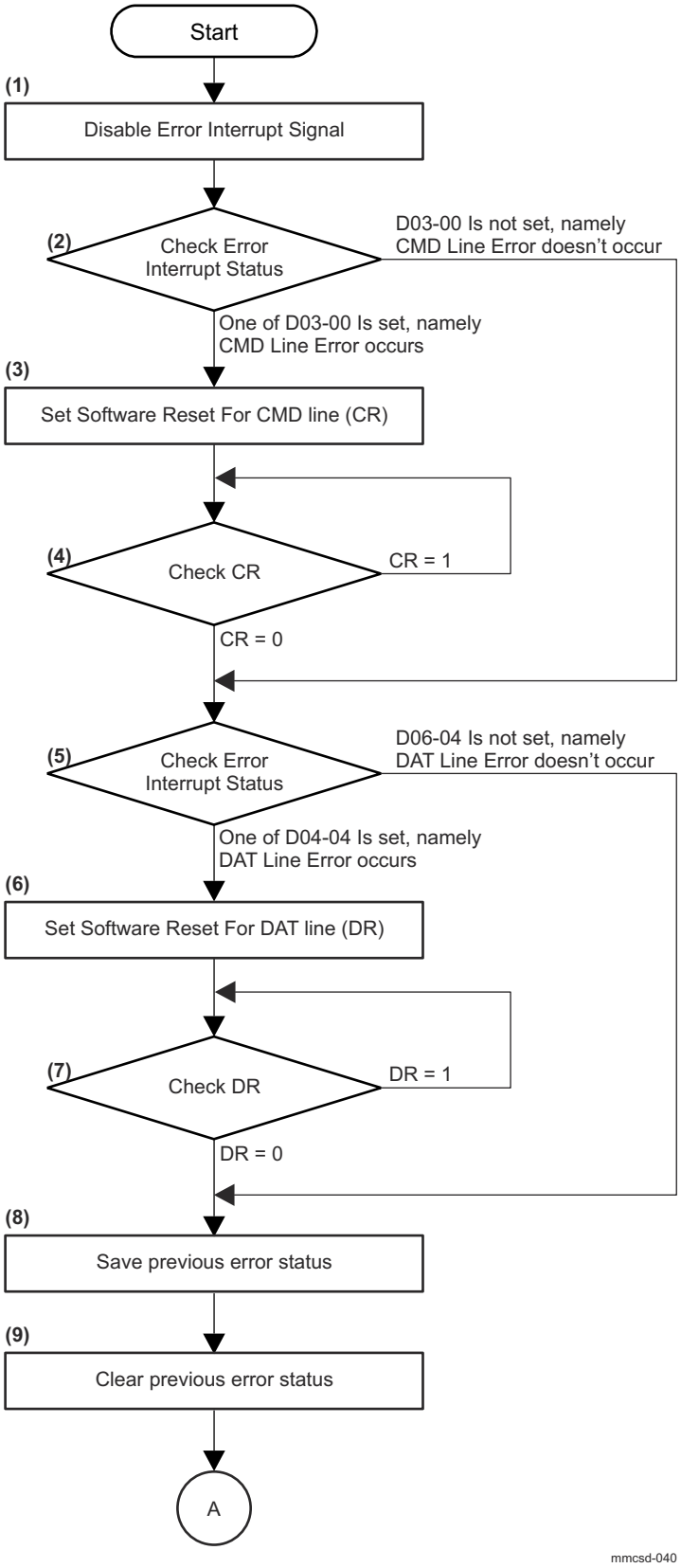


Figure 12-133. Error Interrupt Recovery Sequence (1)

Copyright © 2024 Texas Instruments Incorporated

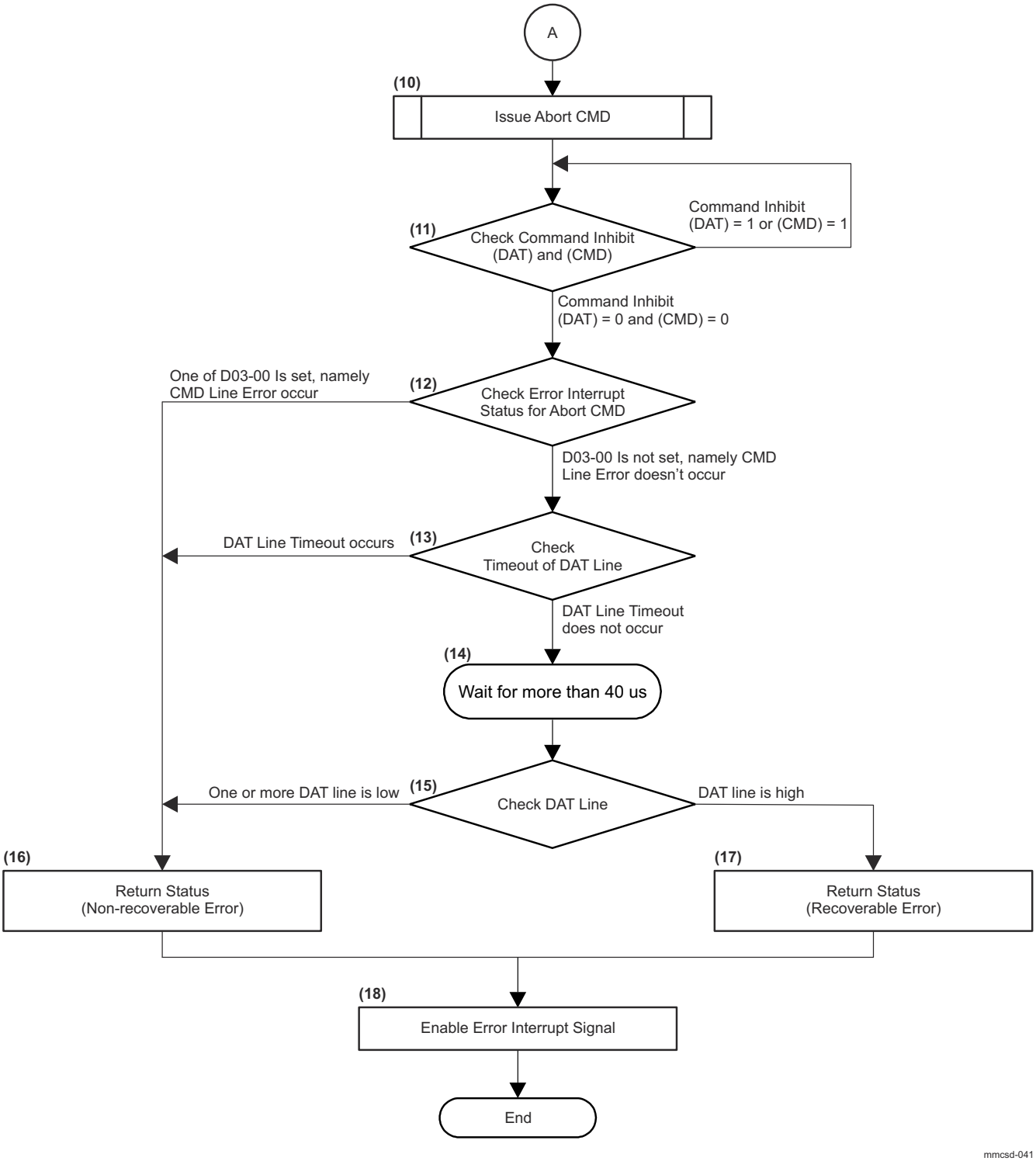


Figure 12-134. Error Interrupt Recovery Sequence (2)

(1) Disable the Error Interrupt Signal (MMCSD0_ERROR_INTR_SIG_ENA).

(2) Check bits D03-00 in the MMCSD0_ERROR_INTR_STS register. If one of these bits (D03-00) is set to 1, go to step (3). If none are set to 1 (all are 0), go to step (5).

(3) Set MMCSD0_SOFTWARE_RESET[1] SWRST_FOR_CMD bit to 1.

Copyright © 2024 Texas Instruments Incorporated

(4) Check MMCSD0_SOFTWARE_RESET[1] SWRST_FOR_CMD bit. If MMCSD0_SOFTWARE_RESET[1] SWRST_FOR_CMD bit is 0, go to step (5). If it is 1, go to step (4).

(5) Check bits D06-04 in the MMCSD0_ERROR_INTR_STS register. If one of these bits (D06-04) is set to 1, go to step (6). If none are set to 1 (all are 0), go to step (8).

(6) Set MMCSD0_SOFTWARE_RESET[2] SWRST_FOR_DAT bit to 1 for software reset of the DAT line.

(7) Check MMCSD0_SOFTWARE_RESET[2] SWRST_FOR_DAT bit. If MMCSD0_SOFTWARE_RESET[2] SWRST_FOR_DAT bit is 0, go to step (8). If it is 1, go to step (7).

(8) Save previous error status.

(9) Clear previous error status with setting them to 1.

(10) Issue Abort Command.

(11) MMCSD0_PRESENTSTATE[1] INHIBIT_DAT bit and MMCSD0_PRESENTSTATE[0] INHIBIT_CMD bit. Repeat this step until both MMCSD0_PRESENTSTATE[1] INHIBIT_DAT bit and MMCSD0_PRESENTSTATE[0] INHIBIT_CMD bit are set to 0.

(12) Check bits D03-00 in the MMCSD0_ERROR_INTR_STS register for Abort Command. If one of these bits is set to 1, go to step (16). If none of these bits are set to 1 (all are 0), go to step (13).

(13) Check MMCSD0_ERROR_INTR_STS[4] DATA_TIMEOUT bit. If this bit is set to 1, go to step (16). If it is 0, go to step (14).

(14) Wait for more than 40 us.

(15) By monitoring the DAT [3:0] Line Signal Level in the MMCSD0_PRESENTSTATE register, judge whether the level of the DAT line is low or not. If one or more DAT lines are low, go to step (16). If the DAT lines are high, go to step (17).

(16) Return Status of "Non-recoverable Error".

(17) Return Status of "Recoverable Error".

(18) Enable the Error Interrupt Signal (MMCSD0_ERROR_INTR_SIG_ENA).

Copyright © 2024 Texas Instruments Incorporated

12.3.6.4.1.10.2 Auto CMD12 Error Recovery

Figure 12-135. Auto CMD12 Error Recovery

The sequence for Auto CMD12 Error Recovery is shown in Figure 12-135. Following four cases A-D shall be covered.

A: An error occurred in CMD_wo_DAT, but not in the SD memory transfer.

B: An error occurred in CMD_wo_DAT, and also occurred in the SD memory transfer.

C: An error did not occur in CMD_wo_DAT, but an error occurred in the SD memory transfer.

D: CMD_wo_DAT was not issued, and an error occurred in the SD memory transfer.

Copyright © 2024 Texas Instruments Incorporated

(1) Check MMCSD0_AUTOCMD_ERR_STS[0] ACMD12_NOT_EXEC bit. If this bit is set to 1, go to step (2). If this bit is set to 0, go to step (6). In addition, the Host Driver shall define PCMD flag, which changes to 1 if MMCSD0_AUTOCMD_ERR_STS[0] ACMD12_NOT_EXEC bit is set to 1.

(2) Wait for Error Interrupt Recovery for CMD_wo_DAT.

(3) Check "Return Status". In the case of "Non-recoverable Error", go to step (16). In the case of "Recoverable Error", go to step (4).

(4) Issue CMD12 .

(5) If the CMD line errors occur for the CMD12 (one of D03-00 is set in the MMCSD0_ERROR_INTR_STS register), "Return Status" is "Non-recoverable Error" and go to step (16). If not CMD line error and busy timeout error occur (D04 is set in the MMCSD0_ERROR_INTR_STS register), "Return Status" is "Recoverable Error" and go to step (11). Otherwise, "Return Status" is "No error" and go to step (17).

(6) Set MMCSD0_SOFTWARE_RESET[1] SWRST_FOR_CMD bit to 1 for software reset of the CMD line.

(7) Check MMCSD0_SOFTWARE_RESET[1] SWRST_FOR_CMD bit. If MMCSD0_SOFTWARE_RESET[1] SWRST_FOR_CMD bit is 0, go to step (8). If it is 1, go to step (7).

(8) Issue CMD12 . Acceptance of CMD12 depends on the state of the card. CMD12 may make the card to return to tran state. If the card is already in tran state, the card does not response to CMD12.

(9) Check "Return Status" for CMD12. If "Return Status" returns "Non-recoverable Error", go to step (16). In the case of "Recoverable Error" or "No error", go to step (10).

(10) Check the MMCSD0_AUTOCMD_ERR_STS[0] ACMD12_NOT_EXEC bit. If this bit is 0, go to step (11). If it is 1, go to step (14).

(11) Set MMCSD0_SOFTWARE_RESET[2] SWRST_FOR_DAT bit to 1 for software reset of the DAT line.

(12) Check MMCSD0_SOFTWARE_RESET[2] SWRST_FOR_DAT bit. If MMCSD0_SOFTWARE_RESET[2] SWRST_FOR_DAT bit is 0, go to step (13). If it is 1, go to step (12).

(13) Check the PCMD flag. If PCMD is 1, go to step (18). If it is 0, go to step (19).

(14) Set MMCSD0_SOFTWARE_RESET[2] SWRST_FOR_DAT bit to 1 for software reset of the DAT line.

(15) Check MMCSD0_SOFTWARE_RESET[2] SWRST_FOR_DAT bit. If MMCSD0_SOFTWARE_RESET[2] SWRST_FOR_DAT bit is 0, go to step (20). If it is 1, go to step (15).

(16) Return Status of "Non-recoverable Error".

(17) Return Status that an error has occurred in CMD_wo_DAT, but not in the SD memory transfer.

(18) Return Status that an error has occurred in both CMD_wo_DAT, and the SD memory transfer.

(19) Return Status that an error has not occurred in CMD_wo_DAT, but has occurred in the SD memory transfer.

(20) Return Status that CMD_wo_DAT has not been issued, and an error has occurred in the SD memory transfer.

12.3.6.4.1.11 Wakeup Control (Optional)

After the Host System goes into standby mode, the Host System can resume from standby via a wakeup event initiated by one of the following three events:

(1) Interrupt from a SD card: If an SD card interrupt occurs, the Host System can resume from standby mode. If the Host System uses this wakeup factor, SD Bus power shall be kept on.

(2) Insertion of SD card: If a SD card is inserted, the Host System can resume from standby mode.

(3) Removal of SD card: If a SD card is removed, the Host System can resume from standby mode.

The sequence for preparing wakeup before the Host System goes into standby mode is shown in Figure 12-136.

Copyright © 2024 Texas Instruments Incorporated

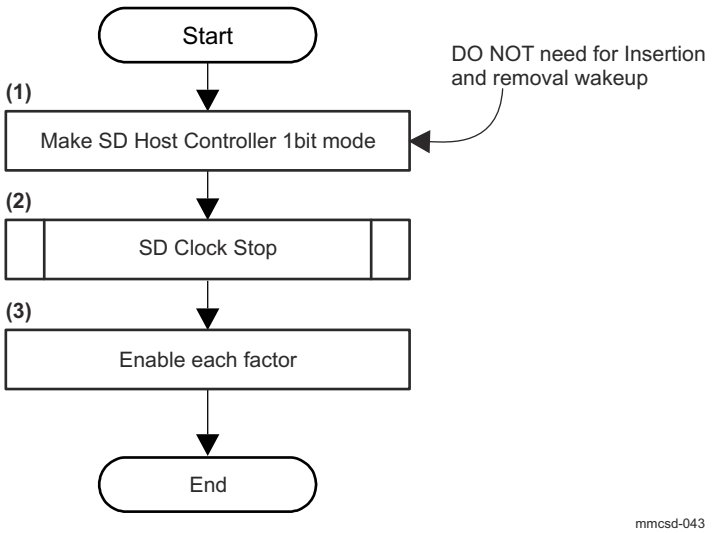


Figure 12-136. Wakeup Control before Standby Mode

(1) Set MMCSD0_HOST_CONTROL1[1] DATA_WIDTH bit to 0.

(2) Execute SD Clock Stop Sequence as described in Section 12.3.6.4.1.2.2, *SD Clock Supply and Stop Sequence*.

(3) Clear the MMCSD0_NORMAL_INTR_STS register and the MMCSD0_NORMAL_INTR_SIG_ENA register, and then set the enable bits of each wakeup event factor to 1 in the MMCSD0_WAKEUP_CONTROL register and set the bits of MMCSD0_ERROR_INTR_STS_ENA register to use wakeup.

The sequence for wakeup once in standby mode is shown in Figure 12-137.

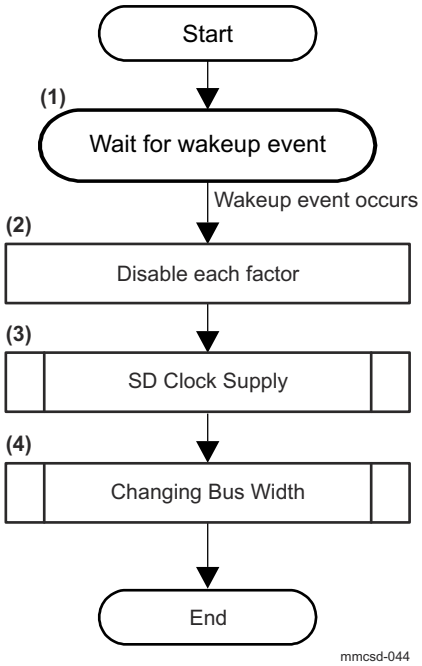


Figure 12-137. Wakeup from Standby

(1) Wait for wakeup event.

(2) Set the enable bits of each wakeup event factor to 0 in the MMCSD0_WAKEUP_CONTROL register and then clear event statuses in the MMCSD0_NORMAL_INTR_STS register. If necessary, set the MMCSD0_NORMAL_INTR_SIG_ENA register.

Copyright © 2024 Texas Instruments Incorporated

(3) Execute SD Clock Supply Sequence (see Section 12.3.6.4.1.2.2, *SD Clock Supply and Stop Sequence*).

(4) Set the SD Bus width (see Section 12.3.6.4.1.4, *Changing Bus Width*).

12.3.6.4.1.12 Suspend/Resume (Optional, Not Supported from Version 4.00)

If a SD card supports suspend and resume functionality, then the Host Controller can initiate suspend and resume. It is necessary for both the Host Controller and the SD card to support the function of "Read Wait". ADMA operation does not support this function.

12.3.6.4.1.12.1 Suspend Sequence

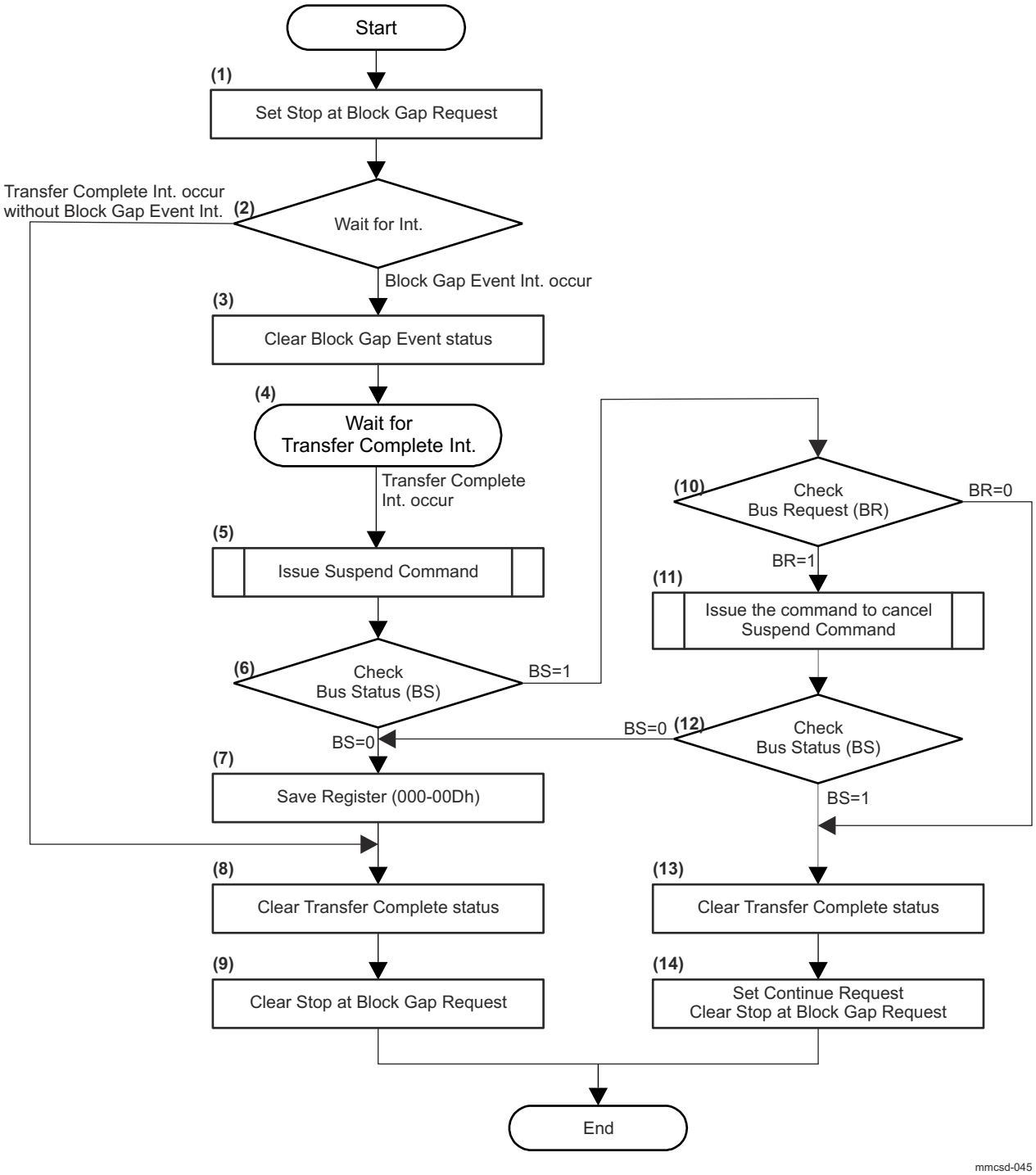


Figure 12-138. The Sequence for Suspend

Copyright © 2024 Texas Instruments Incorporated

(1) Set MMCSD0_BLOCK_GAP_CONTROL[0] STOP_AT_BLK_GAP bit to 1 to stop the SD transaction.

(2) Wait for an Interrupt. If MMCSD0_NORMAL_INTR_STS[2] BLK_GAP_EVENT bit is set to 0 and MMCSD0_NORMAL_INTR_STS[1] XFER_COMPLETE bit is set to 1, go to step (8). If MMCSD0_NORMAL_INTR_STS[2] BLK_GAP_EVENT bit is set to 1, go to step (3).

(3) Set MMCSD0_NORMAL_INTR_STS[2] BLK_GAP_EVENT bit to 1 to clear this bit.

(4) Wait for the Transfer Complete Interrupt (MMCSD0_NORMAL_INTR_STS_ENA[1] XFER_COMPLETE).

(5) Issue the Suspend Command in accordance with Section 12.3.6.4.1.7.1, *Transaction Control without Data Transfer Using DAT Line*.

(6) Check the BS value of the response data. If BS is 0, go to step (7). If BS is 1, go to step (10).

(7) Save the register .

(8) Set MMCSD0_NORMAL_INTR_STS[1] XFER_COMPLETE bit to 1 to clear this bit.

(9) Set MMCSD0_BLOCK_GAP_CONTROL[0] STOP_AT_BLK_GAP bit to 0 to clear this bit.

(10) Check the BR value of the response data. If BR is 1, go to step (11). If BR is 0, go to step (13).

(11) Issues the command to cancel the previous suspend command in accordance with Section 12.3.6.4.1.7.1, *Transaction Control without Data Transfer Using DAT Line*.

(12) Check the BS value of the response data. If BS is 0, go to step (7). If BS is 1, go to step (13).

(13) Set MMCSD0_NORMAL_INTR_STS[1] XFER_COMPLETE bit to 1 to clear this bit.

(14) Set MMCSD0_BLOCK_GAP_CONTROL[1] CONTINUE bit to 1 to continue the transaction. At the same time, write 0 to MMCSD0_BLOCK_GAP_CONTROL[0] STOP_AT_BLK_GAP bit to clear this bit.

Conditions			Suspend/Resume Function	
Host Suspend/Resume Support	**Card Suspend/Resume Support**	**Card Read Wait Support**	**Write Suspend/Resume**	**Read Suspend/Resume**
Not supported	Don’t care	Don’t care	Cannot be used	Cannot be used
Supported	Not supported	Don’t care	Cannot be used	Cannot be used
Supported	Supported	Not supported	Can be used	Cannot be used
Supported	Supported	Supported	Can be used	Can be used

mmcsd-046

Figure 12-139. Suspend/Resume Condition

Copyright © 2024 Texas Instruments Incorporated

12.3.6.4.1.12.2 Resume Sequence

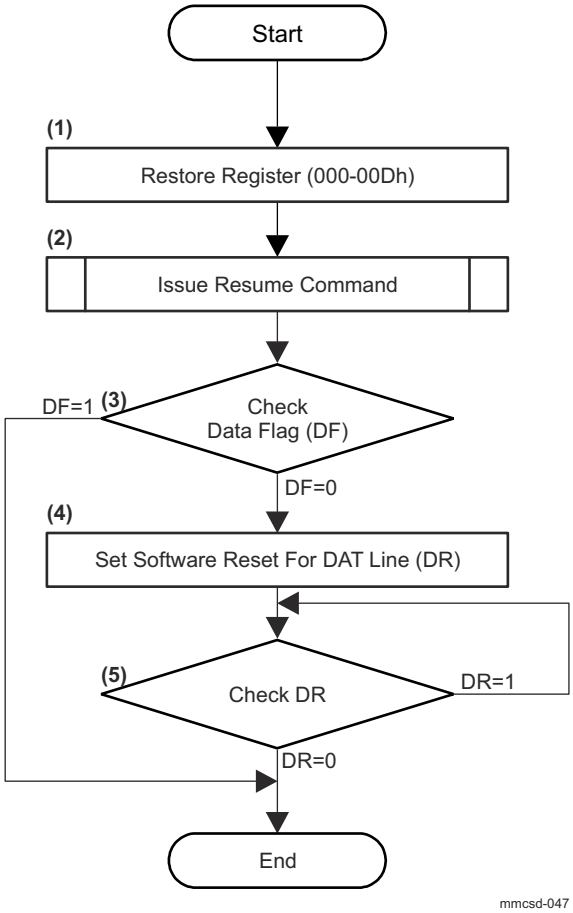


Figure 12-140. The Sequence for Resume

(1) Restore the register .

(2) Issue the Resume Command .

(3) Check the DF value of the response data. If DF is 0, go to step (4). If DF is 1, go to "End".

(4) Set MMCSD0_SOFTWARE_RESET[2] SWRST_FOR_DAT bit to 1 for software reset of the DAT line.

(5) Check MMCSD0_SOFTWARE_RESET[2] SWRST_FOR_DAT bit. If MMCSD0_SOFTWARE_RESET[2] SWRST_FOR_DAT bit is 0, go to "End". If it is 1, go to step (5).

12.3.6.4.1.12.3 Stop At Block Gap/Continue Timing for Read Transaction

The timing of Stop At Block Gap Request and Continue Request. The Transfer Complete interrupt (MMCSD0_NORMAL_INTR_STS[1] XFER_COMPLETE) is always generated by setting MMCSD0_BLOCK_GAP_CONTROL[0] STOP_AT_BLK_GAP bit where data transfer is stopped. However, generation of the Block Gap Event interrupt (MMCSD0_NORMAL_INTR_STS[2] BLK_GAP_EVENT) is dependent on whether the last data block is sent or not. Block Gap Event is not generated If all data blocks are transferred (the last block is transferred). It is not necessary to enable Block Gap Event interrupt. The status can be checked when transfer complete interrupt is detected. It is not necessary to use Continue Request (MMCSD0_BLOCK_GAP_CONTROL[1] CONTINUE) if Block Gap Event status is not set because there is no further data to be transferred. If Read Wait is not supported, Host Controller stops SD Clock at the block gap.

Implementation Note:

Copyright © 2024 Texas Instruments Incorporated

The MMCSD0_BLOCK_GAP_CONTROL[2] RDWAIT_CTRL, MMCSD0_PRESENTSTATE[2] DATA_LINE_ACTIVE, MMCSD0_PRESENTSTATE[9] RD_XFER_ACTIVE bits shall be set and cleared by the Host Controller.

The MMCSD0_BLOCK_GAP_CONTROL[0] STOP_AT_BLK_GAP bit shall be set and cleared by the Host Driver. The MMCSD0_BLOCK_GAP_CONTROL[1] CONTINUE bit shall be set by the Host Driver and be cleared by the Host Controller. The MMCSD0_NORMAL_INTR_STS[2] BLK_GAP_EVENT bit and MMCSD0_NORMAL_INTR_STS[1] XFER_COMPLETE bit shall be set by the Host Controller and be cleared by the Host Driver.

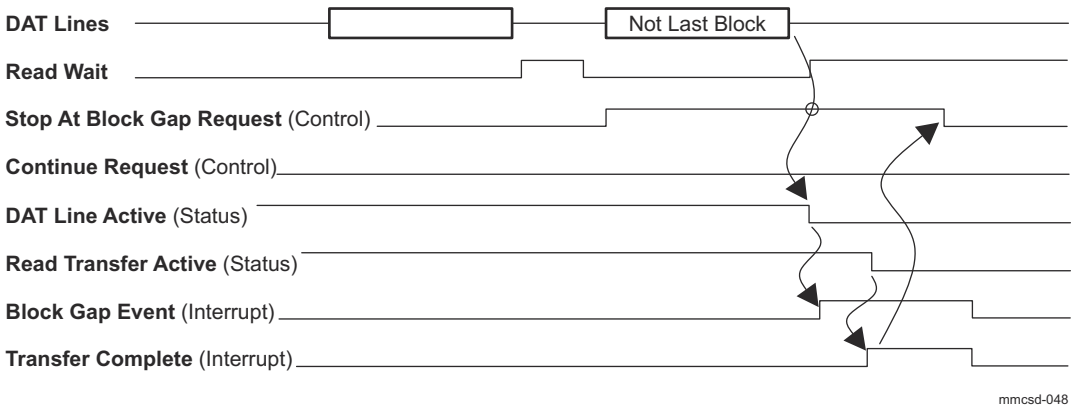


Figure 12-141. Wait Read Transfer by Stop At Block Gap Request

The Host Controller can accept a Stop At Block Gap Request (MMCSD0_BLOCK_GAP_CONTROL[0] STOP_AT_BLK_GAP) when all the following conditions are met.

(1) It is at the block gap.

(2) The Host Controller can assert read wait or it is already asserted.

(3) The MMCSD0_BLOCK_GAP_CONTROL[2] RDWAIT_CTRL bit is set to 1.

After accepting the MMCSD0_BLOCK_GAP_CONTROL[0] STOP_AT_BLK_GAP bit:

(1) Clear MMCSD0_PRESENTSTATE[2] DATA_LINE_ACTIVE bit and generate the Block Gap Event Interrupt (MMCSD0_NORMAL_INTR_STS[2] BLK_GAP_EVENT).

(2) After all valid data has been read (No valid read data remains in the Host Controller), clear the MMCSD0_PRESENTSTATE[9] RD_XFER_ACTIVE bit and generate the Transfer Complete Interrupt (MMCSD0_NORMAL_INTR_STS[1] XFER_COMPLETE).

(3) After accepting MMCSD0_NORMAL_INTR_STS[1] XFER_COMPLETE bit, clear the MMCSD0_BLOCK_GAP_CONTROL[0] STOP_AT_BLK_GAP bit.

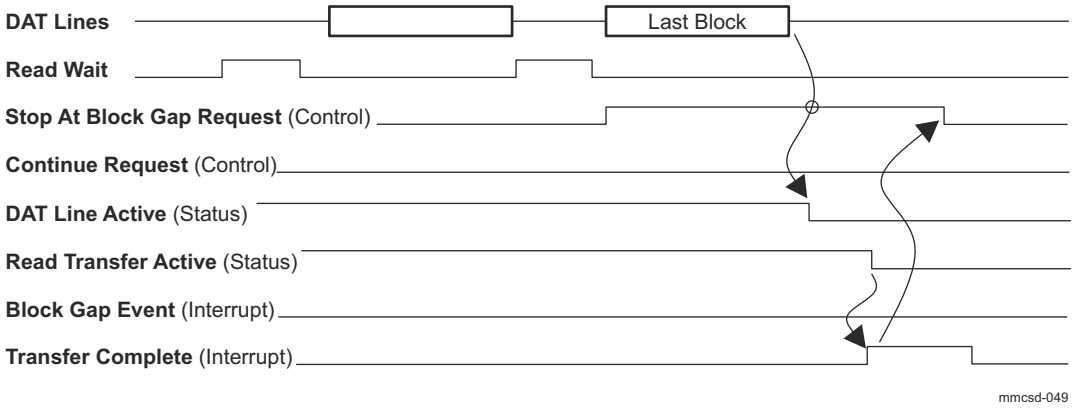


Figure 12-142. Stop At Block Gap Request is Not Accepted at the Last Block of the Read Transfer

Copyright © 2024 Texas Instruments Incorporated

If the MMCSD0_BLOCK_GAP_CONTROL[0] STOP_AT_BLK_GAP bit is set to 1 during the last block transfer, the Host Controller shall not accept the MMCSD0_BLOCK_GAP_CONTROL[0] STOP_AT_BLK_GAP bit and stops the transaction normally. The Block Gap Event Interrupt (MMCSD0_NORMAL_INTR_STS[2] BLK_GAP_EVENT) is not generated. When the Transfer Complete Interrupt (MMCSD0_NORMAL_INTR_STS[1] XFER_COMPLETE) is generated, and if the MMCSD0_NORMAL_INTR_STS[2] BLK_GAP_EVENT bit status is not set to 1, the driver shall clear the MMCSD0_BLOCK_GAP_CONTROL[0] STOP_AT_BLK_GAP bit.

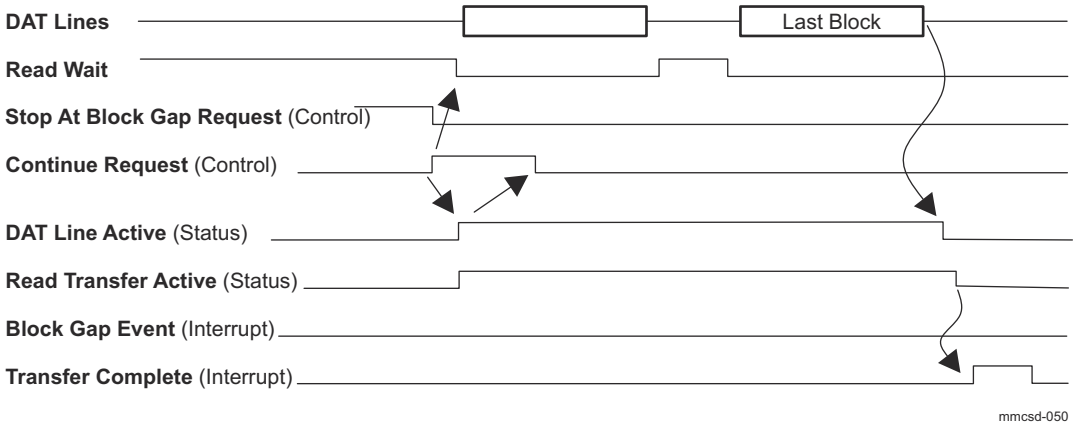


Figure 12-143. Continue Read Transfer by Continue Request

To restart a stopped data transfer, set the MMCSD0_BLOCK_GAP_CONTROL[1] CONTINUE bit to 1 (the MMCSD0_BLOCK_GAP_CONTROL[0] STOP_AT_BLK_GAP bit shall be set to 0).

After accepting the MMCSD0_BLOCK_GAP_CONTROL[1] CONTINUE bit:

(1) Release MMCSD0_BLOCK_GAP_CONTROL[2] RDWAIT_CTRL bit (if the data block can accept the next data).

(2) Set the MMCSD0_PRESENTSTATE[2] DATA_LINE_ACTIVE bit and the MMCSD0_PRESENTSTATE[9] RD_XFER_ACTIVE bit.

(3) The MMCSD0_BLOCK_GAP_CONTROL[1] CONTINUE bit is automatically cleared by (2).

The end of the read transfer is specified by data length.

(1) Clear the MMCSD0_PRESENTSTATE[2] DATA_LINE_ACTIVE bit and do not generate the Block Gap Event Interrupt (MMCSD0_NORMAL_INTR_STS[2] BLK_GAP_EVENT bit).

(2) After all valid data has been read (No valid read data remains in the Host Controller), clear the MMCSD0_PRESENTSTATE[9] RD_XFER_ACTIVE bit and generate the Transfer Complete Interrupt (MMCSD0_NORMAL_INTR_STS[1] XFER_COMPLETE bit).

Copyright © 2024 Texas Instruments Incorporated

12.3.6.4.1.12.4 Stop At Block Gap/Continue Timing for Write Transaction

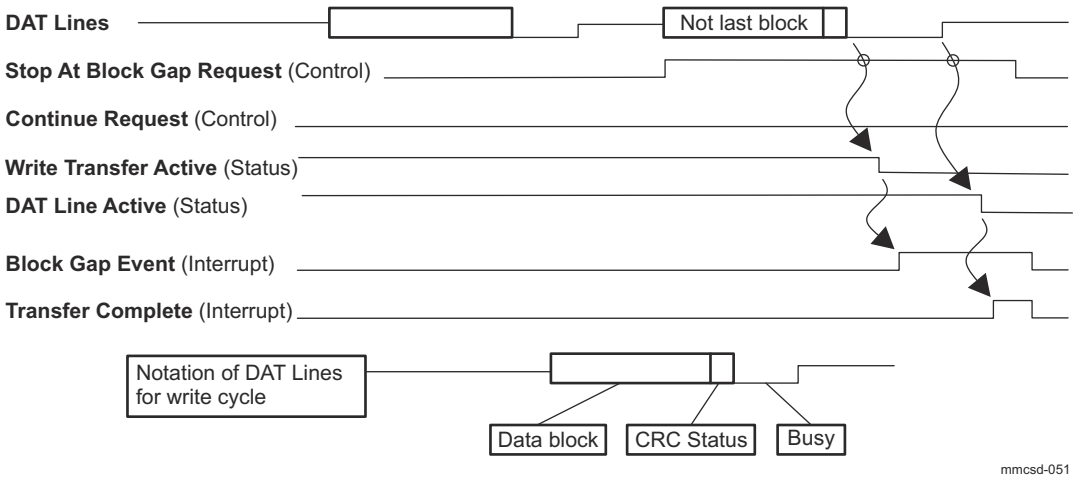


Figure 12-144. Wait Write Transfer by Stop At Block Gap Request

The Host Controller can accept the MMCSD0_BLOCK_GAP_CONTROL[0] STOP_AT_BLK_GAP bit when matches all following conditions:

(1) It is at the block gap.

(2) No valid write data remains in the Host Controller.

After accepting the MMCSD0_BLOCK_GAP_CONTROL[0] STOP_AT_BLK_GAP bit.

(1) Clear the MMCSD0_PRESENTSTATE[8] WR_XFER_ACTIVE bit and generate the Block Gap Event Interrupt (MMCSD0_NORMAL_INTR_STS[2] BLK_GAP_EVENT bit).

(2) After the busy signal is released, clear the MMCSD0_PRESENTSTATE[2] DATA_LINE_ACTIVE bit and generate the Transfer Complete Interrupt (MMCSD0_NORMAL_INTR_STS[1] XFER_COMPLETE bit).

(3) After accepting the Transfer Complete Interrupt (MMCSD0_NORMAL_INTR_STS[1] XFER_COMPLETE bit), clear the MMCSD0_BLOCK_GAP_CONTROL[0] STOP_AT_BLK_GAP bit.

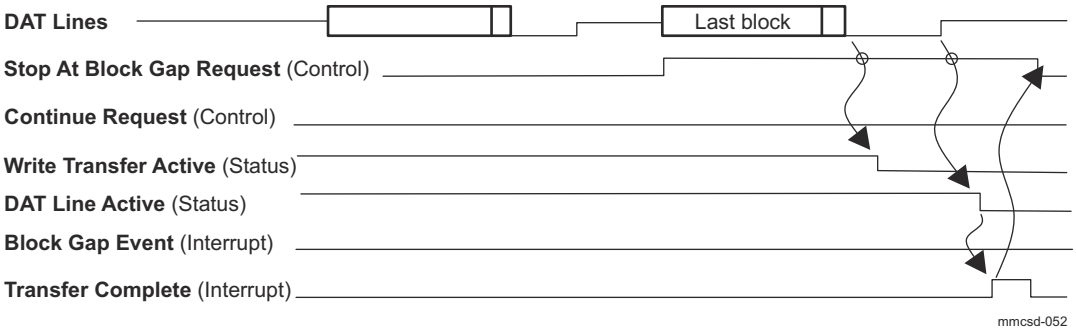


Figure 12-145. Stop At Block Gap Request is Not Accepted at the Last Block of the Write Transfer

If the MMCSD0_BLOCK_GAP_CONTROL[0] STOP_AT_BLK_GAP bit is set to 1 during the last block transfer, the Host Controller shall not accept the MMCSD0_BLOCK_GAP_CONTROL[0] STOP_AT_BLK_GAP bit and terminates the transaction normally. The Block Gap Event Interrupt (MMCSD0_NORMAL_INTR_STS[2] BLK_GAP_EVENT) is not generated. When the Transfer Complete Interrupt (MMCSD0_NORMAL_INTR_STS[1] XFER_COMPLETE) is generated, and if the MMCSD0_NORMAL_INTR_STS[2] BLK_GAP_EVEN bit is not set to 1, the driver shall clear the MMCSD0_BLOCK_GAP_CONTROL[0] STOP_AT_BLK_GAP bit.

Copyright © 2024 Texas Instruments Incorporated

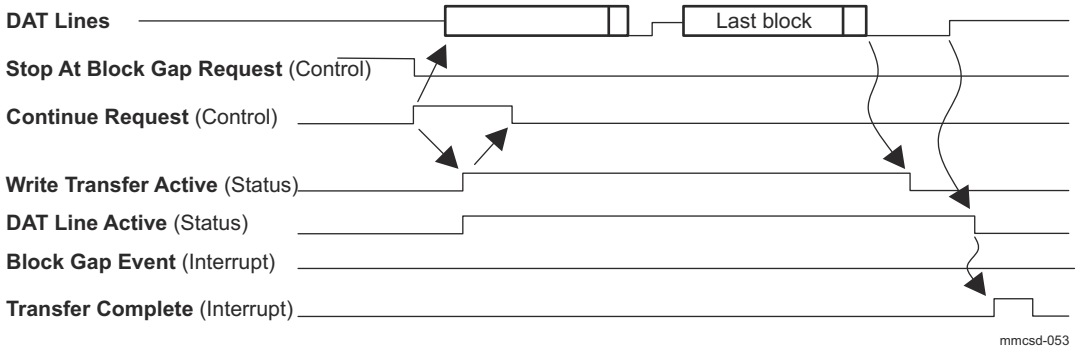


Figure 12-146. Continue Write Transfer by Continue Request

To restart a stopped data transfer, set the MMCSD0_BLOCK_GAP_CONTROL[1] CONTINUE bit to 1 (the MMCSD0_BLOCK_GAP_CONTROL[0] STOP_AT_BLK_GAP bit shall be set to 0).

After accepting the MMCSD0_BLOCK_GAP_CONTROL[1] CONTINUE bit:

(1) Set the MMCSD0_PRESENTSTATE[2] DATA_LINE_ACTIVE bit and MMCSD0_PRESENTSTATE[8] WR_XFER_ACTIVE bit.

(2) The MMCSD0_BLOCK_GAP_CONTROL[1] CONTINUE bit is automatically cleared by (1).

The end of transfer is specified by data length.

(1) Clear the MMCSD0_PRESENTSTATE[8] WR_XFER_ACTIVE bit, and do not generate the (MMCSD0_NORMAL_INTR_STS[2] BLK_GAP_EVEN).

(2) After the busy signal is released, clear the MMCSD0_PRESENTSTATE[2] DATA_LINE_ACTIVE bit and generates the Transfer Complete Interrupt (MMCSD0_NORMAL_INTR_STS[1] XFER_COMPLETE bit).

Copyright © 2024 Texas Instruments Incorporated

12.3.6.4.2 Driver Flow Sequence

Note

This section is applicable for MMCSD0 and MMCSD1 but MMCSD0 registers are used in the content.

Note

UHSII is not supported. For more information, see *MMCSD Not Supported Features*.

12.3.6.4.2.1 Host Controller Setup and Card Detection

12.3.6.4.2.1.1 Host Controller Setup Sequence

Figure 12-147 and Table 12-225 show a Host Controller setup sequence.

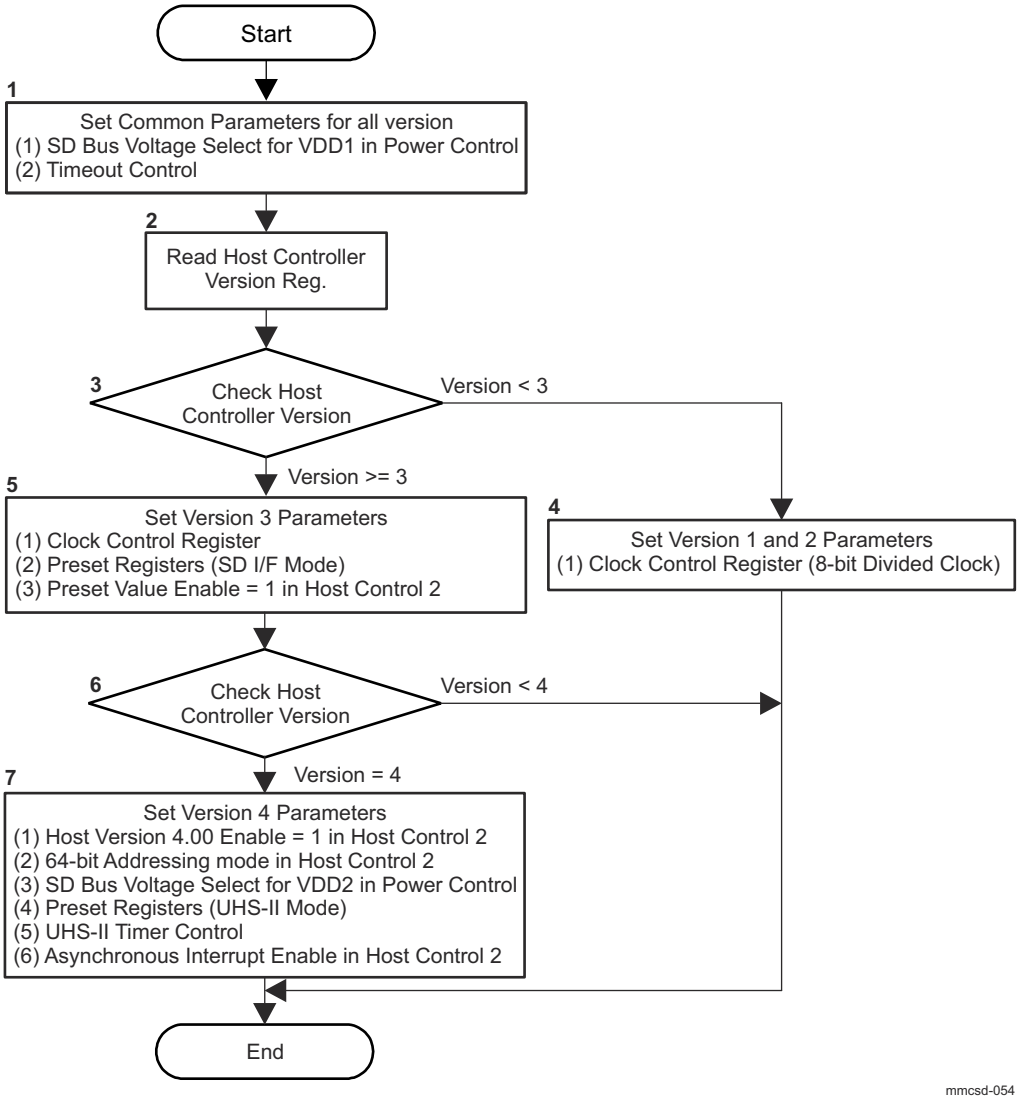


Figure 12-147. Host Controller Setup Sequence

Table 12-225. Host Controller Setup Sequence

Step	Description
1	Set parameters for all Host Controller version. Set the MMCSD0_POWER_CONTROL[3-1] SD_BUS_VOLTAGE and MMCSD0_TIMEOUT_CONTROL[3-0] COUNTER_VALUE bit fields.
2	Read the MMCSD0_HOST_CONTROLLER_VER[7-0] SPEC_VER_NUM bit field.

Copyright © 2024 Texas Instruments Incorporated

Table 12-225. Host Controller Setup Sequence (continued)

Step	Description
3	If Specification Version number (MMCSD0_HOST_CONTROLLER_VER[7-0] SPEC_VER_NUM) is less than version 3, go to step (4), if it is version 3 or later go to step (5).
4	Set the MMCSD0_CLOCK_CONTROL register using 8-bit Divided Clock mode.
5	Set Version 3 parameters. The MMCSD0_CLOCK_CONTROL register is sets in 10-bit Divided Clock Mode or Programmable Clock Mode. If Clock Multiplier in the MMCSD0_CAPABILITIES register is not zero, Programmable Clock Mode should be used. If Preset Value is used, set Preset Values of SD I/F Modes in the Preset Value register (MMCSD0_PRESET_VALUE0 - MMCSD0_PRESET_VALUE10) and set the MMCSD0_HOST_CONTROL2[15] PRESET_VALUE_ENA bit to 1.
6	If Specification Version number (MMCSD0_HOST_CONTROLLER_VER[7-0] SPEC_VER_NUM) is version 4, go to step (7), if it is less than version 4, exits.
7	Set Version 4 parameters. Set the MMCSD0_HOST_CONTROL2[12] HOST_VER40_ENA bit to 1. If the MMCSD0_CAPABILITIES[27] ADDR_64BIT_SUPPORT_V4 bit is set to 1, set the MMCSD0_HOST_CONTROL2[13] BIT64_ADDRESSING bit to 1. If UHS-II Support is set to 1 and 1.8 V VDD2 Support is set to 1 in the MMCSD0_CAPABILITIES register, set SD Bus Voltage Select for VDD2 to 1.8 V mode in the MMCSD0_POWER_CONTROL register, set preset value for UHS-II Mode to Preset Value register (MMCSD0_PRESET_VALUE0 - MMCSD0_PRESET_VALUE10) and set Timeout Counter Value for CMD_RES and Timeout Counter Value for Deadlock in the MMCSD0_UHS2_TIMER_CONTROL register based on Timeout Clock Frequency and Timeout Clock Unit in the MMCSD0_CAPABILITIES register. If Asynchronous Interrupt Support in the MMCSD0_CAPABILITIES register is set to 1, set Asynchronous Interrupt Enable to 1 in the MMCSD0_HOST_CONTROL2 register.

12.3.6.4.2.1.2 Card Interface Detection Sequence

This procedure is invoked by the Card Insertion interrupt. Figure 12-148 and Table 12-226 show a card interface detection sequence.

Copyright © 2024 Texas Instruments Incorporated

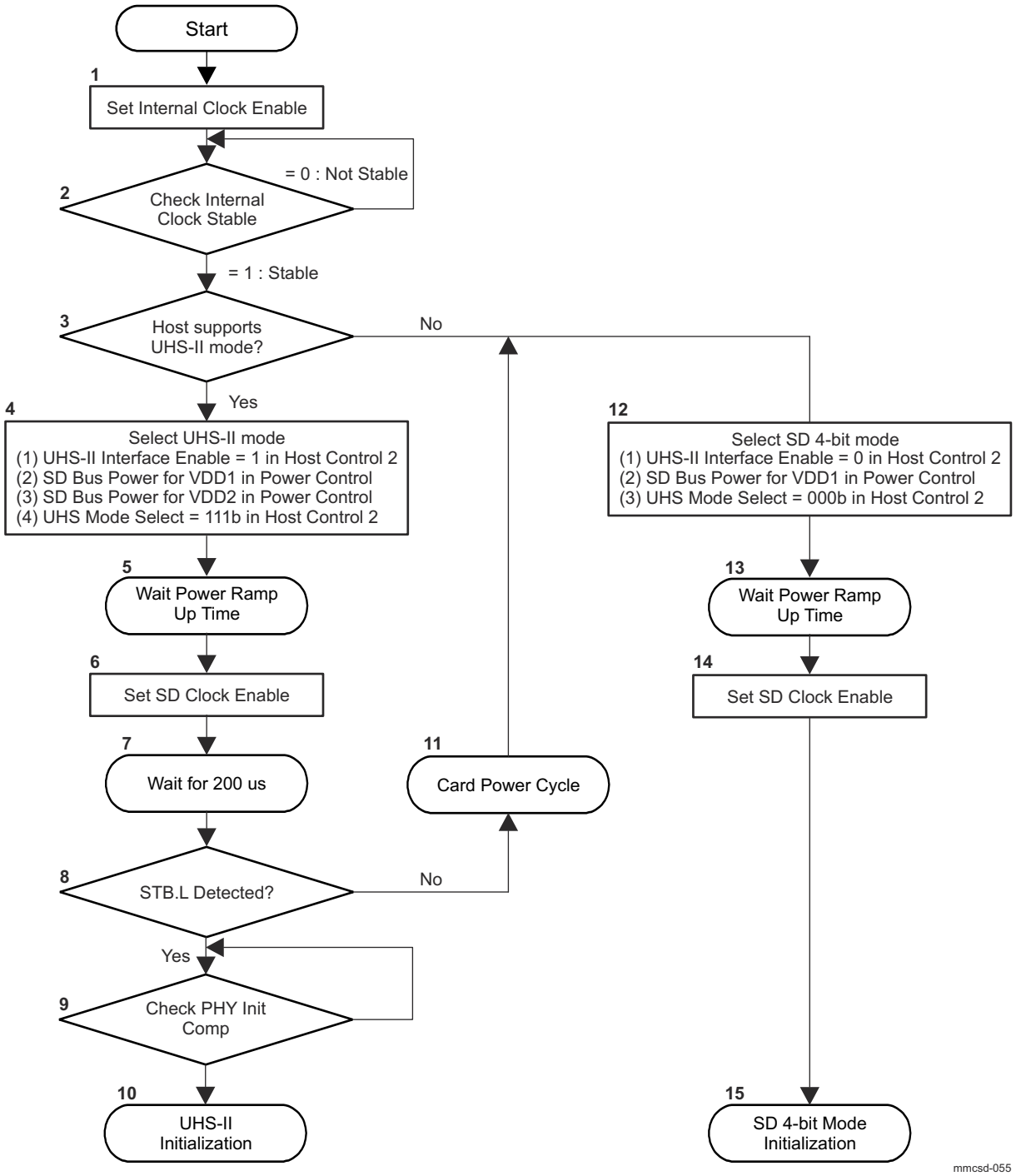


Figure 12-148. Card Interface Detection Sequence

Table 12-226. Card Interface Detection Sequence

Step	Description
1	Set Internal Clock Enable to 1 in the MMCSD0_CLOCK_CONTROL register.
2	Wait until Internal Clock Stable is set to 1 in the MMCSD0_CLOCK_CONTROL register.
3	If Host Supports UHS-II, go to step (4) else go to step (12).
4	Try to initialize a card to UHS-II mode. Set UHS-II Interface Enable to 1 in the MMCSD0_HOST_CONTROL2 register, set SD Bus Power for VDD1 and SD Bus Power for VDD2 to 1 in the MMCSD0_POWER_CONTROL register, and set UHS Mode Select to 111b in the MMCSD0_HOST_CONTROL2 register.
5	Wait power ramp up time. It is dependent on a Host System.

Copyright © 2024 Texas Instruments Incorporated

Table 12-226. Card Interface Detection Sequence (continued)

Step	Description
6	Set SD Clock Enable to 1 in the MMCSD0_CLOCK_CONTROL register.
7	Wait 200 µs to check a card supports UHS-II mode.
8	Check STB.L Detection in the MMCSD0_PRESENTSTATE register. If UHS-II IF is detected, go to step (9). If UHS-II IF is not detected, go to step (11).
9	Wait until Lane Synchronization in the MMCSD0_PRESENTSTATE register is set to 1 (PHY Initialization is completed).
10	Perform UHS-II initialization.
11	Perform card power cycle.
12	Try to initialize a card to SD 4-bit mode. Set UHS-II Interface Enable to 0 in the MMCSD0_HOST_CONTROL2 register, set SD Bus Power for VDD1 to 1 in the MMCSD0_POWER_CONTROL register, and set UHS Mode Select to 000b in the MMCSD0_HOST_CONTROL2 register.
13	Wait power ramp up time. It is dependent on a Host System.
14	Set SD Clock Enable to 1 in the MMCSD0_CLOCK_CONTROL register.
15	Perform SD 4-bit mode initialization. Refer to Section 12.3.6.4.1.6, *Card Initialization and Identification (for SD I/F)*.

12.3.6.4.2.2 Boot Operation

The boot operation sequence is shown in Figure 12-149.

Copyright © 2024 Texas Instruments Incorporated

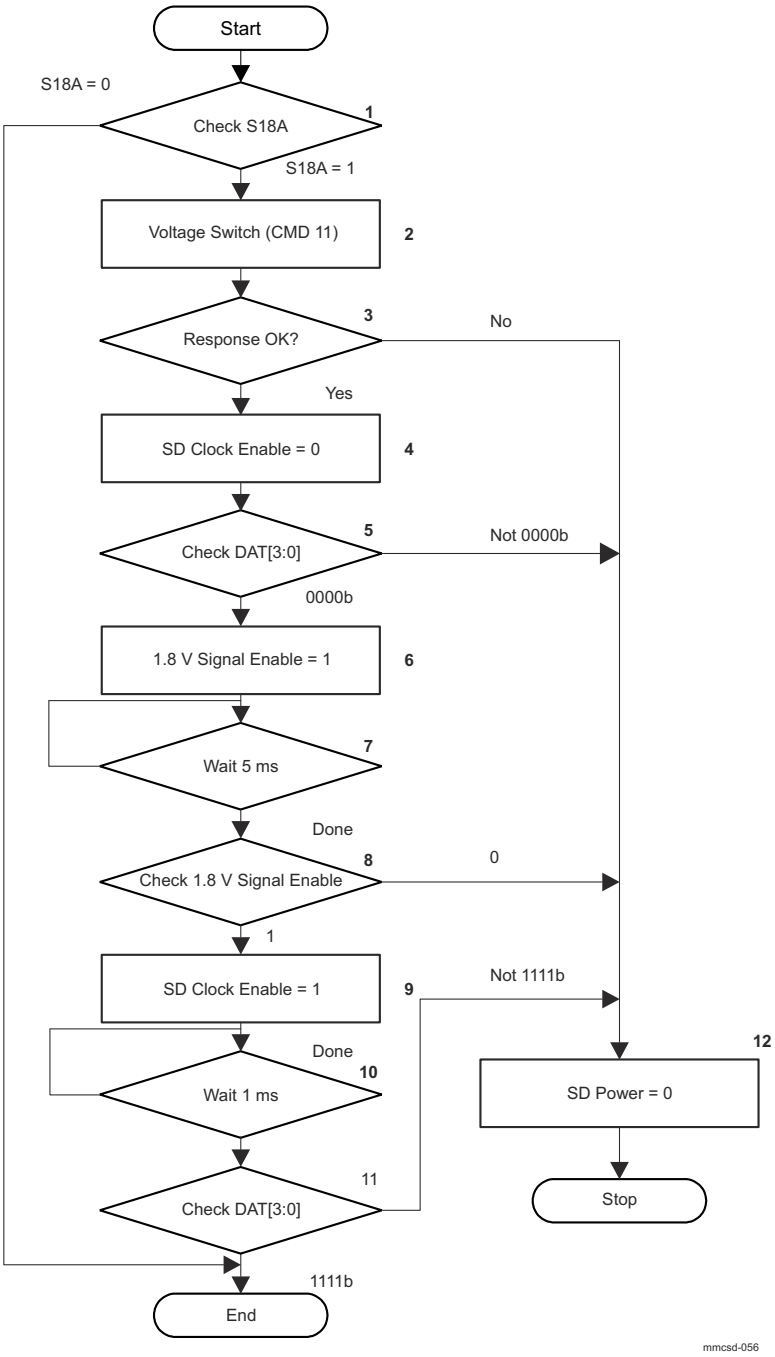


Figure 12-149. Boot Operation Sequence

12.3.6.4.2.2.1 Normal Boot Operation: (For Legacy eMMC 5.0)

The Host driver writes the boot timeout value into MMCSD0_BOOT_TIMEOUT_CONTROL register as per the eMMC4.3+ spec.

When the Host driver sets the "BOOT_ENABLE" to 1 in MMCSD0_BLOCK_GAP_CONTROL register and "DATA_XFER_DIR" bit to "1" in MMCSD0_TRANSFER_MODE register, the Host controller drives the CMD line to "0" for boot operation.

If the Host controller is configured to wait for boot acknowledgement from the eMMC4.3+ device, the controller receives the boot acknowledgement and asserts the "RCV_BOOT_ACK" interrupt to the driver (MMCSD0_NORMAL_INTR_STS[13] RCV_BOOT_ACK). If the Host controller doesn't receive the boot

Copyright © 2024 Texas Instruments Incorporated

acknowledgement from the device within the timeout value, the Host controller will assert the data timeout error interrupt (MMCSD0_ERROR_INTR_STS[4] DATA_TIMEOUT).

After servicing the boot acknowledge interrupt or data timeout error interrupt, the driver programs the MMCSD0_BOOT_TIMEOUT_CONTROL register with boot data timeout value.

The eMMC4.3+ device starts sending the boot data on the data line and Host controller sends the same to system whenever a block of data is received from device. The Host controller terminates the boot operation when the programmed number of blocks is transferred to the System. The Driver can writes "BOOT_ENABLE" as "0" in MMCSD0_BLOCK_GAP_CONTROL register.

The boot operation can also be terminated in between the boot transfer (the Driver can set the "BOOT_ENABLE" as "0" anytime to disable the boot operation). After the boot termination, the Host controller asserts soft reset for CMD line and DATA line internally once the driver clears the BOOT_COMPLETE interrupt (MMCSD0_NORMAL_INTR_STS[14] BOOT_COMPLETE), to reset all the state machines to idle state.

There is no need to perform error recovery sequence in case data timeout interrupt occurs during boot operation (ignore sending abort command, soft reset and just clear the timeout interrupt and proceed with the boot flow).

If the device sends wrong acknowledgement to the Host, the Host controller will assert Data CRC Error interrupt (MMCSD0_ERROR_INTR_STS[5] DATA_CRC) on the Driver. In this case, the Host driver has to stop the boot mode operation by setting "BOOT_ENABLE" as "0" in MMCSD0_BLOCK_GAP_CONTROL register and write soft reset for CMD and data line.

If the device sends end bit as "0" in acknowledgement to the Host, the Host controller will assert Data End Bit Error interrupt (MMCSD0_ERROR_INTR_STS[6] DATA_ENDBIT) on the Driver.

The Host driver stops the boot mode operation by setting "BOOT_ENABLE" as "0" in MMCSD0_BLOCK_GAP_CONTROL register and write soft reset for CMD and data line.

Note: Enable the MMCSD0_BLOCK_GAP_CONTROL[7] BOOT_ACK_ENA bit during boot operation. If failed, the controller will not wait for boot acknowledge from the card and send out data CRC error when the card sends boot acknowledgement first and followed by boot data.

12.3.6.4.2.2.2 Alternate Boot Operation (For Legacy eMMC 5.0):

The Host driver writes the boot timeout value as per the eMMC4.3+ spec into MMCSD0_BOOT_TIMEOUT_CONTROL register.

If the "ALT_BOOT_MODE" and "BOOT_ENABLE" is set to 1 in MMCSD0_BLOCK_GAP_CONTROL register and "DATA_XFER_DIR" bit is set to 1 in MMCSD0_TRANSFER_MODE register, the host controller drives the CMD0 (0xFFFFFFFA) on the CMD line.

The system shall wait for command complete interrupt before "RCV_BOOT_ACK" interrupt (MMCSD0_NORMAL_INTR_STS[13] RCV_BOOT_ACK).

If the Host controller is configured to wait for boot acknowledgement from the eMMC4.3+ device, the controller receives the boot acknowledgement and asserts the "RCV_BOOT_ACK" interrupt to the driver.

The eMMC4.3+ device starts sending the boot data on the data line and Host controller sends the same to system whenever a block of data is received from device. The System needs to send CMD0 to inform the device about the boot operation complete. The system shall program MMCSD0_COMMAND register with argument "00000000h" for the CMD0 and waits for command complete.

The Host controller terminates the boot operation when the programmed number of blocks are transferred to the System and the Driver shall send CMD0 to eMMC card and then program "BOOT_ENABLE" as "0" in MMCSD0_BLOCK_GAP_CONTROL register.

The boot operation can be terminated at any time (the Driver can send CMD0 and set the "BOOT_ENABLE" as "0" anytime to disable the boot operation). After the boot termination, all state machines in the Host controller need to move to IDLE state. The Host controller asserts the soft reset for CMD and data line internally when the driver clears the BOOT_COMPLETE interrupt (MMCSD0_NORMAL_INTR_STS[14] BOOT_COMPLETE) and all the state machine goes to the idle state.

Copyright © 2024 Texas Instruments Incorporated

If the Host controller doesn't receive the acknowledgement from the device with in timeout value the Host controller will assert the data timeout error interrupt (MCSD0_ERROR_INTR_STS[4] DATA_TIMEOUT).

The driver programs the MMCSD0_BOOT_TIMEOUT_CONTROL register with boot data timeout value.

There is no need to perform error recovery sequence in case of data timeout interrupt as mentioned above. If the device sends wrong acknowledgement to the Host, then Host controller will assert Data CRC Error interrupt to the Driver (MCSD0_ERROR_INTR_STS[5] DATA_CRC). The Host driver has to stop the boot mode operation by setting "ALT_BOOT_MODE" and "BOOT_ENABLE" as "0" in MMCSD0_BLOCK_GAP_CONTROL register and write soft reset for CMD and data line.

If the device sends end bit as "0" in acknowledgement to the Host, the Host controller asserts Data End Bit Error interrupt (MCSD0_ERROR_INTR_STS[6] DATA_ENDBIT) on the Driver. The Host driver stops the boot mode operation by setting "ALT_BOOT_MODE" and "BOOT_ENABLE" as "0" in MMCSD0_BLOCK_GAP_CONTROL register and write soft reset for CMD and data line.

Note: Enable the MMCSD0_BLOCK_GAP_CONTROL[7] BOOT_ACK_ENA bit during boot operation . If failed, the controller will not wait for boot acknowledge from the card and send out data CRC error when the card sends boot acknowledgement first and followed by boot data.

12.3.6.4.2.2.3 Boot Code Chunk Read Operation (For Legacy eMMC 5.0):

The following figure explains the boot code read done as chunks of data (the Driver can read the boot data as 2K, 4K, 1K and 8K etc. instead of 128K at a time

Copyright © 2024 Texas Instruments Incorporated

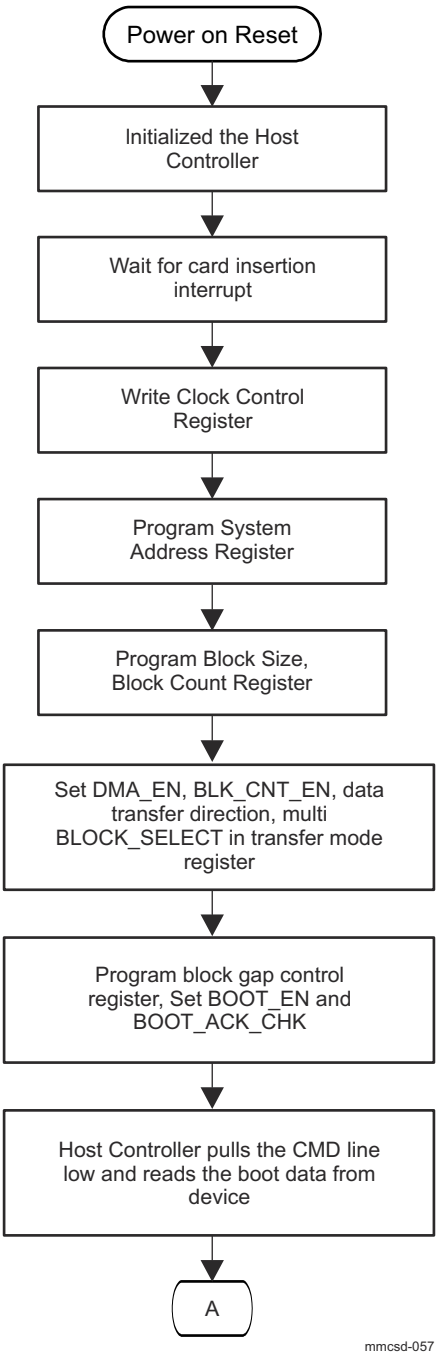


Figure 12-150. Boot Code Access Flow Diagram (1)

Copyright © 2024 Texas Instruments Incorporated

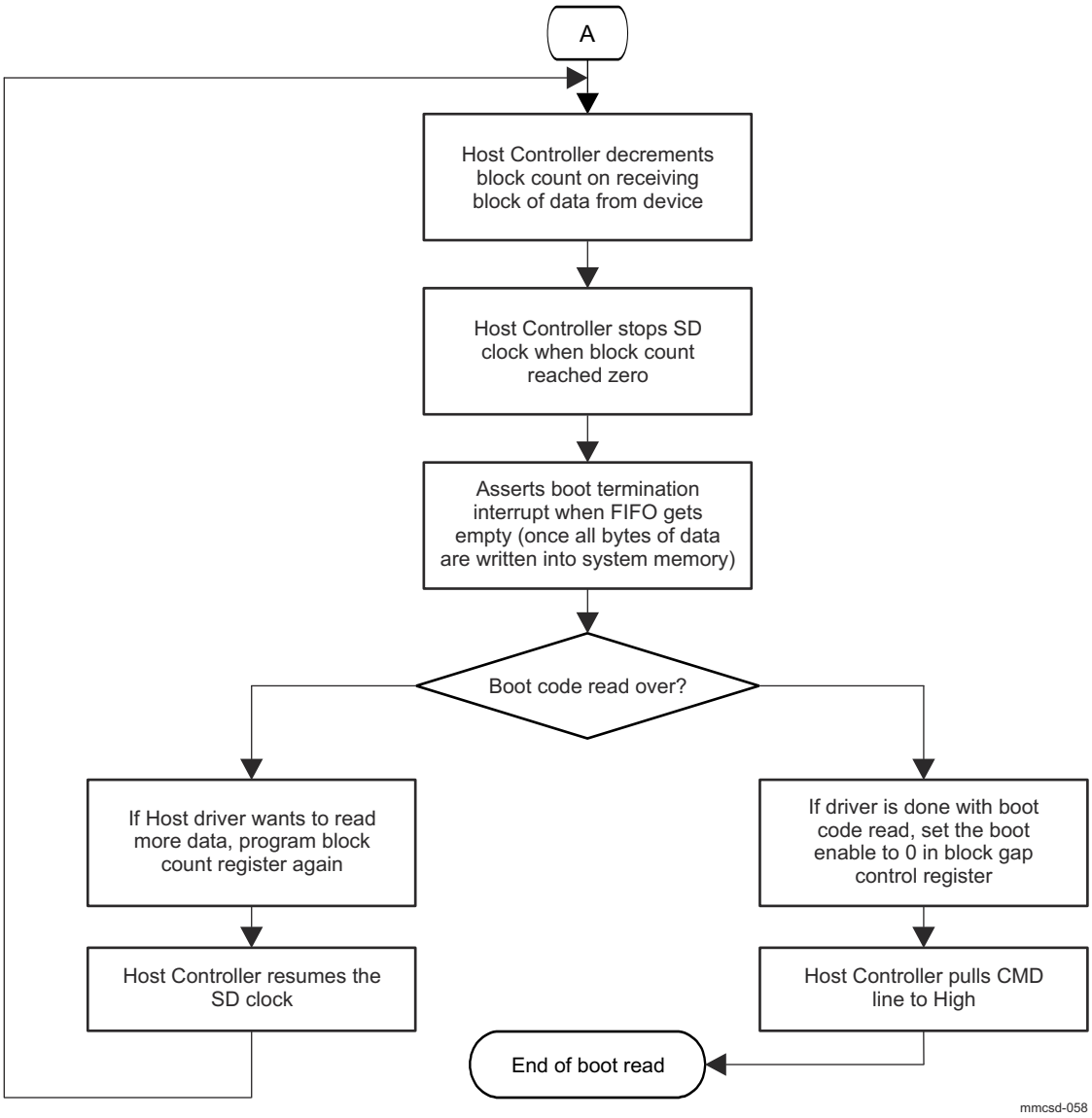


Figure 12-151. Boot Code Access Flow Diagram (2)

12.3.6.4.2.3 Retuning procedure (For Legacy Interface)

12.3.6.4.2.3.1 Sampling Clock Tuning

The SD bus can be operating in high clock frequency mode and then the data window from the card on CMD and DAT[3:0] lines gets smaller. The position of the data window will vary depending on the card and host system implementation. Therefore, the Host Controller shall support a tuning circuit when SDR104 or SDR50.

The Host Controller shall support a tuning circuit when SDR104 or SDR50 (if Use Tuning for SDR50 is set to 1 in the MMCSD0_CAPABILITIES register) is supported by executing the tuning procedure and adjusting the sampling clock. Execute Tuning and Sampling Clock Select in the MMCSD0_HOST_CONTROL2 register are used to control the tuning circuit.

12.3.6.4.2.3.2 Tuning Modes

The re-tuning timing is specified by two methods: Re-Tuning Request generated by Host Controller and expiration of a re-tuning timer prepared by Host Driver.

(1) Re-Tuning Mode 1 The host controller does not have any internal logic to detect when the re-tuning needs to be performed. In this case, the Host Driver should maintain all re-tuning timings by using a Re-Tuning Timer. To

Copyright © 2024 Texas Instruments Incorporated

enable inserting the re-tuning procedure during data transfers, the data length per read/write command shall be limited up to 4 MB.

(2) Re-Tuning Mode 2 The host controller has the capability to indicate the re-tuning timing by Re-Tuning Request during data transfers. Then the data length per read/write command shall be limited up to 4 MB. During non data transfer, re-tuning timing is determined by either Re-Tuning Request or Re-Tuning Timer. If Re-Tuning Request is used, Re-Tuning Timer should be disabled.

12.3.6.4.2.3.3 Re-Tuning Mode 2

The Figure 12-152 defines sampling clock tuning procedure supported by Host Controller. In default, for lower frequency operation, fixed sampling clock is used to receive signals on CMD and DAT[3:0]. Before using SDR104, sampling clock tuning is required. Start of sampling clock tuning is requested by setting Execute Tuning to 1 and Sampling Clock Select to 0.

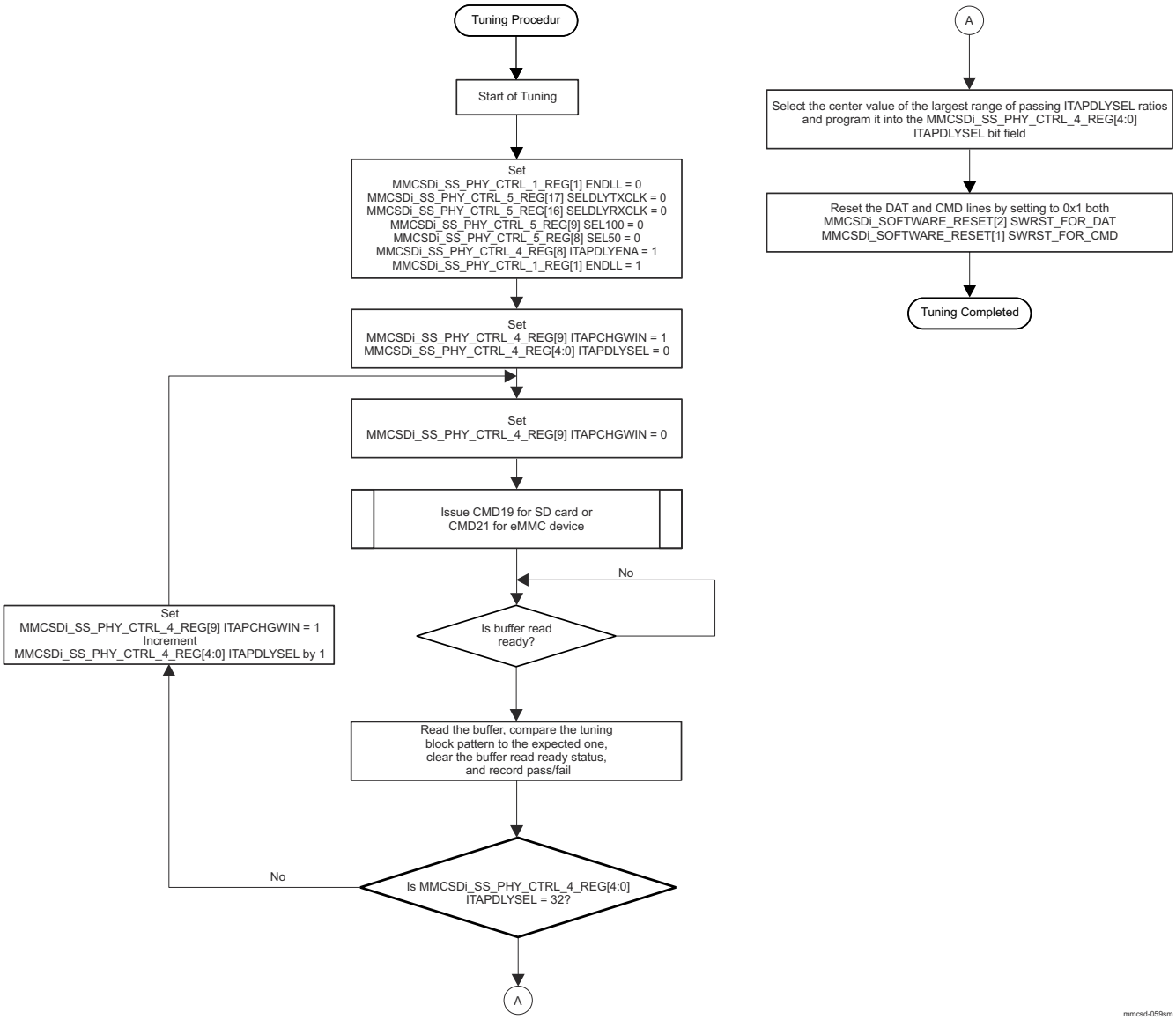


Figure 12-152. MMCSD Clock Tuning

Host driver issue CMD19 repeatedly until the host controller resets Execute Tuning to 0. Host Controller resets Execute Tuning to 0 when tuning is completed or tuning is not completed within 40 times. Host Driver can abort this loop by 40 times CMD19 issue or 150 ms time-out.

Copyright © 2024 Texas Instruments Incorporated

If tuning is completed successfully, Host Controller set Sampling Clock Select to 1 and this means the Host Controller start to use tuned sampling clock. If tuning is failed, Host Controller keeps Sampling Clock Select to 0. By writing Sampling Clock Select to 0, sampling clock is switched from tuned sampling clock to fixed sampling clock. Re-tuning time would be smaller than the first tuning time. CMD19 response errors are not indicated while tuning is performed.

The clock tuning tap delay values are selected using Variable sampling point detection. Fixed tap delay value is used for fixed tuning clock method.

12.3.6.4.2.4 Command Queuing Driver Flow Sequence

12.3.6.4.2.4.1 Command Queuing Initialization Sequence

Figure 12-153 and Table 12-227 show a Command Queuing initialization sequence.

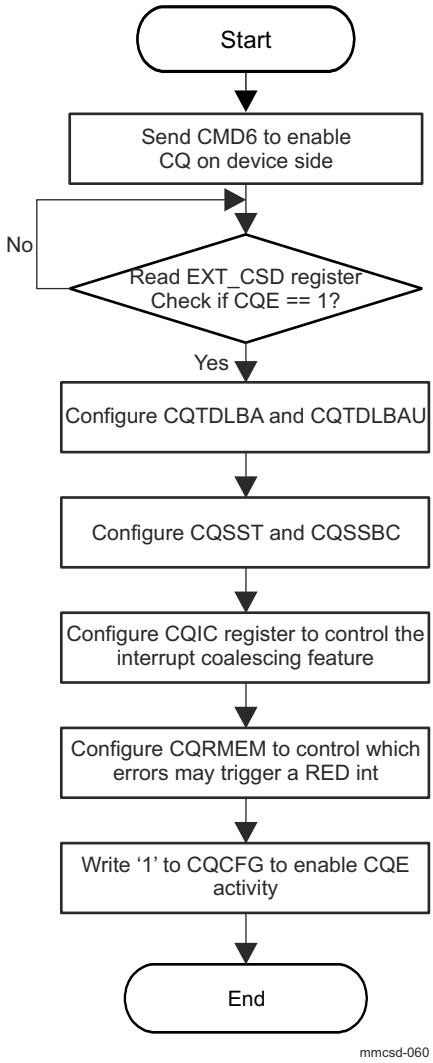


Figure 12-153. Command Queuing Initialization Sequence

Table 12-227. Command Queuing Initialization Sequence

Step	Description
1	Initialize and enable Command Queueing in the device.
2	Configure Task Descriptor size in MMCSD0_CQ_CONFIG register.
3	Configure MMCSD0_CQ_TDL_BASE_ADDR and MMCSD0_CQ_TDL_BASE_ADDR_UPBITS registers to point to the memory location allocated to the TDL in host memory.
4	Configure CQSST and CQSSBC to control when SEND_QUEUE_STATUS commands are sent to the device by CQE.

Copyright © 2024 Texas Instruments Incorporated

Table 12-227. Command Queuing Initialization Sequence (continued)

Step	Description
5	Configure MMCSD0_CQ_INTR_COALESCING register to control the interrupt coalescing feature: enable/disable, set interrupt count and timer protection.
6	Configure MMCSD0_CQ_RESP_ERR_MASK register to control which errors may trigger a RED interrupt (if different from reset values).
7	Write '1' to MMCSD0_CQ_CONFIG register to enable CQE activity.

12.3.6.4.2.4.2 Task Issuance Sequence

Figure 12-154 and Table 12-228 show a task issuance sequence.

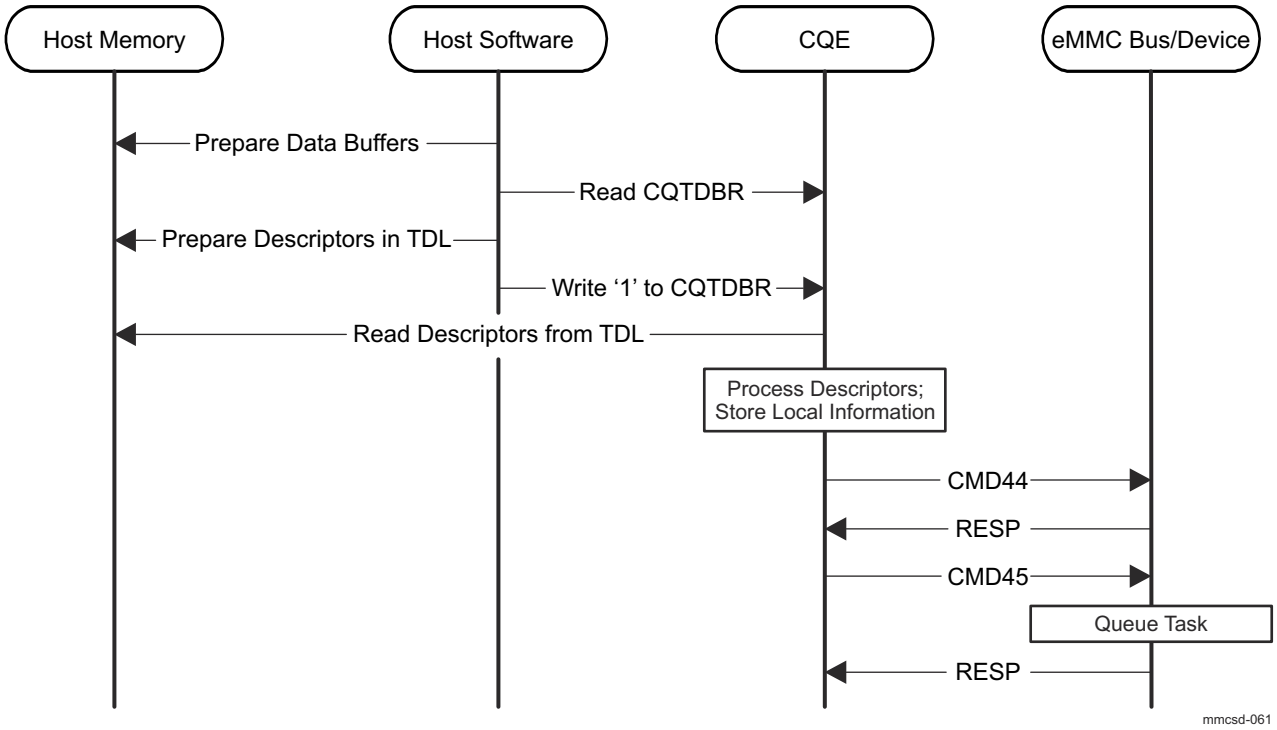


Figure 12-154. Task Issuance Sequence

Table 12-228. Task Issuance Sequence

Step	Description
1	Find an empty transfer request slot by reading the MMCSD0_CQ_TASK_DOOR_BELL register. An empty transfer request slot has its respective bit cleared to '0' in the MMCSD0_CQ_TASK_DOOR_BELL register.
2	Build a Task Descriptor at the 1st entry of the empty slot. Task Descriptor field values: • 1. Valid = 1, to indicate the descriptor is effective. • 2. End = 1, as required for Task Descriptors. • 3. Int = 1, if an interrupt on completion is required. Otherwise, Int = 0. • 4. Act = b101, to indicate a Task Descriptor . • 5. Data Direction = 1 for read commands, and 0 for write commands. • 6. Priority = 1 for high priority, and 0 for simple requests. • 7. QBR = 1 if Queue Barrier functionality is required. 0 otherwise. • 8. Forced Programming, Context ID, Tag Request, Reliable Write, Block Count, and Block Address programmed according to application requirements.

Copyright © 2024 Texas Instruments Incorporated

Table 12-228. Task Issuance Sequence (continued)

Step	Description
3	Build a Transfer Descriptor at the 2nd entry of the empty slot. Transfer Descriptor field values: • 1. Valid = 1, to indicate the descriptor is effective. • 2. End = 1, if a TRAN descriptor is used, to indicate the task only has one data buffer. End = 0 if LINK descriptor is used. • 3. Int = 0. Ignore in Command queueing. • 4. Act = b100, for TRAN descriptors, pointing directly to the task's single data buffer. Act = 401 b110, for LINK descriptors, point to a scatter/gather list (indirect). • 5. Address and Length programmed according to the data buffer supplied by the application.
4	If more than one transfer is requested, repeat step 1-3 for all needed transfers.
5	Set MMCSD0_CQ_TASK_DOOR_BELL register to indicate to the CQE that one or more transfer requests are ready to be sent to the attached device. Host software shall only write a '1' to the bit position that corresponds to new tasks; all other bit positions within MMCSD0_CQ_TASK_DOOR_BELL register should be written with a '0', which indicates no change to their current values.

12.3.6.4.2.4.3 Task Execution and Completion Sequence

The CQE is responsible for task execution, communication with the device and moving the data to the buffers in the host memory.

Figure 12-155 and Table 12-229 show a task execution and completion sequence.

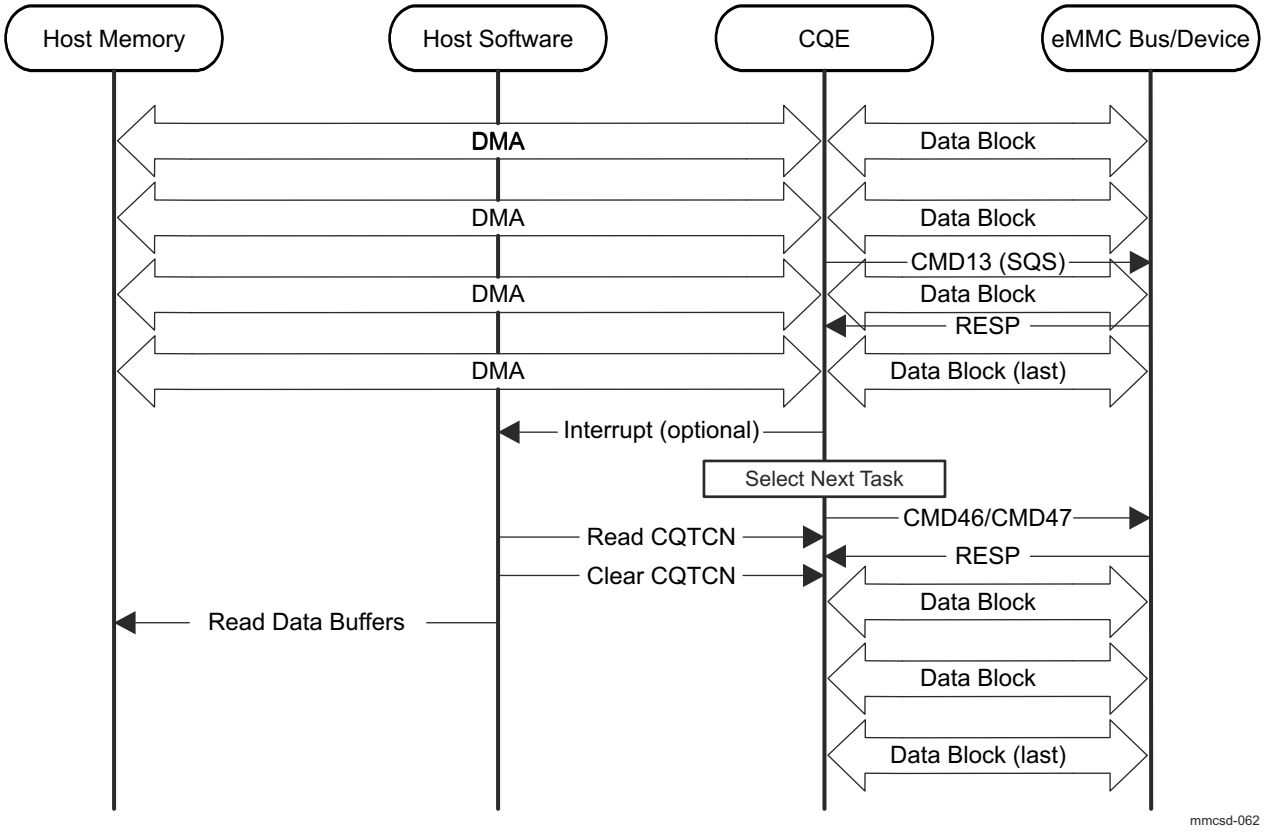


Figure 12-155. Task Execution and Completion Sequence

Table 12-229. Task Execution and Completion Sequence

Step	Description
1	Once the tasks are queued on the device side CQE sends CMD13 to determine the queue status. The queue status lets the CQE know which tasks are ready for execution.
2	The response to CMD13 may indicate no task is ready in which case the CQE is required to send CMD13 again at a later time as controlled CQSST register.

Copyright © 2024 Texas Instruments Incorporated

Table 12-229. Task Execution and Completion Sequence (continued)

Step	Description
3	If a task is ready for execution then the host sends EXECUTE_READ_TASK (CMD46) or EXECUTE_WRITE_TASK (CMD47) to the device with the task id ordering it to execute a task.
4	When the task execution is completed, an interrupt may be generated, if requested, or as determined by Interrupt Coalescing mechanism.
5	The host software reads MMCSD0_CQ_TASK_COMP_NOTIF register to determine which task(s) has(have) been completed. Each bit set in MMCSD0_CQ_TASK_COMP_NOTIF register represents by index which task has completed but hasn't yet been served by software.
6	For every task completed clear the appropriate MMCSD0_CQ_TASK_COMP_NOTIF register bit.
7	Repeat steps 1-6 for the pending tasks.

12.3.6.4.2.4.4 Task Discard and Clear Sequence

Figure 12-156 and Table 12-230 show a task discard and clear sequence.

Copyright © 2024 Texas Instruments Incorporated

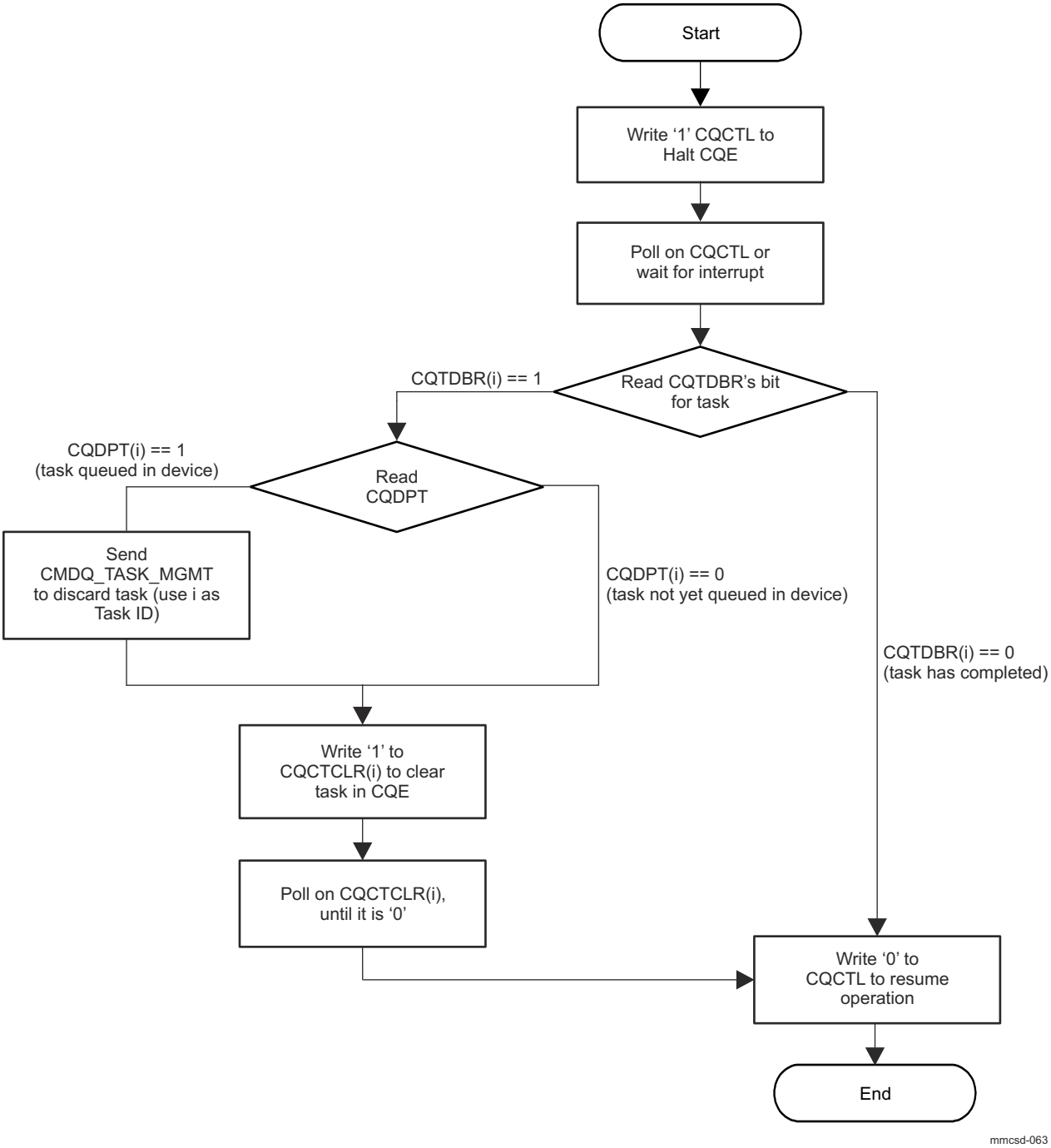


Figure 12-156. Task Discard and Clear Sequence

Table 12-230. Task Discard and Clear Sequence

Step	Description
1	While CQE is enabled write '1' to MMCSD0_CQ_CONTROL[0] HALT_BIT bit to halt CQE.
2	Poll on MMCSD0_CQ_INTR_STS[0] HALT_COMPLETE bit.
3	Read MMCSD0_CQ_TASK_DOOR_BELL register to determine if the task to be dicarded is set to 1.
4	Read MMCSD0_CQ_DEV_PENDING_TASKS register to check if the task is queued in the device.
5	Send CMDQ_TASK_MGMT(CMD48) to discard task using the task id as the argument.
6	Write '1' to CQCTCLR[i] to clear task in CQE.
7	Poll on CQCTCLR[i] until it is '0'.

Copyright © 2024 Texas Instruments Incorporated

Table 12-230. Task Discard and Clear Sequence (continued)

Step	Description
8	Write '0' to MMCSD0_CQ_CONTROL register to resume CQE.

12.3.6.4.2.4.5 Error Detect and Recovery when CQ is enabled

Figure 12-157 and Table 12-231 show an error detect and recovery sequence.

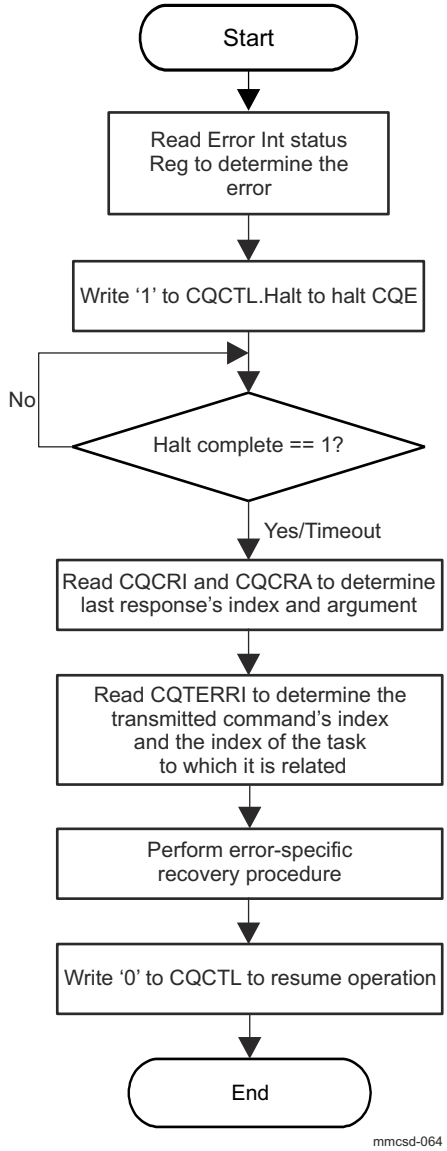


Figure 12-157. Error Detect and Recovery Sequence

Table 12-231. Error Detect and Recovery Sequence

Step	Description
1	Read eMMC host controller MMCSD0_ERROR_INTR_STS register and determine error is related to CQE.
2	Write '1' to MMCSD0_CQ_CONTROL[0] HALT_BIT bit to halt CQE.
3	Wait for MMCSD0_CQ_CONTROL[0] HALT_BIT bit to read '1'. In some error cases, this may not happen, so software should proceed to the next step after a sufficient time-out.
4	Read MMCSD0_CQ_CMD_RESP_INDEX and MMCSD0_CQ_CMD_RESP_ARG registers to determine last response's index and argument.
5	Read MMCSD0_CQ_TASK_ERR_INFO register to determine the transmitted command's index and the index of the task to which it is related.

Copyright © 2024 Texas Instruments Incorporated

Table 12-231. Error Detect and Recovery Sequence (continued)

Step	Description
6	Perform error-specific recovery procedure.
7	Write '0' to MMCSD0_CQ_CONTROL register to resume operation.

Copyright © 2024 Texas Instruments Incorporated

12.4 Industrial and Control Interfaces

This section describes the industrial and control Interfaces in the device.

12.4.1 Enhanced Capture (ECAP) Module

This section describes the Enhanced Capture (ECAP) module in the device.

12.4.1.1 ECAP Overview

The Enhanced Capture (ECAP) module can be used for:

- Sample rate measurements of audio inputs
- Speed measurements of rotating machinery (for example, toothed sprockets sensed via Hall sensors)
- Elapsed time measurements between position sensor pulses
- Period and duty cycle measurements of pulse train signals
- Decoding current or voltage amplitude derived from duty cycle encoded current/voltage sensors

The device has three instances of the ECAP module.

Copyright © 2024 Texas Instruments Incorporated

12.4.1.1.1 ECAP Features

The ECAP module includes the following features:

- 32-bit time base counter
- 4 × 32 bits event time-stamp capture registers (ECAP_CAP1 through ECAP_CAP4)
- 4-stage sequencer (Mod4 counter), synchronized to external events (ECAPx pin edges)
- Independent edge polarity (rising / falling edge) selection for all 4 sequenced time-stamp capture events
- Input capture signal pre-scaling (from 1 to 16)
- One-shot compare register (2 bits) to freeze captures after 1 to 4 time-stamp events
- Continuous mode capture of time-stamps in a four-deep circular buffer
- Interrupt capabilities on any of the 4 capture events
- Absolute time-stamp capture
- Difference (Delta) mode time-stamp capture
- All above resources dedicated to a single input pin
- When not used in capture mode, the ECAP module can be configured as a single channel PWM output

Unsupported Features:

- See the Module Integration section of this document for a list of module features not supported by the integration on this Device.

12.4.1.1.2 ECAP Ports

Table 12-232. ECAP Clocks and Resets

Clocks	
Module Clock Input	**Description**
ECAP_FICLK	ECAP functional and interface clock
Resets	
Module Reset Input	**Description**
ECAP_RST	Module Reset

Table 12-233. ECAP Hardware Requests

Interrupt Requests		
Module Interrupt Signal	**Description**	**Type**
ECAP_ECAP_INT_0	ECAP interrupt	Pulse

12.4.1.2 ECAP Environment

This section describes the ECAP external connections (environment).

Table 12-234 describes the ECAP I/O signals.

Table 12-234. ECAP I/O Signals

Module Pin	I/O[(1)]	Description
ECAP0_CAPIN_APWMOUT	I/O	ECAP0 Capture input / PWM output
ECAP1_CAPIN_APWMOUT	I/O	ECAP1 Capture input / PWM output
ECAP2_CAPIN_APWMOUT	I/O	ECAP2 Capture input / PWM output

(1) I = Input; O = Output

Copyright © 2024 Texas Instruments Incorporated

12.4.1.3 ECAP Functional Description

The ECAP module represents one complete capture channel, which has the following independent key resources:

- Dedicated input capture pin
- 32 events selection mapping capability to the input capture pin
- 32-bit time base counter
- 4 × 32-bit time-stamp capture registers (ECAP_CAP1 through ECAP_CAP4)
- 4-stage sequencer (Mod4 counter) that is synchronized to external events, ECAP pin rising/falling edges.
- Independent edge polarity (rising/falling edge) selection for all 4 events
- Input capture signal prescaling (from 2-62)
- One-shot compare register (2 bits) to freeze captures after 1 to 4 time-stamp events
- Control for continuous time-stamp captures using a 4-deep circular buffer (ECAP_CAP1 through ECAP_CAP4) scheme
- Interrupt capabilities on any of the 4 capture events

Multiple identical ECAP modules can be contained in a system as shown in Figure 12-158. For actual number of the ECAP modules integrated in the device, refer to the *ECAP Integration*. The letter x within a signal or module name is used to indicate a generic ECAP instance on a device. For example, output interrupt request, ECAP1INT belongs to ECAP1, ECAP2INT belongs to ECAP2, and so forth.

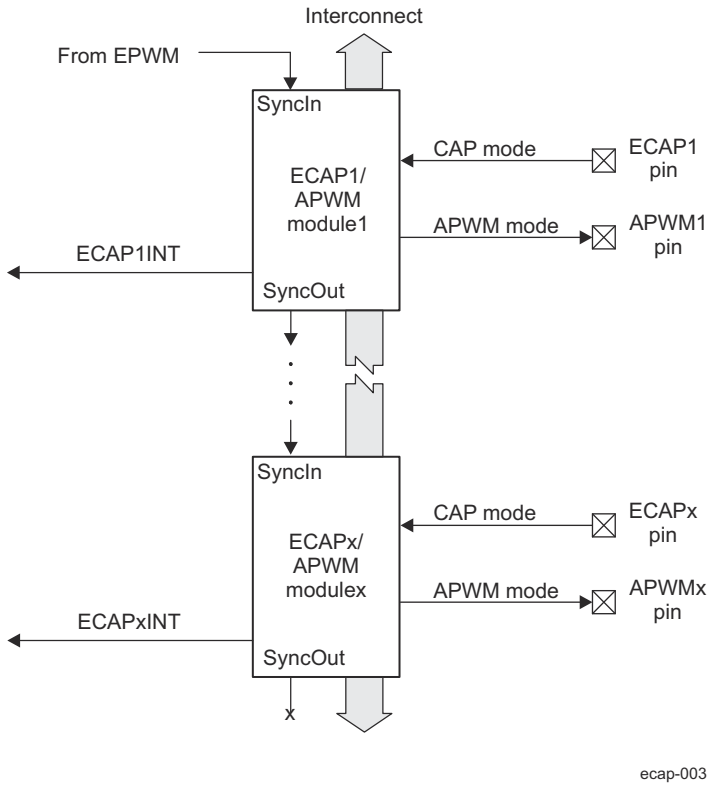
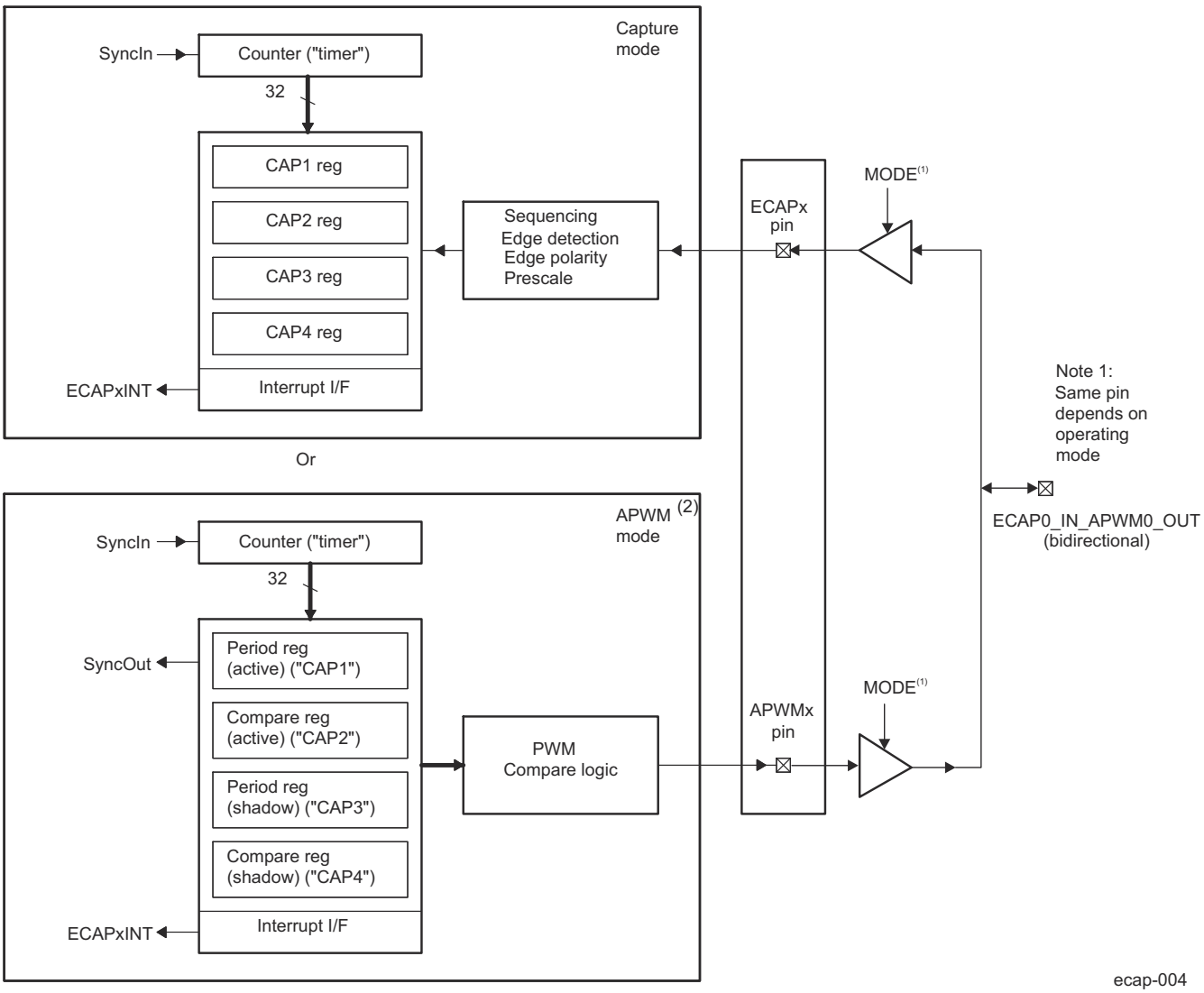


Figure 12-158. Multiple ECAP Modules

12.4.1.3.1 Capture and APWM Operating Modes

The ECAP module resources can be used to implement a single-channel PWM generator (with 32-bit capabilities) when it is not being used for input captures. The counter operates in count-up mode, providing a time-base for asymmetrical pulse width modulation (PWM) waveforms. The ECAP_CAP1 and ECAP_CAP2 registers become the active period and compare registers, respectively, while the ECAP_CAP3 and ECAP_CAP4 registers become the period and capture shadow registers, respectively. Figure 12-159 is a high-level view of both the capture and auxiliary pulse-width modulator (APWM) modes of operation.

Copyright © 2024 Texas Instruments Incorporated



A. A single pin is shared between CAP and APWM functions. In capture mode, it is an input. In APWM mode, it is an output.

B. In APWM mode, writing any value to the ECAP_CAP1/ECAP_CAP2 active registers also writes the same value to the corresponding shadow registers (ECAP_CAP3/ECAP_CAP4). This emulates immediate mode. Writing to the shadow registers (ECAP_CAP3/ECAP_CAP4) invokes the shadow mode.

Figure 12-159. Capture and APWM Modes of Operation

Copyright © 2024 Texas Instruments Incorporated

12.4.1.3.1.1 ECAP Capture Mode Description

Figure 12-160 shows the various components that implement the capture function.

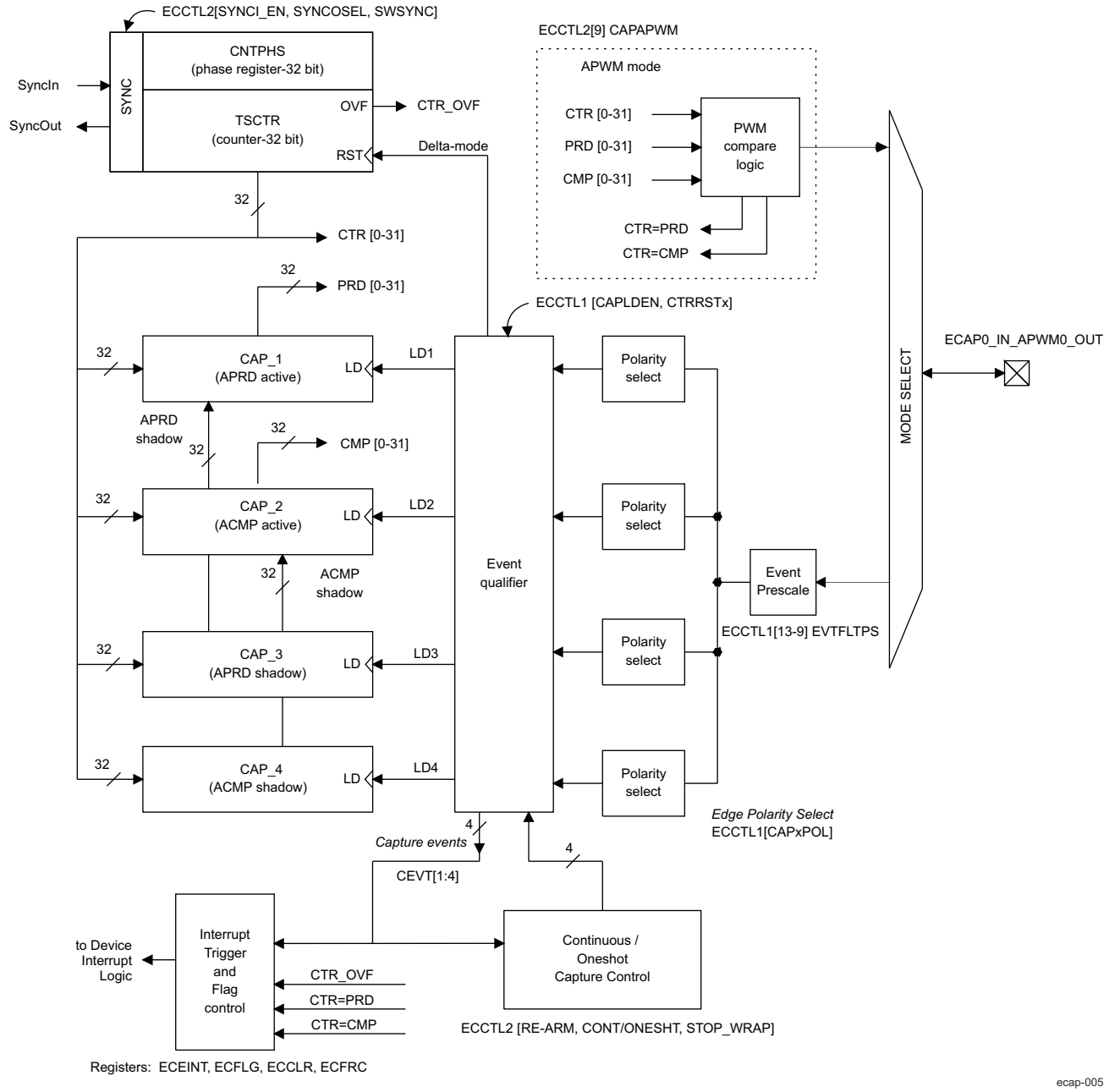
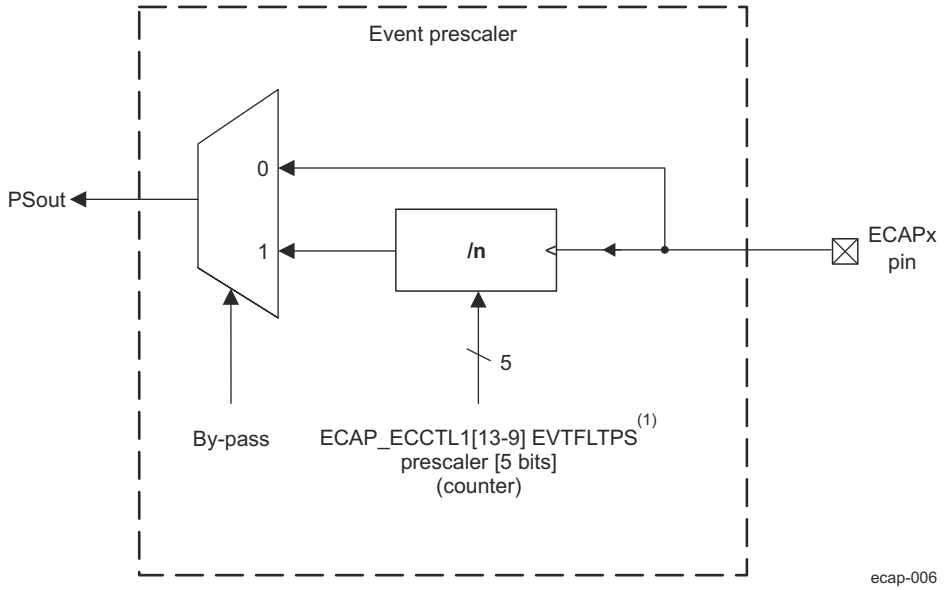


Figure 12-160. Capture Function Diagram

12.4.1.3.1.1.1 ECAP Event Prescaler

An input capture signal (pulse train) can be prescaled by N = 2-62 (in multiples of 2) or can bypass the prescaler. This is useful when very high frequency signals are used as inputs. Figure 12-161 shows a functional diagram and Figure 12-162 shows the operation of the prescale function.

Copyright © 2024 Texas Instruments Incorporated



A. When a prescale value of 1 is chosen (ECAP_ECCTL[13-9] EVTFLTPS = 0b0000) the input capture signal by-passes the prescale logic completely.

Figure 12-161. Event Prescale Control

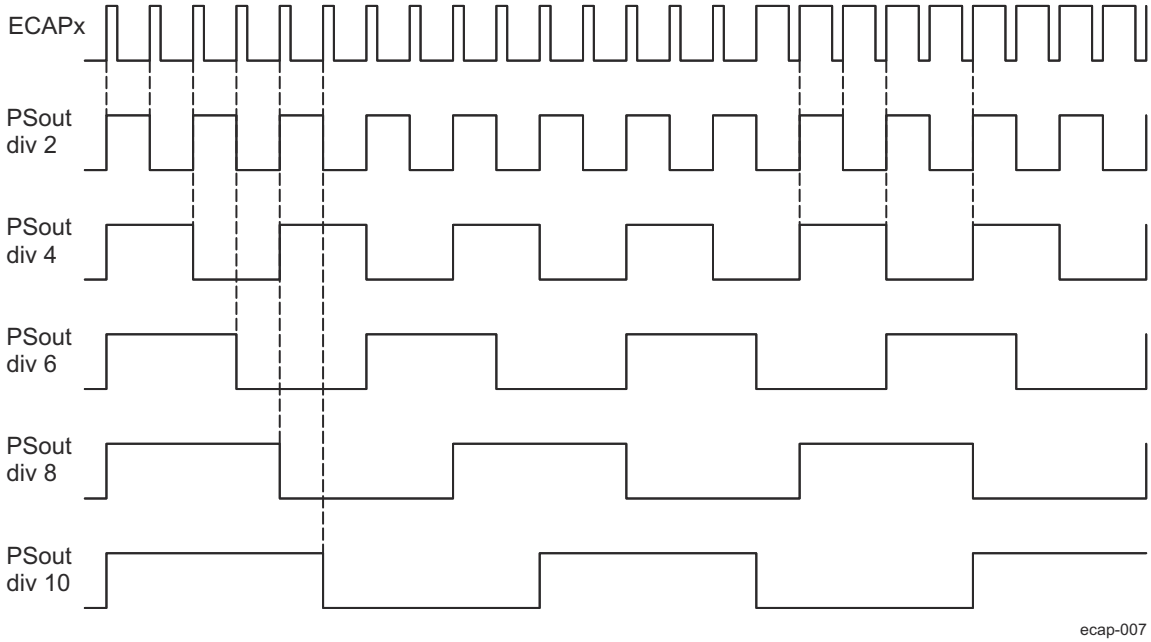


Figure 12-162. Prescale Function Waveforms

Copyright © 2024 Texas Instruments Incorporated

12.4.1.3.1.1.2 ECAP Edge Polarity Select and Qualifier

- Four independent edge polarity (rising edge/falling edge) selection multiplexers are used, one for each capture event.
- Each edge (up to 4) is event qualified by the Mod4 sequencer.
- The edge event is gated to its respective CAP*n* register by the Mod4 counter. The CAP*n* register is loaded on the falling edge.

12.4.1.3.1.1.3 ECAP Continuous/One-Shot Control

- The Mod4 (2-bit) counter is incremented via edge qualified events CEVT1 through CEVT4 (see the ECAP_ECINT_EN_FLG register).
- The Mod4 counter continues counting (0->1->2->3->0) and wraps around unless stopped.
- A 2-bit stop register is used to compare the Mod4 counter output; when equal the Mod4 counter stops and inhibits further loads of the ECAP_CAP1 through ECAP_CAP4 registers. This occurs during one-shot operation.

The continuous/one-shot block (Figure 12-163) controls the start/stop and reset (zero) functions of the Mod4 counter via a mono-shot type of action that can be triggered by the stop-value comparator and re-armed via software control.

Once armed, the ECAP module waits for 1-4 (defined by stop-value) capture events before freezing both the Mod4 counter and contents of the ECAP_CAP1 through ECAP_CAP4 registers (time-stamps).

Re-arming prepares the ECAP module for another capture sequence. Also re-arming clears (to zero) the Mod4 counter and permits loading of the ECAP_CAP1 through ECAP_CAP4 registers again, providing the ECAP_ECCTL[8] CAPLDEN bit is set.

In continuous mode, the Mod4 counter continues to run (0->1->2->3->0), the one-shot action is ignored, and capture values continue to be written to the ECAP_CAP1 through ECAP_CAP4 registers in a circular buffer sequence.

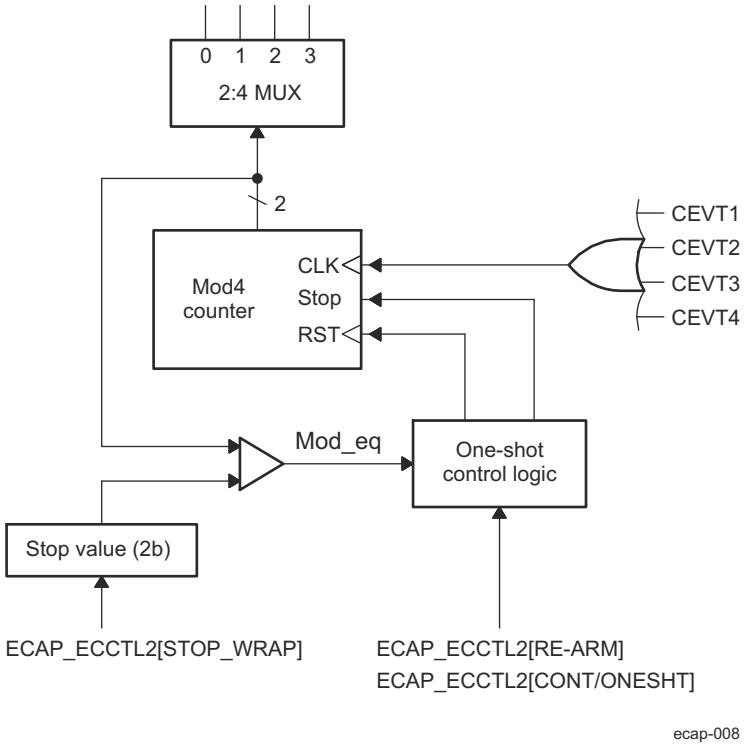


Figure 12-163. ECAP Continuous/One-shot Block Diagram

Copyright © 2024 Texas Instruments Incorporated

12.4.1.3.1.1.4 ECAP 32-Bit Counter and Phase Control

This counter (Figure 12-164) provides the time-base for event captures, and is clocked via the system clock.

A phase register is provided to achieve synchronization with other counters, via a hardware and software forced sync. This is useful in APWM mode when a phase offset between modules is needed.

On any of the four event loads, an option to reset the 32-bit counter is given. This is useful for time difference capture. The 32-bit counter value is captured first, then it is reset to 0 by any of the LD1 through LD4 signals.

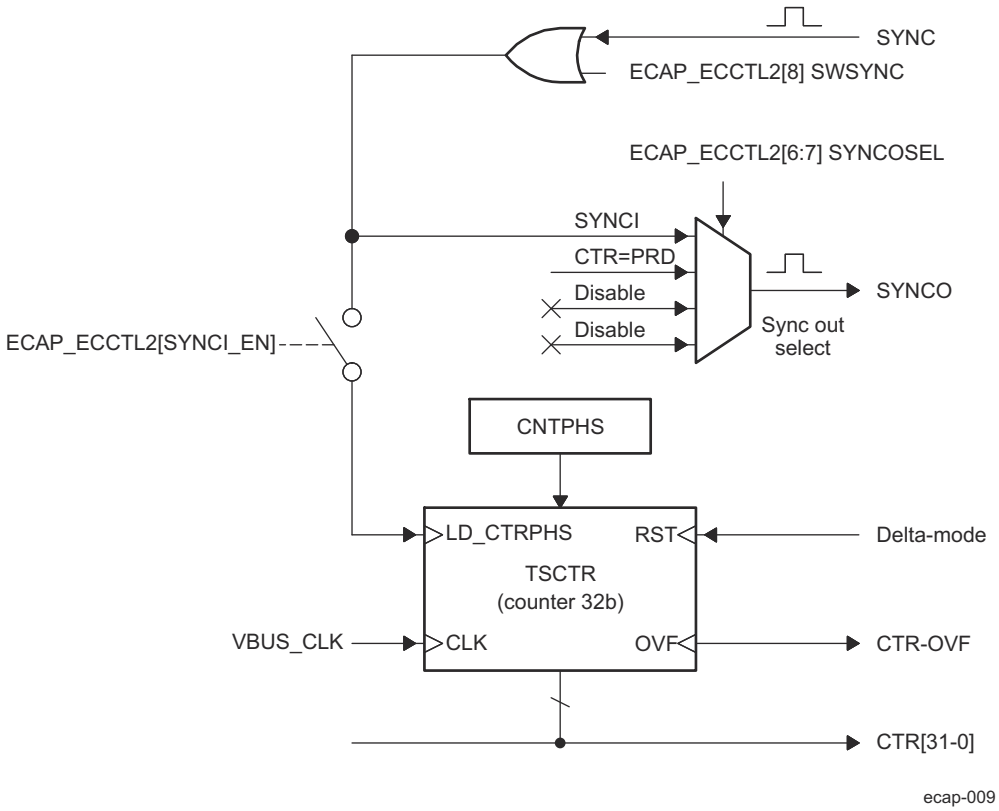


Figure 12-164. ECAP Counter and Synchronization Block Diagram

Copyright © 2024 Texas Instruments Incorporated

12.4.1.3.1.1.5 CAP1-CAP4 Registers

These 32-bit registers are fed by the 32-bit counter timer bus, CTR[0-31] and are loaded (capture a time-stamp) when their respective LD inputs are strobed.

Loading of the capture registers can be inhibited via the control ECAP_ECCTL[8] CAPLDEN bit. During one-shot operation, this bit is cleared (loading is inhibited) automatically when a stop condition occurs, StopValue = Mod4.

The ECAP_CAP1 and ECAP_CAP2 registers become the active period and compare registers, respectively, in APWM mode.

The ECAP_CAP3 and ECAP_CAP4 registers become the respective shadow registers (APRD and ACMP) for the ECAP_CAP1 and ECAP_CAP2 registers during APWM operation.

12.4.1.3.1.1.6 ECAP Interrupt Control

An interrupt can be generated on capture events CEVT1 through CEVT4, CNTOVF (see the ECAP_ECINT_EN_FLG[21] CNTOVF_FLG bit) or APWM events (TSCNT = PRD, TSCNT = CMP). See Figure 12-165.

A counter overflow event (FFFF FFFFh->0000 0000h) is also provided as an interrupt source (CNTOVF).

The capture events are edge and sequencer qualified (that is, ordered in time) by the polarity select and Mod4 gating, respectively.

One of these events can be selected as the interrupt source (from the ECAP*n* module) going to the interrupt controller.

Seven interrupt events (CEVT1, CEVT2, CEVT3, CEVT4, CNTOVF, TSCNT = PRD, TSCNT = CMP) can be generated. The interrupt enable register (ECAP_ECINT_EN_FLG) is used to enable/disable individual interrupt event sources. The interrupt flag register (ECAP_ECINT_EN_FLG) indicates if any interrupt event has been latched and contains the global interrupt flag ECAP_ECINT_EN_FLG[16] INT_FLG bit. An interrupt pulse is generated to the interrupt controller only if any of the interrupt events are enabled, the flag bit is 1h, and the INT_FLG flag bit is 0h. The interrupt service routine must clear the global interrupt flag bit and the serviced event via the interrupt clear register (ECAP_ECINT_CLR_FRC) before any other interrupt pulses are generated. The interrupt force register (ECAP_ECINT_CLR_FRC) can force an interrupt event. This is useful for test purposes.

12.4.1.3.1.1.7 ECAP Shadow Load and Lockout Control

In capture mode, this logic inhibits (locks out) any shadow loading of the ECAP_CAP1 or ECAP_CAP2 registers from APRD and ACMP registers, respectively.

In APWM mode, shadow loading is active and two choices are permitted:

- Immediate - APRD or ACMP are transferred to the ECAP_CAP1 or ECAP_CAP2 register immediately upon writing a new value.
- On period equal, CTR[31-0] = PRD[31-0]

Note

The CEVT1_FLG, CEVT2_FLG, CEVT3_FLG, CEVT4_FLG flags are only active in capture mode (ECAP_ECCTL[25] CAP_APWM == 0h). The TSCNT = PRD, TSCNT = CMP flags are only valid in APWM mode (ECAP_ECCTL[25] CAP_APWM == 1h). CNTOVF_FLG flag is valid in both modes.

Copyright © 2024 Texas Instruments Incorporated

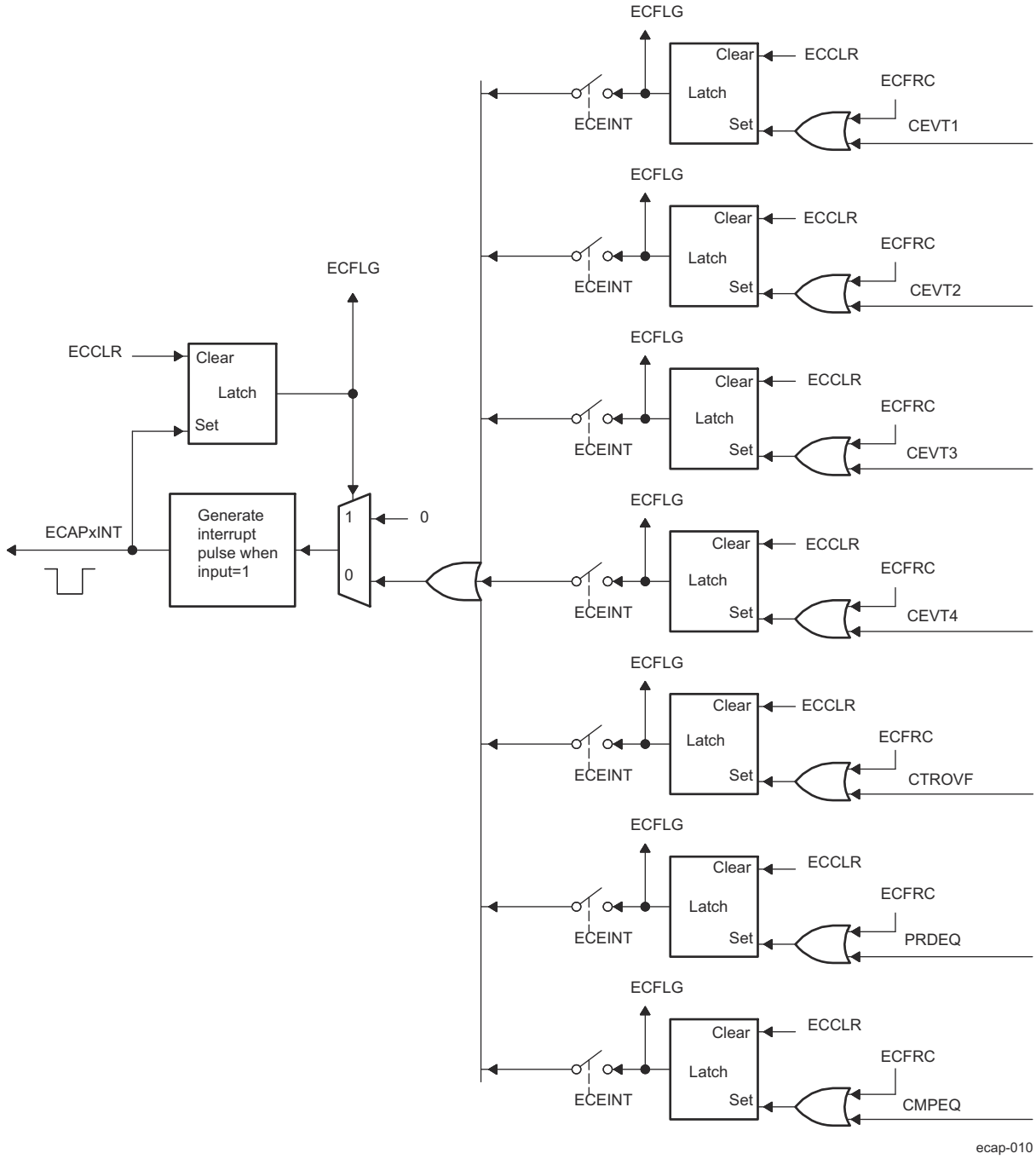


Figure 12-165. Interrupts in ECAP Module

Copyright © 2024 Texas Instruments Incorporated

12.4.1.3.1.2 ECAP APWM Mode Operation

Main operating highlights of the APWM section:

- The time-stamp counter bus is made available for comparison via 2 digital (32-bit) comparators.
- When the ECAP_CAP1/ECAP_CAP2 registers are not used in capture mode, their contents can be used as Period and Compare values in APWM mode.
- Double buffering is achieved via the shadow registers - APRD and ACMP (ECAP_CAP3/ECAP_CAP4). The shadow register contents are transferred over to ECAP_CAP1/ECAP_CAP2 registers either immediately upon a write, or on a TSCNT = PRD trigger.
- In APWM mode, writing to the ECAP_CAP1/ECAP_CAP2 active registers will also write the same value to the corresponding shadow registers (ECAP_CAP3/ECAP_CAP4). This emulates immediate mode. Writing to the shadow registers (ECAP_CAP3/ECAP_CAP4) will invoke the shadow mode.
- During initialization, software must write the PRD and CMP values to the active registers. This automatically copies the initial values into the shadow values. For subsequent compare updates, during run-time, software should use only shadow registers.

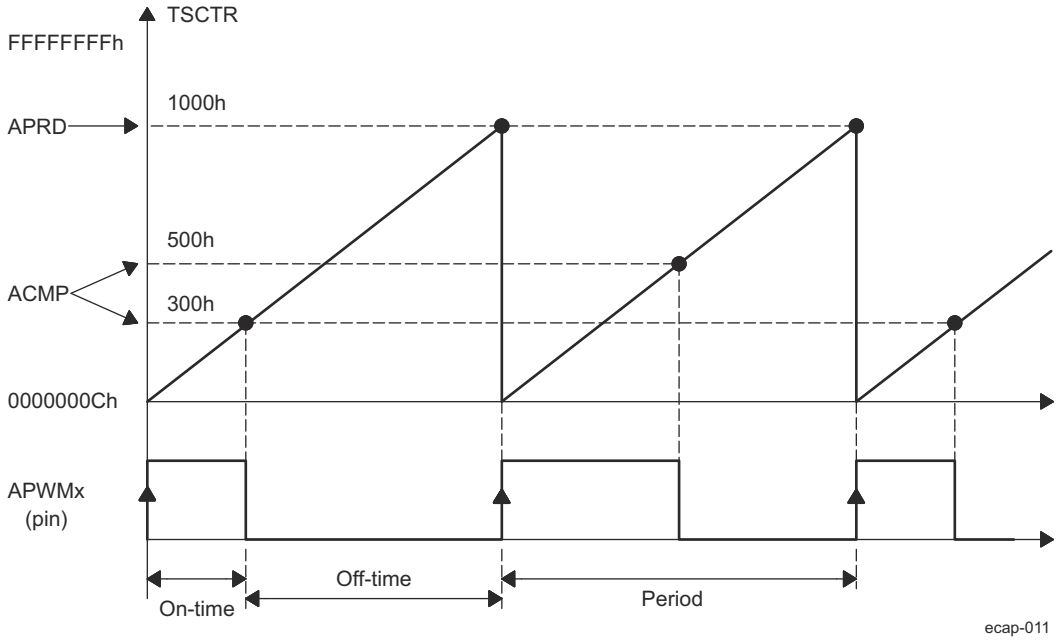


Figure 12-166. PWM Waveform Details of ECAP APWM Mode Operation

The behavior of APWM active-high mode (APWMPOL == 0) is:

```
CMP = 0x00000000, output low for duration of period (0% duty)
```

```
CMP = 0x00000001, output high 1 cycle
```

```
CMP = 0x00000002, output high 2 cycles
```

```
CMP = PERIOD, output high except for 1 cycle (<100% duty)
```

```
CMP = PERIOD+1, output high for complete period (100% duty)
```

```
CMP > PERIOD+1, output high for complete period
```


Copyright © 2024 Texas Instruments Incorporated

The behavior of APWM active-low mode (APWMPOL == 1) is:

```
CMP = 0x00000000, output high for duration of period (0% duty)
```

```
CMP = 0x00000001, output low 1 cycle
```

```
CMP = 0x00000002, output low 2 cycles
```

```
CMP = PERIOD, output low except for 1 cycle (<100% duty)
```

```
CMP = PERIOD+1, output low for complete period (100% duty)
```

```
CMP > PERIOD+1, output low for complete period
```

12.4.1.3.2 Summary of ECAP Functional Registers

Table 12-235 shows the ECAP module control and status register set. All 32-bit registers are aligned on even address boundaries and are organized in little-endian mode.

Note

In APWM mode, writing to the ECAP_CAP1/ECAP_CAP2 active registers also writes the same value to the corresponding shadow registers (ECAP_CAP3/ECAP_CAP4). This emulates immediate mode. Writing to the shadow registers (ECAP_CAP3/ECAP_CAP4) invokes the shadow mode.

Table 12-235. ECAP Control and Status Functional Registers

Offset	Register Name	Description	Size (×16)
0h	ECAP_TSCNT	Time-Stamp Counter Register	2
4h	ECAP_CNTPHS	Counter Phase Offset Value Register	2
8h	ECAP_CAP1	Capture 1 Register	2
Ch	ECAP_CAP2	Capture 2 Register	2
10h	ECAP_CAP3	Capture 3 Register	2
14h	ECAP_CAP4	Capture 4 Register	2
28h	ECAP_ECCTL	Capture Control Register	2
2Ch	ECAP_ECINT_EN_FLG	Capture Interrupt Enable and Flag Register	2
30h	ECAP_ECINT_CLR_FRC	Capture Interrupt Clear and Forcing Register	2
5Ch	ECAP_PID	Revision ID Register	2

Note

For more information on the ECAP registers, see *ECAP Registers.*

Copyright © 2024 Texas Instruments Incorporated

12.4.1.4 ECAP Use Cases

The following sections will provide applications examples and code snippets to show how to configure and operate the ECAP module. For clarity and ease of use, below are useful #defines which will help in the understanding of the examples.

```
// ECCTL1 ( ECAP Control Reg 1)
//==========================
// CAPxPOL bits
#define    EC_RISING              0x0
#define    EC_FALLING             0x1
// CTRRSTx bits
#define    EC_ABS_MODE            0x0
#define    EC_DELTA_MODE          0x1
// PRESCALE bits
#define    EC_BYPASS              0x0
#define    EC_DIV1                0x0
#define    EC_DIV2                0x1
#define    EC_DIV4                0x2
#define    EC_DIV6                0x3
#define    EC_DIV8                0x4
#define    EC_DIV10               0x5
// ECCTL2 ( ECAP Control Reg 2)
//==========================
// CONT/ONESHOT bit
#define    EC_CONTINUOUS          0x0
#define    EC_ONESHOT             0x1
// STOPVALUE bit
#define    EC_EVENT1              0x0
#define    EC_EVENT2              0x1
#define    EC_EVENT3              0x2
#define    EC_EVENT4              0x3
// RE-ARM bit
#define    EC_ARM                 0x1
// TSCTRSTOP bit
#define    EC_FREEZE              0x0
#define    EC_RUN                 0x1
// SYNCO_SEL bit
#define    EC_SYNCIN              0x0
#define    EC_CTR_PRD             0x1
#define    EC_SYNCO_DIS           0x2
// CAP/APWM mode bit
#define    EC_CAP_MODE            0x0
#define    EC_APWM_MODE           0x1
// APWMPOL bit
#define    EC_ACTV_HI             0x0
#define    EC_ACTV_LO             0x1
// Generic
#define    EC_DISABLE             0x0
#define    EC_ENABLE              0x1
#define    EC_FORCE               0x1
```


Copyright © 2024 Texas Instruments Incorporated

12.4.1.4.1 Absolute Time-Stamp Operation Rising Edge Trigger Example

Figure 12-167 shows an example of continuous capture operation (Mod4 counter wraps around). In this figure, TSCTR counts-up without resetting and capture events are qualified on the rising edge only, this gives period (and frequency) information.

On an event, the TSCTR contents (time-stamp) is first captured, then Mod4 counter is incremented to the next state. When the TSCTR reaches FFFF FFFFh (maximum value), it wraps around to 0000 0000h (not shown in Figure 12-167), if this occurs, the CNTOVF_FLG (counter overflow) flag is set, and an interrupt (if enabled) occurs. Captured time-stamps are valid at the point indicated by the diagram, after the 4-th event, hence event CEVT4 can conveniently be used to trigger an interrupt and the CPU can read data from the CAP*n* registers.

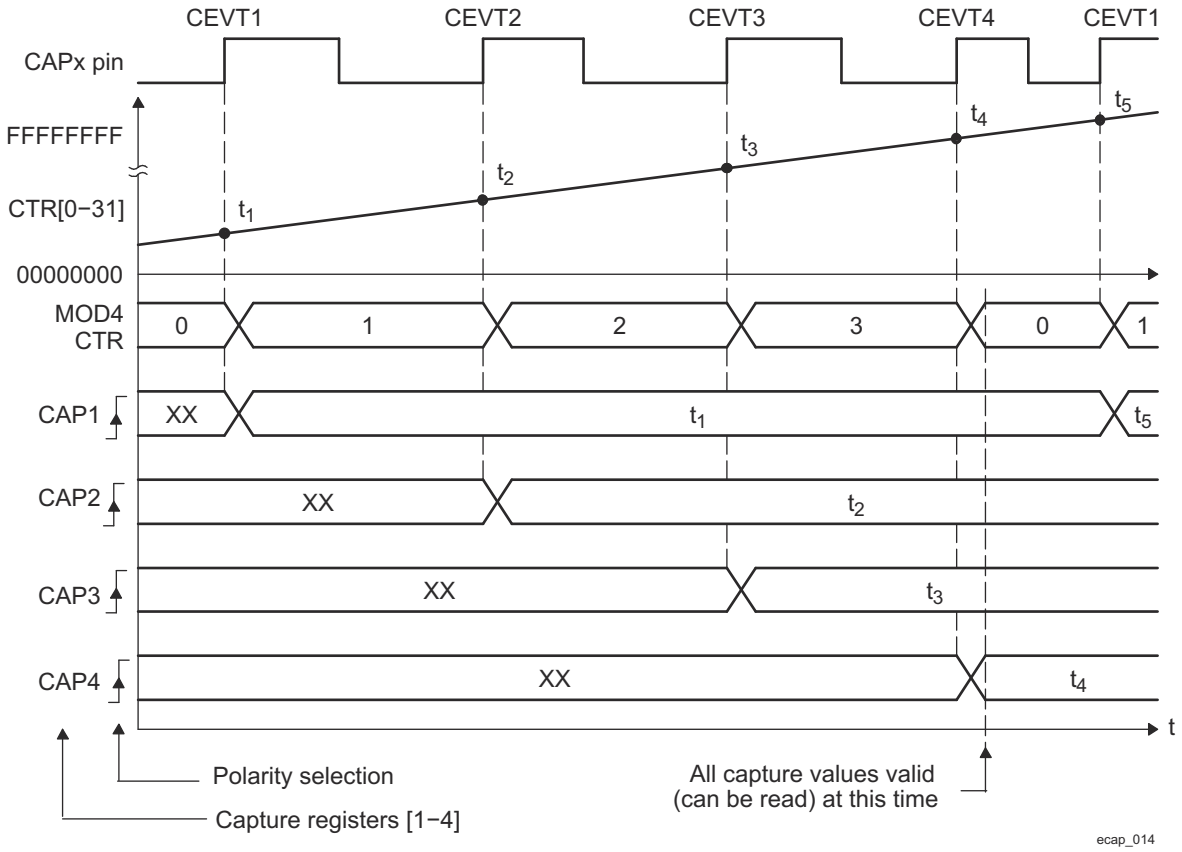


Figure 12-167. Capture Sequence for Absolute Time-Stamp, Rising Edge Detect

Copyright © 2024 Texas Instruments Incorporated

Table 12-236. ECAP Initialization for CAP Mode Absolute Time, Rising Edge Trigger

Register	Bit	Value
ECCTL1	CAP1POL	EC_RISING
ECCTL1	CAP2POL	EC_RISING
ECCTL1	CAP3POL	EC_RISING
ECCTL1	CAP4POL	EC_RISING
ECCTL1	CTRRST1	EC_ABS_MODE
ECCTL1	CTRRST2	EC_ABS_MODE
ECCTL1	CTRRST3	EC_ABS_MODE
ECCTL1	CTRRST4	EC_ABS_MODE
ECCTL1	CAPLDEN	EC_ENABLE
ECCTL1	PRESCALE	EC_DIV1
ECCTL2	CAP_APWM	EC_CAP_MODE
ECCTL2	CONT_ONESHT	EC_CONTINUOUS
ECCTL2	SYNCO_SEL	EC_SYNCO_DIS
ECCTL2	SYNCI_EN	EC_DISABLE
ECCTL2	TSCTRSTOP	EC_RUN

Example 12-1. Code Snippet for CAP Mode Absolute Time, Rising Edge Trigger

```
// Code snippet for CAP mode Absolute Time, Rising edge trigger
// Run Time ( e.g. CEVT4 triggered ISR call)
//=========================================
    TSt1 = ECAPxRegs.CAP1;        // Fetch Time-Stamp captured at t1
    TSt2 = ECAPxRegs.CAP2;        // Fetch Time-Stamp captured at t2
    TSt3 = ECAPxRegs.CAP3;        // Fetch Time-Stamp captured at t3
    TSt4 = ECAPxRegs.CAP4;        // Fetch Time-Stamp captured at t4
    Period1 = TSt2-TSt1;        // Calculate 1st period
    Period2 = TSt3-TSt2;        // Calculate 2nd period
    Period3 = TSt4-TSt3;        // Calculate 3rd period
```


Copyright © 2024 Texas Instruments Incorporated

12.4.1.4.2 Absolute Time-Stamp Operation Rising and Falling Edge Trigger Example

In Figure 12-168 the ECAP operating mode is almost the same as in the previous section except capture events are qualified as either rising or falling edge, this now gives both period and duty cycle information: Period1 = $t_3 – t_1$, Period2 = $t_5 – t_3$, …etc. Duty Cycle1 (on-time %) = $(t_2 – t_1)$ / Period1 x 100%, etc. Duty Cycle1 (off-time %) = $(t_3 – t_2)$ / Period1 x 100%, etc.

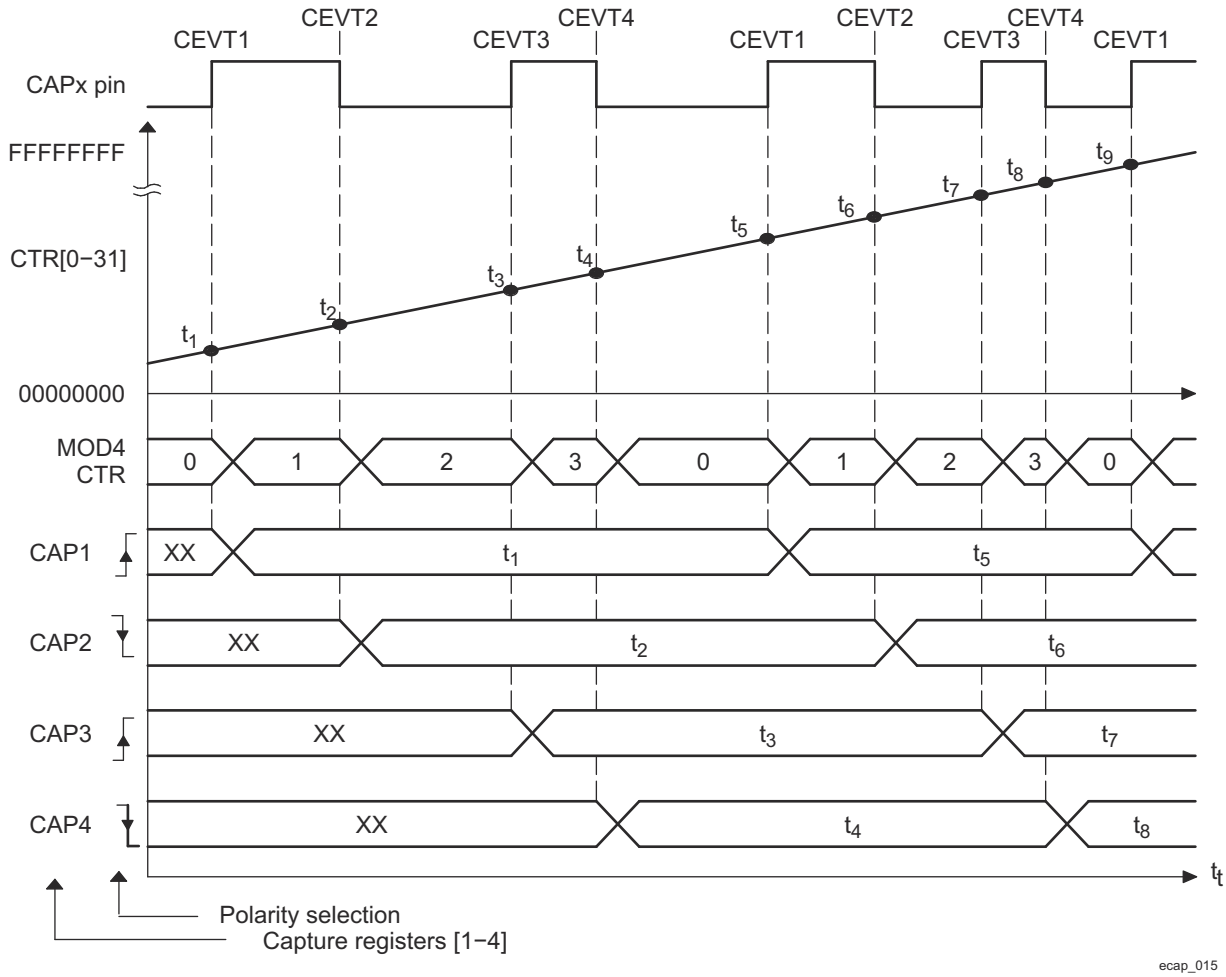


Figure 12-168. Capture Sequence for Absolute Time-Stamp, Rising and Falling Edge Detect

Copyright © 2024 Texas Instruments Incorporated

Table 12-237. ECAP Initialization for CAP Mode Absolute Time, Rising and Falling Edge Trigger

Register	Bit	Value
ECCTL1	CAP1POL	EC_RISING
ECCTL1	CAP2POL	EC_FALLING
ECCTL1	CAP3POL	EC_RISING
ECCTL1	CAP4POL	EC_FALLING
ECCTL1	CTRRST1	EC_ABS_MODE
ECCTL1	CTRRST2	EC_ABS_MODE
ECCTL1	CTRRST3	EC_ABS_MODE
ECCTL1	CTRRST4	EC_ABS_MODE
ECCTL1	CAPLDEN	EC_ENABLE
ECCTL1	PRESCALE	EC_DIV1
ECCTL2	CAP_APWM	EC_CAP_MODE
ECCTL2	CONT_ONESHT	EC_CONTINUOUS
ECCTL2	SYNCO_SEL	EC_SYNCO_DIS
ECCTL2	SYNCI_EN	EC_DISABLE
ECCTL2	TSCTRSTOP	EC_RUN

Example 12-2. Code Snippet for CAP Mode Absolute Time, Rising and Falling Edge Trigger

```
// Code snippet for CAP mode Absolute Time, Rising & Falling edge triggers
// Run Time ( e.g. CEVT4 triggered ISR call)
//=========================================
    TSt1 = ECAPxRegs.CAP1;        // Fetch Time-Stamp captured at t1
    TSt2 = ECAPxRegs.CAP2;        // Fetch Time-Stamp captured at t2
    TSt3 = ECAPxRegs.CAP3;        // Fetch Time-Stamp captured at t3
    TSt4 = ECAPxRegs.CAP4;        // Fetch Time-Stamp captured at t4
    Period1 = TSt3-TSt1;        // Calculate 1st period
    DutyOnTime1 = TSt2-TSt1;    // Calculate On time
    DutyOffTime1 = TSt3-TSt2;    // Calculate Off time
```


Copyright © 2024 Texas Instruments Incorporated

12.4.1.4.3 Time Difference (Delta) Operation Rising Edge Trigger Example

Figure 12-169 shows how the ECAP module can be used to collect Delta timing data from pulse train waveforms. Here Continuous Capture mode (TSCTR counts-up without resetting, and Mod4 counter wraps around) is used. In Delta-time mode, TSCTR is Reset back to Zero on every valid event. Here Capture events are qualified as Rising edge only. On an event, TSCTR contents (time-stamp) is captured first, and then TSCTR is reset to Zero. The Mod4 counter then increments to the next state. If TSCTR reaches FFFF FFFFh (maximum value), before the next event, it wraps around to 0000 0000h and continues, a CNTOVF_FLG (counter overflow) Flag is set, and an Interrupt (if enabled) occurs. The advantage of Delta-time Mode is that the CAP*n* contents directly give timing data without the need for CPU calculations: Period1 = T_1, Period2 = T_2,…etc. As shown in Figure 12-169, the CEVT1 event is a good trigger point to read the timing data, T_1, T_2, T_3, T_4 are all valid here.

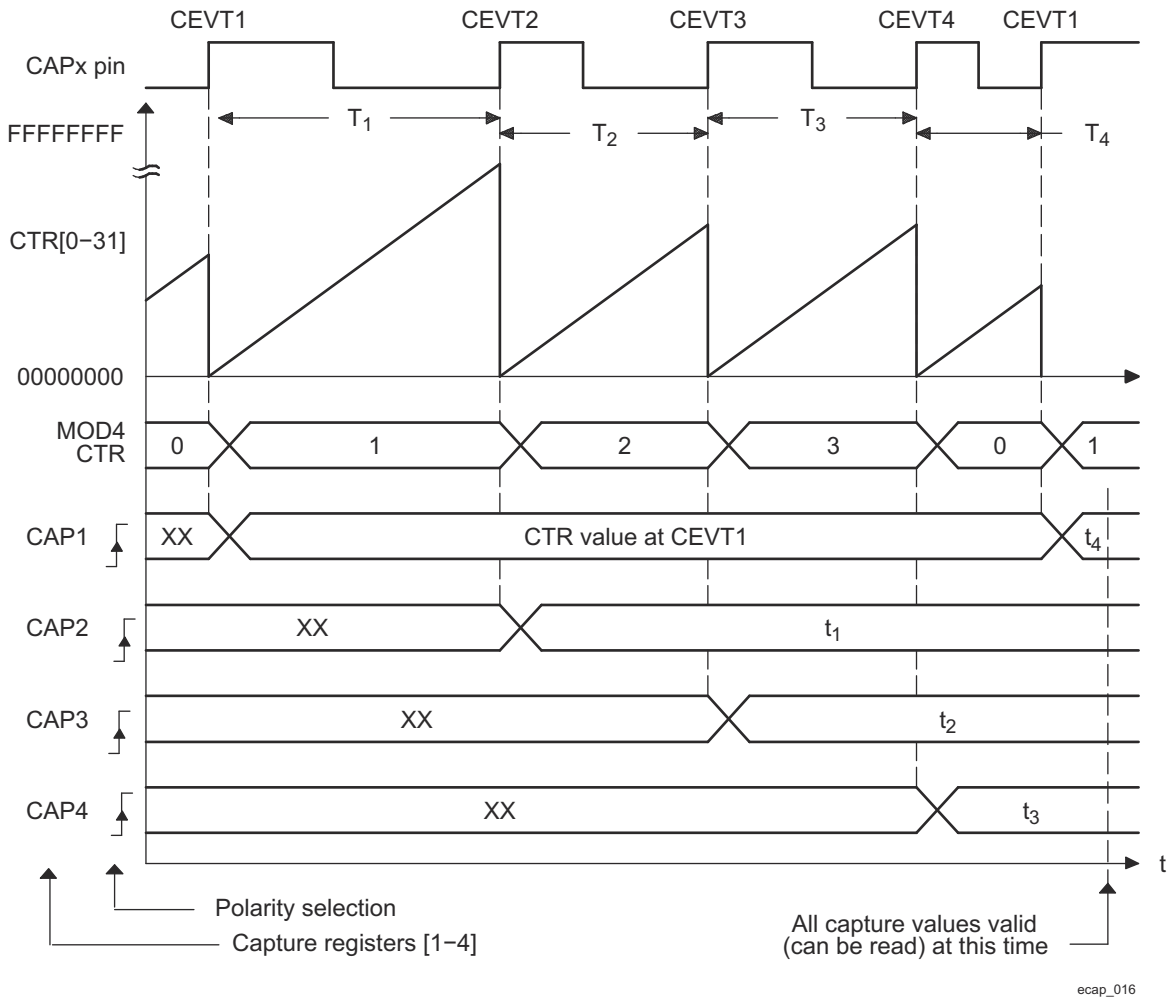


Figure 12-169. Capture Sequence for Delta Mode Time-Stamp, Rising Edge Detect

Copyright © 2024 Texas Instruments Incorporated

Table 12-238. ECAP Initialization for CAP Mode Delta Time, Rising Edge Trigger

Register	Bit	Value
ECCTL1	CAP1POL	EC_RISING
ECCTL1	CAP2POL	EC_RISING
ECCTL1	CAP3POL	EC_RISING
ECCTL1	CAP4POL	EC_RISING
ECCTL1	CTRRST1	EC_DELTA_MODE
ECCTL1	CTRRST2	EC_DELTA_MODE
ECCTL1	CTRRST3	EC_DELTA_MODE
ECCTL1	CTRRST4	EC_DELTA_MODE
ECCTL1	CAPLDEN	EC_ENABLE
ECCTL1	PRESCALE	EC_DIV1
ECCTL2	CAP_APWM	EC_CAP_MODE
ECCTL2	CONT_ONESHT	EC_CONTINUOUS
ECCTL2	SYNCO_SEL	EC_SYNCO_DIS
ECCTL2	SYNCI_EN	EC_DISABLE
ECCTL2	TSCTRSTOP	EC_RUN

Example 12-3. Code Snippet for CAP Mode Delta Time, Rising Edge Trigger

```
// Code snippet for CAP mode Delta Time, Rising edge trigger
// Run Time ( e.g. CEVT1 triggered ISR call)
//=========================================
// Note: here Time-stamp directly represents the Period value.
    Period4 = ECAPxRegs.CAP1;    // Fetch Time-Stamp captured at T1
    Period1 = ECAPxRegs.CAP2;    // Fetch Time-Stamp captured at T2
    Period2 = ECAPxRegs.CAP3;    // Fetch Time-Stamp captured at T3
    Period3 = ECAPxRegs.CAP4;    // Fetch Time-Stamp captured at T4
```


Copyright © 2024 Texas Instruments Incorporated

12.4.1.4.4 Time Difference (Delta) Operation Rising and Falling Edge Trigger Example

In Figure 12-170 the ECAP operating mode is almost the same as in previous section except Capture events are qualified as either Rising or Falling edge, this now gives both Period and Duty cycle information: Period1 = T_1 + T_2, Period2 = T_3 + T_4, …etc Duty Cycle1 (on-time %) = T_1 / Period1 × 100%, etc Duty Cycle1 (off-time %) = T_2 / Period1 × 100%, etc.

During initialization, you must write to the active registers for both period and compare. This will then automatically copy the init values into the shadow values. For subsequent compare updates, that is, during run-time, only the shadow registers must be used.

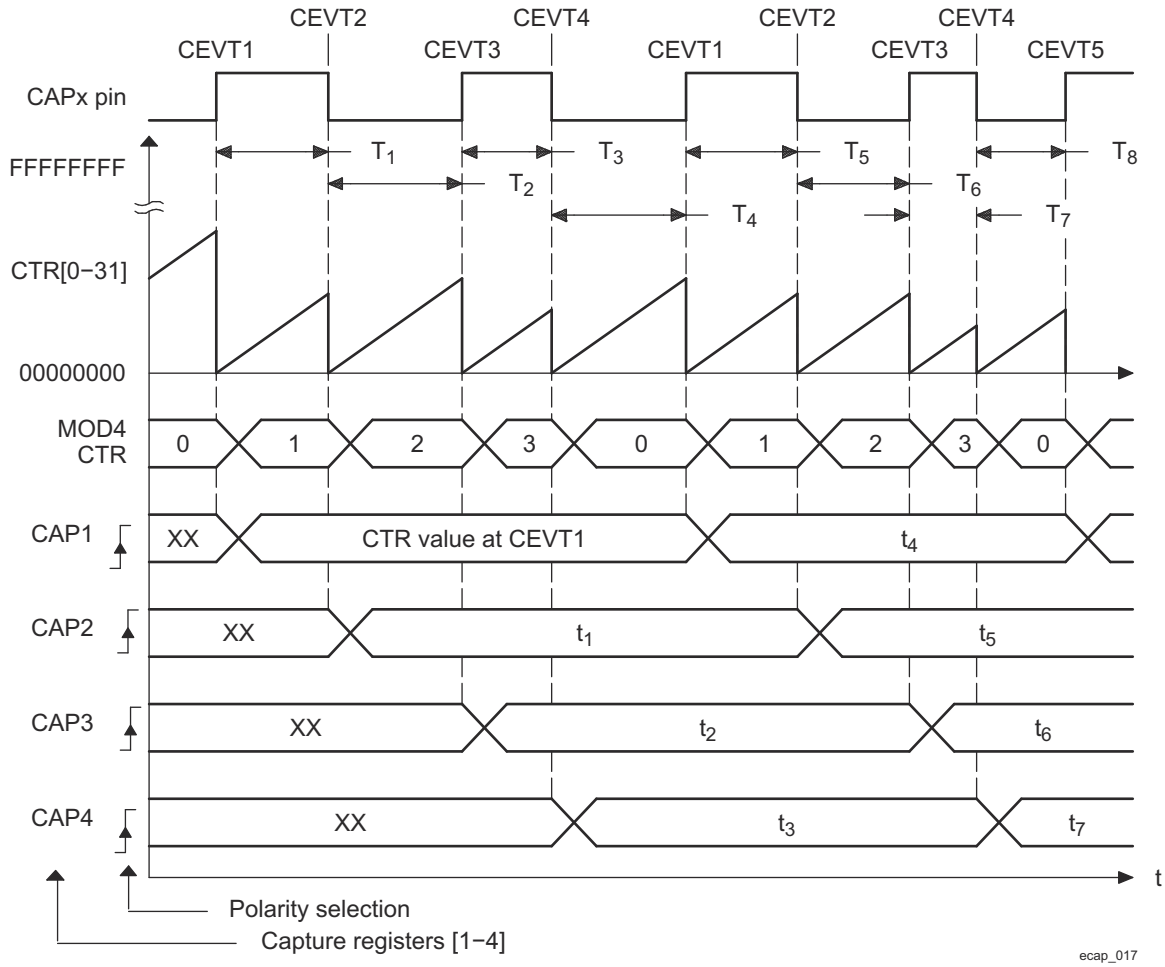


Figure 12-170. Capture Sequence for Delta Mode Time-Stamp, Rising and Falling Edge Detect

Copyright © 2024 Texas Instruments Incorporated

Table 12-239. ECAP Initialization for CAP Mode Delta Time, Rising and Falling Edge Triggers

Register	Bit	Value
ECCTL1	CAP1POL	EC_RISING
ECCTL1	CAP2POL	EC_FALLING
ECCTL1	CAP3POL	EC_RISING
ECCTL1	CAP4POL	EC_FALLING
ECCTL1	CTRRST1	EC_DELTA_MODE
ECCTL1	CTRRST2	EC_DELTA_MODE
ECCTL1	CTRRST3	EC_DELTA_MODE
ECCTL1	CTRRST4	EC_DELTA_MODE
ECCTL1	CAPLDEN	EC_ENABLE
ECCTL1	PRESCALE	EC_DIV1
ECCTL2	CAP_APWM	EC_CAP_MODE
ECCTL2	CONT_ONESHT	EC_CONTINUOUS
ECCTL2	SYNCO_SEL	EC_SYNCO_DIS
ECCTL2	SYNCI_EN	EC_DISABLE
ECCTL2	TSCTRSTOP	EC_RUN

Example 12-4. Code Snippet for CAP Mode Delta Time, Rising and Falling Edge Triggers

```
// Code snippet for CAP mode Delta Time, Rising and Falling edge triggers
// Run Time ( e.g. CEVT1 triggered ISR call)
//=========================================
// Note: here Time-stamp directly represents the Duty cycle values.
    DutyOnTime1 =  ECAPxRegs.CAP2;    // Fetch Time-Stamp captured at T2
    DutyOffTime1 = ECAPxRegs.CAP3;    // Fetch Time-Stamp captured at T3
    DutyOnTime2 =  ECAPxRegs.CAP4;    // Fetch Time-Stamp captured at T4
    DutyOffTime2 = ECAPxRegs.CAP1;    // Fetch Time-Stamp captured at T1
    Period1 = DutyOnTime1 + DutyOffTime1;
    Period2 = DutyOnTime2 + DutyOffTime2;
```


Copyright © 2024 Texas Instruments Incorporated

12.4.1.4.5 Application of the APWM Mode

12.4.1.4.5.1 Simple PWM Generation (Independent Channel/s) Example

In this example, the ECAP module is configured to operate as a PWM generator. Here a very simple single channel PWM waveform is generated from output pin APWM*n*. The PWM polarity is active high, which means that the compare value (CAP2 reg is now a compare register) represents the on-time (high level) of the period. Alternatively, if the APWMPOL bit is configured for active low, then the compare value represents the off-time.

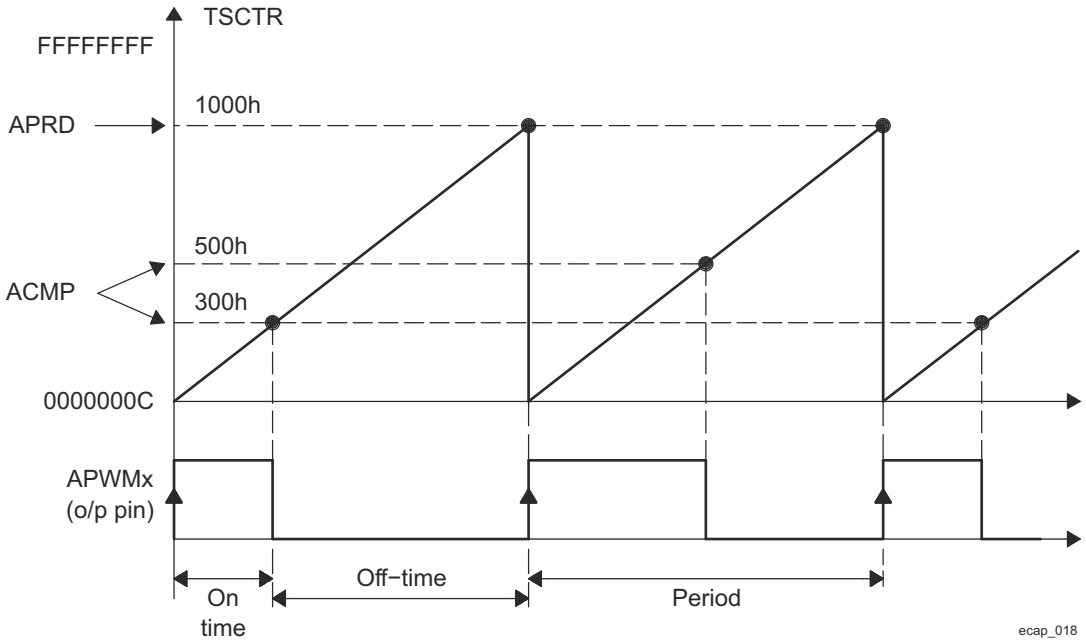


Figure 12-171. PWM Waveform Details of APWM Mode Operation

Copyright © 2024 Texas Instruments Incorporated

Table 12-240. ECAP Initialization for APWM Mode

Register	Bit	Value
CAP1	CAP1	0x1000
CTRPHS	CTRPHS	0x0
ECCTL2	CAP_APWM	EC_APWM_MODE
ECCTL2	APWMPOL	EC_ACTV_HI
ECCTL2	SYNCI_EN	EC_DISABLE
ECCTL2	SYNCO_SEL	EC_SYNCO_DIS
ECCTL2	TSCTRSTOP	EC_RUN

Example 12-5. Code Snippet for APWM Mode

```
// Code snippet for APWM mode Example 1
// Run Time (Instant 1, e.g. ISR call)
//=====================
   ECAPxRegs.CAP2 = 0x300;      // Set Duty cycle i.e. compare value
// Run Time (Instant 2, e.g. another ISR call)
//=====================
   ECAPxRegs.CAP2 = 0x500;      // Set Duty cycle i.e. compare value
```

12.4.1.4.5.2 Multichannel PWM Generation with Synchronization Example

Figure 12-172 takes advantage of the synchronization feature between the ECAP modules. Here 4 independent PWM channels are required with different frequencies, but at integer multiples of each other to avoid "beat" frequencies. Hence one ECAP module is configured as the Master and the remaining 3 are Slaves all receiving their synch pulse (CTR = PRD) from the master. Note the Master is chosen to have the lower frequency (F1 = 1/20,000) requirement. Here Slave2 Freq = 2 × F1, Slave3 Freq = 4 × F1 and Slave4 Freq = 5 × F1. Note here values are in decimal notation. Also, only the APWM1 output waveform is shown.

Copyright © 2024 Texas Instruments Incorporated

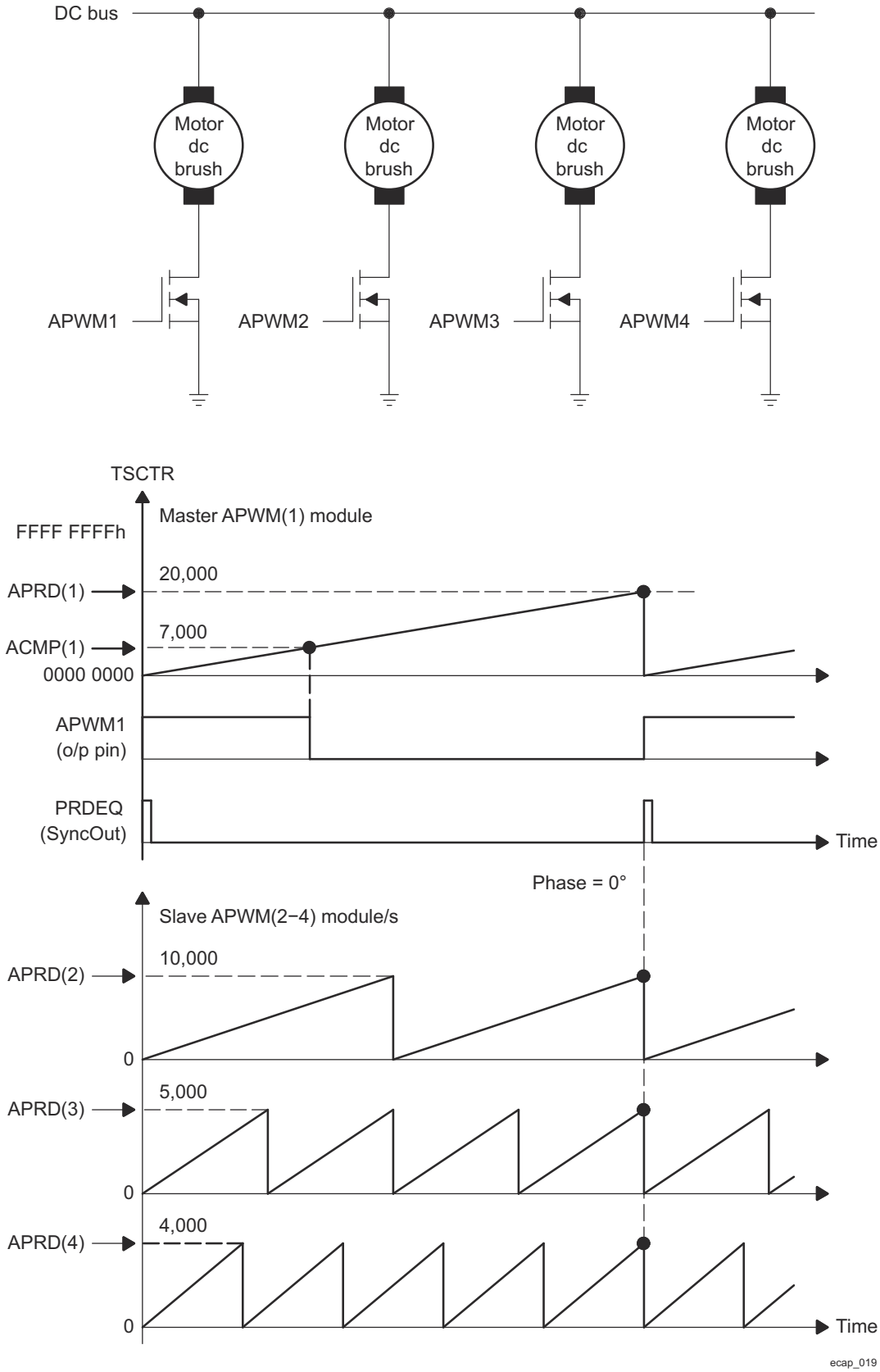


Figure 12-172. Multichannel PWM Example Using 4 ECAP Modules

Copyright © 2024 Texas Instruments Incorporated

Table 12-241. ECAP1 Initialization for Multichannel PWM Generation with Synchronization

Register	Bit	Value
CAP1	CAP1	20000
CTRPHS	CTRPHS	0
ECCTL2	CAP_APWM	EC_APWM_MODE
ECCTL2	APWMPOL	EC_ACTV_HI
ECCTL2	SYNCI_EN	EC_DISABLE
ECCTL2	SYNCO_SEL	EC_CTR_PRD
ECCTL2	TSCTRSTOP	EC_RUN

Table 12-242. ECAP2 Initialization for Multichannel PWM Generation with Synchronization

Register	Bit	Value
CAP1	CAP1	10000
CTRPHS	CTRPHS	0
ECCTL2	CAP_APWM	EC_APWM_MODE
ECCTL2	APWMPOL	EC_ACTV_HI
ECCTL2	SYNCI_EN	EC_ENABLE
ECCTL2	SYNCO_SEL	EC_SYNCI
ECCTL2	TSCTRSTOP	EC_RUN

Table 12-243. ECAP3 Initialization for Multichannel PWM Generation with Synchronization

Register	Bit	Value
CAP1	CAP1	5000
CTRPHS	CTRPHS	0
ECCTL2	CAP_APWM	EC_APWM_MODE
ECCTL2	APWMPOL	EC_ACTV_HI
ECCTL2	SYNCI_EN	EC_ENABLE
ECCTL2	SYNCO_SEL	EC_SYNCI
ECCTL2	TSCTRSTOP	EC_RUN

Table 12-244. ECAP4 Initialization for Multichannel PWM Generation with Synchronization

Register	Bit	Value
CAP1	CAP1	4000
CTRPHS	CTRPHS	0
ECCTL2	CAP_APWM	EC_APWM_MODE
ECCTL2	APWMPOL	EC_ACTV_HI
ECCTL2	SYNCI_EN	EC_ENABLE
ECCTL2	SYNCO_SEL	EC_SYNCO_DIS
ECCTL2	TSCTRSTOP	EC_RUN

Example 12-6. Code Snippet for Multichannel PWM Generation with Synchronization

```
// Code snippet for APWM mode Example 2
// Run Time (Note: Example execution of one run-time instant)
//==========================================================
   ECAP1Regs.CAP2 = 7000;    // Set Duty cycle i.e., compare value = 7000
   ECAP2Regs.CAP2 = 2000;    // Set Duty cycle i.e., compare value = 2000
   ECAP3Regs.CAP2 = 550;     // Set Duty cycle i.e., compare value = 550
   ECAP4Regs.CAP2 = 6500;    // Set Duty cycle i.e., compare value = 6500
```


Copyright © 2024 Texas Instruments Incorporated

12.4.1.4.5.3 Multichannel PWM Generation with Phase Control Example

In Figure 12-173, the Phase control feature of the APWM mode is used to control a 3 phase Interleaved DC/DC converter topology. This topology requires each phase to be off-set by 120° from each other. Hence if "Leg" 1 (controlled by APWM1) is the reference Leg (or phase), that is, 0°, then Leg 2 need 120° off-set and Leg 3 needs 240° off-set. The waveforms in Figure 12-173 show the timing relationship between each of the phases (Legs). Note ECAP1 module is the Master and issues a SyncOut pulse to the slaves (modules 2, 3) whenever TSCTR = Period value.

Copyright © 2024 Texas Instruments Incorporated

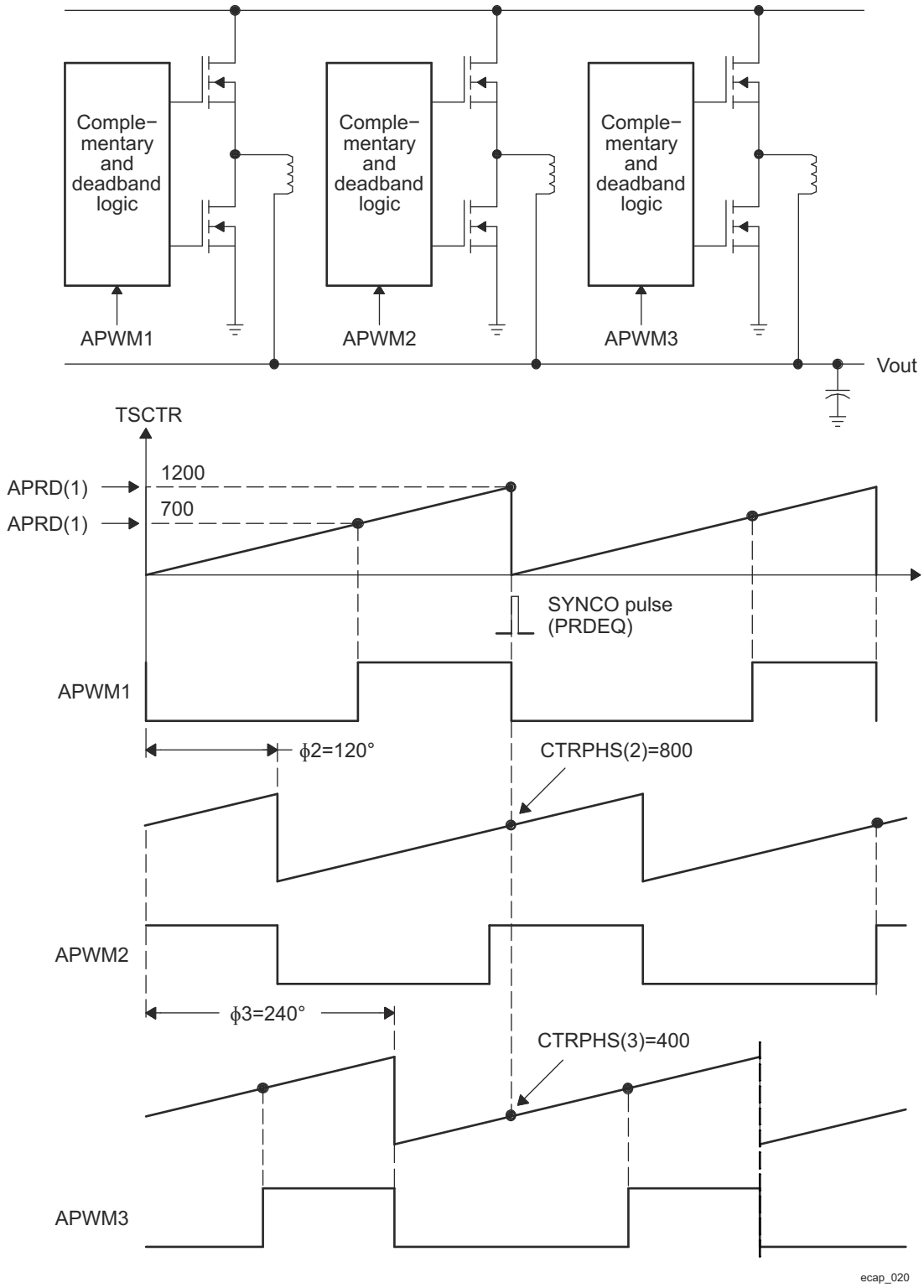


Figure 12-173. Multiphase (channel) Interleaved PWM Example Using 3 ECAP Modules

Copyright © 2024 Texas Instruments Incorporated

Table 12-245. ECAP1 Initialization for Multichannel PWM Generation with Phase Control

Register	Bit	Value
CAP1	CAP1	1200
CTRPHS	CTRPHS	0
ECCTL2	CAP_APWM	EC_APWM_MODE
ECCTL2	APWMPOL	EC_ACTV_HI
ECCTL2	SYNCI_EN	EC_DISABLE
ECCTL2	SYNCO_SEL	EC_CTR_PRD
ECCTL2	TSCTRSTOP	EC_RUN

Table 12-246. ECAP2 Initialization for Multichannel PWM Generation with Phase Control

Register	Bit	Value
CAP1	CAP1	1200
CTRPHS	CTRPHS	800
ECCTL2	CAP_APWM	EC_APWM_MODE
ECCTL2	APWMPOL	EC_ACTV_HI
ECCTL2	SYNCI_EN	EC_ENABLE
ECCTL2	SYNCO_SEL	EC_SYNCI
ECCTL2	TSCTRSTOP	EC_RUN

Table 12-247. ECAP3 Initialization for Multichannel PWM Generation with Phase Control

Register	Bit	Value
CAP1	CAP1	1200
CTRPHS	CTRPHS	400
ECCTL2	CAP_APWM	EC_APWM_MODE
ECCTL2	APWMPOL	EC_ACTV_HI
ECCTL2	SYNCI_EN	EC_ENABLE
ECCTL2	SYNCO_SEL	EC_SYNCO_DIS
ECCTL2	TSCTRSTOP	EC_RUN

Example 12-7. Code Snippet for Multichannel PWM Generation with Phase Control

```
// Code snippet for APWM mode Example 3
// Run Time (Note: Example execution of one run-time instant)
//==========================================================
// All phases are set to the same duty cycle
  ECAP1Regs.CAP2 = 700;     // Set Duty cycle i.e. compare value = 700
  ECAP2Regs.CAP2 = 700;     // Set Duty cycle i.e. compare value = 700
  ECAP3Regs.CAP2 = 700;     // Set Duty cycle i.e. compare value = 700
```

12.4.2 Enhanced Pulse Width Modulation (EPWM) Module

This chapter describes the Enhanced PWM (EPWM) module in the device.

12.4.2.1 EPWM Overview

An effective PWM peripheral must be able to generate complex pulse width waveforms with minimal CPU overhead or intervention. It needs to be highly programmable and very flexible while being easy to understand and use. The EPWM unit described here addresses these requirements by allocating all needed timing and control resources on a per PWM channel basis. Cross coupling or sharing of resources has been avoided; instead, the EPWM is built up from smaller single channel modules with separate resources and that can operate together as required to form a system. This modular approach results in an orthogonal architecture and provides a more transparent view of the peripheral structure, helping users to understand its operation quickly.

In the further description the letter x within a signal or module name is used to indicate a generic EPWM instance on a device. For example, output signals EPWMxA and EPWMxB refer to the output signals from the EPWMx

Copyright © 2024 Texas Instruments Incorporated

instance. Thus, EPWM1A and EPWM1B belong to EPWM1, EPWM2A and EPWM2B belong to EPWM2, and so forth.

The EPWM module represents one complete PWM channel composed of two PWM outputs: EPWMxA and EPWMxB. A given EPWM module functionality can be extended with the so called **High-Resolution Pulse Width Modulator**. Refer to the *EPWM Integration*, to determine which EPWM instances include the HRPWM feature. The HRPWM functionalities are described in Section 12.4.2.3.9.

As also described in *Daisy-Chain Connectivity between EPWM Modules*, the EPWM modules are chained together via a clock synchronization scheme that allows them to operate as a single system when required. Additionally, the EPWM integration allows this synchronization scheme to be extended to the capture peripheral modules (ECAP). The number of modules is device-dependent and based on target application needs. Modules can also operate stand-alone.

The device has six instances of the EPWM modules.

12.4.2.1.1 EPWM Features

Each EPWM module supports the following features:

- Dedicated 16-bit time-base counter with period and frequency control
- Two PWM outputs (EPWMxA and EPWMxB) that can be used in the following configurations:
  - Two independent PWM outputs with single-edge operation
  - Two independent PWM outputs with dual-edge symmetric operation
  - One independent PWM output with dual-edge asymmetric operation
- Asynchronous override control of PWM signals through software
- Programmable phase-control support for lag or lead operation relative to other EPWM modules
- Hardware-locked (synchronized) phase relationship on a cycle-by-cycle basis
- Dead-band generation with independent rising and falling edge delay control
- Programmable trip zone allocation of both cycle-by-cycle trip and one-shot trip on fault conditions
- A trip condition can force either high, low, or high-impedance state logic levels at PWM outputs
- Allows events to trigger both CPU interrupts and ADC start of conversions
- Programmable event prescaling minimizes CPU overhead on interrupts
- PWM chopping by high-frequency carrier signal, useful for pulse transformer gate drives
- High-resolution module with programmable delay line:
  - Programmable on a per PWM period basis
  - Can be inserted either on the rising edge or falling edge of the PWM pulse or both or not at all.

Unsupported Features:

- See the Module Integration section of this document for a list of module features not supported by the integration on this Device.

Each EPWM module is connected to the input/output signals shown in Figure 12-174. The signals are described in detail in subsequent sections.

The order in which the EPWM modules are connected may differ from what is shown in Figure 12-174. See *Daisy-Chain Connectivity between EPWM Modules* for the actual synchronization scheme implemented in the device. Each EPWM module consists of eight submodules and is connected within a system via the signals shown in Figure 12-175.

Copyright © 2024 Texas Instruments Incorporated

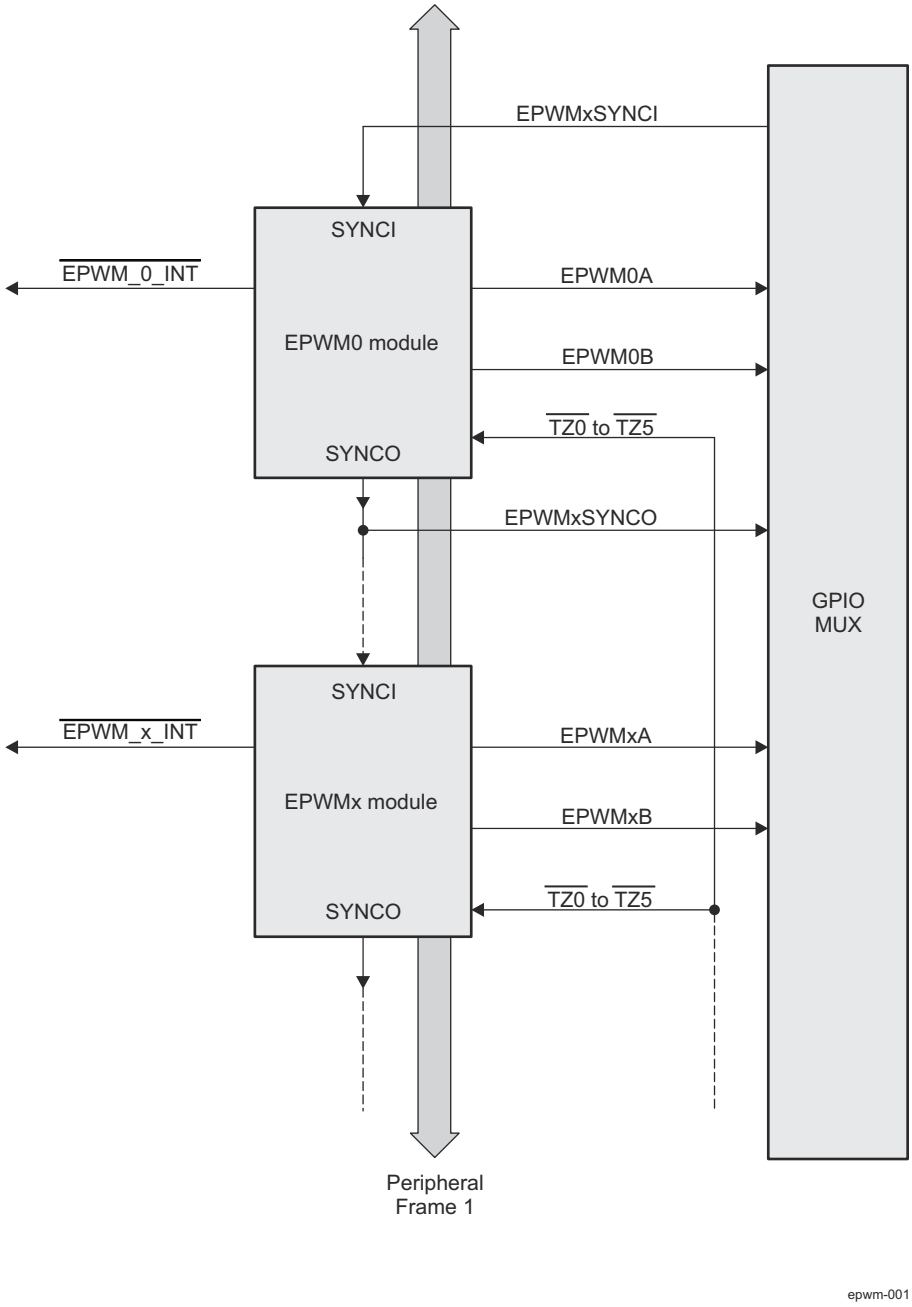


Figure 12-174. Multiple EPWM Modules

Copyright © 2024 Texas Instruments Incorporated

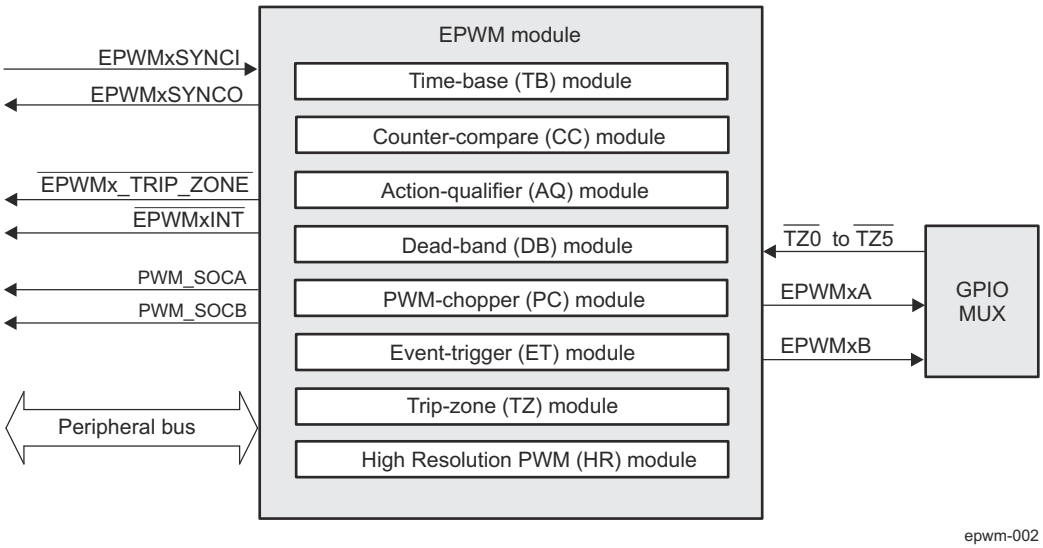


Figure 12-175. Submodules and Signal Connections for an EPWM Module

Figure 12-176 shows more internal details of a single EPWM module. The main signals used by the EPWM module are:

- **PWM output signals (EPWMxA and EPWMxB).** The PWM output signals are made available external to the device through the GPIO peripheral described in the system control and interrupts guide for the device.
- **Trip-zone signals ($\overline{TZ0}$ to $\overline{TZ5}$).** These input signals alert the EPWM module of an external fault condition. Each module on a device can be configured to either use or ignore any of the trip-zone signals. The trip-zone signal can be configured as an asynchronous input through the GPIO peripheral.
- **Time-base synchronization input (EPWMxSYNCI) and output (EPWMxSYNCO) signals.** The synchronization signals daisy chain the EPWM modules together. Each module can be configured to either use or ignore its synchronization input. The clock synchronization input and output signal are brought out to pins for EPWM0 (EPWM module 0) and EPWM3. The EPWM5 synchronization output (EPWM5SYNCO) is also connected to the input SYNCIN of the Enhanced Capture Module (ECAP0).
- **ADC start-of-conversion signals (EPWMxSOCA and EPWMxSOCB).** Each EPWM module has two ADC start of conversion signals (one for each sequencer). Any EPWM module can trigger a start of conversion for either sequencer. Which event triggers the start of conversion is configured in the Event-Trigger submodule of the EPWM.
- **Peripheral Bus.** The peripheral bus is 32-bits wide and allows both 16-bit and 32-bit writes to the EPWM register file.

Figure 12-176 also shows the key internal submodule interconnect signals. Each submodule is described in Section 12.4.2.3.

Copyright © 2024 Texas Instruments Incorporated

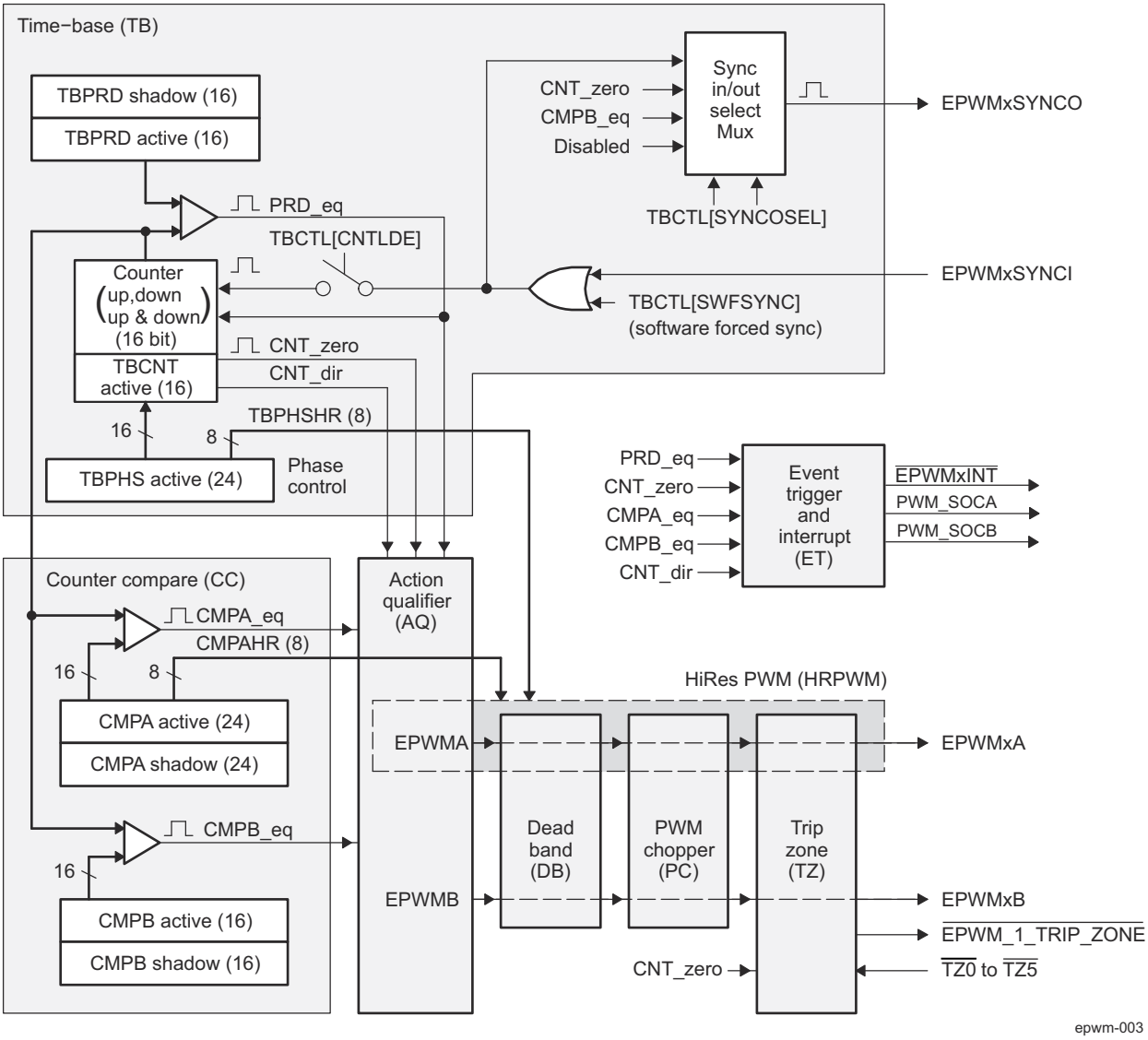


Figure 12-176. EPWM Submodules and Critical Internal Signal Interconnects

12.4.2.1.2 EPWM Ports

Table 12-248. EPWM Clocks and Resets

Clocks	
Module Clock Input	**Description**
EPWM_FICLK	EPWM functional and interface clock
Resets	
Module Reset Input	**Description**
EPWM_RST	Module Reset

Table 12-249. EPWM Hardware Requests

Interrupt Requests		
Module Interrupt Signal	**Description**	**Type**
EHRPWM_EPWM_ETINT_0	EPWM interrupt	Pulse
EHRPWM_EPWM_TRIPZINT_0	EPWM Tripzone interrupt	Pulse
GLUELOGIC_SOCA_INT_GLUE_SOCA_INT_0	EPWM Start of Conversion A event	Pulse
GLUELOGIC_SOCB_INT_GLUE_SOCB_INT_0	EPWM Start of Conversion B event	Pulse

Copyright © 2024 Texas Instruments Incorporated

12.4.2.2 ECAP Environment

This section describes the EPWM external connections (environment).

Table 12-250 describes the EPWM I/O signals.

Table 12-250. EPWM Subsystems I/O Signals

Module Pin	I/O [(1)]	Description
EPWM0A	O	EPWM0 output A
EPWM0B	O	EPWM0 output B
EPWM0SYNCI	I	EPWM0 Sync input
EPWM0SYNCO	O	EPWM0 Sync output
$\overline{\text{EPWM0_TRIP_TZ[0]}}$	I	EPWM0 TripZone input
$\overline{\text{EPWM1_TRIP_TZ[1]}}$	I	EPWM1 TripZone input
$\overline{\text{EPWM2_TRIP_TZ[2]}}$	I	EPWM2 TripZone input
$\overline{\text{EPWM3_TRIP_TZ[3]}}$	I	EPWM3 TripZone input
$\overline{\text{EPWM4_TRIP_TZ[4]}}$	I	EPWM4 TripZone input
$\overline{\text{EPWM5_TRIP_TZ[5]}}$	I	EPWM5 TripZone input
EPWM1A	O	EPWM1 output A
EPWM1B	O	EPWM1 output B
$\overline{\text{EPWM0_TRIP_TZ[0]}}$	I	EPWM0 TripZone input
$\overline{\text{EPWM1_TRIP_TZ[1]}}$	I	EPWM1 TripZone input
$\overline{\text{EPWM2_TRIP_TZ[2]}}$	I	EPWM2 TripZone input
$\overline{\text{EPWM3_TRIP_TZ[3]}}$	I	EPWM3 TripZone input
$\overline{\text{EPWM4_TRIP_TZ[4]}}$	I	EPWM4 TripZone input
$\overline{\text{EPWM5_TRIP_TZ[5]}}$	I	EPWM5 TripZone input
EPWM2A	O	EPWM2 output A
EPWM2B	O	EPWM2 output B
$\overline{\text{EPWM0_TRIP_TZ[0]}}$	I	EPWM0 TripZone input
$\overline{\text{EPWM1_TRIP_TZ[1]}}$	I	EPWM1 TripZone input
$\overline{\text{EPWM2_TRIP_TZ[2]}}$	I	EPWM2 TripZone input
$\overline{\text{EPWM3_TRIP_TZ[3]}}$	I	EPWM3 TripZone input
$\overline{\text{EPWM4_TRIP_TZ[4]}}$	I	EPWM4 TripZone input
$\overline{\text{EPWM5_TRIP_TZ[5]}}$	I	EPWM5 TripZone input
EPWM3A	O	EPWM3 output A
EPWM3B	O	EPWM3 output B
EPWM3SYNCI	I	EPWM3 Sync input
EPWM3SYNCO	O	EPWM3 Sync output
$\overline{\text{EPWM0_TRIP_TZ[0]}}$	I	EPWM0 TripZone input
$\overline{\text{EPWM1_TRIP_TZ[1]}}$	I	EPWM1 TripZone input
$\overline{\text{EPWM2_TRIP_TZ[2]}}$	I	EPWM2 TripZone input
$\overline{\text{EPWM3_TRIP_TZ[3]}}$	I	EPWM3 TripZone input
$\overline{\text{EPWM4_TRIP_TZ[4]}}$	I	EPWM4 TripZone input
$\overline{\text{EPWM5_TRIP_TZ[5]}}$	I	EPWM5 TripZone input
EPWM4A	O	EPWM4 output A
EPWM4B	O	EPWM4 output B
$\overline{\text{EPWM0_TRIP_TZ[0]}}$	I	EPWM0 TripZone input
$\overline{\text{EPWM1_TRIP_TZ[1]}}$	I	EPWM1 TripZone input
$\overline{\text{EPWM2_TRIP_TZ[2]}}$	I	EPWM2 TripZone input
$\overline{\text{EPWM3_TRIP_TZ[3]}}$	I	EPWM3 TripZone input
$\overline{\text{EPWM4_TRIP_TZ[4]}}$	I	EPWM4 TripZone input

Copyright © 2024 Texas Instruments Incorporated

Table 12-250. EPWM Subsystems I/O Signals (continued)

Module Pin	I/O [1]	Description
$\overline{\text{EPWM5_TRIP_TZ[5]}}$	I	EPWM5 TripZone input
EPWM5A	O	EPWM5 output A
EPWM5B	O	EPWM5 output B
$\overline{\text{EPWM0_TRIP_TZ[0]}}$	I	EPWM0 TripZone input
$\overline{\text{EPWM1_TRIP_TZ[1]}}$	I	EPWM1 TripZone input
$\overline{\text{EPWM2_TRIP_TZ[2]}}$	I	EPWM2 TripZone input
$\overline{\text{EPWM3_TRIP_TZ[3]}}$	I	EPWM3 TripZone input
$\overline{\text{EPWM4_TRIP_TZ[4]}}$	I	EPWM4 TripZone input
$\overline{\text{EPWM5_TRIP_TZ[5]}}$	I	EPWM5 TripZone input
PWM_SOCA	O	EPWM start of conversion output A
PWM_SOCB	O	EPWM start of conversion output B

(1) I = Input; O = Output

Copyright © 2024 Texas Instruments Incorporated

12.4.2.3 EPWM Functional Description

8 submodules are included in each EPWM peripheral. Each of these submodules performs specific tasks that can be configured by software.

12.4.2.3.1 EPWM Submodule Features

Table 12-251 lists the eight key submodules together with a list of their main configuration parameters.

Table 12-251. Submodule Configuration Parameters

Submodule	Configuration Parameter or Option
Time-base (TB)	• Scale the time-base clock (TBCLK) relative to the system clock (FICLK). • Configure the PWM time-base counter (TBCNT) frequency or period. • Time-base counter mode selection: – count-up mode: used for asymmetric PWM – count-down mode: used for asymmetric PWM – count-up-and-down mode: used for symmetric PWM • Configure the time-base phase relative to another EPWM module. • Synchronize the time-base counter between modules through hardware or software. • Configure the direction (up or down) of the time-base counter after a synchronization event. • Configure how the time-base counter will behave when the device is halted by an emulator. • Specify the source for the synchronization output of the EPWM module: – Synchronization input signal – Time-base counter equal to zero – Time-base counter equal to counter-compare B (CMPB) – No output synchronization signal generated
Counter-compare (CC)	• Specify the PWM duty cycle for output EPWMxA and/or output EPWMxB • Specify the time at which switching events occur on the EPWMxA or EPWMxB output
Action-qualifier (AQ)	• Specify the type of action taken when a time-base or counter-compare submodule event occurs: – No action taken – Output EPWMxA and/or EPWMxB switched high – Output EPWMxA and/or EPWMxB switched low – Output EPWMxA and/or EPWMxB toggled • Force the PWM output state through software control • Configure and control the PWM dead-band through software
Dead-band (DB)	• Control of traditional complementary dead-band relationship between upper and lower switches • Specify the output rising-edge-delay value • Specify the output falling-edge delay value • Bypass the dead-band module entirely. In this case the PWM waveform is passed through without modification
PWM-chopper (PC)	• Create a chopping (carrier) frequency • Pulse width of the first pulse in the chopped pulse train • Duty cycle of the second and subsequent pulses • Bypass the PWM-chopper module entirely. In this case the PWM waveform is passed through without modification.

Copyright © 2024 Texas Instruments Incorporated

Table 12-251. Submodule Configuration Parameters (continued)

Submodule	Configuration Parameter or Option
Trip-zone (TZ)	• Configure the EPWM module to react to one, all, or none of the trip-zone pins • Specify the tripping action taken when a fault occurs: – Force EPWMxA and/or EPWMxB high – Force EPWMxA and/or EPWMxB low – Force EPWMxA and/or EPWMxB to a high-impedance state – Configure EPWMxA and/or EPWMxB to ignore any trip condition • Configure how often the EPWM will react to the trip-zone pin: – One-shot – Cycle-by-cycle • Enable the trip-zone to initiate an interrupt • Bypass the trip-zone module entirely
Event-trigger (ET)	• Enable the EPWM events that will trigger an interrupt. • Specify the rate at which events cause triggers (every occurrence or every second or third occurrence) • Poll, set, or clear event flags
High-Resolution PWM (HRPWM)	• Enable extended time resolution capabilities • Configure finer time granularity control or edge positioning

Some examples on various EPWM module configurations are shown below. These examples use the constant definitions shown in Example 12-8..

Constant Definitions Used in the EPWM Code Examples

```
// TBCTL (Time-Base Control)
// = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
// TBCNT MODE bits
#define         TB_COUNT_UP        0x0
#define         TB_COUNT_DOWN      0x1
#define         TB_COUNT_UPDOWN    0x2
#define         TB_FREEZE          0x3
// PHSEN bit
#define         TB_DISABLE         0x0
#define         TB_ENABLE          0x1
// PRDLD bit
#define         TB_SHADOW          0x0
#define         TB_IMMEDIATE       0x1
// SYNCOSEL bits
#define         TB_SYNC_IN         0x0
#define         TB_CTR_ZERO        0x1
#define         TB_CTR_CMPB        0x2
#define         TB_SYNC_DISABLE    0x3
// HSPCLKDIV and CLKDIV bits
#define         TB_DIV1            0x0
#define         TB_DIV2            0x1
#define         TB_DIV4            0x2
// PHSDIR bit
#define         TB_DOWN            0x0
#define         TB_UP              0x1
```

```
// CMPCTL (Compare Control)
// = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
// LOADAMODE and LOADBMODE bits
#define         CC_CTR_ZERO        0x0
```


Copyright © 2024 Texas Instruments Incorporated

```
#define         CC_CTR_PRD          0x1
#define         CC_CTR_ZERO_PRD     0x2
#define         CC_LD_DISABLE       0x3
// SHDWAMODE and SHDWBMODE bits
#define         CC_SHADOW           0x0
#define         CC_IMMEDIATE        0x1
// AQCTLA and AQCTLB (Action-qualifier Control)
// = = = = = = = = = = = = = = = = = = = = = = = = = = = =
// ZRO, PRD, CAU, CAD, CBU, CBD bits
#define         AQ_NO_ACTION        0x0
#define         AQ_CLEAR            0x1
#define         AQ_SET              0x2
#define         AQ_TOGGLE           0x3
// DBCTL (Dead-Band Control)
// = = = = = = = = = = = = = = = = = = = = = = = = = = = =
// MODE bits
#define         DB_DISABLE          0x0
#define         DBA_ENABLE          0x1
#define         DBB_ENABLE          0x2
#define DB_FULL_ENABLE 0x3
// POLSEL bits
#define         DB_ACTV_HI          0x0
#define         DB_ACTV_LOC         0x1
#define         DB_ACTV_HIC         0x2
#define         DB_ACTV_LO          0x3
// PCCTL (chopper control)
// = = = = = = = = = = = = = = = = = = = = = = = = = = = =
// CHPEN bit
#define         CHP_ENABLE          0x0
#define CHP_DISABLE 0x1
// CHPFREQ bits
#define         CHP_DIV1            0x0
#define         CHP_DIV2            0x1
#define         CHP_DIV3            0x2
#define         CHP_DIV4            0x3
#define         CHP_DIV5            0x4
#define         CHP_DIV6            0x5
#define         CHP_DIV7            0x6
#define         CHP_DIV8            0x7
// CHPDUTY bits
#define         CHP1_8TH            0x0
#define         CHP2_8TH            0x1
#define         CHP3_8TH            0x2
#define         CHP4_8TH            0x3
#define         CHP5_8TH            0x4
#define         CHP6_8TH            0x5
#define         CHP7_8TH            0x6
// TZSEL (Trip-zone Select)
// = = = = = = = = = = = = = = = = = = = = = = = = = = = =
// CBCn and OSHTn bits
#define         TZ_ENABLE           0x0
#define         TZ_DISABLE          0x1
// TZCTL (Trip-zone Control)
// = = = = = = = = = = = = = = = = = = = = = = = = = = = =
// TZA and TZB bits
#define         TZ_HIZ              0x0
#define         TZ_FORCE_HI         0x1
#define         TZ_FORCE_LO         0x2
#define         TZ_DISABLE          0x3
// ETSEL (Event-trigger Select)
// = = = = = = = = = = = = = = = = = = = = = = = = = = = =
// INTSEL, SOCASEL, SOCBSEL bits
#define         ET_CTR_ZERO         0x1
#define         ET_CTR_PRD          0x2
#define         ET_CTRU_CMPA        0x4
#define         ET_CTRD_CMPA        0x5
#define         ET_CTRU_CMPB        0x6
#define         ET_CTRD_CMPB        0x7
// ETPS (Event-trigger Prescale)
// = = = = = = = = = = = = = = = = = = = = = = = = = = = =
// INTPRD, SOCAPRD, SOCBPRD bits
#define         ET_DISABLE          0x0
#define         ET_1ST              0x1
#define         ET_2ND              0x2
#define         ET_3RD              0x3
```


Copyright © 2024 Texas Instruments Incorporated

12.4.2.3.2 EPWM Time-Base (TB) Submodule

This section describes the Time-Base (TB) submodule in the PWM module.

12.4.2.3.2.1 Overview

Each EPWM module has its own time-base submodule that determines all of the timing events. Built-in synchronization logic allows the time-base of multiple EPWM modules (EPWMx) to work together as a single system. Figure 12-177 illustrates the time-base module's place within the EPWM.

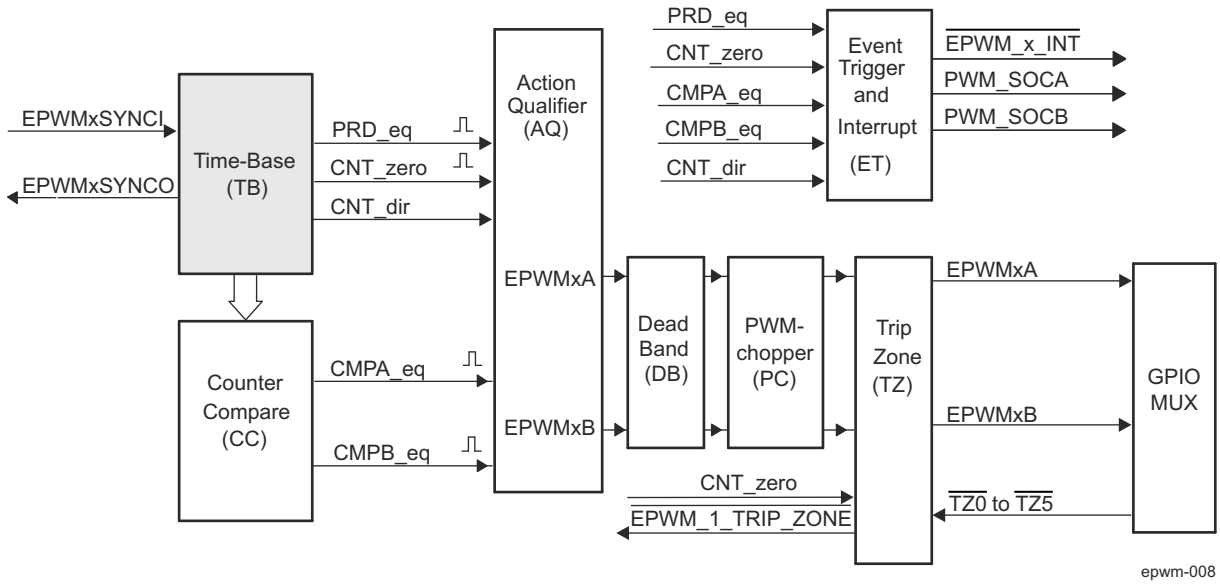


Figure 12-177. EPWM Time-Base Submodule

The TB module generates the period (or frequency) of the PWM output waveforms and consists of a 16-bit counter that can be configured for up, down or up and down counting. The time base for the counter is a pre-scaled version of the system clock (FICLK/n) which can be programmed for a pre-scale of 1, that is same as system clock.

TB module features:

- Specify the EPWMx time-base counter (TBCNT) frequency or period in the EPWM_TBCNT register to control how often events occur.
- Manage time-base synchronization with other EPWMx modules.
- Maintain a phase relationship with other EPWMx modules.
- Set the time-base counter to count-up, count-down, or count-up-and-down mode.
- Generate the following events:
  - PRD_eq (Time-base counter (EPWM_TBCNT register) equal to the specified period in EPWM_TBPRD register (that is TBCNT = TBPRD)).
  - CNT_zero (Time-base counter equal to zero (TBCNT = 0000h)).
- Configure the rate of the time-base clock; a prescaled version of the CPU m clock (FICLK). This allows the time-base counter to increment/decrement at a slower rate.

12.4.2.3.2.2 Controlling and Monitoring the EPWM Time-Base Submodule

Table 12-252 lists the registers used to control and monitor the time-base submodule.

Table 12-252. EPWM Time-Base Submodule Registers

Acronym	Register Description	Address Offset	Shadowed
EPWM_TBCTL	Time-Base Control Register	0h	No
EPWM_TBSTS	Time-Base Status Register	2h	No
HRPWM_TBPHSHR	HRPWM extension Phase Register [(1)]	4h	No
EPWM_TBPHS	Time-Base Phase Register	6h	No

Copyright © 2024 Texas Instruments Incorporated

Table 12-252. EPWM Time-Base Submodule Registers (continued)

Acronym	Register Description	Address Offset	Shadowed
EPWM_TBCNT	Time-Base Counter Register	8h	No
EPWM_TBPRD	Time-Base Period Register	Ah	Yes

(1) This register is available only on EPWM instances that include the high-resolution extension (HRPWM). On EPWM modules that do not include the HRPWM, this location is reserved. See *EPWM Integration* to determine which EPWM instances include this feature.

Figure 12-178 shows the critical signals and registers of the time-base submodule. Table 12-253 provides descriptions of the key signals associated with the time-base submodule.

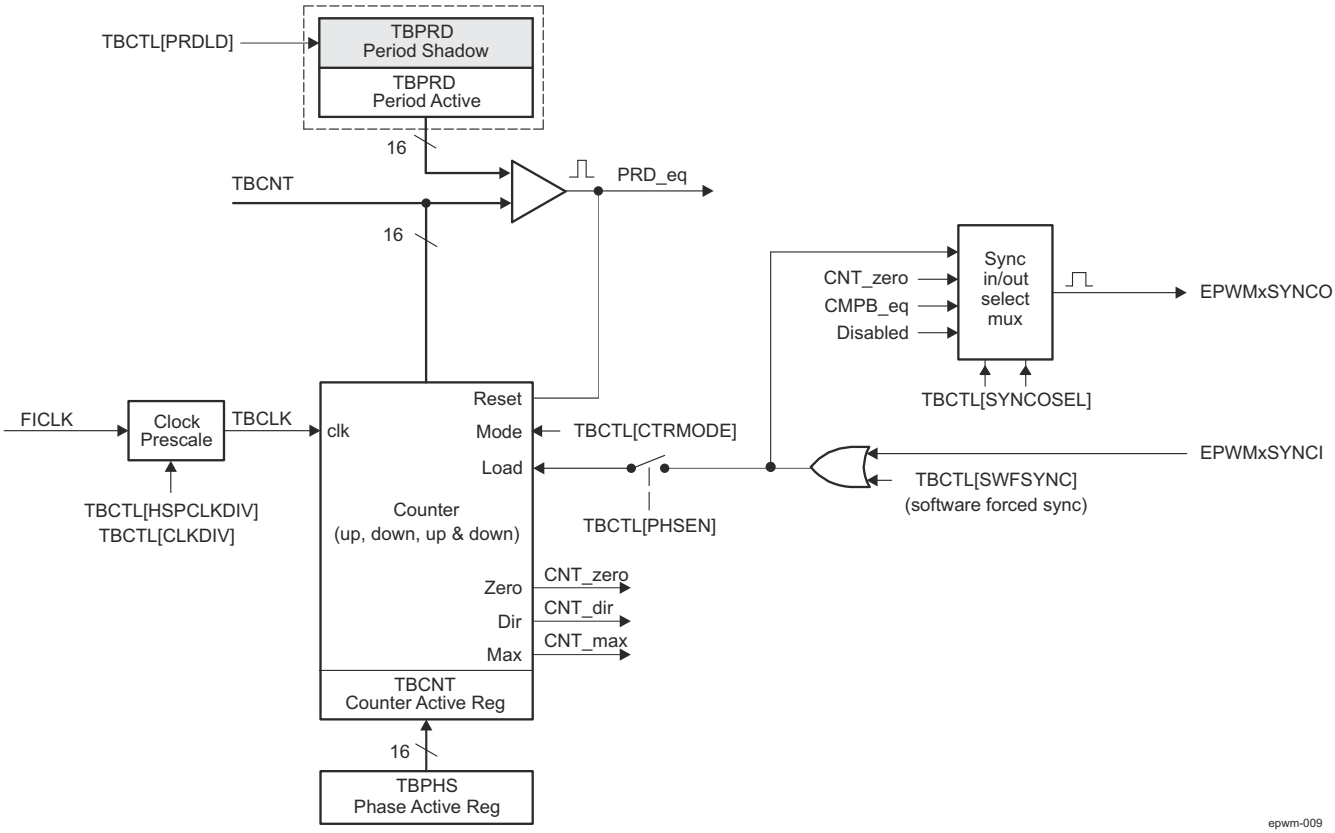


Figure 12-178. EPWM Time-Base Submodule Signals and Registers

Table 12-253. EPWM Time-Base Submodule Key Signals

Signal	Description
EPWMxSYNCI	Time-base synchronization input. Input pulse used to synchronize the time-base counter with the counter of another EPWM module earlier in the synchronization chain. An EPWM peripheral can be configured to use or ignore this signal. For the first EPWM module (EPWM0) this signal comes from a device pin. For subsequent EPWM modules this signal is passed from another EPWM peripheral. For example, EPWM2SYNCI is generated by the EPWM1 peripheral and the EPWM3SYNCI is generated (optional through internal multiplexer) by EPWM2 or from a device pin. See Section 12.4.2.3.2.3.2 for information on the synchronization order of a particular device.
EPWMxSYNCO	Time-base synchronization output. This output pulse is used to synchronize the counter of an EPWM module later in the synchronization chain. The EPWM module generates this signal from one of three event sources: 1. EPWMxSYNCI (Synchronization input pulse) 2. TBCNT = 0: The time-base counter (EPWM_TBCNT register) equal to zero (TBCNT = 0000h). 3. TBCNT = CMPB: The time-base counter (EPWM_TBCNT register) equal to the counter-compare B register — EPWM_CMPB (that is bitfield TBCNT = bitfield CMPB).

Copyright © 2024 Texas Instruments Incorporated

Table 12-253. EPWM Time-Base Submodule Key Signals (continued)

Signal	Description
PRD_eq	Time-base counter equal to the specified period. This signal is generated whenever the counter value is equal to the active period register value. That is when TBCNT = TBPRD.
CNT_zero	Time-base counter equal to zero. This signal is generated whenever the counter value is zero. That is when TBCNT equals 0000h.
CMPB_eq	Time-base counter equal to active counter-compare B register (TBCNT = CMPB). This event is generated by the counter-compare submodule and used by the synchronization out logic.
CNT_dir	Time-base counter direction. Indicates the current direction of the EPWM's time-base counter. This signal is high when the counter is increasing and low when it is decreasing.
CNT_max	Time-base counter equal max value (TBCNT = FFFFh). Generated event when the EPWM_TBCNT register value reaches its maximum value. This signal is only used only as a status bit.
TBCLK	Time-base clock. This is a prescaled version of the system clock — FICLK and is used by all submodules within the EPWMn. This clock determines the rate at which time-base counter increments or decrements.

12.4.2.3.2.3 Calculating PWM Period and Frequency

The frequency of PWM events is controlled by the time-base period register (EPWM_TBPRD) and the mode of the time-base counter. Figure 12-179 shows the period (T_{pwm}) and frequency (F_{pwm}) relationships for the up-count, down-count, and up-down-count time-base counter modes when the period is set to 4 (EPWM_TBPRD[15-0] TBPRD = 0x4). The time increment for each step is defined by the time-base clock (TBCLK) which is a prescaled version of the system clock (FICLK).

The time-base counter has three modes of operation selected by the time-base control register (EPWM_TBCTL):

- **Up-Down-Count Mode:** In up-down-count mode, the time-base counter starts from zero and increments until the period (TBPRD) value is reached. When the period value is reached, the time-base counter then decrements until it reaches zero. At this point the counter repeats the pattern and begins to increment.
- **Up-Count Mode:** In this mode, the time-base counter starts from zero and increments until it reaches the value in the period register (TBPRD). When the period value is reached, the time-base counter resets to zero and begins to increment once again.
- **Down-Count Mode:** In down-count mode, the time-base counter starts from the period (TBPRD) value and decrements until it reaches zero. When it reaches zero, the time-base counter is reset back to the period value and it repeats this pattern.

Copyright © 2024 Texas Instruments Incorporated

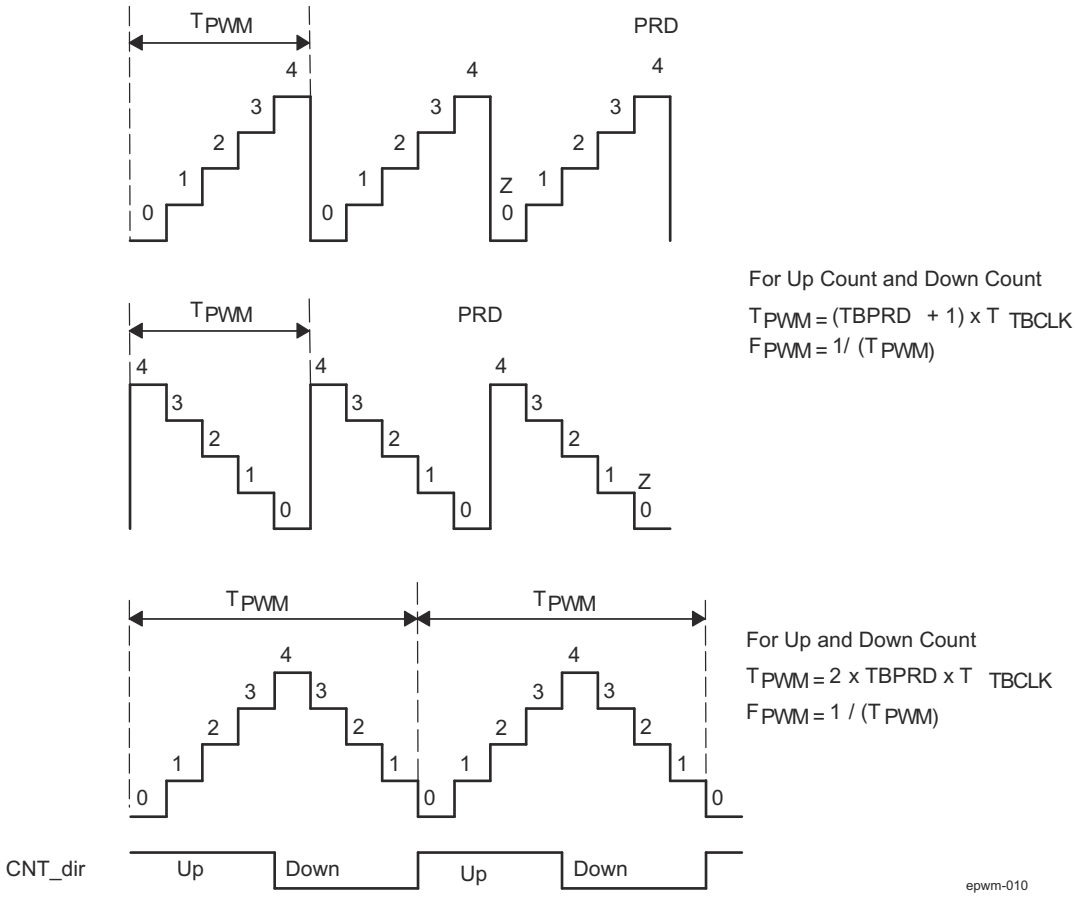


Figure 12-179. EPWM Time-Base Frequency and Period

12.4.2.3.2.3.1 EPWM Time-Base Period Shadow Register

The time-base period register (EPWM_TBPRD) has a shadow register. Shadowing allows the register update to be synchronized with the hardware. The following definitions are used to describe all shadow registers in the EPWM module:

- **Active Register:** The active register controls the hardware and is responsible for actions that the hardware causes or invokes.
- **Shadow Register:** The shadow register buffers or provides a temporary holding location for the active register. It has no direct effect on any control hardware. At a strategic point in time the shadow register's content is transferred to the active register. This prevents corruption or spurious operation due to the register being asynchronously modified by software.

The memory address of the shadow period register is the same as the active register. Which register is written to or read from is determined by the EPWM_TBCTL[3] PRDLD bit. This bit enables and disables the EPWM_TBPRD shadow register as follows:

- **Time-Base Period Shadow Mode:** The EPWM_TBPRD shadow register is enabled when EPWM_TBCTL[3] PRDLD = 0h. Reads from and writes to the EPWM_TBPRD register memory address go to the shadow register. The shadow register contents are transferred to the active register (EPWM_TBPRD (Active) ← EPWM_TBPRD (shadow)) when the time-base counter register (EPWM_TBCNT) equals zero (TBCNT = 0000h). By default the EPWM_TBPRD shadow register is enabled.
- **Time-Base Period Immediate Load Mode:** If immediate load mode is selected (EPWM_TBCTL[3] PRDLD = 1h), then a read from or a write to the TBPRD memory address goes directly to the active register.

12.4.2.3.2.3.2 EPWM Time-Base Counter Synchronization

A time-base synchronization scheme connects all of the EPWM modules on a device. Each EPWM module has a synchronization input (EPWMxSYNCI) and a synchronization output (EPWMxSYNCO). The input

Copyright © 2024 Texas Instruments Incorporated

synchronization for the first instance (EPWM0) comes from an external pin. For the device EPWM environment sync pin details refer to *EPWM Environment*. The possible synchronization connections for the remaining EPWM modules is shown in Figure 12-180.

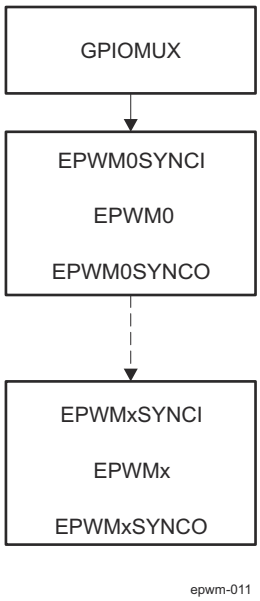


Figure 12-180. EPWM Time-Base Counter Synchronization Scheme 1

Each EPWM module can be configured to use or ignore the synchronization input. If the EPWM_TBCTL[2] PHSEN bit is set, then the time-base counter (TBCNT) of the EPWM module (EPWM_TBCNT register) will be automatically loaded with the phase register (EPWM_TBPHS) contents when one of the following conditions occur:

- **EPWMxSYNCI: Synchronization Input Pulse:** The value of the phase register is loaded into the counter register when an input synchronization pulse is detected (EPWM_TBPHS → EPWM_TBCNT). This operation occurs on the next valid time-base clock (TBCLK) edge.
- **Software Forced Synchronization Pulse:** Writing a 1h to the EPWM_TBCTL[6] SWFSYNC control bit invokes a software forced synchronization. This pulse is ORed with the synchronization input signal, and therefore has the same effect as a pulse on EPWMxSYNCI.

This feature enables the EPWM module to be automatically synchronized to the time base of another EPWM module. Lead or lag phase control can be added to the waveforms generated by different EPWM modules to synchronize them. In up-down-count mode, the EPWM_TBCTL[13] PHSDIR bit configures the direction of the time-base counter immediately after a synchronization event. The new direction is independent of the direction prior to the synchronization event. The TBPHS bit is ignored in count-up or count-down modes. See Figure 12-181 through Figure 12-184 for examples.

Clearing the EPWM_TBCTL[2] PHSEN bit configures the EPWM to ignore the synchronization input pulse. The synchronization pulse can still be allowed to flow-through to the EPWMxSYNCO and be used to synchronize other EPWM modules. In this way, a controller time-base (for example, EPWM1) and downstream modules (EPWM2–EPWMx) can be set and they may select to run in synchronization with the controller.

12.4.2.3.2.4 Phase Locking the Time-Base Clocks of Multiple EPWM Modules

As already described in *EPWM Modules Time-Base Clock Gating*, TB_CLKEN bit in the EPWMn_CTRL (where n = 0 to 5) register of the device CTRL_MMR0 can be used to individually control or globally synchronize the time-base clocks of all enabled EPWM modules on a device. When all TB_CLKEN bits are set to 0b0, the time-base clocks of all EPWMx (where x = 0 to 5) modules are stopped (default). When all TB_CLKEN bits are simultaneously set in software to 0b1, all EPWMx modules time-base clocks are started with the rising edge of TBCLK aligned. For perfectly synchronized TBCLKs, the prescaler bits in the EPWM_TBCTL register of each EPWM module must be set identically. The proper procedure for enabling the EPWM clocks is as follows:

Copyright © 2024 Texas Instruments Incorporated

1. Enable the EPWM module clocks.
2. Set TB_CLKEN = 0. This will stop the time-base clock within any enabled EPWMx module.
3. Configure the prescaler values and desired EPWM modes per each involved EPWMx.
4. Simultaneously set TB_CLKEN bits to 0b1.

12.4.2.3.2.5 EPWM Time-Base Counter Modes and Timing Waveforms

The time-base counter operates in one of four modes:

- Up-count mode which is asymmetrical.
- Down-count mode which is asymmetrical.
- Up-down-count which is symmetrical.
- Frozen where the time-base counter is held constant at the current value.

To illustrate the operation of the first three modes, Figure 12-181 to Figure 12-184 show when events are generated and how the time-base responds to an EPWMxSYNCI signal.

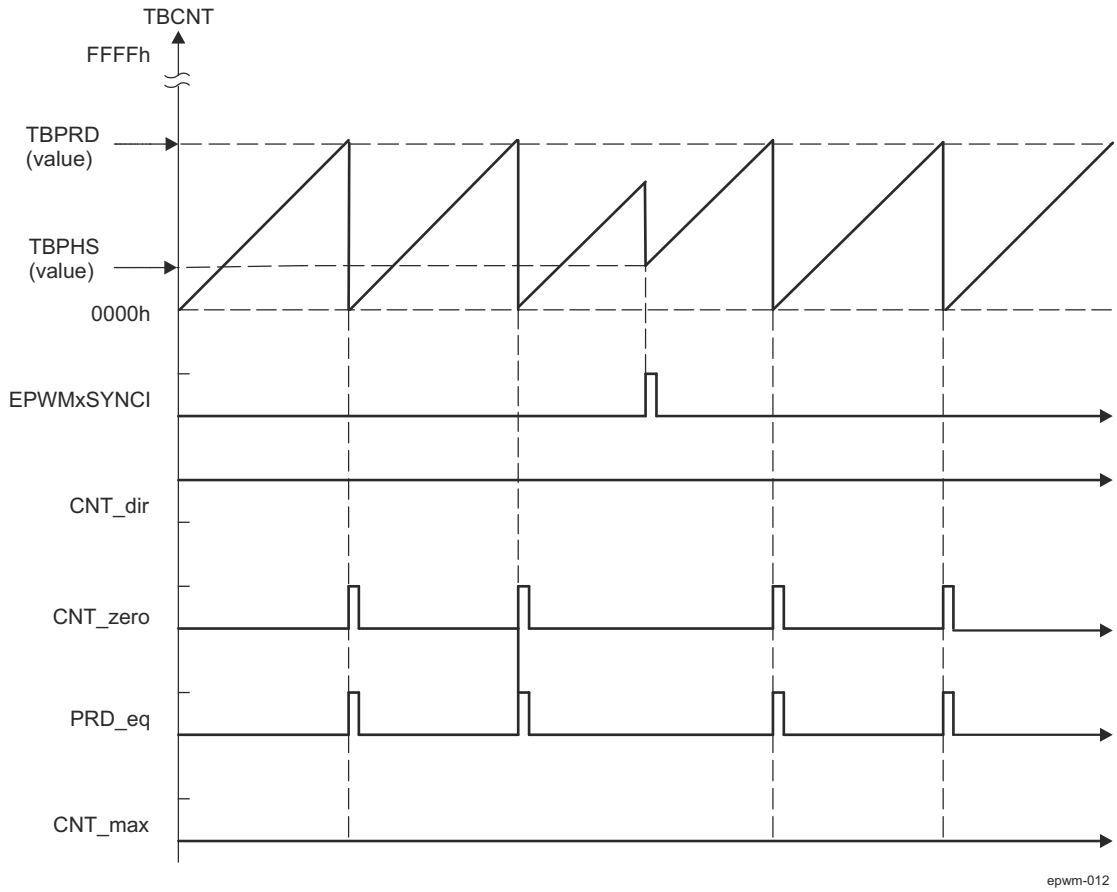


Figure 12-181. EPWM Time-Base Up-Count Mode Waveforms

Copyright © 2024 Texas Instruments Incorporated

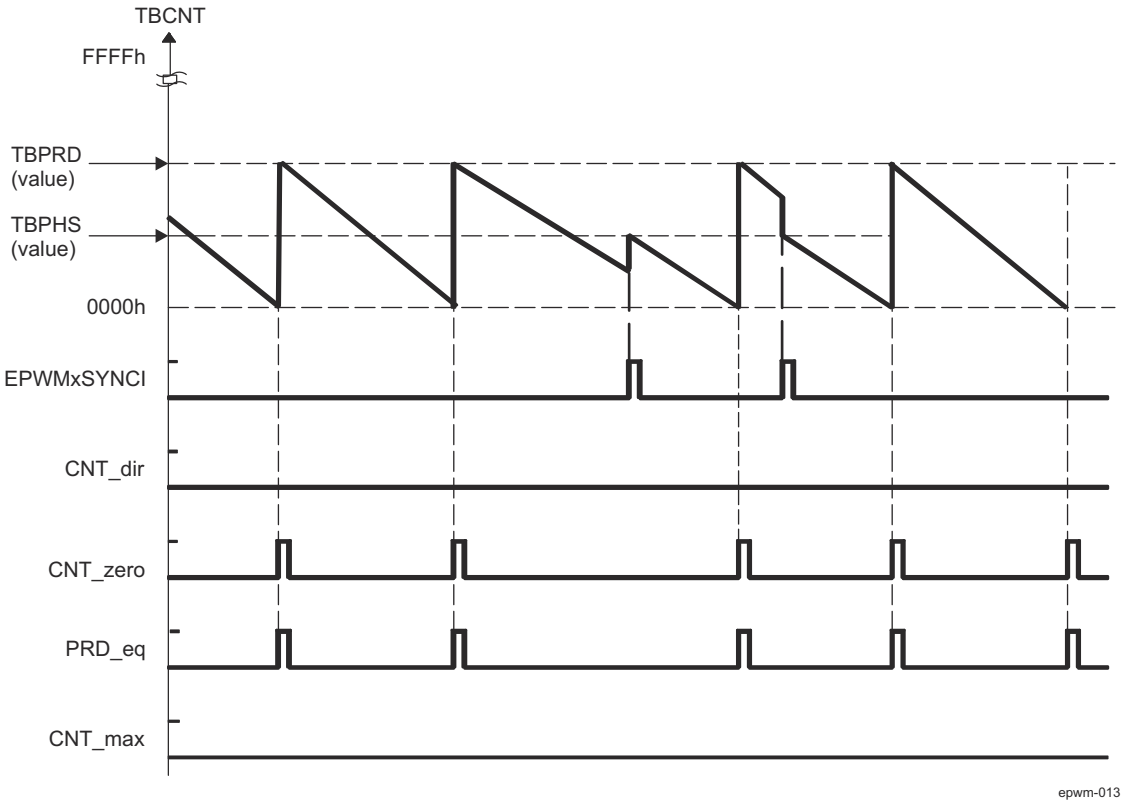


Figure 12-182. EPWM Time-Base Down-Count Mode Waveforms

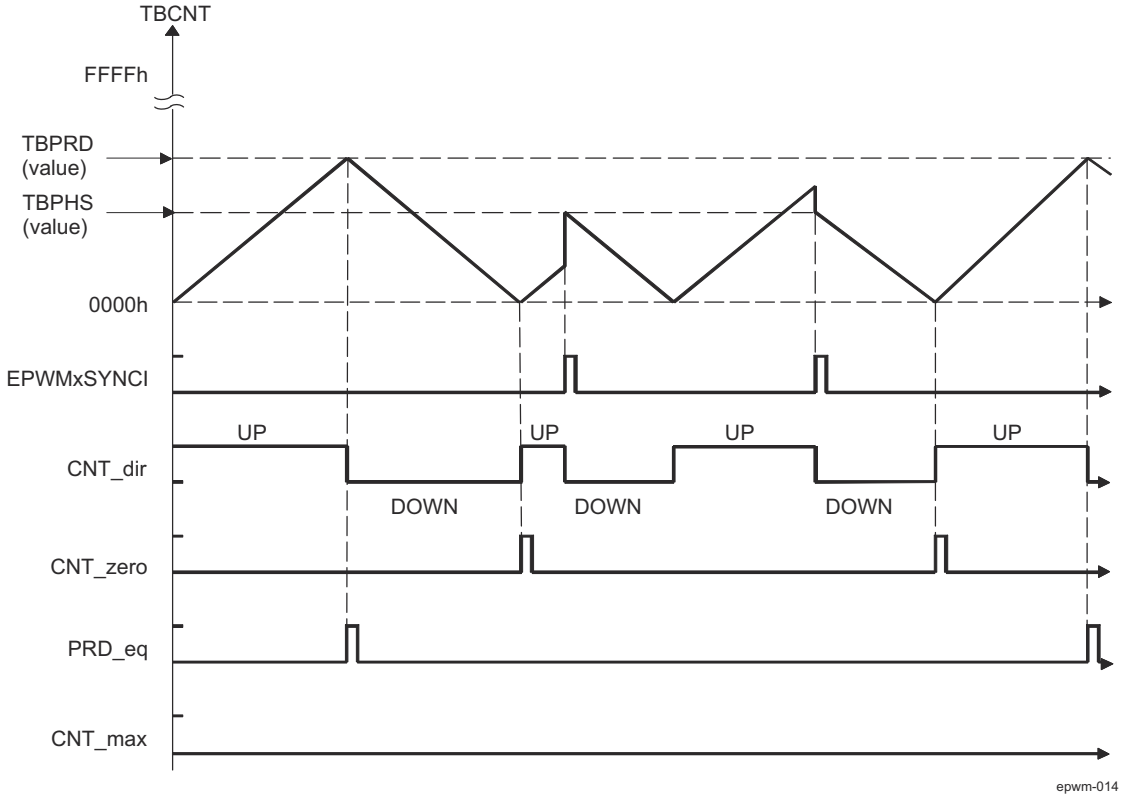


Figure 12-183. EPWM Time-Base Up-Down-Count Waveforms, EPWM_TBCTL[13] PHSDIR = 0 Count Down on Synchronization Event

Copyright © 2024 Texas Instruments Incorporated

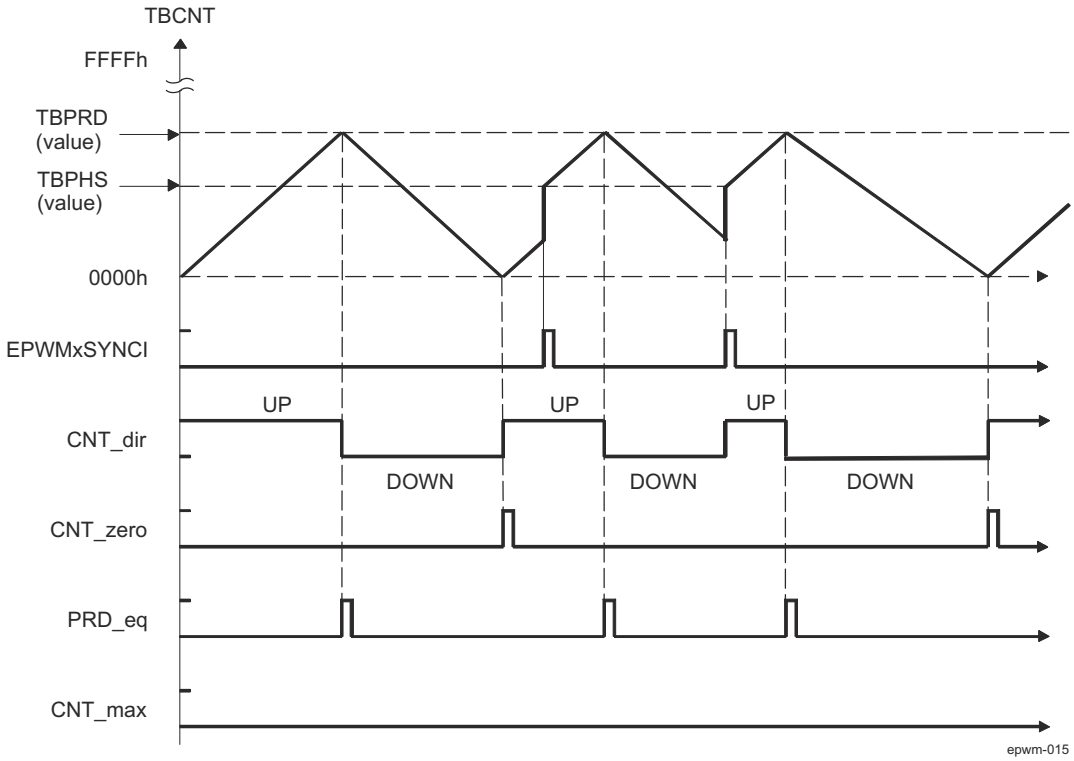


Figure 12-184. EPWM Time-Base Up-Down Count Waveforms, EPWM_TBCTL[13] PHSDIR = 1 Count Up on Synchronization Event

Copyright © 2024 Texas Instruments Incorporated

12.4.2.3.3 EPWM Counter-Compare (CC) Submodule

This section describes the Counter-Compare (CC) submodule in the PWM module.

12.4.2.3.3.1 Overview

Figure 12-185 illustrates the counter-compare submodule within the EPWM. Figure 12-186 shows the basic structure of the counter-compare submodule.

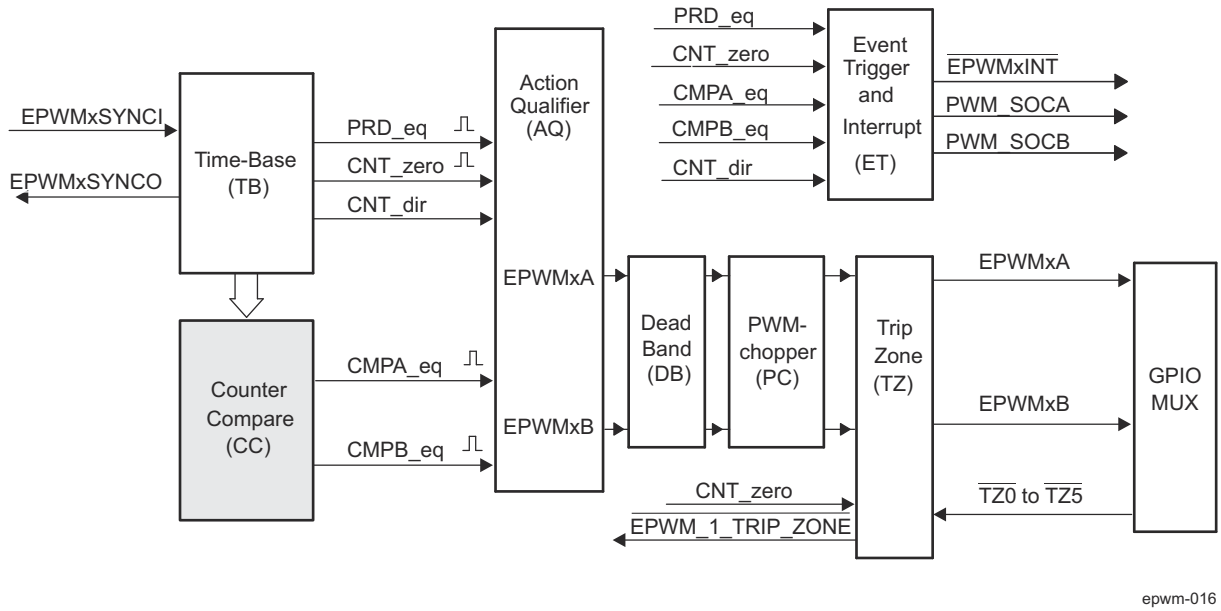


Figure 12-185. EPWM Counter-Compare Submodule

CC module features:

- Generates events based on programmable time stamps using the EPWM_CMPA and EPWM_CMPB registers
  - CMPA_eq (Time-base counter equals counter-compare A register (TBCNT = CMPA)).
  - CMPB_eq (Time-base counter equals counter-compare B register (TBCNT = CMPB)).
- Controls the PWM duty cycle if the action-qualifier submodule is configured appropriately
- Shadows new compare values to prevent corruption or glitches during the active PWM cycle.

The counter-compare submodule takes as input the time-base counter value. This value is continuously compared to the counter-compare A (EPWM_CMPA) and counter-compare B (EPWM_CMPB) registers. When the time-base counter is equal to one of the compare registers, the counter-compare unit generates an appropriate event.

12.4.2.3.3.2 Controlling and Monitoring the EPWM Counter-Compare Submodule

Table 12-254 lists the registers used to control and monitor the counter-compare submodule. Table 12-255 lists the key signals associated with the counter-compare submodule.

Table 12-254. EPWM Counter-Compare Submodule Registers

Acronym	Register Description	Address Offset	Shadowed
EPWM_CMPCTL	Counter-Compare Control Register.	Eh	No
HRPWM_CMPAHR	HRPWM Counter-Compare A Extension Register (1)	10h	Yes
EPWM_CMPA	Counter-Compare A Register	12h	Yes
EPWM_CMPB	Counter-Compare B Register	14h	Yes

(1) This register is available only on the EPWM modules with the high-resolution extension (HRPWM). On the EPWM modules that do not include the HRPWM, this location is reserved. See *EPWM Integration* to determine which EPWM instances include this feature.

Copyright © 2024 Texas Instruments Incorporated

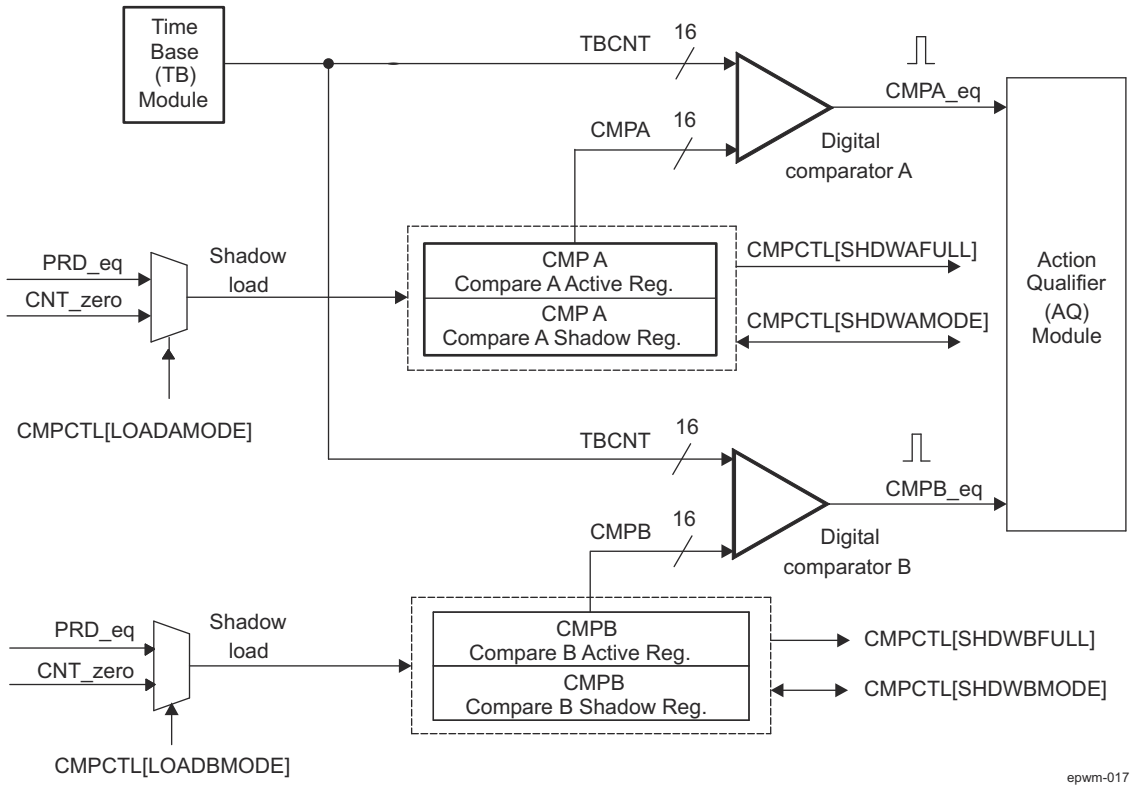


Figure 12-186. EPWM Counter-Compare Submodule Signals and Registers

Table 12-255. EPWM Counter-Compare Submodule Key Signals

Signal	Description of Event	Register Bitfields Compared
CMPA_eq	Time-base counter equal to the active counter-compare A value	TBCNT = CMPA
CMPB_eq	Time-base counter equal to the active counter-compare B value	TBCNT = CMPB
PRD_eq	Time-base counter equal to the active period. Used to load active counter-compare A and B registers from the shadow register	TBCNT = TBPRD
CNT_zero	Time-base counter equal to zero. Used to load active counter-compare A and B registers from the shadow register	TBCNT = 0000h

Copyright © 2024 Texas Instruments Incorporated

12.4.2.3.3.3 Operational Highlights for the EPWM Counter-Compare Submodule

The counter-compare submodule is responsible for generating two independent compare events based on two compare registers:

1. CMPA: Time-base counter equal to counter-compare A register (EPWM_TBCNT = EPWM_CMPA).
2. CMPB: Time-base counter equal to counter-compare B register (EPWM_TBCNT = EPWM_CMPB).

For up-count or down-count mode, each event occurs only once per cycle. For up-down-count mode each event occurs twice per cycle, if the compare value is between 0000h and TBPRD; and occurs once per cycle, if the compare value is equal to 0000h or equal to TBPRD. These events are fed into the action-qualifier submodule where they are qualified by the counter direction and converted into actions if enabled. Refer to Section 12.4.2.3.4 for more details.

The counter-compare EPWM_CMPA and EPWM_CMPB registers each have an associated shadow register. Shadowing provides a way to keep updates to the registers synchronized with the hardware. When shadowing is used, updates to the active registers only occurs at strategic points. This prevents corruption or spurious operation due to the register being asynchronously modified by software. The memory address of the active register and the shadow register is identical. Which register is written to or read from is determined by the EPWM_CMPCTL[4] SHDWAMODE and EPWM_CMPCTL[6] SHDWBMODE bits. These bits enable and disable the EPWM_CMPA shadow register and EPWM_CMPB shadow register respectively. The behavior of the two load modes is described below:

- **Shadow Load Mode:**
  The shadow mode for the EPWM_CMPA register is enabled by clearing the EPWM_CMPCTL[4] SHDWAMODE bit and the shadow register for EPWM_CMPB register is enabled by clearing the EPWM_CMPCTL[6] SHDWBMODE bit. Shadow mode is enabled by default for both EPWM_CMPA and EPWM_CMPB register.

  If the shadow register is enabled then the content of the shadow register is transferred to the active register on one of the following events:
  - PRD_eq: Time-base counter equal to the period (TBCNT = TBPRD).
  - CNT_zero: Time-base counter equal to zero (TBCNT = 0000h)
  - Both PRD_eq and CNT_zero events occurrence

  Which of these three events will drive the counter compare module is specified by the EPWM_CMPCTL[1-0] LOADAMODE and EPWM_CMPCTL[3-2] LOADBMODE register bit fields. Only the active register contents are used by the counter-compare submodule to generate events to be sent to the action-qualifier.
- **Immediate Load Mode:**
  If immediate load mode is selected (EPWM_CMPCTL[4] SHDWAMODE = 1h or EPWM_CMPCTL[6] SHDWBMODE = 1h), then a read from or a write to the register will go directly to the active register.

Figure 12-187 and Figure 12-188 show Compare A and B Dual Shadow registers.

Copyright © 2024 Texas Instruments Incorporated

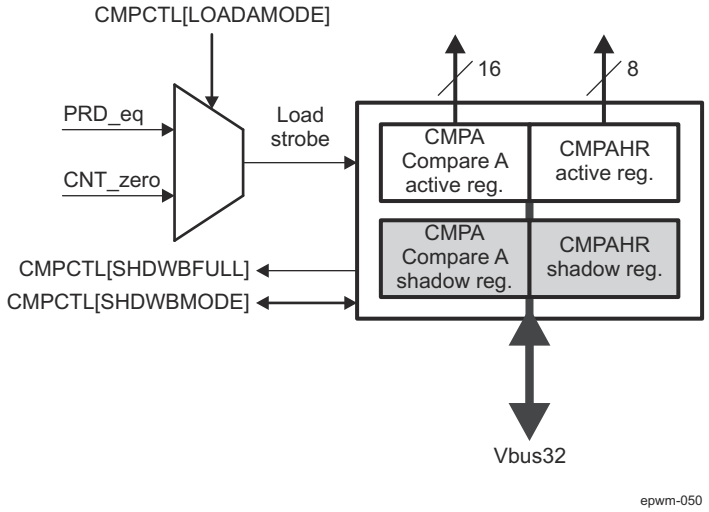


Figure 12-187. Compare A Dual Shadow register

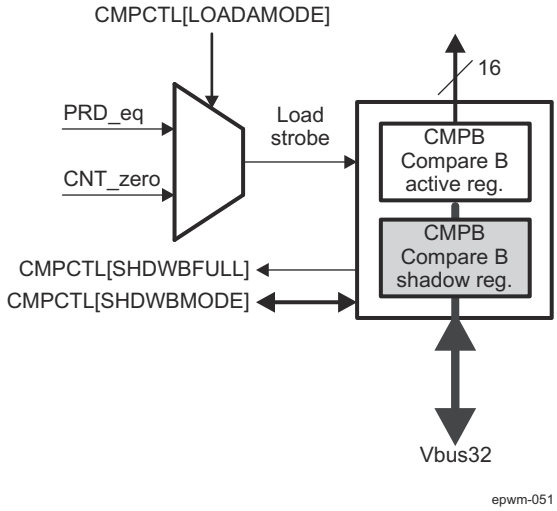


Figure 12-188. Compare B Dual Shadow register

Note

In HRPWM mode the user must perform a single 32-bit write access to both the HRPWM_CMPAHR and EPWM_CMPA registers. Two 16-bit accesses are not allowed.

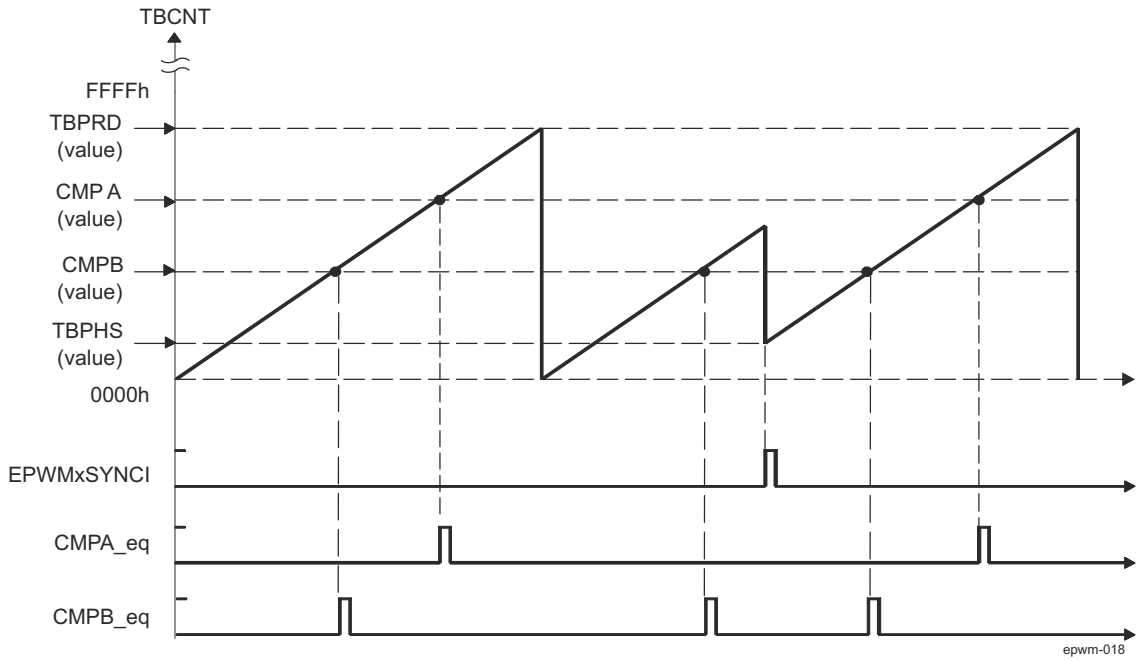
12.4.2.3.3.4 EPWM Counter-Compare Submodule Timing Waveforms

As described in Section 12.4.2.3.2, the Time Base (TB) module can be configured to operate in 3 distinct count modes:

- Up-count mode: used to generate an asymmetrical PWM waveform.
- Down-count mode: used to generate an asymmetrical PWM waveform.
- Up-down-count mode: used to generate a symmetrical PWM waveform.

The timing diagrams in Figure 12-189 to Figure 12-192 show how CMPA and CMPB events are generated in each of the 3 count modes and how the EPWMxSYNCI signal interacts.

Copyright © 2024 Texas Instruments Incorporated



An EPWMxSYNCI external synchronization event can cause a discontinuity in the TBCNT count sequence. This can lead to a compare event being skipped. This skipping is considered normal operation and must be taken into account.

Figure 12-189. EPWM Counter-Compare Event Waveforms in Up-Count Mode

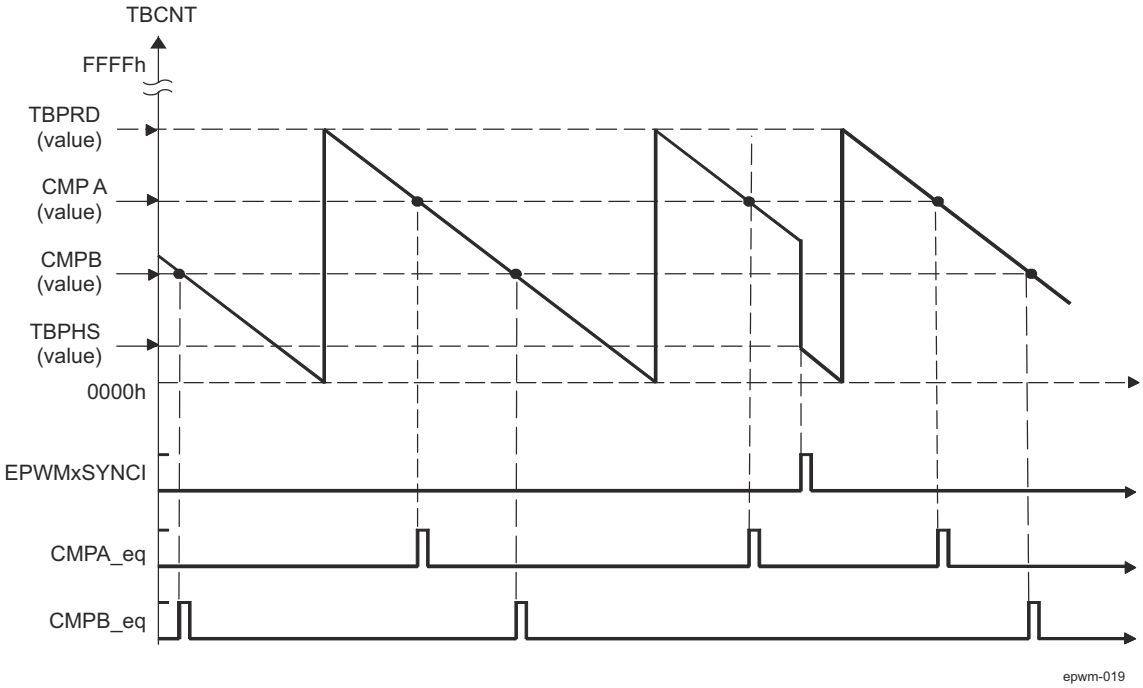


Figure 12-190. EPWM Counter-Compare Events in Down-Count Mode

Copyright © 2024 Texas Instruments Incorporated

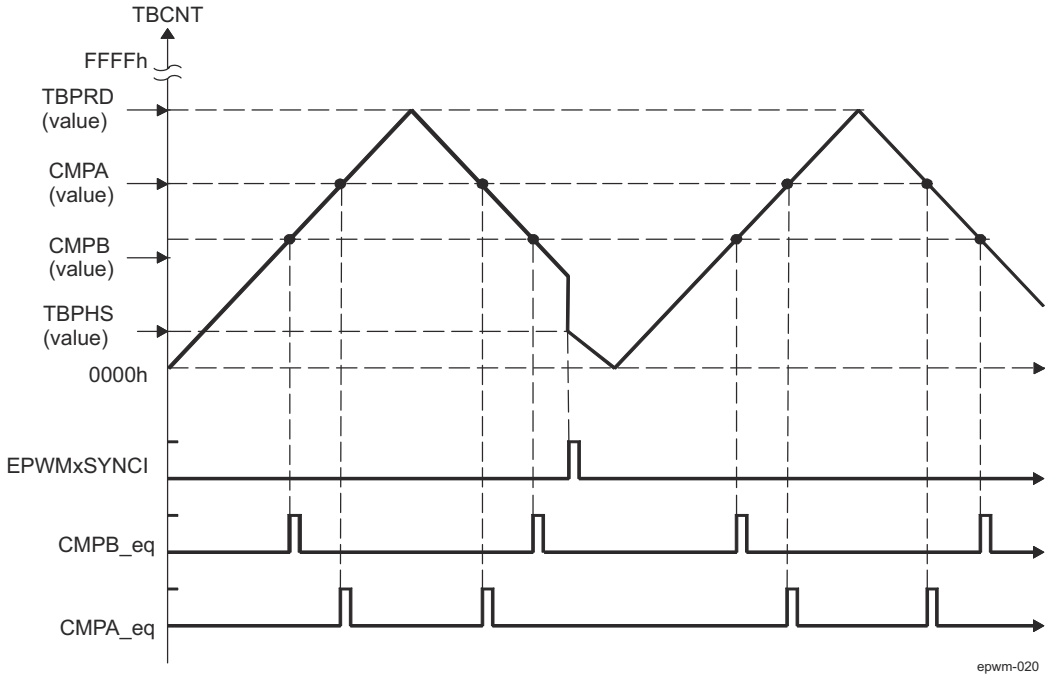


Figure 12-191. EPWM Counter-Compare Events in Up-Down-Count Mode, EPWM_TBCTL[13] PHSDIR = 0 Count Down on Synchronization Event

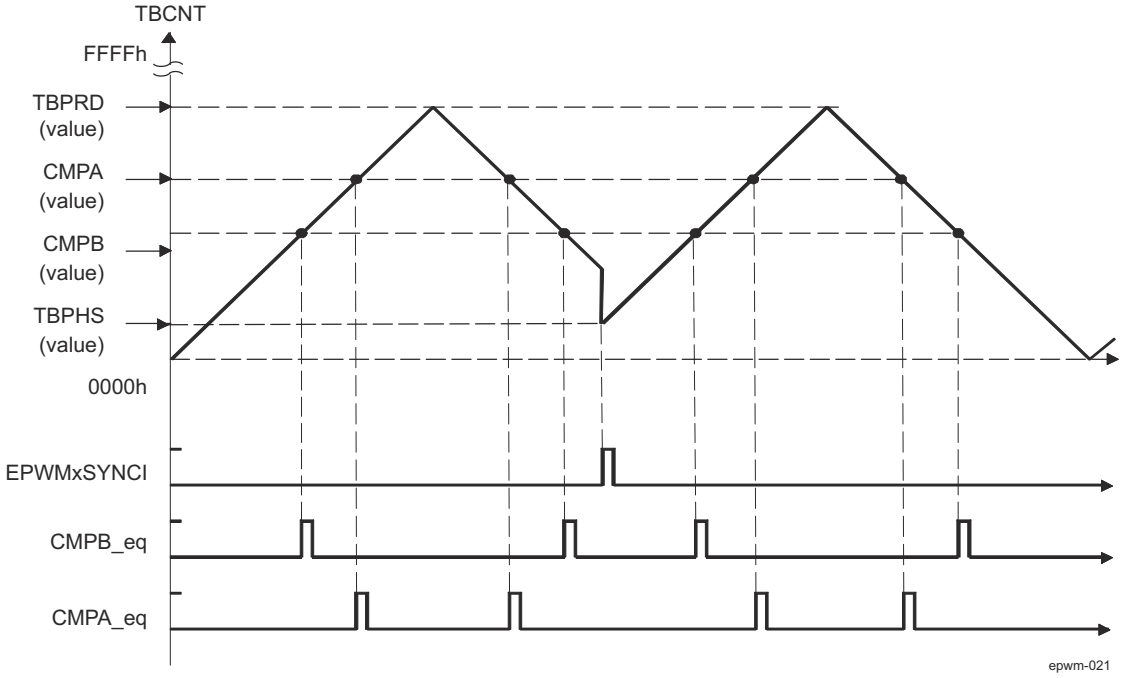


Figure 12-192. EPWM Counter-Compare Events in Up-Down-Count Mode, EPWM_TBCTL[13] PHSDIR = 1 Count Up on Synchronization Event

Copyright © 2024 Texas Instruments Incorporated

12.4.2.3.4 EPWM Action-Qualifier (AQ) Submodule

This section describes the Action-Qualifier (AQ) submodule in the PWM module.

12.4.2.3.4.1 Overview

Figure 12-193 shows the action-qualifier (AQ) submodule in the EPWM system. This submodule has the most important role in waveform construction and PWM generation. It decides which events are converted into various action types, thereby producing the required switched waveforms at the EPWMxA and EPWMxB outputs.

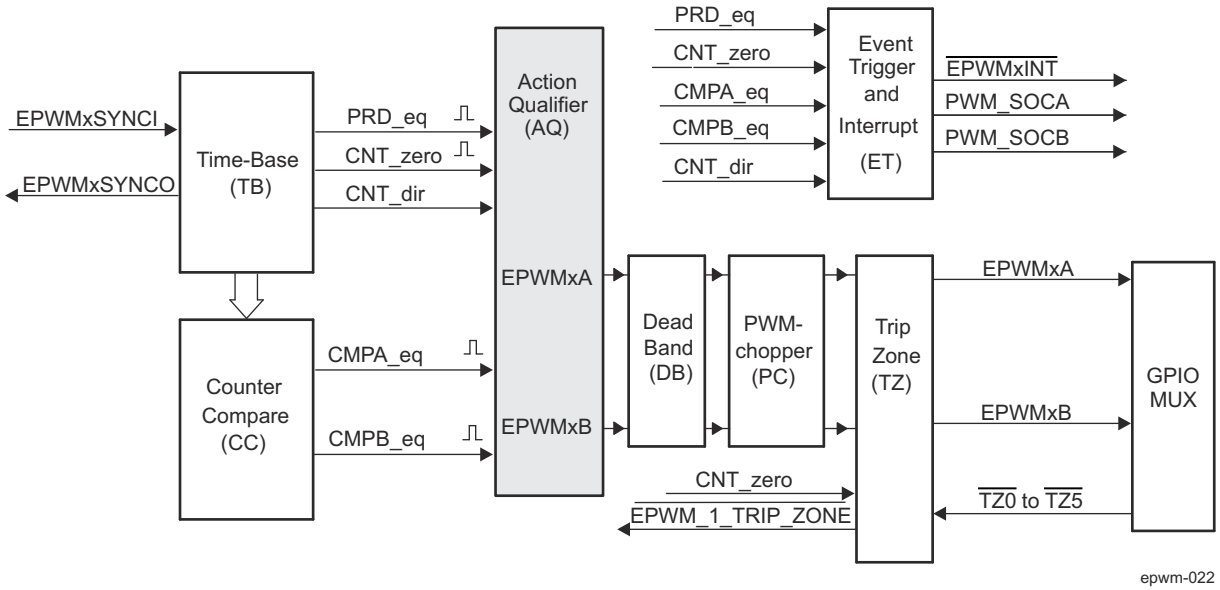


Figure 12-193. EPWM Action-Qualifier Submodule

AQ module features:

- Qualifying and generating actions (set, clear, toggle) based on the following events:
  - PRD_eq: Time-base counter equal to the period (TBCNT = TBPRD)
  - CNT_zero: Time-base counter equal to zero (TBCNT = 0000h)
  - CMPA_eq: Time-base counter equal to the counter-compare A register (TBCNT = CMPA)
  - CMPB_eq: Time-base counter equal to the counter-compare B register (TBCNT = CMPB)
- Managing priority when these events occur concurrently
- Providing independent control of events when the time-base counter is increasing and when it is decreasing

12.4.2.3.4.2 Controlling and Monitoring the EPWM Action-Qualifier Submodule

Table 12-256 lists the registers used to control and monitor the action-qualifier submodule.

Table 12-256. EPWM Action-Qualifier Submodule Registers

Acronym	Register Description	Address Offset	Shadowed
EPWM_AQCTLA	Action-Qualifier Control Register For Output A (EPWMxA)	16h	No
EPWM_AQCTLB	Action-Qualifier Control Register For Output B (EPWMxB)	18h	No
EPWM_AQSFRC	Action-Qualifier Software Force Register	1Ah	No
EPWM_AQCSFRC	Action-Qualifier Continuous Software Force	1Ch	Yes

The action-qualifier submodule is based on event-driven logic. It can be thought of as a programmable cross switch with events at the input and actions at the output, all of which are software controlled via the set of registers as shown in Figure 12-194. The possible input events are summarized again in Table 12-257.

Copyright © 2024 Texas Instruments Incorporated

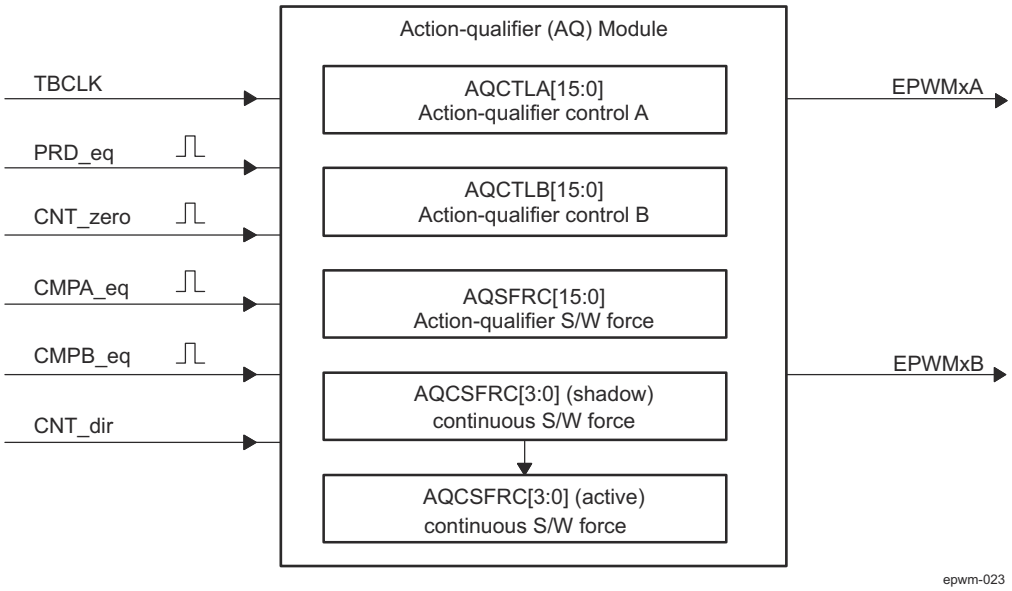


Figure 12-194. EPWM Action-Qualifier Submodule Inputs and Outputs

Table 12-257. EPWM Action-Qualifier Submodule Possible Input Events

Signal	Description	Register Bitfield Compared
PRD_eq	Time-base counter equal to the period value	TBCNT = TBPRD
CNT_zero	Time-base counter equal to zero	TBCNT = 0000h
CMPA_eq	Time-base counter equal to the counter-compare A	TBCNT = CMPA
CMPB_eq	Time-base counter equal to the counter-compare B	TBCNT = CMPB
Software forced event	Asynchronous event initiated by software	

The software forced action is a useful asynchronous event. This control is handled by EPWM_AQSFRC and EPWM_AQCSFRC registers.

The action-qualifier submodule controls how the two outputs EPWMxA and EPWMxB behave when a particular event occurs. The event inputs to the action-qualifier submodule are further qualified by the counter direction (up or down). This allows for independent action on outputs on both the count-up and count-down phases.

The possible actions imposed on outputs EPWMxA and EPWMxB are:

- **Set High:** Set output EPWMxA or EPWMxB to a high level.
- **Clear Low:** Set output EPWMxA or EPWMxB to a low level.
- **Toggle:** If EPWMxA or EPWMxB is currently pulled high, then pull the output low. If EPWMxA or EPWMxB is currently pulled low, then pull the output high.
- **Do Nothing:** Keep outputs EPWMxA and EPWMxB at same level as currently set. Although the "Do Nothing" option prevents an event from causing an action on the EPWMxA and EPWMxB outputs, this event can still trigger interrupts. See the Event_Trigger (ET) submodule description in Section 12.4.2.3.8 for details.

Actions are specified independently for either output (EPWMxA or EPWMxB). Any or all events can be configured to generate actions on a given output. All qualifier actions are configured via the control registers found at the end of this section.

For clarity, the drawings in this chapter use a set of symbolic actions. These symbols are summarized in Figure 12-195. Each symbol represents an action as a marker in time. Some actions are fixed in time (zero and period) while the CMPA and CMPB actions are moveable and their time positions are programmed via the counter-compare A and B registers, respectively. To turn off or disable an action, use the "Do Nothing option"; it is the default at reset.

Copyright © 2024 Texas Instruments Incorporated

S/W force	TB Counter equals:				Actions
	Zero	Comp A	Comp B	Period	
SW ×	Z ×	CA ×	CB ×	P ×	Do Nothing
SW ↓	Z ↓	CA ↓	CB ↓	P ↓	Clear Low
SW ↑	Z ↑	CA ↑	CB ↑	P ↑	Set High
SW T	Z T	CA T	CB T	P T	Toggle

epwm-024

Figure 12-195. Possible Action-Qualifier Actions for EPWMxA and EPWMxB Outputs

Copyright © 2024 Texas Instruments Incorporated

12.4.2.3.4.3 EPWM Action-Qualifier Event Priority

It is possible for the EPWM action qualifier to receive more than one event at the same time. In this case events are assigned a priority by the hardware. The general rule is: events occurring later in time have a higher priority and software forced events always have the highest priority. The event priority levels for up-down-count mode are shown in Table 12-258. A priority level 1 is the highest priority and level 7 is the lowest. The priority changes slightly depending on the direction of TBCNT.

Table 12-258. EPWM Action-Qualifier Event Priority for Up-Down-Count Mode

Priority Level	Event if TBCNT is Incrementing TBCNT = 0 up to TBCNT = TBPRD	Event if TBCNT is Decrementing TBCNT = TBPRD down to TBCNT = 1
1 (Highest)	Software forced event	Software forced event
2	Counter equals CMPB on up-count (CBU)	Counter equals CMPB on down-count (CBD)
3	Counter equals CMPA on up-count (CAU)	Counter equals CMPA on down-count (CAD)
4	Counter equals zero	Counter equals period (TBPRD in EPWM_TBPRD active register)
5	Counter equals CMPB on down-count (CBD) (1)	Counter equals CMPB on up-count (CBU) (1)
6 (Lowest)	Counter equals CMPA on down-count (CAD) (1)	Counter equals CMPA on up-count (CBU) (1)

(1) To maintain symmetry for up-down-count mode, both up-events (CAU/CBU) and down-events (CAD/CBD) can be generated for TBPRD. Otherwise, up-events can occur only when the counter is incrementing and down-events can occur only when the counter is decrementing.

Table 12-259 shows the action-qualifier priority for up-count mode. In this case, the counter direction is always defined as up and thus down-count events will never be taken.

Table 12-259. EPWM Action-Qualifier Event Priority for Up-Count Mode

Priority Level	Event
1 (Highest)	Software forced event
2	Counter equal to period (TBPRD)
3	Counter equal to CMPB on up-count (CBU)
4	Counter equal to CMPA on up-count (CAU)
5 (Lowest)	Counter equal to Zero

Table 12-260 shows the action-qualifier priority for down-count mode. In this case, the counter direction is always defined as down and thus up-count events will never be taken.

Table 12-260. EPWM Action-Qualifier Event Priority for Down-Count Mode

Priority Level	Event
1 (Highest)	Software forced event
2	Counter equal to Zero
3	Counter equal to CMPB on down-count (CBD)
4	Counter equal to CMPA on down-count (CAD)
5 (Lowest)	Counter equal to period (TBPRD)

Copyright © 2024 Texas Instruments Incorporated

It is possible to set the compare value greater than the period. In this case the action will take place as shown in Table 12-261.

Table 12-261. Behavior if CMPA/CMPB is Greater than the Period

Counter Mode	Compare on Up-Count Event CAU/CBU	Compare on Down-Count Event CAU/CBU
Up-Count Mode	If CMPA/CMPB ≤ TBPRD period, then the event occurs on a compare match (TBCNT = CMPA or CMPB). If CMPA/CMPB > TBPRD, then the event will not occur.	Never occurs.
Down-Count Mode	Never occurs.	If CMPA/CMPB < TBPRD, the event will occur on a compare match (TBCNT = CMPA or CMPB). If CMPA/CMPB ≥ TBPRD, the event will occur on a period match (TBCNT = TBPRD).
Up-Down-Count Mode	If CMPA/CMPB < TBPRD and the counter is incrementing, the event occurs on a compare match (TBCNT = CMPA or CMPB). If CMPA/CMPB is ≥ TBPRD, the event will occur on a period match (TBCNT = TBPRD).	If CMPA/CMPB < TBPRD and the counter is decrementing, the event occurs on a compare match (TBCNT = CMPA or CMPB). If CMPA/CMPB ≥ TBPRD, the event occurs on a period match (TBCNT = TBPRD).

12.4.2.3.4.4 Waveforms for Common EPWM Configurations

Note

The waveforms in this chapter show the EPWMs behavior for a static compare register value. In a running system, the active compare registers (EPWM_CMPA and EPWM_CMPB) are typically updated from their respective shadow registers once every period. The user specifies when the update will take place — either when the time-base counter reaches zero or when the time-base counter reaches period. There are some cases when the action based on the new value can be delayed by one period or the action based on the old value can take effect for an extra period. Some PWM configurations avoid this situation. These include, but are not limited to, the following:

Use up-down-count mode to generate a symmetric PWM:

- If EPWM_CMPA / EPWM_CMPB is loaded on zero, then use EPWM_CMPA / EPWM_CMPB values greater than or equal to 1.
- If EPWM_CMPA / EPWM_CMPB is loaded on period, then use EPWM_CMPA / EPWM_CMPB values less than or equal to TBPRD - 1.

  This means there will always be a pulse of at least one TBCLK cycle in a PWM period which, when very short, tend to be ignored by the system.

Use up-down-count mode to generate an asymmetric PWM:

- To achieve 50-0% asymmetric PWM use the following configuration: Load EPWM_CMPA / EPWM_CMPB on period and use the period action to clear the PWM and a compare-up action to set the PWM. Modulate the compare value from 0 to TBPRD to achieve 50-0% PWM duty.

When using up-count mode to generate an asymmetric PWM:

- To achieve 0-100% asymmetric PWM use the following configuration: Load EPWM_CMPA / EPWM_CMPB on TBPRD. Use the Zero action to set the PWM and a compare-up action to clear the PWM. Modulate the compare value from 0 to TBPRD+1 to achieve 0-100% PWM duty.

Copyright © 2024 Texas Instruments Incorporated

Figure 12-196 shows how a symmetric PWM waveform can be generated using the up-down-count mode of the TBCNT. In this mode 0-100% DC modulation is achieved by using equal compare matches on the up count and down count portions of the waveform. In the example shown, CMPA is used to make the comparison. When the counter is incrementing the CMPA match will pull the PWM output high. Likewise, when the counter is decrementing the compare match will pull the PWM signal low. When CMPA = 0, the PWM signal is low for the entire period giving the 0% duty waveform. When EPWM_CMPA = EPWM_TBPRD, the PWM signal is high achieving 100% duty.

When using this configuration in practice, if CMPA/CMPB is loaded on zero, then use CMPA/CMPB values greater than or equal to 1. If CMPA/CMPB is loaded on period, then use CMPA/CMPB values less than or equal to TBPRD - 1. This means there will always be a pulse of at least one TBCLK cycle in a PWM period which, when very short, tend to be ignored by the system.

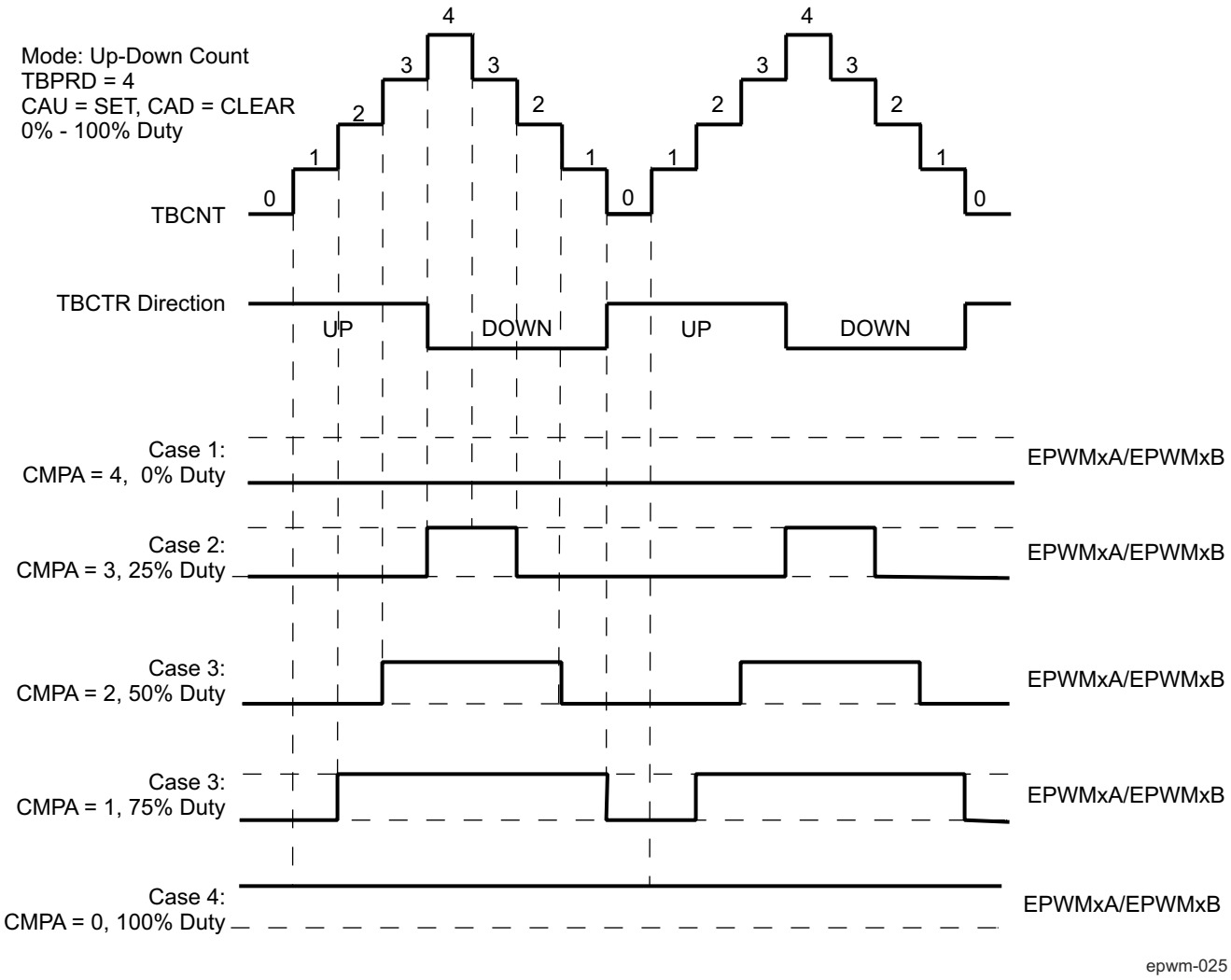


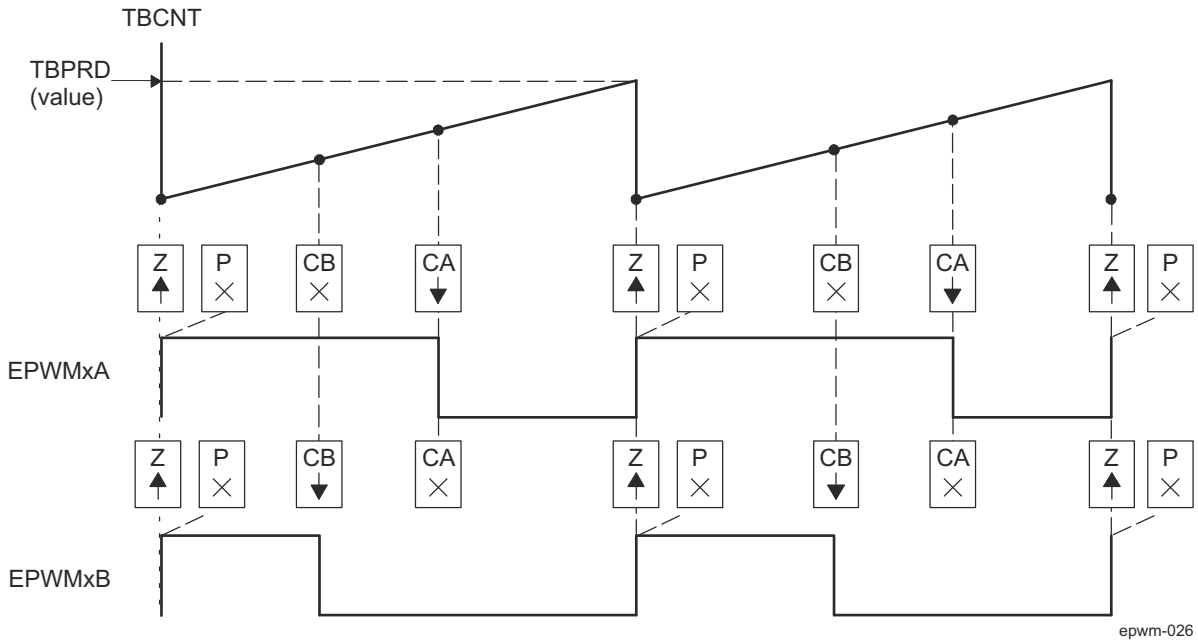
Figure 12-196. EPWM Up-Down-Count Mode Symmetrical Waveform

Copyright © 2024 Texas Instruments Incorporated

The PWM waveforms in Figure 12-197 through Figure 12-202 show some common action-qualifier configurations. Some conventions used in the figures are as follows:

- TBPRD, CMPA, and CMPB refer to the value written in their respective registers (EPWM_TBPRD, EPWM_CMPA, and EPWM_CMPB). The active register, not the shadow register, is used by the hardware.
- CMPx, refers to either CMPA or CMPB.
- EPWMxA and EPWMxB refer to the output signals from EPWMx
- Up-Down means Count-up-and-down mode, Up means up-count mode and Dwn means down-count mode
- Sym = Symmetric, Asym = Asymmetric

Table 12-262 and Table 12-263 contains initialization and runtime register configurations for the waveforms in Figure 12-197.



A. PWM period = (TBPRD + 1) × T_{TBCLK}
B. Duty modulation for EPWMxA is set by CMPA, and is active high (that is, high time duty proportional to CMPA).
C. Duty modulation for EPWMxB is set by CMPB and is active high (that is, high time duty proportional to CMPB).
D. The "Do Nothing" actions (X) are shown for completeness, but will not be shown on subsequent diagrams.
E. Actions at zero and period, although appearing to occur concurrently, are actually separated by one TBCLK period. TBCNT wraps from period to 0000h.

Figure 12-197. Up, Single Edge Asymmetric Waveform, With Independent Modulation on EPWMxA and EPWMxB — Active High

Copyright © 2024 Texas Instruments Incorporated

Table 12-262. EPWMx Initialization for Figure 12-197

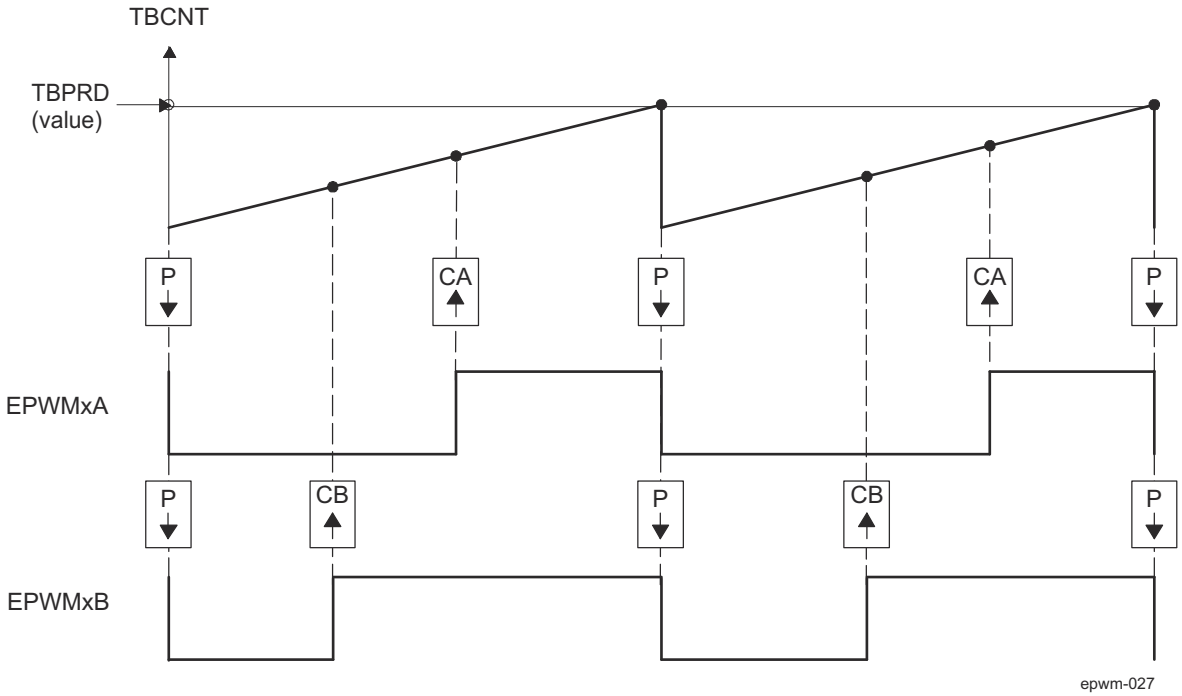
Register	Bitfield	Value	Comments
EPWM_TBPRD	TBPRD	600 (258h)	Period = 601 TBCLK counts
EPWM_TBPHS	TBPHS	0	Clear Phase Register to 0
EPWM_TBCNT	TBCNT	0	Clear TB counter
EPWM_TBCTL	CTRMODE	TB_UP	
	PHSEN	TB_DISABLE	Phase loading disabled
	PRDLD	TB_SHADOW	
	SYNCOSEL	TB_SYNC_DISABLE	
	HSPCLKDIV	TB_DIV1	TBCLK = FICLK
	CLKDIV	TB_DIV1	
EPWM_CMPA	CMPA	350 (15Eh)	Compare A = 350 TBCLK counts
EPWM_CMPB	CMPB	200 (C8h)	Compare B = 200 TBCLK counts
EPWM_CMPCTL	SHDWAMODE	CC_SHADOW	
	SHDWBMODE	CC_SHADOW	
	LOADAMODE	CC_CTR_ZERO	Load on TBCNT = 0
	LOADBMODE	CC_CTR_ZERO	Load on TBCNT = 0
EPWM_AQCTLA	ZRO	AQ_SET	
	CAU	AQ_CLEAR	
EPWM_AQCTLB	ZRO	AQ_SET	
	CBU	AQ_CLEAR	

Table 12-263. EPWMx Run Time Changes for Figure 12-197

Register	Bitfield	Value	Comments
EPWM_CMPA	CMPA	Duty1A	Adjust duty for output EPWM1A
EPWM_CMPB	CMPB	Duty1B	Adjust duty for output EPWM1B

Copyright © 2024 Texas Instruments Incorporated

Table 12-264 and Table 12-265 contains initialization and runtime register configurations for the waveforms in Figure 12-198.



A. PWM period = (TBPRD + 1) × T_{TBCLK}
B. Duty modulation for EPWMxA is set by CMPA, and is active low (that is, the low time duty is proportional to CMPA).
C. Duty modulation for EPWMxB is set by CMPB and is active low (that is, the low time duty is proportional to CMPB).
D. The Do Nothing actions (X) are shown for completeness here, but will not be shown on subsequent diagrams.
E. Actions at zero and period, although appearing to occur concurrently, are actually separated by one TBCLK period. TBCNT wraps from period to 0000h.

Figure 12-198. Up, Single Edge Asymmetric Waveform With Independent Modulation on EPWMxA and EPWMxB — Active Low

Copyright © 2024 Texas Instruments Incorporated

Table 12-264. EPWMx Initialization for Figure 12-198

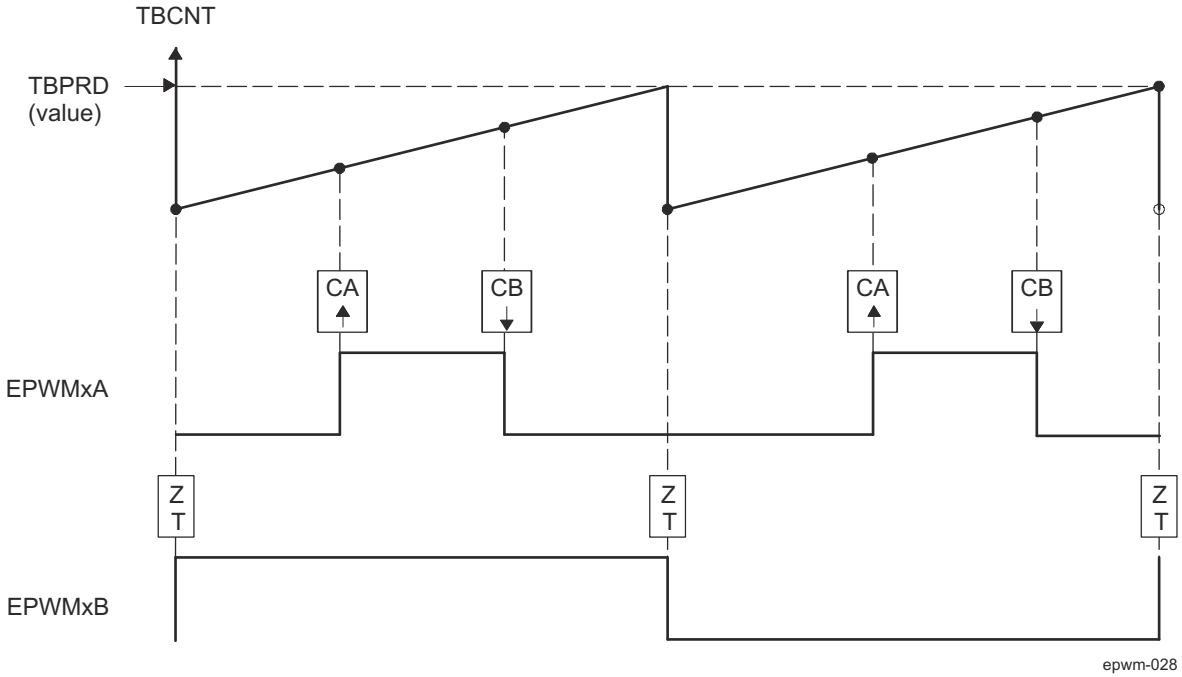
Register	Bitfield	Value	Comments
EPWM_TBPRD	TBPRD	600 (258h)	Period = 601 TBCLK counts
EPWM_TBPHS	TBPHS	0	Clear Phase Register to 0
EPWM_TBCNT	TBCNT	0	Clear TB counter
EPWM_TBCTL	CTRMODE	TB_UP	
	PHSEN	TB_DISABLE	Phase loading disabled
	PRDLD	TB_SHADOW	
	SYNCOSEL	TB_SYNC_DISABLE	
	HSPCLKDIV	TB_DIV1	TBCLK = FICLK
	CLKDIV	TB_DIV1	
EPWM_CMPA	CMPA	350 (15Eh)	Compare A = 350 TBCLK counts
EPWM_CMPB	CMPB	200 (C8h)	Compare B = 200 TBCLK counts
EPWM_CMPCTL	SHDWAMODE	CC_SHADOW	
	SHDWBMODE	CC_SHADOW	
	LOADAMODE	CC_CTR_ZERO	Load on TBCNT = 0
	LOADBMODE	CC_CTR_ZERO	Load on TBCNT = 0
EPWM_AQCTLA	PRD	AQ_CLEAR	
	CAU	AQ_SET	
EPWM_AQCTLB	PRD	AQ_CLEAR	
	CBU	AQ_SET	

Table 12-265. EPWMx Run Time Changes for Figure 12-198

Register	Bit	Value	Comments
EPWM_CMPA	CMPA	Duty1A	Adjust duty for output EPWM1A
EPWM_CMPB	CMPB	Duty1B	Adjust duty for output EPWM1B

Copyright © 2024 Texas Instruments Incorporated

Table 12-266 and Table 12-267 contains initialization and runtime register configurations for the waveforms Figure 12-199. Use the code in *Constant Definitions Used in the EPWM Code Examples* to define the headers.



A. PWM frequency = 1/((TBPRD + 1) × T_{TBCLK})
B. Pulse can be placed anywhere within the PWM cycle (0000h - TBPRD)
C. High time duty proportional to (CMPB - CMPA)
D. EPWMxB can be used to generate a 50% duty square wave with frequency = 1/2 × ((TBPRD + 1) × TBCLK)

Figure 12-199. Up-Count, Pulse Placement Asymmetric Waveform With Independent Modulation on EPWMxA

Copyright © 2024 Texas Instruments Incorporated

Table 12-266. EPWMx Initialization for Figure 12-199

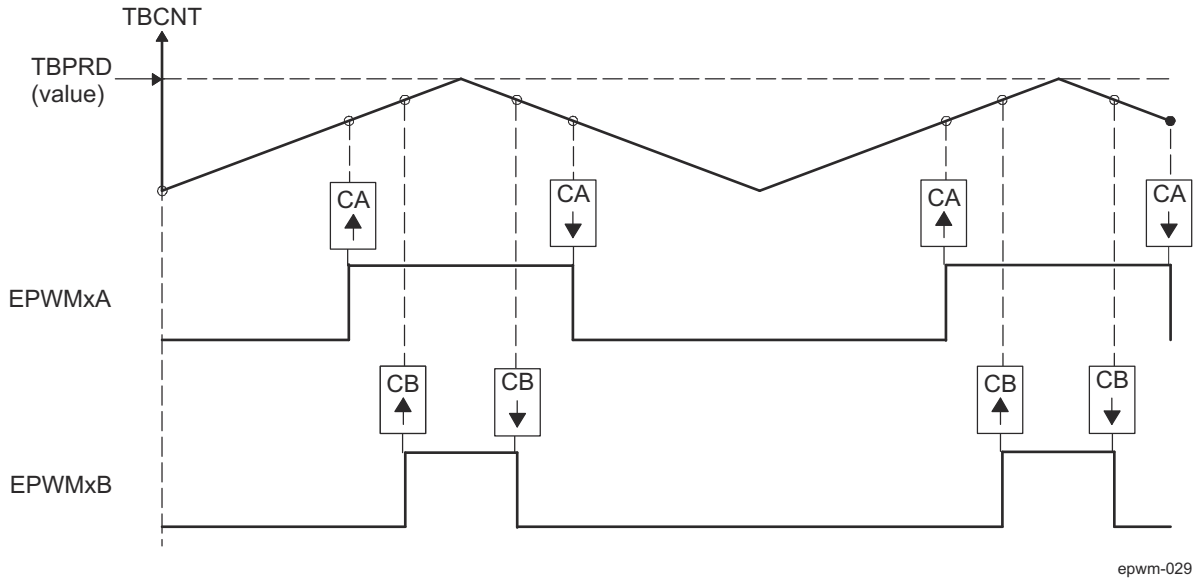
Register	Bitfield	Value	Comments
EPWM_TBPRD	TBPRD	600 (258h)	Period = 601 TBCLK counts
EPWM_TBPHS	TBPHS	0	Clear Phase Register to 0
EPWM_TBCNT	TBCNT	0	Clear TB counter
EPWM_TBCTL	CTRMODE	TB_UP	
	PHSEN	TB_DISABLE	Phase loading disabled
	PRDLD	TB_SHADOW	
	SYNCOSEL	TB_SYNC_DISABLE	
	HSPCLKDIV	TB_DIV1	TBCLK = FICLK
	CLKDIV	TB_DIV1	
EPWM_CMPA	CMPA	200 (C8h)	Compare A = 200 TBCLK counts
EPWM_CMPB	CMPB	400 (190h)	Compare B = 400 TBCLK counts
EPWM_CMPCTL	SHDWAMODE	CC_SHADOW	
	SHDWBMODE	CC_SHADOW	
	LOADAMODE	CC_CTR_ZERO	Load on TBCNT = 0
	LOADBMODE	CC_CTR_ZERO	Load on TBCNT = 0
EPWM_AQCTLA	CAU	AQ_SET	
	CBU	AQ_CLEAR	
EPWM_AQCTLB	ZRO	AQ_TOGGLE	

Table 12-267. EPWMx Run Time Changes for Figure 12-199

Register	Bitfield	Value	Comments
EPWM_CMPA	CMPA	EdgePosA	Adjust duty for output EPWM1A
EPWM_CMPB	CMPB	EdgePosB	

Copyright © 2024 Texas Instruments Incorporated

Table 12-268 and Table 12-269 contains initialization and runtime register configurations for the waveforms in Figure 12-200. Use the code in *Constant Definitions Used in the EPWM Code Examples* to define the headers.



A. PWM period = 2 x TBPRD × T_{TBCLK}
B. Duty modulation for EPWMxA is set by CMPA, and is active low (that is, the low time duty is proportional to CMPA).
C. Duty modulation for EPWMxB is set by CMPB and is active low (that is, the low time duty is proportional to CMPB).
D. Outputs EPWMxA and EPWMxB can drive independent power switches.

Figure 12-200. Up-Down-Count, Dual Edge Symmetric Waveform, With Independent Modulation on EPWMxA and EPWMxB — Active Low

Copyright © 2024 Texas Instruments Incorporated

Table 12-268. EPWMx Initialization for Figure 12-200

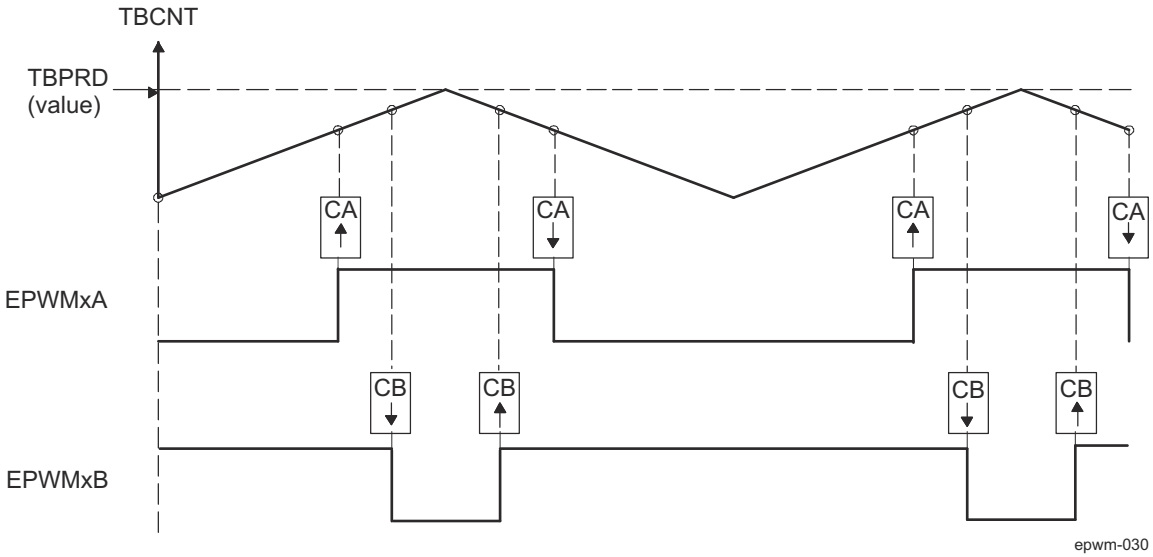
Register	Bitfield	Value	Comments
EPWM_TBPRD	TBPRD	600 (258h)	Period = 601 TBCLK counts
EPWM_TBPHS	TBPHS	0	Clear Phase Register to 0
EPWM_TBCNT	TBCNT	0	Clear TB counter
EPWM_TBCTL	CTRMODE	TB_UPDOWN	
	PHSEN	TB_DISABLE	Phase loading disabled
	PRDLD	TB_SHADOW	
	SYNCOSEL	TB_SYNC_DISABLE	
	HSPCLKDIV	TB_DIV1	TBCLK = FICLK
	CLKDIV	TB_DIV1	
EPWM_CMPA	CMPA	400 (190h)	Compare A = 400 TBCLK counts
EPWM_CMPB	CMPB	500 (1F4h)	Compare B = 500 TBCLK counts
EPWM_CMPCTL	SHDWAMODE	CC_SHADOW	
	SHDWBMODE	CC_SHADOW	
	LOADAMODE	CC_CTR_ZERO	Load on TBCNT = 0
	LOADBMODE	CC_CTR_ZERO	Load on TBCNT = 0
EPWM_AQCTLA	CAU	AQ_SET	
	CAD	AQ_CLEAR	
EPWM_AQCTLB	CBU	AQ_SET	
	CBD	AQ_CLEAR	

Table 12-269. EPWMx Run Time Changes for Figure 12-200

Register	Bitfield	Value	Comments
EPWM_CMPA	CMPA	Duty1A	Adjust duty for output EPWM1A
EPWM_CMPB	CMPB	Duty1B	Adjust duty for output EPWM1B

Copyright © 2024 Texas Instruments Incorporated

Table 12-270 and Table 12-271 contains initialization and runtime register configurations for the waveforms in Figure 12-201. Use the code in *Constant Definitions Used in the EPWM Code Examples* to define the headers.



A. PWM period = 2 × TBPRD × T_{TBCLK}
B. Duty modulation for EPWMxA is set by CMPA, and is active low, that is, low time duty proportional to CMPA.
C. Duty modulation for EPWMxB is set by CMPB and is active high, that is, high time duty proportional to CMPB.
D. Outputs EPWMx can drive upper/lower (complementary) power switches.
E. Dead-band = CMPB - CMPA (fully programmable edge placement by software). Note the dead-band module is also available if the more classical edge delay method is required.

Figure 12-201. Up-Down-Count, Dual Edge Symmetric Waveform, With Independent Modulation on EPWMxA and EPWMxB — Complementary

Copyright © 2024 Texas Instruments Incorporated

Table 12-270. EPWMx Initialization for Figure 12-201

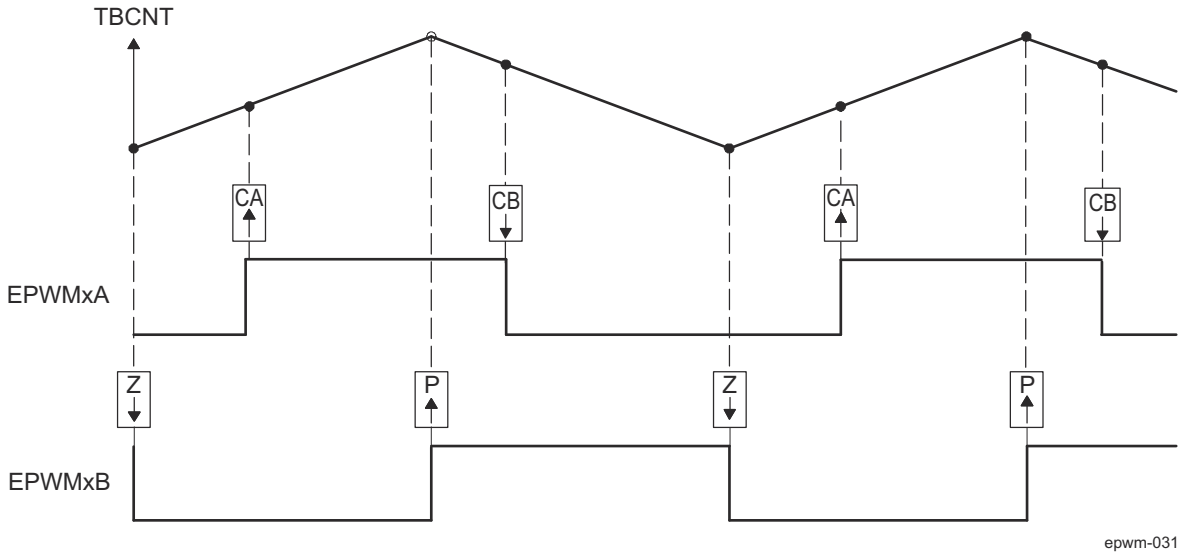
Register	Bitfield	Value	Comments
EPWM_TBPRD	TBPRD	600 (258h)	Period = 601 TBCLK counts
EPWM_TBPHS	TBPHS	0	Clear Phase Register to 0
EPWM_TBCNT	TBCNT	0	Clear TB counter
EPWM_TBCTL	CTRMODE	TB_UPDOWN	
	PHSEN	TB_DISABLE	Phase loading disabled
	PRDLD	TB_SHADOW	
	SYNCOSEL	TB_SYNC_DISABLE	
	HSPCLKDIV	TB_DIV1	TBCLK = FICLK
	CLKDIV	TB_DIV1	
EPWM_CMPA	CMPA	350 (15Eh)	Compare A = 350 TBCLK counts
EPWM_CMPB	CMPB	400 (190h)	Compare B = 400 TBCLK counts
EPWM_CMPCTL	SHDWAMODE	CC_SHADOW	
	SHDWBMODE	CC_SHADOW	
	LOADAMODE	CC_CTR_ZERO	Load on TBCNT = 0
	LOADBMODE	CC_CTR_ZERO	Load on TBCNT = 0
EPWM_AQCTLA	CAU	AQ_SET	
	CAD	AQ_CLEAR	
EPWM_AQCTLB	CBU	AQ_CLEAR	
	CBD	AQ_SET	

Table 12-271. EPWMx Run Time Changes for Figure 12-201

Register	Bitfield	Value	Comments
EPWM_CMPA	CMPA	Duty1A	Adjust duty for output EPWM1A
EPWM_CMPB	CMPB	Duty1B	Adjust duty for output EPWM1B

Copyright © 2024 Texas Instruments Incorporated

Table 12-272 and Table 12-273 contains initialization and runtime register configurations for the waveforms in Figure 12-202. Use the code in *Constant Definitions Used in the EPWM Code Examples* to define the headers.



A. PWM period = 2 × TBPRD × TBCLK
B. Rising edge and falling edge can be asymmetrically positioned within a PWM cycle. This allows for pulse placement techniques.
C. Duty modulation for EPWMxA is set by CMPA and CMPB.
D. Low time duty for EPWMxA is proportional to (CMPA + CMPB).
E. To change this example to active high, CMPA and CMPB actions need to be inverted (that is, Set ! Clear and Clear Set).
F. Duty modulation for EPWMxB is fixed at 50% (utilizes spare action resources for EPWMxB).

Figure 12-202. Up-Down-Count, Dual Edge Asymmetric Waveform, With Independent Modulation on EPWMxA — Active Low

Copyright © 2024 Texas Instruments Incorporated

Table 12-272. EPWMx Initialization for Figure 12-202

Register	Bitfield	Value	Comments
EPWM_TBPRD	TBPRD	600 (258h)	Period = 601 TBCLK counts
EPWM_TBPHS	TBPHS	0	Clear Phase Register to 0
EPWM_TBCNT	TBCNT	0	Clear TB counter
EPWM_TBCTL	CTRMODE	TB_UPDOWN	
	PHSEN	TB_DISABLE	Phase loading disabled
	PRDLD	TB_SHADOW	
	SYNCOSEL	TB_SYNC_DISABLE	
	HSPCLKDIV	TB_DIV1	TBCLK = FICLK
	CLKDIV	TB_DIV1	
EPWM_CMPA	CMPA	250 (FAh)	Compare A = 250 TBCLK counts
EPWM_CMPB	CMPB	450 (1C2h)	Compare B = 450 TBCLK counts
EPWM_CMPCTL	SHDWAMODE	CC_SHADOW	
	SHDWBMODE	CC_SHADOW	
	LOADAMODE	CC_CTR_ZERO	Load on TBCNT = 0
	LOADBMODE	CC_CTR_ZERO	Load on TBCNT = 0
EPWM_AQCTLA	CAU	AQ_SET	
	CBD	AQ_CLEAR	
EPWM_AQCTLB	ZRO	AQ_CLEAR	
	PRD	AQ_SET	

Table 12-273. EPWMx Run Time Changes for Figure 12-202

Register	Bitfield	Value	Comments
EPWM_CMPA	CMPA	EdgePosA	Adjust duty for output EPWM1A
EPWM_CMPB	CMPB	EdgePosB	Adjust duty for output EPWM1B

Copyright © 2024 Texas Instruments Incorporated

12.4.2.3.5 EPWM Dead-Band Generator (DB) Submodule

This section describes the Dead-Band Generator (DB) submodule in the PWM module.

12.4.2.3.5.1 Overview

Figure 12-203 illustrates the dead-band generator submodule within the EPWM module.

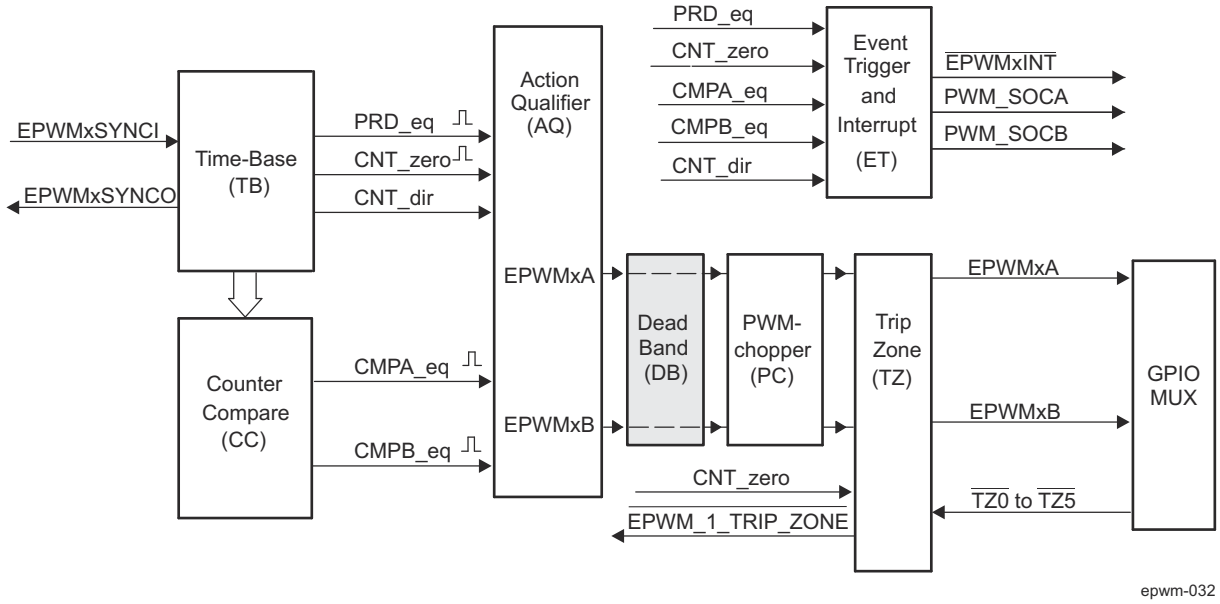


Figure 12-203. Dead-Band Generator Submodule

The "Action-qualifier (AQ) Module" section discussed how it is possible to generate the required dead-band by having full control over edge placement using both the CMPA and CMPB resources of the EPWM module. However, if the more classical edge delay-based dead-band with polarity control is required, then the dead-band generator submodule should be used.

The key functions of the dead-band generator submodule are:

- Generating appropriate signal pairs (EPWMxA and EPWMxB) with dead-band relationship from a single EPWMxA input
- Programming signal pairs for:
  - Active high (AH)
  - Active low (AL)
  - Active high complementary (AHC)
  - Active low complementary (ALC)
- Adding programmable delay to rising edges (RED)
- Adding programmable delay to falling edges (FED)
- Can be totally bypassed from the signal path (note dotted lines in Figure 12-203)

12.4.2.3.5.2 Controlling and Monitoring the EPWM Dead-Band Submodule

The dead-band generator submodule operation is controlled and monitored via the following registers:

Table 12-274. Dead-Band Generator Submodule Registers

Acronym	Register Description	Address Offset	Shadowed
EPWM_DBCTL	Dead-Band Control Register	1Eh	No
EPWM_DBRED	Dead-Band Rising Edge Delay Count Register	20h	No
EPWM_DBFED	Dead-Band Falling Edge Delay Count Register	22h	No

12.4.2.3.5.3 Operational Highlights for the EPWM Dead-Band Generator Submodule

The dead-band submodule has two groups of independent selection options as shown in Figure 12-204.

Copyright © 2024 Texas Instruments Incorporated

- **Input Source Selection:** The input signals to the dead-band module are the EPWMxA and EPWMxB output signals from the action-qualifier. In this section they will be referred to as EPWMxA In and EPWMxB In. Using the EPWM_DBCTL[5-4] IN_MODE control bits, the signal source for each delay, falling-edge or rising-edge, can be selected:
  - EPWMxA In is the source for both falling-edge and rising-edge delay. This is the default mode.
  - EPWMxA In is the source for falling-edge delay, EPWMxB In is the source for rising-edge delay.
  - EPWMxA In is the source for rising edge delay, EPWMxB In is the source for falling-edge delay.
  - EPWMxB In is the source for both falling-edge and rising-edge delay.
- **Output Mode Control:** The output mode is configured by way of the EPWM_DBCTL[1-0] OUT_MODE bit fields. These bits determine if the falling-edge delay, rising-edge delay, neither, or both are applied to the input signals.
- **Polarity Control:** The polarity control (EPWM_DBCTL[3-2] POLSEL) allows to be specified whether the rising-edge delayed signal and/or the falling-edge delayed signal is to be inverted before being sent out of the dead-band submodule.

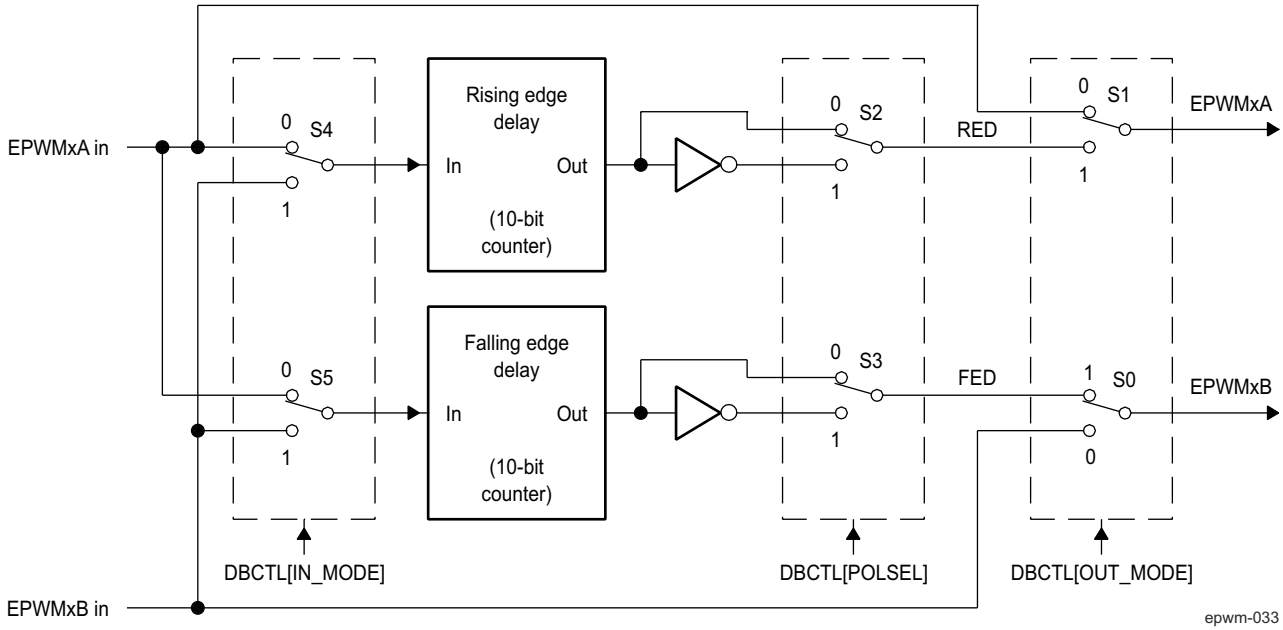


Figure 12-204. Configuration Options for the EPWM Dead-Band Generator Submodule

Copyright © 2024 Texas Instruments Incorporated

Although all combinations are supported, not all are typical usage modes. Table 12-275 lists some classical dead-band configurations. These modes assume that the EPWM_DBCTL[5-4] IN_MODE is configured such that EPWMxA In is the source for both falling-edge and rising-edge delay. Enhanced, or non-traditional modes can be achieved by changing the input signal source. The modes shown in Table 12-275 fall into the following categories:

- **Mode 1: Bypass both falling-edge delay (FED) and rising-edge delay (RED)** Allows to be fully disabled the dead-band submodule from the PWM signal path.
- **Mode 2-5: Classical Dead-Band Polarity Settings** These represent typical polarity configurations that should address all the active high/low modes required by available industry power switch gate drivers. The waveforms for these typical cases are shown in Figure 12-205. Note that to generate equivalent waveforms to Figure 12-205, configure the action-qualifier submodule to generate the signal as shown for EPWMxA.
- **Mode 6: Bypass rising-edge-delay and Mode 7: Bypass falling-edge-delay** Finally the last two entries in Table 12-275 show combinations where either the falling-edge-delay (FED) or rising-edge-delay (RED) blocks are bypassed.

Table 12-275. Classical Dead-Band Operating Modes

		EPWM_DBCTL[3-2] POLSEL		EPWM_DBCTL[1-0] OUT_MODE	
Mode	**Mode Description** (1)	**S3**	**S2**	**S1**	**S0**
1	EPWMxA and EPWMxB Passed Through (No Delay)	x	x	0	0
2	Active High Complementary (AHC)	1	0	1	1
3	Active Low Complementary (ALC)	0	1	1	1
4	Active High (AH)	0	0	1	1
5	Active Low (AL)	1	1	1	1
6	EPWMxA Out = EPWMxA In (No Delay) EPWMxB Out = EPWMxA In with Falling Edge Delay	0 or 1	0 or 1	0	1
7	EPWMxA Out = EPWMxA In with Rising Edge Delay EPWMxB Out = EPWMxB In with No Delay	0 or 1	0 or 1	1	0

(1) These are classical dead-band modes and assume that EPWM_DBCTL[5-4] IN_MODE = 0b00. That is, EPWMxA in is the source for both the falling-edge and rising-edge delays. Enhanced, non-traditional modes can be achieved by changing the IN_MODE configuration.

Figure 12-205 shows waveforms for typical cases where 0% < duty < 100%.

Copyright © 2024 Texas Instruments Incorporated

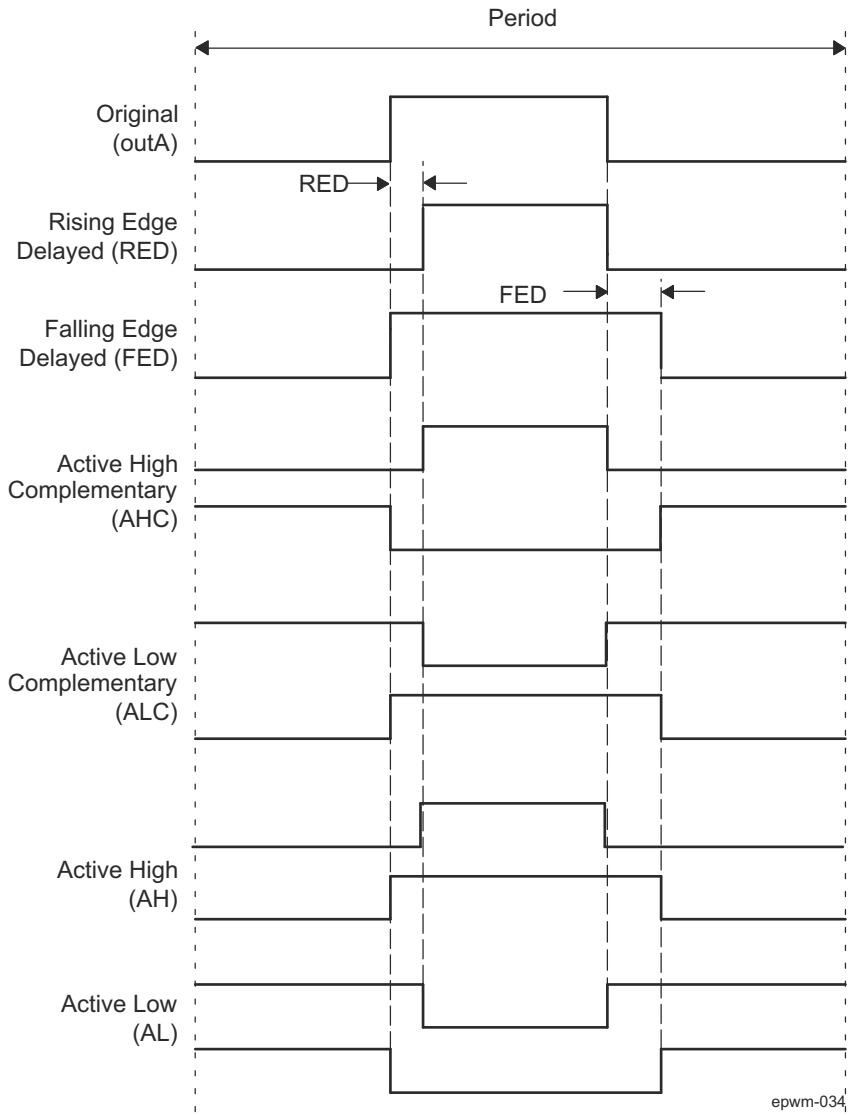


Figure 12-205. Dead-Band Waveforms for Typical Cases (0% < Duty < 100%)

The dead-band submodule supports independent values for rising-edge (RED) and falling-edge (FED) delays. The amount of delay is programmed using the EPWM_DBRED and EPWM_DBFED registers. These are 10-bit registers and their value represents the number of time-base clock, TBCLK, periods a signal edge is delayed by. For example, the formulas to calculate falling-edge-delay and rising-edge-delay are:

FED = EPWM_DBFED × T_{TBCLK}

RED = EPWM_DBRED × T_{TBCLK}

Where T_{TBCLK} is the period of TBCLK, the prescaled version of FICLK.

Copyright © 2024 Texas Instruments Incorporated

12.4.2.3.6 EPWM-Chopper (PC) Submodule

This section describes the PWM-Chopper (PC) submodule in the PWM module.

12.4.2.3.6.1 Overview

Figure 12-206 illustrates the PWM-chopper (PC) submodule within the EPWM module. The PWM-chopper submodule allows a high-frequency carrier signal to modulate the PWM waveform generated by the action-qualifier and dead-band submodules. This capability is important if a pulse transformer-based gate drivers to control the power switching elements is needed.

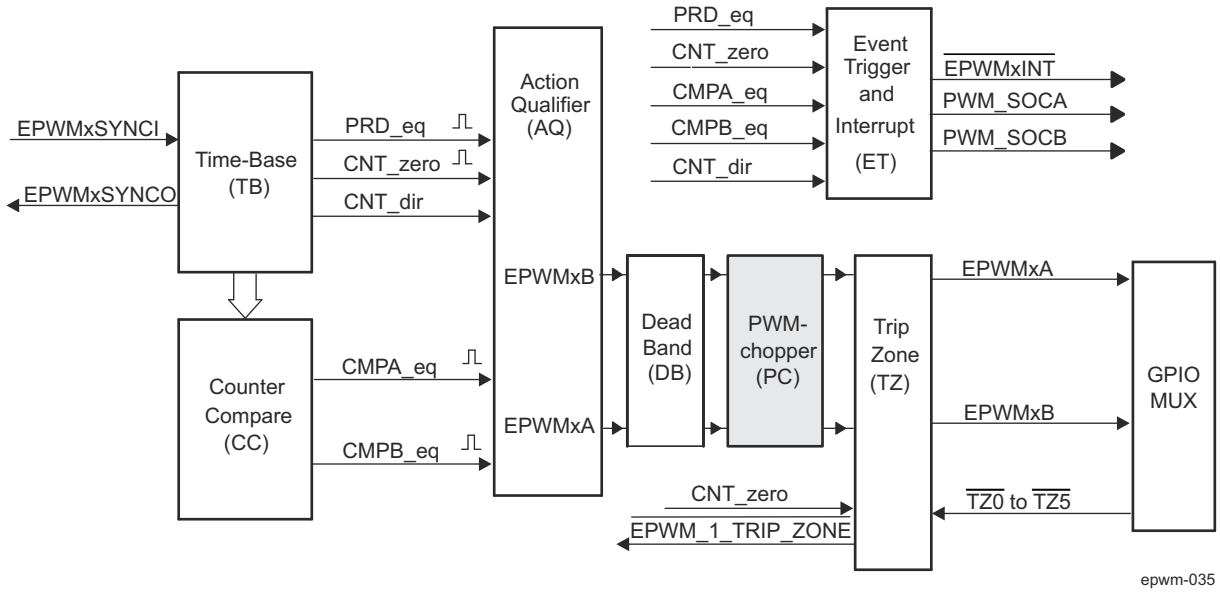


Figure 12-206. PWM-Chopper Submodule

PC module key features:

- Programmable chopping (carrier) frequency
- Programmable pulse width of first pulse
- Programmable duty cycle of second and subsequent pulses
- Can be fully bypassed if not required

12.4.2.3.6.2

PC module key features:

- Programmable chopping (carrier) frequency
- Programmable pulse width of first pulse
- Programmable duty cycle of second and subsequent pulses
- Can be fully bypassed if not required

12.4.2.3.6.3 Controlling the EPWM-Chopper Submodule

The EPWM-chopper submodule operation is controlled via the register in Table 12-276.

Table 12-276. EPWM-Chopper Submodule Registers

Acronym	Register Description	Address Offset	Shadowed
EPWM_PCCTL	PWM-chopper Control Register	3Ch	No

Copyright © 2024 Texas Instruments Incorporated

12.4.2.3.6.4 Operational Highlights for the EPWM-Chopper Submodule

Figure 12-207 shows the operational details of the EPWM-chopper submodule. The carrier clock is derived from FICLK. Its frequency and duty cycle are controlled via the CHPFREQ and CHPDUTY bits in the EPWM_PCCTL register. The one-shot block is a feature that provides a high energy first pulse to ensure hard and fast power switch turn on, while the subsequent pulses sustain pulses, ensuring the power switch remains on. The one-shot width is programmed via the OSHTWTH bits. The PWM-chopper submodule can be fully disabled (bypassed) via the CHPEN bit.

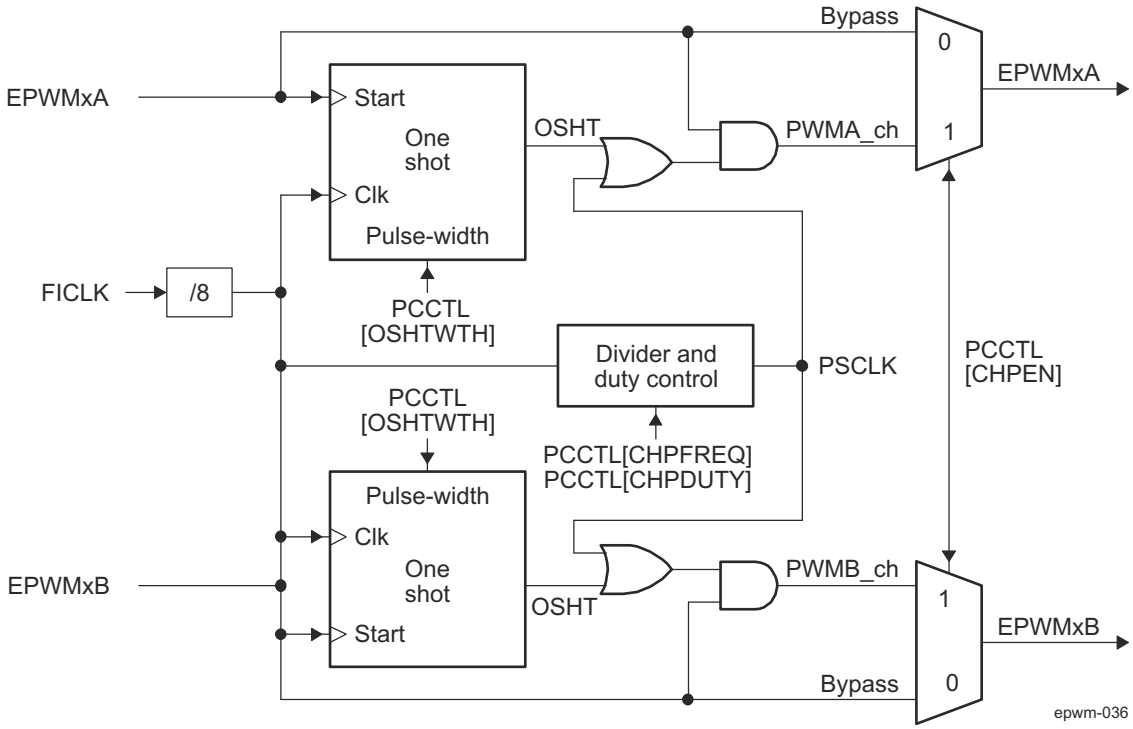


Figure 12-207. PWM-Chopper Submodule Signals and Registers

Copyright © 2024 Texas Instruments Incorporated

12.4.2.3.6.5 EPWM-Chopper Waveforms

Figure 12-208 shows simplified waveforms of the chopping action only; one-shot and duty-cycle control are not shown. Details of the one-shot and duty-cycle control are discussed in the following sections.

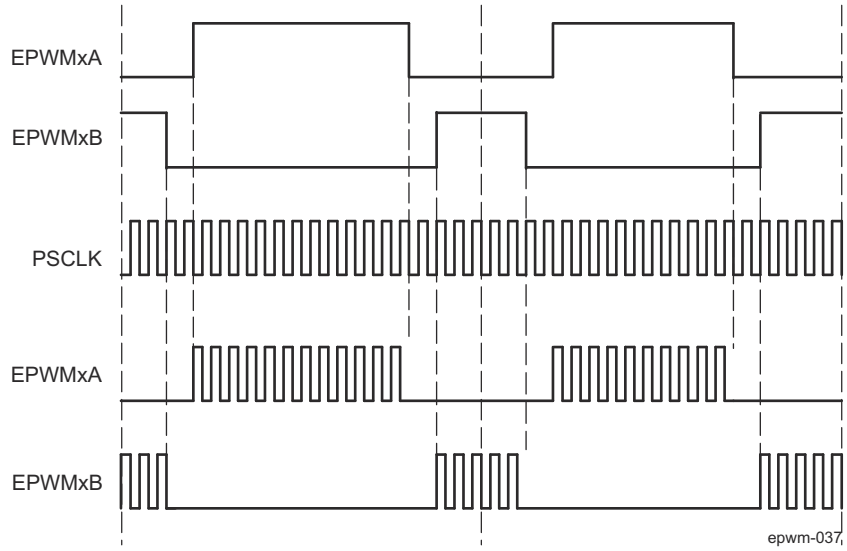


Figure 12-208. Simple EPWM-Chopper Submodule Waveforms Showing Chopping Action Only

12.4.2.3.6.5.1 EPWM-Chopper One-Shot Pulse

The width of the first pulse can be programmed to any of 16 possible pulse width values. The width or period of the first pulse is given by:

$T_{1stpulse} = T_{FICLK} \times 8 \times OSHTWTH$

Where T_{FICLK} is the period of the system clock (FICLK) and OSHTWTH is the four control bits (value from 1 to 16)

Figure 12-209 shows the first and subsequent sustaining pulses.

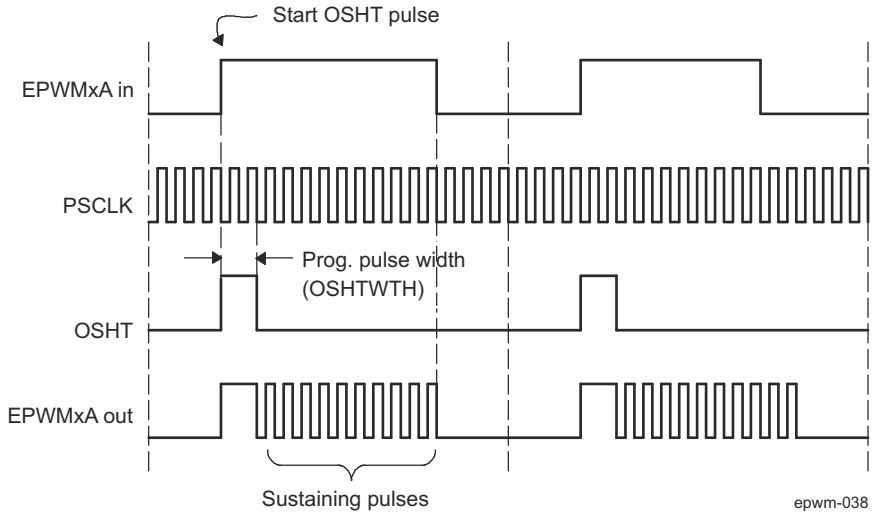


Figure 12-209. EPWM-Chopper Submodule Waveforms Showing the First Pulse and Subsequent Sustaining Pulses

Copyright © 2024 Texas Instruments Incorporated

12.4.2.3.6.5.2 EPWM-Chopper Duty Cycle Control

Pulse transformer-based gate drive designs need to comprehend the magnetic properties or characteristics of the transformer and associated circuitry. Saturation is one such consideration. To assist the gate drive designer, the duty cycles of the second and subsequent pulses have been made programmable. These sustaining pulses ensure the correct drive strength and polarity is maintained on the power switch gate during the on period, and hence a programmable duty cycle allows a design to be tuned or optimized via software control.

Figure 12-210 shows the duty cycle control that is possible by programming the CHPDUTY bits. One of seven possible duty ratios can be selected ranging from 12.5 to 87.5%.

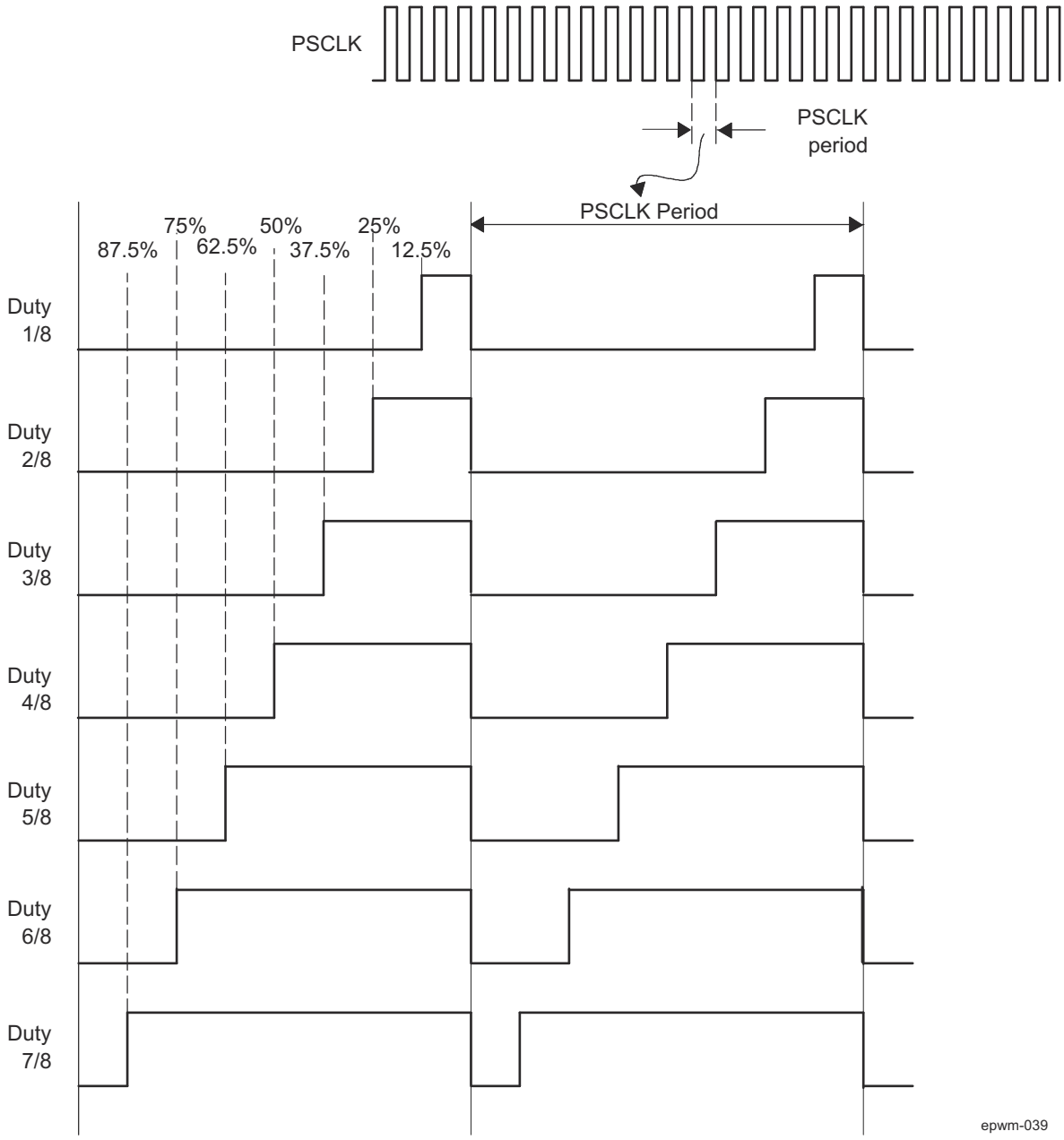


Figure 12-210. EPWM-Chopper Submodule Waveforms Showing the Pulse Width (Duty Cycle) Control of Sustaining Pulses

12.4.2.3.7 EPWM Trip-Zone (TZ) Submodule

This section describes the Trip-Zone (TZ) submodule in the PWM module.

Copyright © 2024 Texas Instruments Incorporated

12.4.2.3.7.1 Overview

Figure 12-211 shows how the trip-zone (TZ) submodule fits within the EPWM module. Each EPWM module is connected to six $\overline{TZ}$ signals ($\overline{TZ0}$ to $\overline{TZ5}$) that are sourced from the GPIO MUX. These signals indicates external fault or trip conditions, and the EPWM outputs can be programmed to respond accordingly when faults occur. See *EPWM Integration* to determine the number of trip-zone pins available for the device.

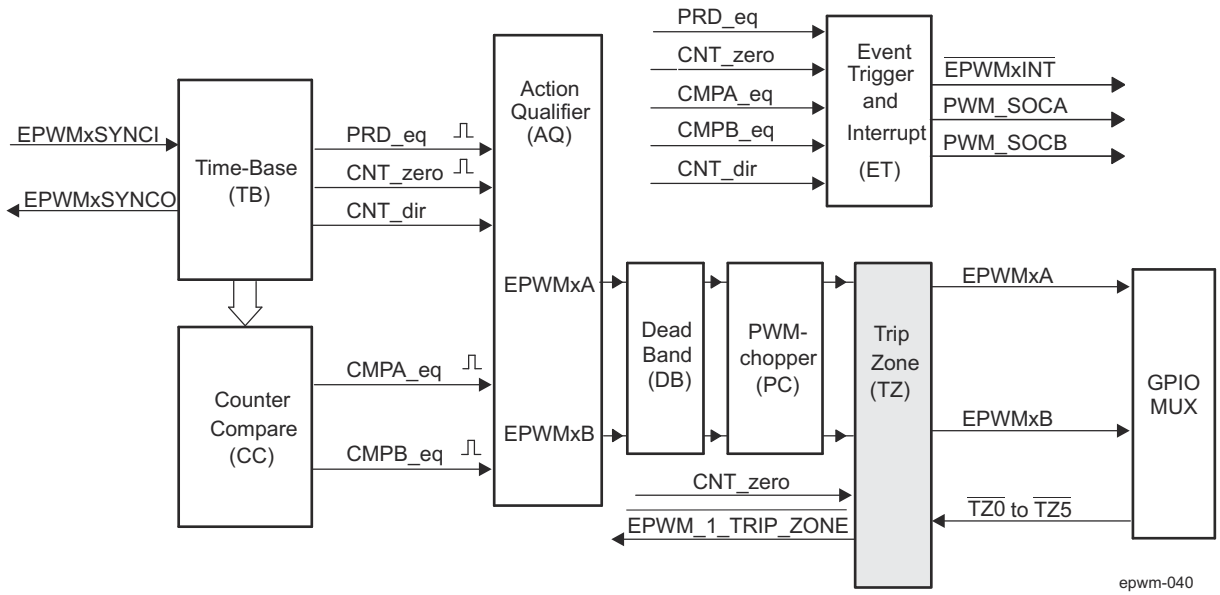


Figure 12-211. EPWM Trip-Zone Submodule

The key functions of the trip-zone submodule are:

- Trip inputs $\overline{TZ0}$ to $\overline{TZ5}$ can be flexibly mapped to any EPWM module.
- Upon a fault condition, outputs EPWMxA and EPWMxB can be forced to one of the following:
  - High
  - Low
  - High-impedance
  - No action taken
- Support for one-shot trip (OSHT) for major short circuits or over-current conditions.
- Support for cycle-by-cycle tripping (CBC) for current limiting operation.
- Each trip-zone input pin can be allocated to either one-shot or cycle-by-cycle operation.
- Interrupt generation is possible on any trip-zone pin.
- Software-forced tripping is also supported.
- The trip-zone submodule can be fully bypassed if it is not required.

Note

For each EPWMx module, 6 tripzone input events are supported ($\overline{EPWMx_TRIP_TZ[5:0]}$). Each tripzone input for all EPWMx modules (where x = 0 to 5) are tied to a common tripzone input pin, so that any EPWMx can be triggered by any of the six tripzone inputs.

Copyright © 2024 Texas Instruments Incorporated

12.4.2.3.7.2 Controlling and Monitoring the EPWM Trip-Zone Submodule

The trip-zone submodule operation is controlled and monitored through the following registers:

Table 12-277. EPWM Trip-Zone Submodule Registers

Acronym	Register Description	Address Offset	Shadowed
EPWM_TZSEL	Trip-Zone Select Register (1)	24h	No
EPWM_TZCTL	Trip-Zone Control Register (1)	28h	No
EPWM_TZEINT	Trip-Zone Enable Interrupt Register (1)	2Ah	No
EPWM_TZFLG	Trip-Zone Flag Register (1)	2Ch	No
EPWM_TZCLR	Trip-Zone Clear Register (1)	2Eh	No
EPWM_TZFRC	Trip-Zone Force Register (1)	30h	No

(1) All trip-zone registers are EALLOW protected and can be modified only after executing the EALLOW instruction.

12.4.2.3.7.3 Operational Highlights for the EPWM Trip-Zone Submodule

The trip-zone signals at pin $\overline{TZ0}$ to $\overline{TZ5}$ is an active-low input signal. When the pin goes low, it indicates that a trip event has occurred. Each EPWM module can be individually configured to ignore or use each of the trip-zone pins. Which trip-zone pins are used by a particular EPWM module is determined by the EPWM_TZSEL register for that specific EPWM module. The trip-zone signal may or may not be synchronized to the system clock (FICLK). A minimum of 1 FICLK low pulse on the $\overline{TZ0}$ to $\overline{TZ5}$ inputs is sufficient to trigger a fault condition in the EPWM module. The asynchronous trip makes sure that if clocks are missing for any reason, the outputs can still be tripped by a valid event present on the $\overline{TZ0}$ to $\overline{TZ5}$ inputs.

The $\overline{TZ0}$ to $\overline{TZ5}$ inputs can be individually configured to provide either a cycle-by-cycle or one-shot trip event for a EPWM module. The configuration is determined by the EPWM_TZSEL[5-0] CBCk and EPWM_TZSEL[13-8] OSHTk bit field (where k = 0 to 5 corresponds to the trip pin), respectively.

- **Cycle-by-Cycle (CBC):** When a cycle-by-cycle trip event occurs, the action specified in the EPWM_TZCTL register is carried out immediately on the EPWMxA and/or EPWMxB output. Table 12-278 lists the possible actions. In addition, the cycle-by-cycle trip event flag (EPWM_TZFLG[1] CBC) is set and a EPWMxTZINT interrupt is generated if it is enabled in the EPWM_TZEINT register.

  The specified condition on the pins is automatically cleared when the EPWM time-base counter reaches zero (EPWM_TBCNT[15-0] TBCNT = 0000h) if the trip event is no longer present. Therefore, in this mode, the trip event is cleared or reset every PWM cycle. The EPWM_TZFLG[1] CBC flag bit will remain set until it is manually cleared by writing to the EPWM_TZCLR[1] CBC bit. If the cycle-by-cycle trip event is still present when the EPWM_TZFLG[1] CBC bit is cleared, then it will again be immediately set.
- **One-Shot (OSHT):** When a one-shot trip event occurs, the action specified in the EPWM_TZCTL register is carried out immediately on the EPWMxA and/or EPWMxB output. Table 12-278 lists the possible actions. In addition, the one-shot trip event flag (EPWM_TZFLG[2] OST) is set and a EPWMxTZINT interrupt is generated if it is enabled in the EPWM_TZEINT register. The one-shot trip condition must be cleared manually by writing to the EPWM_TZCLR[2] OST bit.

The action taken when a trip event occurs can be configured individually for each of the EPWM output pins by way of the EPWM_TZCTL[1-0] TZA and EPWM_TZCTL[3-2] TZB register bit field. One of four possible actions, shown in Table 12-278, can be taken on a trip event.

Copyright © 2024 Texas Instruments Incorporated

Table 12-278. Possible Actions On an EPWM Trip Event

EPWM_TZCTL[1-0] TZA and/or EPWM_TZCTL[3-2] TZB	EPWMxA and/or EPWMxB	Comment
0	High-Impedance	Tripped
1h	Force to High State	Tripped
2h	Force to Low State	Tripped
3h	No Change	Do Nothing. No change is made to the output.

Example of EPWM Trip-Zone Configurations

A one-shot trip event on $\overline{TZ0}$ pulls both EPWM1A, EPWM1B low and also forces EPWM2A and EPWM2B high.

- Configure the EPWM1 registers as follows:
  - EPWM_TZSEL[8] OSHT1 = 1: enables $\overline{TZ}$ as a one-shot event source for EPWM1
  - EPWM_TZCTL[1-0] TZA = 2: EPWM1A will be forced low on a trip event
  - EPWM_TZCTL[3-2] TZB = 2: EPWM1B will be forced low on a trip event
- Configure the EPWM2 registers as follows:
  - EPWM_TZSEL[8] OSHT1 = 1: enables $\overline{TZ}$ as a one-shot event source for EPWM2
  - EPWM_TZCTL[1-0] TZA = 1: EPWM2A will be forced high on a trip event
  - EPWM_TZCTL[3-2] TZB = 1: EPWM2B will be forced high on a trip event

12.4.2.3.7.4 Generating EPWM Trip-Event Interrupts

Figure 12-212 and Figure 12-213 illustrate the trip-zone submodule control and interrupt logic, respectively.

Copyright © 2024 Texas Instruments Incorporated

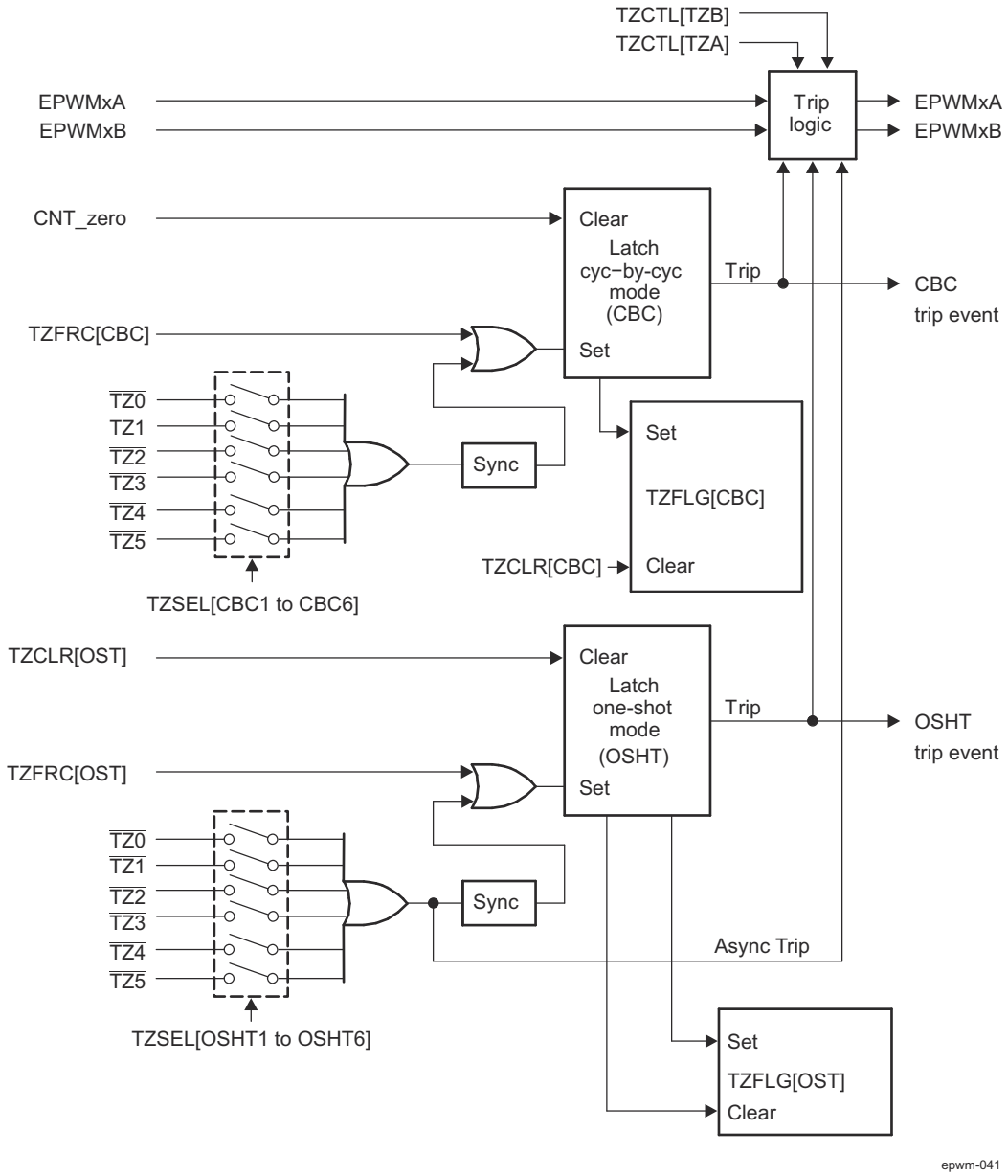


Figure 12-212. EPWM Trip-Zone Submodule Mode Control Logic

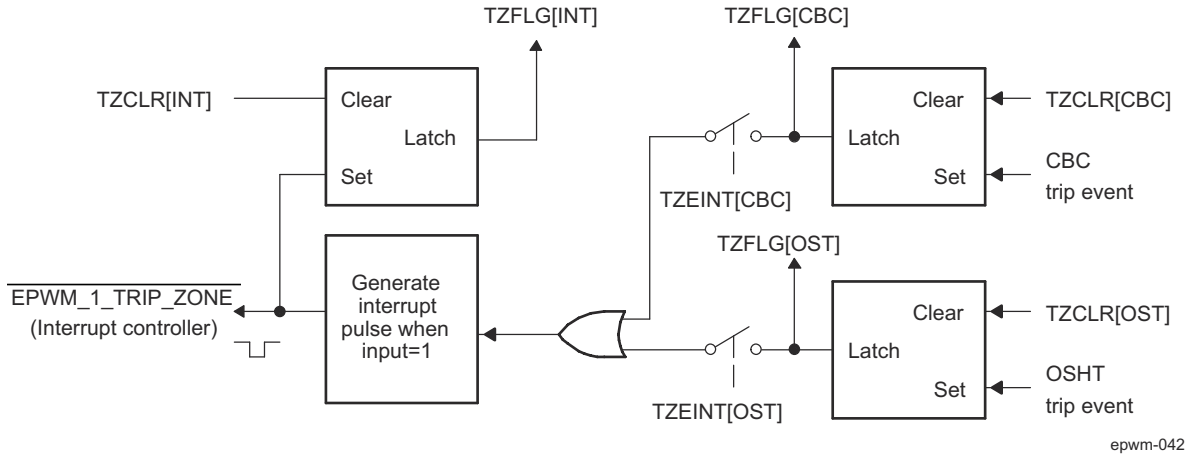


Figure 12-213. EPWM Trip-Zone Submodule Interrupt Logic

Copyright © 2024 Texas Instruments Incorporated

12.4.2.3.8 EPWM Event-Trigger (ET) Submodule

This section describes the Event-Trigger (ET) submodule in the PWM module.

12.4.2.3.8.1 Overview

Figure 12-214 shows the event-trigger (ET) submodule in the EPWM system. The event-trigger submodule manages the events generated by the time-base submodule and the counter-compare submodule to generate an aggregated interrupt request.

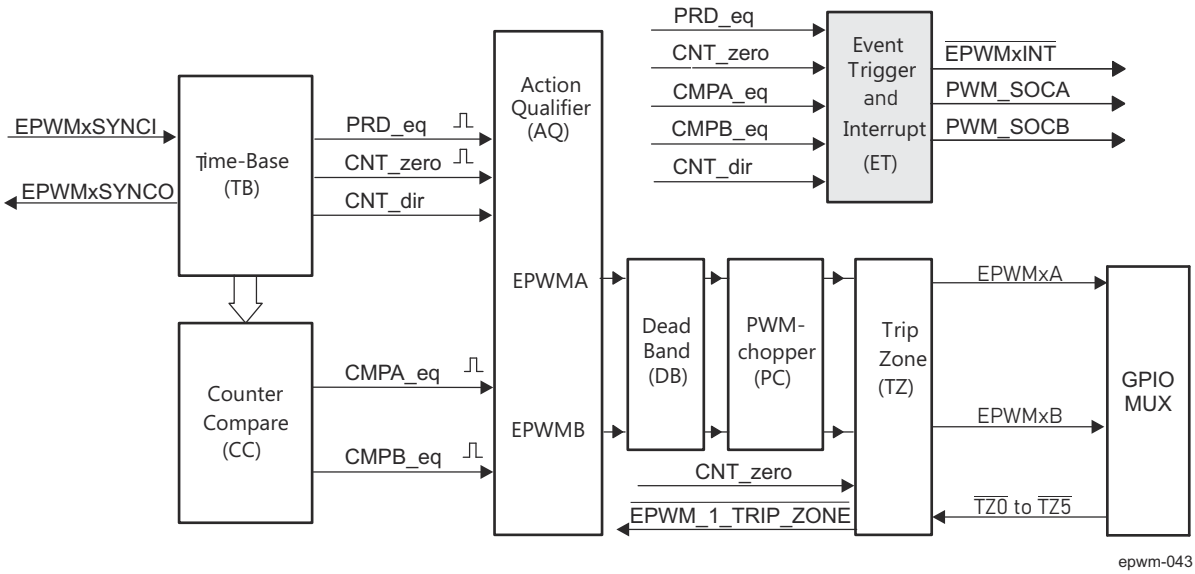


Figure 12-214. EPWM Event-Trigger Submodule

The key functions of the event-trigger submodule are:

- Receives event inputs generated by the time-base and counter-compare submodules
- Uses the time-base direction information for up/down event qualification
- Uses prescaling logic to issue interrupt requests at:
  - Every event
  - Every second event
  - Every third event
- Provides full visibility of event generation via event counters and flags

12.4.2.3.8.2 Controlling and Monitoring the EPWM Event-Trigger Submodule

The key registers used to configure the event-trigger submodule are shown in Table 12-279:

Table 12-279. EPWM Event-Trigger Submodule Registers

Acronym	Register Description	Address Offset	Shadowed
EPWM_ETSEL	Event-Trigger Selection Register	32h	No
EPWM_ETPS	Event-Trigger Prescale Register	34h	No
EPWM_ETFLG	Event-Trigger Flag Register	36h	No
EPWM_ETCLR	Event-Trigger Clear Register	38h	No
EPWM_ETFRC	Event-Trigger Force Register	3Ah	No

12.4.2.3.8.3 Operational Overview of the EPWM Event-Trigger Submodule

Each EPWM module has one interrupt request line as shown in Figure 12-215. Mapping interrupt lines to device host interrupt controllers is device specific and is covered in *EPWM Integration*.

Copyright © 2024 Texas Instruments Incorporated

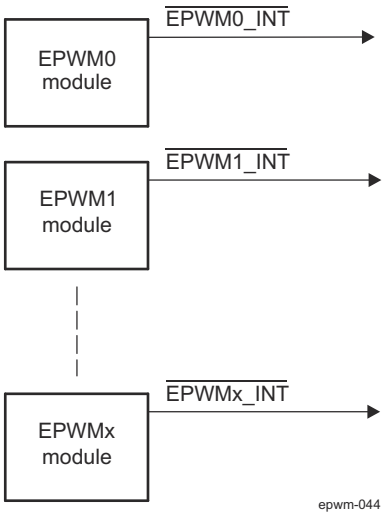


Figure 12-215. EPWM Event-Trigger Submodule Inter-Connectivity to Interrupt Controller

The event-trigger submodule monitors various event conditions (the left side inputs to event-trigger submodule shown in Figure 12-216) and can be configured to prescale these events before issuing an Interrupt request. The event-trigger prescaling logic can issue Interrupt requests at:

- Every event
- Every second event
- Every third event

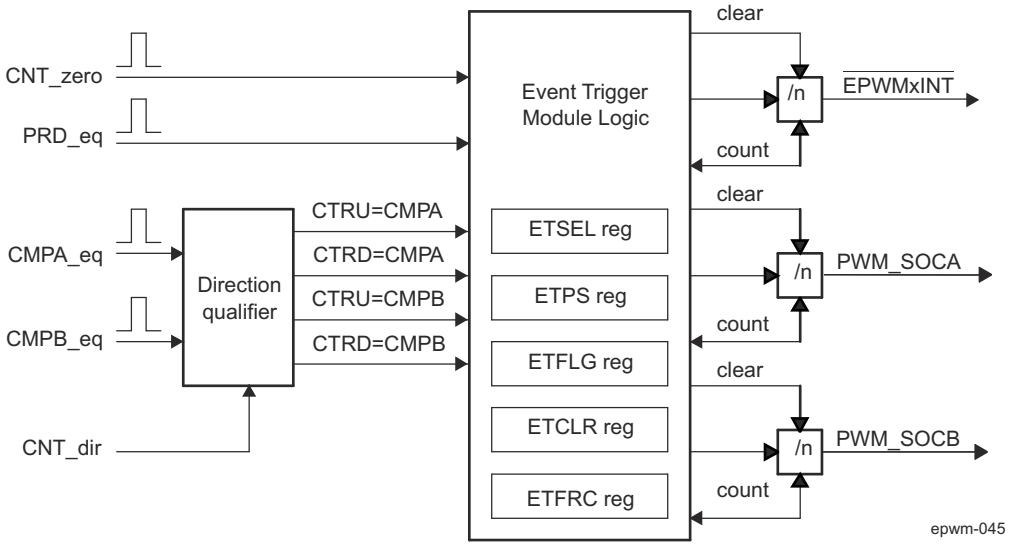


Figure 12-216. EPWM Event-Trigger Submodule Showing Event Inputs and Prescaled Outputs

Copyright © 2024 Texas Instruments Incorporated

- ETSEL — This selects which of the possible events will trigger an interrupt.
- ETPS — This programs the event prescaling options previously mentioned.
- ETFLG — These are flag bits indicating status of the selected and prescaled events.
- ETCLR — These bits allow to clear the flag bits in the EPWM_ETFLG register via software.
- ETFRC — These bits allow software forcing of an event. Useful for debugging or software intervention.

A more detailed look at how the various register bits interact with the Interrupt is shown in Figure 12-217.

Figure 12-217 shows the event-trigger's interrupt generation logic. The interrupt-period (EPWM_ETPS[1-0] INTPRD) bits specify the number of events required to cause an interrupt pulse to be generated. The choices available are:

- Do not generate an interrupt
- Generate an interrupt on every event
- Generate an interrupt on every second event
- Generate an interrupt on every third event

An interrupt cannot be generated on every fourth or more events.

Which event can cause an interrupt is configured by the interrupt selection (EPWM_ETSEL[2-0] INTSEL) bits. The event can be one of the following:

- Time-base counter equal to zero (EPWM_TBCNT[15-0] TBCNT = 0000h).
- Time-base counter equal to period (EPWM_TBCNT[15-0] TBCNT = EPWM_TBPRD[15-0] TBPRD).
- Time-base counter equal to the compare A register (EPWM_CMPA) when the timer is incrementing.
- Time-base counter equal to the compare A register (EPWM_CMPA) when the timer is decrementing.
- Time-base counter equal to the compare B register (EPWM_CMPB) when the timer is incrementing.
- Time-base counter equal to the compare B register (EPWM_CMPB) when the timer is decrementing.

The number of events that have occurred can be read from the interrupt event counter (EPWM_ETPS[3-2] INTCNT) register bits. That is, when the specified event occurs the EPWM_ETPS[3-2] INTCNT bits are incremented until they reach the value specified by the EPWM_ETPS[1-0] INTPRD bit field. When EPWM_ETPS[3-2] INTCNT = EPWM_ETPS[1-0] INTPRD the counter stops counting and its output is set. The counter is only cleared when an interrupt is sent to the interrupt controller.

When EPWM_ETPS[3-2] INTCNT reaches EPWM_ETPS[1-0] INTPRD, one of the following behaviors will occur:

- If interrupts are enabled, EPWM_ETSEL[3] INTEN = 1h and the interrupt flag is clear, EPWM_ETFLG[0] INT = 0h, then an interrupt pulse is generated and the interrupt flag is set, EPWM_ETFLG[0] INT = 1h, and the event counter is cleared EPWM_ETPS[3-2] INTCNT = 0h. The counter will begin counting events again.
- If interrupts are disabled, EPWM_ETSEL[3] INTEN = 0, or the interrupt flag is set, EPWM_ETFLG[0] INT = 1h, the counter stops counting events when it reaches the period value EPWM_ETPS[3-2] INTCNT = EPWM_ETPS[1-0] INTPRD.
- If interrupts are enabled, but the interrupt flag is already set, then the counter will hold its output high until the ENTFLG[0] INT flag is cleared. This allows for one interrupt to be pending while one is serviced.

Writing to the INTPRD bits will automatically clear the counter INTCNT = 0 and the counter output will be reset (so no interrupts are generated). Writing a 1h to the EPWM_ETFRC[0] INT bit will increment the event counter INTCNT. The counter will behave as described above when INTCNT = INTPRD. When INTPRD = 0h, the counter is disabled and hence no events will be detected and the EPWM_ETFRC[0] INT bit is also ignored.

Copyright © 2024 Texas Instruments Incorporated

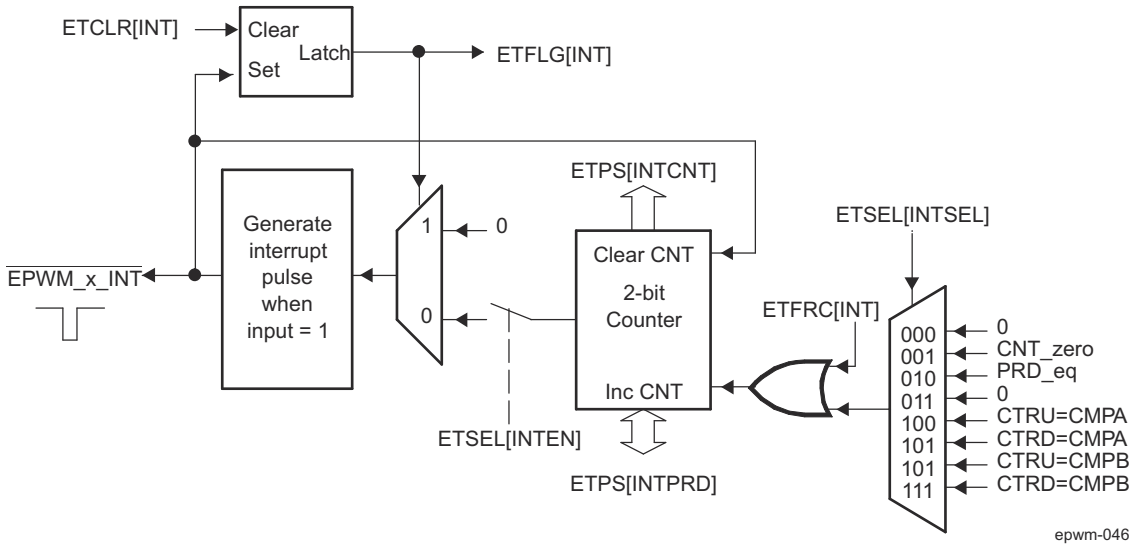


Figure 12-217. EPWM Event-Trigger Interrupt Generator

12.4.2.3.8.4

Figure 12-218 shows the operation of the event-trigger's start-of-conversion-A (SOCA) pulse generator. The EPWM_ETPS[11-10] SOCACNT counter and EPWM_ETPS[9-8] SOCAPRD period values behave similarly to the interrupt generator except that the pulses are continuously generated. That is, the pulse flag EPWM_ETFLG[2] SOCA is latched when a pulse is generated, but it does not stop further pulse generation. The enable/disable EPWM_ETSEL[11] SOCAEN bit stops pulse generation, but input events can still be counted until the period value is reached as with the interrupt generation logic. The event that will trigger an SOCA and SOCB pulse can be configured separately in the EPWM_ETSEL[10-8] SOCASEL and EPWM_ETSEL[14-12] SOCBSEL bit field. The possible events are the same events that can be specified for the interrupt generation logic.

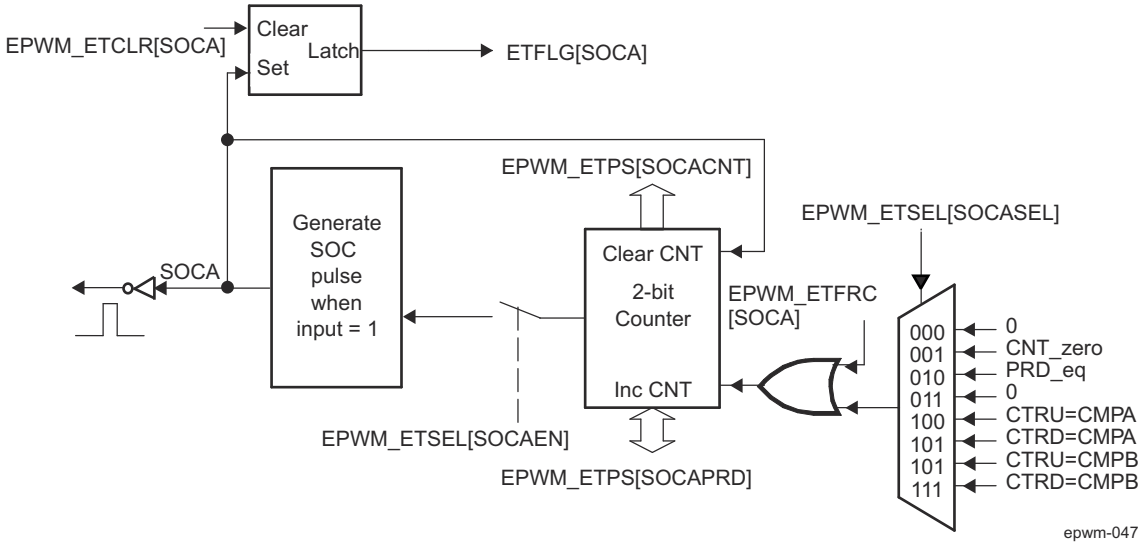


Figure 12-218. EPWM Event-Trigger SOCA Pulse Generator

Figure 12-219 shows the operation of the event-trigger's start-of-conversion-B (SOCB) pulse generator. The event-trigger's SOCB pulse generator operates the same way as the SOCA.

Copyright © 2024 Texas Instruments Incorporated

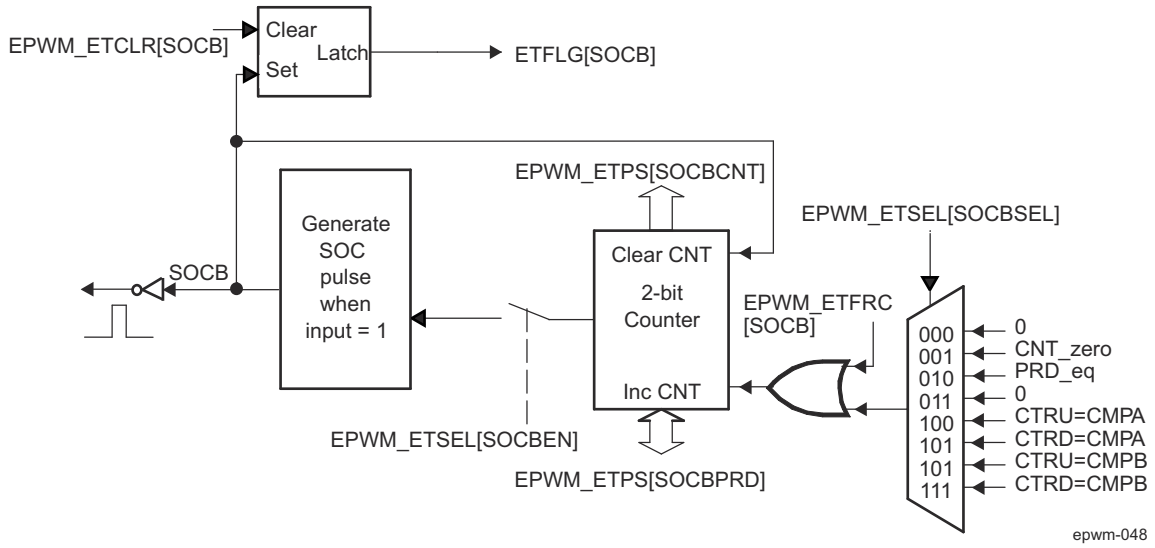


Figure 12-219. EPWM Event-Trigger SOCB Pulse Generator

Copyright © 2024 Texas Instruments Incorporated

12.4.2.3.9 EPWM High Resolution (HRPWM) Submodule

This section describes the High-Resolution PWM (HRPWM) submodule in the PWM module.

12.4.2.3.9.1 Overview

Figure 12-220 shows the high-resolution PWM (HRPWM) submodule in the EPWM system. Some devices include the high-resolution PWM submodule, see *EPWM Integration* to determine which EPWM instances include this feature.

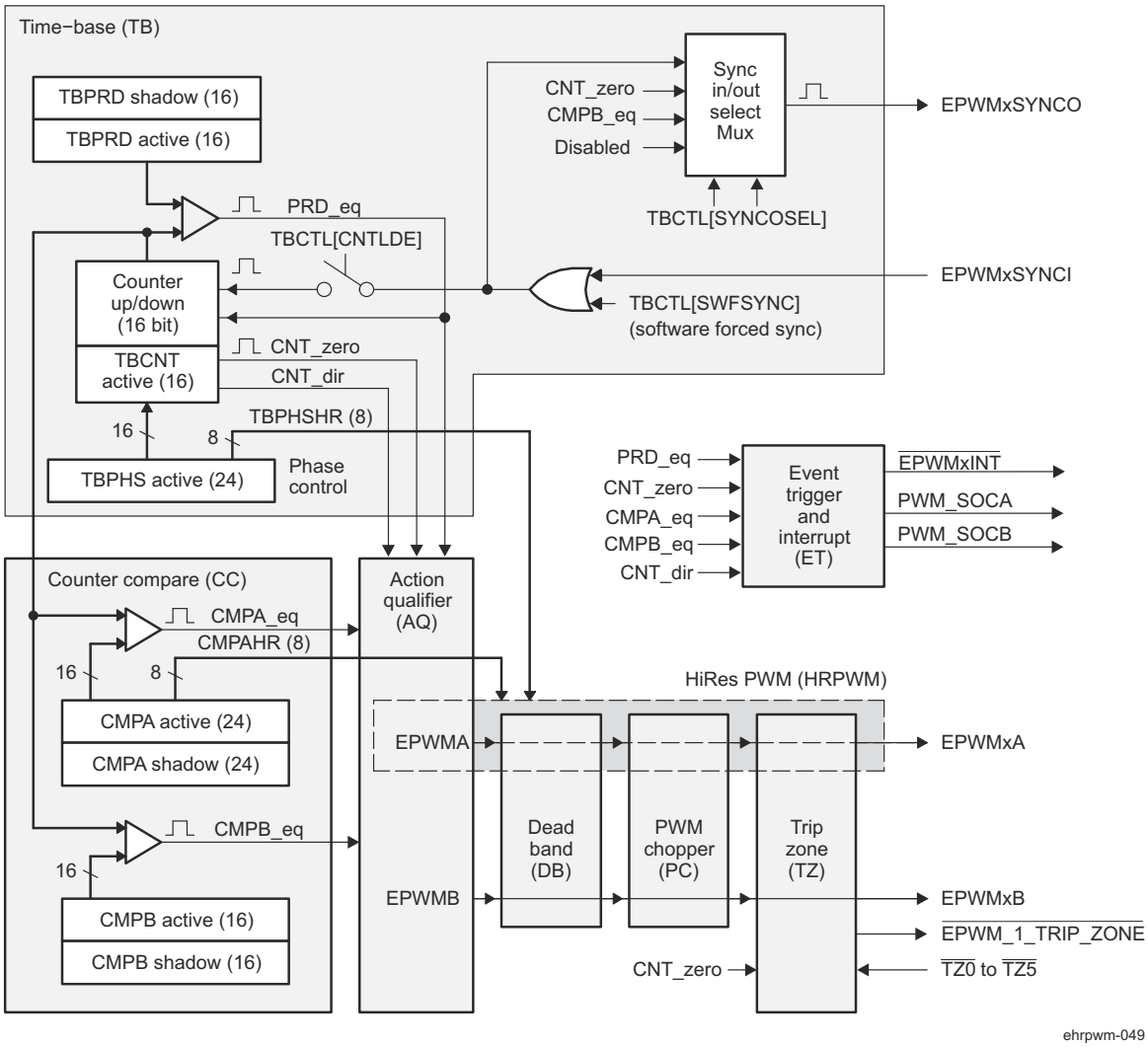


Figure 12-220. HRPWM System Interface

The high-resolution pulse-width modulator (HRPWM) extends the time resolution capabilities of the conventionally derived digital pulse-width modulator (PWM). HRPWM is typically used when PWM resolution falls below ~9-10 bits.
The key features of HRPWM are:

- Extended time resolution capability
- Used in both duty cycle and phase-shift control methods
- Finer time granularity control or edge positioning using extensions to the Compare A and Phase registers
- Implemented using the A signal path of PWM, that is, on the EPWMxA output. **EPWMxB output has conventional PWM capabilities**

Copyright © 2024 Texas Instruments Incorporated

The EPWM peripheral is used to perform a function that is mathematically equivalent to a digital-to-analog converter (DAC). As shown in Figure 12-221, the effective resolution for conventionally generated PWM is a function of PWM frequency (or period) and system clock frequency.

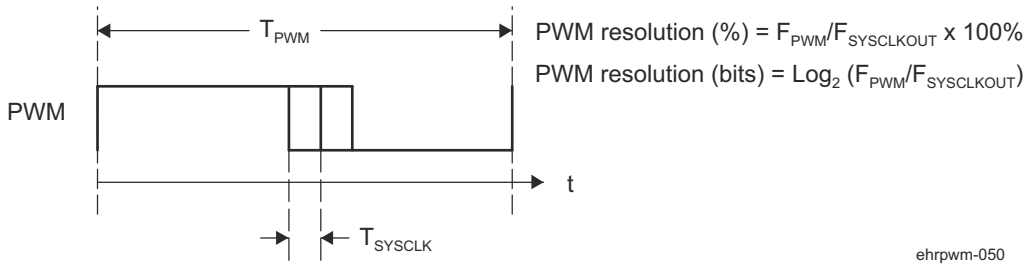


Figure 12-221. Resolution Calculations for Conventionally Generated PWM

Use HRPWM when the required PWM operating frequency does not offer sufficient resolution in PWM mode. As an example of improved performance offered by HRPWM, Table 12-280 shows resolution in bits for various PWM frequencies. Table 12-280 values assume a MEP step size of 180 ps. See the device-specific Datasheet for typical and maximum performance specifications for the MEP.

Table 12-280. Resolution for PWM and HRPWM

	Regular Resolution (PWM)		High Resolution (HRPWM)	
PWM Frequency (kHz)	Bits	%	Bits	%
20	12.3	0.0	18.1	0.000
50	11.0	0.0	16.8	0.001
100	10.0	0.1	15.8	0.002
150	9.4	0.2	15.2	0.003
200	9.0	0.2	14.8	0.004
250	8.6	0.3	14.4	0.005
500	7.6	0.5	13.8	0.007
1000	6.6	1.0	12.4	0.018
1500	6.1	1.5	11.9	0.027
2000	5.6	2.0	11.4	0.036

Although each application may differ, typical low-frequency PWM operation (below 250 kHz) may not require HRPWM. HRPWM capability is most useful for high-frequency PWM requirements of power conversion topologies such as:

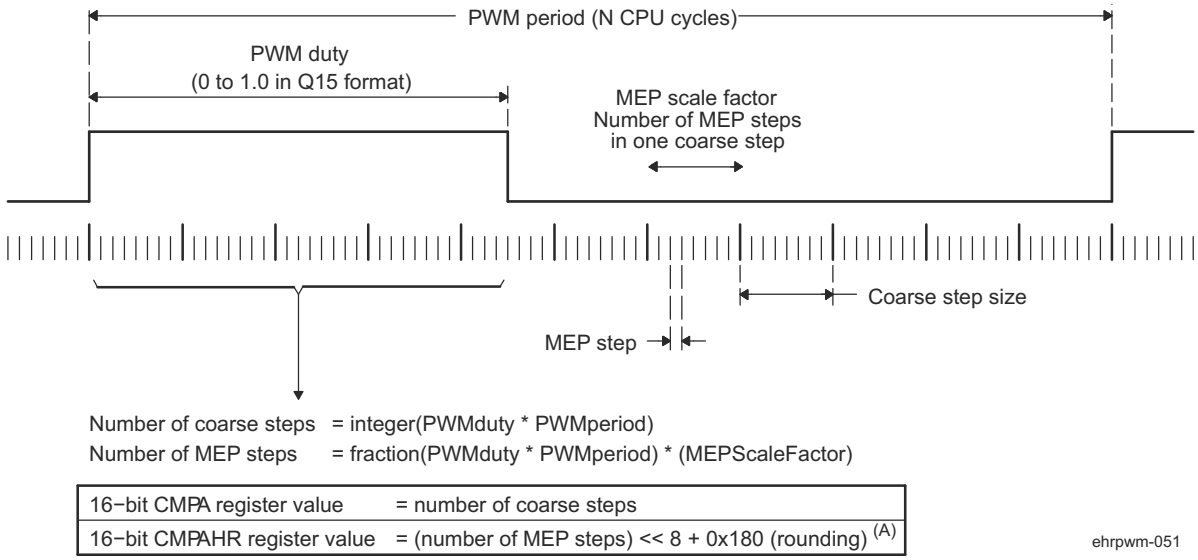
- Single-phase buck, boost, and flyback
- Multi-phase buck, boost, and flyback
- Phase-shifted full bridge
- Direct modulation of D-Class power amplifiers

12.4.2.3.9.2 Architecture of the High-Resolution PWM Submodule

The HRPWM is based on micro edge positioner (MEP) technology. MEP logic is capable of positioning an edge very finely by sub-dividing one coarse system clock of a conventional PWM generator. The time step accuracy is on the order of 150 ps. The HRPWM also has a self-check software diagnostics mode to check if the MEP logic is running optimally, under all operating conditions.

Figure 12-222 shows the relationship between one coarse system clock and edge position in terms of MEP steps, which are controlled via an 8-bit field in the Compare A extension register (HRPWM_CMPAHR).

Copyright © 2024 Texas Instruments Incorporated



A. For MEP range and rounding adjustment.

Figure 12-222. Operating Logic Using MEP

To generate an HRPWM waveform of a given frequency and polarity, the user needs to configure the TBM (Time-Base Module), CCM (Counter-Compare Module), and AQM (Action-Qualifier Module) registers. The HRPWM works together with the TBM, CCM, and AQM registers to extend edge resolution, and should be configured accordingly. Although many programming combinations are possible, only a few are needed and practical.

12.4.2.3.9.3 Controlling and Monitoring the High-Resolution PWM Submodule

The MEP of the HRPWM is controlled by two extension registers, each 8-bits wide. These two HRPWM registers are concatenated with the 16-bit EPWM_TBPHS and EPWM_CMPA registers used to control PWM operation.

- HRPWM_TBPHSHR — Time-Base Phase High-Resolution Register
- HRPWM_CMPAHR — Counter-Compare A High-Resolution Register.

Table 12-281 lists the registers used to control and monitor the high-resolution PWM submodule.

Table 12-281. HRPWM Submodule Registers

Acronym	Register Description	Address Offset	Shadowed
HRPWM_TBPHSHR	Extension Register for HRPWM Phase	4h	No
HRPWM_CMPAHR	Extension Register for HRPWM Duty	10h	Yes
HRPWM_HRCTL	HRPWM Configuration Register	40h	No

12.4.2.3.9.4 Configuring the High-Resolution PWM Submodule

Once the EPWM has been configured to provide conventional PWM of a given frequency and polarity, the HRPWM is configured by programming the HRPWM_HRCTL register located at offset address 40h. This register provides configuration options for the following key operating modes:

- **Edge Mode:** The MEP can be programmed to provide precise position control on the rising edge (RE), falling edge (FE), or both edges (BE) at the same time. FE and RE are used for power topologies requiring duty cycle control, while BE is used for topologies requiring phase shifting, for example, phase shifted full bridge.
- **Control Mode:** The MEP is programmed to be controlled either from the HRPWM_CMPAHR register (duty cycle control) or the HRPWM_TBPHSHR register (phase control). RE or FE control mode should be used with the HRPWM_CMPAHR register. BE control mode should be used with the HRPWM_TBPHSHR register.
- **Shadow Mode:** This mode provides the same shadowing (double buffering) option as in regular PWM mode. This option is valid only when operating from the HRPWM_CMPAHR register and should be chosen to be the same as the regular load option for the CMPA register. If the HRPWM_TBPHSHR register is used, then this option has no effect.

Copyright © 2024 Texas Instruments Incorporated

12.4.2.3.9.5 Operational Highlights for the High-Resolution PWM Submodule

The MEP logic is capable of placing an edge in one of 255 (8 bits) discrete time steps, each of which has a time resolution on the order of 150 ps. The MEP works with the TBM (Time-Base Module) and CCM (Counter-Compare Module) registers to be certain that time steps are optimally applied and that edge placement accuracy is maintained over a wide range of PWM frequencies, system clock frequencies and other operating conditions. Table 12-282 shows the typical range of operating frequencies supported by the HRPWM.

Table 12-282. Relationship Between MEP Steps, PWM Frequency and Resolution

System (MHz)	MEP Steps Per FICLK(1) (2) (3)	PWM Minimum (Hz) (4)	PWM Maximum (MHz)	Resolution at Maximum (Bits) (5)
50.0	111	763	2.50	11.1
60.0	93	916	3.00	10.9
70.0	79	1068	3.50	10.6
80.0	69	1221	4.00	10.4
90.0	62	1373	4.50	10.3
100.0	56	1526	5.00	10.1

(1) System frequency = FICLK, that is, CPU clock. TBCLK = FICLK.
(2) Table data based on a MEP time resolution of 180 ps (this is an example value).
(3) MEP steps applied = T_{FICLK}/180 ps in this example.
(4) PWM minimum frequency is based on a maximum period value, TBPRD = 65 535. PWM mode is asymmetrical up-count.
(5) Resolution in bits is given for the maximum PWM frequency stated.

Copyright © 2024 Texas Instruments Incorporated

12.4.2.3.9.5.1 HRPWM Edge Positioning

In a typical power control loop (switch modes, digital motor control (DMC), uninterruptible power supply (UPS)), a digital controller (PID, 2pole/2zero, lag/lead, and so forth) issues a duty command, usually expressed in a per unit or percentage terms.

In the following example, assume that for a particular operating point, the demanded duty cycle is 0.405 or 40.5% on-time and the required converter PWM frequency is 1.25 MHz. In conventional PWM generation with a system clock of 100 MHz, the duty cycle choices are in the vicinity of 40.5%. Figure 12-223 shows that duty cycle of 40% (a compare value of 32 counts) is the closest to 40.5%. This is equivalent to an edge position of 320 ns instead of the desired 324 ns. This data is shown in Table 12-283.

Utilizing MEP allows to achieve an edge position much closer to the desired point of 324 ns. Table 12-283 shows that in addition to the CMPA value, 22 steps of the MEP (HRPWM_CMPAHR register) will position the edge at 323.96 ns, resulting in almost zero error. In this example, it is assumed that the MEP has a step resolution of 180 ns.

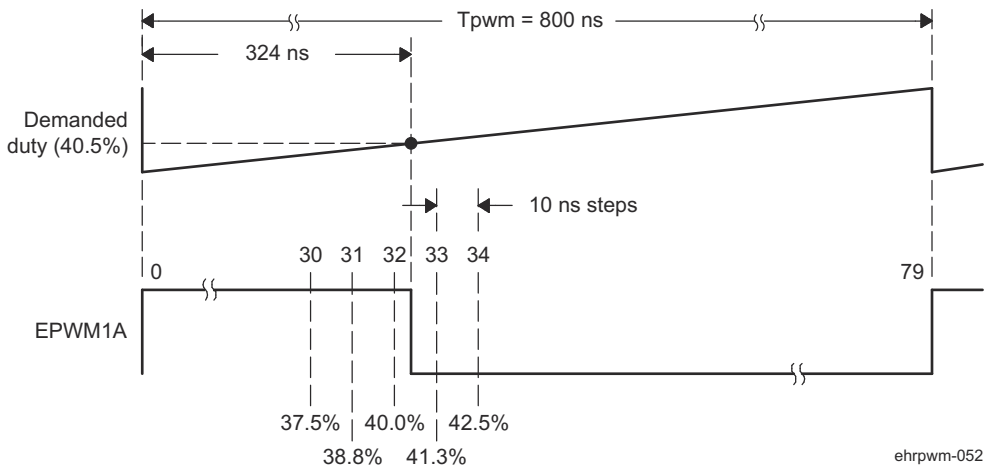


Figure 12-223. Required PWM Waveform for a Requested Duty = 40.5%

Table 12-283. CMPA vs Duty (left), and [CMPA:CMPAHR] vs Duty (right)

CMPA (count)(1) (2) (3)	DUTY (%)	High Time (ns)	CMPA (count)	CMPAHR (count)	Duty (%)	High Time (ns)
28	35.0	280	32	18	40.405	323.24
29	36.3	290	32	19	40.428	323.42
30	37.5	300	32	20	40.450	323.60
31	38.8	310	32	21	40.473	323.78
32	40.0	320	32	22	40.495	323.96
33	41.3	330	32	23	40.518	324.14
34	42.5	340	32	24	40.540	324.32
			32	25	40.563	324.50
Required			32	26	40.585	324.68
32.40	40.5	324	32	27	40.608	324.86

(1) System clock, FICLK and TBCLK = 100 MHz, 10 ns
(2) For a PWM Period register value of 80 counts, PWM Period = 80 × 10 ns = 800 ns, PWM frequency = 1/800 ns = 1.25 MHz
(3) Assumed MEP step size for the above example = 180 ps

Copyright © 2024 Texas Instruments Incorporated

12.4.2.3.9.5.2 HRPWM Scaling Considerations

The mechanics of how to position an edge precisely in time has been demonstrated using the resources of the standard (EPWM_CMPA) and MEP (HRPWM_CMPAHR) registers. In a practical application, however, it is necessary to seamlessly provide the CPU a mapping function from a per-unit (fractional) duty cycle to a final integer (non-fractional) representation that is written to the [CMPA:CMPAHR] register combination.

To do this, first examine the scaling or mapping steps involved. It is common in control software to express duty cycle in a per-unit or percentage basis. This has the advantage of performing all needed math calculations without concern for the final absolute duty cycle, expressed in clock counts or high time in ns. Furthermore, it makes the code more transportable across multiple converter types running different PWM frequencies.

To implement the mapping scheme, a two-step scaling procedure is required.

Assumptions for this example:

System clock, FICLK	=	10 ns (100 MHz)
PWM frequency	=	1.25 MHz (1/800 ns)
Required PWM duty cycle, **PWMDuty**	=	0.405 (40.5%)
PWM period in terms of coarse steps, **PWMperiod** (800 ns/10 ns)	=	80
Number of MEP steps per coarse step at 180 ps (10 ns/180 ps), **MEP_SF**	=	55
Value to keep CMPAHR within the range of 1-255 and fractional rounding constant (default value)	=	180h

Step 1: Percentage Integer Duty value conversion for EPWM_CMPA register

EPWM_CMPA register value	=	int(**PWMDuty** × **PWMperiod**); int means integer part
	=	int(0.405 × 80)
	=	int(32.4)
EPWM_CMPA register value	=	32 (20h)

Step 2: Fractional value conversion for HRPWM_CMPAHR register

HRPWM_CMPAHR register value	=	(frac(**PWMDuty** × **PWMperiod**) × **MEP_SF**) << 8) + 180h; frac means fractional part
	=	(frac(32.4) × 55 <<8) + 180h; Shift is to move the value as CMPAHR high byte
	=	((0.4 × 55) <<8) + 180h
	=	(22 <<8) + 180h
	=	22 × 256 + 180h; Shifting left by 8 is the same multiplying by 256.
	=	5632 + 180h
	=	1600h + 180h
HRPWM_CMPAHR value	=	1780h; HRPWM_CMPAHR value = 1700h, lower 8 bits will be ignored by hardware.

Copyright © 2024 Texas Instruments Incorporated

12.4.2.3.9.5.3 HRPWM Duty Cycle Range Limitation

In high resolution mode, the MEP is not active for 100% of the PWM period. It becomes operational 3 FICLK cycles after the period starts.

Duty cycle range limitations are illustrated in Figure 12-224. This limitation imposes a lower duty cycle limit on the MEP. For example, precision edge control is not available all the way down to 0% duty cycle. Although for the first 3 or 6 cycles, the HRPWM capabilities are not available, regular PWM duty control is still fully operational down to 0% duty. In most applications this should not be an issue as the controller regulation point is usually not designed to be close to 0% duty cycle.

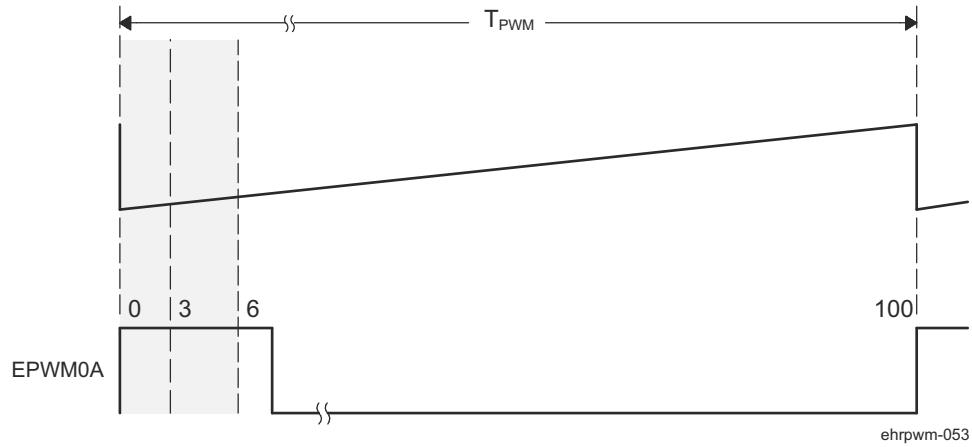


Figure 12-224. Low % Duty Cycle Range Limitation Example When PWM Frequency = 1 MHz

If the application demands HRPWM operation in the low percent duty cycle region, then the HRPWM can be configured to operate in count-down mode with the rising edge position (REP) controlled by the MEP. This is illustrated in Figure 12-225. In this case low percent duty limitation is no longer an issue.

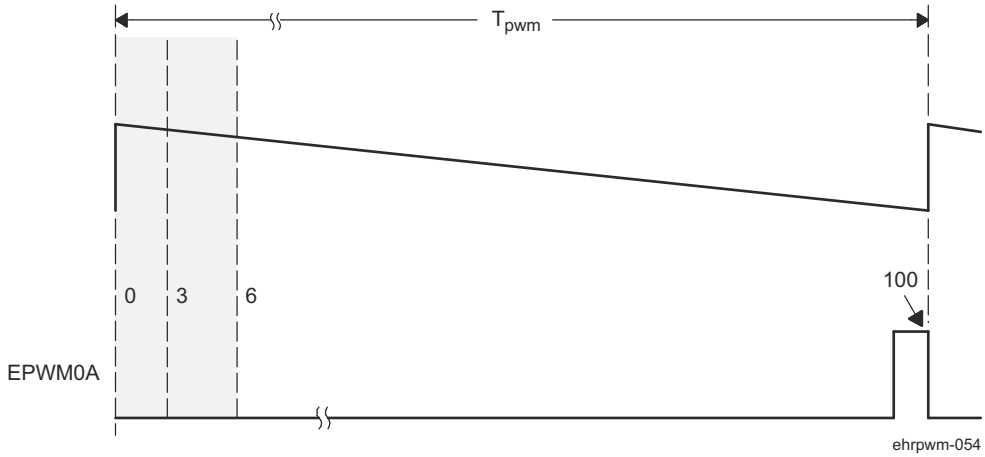


Figure 12-225. High % Duty Cycle Range Limitation Example when PWM Frequency = 1 MHz

12.4.2.3.10 EPWM / HRPWM Functional Register Groups

The Table 12-284 lists the groups of EPWM and the high-resolution PWM module registers according to their functionalities.

Table 12-284. EPWM / HRPWM Module Control and Status Registers Grouped by Submodule

Register Name	Offset	Size (×16)	Shadow	Register Description
Time-Base (TB) Submodule Registers				
EPWM_TBCTL	0h	1	No	Time-Base Control Register
EPWM_TBSTS	2h	1	No	Time-Base Status Register
EPWM_TBPHS	6h	1	No	Time-Base Phase Register

Copyright © 2024 Texas Instruments Incorporated

Table 12-284. EPWM / HRPWM Module Control and Status Registers Grouped by Submodule (continued)

Register Name	Offset	Size (×16)	Shadow	Register Description
EPWM_TBCNT	8h	1	No	Time-Base Counter Register
EPWM_TBPRD	Ah	1	Yes	Time-Base Period Register
Counter-Compare (CC) Submodule Registers				
EPWM_CMPCTL	Eh	1	No	Counter-Compare Control Register
EPWM_CMPA	12h	1	Yes	Counter-Compare A Register
EPWM_CMPB	14h	1	Yes	Counter-Compare B Register
Action-Qualifier (AQ) Submodule Registers				
EPWM_AQCTLA	16h	1	No	Action-Qualifier Control Register for Output A (EPWMxA)
EPWM_AQCTLB	18h	1	No	Action-Qualifier Control Register for Output B (EPWMxB)
EPWM_AQSFRC	1Ah	1	No	Action-Qualifier Software Force Register
EPWM_AQCSFRC	1Ch	1	Yes	Action-Qualifier Continuous S/W Force Register Set
Dead-Band (DB) Generator Submodule Registers				
EPWM_DBCTL	1Eh	1	No	Dead-Band Generator Control Register
EPWM_DBRED	20h	1	No	Dead-Band Generator Rising Edge Delay Count Register
EPWM_DBFED	22h	1	No	Dead-Band Generator Falling Edge Delay Count Register
Trip-Zone (TZ) Submodule Registers				
EPWM_TZSEL	24h	1	No	Trip-Zone Select Register
EPWM_TZCTL	28h	1	No	Trip-Zone Control Register (1)
EPWM_TZEINT	2Ah	1	No	Trip-Zone Enable Interrupt Register (1)
EPWM_TZFLG	2Ch	1	No	Trip-Zone Flag Register (1)
EPWM_TZCLR	2Eh	1	No	Trip-Zone Clear Register (1)
EPWM_TZFRC	30h	1	No	Trip-Zone Force Register (1)
Event-Trigger (ET) Submodule Registers				
EPWM_ETSEL	32h	1	No	Event-Trigger Selection Register
EPWM_ETPS	34h	1	No	Event-Trigger Pre-Scale Register
EPWM_ETFLG	36h	1	No	Event-Trigger Flag Register
EPWM_ETCLR	38h	1	No	Event-Trigger Clear Register
EPWM_ETFRC	3Ah	1	No	Event-Trigger Force Register
PWM-Chopper Submodule Registers				
EPWM_PCCTL	3Ch	1	No	PWM-Chopper Control Register
High-Resolution PWM (HRPWM) Submodule Registers				
HRPWM_TBPHSHR	4h	1	No	Extension for HRPWM Phase Register
HRPWM_CMPAHR	10h	1	No	Extension for HRPWM Counter-Compare A Register
HRPWM_HRCTL	40h	1	No	HRPWM Control Register

(1) EALLOW protected registers.

12.4.2.3.11 Proper EPWM Interrupt Initialization Procedure

When the EPWM peripheral clock is enabled it is possible that interrupt flags may be set due to spurious events as the EPWM registers not being properly initialized. The proper procedure for initializing the EPWM peripheral is:

1. Disable global interrupts (CPU INTM flag)
2. Disable EPWM interrupts
3. Initialize peripheral registers
4. Clear any spurious EPWM flags
5. Enable EPWM interrupts

Copyright © 2024 Texas Instruments Incorporated

6. Enable global interrupts

Copyright © 2024 Texas Instruments Incorporated

12.4.3 Enhanced Quadrature Encoder Pulse (EQEP) Module

This section describes the Enhanced Quadrature Encoder Pulse (EQEP) module in the device.

12.4.3.1 EQEP Overview

The Enhanced Quadrature Encoder Pulse (EQEP) peripheral is used for direct interface with a linear or rotary incremental encoder to get position, direction and speed information from a rotating machine for use in high performance motion and position control system. The disk of an incremental encoder is patterned with a single track of slots patterns, as shown in Figure 12-226. These slots create an alternating pattern of dark and light lines. The disk count is defined as the number of dark/light line pairs that occur per revolution (lines per revolution). As a rule, a second track is added to generate a signal that occurs once per revolution (index signal: QEPI), which can be used to indicate an absolute position. Encoder manufacturers identify the index pulse using different terms such as index, marker, home position and zero reference.

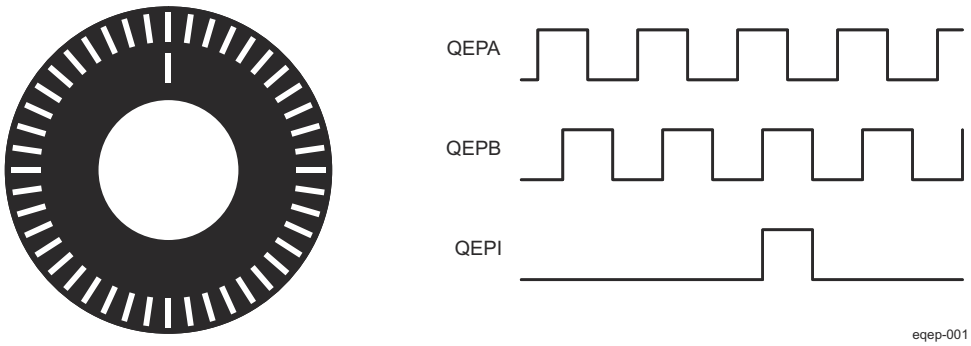


Figure 12-226. Optical Encoder Disk

To derive direction information, the lines on the disk are read out by two different photo-elements that "look" at the disk pattern with a mechanical shift of 1/4 the pitch of a line pair between them. This shift is realized with a reticle or mask that restricts the view of the photo-element to the desired part of the disk lines. As the disk rotates, the two photo-elements generate signals that are shifted 90 degrees out of phase from each other. These are commonly called the quadrature QEPA and QEPB signals. The clockwise direction for most encoders is defined as the QEPA channel going positive before the QEPB channel and vise versa as shown in Figure 12-227.

The encoder wheel typically makes one revolution for every revolution of the motor or the wheel may be at a geared rotation ratio with respect to the motor. Therefore, the frequency of the digital signal coming from the QEPA and QEPB outputs varies proportionally with the velocity of the motor. For example, a 2000-line encoder directly coupled to a motor running at 5000 revolutions per minute (rpm) results in a frequency of 166.6 kHz, so by measuring the frequency of either the QEPA or QEPB output, the processor can determine the velocity of the motor.

Copyright © 2024 Texas Instruments Incorporated

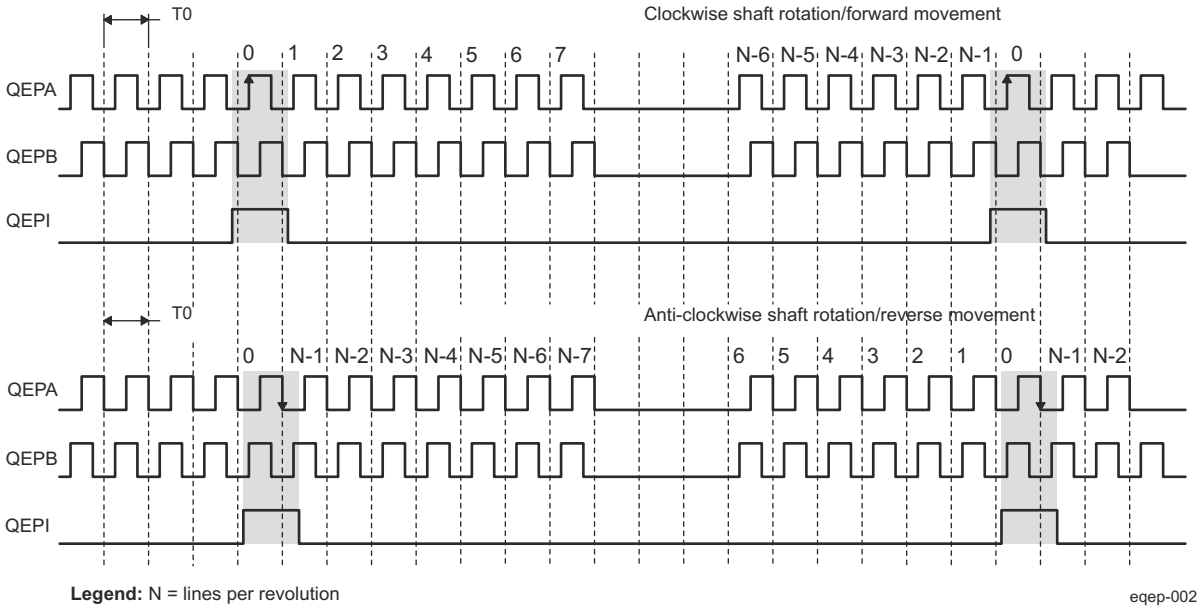


Figure 12-227. QEP Encoder Output Signal for Forward/Reverse Movement

Quadrature encoders from different manufacturers come with two forms of index pulse (gated index pulse or ungated index pulse) as shown in Figure 12-228. A nonstandard form of index pulse is ungated. In the ungated configuration, the index edges are not necessarily coincident with A and B signals. The gated index pulse is aligned to any of the four quadrature edges and width of the index pulse and can be equal to a quarter, half, or full period of the quadrature signal.

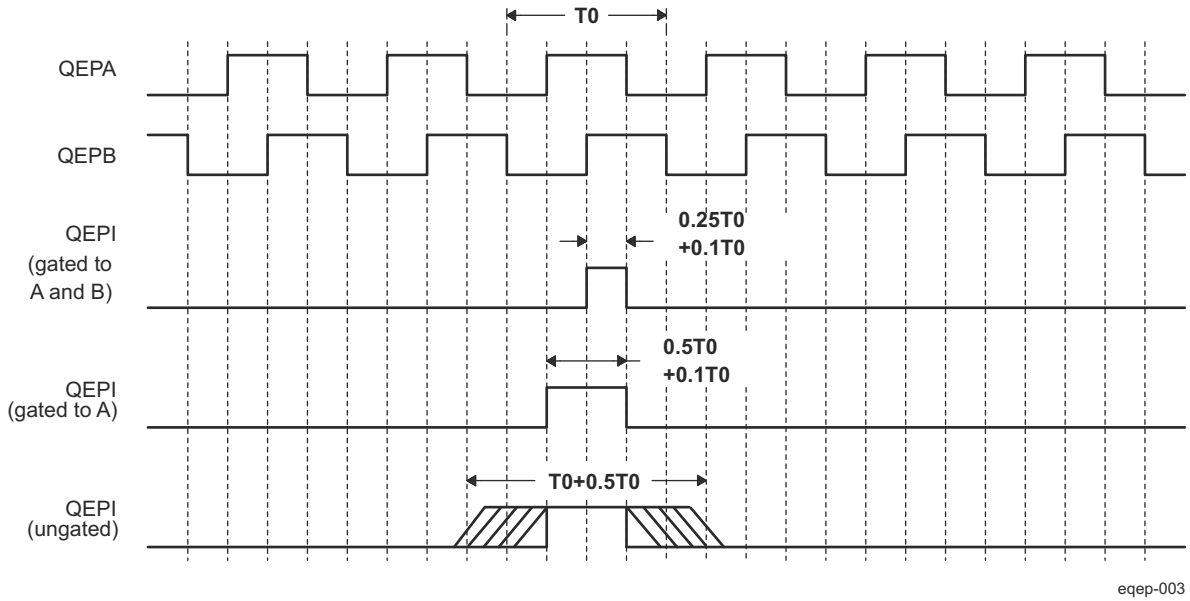


Figure 12-228. Index Pulse Example

Some typical applications for rotary encoders range from fax machines to motor controllers, and even robot localization. Rotary sensors are fundamental to the robotics and motion control systems found today. The optical shaft encoder can be used to improve a robot in various ways. The encoder can measure rotational distance traveled and speed, which can be used to monitor, for example, the angular position of a robot gripper arm or the speed of a robot.

General Issues: Estimating velocity from a digital position sensor is a cost-effective strategy in motor control. Two different first order approximations for velocity may be written as:

Copyright © 2024 Texas Instruments Incorporated

$$V(k) \approx \frac{x(k) - x(k-1)}{T} = \frac{\Delta X}{T}$$

eqep-004 (19)

$$V(k) \approx \frac{X}{t(k) - t(k-1)} = \frac{X}{\Delta T}$$

eqep-005 (20)

where

v(k): Velocity at time instant k

x(k): Position at time instant k

x(k-1): Position at time instant k - 1

T: Fixed unit time or inverse of velocity calculation rate

ΔX: Incremental position movement in unit time

t(k): Time instant "k"

t(k-1): Time instant "k - 1"

X: Fixed unit position

ΔT: Incremental time elapsed for unit position movement.

Equation 19 is the conventional approach to velocity estimation and it requires a time base to provide unit time event for velocity calculation. Unit time is the inverse of the velocity calculation rate.

The encoder count (position) is read once during each unit time event. The quantity [x(k) - x(k-1)] is formed by subtracting the previous reading from the current reading. Then the velocity estimate is computed by multiplying by the known constant 1/T (where T is the constant time between unit time events and is known in advance).

Estimation based on Equation 19 has an inherent accuracy limit directly related to the resolution of the position sensor and the unit time period T. For example, consider a 500-line per revolution quadrature encoder with a velocity calculation rate of 400 Hz. When used for position the quadrature encoder gives a four-fold increase in resolution, in this case, 2000 counts per revolution. The minimum rotation that can be detected is therefore 0.0005 revolutions, which gives a velocity resolution of 12 rpm when sampled at 400 Hz. While this resolution may be satisfactory at moderate or high speeds, for example, 1% error at 1200 rpm, it would clearly prove inadequate at low speeds. In fact, at speeds below 12 rpm, the speed estimate would erroneously be zero much of the time.

At low speed, Equation 20 provides a more accurate approach. It requires a position sensor that outputs a fixed interval pulse train, such as the aforementioned quadrature encoder. The width of each pulse is defined by motor speed for a given sensor resolution. Equation 20 can be used to calculate motor speed by measuring the elapsed time between successive quadrature pulse edges. However, this method suffers from the opposite limitation of Equation 19. A combination of relatively large motor speeds and high sensor resolution makes the time interval ΔT small, and thus more greatly influenced by the timer resolution. This can introduce considerable error into high-speed estimates.

For systems with a large speed range (that is, speed estimation is needed at both low and high speeds), one approach is to use Equation 20 at low speed and have the software switch over to Equation 19 when the motor speed rises above some specified threshold.

The device has three instances of the EQEP modules.

12.4.3.1.1 EQEP Features

The EQEP module includes the following features:

- Input synchronization
- Three stage/six stage digital noise filter
- Quadrature decoder unit

Copyright © 2024 Texas Instruments Incorporated

- Position counter and control unit for position measurement
- Quadrature edge capture unit for low-speed measurement
- Unit time base for speed and frequency measurement
- Watchdog timer for detecting stalls
- EQEP inputs (A/B/INDEX and STROBE) are available at chip level
- EQEP phase error output is also available. The status of the phase error can be observed by software through the CTRLMMR_EQEP_STAT register in the CTRL_MMR0 module.

12.4.3.1.2 EQEP Ports

Table 12-285. EQEP Clocks and Resets

Clocks	
Module Clock Input	**Description**
EQEP_FICLK	EQEP functional and interface clock
Resets	
Module Reset Input	**Description**
EQEP_RST	Module Reset

Table 12-286. EQEP Hardware Requests

Interrupt Requests		
Module Interrupt Signal	**Description**	**Type**
EQEP_EQEP_INT_0	EQEP interrupt	Pulse

12.4.3.2 EQEP Environment

This section describes the EQEP external connections (environment).

Table 12-287 describes the EQEP I/O signals.

Table 12-287. EQEP I/O Signals

Module Pin	I/O Type[(1)]	Description
EQEP0_A	I	EQEP0 Quadrature input A
EQEP0_B	I	EQEP0 Quadrature input B
EQEP0_INDEX	I/O[(2)]	EQEP0 Index input/output
EQEP0_STROBE	I/O[(2)]	EQEP0 Strobe input/output
EQEP1_A	I	EQEP1 Quadrature input A
EQEP1_B	I	EQEP1 Quadrature input B
EQEP1_INDEX	I/O[(2)]	EQEP1 Index input/output
EQEP1_STROBE	I/O[(2)]	EQEP1 Strobe input/output
EQEP2_A	I	EQEP2 Quadrature input A
EQEP2_B	I	EQEP2 Quadrature input B
EQEP2_INDEX	I/O[(2)]	EQEP2 Index input/output
EQEP2_STROBE	I/O[(2)]	EQEP2 Strobe input/output

(1) I = Input; O = Output
(2) Output functionality is not supported. Only inputs are pinned out. For more information see *Device Specific EQEP Features*.

Copyright © 2024 Texas Instruments Incorporated

12.4.3.3 EQEP Functional Description

This section provides the EQEP functional description and corresponding functional details about EQEPx inputs.

Note

Multiple identical EQEP modules can be contained in a system. For actual number of EQEP modules integrated in the device, refer to *EQEP Integration*. The letter x within a signal or module name is used to indicate a generic EQEP instance on a device. For example, output interrupt request, EQEP0_EQEP_INT_0 belongs to EQEP0, EQEP1_EQEP_INT_0 belongs to EQEP1, and so forth.

The EQEP peripheral contains the following major functional units (as shown in Figure 12-229):

- Programmable input qualification for each pin (part of the GPIO MUX)
- Three stage/six stage digital noise filter
- Quadrature decoder unit (QDU)
- Position counter and control unit for position measurement (PCCU)
- Quadrature edge-capture unit for low-speed measurement (QCAP)
- Unit time base for speed/frequency measurement (UTIME)
- Watchdog timer for detecting stalls (QWDOG).

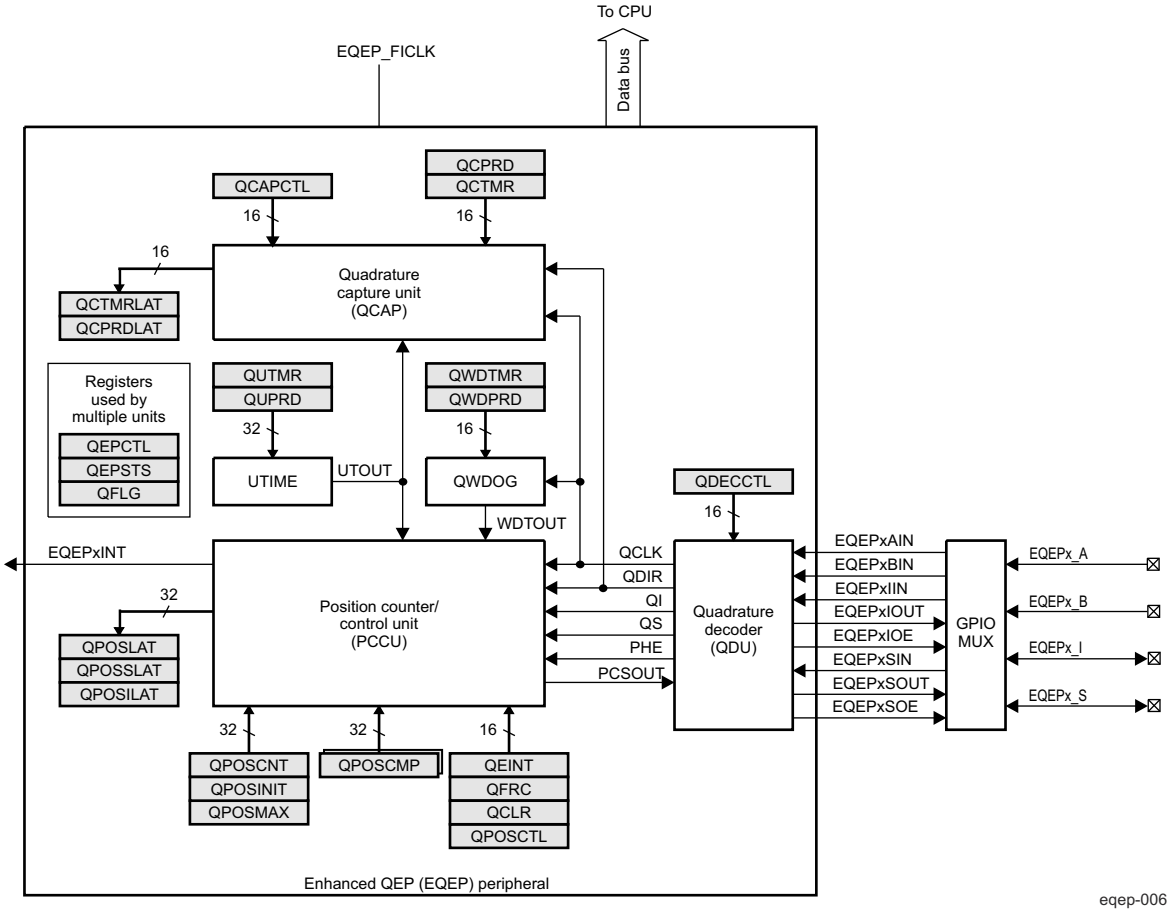


Figure 12-229. Functional Block Diagram of the EQEP Peripheral

12.4.3.3.1 EQEP Inputs

The EQEP inputs include two pins for quadrature-clock mode or direction-count mode, an index (or 0 marker), and a strobe input.

- EQEPx_A and EQEPx_B: These two pins can be used in quadrature-clock mode or direction-count mode.

Copyright © 2024 Texas Instruments Incorporated

  - Quadrature-clock mode: The EQEP encoders provide two square wave signals (A and B) 90 electrical degrees out of phase whose phase relationship is used to determine the direction of rotation of the input shaft and number of EQEP pulses from the index position to derive the relative position information. For forward or clockwise rotation, QEPA signal leads QEPB signal and vice versa. The quadrature decoder uses these two inputs to generate quadrature-clock and direction signals.
  - Direction-count mode: In direction-count mode, direction and clock signals are provided directly from the external source. Some position encoders have this type of output instead of quadrature output. The EQEPx_A pin provides the clock input and the EQEPx_B pin provides the direction input.
- EQEPx_I: Index or Zero Marker: The EQEP encoder uses an index signal to assign an absolute start position from which position information is incrementally encoded using quadrature pulses. This pin is connected to the index output of the EQEP encoder to optionally reset the position counter for each revolution. This signal can be used to initialize or latch the position counter on the occurrence of a desired event on the index pin.
- EQEPx_S: Strobe Input: This general-purpose strobe signal can initialize or latch the position counter on the occurrence of a desired event on the strobe pin. This signal is typically connected to a sensor or limit switch to notify that the motor has reached a defined position.

Copyright © 2024 Texas Instruments Incorporated

12.4.3.3.2 EQEP Quadrature Decoder Unit (QDU)

Figure 12-230 shows a functional block diagram of the QDU.

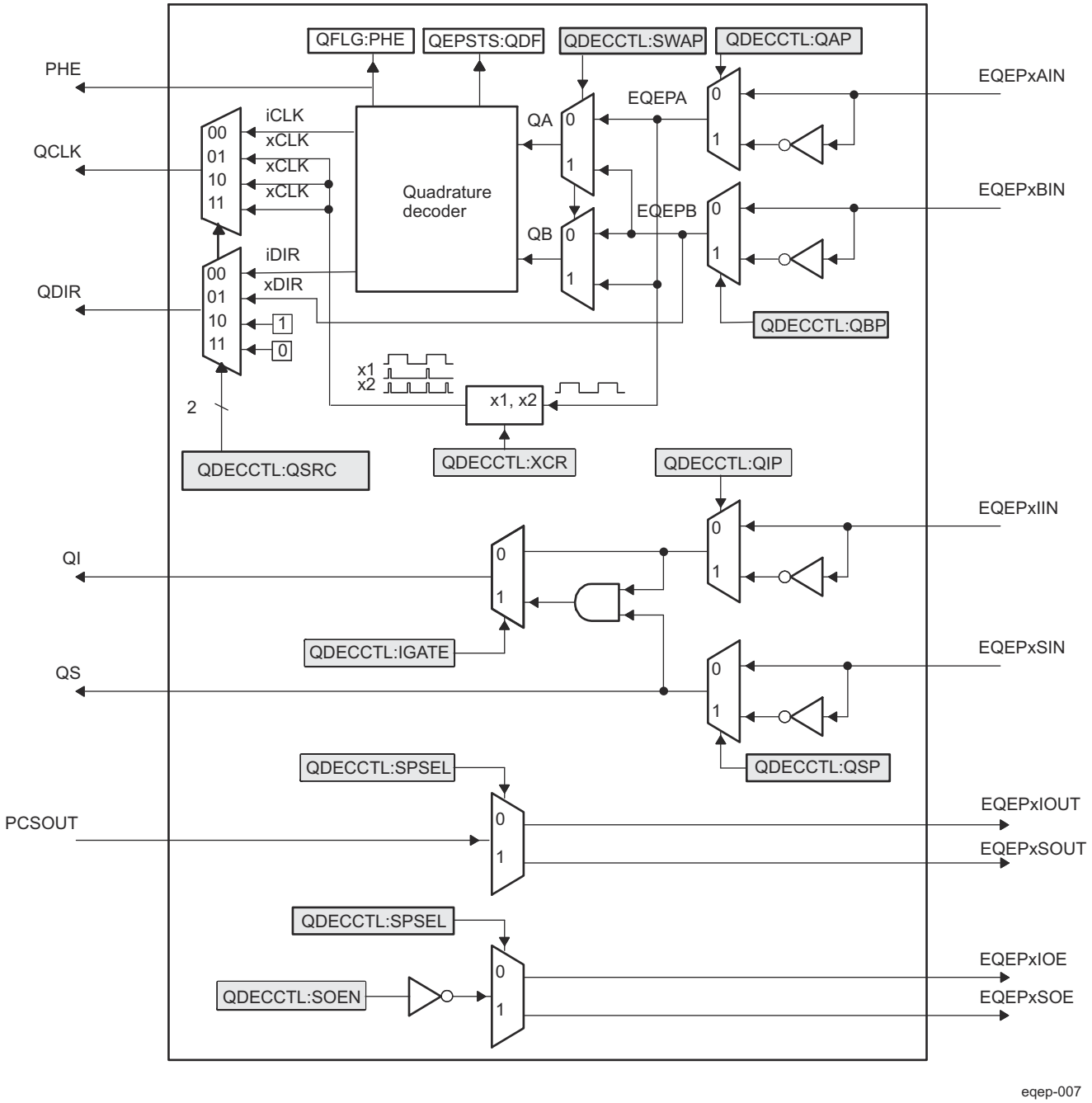


Figure 12-230. Functional Block Diagram of Decoder Unit

Copyright © 2024 Texas Instruments Incorporated

12.4.3.3.2.1 EQEP Position Counter Input Modes

Clock and direction input to position counter is selected using the QSRC bit in the EQEP decoder and control register (EQEP_QDEC_QEP_CTL), based on interface input requirement as follows:

- Quadrature-count mode
- Direction-count mode
- UP-count mode
- DOWN-count mode.

12.4.3.3.2.1.1 Quadrature Count Mode

The quadrature decoder generates the direction and clock to the position counter in quadrature count mode.

Direction Decoding

The direction decoding logic of the EQEP circuit determines which one of the sequences (EQEPA, EQEPB) is the leading sequence and accordingly updates the direction information in the QDF bit in the EQEP status register (EQEP_QEP_STS_CT). Table 12-288 and Figure 12-231 show the direction decoding logic in truth table and state machine form. Both edges of the EQEPA and EQEPB signals are sensed to generate count pulses for the position counter. Therefore, the frequency of the clock generated by the EQEP logic is four times that of each input sequence. Figure 12-232 shows the direction decoding and clock generation from the EQEP input signals.

Phase Error Flag

In normal operating conditions, quadrature inputs EQEPA and EQEPB will be 90 degrees out of phase. The phase error flag (QPEI_FLG) is set in the EQEP_QINT_EN_FLG register when edge transition is detected simultaneously on the EQEPA and EQEPB signals to optionally generate interrupts. State transitions marked by dashed lines in Figure 12-231 are invalid transitions that generate a phase error.

Count Multiplication

The EQEP position counter provides 4× times the resolution of an input clock by generating a quadrature-clock (QCLK) on the rising/falling edges of both EQEP input clocks (EQEPA and EQEPB) as shown in Figure 12-232.

Reverse Count

In normal quadrature count operation, EQEPA input is fed to the QA input of the quadrature decoder and the EQEPB input is fed to the QB input of the quadrature decoder. Reverse counting is enabled by setting the SWAP bit in the EQEP decoder and control register (EQEP_QDEC_QEP_CTL). This will swap the input to the quadrature decoder thereby reversing the counting direction.

Table 12-288. Quadrature Decoder Truth Table

Previous Edge	Present Edge	QDIR	QPOSCNT
QA↑	QB↑	UP	Increment
	QB↓	DOWN	Decrement
	QA↓	TOGGLE	Increment or Decrement
QA↓	QB↓	UP	Increment
	QB↑	DOWN	Decrement
	QA↑	TOGGLE	Increment or Decrement
QB↑	QA↑	DOWN	Increment
	QA↓	UP	Decrement
	QB↓	TOGGLE	Increment or Decrement
QB↓	QA↓	DOWN	Increment
	QA↑	UP	Decrement
	QB↑	TOGGLE	Increment or Decrement

Copyright © 2024 Texas Instruments Incorporated

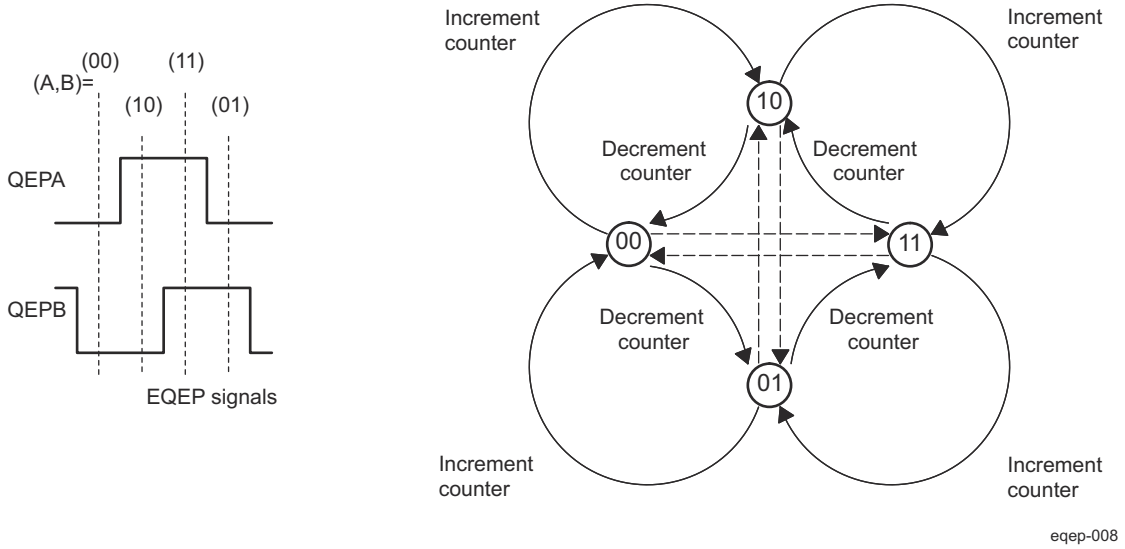


Figure 12-231. Quadrature Decoder State Machine

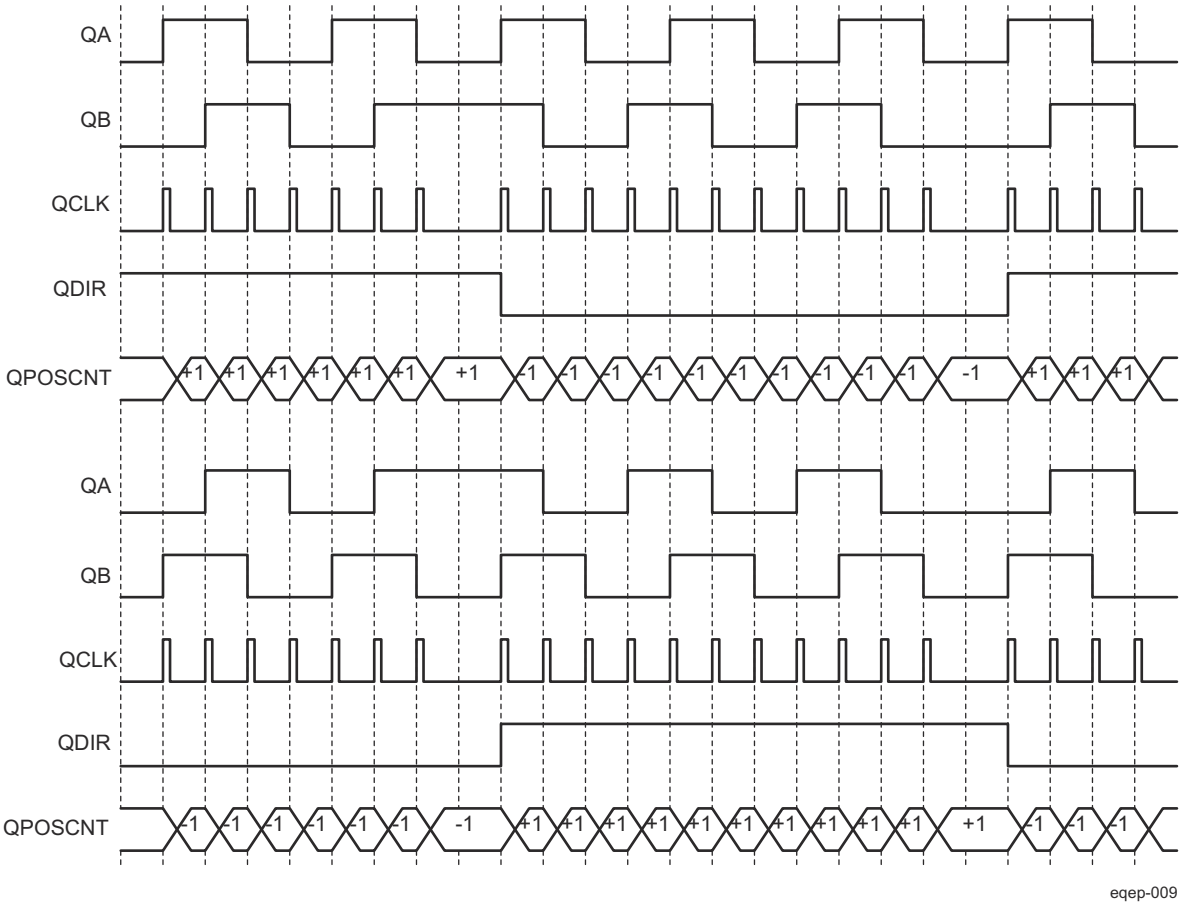


Figure 12-232. Quadrature-clock and Direction Decoding

Copyright © 2024 Texas Instruments Incorporated

12.4.3.3.2.1.2 EQEP Direction-count Mode

Some position encoders provide direction and clock outputs, instead of quadrature outputs. In such cases, direction-count mode can be used. EQEPA input will provide the clock for position counter and the EQEPB input will have the direction information. The position counter is incremented on every rising edge of a EQEPA input when the direction input is high and decremented when the direction input is low.

12.4.3.3.2.1.3 EQEP Up-Count Mode

The counter direction signal is hard-wired for up count and the position counter is used to measure the frequency of the EQEPA input. Setting of the XCR bit in the EQEP decoder and control register (EQEP_QDEC_QEP_CTL) enables clock generation to the position counter on both edges of the QEPA input, thereby increasing the measurement resolution by 2× factor.

12.4.3.3.2.1.4 EQEP Down-Count Mode

The counter direction signal is hardwired for a down count and the position counter is used to measure the frequency of the EQEPA input. Setting of the XCR bit in the EQEP decoder and control register (EQEP_QDEC_QEP_CTL) enables clock generation to the position counter on both edges of a EQEPA input, thereby increasing the measurement resolution by 2× factor.

12.4.3.3.2.2 EQEP Input Polarity Selection

Each EQEP input can be inverted using the in the EQEP decoder and control register (EQEP_QDEC_QEP_CTL[8-5]) control bits. As an example, setting of the QIP bit in EQEP_QDEC_QEP_CTL inverts the index input.

12.4.3.3.2.3 EQEP Position-Compare Sync Output

The EQEP peripheral includes a position-compare unit that is used to generate the position-compare sync signal on compare match between the position counter register (EQEP_QPOSCNT) and the position-compare register (EQEP_QPOSCMP). This sync signal can be output using an index pin or strobe pin of the EQEP peripheral.

Setting the SOEN bit in the EQEP decoder and control register (EQEP_QDEC_QEP_CTL) enables the position-compare sync output and the SPSEL bit in EQEP_QDEC_QEP_CTL selects either an EQEP index pin or an EQEP strobe pin.

12.4.3.3.3 EQEP Position Counter and Control Unit (PCCU)

The position counter and control unit provides two configuration registers (EQEP_QDEC_QEP_CTL and EQEP_QCAP_QPOS_CTL) for setting up position counter operational modes, position counter initialization/latch modes and position-compare logic for sync signal generation.

12.4.3.3.3.1 EQEP Position Counter Operating Modes

Position counter data may be captured in different manners. In some systems, the position counter is accumulated continuously for multiple revolutions and the position counter value provides the position information with respect to the known reference. An example of this is the quadrature encoder mounted on the motor controlling the print head in the printer. Here the position counter is reset by moving the print head to the home position and then position counter provides absolute position information with respect to home position.

In other systems, the position counter is reset on every revolution using index pulse and position counter provides rotor angle with respect to index pulse position.

Position counter can be configured to operate in following four modes

- Position Counter Reset on Index Event
- Position Counter Reset on Maximum Position
- Position Counter Reset on the first Index Event
- Position Counter Reset on Unit Time Out Event (Frequency Measurement).

In all the above operating modes, position counter is reset to 0 on overflow and to QPOSMAX bifield value in EQEP_QPOSMAX register on underflow. Overflow occurs when the position counter counts up after QPOSMAX

Copyright © 2024 Texas Instruments Incorporated

value. Underflow occurs when position counter counts down after "0". Interrupt flag is set to indicate overflow/ underflow in EQEP_QINT_EN_FLG register.

12.4.3.3.3.1.1 EQEP Position Counter Reset on Index Event (EQEP_QDEC_QEP_CTL[29-28] PCRM] = 0b00)

If the index event occurs during the forward movement, then position counter is reset to 0 on the next EQEP clock. If the index event occurs during the reverse movement, then the position counter is reset to the value in the EQEP_QPOSMAX register on the next EQEP clock.

First index marker is defined as the quadrature edge following the first index edge. The EQEP peripheral records the occurrence of the first index marker (EQEP_QEP_STS_CT[1] FIMF) and direction on the first index event marker (EQEP_QEP_STS_CT[6] FIDF) in EQEP_QEP_STS_CT registers, it also remembers the quadrature edge on the first index marker so that same relative quadrature transition is used for index event reset operation.

For example, if the first reset operation occurs on the falling edge of EQEPB during the forward direction, then all the subsequent reset must be aligned with the falling edge of EQEPB for the forward rotation and on the rising edge of EQEPB for the reverse rotation as shown in Figure 12-233.

The position-counter value is latched to the EQEP_QPOSILAT register and direction information is recorded in the EQEP_QEP_STS_CT[4] QDLF bit on every index event marker. The position-counter error flag (EQEP_QEP_STS_CT[0] PCEF) and error interrupt flag (EQEP_QINT_EN_FLG[17] PCEI_FLG) are set if the latched value is not equal to 0 or QPOSMAX. The position-counter error flag (EQEP_QEP_STS_CT[0] PCEF) is updated on every index event marker and an interrupt flag (EQEP_QINT_EN_FLG[17] PCEI_FLG) will be set on error that can be cleared only through software.

The index event latch configuration EQEP_QDEC_QEP_CTL[21-20] IEL bits are ignored in this mode and position counter error flag/interrupt flag are generated only in index event reset mode.

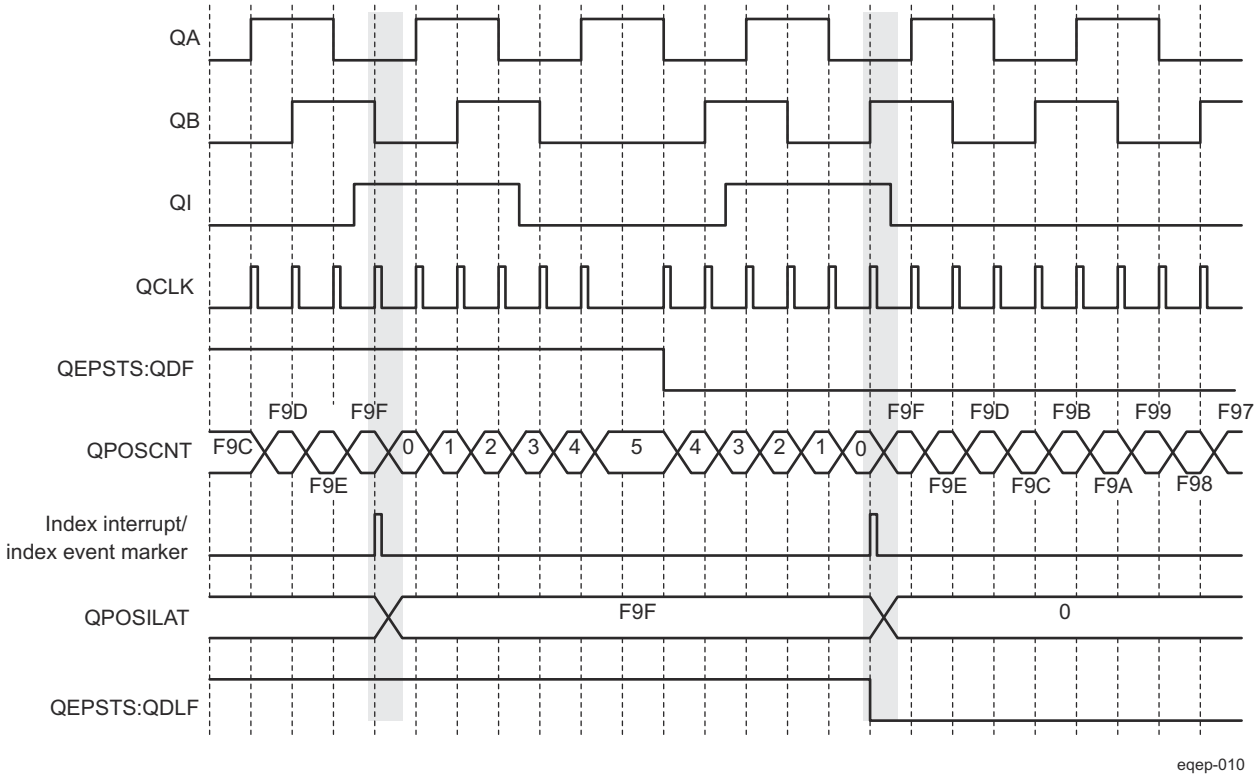


Figure 12-233. Position Counter Reset by Index Pulse for 1000 Line Encoder (QPOSMAX = 3999 or F9Fh)

Copyright © 2024 Texas Instruments Incorporated

12.4.3.3.3.1.2 EQEP Position Counter Reset on Maximum Position (EQEP_QDEC_QEP_CTL[29-28] PCRM=0b01)

If the position counter is equal to QPOSMAX (in EQEP_QPOSMAX register), then the position counter is reset to 0 on the next EQEP clock for forward movement and position counter overflow flag is set. If the position counter is equal to 0, then the position counter is reset to QPOSMAX on the next QEP clock for reverse movement and position counter underflow flag is set. Figure 12-234 shows the position counter reset operation in this mode.

First index marker is defined as the quadrature edge following the first index edge. The EQEP peripheral records the occurrence of the first index marker (EQEP_QEP_STS_CT[1] FIMF) and direction on the first index event marker (EQEP_QEP_STS_CT[6] FIDF); it also remembers the quadrature edge on the first index marker so that the same relative quadrature transition is used for the software index marker (EQEP_QDEC_QEP_CTL[21-20] IEL= 0b11).

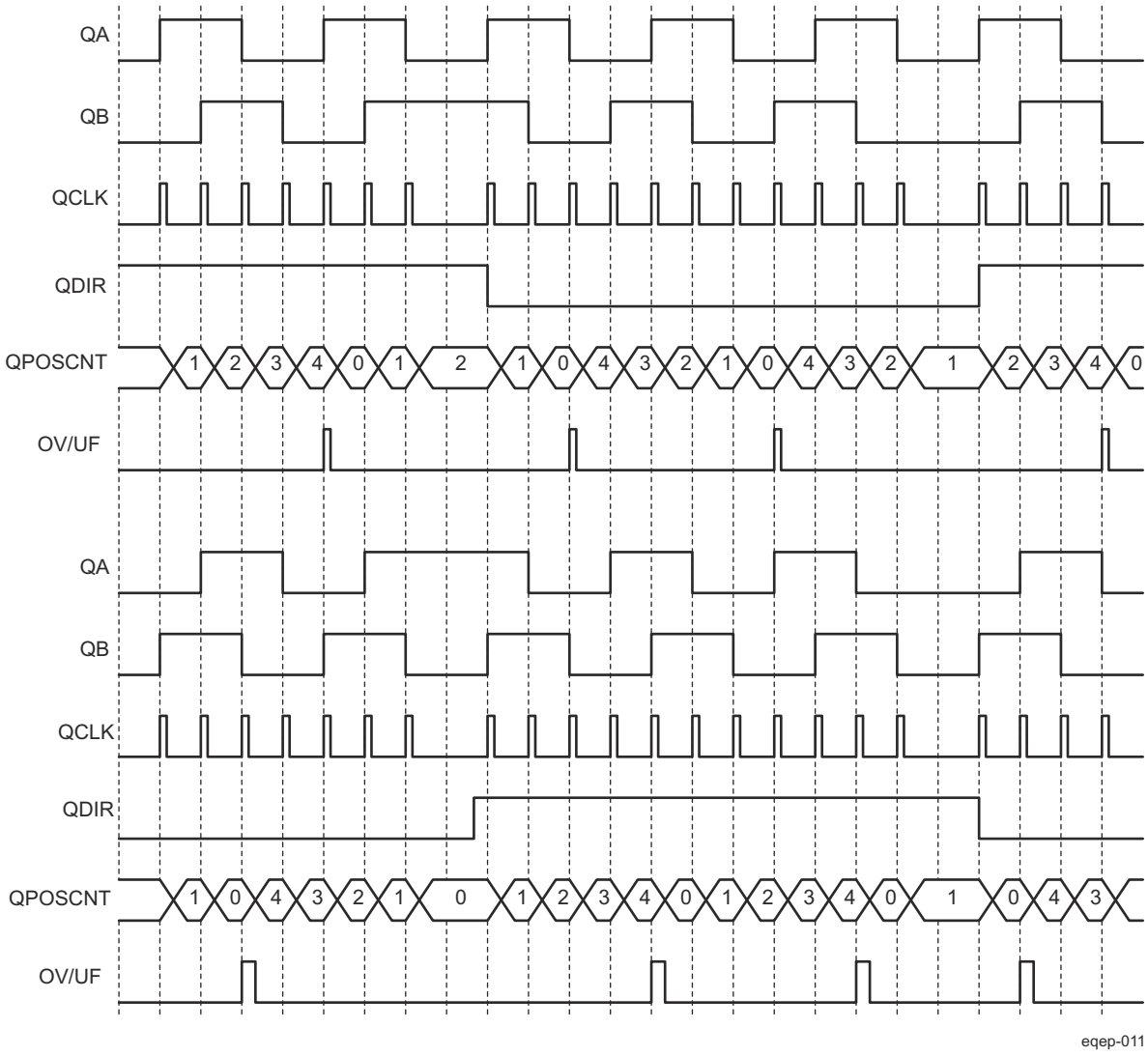


Figure 12-234. Position Counter Underflow/Overflow (QPOSMAX = 4)

Copyright © 2024 Texas Instruments Incorporated

12.4.3.3.3.1.3 Position Counter Reset on the First Index Event (EQEP_QDEC_QEP_CTL[29-28] PCRM = 0b10)

If the index event occurs during forward movement, then the position counter is reset to 0 on the next EQEP clock. If the index event occurs during the reverse movement, then the position counter is reset to the value in the EQEP_QPOSMAX register on the next EQEP clock. Note that this is done only on the first occurrence and subsequently the position counter value is not reset on an index event; rather, it is reset based on maximum position as described in Section 12.4.3.3.3.1.2.

First index marker is defined as the quadrature edge following the first index edge. The EQEP peripheral records the occurrence of the first index marker (EQEP_QEP_STS_CT[1] FIMF) and direction on the first index event marker (EQEP_QEP_STS_CT[6] FIDF) in EQEP_QEP_STS_CT registers. It also remembers the quadrature edge on the first index marker so that same relative quadrature transition is used for software index marker (EQEP_QDEC_QEP_CTL[21-20] IEL= 0b11).

12.4.3.3.3.1.4 Position Counter Reset on Unit Time out Event (EQEP_QDEC_QEP_CTL[29-28] PCRM = 0b11)

In this mode, the QPOSCNT value is latched to the EQEP_QPOSLAT register and then the QPOSCNT field is reset (to 0 or the QPOSMAX value in the EQEP_QPOSMAX register, depending on the direction mode selected by EQEP_QDEC_QEP_CTL[15-14] QSRC bits on a unit time event). This is useful for frequency measurement.

12.4.3.3.3.2 EQEP Position Counter Latch

The EQEP index and strobe input can be configured to latch the position counter QPOSCNT (EQEP_QPOSCNT) into QPOSILAT (EQEP_QPOSILAT register) and QPOSSLAT (EQEP_QPOSSLAT register) bitfields, respectively, on occurrence of a definite event on these pins.

12.4.3.3.3.2.1 Index Event Latch

In some applications, it may not be desirable to reset the position counter on every index event and instead it may be required to operate the position counter in full 32-bit mode (EQEP_QDEC_QEP_CTL[29-28] PCRM = 0b01 and EQEP_QDEC_QEP_CTL[29-28] PCRM = 0b10 modes).

In such cases, the EQEP position counter can be configured to latch on the following events and direction information is recorded in the EQEP_QEP_STS_CT[4] QDLF bit on every index event marker.

- Latch on Rising edge (EQEP_QDEC_QEP_CTL[21-20] IEL = 0b01)
- Latch on Falling edge (EQEP_QDEC_QEP_CTL[21-20] IEL = 0b10)
- Latch on Index Event Marker (EQEP_QDEC_QEP_CTL[21-20] IEL = 0b11)

This is particularly useful as an error checking mechanism to check if the position counter accumulated the correct number of counts between index events. As an example, the 1000-line encoder must count 4000 times when moving in the same direction between the index events.

The index event latch interrupt flag (EQEP_QINT_EN_FLG[26] IELI_FLG) is set when the position counter is latched to the EQEP_QPOSILAT register. The index event latch configuration bits are ignored when EQEP_QDEC_QEP_CTL[29-28] PCRM = 0b00.

When Position counter reset on an index event mode is selected through EQEP_QDEC_QEP_CTL[29-28] PCRM bit field (value: 0h), EQEP_QDEC_QEP_CTL[21-20] IEL bit field must be configured to 0h and position counter value is latched into the EQEP_QPOSILAT[31-0] QPOSILAT register for every index marker.

Latch on Rising Edge (EQEP_QDEC_QEP_CTL[21-20] IEL = 0b01)	The position counter value (QPOSCNT) is latched to the EQEP_QPOSILAT register on every rising edge of an index input.
Latch on Falling Edge (EQEP_QDEC_QEP_CTL[21-20] IEL = 0b10)	The position counter value (QPOSCNT) is latched to the EQEP_QPOSILAT register on every falling edge of index input.
Latch on Index Event Marker/Software Index Marker (EQEP_QDEC_QEP_CTL[21-20] IEL = 0b11)	The first index marker is defined as the quadrature edge following the first index edge. The EQEP peripheral records the occurrence of the first index marker (EQEP_QEP_STS_CT[1] FIMF) and direction on the first index event marker (EQEP_QEP_STS_CT[6] FIDF) in the EQEP_QEP_STS_CT registers. It also remembers the

Copyright © 2024 Texas Instruments Incorporated

quadrature edge on the first index marker so that same relative quadrature transition is used for latching the position counter (EQEP_QDEC_QEP_CTL[21-20] IEL = 0b11).

Figure 12-235 shows the position counter latch using an index event marker.

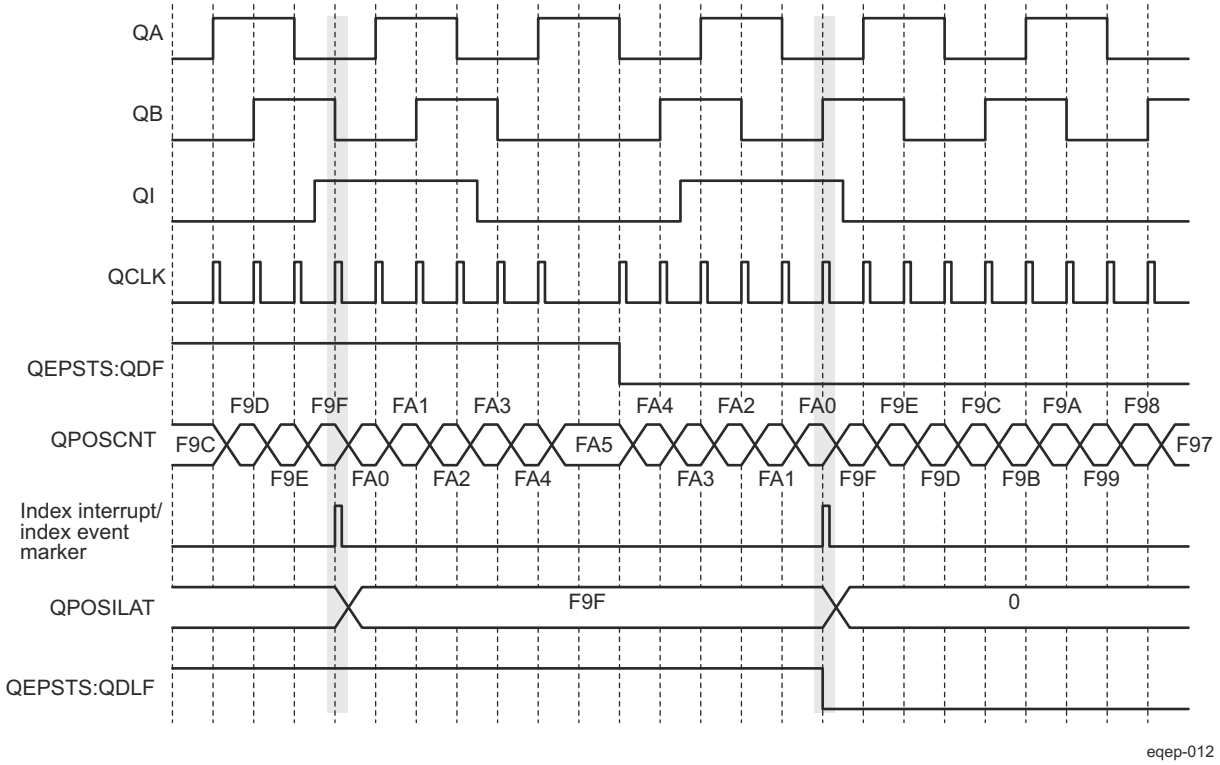


Figure 12-235. Software Index Marker for 1000-line Encoder (EQEP_QDEC_QEP_CTL[21-20] IEL = 0b01)

Copyright © 2024 Texas Instruments Incorporated

12.4.3.3.3.2.2 EQEP Strobe Event Latch

The position-counter value is latched to the EQEP_QPOSSLAT register on the rising edge of the strobe input (QCLK) by clearing the EQEP_QDEC_QEP_CTL[22] SEL bit. Latching on the falling edge of the strobe input (QCLK) can be done by inverting the strobe input using the EQEP_QDEC_QEP_CTL[5] QSP bit.

If the EQEP_QDEC_QEP_CTL[22] SEL bit is set, then the position counter value is latched to the EQEP_QPOSSLAT register on the rising edge of the strobe input for forward direction and on the falling edge of the strobe input for reverse direction as shown in Figure 12-236.

The strobe event latch interrupt flag (EQEP_QINT_EN_FLG[25] SELI_FLG) is set when the position counter is latched to the EQEP_QPOSSLAT register.

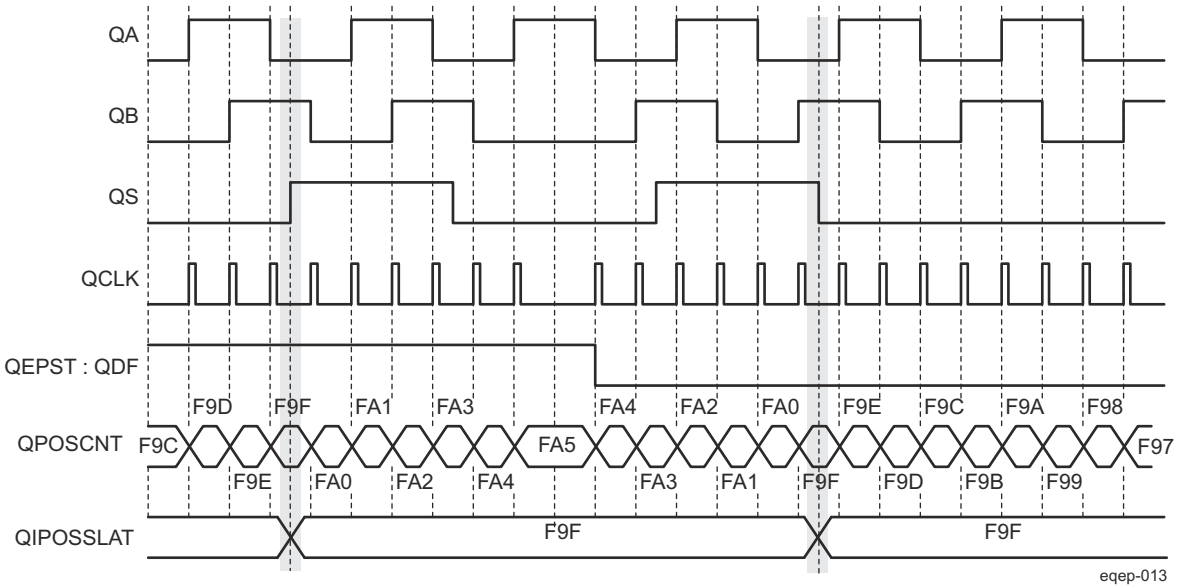


Figure 12-236. EQEP Strobe Event Latch (EQEP_QDEC_QEP_CTL[22] SEL = 0b1)

Copyright © 2024 Texas Instruments Incorporated

12.4.3.3.3.3 EQEP Position Counter Initialization

The position counter can be initialized using following events:

- Index event
- Strobe event
- Software initialization.

Note

When all of the above events occur simultaneously, the sequence of priority is as follows: 1) Software Initialization, 2) strobe event initialization, 3) Index Event Initialization.

Index Event Initialization (IEI)

The EQEPx_I index input can be used to trigger the initialization of the position counter at the rising or falling edge of the index input.

If the EQEP_QDEC_QEP_CTL[25-24] IEI bits are 2h, then the position counter (EQEP_QPOSCNT) is initialized with a value in the EQEP_QPOSINIT register on the rising edge of strobe input for forward direction and on the falling edge of strobe input for reverse direction.

The index event initialization interrupt flag (EQEP_QDEC_QEP_CTL[25-24] IEI) is set when the position counter is initialized with a value in the EQEP_QPOSINIT register.

If EQEP_QDEC_QEP_CTL[25-24] IEI bit field is configured with value 0h (default) or 1h, the index event initialization of position counter is disabled.

If EQEP_QDEC_QEP_CTL[25-24] IEI bit field is configured with value 3h, then the the position counter (EQEP_QPOSCNT) is initialized with a value in the EQEP_QPOSINIT register on the falling edge of strobe input signal.

Strobe Event Initialization (SEI)

If the EQEP_QDEC_QEP_CTL[27-26] SEI bits are 2h, then the position counter is initialized with a value in the EQEP_QPOSINIT register on the rising edge of strobe input.

If the EQEP_QDEC_QEP_CTL[27-26] SEI bits are 3h, then the position counter (EQEP_QPOSCNT) is initialized with a value in the EQEP_QPOSINIT register on the rising edge of strobe input for forward direction and on the falling edge of strobe input for reverse direction.

The strobe event initialization interrupt flag (EQEP_QDEC_QEP_CTL[27-26] SEI) is set when the position counter is initialized with a value in the EQEP_QPOSINIT register.

If EQEP_QDEC_QEP_CTL[27-26] SEI bit field is configured with value 0h (default) or 1h, the strobe event initialization of position counter is disabled.

Software Initialization (SWI)

The position counter can be initialized in software by writing a 1h to the EQEP_QDEC_QEP_CTL[23] SWI bit, which will automatically be cleared after initialization.

12.4.3.3.3.4 EQEP Position-Compare Unit

The EQEP peripheral includes a position-compare unit that is used to generate a sync output and/or interrupt on a position-compare match. Figure 12-237 shows a diagram. The position-compare (EQEP_QPOSCMP) register is shadowed and shadow mode can be enabled or disabled using the EQEP_QCAP_QPOS_CTL[31] PCSHDW bit. If the shadow mode is not enabled, the CPU writes directly to the active position compare register.

Copyright © 2024 Texas Instruments Incorporated

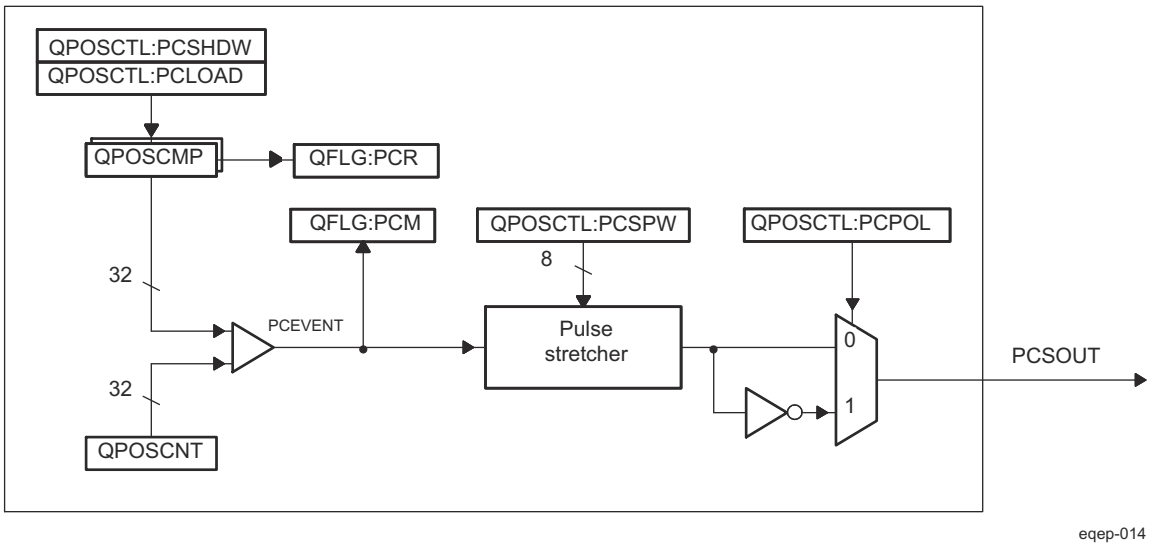


Figure 12-237. EQEP Position-compare Unit

In shadow mode, SW can configure the position-compare unit (EQEP_QCAP_QPOS_CTL[30] PCLOAD) to load the shadow register value into the active register on the following events and to generate the position-compare ready (EQEP_QINT_EN_FLG[23] PCRI_FLG) interrupt after loading.

- Load on compare match
- Load on position-counter zero event.

The position-compare match (EQEP_QINT_EN_FLG[24] PCMI_FLG) is set when the position-counter value (QPOSCNT) matches with the active position-compare register (EQEP_QPOSCMP) and the position-compare sync output of the programmable pulse width is generated on compare match to trigger an external device.

For example, if EQEP_QPOSCMP bitfield POSCMP = 0x2, the position-compare unit generates a position-compare event on the transition from 1 to 2 of the EQEP position counter for forward counting direction and on the transition from 3 to 2 of the EQEP position counter for reverse counting direction (see Figure 12-238).

Note

When EQEP_QCAP_QPOS_CTL[30] PCLOAD=0h, the shadow register is loaded into the active register as soon as POSCNT becomes zero, and it should not generate another shadow load if the POSCNT continues to stay at zero

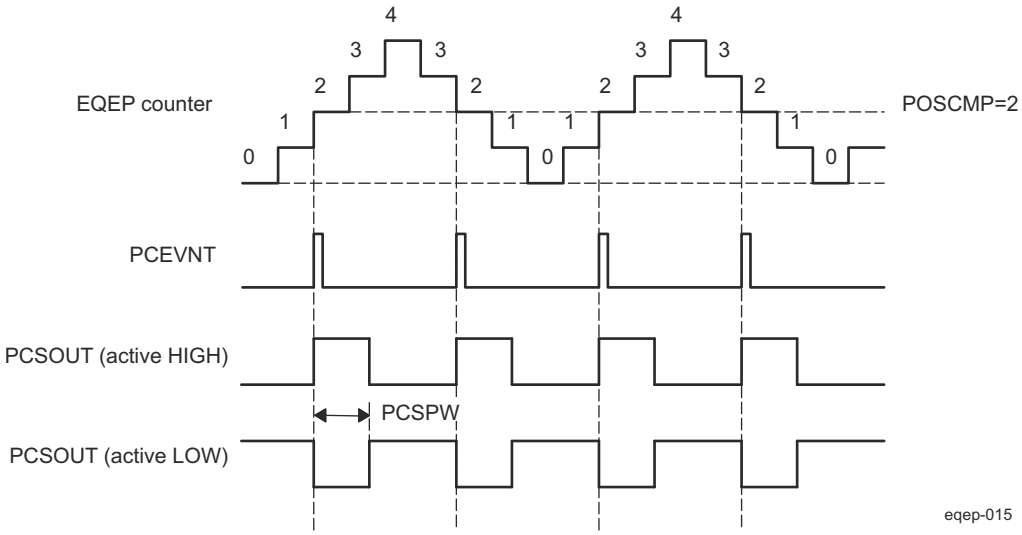


Figure 12-238. EQEP Position-compare Event Generation Points

Copyright © 2024 Texas Instruments Incorporated

The pulse stretcher logic in the position-compare unit generates a programmable position-compare sync pulse output on the position-compare match. In the event of a new position-compare match while a previous position-compare pulse is still active, then the pulse stretcher generates a pulse of specified duration from the new position-compare event as shown in Figure 12-239.

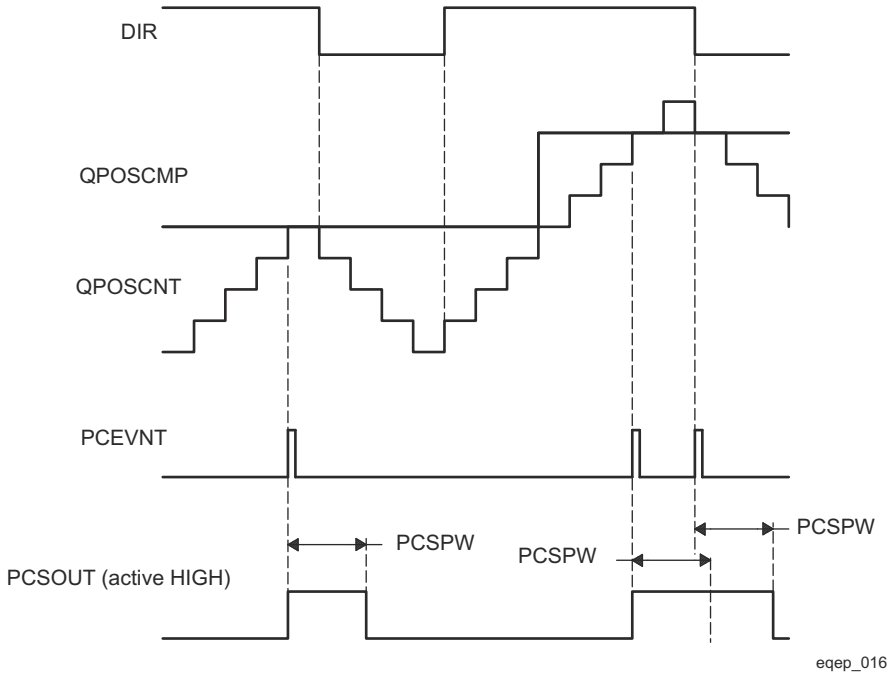


Figure 12-239. EQEP Position-compare Sync Output Pulse Stretcher

12.4.3.3.4 EQEP Edge Capture Unit

The EQEP peripheral includes an integrated edge capture unit to measure the elapsed time between the unit position events as shown in Figure 12-240. This feature is typically used for low speed measurement using the following equation:

$$V(k) = \frac{X}{t(k) - t(k-1)} = \frac{X}{\Delta T} \quad \text{eqep-017} \tag{21}$$

where,

- X - Unit position is defined by integer multiple of quadrature edges (see Figure 12-241)
- ΔT - Elapsed time between unit position events
- v(k) - Velocity at time instant "k"

The EQEP capture timer (QCTMR bitfield in EQEP_QCTMR register) runs from prescaled SYSCLKOUT and the prescaler is programmed by the EQEP_QCAP_QPOS_CTL[6-4] CCPS bits. The capture timer QCTMR value is latched into the capture period register (EQEP_QC_PRD_TLAT) on every unit position event and then the capture timer is reset.

Time measurement (ΔT) between unit position events will be correct if the following conditions are met:

- No more than 65,535 counts have occurred between unit position events.
- No direction change between unit position events.

The capture unit sets the EQEP overflow error flag (EQEP_QEP_STS_CT[3] COEF) in the event of capture timer overflow between unit position events. If a direction change occurs between the unit position events, then an error flag is set in the status register (EQEP_QEP_STS_CT[2] CDEF).

Copyright © 2024 Texas Instruments Incorporated

Capture Timer (EQEP_QCTMR register) and Capture period register (EQEP_QC_PRD_TLAT) can be configured to latch on following events.

- CPU read of EQEP_QPOSCNT register
- Unit time-out event

If the EQEP_QDEC_QEP_CTL[18] QCLM bit is cleared, then the capture timer and capture period values are latched into the EQEP_QC_PRD_TLAT and EQEP_QCPRDLAT registers, respectively, when the CPU reads the position counter in EQEP_QPOSCNT.

Note

Capture timer and capture period values are not latched for an emulation read.

If the EQEP_QDEC_QEP_CTL[18] QCLM bit is set, then the position counter, capture timer, and capture period values are latched into the EQEP_QPOSLAT, EQEP_QC_PRD_TLAT and EQEP_QCPRDLAT registers, respectively, on unit time out.

Figure 12-242 shows the capture unit operation along with the position counter.

Note

The EQEP_QCAP_QPOS_CTL register should not be modified dynamically (such as switching CAPCLK prescaling mode from SYSCLKOUT/4 to SYSCLKOUT/8, where SYSCLKOUT is equivalent to EQEPx__FICLK). The capture unit must be disabled before changing the prescaler.

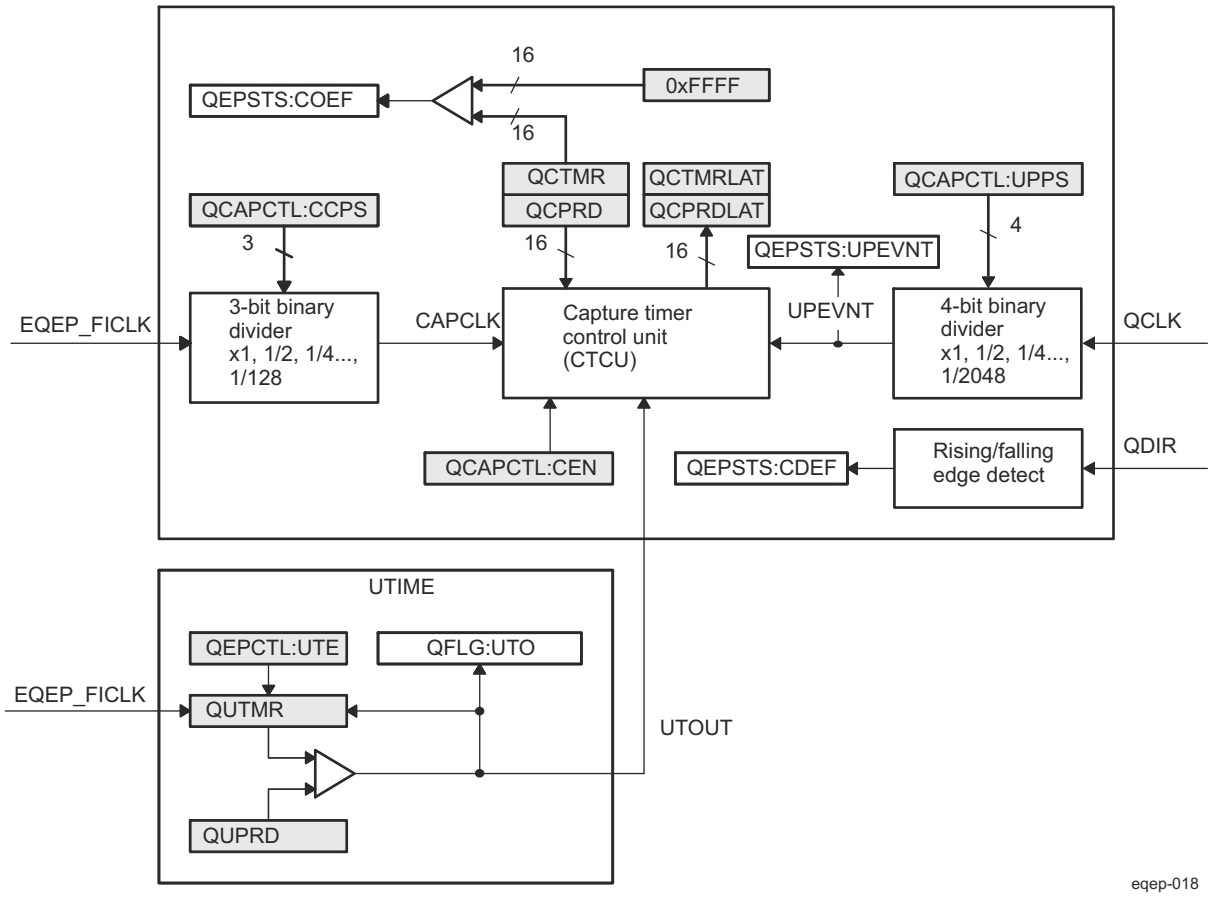
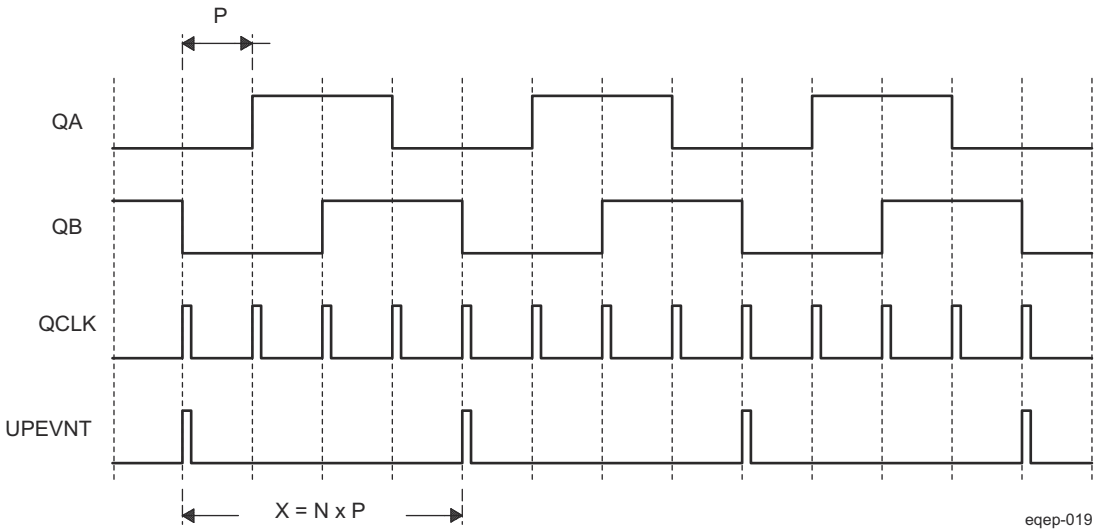


Figure 12-240. EQEP Edge Capture Unit

Copyright © 2024 Texas Instruments Incorporated



N - Number of quadrature periods selected using EQEP_QCAP_QPOS_CTL[3-0] UPPS bits

Figure 12-241. Unit Position Event for Low Speed Measurement (EQEP_QCAP_QPOS_CTL[UPPS] = 0010)

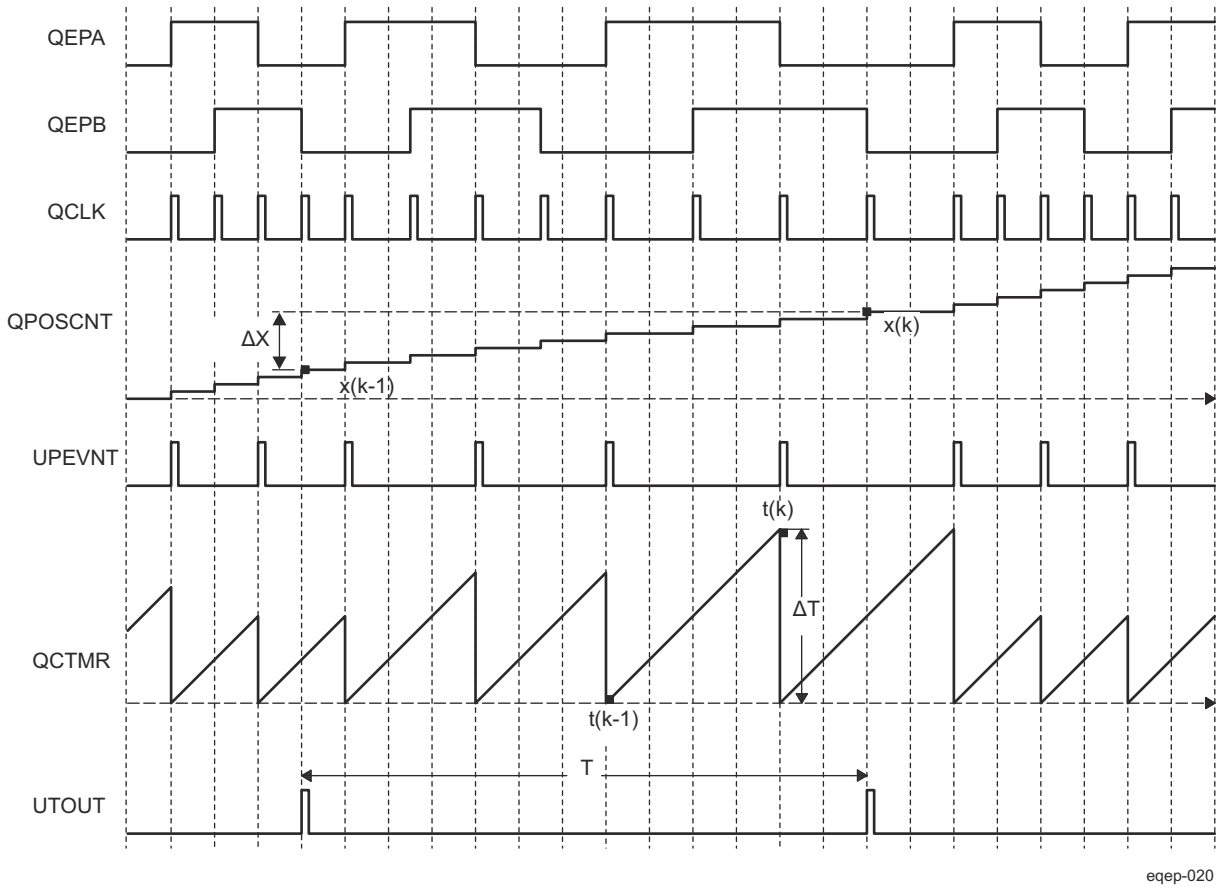


Figure 12-242. EQEP Edge Capture Unit - Timing Details

Velocity Calculation Equations:

Copyright © 2024 Texas Instruments Incorporated

$$V(k) = \frac{x(k) - x(k-1)}{T} = \frac{\Delta X}{T} \quad \text{or}$$
$$V(k) = \frac{X}{t(k) - t(k-1)} = \frac{X}{\Delta T}$$

eqep_021 (22)

where

v(k): Velocity at time instant k

x(k): Position at time instant k

x(k-1): Position at time instant k - 1

T: Fixed unit time or inverse of velocity calculation rate

ΔX: Incremental position movement in unit time

X: Fixed unit position

ΔT: Incremental time elapsed for unit position movement

t(k): Time instant "k"

t(k-1): Time instant "k - 1"

Unit time (T) and unit period (X) are configured using the EQEP_QUPRD and EQEP_QCAP_QPOS_CTL[3-0] UPPS registers. Incremental position output and incremental time output is available in the EQEP_QPOSLAT and EQEP_QCPRDLAT registers.

Parameter	Relevant Register to Configure or Read the Information
T	Unit Period Register (EQEP_QUPRD)
ΔX	Incremental Position = QPOSLAT(k) - QPOSLAT(k - 1)
X	Fixed unit position defined by sensor resolution and QCAPCTL[3-0] UPPS bits
ΔT	Capture Period Latch (QCPRDLAT)

12.4.3.3.5 EQEP Watchdog

The EQEP peripheral contains a 16-bit watchdog timer that monitors the quadrature-clock to indicate proper operation of the motion-control system. The EQEP watchdog timer is clocked from SYSCLKOUT/64 and the quadrate clock event (pulse) resets the watchdog timer. If no quadrature-clock event is detected until a period match (QWDPRD = QWDTMR), then the watchdog timer will time out and the watchdog interrupt flag will be set (EQEP_QINT_EN_FLG[20] WTOI_FLG). The time-out value is programmable through the watchdog period bit field (EQEP_QWD_TMR_PRD[31-16] QWDPRD).

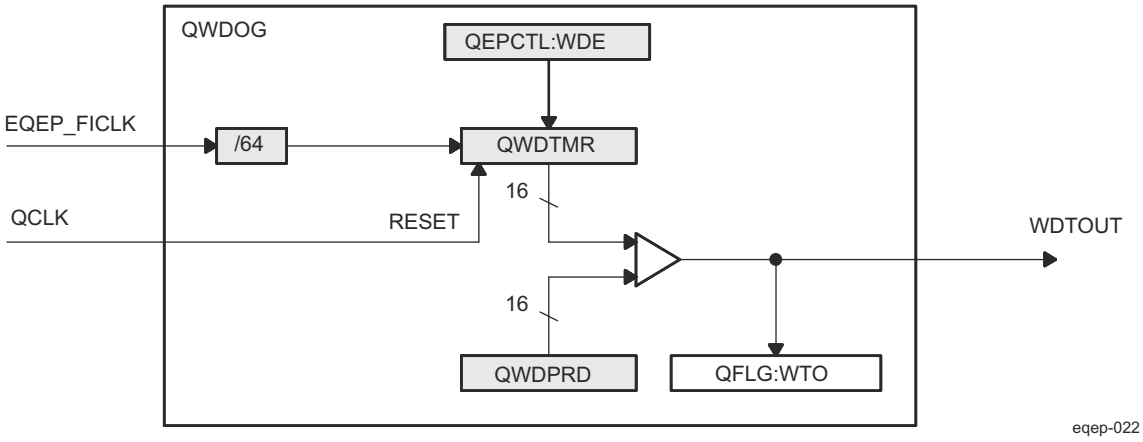


Figure 12-243. EQEP Watchdog Timer

Copyright © 2024 Texas Instruments Incorporated

12.4.3.3.6 Unit Timer Base

The EQEP peripheral includes a 32-bit timer (QUTMR) that is clocked by SYSCLKOUT to generate periodic interrupts for velocity calculations. The unit time out interrupt is set (EQEP_QINT_EN_FLG[27] UTOI_FLG) when the unit timer (QUTMR) matches the unit period register (EQEP_QUPRD).

The EQEP peripheral can be configured to latch the position counter, capture timer, and capture period values on a unit time out event. The latched values are used for velocity calculation as described in Section Section 12.4.3.3.4.

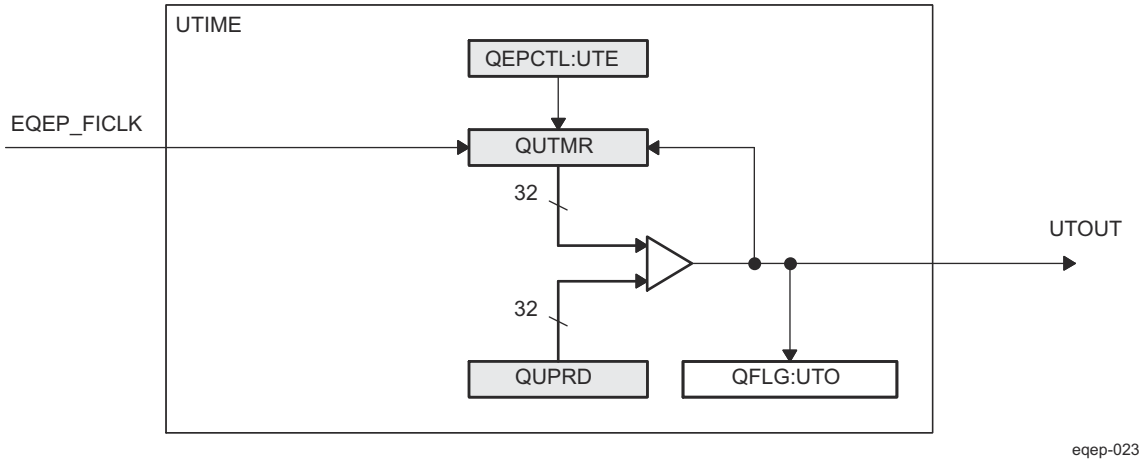


Figure 12-244. EQEP Unit Time Base

12.4.3.3.7 EQEP Interrupt Structure

Figure 12-245 shows how the interrupt mechanism works in the EQEP module.

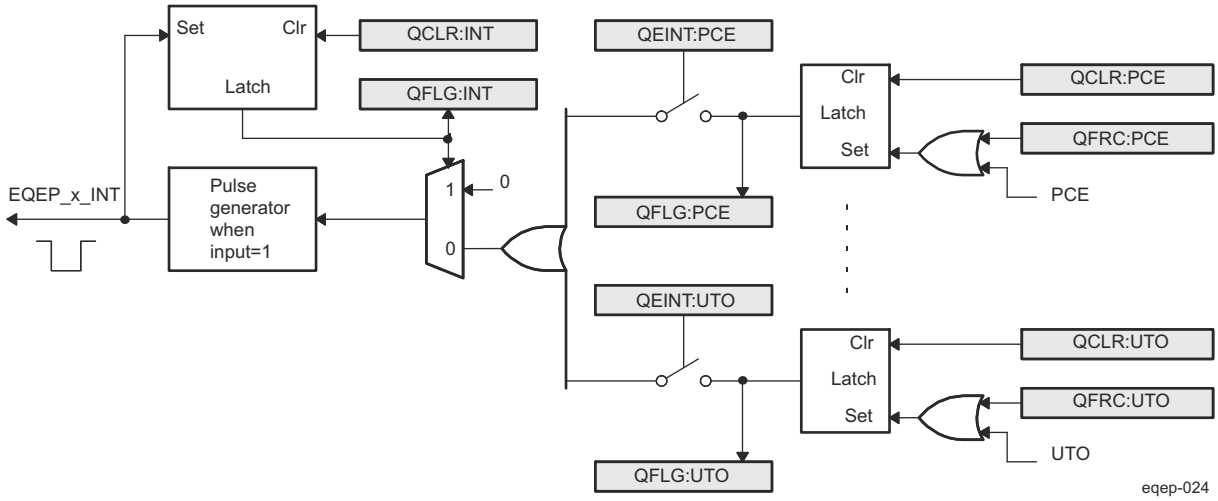


Figure 12-245. EQEP Interrupt Generation

Eleven interrupt events (PCE, PHE, QDC, WTO, PCU, PCO, PCR, PCM, SEL, IEL, and UTO) can be generated. The interrupt control register (EQEP_QINT_EN_FLG) is used to enable/disable individual interrupt event sources. The interrupt flag register (EQEP_QINT_EN_FLG) indicates if any interrupt event has been latched and contains the global interrupt flag bit (INT). An interrupt pulse is only generated to the interrupt controller if any of the interrupt events is enabled, the flag bit is 1 and the INT flag bit is 0. The interrupt service routine will need to clear the global interrupt flag bit and the serviced event, via the interrupt clear register (EQEP_QINT_CLR_FRC), before any other interrupt pulses are generated. SW can force an interrupt event by way of the interrupt force register (EQEP_QINT_CLR_FRC), which is useful for test purposes.

Copyright © 2024 Texas Instruments Incorporated

12.4.3.3.8 Summary of EQEP Functional Registers

Table 12-289 lists the registers with their memory locations, sizes, and reset values.

Table 12-289. EQEP Control and Status Functional Registers

Offset	Acronym	Register Description	Size (×16)/ #shadow
0h	EQEP_QPOSCNT	EQEP Position Counter Register	2/0
4h	EQEP_QPOSINIT	EQEP Position Counter Initialization Register	2/0
8h	EQEP_QPOSMAX	EQEP Maximum Position Count Register	2/0
Ch	EQEP_QPOSCMP	EQEP Position-Compare Register	2/1
10h	EQEP_QPOSILAT	EQEP Index Position Latch Register	2/0
14h	EQEP_QPOSSLAT	EQEP Strobe Position Latch Register	2/0
18h	EQEP_QPOSLAT	EQEP Position Counter Latch Register	2/0
1Ch	EQEP_QUTMR	EQEP Unit Timer Register	2/0
20h	EQEP_QUPRD	EQEP Unit Period Register	2/0
24h	EQEP_QWD_TMR_PRD	EQEP Watchdog Timer and Period Register	2/0
28h	EQEP_QDEC_QEP_CTL	EQEP Decoder and EQEP Control Register	2/0
2Ch	EQEP_QCAP_QPOS_CTL	EQEP Capture and Position Compare Control Register	2/0
30h	EQEP_QINT_EN_FLG	EQEP Interrupt Enable and Flag Register	2/0
34h	EQEP_QINT_CLR_FRC	EQEP Interrupt Clear and Forcing Register	2/0
38h	EQEP_QEP_STS_CT	EQEP Status and Capture Timer Register	2/0
3Ch	EQEP_QC_PRD_TLAT	EQEP Capture Period ant Timer Latch Register	2/0
40h	EQEP_QCPRDLAT	EQEP Capture Period Latch Register	1/0
5Ch	EQEP_PID	EQEP Revision ID Register	2/0

Copyright © 2024 Texas Instruments Incorporated

12.4.4 Controller Area Network (MCAN)

This section describes the Controller Area Network (MCAN) modules in the device.

12.4.4.1 MCAN Overview

The Controller Area Network (CAN) is a serial communications protocol which efficiently supports distributed real-time control. CAN has high immunity to electrical interference. In a CAN network, many short messages are broadcast to the entire network, which provides for data consistency in every node of the system.

The MCAN module supports both classic CAN and CAN FD (CAN with Flexible Data-Rate) specifications. CAN FD feature allows high throughput and increased payload per data frame. The classic CAN and CAN FD devices can coexist on the same network without any conflict.

They connect to the physical layer of the CAN network through external (for the device) transceivers. Each MCAN module supports flexible bit rates greater than 1 Mbps and is compliant to ISO 11898-1:2015.

12.4.4.1.1 MCAN Features

Each MCAN module implements the following features:

- Conforms with CAN Protocol 2.0 A, B and ISO 11898-1:2015
- Full CAN FD support (up to 64 data bytes)
- SAE J1939 support
- AUTOSAR support
- Up to 32 dedicated Transmit Buffers
- Configurable Transmit FIFO, up to 32 elements
- Configurable Transmit Queue, up to 32 elements
- Configurable Transmit Event FIFO, up to 32 elements
- Up to 64 dedicated Receive Buffers
- Two configurable Receive FIFOs, up to 64 elements each
- Up to 128 filter elements
- Internal Loopback mode for self-test
- Maskable interrupts, two interrupt lines
- Two clock domains (CAN clock/Host clock)
- Parity/ECC support - Message RAM single error correction and double error detection (SECDED) mechanism
- Local power-down and wakeup support
- Timestamp Counter

Unsupported Features:

- See the Module Integration section of this document for a list of module features not supported by the integration on this Device.

12.4.4.1.2 MCAN Ports

Table 12-290. MCAN Clocks and Resets

Clocks	
Module Clock Input	**Description**
MCAN_ICLK	Interface Clock
MCAN_FCLK	Functional Clock
Resets	
Module Reset Input	**Description**
MCAN_RST	Asynchronous Module Reset

Table 12-291. MCAN Hardware Requests

Interrupt Requests		
Module Interrupt Signal	**Description**	**Type**
MCAN_MCANSS_MCAN_LVL_INT_0	MCAN Line 0 Interrupt Request	Level
MCAN_MCANSS_MCAN_LVL_INT_1	MCAN Line 1 Interrupt Request	Level

Copyright © 2024 Texas Instruments Incorporated

Table 12-291. MCAN Hardware Requests (continued)

MCAN_MCANSS_EXT_TS_ROLLOVER_LVL_INT_0	MCAN External TimeStamp Counter Rollover Interrupt	Level
MCAN_MCANSS_ECC_CORR_LVL_INT_0	MCAN ECC Correctable Error Interrupt Request	Level
MCAN_MCANSS_ECC_UNCORR_LVL_INT_0	MCAN ECC Uncorrectable Error Interrupt Request	Level

12.4.4.2 MCAN Environment

This section describes the MCAN external connections (environment).

Table 12-292 describes the MCAN I/O signals.

Table 12-292. MCAN I/O Signals

Module Pin	I/O[(1)]	Description
RX	I	Serial data input from external CAN transceiver
TX	O	Serial data output to external CAN transceiver

(1) I = Input; O = Output; HiZ = High Impedance

Copyright © 2024 Texas Instruments Incorporated

12.4.4.3 MCAN Functional Description

The MCAN module performs CAN protocol communication according to ISO 11898-1:2015. The bit rate can be programmed to values greater than 1 Mbps. Additional transceiver hardware is required for the connection to the physical layer (CAN bus).

For communication on a CAN network, individual message frames can be configured. The message frames and identifier masks are stored in the Message RAM.

All functions concerning the handling of messages are implemented in the Message Handler.

The register set of the MCAN module can be accessed directly via the module interface. These registers are used to control and configure the CAN core and the Message Handler, and to access the Message RAM.

Figure 12-246 shows the MCAN module block diagram.

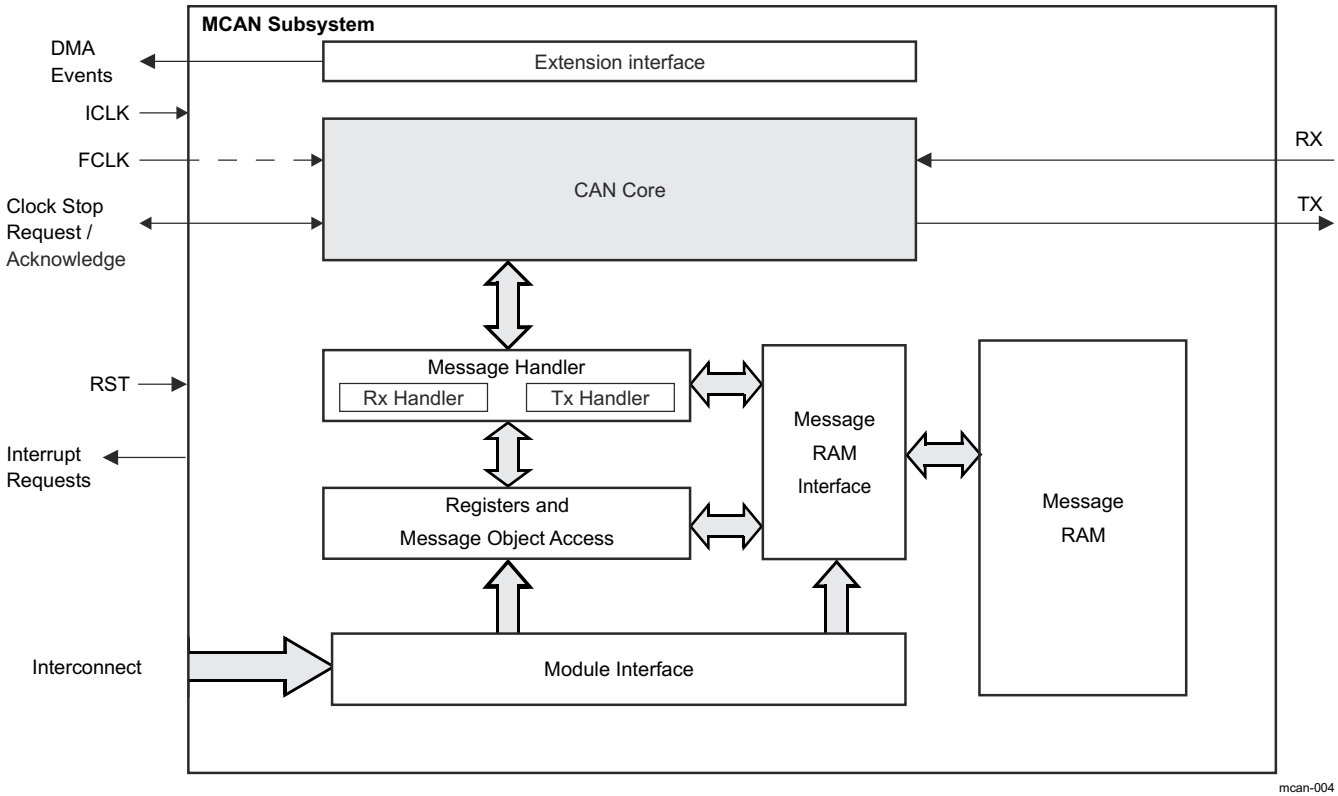


Figure 12-246. MCAN Block Diagram

The MCAN module blocks description:

- **CAN Core**: the CAN core consists of the CAN protocol controller and the Rx/Tx shift register. It handles all ISO 11898-1:2015 protocol functions and supports 11-bit and 29-bit identifiers.
- **Message Handler**: the Message Handler (Rx Handler and Tx Handler) is a state machine that controls the data transfer between the single-ported Message RAM and the CAN core's Rx/Tx shift register. It also handles the acceptance filtering and the Interrupt/DMA request generation as programmed in the control registers.
- **Message RAM**: the main purpose of the Message RAM is to store Rx/Tx messages, Tx Event elements, and Message ID Filter elements (for more information, see *Message RAM*).
- **Message RAM Interface**: enables connection between the Message RAM and the other blocks in the MCAN module.
- **Registers and Message Object Access**: data consistency is ensured by indirect accesses to the message objects. The interface registers have the same word-length as the Message RAM.
- **Module Interface**: provides connection to the Registers and Message Object Access block and Message RAM Interface block

Copyright © 2024 Texas Instruments Incorporated

- **Clocking**: two clocks are provided to the MCAN module: the peripheral synchronous clock (interface clock ICLK) and the peripheral asynchronous clock (functional clock - FCLK). For more information, see *MCU_MCAN Clocks and Resets* and *MCAN Clocks and Resets*.
- **Extension Interface**: this interface is used for DMA requests signaling (see *DMA Requests*).

12.4.4.3.1 Module Clocking Requirements

Two clocks are provided to the MCAN module:

- the peripheral synchronous clock (ICLK) as the general module clock source
- and the peripheral asynchronous clock (FCLK) provided to the CAN core for generating the CAN bit timing.

Within the MCAN module there is a synchronization mechanism implemented to ensure safe data transfer between the two clock domains. There is synchronization between the signals from the Host clock domain to the CAN clock domain and vice versa and between the reset signal to the Host clock domain and to the CAN clock domain.

Note

ICLK must always be higher or equal to FCLK, in order to achieve a stable functionality of the MCAN module. Here, also the frequency shift of the modulated ICLK has to be considered:

$f_{0,ICLK} \pm \Delta f_{FM,ICLK} \geq f_{FCLK}$

For more information on how to configure the relevant clock source registers, see *Clocking* and the device-specific Datasheet.

12.4.4.3.2 Interrupt and DMA Requests

The MCAN module provides interrupt and DMA requests. They are configured via the Host CPU. The Suspend Mode is requesting or forcing (based on MCANSS_CTRL[3] DBGSUSP_FREE bit) the MCAN module to go into initialization mode (see MCAN_CCCR[0] INIT bit) in which new interrupts and DMA requests will not be issued, that is to prevents the interrupt and DMA requests from propagating to the Host CPU (for more information, see Section 12.4.4.3.3.8.2, *Suspend Mode*).

12.4.4.3.2.1 Interrupt Requests

The MCAN module has two interrupt lines. There are 30 internal interrupt sources. Each source can be configured to drive one of the two interrupt lines. The interrupts are 'level high' interrupts.

The MCAN core provides two interrupt requests (for Line 0 and Line 1).

For more information, see the following registers:

- Interrupt Register (MCAN_IR)
- Interrupt Enable (MCAN_IE)
- Interrupt Line Select (MCAN_ILS)
- Interrupt Line Enable (MCAN_ILE)

The MCAN module is capable of issuing ECC interrupts. After clearing the ECC interrupt source, the application software must also write 1 to EOI register (MCANSS_ECC_SEC_EOI_REG/MCANSS_ECC_DED_EOI_REG). For more information, see *ECC Aggregator*.

The MCAN module supports External Timestamp Counter. When the External Timestamp Counter rolls over it produces an interrupt (see *External Timestamp Counter*).

For more information, see the following registers:

- Interrupt Clear Shadow Register (MCANSS_ICS)
- Interrupt Raw Status Register (MCANSS_IRS)
- Interrupt Enable Clear Shadow Register (MCANSS_IECS)
- Interrupt Enable Register (MCANSS_IE)
- Interrupt Enable Status Register (MCANSS_IES)
- End Of Interrupt Register (MCANSS_EOI)
- External Timestamp Prescaler Register (MCANSS_EXT_TS_PRESCALER)

Copyright © 2024 Texas Instruments Incorporated

- External Timestamp Unserviced Interrupts Counter Register (MCANSS_EXT_TS_UNSERVICED_INTR_CNTR)

For more information about available Interrupt Requests, see *MCU_MCAN Hardware Requests* and *MCAN Hardware Requests*.

12.4.4.3.2.2 DMA Requests

Functional transmit and Filter DMA requests are generated by the MCAN module based on the signaling in the Extension Interface. The DMA signaling uses a simple DMA request active high pulse.

The active high pulse indicates a pending message is transmitted (see MCAN_TXBRP). This pulse can be used to transfer another message to the Tx Buffer, which would need to be followed by writing 1 to the corresponding MCAN_TXBAR[0] AR bit to mark a new Tx message pending transmission.

The Parity on Tx DMA Events is available using an EDC Controller which can be accessed through the ECC Aggregator.

Standard and Extended message filters can be set to issue a pulse when a filter match occurs. These 'Filter Events' can be used to DMA messages from the Rx FIFO. The events are high level single clock cycle pulses (ICLK).

12.4.4.3.3 Operating Modes

12.4.4.3.3.1 Software Initialization

Setting the MCAN_CCCR[0] INIT bit to 1 starts a software initialization. This is done either by software or by a hardware reset, when an uncorrected bit error was detected in the Message RAM, or by going Bus_Off state. While the MCAN_CCCR[0] INIT bit is set, the message transfer is stopped and the status of the output TX pin is recessive (high). The counters of the Error Management Logic (EML) are unchanged. Setting the MCAN_CCCR[0] INIT bit does not change any configuration register. Resetting the MCAN_CCCR[0] INIT bit finishes the software initialization. After waiting for the occurrence of a sequence of 11 consecutive recessive bits (indication for Bus_Idle state) the message transfer starts.

Access to the MCAN configuration registers is only enabled when both MCAN_CCCR[0] INIT and MCAN_CCCR[1] CCE bits are set (write protection).

The MCAN_CCCR[1] CCE bit can only be set/reset while the MCAN_CCCR[0] INIT = 1. The MCAN_CCCR[1] CCE bit is automatically reset when the MCAN_CCCR[0] INIT bit is reset.

The following registers are reset when the MCAN_CCCR[1] CCE bit is set:

- MCAN_HPMS - High Priority Message Status
- MCAN_RXF0S - Rx FIFO 0 Status
- MCAN_RXF1S - Rx FIFO 1 Status
- MCAN_TXFQS - Tx FIFO/Queue Status
- MCAN_TXBRP - Tx Buffer Request Pending
- MCAN_TXBTO - Tx Buffer Transmission Occurred
- MCAN_TXBCF - Tx Buffer Cancellation Finished
- MCAN_TXEFS - Tx Event FIFO Status

The Timeout Counter value MCAN_TOCV[15-0] TOC field is preset to the value configured by the MCAN_TOCC[31-16] TOP field when the MCAN_CCCR[1] CCE bit is set.

In addition the Tx Handler and Rx Handler are held in idle state while MCAN_CCCR[1] CCE = 1.

The following registers are only writeable while MCAN_CCCR[1] CCE = 0

- MCAN_TXBAR - Tx Buffer Add Request
- MCAN_TXBCR - Tx Buffer Cancellation Request

MCAN_CCCR[7] TEST and MCAN_CCCR[5] MON bits can only be set by the Host CPU while MCAN_CCCR[0] INIT = 1 and MCAN_CCCR[1] CCE = 1. Both bits may be reset at any time. The MCAN_CCCR[6] DAR bit can only be set/reset while MCAN_CCCR[0] INIT = 1
and MCAN_CCCR[1] CCE = 1.

Copyright © 2024 Texas Instruments Incorporated

Table 12-293 shows the steps to configure the MCAN module.

Table 12-293. Steps to Configure MCAN Module

Step	Operation	Description	Pseudo Code
1	Initialize MCAN_CCCR	Set MCAN_CCCR[0] INIT bit and check that it has been set	INIT = 1; If INIT ≠ 1, wait until it is
2	Unlock protected registers	Set MCAN_CCCR[1] CCE bit	CCE = 1;
3	Configure CAN mode	Set MCAN_CCCR[8] FDOE bit to CAN FD	FDOE = 1 for CAN FD FDOE = 0 for CAN
4	Configure Bit Rate Switching	Set MCAN_CCCR[9] BRSE bit	BRSE = 1 with bit rate switching BRSE = 0 without bit rate switching
5	Set bit timing	Set MCAN_NBTP register	
6	Lock protected registers	Clear MCAN_CCCR[1] CCE bit	CCE = 0;
7	Return MCAN module to normal operation	Clear MCAN_CCCR[0] INIT bit and check it has been cleared	INIT = 0; If INIT ≠ 0, wait until it is

12.4.4.3.3.2 Normal Operation

Once the MCAN module is initialized and the MCAN_CCCR[0] INIT bit is reset to zero, the MCAN module synchronizes itself to the CAN bus and is ready for communication. After passing the acceptance filtering, received messages including Message Identifier (ID) and Data Length Code (DLC) are stored into a dedicated Rx Buffer or into Rx FIFO 0/Rx FIFO 1.

For messages to be transmitted dedicated Tx Buffers and/or a Tx FIFO or a Tx Queue can be initialized or updated.

Note

Automated transmission on reception of remote frames is not supported.

12.4.4.3.3.3 CAN FD Operation

The CAN FD standard allows extended frames to be sent, up to 64 data bytes in a single frame at a higher bit rate for the data phase of a frame, up to 8 Mbps. The CAN FD standard introduces the ability to switch from one bit rate to another. Extended Data Length (EDL), as shown in Figure 12-247, sets a data length of up to 8 or up to 64 data bytes. Bit Rate Switching (BRS) indicates whether two bit rates (the data phase is transmitted at a different bit rate to the arbitration phase) are enabled.

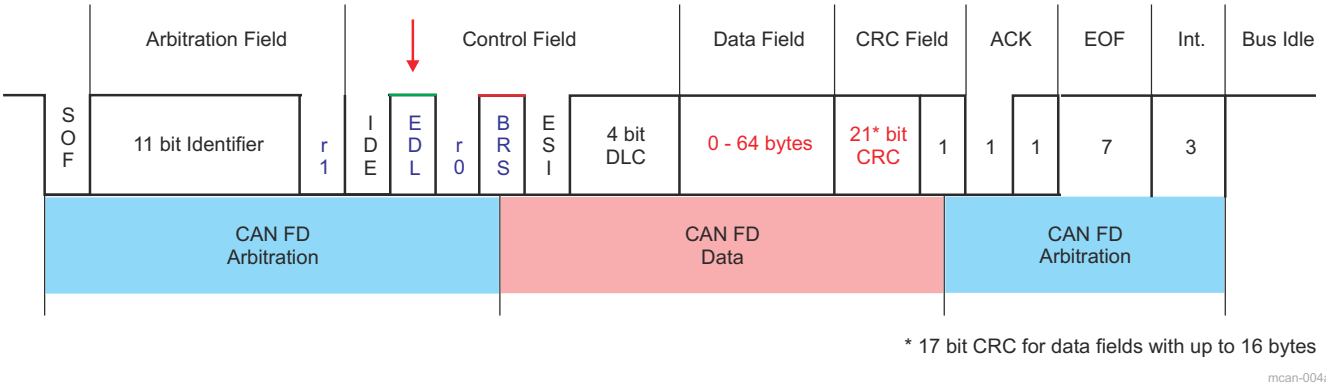


Figure 12-247. CAN FD Frame

Copyright © 2024 Texas Instruments Incorporated

Note

Figure 12-247 presents CAN FD frame according to the Non-ISO CAN FD (legacy) protocol. In the new ISO CAN FD protocol the CRC Field includes additional 5 bits (three stuff bit counter (SBC) bits and two parity bits). With these additional bits, a weakness identified in the error detection scheme chosen by the original protocol is removed. By setting MCAN_CCCR[15] NISO bit, the ISO or Non-ISO CAN FD format can be chosen. In CAN network ISO CAN FD and non-ISO CAN FD devices should never mix.

There are two variants of CAN FD frame transmission:

- CAN FD frame transmission without bit rate switching
- CAN FD frame transmission where control field, data field, and CRC field are transmitted with a higher bit rate than the beginning and the end of the frame

In the CAN frames FDF = recessive (logical 1) signifies a CAN FD frame, FDF = dominant (logical 0) signifies a Classic CAN frame. In a CAN FD frame, the two bits following FDF - res and BRS, decide whether the bit rate inside of this CAN FD frame is switched. A CAN FD bit rate switch is signified by res = dominant and BRS = recessive. Note that the coding of res = recessive is reserved for future expansion of the protocol. In case the MCAN module receives a frame with FDF = recessive and res = recessive, it will signal a Protocol Exception Event by setting the MCAN_PSR[14] EXE bit. When Protocol Exception Handling is enabled (MCAN_CCCR[12] PXHD = 0), this causes the operation state to change from Receiver (MCAN_PSR[4-3] ACT = 10) to Integrating (MCAN_PSR[4-3] ACT = 00) at the next sample point. In case Protocol Exception Handling is disabled (MCAN_CCCR[12] PXHD = 1), the MCAN will treat a recessive res bit as an form error and will respond with an error frame.

CAN FD operation is enabled by programming the MCAN_CCCR[8] FDOE bit. In case MCAN_CCCR[8] FDOE = 1, transmission and reception of CAN FD frames is enabled. Transmission and reception of Classic CAN frames is always possible. Whether a CAN FD frame or a Classic CAN frame is transmitted can be configured via the FDF bit in the respective Tx Buffer element.

With MCAN_CCCR[8] FDOE = 0, received frames are interpreted as Classic CAN frames, which leads to the transmission of an error frame when receiving a CAN FD frame. When CAN FD operation is disabled, no CAN FD frames are transmitted even if the FDF bit of a Tx Buffer element is set. The MCAN_CCCR[8] FDOE and MCAN_CCCR[9] BRSE bits can only be changed while the MCAN_CCCR[0] INIT and MCAN_CCCR[1] CCE bits are both set. With MCAN_CCCR[8] FDOE = 0, the setting of bits FDF and BRS is ignored and frames are transmitted in Classic CAN format.

With MCAN_CCCR[8] FDOE = 1 and MCAN_CCCR[9] BRSE = 0, only FDF bit of a Tx Buffer element is evaluated. With MCAN_CCCR[8] FDOE = 1 and MCAN_CCCR[9] BRSE = 1, transmission of CAN FD frames with bit rate switching is enabled. All Tx Buffer elements with bits FDF and BRS set are transmitted in CAN FD format with bit rate switching.

A mode change during CAN operation is only recommended under the following conditions:

- The failure rate in the CAN FD data phase is significant higher than in the CAN FD arbitration phase. In this case disable the CAN FD bit rate switching option for transmissions.
- During system startup all nodes are transmitting Classic CAN messages until it is verified that they are able to communicate in CAN FD format. If this is true, all nodes switch to CAN FD operation.
- Wakeup messages in CAN Partial Networking have to be transmitted in Classic CAN format.
- End-of-line programming in case not all nodes are CAN FD capable. Non CAN FD nodes are held in Silent mode until programming has completed. Then all nodes switch back to Classic CAN communication.

In the CAN FD format, the DLC coding differs from the standard CAN format (see Table 12-294).

Table 12-294. DLC Coding

DLC	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Number of Data Bytes in Standard CAN	0	1	2	3	4	5	6	7	8	8	8	8	8	8	8	8
Number of Data Bytes in CAN FD	0	1	2	3	4	5	6	7	8	12	16	20	24	32	48	64

Copyright © 2024 Texas Instruments Incorporated

For CAN FD frames, the bit timing will be switched inside the frame after the BRS (Bit Rate Switch) bit in case this bit is recessive. In the CAN FD arbitration phase, before the BRS bit, the nominal CAN bit timing (see Figure 12-248) is used as configured by the Nominal Bit Timing and Prescaler Register MCAN_NBTP. In the following CAN FD data phase, the data phase bit timing is used as configured by the Data Bit Timing and Prescaler Register MCAN_DBTP. The bit timing is switched back from the data phase timing at the CRC delimiter or when an error is detected, whichever occurs first.

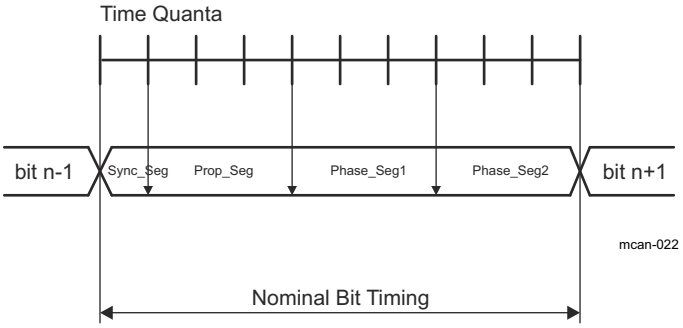


Figure 12-248. CAN Bit Timing

The maximum configurable data phase bit timing depends on the CAN clock frequency (FCLK). Example: with FCLK = 20 MHz and the shortest configurable bit time of 4 t_q (time quanta), the bit rate in the data phase is 5 Mbps.

In both data frame formats, CAN FD and CAN FD with bit rate switching, the value of the ESI (Error Status Indicator) bit depends on transmitter's error state (see MCAN_PSR[11] RESI bit) monitored at the start of the transmission. If the transmitter has error passive flag the ESI bit is transmitted recessive, else it is transmitted dominant.

12.4.4.3.3.4 Transmitter Delay Compensation

12.4.4.3.3.4.1 Description

When only one CAN FD node is transmitting and all others are receivers the length of the bus line has no impact. When transmitting via the TX pin the MCAN module receives the transmitted data from its local CAN transceiver via the RX pin. The received data is delayed. If the transmitter delay is greater than TSEG1 (time segment before sample point), a bit error is detected.

The MCAN module provides a delay compensation mechanism to compensate the transmitter delay. The compensation mechanism enables transmission with higher bit rates during the CAN FD data phase independent of the delay of a specific CAN transceiver. Without transmitter delay compensation the bit rate in the data phase is limited by the transmitter delay.

The mechanism enables configurations where the data bit time is shorter than the transmitter delay (it is described in detail in ISO 11898-1:2015). The transmitter delay compensation is enabled by setting the MCAN_DBTP[23] TDC bit to 1.

The delayed transmit data is compared against the received data at the Secondary Sample Point (SSP) in order to check for bit errors during the data phase of transmitting nodes. If a bit error is detected, the transmitter will react on this bit error at the next following regular sample point. During arbitration phase the delay compensation is always disabled.

The received bit is compared against the transmitted bit at the SSP. The SSP position is defined as the sum of the measured delay from the MCAN's transmit output TX pin through the transceiver to the receive input RX pin plus the transmitter delay compensation offset configured by the MCAN_TDCR[14-8] TDCO field (see Figure 12-249). The transmitter delay compensation offset is used to adjust the position of the SSP inside the received bit (example: half of the bit time in the data phase). The position of the SSP is rounded down to the next integer number of mtq.

Copyright © 2024 Texas Instruments Incorporated

The actual transmitter delay compensation value can be checked by reading the MCAN_PSR[22-16] TDCV field. This field is cleared when the MCAN_CCCR[0] INIT bit is set and is updated at each transmission of CAN FD frame while the MCAN_DBTP[23] TDC bit is set.

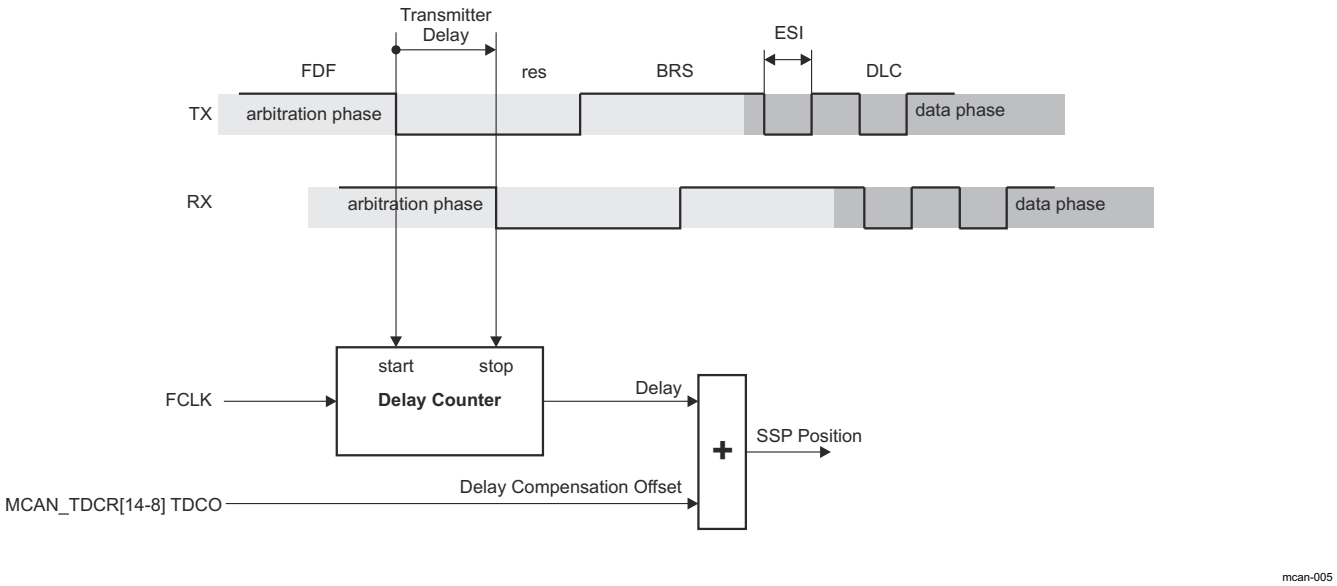


Figure 12-249. Transmitter Delay Measurement

12.4.4.3.3.4.2 Transmitter Delay Compensation Measurement

When transmitter delay compensation is enabled (by programming MCAN_DBTP[23] TDC = 1), the measurement is started within each transmitted CAN FD frame at the falling edge of FDF bit to bit res. The measurement is stopped when this edge is seen at the receive input RX pin of the transmitter. The resolution of this measurement is one mtq (see Figure 12-249). The mtq (minimum time quantum) dimension is equal to the CAN clock period (FCLK).

The use of a transmitter delay compensation filter window can be enabled by programming MCAN_TDCR[6-0] TDCF field. This filter feature defines a minimum value for the SSP position to avoid the case in which a dominant glitch inside the received FDF bit ends the delay compensation measurement before the falling edge of the received res bit, resulting in an early taken SSP position. Dominant edges on the RX pin, that would result in an earlier SSP position are ignored for transmitter delay measurement. The measurement is stopped when the SSP position is at least MCAN_TDCR[6-0] TDCF field and the RX pin is low.

The following boundary conditions have to be considered:

- The sum of the measured delay from the TX pin to the RX pin and the configured transmitter delay compensation offset (MCAN_TDCR[14-8] TDCO field) has to be less than 6 bit times in the data phase.
- The sum of the measured delay from the TX pin to the RX pin and the configured transmitter delay compensation offset (MCAN_TDCR[14-8] TDCO) field has to be less or equal 127 mtq. In case this sum exceeds 127 mtq, the maximum value of 127 mtq is used for transmitter delay compensation.
- The data phase ends at the sample point of the CRC delimiter, that stops checking of receive bits at the SSPs.

12.4.4.3.3.5 Restricted Operation Mode

In Restricted Operation Mode the CAN node is able to receive data and remote frames and to give acknowledge to valid frames, but it does not send data frames, remote frames, active error frames, or overload frames. In case of an error condition or overload condition, it does not send dominant bits, instead it waits for the occurrence of bus idle condition to resynchronize itself to the CAN communication. The recieve and transmit error counters (MCAN_ECR[14-8] REC and MCAN_ECR[7-0] TEC) are frozen while CAN error logging (MCAN_ECR[23-16] CEL) is active. The Host CPU can set the MCAN module into Restricted Operation Mode by setting MCAN_CCCR[2] ASM bit. The bit can only be set by the Host CPU at any time when both MCAN_CCCR[2] CCE and MCAN_CCCR[0] INIT bits are set to 1.

Copyright © 2024 Texas Instruments Incorporated

The Restricted Operation Mode is automatically entered when the Tx Handler was not able to read data from the Message RAM in time. To leave Restricted Operation Mode, the Host CPU has to reset MCAN_CCCR[2] ASM bit. This mode can be used in applications that adapt themselves to different CAN bit rates. In this case the application tests different bit rates and leaves the Restricted Operation Mode after it has received a valid frame.

Note

The Restricted Operation Mode must not be combined with the Internal Loopback Mode .

12.4.4.3.3.6 Bus Monitoring Mode

Entering Bus Monitoring Mode is done by setting the MCAN_CCCR[5] MON bit to 1. In this mode (see ISO 11898-1:2015, *Bus Monitoring* section), the MCAN module is able to receive valid data and remote frames, but cannot start a transmission. The MCAN module sends only recessive bits on the CAN bus. If the MCAN module is required to send a dominant bit (ACK bit, overload flag, active error flag), the bit is rerouted internally so that the MCAN module monitors this dominant bit, although the CAN bus may remain in recessive state. In Bus Monitoring Mode the MCAN_TXBRP register is held in reset state. The Bus Monitoring Mode can be used to analyze the traffic on a CAN bus without affecting it by the transmission of dominant bits. Figure 12-250 shows the connection of the TX and RX signals to the MCAN module in Bus Monitoring Mode.

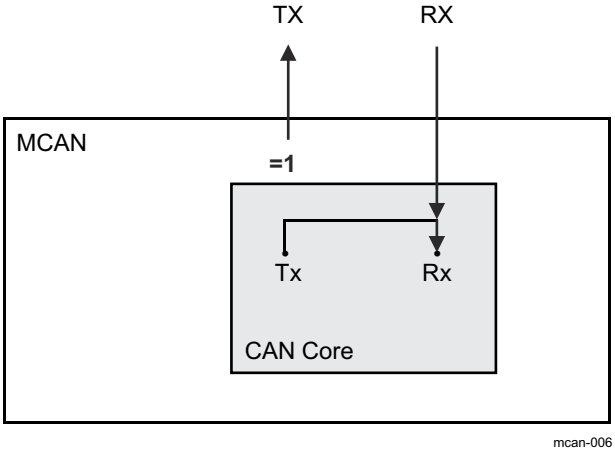


Figure 12-250. Connection of Signals in Bus Monitoring Mode

12.4.4.3.3.7 Disabled Automatic Retransmission (DAR) Mode

According to the CAN Specification (see ISO11898-1:2015, *Recovery Management* section), the MCAN module provides means for automatic retransmission of frames that have lost arbitration or that have been disturbed by errors during transmission. By default automatic retransmission is enabled (see the MCAN_CCCR[6] DAR bit).

12.4.4.3.3.7.1 Frame Transmission in DAR Mode

In DAR mode all transmissions are automatically cancelled after they started on the CAN bus. A Tx Buffer's Tx Request Pending MCAN_TXBRP[xx] TRPx bit is reset after successful transmission, when a transmission has not yet been started at the point of cancellation, has been aborted due to lost arbitration, or when an error occurred during frame transmission.

Successful transmission:

- Corresponding Tx Buffer Transmission Occurred MCAN_TXBTO[xx] TOx bit is set
- Corresponding Tx Buffer Cancellation Finished MCAN_TXBCF[xx] CFx bit is not set

Successful transmission in spite of cancellation:

- Corresponding Tx Buffer Transmission Occurred MCAN_TXBTO[xx] TOx bit is set
- Corresponding Tx Buffer Cancellation Finished MCAN_TXBCF[xx] CFx bit is set

Arbitration lost or frame transmission disturbed:

- Corresponding Tx Buffer Transmission Occurred MCAN_TXBTO[xx] TOx bit is not set

Copyright © 2024 Texas Instruments Incorporated

- Corresponding Tx Buffer Cancellation Finished MCAN_TXBCF[xx] CFx bit is set

In case of a successful frame transmission, and if storage of Tx events is enabled, a Tx Event FIFO element is written with Event Type ET = 10 (transmission in spite of cancellation).

12.4.4.3.3.8 Power Down (Sleep Mode)

The entering in Power Down mode is controlled via two sources:
- PSC (via clock stop request signal)
- Software (by writing to the MCAN_CCCR[4] CSR bit)

As long as the clock stop request signal is active, the MCAN_CCCR[4] CSR bit is read as 1.

When all pending transmission requests have completed, the MCAN module waits until bus idle state is detected. Then the MCAN module sets the MCAN_CCCR[0] INIT bit to 1 to prevent any further CAN transfers.

The MCAN module acknowledges that it is ready for power down:
- By asserting clock stop acknowledge signal to the PSC (in case of PSC source).
- By setting the MCAN_CCCR[3] CSA flag bit to 1 (in case of Software source).

In this state, before the clocks are switched off, further register accesses can be made. Now the module clock inputs ICLK and FCLK may be switched off.

To leave power down mode, the application has to turn on the module clocks before resetting the input clock stop request signal respectively the MCAN_CCCR[4] CSR flag bit. The MCAN will acknowledge this by resetting the output clock stop acknowledge signal respectively the MCAN_CCCR[3] CSA flag bit. Afterwards, the application can restart CAN communication by resetting the MCAN_CCCR[0] INIT bit.

Restoring the clocks from clock stop mode, needs to be done according to how the clock stop was initiated:
- If Software asserts the MCAN_CCCR[3] CSA flag bit, once the MCAN module goes idle, the MCAN_CCCR[0] INIT bit is set. To get it started again, Software needs to write 0 to the MCAN_CCCR[0] INIT bit.
- If PSC is issuing a clock stop request, than there are two options for waking up:
  - After removing clock stop request signal, Software would need to write 0 to the MCAN_CCCR[0] INIT bit, or
  - If the MCANSS_CTRL[5] AUTOWAKEUP bit is set, than after removing clock stop request signal, an FSM inside the MCAN module will reset the MCAN_CCCR[0] INIT bit (without Software).

12.4.4.3.3.8.1 External Clock Stop Mode

The MCAN module supports two external clock stop modes:

- Immediate
- Graceful

In a graceful clock stop mode, when the clock stop request is asserted, the MCAN core will respond with clock stop acknowledge when all pending Tx messages have been processed and an Idle line had been detected. The MCAN_CCCR[0] INIT bit will be set, the MCAN core will go and stay Idle.

The automatic wakeup feature is enabled by setting the MCANSS_CTRL[5] AUTOWAKEUP and MCANSS_CTRL[4] WAKEUPREQEN bits to 1 (for more information, see Section 12.4.4.3.3.8.3, *Wakeup request*). When external clock stop request is removed and no suspend request is active, a read-modify-write to the MCAN_CCCR[0] INIT bit is performed to clear it.

12.4.4.3.3.8.2 Suspend Mode

The MCAN module supports two suspend modes:

- Immediate
- Graceful

In a graceful suspend mode (see the MCANSS_CTRL[3] DBGSUSP_FREE bit), when the suspend request is asserted, a clock stop request to the MCAN core is performed. The MCAN core will respond with clock stop acknowledge when all pending Tx messages have been processed and an Idle line had been detected. At that

Copyright © 2024 Texas Instruments Incorporated

point the MCAN_CCCR[0] INIT bit will be set, the MCAN core will go and stay Idle. The suspend state can be verified by reading MCAN_CCCR[0] INIT bit.

The automatic wakeup feature is enabled by setting the MCANSS_CTRL[5] AUTOWAKEUP and MCANSS_CTRL[4] WAKEUPREQEN bits to 1 (for more information, see Section 12.4.4.3.3.8.3, *Wakeup request*). When suspend request is removed, if no external clock stop request is active, a read-modify-write to the MCAN_CCCR[0] INIT bit is performed to clear it.

During suspend mode the auto-clear feature is disabled. The following register fields have an auto-clear feature:

- MCAN_ECR[23-16] CEL
- MCAN_PSR[2-0] LEC
- MCAN_PSR[10-8] DLEC
- MCAN_PSR[11] RESI
- MCAN_PSR[12] RBRS
- MCAN_PSR[13] RFDF
- MCAN_PSR[14] PXE

12.4.4.3.3.8.3 Wakeup request

Issuing a clock stop request puts the MCAN module into Power Down mode (Sleep Mode). During transition from IDLE to ACTIVE, if the MCANSS_CTRL[5] AUTOWAKEUP and MCANSS_CTRL[4] WAKEUPREQEN bits are enabled, after the MCAN Core respond to the removal of the clock stop request with removing the clock stop acknowledge, a read-modify-write will be issued to clear the MCAN_CCCR[0] INIT bit and the MCAN core will resume operation.

If the MCANSS_CTRL[4] WAKEUPREQEN bit is set, the MCAN module provides a wakeup request on the following wakeup event:

- The receive RX pin is dominant (logical 0)

The wakeup request is de-asserted when any of the following conditions occur:

- Clock stop request is removed and clock stop acknowledge is de-asserted
- A reset is applied to the MCAN module

12.4.4.3.3.9 Test Modes

The MCAN_TEST register write access is enabled by setting the test mode enable MCAN_CCCR[7] TEST bit to 1. The MCAN_TEST register allows the configuration of the test modes and test functions.

The CAN transmit TX pin has four output functions. One of those functions can be selected by programming the MCAN_TEST[6-5] TX field. Additionally to its default function (the serial data output) it can drive the CAN Sample Point signal to monitor the MCAN's bit timing and it can drive constant dominant or recessive values.

The actual value of the CAN receive RX pin can be monitored from MCAN_TEST[7] RX bit. Both functions can be used to check the CAN bus physical layer. Due to the synchronization mechanism between CAN clock (FCLK) and Host clock (ICLK) domain, there may be a delay of several Host clock periods between writing to the MCAN_TEST[6-5] TX field until the new configuration is visible at the output TX pin. This applies also when reading input RX pin via the MCAN_TEST[7] RX bit.

Note

Test modes should be used for self test only. The software control for TX pin interferes with all CAN protocol functions. It is not recommended to use test modes for application.

12.4.4.3.3.9.1 Internal Loopback Mode

The MCAN module can be set into Internal Loopback Mode by programming MCAN_TEST[4] LBCK and MCAN_CCCR[5] MON bits to 1. The Internal Loopback Mode is used for a 'Hot Selftest'. The 'Hot Selftest' allows the MCAN module to be tested without affecting a running CAN system connected to the TX and RX pins. In this mode RX pin is disconnected from the MCAN module and TX pin is held recessive. Figure 12-251 shows the connection of the TX and RX pins to the MCAN module in case of Internal Loopback Mode.

Copyright © 2024 Texas Instruments Incorporated

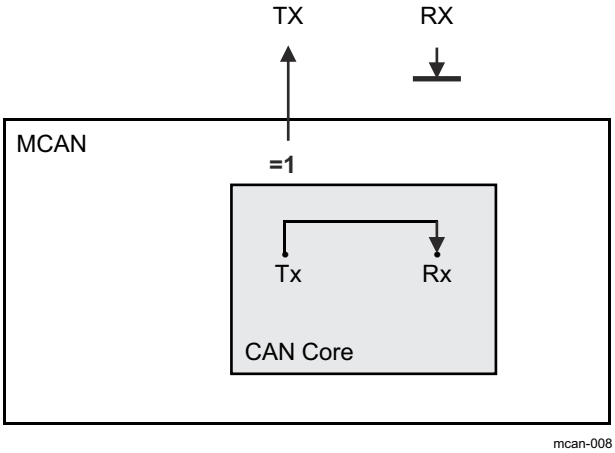


Figure 12-251. Internal Loopback Mode

12.4.4.3.4 Timestamp Generation

The MCAN module has integrated a 16-bit wrap-around counter for timestamp generation. The timestamp counter prescaler MCAN_TSCC[19-16] TCP field can be configured to clock the counter in multiples of CAN bit times (1-16). The counter is readable via the MCAN_TSCV[15-0] TSC field. A write access to the MCAN_TSCV register resets the counter to zero. When the timestamp counter wraps around the interrupt MCAN_IR[16] TSW flag is set. On start of a frame reception/transmission the counter value is captured and stored into the timestamp section of an Rx Buffer/Rx FIFO (RXTS[15-0]) or Tx Event FIFO (TXTS[15-0]) element. For more information, see Section 12.4.4.3.10, *Message RAM*.

12.4.4.3.4.1 External Timestamp Counter

For CAN FD operation mode the MCAN core requires an External Timestamp Counter. An externally generated 16-bit vector may substitute the integrated 16-bit CAN bit time counter (internal timestamp counter) for receive and transmit timestamp generation. An external 16-bit timestamp counter can be used by programming the MCAN_TSCC[1-0] TSS field.

The External Timestamp Counter uses the interface clock (ICLK) as a reference clock. The MCAN Core accepts a 16-bit timestamp. A 24-bit prescaler provides a programmable resolution for the timestamp (see MCANSS_EXT_TS_PRESCALER[23-0] PRESCALER bit field). The External Timestamp Counter counter can be enabled or disabled through the MCANSS_CTRL[6] EXT_TS_CNTR_EN bit. When disabled the counter is reset back to zero. While enabled the counter keeps incrementing. When the timestamp rolls over the MCAN timestamp interrupt is generated.

When the timestamp rolls over the MCANSS_IRS register is set (see Figure 12-252). The MCANSS_IE register can be affected by writing to the *MCANSS_IESS* register to set or to the MCANSS_IECS register to clear. The *MCANSS_IESS* register is a shadow register mapped to the same address as the MCANSS_IE register. The level interrupt is a reflection of both MCANSS_IRS and MCANSS_IE being set. The MCANSS_IES register reflects the level interrupt. When an rollover event occurs the interrupt counter is incremented. Writing to the MCANSS_ICS register to clear the MCANSS_IRS register will also decrement the interrupt counter. Writing to the MCANSS_EOI register will issue another pulse if the interrupt counter is not zero.

The rollover event can be artificially simulated by software through writing to the Interrupt Set Shadow register (*MCANSS_ISS*). The *MCANSS_ISS* register is a shadow register mapped to the same address as the MCANSS_IRS register.

Copyright © 2024 Texas Instruments Incorporated

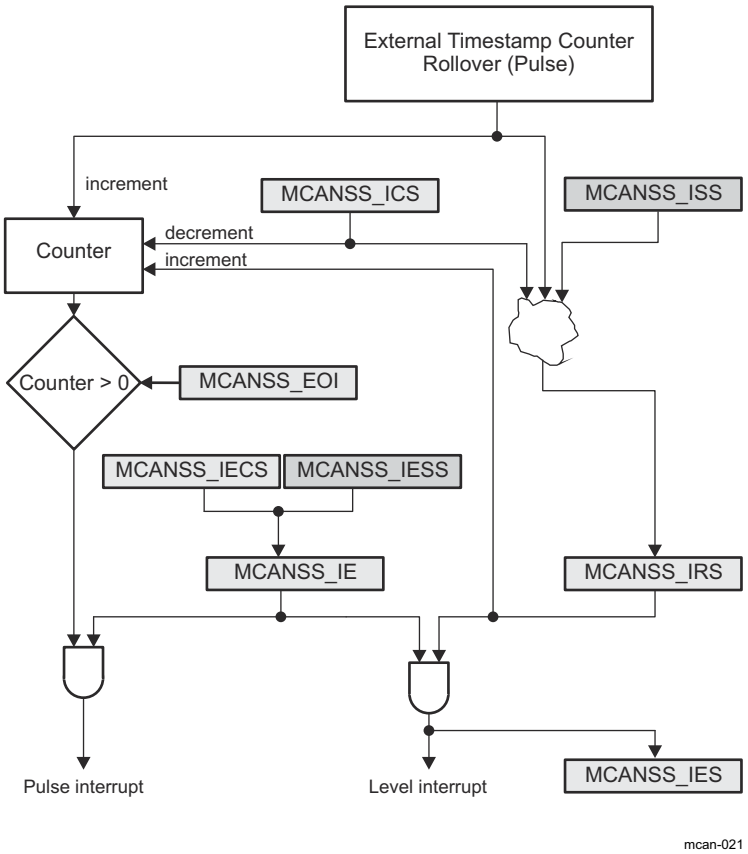


Figure 12-252. External Timestamp Counter Interrupt

12.4.4.3.5 Timeout Counter

The MCAN module has integrated a 16-bit Timeout Counter. It is used to signal timeout conditions for the Rx FIFO 0, Rx FIFO 1, and Tx Event FIFO Message RAM elements. The Timeout Counter is configured via the MCAN_TOCC register. It is enabled via the MCAN_TOCC[0] ETOC bit. The Timeout Counter operates as down-counter and uses the same prescaler programmed by the MCAN_TSCC[19-16] TCP field as the Timestamp Counter. The actual counter value can be monitored from the MCAN_TOCV[15-0] TOC field. The Timeout Counter can be started only when MCAN_CCCR[0] INIT = 0 and stopped when MCAN_CCCR[0] INIT = 1 (example: when the MCAN enters Bus_Off state). The operation mode is selected by the MCAN_TOCC[2-1] TOS field. When Continuous Mode is selected, the counter starts when MCAN_CCCR[0] INIT = 0, a write to the MCAN_TOCV register presets the counter to the value configured by the MCAN_TOCC[31-16] TOP field and continues down-counting.

In case the Timeout Counter is controlled by one of the FIFOs, an empty FIFO presets the counter to the value configured by the MCAN_TOCC[31-16] TOP field. Down-counting is started when the first FIFO element is stored. Writing to the MCAN_TOCV register has no effect. When the counter reaches zero, the interrupt MCAN_IR[18] TOO flag is set.

In Continuous Mode, the counter is immediately restarted at the value configured by the MCAN_TOCC[31-16] TOP field.

12.4.4.3.6 ECC Support

The Message Memory is wrapped in an ECC wrapper providing SECDED parity functionality. The ECC wrapper is controlled by an ECC Aggregator.

12.4.4.3.6.1 ECC Wrapper

The ECC wrapper provides Single Error Correction (SEC) and Double Error Detection (DED) parity to the Message Memory content. It has side band signals for error notification. The ECC Wrapper implements an error injection test mode.

Copyright © 2024 Texas Instruments Incorporated

The error correction is done using a lazy write back. When an error is detected, it is noted in a FIFO Queue which waits for an access gap to write the data back and refresh the memory. If a transaction writes new data to the compromised entry before the lazy write back completes, the write back is discarded.

12.4.4.3.6.2 ECC Aggregator

Note

For more information about ECC Aggregator, refer to Section 12.10.4, *ECC Aggregator*.

12.4.4.3.7 Rx Handling

The Rx Handler controls the following operations:

- Acceptance filtering
- The transfer of received messages to the Rx Buffers or to one of the two Rx FIFOs (Rx FIFO 0 or Rx FIFO 1)
- Rx FIFO Put and Get Index operations

12.4.4.3.7.1 Acceptance Filtering

The MCAN module is capable to configure two sets of acceptance filters - one set for standard and one set for extended identifiers. These filters can be assigned to an Rx Buffer or to one of the two Rx FIFOs.

The main features of the filter elements are:

- Each filter element can be configured as:
  - Range Filter (from - to)
  - Filter for specific IDs (for one or two dedicated IDs)
  - Classic Bit Mask Filter
- Each filter element can be enabled/disabled individually
- Each filter element can be configured for acceptance or rejection filtering
- Filters are checked sequentially and execution (acceptance filtering procedure) stops at the first matching filter element or when the end of the filter list is reached

Related configuration registers are:

- Global Filter Configuration (MCAN_GFC) register
- Standard ID Filter Configuration (MCAN_SIDFC) register
- Extended ID Filter Configuration (MCAN_XIDFC) register
- Extended ID AND Mask (MCAN_XIDAM) register

Depending on the configuration of the filter element (see SFEC/EFEC in Section 12.4.4.3.10, *Message RAM*) if filter matches, one of the following actions is performed:

- Received frame is stored in FIFO 0 or FIFO 1
- Received frame is stored in Rx Buffer
- Received frame is stored in Rx Buffer and generation of pulse at filter event pin is performed. This is high level single ICLK pulse. For more information, see Section 12.4.4.3.2.1, *DMA Requests*.
- Received frame is rejected
- Set High Priority Message interrupt flag MCAN_IR[8] HPM
- Set High Priority Message interrupt flag MCAN_IR[8] HPM and store received frame in FIFO 0 or FIFO 1

Acceptance filtering starts when complete Message ID is received. Acceptance filtering stops at the first matching enabled filter element or when the end of the filter list is reached. If a filter element matches - the Rx Handler starts writing the received message data in portions of 32 bit to the matching Rx Buffer or Rx FIFO. If an error condition occurs (for example: CRC error), this message is rejected with the following impact on the affected Rx Buffer or Rx FIFO:

- Rx Buffer:

Copyright © 2024 Texas Instruments Incorporated

New Data flag (MCAN_NDAT1/MCAN_NDAT2) of matching Rx Buffer is not set, but Rx Buffer (partly) overwritten with received data (for error type see MCAN_PSR[2-0] LEC respectively MCAN_PSR[10-8] DLEC fields).

- Rx FIFO:
  Put index of matching Rx FIFO is not updated, but related Rx FIFO element (partly) overwritten with received data (for error type see MCAN_PSR[2-0] LEC respectively MCAN_PSR[10-8] DLEC fields). If matching Rx FIFO is configured to operate in overwrite mode, the boundary conditions described in Section 12.4.4.3.7.2.2 have to be considered.

12.4.4.3.7.1.1 Range Filter

Each filter element can be configured to operate as Range Filter (Standard Filter Type SFT = 00/Extended Filter Type EFT = 00). The filter matches for all received message frames with IDs in the range from SFID1 to SFID2 (SFID2 ≥ SFID1) respectively in the range from EFID1 to EFID2 (EFID2 ≥ EFID1). For more information see Section 12.4.4.3.10.5, *Standard Message ID Filter Element* and Section 12.4.4.3.10.6, *Extended Message ID Filter Element*.

There are two options for range filtering of extended frames:

- Extended Filter Type EFT = 00: The Extended ID AND Mask (MCAN_XIDAM) is used for Range Filtering. The Message ID of received frames is ANDed with the Extended ID AND Mask (MCAN_XIDAM) before the range filter is applied.
- Extended Filter Type EFT = 11: The Extended ID AND Mask (MCAN_XIDAM) is not used for Range Filtering.

12.4.4.3.7.1.2 Filter for specific IDs

Each filter element can be configured to filter one or two dedicated Message IDs (Standard Filter Type SFT =01/ Extended Filter Type EFT =01). To filter only one specific Message ID, the filter element has to be configured with SFID1 = SFID2 respectively EFID1 = EFID2. For more information see Section 12.4.4.3.10.5, *Standard Message ID Filter Element* and Section 12.4.4.3.10.6, *Extended Message ID Filter Element*.

12.4.4.3.7.1.3 Classic Bit Mask Filter

Classic bit mask filtering can filter groups of Message IDs (Standard Filter Type SFT =10/Extended Filter Type EFT =10). This is done by masking single bits of a received Message ID. In this case SFID1/EFID1 element is used as Message ID filter, while SFID2/EFID2 element is used as filter mask.

A 0 bit at the filter mask (SFID2/EFID2) will mask out the corresponding bit position of the configured Message ID filter (SFID1/EFID1) and the value of the received Message ID at that bit position is not relevant for acceptance filtering. Only those bits of the received Message ID where the corresponding mask bits are 1 are relevant for acceptance filtering.

There are two interesting cases:

- All mask bits are 1: a match occurs only when the received Message ID and the configured Message ID filter are identical.
- All mask bits are 0: all Message IDs match.

12.4.4.3.7.1.4 Standard Message ID Filtering

The standard Message ID (11-bit ID) filtering flow is shown in Figure 12-253. Section 12.4.4.3.10.5, *Standard Message ID Filter Element* describes the standard Message ID filter element.

The Remote Transmission Request (RTR) and Extended Identifier (XTD) bits of the received frames are compared against the list of configured filter elements. This is controlled by the following registers:

- Global Filter Configuration (MCAN_GFC) register
- Standard ID Filter Configuration (MCAN_SIDFC) register

Copyright © 2024 Texas Instruments Incorporated

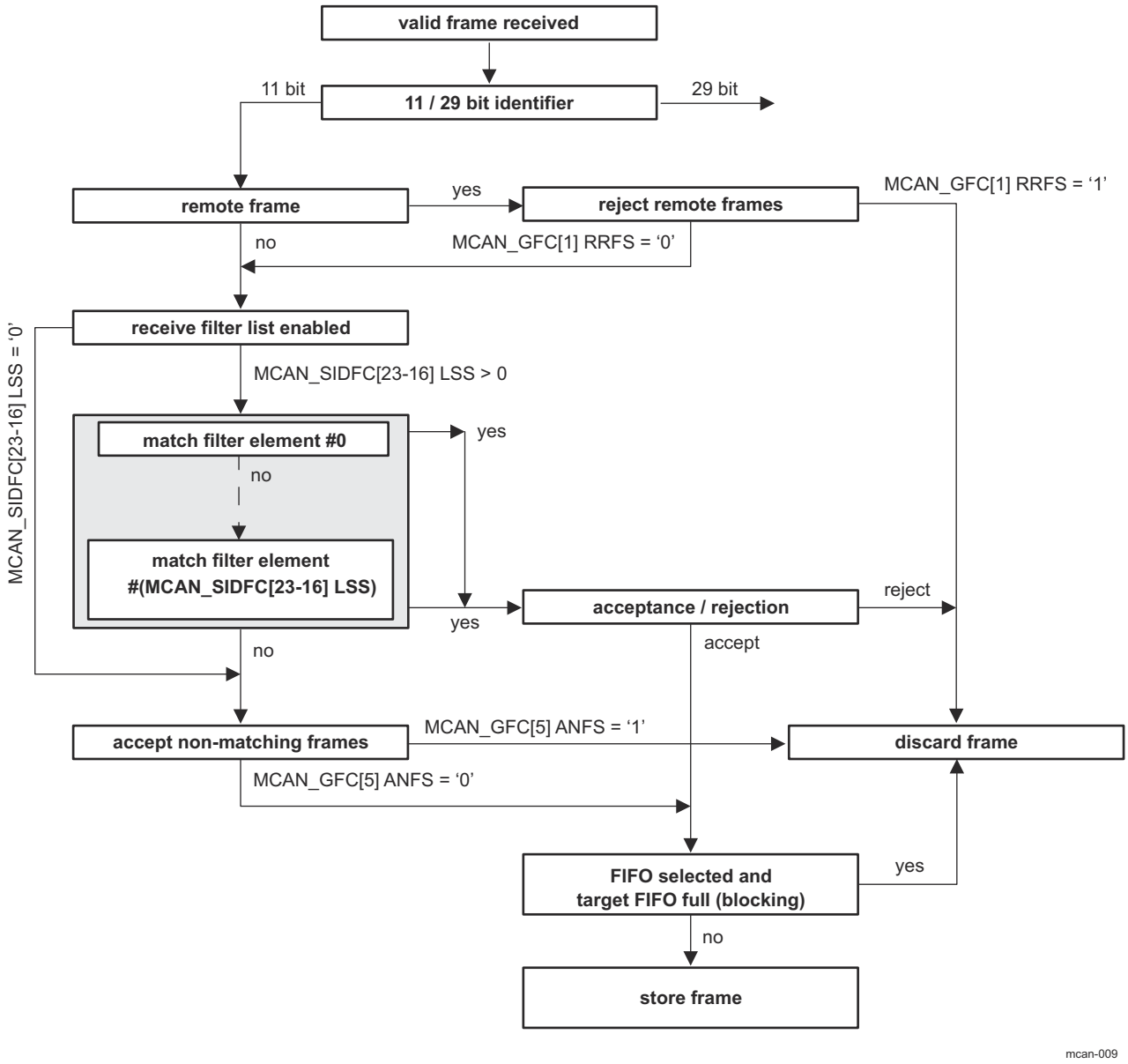


Figure 12-253. Standard Message ID Filter Path

12.4.4.3.7.1.5 Extended Message ID Filtering

The extended Message ID (29-bit ID) filtering flow is shown in Figure 12-254. Section 12.4.4.3.10.6, *Extended Message ID Filter Element* describes the extended Message ID filter element.

The Remote Transmission Request (RTR) and Extended Identifier (XTD) bits of the received frames are compared against the list of configured filter elements. This is controlled by the following registers:

- Global Filter Configuration (MCAN_GFC) register
- Extended ID Filter Configuration (MCAN_XIDFC) register

Note that before the filter list is executed the received identifier is ANDed with the Extended ID AND Mask (MCAN_XIDAM).

Copyright © 2024 Texas Instruments Incorporated

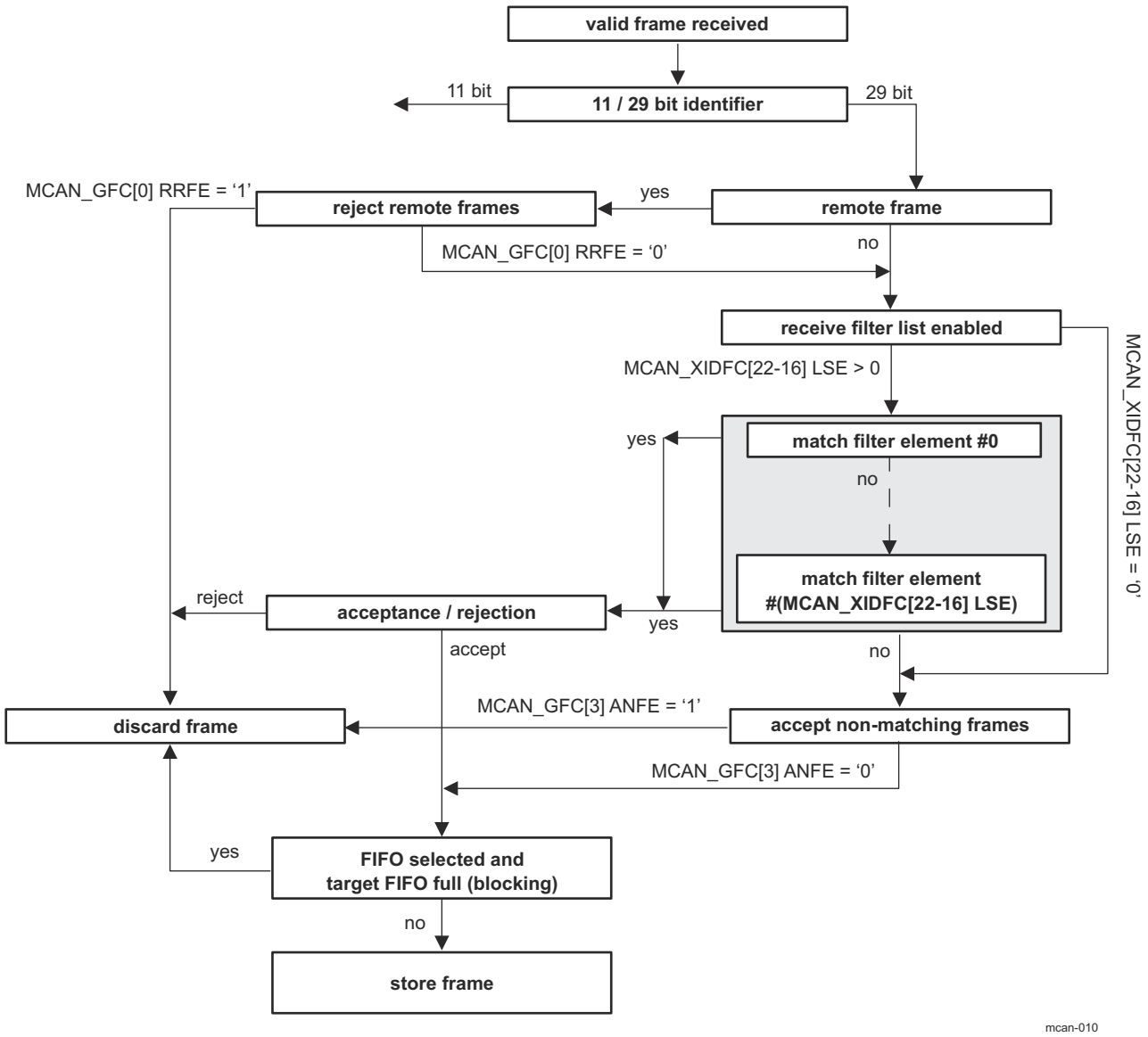


Figure 12-254. Extended Message ID Filter Path

12.4.4.3.7.2 Rx FIFOs

The configuration of the Rx FIFOs (Rx FIFO 0 and Rx FIFO 1) can be done via the MCAN_RXF0C and MCAN_RXF1C registers. Each Rx FIFO can be configured to store up to 64 received messages.

After acceptance filtering the received messages that passed are transferred to the Rx FIFO. The filter mechanisms available for the Rx FIFO 0 and Rx FIFO 1 is described in Section 12.4.4.3.7.1, *Acceptance Filtering*. Section 12.4.4.3.10.2, *Rx Buffer and FIFO Element* describes the Rx FIFO element.

The Rx FIFO watermark can be used to prevent an Rx FIFO overflow. If the Rx FIFO fill level reaches the Rx FIFO watermark configured by the MCAN_RXFnC[30-24] FnWM field (where: n = 0 or 1) an interrupt flag MCAN_IR[1] RF0W/MCAN_IR[5] RF1W is set.

When the Rx FIFO Put Index reaches the Rx FIFO Get Index (MCAN_RXFnS[21-16] FnPI = MCAN_RXFnS[13-8] FnGI) an Rx FIFO Full condition is signalled by the MCAN_RXFnS[24] FnF status bit and interrupt flag MCAN_IR[2] RF0F/MCAN_IR[6] RF1F is set. Figure 12-255 shows Rx FIFO Status. The FIFOs fill level is presented in the MCAN_RXFnS[6-0] FnFL field (the number of elements stored in Rx FIFO).

Copyright © 2024 Texas Instruments Incorporated

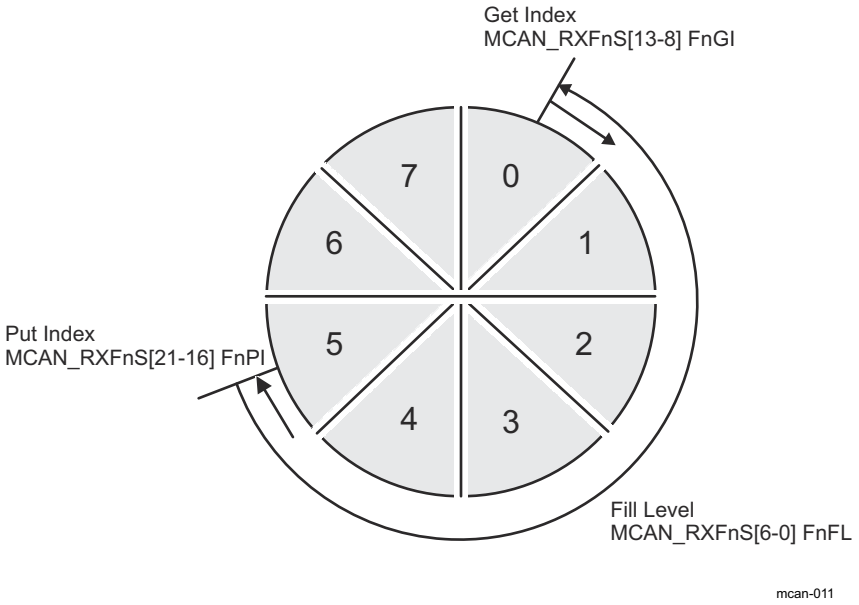


Figure 12-255. Rx FIFO Status

Rx FIFOs start address in the Message RAM (MCAN_RXFnC[15-2] FnSA field) have to be configured when reading from an Rx FIFO (Rx FIFO Get Index - MCAN_RXFnS[13-8] FnGI). Table 12-295 presents Rx Buffer/Rx FIFO Element Size for different Rx Buffer/Rx FIFO Data Field Size which is configured via the MCAN_RXESC register.

Table 12-295. Rx Buffer/Rx FIFO Element Size

MCAN_RXESC[10-8] RBDS MCAN_RXESC[2-0] F0DS/ MCAN_RXESC[6-4] F1DS	Data Field [bytes]	FIFO Element Size [RAM words]
000	8	4
001	12	5
010	16	6
011	20	7
100	24	8
101	32	10
110	48	14
111	64	18

12.4.4.3.7.2.1 Rx FIFO Blocking Mode

The Rx FIFO blocking mode is the default operation mode for the Rx FIFOs. It is configured by the MCAN_RXFnC[31] FnOM = 0.

If an Rx FIFO full condition is reached (MCAN_RXFnS[21-16] FnPI = MCAN_RXFnS[13-8] FnGI), no further messages are written to the corresponding Rx FIFO until at least one message has been read out and the Rx FIFO Get Index has been incremented. An Rx FIFO full condition is signalled by the MCAN_RXFnS[24] FnF = 1 and interrupt flag MCAN_IR[2] RF0F/MCAN_IR[6] RF1F is set.

In case a message is received while the corresponding Rx FIFO is full, this message is rejected and the message lost condition is signalled by MCAN_RXFnS[25] RFnL = 1 and interrupt flag MCAN_IR[3] RFnL/ MCAN_IR[25] RFnL is set.

12.4.4.3.7.2.2 Rx FIFO Overwrite Mode

The Rx FIFO overwrite mode is configured by the MCAN_RXFnC[31] FnOM = 1. When an Rx FIFO full condition is reached (MCAN_RXFnS[21-16] FnPI = MCAN_RXFnS[13-8] FnGI) signalled by MCAN_RXFnS[24] FnF = 1,

Copyright © 2024 Texas Instruments Incorporated

the next accepted message for the FIFO will overwrite the oldest FIFO message. Put index/Get index are both incremented by one.

In overwrite mode if an Rx FIFO full condition is signalled, reading of the Rx FIFO elements should start at least at get index + 1. The reason for that is, that it might happen, that a received message is written to the Message RAM (Put index) while the Host CPU is reading from the Message RAM (Get index). In this case inconsistent data may be read from the respective Rx FIFO element. The problem is solved by adding an offset to the Get index when reading from the Rx FIFO. The offset depends on how fast the Host CPU accesses the Rx FIFO. Figure 12-256 shows an offset of two with respect to the Get index when reading the Rx FIFO. In this case the two messages stored in element 1 and 2 are lost.

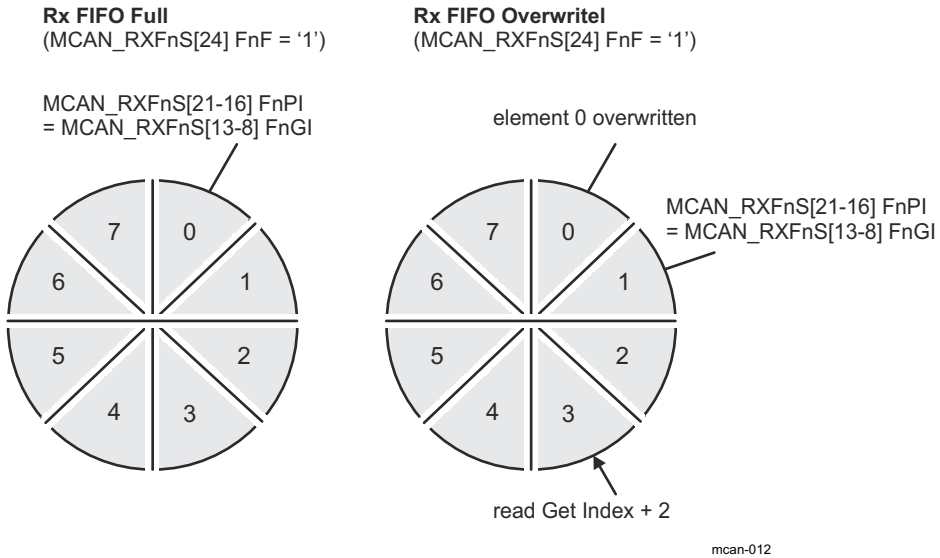


Figure 12-256. Rx FIFO Overflow Handling

After reading from the Rx FIFO, the number of the last element read has to be written to the Rx FIFO Acknowledge Index MCAN_RXFnA[5-0] FnAI. This increments the get index to that element number. In case the Put index has not been incremented to this Rx FIFO element, the Rx FIFO full condition is reset (MCAN_RXFnS[24] FnF = 0).

12.4.4.3.7.3 Dedicated Rx Buffers

The MCAN supports up to 64 dedicated Rx Buffers. The start address of the Rx Buffers section in the Message RAM is configured via MCAN_RXBC[15-2] RBSA field. To store in an Rx Buffer a Standard or Extended Message ID Filter Element with SFEC/EFEC = 111 and SFID2/EFID2[10-9] = 00 has to be configured (see Section 12.4.4.3.10.5, *Standard Message ID Filter Element* and Section 12.4.4.3.10.6, *Extended Message ID Filter Element*).

After a received message has been accepted by a filter element, the message is stored into the Rx Buffer in the Message RAM referenced by the filter element (the format is the same as for an Rx FIFO element). In addition the flag MCAN_IR[19] DRX (Message stored in Dedicated Rx Buffer) is set.

Table 12-296 shows Example Filter Configuration for Rx Buffers.

Table 12-296. Example Filter Configuration for Rx Buffers

Filter Element	SFID1[10-0] EFID1[28-0]	SFID2[10-9] EFID2[10-9]	SFID2[5-0] EFID2[5-0]
0	ID message 1	00	00 0000
1	ID message 2	00	00 0001
2	ID message 3	00	00 0010

After the last word of a matching received message has been written to the Message RAM, the respective New Data flag in register MCAN_NDAT1/MCAN_NDAT2 is set. As long as the New Data flag is set, the respective

Copyright © 2024 Texas Instruments Incorporated

Rx Buffer is locked against updates from received matching frames. The New Data flags have to be reset by the Host CPU by writing a 1 to the respective bit position.

While an Rx Buffer's New Data flag is set, a Message ID Filter Element referencing this specific Rx Buffer will not match, causing the acceptance filtering to continue. Following Message ID Filter Elements may cause the received message to be stored into another Rx Buffer, or into an Rx FIFO, or the message may be rejected, depending on filter configuration.

12.4.4.3.7.3.1 Rx Buffer Handling

Rx Buffer Handling include the following steps:

- Reset interrupt flag MCAN_IR[19] DRX
- Read New Data registers
- Read messages from Message RAM
- Reset New Data flags of processed messages

12.4.4.3.7.4 Debug on CAN Support

Debug DMA is not supported feature. Debug messages can be traced through the RX FIFO (see Section 12.4.4.3.7.2).

12.4.4.3.8 Tx Handling

The Tx Handler is used to handle the Tx requests. It controls the transfer of transmit messages from the dedicated Tx Buffers, the Tx FIFO, and the Tx Queue to the CAN Core, the Tx Event FIFO, and the Put and Get Index operations. The MCAN module supports up to 32 Tx Buffers. These Tx Buffers can be configured as dedicated Tx Buffers, Tx FIFO, or Tx Queue and as combination of dedicated Tx Buffers/Tx FIFO or dedicated Tx Buffers/Tx Queue. For each Tx Buffer element Classical CAN or CAN FD transmission mode can be configured. Section 12.4.4.3.10.3 describes the Tx Buffer Element. Table 12-297 shows the possible configurations for message transmission.

Table 12-297. Possible Configurations for Message Transmission

MCAN_CCCR		Tx Buffer Element		Frame Transmission
MCAN_CCCR[9] BRSE	MCAN_CCCR[8] FDOE	FDF	BRS	
ignored	0	ignored	ignored	Classic CAN
0	1	0	ignored	Classic CAN
0	1	1	ignored	CAN FD without bit rate switching
1	1	0	ignored	Classic CAN
1	1	1	0	CAN FD without bit rate switching
1	1	1	1	CAN FD with bit rate switching

When the Tx Buffer Request Pending MCAN_TXBRP register is updated, or when a transmission has been started the Tx Handler starts scanning to check for the highest priority pending Tx request. The Tx Buffer with lowest Message ID has highest priority.

Note

AUTOSAR requires at least three Tx Queue Buffers and support of transmit cancellation.

12.4.4.3.8.1 Transmit Pause

The transmit pause feature is intended for use in CAN networks where the CAN Message IDs are specific and cannot easily be changed. These Message IDs may have a higher priority than other defined Message IDs, while in a specific application their relative priority should be inverse. This allows for a case where one ECU sends a burst of CAN messages that cause another ECU's CAN messages to be delayed (paused).

The transmit pause feature is enabled by the MCAN_CCCR[14] TXP bit. By default this bit is disabled (MCAN_CCCR[14] TXP = 0). Each time after successfully transmitted message, a pause for two CAN bit times

Copyright © 2024 Texas Instruments Incorporated

occurs before the start of the next transmission. This allows the other CAN nodes in the network to transmit messages even if their Message IDs have lower priority.

12.4.4.3.8.2 Dedicated Tx Buffers

Dedicated Tx Buffers are intended for message transmission under complete control of the Host CPU.

There are two options:

- Each dedicated Tx Buffer is configured with a specific Message ID.
- Two or more dedicated Tx Buffers are configured with the same Message ID. In this case the Tx Buffer with the lowest buffer number is transmitted first.

After the data section has been updated, a transmission is requested by an Add Request. This is done via the MCAN_TXBAR[x]ARn bit (where x = 0 - 31). The requested messages arbitrate internally with messages from an optional Tx FIFO or Tx Queue and externally with messages on the CAN bus, and are sent out according to their Message ID.

Table 12-298 shows Tx Buffer/Tx FIFO/Tx Queue Element Size. A Dedicated Tx Buffer allocates Element Size 32-bit words in the Message RAM. The start address of a dedicated Tx Buffer in the Message RAM is calculated by adding transmit buffer index from 0 to 31 (MCAN_TXFQS[20-16] TFQPI) × Element Size to the Tx Buffer Start Address MCAN_TXBC[15-2] TBSA field.

Table 12-298. Tx Buffer/Tx FIFO/Tx Queue Element Size

MCAN_TXESC[2-0] TBDS	Data Field [bytes]	Element Size [RAM words]
000	8	4
001	12	5
010	16	6
011	20	7
100	24	8
101	32	10
110	48	14
111	64	18

12.4.4.3.8.3 Tx FIFO

Tx FIFO mode is configured by setting bit MCAN_TXBC[30] TFQM = 0. The stored in the Tx FIFO messages are transmitted starting with the message referenced by the Get Index MCAN_TXFQS[12-8] TFGI field. After each transmission the Get Index is incremented until the Tx FIFO is empty. The Tx FIFO Free Level MCAN_TXFQS[5-0] TFFL field indicates the number of the available free Tx FIFO elements. The Tx FIFO allows transmission of messages with the same Message ID from different Tx Buffers in the order these messages have been written to the Tx FIFO.

New transmit messages have to be written to the Tx FIFO starting with the Tx Buffer referenced by the Put Index MCAN_TXFQS[20-16] TFQPI field. After each Add Request (MCAN_TXBAR[x] ARn = 1) the Put Index is incemented to the next free Tx FIFO element. When the Put Index reaches the Get Index (MCAN_TXFQS[20-16] TFQPI = MCAN_TXFQS[12-8] TFGI), Tx FIFO Full condition is signalled by bit MCAN_TXFQS[21] TFQF = 1. In this case no further messages should be written to the Tx FIFO until the next message has been transmitted and the Get Index has been incremented.

The number of requested Tx buffers should not exceed the number of free Tx Buffers as indicated by the Tx FIFO Free Level MCAN_TXFQS[5-0] TFFL field.

In case a transmission request for the Tx Buffer referenced by the Get Index is cancelled, the Get Index is incremented to the next Tx Buffer with pending transmission request and the Tx FIFO Free Level MCAN_TXFQS[5-0] TFFL field is recalculated. In case transmission cancellation is applied to any other Tx Buffer - the Get Index and the FIFO Free Level remain unchanged.

A Tx FIFO element allocates Element Size 32-bit words in the Message RAM (see Table 12-298). The start address of the next available (free) Tx FIFO Buffer is calculated by adding Tx FIFO/Queue Put Index

Copyright © 2024 Texas Instruments Incorporated

MCAN_TXFQS[20-16] TFQPI (from 0 to 31) × Element Size to the Tx Buffer Start Address MCAN_TXBC[15-2] TBSA field.

12.4.4.3.8.4 Tx Queue

Tx Queue mode is configured by setting bit MCAN_TXBC[30] TFQM = 1. The stored in the Tx Queue messages are transmitted starting with the highest priority message (lowest Message ID). In case two or more Queue Buffers are configured with the same Message ID, the Queue Buffer with the lowest buffer number is transmitted first.

New transmit messages have to be written to the Tx FIFO starting with the Tx Buffer referenced by the Put Index MCAN_TXFQS[20-16] TFQPI field. Each Add Request cyclically increments the Put Index to the next free Tx Buffer. In case of Tx Queue Full condition (MCAN_TXFQS[21] TFQF = 1), the Put Index is not valid and no further message should be written to the Tx Queue until at least one of the requested messages has been sent out or a pending transmission request has been cancelled.

The application may use the MCAN_TXBRP register instead of the Put Index and may place messages to any Tx Buffer without pending transmission request.

A Tx Queue Buffer allocates Element Size 32-bit words in the Message RAM (see Table 12-298). The start address of the next available (free) Tx Queue Buffer is calculated by adding Tx FIFO/Queue Put Index MCAN_TXFQS[20-16] TFQPI (from 0 to 31) × Element Size to the Tx Buffer Start Address MCAN_TXBC[15-2] TBSA field.

12.4.4.3.8.5 Mixed Dedicated Tx Buffers/Tx FIFO

For this combination the Tx Buffers section in the Message RAM is separated in two parts:

- Dedicated Tx Buffers: the number of Dedicated Tx Buffers is configured by the MCAN_TXBC[21-16] NDTB field
- Tx FIFO: the number of Tx Buffers assigned to the Tx FIFO is configured by the MCAN_TXBC[29-24] TFQS field

If the MCAN_TXBC[29-24] TFQS field is empty (zero) - only Dedicated Tx Buffers are used.

Tx prioritization:

- Scan Dedicated Tx Buffers and oldest pending Tx FIFO Buffer (referenced by the MCAN_TXFQS[12-8] TFGI field)
- Buffer with lowest Message ID gets highest priority and is transmitted next

Figure 12-257 shows Mixed Dedicated Tx Buffers/Tx FIFO example.

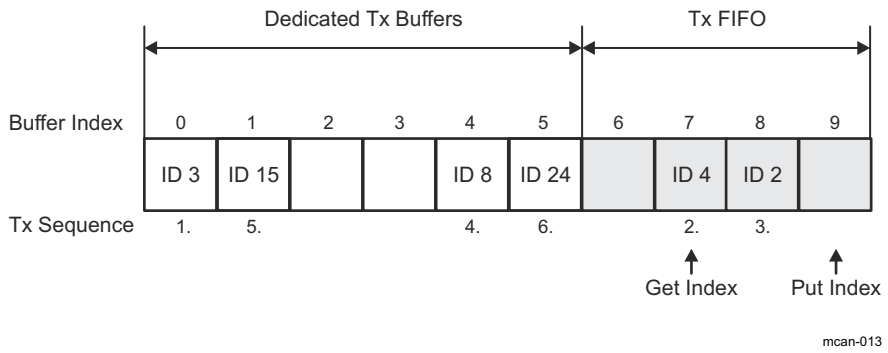


Figure 12-257. Mixed Dedicated Tx Buffers/Tx FIFO (example)

12.4.4.3.8.6 Mixed Dedicated Tx Buffers/Tx Queue

For this combination the Tx Buffers section in the Message RAM is separated in two parts:

- Dedicated Tx Buffers: the number of Dedicated Tx Buffers is configured by the MCAN_TXBC[21-16] NDTB field
- Tx Queue: the number of Tx Buffers assigned to the Tx Queue is configured by the MCAN_TXBC[29-24] TFQS field

Copyright © 2024 Texas Instruments Incorporated

If MCAN_TXBC[29-24] TFQS field is empty (zero) - only Dedicated Tx Buffers are used.

Tx prioritization:

- Scan all Tx Buffers with activated transmission request
- Tx Buffer with lowest Message ID gets highest priority and is transmitted next

Figure 12-258 shows Mixed Dedicated Tx Buffers/Tx Queue example.

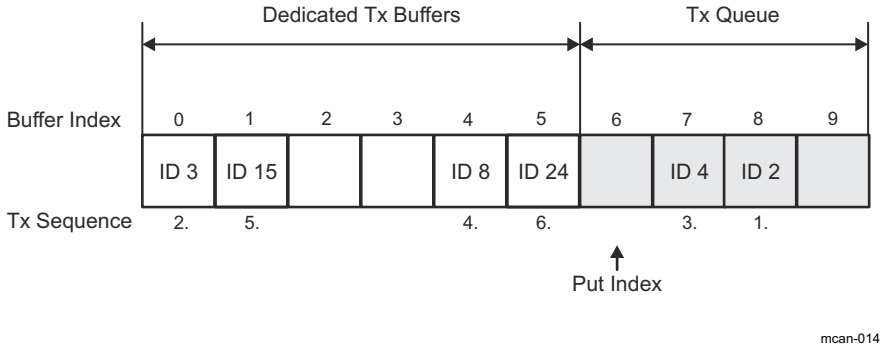


Figure 12-258. Mixed Dedicated Tx Buffers/Tx Queue (example)

12.4.4.3.8.7 Transmit Cancellation

This feature is especially intended for gateway and AUTOSAR based applications. The Host CPU can cancel a requested transmission from a dedicated Tx Buffer or a Tx Queue Buffer by setting bit MCAN_TXBCR[n] CRn = 1 (where n = 0 - 31). The corresponding bit position n is equivalent to the number of the Tx Buffer.

Transmit cancellation is not intended for Tx FIFO operation.

Successful cancellation is signalled by setting the corresponding bit of the MCAN_TXBCF register (MCAN_TXBCF[n] CFn = 1).

If transmission from a Tx Buffer is already ongoing and a transmit cancellation is requested, the corresponding MCAN_TXBRP[n] TRPn bit remains set as long as the transmission is in progress. If the transmission was successful, the corresponding MCAN_TXBTO[n] TOn and MCAN_TXBCF[n] CFn bits are set. If the transmission was not successful, only the corresponding bit MCAN_TXBCF[n] CFn = 1.

Note

If pending transmission is cancelled immediately before this transmission could have been started, a short time window occurs where no transmission is started even if another message is also pending in this node. This may enable another node to transmit a message which may have a lower priority than the second message in this node.

12.4.4.3.8.8 Tx Event Handling

To support Tx Event Handling the Message RAM has implemented a Tx Event FIFO section. Up to 32 Tx Event FIFO elements can be configured. Section 12.4.4.3.10.4 describes the Tx Event FIFO element. After message transmission on the CAN bus, Message ID and Timestamp are stored in a Tx Event FIFO element. To link a Tx Event to a Tx Event FIFO element, the Message Marker from the transmitted Tx Buffer is copied into the Tx Event FIFO element.

A Tx Event FIFO full condition is signalled by the MCAN_IR[14] TEFF bit. In this case no further elements are written to the Tx Event FIFO until at least one element has been read out and the Tx Event FIFO Get Index has been incremented (MCAN_TXEFS[12-8] EFGI). In case a Tx Event occurs while the Tx Event FIFO is full, this event is rejected and interrupt flag MCAN_IR[15] TEFL bit is set.

The Tx Event FIFO watermark can be configured to avoid a Tx Event FIFO overflow. When the Tx Event FIFO fill level reaches the Tx Event FIFO watermark configured by the MCAN_TXEFC[29-24] EFWM field, interrupt flag MCAN_IR[13] TEFW is set. When reading from the Tx Event FIFO, two times the Tx Event FIFO Get Index

Copyright © 2024 Texas Instruments Incorporated

MCAN_TXEFS[12-8] EFGI field has to be added to the Tx Event FIFO start address MCAN_TXEFC[15-2] EFSA field.

12.4.4.3.9 FIFO Acknowledge Handling

The Get Indices of the two Rx FIFOs (Rx FIFO 0 or Rx FIFO 1) and the Tx Event FIFO are controlled by writing to the corresponding FIFO Acknowledge Index (see MCAN_RXF0A, MCAN_RXF1A, and MCAN_TXEFA). Writing to the FIFO Acknowledge Index will set the FIFO Get Index to the FIFO Acknowledge Index plus one and thereby updates the FIFO Fill Level.

There are two use cases:

- A single element has been read from the FIFO: the Get Index value is written to the FIFO Acknowledge Index.
- A sequence of elements has been read from the FIFO: the Get Index value (Index of the last element read) is written to the FIFO Acknowledge Index at the end of that read sequence.

The Host CPU has free access to the Message RAM. The special care has to be taken when reading FIFO elements in an arbitrary order (Get Index not considered). This can be useful when reading a High Priority Message from one of the two Rx FIFOs. In this case the FIFO's Acknowledge Index should not be written because this would set the Get Index to a wrong position and also changes the FIFO's Fill Level. In this case some of the older FIFO elements would be lost.

Note

The application has to ensure that a valid value is written to the FIFO Acknowledge Index. The MCAN module does not check for erroneous values.

Copyright © 2024 Texas Instruments Incorporated

12.4.4.3.10 Message RAM

The MCAN module has implemented Message RAM. The main purpose of the Message RAM is to store:

- Receive Messages
- Transmit Messages
- Tx Event Elements
- Message ID Filter Elements

12.4.4.3.10.1 Message RAM Configuration

The MCAN module is configured to allocate 4352 words in the Message RAM. The Message RAM has a width of 32 bits.

The Message RAM is capable to include each of the sections listed in Figure 12-259. It is not necessary to configure each of the sections (a section in the Message RAM may be 0) and there is not restriction with respect to the sequence of the sections. For parity checking or ECC a respective number of bits has to be added to each word.

When the MCAN module addresses the Message RAM it addresses 32-bit words. The start addresses are configurable and they are 32-bit word addresses.

The element size can be configured for:

- Rx FIFO 0 via the MCAN_RXESC[2-0] F0DS field
- Rx FIFO 1 via the MCAN_RXESC[6-4] F1DS field
- Rx Buffers via the MCAN_RXESC[10-8] RBDS field
- Tx Buffers via the MCAN_TXESC[2-0] TBDS field

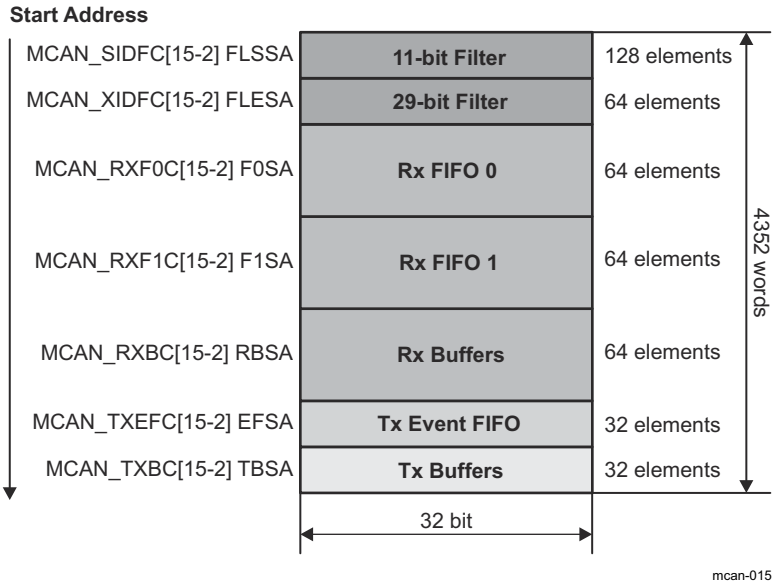


Figure 12-259. Message RAM Configuration

The Host CPU configures the following information in the Message RAM:

- Start addresses of the memory sections
- Number of elements in each section
- The size of the elements in some sections

Note

The MCAN module does not check for errors in the Message RAM configuration. The configuration of the start addresses of the different sections and the number of elements of each section has to be done carefully. This will prevent falsification or loss of data.

Copyright © 2024 Texas Instruments Incorporated

12.4.4.3.10.2 Rx Buffer and FIFO Element

Up to 64 Rx Buffers and two Rx FIFOs can be configured in the Message RAM. Each Rx FIFO section can be configured to store up to 64 received messages. The element size can be configured for storage of CAN FD messages with up to 64 bytes data field via the MCAN_RXESC register.

Figure 12-260 shows Rx Buffer/Rx FIFO element structure.

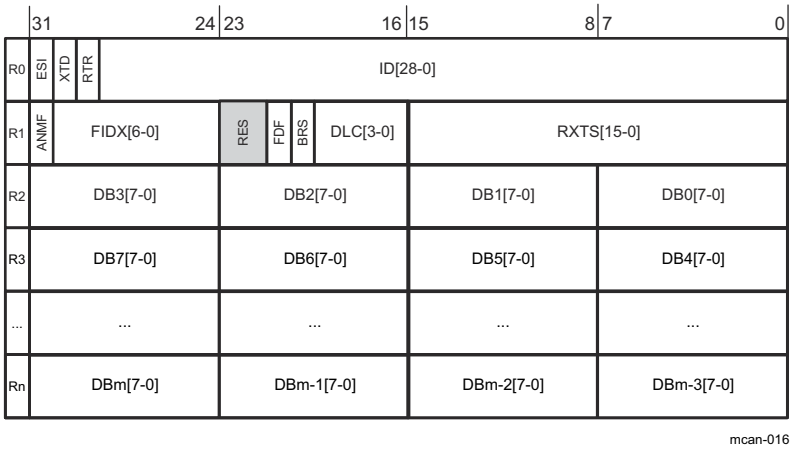


Figure 12-260. Rx Buffer/Rx FIFO Element Structure

Table 12-299 shows Rx Buffer/Rx FIFO element field descriptions.

Table 12-299. Rx Buffer/Rx FIFO Element Field Descriptions

Word	Bits	Field Name	Description
R0	31	ESI	Error State Indicator • 0h = Transmitting node is error active • 1h = Transmitting node is error passive
	30	XTD	Extended Identifier Signals to the Host CPU whether the received frame has a standard or extended identifier. • 0h = 11-bit standard identifier • 1h = 29-bit extended identifier
	29	RTR	Remote Transmission Request Signals to the Host CPU whether the received frame is a data frame or a remote frame. • 0h = Received frame is a data frame • 1h = Received frame is a remote frame **Note:** There are no remote frames in CAN FD format. In case a CAN FD frame was received (FDF = 1), RTR bit reflects the state of the reserved r1 bit (RES[23]).
	28-0	ID[28-0]	Identifier Standard or extended identifier depending on XTD bit. A standard identifier is stored into ID[28-18].

Copyright © 2024 Texas Instruments Incorporated

Table 12-299. Rx Buffer/Rx FIFO Element Field Descriptions (continued)

Word	Bits	Field Name	Description
R1	31	ANMF	Accepted Non-matching Frame Acceptance of non-matching frames may be enabled via the MCAN_GFC[5-4] ANFS and MCAN_GFC[3-2] ANFE fields. • 0h = Received frame matching filter index FIDX field • 1h = Received frame did not match any Rx filter element
	30-24	FIDX[6-0]	Filter Index 0h-7Fh (0-127): Index of matching Rx acceptance filter element (invalid if ANMF = 1). Range is 0 to MCAN_SIDFC[23-16] LSS - 1 respectively MCAN_XIDFC[22-16] LSE - 1.
	23-22	RES	Reserved
	21	FDF	FD Format • 0h = Standard frame format • 1h = CAN FD frame format (new DLC-coding and CRC)
	20	BRS	Bit Rate Switch • 0h = Frame received without bit rate switching • 1h = Frame received with bit rate switching
	19-16	DLC[3-0]	Data Length Code • 0h-8h (0-8) = CAN + CAN FD: received frame has 0-8 data bytes • 9h-Fh (9-15) = CAN: received frame has 8 data bytes • 9h-Fh (9-15) = CAN FD: received frame has 12/16/20/24/32/48/64 data bytes
	15-0	RXTS[15-0]	Rx Timestamp Timestamp Counter value captured on start of frame reception. Resolution depending on configuration of the Timestamp Counter Prescaler MCAN_TSCC[19-16] TCP.
R2	31-24	DB3[7-0]	Data Byte 3
	23-16	DB2[7-0]	Data Byte 2
	15-8	DB1[7-0]	Data Byte 1
	7-0	DB0[7-0]	Data Byte 0
R3	31-24	DB7[7-0]	Data Byte 7
	23-16	DB6[7-0]	Data Byte 6
	15-8	DB5[7-0]	Data Byte 5
	7-0	DB4[7-0]	Data Byte 4
...	...	...	...
Rn	31-24	DBm[7-0]	Data Byte m
	23-16	DBm-1[7-0]	Data Byte m-1
	15-8	DBm-2[7-0]	Data Byte m-2
	7-0	DBm-3[7-0]	Data Byte m-3

Note: Depending on the configuration of the element size (MCAN_RXESC), between two and sixteen 32-bit words (Rn = 3-17) are used for storage of a CAN message's data field.

Copyright © 2024 Texas Instruments Incorporated

12.4.4.3.10.3 Tx Buffer Element

The Tx Buffers section can be configured to hold dedicated Tx Buffers as well as a Tx FIFO/Tx Queue. In case that the Tx Buffers section is shared by dedicated Tx buffers and a Tx FIFO/Tx Queue, the dedicated Tx Buffers start at the beginning of the Tx Buffers section followed by the buffers assigned to the Tx FIFO or Tx Queue. The Tx Handler makes difference between dedicated Tx Buffers and Tx FIFO/Tx Queue via the MCAN_TXBC[29-24] TFQS and MCAN_TXBC[21-16] NDTB fields. The element size can be configured for storage of CAN FD messages with up to 64 bytes data field via the MCAN_TXESC register.

Figure 12-261 shows Tx Buffer element structure.

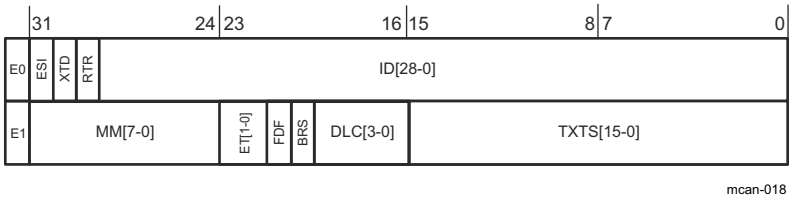


Figure 12-261. Tx Buffer Element Structure

Table 12-300 shows Tx Buffer element field descriptions.

Table 12-300. Tx Buffer Element Field Descriptions

Word	Bits	Field Name	Description
T0	31	ESI	Error State Indicator • 0h = ESI bit in CAN FD format depends only on error passive flag • 1h = ESI bit in CAN FD format transmitted recessive **Note:** The ESI bit of the transmit buffer is or'ed with the error passive flag to decide the value of the ESI bit in the transmitted CAN FD frame. As required by the CAN FD protocol specification, an error active node may optionally transmit the ESI bit recessive, but an error passive node will always transmit the ESI bit recessive.
	30	XTD	Extended Identifier • 0h = 11-bit standard identifier • 1h = 29-bit extended identifier
	29	RTR	Remote Transmission Request • 0h = Transmit data frame • 1h = Transmit remote frame **Note:** When RTR = 1, the MCAN module transmits a remote frame according to ISO11898-1:2015, even if the MCAN_CCCR[8] FDOE bit enables the transmission in CAN FD format.
	28-0	ID[28-0]	Identifier Standard or extended identifier depending on XTD bit. A standard identifier has to be written to ID[28-18].

Copyright © 2024 Texas Instruments Incorporated

Table 12-300. Tx Buffer Element Field Descriptions (continued)

Word	Bits	Field Name	Description
T1	31-24	MM[7-0]	Message Marker Written by Host CPU during Tx Buffer configuration. Copied into Tx Event FIFO element for identification of Tx message status (see also MM[7-0] field in Table 12-301).
	23	EFC	Event FIFO Control • 0h = Don't store Tx events • 1h = Store Tx events
	22	RES	Reserved
	21	FDF	FD Format • 0h = Frame transmitted in Classic CAN format • 1h = Frame transmitted in CAN FD format
	20	BRS	Bit Rate Switch • 0h = CAN FD frames transmitted without bit rate switching • 1h = CAN FD frames transmitted with bit rate switching **Note:** ESI, FDF, and BRS bits are only evaluated when CAN FD operation is enabled vie the MCAN_CCCR[8] FDOE bit. BRS bit is only evaluated when in addition the MCAN_CCCR[9] BRSE = 1.
	19-16	DLC[3-0]	Data Length Code • 0h-8h (0-8) = CAN + CAN FD: transmit frame has 0-8 data bytes • 9h-Fh (9-15) = CAN: transmit frame has 8 data bytes • 9h-Fh (9-15) = CAN FD: transmit frame has 12/16/20/24/32/48/64 data bytes
	15-0	RES	Reserved
T2	31-24	DB3[7-0]	Data Byte 3
	23-16	DB2[7-0]	Data Byte 2
	15-8	DB1[7-0]	Data Byte 1
	7-0	DB0[7-0]	Data Byte 0
T3	31-24	DB7[7-0]	Data Byte 7
	23-16	DB6[7-0]	Data Byte 6
	15-8	DB5[7-0]	Data Byte 5
	7-0	DB4[7-0]	Data Byte 4
...	...	...	...
Tn	31-24	DBm[7-0]	Data Byte m
	23-16	DBm-1[7-0]	Data Byte m-1
	15-8	DBm-2[7-0]	Data Byte m-2
	7-0	DBm-3[7-0]	Data Byte m-3

Note: Depending on the configuration of the element size (MCAN_TXESC), between two and sixteen 32-bit words (Tn = 3-17) are used for storage of a CAN message's data field.

Copyright © 2024 Texas Instruments Incorporated

12.4.4.3.10.4 Tx Event FIFO Element

Each element stores information about transmitted messages. By reading the Tx Event FIFO the Host CPU gets this information in the order the messages were transmitted. Status information about the Tx Event FIFO can be obtained from the MCAN_TXEFS register.

Figure 12-262 shows Tx Event FIFO element structure.

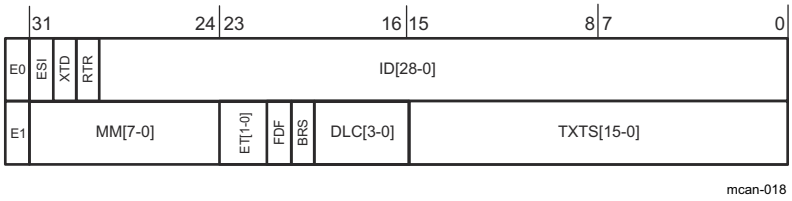


Figure 12-262. Tx Event FIFO Element Structure

Table 12-301 shows Tx Event FIFO element field descriptions.

Table 12-301. Tx Event FIFO Element Field Descriptions

Word	Bits	Field Name	Description
E0	31	ESI	Error State Indicator • 0h = Transmitting node is error active • 1h = Transmitting node is error passive
	30	XTD	Extended Identifier • 0h = 11-bit standard identifier • 1h = 29-bit extended identifier
	29	RTR	Remote Transmission Request • 0h = Data frame transmitted • 1h = Remote frame transmitted
	28-0	ID[28-0]	Identifier Standard or extended identifier depending on XTD bit. A standard identifier has to be written to ID[28-18].

Copyright © 2024 Texas Instruments Incorporated

Table 12-301. Tx Event FIFO Element Field Descriptions (continued)

Word	Bits	Field Name	Description
E1	31-24	MM[7-0]	Message Marker Copied from Tx Buffer into Tx Event FIFO element for identification of Tx message status (see also MM[7-0] field in Table 12-300).
	23-22	ET[1-0]	Event Type • 0h = Reserved • 1h = Tx event • 2h = Transmission in spite of cancellation (always set for transmissions in DAR mode) • 3h = Reserved
	21	FDF	FD Format • 0h = Standard frame format • 1h = CAN FD frame format (new DLC-coding and CRC)
	20	BRS	Bit Rate Switch • 0h = Frame transmitted without bit rate switching • 1h = Frame transmitted with bit rate switching
	19-16	DLC[3-0]	Data Length Code • 0h-8h (0-8) = CAN + CAN FD: frame with 0-8 data bytes transmitted • 9h-Fh (9-15) = CAN: frame with 8 data bytes transmitted • 9h-Fh (9-15) = CAN FD: frame with 12/16/20/24/32/48/64 data bytes transmitted
	15-0	TXTS[15-0]	Tx Timestamp Timestamp Counter value captured on start of frame transmission. Resolution depending on configuration of the Timestamp Counter Prescaler MCAN_TSCC[19-16] TCP field.

12.4.4.3.10.5 Standard Message ID Filter Element

Up to 128 filter elements can be configured for 11-bit standard IDs. When accessing a Standard Message ID Filter element, its address is the Filter List Standard Start Address MCAN_SIDFC[15-2] FLSSA field plus the index of the filter element (0-127).

Figure 12-263 shows Standard Message ID Filter element structure.

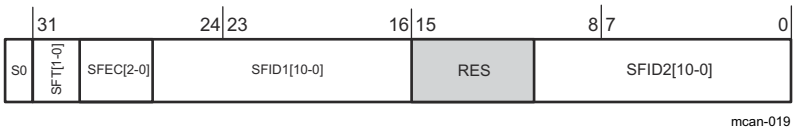


Figure 12-263. Standard Message ID Filter Element Structure

Table 12-302 shows Standard Message ID Filter element field descriptions.

Copyright © 2024 Texas Instruments Incorporated

Table 12-302. Standard Message ID Filter Element Field Descriptions

Word	Bits	Field Name	Description
S0	31-30	SFT[1-0]	Standard Filter Type • 0h = Range filter from SFID1 to SFID2 (SFID2 ≥ SFID1) • 1h = Dual ID filter for SFID1 or SFID2 • 2h = Classic filter: SFID1 = filter; SFID2 = mask • 3h = Filter element disabled **Note:** With SFT = 11 the filter element is disabled and the acceptance filtering continues (same behaviour as with SFEC = 000)
	29-27	SFEC[2-0]	Standard Filter Element Configuration All enabled filter elements are used for acceptance filtering of standard frames. Acceptance filtering stops at the first matching enabled filter element or when the end of the filter list is reached. If SFEC = 100, 101, or 110 a match sets interrupt flag MCAN_IR[8]HPM and, if enabled, an interrupt is generated. In this case the MCAN_HPMS register is updated with the status of the priority match. • 0h = Disable filter element • 1h = Store in Rx FIFO 0 if filter matches • 2h = Store in Rx FIFO 1 if filter matches • 3h = Reject ID if filter matches • 4h = Set priority if filter matches • 5h = Set priority and store in FIFO 0 if filter matches • 6h = Set priority and store in FIFO 1 if filter matches • 7h = Store into Rx Buffer , configuration of SFT[1-0] ignored
	26-16	SFID1[10-0]	Standard Filter ID 1 When filtering for Rx Buffers this field defines the ID of a standard message to be stored. The received identifiers must match exactly, no masking mechanism is used.
	15-11	RES	Reserved
	10-0	SFID2[10-0]	Standard Filter ID 2 This bit field has a different meaning depending on the configuration of SFEC: • 1) SFEC = 001 - 110 Second ID of standard ID filter element • 2) SFEC = 111 Filter for Rx Buffers
		SFID2[10-9]	This field is decides whether the received message is stored into an Rx Buffer or treated as message A, B, or C of the debug message sequence. • 0h = Store message into an Rx Buffer • 1h = Debug Message A • 2h = Debug Message B • 3h = Debug Message C **Note:** Debug feature is not supported.
		SFID2[8-6]	This field is used to control the filter event pins at the Extension Interface. A one at the respective bit position enables generation of a pulse at the related filter event pin with the duration of one MCAN_ICKL period in case the filter matches. **Note:** Only three filter event pins are supported.
		SFID2[5-0]	This field defines the offset to the Rx Buffer Start Address MCAN_RXBC[15-2] RBSA field for storage of a matching message.

Copyright © 2024 Texas Instruments Incorporated

12.4.4.3.10.6 Extended Message ID Filter Element

Up to 64 filter elements can be configured for 29-bit extended IDs. When accessing an Extended Message ID Filter element, its address is the Filter List Extended Start Address MCAN_XIDFC[15-2] FLESA field plus two times the index of the filter element (0-63).

Figure 12-264 shows Extended Message ID Filter element structure.

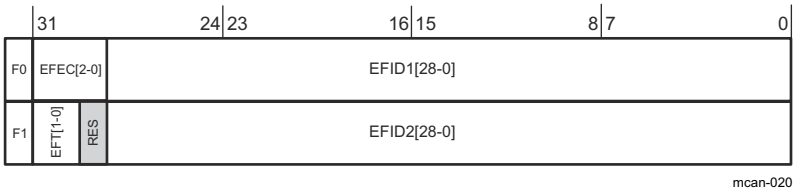


Figure 12-264. Extended Message ID Filter Element Structure

Table 12-303 shows Extended Message ID Filter element field descriptions.

Table 12-303. Extended Message ID Filter Element Field Descriptions

Word	Bits	Field Name	Description
F0	31-29	EFEC[2-0]	Extended Filter Element Configuration All enabled filter elements are used for acceptance filtering of extended frames. Acceptance filtering stops at the first matching enabled filter element or when the end of the filter list is reached. If EFEC = 100, 101, or 110 a match sets interrupt flag MCAN_IR[8]HPM and, if enabled, an interrupt is generated. In this case the MCAN_HPMS register is updated with the status of the priority match. • 0h = Disable filter element • 1h = Store in Rx FIFO 0 if filter matches • 2h = Store in Rx FIFO 1 if filter matches • 3h = Reject ID if filter matches • 4h = Set priority if filter matches • 5h = Set priority and store in FIFO 0 if filter matches • 6h = Set priority and store in FIFO 1 if filter matches • 7h = Store into Rx Buffer or as debug message, configuration of EFT[1-0] ignored
	28-0	EFID1[28-0]	Extended Filter ID 1 First ID of extended ID filter element. When filtering for Rx Buffers this field defines the ID of an extended message to be stored. The received identifiers must match exactly, only XIDAM masking mechanism (see Section 12.4.4.3.7.1.5, *Extended Message ID Filtering*) is used.

Copyright © 2024 Texas Instruments Incorporated

Table 12-303. Extended Message ID Filter Element Field Descriptions (continued)

Word	Bits	Field Name	Description
F1	31-30	EFT[1-0]	Extended Filter Type • 0h = Range filter from EFID1 to EFID2 (EFID2 ≥ EFID1) • 1h = Dual ID filter for EFID1 or EFID2 • 2h = Classic filter: EFID1 = filter, EFID2 = mask • 3h = Range filter from EFID1 to EFID2 (EFID2 ≥ EFID1), XIDAM mask not applied
	29	RES	Reserved
	28-0	EFID2[28-0]	Extended Filter ID 2 This bit field has a different meaning depending on the configuration of EFEC: • 1) EFEC = 001 - 110 Second ID of extended ID filter element • 2) EFEC = 111 Filter for Rx Buffers
		EFID2[10-9]	This field decides whether the received message is stored into an Rx Buffer or treated as message A, B, or C of the debug message sequence. • 0h = Store message into an Rx Buffer • 1h = Debug Message A • 2h = Debug Message B • 3h = Debug Message C **Note:** Debug feature is not supported.
		EFID2[8-6]	This field is used to control the filter event pins at the Extension Interface. A one at the respective bit position enables generation of a pulse at the related filter event pin with the duration of one MCAN_ICKL period in case the filter matches. **Note:** Only three filter event pins are supported.
		EFID2[5-0]	This field defines the offset to the Rx Buffer Start Address MCAN_RXBC[15-2] RBSA field for storage of a matching message.

Copyright © 2024 Texas Instruments Incorporated

12.5 Audio Interfaces

Copyright © 2024 Texas Instruments Incorporated

12.5.1 Audio Tracking Logic (ATL)

This chapter describes the integration of the Audio Tracking Logic (ATL) subsystem in the device.

12.5.1.1 ATL Overview

The Audio Tracking Logic (ATL) is used by HD Radio™ applications to synchronize the digital audio output to the baseband clock. This same IP can also be used generically to track errors between two reference signals (such as frame syncs) and generate a modulated clock output (using software-controlled cycle stealing) which averages to some desired frequency. This process can be used as a hardware assist for asynchronous sample rate conversion algorithms.

12.5.1.1.1 ATL Features Overview

The ATL module supports the following features:

- One ATL module, containing four ATL instances, for HD Radio support and asynchronous sample rate conversion assistance
- Each instance tracks the time error between two syncs (local audio word select [AWS] and baseband word select [BWS])
- Each instance generates modulated ATCLK clock signals with software-initiated pulse stealing. The intention is to use these clocks to derive audio output bit clock on MCASP
- Selection between ATL_VCLK clock or functional ATL_PCLK to run error counting timers and to derive modulated ATCLK clock outputs.
- Clock and reset management: Receives clock and reset signals from the device PSC module. The ATL module receives hardware reset.
- Power management: The ATL belongs to the PD0 power domain.
- Local software reset

Unsupported Features:

- See the Module Integration section of this document for a list of module features not supported by the integration on this Device.

12.5.1.1.2 ATL Ports

Table 12-304. ATL Clocks and Resets

Clocks	
Module Clock Input	**Description**
ATL_VCLK	ATL interface clock
ATL_PCLK	ATL functional clock.
Resets	
Module Reset Input	**Description**
ATL_RST	ATL reset

Copyright © 2024 Texas Instruments Incorporated

12.5.2 Multichannel Audio Serial Port (MCASP)

This section describes the Multichannel Audio Serial Port (MCASP).

12.5.2.1 MCASP Overview

This section introduces the Multichannel Audio Serial Port (MCASP) module and describes its main functions and connections in the device.

The MCASP functions as a general-purpose audio serial port are optimized to the requirements of various audio applications. The MCASP module can operate in both transmit and receive modes. The MCASP is useful for time-division multiplexed (TDM) stream, Inter-IC Sound (I2S) protocols reception and transmission as well as for an inter-component digital audio interface transmission (DIT). The MCASP has the flexibility to gluelessly connect to a Sony/Philips digital interface (S/PDIF) transmit physical layer component.

Although inter-component digital audio interface reception (DIR) mode (this is, S/PDIF stream receiving) is not natively supported by the MCASP module, a specific TDM mode implementation for the MCASP receivers allows an easy connection to external DIR components (for example, S/PDIF to I2S format converters).

12.5.2.1.1 MCASP Features

Each MCASP module includes the following main features:

- Independent serializer for each AXRx channel of each MCASP module
- Clock stop request/acknowledge protocol
- A single 32-bit buffer per serializer for transmit and receive operations
- Interconnect interface port for CBASS0
- Two independent clock generator modules for transmit and receive (clocking flexibility allows the MCASP to receive and transmit at different rates. For example, the MCASP can receive data at 48 kHz, but output up-sampled data at 96 kHz or 192 kHz)
- Each MCASP module functional clock can be generated:
  - Internally (master mode)
  - Supplied over MCASP serial interface (slave mode)
  - Has a controllable functional clock divide ratio
- Independent transmit and receive modules, each includes:
  - Programmable clock and frame sync generator
  - TDM streams from 2 to 32, and 384 time slots
  - Support for time slot sizes of 8, 12, 16, 20, 24, 28, and 32 bits
  - Data formatter for bit manipulation
- Glueless connection to audio Analog-to-Digital Converters (ADC), Digital-to-Analog Converters (DAC), codec, digital audio interface receiver (DIR), and S/PDIF transmit physical layer components
- Wide variety of I2S and similar bit-stream format
- Integrated digital audio interface transmitter (DIT):
  - S/PDIF, IEC60958-1, AES-3 formats
  - Enhanced channel status/user data RAM
- 384-slot TDM with external digital audio interface receiver (DIR) device
  - For DIR reception, an external DIR receiver integrated circuit should be used with I2S output format and connected to the MCASP receive section
- Support for 2 × DMA requests (one per direction):
  - 1 level-sensitive transmit direct memory access (DMA) request common for all of the MCASP serializers
  - 1 level-sensitive receive direct memory access (DMA) request common for all of the MCASP serializers
  - All transmit DMA requests are mapped to the device DMA controllers
- One transmit interrupt request common for all serializers
- One receive interrupt request common for all serializers
- Each of the Rx and Tx interrupts is propagated to different host processors via the device Interrupts

Copyright © 2024 Texas Instruments Incorporated

Note

Because a serializer receive and transmit channels data is shared on the same MCASP data pin, user can choose to have either Tx or Rx function from a serializer, not both at the same time.

Unsupported Features:

- See the Module Integration section of this document for a list of module features not supported by the integration on this Device.

12.5.2.1.2 MCASP Ports

Table 12-305. MCASP Clocks and Resets

Clocks	
Module Clock Input	**Description**
MCASP_ICLK	MCASP interface clock
MCASP_AUX_CLK	MCASP functional clock (Master Mode).
Resets	
Module Reset Input	**Description**
MCASP_RST	MCASP reset

Table 12-306. MCASP Hardware Requests

Interrupt Requests		
Module Interrupt Signal	**Description**	**Type**
MCASP_REC_INTR_PEND_0	MCASP receive interrupt request	Level
MCASP_XMIT_INTR_PEND_0	MCASP transmit interrupt request	Level
DMA Events		
Module DMA Event	**Description**	**Type**
MCASP_REC_DMA_EVT	MCASP receive request line	Pulse
MCASP_XMIT_DMA_EVT	MCASP transmit request line	Pulse

12.5.2.2 MCASP Environment

This section describes the MCASP application fields from an environment point of view (external connections).

Table 12-307 describes the MCASP I/O signals.

Table 12-307. MCASP I/O Signals

Module Pin	I/O[(1)]	Description
AXR0	I/O	Audio transmit/receive data - channel 0
AXR1	I/O	Audio transmit/receive data - channel 1
AXR2	I/O	Audio transmit/receive data - channel 2
AXR3	I/O	Audio transmit/receive data - channel 3
AXR4	I/O	Audio transmit/receive data - channel 4
AXR5	I/O	Audio transmit/receive data - channel 5
AXR6	I/O	Audio transmit/receive data - channel 6
AXR7	I/O	Audio transmit/receive data - channel 7
AXR8	I/O	Audio transmit/receive data - channel 8
AXR9	I/O	Audio transmit/receive data - channel 9
AXR10	I/O	Audio transmit/receive data - channel 10
AXR11	I/O	Audio transmit/receive data - channel 11
AXR12	I/O	Audio transmit/receive data - channel 12
AXR13	I/O	Audio transmit/receive data - channel 13
AXR14	I/O	Audio transmit/receive data - channel 14
AXR15	I/O	Audio transmit/receive data - channel 15

Copyright © 2024 Texas Instruments Incorporated

Table 12-307. MCASP I/O Signals (continued)

Module Pin	I/O[1]	Description
ACLKX	I/O	Transmit bit clock
AFSX	I/O	Transmit frame synchronization
ACLKR	I/O	Receive bit clock
AFSR	I/O	Receive frame synchronization
MCASP0_AHCLKX_I/O	I/O	Transmit high-frequency master clock. See *MCASP Integration*
MCASP0_AHCLKR_I/O	I/O	Receive high-frequency master clock. See *MCASP Integration*
AXR0	I/O	Audio transmit/receive data - channel 0
AXR1	I/O	Audio transmit/receive data - channel 1
AXR2	I/O	Audio transmit/receive data - channel 2
AXR3	I/O	Audio transmit/receive data - channel 3
AXR4	I/O	Audio transmit/receive data - channel 4
ACLKX	I/O	Transmit bit clock
AFSX	I/O	Transmit frame synchronization
ACLKR	I/O	Receive bit clock
AFSR	I/O	Receive frame synchronization
MCASP1_AHCLKX_I/O	I/O	Transmit high-frequency master clock. See *MCASP Integration*
MCASP1_AHCLKR_I/O	I/O	Receive high-frequency master clock. See *MCASP Integration*
AXR0	I/O	Audio transmit/receive data - channel 0
AXR1	I/O	Audio transmit/receive data - channel 1
AXR2	I/O	Audio transmit/receive data - channel 2
AXR3	I/O	Audio transmit/receive data - channel 3
AXR4	I/O	Audio transmit/receive data - channel 4
ACLKX	I/O	Transmit bit clock
AFSX	I/O	Transmit frame synchronization
ACLKR	I/O	Receive bit clock
AFSR	I/O	Receive frame synchronization
MCASP2_AHCLKX_I/O	I/O	Transmit high-frequency master clock. See *MCASP Integration*
MCASP2_AHCLKR_I/O	I/O	Receive high-frequency master clock. See *MCASP Integration*

(1) I = Input; O = Output; I/O = Bidirectional

Copyright © 2024 Texas Instruments Incorporated

12.5.2.3 MCASP Functional Description

In the text throughout this section a single instance of MCASP is described assuming that all modules are functionally identical. For module availability and integration differences, see *MCASP Environment* and *MCASP Integration*.

12.5.2.3.1 MCASP Block Diagram

Figure 12-265 shows the major blocks of the MCASP module. The MCASP0 has 16 serializers, the MCASP1 has 12 serializers, the MCASP2 has 6 serializers. The serializers share a clock and frame-sync generator, format unit, and error-checking logic independently for the receive and transmit part.

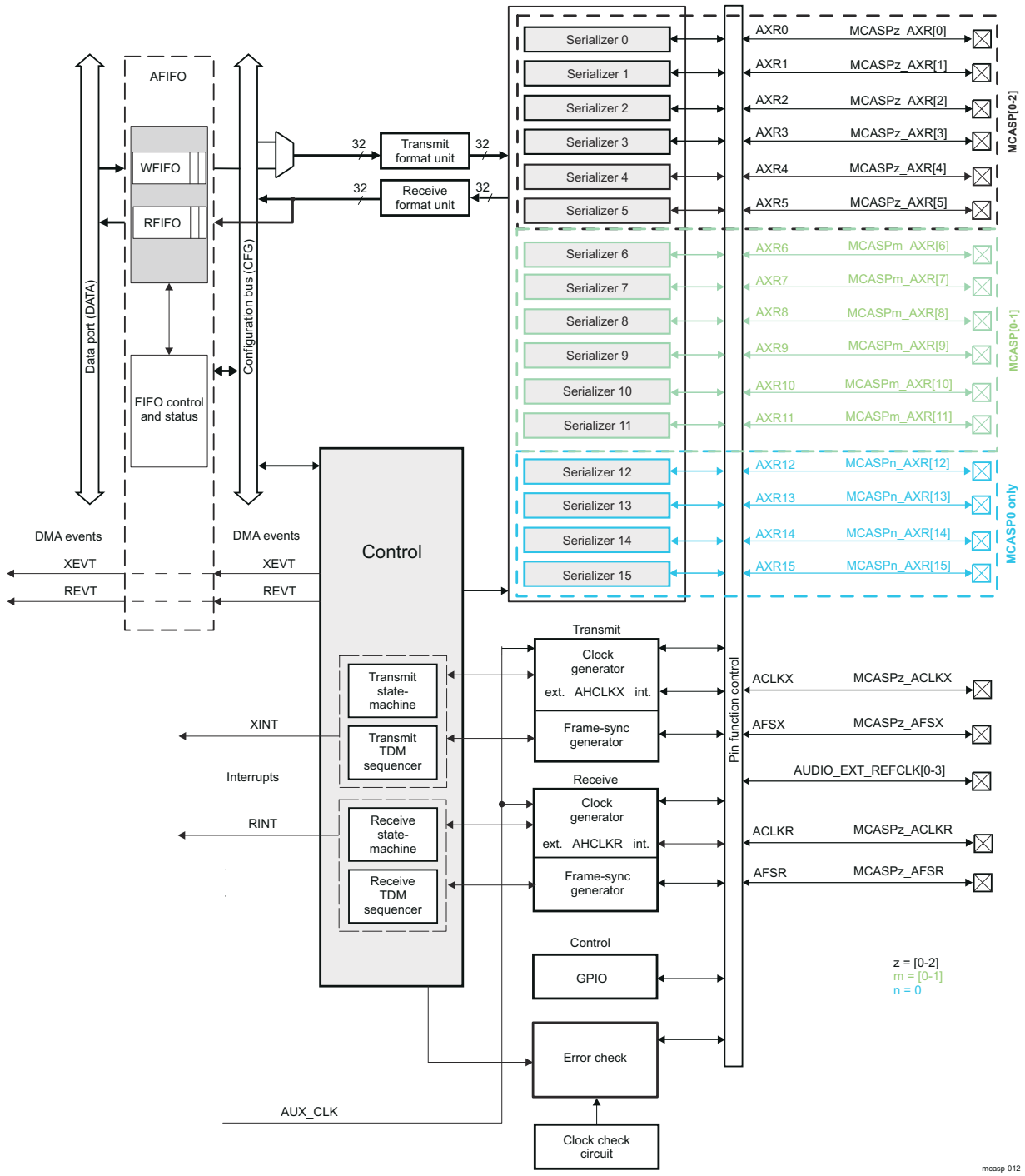


Figure 12-265. MCASP Module Block Diagram

Copyright © 2024 Texas Instruments Incorporated

Note

The internal and external clocks mentioned in this section are with respect to clock and frame-sync generator modules.

12.5.2.3.2 MCASP Clock and Frame-Sync Configurations

There are three scenarios to provide clock source signals for the Tx part and four scenarios for the Rx part of the MCASP serializers. The first three scenarios are identical between the Tx and Rx part of the MCASP. They feature an asynchronous operation between receiver and transmitter channels using independent Tx/Rx bit rate clock sources (either internal or external).

In the first scenario, the transmit - XCLK and receive - RCLK serial clocks (clock at the bit rate) are generated internally by passing through a couple of clock dividers off the internal functional clock source (AUXCLK). In this case, the bit rate clock is generated internally and is driven out on the pin ACLKX for the Tx part and pin ACLKR for the Rx part, respectively. An internally generated high-frequency clock can be optionally driven out onto the AHCLKX pin for the Tx part to serve as a reference clock for other components in the system.

In the second scenario, an external for the device clock, is passed on the ACLKX (for the TX part) and ACLKR (for the RX part) pins which are configured as inputs. In this case the Rx- /Tx- high-speed clock logic is bypassed for the XCLK/RCLK generation.

In the third (mixed) scenario, an externally driven (master) high-frequency clock is applied on the AHCLKX (for the TX part) pin, which is configured as input. In this case the AHCLKX clock frequency can be divided down via programming the ACLKX associated divider to produce the necessary bit rate clock. The high-speed clock divider can NOT be used.

In the fourth clock generation scenario the bit rate clock for MCASP receivers - RCLK is derived from the bit rate clock of the MCASP transmitters - XCLK for a synchronous operation between transmitters and receivers. Hence, the whole receiver clock generator logic is bypassed.

A typical role of the MCASP frame sync signal is to carry the left/right clock (LRCLK) signal when transmitting and receiving stereo data.

For an asynchronous operation, the AFSX (Tx part) and AFSR (Rx part) frame synchronization signals can be sourced internally or delivered externally independently for the Tx and Rx channels. During synchronous operation the receive frame sync - AFSR signal is derived from the transmit frame sync - AFSX signal. A synchronous and asynchronous mode applies to bit rate clock and frame sync signals at the same time.

12.5.2.3.2.1 MCASP Transmit Clock

The transmit high-speed and transmit clock configuration is controlled by the following registers:

- MCASP_ACLKXCTL
- MCASP_AHCLKXCTL

In case, the transmit bit clock, ACLKX, is generated internally, the MCASP_ACLKXCTL[5] CLKXM bit must be set to 1. Thus, the clock is divided down by a programmable bit clock divider (the MCASP_ACLKXCTL[4-0] CLKXDIV bit field) from the source signal.

If the transmit high-frequency master clock, AHCLKX, is also sourced internally (that is first scenario described in Section 12.5.2.3.2, the MCASP_AHCLKXCTL[15] HCLKXM bit must be set to 1. Thus, the clock is divided down by a programmable high-clock divider (the MCASP_AHCLKXCTL[11-0] HCLKXDIV bit field) from the MCASP internal clock source AUXCLK.

Internally, the MCASP always shifts transmit data at the rising edge of the internal transmit clock - XCLK, (see Figure 12-266). The CLKXP mux determines if ACLKX needs to be inverted to become XCLK. If MCASP_ACLKXCTL[7] CLKXP = 0, the CLKXP mux directly passes ACLKX signal to XCLK. As a result, the MCASP shifts transmit data at the rising edge of ACLKX. If MCASP_ACLKXCTL[7] CLKXP = 1, the CLKX mux passes the inverted version of ACLKX to XCLK. As a result, the MCASP shifts transmit data at the falling edge of ACLKX.

Copyright © 2024 Texas Instruments Incorporated

It can be seen in Figure 12-266 that XCLK is propagated to the Rx clock logic, to allow an internally synchronous operation between MCASP transmitters and receivers. This is used for example in the MCASP loopback mode.

Note

The polarity of ACLKX can be controlled in the MCASP_ACLKXCTL[7] CLKXP bit, regardless of ACLKX signal being internally or externally sourced.

In addition, there is an option to invert polarity of the AHCLKX master high speed clock via writing the MCASP_AHCLKXCTL[14] HCLKXP bit.

Note

In a similar way, the polarity of AHCLKX clock can be controlled in the MCASP_AHCLKXCTL[14] HCLKXP bit, regardless of the AHCLKX signal being internally or externally sourced.

Figure 12-266 presents the block diagram of the transmit clock generator.

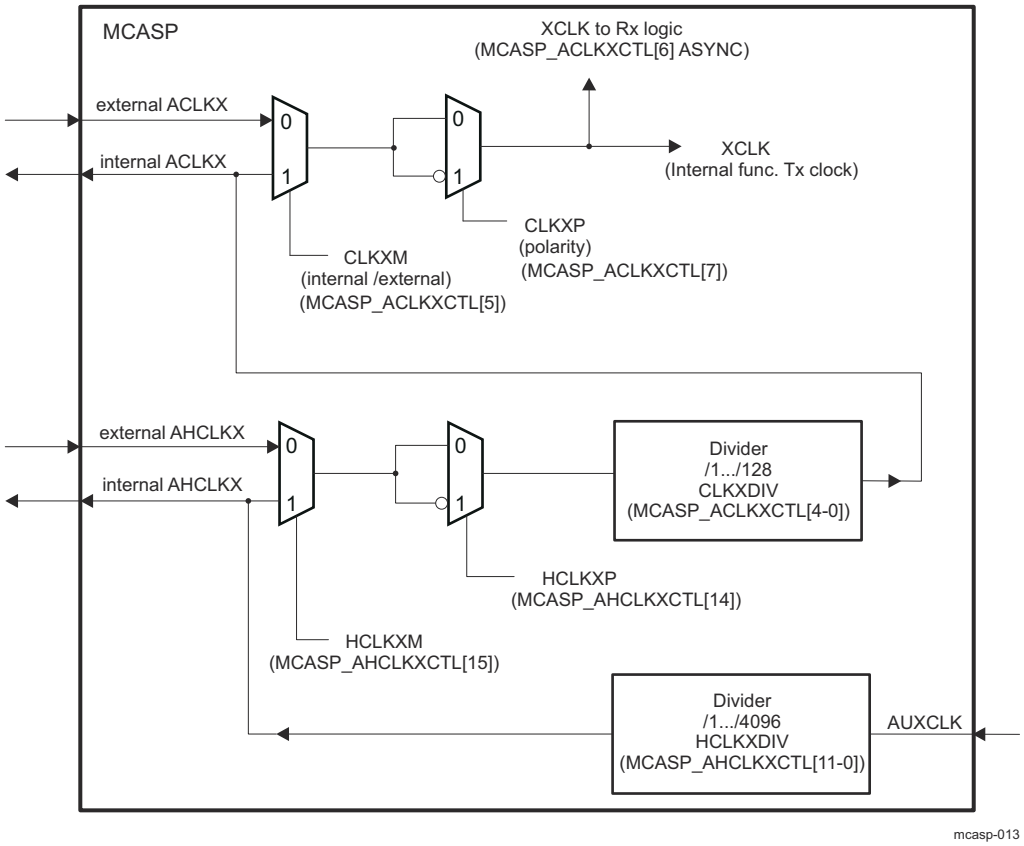


Figure 12-266. Transmit Clock Generator Block Diagram

Note

In this device:

- ACLKX is mapped on the device ball ACLKX
- internal AHCLKX is mapped on the device balls AUDIO_EXT_REFCLK[0-1]
- external AHCLKX is mapped on MCASPi_AHCLKX clock from the Device Configuration

For more information about MCASP integration, see *MCASP Environment* and *MCASP Integration*.

Copyright © 2024 Texas Instruments Incorporated

12.5.2.3.2.2 MCASP Receive Clock

The MCASP receive clock generator is built on a very similar to the transmit clock generator (but independent) circuit.

The receive clock configuration is controlled by the following registers:

- MCASP_ACLKRCTL
- MCASP_AHCLKRCTL

In case, the receive bit clock, ACLKR, is generated internally (but asynchronously to XCLK), the MCASP_ACLKRCTL[5] CLKRM bit must be set to 1. Thus, the clock is divided down by a programmable bit clock divider (the MCASP_ACLKRCTL[4-0] CLKRDIV bit field) from the source signal.

If the receive high-frequency master clock, AHCLKR, is also sourced internally (that is, first scenario described in Section 12.5.2.3.2) and the MCASP_AHCLKRCTL[15] HCLKRM bit must be set to 1. Thus, the clock is divided down by a programmable high-clock divider (the MCASP_AHCLKRCTL[11-0] HCLKRDIV bit field) from the MCASP internal clock source AUXCLK.

Note

The polarity of ACLKR can be controlled in the MCASP_ACLKRCTL[7] CLKRP bit, regardless of ACLKR signal being internally or externally sourced.

In a similar way, the polarity of AHCLKR clock can be controlled in the MCASP_AHCLKRCTL[14] HCLKRP bit, regardless of the AHCLKR signal being internally or externally sourced.

There is an option for the MCASP receiver to be configured to operate synchronously to the ACLKX and AFSX signals. The XCLK output of the Tx Clock generator (see Figure 12-266 and Figure 12-267) becomes source of the receive clock (RCLK output), when the MCASP_ACLKXCTL[6] ASYNC bit in the transmit clock control register is set to '0b0'. For more information, refer to Section 12.5.2.3.2.4.

Figure 12-267 presents the block diagram of the receive clock generator.

Copyright © 2024 Texas Instruments Incorporated

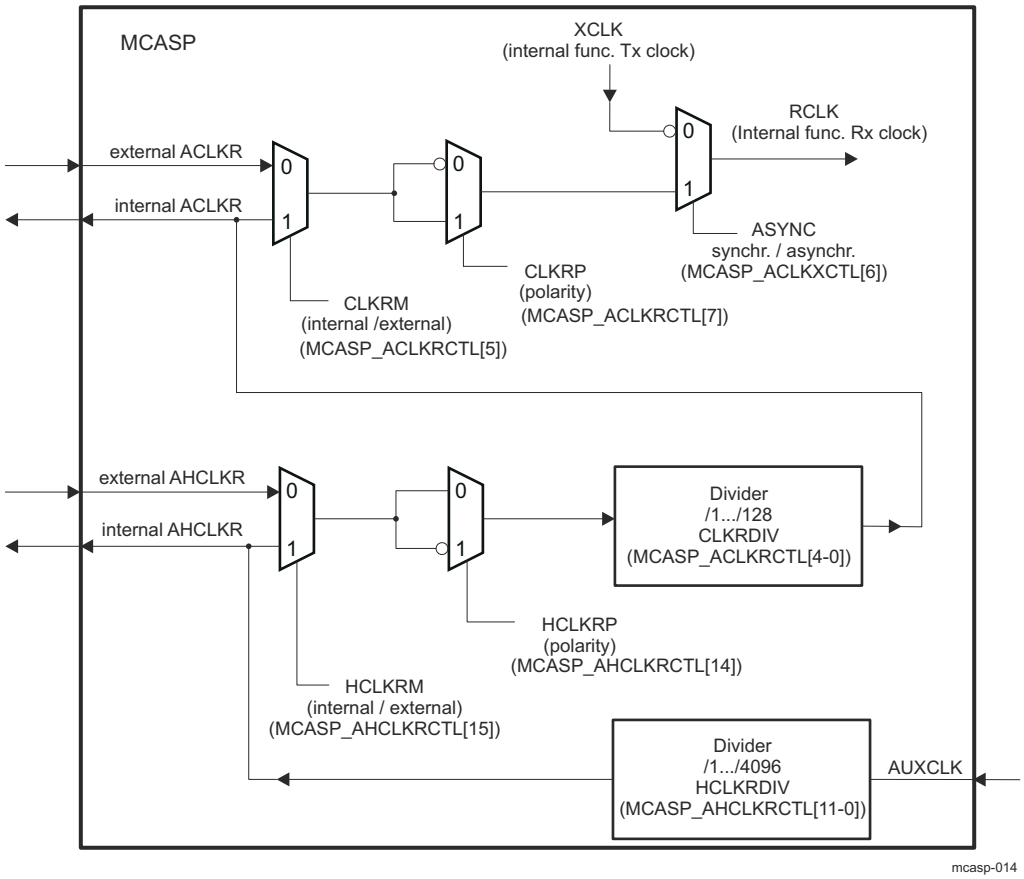


Figure 12-267. Receive Clock Generator Block Diagram

Note

In this device:

- ACLKR is mapped on the device ball ACLKR
- internal AHCLKX is mapped on the device balls AUDIO_EXT_REFCLK[0-1]
- external AHCLKR is mapped on Device Configuration MCASPi_AHCLKR from Device Configuration

12.5.2.3.2.3 Frame-Sync Generator

There are two different modes for frame sync: burst and TDM. The MCASP frame sync generator logic is illustrated in Figure 12-268.The I/O buffers are not part of the MCASP module, and are not shown in the figure.

Copyright © 2024 Texas Instruments Incorporated

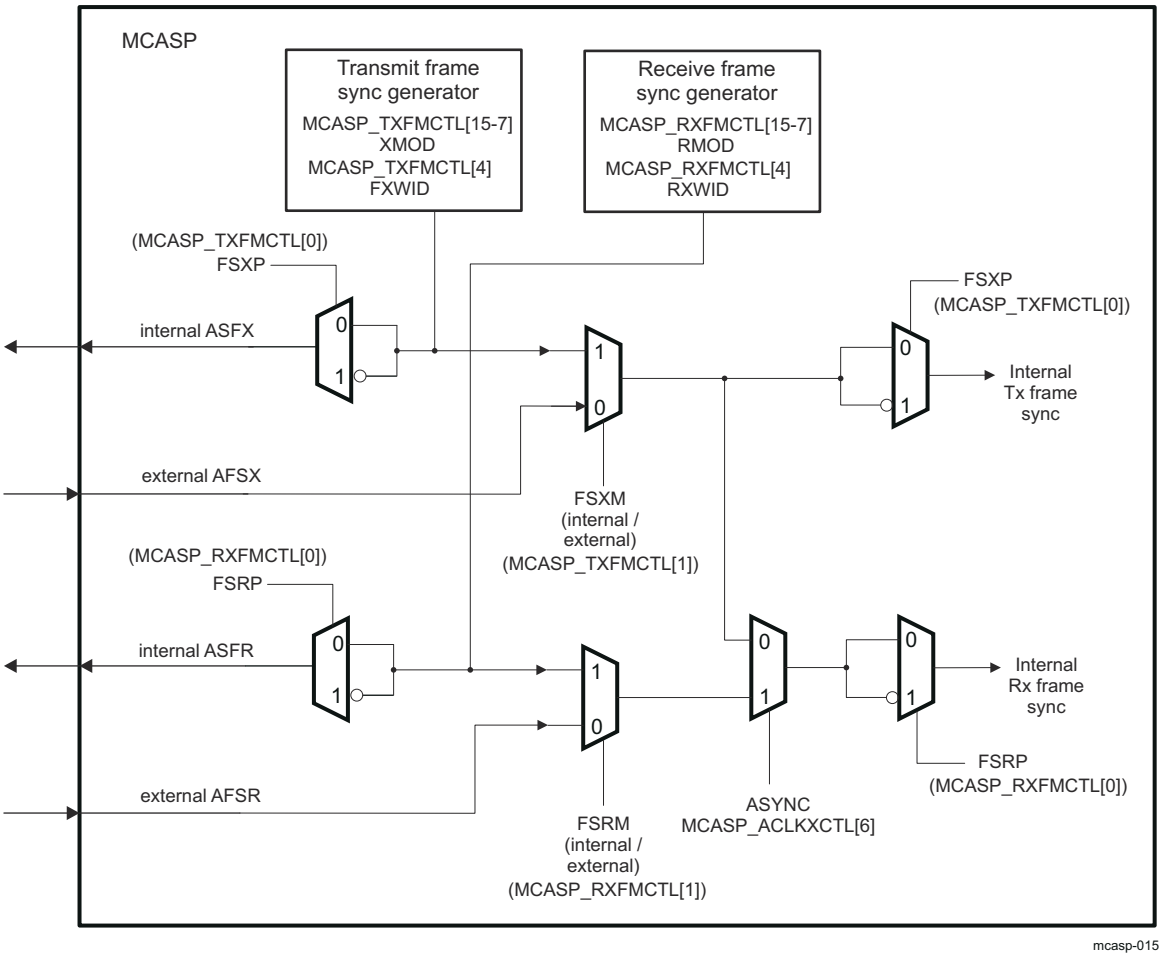


Figure 12-268. Frame Sync Generator Block Diagram

For the transmit logic, following frame-sync generator configurations can be selected:

- Internally/externally generated frame-sync via configuring MCASP_AFSXCTL[1] FSXM bit
- Frame-sync polarity: Rising edge or falling edge via configuring MCASP_AFSXCTL[0] FSXP bit
- Frame-sync width: "single bit" or "single word" via configuring MCASP_AFSXCTL[4] FXWID bit
- Frame sync mode - the appropriate frame sync generation pattern for the selected transfer mode is defined in the MCASP_AFSXCTL[15-7] XMOD bit field, as follows:
  - For DIT mode (384 slots) - MCASP_AFSXCTL[15-7] XMOD = 0x180
  - For I2S mode (2 TDM slots) - MCASP_AFSXCTL[15-7] XMOD = 0x2
  - For TDM mode (from 3 to 32 TDM slots) - MCASP_AFSXCTL[15-7] XMOD bit field set in range 0x3 - 0x20
- Bit delay: 0, 1, or 2 cycles before the first data bit. This delay is defined in MCASP_XFMT[17-16] XDATDLY bit field

For the receive logic, following frame-sync generator configurations can be selected:

- Internally/externally generated frame-sync via configuring MCASP_AFSRCTL[1] FSRM bit
- Frame-sync polarity: Rising edge or falling edge via configuring MCASP_AFSRCTL[0] FSRP bit
- Frame-sync width: "single bit" or "single word" via configuring MCASP_AFSRCTL[4] FRWID bit
- Frame sync mode - the appropriate frame sync generation pattern for the selected transfer mode is defined in the MCASP_AFSRCTL[15-7] RMOD bit field, as follows:
  - For I2S mode (2 TDM slots) - MCASP_AFSRCTL[15-7] RMOD = 0x2
  - For TDM mode (from 3 to 32 TDM slots) - MCASP_AFSRCTL[15-7] RMOD set in range 0x3 - 0x20
  - For the special 384-slot TDM mode - MCASP_AFSRCTL[15-7] RMOD = 0x180
- Bit delay: 0, 1, or 2 cycles before the first data bit. This delay is defined in the MCASP_RFMT[17-16] RDATDLY bit field

Copyright © 2024 Texas Instruments Incorporated

- Selecting the source (AFSX or AFSR) of receiver internal frame synchronization. This is done in the same bit - MCASP_ACLKXCTL[6] ASYNC, used to define the receiver internal clock source. For more details, refer to Section 12.5.2.3.2.4, *Synchronous and Asynchronous Transmit and Receive Operations*.

Regardless of the AFSX/AFSR being internally generated or externally sourced, the polarity of AFSX/AFSR is determined by FSXP/FSRP, respectively, to be either rising or falling edge. If FSXP/FSRP = 0, the frame sync polarity is rising edge. If FSXP/FSRP = 1, the frame sync polarity is falling edge.

Note

Certain restrictions apply to the receive and transmit logic settings, when the MCASP_ACLKXCTL[6] ASYNC bit is set to 0b0. They are described in Section 12.5.2.3.2.4.

12.5.2.3.2.4 Synchronous and Asynchronous Transmit and Receive Operations

Synchronous Transmit and Receive Operations -

When the MCASP_ACLKXCTL[6] ASYNC bit is written to 0b0, the transmit and receive sections operate synchronously to the transmit section clock and transmit frame sync signals.

Though Rx section may have a different data format, it has to be configured to have the same slot size than the transmit section one. As shown on the Figure 12-267, with the ASYNC bit set to 0b0, the RCLK becomes an inverted version of the transmit clock generator XCLK output.

When the MCASP_ACLKXCTL[6] ASYNC = 0b0, both Rx and Tx sections use the same clock and frame sync signals. For this reason, they must be aligned on the following settings:

- MCASP_DITCTL[0] DITEN = 0 (that is, transmission in TDM mode is enabled)
- The total number of bits per frame must be the same (that is, RSSZ * RMOD product value must equal XSSZ * XMOD product value)
- The settings in the MCASP_ACLKRCTL register are NOT considered
- FSXM must match FSRM
- FXWID must match FRWID

For all other settings, the transmit and receive sections may be programmed independently.

Asynchronous Transmit and Receive Operations -

When the MCASP_ACLKXCTL[6] ASYNC = 0b1, Tx and Rx operate independently from each other with separate clock and frame sync signals.

Note

Synchronous transmit and receive operations are allowed only in the MCASP TDM (I2S) mode (this is, when MCASP_DITCTL[0] DITEN = 0b0).

12.5.2.3.3 MCASP Frame Sync Feedback for Cross Synchronization

The device level requirement is to align frame syncs across McASP that may be feeding DACs with different formats but the same frequency.

The intent of is to provide a mechanism to check for the alignment of frame syncs. This works by feeding the frame sync of McASP into the a receive data pin of McASP. If this serializer is enabled as a receiver, then the frame sync of McASP will appear as receive data for McASP.

To adjust the McASP transmit bit clock, the MCASP_ACLKXCTL[17-16] CLKXADJ register bit field can be used to lengthen (write 0x2) or reduce (write 0x1) the McASP bit clock period in a one-shot fashion. After each adjustment the frame sync should be checked again and the process repeated until the two frame syncs are aligned.

This process is expected to be carried out once, before any audio transfers occur. It is not intended to be used for sample rate conversion but rather initial phase alignment of transmit data across McASP.

Copyright © 2024 Texas Instruments Incorporated

12.5.2.3.4 MCASP Serializers

The MCASP serializers shift serial data in (Rx) and out (Tx) of the MCASP. A given serializer n consists of a shift register (XRSRn) with a single-entry data buffer XRBUFn used either for transmitting (write accessible in the MCASP_XBUFn register) or for receiving (read accessible in the MCASP_RBUFn register) data. In addition, each serializer has a dedicated control register (MCASP_SRCTLn) and a serial bidirectional data pin - AXRn. The register MCASP_SRCTLn allows n-th serializer to be configured as a transmitter, receiver, or as inactive. There are transmit and receive data formatting units to support data alignment options of the MCASP which are shared between all Tx and Rx serializers, respectively.

A given serializer XRSRn shifter configured as a receiver in the MCASP_SRCTLn register, shifts in data through MCASP corresponding device level bidirectional data pad AXRn. A given serializer XRSRn shifter configured as a transmitter in the MCASP_SRCTLn register, shifts out data on MCASP corresponding device level bidirectional data pad AXRn (n = 0 to 15).

The serializer is clocked from the transmit section clock (ACLKX signal) if configured to transmit or clocked from the receive section clock (ACLKR signal) if configured to receive. A serializer configured to transmit and receive operates in lockstep, which means that for MCASP there are at most a couple of zones, one for transmit and one for receive.

Figure 12-269 shows the serializer block diagram.

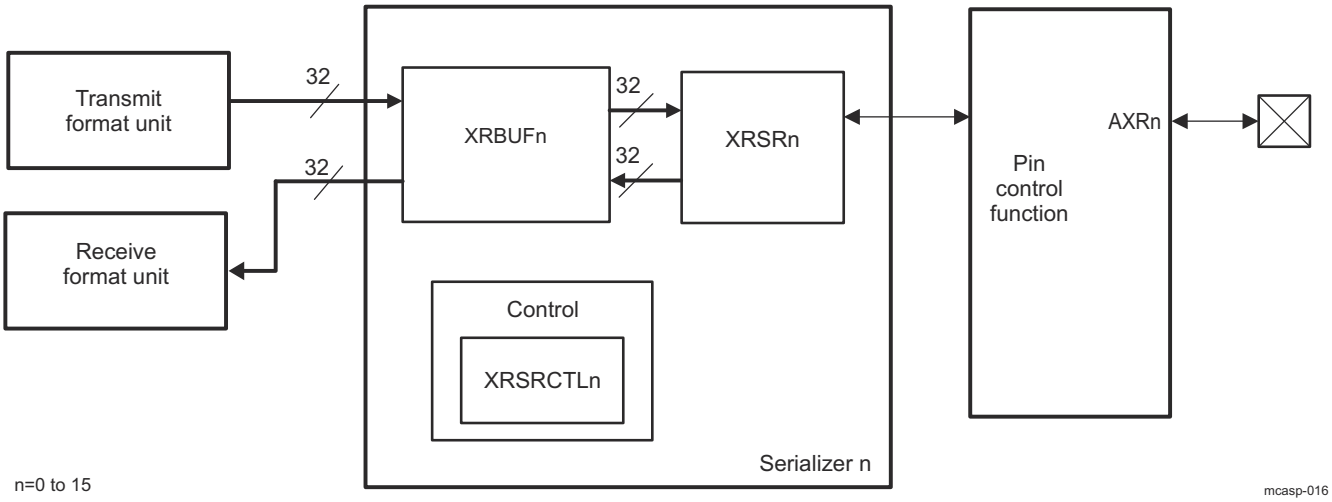


Figure 12-269. Individual Serializer and Connections Within MCASP

Transmission on the n-th serializer is performed as follows:

The MCASP is serviced by writing data into the MCASP_XBUFn register, which is an alias of the serializer data buffer - XRBUFn for transmit function. The data automatically passes through the transmit format unit before reaching the XRBUFn register in the serializer. The data is then copied from the XRBUFn to XRSRn and shifted out from AXRn synchronously to the serial clock.

Reception from the n-th serializer is performed as follows:

The data is shifted into the MCASP XRSRn serializer register through the AXRn pin, bit by bit. Once the entire slot becomes available within the XRSRn shift register, the data is copied into the serializer data buffer - XRBUFn, and can be accessed in the MCASP_RBUFn register, which is an alias of the serializer data buffer - XRBUFn for receive function. When software reads the data from this register, the MCASP passes the data through the receive format unit, hence it returns the formatted data.

Serializer controls:

A serializer n is configured as inactive via setting MCASP_SRCTLn[1-0] SRMOD bit field to 0x0.

For a transmitting serializer, the MCASP_SRCTLn[3-2] DISMOD bitfield, defines the AXRn pin output state, during inactive slots (HIGH, LOW or Hi-Z).

Copyright © 2024 Texas Instruments Incorporated

Transmit function for the n-th serializer is selected via setting MCASP_SRCTLn[1-0] SRMOD bit field to 0x1.

Receive function for the n-th serializer is selected via setting MCASP_SRCTLn[1-0] SRMOD bit field to 0x2 (n = 0 to 15).

In the DIT-transmission mode (that is S/PDIF format data transmission): in addition to the data, the serializer shifts out other DIT-specific information accordingly (preamble, user data, etc.). For more information, see *S/PDIF Coding Format*.

12.5.2.3.5 MCASP Format Units

The MCASP has one transmit data formatting unit and one receive data formatting unit, shared between the device MCASP serializers. These units automatically remap the data bits within the transmitted or received words between a natural format for the device processors (for example, a Q31 representation) and the required format for the external serial device (for example I2S format). During the remapping process, the format unit can also mask off certain bits.

Since all transmitters share the same data formatting unit, the MCASP only supports one transmit format at a time. For example, the MCASP does NOT transmit in "I2S format" on serializer 0, while transmitting "Left Justified" on serializer 1. Likewise, the receiver section of the MCASP only supports one data format at a time, and this format applies to all receiving serializers.

Note

The MCASP can transmit in one format while receiving in a completely different format.

The bit mask and pad stage of each of Tx and Rx format units includes a full 32-bit mask register, allowing selected individual bits to either pass through the stage unchanged, or be masked off. The bit mask and pad then pad the value of the masked off bits by inserting either a 0, a 1, or one of the original 32 bits as the pad value. The last option allows for sign-extension when the sign bit is selected to pad the remaining bits. The rotate right stage performs bitwise rotation by a multiple of 4 bits (between 0 and 28 bits), programmable by the MCASP_RFMT/MCASP_XFMT register. Note that this is a rotation process, not a shifting process, so bit 0 gets shifted back into bit 31 during the rotation. The bit order - reversal stage either passes all 32 bits directly through, or swaps them. This allows for either MSB or LSB first data formats. If bit order reversal is not enabled, then the MCASP will naturally transmit and receive in an LSB first order. Finally, note that the RDATDLY/XDATDLY bits in the MCASP_RFMT/MCASP_XFMT also determine the data format. For example, the difference between I2S format and left-justified is determined by the delay between the frame sync edge and the first data bit of a given time slot. For I2S format, RDATDLY/XDATDLY should be set to a 1-bit delay, whereas for left-justified format, it should be set to a 0-bit delay. The combination of all the options in the MCASP_RFMT/MCASP_XFMT register means that the MCASP supports a wide variety of data formats, both on the serial data lines, and in the device CPU data representation.

12.5.2.3.5.1 Transmit Format Unit

The MCASP transmit formatting unit consists of three stages :

- Bit mask (masks off bits)
- Rotate right (aligns data within word)
- Bit reversal (selects between MSB-first or LSB-first)

Figure 12-270 shows the transmit formatting unit.

The MCASP transmitter supports serial formats of:

- Slot (or Time slot) size = 8, 12, 16, 20, 24, 28, 32 bits
- Word size ≤ Slot size
- Alignment: when more bits/slot than bits/words, then:
  - Left aligned = word shifted first, remaining bits are pad
  - Right aligned = pad bits are shifted first, word occupies the last bits in slot
- Order of bits shifted out:
  - MSB: most-significant bit of word is shifted out first, last bit is LSB

Copyright © 2024 Texas Instruments Incorporated

– LSB: least-significant bit of word is shifted out last, last bit is MSB

Hardware support for these serial formats comes from the programmable options in the bitstream format register - MCASP_XFMT:

- XRVRS: bit reverse (1) or no bit reverse (0)
- XROT: rotate right by 0, 4, 8, 12, 16, 20, 24, or 28 bits
- XSSZ: transmit slot size of 8, 12, 16, 20, 24, 28, or 32 bits

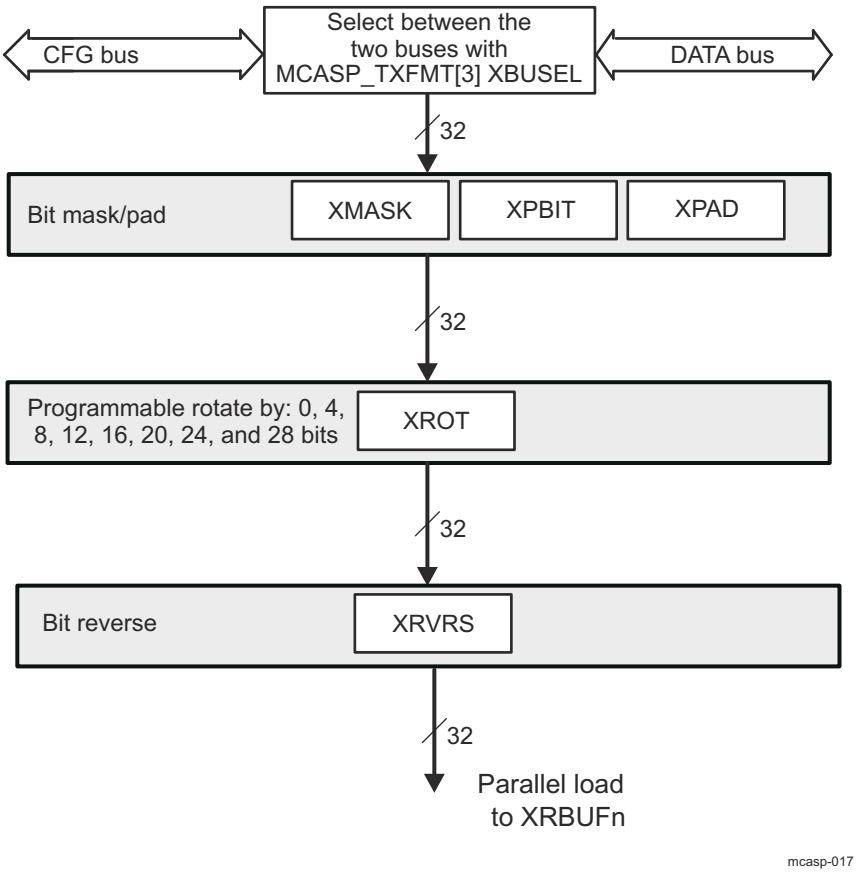


Figure 12-270. Transmit Format Unit

As shown in Figure 12-270, the data to the transmit format unit can come from the configuration port (CFG) or the data port (DATA). The selection is made through the MCASP_XFMT[3] XBUSEL bit. According to port type selected, data transfer has different behaviour. For more details, refer to the Section 12.5.2.3.11.1.3, *Transfers Through the DATA Port*, and Section 12.5.2.3.11.1.4, *Transfers Through the Configuration (CFG) Bus*.

In the transmit format unit (TFU), the input data bits are first masked-off with the MCASP_XMASK[31-0] XMASK contents. The masked data is then right-rotated to the MCASP_XFMT[2-0] XROT bit field positions, to produce the output word for a TDM- or DIT- transmission.

The bit mask stage includes a full 32-bit mask register, allowing selected individual bits to pass through the stage unchanged or be masked off.

12.5.2.3.5.1.1 TDM Mode Transmission Data Alignment Settings

The TDM-mode transmission settings are relevant for I2S-protocol and protocols using more than 2 TDM-slots.

XSSZ should always be programmed to match the slot size of the serial stream.

Copyright © 2024 Texas Instruments Incorporated

Note

Note that, TDM word size is not directly programmed into the MCASP, but rather is used to determine the rotation needed in the XROT field.

The Table 12-308 show the XRVRS and XROT fields for each serial format and for both integer and Q31 fractional internal representations.

The Table 12-308 assumes that all slot size (SLOT in Table 12-308) and word size (WORD in Table 12-308) options are multiples of 4, since the transmit rotate right unit only supports rotation by multiples of 4. However, the bit mask/pad unit does allow for any number of significant digits. For example, a Q31 number may have 19 significant digits (word) and be transmitted in a 24-bit slot; this would be formatted as a word size of 20 bits and a slot size of 24 bits. However, it is possible to set the bit mask unit to only pass the 19 most-significant digits (program the mask value to FFFF E000h). The digits that are not significant can be set to a selected pad value, which can be any one of the significant digits, a fixed value of 0, or a fixed value of 1.

The transmit bit mask/pad unit operates on data as an initial step of the transmit format unit, and the data is aligned in the same representation as it is written to the transmitter (typically Q31 or integer).

Table 12-308. MCASP TFU TDM Mode Settings

Bit Stream Order	Bit Stream Alignment	Internal Numeric Representation	MCASP_XFMT bits	
			XROT[(1)]	XRVRS
MSB first[(2)]	Left aligned	Q31 fraction	0	1
MSB first	Right aligned	Q31 fraction	SLOT - WORD	1
LSB first	Left aligned	Q31 fraction	32 - WORD	0
LSB first	Right aligned	Q31 fraction	32 - SLOT	0
MSB first[(2)]	Left aligned	Integer	WORD	1
MSB first	Right aligned	Integer	SLOT	1
LSB first	Left aligned	Integer	0	0
LSB first	Right aligned	Integer	(32 - (SLOT - WORD)) % 32	0

(1) WORD = Word size rounded up to the nearest multiple of 4; SLOT = slot size; % = modulo operator
(2) To transmit in I2S format, select MSB first, left aligned, and also select XDATDLY = 01 (1 bit delay)

12.5.2.3.5.1.2 DIT Mode Transmission Data Alignment Settings

In case of a DIT-mode (S/PDIF protocol) transmission, while left-aligned Q31 data should be right-rotated to a multiple by 4 positions, no right-rotation is required for a right-aligned Q31 data. Because this is a rotation process, not a shifting process, bit 0 gets shifted back into bit 31 during the process.

The MCASP_XFMT[17-16] XDATDLY bit field must be set to a 0-bit delay (0x0 value).

For left-aligned Q31 data, the following transmit format unit settings process the data into right-aligned data, ready for transmission:

- MCASP_XFMT[2-0] XROT =
  - 0x2 (rotate right by 8 bits) - for a 24-bit output audio data
  - 0x3 (rotate right by 12 bits) - for a 20-bit output audio data
  - 0x4 (rotate right by 16 bits) - for a 16-bit output audio data
- MCASP_XFMT[15] XRVRS = 0x0 – Bit reversal is not enabled; the MCASP naturally transmits and receives in a LSB-first order.
- MCASP_XMASK[32] XMASK = 0xFFFFFF00 – 0xFFFF0000
- MCASP_XFMT[14-13] XPAD = 0x0 (Pad extra bits with 0s.)

For right-aligned data, the following transmit format unit settings process the data into right-aligned audio data ready for transmission:

- MCASP_XFMT[2-0] XROT = 0x0 (rotate right by 0 bits regardless of the audio word length)
- MCASP_XFMT[15] XRVRS = 0x0 – Bit reversal is not enabled; the MCASP naturally transmits and receives in a LSB-first order.
- MCASP_XMASK[32] XMASK = 0x00FFFFFF – 0x0000FFFF

Copyright © 2024 Texas Instruments Incorporated

- MCASP_XFMT[14-13] XPAD = 0x0 (Pad extra bits with 0s.)

The example settings provided in Table 12-309 should be applied to MCASP in cases of DIT-transmitting a Q31 data as a 24-bit, 20-bit and 16-bit left- or right- aligned audio word, respectively. Note that the listed settings let the MCASP TFU preserve the most significant bits and cut only the LSBs of the original Q31 CPU data:

Table 12-309. MCASP TFU DIT-Mode Example Settings

Output Audio Word Alignment	Audio Word Length	Right-rotation (multiple of 4-bit positions)	XMASK	XROT
LEFT	16	16	0xFFFF0000	0x4
LEFT	20	12	0xFFFFF000	0x3
LEFT	24	8	0xFFFFFF00	0x2
RIGHT	16	0	0x0000FFFF	0x0
RIGHT	20	0	0x000FFFFF	0x0
RIGHT	24	0	0x00FFFFFF	0x0

Assuming that a Q31 data word 0xFA5AFxxx (where x-marked nibbles of the data are applied as padding bits of the word) is generated on the MCASP CFG (peripheral) port. To transmit a left-aligned 20-bit version of same word, preserving the MSBs, according to the Table 12-309, the user must set XMASK = 0xFFFFF000, and to select a right-rotation to 12 positions (XROT = 0x3).

- After applying 0-s (XPAD = 0) as masking-off bits at the first TFU stage, word is transformed to the word 0xFA5AF000.
- After a rotation by 12 positions to the right is performed in TFU, the 20-bit output word obtained is: 0x000FA5AF. Thus the word gets ready for transmission being mapped with its LS-bit as bit 8 and its MS-bit as bit 27 within a S/PDIF bitstream. This word is shifted in a LSB-to-MSB order (XRVRS = 0x0) out of the XRSR register during a DIT-transmission.

Assuming that a right-aligned Q31 data word - 0x yyyyE4B4 is generated by software on the MCASP CFG (peripheral) port (with the presumption that y-marked nibbles of the input data are applied as padding bits). To transmit a right-aligned 16-bit version of same word, preserving the MSBs, according to the table MCASP TFU Example Settings, user is supposed to set XMASK = 0x0000FFFF, and to select right-rotation to 0 positions (XROT = 0x0).

- After masking-off with 0s at first TFU stage, word is transformed to 0x0000E4B4.
- Since no rotation is applied, the 16-bit output word obtained is actually the one obtained in the masking stage – 0x0000E4B4.

The above examples use internal representation in integer and Q31 notation, but other fractional notations are also possible.

12.5.2.3.5.2 Receive Format Unit

The MCASP receive formatting unit consists of three stages:

- Bit mask (masks off bits)
- Rotate right (aligns data within word)
- Bit reversal (selects between MSB first or LSB first)

Figure 12-271 shows the receive format unit (RFU).

Copyright © 2024 Texas Instruments Incorporated

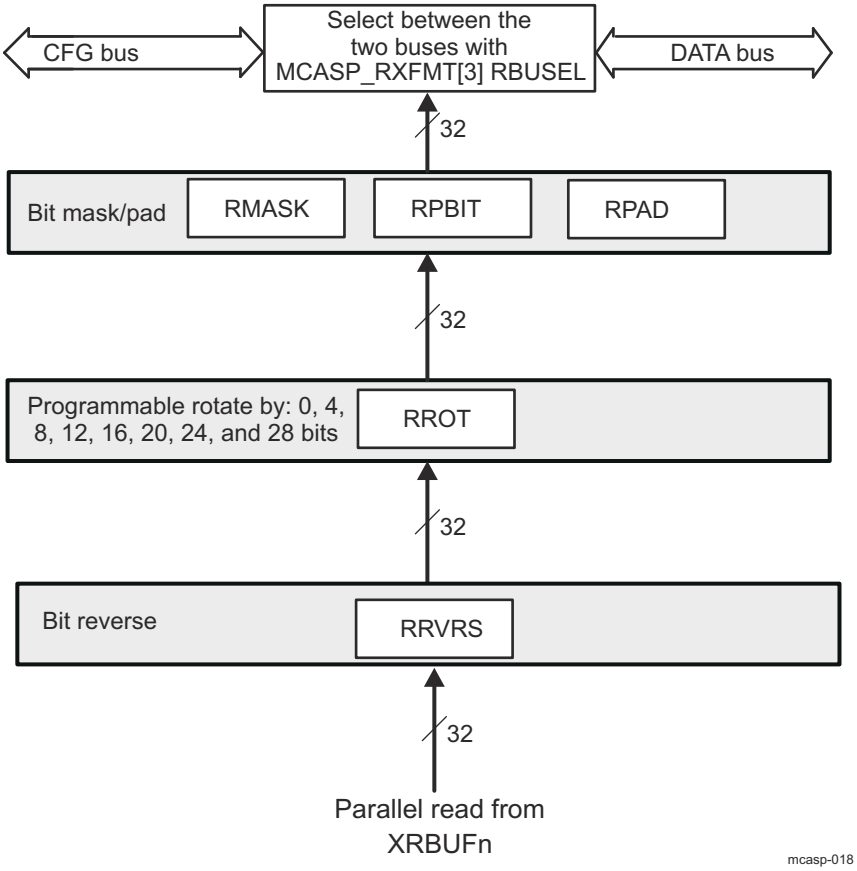


Figure 12-271. Receive Format Unit

The MCASP receiver supports serial formats of:

- Slot or time slot size = 8, 12, 16, 20, 24, 28, 32 bits
- Word size ≤ Slot size
- Alignment when more bits are available per slot than bits per word within the slot, then:
  - Left aligned = word shifted first, remaining bits are pad
  - Right aligned = pad bits are shifted first, word occupies the last bits in slot
- Order of bits shifted out:
  - MSB: most-significant bit of word is shifted out first, last bit is LSB
  - LSB: least-significant bit of word is shifted out last, last bit is MSB

Hardware support for these serial formats comes from the programmable options in the receive bitstream format register - MCASP_RFMT:

- RRVRS: bit reverse (1) or no bit reverse (0)
- RROT: rotate right by 0, 4, 8, 12, 16, 20, 24, or 28 bits
- RSSZ: receive slot size of 8, 12, 16, 20, 24, 28, or 32 bits

As shown on Figure 12-271, the data processed in the RFU can be output to host CPU through the configuration port (CFG) or the data port (DATA). The selection is made through the MCASP_RFMT[3] RBUSEL bit. According to port type selected, data transfer has different behaviour. For more details, refer to the Section 12.5.2.3.11.1.3, *Transfers Through the DATA Port*, and Section 12.5.2.3.11.1.4, *Transfers Through the Configuration (CFG) Bus*.

12.5.2.3.5.2.1 TDM Mode Reception Data Alignment Settings

RSSZ should always be programmed to match the slot size of the serial stream.

Copyright © 2024 Texas Instruments Incorporated

Note

Note that the word size is not directly programmed into the MCASP, but rather is used to determine the rotation needed in the RROT field.

Table 12-310 shows the RRVRS and RROT fields for each serial format and for both integer and Q31 fractional internal representations.

Table 12-310. MCASP RFU Settings

Bit Stream Order	Bit Stream Alignment	Internal Numeric Representation	MCASP_RFMT bits	
			RROT[1]	RRVRS
MSB first[2]	Left aligned	Q31 fraction	SLOT	1
MSB first	Right aligned	Q31 fraction	WORD	1
LSB first	Left aligned	Q31 fraction	(32 - (SLOT - WORD)) % 32	0
LSB first	Right aligned	Q31 fraction	0	0
MSB first[2]	Left aligned	Integer	SLOT - WORD	1
MSB first	Right aligned	Integer	0	1
LSB first	Left aligned	Integer	32 - SLOT	0
LSB first	Right aligned	Integer	32 - WORD	0

(1) WORD = Word size rounded up to the nearest multiple of 4; SLOT = slot size; % = modulo operator
(2) To receive in I2S format, select MSB first, left aligned, and also select RDATDLY = 01 (1 bit delay)

The Table 12-310 assumes that all slot size and word size options are multiples of 4; since the receive rotate right unit only supports rotation by multiples of 4. However, the bit mask/pad unit does allow for any number of significant digits. For example, a Q31 number may have 19 significant digits (word) and be received in a 24-bit slot; this would be formatted as a word size of 20 bits and a slot size of 24 bits. However, it is possible to set the bit mask unit to only pass the 19 most-significant digits (program the mask value to FFFF E000h). The digits that are not significant can be set to a selected pad value, which can be any one of the significant digits, a fixed value of 0, or a fixed value of 1. The receive bit mask/pad unit operates on data as the final step of the receive format unit (see Figure 12-271), and the data is aligned in the same representation as it is read from the receiver (typically Q31 or integer).

12.5.2.3.6 MCASP State-Machines

The receive and transmit sections have independent state machines.

Each state-machine controls the interactions between the various units in the MCASP Rx and Tx sections, respectively. In addition, each state-machine keeps track of error conditions and serial port status. No serial transfers can occur until the RX/TX state-machine is released from reset.

The transmit state-machine is controlled by the transmit bitstream format register (MCASP_XFMT) and it reports the MCASP status and error conditions in the transmitter status register (MCASP_XSTAT).

Similarly, the receive state-machine is controlled by the receive bitstream format register (MCASP_RFMT) and it reports the MCASP status and error conditions in the receiver status register (MCASP_RSTAT).

12.5.2.3.7 MCASP TDM Sequencers

There are separate TDM sequencers for the transmit section and the receive section. Each TDM sequencer keeps track of the slot count. In addition, the TDM sequencer checks the bits of the MCASP_RTDM/ MCASP_XTDM register and determines if the MCASP should receive/transmit in that time slot.

There are two possibilities for a slot: The MCASP either performs Rx/Tx operations during the time slot (transmit/ receive bit is active), or the MCASP skips Rx/Tx operations during the time slot (transmit/receive bit is inactive). In the latter case, no transfers between the XRBUF and XRSR registers in the serializer would occur during that time slot.

In addition, during time of inactive slots, the serializers programmed as transmitters place their data output pins - AXRn in a predetermined state - logic low, high, or high impedance (tri-stated) as programmed in each serializer control MCASP_SRCTLn[3-2] DISMOD register. Refer also to Section 12.5.2.3.10.2.1, *TDM Time Slots*

Copyright © 2024 Texas Instruments Incorporated

Generation and Processing, for details on how DMA event or interrupt generations are handled during inactive time slots in TDM mode.

In case of a DIT-transmission (S/PDIF transfers): the time division multiplexing (TDM) sequencer is used to count the 384 subframes (slots) in the DIT block. If currently transmitting slot 1, slot 2 (next value of the TDM slot counter) should be used during the encode phase to select the appropriate C, V, and U bit, because the data encoded and written to a MCASP_XBUFn register during the current time slot (slot 1) is actually shifted out on the next time slot (n = 0 to 15).

The transmit TDM sequencer is controlled by the MCASP_XTDM register and reports the current transmit slot to the MCASP_XTDMSLOT[9-0] XSLOTCNT bit field.

12.5.2.3.8 MCASP Software Reset

The MCASP can be put into reset through the global transmit and receive control register (MCASP_GBLCTL). A valid serial clock must be supplied to the MCASP to assert the software reset bits in the MCASP_GBLCTL register.

12.5.2.3.9 MCASP Power Management

Table 12-311 describes power-management features available to the MCASP.

Table 12-311. Local Power-Management Features

Feature	Registers	Description
Slave clock stop modes	MCASP_PWRIDLESYSCONFIG[1-0] IDLE_MODE	Force-clock stop, no-clock stop, and smart-clock stop modes are available.

CAUTION

No wakeup schema is supported for the MCASP. To ensure a correct behavior after enabling MCASP at device Device Configuration level, the user software is strongly recommended to choose *No Idle* mode, setting MCASP_PWRIDLESYSCONFIG[1-0] IDLE_MODE bit field to 0x1. Before disabling MCASP at device Device Configuration level, user software is strongly recommended to choose a *Smart-Idle* mode, setting MCASP_PWRIDLESYSCONFIG[1-0] IDLE_MODE bit field to 0x2.

12.5.2.3.10 MCASP Transfer Modes

12.5.2.3.10.1 Burst Transfer Mode

The MCASP supports a burst transfer mode, which is useful for nonaudio data such as passing control information between two processors. Burst transfer mode uses a synchronous serial format similar to the TDM mode. The frame sync generation is not periodic or time-driven as in TDM mode, but data driven, and the frame sync is generated for each data word transferred.

When operating in burst frame sync mode (see Figure 12-272), as specified for transmit (MCASP_AFSXCTL[15-7] = 0) and receive (MCASP_AFSRCTL[15-7] RMOD = 0), one slot is shifted for each active edge of the frame sync signal that is recognized. Additional clocks after the slot and before the next frame sync edge are ignored.

In burst frame sync mode, the frame sync delay may be specified as 0, 1, or 2 serial clock cycles. This is the delay between the frame sync active edge and the start of the slot. The frame sync signal lasts for a single bit clock duration (MCASP_AFSRCTL[4] FRWID = 0, MCASP_AFSXCTL[4] FXWID = 0).

For transmit, when generating the transmit frame sync internally, the frame sync begins when the previous transmission has completed and when all the XBUFn (for every serializer set to operate as a transmitter) has been updated with new data.

For receive, when generating the receive frame sync internally, frame sync begins when the previous transmission has completed and when all the RBUFn (for every serializer set to operate as a receiver) has been read.

Copyright © 2024 Texas Instruments Incorporated

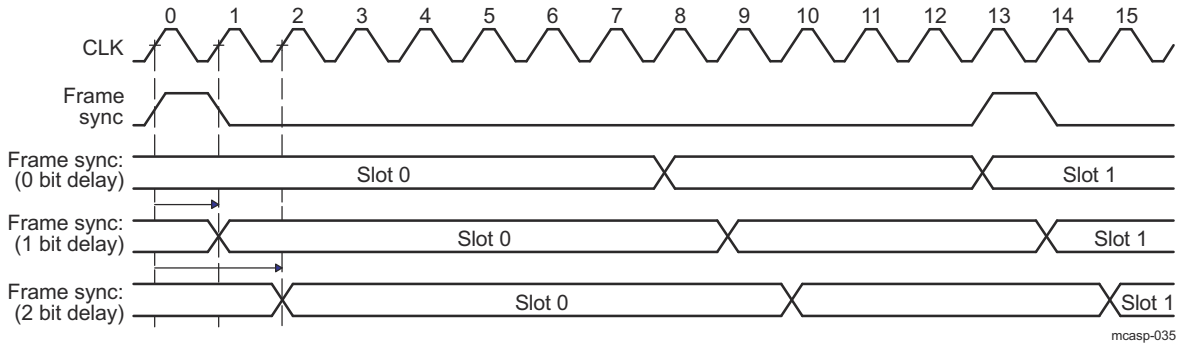


Figure 12-272. Burst Frame Sync Mode

The control registers must be configured as follows for the burst transfer mode. The burst mode specific bit fields are in bold face:

- MCASP_PFUNC: The clock, frame, data pins must be configured for MCASP function.
- MCASP_PDIR: The clock, frame, data pins must be configured to the direction desired.
- MCASP_PDOUT, MCASP_PDIN, MCASP_PDCLR: Not applicable. Leave at default.
- MCASP_GBLCTL: Follow the initialization sequence in *MCASP Global Initialization*, to configure this register.
- MCASP_AMUTE: Not applicable. Leave at default.
- MCASP_DLBCTL: If loopback mode is desired, configure this register according to Section 12.5.2.3.15, *MCASP Loopback Modes*, otherwise leave this register at default.
- MCASP_DITCTL: DITEN must be left at default 0 to select non-DIT mode. Leave the register at default.
- MCASP_RMASK/MCASP_XMASK: Mask desired bits according to Section 12.5.2.3.5, *MCASP Format Units*.
- MCASP_RFMT/MCASP_XFMT: Program all fields according to data format desired. See Section 12.5.2.3.5, *MCASP Format Units*.
- MCASP_RFMT/MCASP_XFMT: Clear RMOD/XMOD bits to 0 to indicate burst mode. Clear FRWID/FXWID bits to 0 for single bit frame sync duration. Configure other fields as desired.
- MCASP_ACLKRCTL/MCASP_ACLKXCTL: Program all fields according to bit clock desired. See Section 12.5.2.3.2, *MCASP Clock and Frame-Sync Configurations*.
- MCASP_AHCLKRCTL/MCASP_AHCLKXCTL: Program all fields according to high-frequency clock desired. See Section 12.5.2.3.2, *MCASP Clock and Frame-Sync Configurations*.
- MCASP_RTDM/MCASP_XTDM: Program RTDMS0/XTDMS0 to 1 to indicate one active slot only. Leave other fields at default.
- MCASP_RINTCTL/MCASP_XINTCTL: Program all fields according to interrupts desired.
- MCASP_RCLKCHK/MCASP_XCLKCHK: Not applicable. Leave at default.
- MCASP_SRCTLn: Program SRMOD to inactive/transmitter/receiver as desired. DISMOD is not applicable and should be left at default (n = 0 to 15).
- MCASP_DITCSRAi, MCASP_DITCSRBi, MCASP_DITUDRAi, MCASP_DITUDRBi: Not applicable. Leave at default (i = 0 to 5).

12.5.2.3.10.2 Time-Division Multiplexed (TDM) Transfer Mode

The MCASP time-division multiplexed (TDM) transfer mode supports the TDM format discussed in *TDM Format*.

Transmitting data in the TDM transfer mode requires a minimum set of pins:

- ACLKX - transmit bit clock
- AFSX - transmit frame sync (or commonly called left/right clock)
- One or more serial data pins, AXRn, whose serializers are configured to transmit

For more details on MCASP transmitting serializers clock and frame sync options, refer to the Section 12.5.2.3.2.1, *Transmit Clock*, and Section 12.5.2.3.2.3, *Frame-Sync Generator*.

Similarly, to receive data in the TDM transfer mode requires a minimum set of pins:

- ACLKR - receive bit clock
- AFSR - receive frame sync (or commonly called left/right clock)
- One or more serial data pins, AXRn, whose serializers are configured to receive

Copyright © 2024 Texas Instruments Incorporated

For more details on MCASP receiving serializers clock and frame sync options, refer to Section 12.5.2.3.2.2, *Receive Clock*, and Section 12.5.2.3.2.3, *Frame-Sync Generator*.

The control registers must be configured as follows for the TDM mode. The TDM mode specific bit fields are highlighted in bold:

- MCASP_PFUNC: The clock, frame, data pins must be configured for MCASP function.
- MCASP_PDIR: The clock, frame, data pins must be configured to the direction desired.
- MCASP_PDOUT, MCASP_PDIN, MCASP_PDCLR: Not applicable. Leave at default.
- MCASP_GBLCTL: Follow the initialization sequence is described in the Section 12.5.2.4.1, *MCASP Operational Modes Configuration*.
- MCASP_AMUTE: Leave this register at default state.
- MCASP_DLBCTL: If loopback mode is desired, configure this register according to Section 12.5.2.3.15, otherwise leave this register at default.
- MCASP_DITCTL: DITEN must be left at default 0 to select TDM mode (transmitters only).
- MCASP_RMASK/MCASP_XMASK: Mask desired bits according to Section 12.5.2.3.5, *MCASP Format Units*.
- MCASP_RFMT/MCASP_XFMT: Program all fields according to data format desired. See the Section 12.5.2.3.5, *MCASP Format Units*.
- MCASP_AFSRCTL/MCASP_AFSXCTL: Set RMOD/XMOD bits to (0x2 - 0x20) for Rx/Tx (2- 32 slots) TDM mode. In addition, set RMOD to 0x180 if 384-slot TDM stream has to be received by MCASP. Configure other fields as desired.
- MCASP_ACLKRCTL/MCASP_ACLKXCTL: Program all fields according to bit clock desired. For more inforamtion, refer to Section 12.5.2.3.2, *MCASP Clock and Frame-Sync Configurations*.
- MCASP_AHCLKRCTL/MCASP_AHCLKXCTL: Program all fields according to high-frequency clock desired. For more details, refer to Section 12.5.2.3.2, *MCASP Clock and Frame-Sync Configurations*.
- MCASP_RTDM/MCASP_XTDM: Program all fields according to the time slot characteristics desired.
- MCASP_XINTCTL: Program all fields according to transmit interrupts desired.
- MCASP_RCLKCHK/MCASP_XCLKCHK: Program all fields according to clock checking desired.
- MCASP_SRCTLn: Program all fields according to serializer operation desired (n = 0 to 15).

Note

The MCASP_DITCSRAi, MCASP_DITCSRBi, MCASP_DITUDRAi, MCASP_DITUDRBi (i = 0 to 5) settings are NOT applicable in TDM transfer modes. They have to be kept at their default values.

12.5.2.3.10.2.1 TDM Time Slots Generation and Processing

TDM mode on the MCASP can extend to support multiprocessor applications, with up to 32 time slots per frame. For each of the time slots, the MCASP may be configured to participate or to be inactive by configuring MCASP_XTDM and/or MCASP_RTDM registers.

The TDM sequencer (separate ones for transmit and receive) functions in this mode. The TDM sequencer counts the slots beginning with the frame sync. For each slot, the TDM sequencer checks the respective bit in either MCASP_XTDM or MCASP_RTDM register to determine if the MCASP transmits/receives in that time slot.

Note

If a MCASP_XTDM/MCASP_RTDM bit defines an active slot (number of slot matches the bit position), the MCASP functions normally during that time slot; otherwise, the MCASP is inactive during that time slot; no update to the buffer occurs, and no event is generated. MCASP (transmit only) data pins are automatically set to a high-impedance state, 0, or 1 during that slot, as determined by bitfield MCASP_SRCTLn[3-2] DISMOD (n = 0 to 15).

Figure 12-273 shows when the transmit DMA event - XINT is generated. See Section 12.5.2.3.11.1, *Data Ready Status and Event/Interrupt Generation* for details on data ready and the initialization period indication. The transmit DMA event for an active time slot (slot N) is generated during the previous time slot (slot N - 1), regardless of the previous time slot (slot N - 1) being active or inactive.

Copyright © 2024 Texas Instruments Incorporated

During an active transmit time slot (slot N), if the next time slot (slot N + 1) is configured to be active, the copy from XRBUFn to XRSRn generates the DMA event for time slot N + 1. If the next time slot (slot N + 1) is configured to be inactive, then the DMA event will be delayed to time slot M - 1. In this case, slot M is the next active time slot. The DMA event for time slot M is generated during the first bit time of slot M - 1.

The receive DMA event is generated after data is received in the buffer (looks back in time). If a time slot is disabled, then no data is copied to the buffer for that time slot and no DMA event is generated.

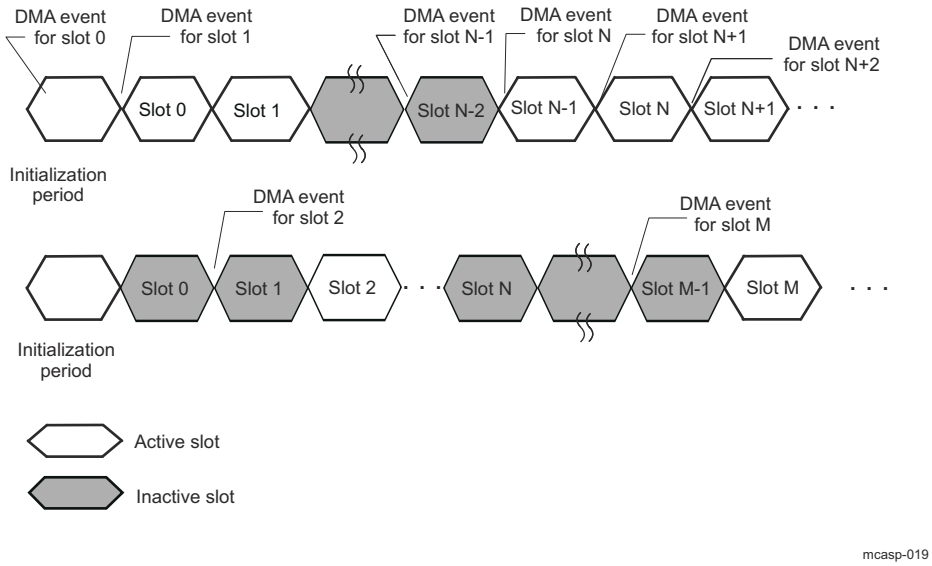


Figure 12-273. Transmit DMA Event (XINT) Generation in TDM Time Slots

12.5.2.3.10.2.2 Special 384-Slot TDM Mode for Connection to External DIR

The MCASP receiver also supports a 384 time slot TDM mode (DIR mode), to support S/PDIF receiver ICs whose natural block (block corresponds to MCASP frame) size is 384 samples. The receive TDM time slot register (MCASP_RTDM) should be programmed to all 1s during reception of a DIR block. Other TDM functionalities (for example, inactive slots) are not supported (only the slot counter counts the 384 subframes in a block). To receive data in DIR mode, the following pins are typically needed:

- ACLKR - receive bit clock
- AFSR - receive frame sync (or commonly called left/right clock)
- In this mode, AFSR should be connected to a DIR which outputs a start of block signal, instead of LRCLK
- One or more serial data pins, AXRn, whose serializers have been configured to receive
- For this special DIR mode, the control registers can be configured just as for TDM mode, except set MCASP_AFSRCTL[15-7] RMOD bit field to 384 (0x180) to receive 384 time slots

12.5.2.3.10.3 DIT Transfer Mode

The DIT transfer mode of the MCASP also supports transmission of audio data in S/PDIF, AES-3, and IEC-60958 formats. These formats are designed to carry audio data between different systems through an optical or coaxial cable. The DIT mode applies only to a serializer configured as transmitter, not as receiver. For a description of the S/PDIF format, see *S/PDIF Coding Format*.

12.5.2.3.10.3.1 Transmit DIT Encoding

When the MCASP operates in DIT mode, the data transmitted is output as a biphase-mark encoded bitstream, with preamble, channel status, user data, validity, and parity automatically stuffed into the bitstream by the MCASP. The MCASP includes separate validity bits for even/odd subframes and two 384-bit RAM modules to hold channel status and user data bits.

Copyright © 2024 Texas Instruments Incorporated

Note

The transmit TDM time slot register (MCASP_XTDM) should be programmed to all 1s during DIT mode. TDM functionality is not supported in DIT mode, except that the TDM slot counter counts the DIT subframes.

To transmit data in DIT mode, the following pins are typically required:

- AHCLKX – transmit high-frequency master clock (The internal clock source can be used instead.)
- One serial data pin (AXRn) of a serializer n configured to transmit.

For DIT Mode Transmission Data Alignment Settings see Section 12.5.2.3.5.1.2, *DIT Mode Transmission Data Alignment Settings*.

If the MCASP is configured to transmit in the DIT mode on more than one serial data pin, the bit streams on all pins will be synchronized. In addition, although they will carry unique audio data, they will carry the same channel status, user data, and validity information.

The actual 24-bit audio data must always be in bit positions 23–0 after passing through the first three stages of the transmit format unit.

12.5.2.3.10.3.2 Transmit DIT Clock and Frame-Sync Generation

The DIT transmitter works only in the following configuration:

- In the transmit frame control register (MCASP_AFSXCTL):
  - Internally generated transmit frame sync, FSXM = 1
  - Rising-edge frame sync, FSXP = 0
  - Bit-width frame sync, FXWID = 0
  - 384-slot TDM, XMOD = 1 1000 0000b
- In the transmit clock control register (MCASP_ACLKXCTL), ASYNC = 1
- In the transmit bitstream format register (MCASP_XFMT), XSSZ = 1111 (32-bit slot size)

All combinations of AHCLKX and ACLKX are supported.

The following summarizes the register configurations required for DIT mode. DIT mode-specific bit fields are in bold face:

- MCASP_PFUNC: The data pin - AXRn must be configured for MCASP function. If AHCLKX is used, it must also be configured for MCASP function. Other pins can be configured to function as GPIOs, if desired.
- MCASP_PDIR: The data pin must be configured as output. If internal clock source AUXCLK is used as the reference clock, it may be output as the AHCLKX device level signal by configuring AHCLKX pin as an output.
- MCASP_GBLCTL: Global initialization
- MCASP_AMUTE: Leave this register at default state.
- MCASP_DITCTL: The DITEN bit must be set to 0b1 to enable DIT mode. Configure other bits as desired.
- MCASP_XMASK: Mask the desired bits, depending upon left-aligned or right-aligned internal data.
- MCASP_XFMT: XDATDLY = 0. XRVRS = 0. XPAD = 0. XSSZ = Fh (32-bit slot). XBUSEL = configured as desired. The XROT bit is configured, as described in the Section 12.5.2.3.5.1.2 .
- MCASP_AFSXCTL: Configure the bits according to former discussions.
- MCASP_ACLKXCTL: ASYNC = 1. Program the CLKXDIV bits to obtain the bit clock rate desired. CLKXM = 1.
- MCASP_AHCLKXCTL: Program the HCLKXDIV bits to obtain the high-frequency bit clock rate desired.
- MCASP_XTDM: Set to FFFF FFFFh for all active slots for DIT transfers.
- MCASP_XINTCTL: Program all fields according to the interrupts desired.
- MCASP_XCLKCHK: Program all fields according to the clock checking desired.
- MCASP_SRCTLn: Set SRMOD = 1 (transmitter) for the DIT pins (n = 0 to 15).
- MCASP_DITCSRAi and MCASP_DITCSRBi: Program the channel status bits as desired (i = 0 to 5).
- MCASP_DITUDRAi and MCASP_DITUDRBi: Program the user data bits as desired (i = 0 to 5).

Copyright © 2024 Texas Instruments Incorporated

Note

In DIT mode, the transmitter can support a 192 kHz frame rate (stereo) on up to 2 serial data pins simultaneously (note that the internal bit clock for DIT runs two times faster than the equivalent bit clock for TDM (I2S) mode, due to the need to generate Biphase Mark Encoded Data - see *Biphase-Mark Code*).

12.5.2.3.10.3.3 DIT Channel Status and User Data Register Files

The channel status registers (MCASP_DITCSRAi and MCASP_DITCSRBi) and user data registers (MCASP_DITUDRAi and MCASP_DITUDRBi) are not double-buffered. Typically, programmers use one of the synchronizing interrupts, such as the last slot, to create an event at a safe time so the register may be updated. In addition, the software reads the transmit TDM slot counter to determine which word of the register is being used (i = 0 to 5).

It is a software requirement to avoid writing to the word of user data and channel status that are being used to encode the current time slot; otherwise, it is undetermined whether old or new data is used to encode the bitstream.

The DIT subframe format is defined in *S/PDIF Subframe Format*. The channel status information (C) and user data (U) are defined in the following DIT control registers:

- MCASP_DITCSRA0 to MCASP_DITCSRA5: The 192 bits in these six registers contain the channel status information for the left channel within each frame.
- MCASP_DITCSRB0 to MCASP_DITCSRB5: The 192 bits in these six registers contain the channel status information for the right channel within each frame.
- MCASP_DITUDRA0 to MCASP_DITUDRA5: The 192 bits in these six registers contain the user data information for the left channel within each frame.
- MCASP_DITUDRB0 to MCASP_DITUDRB5: The 192 bits in these six registers contain the user data information for the right channel within each frame.
- The S/PDIF block format is shown in *S/PDIF Frame Format*. There are 192 frames within a block (frame 0 to frame 191). There are two subframes within each frame (subframes 1 and 2 for the left and right channels, respectively).

The channel status and user data information sent on each subframe is summarized in Table 12-312.

Table 12-312. Channel Status and User Data for Each DIT Block

Frame	Subframe	Preamble	Channel Status Defined in:	User Data Defined in:
Defined by DITCSRA0, DITCSRB0, DITUDRA0, DITUDRB0				
0	1 (L)	B	DITCSRA0[0]	DITUDRA0[0]
0	2 (R)	W	DITCSRB0[0]	DITUDRB0[0]
1	1 (L)	M	DITCSRA0[1]	DITUDRA0[1]
1	2 (R)	W	DITCSRB0[1]	DITUDRB0[1]
2	1 (L)	M	DITCSRA0[2]	DITUDRA0[2]
2	2 (R)	W	DITCSRB0[2]	DITUDRB0[2]
…	…	…	…	…
31	1 (L)	M	DITCSRA0[31]	DITUDRA0[31]
31	2 (R)	W	DITCSRB0[31]	DITUDRB0[31]
Defined by DITCSRA1, DITCSRB1, DITUDRA1, DITUDRB1				
32	1 (L)	M	DITCSRA1[0]	DITUDRA1[0]
32	2 (R)	W	DITCSRB1[0]	DITUDRB1[0]
…	…	…	…	…
63	1 (L)	M	DITCSRA1[31]	DITUDRA1[31]
63	2 (R)	W	DITCSRB1[31]	DITUDRB1[31]
Defined by DITCSRA2, DITCSRB2, DITUDRA2, DITUDRB2				
64	1 (L)	M	DITCSRA2[0]	DITUDRA2[0]
64	2 (R)	W	DITCSRB2[0]	DITUDRB2[0]

Copyright © 2024 Texas Instruments Incorporated

Table 12-312. Channel Status and User Data for Each DIT Block (continued)

Frame	Subframe	Preamble	Channel Status Defined in:	User Data Defined in:
…	…	…	…	…
95	1 (L)	M	DITCSRA2[31]	DITUDRA2[31]
95	2 (R)	W	DITCSRB2[31]	DITUDRB2[31]
Defined by DITCSRA3, DITCSRB3, DITUDRA3, DITUDRB3				
96	1 (L)	M	DITCSRA3[0]	DITUDRA3[0]
96	2 (R)	W	DITCSRB3[0]	DITUDRB3[0]
…	…	…	…	…
127	1 (L)	M	DITCSRA3[31]	DITUDRA3[31]
127	2 (R)	W	DITCSRB3[31]	DITUDRB3[31]
Defined by DITCSRA4, DITCSRB4, DITUDRA4, DITUDRB4				
128	1 (L)	M	DITCSRA4[0]	DITUDRA4[0]
128	2 (R)	W	DITCSRB4[0]	DITUDRB4[0]
…	…	…	…	…
159	1 (L)	M	DITCSRA4[31]	DITUDRA4[31]
159	2 (R)	W	DITCSRB4[31]	DITUDRB4[31]
Defined by DITCSRA5, DITCSRB5, DITUDRA5, DITUDRB5				
160	1 (L)	M	DITCSRA5[0]	DITUDRA5[0]
160	2 (R)	W	DITCSRB5[0]	DITUDRB5[0]
…	…	…	…	…
191	1 (L)	M	DITCSRA5[31]	DITUDRA5[31]
191	2 (R)	W	DITCSRB5[31]	DITUDRB5[31]

12.5.2.3.11 MCASP Data Transmission and Reception

The MCASP is serviced by writing data to the MCASP_XBUFn registers for transmit operations, and by reading data from the MCASP_RBUFn registers for receive operations. The MCASP sets status flags and notifies the software whenever data is ready to be serviced. The Section 12.5.2.3.11.1, *Data Ready Status and Event/ Interrupt Generation*, discusses data-ready status in details (n = 0 to 15).

The MCASP transmit/receive XRBUFn buffer can be accessed through one of the two peripheral ports of the device:

- DATA port: This port is dedicated to DMA initiated data transfers on the device for MCASP transmit (Tx) purposes.
- Configuration bus (CFG): The configuration bus- CFG port is used for peripheral configuration control and receive/transmit data transfers initiated by the host CPU in the device.

Section 12.5.2.3.11.1.3, *Transfers Through the Data Port (DATA)*, and Section 12.5.2.3.11.1.4, *Transfers Through the Configuration Bus (CFG),* discuss how to perform transfers through the data port (DATA) and the configuration port (CFG), respectively.

A device CPU and DMA usages are discussed in Section 12.5.2.3.11.1.5, *Using the device CPUs for MCASP Servicing*, and Section 12.5.2.3.11.1.6, *Using the DMA for MCASP Servicing*, respectively.

MCASP DATA port allows DMAs to access the MCASP transmit buffer more efficiently on the CBASS0, using burst transfers. The physical addresses to access these registers are listed in *DMA Registers*.

12.5.2.3.11.1 Data Ready Status and Event/Interrupt Generation

12.5.2.3.11.1.1 Transmit Data Ready

The transmit data ready flag - XDATA in the MCASP_XSTAT register reflects the data ready status of XRBUFn buffers for all of the active slot transmitting serializers. The XDATA flag is set whenever data is transferred from a transmitting serializer buffer - XRBUFn to its corresponding XRSRn shift register. Thus, the XDATA bit indicates the global event that some of the serializers data buffer - XRBUFn is emptied and ready to accept new data from the host (CPU or DMA). The transmit data ready event is individually indicated per serializer in

Copyright © 2024 Texas Instruments Incorporated

its corresponding control register MCASP_SRCTLn[4] XRDY status bit. When this bit is set to 0b1, it notifies to host that this serializer Tx buffer must be serviced (written). When MCASP_XBUFn is written to by the host, the MCASP_SRCTLn[4] XRDY is deasserted to 0b0. As XDATA global flag is an OR-event of all active serializers XRDY flags, it indicates to software the moment, when write service operation has to be initiated by the MCASP host (XDATA=0b1). The XRDY flags have to be sequentially scanned by user software to determine which serializer MCASP_XBUFn register has to be currently written. Once all requested MCASP_XBUFn are written, the serializers control XRDY flags are cleared to 0b0. As a consequence, XDATA flag is deasserted to 0b0, to indicate to SW that write operation is completed for all serializers.

The global XDATA flag can be cleared when the MCASP_XSTAT[5] XDATA bit is written to 0b1, or once MCASP_XBUFn registers of all the serializers, that have previously raised their XRDY flags, are written with corresponding active slot data by the host.

Whenever XDATA is set, the XINT event is automatically generated on MCASP[0-2]_XMIT_DMA_EVT line (if enabled in the MCASP_XEVTCTL register) to notify the DMA of the MCASP_XBUFn empty status. An interrupt - MCASP[0-2]_XMIT_INTR_PEND can be also generated if the XDATA interrupt is enabled in the MCASP_XINTCTL register (for details, see Section 12.5.2.3.13.1, *Transmit Data Ready Interrupt*).

For DMA requests, the MCASP does not require that MCASP_XSTAT register be read between DMA events. This means that, even if MCASP_XSTAT register already has the XDATA flag set to 1 from a previous request, the next transfer triggers another DMA request.

Because the serializer acts in lockstep, only one DMA event is generated to indicate that the transmit serializer is ready to be written to with new data.

Figure 12-274 shows the timing details of when XINT is generated at the MCASP boundary. In this example, as soon as the last bit (A0) of word A is transmitted, the MCASP sets the XDATA flag and generates an XINT event. However, it takes up to five MCASP interface clocks (XINT latency) before XINT is active at the MCASP boundary. Upon XINT, the CPU can begin servicing the MCASP by writing word C into the MCASP_XBUFn (service time). The CPU must write word C into the MCASP_XBUFn within the setup time required by the MCASP (setup time) (n = 0 to 15).

The maximum service time (see Figure 12-274) can be calculated as:

Service Time = Time Slot – XINT Latency –Setup Time

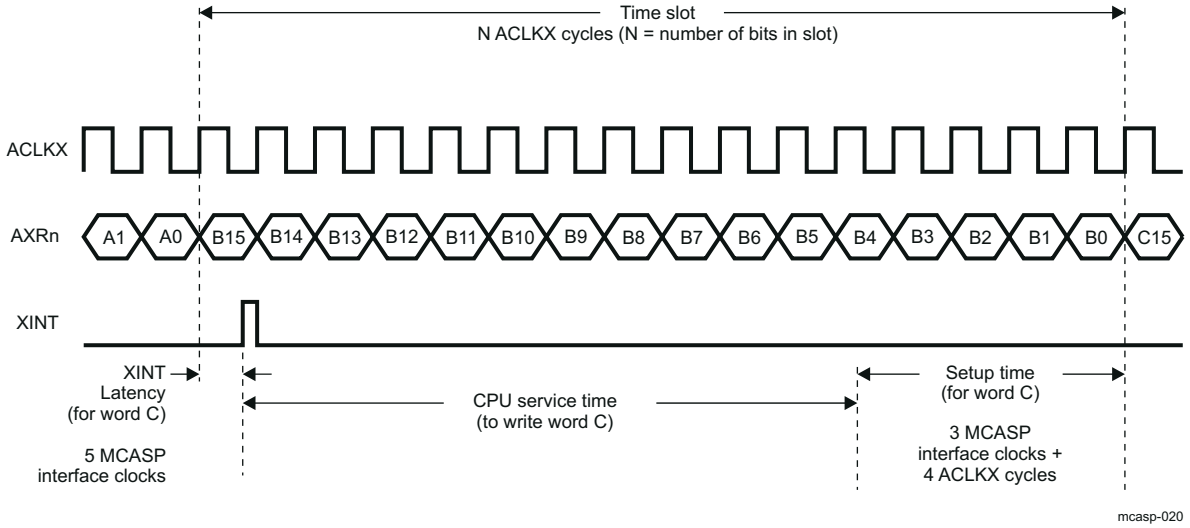


Figure 12-274. Service Time Upon Transmit DMA Event (XINT)

12.5.2.3.11.1.2 Receive Data Ready

Similarly, the receive data ready flag - RDATA in the MCASP_RSTAT register reflects the data ready status of XRBUFn buffers for all of the active slot receiving serializers. The RDATA flag is set whenever data is transferred from a receiving serializer shift register XRSRn to its corresponding XRBUFn data buffer. Thus, the RDATA bit

Copyright © 2024 Texas Instruments Incorporated

indicates the global event that some of the receivers data buffer - RXBUFn already contains received data (this is, a buffer is full) and is ready to transfer it to the host. The receive data ready event is individually indicated per serializer in its corresponding control register MCASP_SRCTLn [5] RRDY status bit. When this bit is set to 0b1, it notifies to host that this serializer Rx buffer must be serviced (read). When MCASP_RBUFn register is read from the host, the MCASP_SRCTLn [5] RRDY bit is deasserted to 0b0. As RDATA global flag is an OR-event of all active serializers RRDY flags, it indicates to software the moment, when read service operation has to be initiated by the MCASP host (RDATA = 0b1). The RRDY flags have to be sequentially scanned by user software to determine which serializer MCASP_RBUFn register has to be currently read. Once all requested MCASP_RBUFn registers are read, the serializers control RRDY flags are cleared to 0b0. As a consequence, RDATA flag is deasserted to 0b0, to indicate to SW that read operation is completed for all serializers.

The global RDATA flag can be cleared when the MCASP_RSTAT[5] RDATA bit is written to 0b1, or once MCASP_RBUFn registers of all the serializers, that have previously raised their RRDY flags, are read by the host.

Whenever RDATA flag is set, the RINT event is automatically generated on MCASP[0-2]_REC_DMA_EVT line (if enabled in the MCASP_PIDTCTL register) to notify the DMA of the MCASP_RBUFn full status. An interrupt - MCASP[0-2]_REC_INTR_PEND can be also generated if the RDATA interrupt is enabled in the MCASP_RINTCTL register (for details, see Section 12.5.2.3.13.1, *Receive Data Ready Interrupt*).

Figure 12-275 shows the timing details of when RINT event is generated at the MCASP boundary. In this example, as soon as the last bit (bit A0) of Word A is received, the MCASP sets the RDATA flag and generates an RINT event. However, it takes up to five MCASP interface clocks (RINT Latency) before RINT is active at the MCASP boundary. Upon RINT, the CPU can begin servicing the MCASP by reading Word A from the MCASP_RBUFn (service time). The CPU must read Word A from the MCASP_RBUFn register no later than the setup time required by the MCASP (Setup Time) (n = 0 to 15).

The maximum service time (see Figure 12-275) can be calculated as:
Service Time = Time Slot - RINT Latency - Setup Time

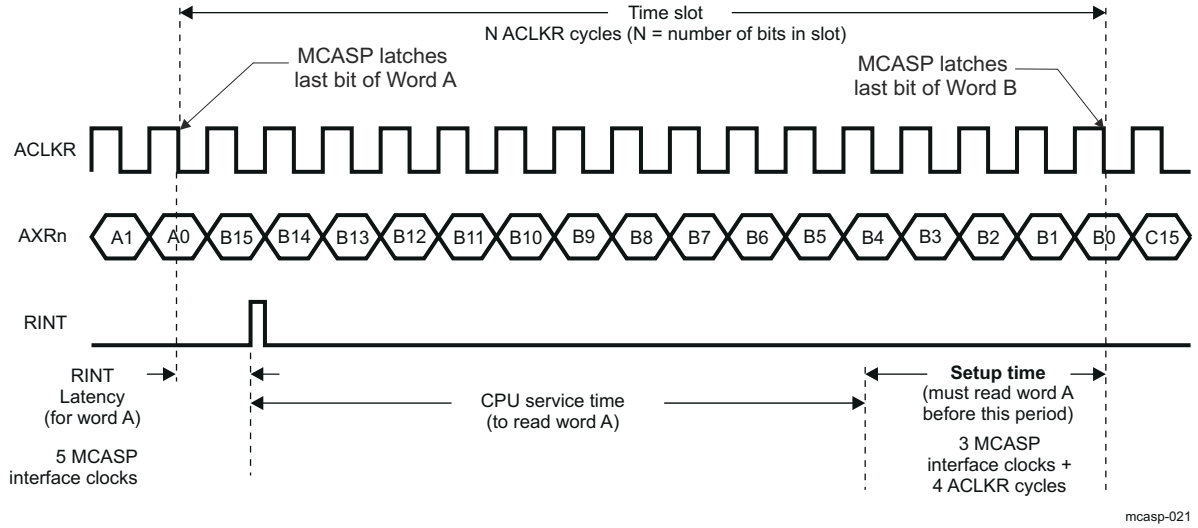


Figure 12-275. CPU Service Time Upon Receive Event (RINT)

12.5.2.3.11.1.3 Transfers Through the Data Port (DATA)

CAUTION

To perform internal transfers through the DATA port, clear the XBUSEL/RBUSEL bit to 0b0 in the MCASP_XFMT/MCASP_RFMT register, respectively. Failure to do so may result in software malfunction.

Copyright © 2024 Texas Instruments Incorporated

In a typical MCASP transfer scenario, the DMA Controller write accesses the XRBUFn transmit buffer through the MCASP data port (DATA) on CBASS0 Interconect. CPU hosts can access both XRBUFn transmit and receive data buffers on their corresponding DATA port address via DATA port corresponding address. To perform transfers through the DATA port, simply have the DMA Controller write the MCASP Tx buffer through Interconect DATA port location. Refer to *DMA Registers*. Although the transfer is passed through an integrated AFIFO transmit/receive buffer, the host (DMA or CPU) must follow the described below procedure to access the data buffers of each serializer, regardless the AFIFO is enabled or disabled. The AFIFO operation is described in Section 12.5.2.3.12.

For accesses through the DATA port, the DMA/CPU services all the serializers through accessing only a single address. In addition, as can be seen in *DMA Registers*, the same physical DATA port address is used regardless of a read or write access is performed. The MCASP automatically cycles through the active slot transmitting/ receiving serializers, internally generating the appropriate offsets.

Note

DATA port allows the DMA/CPU to automatically access only the data buffers. There is no way for DMA/CPU to access the MCASP configuration registers addressing their corresponding MCASP DATA port.

For transmit operations through the DATA port, the host must always write to the same transmit buffer DATA port address (which is same than the receive buffer DATA port adress) to service all of the active slot transmitting serializers. Regardless of MCASP serializer 0 being configured inactive or active, the user software must always configure the destination address to match the DATA port location of TXBUF buffer (See *DMA Registers*).

In addition, the DMA/CPU must write the buffers of all transmitting serializers in incremental (although not necessarily consecutive) order. For example, if only serializers 1 and 3 are set up as active transmitters, to the same transmit buffer DATA port address twice - first data for serializer 1 and second data for serializer 3 upon each transmit data ready event. This exact servicing order must be followed so that data appears in the appropriate serializers.

Note

For write transfers through MCASP DATA port it is preferable to use DMA on corresponding Interconect. This is because DMAs initiated traffic gets better advantage of the burst transfers supported by DATA port.

For receive operations through the DATA port, the DMA/CPU must always read from the same receive buffer DATA port address (which is same than the transmit buffer DATA port adress) to service all of the active slot receiving serializers. Regardless of MCASP serializer 0 being configured inactive or active, the user software must always configure the DMA/CPU source address to match the DATA port location of RXBUF buffer (See *DMA Registers*).

In addition, reads from the receive buffer for all active slot receiving serializers through the Rx DATA port return data in incremental (although not necessarily consecutive) order. For example, if serializers 0, 1 and 3 are set up as active receivers, the CPU should read from the same receive buffer DATA port address three times to obtain data for serializers 0,1 and 3 in this exact order, upon each receive data ready event.

Note

To service a serializer for a transmit or receive operation through the MCASP DATA port, the initiator always writes (preferably DMA) and reads from the same address (refer to *DMA Registers*), respectively.

Copyright © 2024 Texas Instruments Incorporated

Note

When transmitting through the DATA port, the DMA/CPU must write data (at the same address) to each serializer configured as *active* (active slot selected in MCASP_XTDM) and *transmit* (Tx enabled in MCASP_SRCTLn) within each time slot. Failure to do so results in a buffer underrun condition (see Section 12.5.2.3.16.1, *Buffer Underrun Error - Transmitter*). Similarly, when DMA/CPU receives, data must be read from each serializer configured as *active* (active slot selected in MCASP_RTDM) and *receive* (Rx enabled in MCASP_SRCTLn) within each time slot. Failure to do so results in a buffer overrun condition (see Section 12.5.2.3.16.2, *Buffer Overrun Error - Receiver*) (n = 0 to 15).

12.5.2.3.11.1.4 Transfers Through the Configuration Bus (CFG)

CAUTION

To perform internal transfers through the configuration bus, set the XBUSEL/RBUSEL bit to 1 in the MCASP_XFMT/MCASP_RFMT registers, respectively. Failure to do so may result in software malfunction.

Note

MCASP[0-2] whose data ports are accessible directly via CBASS0 do not support FIFO/constant addressing modes. Incrementing transfers must be used instead.

In this method, the CPU accesses the XRBUFn transmit or receive buffer through corresponding configuration bus (CFG) address.

The exact XRBUFn transmit/receive buffer physical address for any particular serializer is determined by adding the transmit/receive buffer alias register offset for that particular serializer to the base address of MCASP CFG port actual for CBASS0 accesses. The XRBUFn buffer of the n-th serializer configured as a transmitter is aliased - MCASP_XBUFn in the CFG port address space. For example, the XRBUF2 transmit buffer is mapped as the MCASP_XBUF2 register. Similarly, the XRBUFn buffer of the n-th serializer configured as a receiver is aliased - MCASP_RBUFn in the CFG port address space. For example, the XRBUF3 receive buffer is mapped as the MCASP_RBUF3 register.

Accessing the XRBUF through the DATA port (see Section 12.5.2.3.11.1.3) is different than CFG port accesses because the DATA port access demands the same physical address, regardless of transfer direction or current channel index , while accessing through the peripheral configuration port - CFG, the CPU must provide the exact MCASP_XBUFn or MCASP_RBUFn address upon accessing n-th serializer TX or RX buffer, respectively. For more details about MCASP_XBUFn and MCASP_RBUFn addresses corresponding to MCASP CFG port, see *CFG Registers* (n = 0 to 15).

12.5.2.3.11.1.5 Using a Device CPU for MCASP Servicing

The device CPUs can be used to service the MCASP transmit channels through interrupts (upon MCASP[0-2]_XMIT_INTR_PEND and MCASP[0-2]_REC_INTR_PEND interrupts). Because these interrupt events are connected to device COMPUTE_CLUSTER0, PRU_ICSSG0/1, MAIN2MCU_LVL_INTRTR0, R5FSS0/1_INTRTR0, C66SS0/1_INTRTR0, R5FSS0/1 modules, they could be software mapped to input interrupt lines of any device CPU. Another way to service the transmit and receive channels, a polling of the XDATA bit in the MCASP_XSTAT register and RDATA bit in the MCASP_RSTAT register can be performed by device CPUs, respectively. As discussed in Section 12.5.2.3.11.1.3, *Transfers Through the Data Port (DATA)*, and Section 12.5.2.3.11.1.4, *Transfers Through the Configuration Bus (CFG)*, the device CPUs can access MCASP XRBUF serializer buffer through their corresponding DATA and CFG port locations.

To use the device CPUs to service the MCASP through interrupts, the XDATA/RDATA bit must be enabled in the respective MCASP_XINTCTL/MCASP_RINTCTL registers, to generate interrupts MCASP[0-2]_XMIT_INTR_PEND/MCASP[0-2]_REC_INTR_PEND to the device CPUs upon data ready

Copyright © 2024 Texas Instruments Incorporated

12.5.2.3.11.1.6 Using the DMA for MCASP Servicing

Note

The associated static Transfer Request (TR) operations of PDMA0, located in front of MCASP, must be configured to match the MCASP configuration. For more information, refer to *DMA Controllers*.

The typical scenario is to use the DMA to service the MCASP transmit and receive logic through the DATA port. The transfer passes through integrated AFIFO transmit/receive buffer. If AFIFO is enabled, DMA requests are collected and fed to a device DMA controller (see Figure 12-265). The data transfer is managed by the AFIFO according to generated transmit and receive events in the MCASP and data is fed to transmit buffers and fetched from receive buffers as described in Section 12.5.2.3.12, *MCASP Audio FIFO (AFIFO)*. The generation of transmit and receive request is described below. After generation of transmit/receive DMA events from MCASP module, these events are collected in AFIFO and on specific AFIFO conditions described in Section 12.5.2.3.12, *MCASP Audio FIFO (AFIFO)* the requests (transmit or receive) are forwarded to a DMA controller via MCASP[0-2]_XMIT_DMA_EVT and MCASP[0-2]_REC_DMA_EVT outputs. If the AFIFO is disabled (default state) it is transperrant for the MCASP module and all request are directly sent to the DMA controller.

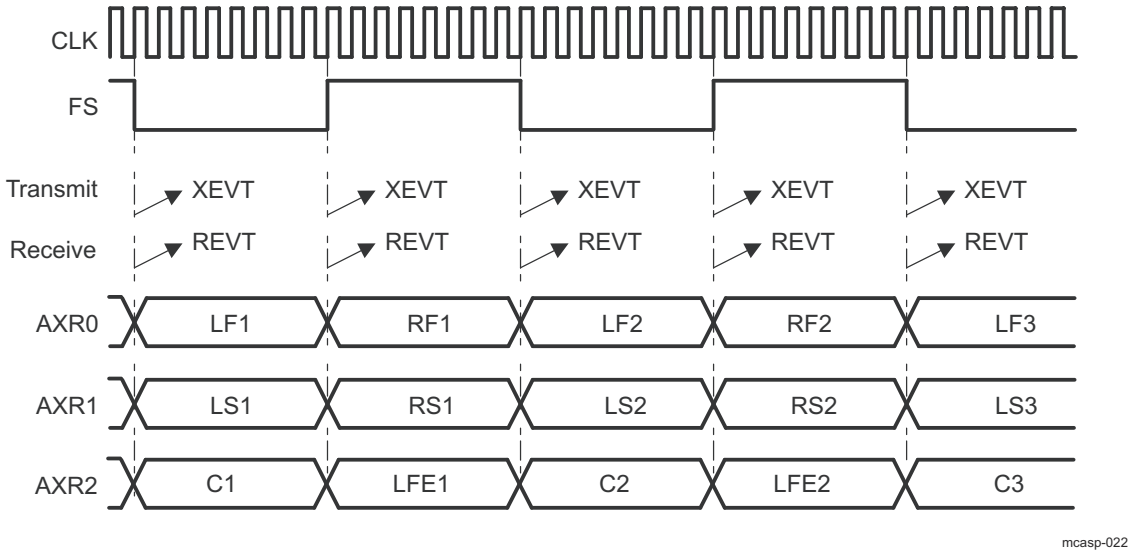


Figure 12-276. DMA Transmit and Receive Event in an Audio Example – One Event

In transmit mode, the DMA event - XINT (MCASP[0-2]_XMIT_DMA_EVT output), which is triggered upon each XDATA transition from 0 to 1, is used to service the MCASP TXBUFn transmit buffers. In receive mode, the DMA event RINT (MCASP[0-2]_REC_DMA_EVT output) which is triggered upon each RDATA transition from 0 to 1, is used to service the MCASP RXBUFn receive buffers.

Figure 12-276 is an example of an audio system with six audio channels (LF, RF, LS, RS, C and LFE) transmitted or received through the MCASP signals - AXR0, AXR1 and AXR2. It shows the points at which events XINT/RINT are triggered.

In Figure 12-276, a Tx DMA event XINT is triggered on each time slot. In the example, XINT is triggered for each of the transmit audio channel time slot (time slot for channels LF, LS, and C; and time slot for channels RF, RS, LFE). Transmit DMA events are generated automatically upon transmit data ready, provided that DMA TX requests generation is enabled in the MCASP_XEVTCTL register. Similarly, Rx DMA event RINT is triggered for each of the receive audio channel time slot. Receive DMA events are generated automatically upon receive data ready, provided that DMA RX requests generation is enabled in the MCASP_PIDTCTL register.

12.5.2.3.12 MCASP Audio FIFO (AFIFO)

The AFIFO contains two FIFOs: one Read FIFO (RFIFO), and one Write FIFO (WFIFO). The RFIFO and the WFIFO are the same size: 64 32-bit Words. To ensure backward compatibility with existing software, both the Read and Write FIFOs are disabled by default. See Figure 12-277 for a high-level block diagram of the AFIFO.

Copyright © 2024 Texas Instruments Incorporated

The AFIFO may be enabled/disabled and configured via the MCASP_WFIFOCTL and MCASP_RFIFOCTL registers. Note that if the Read or Write FIFO is to be enabled, it must be enabled prior to initializing the receive/transmit section of the MCASP.

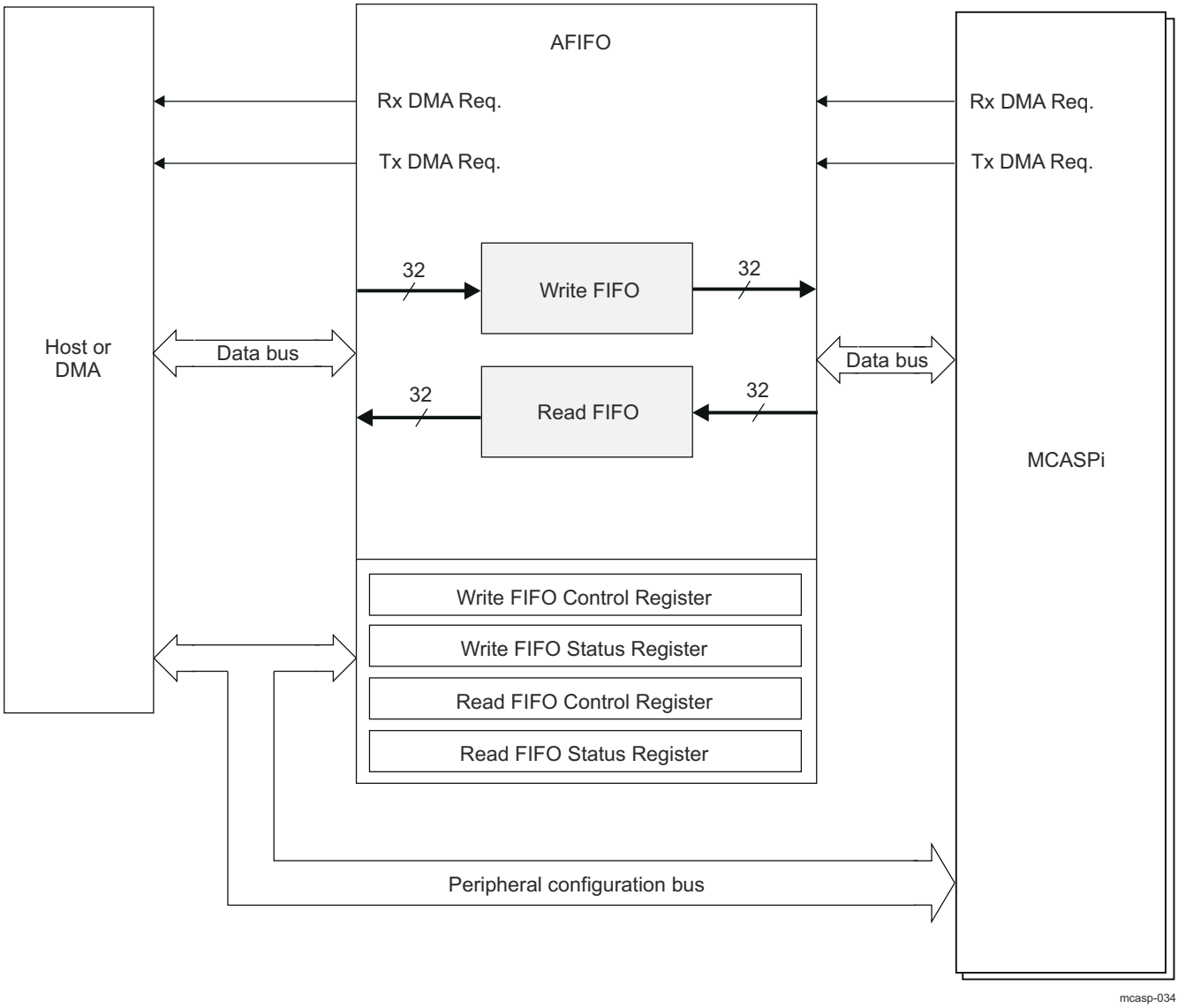


Figure 12-277. MCASP Audio FIFO (AFIFO) Block Diagram

12.5.2.3.12.1 AFIFO Data Transmission

When the Write FIFO is disabled, transmit DMA requests pass through directly from the MCASP to the host/DMA controller. Whether the WFIFO is enabled or disabled, the MCASP generates transmit DMA requests as needed; the AFIFO is “invisible” to the MCASP. When the Write FIFO is enabled, transmit DMA requests from the MCASP are sent to the AFIFO, which in turn generates transmit DMA requests to the host/DMA controller. If the Write FIFO is enabled, upon a transmit DMA request from the MCASP, the WFIFO writes WNUMDMA 32-bit words to the MCASP if and when there are at least WNUMDMA words in the Write FIFO. If there are not, the WFIFO waits until this condition has been satisfied. At that point, it writes WNUMDMA words to the MCASP (see description for the MCASP_WFIFOCTL[7-0] WNUMDMA). If the host CPU writes to the Write FIFO, independent of a transmit DMA request, the WFIFO will accept host writes until full. After this point, excess data will be discarded. Note that when the WFIFO is first enabled, it will immediately issue a transmit DMA request to the host. This is because it begins in an empty state, and is therefore ready to accept data.

Copyright © 2024 Texas Instruments Incorporated

12.5.2.3.12.1.1 Transmit DMA Event Pacer

The AFIFO may be configured to delay making a transmit DMA request to the host until the Write FIFO has enough space for a specified number of words. In this situation, the number of transmit DMA requests to the host or DMA controller is reduced. If the Write FIFO has space to accept WNUMEVT 32-bit words, it generates a transmit DMA request to the host and then waits for a response. Once WNUMEVT words have been written to the FIFO, it checks again to see if there is space for WNUMEVT 32-bit words. If there is space, it generates another transmit DMA request to the host, and so on. In this fashion, the Write FIFO will attempt to stay filled. Note that if transmit DMA event pacing is desired, MCASP_WFIFOCTL[15-8] WNUMEVT bit field should be set to a non-zero integer multiple of the value in MCASP_WFIFOCTL[7-0] WNUMDMA bit field. If transmit DMA event pacing is not desired, then the value in MCASP_WFIFOCTL[15-8] WNUMEVT bit field should be set equal to the value in MCASP_WFIFOCTL[7-0] WNUMDMA bit field.

12.5.2.3.12.2 AFIFO Data Reception

When the Read FIFO is disabled, receive DMA requests pass through directly from MCASP to the host/DMA controller. Whether the RFIFO is enabled or disabled, the MCASP generates receive DMA requests as needed; the AFIFO is “invisible” to the MCASP. When the Read FIFO is enabled, receive DMA requests from the MCASP are sent to the AFIFO, which in turn generates receive DMA requests to the host/DMA controller. If the Read FIFO is enabled and the MCASP makes a receive DMA request, the RFIFO reads RNUMDMA 32-bit words from the MCASP, if and when the RFIFO has space for RNUMDMA words. If it does not, the RFIFO waits until this condition has been satisfied; at that point, it reads RNUMDMA words from the MCASP (see description for theMCASP_RFIFOCTL[7-0] RNUMDMA). If the host CPU reads the Read FIFO, independent of a receive DMA request, and the RFIFO at that time contains less than RNUMEVT words, those words will be read correctly, emptying the FIFO.

12.5.2.3.12.2.1 Receive DMA Event Pacer

The AFIFO may be configured to delay making a receive DMA request to the host until the Read FIFO contains a specified number of words. In this situation, the number of receive DMA requests to the host or DMA controller is reduced. If the Read FIFO contains at least RNUMEVT 32-bit words, it generates a receive DMA request to the host and then waits for a response. Once RNUMEVT 32-bit words have been read from the RFIFO, the RFIFO checks again to see if it contains at least another RNUMEVT words. If it does, it generates another receive DMA request to the host, and so on. In this fashion, the Read FIFO will attempt to stay empty. Note that if receive DMA event pacing is desired, MCASP_RFIFOCTL[15-8] RNUMEVT bit field should be set to a non-zero integer multiple of the value in MCASP_RFIFOCTL[7-0] RNUMDMA bit field. If receive DMA event pacing is not desired, then the value in MCASP_RFIFOCTL[15-8] RNUMEVT bit field should be set equal to the value in MCASP_RFIFOCTL[7-0] RNUMDMA bit field.

12.5.2.3.12.3 Arbitration Between Transmit and Receive DMA Requests

If both the WFIFO and the RFIFO are enabled and a transmit DMA request and receive DMA request occur simultaneously, priority is given to the transmit DMA request. Once a transfer is in progress, it is allowed to complete. If only the WFIFO is enabled and a transmit DMA request and receive DMA request occur simultaneously, priority is given to the transmit DMA request. Once a transfer is in progress, it is allowed to complete. If only the RFIFO is enabled and a transmit DMA request and receive DMA request occur simultaneously, priority is given to the receive DMA request. Once a transfer is in progress, it is allowed to complete.

12.5.2.3.13 MCASP Events and Interrupt Requests

Table 12-313 lists all the transmit event flags. Table 12-314 lists all the Receive event flags. Source of each of these TX/RX events can be a TX/RX channel from any MCASP serializer configured as transmitter or receiver respectively.

Table 12-313. TX Events

Event Mask[(1)]	Event Flag	Map to [(1)]	Description
MCASP_XINTCTL[0] XUNDRN	MCASP_XSTAT[0] XUNDRN	MCASP[0-2]_XMIT_INTR_PEND	Transmit buffer underrun
MCASP_XINTCTL[1] XSYNCERR	MCASP_XSTAT[1] XSYNCERR	MCASP[0-2]_XMIT_INTR_PEND	Unexpected transmit frame sync

Copyright © 2024 Texas Instruments Incorporated

Table 12-313. TX Events (continued)

Event Mask[(1)]	Event Flag	Map to [(1)]	Description
MCASP_XINTCTL[2] XCKFAIL	MCASP_XSTAT[2] XCKFAIL	MCASP[0-2]_XMIT_INTR_PEND	Transmit clock failure
MCASP_XINTCTL[3] XDMAERR	MCASP_XSTAT[7] XDMAERR	MCASP[0-2]_XMIT_INTR_PEND	DATA port transmit error
MCASP_XINTCTL[4] XLAST	MCASP_XSTAT[4] XLAST	MCASP[0-2]_XMIT_INTR_PEND	Transmit last slot interrupt
MCASP_XINTCTL[5] XDATA	MCASP_XSTAT[5] XDATA	MCASP[0-2]_XMIT_INTR_PEND	Transmit data-ready interrupt
MCASP_XINTCTL[7] XSTAFRM	MCASP_XSTAT[6] XSTAFRM	MCASP[0-2]_XMIT_INTR_PEND	Transmit start of frame interrupt
n.a.	MCASP_XSTAT[8] XERR	n.a.	OR-event of all Tx-error events: (XDMAERR | XCKFAIL | XUNDRN | XSYNCERR). It is cleared ONLY when all error flags are cleared
n.a.	MCASP_XSTAT[3] XTDMSLOT	n.a.	Qualifies the current TDM slot as an odd or an even slot.

(1) Every MCASP module generates separate interrupt event.

Table 12-314. RX Events

Event Mask[(1)]	Event Flag	Map to[(1)]	Description
MCASP_RINTCTL[0] ROVRN	MCASP_RSTAT[0] ROVRN	MCASP[0-2]_REC_INTR_PEND	Receive buffer overrun
MCASP_RINTCTL[1] RSYNCERR	MCASP_RSTAT[1] RSYNCERR	MCASP[0-2]_REC_INTR_PEND	Unexpected receive frame sync
MCASP_RINTCTL[2] RCKFAIL	MCASP_RSTAT[2] RCKFAIL	MCASP[0-2]_REC_INTR_PEND	Receive clock failure
MCASP_RINTCTL[3] RDMAERR	MCASP_RSTAT[7] RDMAERR	MCASP[0-2]_REC_INTR_PEND	DATA port receive error
MCASP_RINTCTL[4] RLAST	MCASP_RSTAT[4] RLAST	MCASP[0-2]_REC_INTR_PEND	Receive last slot
MCASP_RINTCTL[5] RDATA	MCASP_RSTAT[5] RDATA	MCASP[0-2]_REC_INTR_PEND	Receive data-ready
MCASP_RINTCTL[7] RSTAFRM	MCASP_RSTAT[6] RSTAFRM	MCASP[0-2]_REC_INTR_PEND	Receive start of frame
n.a.	MCASP_RSTAT[8] RERR	n.a.	OR-event of all Rx-error events: (RDMAERR | RCKFAIL | ROVRN | RSYNCERR). RERR event is cleared once all error flags are cleared.
n.a.	MCASP_RSTAT[3] RTDMSLOT	n.a.	Qualifies the current TDM slot as an odd or an even slot.

(1) Every MCASP module generates separate interrupt event.

Software has to read the MCASP_XSTAT/MCASP_RSTAT register to determine which event occurs at a global level for MCASP Tx/Rx logic. In addition user software has to scan the XRDY/RRDY read-only flags in the MCASP_SRCTLn registers to determine which active serializer is the actual source of the event.

A Tx interrupt line (MCASP[0-2]_XMIT_INTR_PEND) is asserted (active high) when one of the MCASP_XSTAT notified events occurs, provided that it is enabled in its corresponding MCASP_XINTCTL bit. Similarly, a Rx interrupt line (MCASP[0-2]_REC_INTR_PEND) is asserted (active high) when one of MCASP_RSTAT notified events occurs, provided that it is enabled in its corresponding MCASP_RINTCTL bit. See also Section 12.5.2.3.13.4, *Multiple Interrupts* and the Section 12.5.2.3.11.1, *Data Ready Status and Event/Interrupt Generation* (n = 0 to 15).

Copyright © 2024 Texas Instruments Incorporated

12.5.2.3.13.1 Transmit Data Ready Event and Interrupt

The transmit data-ready interrupt (XDATA) is generated if the MCASP_XSTAT[5] XDATA bit is 1 and MCASP_XINTCTL[5] XDATA bit is enabled. The Section 12.5.2.3.11.1, *Data Ready Status and Event/Interrupt Generation*, provides details on when XDATA is set in the MCASP_XSTAT register.

A transmit-start-of-frame interrupt (XSTAFRM) is triggered by the recognition of a transmit frame sync.

A transmit-last-slot interrupt (XLAST) is a qualified version of the data-ready interrupt (XDATA). It has the same behavior than the data-ready interrupt, but is further qualified by having the data requested belonging to the last slot (the slot that just ended is the next-to-last TDM slot, the current slot is the last slot).

12.5.2.3.13.2 Receive Data Ready Event and Interrupt

The receive data-ready interrupt (RDATA) is generated if the MCASP_RSTAT[5] RDATA bit is 1 and MCASP_RINTCTL[5] RDATA bit is enabled. The Section 12.5.2.3.11.1, *Data Ready Status and Event/Interrupt Generation*, provides details on when the MCASP_RSTAT[5] RDATA bit is set.

A receiver start of frame (RSTAFRM) interrupt is triggered by the recognition of a receiver frame sync.

A receiver last slot (RLAST) interrupt is a qualified version of the data ready interrupt (RDATA). It has the same behavior as the data ready interrupt, but is further qualified by having the data in the buffer come from the last TDM time slot (the slot that just ended was last TDM slot).

12.5.2.3.13.3 Error Interrupt

Upon detection, the following error conditions generate interrupt flags:

In the transmit status register (MCASP_XSTAT):

- Transmit underrun (XUNDRN)
- Unexpected transmit frame sync (XSYNCERR)
- Transmit clock failure (XCKFAIL)
- Transmit DATA port error (XDMAERR)

Each interrupt source also has a corresponding enable bit in the transmit interrupt control register (MCASP_XINTCTL). If the enable bit is set, an interrupt is requested when the interrupt flag is set in MCASP_XSTAT. If the enable bit is not set, no interrupt request is generated. However, the interrupt flag may be polled.

In the receive status register (MCASP_RSTAT) :

- Receiver overrun (ROVRN)
- Unexpected receive frame sync (RSYNCERR)
- Receive clock failure (RCKFAIL)
- Receive DATA port error (RDMAERR)

Each interrupt source also has a corresponding enable bit in the receive interrupt control register (MCASP_RINTCTL). If the enable bit is set, an interrupt is requested when the interrupt flag is set in MCASP_RSTAT. If the enable bit is not set, no interrupt request is generated. However, the interrupt flag may be polled.

12.5.2.3.13.4 Multiple Interrupts

This only applies to interrupts and not to DMA requests. The following terms are defined:

- **Active Interrupt Request:** a flag in MCASP_XSTAT is set and the interrupt is enabled in MCASP_XINTCTL.
- **Outstanding Interrupt Request:** An interrupt request has been issued on one of the MCASP transmit interrupt port, but that request has not yet been serviced.
- **Serviced:** The CPUs write to MCASP_XSTAT to clear one or more of the active interrupt request flags.

The first interrupt request to become active for the serializer with the interrupt flag set in MCASP_XSTAT/MCASP_RSTAT and the interrupt enabled in MCASP_XINTCTL/MCASP_RINTCTL generates a request on the MCASP transmit or receive interrupt port.

Copyright © 2024 Texas Instruments Incorporated

If more than one interrupt request becomes active in the same cycle, a single interrupt request is generated on the MCASP transmit or receive interrupt port. Subsequent interrupt requests that become active while the first interrupt request is outstanding do not immediately generate a new request pulse on the MCASP transmit or receive interrupt port.

The interrupt is serviced with the CPU writing to MCASP_XSTAT/MCASP_RSTAT. If any interrupt requests are active after the write, a new request is generated on the MCASP transmit or receive interrupt port.

One outstanding interrupt request is allowed on each port, so a transmit and a receive interrupt request may both be outstanding at the same time.

12.5.2.3.14 MCASP DMA Requests

The MCASP can generate one DMA request to the DMA controller to transmit (MCASP[0-2]_XMIT_DMA_EVT) or receive (MCASP[0-2]_REC_DMA_EVT) data. A DMA request to transmit data is generated if the MCASP_XEVTCTL[0] XDATDMA bit is cleared. A DMA request to receive data is generated if the MCASP_PIDTCTL[0] RDATDMA bit is cleared.

12.5.2.3.15 MCASP Loopback Modes

The MCASP features a digital loopback mode (DLB) that allows loopback test transfers in TDM mode between MCASP transmitters and receivers within the same device. In loopback mode, the output of a transmit serializer is connected internally to the input of a receive serializer. Therefore, a receiver data can be checked against a transmitter data to ensure that the MCASP settings are correct. Digital loopback mode applies to TDM mode only (2 to 32 slots in a frame). It does not apply to DIT mode (XMOD = 0x180) or burst mode (XMOD = 0).

Figure 12-278 shows the basic logical connection of the serializers in loopback mode.

Copyright © 2024 Texas Instruments Incorporated

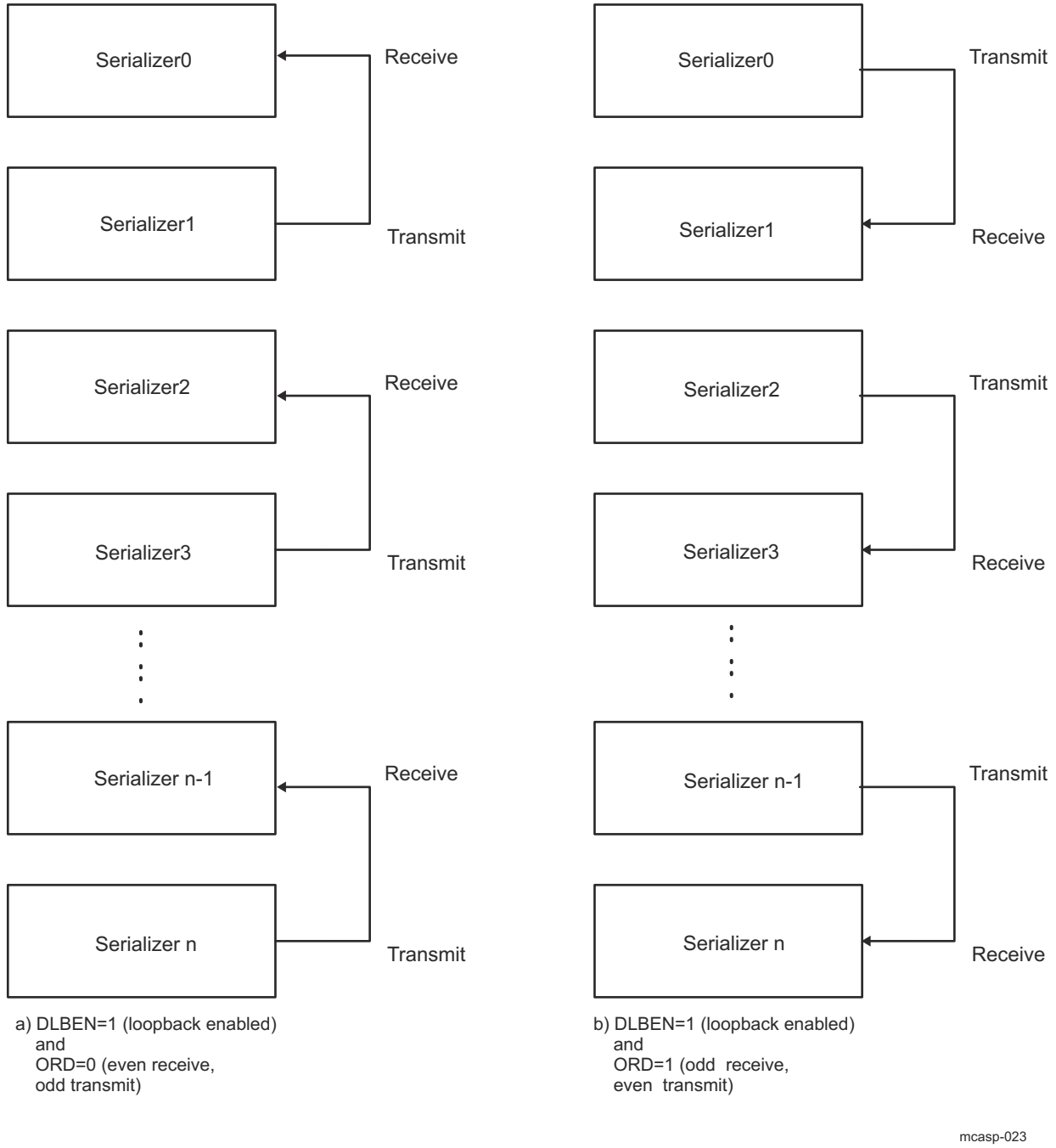


Figure 12-278. MCASP Serializers Operation in Loopback Mode

Two types of loopback connections are possible, selected by the ORD bit in the digital loopback control register - MCASP_DLBCTL as follows:

- ORD = 0: Outputs of odd serializers are connected to inputs of even serializers. If this mode is selected, the odd serializers must be configured as transmitters and even serializers as receivers.
- ORD = 1: Outputs of even serializers are connected to inputs of odd serializers. If this mode is selected, the even serializers must be configured as transmitters and odd serializers as receivers.

User can choose in software (the MCASP_DLBCTL[4] IOLBEN bit) between a MCASP module internal loopback and a device I/O level loopback.

When a **MCASP internal loopback** is selected (MCASP_DLBCTL[4] IOLBEN = 0b0), it is NOT necessary to configure MCASP_PFUNC and MCASP_PDIR registers for MCASP pin settings. Nevertheless, data can be

Copyright © 2024 Texas Instruments Incorporated

optionally made externally visible at the I/O pin of the transmit serializer, if the pin is configured as a MCASP output pin by setting the corresponding MCASP_PFUNC bit to 0 (this is, to function as MCASP, not GPIO) and MCASP_PDIR bit to 1 (output).

When a **device I/O level loopback** is selected (MCASP_DLBCTL[4] IOLBEN = 0b1), the MCASP_PFUNC and MCASP_PDIR registers must be configured with the appropriate settings for all AXRn pins, according to ORD bit configuration.

In case of device I/O loopback, the connectivity is externally applied between device pads (this is, reaching device I/O buffers).

When in loopback mode, the transmit clock and frame sync are used by both the transmit and receive sections of the MCASP. The transmit and receive sections operate synchronously. This is achieved by setting the MCASP_DLBCTL[3-2] MODE bit field to 0x1 and the MCASP_ACLKXCTL[6] ASYNC bit to 0b0.

12.5.2.3.15.1 Loopback Mode Configurations

This is a summary of the settings required for digital loopback mode for TDM format :

- The MCASP_DLBCTL[0] DLBEN bit must be set to 0b1 to enable a loopback mode. It must be kept at 0b0 during normal MCASP operation.
- The MCASP_DLBCTL[4] IOLBEN bit must be set to select between internal (MCASP local) loopback mode or device I/O level loopback mode.
- The MCASP_DLBCTL[3-2] MODE bit field must be set to 0x1 for both the transmit and receive sections to use the transmit clock and frame sync generator.
- The MCASP_DLBCTL[1] ORD bit must be programmed appropriately to select odd or even serializers to be transmitters or receivers.
- The corresponding serializers must be configured accordingly.
- The MCASP_ACLKXCTL[6] ASYNC bit must be cleared to 0b0 to ensure synchronous transmit and receive operations.
- The MCASP_AFSRCTL[15-7] RMOD bit field and MCASP_AFSXCTL[15-7] XMOD bit field must be set within range (0x2 - 0x20) to indicate TDM mode.

Note

Loopback mode does not apply to DIT or burst mode, because MCASP receivers do NOT natively support DIR - reception.

12.5.2.3.16 MCASP Error Reporting

The MCASP includes error-checking capability for the serial protocol and data underrun. In addition, the MCASP includes a timer that continually measures the high-frequency master clock every 32 AHCLKX clock cycles. The value of the timer can be read to get a measurement of the clock frequency and has a minimum and maximum range setting that can set an error flag if the master clock goes out of a specified range.

When one or more errors (software selectable) are detected, an interrupt can be generated if desired, based on one or more error sources.

12.5.2.3.16.1 Buffer Underrun Error -Transmitter

A buffer underrun occurs when a serializer is instructed by the transmit state-machine to transfer data from XRBUFn buffer to XRSRn shift register, but the corresponding (MCASP_XBUFn) register has not yet been written with new data since the last time the transfer occurred. When this occurs, the transmit state-machine sets the XUNDRN flag.

An underrun is checked only once per time slot. The MCASP_XSTAT[0] XUNDRN flag is set when an underrun condition occurs. Once set, the XUNDRN flag remains set until the host explicitly writes 1 to the XUNDRN bit to clear it (n = 0 to 15).

In DIT mode, a pair of BMC zeros is shifted out when an underrun occurs (four bit times at 128 bfs). By shifting out a pair of zeros, a clock can be recovered on the receiver. To recover, reset the MCASP and restart with the proper initialization.

Copyright © 2024 Texas Instruments Incorporated

In TDM mode, during an underrun case, a long stream of zeros are shifted out causing the DACs to mute. To recover, reset the MCASP and start again with the proper initialization.

12.5.2.3.16.2 Buffer Overrun Error-Receiver

A buffer overrun occurs when a serializer is instructed to transfer data from XRSRn shift register to XRBUFn receiver buffer, but the corresponding MCASP_RBUFn register has not yet been read since the last time the transfer occured. When this occurs, the receiver state machine sets the overrun flag - ROVRN. However, the individual serializer writes over the data in the XRBUFn buffer register (destroying the previous sample) and continues shifting (n = 0 to 15).

An overrun is checked only once per time slot. The MCASP_RSTAT[0] ROVRN flag is set when an overrun condition occurs. It is possible that an overrun occurs on one time slot but then the host catches up and does not cause an overrun on the following time slots. However, once the ROVRN flag is set, it remains set until the host explicitly writes a 1 to the ROVRN bit to clear the ROVRN bit.

12.5.2.3.16.3 DATA Port Error - Transmitter

A transmit DATA port error, as indicated by the MCASP_XSTAT[7] XDMAERR bit, occurs when the DMA or device CPU writes more words to the DATA port of the MCASP than it should.

The MCASP_XSTAT[7] XDMAERR = 0b1 indicates that the DMA or device CPU wrote too many words to the MCASP DATA port for a given transmit DMA event. Writing too few words results in a transmit underrun error setting the MCASP_XSTAT[0] XUNDRN bit.

While XDMAERR occurs infrequently, an occurrence indicates a serious loss of synchronization between the MCASP and the DMA or device CPU. The MCASP transmitter and the DMA must be reinitialized to resynchronize them.

12.5.2.3.16.4 DATA Port Error - Receiver

A receive DATA port error, as indicated by the MCASP_RSTAT[7] RDMAERR bit, occurs when the DMA or device CPU reads more words from the DATA port of the MCASP than it should.

The MCASP_RSTAT[7] RDMAERR bit indicates that the DMA or device CPU read too many words from the MCASP DATA port for a given receive RINT event. Reading too few words results in a receiver overrun error setting the MCASP_RSTAT[0] ROVRN bit.

While RDMAERR occurs infrequently, an occurrence indicates a serious loss of synchronization between the MCASP and the DMA or device CPU. The MCASP receiver and the DMA must be reinitialized to resynchronize them.

12.5.2.3.16.5 Unexpected Frame Sync Error

An unexpected frame sync occurs in when:

- in burst mode and TDM mode, the next active edge of the frame sync occurs early such that the current slot will not be completed by the time the next slot is scheduled to begin.
- in TDM mode, an unexpected frame sync occurs also if the frame sync does NOT occur exactly during the correct bit clock (not a cycle earlier or later) and before slot 0.

When an unexpected frame sync occurs, there are two possible actions depending upon when the unexpected frame sync occurs:

1. **Early:** An early unexpected frame sync occurs when the MCASP is in the process of completing the current frame and a new frame sync is detected (not including overlap that occurs due to a 1 or 2 bit frame sync delay). When an early unexpected frame sync occurs:
   - Error event flag is set (XSYNCERR, if an unexpected transmit frame sync occurs; RSYNCERR, if an unexpected receive frame sync occurs).
   - Current frame is not resynchronized. The number of bits in the current frame is completed. The next frame sync, which occurs after the current frame is completed, will be resynchronized.
2. **Late:** A late unexpected frame sync occurs when there is a gap or delay between the last bit of the previous frame and the first bit of the next frame. When a late unexpected frame sync occurs (as soon as the gap is detected):

Copyright © 2024 Texas Instruments Incorporated

- Error event flag is set (XSYNCERR, if an unexpected transmit frame sync occurs; RSYNCERR, if an unexpected receive frame sync occurs).
- Resynchronization occurs upon the arrival of the next frame sync.

Late frame sync is detected the same way in burst mode and TDM mode. However, in burst mode, late frame sync is not meaningful and its interrupt enable should not be set.

12.5.2.3.16.6 Clock Failure Detection

12.5.2.3.16.6.1 Clock Failure Check Startup

It is initially expected of the clock failure circuits to generate an error until at least one measurement is taken. Therefore, the clock failure interrupts, clock switch, and mute functions should not be enabled immediately, but only after a specific startup procedure.

To start the transmit clock failure check procedure:

1. Configure the transmit clock failure detect logic (XMIN, XMAX, XPS) in the transmit clock check control register (MCASP_XCLKCHK).
2. Clear the transmit clock failure flag (XCKFAIL) in the transmit status register (MCASP_XSTAT).
3. Wait until the first measurement is taken (> 32 AHCLKX clock periods).
4. Verify that no clock failure is detected.
5. Repeat Step 2 through Step 4 until the clock is running and is no longer issuing clock failure errors.
6. After the transmit clock is measured and falls within the acceptable range, the following can be enabled:
   a. The transmit clock failure interrupt enable bit (XCKFAIL) in the transmitter interrupt control register (MCASP_XINTCTL)

To start the receive clock failure check procedure:

1. Configure receive clock failure detect logic (RMIN, RMAX, RPS) in the receive clock check control register (MCASP_RCLKCHK).
2. Clear receive clock failure flag (RCKFAIL) in the receive status register (MCASP_RSTAT).
3. Wait until first measurement is taken (> 32 AHCLKR clock periods).
4. Verify no clock failure is detected.
5. Repeat steps 2–4 until clock is running and is no longer issuing clock failure errors.
6. After the receive clock is measured and falls within the acceptable range, the following may be enabled:
   a. the receive clock failure (RCKFAIL) interrupt enable bit in the receive interrupt control register (MCASP_RINTCTL)

12.5.2.3.16.6.2 Transmit Clock Failure Check and Recovery

The transmit clock failure check circuit (see Figure 12-279) works off the internal MCASP interface clock and the external high-frequency serial clock (AHCLKX). It continually counts the number of interface clocks for every 32 high-rate serial clock (AHCLKX) periods, and stores the count in XCNT of the transmit clock check control register (MCASP_XCLKCHK) every 32 high-rate serial clock cycles.

The logic compares the count against a user-defined minimum allowable boundary (XMIN), and automatically flags an interrupt (the MCASP_XSTAT[2] XCKFAIL bit) when an out-of-range condition occurs. An out-of-range minimum condition occurs when the count is less than XMIN. The logic continually compares the current count (from the running interface clock counter) to the maximum allowable boundary (XMAX). This is so that if the external clock completely stops, the counter value is not copied to XCNT. An out-of-range maximum condition occurs when the count is greater than XMAX. The XMIN and XMAX fields are 8-bit unsigned values, and the comparison is performed using unsigned arithmetic.

An out-of-range count may indicate that an unstable clock was detected or that the audio source has changed and a new sample rate is being used.

For the transmit clock failure check circuit to operate correctly, the high-frequency serial clock divider must be taken out of reset.

If a clock failure is detected, the MCASP_XSTAT[2] XCKFAIL transmit clock failure flag is set. This causes an interrupt if the MCASP_XINTCTL[2] XCKFAIL transmit clock failure interrupt enable bit is set.

Copyright © 2024 Texas Instruments Incorporated

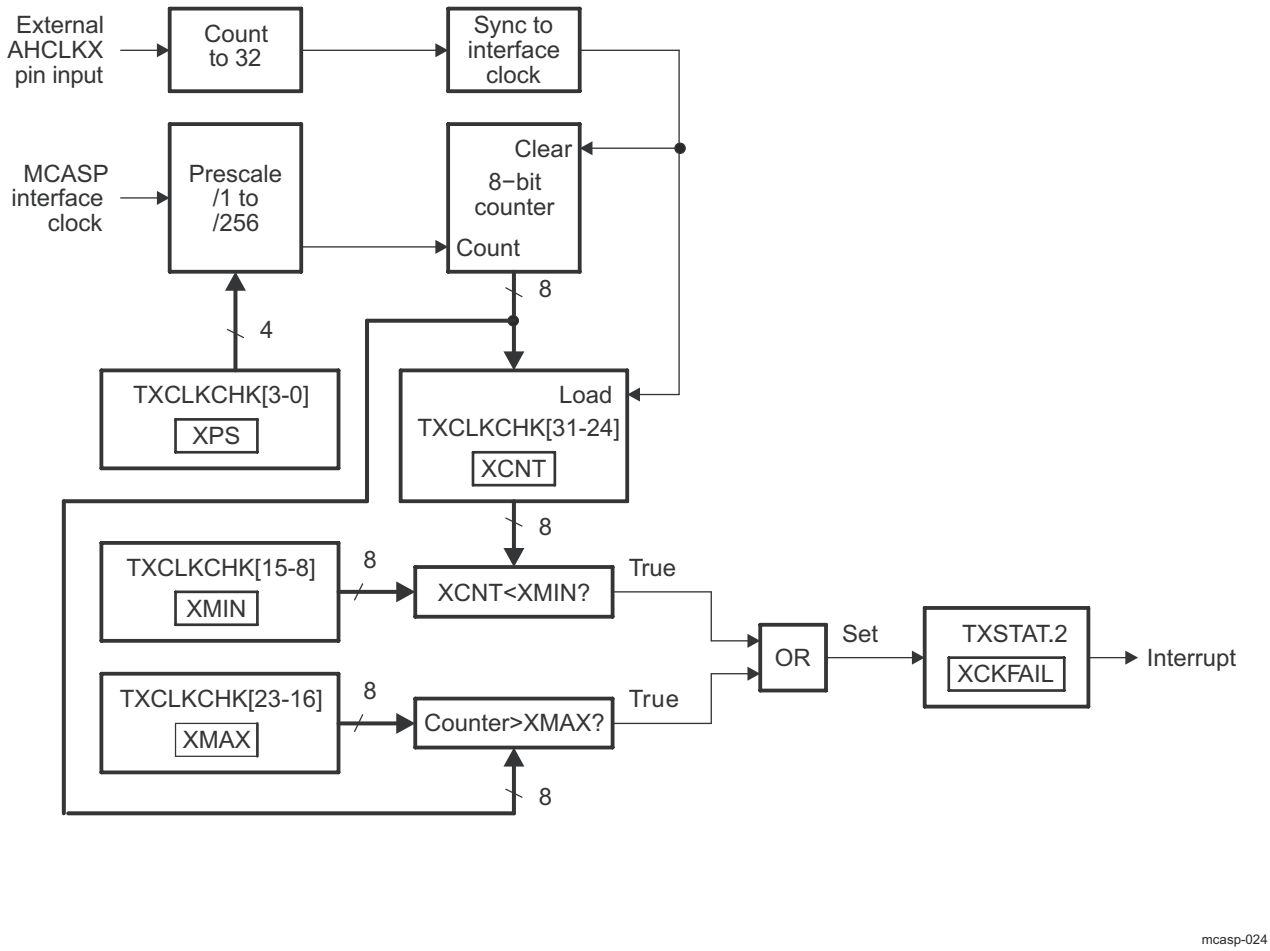


Figure 12-279. Transmit Clock Failure Detection Circuit Block Diagram

12.5.2.3.16.6.3 Receive Clock Failure Check and Recovery

The receive clock failure check circuit (see Figure 12-280) works off both the internal MCASP interface clock and the external high-frequency serial clock (AHCLKR). It continually counts the number of interface clocks for every 32 high rate serial clock (AHCLKR) periods, and stores the count in RCNT of the receive clock check control register (MCASP_RCLKCHK) every 32 high rate serial clock cycles.

The logic compares the count against a user-defined minimum allowable boundary (RMIN) and automatically flags an event (the MCASP_RSTAT[2] RCKFAIL bit) when an out-of-range condition occurs. An out-of-range minimum condition occurs when the count is smaller than RMIN. The logic continually compares the current count (from the running interface clock counter) against the maximum allowable boundary (RMAX). This is in case the external clock completely stops, so that the counter value is not copied to RCNT. An out-of-range maximum condition occurs when the count is greater than RMAX. Note that the RMIN and RMAX fields are 8-bit unsigned values, and the comparison is performed using unsigned arithmetic.

An out-of-range count may indicate either that an unstable clock was detected or that the audio source has changed and a new sample rate is being used.

In order for the receive clock failure check circuit to operate correctly, the high-frequency serial clock divider must be taken out of reset.

Copyright © 2024 Texas Instruments Incorporated

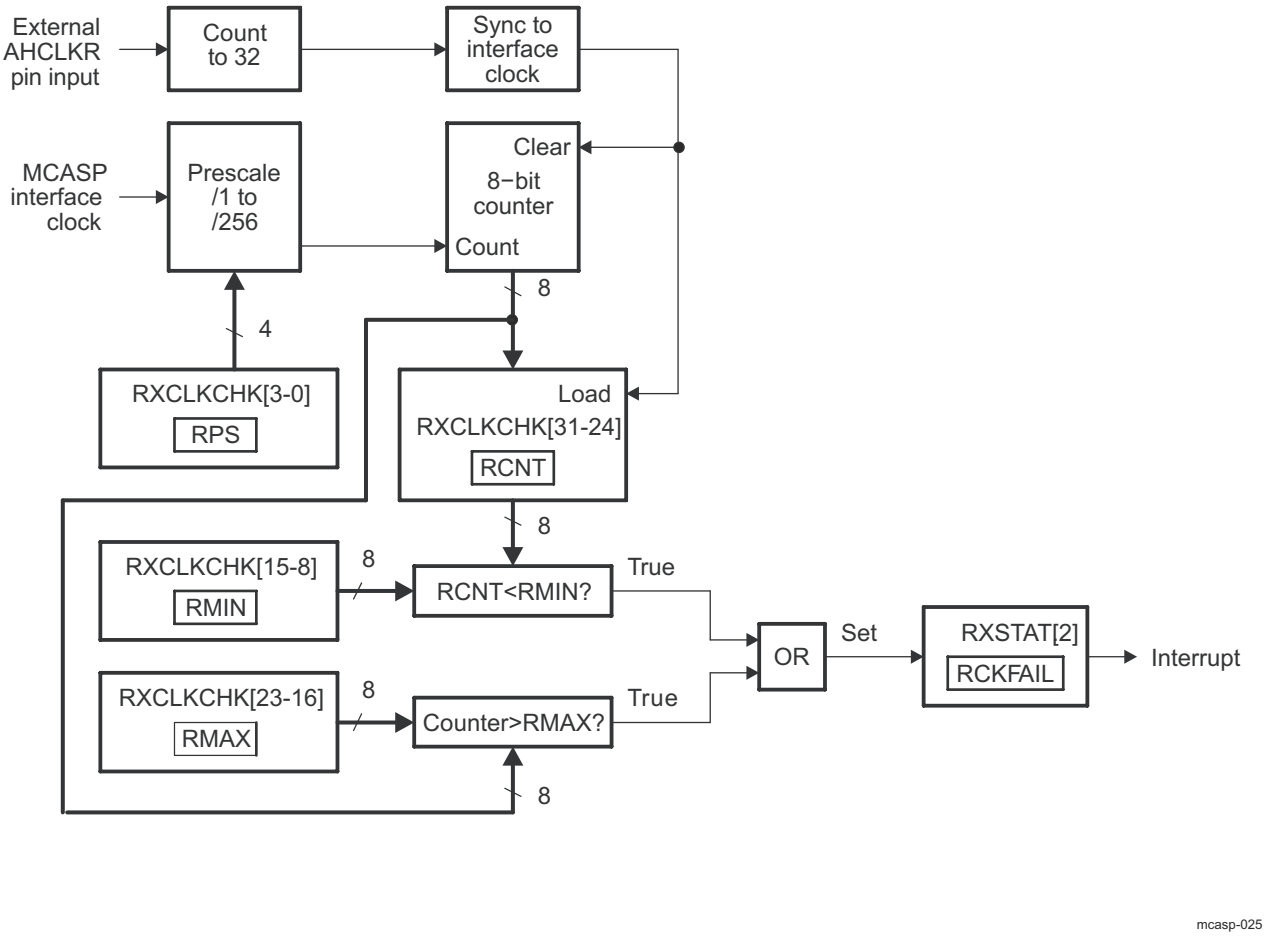


Figure 12-280. Receive Clock Failure Detection Circuit Block Diagram

Copyright © 2024 Texas Instruments Incorporated

12.5.2.4 MCASP Programming Guide

This section describes the low-level hardware programming sequences for the configuration and use of the MCASP module.

12.5.2.4.1 MCASP Operational Modes Configuration

12.5.2.4.1.1 MCASP Transmission Modes

12.5.2.4.1.1.1 Main Sequence – MCASP DIT- /TDM- Polling Transmission Method

Figure 12-281 shows the MCASP DIT-/TDM- polling method.

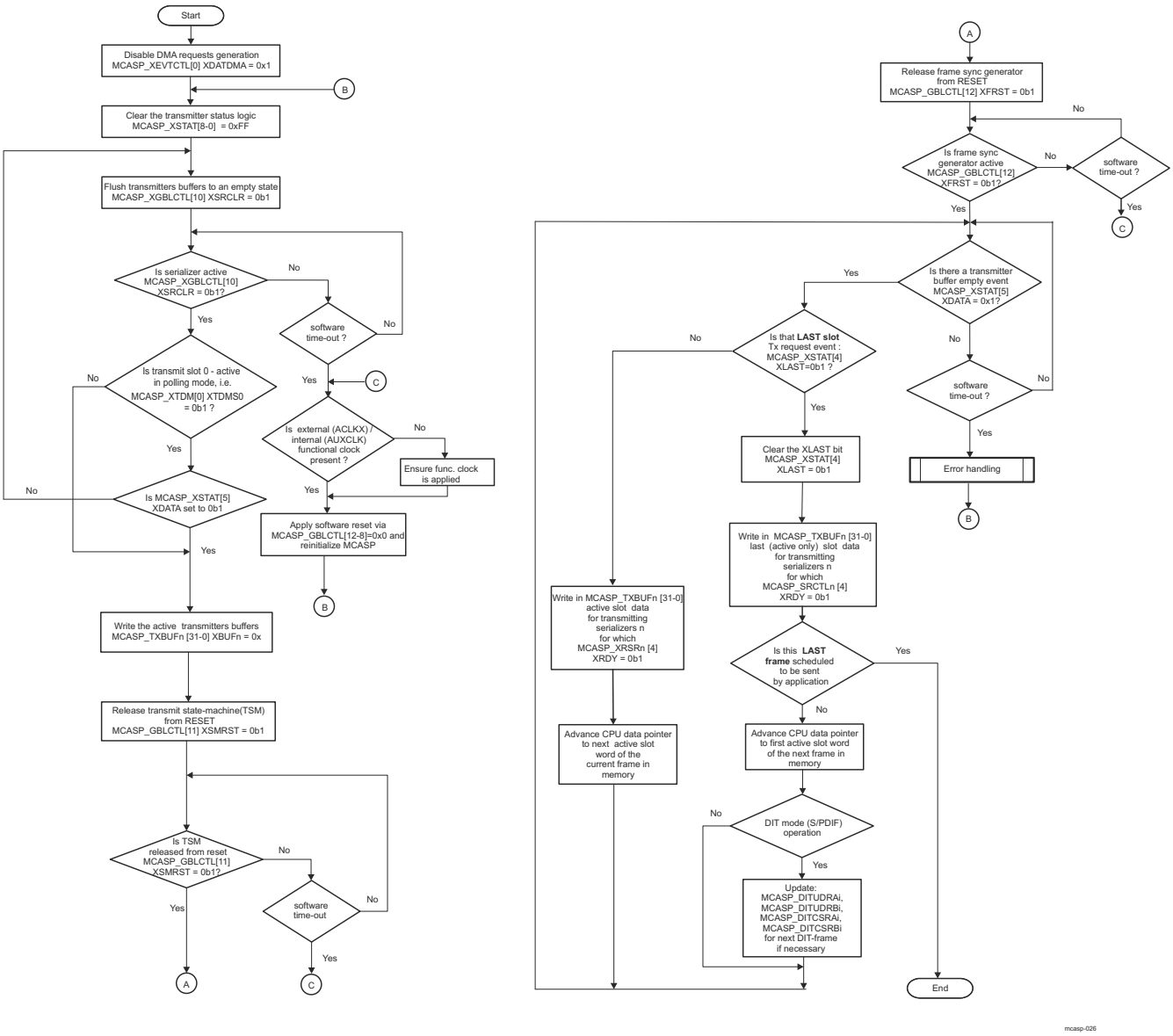


Figure 12-281. MCASP DIT- /TDM- Transmission Polling Method

These registers are for MCASP DIT-/TDM- transmission polling method: MCASP_XEVTCTL, MCASP_XSTAT, MCASP_GBLCTL, MCASP_XTDM, MCASP_XBUFn, MCASP_SRCTLn, MCASP_DITUDRAi, MCASP_DITUDRBi, MCASP_DITCSRAi, MCASP_DITCSRBi (n = 0 to 15 and i = 0 to 5).

Table 12-315 summarizes the subprocess call for the DIT-/TDM- transmission polling mode.

Copyright © 2024 Texas Instruments Incorporated

Table 12-315. Subprocess Call Summary for Main Sequence – MCASP DIT-/TDM- Transmission Polling Method

Subprocess Name	Cross-Reference
Error handling	Figure 12-287

12.5.2.4.1.1.2 Main Sequence – MCASP DIT- /TDM - Interrupt Transmission Method

Figure 12-282 shows the initial setup for interrupt-based transmission.

Copyright © 2024 Texas Instruments Incorporated

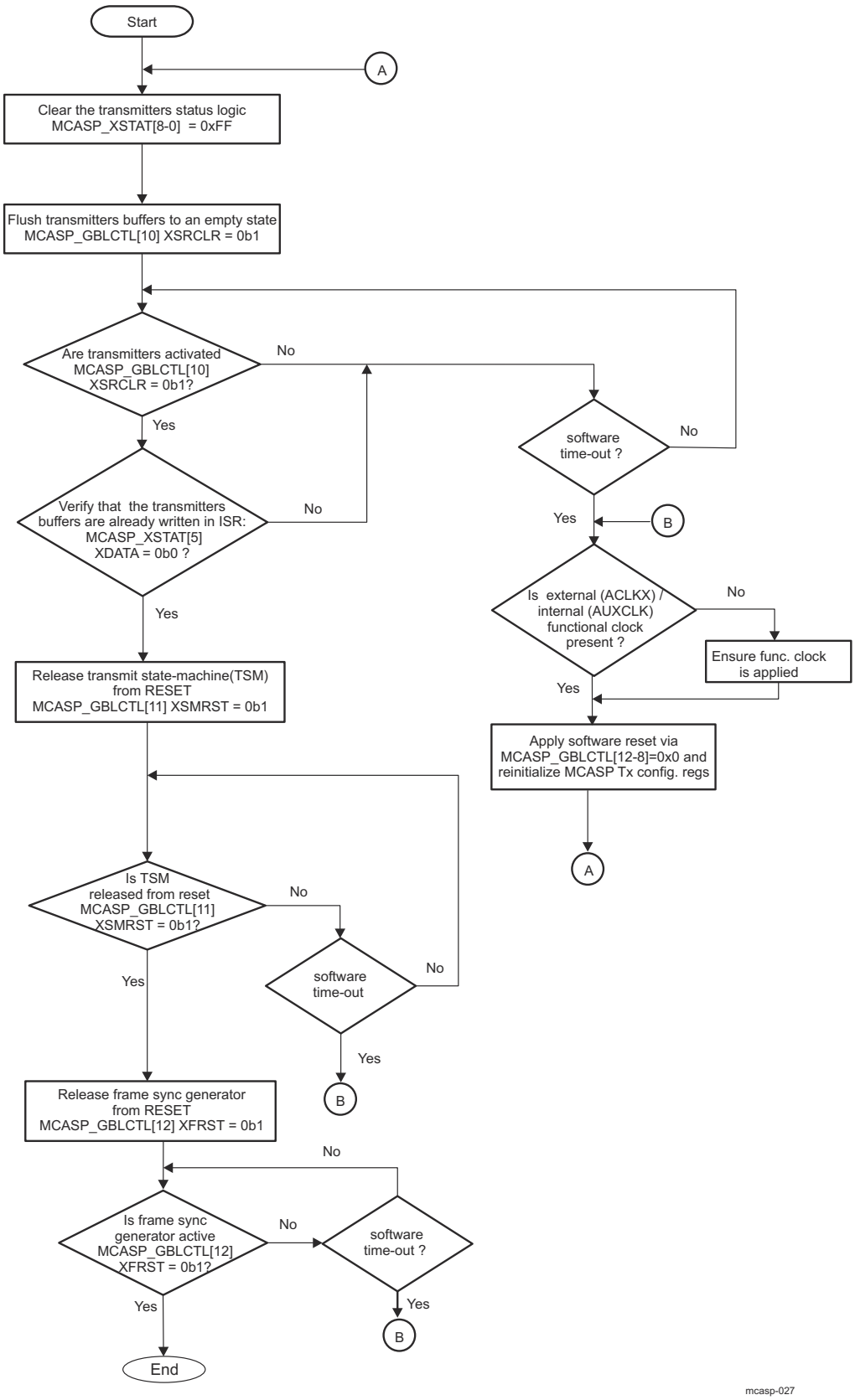


Figure 12-282. Subsequence – DIT-/TDM- Transmission Startup Procedure

Table 12-316 shows the configuration of the MCASP using an interrupt method for DIT-/TDM- transmission.

Copyright © 2024 Texas Instruments Incorporated

Table 12-316. MCASP DIT-/TDM- Interrupt Transmission Model

Step	Register/Bit Field/Programming Model	Value
Disable Tx DMA requests generation.	MCASP_XEVTCTL[0] XDATDMA	0x1
Enable the data ready event transmit interrupt.	MCASP_XINTCTL[5] XDATA	0x1
Optional: Enable the transmit error event interrupts.	MCASP_XINTCTL[2] XCKFAIL MCASP_XINTCTL[1] XSYNCERR MCASP_XINTCTL[0] XUNDRN	0x1 0x1 0x1
Optional: Enable the start of frame interrupt. Optional: Enable the last slot data interrupt (useful for DIT user data/ channel status next S/PDIF frame info update).	MCASP_XINTCTL[7] XSTAFRM MCASP_XINTCTL[4] XLAST	0x1 0x1
IFwrite transfer is through the MCASP DATA port (MCASP_XFMT[3] XBUSEL is set to 0b0).	Software test condition (setting is done in step4 of the *MCASP Transmitters Global Initialization* - see *MCASP Transmitters Global Initialization for DIT-Mode Operation*)	
Enable the DATA port error based interrupt.	MCASP_XINTCTL[3] XDMAERR	0x1
ELSE		
Disable the DATA port error based interrupt.	MCASP_XINTCTL[3] XDMAERR	0x0
ENDIF		
DIT/TDM - Transmission Startup Procedure	See Figure 12-282.	

These registers are for MCASP DIT-/TDM- transmission startup procedure: MCASP_GBLCTL, MCASP_XSTAT.

12.5.2.4.1.1.3 Main Sequence –MCASP DIT- /TDM - Mode DMA Transmission Method

Table 12-317 shows the configuration of the ↓ using the DMA method for transmission. Possible interrupt error event servicing is also considered. shows the initial setup for DMA - based transmission.

Note

Because of the DATA port burst access capability with the DMA method, it is strongly recommended that DMA transfers are initiated through the MCASP DATA port.

Table 12-317. MCASP DMA Transmission Model with Interrupt Events Servicing

Step	Register/Bit Field/Programming Model	Value
Recommended: Select DATA port to access the transmit buffers.	MCASP_XFMT[3] XBUSEL	0x0
Enable the Tx DMA requests generation.	MCASP_XEVTCTL[0] XDATDMA	0x0
Enable the Tx DMA error event, because of MCASP DATA port usage.	MCASP_XINTCTL[3] XDMAERR	0x1
Optional: Enable the transmit error event interrupts.	MCASP_XINTCTL[2] XCKFAIL MCASP_XINTCTL[1] XSYNCERR MCASP_XINTCTL[0] XUNDRN	0x1 0x1 0x1
Optional: Enable the start of frame interrupt. Optional: Enable the last slot data interrupt.	MCASP_XINTCTL[7] XSTAFRM MCASP_XINTCTL[4] XLAST	0x1 0x1
Disable the data ready event transmit interrupt, as DMA is used to service this request.	MCASP_XINTCTL[5] XDATA	0x0
DMA startup transmission procedure. This procedure is identical than the one shown in Figure 12-282. The only difference is that DMA automatically services all the XINT events raised by the MCASP, and no CPU data processing intervention is required. The CPU is involved only in error handling shown in Figure 12-285.	See Figure 12-282.	

12.5.2.4.1.2 MCASP Reception Modes

12.5.2.4.1.2.1 Main Sequence – MCASP Polling Reception Method

Figure 12-283 shows the MCASP polling reception method.

Copyright © 2024 Texas Instruments Incorporated

Note

The MCASP polling reception model considers thedevice CPUs as the accessor of audio data from the MCASP receive buffers.

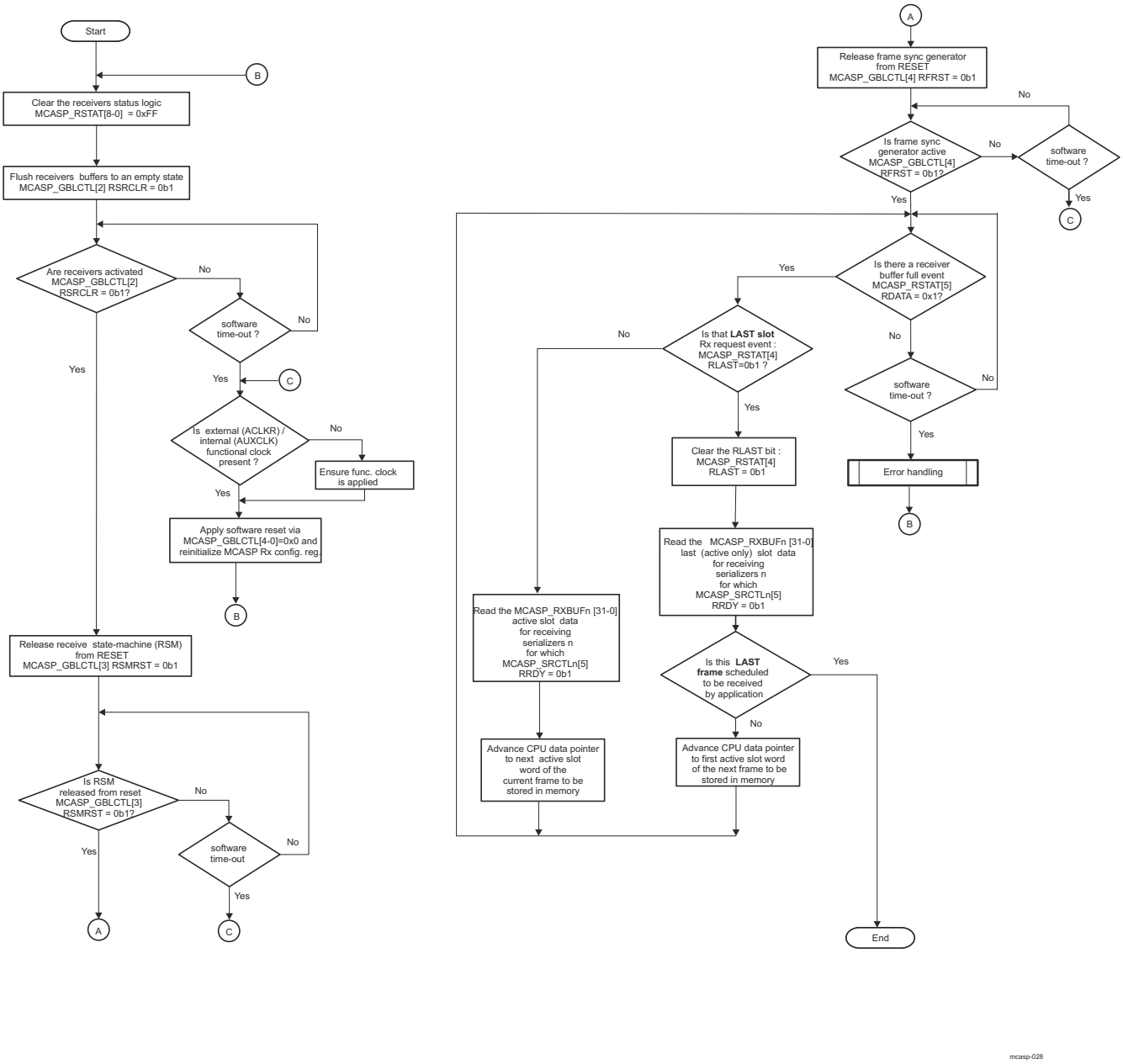


Figure 12-283. MCASP Polling Reception Method

These registers are for MCASP reception polling method: MCASP_RSTAT, MCASP_XGBLCTL, MCASP_RBUFn, MCASP_SRCTLn (n = 0 to 15).

Copyright © 2024 Texas Instruments Incorporated

Table 12-318 summarizes the subprocess call for the polling mode.

Table 12-318. Subprocess Call Summary for Main Sequence – MCASP Reception Polling Method

Subprocess Name	Cross-Reference
Error handling	Figure 12-288

12.5.2.4.1.2.2 Main Sequence – MCASP TDM - Interrupt Reception Method

Figure 12-284 shows the initial setup for interrupt-based reception.

Copyright © 2024 Texas Instruments Incorporated

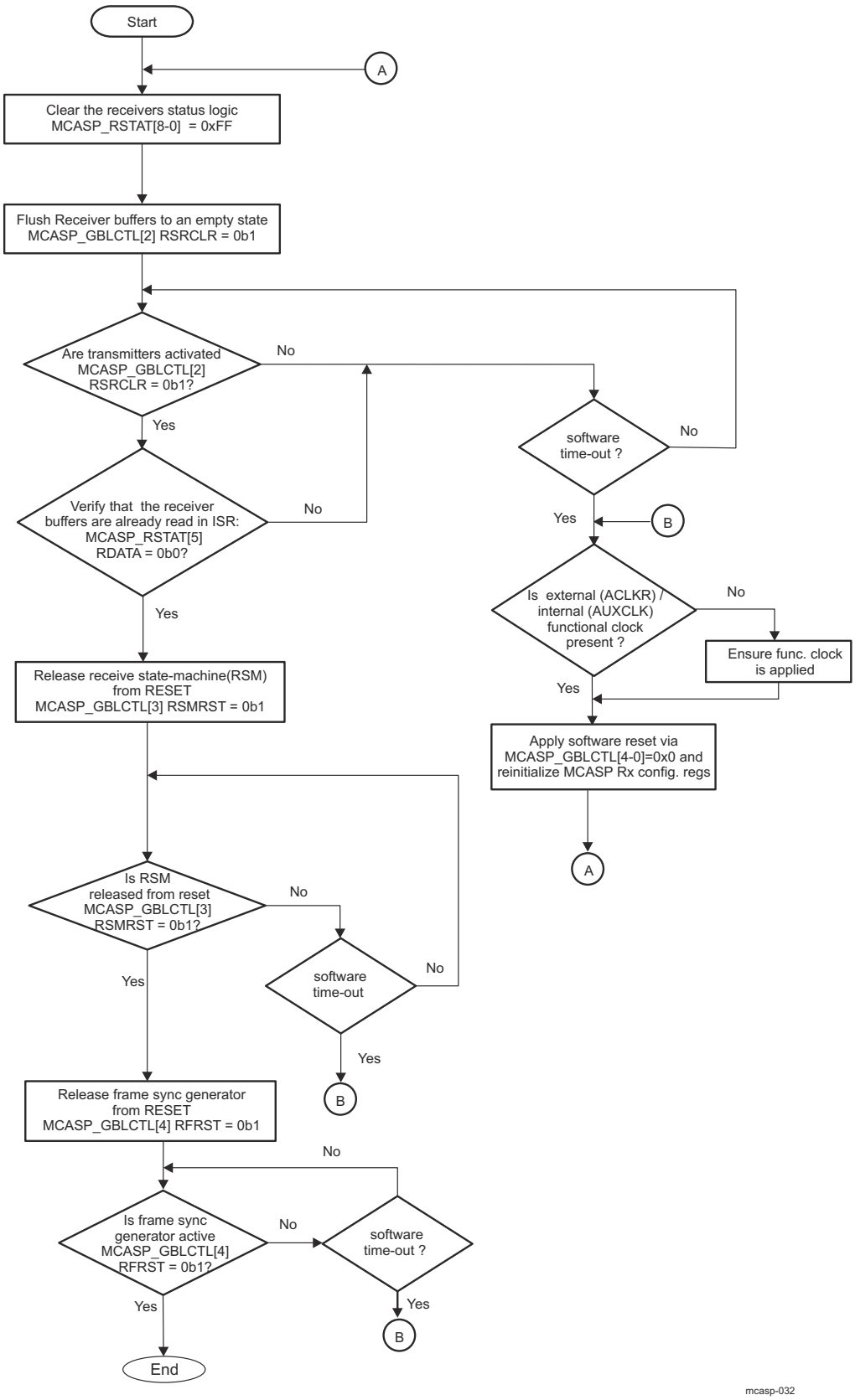


Figure 12-284. Subsequence – TDM - Reception Startup Procedure

Table 12-319 shows the configuration of the MCASP using an interrupt method for TDM- reception.

Copyright © 2024 Texas Instruments Incorporated

Table 12-319. MCASP TDM- Interrupt Reception Model

Step	Register/Bit Field/Programming Model	Value
Disable Rx DMA requests generation.	MCASP_PIDTCTL[0] RDATDMA	0x1
Enable the data ready event receive interrupt.	MCASP_RINTCTL[5] RDATA	0x1
Optional: Enable the receive error event interrupts.	MCASP_RINTCTL[2] RCKFAIL MCASP_RINTCTL[1] RSYNCERR MCASP_RINTCTL[0] ROVRN	0x1 0x1 0x1
Optional: Enable the start of frame interrupt. Optional: Enable the last slot data interrupt	MCASP_RINTCTL[7] RSTAFRM MCASP_RINTCTL[4] RLAST	0x1 0x1
IF read transfer is through the MCASP DATA port (MCASP_RFMT[3] RBUSEL is set to 0b0).	Software test condition (setting is done in step4 of the *MCASP Receivers Global Initialization for TDM-Mode Operation* - see *MCASP Receivers Global Initialization for TDM-Mode Operation*)	
Enable the DATA port error based interrupt.	MCASP_RINTCTL[3] RDMAERR	0x1
ELSE		
Disable the DATA port error based interrupt.	MCASP_RINTCTL[3] RDMAERR	0x0
ENDIF		
TDM - Transmission Startup Procedure	See Figure 12-284.	

These registers are for MCASP TDM- interrupt reception model: MCASP_XGBLCTL, MCASP_RSTAT.

12.5.2.4.1.2.3 Main Sequence – MCASP TDM - Mode DMA Reception Method

Table 12-320 shows the configuration of the MCASP using the DMA method for reception. Possible interrupt error event servicing is also considered. shows the initial setup for DMA - based transmission.

Note

Because of the DATA port burst access capability with the DMA method, it is strongly recommended that DMA transfers are initiated through the MCASP DATA port.

Table 12-320. MCASP DMA Reception Model with Interrupt Events Servicing

Step	Register/Bit Field/Programming Model	Value
Recommended: Select DATA port to access the transmit buffers.	MCASP_RFMT[3] RBUSEL	0x0
Enable the Rx DMA requests generation.	MCASP_PIDTCTL[0] RDATDMA	0x0
Enable the Rx DMA error event, because of MCASP DATA port usage.	MCASP_RINTCTL[3] RDMAERR	0x1
Optional: Enable the receive error event interrupts.	MCASP_RINTCTL[2] RCKFAIL MCASP_RINTCTL[1] RSYNCERR MCASP_RINTCTL[0] ROVRN	0x1 0x1 0x1
Optional: Enable the start of frame interrupt. Optional: Enable the last slot data interrupt.	MCASP_RINTCTL[7] RSTAFRM MCASP_RINTCTL[4] RLAST	0x1 0x1
Disable the data ready event receive interrupt, as DMA is used to service this request.	MCASP_RINTCTL[5] RDATA	0x0
DMA startup reception procedure. This procedure is identical than the one shown in Figure 12-284. The only difference is that DMA automatically services all the RINT events raised by the MCASP, and no CPU data processing intervention is required. The CPU is involved only in error handling shown in Figure 12-286.	See Figure 12-284.	

12.5.2.4.1.3 MCASP Event Servicing

12.5.2.4.1.3.1 MCASP DIT-/TDM- Transmit Interrupt Events Servicing

Figure 12-285 shows the flow of DIT-/TDM- mode transmit interrupt events servicing for the MCASP module.

Copyright © 2024 Texas Instruments Incorporated

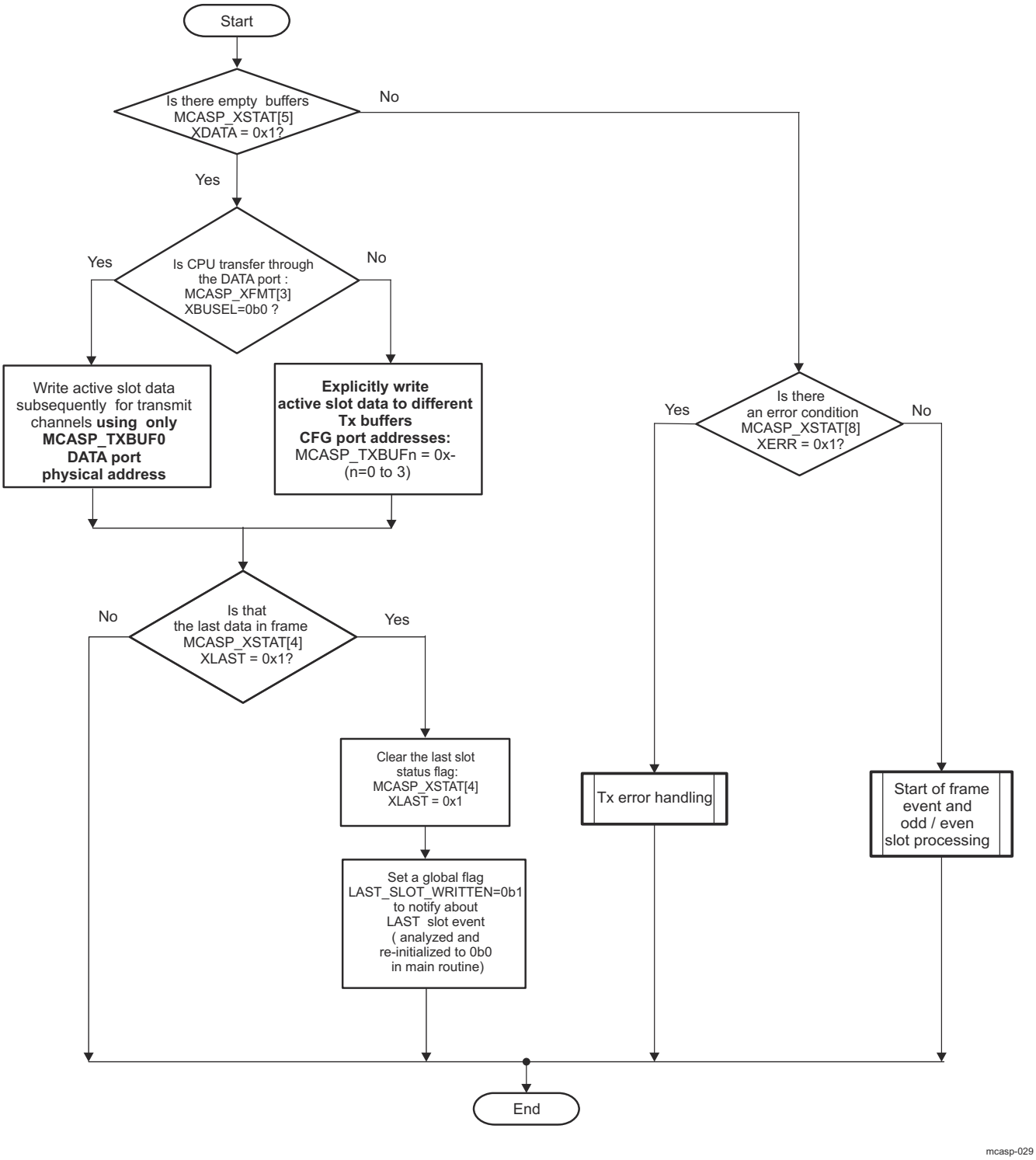


Figure 12-285. MCASP Transmit Interrupt Events Servicing

12.5.2.4.1.3.2 MCASP TDM- Receive Interrupt Events Servicing

Figure 12-286 shows the flow of DIT-/TDM- mode transmit interrupt events servicing for the MCASP module.

Copyright © 2024 Texas Instruments Incorporated

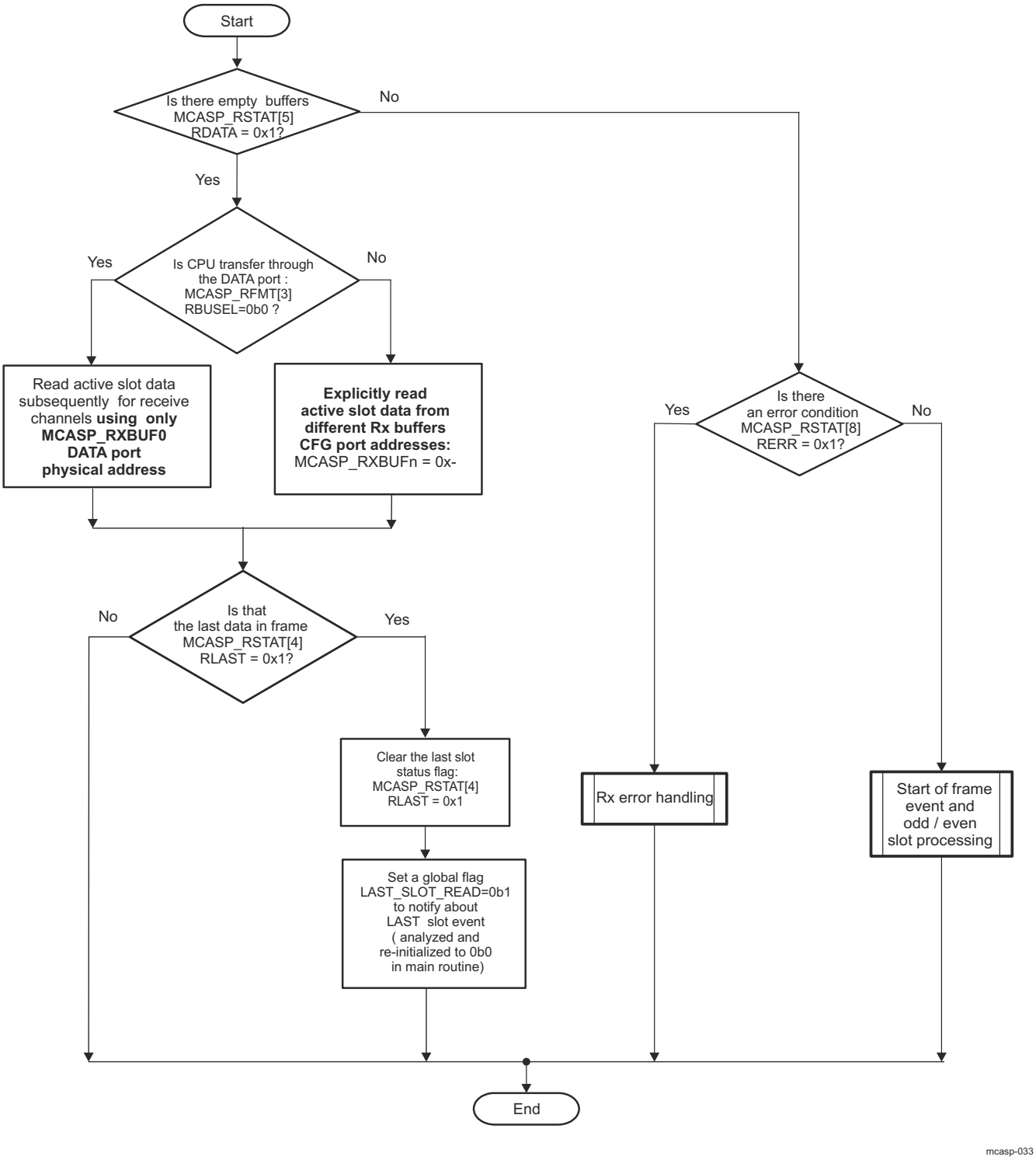


Figure 12-286. MCASP Receive Interrupt Events Servicing

These registers are for MCASP receive interrupt events servicing: MCASP_RSTAT, MCASP_RBUFn, MCASP_RFMT (n = 0 to 15).

Table 12-321 lists the subprocess call summary for receive interrupt events servicing.

Table 12-321. Subprocess Call Summary for Receive Interrupt Events Servicing

Subprocess Name	Cross-Reference
MCASP receive error handling	Figure 12-288

Copyright © 2024 Texas Instruments Incorporated

Table 12-321. Subprocess Call Summary for Receive Interrupt Events Servicing (continued)

Subprocess Name	Cross-Reference
Start of frame handling	Section 12.5.2.3.13.2, *Receive Data Ready Event and Interrupt*

12.5.2.4.1.3.3 Subsequence – MCASP DIT-/TDM -Modes Transmit Error Handling

Figure 12-287 shows the transmit error handling schema for the MCASP, which can be implemented as part of the Tx interrupt service routine or as part of the Tx polling sequence.

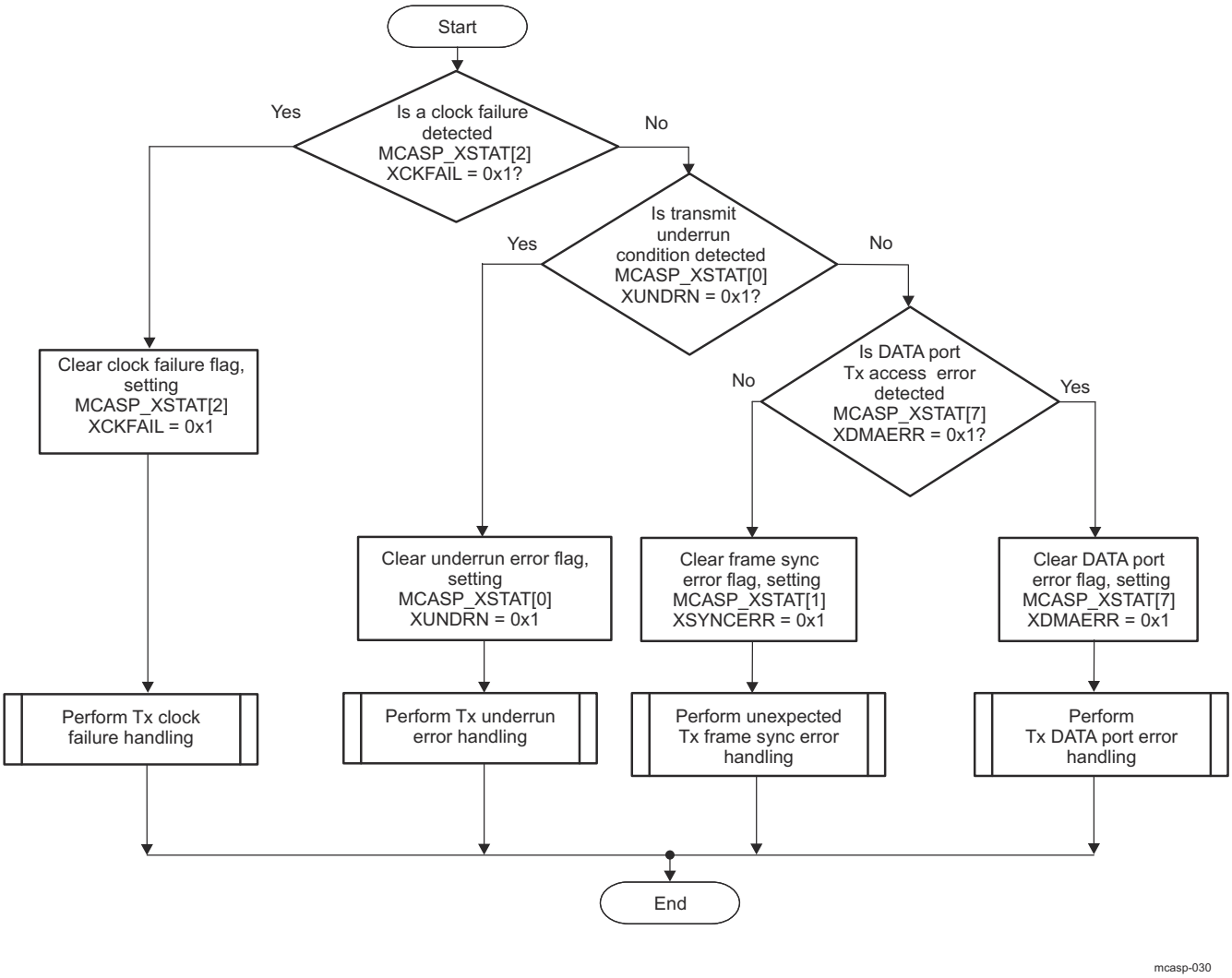


Figure 12-287. MCASP Transmit Error Handling

Note

- For more information about transmit clock failure handling, see Section 12.5.2.3.16.6.2, *Transmit Clock Failure Check and Recovery*.
- For more information about transmit buffer underrun handling, see Section 12.5.2.3.16.1, *Buffer Underrun Error -Transmitter*.
- For more information about DATA port Tx error handling, see Section 12.5.2.3.16.3, *DATA Port Error - Transmitter*.
- For more information about unexpected Tx frame sync error handling, see Section 12.5.2.3.16.5, *Unexpected Frame Sync Error*.

Copyright © 2024 Texas Instruments Incorporated

12.5.2.4.1.3.4 Subsequence – MCASP Receive Error Handling

Figure 12-288 shows the receive error handling schema for the MCASP, which can ONLY be implemented as part of the Rx polling sequence.

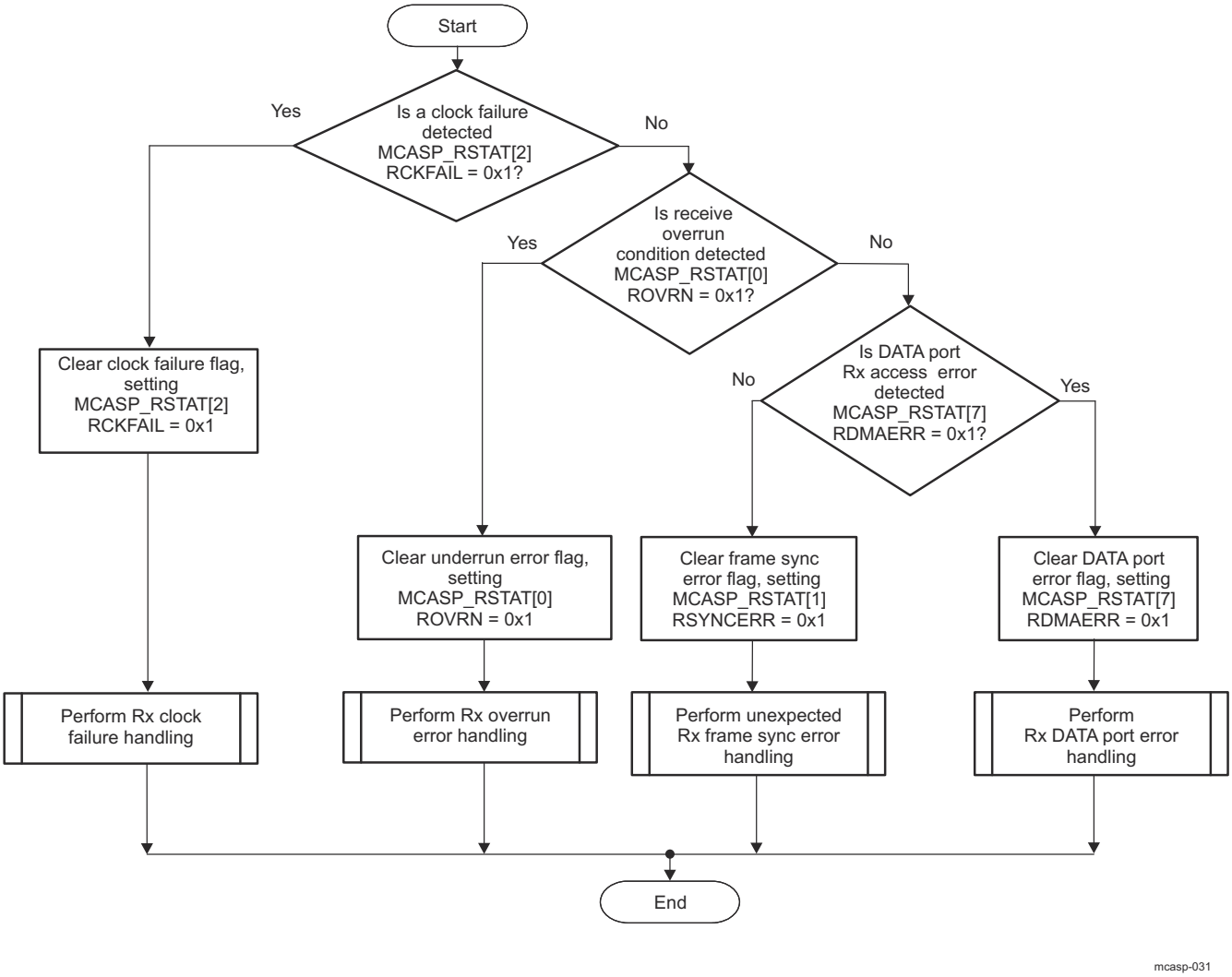


Figure 12-288. MCASP Receive Error Handling

This register is for MCASP receive error handling: MCASP_RSTAT.

Note

- For more information about receive clock failure handling, see Section 12.5.2.3.16.6.3, *Receive Clock Failure Check and Recovery*.
- For more information about receive buffer overrun handling, see Section 12.5.2.3.16.2, *Buffer Overrun Error - Receiver*.
- For more information about DATA port Rx error handling, see Section 12.5.2.3.16.4, *DATA Port Error - Receiver*.
- For more information about unexpected Rx frame sync error handling, see Section 12.5.2.3.16.5, *Unexpected Frame Sync Error*.

Copyright © 2024 Texas Instruments Incorporated

12.6 Display Subsystem (DSS) and Peripherals

This section describes the Display Subsystem (DSS) in the device, integrated together with display peripherals, such as the MIPI Display Serial Interface (DSI) transmitter host controller with associated MIPI DSI (Physical Layer) D-PHY module (DPHY_TX), and the Embedded Display Port (eDP) transmitter with associated SerDes and auxiliary complex I/O (eDP AUXPHY) modules.

12.6.1 DSS Overview

The DSS is a flexible composition-enabled display subsystem, that supports multiple high resolution display outputs. It consists of a Display Controller (DISPC) and a Frame Buffer Decompression Core (FBDC). The DISPC supports a multi-layer blending and transparency for each of its display outputs. The DISPC also supports a write-back pipeline with scaling to enable memory-to-memory composition and/or to capture a display output for Ethernet video encoding. The DISPC supports gamma correction and programmable color control in both source and destination pipelines for enhanced color space control and video output quality. The DISPC video outputs connect to various display interface controllers, integrated at the SoC level, to drive specific displays in the end application. The DSS also includes freeze frame detection and data signature check capabilities for each display output path.

The MIPI DSI v1.3.1 Controller (DSITX) implements the stream arbitration and low-level protocol layer functionalities required by MIPI DSI 1.3 standard. It supports up to 4 x 2.5 Gbps D-PHY data lanes in a single-link configuration and handles the byte lane mapping per use case (1, 2, 3, or 4-lanes). The accompaning DSI (Physical Layer) D-PHY module (DPHY_TX) provides the video output interfacing by implementing a four-lane MIPI D-PHY transmitter.

The VESA DP1.4/eDP1.4 Compliant Transmitter Host Controller (EDP) can output up to 4 video streams (through Multiple Stream Transport / MST) and one audio stream through the 4-lane accompaning SerDes module. It provides up to 25.92 Gbps of application bandwidth. An additional auxiliary PHY (AUXPHY) module implements a doubly-terminated differential pair required for 1 Mbps data rates over a long (15m) cable. This is a half duplex, Manchester II encoded connection, used by the EDP to configure the link. The AUX channel is used to access the Display Port Configuration Data (DPCD) memory area in the sink device and to implement VESA Enhanced Extended Display Identification Data (EDID) function.

12.6.1.1 DSS Features

The Display Controller (DISPC) supports:

- An embedded DMA Controller with the following features:
  - Support for 1D-only DMA transfers
  - Support for 48b addressable memory space
  - Support for memory fragmentation through external PAT (Page Address Translation) table at SoC level
  - Integrated shared buffer management for pipelines within the same DMA controller group
    - Support for up to 8K-pixels wide frame buffer for 8/16/32/64 bits per pixel (total buffer space requirement not exceeding the total DMA buffer size of the DISPC)
  - Programmable DMA request management
    - Programmable buffer thresholds for request priority
    - Bandwidth limiter on write requests (insertion on idle cycles between requests)
    - Self-refresh using the DMA buffers
    - Arbitration between normal/low priority pipelines
  - Support for source image flip along X and Y-axis
  - Support for secure access to firewall protected frame buffer in DDR memory
- Two input display processing Video Pipelines, each supporting:
  - Input RGB source pixel formats:
    - Bitmap-1, Bitmap-2, Bitmap-4, Bitmap-8
    - ARGB16-4444, ABGR16-4444, RGBA16-4444, RGB16-565, BGR16-565, ARGB16-1555, ABGR16-1555
    - RGB16-565/BGR16-565 with a separate A8 plane
    - RGB24-888, BGR24-888

Copyright © 2024 Texas Instruments Incorporated

    - ARGB32-8888, ABGR32-8888, BGRA32-8888, RGBA32-8888, ARGB32-2101010, ABGR32-2101010
    - ARGB64-16161616, RGBA64-16161616
    - Additionally, equivalent RGBx, xRGB, xBGR, and BGRx pixel formats defined, considering that A component of RGBA, ARGB, ABGR, BGRA pixel formats is ignored by HW (for example, ARGB —> xRGB); ("A" can be ignored by not selecting Alpha pixel)
    - Pre-multiplied ARGB/RGBA formats
  - Input YUV source pixel formats:
    - Packed: YUV422-UYVY, YUV422-YUV2
    - 2-plane: YUV420-NV12, YUV420-NV21, YUV422-NV12, YUV422-NV21
    - 8-bit per component support for all YUV formats
    - 10-bit IMG pack-format and 12-bit TI fully packed format support for 420/422 (internally processed as 10-bit component data)
    - 16-bit unpacked source format (internally processed as 10-bit component data)
  - Programmable poly-phase filter (scaler):
    - Independent horizontal and vertical re-sampling: up-sampling (up to x16) and down-sampling (down to x0.25)
    - Maximum input width of 2048 pixels using ARGB pixels and 5-tap, 4096 pixels using YUV pixels and 5-tap, and 4096 pixels using ARGB pixels and 3-tap. No limitation on the input height.
    - Alpha blending factor is re-scaled like the R, G and B color components
    - Implementation with 16 phases with symmetrical coefficients
  - Programmable color space conversion from YUV422/YUV420 (chroma upsampled to YUV444 using the scaler) into ARGB48-12121212
  - Programmable VC1 range mapping
  - Programmable Brightness/Contrast/Hue/Saturation
  - Programmable Gamma Correction LUT
  - Luma Key generation
  - 10-bit processing pipeline
- Two input display processing Video Lite Pipelines, each supporting:
  - Input RGB source pixel formats:
    - Bitmap-1, Bitmap-2, Bitmap-4, Bitmap-8
    - ARGB16-4444, ABGR16-4444, RGBA16-4444, RGB16-565, BGR16-565, ARGB16-1555, ABGR16-1555
    - RGB16-565/BGR16-565 with a separate A8 plane
    - RGB24-888, BGR24-888
    - ARGB32-8888, ABGR32-8888, BGRA32-8888, RGBA32-8888, ARGB32-2101010, ABGR32-2101010
    - ARGB64-16161616, RGBA64-16161616
    - Additionally, equivalent RGBx, xRGB, xBGR, and BGRx pixel formats defined, considering that A component of RGBA, ARGB, ABGR, BGRA pixel formats is ignored by HW (for example, ARGB —> xRGB); ("A" can be ignored by not selecting Alpha pixel)
    - Pre-multiplied ARGB/RGBA formats
  - Input YUV source pixel formats:
    - Packed: YUV422-UYVY, YUV422-YUV2
    - 2-plane: YUV420-NV12, YUV420-NV21, YUV422-NV12, YUV422-NV21
    - 8-bit per component support for all YUV formats
    - 10-bit IMG pack-format and 12-bit TI fully packed format support for 420/422 (internally processed as 10-bit component data)
    - 16-bit unpacked source format (internally processed as 10-bit component data)
  - YUV420 to YUV422 chroma upsampling using an average filter
  - YUV422 to YUV444 chroma upsampling using a 4-tap filter based on Catmull-Rom algorithm
  - Programmable color space conversion from YUV422/YUV420 into ARGB48-12121212
  - Programmable VC1 range mapping
  - Programmable Brightness/Contrast/Hue/Saturation
  - Programmable Gamma Correction LUT
  - Luma Key generation

Copyright © 2024 Texas Instruments Incorporated

  - 10-bit processing pipeline
- One Write-back (WB) pipeline, supporting:
  - Destination RGB pixel formats:
    - ARGB16-4444, ABGR16-4444, RGBA16-4444, RGB16-565, BGR16-565, ARGB16-1555, ABGR16-1555
    - RGB24-888 (no support for BGR24-888)
    - RGB16-565/BGR16-565 with a separate A8 plane
    - ARGB32-8888, ABGR32-8888, BGRA32-8888, RGBA32-8888, ARGB32-2101010, ABGR32-2101010
    - ARGB64-16161616, RGBA64-16161616
  - Destination YUV pixel formats:
    - Packed: YUV422-UYVY, YUV422-YUV2
    - 2-plane: YUV420-NV12, YUV420-NV21, YUV422-NV12, YUV422-NV21
    - Only 8-bit output support for YUV formats
  - Programmable poly-phase filter (scaler):
    - Independent horizontal and vertical re-sampling: up-sampling (up to x16) and down-sampling (down to x0.25). When the output format of the WB pipeline includes a format change RGB/YUV422 —> YUV420, the maximum downscaling provided by the WB scaler is x0.5.
    - Maximum input width of 2048 pixels using ARGB pixels and 5-tap, 4096 pixels using YUV pixels and 5-tap, and 4096 pixels using ARGB pixels and 3-tap. No limitation on the input height.
    - Alpha blending factor is re-scaled like the R, G and B color components
    - Implementation with 16 phases with symmetrical coefficients
  - Operation modes:
    - Output capture mode (to save one of the display outputs)
    - Memory-to-memory (M2M) mode (to save a M2M composition/conversion operation result)
- Four Overlay Managers (OVR), each supporting:
  - Input pixel format: ARGB48-12121212
  - Output pixel format: ARGB48-12121212 ("A" component data is only used for the Write-back path)
  - Overlay of the input pipelines (fully mapped to all input pipelines)
  - Up to 4 input layers blending, plus background layer
  - Transparency color key (source and destination)
  - Alpha blending support: Embedded pixel alpha (ARGB and RGBA), global pixel, and combination of global pixel and pixel alpha
  - Z-order programmable (full flexibility)
  - Color bar test pattern insertion
  - Any overlay output can be selected to drive the Write-back pipeline
- Four Video Port (VP) display outputs, each supporting:
  - 36-bit per pixel on the RGB output interface (12-bits per component)
  - Independent programmable timing generator, supporting up to 600 MHz pixel clock video formats (actual support limited by the maximum pixel clock provided to DSS at SoC level)
  - Independent programmable 10-bit gamma correction
  - Independent programmable multiple cycles output format on 8/9/12/16-bit interface (TDM)
  - Selection between RGB and YUV422 output pixel format (YUV422 only available when BT.656/BT.1120 output is enabled)
  - Configurable VP output mode: progressive mode only on RGB output, or progressive/interlaced mode on BT.656/BT.1120 output
- Internal data check diagnostic features:
  - Supports up to 4 programmable (position/size) check regions on the DISPC video port display outputs
  - Support for 1 check region on each input video pipeline output
  - MISR (Multiple Input Signature Register) used on each check region to perform data correctness check and/or freeze frame detection
- Local power management features:
  - Low power saving modes
  - Capability to associate all buffers with a single pipeline for a display self-refresh
- System interconnect ports:

Copyright © 2024 Texas Instruments Incorporated

  - Two 128-bit VBUSM master interfaces for data read/write
  - One 32-bit VBUSP slave interfaces for configuration

MIPI Display Serial Interface (DSI) transmitter host controller supports:

- Compliance with MIPI DSI v1.3.1 protocol specification and previous specifications
- Compliance with MIPI Stereoscopic Display Formats (MIPI SDF v1.0) specification
- Video and command operational modes
- Both burst and non-burst modes for video mode data transmission (with either sync pulses or sync events)
- Up to 4 virtual channels via command mode
  - Supports data interleaving support for one synchronous stream (video mode) from the display controller, and up to three interleaved asynchronous streams (command mode) from the interconnect concurrently
  - Supports data interleaving for up to four interleaved asynchronous streams (virtual channels in command mode) from the interconnect
- Bi-directional communication and escape mode (only on Lane-0)
- Pixel clock rate range 25-330 MHz (10-bit/component, 4MPix @60fps performance). Actual performance is limited by the maximum pixel clock provided to the DSI controller at SoC level.
- Programmable display resolutions with maximum clock rate not exceeding the total available bandwidth over 4MPix @60fps (10-bit/component RGB)
- 16/18/24/30/36-bit RGB input data formats for video mode
- RGB16, RGB18 packed, and RGB24 input data formats for command mode
- All generic data types defined by MIPI
- Display Command Set (DCS): transparent to the protocol engine, no decoding and interpretation of the information from and to the peripheral
- ECC on the APB interface
- Data splitter for 2-,3-, or 4-data lane configuration
- Connection to a single MIPI D-PHY complex I/O through an 8-bit Protocol Peripheral Interface (PPI)
- Bus contention recovery
- Video mode pattern generator: color bar pattern image and D-PHY BET testing pattern
- APB slave interface with 32-bit data and address for configuration

MIPI DSI (Physical Layer) D-PHY module supports:

- Compliance with MIPI D-PHY 1.2 physical layer interface specification and features
- 1, 2 or 4 data lanes, in addition to clock signaling
- Maximum data rate up to 2.5 Gbps per data lane
- Protocol Peripheral Interface (PPI)
- HS continuous and burst mode
- LP (Low-Power), ULPM (Ultra-Lower Power Mode), and Shutdown modes
- Forward direction and reverse direction escape modes (only on Lane-0)
- Automatic termination control in both high-speed and low-power modes
- DSI D-PHY IO pad signals work at/with electrical specification specified by requested standards
- Single 32-bit VBUSP slave interface

The Embedded Display Port (eDP) transmitter host controller supports:

- Compliance with VESA Display Port (DP) 1.3 (with 1.4 DSC/FEC support) specification
- Compliance with VESA embedded Display Port (eDP) 1.4 specification
- Static configuration of either DP or eDP mode
- Link rates up to HBR3 (maximum application bandwidth up to 25.92Gbps)
- Pixel clock rate range 25-600 MHz (10-bit/component, 8MPix@60fps performance).
- Transceiver AUX CH (1 MHz Manchester II coding mode) for access of DPCD and EDID
- 8, 10, and 12 bpc (bits per component), in RGB/YCbCr444 colorimetry formats (CEA-861 compliant) and YCbCr422 (via simple decimation)
- Data splitter for 1-, 2-, or 4-data lane configuration
- SST (Single Stream Transport)
- MST (Multiple Stream Transport):
  - Up to 4 video and up to 1 audio sources

Copyright © 2024 Texas Instruments Incorporated

  - Support for up to 25 Gbps throughput (equivalent to approximately 4K + 2xFHD streams) use case
  - HDCP support on one video source
  - DSC (Display Stream Compression) support on one 4K or 2x2.5K streams
- HDCP 1.4 and HDCP 2.2 with True Random Number Generator (TRNG with 8 FRO)
- DSC (Display Stream Compression) encoded stream data transport via an embedded DSC core:
  - Supports VESA DSC 1.1 compliant video compression at 2x~3x compression ratios
  - Supports all DSC 1.1 mandatory encoding mechanisms (MMAP, BP, MPP and ICH)
  - Supports configurable maximum display resolution up to 8Mpix @60fps including (but not limited to) 4K@60 (4096x2160), and up to 8K wide-aspect-ratio resolution displays
  - Supports two hard encoder slices (peak pixel ratio <= 340MP/s on each slice)
  - Supports 8 and 10 bits per video component
  - Supports RGB/YCbCR4:4:4 video input format (Native Encoding) only
  - Supports Dual pixel streams or Split panel input streams
  - Supports dual or single transport link
- Forward Error Correction (FEC) encoder with/without DSC enabled in DP mode
- Various eDP specific features:
  - eDP DPCD registers
  - Full and Fast Link Training
  - (Regional) backlights and multi-touch command over AUX channel
  - Alternate Scrambler Reset
- Audio transport features:
  - I2S (LPCM/IEC60948/619376) audio input stream
  - Audio transport of uncompressed multichannel (up to eight channels) via SDP (Secondary-data Packet)
- Metadata transport via MSA (Main Stream Attribute) packet or via SDP
- Display Port transmitter functionalities:
  - Scrambler
  - 8/10-bit Encoder (in the DPTX core)
  - Inter Lane Skew Insertion
  - Training Pattern Generation – TPS1,2,3,4 PRBS7 and 80-bit custom training pattern generation (bypassing the scrambler and encoder)
- PAPB Interface:
  - The primary APB slave port controls the HD Display controller from the host processor
  - During the boot, the primary APB slave port enables access to the I-MEM and D-MEM, and to the full address space for debugging purposes
  - After the boot, the primary APB slave port enables direct register access to designated HW modules and enables communication with uCPU through a mailbox channel
- SAPB Interface
  - For configuring the embedded HDCP, this bus is considered as a secured APB to carry secured commands over the mailbox channel
- Internal diagnostic features:
  - ECC on the critical memories
  - Parity check on the configuration interface
  - Encoder self-check diagnostics support in the DSC core
  - Corrected/Uncorrected error interrupt generation
  - Injection of ECC and parity errors

The DP (Physical Layer) SERDES and Aux PHY modules support:

- DP1.3 HBR3 and eDP1.4a HBR3 throughput
- 1, 2, or 4 lanes at 1.62 Gbps, 2.7 Gbps, 5.4 Gbps, and 8.1 Gbps per lane
- Additional link rates (2.16, 2.43, 3.24, 4.32 Gbps) per lane in eDP mode
- Reduced differential voltage swing (0.2/0.25/0.30/0.35/0.40/0.45) in eDP mode
- Hot Plug Detect (HPD) for connection detection and interrupt from sink
- Integrated Low Jitter, Fixed Bandwidth PLL
- 1 Mbps AUX PHY for link training, DPCD register access, HDCP authentication and EDID access.

Copyright © 2024 Texas Instruments Incorporated

12.6.1.2 DSS Ports

Table 12-322. DISPC Clocks and Resets

Clocks	
Module Clock Input	**Description**
DSS_FUNC_CLK	DISPC functional clock.
DSS_DPI_0_PCLK	DISPC peripheral pixel clocks for VP1.
DSS_DPI_0_DIV_PCLK	
DSS_DPI_1_PCLK	DISPC peripheral pixel clock for VP2.
DSS_DPI_2_PCLK	DISPC peripheral pixel clocks for VP3.
DSS_DPI_2_DIV_PCLK	
DSS_DPI_3_PCLK	DISPC peripheral pixel clock for VP4.
DSS_DPI_0_OUT_PCLK_2X	DISPC output pixel clocks to display peripherals.
DSS_DPI_0_OUT_PCLK	
DSS_DPI_1_OUT_PCLK_2X	
DSS_DPI_1_OUT_PCLK	
DSS_DPI_2_OUT_PCLK_2X	
DSS_DPI_2_OUT_PCLK	
DSS_DPI_3_OUT_PCLK_2X	
DSS_DPI_3_OUT_PCLK	
Resets	
Module Reset Input	**Description**
DSS_DPI_RST_0	DISPC reset

Table 12-323. DISPC Hardware Requests

Interrupt Requests		
Module Interrupt Signal	**Description**	**Type**
DSS_DSS_INST0_DISPC_FUNC_IRQ_PROC0_0	DISPC functional interrupt request	Level
DSS_DSS_INST0_DISPC_FUNC_IRQ_PROC1_0	DISPC functional interrupt request	Level
DSS_DSS_INST0_DISPC_FUNC_IRQ_PROC2_0	DISPC functional interrupt request	Level
DSS_DSS_INST0_DISPC_FUNC_IRQ_PROC3_0	DISPC0 functional interrupt request	Level
DSS_DSS_INST0_DISPC_SAFETY_ERROR_IRQ_PROC0_0	DISPC internal diagnostic error interrupt request	Level
DSS_DSS_INST0_DISPC_SAFETY_ERROR_IRQ_PROC1_0	DISPC internal diagnostic error interrupt request	Level
DSS_DSS_INST0_DISPC_SAFETY_ERROR_IRQ_PROC2_0	DISPC internal diagnostic error interrupt request	Level
DSS_DSS_INST0_DISPC_SAFETY_ERROR_IRQ_PROC3_0	DISPC internal diagnostic error interrupt request	Level
DSS_DSS_INST0_DISPC_SECURE_IRQ_PROC0_0	DISPC secure interrupt request	Level
DSS_DSS_INST0_DISPC_SECURE_IRQ_PROC1_0	DISPC secure interrupt request	Level
DSS_DSS_INST0_DISPC_SECURE_IRQ_PROC2_0	DISPC secure interrupt request	Level
DSS_DSS_INST0_DISPC_SECURE_IRQ_PROC3_0	DISPC secure interrupt request	Level

Table 12-324. DSI Clocks and Resets

Clocks	
Module Clock Input	**Description**
DSI_SYS_CLK	DSI system clock.
DSI_DPI_0_CLK	DSI DPI clock.
DSI_DPHY_0_TX_ESC_CLK	DSI DPHY clock.
DSI_DPHY_0_RX_ESC_CLK	DSI DPHY clock.

Copyright © 2024 Texas Instruments Incorporated

Table 12-324. DSI Clocks and Resets (continued)

DPHY_CLK	DPHY_TX0 clocks.
DPHY_PSM_CLK	
DPHY_PPI_K_M_TXCLKESCCLK_CL	
DPHY_PPI_K_LN0_M_TXCLKESC_DL	
DPHY_PPI_K_LN1_M_TXCLKESC_DL	
DPHY_PPI_K_LN2_M_TXCLKESC_DL	
DPHY_PPI_K_LN3_M_TXCLKESC_DL	
DPHY_REF_CLK	DPHY_TX0 reference clock.
DPHY_TX_P_CL_L	DSI_DPHY_TX0 input pin clock.
DPHY_TX_M_CL_L	DSI_DPHY_TX0 input pin clock.
Resets	
Module Reset Input	**Description**
DSI_RST_0	DSI reset.

Table 12-325. DSI Hardware Requests

Interrupt Requests		
Module Interrupt Signal	**Description**	**Type**
DSS_DSI_DSI_0_FUNC_INTR_0	DSI functional interrupt request.	Level
DSS_DSI_DSI_0_SAFETY_ERROR_NONFATAL_INTR_0	DSI ASF non-fatal interrupt request.	Level
DSS_DSI_DSI_0_SAFETY_ERROR_FATAL_INTR_0	DSI ASF fatal interrupt request.	Level
DSS_DSI_ECC_INTR_UNCORR_LEVEL_SYS_0	DSI ECC Aggregator uncorrectable error interrupt request.	Level

Table 12-326. EDP Clocks and Resets

Clocks	
Module Clock Input	**Description**
EDP_DPTX_CLK	EDP main clock.
EDP_DPI_2_CLK	EDP video stream pixel clocks.
EDP_DPI_2_2X_CLK	
EDP_DPI_3_CLK	
EDP_DPI_4_CLK	
EDP_DPI_5_CLK	
EDP_AIF_I2S_CLK	EDP audio stream clock.
PHY_LN0_REFCLK	EDP lane input clocks.
PHY_LN0_TXFCLK	
PHY_LN0_TXMCLK	
PHY_LN0_RXFCLK	
PHY_LN0_RXCLK	
PHY_LN1_REFCLK	
PHY_LN1_TXFCLK	
PHY_LN1_TXMCLK	
PHY_LN1_RXFCLK	
PHY_LN1_RXCLK	
PHY_LN2_REFCLK	
PHY_LN2_TXFCLK	
PHY_LN2_TXMCLK	
PHY_LN2_RXFCLK	
PHY_LN2_RXCLK	
PHY_LN3_REFCLK	

Copyright © 2024 Texas Instruments Incorporated

Table 12-326. EDP Clocks and Resets (continued)

PHY_LN3_TXFCLK	
PHY_LN3_TXMCLK	
PHY_LN3_RXFCLK	
PHY_LN3_RXCLK	
IP1_LN0_TXCLK	SERDES4 TX return clocks.
IP1_LN1_TXCLK	
IP1_LN2_TXCLK	
IP1_LN3_TXCLK	
Resets	
Module Reset Input	**Description**
EDP_RST_0	EDP reset.

Table 12-327. EDP Hardware Requests

Interrupt Requests		
Module Interrupt Signal	**Description**	**Type**
DSS_EDP_INTR_0	EDP interrupt request.	Level
DSS_EDP_INTR_1	EDP interrupt request.	Level
DSS_EDP_INTR_2	EDP interrupt request.	Level
DSS_EDP_INTR_3	EDP interrupt request.	Level
DSS_EDP_INTR_ASF_0	EDP ASF interrupt requests.	Level
DSS_EDP_INTR_ASF_1		Level
DSS_EDP_INTR_ASF_2		Level
DSS_EDP_INTR_ASF_3		Level
DSS_EDP_INTR_ASF_4		Level
DSS_EDP_INTR_ASF_5		Level
DSS_EDP_INTR_ASF_6		Level

12.6.2 DSS Environment

This section describes the interfaces handled by the Display Subsystem.

The DSS is capable of driving multiple displays in parallel, through a combination of interfaces:

- Two DPI parallel interfaces, directly delivered on SoC pads from DISPC Video Port (VP) outputs
- One MIPI Display Serial Interface (DSI)
- One Embedded DisplayPort (DP/eDP) interface

The pixel format mapping between the DISPC VP outputs and DPI, eDP, and DSI display peripherals varies depending on the interface used. The pixel bit color mapping for the DISPC VP outputs is as shown in Figure 12-289.

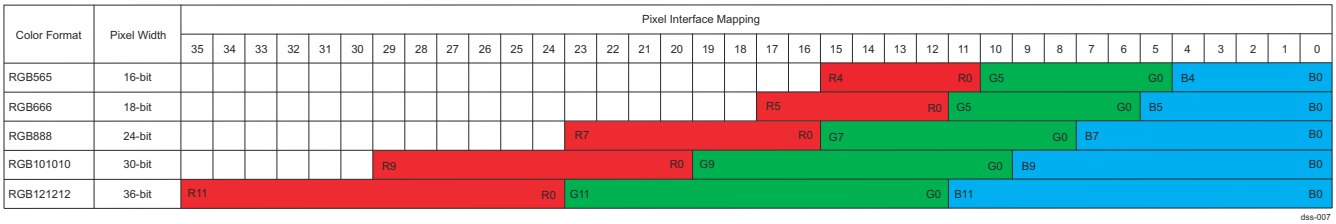


Figure 12-289. DSS Pixel Mapping on the DISPC Video Port Outputs

The pixel format support at a DISPC VP output when connected to each of the DPI, eDP, and DSI peripherals is as listed in Table 12-328.

Copyright © 2024 Texas Instruments Incorporated

Table 12-328. DSS Pixel Format Interoperability between DISPC VP Output and DPI, eDP, and DSI

Pixel Format	DISPC VP Output Pixel Format	DPI Parallel Out	DSI		eDP
	DSS0_VP_CONTROL [10-8] DATALINES Register Field Value		DSI-DPI Interface	DSI-SDI Interface	
RGB565	0x1	Supported	Supported	Supported	Not supported
RGB666	0x2	Supported	Supported	Supported	Not supported
RGB888	0x3	Supported	Supported	Supported	Not supported
RGB101010	0x4	Not supported	Supported	Not supported	Supported
RGB121212	0x5	Not supported	Supported	Not supported	Supported

Note

The DISPC VP outputs are connected to the display peripherals through the Merge-Split-Sync (MSS) block. For more details, see *DISPC VP Merge-Split-Sync (MSS) Module*.

12.6.2.1 DISPC Environment

Figure 12-290 shows the DSS DPI parallel interface signals. The output data bus of the DISPC video ports is 36 bits wide. The DSS DPI parallel interface uses up to the 24 LSB bits [23:0] of the data bus.

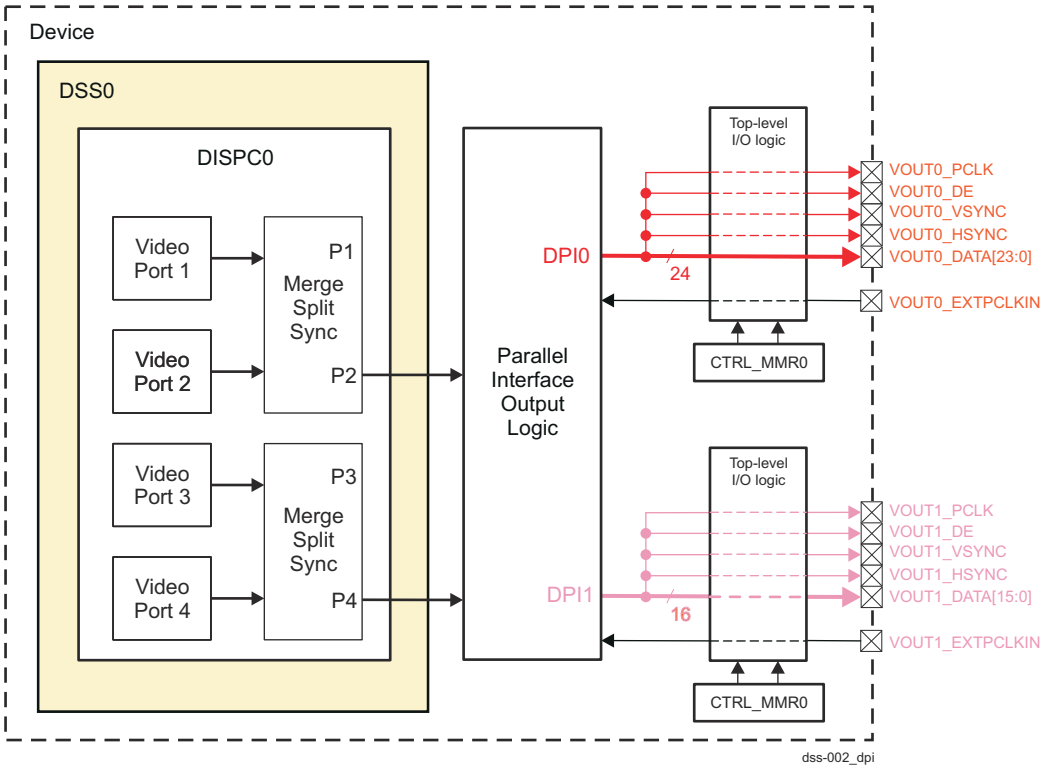


Figure 12-290. DSS DPI Parallel Interface Signals

Each DISPC video port in Figure 12-290 can be connected to any DPI output via the parallel interface output logic.

The [3-0] DPI_0_CONN and [7-4] DPI_1_CONN fields in the DSS0_COMMON_DISPC_CONNECTIONS register define the DISPC VP connections to DPI0 and DPI1 outputs, respectively.

The DISPC video ports output the required data and control signals to device pads to support one of the following display interface modes:

Copyright © 2024 Texas Instruments Incorporated

- Parallel MIPI DPI 2.0 (Digital Pixel Interface): RGB 16/18/24-bit output with separate sync signals.
- BT.656/BT.1120 interface: YUV422 output (8/10-bit modes) with embedded syncs.
- BT.601 interface: YUV422 output with discrete syncs. For more details, see *VSYNC/HSYNC/DE Signal Export to SoC Boundary*.

12.6.2.1.1 RGB Data Output

This section describes the pixel data bus for RGB formats and shows timing diagrams of transactions and synchronizations.

For the active matrix display type, one pixel per pixel clock is displayed. The diagrams represent the configuration of assertion of the data on the rising edge of the pixel clock. It is possible to program the interface timings to output the data on the falling edge of the pixel clock.

Figure 12-291 through Figure 12-294 show the interface to 12-, 16-, 18-, and 24-bit RGB active matrix displays. Each vertical line represents one output pixel. The width of the data bus can be configured through DSS0_VP_CONTROL[10-8] DATALINES register bitfield.

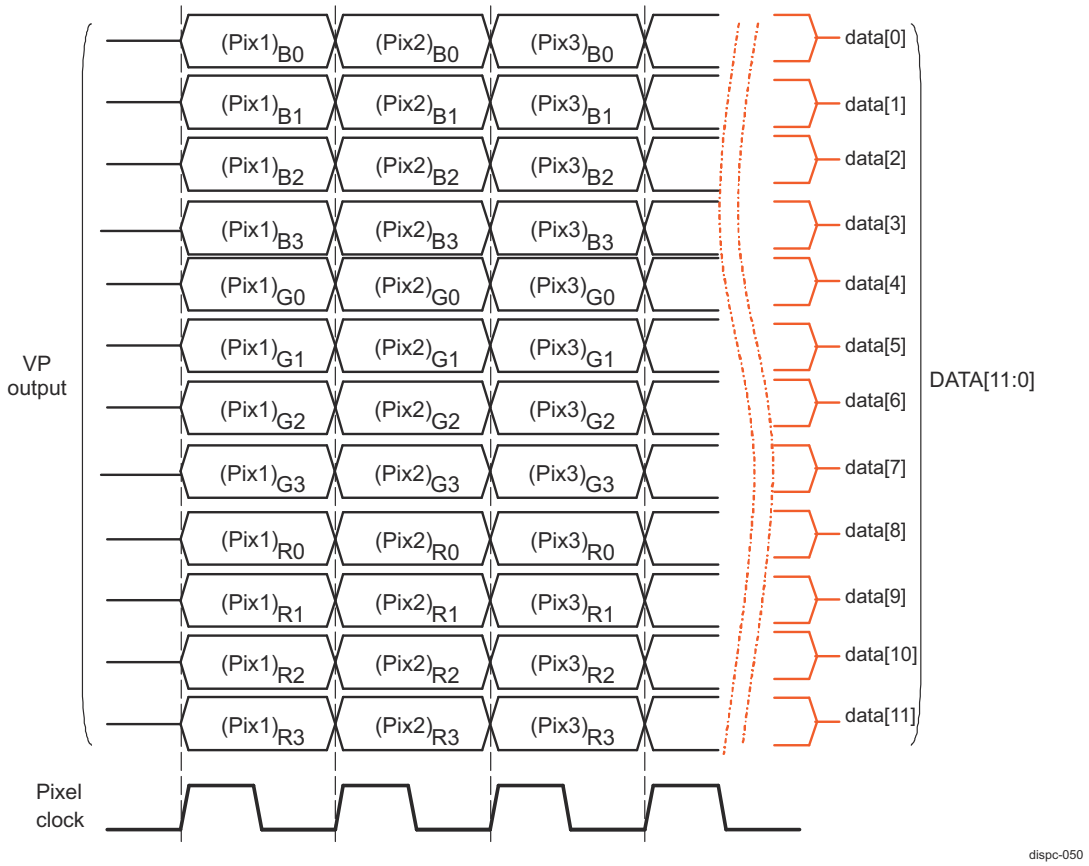


Figure 12-291. DISPC Video Port Pixel Data - 12-bit RGB Active Matrix

Copyright © 2024 Texas Instruments Incorporated

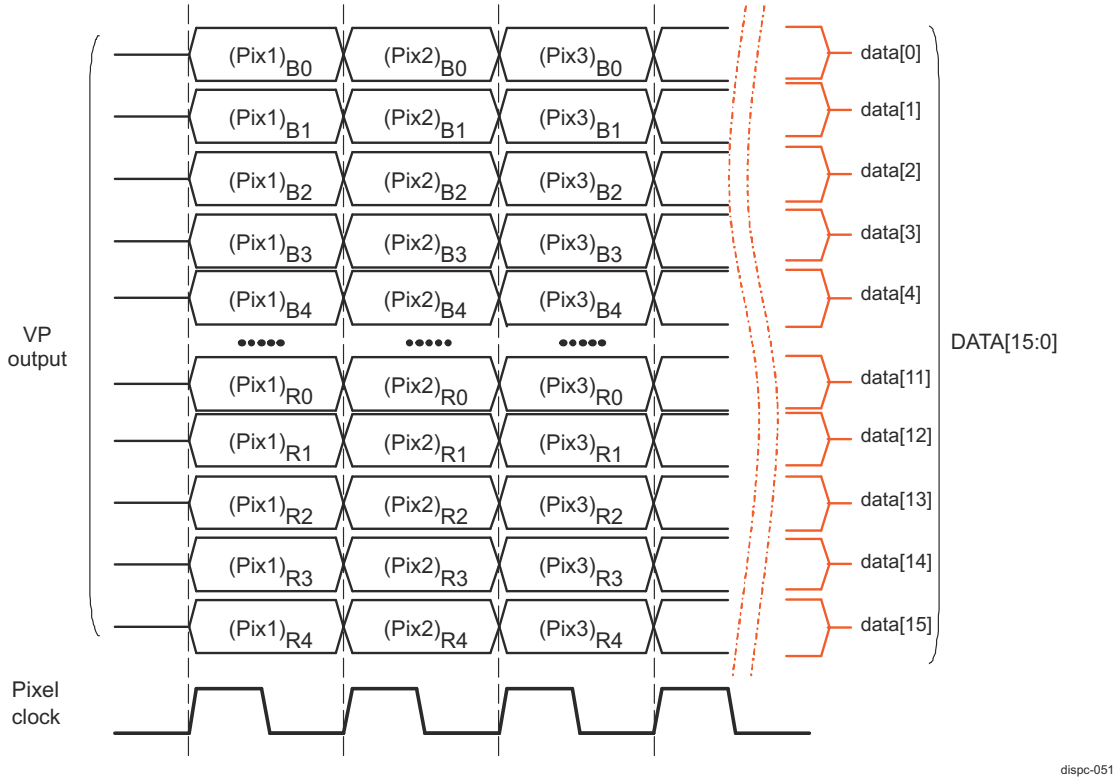


Figure 12-292. DISPC Video Port Pixel Data - 16-bit RGB Active Matrix

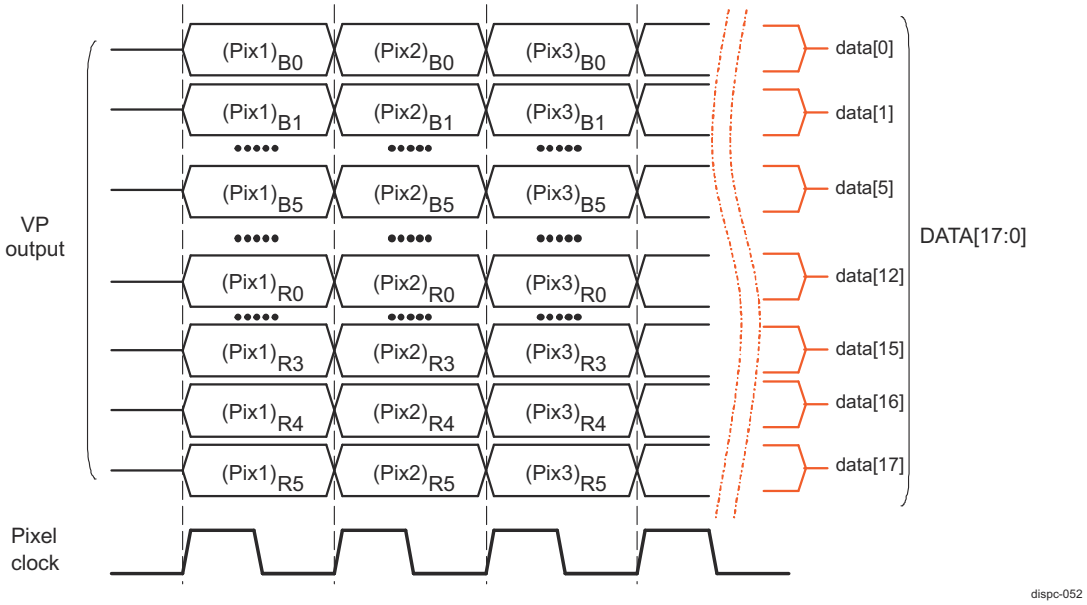


Figure 12-293. DISPC Video Port Pixel Data - 18-bit RGB Active Matrix

Copyright © 2024 Texas Instruments Incorporated

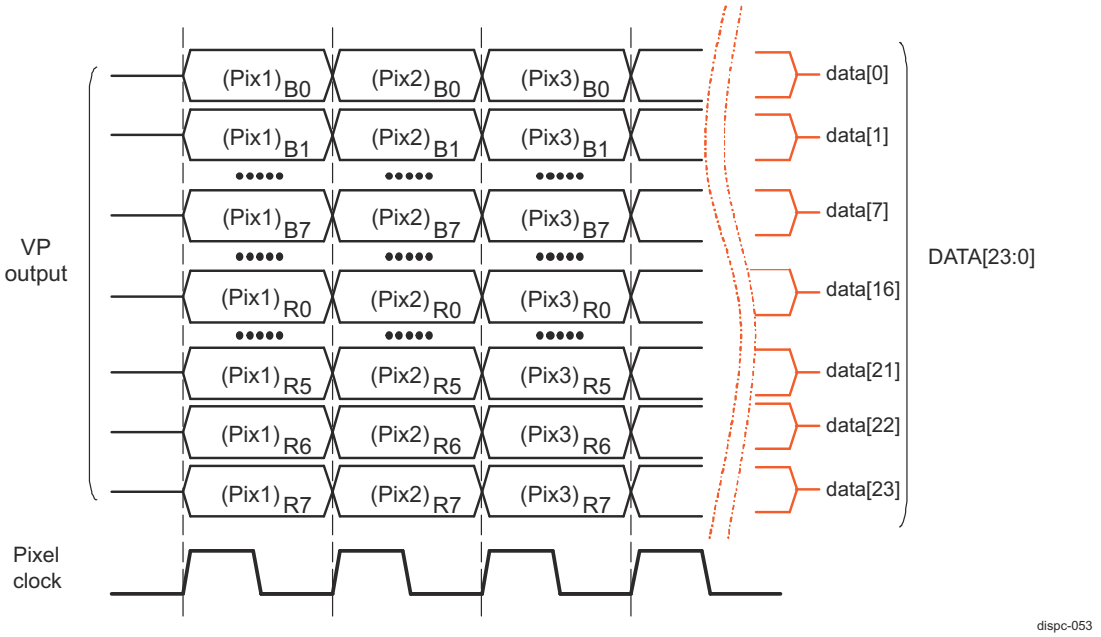


Figure 12-294. DISPC Video Port Pixel Data - 24-bit RGB Active Matrix

12.6.2.1.2 YUV Data Output (BT.656/BT.1120)

Figure 12-295 shows the signal mapping on the DATA[23:0] output data bus for the BT.656 mode. Bits [9-0] are dedicated for BT.656 mode (10-bit). In BT.656 mode however, for compatibility with existing 8-bit interfaces, the two LSBs are ignored and only bits [9-2] are effectively used.

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Unused														BT.656									

Figure 12-295. DISPC Video Port Data Mapping for BT.656 Mode

Figure 12-296 shows the signal mapping on a DATA[23:0] output data bus for the BT.1120 mode. Bits [19-10] (CbCr) and [9-0] (Y) are used in 20-bit mode. Bits [19-12] (CbCr) and [9-2] (Y) are used in 16-bit mode (YCbCr422).

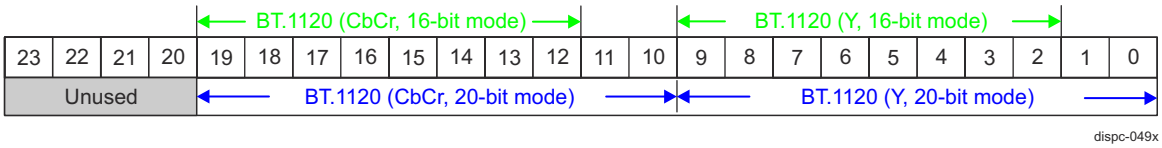


Figure 12-296. DISPC Video Port Data Mapping for BT.1120 Mode

The DISPC VP outputs support both interlace and progressive content in BT.656/BT.1120 modes. For more information on timings configuration, see Section 12.6.3.11.8, *DISPC VP Timing Generator and Display Panel Settings*.

Note

In progressive BT.656/BT.1120 mode the maximum output resolution will be limited, as it requires two pixel clock cycles to send out one pixel.

12.6.2.1.3 Display Timing Diagrams

Figure 12-297 through Figure 12-300 show examples with timing diagrams of synchronization signals and pixel clocks for active matrix panels. The DISPC video ports directly drive these signals, which are related to the

Copyright © 2024 Texas Instruments Incorporated

programmable fields listed in Table 12-329. For more information, see also Section 12.6.3.11.8, *DISPC VP Timing Generator and Display Panel Settings*.

Table 12-329. DISPC Video Port Register Fields for Active Matrix Display

Name	Register	Description
PPL	DSS0_VP_SIZE_SCREEN[13-0] PPL value + 1	Pixels per line
LPP	DSS0_VP_SIZE_SCREEN[29-16] LPP value + 1	Lines per panel
HBP	DSS0_VP_TIMING_H[31-20] HBP value + 1	Horizontal back porch
HFP	DSS0_VP_TIMING_H[19-8] HFP value + 1	Horizontal front porch
HSW	DSS0_VP_TIMING_H[7-0] HSW value + 1	Horizontal synchronization pulse width
VBP	DSS0_VP_TIMING_V[31-20] VBP value	Vertical back porch
VFP	DSS0_VP_TIMING_V[19-8] VFP value	Vertical front porch
VSW	DSS0_VP_TIMING_V[7-0] VSW value + 1	Vertical synchronization pulse width
ALIGN	DSS0_VP_POL_FREQ[18] ALIGN	Alignment between HSYNC and VSYNC assertion
ONOFF	DSS0_VP_POL_FREQ[17] ONOFF	HSYNC and VSYNC pixel clock control
RF	DSS0_VP_POL_FREQ[16] RF	HSYNC and VSYNC pixel clock edge control
IEO	DSS0_VP_POL_FREQ[15] IEO	Invert output enable
IPC	DSS0_VP_POL_FREQ[14] IPC	Invert PCLK
IHS	DSS0_VP_POL_FREQ[13] IHS	Invert HSYNC
IVS	DSS0_VP_POL_FREQ[12] IVS	Invert VSYNC

- Active matrix timing configuration 1:
  - DSS0_VP_POL_FREQ[17] ONOFF = 0
  - DSS0_VP_POL_FREQ[16] RF = 0

    The HSYNC and VSYNC signals are driven on the opposite edge of PCLK from the pixel data.
  - DSS0_VP_POL_FREQ[15] IEO = 0

    The DE signal is active high.
  - DSS0_VP_POL_FREQ[14] IPC = 0

    The pixel data are driven on the rising edge of PCLK.
  - DSS0_VP_POL_FREQ[13] IHS = 0

    The HSYNC signal is active high.
  - DSS0_VP_POL_FREQ[12] IVS = 0

    The VSYNC signal is active high.
  - DSS0_VP_POL_FREQ[18] ALIGN = 0

    The VSYNC and HSYNC assertion is not aligned.

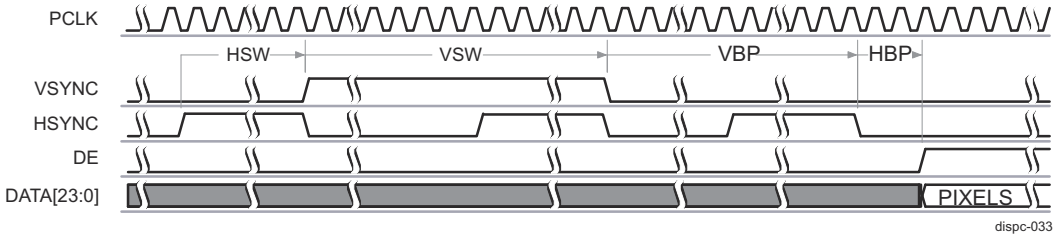


Figure 12-297. DISPC Display Timing Diagram of Configuration 1 (Start of Frame)

Copyright © 2024 Texas Instruments Incorporated

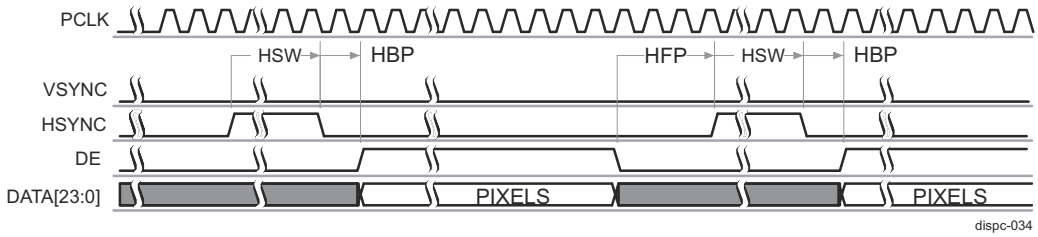


Figure 12-298. DISPC Display Timing Diagram of Configuration 1 (Between Lines)

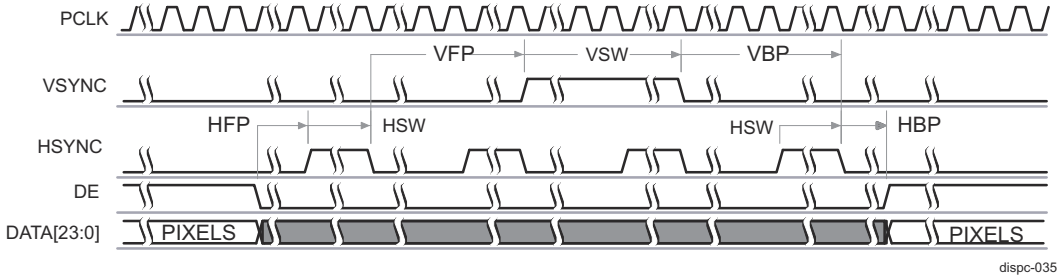


Figure 12-299. DISPC Display Timing Diagram of Configuration 1 (Between Frames)

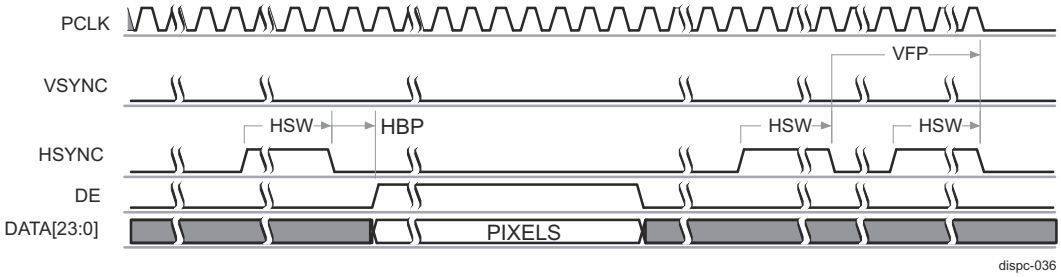


Figure 12-300. DISPC Display Timing Diagram of Configuration 1 (End of Frame)

- Active matrix timing configuration 2:
  - DSS0_VP_POL_FREQ[17] ONOFF = 1
  - DSS0_VP_POL_FREQ[16] RF = 1

    The HSYNC and VSYNC signals are driven on the rising edge of PCLK.
  - DSS0_VP_POL_FREQ[15] IEO = 1

    The DE signal is active low.
  - DSS0_VP_POL_FREQ[14] IPC = 1

    The pixel data is driven on the falling edge of PCLK.
  - DSS0_VP_POL_FREQ[13] IHS = 1

    The HSYNC signal is active low.
  - DSS0_VP_POL_FREQ[12] IVS = 1

    The VSYNC signal is active low.
  - DSS0_VP_POL_FREQ[18] ALIGN = 0

    The VSYNC and HSYNC assertion is not aligned.

Copyright © 2024 Texas Instruments Incorporated

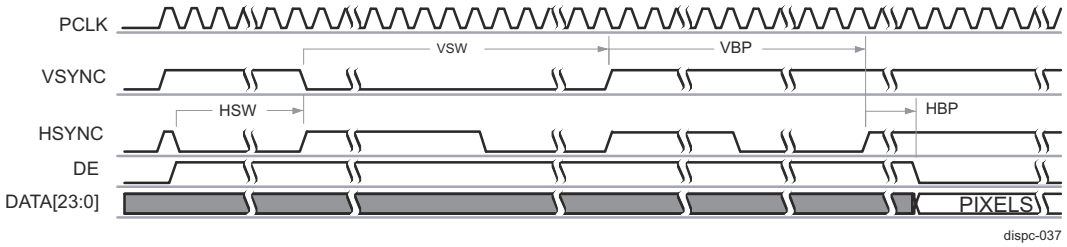


Figure 12-301. DISPC Display Timing Diagram of Configuration 2 (Start of Frame)

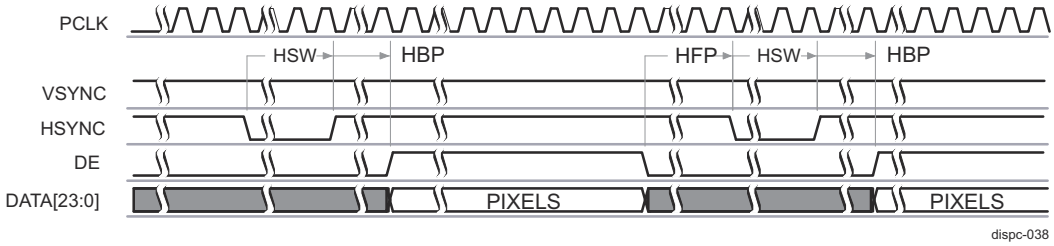


Figure 12-302. DISPC Display Timing Diagram of Configuration 2 (Between Lines)

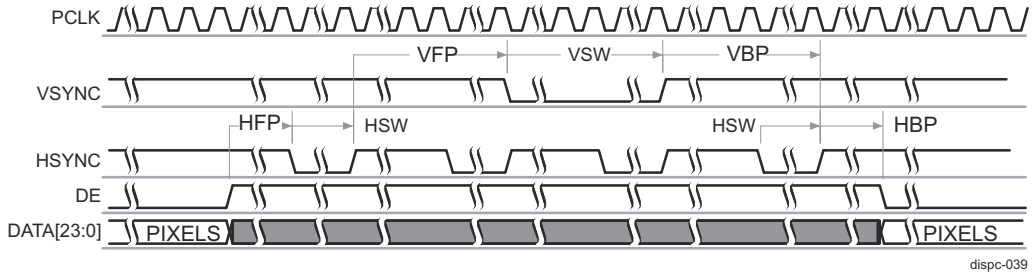


Figure 12-303. DISPC Display Timing Diagram of Configuration 2 (Between Frames)

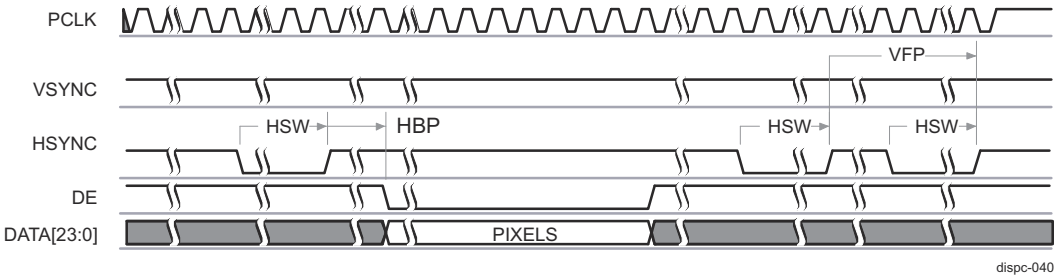


Figure 12-304. DISPC Display Timing Diagram of Configuration 2 (End of Frame)

- Active matrix timing configuration 3:
  - DSS0_VP_POL_FREQ[17] ONOFF = 1
  - DSS0_VP_POL_FREQ[16] RF = 1

    The HSYNC and VSYNC signals are driven on the rising edge of PCLK.
  - DSS0_VP_POL_FREQ[15] IEO = 0

    The DE signal is active high.
  - DSS0_VP_POL_FREQ[14] IPC = 0

    The pixel data are driven on the rising edge of PCLK.
  - DSS0_VP_POL_FREQ[13] IHS = 0

    The HSYNC signal is active high.
  - DSS0_VP_POL_FREQ[12] IVS = 0

Copyright © 2024 Texas Instruments Incorporated

The VSYNC signal is active high.

– DSS0_VP_POL_FREQ[18] ALIGN = 0

The VSYNC and HSYNC assertion is not aligned.

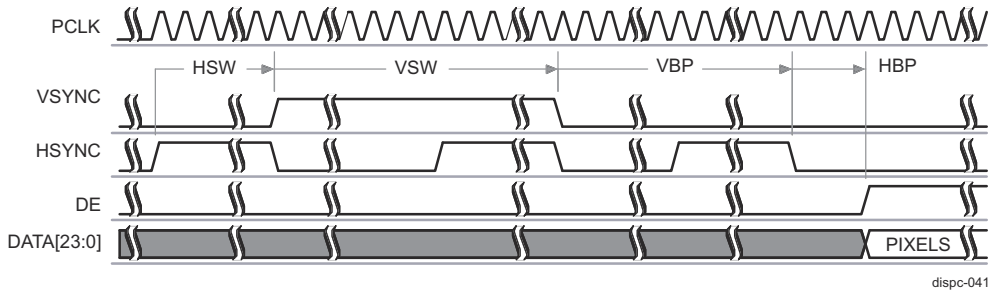


Figure 12-305. DISPC Display Timing Diagram of Configuration 3 (Start of Frame)

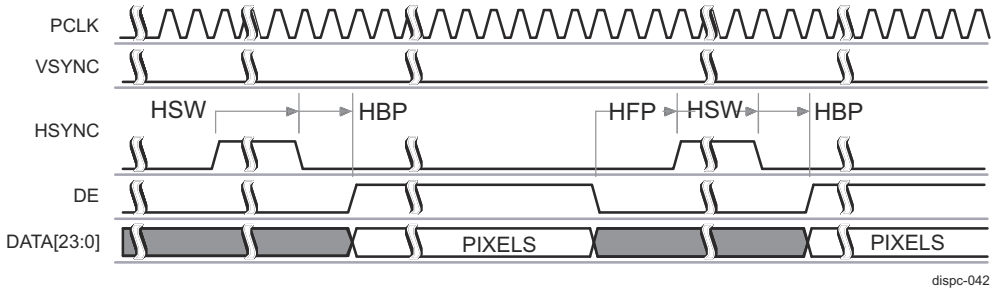


Figure 12-306. DISPC Display Timing Diagram of Configuration 3 (Between Lines)

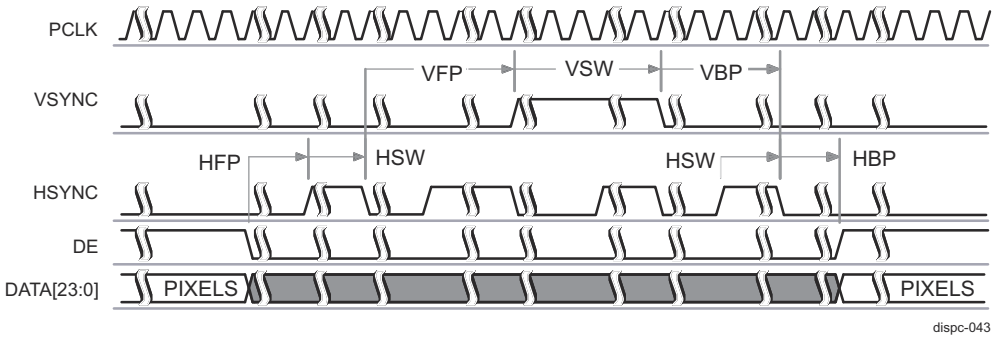


Figure 12-307. DISPC Display Timing Diagram of Configuration 3 (Between Frames)

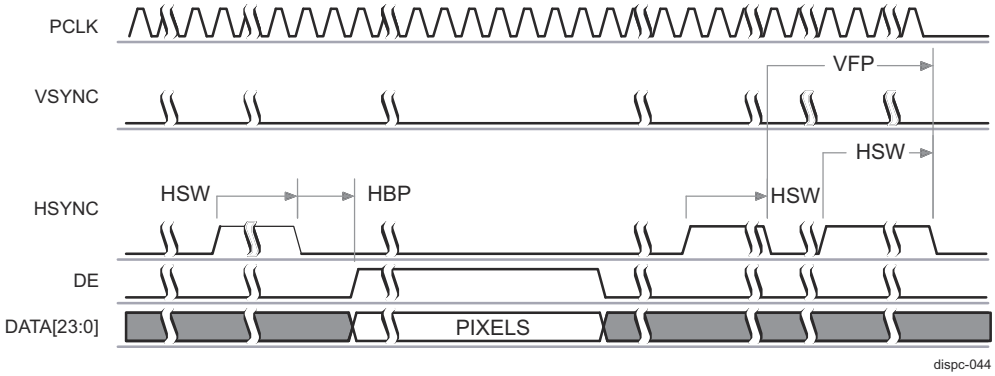


Figure 12-308. DISPC Display Timing Diagram of Configuration 3 (End of Frame)

Copyright © 2024 Texas Instruments Incorporated

12.6.2.1.4 VSYNC/HSYNC/DE Signal Export to SoC Boundary

The DSS exports the VSYNC, HSYNC and DE signals of all DISPC video port outputs to the SoC boundary. This feature is supplementary to the signals already exported out on the parallel DPI interfaces (DPI0/DPI1). This allows the support of the following use cases:

- BT.601 interface (YUV output with discrete syncs)
- FSYNC support for external camera sensor

The necessary singnal muxing is achieved at SoC level (via I/O pin multiplexing).

BT.601 Standard Support

The DSS only supports YUV output with embedded syncs (BT.656 and BT.1120 standards). YUV output with discrete sync (BT.601 standard) is not supported natively on the DISPC VP outputs.

At SoC level, it is possible to achieve BT.601 operation on a DPI parallel out (DPI0/DPI1) by combining two DISPC VP outputs as follows:

- Enable one DISPC VP to output YUV422 data in BT.656 mode on either DPI0 or DPI1 output.
- Use an alternative DISPC VP, that is synchronized and programmed (in RGB TDM mode), to output the discrete sync signals on the same DPI output. This can be achieved by configuring the I/O pin multiplexing at SoC level.
- The following restrictions apply:
  - If DPI0 is used to output YUV422 data, then the VSYNC/HSYNC/DE signals of either VP1 or VP3 can be output on DPI0.
  - If DPI1 is used to output YUV422 data, then the VSYNC/HSYNC/DE signals of only VP1 can be output on DPI1.

FSYNC Support for External Camera Sensor

An external camera sensors can use the VSYNC output from DSS as a synchronization input (FSYNC input). To make this connection, the VSYNC signals of all DISPC video ports are made available at SoC boundary.

Generally, in the cases where the DISPC VP is connected to DSI or eDP, the VSYNC information is "lost" as a separate signal at SoC boundary, although this VSYNC is still available at DISPC module boundary. For such cases, the VSYNCs are exported to SoC level in parallel to the DSI/eDP connection.

12.6.2.2 DSI Environment

Figure 12-309 shows the DSI interface signals.

Copyright © 2024 Texas Instruments Incorporated

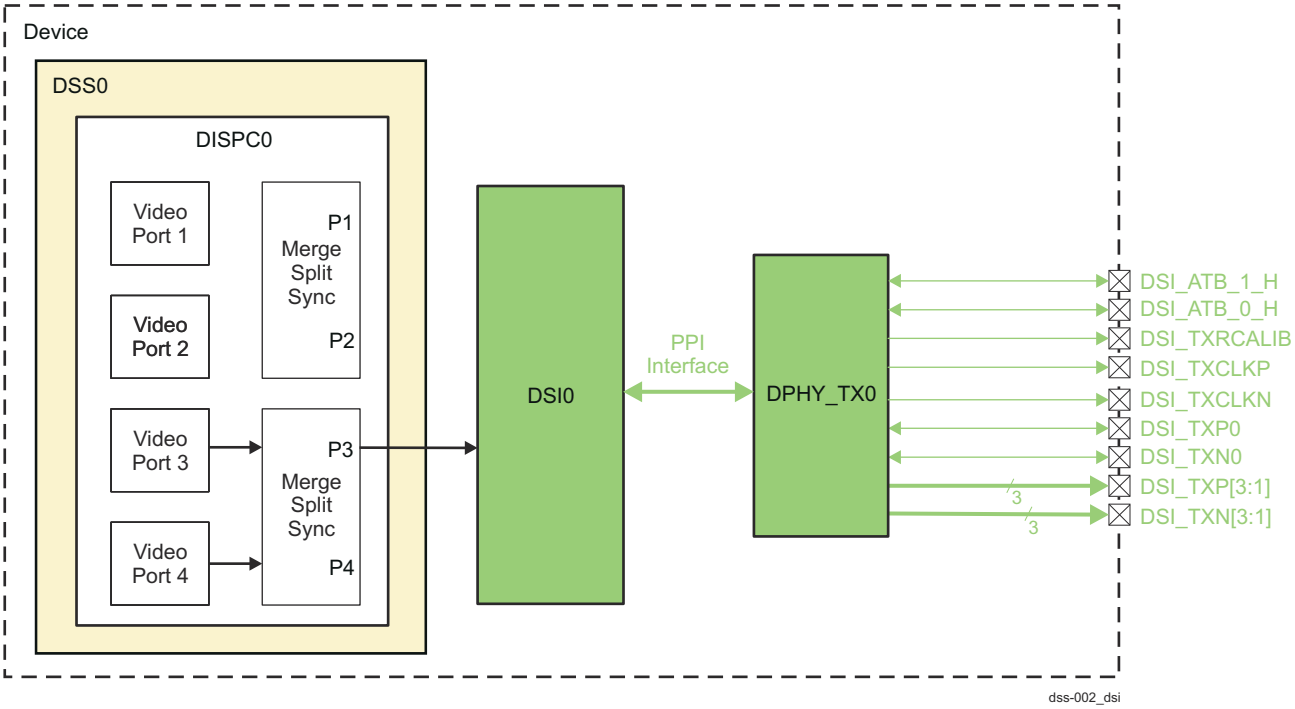


Figure 12-309. DSI Interface Signals

12.6.2.3 EDP Environment

Figure 12-310 shows the EDP interface signals.

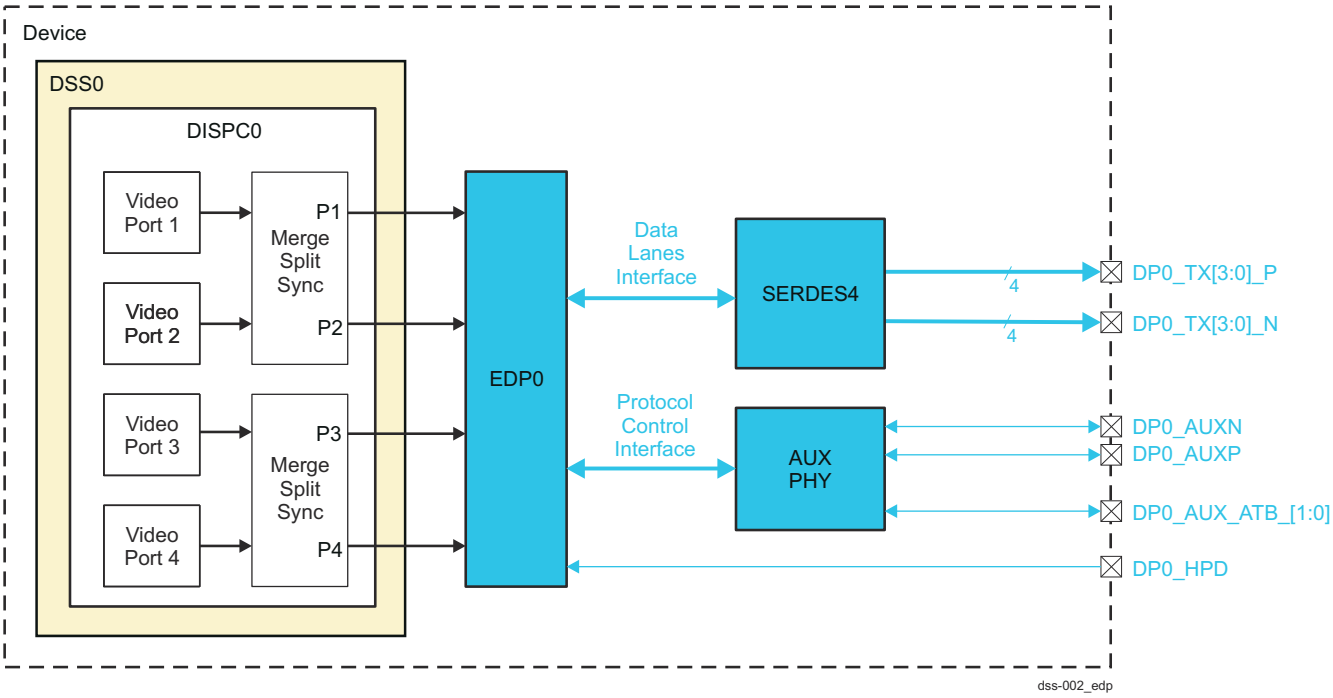


Figure 12-310. DP/eDP Interface Signals

12.6.3 Display Subsystem Controller (DISPC) with Frame Buffer Decompression Core (FBDC)

12.6.3.1 DISPC Overview

Figure 12-311 is a simplified block diagram of DISPC.

Copyright © 2024 Texas Instruments Incorporated

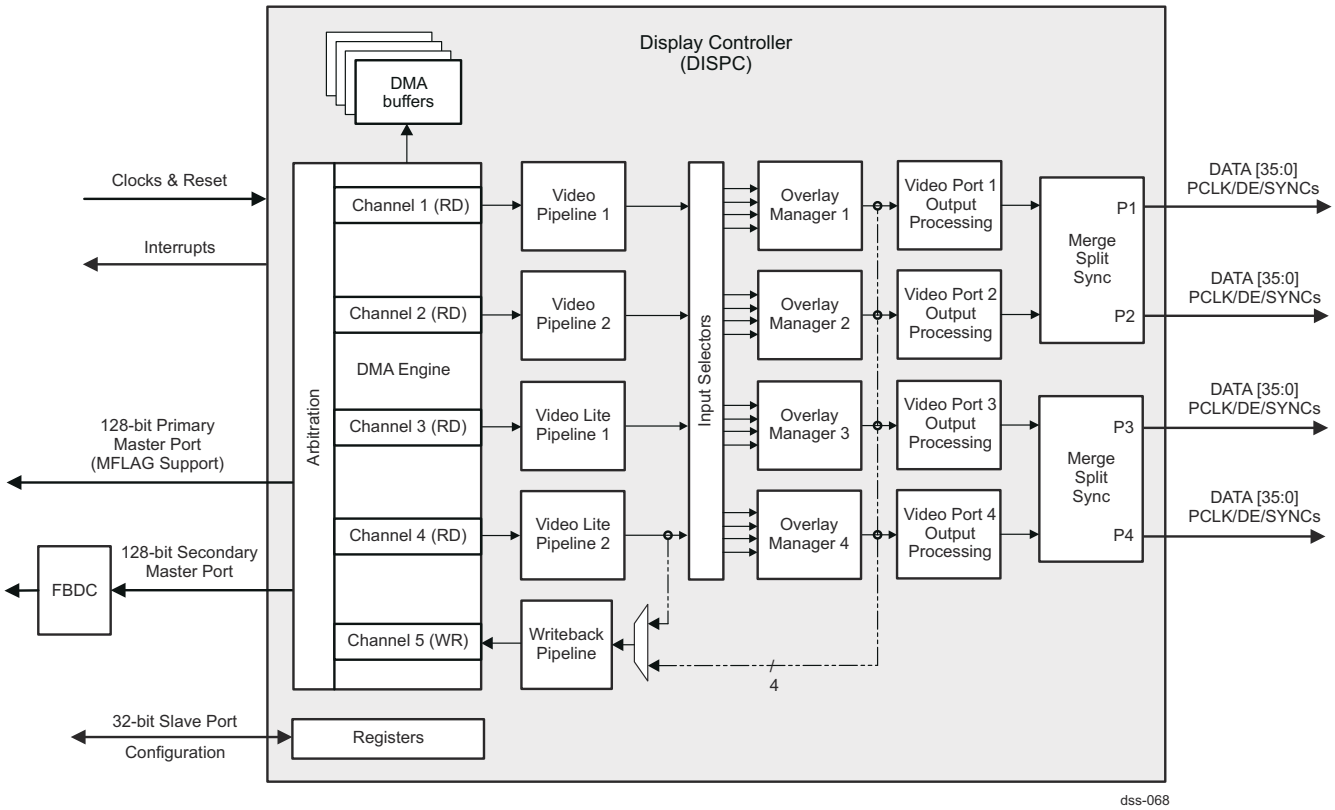


Figure 12-311. DISPC Architecture Overview

The DISPC integrated within DSS is capable of fetching pixel data from the device system memory through its dual master ports, performing various pixel processing, and then providing the processed pixels to an external display panel. The internal DMA engine tightly coupled to the DISPC is used for the pixel data transfer from system memory (frame buffer). The DISPC DMA engine is in charge of scheduling the memory requests. Several processes are configurable in order to manage the video pipeline features (color space conversion, up-sampling, down-sampling) and overlay features. The internal timing generator logic generates the video port output signals based on VESA DMT and CEA-861 standards. The DISPC video port outputs can be connected to display panels either directly (for MIPI DPI 2.0 or BT.656/BT.1120 support), or through either MIPI DSI or DisplayPort (DP/eDP) interfaces.

Note

The DISPC does not support any tiled frame buffer. The DISPC has no internal capability to support rotation of the frame buffer. The support for a compressed frame buffer is provided via the Frame Buffer Decompression (FBDC) module connected to the DISPC secondary master port.

Note

The display resolution is programmable and can be any width in the range [1:4096] pixels. The following limitations apply, related to the type of display or the processing done:

- Active Matrix screen + dithering forces a width multiple of 2 pixels
- Active Matrix + TDM may force a width multiple of 2 pixels

The display buffers in the system memory must consist of contiguous pixels.

12.6.3.2 DISPC Clocks

The DISPC has one clock domain for its internal logic and separate domains for each video port output.

Copyright © 2024 Texas Instruments Incorporated

The DISPC functional clock (DSS_FUNC_CLK) serves as the internal logic clock and also acts as the interface clock for the DISPC master and slave ports to system interconnect. There is no internal divider on this clock.

The DISPC pixel clocks (DSS_DPI_p_PCLK and DSS_DPI_p_DIV_PCLK, where p = 0 to 3) serve as the clocks for the output display interface. The DSS_DPI_p_PCLK clock is the 2x version of the DSS_DPI_p_DIV_PCLK clock. There are no internal dividers on the pixel clocks.

The relationship between the outgoing pixel clock and the input pixel clocks is as shown in Section 12.6.3.11.9.1, *MSS Clocking Scheme*.

The frequency of the display controller logic clock (DSS_FUNC_CLK) has to be greater than the frequency of the DSS_DPI_p_PCLK clocks, in order to get the DISPC internal logic to function properly. The frequency of the DSS_DPI_p_PCLK and DSS_DPI_p_DIV_PCLK clocks depend on the required output display resolution and frame rate. For information on the maximum supported frequency ratings, see the device-specific Datasheet.

The DSS_FUNC_CLK is asynchronous to DSS_DPI_p_PCLK and DSS_DPI_p_DIV_PCLK clocks. They can be generated by different sources.

The DSS_DPI_p_PCLK and DSS_DPI_p_DIV_PCLK clocks are synchronous to one another (for the same value of *p*).

The DSS0_COMMON_DSS_SYSCONFIG[0] AUTOCLKGATING regiser bit is set by default to allow the auto-gating of the interface and functional clocks. The AUTOCLKGATING bit can be reset to disable the auto-gating of the clocks, if required.

The DISPC provides also a clock-gating control on sub-module level, via configuration of the appropriate DSS0_COMMON_DISPC_CLKGATING_DISABLE register fields.

12.6.3.3 DISPC Resets

DISPC receives a single hardware reset signal. For more information, see *DSS Integration*.

To perform a software reset on the DISPC, set the DSS0_COMMON_DSS_SYSCONFIG[1] SOFTRESET bit to 0x1. The DSS0_COMMON_DSS_SYSSTATUS[0] DISPC_FUNC_RESETDONE bit indicates that the software reset is complete (for the DISPC internal logic) when its value is 0x1. When the software reset completes, the DSS0_COMMON_DSS_SYSCONFIG[1] SOFTRESET bit is automatically reset. Software must ensure that the software reset completes before performing DISPC operations.

The completion of the software reset for the video ports logic is indicated in the DSS0_COMMON_DSS_SYSSTATUS[3-1] DISPC_VP_RESETDONE register bit-field.

12.6.3.4 DISPC Power Management

DISPC supports a power management protocol with the device Power Sleep Controller (PSC).

The (software) sequence, while performing a *clkstop_req* to DISPC, is as follows:

1. Disable DISPC by programing the [0] ENABLE bit of DSS0_VP_CONTROL register to 0.
2. DISPC hardware completes the output of the current frame. Then hardware sets the DSS0_COMMON_DSS_SYSSTATUS[9] DISPC_IDLE_STATUS register bit to 1.
3. Poll the DSS_IDLE_STATUS bit to ensure that DISPC is in idle mode.
4. PSC initiates *clkstop_req* to DISPC.
5. DISPC hardware acknowledges with *clkstop_ack* immediately.

Copyright © 2024 Texas Instruments Incorporated

12.6.3.5 DISPC Interrupt Requests

The DISPC supports twelve interrupt output lines. The DISPC interrupt events are classified into functional, internal diagnostic error, and secure interrupt sub-categories, and then mapped to the respective sub-category interrupt line. Multiple sets of IRQ aggregation registers and IRQ generators enable fully independent monitoring and control of the interrupt events by up to 4 processor hosts at SoC level, as shown in Figure 12-312. For more details on the mapping of the interrupt lines to SoC-level resources, see *DISPC Integration*.

- The 4 functional interrupt (FUNC_IRQ) lines are DISPC_FUNC_IRQ_PROC0_0 through DISPC_FUNC_IRQ_PROC3_0.
- The 4 internal diagnostic error interupt (SAFETY_ERROR_IRQ) lines are DISPC_SAFETY_ERROR_IRQ_PROC0_0 through DISPC_SAFETY_ERROR_IRQ_PROC3_0.
- The 4 secure interrupt (SECURE_IRQ) lines are DISPC_SECURE_IRQ_PROC0_0 through DISPC_SECURE_IRQ_PROC3_0.

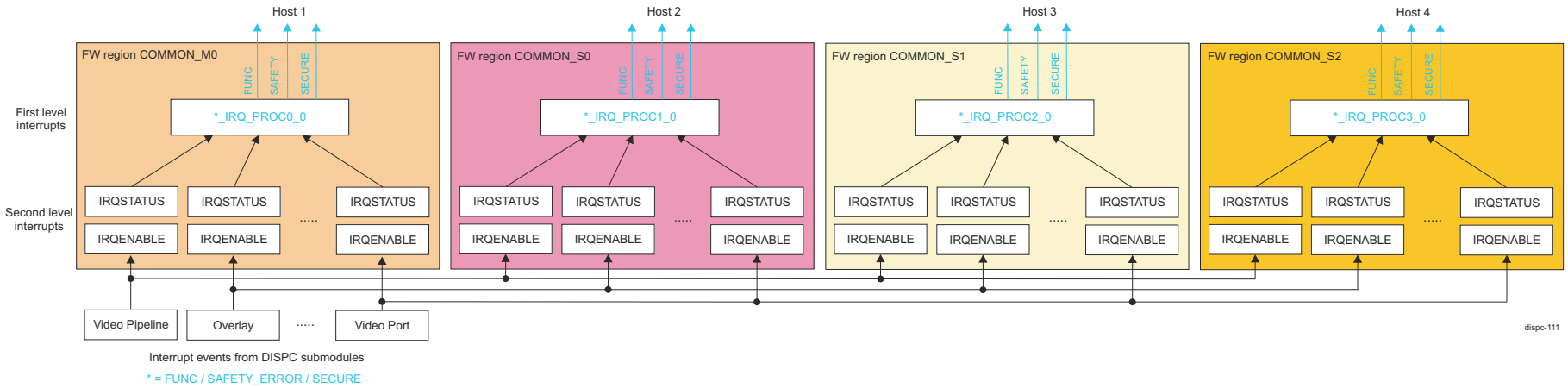


Figure 12-312. DISPC Interrupts Generation

Each of the interrupt signals indicates that one or more interrupt events are detected by the hardware. Each event is independently maskable for each interrupt output.

There are two level of interrupt events, as shown in Figure 12-312. The first level is used to indicate common events and is also source for the second level of interrupts. The second level of interrupt events consists of status and enable interrupt registers for each video pipeline and each video port.

Table 12-330 describes the first level of interrupt events with associated mask and status register fields. Each interrupt event is captured in an interrupt status register. Equivalent DSS0_COMMON_DISPC_IRQSTATUS_RAW register exist, which is updated even if interrupts are not enabled. This allows software to get access to updated status for all interrupt events.

Table 12-330. DISPC Interrupts - First Level

Interrupt Name	Set Interrupt Enable Register DSS0_COMMON_DISPC_IRQENABLE_SET	Clear Interrupt Enable Register DSS0_COMMON_DISPC_IRQENABLE_CLR	Interrupt Status Register DSS0_COMMON_DISPC_IRQSTATUS	Description
VID1_IRQ	[4] SET_VID_IRQ	[4] CLR_VID_IRQ	[4] VID_IRQ	At least one event of the VID1 pipeline interrupt events has occurred. See Table 12-331 for more details.

Copyright © 2024 Texas Instruments Incorporated

Table 12-330. DISPC Interrupts - First Level (continued)

Interrupt Name	Set Interrupt Enable Register DSS0_COMMON_DISPC_IRQENABLE_SET	Clear Interrupt Enable Register DSS0_COMMON_DISPC_IRQENABLE_CLR	Interrupt Status Register DSS0_COMMON_DISPC_IRQSTATUS	Description
VIDL1_IRQ	[5] SET_VID_IRQ	[5] CLR_VID_IRQ	[5] VID_IRQ	At least one event of the VIDL1 pipeline interrupt events has occurred. See Table 12-332 for more details.
VID2_IRQ	[6] SET_VID_IRQ	[6] CLR_VID_IRQ	[6] VID_IRQ	At least one event of the VID2 pipeline interrupt events has occurred.
VIDL2_IRQ	[7] SET_VID_IRQ	[7] CLR_VID_IRQ	[7] VID_IRQ	At least one event of the VID2L pipeline interrupt events has occurred.
VP1_IRQ	[0] SET_VP_IRQ	[0] CLR_VP_IRQ	[0] VP_IRQ	At least one event of the VP1 interrupt events has occurred. See Table 12-334 for more details.
VP2_IRQ	[1] SET_VP_IRQ	[1] CLR_VP_IRQ	[1] VP_IRQ	At least one event of the VP2 interrupt events has occurred. See Table 12-335 for more details.
VP3_IRQ	[2] SET_VP_IRQ	[2] CLR_VP_IRQ	[2] VP_IRQ	At least one event of the VP3 interrupt events has occurred.
VP4_IRQ	[3] SET_VP_IRQ	[3] CLR_VP_IRQ	[3] VP_IRQ	At least one event of the VP4 interrupt events has occurred.
WB_IRQ	[14] SET_WB_IRQ	[14] CLR_WB_IRQ	[14] WB_IRQ	At least one event of the WB pipeline interrupt events has occurred.

Table 12-331 describes the second level of interrupts for VID1 and VID2 pipelines with associated mask and status register bits.

Table 12-331. DISPC Interrupts - Second Level - VID1 and VID2 Pipeline

Interrupt Name	VID1 Interrupt Mask DSS0_COMMON_VID_IRQENABLE_0	VID1 Interrupt Status DSS0_COMMON_VID_IRQSTATUS_0	VID2 Interrupt Mask DSS0_COMMON_VID_IRQENABLE_2	VID2 Interrupt Status DSS0_COMMON_VID_IRQSTATUS_2	Description
VIDBUFFERUNDERFLOW_IRQ	[0] VIDBUFFERUNDERFLOW_EN	[0] VIDBUFFERUNDERFLOW_IRQ	[0] VIDBUFFERUNDERFLOW_EN	[0] VIDBUFFERUNDERFLOW_IRQ	Video DMA buffer underflow (FUNC_IRQ): The input video DMA buffer goes underflow. This does not necessary means that the buffer is empty (out of order refill), but simply that the required pixel is not in yet.
VIDENDWINDOW_IRQ	[1] VIDENDWINDOW_EN	[1] VIDENDWINDOW_IRQ	[1] VIDENDWINDOW_EN	[1] VIDENDWINDOW_IRQ	End of the video window (FUNC_IRQ): The DMA engine has fetched all the data from memory for the video for the current frame.

Copyright © 2024 Texas Instruments Incorporated

Table 12-331. DISPC Interrupts - Second Level - VID1 and VID2 Pipeline (continued)

Interrupt Name	VID1 Interrupt Mask DSS0_COMMON_VID_IRQENABLE_0	VID1 Interrupt Status DSS0_COMMON_VID_IRQSTATUS_0	VID2 Interrupt Mask DSS0_COMMON_VID_IRQENABLE_2	VID2 Interrupt Status DSS0_COMMON_VID_IRQSTATUS_2	Description
VIDSAFETYREGION_IRQ	[2] SAFETYREGION_EN	[2] SAFETYREGION_IRQ	[2] SAFETYREGION_EN	[2] SAFETYREGION_IRQ	Video output MISR signature mismatch OR Video output freeze frame detect (SAFETY_ERROR_IRQ). The MISR signature generated does not match the expected signature OR the Video output frame freeze detection has triggered.
FBDC_CORRUPTTILE_IRQ	[3] FBDC_CORRUPTTILE_EN	[3] FBDC_CORRUPTTILE_IRQ	[3] FBDC_CORRUPTTILE_EN	[3] FBDC_CORRUPTTILE_IRQ	Corrupt tile is detected (FUNC_IRQ).
FBDC_ILLEGALTILEREQ_IRQ	[4] FBDC_ILLEGALTILEREQ_EN	[4] FBDC_ILLEGALTILEREQ_IRQ	[4] FBDC_ILLEGALTILEREQ_EN	[4] FBDC_ILLEGALTILEREQ_IRQ	Illegal tile request is detected (FUNC_IRQ).

Table 12-332 describes the second level of interrupts for VIDL1 and VIDL2 pipelines with associated mask and status register bits.

Table 12-332. DISPC Interrupts - Second Level - VIDL1 and VIDL2 Pipelines

Interrupt Name	VIDL1 Interrupt Mask DSS0_COMMON_VID_IRQENABLE_1	VIDL1 Interrupt Status DSS0_COMMON_VID_IRQSTATUS_1	VIDL2 Interrupt Mask DSS0_COMMON_VID_IRQENABLE_3	VIDL2 Interrupt Status DSS0_COMMON_VID_IRQSTATUS_3	Description
VIDBUFFERUNDERFLOW_IRQ	[0] VIDBUFFERUNDERFLOW_EN	[0] VIDBUFFERUNDERFLOW_IRQ	[0] VIDBUFFERUNDERFLOW_EN	[0] VIDBUFFERUNDERFLOW_IRQ	Video DMA buffer underflow (FUNC_IRQ): The input video DMA buffer goes underflow. This does not necessary means that the buffer is empty (out of order refill), but simply that the required pixel is not in yet.
VIDENDWINDOW_IRQ	[1] VIDENDWINDOW_EN	[1] VIDENDWINDOW_IRQ	[1] VIDENDWINDOW_EN	[1] VIDENDWINDOW_IRQ	End of the video window (FUNC_IRQ): The DMA engine has fetched all the data from memory for the video for the current frame.

Copyright © 2024 Texas Instruments Incorporated

Table 12-332. DISPC Interrupts - Second Level - VIDL1 and VIDL2 Pipelines (continued)

Interrupt Name	VIDL1 Interrupt Mask DSS0_COMMON_VID_IRQENABLE_1	VIDL1 Interrupt Status DSS0_COMMON_VID_IRQSTATUS_1	VIDL2 Interrupt Mask DSS0_COMMON_VID_IRQENABLE_3	VIDL2 Interrupt Status DSS0_COMMON_VID_IRQSTATUS_3	Description
VIDSAFETYREGION_IRQ	[2] SAFETYREGION_EN	[2] SAFETYREGION_IRQ	[2] SAFETYREGION_EN	[2] SAFETYREGION_IRQ	Video output MISR signature mismatch OR Video output freeze frame detect (SAFETY_ERROR_IRQ). The MISR signature generated does not match the expected signature OR the Video output frame freeze detection has triggered.
FBDC_CORRUPTTILE_IRQ	[3] FBDC_CORRUPTTILE_EN	[3] FBDC_CORRUPTTILE_IRQ	[3] FBDC_CORRUPTTILE_EN	[3] FBDC_CORRUPTTILE_IRQ	Corrupt tile is detected (FUNC_IRQ).
FBDC_ILLEGALTILEREQ_IRQ	[4] FBDC_ILLEGALTILEREQ_EN	[4] FBDC_ILLEGALTILEREQ_IRQ	[4] FBDC_ILLEGALTILEREQ_EN	[4] FBDC_ILLEGALTILEREQ_IRQ	Illegal tile request is detected (FUNC_IRQ).

Table 12-333 describes the second level of interrupts for the WB pipeline with associated mask and status register bits.

Table 12-333. DISPC Interrupts - Second Level - WB Pipeline

Interrupt Name	WB Interrupt Mask DSS0_COMMON_WB_IRQENABLE	WB Interrupt Status DSS0_COMMON_WB_IRQSTATUS	Description
BUFFEROVERFLOW_IRQ	[0] WBBUFFEROVERFLOW_EN	[0] WBBUFFEROVERFLOW_IRQ	Write-back DMA buffer Overflow (FUNC_IRQ): The output Write-back DMA buffer goes overflow. It cannot occur when write-back channel is used in memory to memory transfer mode but only in capture mode. In capture mode the timings are defined by the timer associated with the output. In memory-to-memory mode, there is a timing constraint.
UNCOMPLETEERROR_IRQ	[1] WBUNCOMPLETEERROR_EN	[1] WBUNCOMPLETEERROR_IRQ	Write-back un-complete error (FUNC_IRQ): The WB pipeline is reset before all data of the frame currently written back are output to the interconnect interface.

Copyright © 2024 Texas Instruments Incorporated

Table 12-333. DISPC Interrupts - Second Level - WB Pipeline (continued)

Interrupt Name	WB Interrupt Mask DSS0_COMMON_WB_IRQENABLE	WB Interrupt Status DSS0_COMMON_WB_IRQSTATUS	Description
FRAMEDONE_IRQ	[2] WBFRAMEDONE_EN	[2] WBFRAMEDONE_IRQ	Write-back Frame Done (FUNC_IRQ): The WB frame done in memory-to-memory mode of operation.
SECURITYVIOLATION_IRQ	[3] SECURITYVIOLATION_EN	[3] SECURITYVIOLATION_IRQ	Security Violation for WB output (SECURE_IRQ): A security violation (for example, a secure VID pipeline connected to a non-secure WB channel) has occurred.
SYNC_IRQ	[4] WBSYNC_EN	[4] WBSYNC_IRQ	Write-back sync (FUNC_IRQ): A configuration is copied from shadow registets to work for WB for next frame.

Table 12-334 describes the second level of interrupts for VP1 and VP2 outputs with associated mask and status register bits.

Table 12-334. DISPC Interrupts - Second Level - VP1 and VP2 Outputs

Interrupt Name	VP1 Interrupt Mask DSS0_COMMON_VP_IRQENABLE_0	VP1 Interrupt Status DSS0_COMMON_VP_IRQSTATUS_0	VP2 Interrupt Mask DSS0_COMMON_VP_IRQENABLE_1	VP2 Interrupt Status DSS0_COMMON_VP_IRQSTATUS_1	Description
FRAMEDONE_IRQ	[0] VPFRAMEDONE_EN	[0] VPFRAMEDONE_IRQ	[0] VPFRAMEDONE_EN	[0] VPFRAMEDONE_IRQ	Frame done for VP output (FUNC_IRQ). After disabling the VP output of the DISPC, the interrupt is set when the active frame related to the VP has completed.
VSYNC_IRQ	[1] VPVSYNC_EN	[1] VPVSYNC_IRQ	[1] VPVSYNC_EN	[1] VPVSYNC_IRQ	VSYNC for VP output (FUNC_IRQ): VSYNC interrupt for the VP has occurred at the end of the frame.
VSYNC_ODD_IRQ	[2] VPVSYNC_ODD_EN	[2] VPVSYNC_ODD_IRQ	[2] VPVSYNC_ODD_EN	[2] VPVSYNC_ODD_IRQ	VSYNC for odd field (FUNC_IRQ). VSYNC_ODD interrupt has occurred at the end of the frame (EVSYNC received and the field polarity is odd).
PROGRAMMEDLINENUMBER_IRQ	[3] VPPROGRAMMEDLINENUMBER_EN	[3] VPPROGRAMMEDLINENUMBER_IRQ	[3] VPPROGRAMMEDLINENUMBER_EN	[3] VPPROGRAMMEDLINENUMBER_IRQ	Programmed line number (FUNC_IRQ). The VP has reached the user-programmed line number.

Copyright © 2024 Texas Instruments Incorporated

Table 12-334. DISPC Interrupts - Second Level - VP1 and VP2 Outputs (continued)

Interrupt Name	VP1 Interrupt Mask DSS0_COMMON_VP_IRQENABLE_0	VP1 Interrupt Status DSS0_COMMON_VP_IRQSTATUS_0	VP2 Interrupt Mask DSS0_COMMON_VP_IRQENABLE_1	VP2 Interrupt Status DSS0_COMMON_VP_IRQSTATUS_1	Description
SYNCLOST_IRQ	[4] VPSYNCLOST_EN	[4] VPSYNCLOST_IRQ	[4] VPSYNCLOST_EN	[4] VPSYNCLOST_IRQ	Synchronization lost on VP output (FUNC_IRQ): Occurs when VSYNC width/ front or back porches are not wide enough to load the pipeline with data (VP output).
ACBIASCOUNTSTATUS_IRQ	[5] ACBIASCOUNTSTATUS_EN	[5] ACBIASCOUNTSTATUS_IRQ	[5] ACBIASCOUNTSTATUS_EN	[5] ACBIASCOUNTSTATUS_IRQ	ACBIASCOUNTSTATUS for VP output (FUNC_IRQ): AC BIAS transition counter has decremented to zero. Refer to the DSS0_VP_POL_FREQ[11-8] ACBI and [7-0] ACB register field descriptions.
VPSAFETYREGION_IRQ	[9-6] SAFETYREGION_EN Bit [9] = Internal Diagnostic Region 3 Bit [8] = Internal Diagnostic Region 2 Bit [7] = Internal Diagnostic Region 1 Bit [6] = Internal Diagnostic Region 0	[9-6] SAFETYREGION_IRQ Bit [9] = Internal Diagnostic Region 3 Bit [8] = Internal Diagnostic Region 2 Bit [7] = Internal Diagnostic Region 1 Bit [6] = Internal Diagnostic Region 0	[9-6] SAFETYREGION_EN Bit [9] = Internal Diagnostic Region 3 Bit [8] = Internal Diagnostic Region 2 Bit [7] = Internal Diagnostic Region 1 Bit [6] = Internal Diagnostic Region 0	[9-6] SAFETYREGION_IRQ Bit [9] = Internal Diagnostic Region 3 Bit [8] = Internal Diagnostic Region 2 Bit [7] = Internal Diagnostic Region 1 Bit [6] = Internal Diagnostic Region 0	VP output MISR signature mismatch, or VP output freeze frame detect (SAFETY_ERROR_IRQ). The MISR signature generated does not match the expected signature, or the VP output frame freeze detection has triggered.
SECURITYVIOLATION_IRQ	[10] SECURITYVIOLATION_EN	[10] SECURITYVIOLATION_IRQ	[10] SECURITYVIOLATION_EN	[10] SECURITYVIOLATION_IRQ	Security violation for VP output (SECURE_IRQ). A security violation (for example, a secure video pipeline connected to a non-secure VP/OVR) has occurred.

Copyright © 2024 Texas Instruments Incorporated

Table 12-334. DISPC Interrupts - Second Level - VP1 and VP2 Outputs (continued)

Interrupt Name	VP1 Interrupt Mask DSS0_COMMON_VP_IRQENABLE_0	VP1 Interrupt Status DSS0_COMMON_VP_IRQSTATUS_0	VP2 Interrupt Mask DSS0_COMMON_VP_IRQENABLE_1	VP2 Interrupt Status DSS0_COMMON_VP_IRQSTATUS_1	Description
VPSYNC_IRQ	[11] VPSYNC_EN	[11] VPSYNC_IRQ	[11] VPSYNC_EN	[11] VPSYNC_IRQ	Sync for VP output (FUNC_IRQ). Shadow to work copy of registers associated with VP1 has occurred. DSS0_VP_CONTROL[5] GOBIT register bit is cleared. This interrupt can trigger under the following cases: 1. NORMAL Operation: OVRn is connected to VPn. VPSYNC indicates sync of VPn. 2. Overlay Cascade: OVRn is connected to VPk. VPSYNC indicates sync of VPk. 3. Overlay M2M: OVRn is connected to WB. VPSYNC indicates sync event (WB start) of WB.
VPSAFETYREGION1_IRQ	[16-13] SAFETYREGION1_EN Bit [16] = Internal Diagnostic Region 7 Bit [15] = Internal Diagnostic Region 6 Bit [14] = Internal Diagnostic Region 5 Bit [13] = Internal Diagnostic Region 4	[16-13] SAFETYREGION1_IRQ Bit [16] = Internal Diagnostic Region 7 Bit [15] = Internal Diagnostic Region 6 Bit [14] = Internal Diagnostic Region 5 Bit [13] = Internal Diagnostic Region 4	[16-13] SAFETYREGION1_EN Bit [16] = Internal Diagnostic Region 7 Bit [15] = Internal Diagnostic Region 6 Bit [14] = Internal Diagnostic Region 5 Bit [13] = Internal Diagnostic Region 4	[16-13] SAFETYREGION1_IRQ Bit [16] = Internal Diagnostic Region 7 Bit [15] = Internal Diagnostic Region 6 Bit [14] = Internal Diagnostic Region 5 Bit [13] = Internal Diagnostic Region 4	VP output MISR signature mismatch, or VP output freeze frame detect (SAFETY_ERROR_IRQ). The MISR signature generated does not match the expected signature, or the VP output frame freeze detection has triggered.

Table 12-335 describes the second level of interrupts for VP3 and VP4 outputs with associated mask and status register bits.

Copyright © 2024 Texas Instruments Incorporated

Table 12-335. DISPC Interrupts - Second Level - VP3 and VP4 Outputs

Interrupt Name	VP3 Interrupt Mask DSS0_COMMON_VP_IRQENABLE_2	VP3 Interrupt Status DSS0_COMMON_VP_IRQSTATUS_2	VP4 Interrupt Mask DSS0_COMMON_VP_IRQENABLE_3	VP4 Interrupt Status DSS0_COMMON_VP_IRQSTATUS_3	Description
FRAMEDONE_IRQ	[0] VPFRAMEDONE_EN	[0] VPFRAMEDONE_IRQ	[0] VPFRAMEDONE_EN	[0] VPFRAMEDONE_IRQ	Frame done for VP output (FUNC_IRQ). After disabling the VP output of the DISPC, the interrupt is set when the active frame related to the VP has completed.
VSYNC_IRQ	[1] VPVSYNC_EN	[1] VPVSYNC_IRQ	[1] VPVSYNC_EN	[1] VPVSYNC_IRQ	VSYNC for VP output (FUNC_IRQ): VSYNC interrupt for the VP has occurred at the end of the frame.
VSYNC_ODD_IRQ	[2] VPVSYNC_ODD_EN	[2] VPVSYNC_ODD_IRQ	[2] VPVSYNC_ODD_EN	[2] VPVSYNC_ODD_IRQ	VSYNC for odd field (FUNC_IRQ). VSYNC_ODD interrupt has occurred at the end of the frame (EVSYNC received and the field polarity is odd).
PROGRAMMEDLINENUMBER_IRQ	[3] VPPROGRAMMEDLINENUMBER_EN	[3] VPPROGRAMMEDLINENUMBER_IRQ	[3] VPPROGRAMMEDLINENUMBER_EN	[3] VPPROGRAMMEDLINENUMBER_IRQ	Programmed line number (FUNC_IRQ). The VP has reached the user-programmed line number.
SYNCLOST_IRQ	[4] VPSYNCLOST_EN	[4] VPSYNCLOST_IRQ	[4] VPSYNCLOST_EN	[4] VPSYNCLOST_IRQ	Synchronization lost on VP output (FUNC_IRQ): Occurs when VSYNC width/ front or back porches are not wide enough to load the pipeline with data (VP output).
ACBIASCOUNTSTATUS_IRQ	[5] ACBIASCOUNTSTATUS_EN	[5] ACBIASCOUNTSTATUS_IRQ	[5] ACBIASCOUNTSTATUS_EN	[5] ACBIASCOUNTSTATUS_IRQ	ACBIASCOUNTSTATUS for VP output (FUNC_IRQ): AC BIAS transition counter has decremented to zero. Refer to the DSS0_VP_POL_FREQ[11-8] ACBI and [7-0] ACB register field descriptions.

Copyright © 2024 Texas Instruments Incorporated

Table 12-335. DISPC Interrupts - Second Level - VP3 and VP4 Outputs (continued)

Interrupt Name	VP3 Interrupt Mask DSS0_COMMON_VP_IRQENABLE_2	VP3 Interrupt Status DSS0_COMMON_VP_IRQSTATUS_2	VP4 Interrupt Mask DSS0_COMMON_VP_IRQENABLE_3	VP4 Interrupt Status DSS0_COMMON_VP_IRQSTATUS_3	Description
VPSAFETYREGION_IRQ	[9-6] SAFETYREGION_EN Bit [9] = Internal Diagnostic Region 3 Bit [8] = Internal Diagnostic Region 2 Bit [7] = Internal Diagnostic Region 1 Bit [6] = Internal Diagnostic Region 0	[9-6] SAFETYREGION_IRQ Bit [9] = Internal Diagnostic Region 3 Bit [8] = Internal Diagnostic Region 2 Bit [7] = Internal Diagnostic Region 1 Bit [6] = Internal Diagnostic Region 0	[9-6] SAFETYREGION_EN Bit [9] = Internal Diagnostic Region 3 Bit [8] = Internal Diagnostic Region 2 Bit [7] = Internal Diagnostic Region 1 Bit [6] = Internal Diagnostic Region 0	[9-6] SAFETYREGION_IRQ Bit [9] = Internal Diagnostic Region 3 Bit [8] = Internal Diagnostic Region 2 Bit [7] = Internal Diagnostic Region 1 Bit [6] = Internal Diagnostic Region 0	VP output MISR signature mismatch, or VP output freeze frame detect (SAFETY_ERROR_IRQ). The MISR signature generated does not match the expected signature, or the VP output frame freeze detection has triggered.
SECURITYVIOLATION_IRQ	[10] SECURITYVIOLATION_EN	[10] SECURITYVIOLATION_IRQ	[10] SECURITYVIOLATION_EN	[10] SECURITYVIOLATION_IRQ	Security violation for VP output (SECURE_IRQ). A security violation (for example, a secure video pipeline connected to a non-secure VP/OVR) has occurred.
VPSYNC_IRQ	[11] VPSYNC_EN	[11] VPSYNC_IRQ	[11] VPSYNC_EN	[11] VPSYNC_IRQ	Sync for VP output (FUNC_IRQ). Shadow to work copy of registers associated with VP1 has occurred. DSS0_VP_CONTROL[5] GOBIT register bit is cleared. This interrupt can trigger under the following cases: 1. NORMAL Operation: OVRn is connected to VPn. VPSYNC indicates sync of VPn. 2. Overlay Cascade: OVRn is connected to VPk. VPSYNC indicates sync of VPk. 3. Overlay M2M: OVRn is connected to WB. VPSYNC indicates sync event (WB start) of WB.

Copyright © 2024 Texas Instruments Incorporated

Table 12-335. DISPC Interrupts - Second Level - VP3 and VP4 Outputs (continued)

Interrupt Name	VP3 Interrupt Mask DSS0_COMMON_VP_IRQENABLE_2	VP3 Interrupt Status DSS0_COMMON_VP_IRQSTATUS_2	VP4 Interrupt Mask DSS0_COMMON_VP_IRQENABLE_3	VP4 Interrupt Status DSS0_COMMON_VP_IRQSTATUS_3	Description
VPSAFETYREGION1_IRQ	[16-13] SAFETYREGION1_EN Bit [16] = Internal Diagnostic Region 7 Bit [15] = Internal Diagnostic Region 6 Bit [14] = Internal Diagnostic Region 5 Bit [13] = Internal Diagnostic Region 4	[16-13] SAFETYREGION1_IRQ Bit [16] = Internal Diagnostic Region 7 Bit [15] = Internal Diagnostic Region 6 Bit [14] = Internal Diagnostic Region 5 Bit [13] = Internal Diagnostic Region 4	[16-13] SAFETYREGION1_EN Bit [16] = Internal Diagnostic Region 7 Bit [15] = Internal Diagnostic Region 6 Bit [14] = Internal Diagnostic Region 5 Bit [13] = Internal Diagnostic Region 4	[16-13] SAFETYREGION1_IRQ Bit [16] = Internal Diagnostic Region 7 Bit [15] = Internal Diagnostic Region 6 Bit [14] = Internal Diagnostic Region 5 Bit [13] = Internal Diagnostic Region 4	VP output MISR signature mismatch, or VP output freeze frame detect (SAFETY_ERROR_IRQ). The MISR signature generated does not match the expected signature, or the VP output frame freeze detection has triggered.

Copyright © 2024 Texas Instruments Incorporated

12.6.3.6 DISPC DMA Controller

The DISPC DMA controller is based on a 5-channel DMA engine that:

- Requests and supplies data from system memory to the VID and VIDL pipelines (up to 4 input layers) through the SoC system interconnect, based on the configuration of the pipeline settings.
- Stores the composed frames back into system memory by using the WB pipeline.
- Is fully programmable and fetches pixel data via 128-bit requests using 1D burst.

The DISPC DMA engine has an additional (secondary) 128-bit master port. By default, all transactions from every channel go through the primary 128-bit master port. If compression is enabled for a particular channel (by using the FBDC module), then all the transactions of that channel go through the secondary master port. It is also possible to force transactions from a particular channel to the secondary master port (even without compression) by setting the DSS0_VID_ATTRIBUTES2[25] MPORTSEL register bit to 0x1. This is possible only when compression through the FBDC is completely disabled (that is, none of the read channels are fetching compressed data).

Each pipeline has a dedicated DMA buffer and channel with independent settings. If a pipeline is disabled (that is, not used), then its DMA buffer can be assigned to another pipeline by configuring the DSS0_COMMON_DISPC_GLOBAL_BUFFER register. For example, unused buffers for a VIDL pipeline can be used by a VID pipeline.

Each DMA channel supports a total of 8 line buffers, each of which can store 2048 32-bit pixels.

- The size of each DMA buffer associated to the VID, VIDL and WB pipelines is 8x512x128-bit.

The DMA engine fetches encoded pixels from the system memory only when the video layer is enabled (a valid configuration has been programmed for the video layer), that is, when the video window is present and the video pipeline is active.

12.6.3.6.1 DISPC DMA Addressing and Bursts

For each line to be fetched, the DMA engine address generator:

- Calculates the pixel address
- Aligns the address
- Determines byte enable pattern
- Determines 1D burst length

The meaning of the address bits is as follows:

- Bits [31-0]: system (DDR) memory address (pixel address)
- Bits [47-32]: 32 bits + address extension

The address extension bits are defined by the programmable parameter DSS0_VID_BA_EXT_0/ DSS0_VID_BA_EXT_1 and DSS0_VID_BA_UV_EXT_0/DSS0_VID_BA_UV_EXT_1 registers, optionally used to extend the BA memory addressing to 48-bit addressed external memory space (that is, to extend DISPC address space into 4GB+ space).

The DDR scan pixel addresses are generated by the DMA engine in order to read data from the system memory. The base address defines the start address of the first pixel, and then the address is incremented based on the number of pixels per line, offset between two consecutive lines and number of lines. The DSS0_VID_ROW_INC register allow the access to a frame using 1D bursts (but as a two-dimensional block) by adding a fixed address offset at the end of a line. The ROW_INC can also be used to skip lines from the input frame.

The byte address of each pixel in the frame buffer located in the system memory is determined by:

Pixel address = base_address + x × (bpp/8) + y × (width × (bpp/8) + increment), where:

- "base_address" corresponds to the base address defined by:
  - DSS0_VID_BA_0/DSS0_VID_BA_1[31-0] BA register bit-fields for all formats, and Y frame buffer in case of YUV-NV12 format. Optional extension bits in DSS0_VID_BA_EXT_0/DSS0_VID_BA_EXT_1 registers.
  - DSS0_VID_BA_UV_0/DSS0_VID_BA_UV_1[31-0] BA register bit-fields for UV frame buffer in case of YUV-NV12 format. Optional extension bits in DSS0_VID_BA_UV_EXT_0/DSS0_VID_BA_UV_EXT_1 registers.

Copyright © 2024 Texas Instruments Incorporated

- "bpp" corresponds to the number of bits per pixel defined by the DSS0_VID_ATTRIBUTES[6-1] FORMAT register bit-field.
- "width" corresponds to the number of pixels per line defined by the DSS0_VID_PICTURE_SIZE[11-0] MEMSIZEX + 1 register bit-field.
- "increment" corresponds to the number of bytes to skip between two contiguous lines defined by the DSS0_VID_ROW_INC [31-0] ROWINC – 1 register bit-field.
- "x" corresponds to the pixel position on the x-axis.
- "y" corresponds to the pixel position on the y-axis.

In cases where the DMA controller is doing a flip/mirror (see Section 12.6.3.6.4, *DISPC Flip/Mirror Support*) or fetching a compressed frame buffer (see *DISPC Compressed Data Format Support*), the above equation for pixel byte address needs to be modified according to those features.

Note

Since the base address is aligned on pixel size boundary the horizontal resolution is one pixel. In case of YUV422 formats, the resolution is 4 bytes (2 pixels). In case of RGB24 packed format the resolution is 4 pixels. In case of Y frame buffer (YUV-NV12 format) the resolution is one byte. The vertical resolution is one line.

In case of YUV422 non-planar format, the number of pixels per line shall be a multiple of 2 pixels and the size of a pixel shall be considered as 2 bytes. In case of YUV planar format (YUV420-NV12, YUV420-NV21, YUV422-NV12, YUV422-NV21), the Y buffer shall be considered as an 8-bit frame buffer, and the CbCr shall be considered as a 16-bit frame buffer. The pixel size is 1 byte and 2 bytes respectively for Y and CbCr.

For two-plane pixel formats (YUV420-NV12, YUV420-NV21, YUV422-NV12, YUV422-NV21, RGB565-A8), the pixel values are defined in two separate buffers (Y and UV buffers). The first buffer consists of Y values (8 bits for each Y sample) or 16-bit RGB value. The second buffer consists of CbCr values (16 bits for each pair of CbCr samples) or 8-bit Alpha value. The base address, the number of bytes to skip between pixels and between lines of each buffer are defined by separate registers:

- The DSS0_VID_BA_0/DSS0_VID_BA_1, DSS0_VID_PIXEL_INC and DSS0_VID_ROW_INC registers define the values to use for the first buffer
- The DSS0_VID_BA_UV_0/DSS0_VID_BA_UV_1 and DSS0_VID_ROW_INC_UV define the values to use for the second buffer
- Internal to the DMA controller, the two planes use the same shared memory space, which is allocated to the relevant pipeline which has the two-plane pixel format set

In case of interlaced mode, DSS0_VID_BA_0 and DSS0_VID_BA_UV_0 registers define the base address of the even field, and DSS0_VID_BA_1 and DSS0_VID_BA_UV_1 registers define the base address of the odd field.

Table 12-336 summarizes the register settings for a simple access of a picture in the system memory.

Table 12-336. DISPC Register Settings for Accessing Image in Internal Memory

Pipeline Registers	Value
DSS0_VID_BA_0[(1)] and DSS0_VID_BA_1[(2)] / DSS0_WB_BA_0 and DSS0_WB_BA_1	The physical base address (PBA) of image in the memory for all formats and Y buffer.
DSS0_VID_BA_EXT_0 and DSS0_VID_BA_EXT_1 / DSS0_WB_BA_EXT_0 and DSS0_WB_BA_EXT_1	Address extension bits of PBA for all formats and Y buffer.
DSS0_VID_BA_UV_0[(1)] and DSS0_VID_BA_UV_1[(2)] / DSS0_WB_BA_UV_0 and DSS0_WB_BA_UV_1	The physical base address (PBA) of UV buffers image in the memory.
DSS0_VID_BA_UV_EXT_0 and DSS0_VID_BA_UV_EXT_1 / DSS0_WB_BA_UV_EXT_0 and DSS0_WB_BA_UV_EXT_1	Address extension bits of PBA for UV buffers.
DSS0_VID_PIXEL_INC / DSS0_WB_PIXEL_INC	1 or other in pixel incremental value.
DSS0_VID_ROW_INC / DSS0_WB_ROW_INC	1 or other in row incremental value. Used for Y buffer.

Copyright © 2024 Texas Instruments Incorporated

Table 12-336. DISPC Register Settings for Accessing Image in Internal Memory (continued)

Pipeline Registers	Value
DSS0_VID_ROW_INC_UV / DSS0_WB_ROW_INC_UV	1 or other in row incremental value. Used for UV buffer.

(1) The BA_0 and BA_UV_0 registers define the base address of even field, in case of interlaced mode.
(2) The BA_1 and BA_UV_1 registers define the base address of odd field, in case of interlaced mode.

An interconnect request (128 bits) corresponds to one or several pixels, depending on the bits per pixel. Therefore, the DMA engine determines the appropriate burst sequence to optimize the fetching of each new line. The DMA engine must prevent a single burst from crossing two lines. The DMA engine supports only 1D burst. 1D burst is used, if the fetch data is linear in memory. The size of the burst can be one of the following values:

- 1x128-bit, 2x128-bit, 4x128-bit: Only during start of the row and end of the row, if there is a misaligned address
- 8x128-bit: Steady state burst size
- 16x128-bit: Only while accessing a compressed frame buffer

The DMA controller supports the start address of a row of pixels to be aligned to any arbitrary boundary in memory (with the restriction that a 32-bit format is aligned to 32-bit boundary, a 16-bit format is aligned to a 16-bit boundary, a 8-bit format is aligned to a 8-bit boundary, etc.). However, it operates most efficiently when the start address of a row is aligned to the max burst boundary of 8x128-bit. For all other cases of unaligned rows the DMA will need to go through non-optimal transactions at the start and end of each row, till it can issue a max burst of 8x128.

While accessing a compressed frame buffer (through the FBDC module), the base-address needs to be aligned to 256 bytes. In this case, the DMA will always issue a burst of 16x128-bit.

12.6.3.6.2 DISPC Read DMA Buffers

The read DMA buffers are used by the VID and VIDL pipelines.

When the vertical front porch (VFP) period starts after the last horizontal front porch (HFP) of the last line, the DMA buffers are flushed according to the video port output associated with the particular video pipeline. The DMA engine restarts fetching new frame data from the memory through the DISPC master port. Enabling or disabling the DISPC also flushes the DMA buffers.

Programmable high and low thresholds, independent for each DMA buffer, are used by the DMA engine to start and stop requesting data thourgh the master port.

- When low threshold (set in the DSS0_VID_BUF_THRESHOLD [15-0] BUFLOWTHRESHOLD register bit-field) is reached, the DMA engine starts a request on the device interconnect to fill the DMA buffer.
- When high threshold (set in the DSS0_VID_BUF_THRESHOLD [31-16] BUFHIGHTHRESHOLD register bit-field) is reached, the DMA engine stops requesting encoded pixels.

Note

The configuration of thresholds for optimal performance can be defined using the DSS0_VID_BUF_SIZE_STATUS[15-0] BUFSIZE register field value, as follows:

- For high threshold: BUFSIZE (in number of 128-bit words) – 1
- For low threshold: BUFSIZE (in number of 128-bit words) – burst size (in number of 128-bit words)

The following limitations for BUFLOWTHRESHOLD values must be also considered:

- If the scaler in VID pipeline is disabled, BUFLOWTHRESHOLD can be programmed as low as interconnect latency and pixel output rate allow it.
- If the scaler in VID pipeline is enabled, BUFLOWTHRESHOLD must be programmed to guarantee that at least four full lines can be stored.

To avoid underflow at the beginning of a frame and have sufficient encoded pixel data to start some processing, a preloading of the DMA buffer is configurable between a fixed value of bytes and the high threshold value. When the preload value is reached, the associated channel will start pulling pixels out of the DMA buffer. To enable the preload based on the value entered in the DSS0_VID_PRELOAD[11-0] PRELOAD register bit-field, the DSS0_VID_ATTRIBUTES[19] BUFPRELOAD register bit must be set to 0x0.

Copyright © 2024 Texas Instruments Incorporated

The vertical blanking between two frames must be long enough to allow fetching the number of pixels defined by the DSS0_VID_PRELOAD register and preloading the whole video pipeline. If the value set in the preload register is greater than some overflow conditions detected by the hardware, then data will start to be read from the video DMA buffer before the preload value is reached. If SYNCLOST_IRQ event occurs the video buffer needs to be increased (buffer merge). Preload value must be greater or equal to low threshold, and smaller or equal to high threshold value.

Note

When self-refresh mode is selected (which means that the data in the DMA buffer are used for multiple frames) the DMA buffers are not flushed at the end of each frame. Each DMA buffer has an independent control for selecting the self-refresh mode. For more information, see Section 12.6.3.6.10, *DISPC DMA Ultra-Low Power Mode*.

12.6.3.6.3 DISPC Write DMA Buffer

The write DMA buffer is used by the WB pipeline.

Two modes for filling the DMA buffer are supported by the WB channel, selectable through the DSS0_WB_ATTRIBUTES[19] WRITEBACKMODE bit:

- Capture mode, WRITEBACKMODE bit set to 0: One of the overlay outputs connected to an external interface is captured at the same time the data are sent on the output. The WB timings are controlled by the VP timings.
- Memory-to-memory mode, WRITEBACKMODE bit set to 1: One of the overlay outputs or one of the pipelines is captured to perform a memory-to-memory transfer, with some processing by the DISPC (rescaling, overlaying, color space conversion, etc.).

In capture mode: Transfer is synchronous (that is, tightly coupled) to one of the display outputs. In this mode, the write back buffer starts receiving data after DSS0_WB_ATTRIBUTES[0] ENABLE register bit has been set and the associated output vertical sync has triggered the start of the transfer (VFP). In addition, the WB ENABLE bit shall be set prior to the enable of the captured channel output in order to capture the first frame.

In memory-to-memory mode: The data transfer is not synchronous to any of the display outputs. In this mode, the write back buffer starts receiving data after the DSS0_WB_ATTRIBUTES[0] ENABLE has been set. The WB ENABLE bit also is triggering the update of the shadow register in the WB pipeline.

Programmable high and low thresholds are used by the DMA engine to start and stop sending data to the system interconnect.

- When high threshold (set in the DSS0_WB_BUF_THRESHOLD[31:16] BUFHIGHTHRESHOLD bit field) is reached, and there are enough data for at least one burst of pixels in the DMA buffer ready for transfer, the DMA engine generates a request to the arbitration logic. The size of the burst is defined by the user.
- When low threshold (set in the DSS0_WB_BUF_THRESHOLD[15:0] BUFLOWTHRESHOLD bit field) is reached, the DMA engine stops sending encoded pixels.

At the end of the frame, to completely drain the DMA buffer, some smaller bursts (even single requests) may have to be issued. To limit the number of interconnect requests from the DISPC (that is, to limit the throughput of the write-back channel to the memory), a number of IDLE cycles between requests can be inserted. IDLE cycles can be inserted only when WB is used in memory-to-memory mode. It is ignored when WB is in capture mode.

The number of IDLE cycles between requests can be activated and determined by:

- Setting the DSS0_WB_ATTRIBUTES[27] IDLESIZE bit to 0x0 (default value) and entering the number of idles between requests in the DSS0_WB_ATTRIBUTES[31:28] IDLENUMBER bit field. Idle numbers vary from 0 to 15.
- Setting the DSS0_WB_ATTRIBUTES[27] IDLESIZE bit to 0x1, which considers the size of the burst (the DSS0_WB_ATTRIBUTES[15:14] BURSTSIZE bit field) to determine the number of IDLE cycles.
  - If BURSTSIZE = 0x0, then the number of IDLE cycles equals IDLENUMBER (0 to 15).
  - If BURSTSIZE = 0x1, then the number of IDLE cycles equals IDLENUMBER × 4 (0 to 60).
  - If BURSTSIZE = 0x2, then the number of IDLE cycles equals IDLENUMBER × 8 (0 to 120).

Copyright © 2024 Texas Instruments Incorporated

12.6.3.6.4 DISPC Flip/Mirror Support

The DISPC supports on-the-fly source image flip along the x/y-axis for 8-bit/component formats (ARGB or YUV) to create a mirror effect on the source data. The DMA engine reads the source frame from right to left by requesting burst transfers for each line in negative address increments while performing each burst transfer as a linear incremental burst. The read data is repacked and stored in the line buffer incrementally - effectively storing the frame as a flipped image for the processing pipeline. The configuration of the flip/mirror operation is explained in Table 12-337.

Table 12-337. DISPC Flip/Mirror Configuration

Flip Direction	FLIP Bit	Base Address	Byte Increment
Along Y-axis (vertical mirror)	1	Start of window	1
Along X-axis (horizontal mirror)	0	Start of last line of the window	-(2*(width of the line in bytes))
Along both X-axis and Y-axis (horizontal and vertical mirror)	1	Start of last line of the window	-(2*(width of the line in bytes))

In Table 12-337:

- The FLIP bit is located in DSS0_VID_ATTRIBUTES register.
- The base address of the video buffer must be configured as explained in Section 12.6.3.6.1, *DISPC DMA Addressing and Bursts*.
- The the number of bytes to increment at the end of the row in the video buffer can be configured through DSS0_VID_ROW_INC and DSS0_VID_ROW_INC_UV registers. For more information, see Section 12.6.3.6.1, *DISPC DMA Addressing and Bursts*, and Section 12.6.3.6.5, *DISPC DMA Predecimation*.

The flip/mirror feature is not supported for compressed data formats (when the DMA engine receives daat dreom the FBDC module).

12.6.3.6.5 DISPC DMA Predecimation

The predecimation process consists of downscaling an image by fetching only the necessary pixels out of the memory. Vertical and horizontal predecimation are possible:

- Vertical predecimation: The picture stored in memory can be predecimated vertically by skipping lines. Burst mode is used to fetch the data when skipping lines. Only the lines that will be used by the DISPC are fetched from memory; the other lines are skipped. The DMA engine sends requests only for the useful lines using 1D burst. The base address indicates the first valid pixel to fetch from memory. The number of lines to skip is set in the ROW_INC and ROW_INC_UV registers (see Section 12.6.3.6.1, *DISPC DMA Addressing and Bursts*).
- Horizontal predecimation: When fetching data from memory, it is possible to skip 1 out of 2 pixels, up to 1 of ouf 2047 pixels, by setting the PIXEL_INC register (see Section 12.6.3.6.1, *DISPC DMA Addressing and Bursts*) to the number of pixels to skip (n), multiplied by the size of a pixel (in bytes), +1. The condition to generate a burst is that there is at least one useful pixel per 128-bit OCP request. Therefore, when the pixels are 16/32-bit, the maximum number of pixels that can be skipped is 8/4. If *PixelWidthInBytes* + *BytesToSkip* is greater than 16, then the programmed burst is changed into single request.

No decimation is supported when the input format is 1, 2, 4, or 8-bit BITMAP.

For RGB and YUV420 data formats, each pixel data container in memory holds 1 pixel. Thus, when configuring the PIXEL_INC register, the value of n equals the number of pixels to skip:

- For RGB format, one pixel data container = 32 bits = 1 pixel
- For YUV format:
  - One Y pixel data container = 8 bits = 1 pixel
  - One UV pixel data container = 16 bits = 1 pixel

For YUV422 format, each 32-bit pixel data container holds the Luma components for 2 pixels, and the Chrominance component of 1 pixel (see Figure 12-313). Therefore, for the valid values of the PIXELINC bit field in the case of the following YUV422 format, caution must be taken because n equals the number of pixel data containers to skip, and not the number of pixels:

- For n = 1, PIXELINC = 5
- For n = 2, PIXELINC = 9

Copyright © 2024 Texas Instruments Incorporated

- For n = 3, PIXELINC = 13
- For n = 4, PIXELINC = 17, etc.

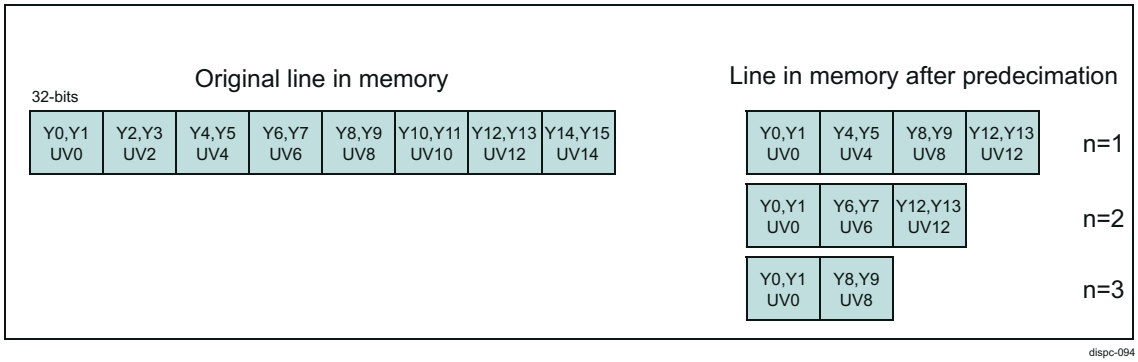


Figure 12-313. DISPC YUV422 Predecimation

12.6.3.6.6 DISPC DMA Buffer Sharing

The DISPC DMA controller supports dynamic re-partition of the memory among the different channels.

At reset, each channel in DMA is allocated a uniform buffer size of 64KB. This 64KB is further partitioned into four 16KB blocks as shown in Figure 12-314. A 64KB buffer is enough to support ping-pong mode for resolutions up to 4K for YUV420/YUV422 formats and up to 2K for ARGB32 formats.

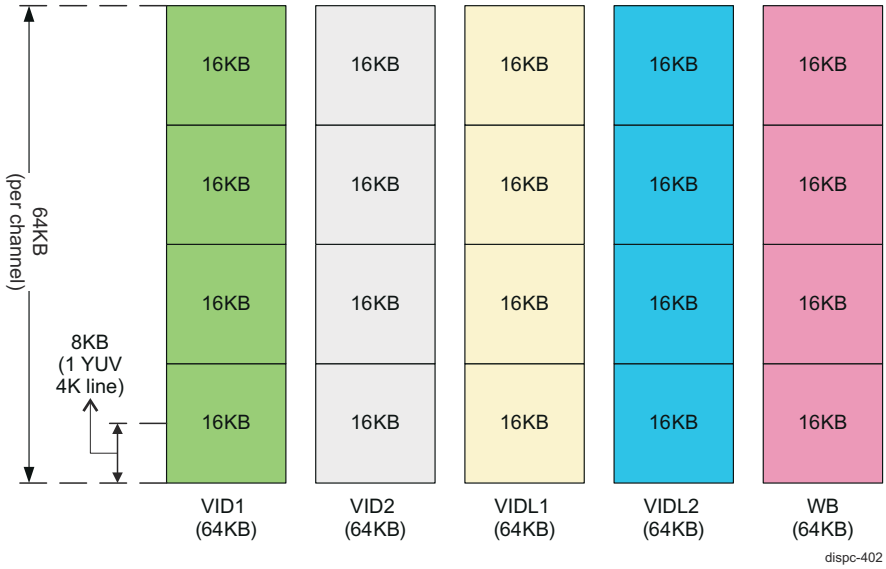


Figure 12-314. DISPC DMA Channel Memory Allocation at Reset

To support resolutions larger than 4K YUV or 2K ARGB (with ping-pong configuration), the default buffer size of 64KB per read channel is not sufficient. In such cases, the DISPC allows dynamic repartition of the allocated buffers across the different read channels. In this way each read channel can have a variable buffer size allocated to it, starting from a minimum of 16KB to a maximum of 256KB, in 16KB increments. The WB channel buffers cannot be re-allocated and remain fixed. This is the case even when WB channel is disabled. A particular configuration (for a specific example use-case) is as shown in Figure 12-315.

Copyright © 2024 Texas Instruments Incorporated

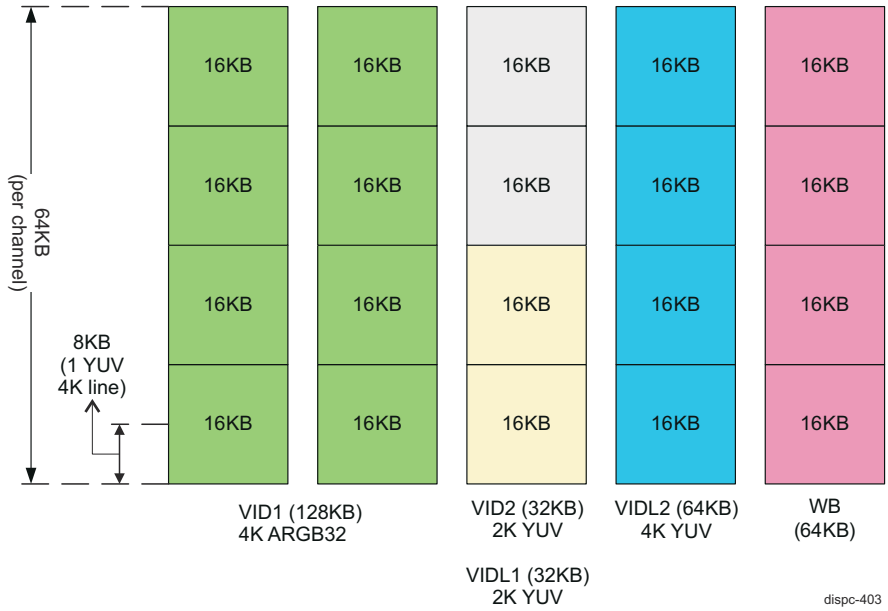


Figure 12-315. DISPC Dynamic Buffer Repartition between DMA Read Channels

When the size of the buffer is changed, the thresholds shall be re-programmed by the user to reflect the new DMA buffer configuration.

The activity on a read channel is controlled by the timing of the Overlay/VP to which the channel is connected (or to the WB, if the channel is connected to WB in M2M mode). Due to this, the DISPC HW ensures that any change in a channel buffer setting is synchronized with regard to the Overlay/VP or WB (M2M mode) to which the channel is connected. In addition, the software also must follow the following programming sequences:

- Initialization sequence (before enabling any pipe or VP or WB):
  1. Allocate the total available memory among the different threads (configurable through the DSS0_COMMON_GLOBAL_DMA_THREADSIZE[19-0] VP#nTHREADSIZE and [24-20] WBTHREADSIZE register bit-fields).
  2. Wait for the DISPC HW to sync to the programmed THREADSIZE (read the value from DSS0_COMMON_GLOBAL_DMA_THREADSIZESTATUS[19-0] VP#nTHREADSIZE and [24-20] WBTHREADSIZE register bit-fields).
  3. Allocate a buffer for each of the (active) read channels (configurable through the DSS0_VID_DMA_BUFSIZE[4-0] BUFSIZE register field). Software needs to ensure that the total size allocated to all the channels in a thread does not surpass the total size allocated to the thread.
  4. Enable the read channels as well as the VP output.
- Intra thread re-allocation sequence. Within the thread the re-allocation always happens at the next VSYNC. There is no handshake needed on the part of software.
  1. Allocate the new buffer size (configurable through the DSS0_VID_DMA_BUFSIZE[4-0] BUFSIZE register field).
- Inter thread re-allocation sequence. The synchronization events (VSYNC) between two threads will be different. Therefore, any re-allocation between two threads needs to go through a software managed hand-shake sequence as follows:
  1. Free the required amount of memory from Thread-#n (configurable through the DSS0_COMMON_GLOBAL_DMA_THREADSIZE register).
  2. Wait till the synchronization event of Thread-#n (achieved by looking at the status reflected in the DSS0_COMMON_GLOBAL_DMA_THREADSIZESTATUS register).
  3. Allocate the freed buffer space to a different thread, Thread-#m (configurable through the DSS0_COMMON_GLOBAL_DMA_THREADSIZE register).
  4. Further allocate this space to the different read channels of Thread-#m (configurable through the DSS0_COMMON_GLOBAL_DMA_THREADSIZESTATUS register).

Copyright © 2024 Texas Instruments Incorporated

5. Further allocate the new buffer size to the read channels in Thread-#m (configurable through DSS0_VID_DMA_BUFSIZE register).

Note

In a case where a channel is connected to multiple Overlay/VP (multi-cast use-case), then the channel belongs to the thread corresponding to the first (lowest numbered) Overlay/VP to which it is connected.

12.6.3.6.7 DISPC DMA MFLAG Mechanism

The MFLAG mechanism allows a dynamic increase of the priority of DISPC real-time traffic, when required, based on the fullness of the DISPC DMA read buffers.

The MFLAG mechanism is used when fullness of the DMA buffers is critical (close to underflow). The mechanism is implemented for all DMA buffers of the video pipelines.

Programmable buffer thresholds (forming hysteresis) are used to configure when a local MFLAG signal is generated. The 1-bit MFLAG signal is generated on DISPC master port in order to inform the system that the outstanding requests from DISPC shall be considered with higher priority in order to get faster interconnect responses. The MFLAG signal is asynchronous to any ongoing interconnect transaction.

Each pipeline maintains its own MFLAG bit. The MFLAG bit is asserted depending on the fullness of the DMA buffers associated with the pipeline and depending on the thresholds programmed by software. All MFLAG signals are OR-ed together to generate the single 1-bit MFLAG for the master port connected to the DISPC DMA engine.

The threshold for video pipelines corresponds to the fullness of the associated DMA buffer, and is defined by two threshold parameters:

- HT_MFLAG: High threshold.
  - For read access from video pipelines, when the pipeline buffer reaches the programmed value, the associated local MFLAG signal goes low (deasserted).
  - This threshold can be programmed in the DSS0_VID_MFLAG_THRESHOLD [31-16] HT_MFLAG register field.
- LT_MFLAG: Low threshold.
  - For read access from video pipelines, when the pipeline buffer reaches the programmed value, the associated local MFLAG signal goes high (asserted).
  - This threshold can be programmed in the DSS0_VID_MFLAG_THRESHOLD [15-0] LT_MFLAG register field.

Summary of the MFLAG value, based on DMA read buffer fullness:

- If DMA read buffer fullness < LT_MFLAG, then MFLAG signal = 1
- If LT_MFLAG < DMA read buffer fullness < HT_MFLAG, then MFLAG signal = 1
- If DMA read buffer fullness > HT_MFLAG, then MFLAG signal = 0

Similarly, the MFLAG is set based on fullness and the transition history on the out bound writeback pipeline. There is no pre-fetch state for out bound DMA controller. The MFLAG is set when the buffer fullness rises above the HT_MFLAG bit-field value. The MFLAG is cleared only when the buffer fullness falls back below the LT_MFLAG bit-field value. In between these two fullness states, the MFLAG bit keeps the previous value.

By default, the MFLAG mechanism is disabled (DSS0_COMMON_DISPC_GLOBAL_MFLAG_ATTRIBUTE[1-0] MFLAG_CTRL register field = 0x0), and the MFLAG signal is low (de-asserted). The arbitration scheme for the pipelines is the same as described in Section 12.6.3.6.9, *DISPC DMA Arbitration*. That is, round-robin either between high-priority pipelines, or between normal-priority pipelines (if all pipelines are of normal priority).

When the MFLAG_CTRL register field is set to 0x2, the MFLAG mechanism is enabled, and the MFLAG signal is dynamically set to 0 or 1, depending on DMA buffer fullness and programmed threshold levels, as explained previously in this section. In this case, the arbitration scheme for the pipelines is round-robin between those high-priority pipelines, which have asserted their local MFLAG signals. If there are no high-priority pipelines with

Copyright © 2024 Texas Instruments Incorporated

their local MFLAG signals asserted, then the arbitration scheme is the same as described in Section 12.6.3.6.9, *DISPC DMA Arbitration*.

The DSS0_COMMON_DISPC_GLOBAL_MFLAG_ATTRIBUTE[6] MFLAG_START bit defines the following additional rules for the MFLAG mechanism:

- If the MFLAG_START bit is set to 0x0 (default value), then when the DMA buffer is empty at the beginning of the frame, the MFLAG signal of each pipeline is kept at 0 until PRELOAD is reached (for more information on preloading, see Section 12.6.3.6.2, *DISPC Read DMA Buffers*). Then, based on the setting of the DSS0_COMMON_DISPC_GLOBAL_MFLAG_ATTRIBUTE[1-0] MFLAG_CTRL register field, the MFLAG signal is generated and DISPC internal logic is arbitrating between pipeline requests.
- If the MFLAG_START bit is set to 0x1, then even in the beginning of the frame when the DMA buffer is empty, the configuration of DSS0_COMMON_DISPC_GLOBAL_MFLAG_ATTRIBUTE[1-0] MFLAG_CTRL register field determines the generation of the MFLAG signal.

12.6.3.6.8 DISPC DMA Priority Requests Control

The DSS0_COMMON_DSS_CBA_CFG register controls the priority level for DMA requests going out to the memory through the system interconnect.

As explained in Section 12.6.3.6.7, *DISPC DMA MFLAG Mechanism*, the DISPC master port generates a 1-bit MFLAG output signal to raise the priority of all requests made on that port, if any of its DMA buffers runs critically low (determined by a set of user programmable threshold values for each buffer).

DSS uses the MFLAG signal from DISPC to set a 3-bit priority level output (*Mpriority*) for the master port to either a low or high value (configurable in DSS0_COMMON_DSS_CBA_CFG[2-0] PRI_LO and [5-3] PRI_HI register fields with optional values of 0~7) as follows:

- When MFLAG = 0, the PRI_LO register field determines the value of the Mpriority output for the normal transactions.
- When MFLAG = 1, the PRI_HI register field determines the value of the Mpriority output for the high-priority transactions.

This Mpriority output directly drives the respective input of the system interconnect port, which corresponds to the DISPC DMA master port.

CAUTION

Upon a hardware reset, DSS0_COMMON_DSS_CBA_CFG[2-0] PRI_LO and [5-3] PRI_HI register fields are set by default to 4 and 1, respectively. Afterwards, these priority level register fields can only be modified by a secure host.

12.6.3.6.9 DISPC DMA Arbitration

The read requests sent to the system interconnect are pipelined and arbitrated in a round-robin scheme. The default arbitration scheme can be modified by setting the priority attribute of each pipeline as defined in the DSS0_VID_ATTRIBUTES[23] ARBITRATION register bit.

By default, all pipelines have the same priority (normal priority), which means all pipeline requests are treated in a round-robin order manner. If one or more pipelines require a higher number of requests going to the system interconnect, its priority can be moved up to high priority. In this case, the high-priority pipeline is granted access before any pipeline in normal priority. If more than one active pipeline is in high priority, then the behavior is the same as all active pipelines in normal priority. Normal active pipelines are not treated until all high active pipelines are finished. The ARBITRATION bit cannot be modified during the entire frame.

In addition to the priority bit-field, the MFLAG mechanism can also result in a higher priority for a pipeline. An MFLAG priority level is set, for all pipelines for which MFLAG bit is set, which is defined as the highest priority level for arbitration. For more details on MFLAG mechanism, see Section 12.6.3.6.7, *DISPC DMA MFLAG Mechanism*. Figure 12-316 shows the transition between the different priority levels.

Copyright © 2024 Texas Instruments Incorporated

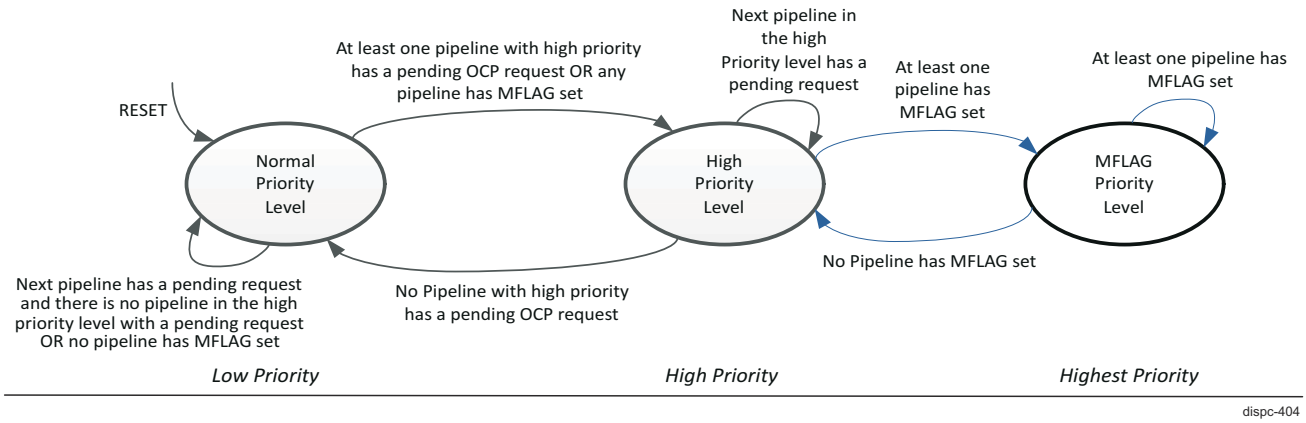


Figure 12-316. DISPC DMA High/Low Priority Arbitration

12.6.3.6.10 DISPC DMA Ultra-Low Power Mode

In ultra-low power mode, the system interconnect is used to fill up the DMA buffers to store all the data required to display a full frame. Then, the system interconnect is not used anymore to fetch new pixels for the consequent frames. The data are fetched once into the DMA buffer and then the following frames re-use the DMA buffer to display on the screen.

The programming of the ultra-low power mode is independent for each pipeline and is achieved via the DSS0_VID_ATTRIBUTES[24] SELFREFRESH register bit. One pipeline may have all frame pixels into the DMA buffer and other pipeline may have to refill the DMA buffers along the display scan, because the frame buffer is too big to be stored in the DMA buffer.

The DMA buffers can be merged in order to optimize the system interconnect OFF time. Each DMA buffer dedicated to one pipeline can be split into two buffers. The merge of the DMA buffers into a single one can be used at the same time to improve the Ultra-Low Power mode.

Two ultra-low power modes can be entered, manual or automatic mode:

- Manual self-refresh mode: Starting self-refresh mode is done manually by setting the SELFREFRESH bit to 0x1 after capturing a frame in the DMA buffers. Self-refresh mode is stopped by setting the SELFREFRESH bit to 0x0. Software must first disable the SELFREFRESH bit during at least one frame in order to capture the data (by setting the GOBIT register bit of the video port the pipeline is associated with). Once the DSS0_VP_CONTROL[5] GOBIT bit has been set, the software must read the GOBIT bit to ensure that the frame has been loaded into the buffer, then it can set the SELFREFRESH bit to 1. Once SELFREFRESH is enabled, the fetch of data from the system memory is stopped for the following frames. The software needs to reset the SELFREFRESH bit in order to restart fetching data from system memory.
- Automatic self-refresh mode: By setting the DSS0_VID_ATTRIBUTES[17] SELFREFRESHAUTO bit to 0x1, the transition from disabled to enabled for self-refresh mode is controlled by hardware. This allows the software to reset the SELFREFRESH bit to "disabled", and then automatically after the fetch of the first frame the hardware switches back SELFREFRESH bit to "enabled". The SELFREFRESH must be disabled during at least one frame in order to capture the data, so software must reset the bit to 0 every time the data in the DMA buffer needs to be updated. The hardware reads the data inside the DMA buffer without accessing the interconnect and system memory during the frame, then modifies the SELFREFRESH bit to reflect the current state of the self-refresh mode by setting the bit to 0x1.

12.6.3.6.11 DISPC Compressed Data Format Support

The DISPC is capable of reading a compressed frame buffer through the FBDC block. The FBDC is a Frame Buffer Decompression module that is compatible with the lossless compression module (FBC) in the GPU in the SoC. The FBDC performs the lossless decompression of the compressed images on a tile-by-tile basis. The size of the tiles is 16 pixels by 4 lines or 32 pixels by 2 lines. The FBDC is enabled by setting the DSS0_VID_FBDC_ATTRIBUTES[0] ENABLE register bit.

Copyright © 2024 Texas Instruments Incorporated

12.6.3.6.11.1 FBDC Tile Request

Since the compression/decompression is tile based, the DMA engine generates a 256-byte (16x128-bit) 1D burst request for each tile (a tile is a 16x4 or 32x2 2D structure of 32-bit elements) to FBDC which handles the DMA transfer and decompression of the requested tile. The de-compression phases are transparent to the DISPC hardware. Only ordering of the pixels from a returned tile is taken care by the DISPC DMA engine using 1D burst. The data returned from the FBDC, for a single tile over different OCP data-phases (DP), is as shown in Figure 12-317.

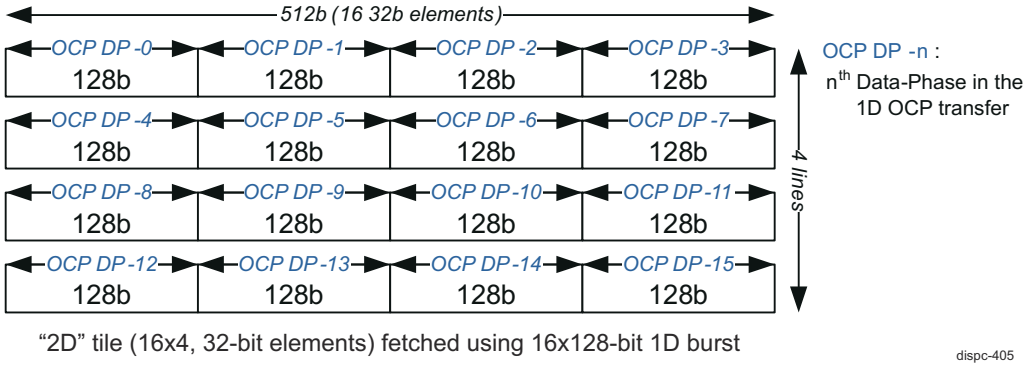


Figure 12-317. DISPC FBDC Tile Request (16x4 Tile)

Note

Due to the tile nature of the data, the following DMA features are not available with compressed data formats:

- Flip/Mirror support
- ROW_INC on a line basis (that is, ROW_INC cannot be used to skip lines). It can only be specified in such a way as to skip entire tile rows.

12.6.3.6.11.2 FBDC Source Cropping

The FBDC supports a source cropping of the compressed frame buffer. The source cropping is achieved as a two-step process.

- Cropping at a tile boundary is achieved by the DMA controller.
- Cropping within a tile (to a pixel or a line) is achieved by the VID/VIDL pipeline. That is a regular crop feature in the VID/VIDL pipeline which is useful for non-compressed frames as well.

Copyright © 2024 Texas Instruments Incorporated

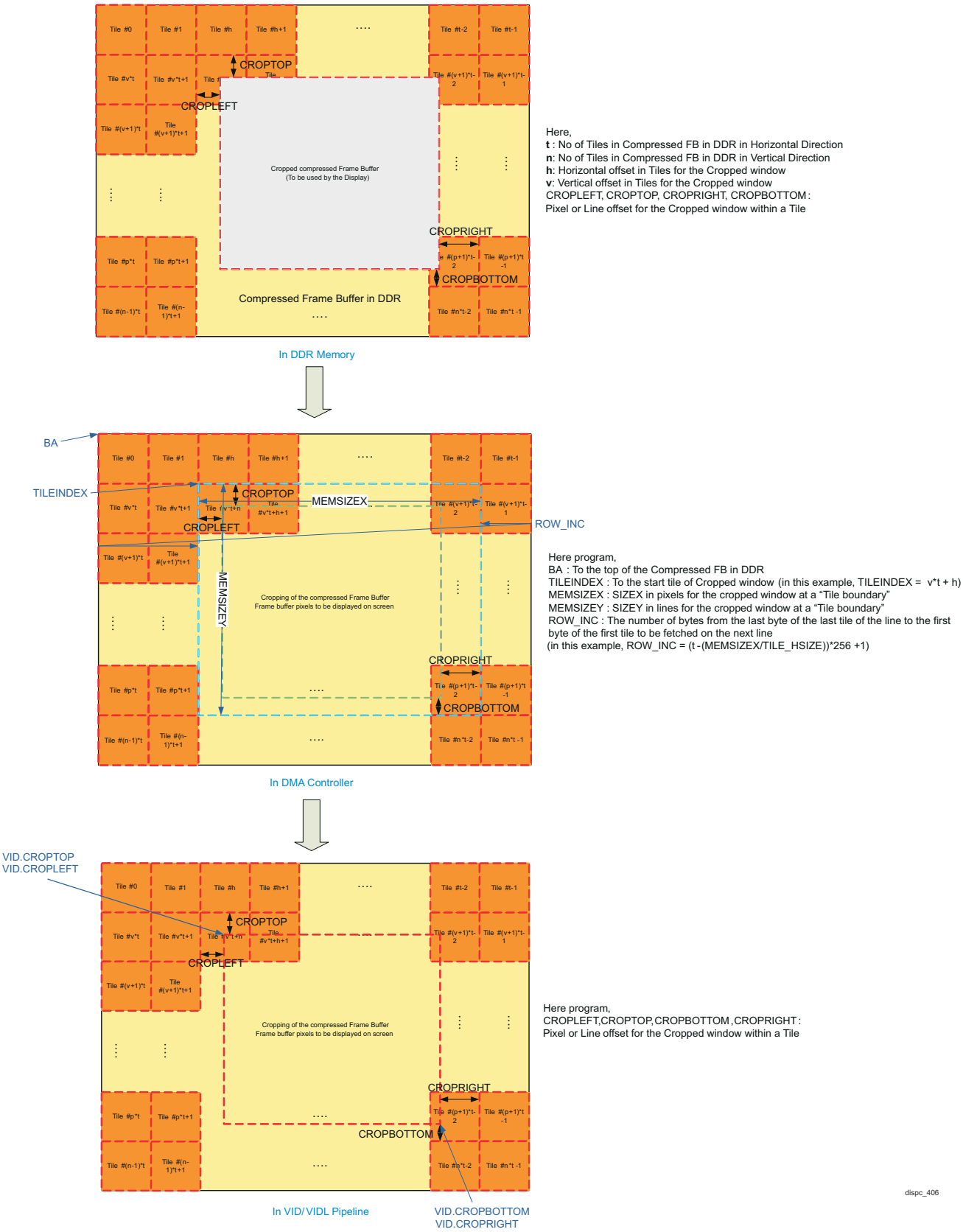


Figure 12-318. DISPC FBDC Example of a Frame Buffer Source Cropping (Decompression)

Copyright © 2024 Texas Instruments Incorporated

The necessary programming for cropping at a tile boundary within the DMA Controller (based on Figure 12-318) is as follows:

- BA must be programmed to the top of the compressed frame buffer in DDR.
- TILEINDEX must be programmed to the start tile of the cropped window in the DSS0_VID_TILE[22-0] TILEINDEX register field. In this example, TILEINDEX = v*t + h.
- MEMSIZEX is the SIZEX in pixels for the cropped window at a "Tile boundary".
- MEMSIZEY is the SIZEY in lines for the cropped window at a "Tile boundary".
- ROW_INC is the number of bytes from the last byte of the last tile of the line to the first byte of the first tile to be fetched on the next line. In this example, ROW_INC = (t-(MEMSIZEX/TILE_HSIZE))*256 + 1, where TILE_HSIZE = 16 or 32 depending on whether 16x4 tile-type is used or a 32x2 tile-type is used. The tile type must be programmed in the DSS0_VID_FBDC_ATTRIBUTES[9-8] TILETYPE register field.

For details on the register programming for the crop feature in the VID/VIDL pipeline, see Section 12.6.3.8.9, *DISPC VID Cropping Support*.

12.6.3.7 DISPC Pixel Data Formats

The DISPC pipelines support various types of memory formats, as listed in Table 12-338.

For BITMAP formats the nibble mode (pixels in each byte are packed in reverse order) can be enabled by setting the DSS0_VID_ATTRIBUTES[10] NIBBLEMODE register bit to 0x1. The nibble mode is suported only for the VID/VIDL pipelines, but not for the WB pipeline.

The pixel data format can be selected by loading the corresponding value from Table 12-338 in the DSS0_VID_ATTRIBUTES/DSS0_WB_ATTRIBUTES[6-1] FORMAT register bit-field.

Table 12-338. DISPC Supported Pixel Data Formats

FORMAT Register Field Value		Pixel Format[4]		Component Bit Depth
Alpha	Alpha-X	VID and VIDL Pipelines	WB Pipeline	
0x00	0x20	ARGB16-4444	ARGB16-4444	4
0x01	0x21	ABGR16-4444	ABGR16-4444	4
0x02	0x22	RGBA16-4444	RGBA16-4444	4
0x03	NA	RGB16-565	RGB16-565	5(R,B), 6(G)
0x04	NA	BGR16-565	BGR16-565	5(R,B), 6(G)
0x05	0x25	ARGB16-1555	ARGB16-1555	1(A), 5(R,G,B)
0x06	0x26	ABGR16-1555	ABGR16-1555	1(A), 5(R,G,B)
0x07	0x27	ARGB32-8888	ARGB32-8888	8
0x08	0x28	ABGR32-8888	ABGR32-8888	8
0x09	0x29	RGBA32-8888	RGBA32-8888	8
0x0A	0x2A	BGRA32-8888	BGRA32-8888	8
0x0B	NA	RGB24-888	RGB24-888	8
0x0C	NA	BGR24-888	BGR24-888	8
0x0E	0x2E	ARGB32-2101010	ARGB32-2101010	2(A),10(R,G,B)
0x0F	0x2F	ABGR32-2101010	ABGR32-2101010	2(A),10(R,G,B)
0x10	0x30	ARGB64-16161616	ARGB64-16161616	16
0x11	0x31	RGBA64-16161616	RGBA64-16161616	16
0x12	NA	BITMAP1	-	1
0x13	NA	BITMPA2	-	2
0x14	NA	BITMAP4	-	4
0x15	NA	BITMAP8	-	8
0x16	NA	RGB565A8 [1]	RGB565A8 [1]	5(R,B), 6(G), separate 8(A)
0x17	NA	BGR565A8 [1]	BGR565A8 [1]	5(R,B), 6(G), separate 8(A)

Copyright © 2024 Texas Instruments Incorporated

Table 12-338. DISPC Supported Pixel Data Formats (continued)

FORMAT Register Field Value		Pixel Format[4]		Component Bit Depth
Packed	**Planar**	**Pixel Format**	**Pixel Format**	**Component Bit Depth [5]**
0x3E	NA	YUV422-YUV2	YUV422-YUV2	8/10/12 [3]
0x3F	NA	YUV422-UYVY	YUV422-UYVY	8/10/12 [3]
NA	0x3C	YUV422-NV12	YUV422-NV12	8/10/12 [3]
NA	0x3D	YUV420-NV12	YUV420-NV12	8/10/12 [3]
NA	See [2]	YUV420-NV21 YUV422-NV21	YUV420-NV21 YUV422-NV21	8/10/12

(1) The video and writeback pipelines support an optional 8-bit Alpha plane (separate buffer), if the pixel format of the source/destination buffer is either RGB16-565 or BGR16-565.
(2) NV21 formats for YUV420/YUV422 are indirectly supported through chroma swapping during the color space conversion.
(3) The 10-bit/12-bit versions of these YUV formats are also supported in both packed and unpacked (in 16-bit container) formats. For unpacked formats, both LSB or MSB alignments are supported. These configurations are set using a separate configuration registers: DSS0_VID_ATTRIBUTES2 and DSS0_WB_ATTRIBUTES2. The default is an 8-bit packed format support.
(4) All RGB formats with alpha component include both pre/non-pre-multiplied data support. Also, alpha can be disabled to work as X (can be replaced with global alpha value).
(5) The writeback path supports only 8-bit and 10-bit YUV data formats.

Figure 12-319 shows the pixel data memory organization for the bitmap pixel formats.

Copyright © 2024 Texas Instruments Incorporated

BITMAP 1-bpp (0x12)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P31	P30	P29	P28	P27	P26	P25	P24	P23	P22	P21	P20	P19	P18	P17	P16	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0

BITMAP 2-bpp (0x13)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P15		P14		P13		P12		P11		P10		P9		P8		P7		P6		P5		P4		P3		P2		P1		P0	

BITMAP 1-bpp (0x14)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Pixel 7				Pixel 6				Pixel 5				Pixel 4				Pixel 3				Pixel 2				Pixel 1				Pixel 0			

BITMAP 1-bpp (0x15)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Pixel 3								Pixel 2								Pixel 1								Pixel 0							

For BITMPA P-1/2/4 bpp, the Nibble mode (pixels in each byte are packed in reverse order) is also supported.

BITMAP 1-bpp (0x12) – Nibble Mode

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P24	P25	P26	P27	P28	P29	P30	P31	P16	P17	P18	P19	P20	P21	P22	P23	P8	P9	P10	P11	P12	P13	P14	P15	P0	P1	P2	P3	P4	P5	P6	P7

BITMAP 2-bpp (0x13) – Nibble Mode

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P12		P13		P14		P15		P8		P9		P10		P11		P4		P5		P6		P7		P0		P1		P2		P3	

BITMAP 4-bpp (0x14) – Nibble Mode

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Pixel 6				Pixel 7				Pixel 4				Pixel 5				Pixel 2				Pixel 3				Pixel 0				Pixel 1			

disp-301

Figure 12-319. DISPC Bitmap Pixel Formats

Figure 12-320 and Figure 12-321 show the pixel data memory organization for the RGB 16-bit pixel formats.

Copyright © 2024 Texas Instruments Incorporated

ARGB16-4444 (0x00)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A1				R1				G1				B1				A0				R0				G0				B0			

xRGB16-4444 (0x20)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Unused				R1				G1				B1				Unused				R0				G0				B0			

ABGR16-4444 (0x01)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A1				B1				G1				R1				A0				B0				G1				R0			

xBGR16-4444 (0x21)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Unused				B1				G1				R1				Unused				B0				G0				R0			

RGBA16-4444 (0x02)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R1				G1				B1				A1				R0				G0				B0				A0			

RGBx16-4444 (0x22)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R1				G1				B1				Unused				R0				G0				B0				Unused			

RGB16-565 (0x03)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R1					G1					B1					R0					G0					B0						

BGR16-565 (0x04)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
B1					G1					R1					B0					G0					R0						

dispc-302

Figure 12-320. DISPC RGB 16-bit Pixel Formats 1

Copyright © 2024 Texas Instruments Incorporated

RGB565A8 (0x16) and BGR565A8 (0x17) are RGB16-565 and BGR16-565, respectively, with a separate Alpha-8bit plane.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A3								A2								A1								A0							

ARGB16-1555 (0x05)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A1	R1					G1					B1					A0	R0					G0					B0				

xRGB16-1555 (0x25)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	R1					G1					B1						R0					G0					B0				

ABGR16-1555 (0x06)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A1	B1					G1					R1					A0	B0					G0					R0				

xBGR16-1555 (0x26)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	B1					G1					R1						B0					G0					R0				

dispc-303

Figure 12-321. DISPC RGB 16-bit Pixel Formats 2

Figure 12-322 shows the pixel data memory organization for the RGB 24-bit pixel formats.

RGB24-888 (0x0B)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
B1								R0								G0								B0								+0x00
G2								B2								R1								G1								+0x04
R3								G3								B3								R2								+0x08

BGR24-888 (0x0C)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
R1								B0								G0								R0								+0x00
G2								R2								B1								G1								+0x04
B3								G3								R3								B2								+0x08

dispc-304

Figure 12-322. DISPC RGB 24-bit Pixel Formats

Figure 12-323 and Figure 12-324 show the pixel data memory organization for the RGB 32-bit pixel formats.

Copyright © 2024 Texas Instruments Incorporated

ARGB32-8888 (0x07)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A								R								G								B							

xRGB32-8888 (0x27)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Unused								R								G								B							

ABGR32-8888 (0x08)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A								B								G								R							

xBGR32-8888 (0x28)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Unused								B								G								R							

RGBA32-8888 (0x09)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R								G								B								A							

RGBx32-8888 (0x29)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R								G								B								Unused							

dispc-305

Figure 12-323. DISPC RGB 32-bit Pixel Formats 1

Copyright © 2024 Texas Instruments Incorporated

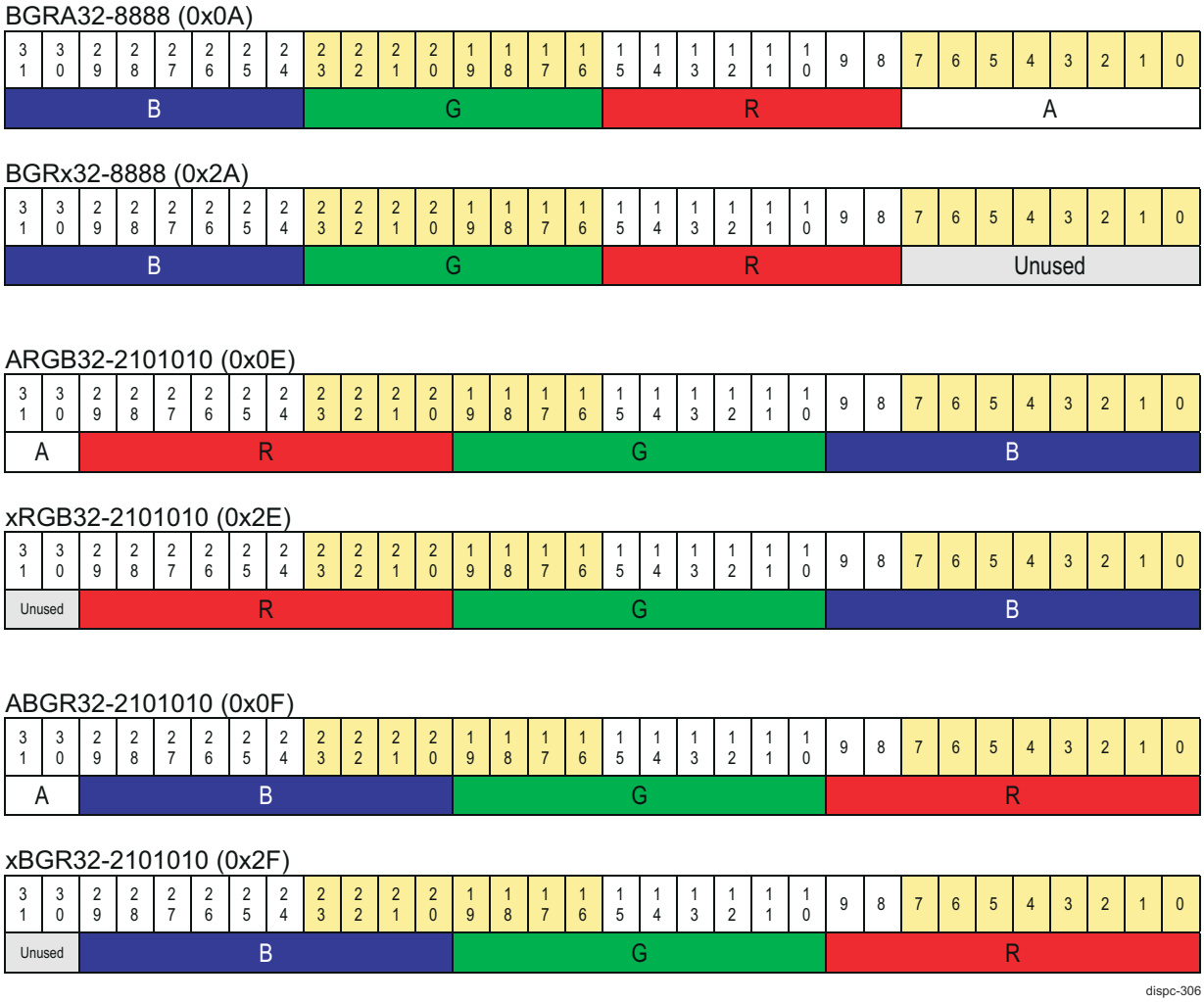


Figure 12-324. DISPC RGB 32-bit Pixel Formats 2

Figure 12-325 shows the pixel data memory organization for the RGB 64-bit pixel formats.

Copyright © 2024 Texas Instruments Incorporated

ARGB64-16161616 (0x10)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
G																B																+0x00
A																R																+0x04

xRGB64-16161616 (0x30)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
G																B																+0x00
Unused																R																+0x04

RGBA64-16161616 (0x11)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
B																A																+0x00
R																G																+0x04

RGBx64-16161616 (0x31)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
B																Unused																+0x00
R																G																+0x04

dispc-307

Figure 12-325. DISPC RGB 64-bit Pixel Formats

Figure 12-326 and Figure 12-327 show the pixel data memory organization for the YUV 8-bit pixel formats, together with some specific register settings.

ATTRIBUTES2.YUV_SIZE = 0 (8b)
ATTRIBUTES2.YUV_MODE = 0 (N/A)
ATTRIBUTES2.YUV_ALIGN = 0 (N/A)

YUV422 (1-plane/Packed)

YUV2 4:2:2 (0x3E)

31	30	29	28	27		26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
	Cr0								Y1								Cb0								Y0								+0x0
	Cr1								Y3								Cb1								Y2								+0x4
	Cr2								Y5								Cb2								Y4								+0x8
	Cr3								Y7								Cb3								Y6								+0xC

UYVY 4:2:2 (0x3F)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Y1								Cr0								Y0								Cb0								+0x0
Y3								Cr1								Y2								Cb1								+0x4
Y5								Cr2								Y4								Cb2								+0x0
Y7								Cr3								Y6								Cb3								+0x4

dispc-308

Figure 12-326. DISPC YUV 8-bit Pixel Formats 1

Copyright © 2024 Texas Instruments Incorporated

YUV420/422 (2-plane/Planar)

YUV 4:2:0 – NV12 (0x3D), YUV 4:2:2 – NV12 (0x3C)

31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16	15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0	
Y3	Y2	Y1	Y0	+0x0
Y7	Y6	Y5	Y4	+0x4
Y11	Y10	Y9	Y8	+0x8
Y15	Y14	Y13	Y12	+0xC

31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16	15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0	
Cr1	Cb1	Cr0	Cb0	+0x0
Cr3	Cb3	Cb2	Cb2	+0x4
Cr5	Cb5	Cr4	Cb4	+0x8
Cr7	Cb7	Cb6	Cb6	+0xC

YUV 4:2:0 – NV21 (0x3D), YUV 4:2:2 – NV21 (0x3C)

31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16	15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0	
Y3	Y2	Y1	Y0	+0x0
Y7	Y6	Y5	Y4	+0x4
Y11	Y10	Y9	Y8	+0x8
Y15	Y14	Y13	Y12	+0xC

31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16	15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0	
Cb1	Cr1	Cb0	Cr0	+0x0
Cb3	Cr3	Cr2	Cr2	+0x4
Cb5	Cr5	Cb4	Cr4	+0x8
Cb7	Cr7	Cr6	Cr6	+0xC

NV21 is not directly supported. It must be specified as NV12 and then the chroma components must be swapped during the YUV to RGB conversion process.

dispc-309

Figure 12-327. DISPC YUV 8-bit Pixel Formats 2

Figure 12-328 shows the pixel data memory organization for the YUV 10-bit packed pixel formats, together with some specific register settings.

Copyright © 2024 Texas Instruments Incorporated

YUV 10-bit formats have the same component packing order as 8-bit formats except that the packing is done across a multiple 32-bit word with 2 MSB in each 32-bit word not used.
Note: Each new line in memory must start at a 128-bit aligned address for this format

ATTRIBUTES2.YUV_SIZE = 1 (10b)
ATTRIBUTES2.YUV_MODE = 0 (Packed)
ATTRIBUTES2.YUV_ALIGN = 0 (N/A)

YUV422 (1-plane/Packed)

YUV2 4:2:2 – 10bit (0x3E)

31 30	29 28 27 26 25 24	23 22 21 20 19 18 17 16	15 14 13 12 11 10	9 8 7 6 5 4 3 2 1 0	
	Y1	Cb0		Y0	+0x0
	Cb1	Y2		Cr0	+0x4
	Y4	Cr1		Y3	+0x8
	Cr2	Y5		Cb2	+0xC

UYVY 4:2:2 – 10bit (0x3F)

31 30	29 28 27 26 25 24	23 22 21 20 19 18 17 16	15 14 13 12 11 10	9 8 7 6 5 4 3 2 1 0	
	Cr0	Y0		Cb0	+0x0
	Y2	Cb1		Y1	+0x4
	Cb2	Y3		Cr1	+0x8
	Y5	Cr2		Y4	+0xC

YUV420/YUV422 (2-plane/Planar)

YUV 4:2:0 – NV12 (10-bit) (0x3D), YUV 4:2:2 – NV12 (10-bit) (0x3C)

31 30	29 28 27 26 25 24	23 22 21 20 19 18 17 16	15 14 13 12 11 10	9 8 7 6 5 4 3 2 1 0	
	Y2	Y1		Y0	+0x0
	Y5	Y4		Y3	+0x4
	Y8	Y7		Y6	+0x0
	Y11	Y10		Y9	+0x4

31 30	29 28 27 26 25 24	23 22 21 20 19 18 17 16	15 14 13 12 11 10	9 8 7 6 5 4 3 2 1 0	
	Cb1	Cr0		Cb0	+0x0
	Cr2	Cb2		Cr1	+0x4
	Cb4	Cr3		Cb3	+0x8
	Cr5	Cb5		Cr4	+0xC

dispc-310

Figure 12-328. DISPC YUV 10-bit Pixel Packed Formats

Figure 12-329 and Figure 12-330 show the pixel data memory organization for the YUV 12-bit packed pixel formats, together with some specific register settings.

Copyright © 2024 Texas Instruments Incorporated

YUV 12-bit formats have the same component packing order as 8-bit formats except that the packing is done across a multiple 64-bit word with 4 MSB in each 64-bit word not used.
Note: Each new line in memory must start at a 128-bit aligned address for this format

ATTRIBUTES2.YUV_SIZE = 2 (12b)
ATTRIBUTES2.YUV_MODE = 0 (Packed)
ATTRIBUTES2.YUV_ALIGN = 0 (N/A)

YUV422 (1-plane)

YUV2 4:2:2 – 12bit (0x3E)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Y1[7:0]								Cb0												Y0												+0x0
				Y2												Cr0												Y1 [11:8]				+0x4
Cr1[7:0]								Y3												Cb1												+0x8
				Cb2												Y4												Cr1[11:8]				+0xC
Y6[7:0]								Cr2												Y5												+0x10
				Y7												Cb3												Y6[11:8]				+0x14
Cb4[7:0]								Y8												Cr3												+018
				Cr4												Y9												Cb4[11:8]				+1C

UYVY 4:2:2 – 12bit (0x3F)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Cr0[7:0]								Y0												Cb0												+0x0
				Cb1												Y1												Cr0 [11:8]				+0x4
Y3[7:0]								Cr1												Y2												+0x8
				Y4												Cb2												Y3[11:8]				+0xC
Cb3[7:0]								Y5												Cr2												+0x10
				Cr3												Y6												Cb3[11:8]				+0x14
Y8[7:0]								Cb4												Y7												+018
				Y9												Cr4												Y8[11:8]				+1C

dispc-311

Figure 12-329. DISPC YUV 12-bit Packed Pixel Formats 1

Copyright © 2024 Texas Instruments Incorporated

YUV420/YUV422 (2-plane)

YUV 4:2:0 – NV12 (12-bit) (0x3D), YUV 4:2:2 – NV12 (12-bit) (0x3C)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Y2[7:0]								Y1												Y0												+0x0
				Y4												Y3												Y2[11:8]				+0x4
Y7[7:0]								Y6												Y5												+0x8
				Y9												Y8												Y7[11:8]				+0xC

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Cb1[7:0]								Cr0												Cb0												+0x0
				Cb2												Cr1												Cb1 [11:8]				+0x4
Cr3[7:0]								Cb3												Cr2												+0x8
				Cr4												Cb4												Cr3[11:8]				+0xC

dispc-312

Figure 12-330. DISPC YUV 12-bit Packed Pixel Formats 2

Figure 12-331 through Figure 12-333 show the pixel data memory organization for the YUV 10-bit/12-bit unpacked pixel formats in 16-bit container, together with some specific register settings.

Copyright © 2024 Texas Instruments Incorporated

YUV 10-bit/12-bit unpacked formats have the same component packing order as 8-bit formats except that each component is stored in a 16-bit container (with MSB or LSB bits within the 16-bit container not used depending on the MSB/LSB alignment).

ATTRIBUTES2.YUV_SIZE = 1 or 2 (10b or 12b)
ATTRIBUTES2.YUV_MODE = 1 (Unpacked)
ATTRIBUTES2.YUV_ALIGN = 0 or 1 (LSB or MSB aligned)

YUV422 (1-plane)

10-bit unpacked LSB aligned

YUV2 4:2:2 – 10bit (0x3E)

31 30 29 28 27 26	25 24 23 22 21 20 19 18 17 16	15 14 13 12 11 10	9 8 7 6 5 4 3 2 1 0	
unused	Cb0	unused	Y0	+0x0
unused	Cr0	unused	Y1	+0x4

UYVY 4:2:2 – 10bit (0x3F)

31 30 29 28 27 26	25 24 23 22 21 20 19 18 17 16	15 14 13 12 11 10	9 8 7 6 5 4 3 2 1 0	
unused	Y0	unused	Cb0	+0x0
unused	Y1	unused	Cr0	+0x4

10-bit unpacked MSB aligned

YUV2 4:2:2 – 10bit (0x3E)

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16	15 14 13 12 11 10 9 8 7 6	5 4 3 2 1 0		
Cb0	unused	Y0	unused	+0x0
Cr0	unused	Y1	unused	+0x4

UYVY 4:2:2 – 10bit (0x3F)

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16	15 14 13 12 11 10 9 8 7 6	5 4 3 2 1 0		
Y0	unused	Cb0	unused	+0x0
Y1	unused	Cr0	unused	+0x4

dispc-313

Figure 12-331. DISPC YUV 10-bit Unpacked Pixel Formats 1

Copyright © 2024 Texas Instruments Incorporated

YUV422 (1-plane)

12-bit unpacked LSB aligned

YUV2 4:2:2 – 12bit (0x3E)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
unused				Cb0												unused				Y0												+0x0
unused				Cr0												unused				Y1												+0x4

UYVY 4:2:2 – 12bit (0x3F)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
unused				Y0												unused				Cb0												+0x0
unused				Y1												unused				Cr0												+0x4

12-bit unpacked MSB aligned

YUV2 4:2:2 – 12bit (0x3E)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Cb0												unused				Y0												unused				+0x0
Cr0												unused				Y1												unused				

UYVY 4:2:2 – 12bit (0x3F)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Y0												unused				Cb0												unused				0x0
Y1												unused				Cr0												unused				

dispc-314

Figure 12-332. DISPC YUV 12-bit Unpacked Pixel Formats 2

Copyright © 2024 Texas Instruments Incorporated

YUV420/YUV422 (2-plane)

10-bit unpacked LSB aligned

YUV 4:2:0 – NV12 (10-bit) (0x3D), YUV 4:2:2 – NV12 (10-bit) (0x3C)

31 30 29 28 27 26	25 24 23 22 21 20	19 18 17 16	15 14 13 12 11 10	9 8 7 6 5 4 3 2 1 0	
unused	Y1		unused	Y0	+0x0
unused	Y3		unused	Y2	+0x4

31 30 29 28 27 26	25 24 23 22 21 20	19 18 17 16	15 14 13 12 11 10	9 8 7 6 5 4 3 2 1 0	
unused	Cr0		unused	Cb0	+0x0
unused	Cr1		unused	Cb1	+0x4

10-bit unpacked MSB aligned

YUV 4:2:0 – NV12 (10-bit) (0x3D), YUV 4:2:2 – NV12 (10-bit) (0x3C)

31 30 29 28 27 26	25 24 23 22	21 20 19 18	17 16	15 14 13 12 11 10	9 8 7 6 5 4 3 2 1 0	
Y1		unused		Y0	unused	+0x0
Y3		unused		Y2	unused	+0x4

31 30 29 28 27 26	25 24 23 22	21 20 19 18	17 16	15 14 13 12 11 10	9 8 7 6 5 4 3 2 1 0	
Cr0		unused		Cb0	unused	+0x0
Cr1		unused		Cb1	unused	+0x4

12-bit unpacked LSB aligned

YUV 4:2:0 – NV12 (12-bit) (0x3D),YUV 4:2:2 – NV12 (12-bit) (0x3C)

31 30 29 28	27 26 25 24	23 22 21 20	19 18 17 16	15 14 13 12	11 10 9 8 7 6 5 4 3 2 1 0	
unused	Y1			unused	Y0	+0x0
unused	Y3			unused	Y2	+0x4

31 30 29 28	27 26 25 24	23 22 21 20	19 18 17 16	15 14 13 12	11 10 9 8 7 6 5 4 3 2 1 0	
unused	Cr0			unused	Cb0	+0x0
unused	Cr1			unused	Cb1	+0x4

12-bit unpacked MSB aligned

YUV 4:2:0 – NV12 (12-bit) (0x3D), YUV 4:2:2 – NV12 (12-bit) (0x3C)

31 30 29 28 27 26 25 24	23 22 21 20	19 18 17 16	15 14 13 12 11 10	9 8 7 6 5 4	3 2 1 0	
Y1		unused	Y0		unused	+0x0
Y3		unused	Y2		unused	+0x4

31 30 29 28 27 26 25 24	23 22 21 20	19 18 17 16	15 14 13 12 11 10	9 8 7 6 5 4	3 2 1 0	
Cr0		unused	Cb0		unused	+0x0
Cr1		unused	Cb1		unused	+0x4

dispc-315

Figure 12-333. DISPC YUV 10-bit/12-bit Unpacked Pixel Formats 3

Copyright © 2024 Texas Instruments Incorporated

12.6.3.8 DISPC Video Pipeline

The DISPC includes two types of input video pipelines:

- Video pipeline (VID1 and VID2)
- Video lite pipeline (VIDL1 and VIDL2)

The video pipeline (VID) consists of:

- VC-1 range mapping unit for YUV422/YUV420 input formats;
- Replication logic for ARGB input formats;
- One 256 x 24-bit entries Color Look-up Table (CLUT);
- Polyphase-filter based resizer unit (scaler) supporting chroma upsampling;
- Luma-key support;
- Color Space Conversion (CSC) unit (YUV to RGB), which can be used also for programmable BCHS (Brightness/Contrast/Hue/Saturation) control;

The video lite pipeline (VIDL) is identical to the video pipeline (VID), except for:

- Polyphase-filter based resizer (scaler) is not included;
- Chroma upsampling is done using dedicated YUV420-to-422 and YUV422-to-444 upsamplers (instead of using the scaler);

Each pipeline processing block can be independently bypassed.

Note

The DISPC input video pipelines, VID and VID1L, will be commonly refered to as *video pipeline (VID)* in the following sections. Any differences in their functionality will be highlighted.

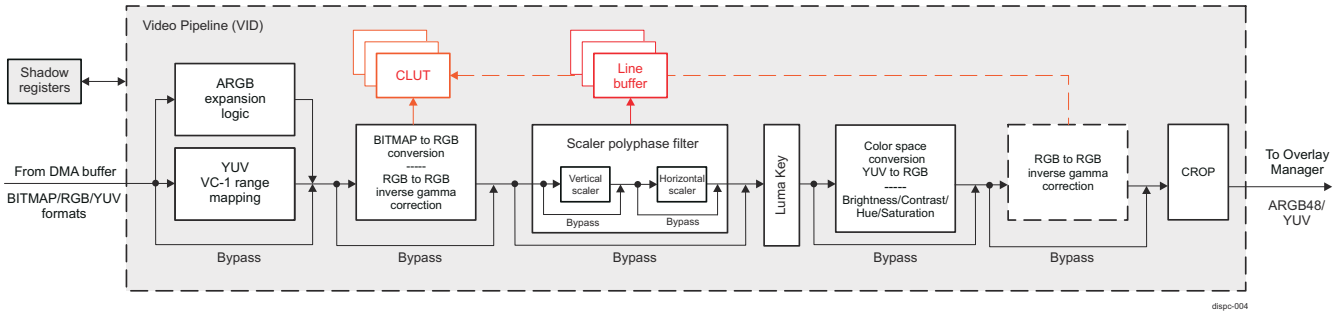


Figure 12-334. DISPC Video Pipeline Configuration

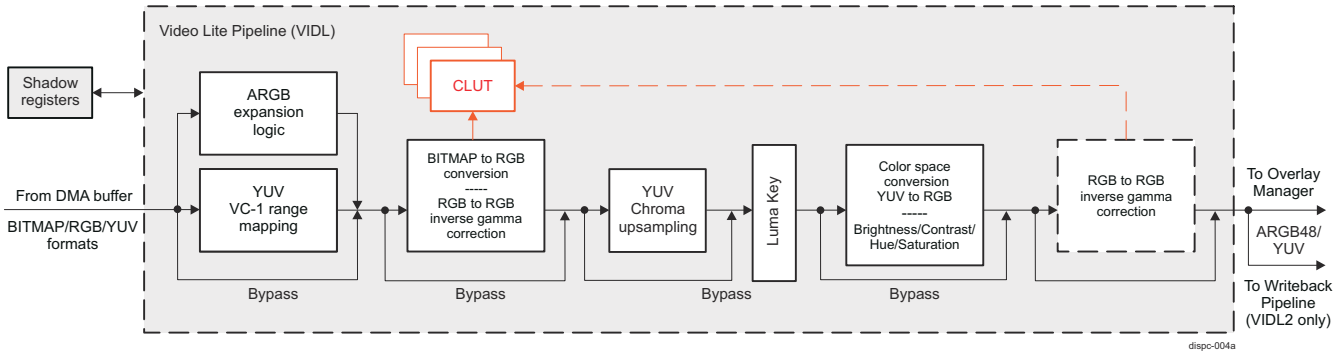


Figure 12-335. DISPC Video Lite Pipeline Configuration

The input of the VID pipeline is connected to the video DMA buffer controller. The pixel output of the VID pipeline is connected to the overlay managers. The VID pipeline configuration supports various BITMAP, RGB (ARGB and RGBA), and YUV formats, as listed in Table 12-338, *DISPC Pixel Data Formats*.

The 256-entry CLUT is either used to convert BITMAP (1, 2, 4, or 8-bit indexed formats) into RGB format, or for RGB to RGB inverse gamma correction. For a BITMAP format data, scaling is not supported. Scaling and

Copyright © 2024 Texas Instruments Incorporated

color look-up table features are mutually exclusive. If the color look-up feature is enabled, then the video pipeline scaler has to be disabled.

For chroma sub-sampled YUV formats (YUV422 and YUV420-NV12/NV21):

- VID pipeline: the scaler unit upsamples the chrominance data using both vertical and horizontal polyphase filters;
- VIDL pipeline: dedicated YUV420 to YUV422 and YUV422 to YUV444 chroma upsamplers are used;

For ARGB source data with less than/equal to 10-bit component data size the replication logic (ARGB expansion) converts the data to ARGB48 by replicating the MSBs into the LSBs:

- When scaling is disabled (or no scaler supported), the resulting ARGB48 data is directly provided to the pipeline output;
- When vertical scaling is engaged, the resulting ARGB48 data is first truncated to ARGB8888, and then converted to ARGB10101010 (by MSBs replication into LSBs), before being fed to the vertical scaler input;
- When vertical scaling is disabled, but horizontal scaling is engaged, the resulting ARGB48 data is directly provided to the horizontal scaler input;

Note

There are limitations on using the inverse gamma (InvGamma) operation along with scaling. If the scaler is enabled, the InvGamma operation must always take place after the scaler in the data-path (as shown in Figure 12-334). This position of the InvGamma operation is controllable through the DSS0_VID_ATTRIBUTES2[29] GAMMAINVERSIONPOS register field.

All the memories (line buffers attached to the scaler, LUT and chroma upsamplers) are sized to support 10-bit per color component. This allows full 10-bit support inside the video pipeline.

The VID pipeline can be enabled by setting the DSS0_VID_ATTRIBUTES[0] ENABLE register bit. If the video pipeline is disabled, the video window does not exist on the screen and the whole video pipeline and its DMA are inactive. Prior to enabling the video layer a valid configuration has to be set by the user.

12.6.3.8.1 DISPC VID Replication Logic

The replication logic (ARGB expansion) converts ARGB pixel formats into ARGB48 format by replicating the MSBs into the LSBs. The logic is always enabled.

Table 12-339 shows how some of the ARGB formats supported by VID are remapped into ARGB48 by default.

Table 12-339. DISPC VID Replication: ARGB Pixel Formats Remapping into ARGB48-12121212

Formats	A[11:0]	R[11:0]	G[11:0]	B[11:0]
	MSB - LSB	MSB - LSB	MSB - LSB	MSB - LSB
xRGB12-4444	111111111111	R[3:0]R[3:0]R[3:0]	G[3:0]G[3:0]G[3:0]	B[3:0]B[3:0]B[3:0]
RGBx12-4444	111111111111	R[3:0]R[3:0]R[3:0]	G[3:0]G[3:0]G[3:0]	B[3:0]B[3:0]B[3:0]
RGB16-565	111111111111	R[4:0]R[4:0]R[4:3]	G[5:0]G[5:0]	B[4:0]B[4:0]B[4:3]
xRGB16-1555	111111111111	R[4:0]R[4:0]R[4:3]	G[4:0]G[4:0]G[4:3]	B[4:0]B[4:0]B[4:3]
ARGB16-1555	AAAAAAAAAA	R[4:0]R[4:0]R[4:3]	G[4:0]G[4:0]G[4:3]	B[4:0]B[4:0]B[4:3]
ARGB16-4444	A[3:0]A[3:0]A[3:0]	R[3:0]R[3:0]R[3:0]	G[3:0]G[3:0]G[3:0]	B[3:0]B[3:0]B[3:0]
RGBA16-4444	A[3:0]A[3:0]A[3:0]	R[3:0]R[3:0]R[3:0]	G[3:0]G[3:0]G[3:0]	B[3:0]B[3:0]B[3:0]
ARGB32-8888	A[7:0]A[7:4]	R[7:0]R[7:4]	G[7:0]G[7:4]	B[7:0]B[7:4]

12.6.3.8.2 DISPC VID VC-1 Range Mapping Unit

The VC-1 range mapping unit is used when the video frame picture is decoded using a VC-1 codec by the video accelerator. It remaps the Y, Cb, and Cr components to the full range (from the reduced data range) before displaying. The unit is used primarily for YUV420-NV12 (NV21) pixel format, but also can be applied to YUV422 pixel formats (YUV2 and UYVY).

The VC-1 range mapping unit is enabled by setting the DSS0_VID_ATTRIBUTES2[0] VC1ENABLE bit to 0x1. The VC-1 range mapping must be enabled only for 8 bits/component YUV input data.

Copyright © 2024 Texas Instruments Incorporated

The DSS0_VID_ATTRIBUTES2[3:1] VC1_RANGE_Y and [6:4]VC1_RANGE_CBCR register bitfields are two 3-bit values programmed by the user.

The equations for the mapping process are:

Y_{out} = CLIP((((Y_{int} – 128) × (VC1_RANGE_Y + 9) + 4) / 8) + 128)

C_b = CLIP((((C_b – 128) × (VC1_RANGE_CBCR + 9) + 4) / 8) + 128)

C_r = CLIP((((C_r – 128) × (VC1_RANGE_CBCR + 9) + 4) / 8) + 128)

Note

The input and output pixel values are unsigned (Y, Cr, and Cb).

The function CLIP() clips to 0 or 255 when minimum or maximum, respectively, is reached. Otherwise, the resulting output remains identical.

12.6.3.8.3 DISPC VID Color Look-Up Table (CLUT)

The video pipeline supports a look up table to perform either of the following operations:

- Conversions of BITMAP formats (1, 2, 4, or 8-bit indexed) into RGB format (CLUT mode), or
- Inverse gamma correction on non-linear RGB source data (gamma mode)

The look-up table consists of 3 separate 1024 x 10-bit memories and is indexed either by the source BITMAP data or by R/G/B component data. The table is loaded through direct register access by writing to the CLUT registers.

The sequence to load the table is:

1. SW writes (only writes are supported) 32-bit values for both color and monochrome mode using single access, or burst access in linear increment burst mode, into the DSS0_VID_CLUT_0 through DSS0_VID_CLUT_15 registers. The LSB 30 bits [29:0] are used for the value and the MSB 1 bit [31] is used to reset the index of the table for the color mode, while the LSB 8 bits are used for the value and the MSB 1 bit to reset the index of the table for monochrome mode. Figure 12-336 describes the format of one of the palette value in the memory.
2. Loop to Step 1, if there is a new access to the CLUT registers. Software can access other registers between two accesses to the CLUT registers.

SW needs to ensure that there is no visible effect when modifying the table, since it is not under hardware control.

The usage of the look-up table in CLUT mode is activated when a BITMAP format is selected in the DSS0_VID_ATTRIBUTES[6:1] FORMAT register bit-field.

The usage of the look-up table for RGB inverse gamma correction can be enabled by setting DSS0_VID_ATTRIBUTES[30] GAMMAINVERSION register bit.

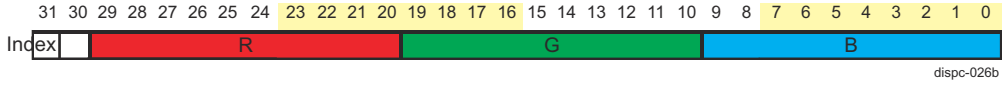


Figure 12-336. DISPC VID CLUT/Gamma Data Memory Organization

Note

Scaling and color look-up table features are mutually exclusive. If color look-up feature is enabled, then video pipeline scaler has to be disabled.

Copyright © 2024 Texas Instruments Incorporated

12.6.3.8.4 DISPC VID Chrominance Resampling

12.6.3.8.4.1 Chrominance Resampling for VID Pipeline

The chrominance resampling from 8-bpc or 10-bpc to 12-bit color components is always performed using filtering (the scaler unit filter). Chrominance resampling and rescaling can be combined to support native rescaling of YUV formats.

The usage of the scaler unit for resampling the chrominance of YUV420 and YUV422 is shown in Figure 12-337 and Figure 12-338, respectively. The settings of the scaler unit to perform chrominance resampling are described in Section 12.6.3.8.5, *DISPC VID Scaler Unit*.

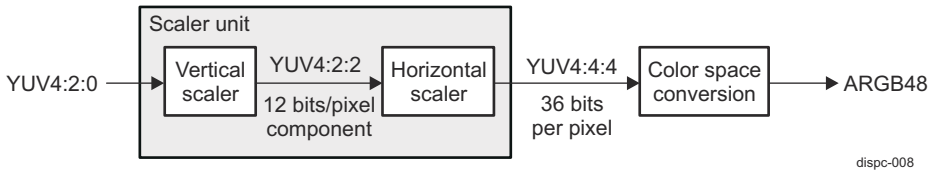


Figure 12-337. DISPC VID YUV420 to ARGB48 Using Scaler Unit for Resampling Chrominance

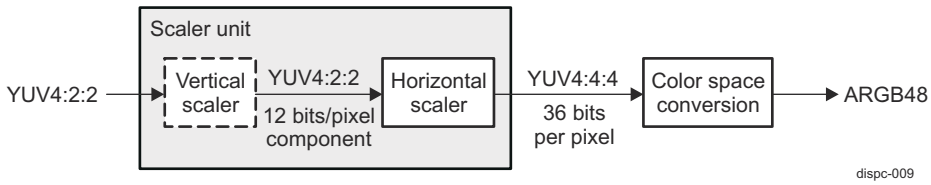


Figure 12-338. DISPC VID YUV422 to ARGB48 Using Scaler Unit for Resampling Chrominance

The vertical scaler is not a mandatory block for resampling the chrominance of YUV422 data, as shown in Figure 12-338.

Using the filter it is possible to re-sample the chrominance from either MPEG1 or MPEG2 sub-sampled chroma source.

12.6.3.8.4.2 Chrominance Resampling for VIDL Pipeline

The VIDL pipeline does not include a scaler unit. Chroma upsampling is done using dedicated YUV420 to YUV422, and YUV422 to YUV444 chroma upsamplers.

The VIDL pipeline converts YUV420 (Chroma) data to YUV422 (Chroma) using a simple vertical average filter (average of two adjacent chroma lines to generate a missing line except for the very bottom line which is generated by repeating the previous chroma line). If the video image is a sub-image of a larger video frame data, then the vertical Luma line offset from the top should be an even number.

The VIDL pipeline converts YUV422 data to YUV444 format using a 4-tap filter (implementing the Catmull-Rom algorithm) to generate missing chroma data as follows:

Cout[2*i] = Cin[i]

Cout[2*i+1] = CLIP (-1/16*Cin[i-1] + 9/16*Cin[i] + 9/16*Cin[i+1] – 1/16*Cin[i+2])

The missing chroma data is generated as a simple weighted average of the surrounding 4 chroma values, which is equivalant to a 4x1 filter kernel with values: [-1/16, 9/16, 9/16, -1/16]. In this algorithm, edge effects are treated by repeating the first and last pixel.

12.6.3.8.5 DISPC VID Scaler Unit

Note

Only VID pipeline includes a resizer unit (scaler). The VIDL pipeline does not include a scaler.

The programmable scaler filter works with all supported video formats, including formats with alpha channel. The alpha channel is scaled with the same parameters as the RGB color components. For the YUV formats,

Copyright © 2024 Texas Instruments Incorporated

Y and Cb/Cr are processed independently. A RGB 64-bit source (16 bits per component) is truncated to 8-bit/compontent, if the scaler is enabled. Otherwise, it will be truncated to ARGB48 format in the scaler bypass mode. A 10-bit/12-bit YUV source is also truncated to 8-bit, since the scaler is enabled to either upsample the chroma component and/or resize the YUV frame data directly (for memory-to-memory operation with YUV format as the memory destination type).

The filter is based on a finite impulse response (FIR) filter with 16 phases. The filter is a 5-tap for horizontal filtering, and can be configured for 3 or 5 taps for vertical filtering. The filtering can be used for various processing:

- Up-sampling of the picture
- Down-sampling of the picture
- Anti-aliasing reduction
- Chrominance resampling in case of YUV data formats

The following limitations must be considered:

- The up-sampling ratio is up to x16.
- The down-sampling ratio using 3-tap configuration is down to x0.5 for RGB format.
- The down-sampling ratio using 5-tap configuration is down to x0.25 for RGB format.

Note

The user must set the correct size and position of the original video before resize in order for the up-sampled/down-sampled video to be displayed inside the screen boundaries.

Figure 12-339 shows an example of video up-sampling, with corresponding video window attributes.

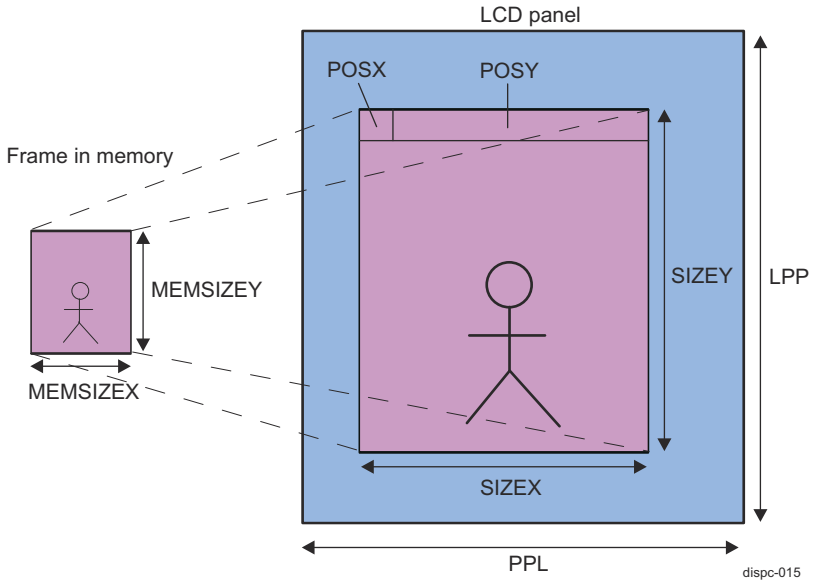


Figure 12-339. DISPC Video Window Attributes

The video window attributes can be configured in the following register bit-fields:

- Source data format (input to the scaler unit) in DSS0_VID_ATTRIBUTES[6-1] FORMAT bit-field
- Video picture width in system memory (frame buffer) in DSS0_VID_PICTURE_SIZE[13-0] MEMSIZEX bit-field. The window width is from 1 up to 4096 pixels. All the integer values in the range [1:4096] are allowed except for YUV422 formats. In case of YUV2422 and UYVY422 formats the width has to be a multiple of 2 pixels.
- Video picture height in system memory (frame buffer) in DSS0_VID_PICTURE_SIZE[29-16] MEMSIZEY bit-field. The window width is from 1 up to 4096 pixels. All the integer values in the range [1:4096] are allowed.

Copyright © 2024 Texas Instruments Incorporated

- Video window width at pipeline output (after resizing) in DSS0_VID_SIZE[13-0] SIZEX bit-field. The window width is from 1 up to 4096 pixels. All the integer values in the range [1:4096] are allowed up to the screen width minus the horizontal start location of the window on the screen.
- Video window height at pipeline output (after resizing) in DSS0_VID_SIZE[29-16] SIZEY bit-field. The window height is from 1 up to 4096 pixels. All the integer values in the range [1:4096] are allowed up to the screen height minus the vertical start location of the window on the screen.
- Video window overlay X-position in DSS0_OVR_ATTRIBUTES2_0 through DSS0_OVR_ATTRIBUTES2_4[13-0] POSX bit-field. See Section 12.6.3.10, *DISPC Overlay Managers*.
- Video window overlay Y-position in DSS0_OVR_ATTRIBUTES2 through DSS0_OVR_ATTRIBUTES2_4[29-16] POSY bit-field. See Section 12.6.3.10, *DISPC Overlay Managers*.
- For configuration of LPP and PPL video port output display parameters, see Section 12.6.3.11.8, *DISPC VP Timing Generator and Display Panel Settings*.

For vertical up-sampling and down-sampling in a 3-tap configuration, the equations are:

For RGB formats

$$Aout(n) = \left(\sum_{i=-1}^{i=1} Ci(\Phi) \times Ain(n+i)\right) >> (Cwidth - (OutWidth - InWidth) - 1)$$

$$Rout(n) = \left(\sum_{i=-1}^{i=1} Ci(\Phi) \times Rin(n+i)\right) >> (Cwidth - (OutWidth - InWidth) - 1)$$

$$Gout(n) = \left(\sum_{i=-1}^{i=1} Ci(\Phi) \times Gin(n+i)\right) >> (Cwidth - (OutWidth - InWidth) - 1)$$

$$Bout(n) = \left(\sum_{i=-1}^{i=1} Ci(\Phi) \times Bin(n+i)\right) >> (Cwidth - (OutWidth - InWidth) - 1)$$

For YUV formats

$$Yout(n) = \left(\sum_{i=-1}^{i=1} Cyi(\Phi y) \times Yin(n+i)\right) >> (Cwidth - (OutWidth - InWidth) - 1)$$

$$Cbout(n) = \left(\sum_{i=-1}^{i=1} Cci(\Phi c) \times Cbin(n+i)\right) >> (Cwidth - (OutWidth - InWidth) - 1)$$

$$Crout(n) = \left(\sum_{i=-1}^{i=1} Cci(\Phi c) \times Crin(n+i)\right) >> (Cwidth - (OutWidth - InWidth) - 1)$$

Component	Vertical	Horizontal
Cwidth (Coefficient Width)	10 bits	10 bits
InWidth (Input Width)	10 bits	12 bits
OutWidth (Output Width)	12 bits	12 bits

dispc-013 (23)

For vertical and horizontal up-sampling and down-sampling in a 5-tap configuration, the equations are:

For RGB formats

$$Aout(n) = \left(\sum_{i=-2}^{i=2} Ci(\Phi) \times Ain(n+i)\right) >> (Cwidth - (OutWidth - InWidth) - 1)$$

$$Rout(n) = \left(\sum_{i=-2}^{i=2} Ci(\Phi) \times Rin(n+i)\right) >> (Cwidth - (OutWidth - InWidth) - 1)$$

$$Gout(n) = \left(\sum_{i=-2}^{i=2} Ci(\Phi) \times Gin(n+i)\right) >> (Cwidth - (OutWidth - InWidth) - 1)$$

$$Bout(n) = \left(\sum_{i=-2}^{i=2} Ci(\Phi) \times Bin(n+i)\right) >> (Cwidth - (OutWidth - InWidth) - 1)$$

For YUV formats

$$Yout(n) = \left(\sum_{i=-2}^{i=2} Cyi(\Phi y) \times Yin(n+i)\right) >> (Cwidth - (OutWidth - InWidth) - 1)$$

$$Cbout(n) = \left(\sum_{i=-2}^{i=2} Cci(\Phi c) \times Cbin(n+i)\right) >> (Cwidth - (OutWidth - InWidth) - 1)$$

$$Crout(n) = \left(\sum_{i=-2}^{i=2} Cci(\Phi c) \times Crin(n+i)\right) >> (Cwidth - (OutWidth - InWidth) - 1)$$

Component	Vertical	Horizontal
Cwidth (Coefficient Width)	10 bits	10 bits
InWidth (Input Width)	10 bits	12 bits
OutWidth (Output Width)	12 bits	12 bits

dispc-012 (24)

Note

The pixel (n + 1) is the previous pixel with respect to pixel (n). The line (n + 1) is the previous line with respect to line (n).

The coefficients Ci() depend on the phase between input and output pixels.

The coefficients are different for Y and Cr/Cb filtering because the calculations are independent due to the chrominance resampling for YUV422 and YUV420.

Copyright © 2024 Texas Instruments Incorporated

First, the vertical filter is applied to the encoded input pixel data, and then the horizontal filter is applied on the resulting pixel values to generate the output pixel values. The vertical input of the filter consists of:

- Six lines of 2048 × 32 bits for 5-tap configuration
- Three lines of 4096 x 32 bits for 3-tap configuration

Table 12-340 lists some of the scaler supported configurations.

Table 12-340. DISPC VID Line Buffer Width for Scaler Unit

Pixel Format	Maximum Input Width (Pixels) for 5-tap	Maximum Input Width (Pixels) for 3-tap
32 bits per pixel (ARGB32-8888, ARGB-2101010, etc.)	2048 pixels wide (1)	4096 pixels wide
YUV420, YUV422	4096 pixels wide	4096 pixels wide

(1) For the 5-tap configuration, the 6th video line is used on the output of the horizontal scaler in order to start the next output line generation while the video pipeline output is being stalled by the overlay manager (either due to display in blanking period and/or in background pixel period). The 6th line buffer is actually 48-bit wide to allow storage of ARGB48 format pixel data.

At the beginning of frame scaling processing, the first line may be duplicated multiple times depending on the initial vertical phase programmed for the poly-phase filter.

At the end of frame scaling processing the last line is duplicated, if the scaling logic requires loading more lines and the last line has been reached.

Similarly, the first pixel may be duplicated multiple times depending on the initial horizontal phase programmed for the poly-phase filter. The last pixel is duplicated, if the scaling logic requires loading more pixels and the last pixel has been reached.

The programmable coefficients of the polyphase filters are signed 10-bit values (except for the central coefficient, which is unsigned). The vertical video scaler has an 10-bit input and a 12-bit output. The horizontal scaling stage takes the resulting 12-bit input and produces 12-bit output.

Figure 12-340 and Figure 12-341 show the scaler macro-architecture for components A, R, G, and B. Figure 12-342 and Figure 12-343 show the scaler macro-architecture for components Y, Cr, and Cb.

Note

The scaling and CSC clipping is set by the same bit, DSS0_VID_ATTRIBUTES[11] FULLRANGE.

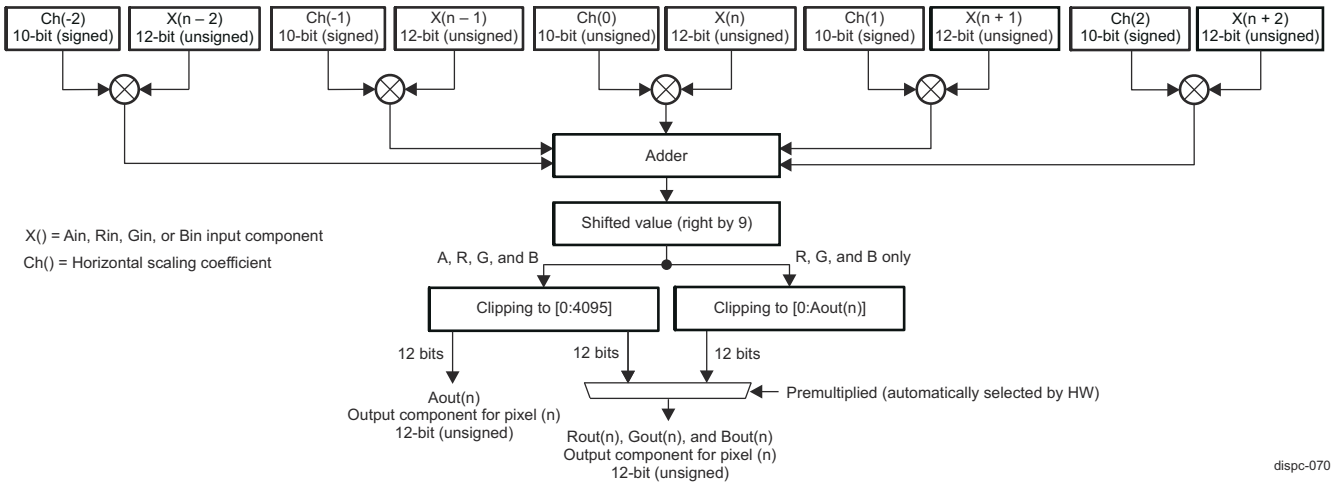


Figure 12-340. DISPC VID Macro-Architecture of the Horizontal Scaling for A, R, G, and B Components (5-tap Restriction)

Copyright © 2024 Texas Instruments Incorporated

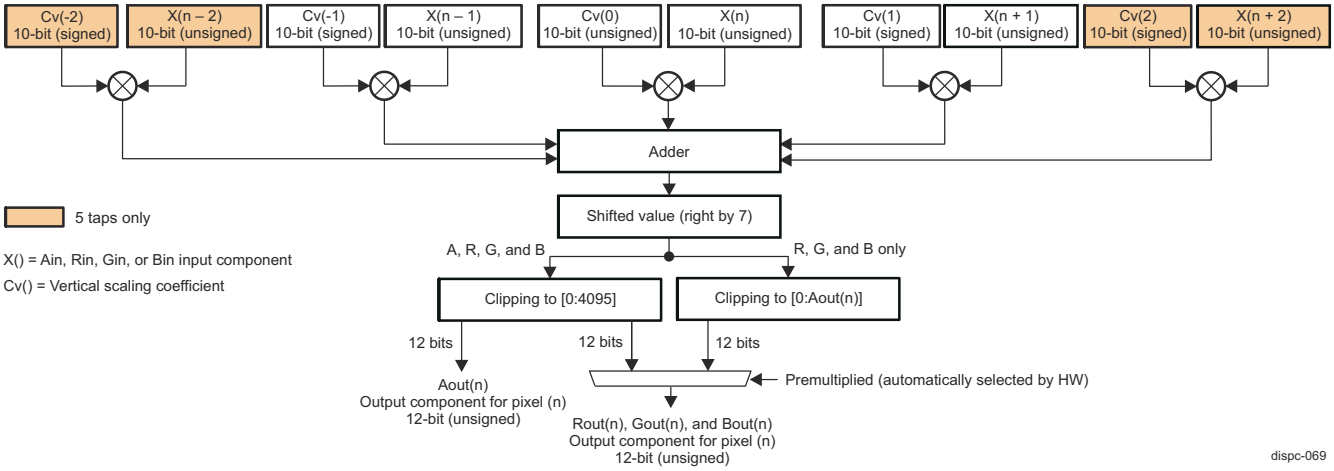


Figure 12-341. DISPC VID Macro-Architecture of the Vertical Scaling for A, R, G, and B Components (5 and 3 taps)

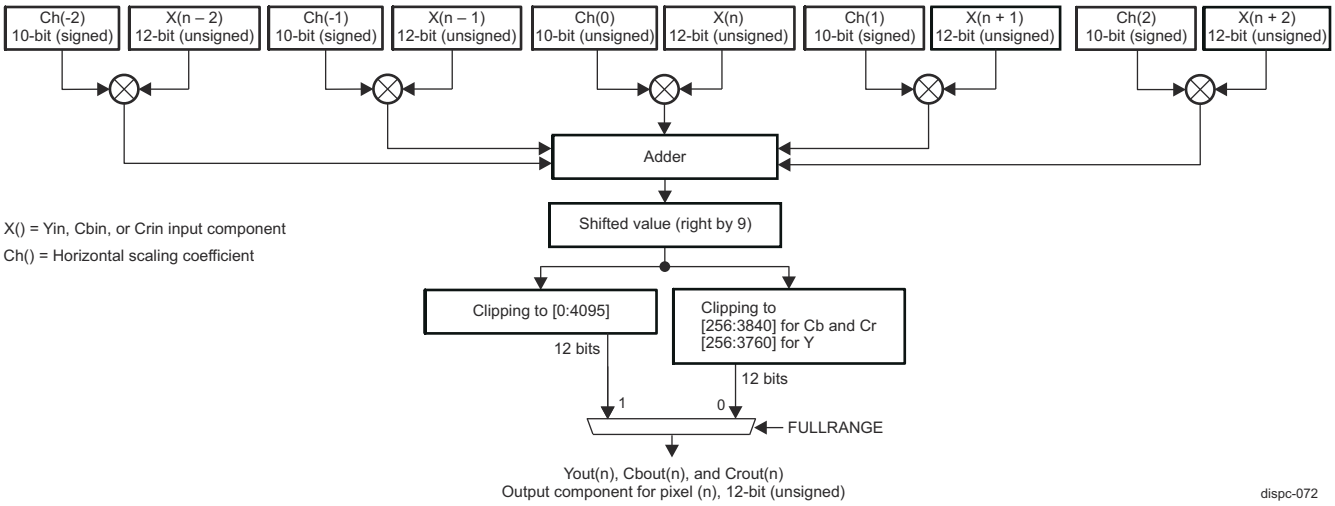


Figure 12-342. DISPC VID Macro-Architecture of the Horizontal Scaling for Y, Cr, and Cb Components (5-tap Restriction)

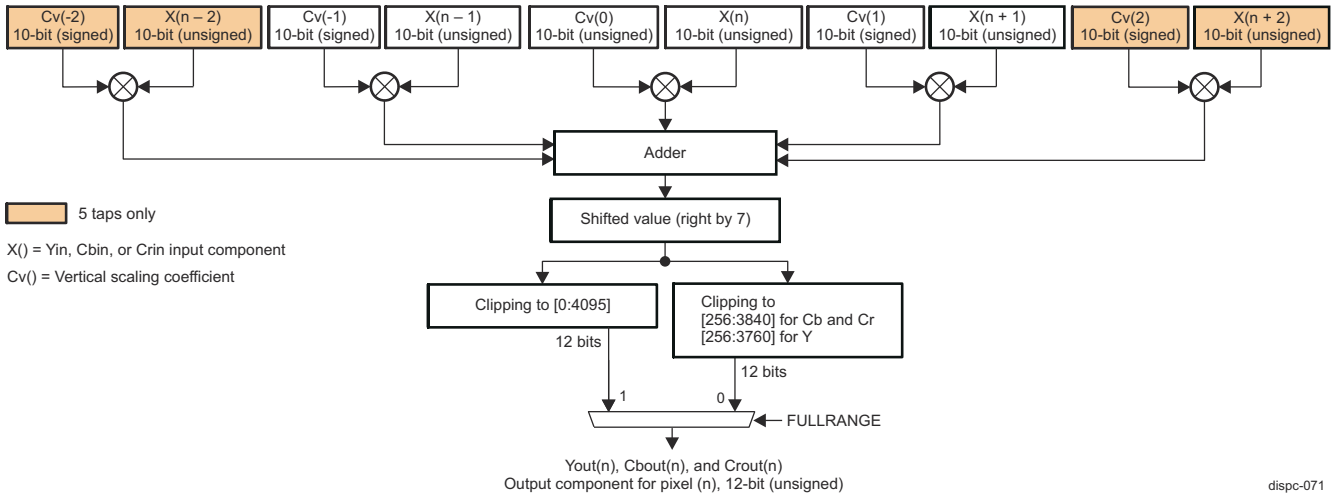


Figure 12-343. DISPC VID Macro-Architecture of the Vertical Scaling for Y, Cr, and Cb Components (5 and 3 taps)

Copyright © 2024 Texas Instruments Incorporated

Table 12-341 list the register fields in the function of the coefficients for the VID horizontal scaler in the DSS0_VID_FIR_COEF_H0_0 to DSS0_VID_FIR_COEF_H0_8, and DSS0_VID_FIR_COEF_H12_0 to DSS0_VID_FIR_COEF_H12_15 registers.

Table 12-341. DISPC Register Fields Associated to Coefficients for ARGB and Y Configuration in VID Horizontal Scaler

Phases	Ch(2)	Ch(1)	Ch(0)	Ch(-1)	Ch(-2)
	Signed coefficient [29-20] FIRHC2 bitfield	Signed coefficient [19-10] FIRHC1 bitfield	Unsigned central coefficient [9-0] FIRHC0 bitfield	Signed coefifcient [19-10] FIRHC1 bitfield	Signed coefficient [29-20] FIRHC2 bitfield
0	DSS0_VID_FIR_COEF_H12_0	DSS0_VID_FIR_COEF_H12_0	DSS0_VID_FIR_COEF_H0_0	DSS0_VID_FIR_COEF_H12_0	DSS0_VID_FIR_COEF_H12_0
1	DSS0_VID_FIR_COEF_H12_1	DSS0_VID_FIR_COEF_H12_1	DSS0_VID_FIR_COEF_H0_1	DSS0_VID_FIR_COEF_H12_15	DSS0_VID_FIR_COEF_H12_15
2	DSS0_VID_FIR_COEF_H12_2	DSS0_VID_FIR_COEF_H12_2	DSS0_VID_FIR_COEF_H0_2	DSS0_VID_FIR_COEF_H12_14	DSS0_VID_FIR_COEF_H12_14
3	DSS0_VID_FIR_COEF_H12_3	DSS0_VID_FIR_COEF_H12_3	DSS0_VID_FIR_COEF_H0_3	DSS0_VID_FIR_COEF_H12_13	DSS0_VID_FIR_COEF_H12_13
4	DSS0_VID_FIR_COEF_H12_4	DSS0_VID_FIR_COEF_H12_4	DSS0_VID_FIR_COEF_H0_4	DSS0_VID_FIR_COEF_H12_12	DSS0_VID_FIR_COEF_H12_12
5	DSS0_VID_FIR_COEF_H12_5	DSS0_VID_FIR_COEF_H12_5	DSS0_VID_FIR_COEF_H0_5	DSS0_VID_FIR_COEF_H12_11	DSS0_VID_FIR_COEF_H12_11
6	DSS0_VID_FIR_COEF_H12_6	DSS0_VID_FIR_COEF_H12_6	DSS0_VID_FIR_COEF_H0_6	DSS0_VID_FIR_COEF_H12_10	DSS0_VID_FIR_COEF_H12_10
7	DSS0_VID_FIR_COEF_H12_7	DSS0_VID_FIR_COEF_H12_7	DSS0_VID_FIR_COEF_H0_7	DSS0_VID_FIR_COEF_H12_9	DSS0_VID_FIR_COEF_H12_9
8	DSS0_VID_FIR_COEF_H12_8	DSS0_VID_FIR_COEF_H12_8	DSS0_VID_FIR_COEF_H0_8	DSS0_VID_FIR_COEF_H12_8	DSS0_VID_FIR_COEF_H12_8
9	DSS0_VID_FIR_COEF_H12_9	DSS0_VID_FIR_COEF_H12_9	DSS0_VID_FIR_COEF_H0_7	DSS0_VID_FIR_COEF_H12_7	DSS0_VID_FIR_COEF_H12_7
10	DSS0_VID_FIR_COEF_H12_10	DSS0_VID_FIR_COEF_H12_10	DSS0_VID_FIR_COEF_H0_6	DSS0_VID_FIR_COEF_H12_6	DSS0_VID_FIR_COEF_H12_6
11	DSS0_VID_FIR_COEF_H12_11	DSS0_VID_FIR_COEF_H12_11	DSS0_VID_FIR_COEF_H0_5	DSS0_VID_FIR_COEF_H12_5	DSS0_VID_FIR_COEF_H12_5
12	DSS0_VID_FIR_COEF_H12_12	DSS0_VID_FIR_COEF_H12_12	DSS0_VID_FIR_COEF_H0_4	DSS0_VID_FIR_COEF_H12_4	DSS0_VID_FIR_COEF_H12_4
13	DSS0_VID_FIR_COEF_H12_13	DSS0_VID_FIR_COEF_H12_13	DSS0_VID_FIR_COEF_H0_3	DSS0_VID_FIR_COEF_H12_3	DSS0_VID_FIR_COEF_H12_3
14	DSS0_VID_FIR_COEF_H12_14	DSS0_VID_FIR_COEF_H12_14	DSS0_VID_FIR_COEF_H0_2	DSS0_VID_FIR_COEF_H12_2	DSS0_VID_FIR_COEF_H12_2
15	DSS0_VID_FIR_COEF_H12_15	DSS0_VID_FIR_COEF_H12_15	DSS0_VID_FIR_COEF_H0_1	DSS0_VID_FIR_COEF_H12_1	DSS0_VID_FIR_COEF_H12_1

Note

In Table 12-341, the cells without color are duplicated from the grey cells.

Similar table approach applies to the vertical scaler (registers DSS0_VID_FIR_COEF_V0_0 to DSS0_VID_FIR_COEF_V0_8, and DSS0_VID_FIR_COEF_V12_0 to DSS0_VID_FIR_COEF_V12_15 are used).

Similar table approach applies to the coefficients for Cb/Cr filtering in case of YUV format (registers DSS0_VID_FIR_COEF_H0_C_0 to DSS0_VID_FIR_COEF_H0_C_8, and DSS0_VID_FIR_COEF_H12_C_0 to DSS0_VID_FIR_COEF_H12_C_15, and DSS0_VID_FIR_COEF_V0_C_0 to DSS0_VID_FIR_COEF_V0_C_8 and DSS0_VID_FIR_COEF_V12_C_0 to DSS0_VID_FIR_COEF_V12_C_15 are used) .

Copyright © 2024 Texas Instruments Incorporated

The VID scaler unit vertical and/or horizontal sampling is selected by configuring the DSS0_VID_ATTRIBUTES[8-7] RESIZEENABLE bit field.

Prior to enabling the video up/down-sampling block a valid configuration has to be set by the user. After configuring the required VID registers change the DSS0_VP_CONTROL[5] GOBIT register bit of the video port the video pipeline is associated with. The software has to wait before setting the GO bit that the hardware has reset the bit. The software reset is not recommended since the application cannot guarantee to be able to reset it before the HW.

The following fields define the configuration of the video up-sampling/down-sampling block in the VID pipeline for ARGB and YUV formats:

- Vertical upsampling and downsampling increment value in the [23-0] FIRVINC bit field of DSS0_VID_FIRV (for ARGB and Y) and DSS0_VID_FIRV2 (for CbCr) registers. The unsigned integer value range is 2^{23}. Software calculates the value using the following equation:

$$FIRVINC = 2^{21} * \left(\frac{MEMSIZEY+1}{SIZEY+1}\right)$$

dispc-066

- Horizontal upsampling and downsampling increment value in the [23-0] FIRHINC bit field of the DSS0_VID_FIRH (for ARGB and Y) and DSS0_VID_FIRH2 (for CbCr) registers. The unsigned integer value range is [1:16384]. Software calculates the value using the following equation:

$$FIRHINC = 2^{21} * \left(\frac{MEMSIZEH+1}{SIZEH+1}\right)$$

dispc-067

- Vertical up/downsampling accumulator value in the [23-0] VERTICALACCU bit field of the DSS0_VID_ACCUV_0 and DSS0_VID_ACCUV_1 (for ARGB and Y), and DSS0_VID_ACCUV2_0 and DSS0_VID_ACCUV2_1 (for CbCr) registers. The accumulator value indicates on which phase the vertical filtering starts. The DSS0_VID_ACCUV_0 register is used for progressive output, while for interlace output both DSS0_VID_ACCUV_0 and DSS0_VID_ACCUV_1 registers are used. The DSS0_VID_ACCUV2_0 and DSS0_VID_ACCUV2_1 registers are used in the same manner for progressive or interlace output.
- Vertical upsampling and downsampling line buffer configuration via the DSS0_VID_ATTRIBUTES[21] VERTICALTAPS bit: The default value at reset time is 0x0 (3-tap configuration is used). If the bit field is reset, the 3-tap configuration is used.
- Horizontal upsampling and downsampling accumulator value in the [23-0] HORIZONTALACCU bit field of the DSS0_VID_ACCUH_0 and DSS0_VID_ACCUH_1 (for ARGB and Y), and DSS0_VID_ACCUH2_0 and DSS0_VID_ACCUH2_1 (for CbCr) registers. The accumulator value indicates on which phase the horizontal filtering starts. The register DSS0_VID_ACCUH_0 is used for progressive output, while for interlace output both DSS0_VID_ACCUH_0 and DSS0_VID_ACCUH_1 registers are used. The DSS0_VID_ACCUH2_0 and DSS0_VID_ACCUH2_1 are used in the same manner for progressive or interlace output.

Table 12-342 lists the scaler vertical and horizonatal accumulator values and phases.

Table 12-342. DISPC VID Scaler Vertical and Horizontal Accumulator Phases

Accumulator Value (MSB bits)	Phases f
0	0
256 or -3840	1
512 or -3584	2
768 or -3328	3
1024 or -3072	4
1280 or -2816	5
1536 or -2560	6
1792 or -2304	7
2048 or -2048	8

Copyright © 2024 Texas Instruments Incorporated

Table 12-342. DISPC VID Scaler Vertical and Horizontal Accumulator Phases (continued)

Accumulator Value (MSB bits)	Phases f
2304 or -1792	9
2560 or -1536	10
2816 or -1280	11
3072 or -1024	12
3328 or -768	13
3584 or -512	14
3840 or -256	15

- Vertical upsampling and downsampling central coefficients:
  - For ARGB and Y, the vertical upsampling and downsampling central coefficients are defined in the DSS0_VID_FIR_COEF_V0_0 to DSS0_VID_FIR_COEF_V0_8 registers. There are 9 registers for the 16 phases with 1 coefficient for each of them. Symetrical implementation is used, so only 9 coefficients are used. Each register contains one 10-bit unsigned coefficient (the central one).
  - Four CbCr, the vertical upsampling and downsampling central coefficients are set in DSS0_VID_FIR_COEF_V0_C_0 to DSS0_VID_FIR_COEF_V0_C_8 registers.
- Vertical upsampling and downsampling coefficients:
  - For ARGB and Y, the vertical upsampling and downsampling coefficients are defined in the DSS0_VID_FIR_COEF_V12_0 to DSS0_VID_FIR_COEF_V12_15 registers. There are 16 registers for the 16 phases with 2 coefficient for each of them, so a total of 32 programmable coefficients for the vertical up/down-sampling block. Each register contains two 10-bit signed coefficients.
  - Four CbCr, the vertical upsampling and downsampling coefficients are set in DSS0_VID_FIR_COEF_V12_C_0 to DSS0_VID_FIR_COEF_V12_C_15 registers.
- Horizontal upsampling and downsampling central coefficients:
  - For ARGB and Y, the horizontal upsampling and downsampling central coefficients are defined in the DSS0_VID_FIR_COEF_H0_0 to DSS0_VID_FIR_COEF_H0_8 registers. There are 9 registers for the 16 phases with 1 coefficient for each of them. Symetrical implementation is used, so only 9 coefficients are used. Each register contains one 10-bit unsigned coefficient (the central one).
  - Four CbCr, the horizontal upsampling and downsampling central coefficients are set in DSS0_VID_FIR_COEF_H0_C_0 to DSS0_VID_FIR_COEF_H0_C_8 registers.
- Horizontal upsampling and downsampling coefficients:
  - For ARGB and Y, the horizontal upsampling and downsampling coefficients are defined in the DSS0_VID_FIR_COEF_H12_0 to DSS0_VID_FIR_COEF_H12_15 registers. There are 16 registers for the 16 phases with 2 coefficient for each of them, so a total of 32 programmable coefficients for the horizontal up/down-sampling block. Each register contains two 10-bit signed coefficients.
  - Four CbCr, the horizontal upsampling and downsampling coefficients are set in DSS0_VID_FIR_COEF_H12_C_0 to DSS0_VID_FIR_COEF_H12_C_15 registers.

The YUV filtering is based on the equations of the ARGB filtering. In addition to the registers used for ARGB filtering configuration, a second set of registers for filtering is used. The first set of registers is used for Y configuration (instead of ARGB configuration) and the second set of registers is used for CbCr filtering configuration. The two sets of registers can be the same when the YUV format is not converted to RGB after filtering. When the RGB conversion is required after filtering, then the chrominance needs to be re-sampled with a different filtering configuration because:

- Chrominance samples are offset to the luminance samples. That is, DSS0_VID_FIRH2 and DSS0_VID_FIRV2, and DSS0_VID_ACCUV2_0/DSS0_VID_ACCUV2_1 and DSS0_VID_ACCUH2_0/DSS0_VID_ACCUH2_1 registers need to be configured differently than DSS0_VID_FIRH and DSS0_VID_FIRV, and DSS0_VID_ACCUH_0/DSS0_VID_ACCUH_1 and DSS0_VID_ACCUV_0/DSS0_VID_ACCUV_1 registers used for the luminance filtering.
- Chrominance is sub-sampled by two horizontally only in case of YUV422, and horizontally and vertically in case of YUV420. The coefficients for the vertical chrominance re-sampling are identical to the coefficients for

Copyright © 2024 Texas Instruments Incorporated

vertical luminance filtering in case of YUV422. The coefficients for the horizontal and vertical chrominance re-sampling are different than the ones for luminance filtering in case of YUV420.

12.6.3.8.6 DISPC VID Color Space Conversion YUV to RGB

The Color Space Conversion (CSC) unit converts the video-encoded pixel values from YUV444 format into ARGB48 format (12-bit value per component A, R, G, and B, with A fixed at 0xFFF).

In case of YUV420 or YUV422 formats, a chrominance resampling to YUV444 is performed before converting the YUV into RGB values (see Section 12.6.3.8.4, *DISPC VID Chrominance Resampling*). The YUV422/YUV420 to YUV444 chrominance resampling is a pre-processing to the color space conversion.

Figure 12-344 and Figure 12-345 show the 3 × 3 11-bit coefficients used to convert from YUV444 into ARGB48. The value of A component is fixed at 0xFFF in the output. The coefficients are set according to the standard used to encode the pixel data in YUV color space. Table 12-343 summarizes the coefficients with their respective register bit fields.

Table 12-343. DISPC VID CSC - YUV to RGB Register Settings

Coefficients	Register Fields
R_Y	DSS0_VID_CSC_COEF0[10-0] C00
R_{Cr}	DSS0_VID_CSC_COEF0[26-16] C01
R_{Cb}	DSS0_VID_CSC_COEF1[10-0] C02
G_Y	DSS0_VID_CSC_COEF1[26-16] C10
G_{Cr}	DSS0_VID_CSC_COEF2[10-0] C11
G_{Cb}	DSS0_VID_CSC_COEF2[26-16] C12
B_Y	DSS0_VID_CSC_COEF3[10-0] C20
B_{Cr}	DSS0_VID_CSC_COEF3[26-16] C21
B_{Cb}	DSS0_VID_CSC_COEF4[10-0] C22
Y offset	DSS0_VID_CSC_COEF5[15-3] PREOFFSET1
Cr offset	DSS0_VID_CSC_COEF5[31-19] PREOFFSET2
Cb offset	DSS0_VID_CSC_COEF6[15-3] PREOFFSET3

- Limited data range conversion (video data range)

  If the active range for the luminance samples (Y) is [256:3760] and for the chrominance samples (Cb and Cr) is [256:3840], the following equation in Figure 12-344 (based on 11-bit coefficients) must be used to convert the YUV to RGB. To clip the values of R, G, and B output components to the full output data range [0:4095], set the DSS0_VID_ATTRIBUTES[11] FULLRANGE bit to 0x1.

$$\begin{bmatrix} R_{OUT} \\ G_{OUT} \\ B_{OUT} \end{bmatrix} = \frac{1}{256} * \begin{bmatrix} R_Y & R_{cr} & R_{cb} \\ G_Y & G_{cr} & G_{cb} \\ B_Y & B_{cr} & B_{cb} \end{bmatrix} * \begin{bmatrix} Y_{IN} - 256 \\ Cr_{IN} - 2048 \\ Cb_{IN} - 2048 \end{bmatrix}$$

dispc-005

Figure 12-344. DISPC VID CSC YCbCr to RGB Equation (Limited Data Range), 12-Bit Outputs

  In Figure 12-344, the offset values (-256, -2048, -2048) are also programmed via corresponding register bit-fields shown in Table 12-343, *DISPC VID CSC - YUV to RGB Register Settings*).

- Full data range conversion (graphics data range)

  If the active range for the luminance samples (Y) and chrominance samples (Cb and Cr) is [0:4095], the following equation in Figure 12-345 (based on 11-bit coefficients) must be used to convert the YUV to RGB. To clip the values of R, G, and B output components to the full output data range [0:4095], set the DSS0_VID_ATTRIBUTES[11] FULLRANGE bit to 0x1.

Copyright © 2024 Texas Instruments Incorporated

$$\begin{bmatrix} R_{OUT} \\ G_{OUT} \\ B_{OUT} \end{bmatrix} = \frac{1}{256} * \begin{bmatrix} R_Y & R_{cr} & R_{cb} \\ G_Y & G_{cr} & G_{cb} \\ B_Y & B_{cr} & B_{cb} \end{bmatrix} * \begin{bmatrix} Y_{IN} \\ Cr_{IN} - 2048 \\ Cb_{IN} - 2048 \end{bmatrix}$$

dispc-006

Figure 12-345. DISPC VID CSC YCbCr to RGB Equation (Full Data Range), 12-Bit Outputs

In Figure 12-344 and Figure 12-345, the offset values (0, -256, -2048) can be programmed via the corresponding register bit-fields shown in Table 12-343, *DISPC VID CSC - YUV to RGB Register Settings*).

Note

In case of YUV420/YUV422-NV21 data, the coefficients for Cr and Cb must be swapped in order to correctly support YUV420/YUV422-NV21 format.

The scaling and CSC clipping is set by the same bit, DSS0_VID_ATTRIBUTES[11] FULLRANGE.

Output Clipping

It is possible for the output results to be larger (overflow) than the maximum or smaller (underflow) than the minimum value the output representation supports. The Color Space Conversion module clips the overflow/underflow result to the maximum/minimum value as follows:

- FULLRANGE = 1 (Graphics range)
  - Maximum: 4095
  - Minimum : 0
- FULLRANGE = 0 (Video range)
  - Maximum: 3760 for Luma, 3840 for Chroma
  - Minimum: 256 for both Luma and Chroma

The FULLRANGE = 0 setting should only be used when the CSC logic is used to convert RGB to YCrCb and the output needs to be in the video data range.

For YCrCb to RGB conversion and other RGB to RGB processing, the FULLRANGE bit should be set to 1 to enable full data range in the output.

12.6.3.8.7 DISPC VID Brightness/Contrast/Saturation/Hue Control

The brightness/contrast/saturation controls are implemented in the same Color Space Conversion (CSC) block by making adjustments to the conversion coefficients and input/output offset values as shown in Figure 12-346:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} m*A0 & m*k*B0 & m*k*C0 \\ m*A1 & m*k*B1 & m*k*C1 \\ m*A2 & m*k*B2 & m*k*C2 \end{bmatrix} \begin{bmatrix} Y + Y_offset_adj \\ Cr + Cr_offset \\ Cb + Cb_offset \end{bmatrix} + \begin{bmatrix} R_offset \\ G_offset \\ B_offset \end{bmatrix}$$

dispc-101

Figure 12-346. DISPC VID Brightness/Contrast/Saturation Equation for YUV to RGB

In Figure 12-346 the following terminology is used:

- A/B/C coefficients and the offset parameters are for YUV to RGB conversion. See Table 12-344 below for coefficients and offsets mapping to register fields.
- The gain term (m) is used for contrast control. The allowed range is: 0 < m <= 2.0 (where m=1 means contrast bypass).
- The gain term (k) is used for saturation control. The allowed range is: 0 < k <= 2.0 (where k=1 means saturation bypass).
- The brightness offset (b) is added to Y_offset_adj. The allowed range (for 8-bit example) is: -128 <= b <= +127 (where b=0 means brightness bypass).

The hue adjustment can also be built into the above equation (Figure 12-346) to comprehend the following Cb/Cr hue angle adjustments:

Copyright © 2024 Texas Instruments Incorporated

Cb_hue_adj = (Cb*cosØ + Cr* sinØ)

Cr_hue_adj = (Cr*cosØ - Cb* sinØ), where Ø is the desired hue angle (with Ø=0 meaning hue adjustment bypass)

The final combined matrix equation for controlling brightness/contrast/saturation/hue (BCSH) during YUV to RGB conversion is shown in Figure 12-347

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} m*A0 & m*k*[B0*\cos(Hue)+C0*\sin(Hue)] & m*k*[C0*\cos(Hue)-B0*\sin(Hue)] \\ m*A1 & m*k*[B1*\cos(Hue)+C1*\sin(Hue)] & m*k*[C1*\cos(Hue)-B1*\sin(Hue)] \\ m*A2 & m*k*[B2*\cos(Hue)+C2*\sin(Hue)] & m*k*[C2*\cos(Hue)-B2*\sin(Hue)] \end{bmatrix} \begin{bmatrix} Y+Y_offset_adj \\ Cr+Cr_offset \\ Cb+Cb_offset \end{bmatrix} + \begin{bmatrix} R_offset \\ G_offset \\ B_offset \end{bmatrix}$$

dispc-102

Figure 12-347. DISPC VID Brightness/Contrast/Saturation/Hue Equation for YUV to RGB

Gain and offset terms in the above equation (Figure 12-347) are programmed in the hardware registers as adjusted CSC coefficients (11-bit signed) and offset values (13-bit signed) to control the brightness, contrast, saturation and hue. Mapping of the coefficients and offsets to register fields is provided in Table 12-344. Outputs are clipped to keep the output values in the proper range.

Table 12-344. DISPC VID BCSH Register Settings for YUV to RGB

Coefficients	Register Fields
A0	DSS0_VID_CSC_COEF0[10-0] C00
B0	DSS0_VID_CSC_COEF0[26-16] C01
C0	DSS0_VID_CSC_COEF1[10-0] C02
A1	DSS0_VID_CSC_COEF1[26-16] C10
B1	DSS0_VID_CSC_COEF2[10-0] C11
C1	DSS0_VID_CSC_COEF2[26-16] C12
A2	DSS0_VID_CSC_COEF3[10-0] C20
B2	DSS0_VID_CSC_COEF3[26-16] C21
C2	DSS0_VID_CSC_COEF4[10-0] C22
Y offset	DSS0_VID_CSC_COEF5[15-3] PREOFFSET1
Cr offset	DSS0_VID_CSC_COEF5[31-19] PREOFFSET2
Cb offset	DSS0_VID_CSC_COEF6[15-3] PREOFFSET3
R offset	DSS0_VID_CSC_COEF6[31-19] POSTOFFSET1
G offset	DSS0_VID_CSC_COEF7[15-3] POSTOFFSET2
B offset	DSS0_VID_CSC_COEF7[31-19] POSTOFFSET3

For RGB input formats, the same CSC block can be used to adjust contrast, brightness, saturation, and hue using the matrix equation from Figure 12-348.

$$\begin{bmatrix} Rout \\ Gout \\ Bout \end{bmatrix} = \begin{bmatrix} m*A0 & m*k*[B0*\cos(Hue)+C0*\sin(Hue)] & m*k*[C0*\cos(Hue)-B0*\sin(Hue)] \\ m*A1 & m*k*[B1*\cos(Hue)+C1*\sin(Hue)] & m*k*[C1*\cos(Hue)-B1*\sin(Hue)] \\ m*A2 & m*k*[B2*\cos(Hue)+C2*\sin(Hue)] & m*k*[C2*\cos(Hue)-B2*\sin(Hue)] \end{bmatrix} \begin{bmatrix} a0 & b0 & c0 \\ a1 & b1 & c1 \\ a2 & b2 & c2 \end{bmatrix} \begin{bmatrix} Rin \\ Gin \\ Bin \end{bmatrix} + \begin{bmatrix} R_offset \\ G_offset \\ B_offset \end{bmatrix}$$

Coefficients a/b/c are related to A/B/C (for RGB to YUV and YUV to RGB conversions) as follows:

$$\begin{bmatrix} a0 & b0 & c0 \\ a1 & b1 & c1 \\ a2 & b2 & c2 \end{bmatrix} = \begin{bmatrix} A0 & B0 & C0 \\ A1 & B1 & C1 \\ A2 & B2 & C2 \end{bmatrix}^{-1}$$

dispc-103

Figure 12-348. DISPC VID Brightness/Contrast/Saturation/Hue Equation for RGB Input

In Figure 12-348 the following specifics apply:

- A/B/C coefficients and the RGB offset parameters mapping to register fields is provided in Table 12-344.

Copyright © 2024 Texas Instruments Incorporated

- The gain term (m) is used for contrast control. The allowed range is: 0 < m <= 2.0 (where m=1 means contrast bypass).
- The gain term (k) is used for saturation control. The allowed range is: 0 < k <= 2.0 (where k=1 means saturation bypass).
- The brightness is controlled by the RGB offset values.
- The Hue angle is used to adjust hue. Hue Ø=0 means hue adjustment bypass.

Outputs are clipped to keep the output values in the proper range.

12.6.3.8.8 DISPC VID Luma Key Support

The video pipeline supports Luma key transparency for Blue-ray video layer composition. When enabled, Y value of each pixel will be checked against luma key range values. The range values are defined in DSS0_VID_LUMAKEY register fields, as follows: [11-0] LUMAKEYMIN < Y < [27-16] LUMAKEYMAX. If this condition is true, then the pixel alpha value will be forced to zero (transparent) to make the pixel transparent during the blending process in the overlay manager.

12.6.3.8.9 DISPC VID Cropping Support

At the output of the video pipeline a final crop stage is added which allows further cropping of the video window to the required dimensions. The crop feature is enabled by DSS0_VID_ATTRIBUTES[13] CROP register field and the cropping dimensions are further set by DSS0_VID_CROP[...] CROPLEFT/CROPTOP/CROPBOTTOM/CROPRIGHT register fields. Cropping of up to 31 pixels/lines is supported in every direction.

The crop feature is especially useful while dealing with the compressed frame buffer using the FBDC module, as explained in Section 12.6.3.6.11, *DISPC Compressed Data Format Support*.

12.6.3.9 DISPC Write-Back Pipeline

The write-back (WB) pipeline is used to store in the system memory the capture of the overlay output or the output of one of the pipelines. The WB pipeline consists of a CSC unit, a scaler unit, and an RGB truncation logic. The format from the overlay managers is always ARGB48. The format from the output of the VID pipeline can be YUV422, YUV420 or ARGB48. Because the overlay works on ARGB48 format and the video accelerator works on YUV format, the color space conversion from RGB to YUV is used to directly output to memory the format that can be encoded with no extra processing.

The write-back pipeline can be connected either to the VID pipeline output or to the output of one of the overlay managers, either in M2M (memory-to-memory) or Capture mode (see Section 12.6.3.6.3, *DISPC Write DMA Buffer*), by configuring the DSS0_COMMON_DISPC_CONNECTIONS[20-16] WB_CONN register field.

- In M2M mode, the pipeline or overlay manager connected to WB must not be connected to any VP.
- In Capture mode, the pipeline or overlay manager connected to WB must be connected to at least one VP.

The capture frame rate can be set in the DSS0_WB_ATTRIBUTES[26:24] CAPTUREMODE bit field.

The write-back pipeline also supports input source cropping to allow a sub-frame capture.

The ARGB48 format is truncated (LSB drop) or expanded to match the output color depth formats. No dithering on the output is supported. When there is no alpha field mentioned by x in the pixel format description, 0's shall be used. For example, for RGB12 pixel format the upper 4 bits are 0’s since RGB value is only 12 bits inside a 16-bit container.

The WB pipeline is enabled by setting the DSS0_WB_ATTRIBUTES[0] ENABLE bit to 0x1.

Figure 12-349 shows the WB pipeline.

Copyright © 2024 Texas Instruments Incorporated

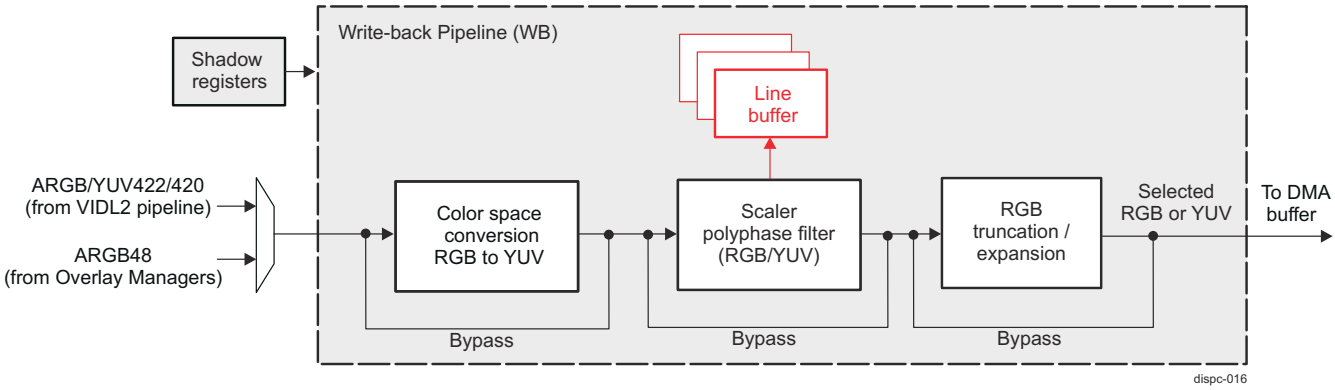


Figure 12-349. DISPC Write-Back Pipeline

12.6.3.9.1 DISPC WB Color Space Conversion RGB to YUV

The WB CSC unit converts the encoded pixel values from RGB48 into YUV444 format. For YUV420 or YUV422 as output formats, a chrominance subsampling (YUV444 to YUV422, and YUV422 to YUV420) is performed in the WB scaler block afer the CSC operation.

Figure 12-350 and Figure 12-351 show the 3 × 3 11-bit coefficients used for the CSC operation. The user sets the coefficients according to the standard used to encode the pixel data in YUV color space. Table 12-345 lists the coefficients with their respective bit fields, together with the post offsets (that are 13-bit signed integer numbers).

Table 12-345. DISPC WB CSC - RGB to YUV Register Settings

Coefficients	Register Bit Fields
Y_R	DSS0_WB_CSC_COEF0[10:0] C00
Y_G	DSS0_WB_CSC_COEF0[26:16] C01
Y_B	DSS0_WB_CSC_COEF1[10:0] C02
Cr_R	DSS0_WB_CSC_COEF1[26:16] C10
Cr_G	DSS0_WB_CSC_COEF2[10:0] C11
Cr_B	DSS0_WB_CSC_COEF2[26:16] C12
Cb_R	DSS0_WB_CSC_COEF3[10:0] C20
Cb_G	DSS0_WB_CSC_COEF3[26:16] C21
Cb_B	DSS0_WB_CSC_COEF4[10:0] C22
$R_{POSTOFFSET}$	DSS0_WB_CSC_COEF6[31-19] POSTOFFSET1
$G_{POSTOFFSET}$	DSS0_WB_CSC_COEF7[15-3] POSTOFFSET2
$B_{POSTOFFSET}$	DSS0_WB_CSC_COEF7[31-19] POSTOFFSET3

Limited data range conversion (video data range)

If the active range for the 12-bit output luminance samples (Y) is to be [256:3760] and for the output chrominance samples (Cb and Cr) is to be [256:3840], then Figure 12-350 gives the equation (based on 11-bit coefficients) to convert the RGB to YUV. The range selection is done by setting the DSS0_WB_ATTRIBUTES[11] FULLRANGE bit to 0x0. The [256, 2048, 2048] offset values are programmable through the registers from Table 12-345.

$$\begin{bmatrix} Y_{OUT} \\ Cb_{OUT} \\ Cr_{OUT} \end{bmatrix} = \frac{1}{256} * \begin{bmatrix} Y_R & Y_G & Y_B \\ Cb_R & Cb_G & Cb_B \\ Cr_R & Cr_G & Cr_B \end{bmatrix} * \begin{bmatrix} R_{IN} \\ G_{IN} \\ B_{IN} \end{bmatrix} + \begin{bmatrix} 256 \\ 2048 \\ 2048 \end{bmatrix}$$

dispc-017

Figure 12-350. DISPC WB CSC RGB to YCbCr Equation (Limited Data Range), 12-bit Outputs

Copyright © 2024 Texas Instruments Incorporated

Full data range conversion (graphics data range)

If the active range for the output luminance samples (Y) and or the chrominance samples (Cb and Cr) is [0:4095], then Figure 12-351 gives the equation (based on 11-bit coefficients) to convert the RGB to YUV. The [0, 2048, 2048] offset values are programmable through the registers from Table 12-345.

$$\begin{bmatrix} Y_{OUT} \\ Cb_{OUT} \\ Cr_{OUT} \end{bmatrix} = \frac{1}{256} * \begin{bmatrix} Y_R & Y_G & Y_B \\ Cb_R & Cb_G & Cb_B \\ Cr_R & Cr_G & Cr_B \end{bmatrix} * \begin{bmatrix} R_{IN} \\ G_{IN} \\ B_{IN} \end{bmatrix} + \begin{bmatrix} 0 \\ 2048 \\ 2048 \end{bmatrix}$$

dispc-018

Figure 12-351. DISPC WB CSC RGB to YCbCr Equation (Full Data Range), 12-bit Outputs

12.6.3.9.2 DISPC WB Scaler Unit

The functional aspect of the WB pipeline scaler unit is identical to the VID pipeline scaler unit (see Section 12.6.3.8.5, *DISPC VID Scaler Unit*), except that the WB scaler is 8-bit only and takes only the 8 most significant bits of the data fed through. The WB scaler, similarly to the one in the video pipeline, can perform scaling in the native ARGB, YUV422, or YUV420 formats. The support for scaling in YUV formats allows a YUV source to be scaled through video and write-back scalers without chroma up and downsampling, if the output is to remain in the same format. In addition, the scaling capability of VID and WB pipelines can be combined to double the maximum rescaling factor (applicable for memory-to-memory operations only).

The downscaling capability is further reduced, if the pipeline is doing a format conversion where downsampling is inherent. When doing RGB/YUV422-to-YUV420 conversion in WB pipeline, the downsampling capability of WB scaler is reduced to:

- If 5 taps is selected, then the downsampling is limited to 0.5 in RGB
- If 3 taps is selected, then no down-scaling is available

The write-back window attributes of the WB pipeline can be configured in the following register bit-fields:

- Source data format for write-back (input to the scaler unit) in DSS0_WB_ATTRIBUTES[6-1] FORMAT bit-field
- Start position (offset) of the window on the overlay which WB will capture in DSS0_WB_POSITION[29-16] POSY and [13-0] POSX register field. Only applicable when the WB pipeline is operating in capture_mode.
- Write-back picture width in system memory (frame buffer) in DSS0_WB_PICTURE_SIZE[13-0] MEMSIZEX bit-field. The window width is from 1 up to 4096 pixels. All the integer values in the range [1:4096] are allowed in the case of RGB16 and RGB24 video data. In the case of YUV2422 and UYVY422 formats the width has to be a multiple of 2 pixels.
- Write-back picture height in system memory (frame buffer) in DSS0_WB_PICTURE_SIZE[29-16] MEMSIZEY bit-field. The window width is from 1 up to 4096 pixels. All the integer values in the range [1:4096] are allowed. In the case of YUV420 the height has to be a multiple of 2 lines.
- Write-back window width at pipeline input (before resizing) in DSS0_WB_SIZE[13-0] SIZEX bit-field. The window width is from 1 up to 4096 pixels. The width of the image sent to the write back pipe has to be a multiple of 2 pixels, if its format (input) is YUV420/YUV422.
- Write-back window height at pipeline input (before resizing) in DSS0_WB_SIZE[29-16] SIZEY bit-field. The window height is from 1 up to 4096 pixels. The width of the image sent to the write back pipe has to be a multiple of 2 lines, if its format (input) is YUV420.

Note

The WB pipeline supports input source cropping to allow a sub-frame capture by specifying the offset position and the size of the sub-frame. With the offset position parameters (POSY, POSX) set to (0,0) and the size prameters (SIZEY, SIZEX) set to the full size, no clipping should be applied. The software must make sure that the clipping parameters are set up correctly to avoid going over the WB input frame boundaries. Otherwise, the pipeline may lock up.

Table 12-346 list the register fields in the function of the coefficients for the VID horizontal scaler in the DSS0_WB_FIR_COEF_H0_0 to DSS0_WB_FIR_COEF_H0_8, and DSS0_WB_FIR_COEF_H12_0 to DSS0_WB_FIR_COEF_H12_15 registers.

Copyright © 2024 Texas Instruments Incorporated

Table 12-346. DISPC Register Fields Associated to Coefficients for ARGB and Y Configuration in WB Horizontal Scaler

Phases	Ch(2)	Ch(1)	Ch(0)	Ch(-1)	Ch(-2)
	Signed coefficient [29-20] FIRHC2 bitfield	Signed coefficient [19-10] FIRHC1 bitfield	Unsigned central coefficient [9-0] FIRHC0 bitfield	Signed coefifcient [19-10] FIRHC1 bitfield	Signed coefficient [29-20] FIRHC2 bitfield
0	DSS0_WB_FIR_COEF_H12_0	DSS0_WB_FIR_COEF_H12_0	DSS0_WB_FIR_COEF_H0_0	DSS0_WB_FIR_COEF_H12_0	DSS0_WB_FIR_COEF_H12_0
1	DSS0_WB_FIR_COEF_H12_1	DSS0_WB_FIR_COEF_H12_1	DSS0_WB_FIR_COEF_H0_1	DSS0_WB_FIR_COEF_H12_15	DSS0_WB_FIR_COEF_H12_15
2	DSS0_WB_FIR_COEF_H12_2	DSS0_WB_FIR_COEF_H12_2	DSS0_WB_FIR_COEF_H0_2	DSS0_WB_FIR_COEF_H12_14	DSS0_WB_FIR_COEF_H12_14
3	DSS0_WB_FIR_COEF_H12_3	DSS0_WB_FIR_COEF_H12_3	DSS0_WB_FIR_COEF_H0_3	DSS0_WB_FIR_COEF_H12_13	DSS0_WB_FIR_COEF_H12_13
4	DSS0_WB_FIR_COEF_H12_4	DSS0_WB_FIR_COEF_H12_4	DSS0_WB_FIR_COEF_H0_4	DSS0_WB_FIR_COEF_H12_12	DSS0_WB_FIR_COEF_H12_12
5	DSS0_WB_FIR_COEF_H12_5	DSS0_WB_FIR_COEF_H12_5	DSS0_WB_FIR_COEF_H0_5	DSS0_WB_FIR_COEF_H12_11	DSS0_WB_FIR_COEF_H12_11
6	DSS0_WB_FIR_COEF_H12_6	DSS0_WB_FIR_COEF_H12_6	DSS0_WB_FIR_COEF_H0_6	DSS0_WB_FIR_COEF_H12_10	DSS0_WB_FIR_COEF_H12_10
7	DSS0_WB_FIR_COEF_H12_7	DSS0_WB_FIR_COEF_H12_7	DSS0_WB_FIR_COEF_H0_7	DSS0_WB_FIR_COEF_H12_9	DSS0_WB_FIR_COEF_H12_9
8	DSS0_WB_FIR_COEF_H12_8	DSS0_WB_FIR_COEF_H12_8	DSS0_WB_FIR_COEF_H0_8	DSS0_WB_FIR_COEF_H12_8	DSS0_WB_FIR_COEF_H12_8
9	DSS0_WB_FIR_COEF_H12_9	DSS0_WB_FIR_COEF_H12_9	DSS0_WB_FIR_COEF_H0_7	DSS0_WB_FIR_COEF_H12_7	DSS0_WB_FIR_COEF_H12_7
10	DSS0_WB_FIR_COEF_H12_10	DSS0_WB_FIR_COEF_H12_10	DSS0_WB_FIR_COEF_H0_6	DSS0_WB_FIR_COEF_H12_6	DSS0_WB_FIR_COEF_H12_6
11	DSS0_WB_FIR_COEF_H12_11	DSS0_WB_FIR_COEF_H12_11	DSS0_WB_FIR_COEF_H0_5	DSS0_WB_FIR_COEF_H12_5	DSS0_WB_FIR_COEF_H12_5
12	DSS0_WB_FIR_COEF_H12_12	DSS0_WB_FIR_COEF_H12_12	DSS0_WB_FIR_COEF_H0_4	DSS0_WB_FIR_COEF_H12_4	DSS0_WB_FIR_COEF_H12_4
13	DSS0_WB_FIR_COEF_H12_13	DSS0_WB_FIR_COEF_H12_13	DSS0_WB_FIR_COEF_H0_3	DSS0_WB_FIR_COEF_H12_3	DSS0_WB_FIR_COEF_H12_3
14	DSS0_WB_FIR_COEF_H12_14	DSS0_WB_FIR_COEF_H12_14	DSS0_WB_FIR_COEF_H0_2	DSS0_WB_FIR_COEF_H12_2	DSS0_WB_FIR_COEF_H12_2
15	DSS0_WB_FIR_COEF_H12_15	DSS0_WB_FIR_COEF_H12_15	DSS0_WB_FIR_COEF_H0_1	DSS0_WB_FIR_COEF_H12_1	DSS0_WB_FIR_COEF_H12_1

Note

In Table 12-346, the cells without color are duplicated from the grey cells.

Similar table approach applies to the vertical scaler (registers DSS0_WB_FIR_COEF_V0_0 to DSS0_WB_FIR_COEF_V0_8, and DSS0_WB_FIR_COEF_V12_0 to DSS0_WB_FIR_COEF_V12_15 are used).

Similar table approach applies to the coefficients for Cb/Cr filtering in case of YUV format (registers DSS0_WB_FIR_COEF_H0_C_0 to DSS0_WB_FIR_COEF_H0_C_8, and DSS0_WB_FIR_COEF_H12_C_0 to DSS0_WB_FIR_COEF_H12_C_15, and DSS0_WB_FIR_COEF_V0_C_0 to DSS0_WB_FIR_COEF_V0_C_8 and DSS0_WB_FIR_COEF_V12_C_0 to DSS0_WB_FIR_COEF_V12_C_15 are used).

The WB scaler unit vertical and/or horizontal sampling is selected by configuring the DSS0_WB_ATTRIBUTES[8-7] RESIZEENABLE register bit field.

Prior to enabling the WB scaler a valid configuration has to be set by the user.

Copyright © 2024 Texas Instruments Incorporated

In case of capturing data of one of the output channels, the corresponding DSS0_VP_CONTROL[5] GOBIT bit has to be set to update the configuration depending to which VP output the write-back pipeline is associated with. Refers also to Section 12.6.3.15, *DISPC Shadow Mechanism for Registers* when write back channel is active. The SW has to wait before setting the GOBIT bit that the HW has reset the same bit. The DSS0_WB_ATTRIBUTES[0] ENABLE bit can be set to update those registers, if it has been previously disabled.

The following register fields define the configuration of the video up/downsampling block in the WB pipeline:

- Vertical up/downsampling increment value in the[23-0] FIRVINC bit field of DSS0_WB_FIRV (for ARGB and Y) and DSS0_WB_FIRV2 (for CbCr) registers. Software calculates the value using the following equation:

$$FIRVINC = 2^{21} * \left(\frac{SIZEY+1}{MEMSIZEY+1}\right) \quad \text{dispc-066} \tag{25}$$

- Horizontal up/downsampling increment value in the [23-0] FIRHINC bit field of the DSS0_WB_FIRH (for ARGB and Y) and DSS0_WB_FIRH2 (for CbCr) registers. Software calculates the value using the following equation:

$$FIRHINC = 2^{21} * \left(\frac{SIZEX+1}{MEMSIZEX+1}\right) \quad \text{dispc-067} \tag{26}$$

- Vertical up/downsampling accumulator value in the [23-0] VERTICALACCU bit field of the DSS0_WB_ACCUV_0 and DSS0_WB_ACCUV_1 (for ARGB and Y), and DSS0_WB_ACCUV2_0 and DSS0_WB_ACCUV2_1 (for CbCr) registers. The accumulator value indicates on which phase the vertical filtering starts.
- Vertical upsampling and downsampling line buffer configuration via the DSS0_WB_ATTRIBUTES[21] VERTICALTAPS bit. The default value at reset time is 0x0 (3-tap configuration is used). If the bit field is reset, the 3-tap configuration is used.
- Horizontal upsampling and downsampling accumulator value in the [23-0] HORIZONTALACCU bit field of the DSS0_WB_ACCUH_0 and DSS0_WB_ACCUH_1 (for ARGB and Y), and DSS0_WB_ACCUH2_0 and DSS0_WB_ACCUH2_1 (for CbCr) registers. The accumulator value indicates on which phase the horizontal filtering starts.

Table 12-347 lists the DISPC vertical and horizontal accumulator values and phases. Other accumulator values are also supported. The HW determines the nearest phase value in order to load the corresponding one.

Table 12-347. DISPC WB Scaler Vertical and Horizontal Accumulator Phases

Accumulator Value (MSB bits)	Phases f
0	0
256 or -3840	1
512 or -3584	2
768 or -3328	3
1024 or -3072	4
1280 or -2816	5
1536 or -2560	6
1792 or -2304	7
2048 or -2048	8
2304 or -1792	9
2560 or -1536	10
2816 or -1280	11
3072 or -1024	12
3328 or -768	13
3584 or -512	14

Copyright © 2024 Texas Instruments Incorporated

Table 12-347. DISPC WB Scaler Vertical and Horizontal Accumulator Phases (continued)

Accumulator Value (MSB bits)	Phases f
3840 or -256	15

- Vertical upsampling and downsampling central coefficients:
  - For ARGB and Y, the vertical upsampling and downsampling central coefficients are defined in the DSS0_WB_FIR_COEF_V0_0 to DSS0_WB_FIR_COEF_V0_8 registers. There are 9 registers for the 16 phases with 1 coefficient for each of them. Symetrical implementation is used, so only 9 coefficients are used. Each register contains one 10-bit unsigned coefficient (the central one).
  - Four CbCr, the vertical upsampling and downsampling central coefficients are set in DSS0_WB_FIR_COEF_V0_C_0 to DSS0_WB_FIR_COEF_V0_C_8 registers.
- Vertical upsampling and downsampling coefficients:
  - For ARGB and Y, the vertical upsampling and downsampling coefficients are defined in the DSS0_WB_FIR_COEF_V12_0 to DSS0_WB_FIR_COEF_V12_15 registers. There are 16 registers for the 16 phases with 2 coefficient for each of them, so a total of 32 programmable coefficients for the vertical up/down-sampling block. Each register contains two 10-bit signed coefficients.
  - Four CbCr, the vertical upsampling and downsampling coefficients are set in DSS0_WB_FIR_COEF_V12_C_0 to DSS0_WB_FIR_COEF_V12_C_15 registers.
- Horizontal upsampling and downsampling central coefficients:
  - For ARGB and Y, the horizontal upsampling and downsampling central coefficients are defined in the DSS0_WB_FIR_COEF_H0_0 to DSS0_WB_FIR_COEF_H0_8 registers. There are 9 registers for the 16 phases with 1 coefficient for each of them. Symetrical implementation is used, so only 9 coefficients are used. Each register contains one 10-bit unsigned coefficient (the central one).
  - Four CbCr, the horizontal upsampling and downsampling central coefficients are set in DSS0_WB_FIR_COEF_H0_C_0 to DSS0_WB_FIR_COEF_H0_C_8 registers.
- Horizontal upsampling and downsampling coefficients:
  - For ARGB and Y, the horizontal upsampling and downsampling coefficients are defined in the DSS0_WB_FIR_COEF_H12_0 to DSS0_WB_FIR_COEF_H12_15 registers. There are 16 registers for the 16 phases with 2 coefficient for each of them, so a total of 32 programmable coefficients for the horizontal up/down-sampling block. Each register contains two 10-bit signed coefficients.
  - Four CbCr, the horizontal upsampling and downsampling coefficients are set in DSS0_WB_FIR_COEF_H12_C_0 to DSS0_WB_FIR_COEF_H12_C_15 registers.

The YUV filtering is based on the equations of the ARGB filtering. In addition to the registers used for ARGB filtering configuration, a second set of registers for filtering is used. The first set of registers is used for Y configuration (instead of ARGB configuration) and the second set of registers is used for CbCr filtering configuration. The two sets of registers can be the same when the YUV format is not converted to RGB after filtering. When the RGB conversion is required after filtering, then the chrominance needs to be re-sampled with a different filtering configuration because:

- Chrominance samples are offset to the luminance samples. That is, DSS0_WB_FIRH2 and DSS0_WB_FIRV2, and DSS0_WB_ACCUV2_0/DSS0_WB_ACCUV2_1 and DSS0_WB_ACCUH2_0/DSS0_WB_ACCUH2_1 registers need to be configured differently than DSS0_WB_FIRH and DSS0_WB_FIRV, and DSS0_WB_ACCUH_0/DSS0_WB_ACCUH_1 and DSS0_WB_ACCUV_0/DSS0_WB_ACCUV_1 registers used for the luminance filtering.
- Chrominance is sub-sampled by two horizontally only in case of YUV422, and horizontally and vertically in case of YUV420. The coefficients for the vertical chrominance re-sampling are identical to the coefficients for vertical luminance filtering in case of YUV422. The coefficients for the horizontal and vertical chrominance re-sampling are different than the ones for luminance filtering in case of YUV420.

12.6.3.10 DISPC Overlay Manager

The overlay manager (OVR) is responsible for compositing selected input layers together to generate the final display output frame. The DISPC implements four identical overlay managers, one for each display output (see Figure 12-311, *DISPC Architecture Overview*). Any of these overlay managers can be used to source also the Write-back pipeline.

Copyright © 2024 Texas Instruments Incorporated

- OVR1 is connected to Video Port 1 (VP1).
- OVR2 is connected to Video Port 2 (VP2).
- OVR3 is connected to Video Port 3 (VP3).
- OVR4 is connected to Video Port 4 (VP4).

Each OVR is preceded by a programmable input pipeline selector which re-maps the output of the selected input pipelines according to the overlay manager Z-order configuration.

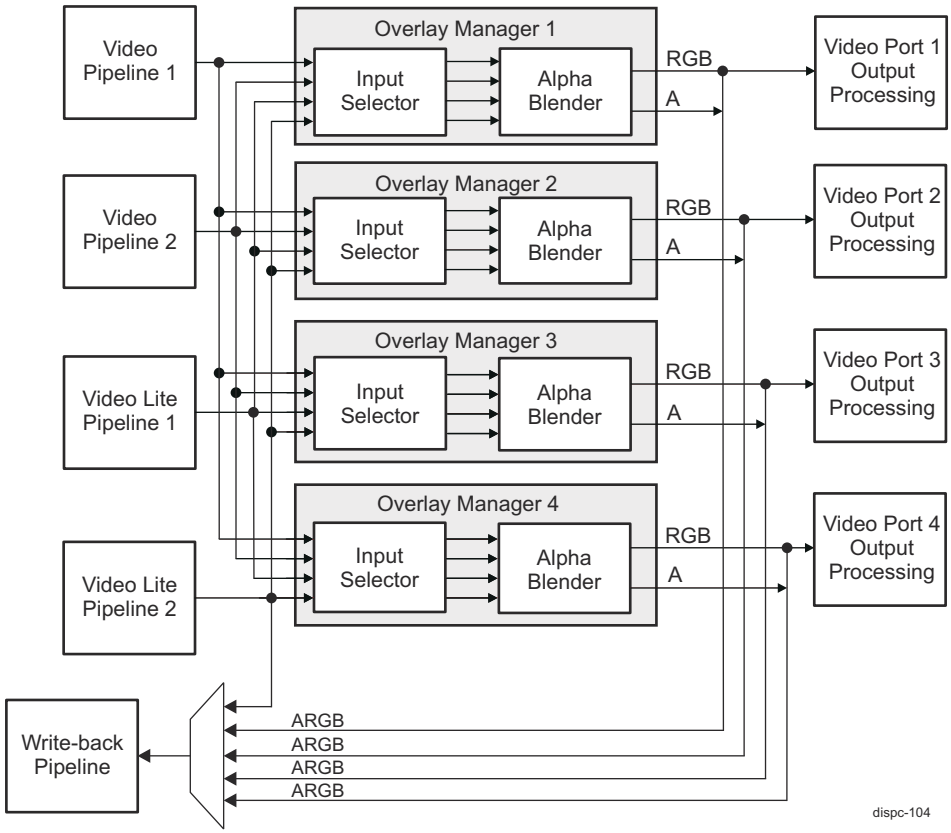


Figure 12-352. DISPC Overlay Manager and Input Selector

Note

The DISPC overlay managers, OVR1 through OVR4, will be commonly refered to as *overlay manager (OVR)* in the following sections.

12.6.3.10.1 DISPC Overlay Input Selector

The overlay input selector before the overlay manager is a n-input to n-output crossbar switch. As such, any input can be connected to any output in the selector. The selection is based on the Z-order layer selection in the overlay configuration through the following register fields:

- DSS0_OVR_ATTRIBUTES_0[4-1] CHANNELIN field for layer 0, Z-order 0
- DSS0_OVR_ATTRIBUTES_1[4-1] CHANNELIN field for layer 1, Z-order 1
- DSS0_OVR_ATTRIBUTES_2[4-1] CHANNELIN field for layer 2, Z-order 2
- DSS0_OVR_ATTRIBUTES_3[4-1] CHANNELIN field for layer 3, Z-order 3
- DSS0_OVR_ATTRIBUTES_4[4-1] CHANNELIN field for layer 4, Z-order 4

The z-order determines the order in which the selected layers are blended together to generate the final output by the overlay manager:

- The lowest enabled order input is the bottom-most layer, just above the background color
- The highest enabled order input is the top-most layer

Copyright © 2024 Texas Instruments Incorporated

The OVR input selector also passes the following attributes from the selected input pipeline to the corresponding selector output port (*zsel*):

- Source pipeline size attributes:
  - Number of pixels per line (from DSS0_VID_SIZE[13-0] SIZEX register field)
  - Number of lines (from DSS0_VID_SIZE[29-16] SIZEY regiser field)
- Source pipeline global blending level (from DSS0_VID_GLOBAL_ALPHA register)

Then, the overlay manager uses the above listed attributes as input layer attributes along with POSX and POSY overlay manager configuration parameters. The POSX and POSY can be configured in the [13-0] POSX and [29-16] POSY fields of the following registers:

- For layer 0, Z-order 0, in DSS0_OVR_ATTRIBUTES2_0 register
- For layer 1, Z-order 1, in DSS0_OVR_ATTRIBUTES2_1 register
- For layer 2, Z-order 2, in DSS0_OVR_ATTRIBUTES2_2 register
- For layer 3, Z-order 3, in DSS0_OVR_ATTRIBUTES2_3 register
- For layer 4, Z-order 4, in DSS0_OVR_ATTRIBUTES2_4 register

Note

The output of an input pipeline can be mapped to more than one overlay manager simultaneously, if and only if the following conditions are true:

- All video port timing generators controlling the overlay managers are identically programmed (same frame width/height with same blanking parameters running at same frame rate) and clocked by the same pixel clock source, AND ...
- The pipeline output is positioned on each display output at the same frame position, OR ...
- One of the overlay managers is connected to the write-back path only.

The software is responsible for managing the pipeline assignment and setting up the video port timing generators properly in order to avoid any resource conflicts that may cause the DISPC to lock up (sync loss).

12.6.3.10.2 DISPC Overlay Mechanism

The overlay mechanism consists of displaying more than one input layer (graphics and/or video layers) using:

- A priority rule based on the display Z-order (selected by configuring the overlay input selector, as described in Section 12.6.3.10.1)
- Transparency color keys configuration (destination and source transparency)
- Alpha blending values (using the alpha component of each pixel or a global alpha value set by the user)
- Size and position attributes of the source window data, as described in Section 12.6.3.10.1

The overlay manager is configured by using the Z-order parameter. The Z-order value defined for each pipeline indicates the visibility order of the window on the screen. If the Z-order value of window A is lower than the Z-order of layer B, then layer A is displayed below layer B. The transparency color keys and the alpha blending factors are then used to blend the layers together (see Section 12.6.3.10.2.1, *Overlay Alpha Blender*, and Section 12.6.3.10.2.2, *Overlay Transparency Color Keys*).

Figure 12-353 gives an simplified example of how two input pipeline frame outputs are merged together to generate the final display output.

Copyright © 2024 Texas Instruments Incorporated

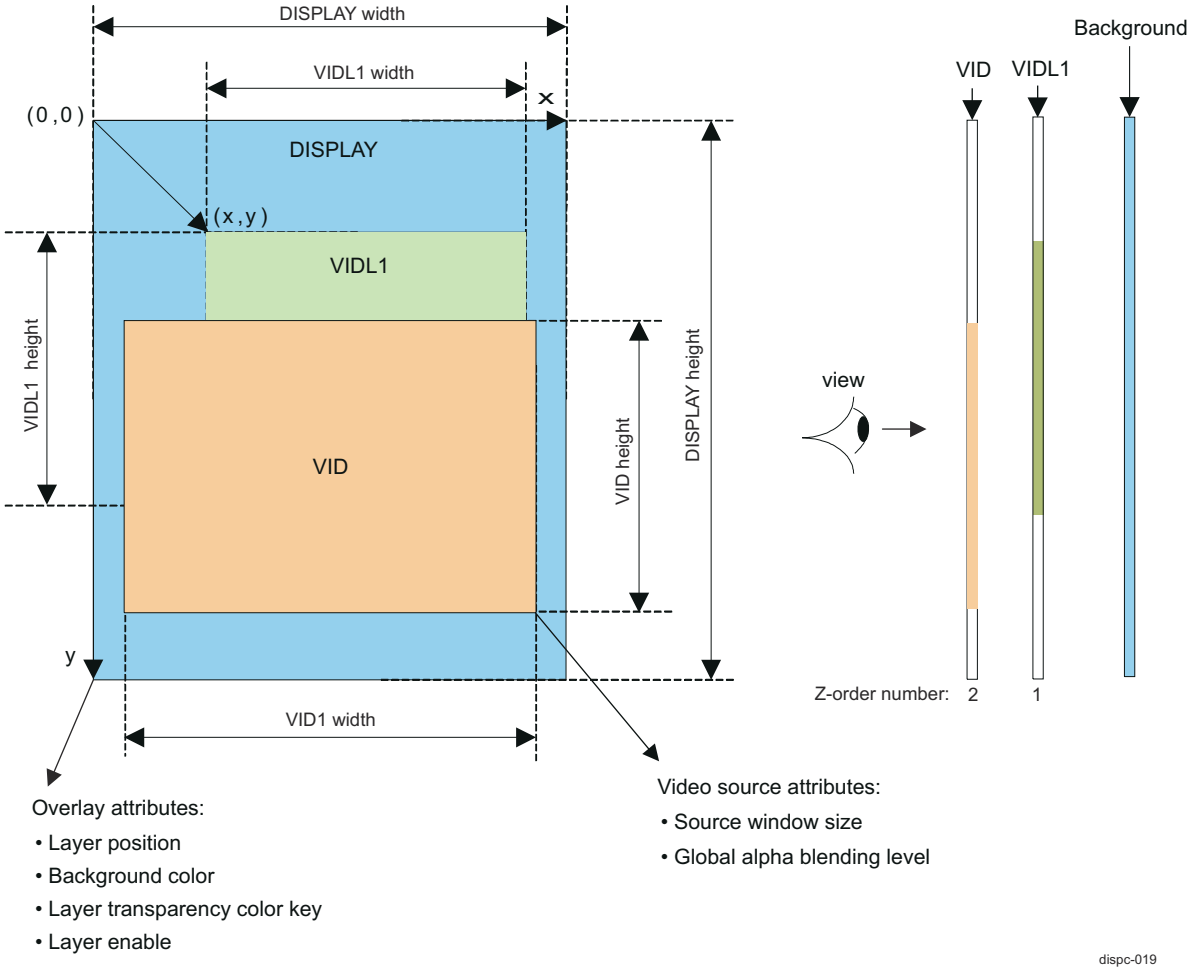


Figure 12-353. DISPC Overlay Example and Display Attributes

As shown in Figure 12-353, each input pipeline generates one rectangular sub-frame output in ARGB48 format. The overlay is responsible for positioning the sub-frame in the display output frame (specified by DSS0_VP_SIZE_SCREEN[13-0] PPL and [29-16] LPP video port register fields), using the overlay window position parameters (specified by the [13-0] POSX and [29-16] POSY overlay fields in the DSS0_OVR_ATTRIBUTES2_0 through DSS0_OVR_ATTRIBUTES2_4 registers), and the window size parameters (specified by the [13-0] SIZEX and [29-16] SIZEY input pipeline fields in the DSS0_VID_SIZE registers). When the overlay manager does not require the input pipeline data (either because it is currently outputting the background color pixel or it is being stalled by the VP output module), the overlay manager stalls the input pipeline.

When there are no video-encoded pixels at a specific position, the programmable solid background color appears. The solid background color can be set in the DSS0_OVR_DEFAULT_COLOR and DSS0_OVR_DEFAULT_COLOR2 registers.

The entire pixels of the video window have to be inside the display screen. Depending on the width of the buffer to be displayed in the video layer and the position, the width must be adjusted by software to limit the right edge of the window inside the display screen. The same is also true for the video layer height in the input pipe line configuration.

12.6.3.10.2.1 Overlay Alpha Blender

The overlay manager can blend as many as four input pipelines using three cascaded stages of basically the same alpha-blending logic. Alpha output generation is supported only for the write-back path.

The alpha blending value is defined by:

Copyright © 2024 Texas Instruments Incorporated

- The component value A when using an ARGB or RGBA pixel format (alpha in the source pixel data is converted to 8-bit alpha value in the input pipeline logic):
  - For ARGB-1555, the alpha blending is defined using a 1-bit value. It is converted into an 8-bit value by duplicating the 1-bit value (see Table 12-348).
  - For ARGB-4444, the alpha blending is defined using a 4-bit value. It is converted into an 8-bit value by duplicating the 4-bit value (see Table 12-348).
  - If the pixel format contains no alpha blending value, the pixel alpha value is considered to be 0xFF, and if alpha is equal to 0xFF, there is no multiplication.
  - For BITMAP or YUV formats, there is no alpha blending factor associated with each pixel value. Only the global alpha blending factor associated with the window displaying the BITMAP or YUV format is used.
- The global alpha blending value can be set in the DSS0_VID_GLOBAL_ALPHA register of the input pipeline, and is updated in synch with the selected output channel.
- The modulated alpha blending value (that is, a total alpha blending value, when a combination of the pixel alpha blending value A and a global alpha blending ia present) is determined as: Alpha = (Pixel_Alpha × Global_Alpha) / 256.

Table 12-348 shows the remapping and the percentage of alpha blending achieved.

Table 12-348. DISPC Overlay Alpha Blending – ARGB

Alpha Blending 1-Bit Value (ARGB16-1555)	Alpha Blending 4-Bit Value (ARGB16-4444)	Alpha Blending 8-Bit Value (Converted Value)	% Blending
0x0	0x0	0x00	100% (transparent)
N/A	0x1	0x11	93.33%
N/A	0x2	0x22	86.6%
N/A	...	...	...
N/A	0xE	0xEE	6.6%
0x1	0xF	0xFF	0% (opaque)

Premultiplied Alpha and Alpha Output Generation

The source image in ARGB format may have its RGB component already premultiplied with the alpha (AR'G'B') where:

- R' = A*R
- G' = A*G
- B' = A*B

In that case, the processing is as follows:

- Color components of premultiplied layers are multiplied with the global alpha, if Global_Alpha is not equal to 0.
- Color components of the composed underlying layer are multiplied with (1 - A * Global_Alpha).

The premultiplied alpha option is accessible through the DSS0_VID_ATTRIBUTES[28] PREMULTIPLYALPHA register bit of the input pipeline. The following settings are available:

- PREMULTIPLYALPHA bit = 0: Source is not premultiplied with alpha. Full blending is done in the overlay manager.
- PREMULTIPLYALPHA bit = 1: Source is premultiplied with alpha. Partial blending is done.

When the write-back channel copies back to memory the premultiplied color component, the the alpha value is computed as follows:

A(dst) = A(src)+(1-A(src)A(dst))

When the DSS0_WB_ATTRIBUTES[9] ALPHAENABLE register bit is cleared or when the overlay channel is not selected for write-back, the computation of the A(dst) is disabled.

Copyright © 2024 Texas Instruments Incorporated

Note

There is no alpha blending between the background and the next layer (layer-0) above it.

Layer-0 cannot be pre-multiplied, if the overlay output is driving the display, since that layer (or the background color) defines the bottom most layer. In this case, layer-0 alpha is 1 always.

Lyaer-0 can have an alpha value other than 1 when the overlay output is only used to generate an intermediate layer composition result in a memory-to-memory operation (through the WB pipeline). But, if alpha is not 1, the resulting data will be "pre-multiplied". If a non-premultiplied data is required, the resulting overlay output must be fed through a Porter-Duff compositing engine to have the result divided by the overlay alpha output.

12.6.3.10.2.2 Overlay Transparency Color Keys

The overlay manager supports two color transparency modes - source and destination. These modes cannot be active at the same time.

The source transparency is tied to the top most layer, so the layer to have this transparency applied to has to be mapped to the highest Z-ordered input (layer-4) of the overlay manager. The destination transparency is tied to the bottom most layer, so the layer to have this transparency applied to has to be mapped to the lowest Z-ordered input (layer-0) of the overlay manager.

The source and destination transparencies are defined as following:

- Source color transparency: Top layer pixel that meets the source transparency color check is made transparent.
- Destination color transparency: Top layer pixel is made transparent, only if the bottom layer pixel does not meet the destination transparency color check.

The transparency color key is enabled via the DSS0_OVR_CONFIG[10] TCKLCDENABLE register bit.

The minimum transparency color values are specified in the TRANSCOLORKEY field of the DSS0_OVR_TRANS_COLOR_MIN and DSS0_OVR_TRANS_COLOR_MIN2 registers.

The maximum transparency color values are specified in the TRANSCOLORKEY field of the DSS0_OVR_TRANS_COLOR_MAX and DSS0_OVR_TRANS_COLOR_MAX2 registers.

The transparency color key values defined in the registers are compared with the pixel values. If each color component (R, G, and B) is in the range defined by MIN and MAX, then there is a match. If at least one of the color component is not in the range, then there is no match.

- **Source transparency color key**

  The values of the source transparency color key define the range of encoded pixel data considered as a transparent pixel. The encoded pixel values with the source color key value inside the valid range are not visible, and the encoded pixel values of the under layers or solid background color are visible.

  The source transparency color key can be used with YUV and RGB formats (ARGB, RGB, RGBA, xRGB, and RGBx). In that case the A information is ignored for the comparison between the pixel value and the color key value. It is possible to use YUV formats with some care, since the comparison is between the input pixel value of the overlay manager from pipeline and the color key value. The YUV data is converted to RGB format. The user must consider the color space conversion processing in order to define the RGB color key value used for the comparison, in case the original format is YUV.

  The scaler can be enabled as a preprocessor in the video pipelines. The pixel scaling processing can be considered in order to define the color key value to be used after the rescaling for the comparison between the input pixel value to the overlay manager and the color key value.

  The source transparency color key mode is selected by setting the DSS0_OVR_CONFIG[11] TCKLCDSELECTION register bit to 0x1.

  Figure 12-354 shows an example of source transparency color key usage. The pixels with the transparency color key are not displayed. Instead, pixels of the resulting layer underneath are shown.

Copyright © 2024 Texas Instruments Incorporated

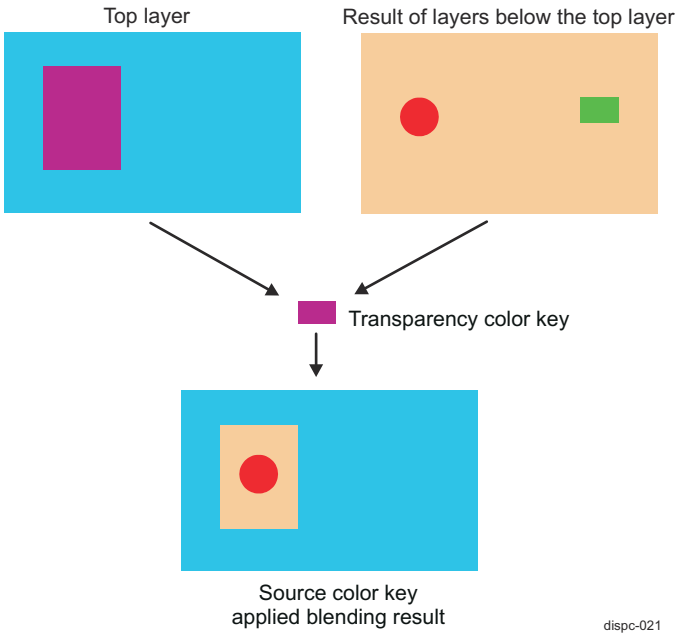


Figure 12-354. DISPC Overlay Source Transparency Color Key Example

- **Destination transparency color key**

  The destination transparency color key values define the range of the encoded pixels in layer-0 (bottom layer), which are not going to be displayed. That is, the encoded pixel values matching the destination color key value range are pixels not visible on the screen. Pixels at the same position in the layers above layer-0 are visible. The destination transparency color key is applicable in the layer in the background only when there is an overlap between the layer in background position and the layer just above it, otherwise, the destination transparency color key is ignored. See Section 12.6.3.10.2, *DISPC Overlay Mechanism*, for details on layer position depending on the Z-order parameter.

  The destination transparency color key can be used only with RGB formats (ARGB, RGB, xRGB, RGBA, and RGBx). In that case the A information is ignored for the comparison between the pixel value and the color key value. It is possible to use YUV formats with some care since the comparison is between the input pixel value of the overlay manager from pipeline and the color key value. The YUV data is converted to RGB format. The user must consider the color space conversion processing in order to define the RGB color key value used for the comparison in case the original format is YUV.

  The scaler can be enabled as a preprocessor in the pipelines. The pixel scaling processing must be considered in order to define the color key value to be used after rescaling for the comparison between the input pixel value to the overlay manager and the color key value.

  The source transparency color key mode is selected by setting the DSS0_OVR_CONFIG[11] TCKLCDSELECTION register bit to 0x0.

  Figure 12-355 shows an example of the destination color key usage. The pixels from layer-0 (bottom layer) equal to the transparency color key are not displayed and are replaced by the pixels from layer-1 (top layer). All other layer-0 pixels, different from the transparency color key, are displayed over layer-1.

Copyright © 2024 Texas Instruments Incorporated

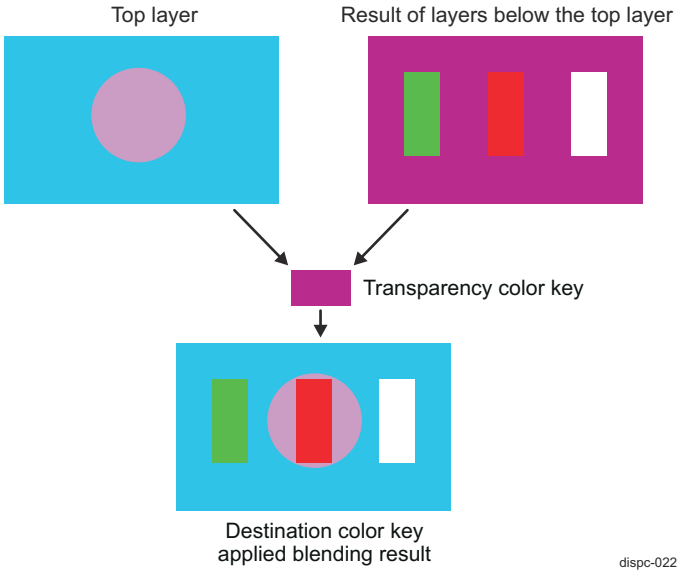


Figure 12-355. DISPC Overlay Destination Transparency Color Key Example

12.6.3.10.3 Overlay 3D Support

In order to support S3D (Stereoscopic 3D) by combining in hardware left and right frames into the same output frame, the following configurations (defined by HDMI 1.4b standard specification) can be used:

- Separate pipeline mode: Left/Right or Top/Bottom. One layer is used for left/top and another layer is used for right/bottom.

12.6.3.10.4 Overlay Color Bar Insertion

Each overlay manager supports a simple internal color bar insertion in each display output path to enable testing of display output interface without using the frame buffer data from the memory.

The colors are: White, Yellow, Cyan, Green, Magenta, Red, Blue, Black, as shown in below. These make three primary colors, three secondary colors, white, and black.

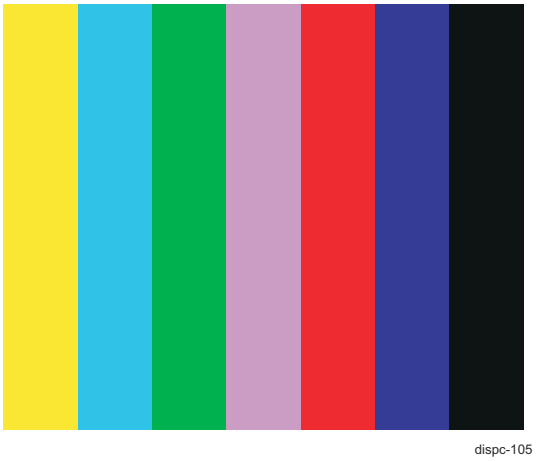


Figure 12-356. DISPC Overlay Internal Color Bar

When the DSS0_OVR_CONFIG[1] COLORBAREN register bit is set to 1, the overlay output data is replaced by the predefined ARGB48 color bar data.

When color bar mode is used, the color space conversion matrix should be appropriately programmed, so that it can convert the full-range RGB to the desired color format.

Copyright © 2024 Texas Instruments Incorporated

Table 12-349. DISPC Overlay Color Bar Table

Color	G	B	R
White	0xFFF	0xFFF	0xFFF
Yellow	0xFFF	0	0xFFF
Cyan	0xFFF	0xFFF	0
Green	0xFFF	0	0
Magenta	0	0xFFF	0xFFF
Red	0	0	0xFFF
Blue	0	0xFFF	0
Black	0	0	0

12.6.3.11 DISPC Video Port Output

The DISPC implements four identical Video Port (VP) outputs:

- VP1 processes data received from Overlay Manager 1 (OVR1)
- VP2 processes data received from Overlay Manager 2 (OVR2)
- VP3 processes data received from Overlay Manager 3 (OVR3)
- VP4 processes data received from Overlay Manager 4 (OVR4)

The VP output path consists of several processing blocks (see Figure 12-357):

- Gamma correction unit
- Color phase rotation (CPR) unit, also used for RGB to YUV color space conversion (CSC)
- Active matrix dithering with TDM (multiple cycle output format)
- BT.656/BT.1120 block
- Timing generator
- Async buffer

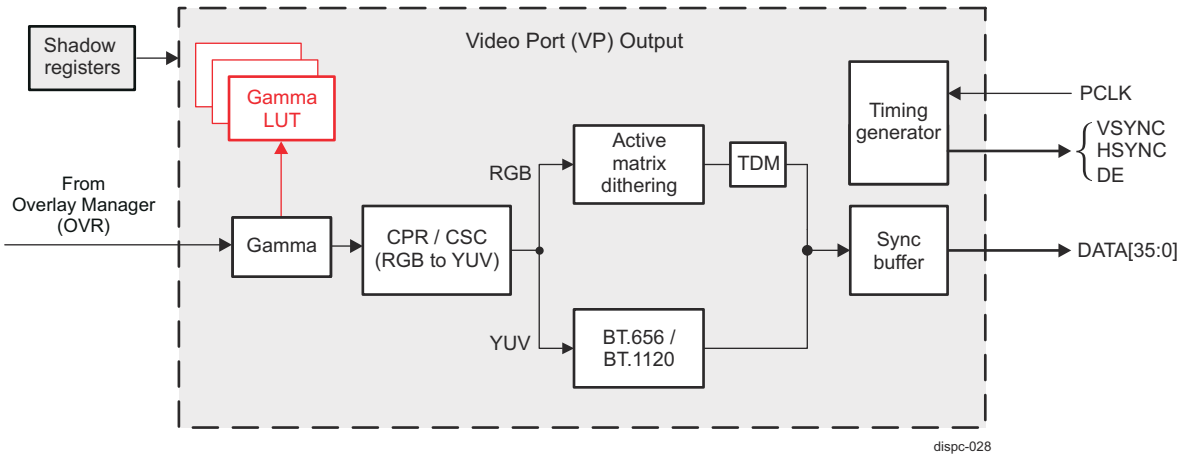


Figure 12-357. DISPC VP Output Architecture

The CSC processing can be configured to occur either before or after the gamma correction in the VP output module via the DSS0_VP_CONFIG[26] COLORCONVPOS register bit. For RGB to YUV color space conversion, the conversion must be done after the gamma correction. For other RGB output applications, the conversion must be done before the final gamma correction.

Note

The DISPC video port (VP) outputs, VP1 through VP4, will be commonly refered to as *video port (VP)* in the following sections.

Copyright © 2024 Texas Instruments Incorporated

12.6.3.11.1 DISPC VP Gamma Correction Unit

The VP supports a look up table to perform the gamma correction on the display output. The look up table consists of 3 separate 1024x10-bit memories, which are indexed by the R/G/B component data.

When performing gamma correction, the selected encoded pixel values from the video pipeline path are sent by the overlay manager to the gamma curve table. Each R/G/B component of the encoded pixel value is used as a pointer to index 1 out of 1024x30-bit gamma curve entries in the table. Each 10-bit component is replaced with the 10-bit table value corresponding to R, G, or B component. The table is loaded by software via the DSS0_VP_GAMMA_TABLE_0 through DSS0_VP_GAMMA_TABLE_15 registers. It is possible to load only part of the table.

The sequence to load the gamma table is:

1. SW writes (only writes are supported) 32-bit gamma correction values using single access, or burst access in linear increment burst mode, into the 64-byte region strating at DSS0_VP_GAMMA_TABLE_0 register address. The LSB 30 bits [29:0] are used for the value, and the MSB 1 bit [31] is used for reseting the index into the table at the end.
2. Loop to step #1, if there is a new access to the gamma table register. The software can access other registers between two accesses to the gamma table register.

Software needs to ensure that there is no visible effect when modifying the table, since it is not under hardware control.

The usage of the gamma table is activated by setting DSS0_VP_CONFIG[2] GAMMAENABLE register bit.

Figure 12-358 describes the format of one of the gamma curve values in the memory.

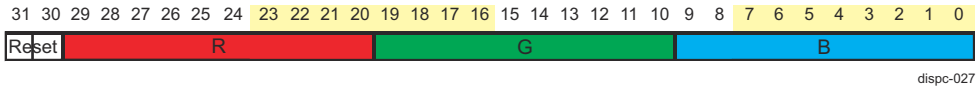


Figure 12-358. DISPC VP Output Gamma Table Write Data Format

12.6.3.11.2 DISPC VP Color Phase Rotation Unit

If required, a color phase rotation (CPR) processing can be applied on the "gamma" data before the spatial/temporal dithering.

The CPR can be selected to correct the non-pure white backlight of the display panel by using a programmable matrix to convert the 36-bit RGB pixel value into a new 36-bit RGB pixel value. The matrix is programmed through a set of nine 11-bit signed coefficients. The output of the calculation is clipped to [0:4095]. The CPR is enabled by setting the DSS0_VP_CONFIG[15] CPR bit to 0x1.

The CPR processing is expressed by the equation shown in Figure 12-359. Table 12-350 lists all coefficients with their respective register bitfields for configuration.

$$\begin{bmatrix} R_{out} \\ G_{out} \\ B_{out} \end{bmatrix} = \frac{1}{256} * \begin{bmatrix} Rr & Rg & Rb \\ Gr & Gg & Gb \\ Br & Bg & Bb \end{bmatrix} * \begin{bmatrix} R_{in} \\ G_{in} \\ B_{in} \end{bmatrix}$$

dispc-023

Figure 12-359. DISPC VP CPR Matrix

Table 12-350. DISPC VP CPR and CSC Coefficients with Associated Register Fields

Register Field	Color Phase Rotation	Color Space Conversion RGB to YUV
DSS0_VP_CSC_COEF0[10-0] C00	RR	YR
DSS0_VP_CSC_COEF0[26-16] C01	RG	YG
DSS0_VP_CSC_COEF1[10-0] C02	RB	YB
DSS0_VP_CSC_COEF1[26-16] C10	GR	CrR

Copyright © 2024 Texas Instruments Incorporated

Table 12-350. DISPC VP CPR and CSC Coefficients with Associated Register Fields (continued)

Register Field	Color Phase Rotation	Color Space Conversion RGB to YUV
DSS0_VP_CSC_COEF2[10-0] C11	GG	CrG
DSS0_VP_CSC_COEF2[26-16] C12	GB	CrB
DSS0_VP_CSC_COEF3[10-0] C20	BR	CbR
DSS0_VP_CSC_COEF3[26-16] C21	BG	CbG
DSS0_VP_CSC_COEF4[10-0] C22	BB	CbB
DSS0_VP_CSC_COEF6[31-19] POSTOFFSET1	-	R offset
DSS0_VP_CSC_COEF7[15-3] POSTOFFSET2	-	G offset
DSS0_VP_CSC_COEF7[31-19] POSTOFFSET3	-	B offset

12.6.3.11.3 DISPC VP Color Space Conversion - RGB to YUV

The RGB to YUV color space conversion (CSC) in the video port is done by using the same programable logic used for CPR. If CPR is also required, then the process can be combined with the CSC processing by adjusting matrix coefficients. Table 12-350 lists the associated coefficients with their respective register bit-fields. The CSC processing is enabled by setting the DSS0_VP_CONFIG[24] COLORCONVENABLE register bit.

The color space conversion can be selected in order to convert from 36-bit RGB to YUV444. The conversion matrix is programmed through a set of nine 11-bit signed coefficients. The result is clipped to [256:3760] for the luminance and [256:3840] for the chrominance, when a full-range YUV output is not desired (equivalent to clipping to [16..235] and [16..240] in 8-bit video). In this case the DSS0_VP_CONFIG[25] FULLRANGE register bit must be set to 0x0.

$$\begin{bmatrix} Y_{OUT} \\ Cr_{OUT} \\ Cb_{OUT} \end{bmatrix} = \frac{1}{256} * \begin{bmatrix} Y_R & Y_G & Y_B \\ Cr_R & Cr_G & Cr_B \\ Cb_R & Cb_G & Cb_B \end{bmatrix} * \begin{bmatrix} R_{IN} \\ G_{IN} \\ B_{IN} \end{bmatrix} + \begin{bmatrix} 256 \\ 2048 \\ 2048 \end{bmatrix}$$

dispc-098

Figure 12-360. DISPC VP CSC RGB to YUV Equation (FULLRANGE=0)

If the programmed active range for the luminance samples (Y) and chrominance samples (Cb and Cr) is [0:4095], then the values Y, Cb, and Cr are clipped to the range [0:4095]. The following equation gives the 11-bit coefficients of the RGB to YUV color space conversion, for full range (when DSS0_VP_CONFIG[25] FULLRANGE = 0x1).

$$\begin{bmatrix} Y_{OUT} \\ Cr_{OUT} \\ Cb_{OUT} \end{bmatrix} = \frac{1}{256} * \begin{bmatrix} Y_R & Y_G & Y_B \\ Cr_R & Cr_G & Cr_B \\ Cb_R & Cb_G & Cb_B \end{bmatrix} * \begin{bmatrix} R_{IN} \\ G_{IN} \\ B_{IN} \end{bmatrix} + \begin{bmatrix} 0 \\ 2048 \\ 2048 \end{bmatrix}$$

dispc-099

Figure 12-361. DISPC VP CSC RGB to YUV Equation (FULLRANGE=1)

In order to provide YUV422 data to the BT.656 / BT.1120 block, the YUV444 format is further converted to YUV422 by sub-sampling the chrominance values. The hardware does this by averaging two-by-two the chrominance samples. The conversion from YUV444 to YUV422 is performed when the color space conversion in the VP is enabled.

Copyright © 2024 Texas Instruments Incorporated

12.6.3.11.4 DISPC VP BT.656 and BT.1120 Modes

Each VP output can be configured in BT.656 or BT.1120 mode. The following standards are not supported in BT.656 mode:

- BT.470 (Support for conventional Analog TV Systems)
- BT.803 (The avoidance of interference generated by digital television studio equipment)
- BT.1364 (Format of ancillary data signals carried in digital component studio interfaces)
- BT.601 (Studio encoding parameters for digital TV standards for standard 4:3 and wide 16:9 aspect ratios)

Unsupported formats when BT.1120 mode is used:

- BT.1364 (Support for Ancillary Data during blanking period)
- BT.709 (Square pixel support)

BT.656/BT.1120 modes use embedded EAV/SAV syncs.

Enabling BT.656 or BT.1120 format for a VP ouput is done by setting DSS0_VP_CONFIG[20] BT656ENABLE or [21] BT1120ENABLE register bit, respectively

Note

It is not possible to enable BT.656 and BT.1120 modes simultaneously on the same VP output.

12.6.3.11.4.1 DISPC BT Mode Blanking

During the transmission of the video signal, the portion of the stream in-between active video data segments is known as the horizontal blanking interval.

Strictly speaking this entire region is the blanking interval, but this interval also includes the EAV and SAV codes. The remaining bytes of information in a digital blanking interval are filled with values corresponding to the blanking levels of the Cb, Y and Cr signals respectively, and in accordance with the standard multiplex sequence for the stream (CbYCrY..). The blanking levels are as follows:

- Cb = 80h
- Y = 10h
- Cr = 80h.

The sequence in the BLANKING region of the data stream is therefore: *80h, 10h, 80h, 10h.....80h, 10h*

For more details on setting the blanking timing values for BT.1120 and/or BT.656 mode, see Section 12.6.3.11.8, *DISPC VP Timing Generator and Display Panel Settings*.

12.6.3.11.4.2 DISPC BT Mode EAV and SAV

The End of Active Video (EAV) and Start of Active Video (SAV) parts of the stream are timing codes. Their function can be summarized as follows:

- EAV – marks the end of the active video data within the current line and therefore also the start of the subsequent line.
- SAV – heralds the start of the active video data within the current line.

These codes are embedded within the BT.656 video data stream, thereby eliminating the need for additional timing signals (HSYNC, VSYNC) to be included as part of the interface.

Both EAV and SAV codes are comprised of a sequence of four bytes (FFh – 00h – 00h - XY). The first three bytes in the sequence constitute a fixed preamble. The fourth byte, contains information about the field being transmitted (Field 1 or Field 2 in an interlaced video signal), the state of field blanking (Vertical) and the state of line blanking (Horizontal). The bit assignment for this byte of the code is shown in Figure 12-362, with the function of each bit described inTable 12-351.

MSB							LSB
1	F	V	H	P3	P2	P1	P0

Figure 12-362. DISPC BT Mode Bit-Assignment for the Fourth Byte of EAV/SAV Codes

Copyright © 2024 Texas Instruments Incorporated

Table 12-351. DISPC BT Mode Bit Function

Bit	Symbol	Function
7	1	Always set to '1'.
6	F	Field bit. 0 – Field 1 1 – Field 2
5	V	Vertical Blanking Status bit. This bit goes High during a vertical field blanking interval, otherwise it remains Low.
4	H	Horizontal Blanking Status bit. 0 – byte is part of SAV code (i.e. stream is entering an active video data region for the current line) 1 – byte is part of EAV code (i.e. stream has entered a horizontal blanking interval - start of a new line)
3	P3	Protection bit 3
2	P2	Protection bit 2
1	P1	Protection bit 1
0	P0	Protection bit 0

The protection bits allow for detection and correction of 1-bit errors and the detection of 2-bit errors. The status of P3, P2, P1 and P0 depend on the states of bits F, V, and H. This dependency is shown in Table 12-352.

Table 12-352. DISPC BT Mode Status of Protection Bits in Function of F, V, and H

F	V	H	P3	P2	P1	P0
0	0	0	0	0	0	0
0	0	1	1	1	0	1
0	1	0	1	0	1	1
0	1	1	0	1	1	0
1	0	0	0	1	1	1
1	0	1	1	0	1	0
1	1	0	1	1	0	0
1	1	1	0	0	0	1

12.6.3.11.5 DISPC VP Spatial/Temporal Dithering

The active spatial/temporal dithering logic can be enabled to minimize the color banding when displaying the data on a LCD panel with color depth lower than 24-bit. The dithering logic is integrated after the color/gamma conversion blocks and before the TDM (Time Division Multiplexing) block. The spatial/temporal dithering algorithm is based on the (x,y) pixel position and frame rate control (the value of removed bits and the frame number). The dithering logic can process the pixels over one frame, two frames, or four frames. The number of frames is selected by setting the DSS0_VP_CONTROL[31-30] SPATIALTEMPORALDITHERINGFRAMES register field. In the case of a single frame, only spatial processing is applied. In case of multiple frames, both spatial and temporal processing are applied to the pixels. The number of frame is initialized before enabling the spatial/temporal dithering logic. It must be never changed by the software while the spatial/temporal logic is enabled. The spatial/temporal dithering logic is enabled by setting the DSS0_VP_CONTROL[7] STDITHERENABLE bit to 0x1.

Note

- If the interface data bus is smaller than the pixel format size and the spatial/temporal dithering is not enabled, the MSBs of the pixel color components are output on the interface data bus.
- If the interface data bus is larger than the pixel format size, then by programming the pixel components replication active/inactive the MSB is replicated to the LSB of the interface data bus.

12.6.3.11.6 DISPC VP Multiple Cycle Output Format (TDM)

The pixels (only RGB compontents) after the active matrix dithering unit are formatted on one or multiple cycles (from 1 to 3 cycles). On three cycles, two pixels can concatenate and send to the panel. The cycle format is

Copyright © 2024 Texas Instruments Incorporated

selected through the DSS0_VP_CONTROL[24-23] TDMCYCLEFORMAT bit field. The number of bits for each cycle is set in the DSS0_VP_DATA_CYCLE_0 register for the first cycle, the DSS0_VP_DATA_CYCLE1 register for the second cycle, and the DSS0_VP_DATA_CYCLE2 register for the third cycle. The output interface data bus width, when TDM mode is enabled (DSS0_VP_CONTROL[20] TDMENABLE register bit = 1), can be 8, 9, 12, or 16 bits, configurable through the DSS0_VP_CONTROL[22-21] TDMPARALLELMODE register field.

When the TDM is disabled (DSS0_VP_CONTROL[20] TDMENABLE = 0), the video port output interface data bus width is configured through the DSS0_VP_CONTROL[10-8] DATALINES register field.

When using TDM mode, only up to 24 bits per pixel can be output on the interface. For higher color depth, only the upper bits are kept before converting each pixel into TDM output.

Figure 12-363 through Figure 12-366 show various examples of TDM settings in the function of pixel data formats and the interface data bus width.

	24-bpp		
	1st cycle	2nd cycle	3rd cycle
Data[7]	R0[7]	G0[7]	B0[7]
Data[6]	R0[6]	G0[6]	B0[6]
Data[5]	R0[5]	G0[5]	B0[5]
Data[4]	R0[4]	G0[4]	B0[4]
Data[3]	R0[3]	G0[3]	B0[3]
Data[2]	R0[2]	G0[2]	B0[2]
Data[1]	R0[1]	G0[1]	B0[1]
Data[0]	R0[0]	G0[0]	B0[0]

CONTROL[24-23] TDMCYCLEFORMAT = 0x2
DATA_CYCLE0 = 0x00000008
DATA_CYCLE1 = 0x00000008
DATA_CYCLE2 = 0x00000008

	18-bpp		
	1st cycle	2nd cycle	3rd cycle
Data[7]	R0[5]	G0[3]	x
Data[6]	R0[4]	G0[2]	x
Data[5]	R0[3]	G0[1]	x
Data[4]	R0[2]	G0[0]	x
Data[3]	R0[1]	B0[5]	x
Data[2]	R0[0]	B0[4]	x
Data[1]	G0[5]	B0[3]	B0[1]
Data[0]	G0[4]	B0[2]	B0[0]

CONTROL[24-23] TDMCYCLEFORMAT = 0x2
DATA_CYCLE0 = 0x00000008
DATA_CYCLE1 = 0x00000008
DATA_CYCLE2 = 0x00000002

	16-bpp	
	1st cycle	2nd cycle
Data[7]	R0[4]	G0[2]
Data[6]	R0[3]	G0[1]
Data[5]	R0[2]	G0[0]
Data[4]	R0[1]	B0[4]
Data[3]	R0[0]	B0[3]
Data[2]	G0[5]	B0[2]
Data[1]	G0[4]	B0[1]
Data[0]	G0[3]	B0[0]

CONTROL[24-23] TDMCYCLEFORMAT = 0x1
DATA_CYCLE0 = 0x00000008
DATA_CYCLE1 = 0x00000008

	12-bpp	
	1st cycle	2nd cycle
Data[7]	R0[3]	x
Data[6]	R0[2]	x
Data[5]	R0[1]	x
Data[4]	R0[0]	x
Data[3]	G0[3]	B0[3]
Data[2]	G0[2]	B0[2]
Data[1]	G0[1]	B0[1]
Data[0]	G0[0]	B0[0]

CONTROL[24-23] TDMCYCLEFORMAT = 0x1
DATA_CYCLE0 = 0x00000008
DATA_CYCLE1 = 0x00000004

dispc-057

Figure 12-363. DISPC VP TDM 8-Bit Interface Settings

Copyright © 2024 Texas Instruments Incorporated

	24-bpp		
	1st cycle	2nd cycle	3rd cycle
Data[8]	R0[7]	G0[6]	x
Data[7]	R0[6]	G0[5]	x
Data[6]	R0[5]	G0[4]	x
Data[5]	R0[4]	G0[3]	B0[5]
Data[4]	R0[3]	G0[2]	B0[4]
Data[3]	R0[2]	G0[1]	B0[3]
Data[2]	R0[1]	G0[0]	B0[2]
Data[1]	R0[0]	B0[7]	B0[1]
Data[0]	G0[7]	B0[6]	B0[0]

CONTROL[24-23] TDMCYCLEFORMAT = 0x2
DATA_CYCLE0 = 0x00000009
DATA_CYCLE1 = 0x00000009
DATA_CYCLE2 = 0x00000006

	18-bpp	
	1st cycle	2nd cycle
Data[8]	R0[5]	G0[2]
Data[7]	R0[4]	G0[1]
Data[6]	R0[3]	G0[0]
Data[5]	R0[2]	B0[5]
Data[4]	R0[1]	B0[4]
Data[3]	R0[0]	B0[3]
Data[2]	G0[5]	B0[2]
Data[1]	G0[4]	B0[1]
Data[0]	G0[3]	B0[0]

CONTROL[24-23] TDMCYCLEFORMAT = 0x1
DATA_CYCLE0 = 0x00000009
DATA_CYCLE1 = 0x00000009

	16-bpp	
	1st cycle	2nd cycle
Data[8]	R0[4]	x
Data[7]	R0[3]	x
Data[6]	R0[2]	G0[1]
Data[5]	R0[1]	G0[0]
Data[4]	R0[0]	B0[4]
Data[3]	G0[5]	B0[3]
Data[2]	G0[4]	B0[2]
Data[1]	G0[3]	B0[1]
Data[0]	G0[2]	B0[0]

CONTROL[24-23] TDMCYCLEFORMAT = 0x1
DATA_CYCLE0 = 0x00000009
DATA_CYCLE1 = 0x00000007

	12-bpp	
	1st cycle	2nd cycle
Data[8]	R0[3]	x
Data[7]	R0[2]	x
Data[6]	R0[1]	x
Data[5]	R0[0]	x
Data[4]	G0[3]	x
Data[3]	G0[2]	x
Data[2]	G0[1]	B0[2]
Data[1]	G0[0]	B0[1]
Data[0]	B0[3]	B0[0]

CONTROL[24-23] TDMCYCLEFORMAT = 0x1
DATA_CYCLE0 = 0x00000009
DATA_CYCLE1 = 0x00000003

dispc-058

Figure 12-364. DISPC VP TDM 9-Bit Interface Settings

Copyright © 2024 Texas Instruments Incorporated

	24-bpp	
	1st cycle	2nd cycle
Data[11]	R0[7]	G0[3]
Data[10]	R0[6]	G0[2]
Data[9]	R0[5]	G0[1]
Data[8]	R0[4]	G0[0]
Dat a[7]	R0[3]	B0[7]
Data[6]	R0[2]	B0[6]
Data[5]	R0[1]	B0[5]
Data[4]	R0[0]	B0[4]
Data[3]	G0[7]	B0[3]
Data[2]	G0[6]	B0[2]
Data[1]	G0[5]	B0[1]
Data[0]	G0[4]	B0[0]

CONTROL[24-23] TDMCYCLEFORMAT = 0x1
DATA_CYCLE0 = 0x0000000C
DATA_CYCLE1 = 0x0000000C

	18-bpp		
	1st cycle	2nd cycle	3rd cycle
Data[11]	R0[5]	B0[5]	G1[5]
Data[10]	R0[4]	B0[4]	G1[4]
Data[9]	R0[3]	B0[3]	G1[3]
Data[8]	R0[2]	B0[2]	G1[2]
Data[7]	R0[1]	B0[1]	G1[1]
Data[6]	R0[0]	B0[0]	G1[0]
Data[5]	G0[5]	R1[5]	B1[5]
Data[4]	G0[4]	R1[4]	B1[4]
Data[3]	G0[3]	R1[3]	B1[3]
Data[2]	G0[2]	R1[2]	B1[2]
Data[1]	G0[1]	R1[1]	B1[1]
Data[0]	G0[0]	R1[0]	B1[0]

CONTROL[24-23] TDMCYCLEFORMAT = 0x3
DATA_CYCLE0 = 0x0000000C
DATA_CYCLE1 = 0x00060606
DATA_CYCLE2 = 0x000C0000

	16-bpp	
	1st cycle	2nd cycle
Data[11]	R0[4]	x
Data[10]	R0[3]	x
Data[9]	R0[2]	x
Data[8]	R0[1]	x
Data[7]	R0[0]	x
Data[6]	G0[5]	x
Data[5]	G0[4]	x
Data[4]	G0[3]	x
Data[3]	G0[2]	B0[3]
Data[2]	G0[1]	B0[2]
Data[1]	G0[0]	B0[1]
Data[0]	B0[4]	B0[0]

CONTROL[24-23] TDMCYCLEFORMAT = 0x1
DATA_CYCLE0 = 0x0000000C
DATA_CYCLE1 = 0x00000004

	12-bpp
	1st cycle
Data[11]	R0[3]
Data[10]	R0[2]
Data[9]	R0[1]
Data[8]	R0[0]
Data[7]	G0[3]
Data[6]	G0[2]
Data[5]	G0[1]
Data[4]	G0[0]
Data[3]	B0[3]
Data[2]	B0[2]
Data[1]	B0[1]
Data[0]	B0[0]

CONTROL[24-23] TDMCYCLEFORMAT = 0x0
DATA_CYCLE0 = 0x0000000C

dispc-059

Figure 12-365. DISPC VP TDM 12-Bit Interface Settings

Copyright © 2024 Texas Instruments Incorporated

	24-bpp		
	1st cycle	2nd cycle	3rd cycle
Data[15]	R0[7]	B0[7]	G1[7]
Data[14]	R0[6]	B0[6]	G1[6]
Data[13]	R0[5]	B0[5]	G1[5]
Data[12]	R0[4]	B0[4]	G1[4]
Data[11]	R0[3]	B0[3]	G1[3]
Data[10]	R0[2]	B0[2]	G1[2]
Data[9]	R0[1]	B0[1]	G1[1]
Data[8]	R0[0]	B0[0]	G1[0]
Data[7]	G0[7]	R1[7]	B1[7]
Data[6]	G0[6]	R1[6]	B1[6]
Data[5]	G0[5]	R1[5]	B1[5]
Data[4]	G0[4]	R1[4]	B1[4]
Data[3]	G0[3]	R1[3]	B1[3]
Data[2]	G0[2]	R1[2]	B1[2]
Data[1]	G0[1]	R1[1]	B1[1]
Data[0]	G0[0]	R1[0]	B1[0]

CONTROL[24-23] TDMCYCLEFORMAT = 0x3
DATA_CYCLE0 = 0x00000010
DATA_CYCLE1 = 0x00080808
DATA_CYCLE2 = 0x00100000

	18-bpp	
	1st cycle	2nd cycle
Data[15]	R0[5]	x
Data[14]	R0[4]	x
Data[13]	R0[3]	x
Data[12]	R0[2]	x
Data[11]	R0[1]	x
Data[10]	R0[0]	x
Data[9]	G0[5]	x
Data[8]	G0[4]	x
Data[7]	G0[3]	X
Data[6]	G0[2]	x
Data[5]	G0[1]	x
Data[4]	G0[0]	x
Data[3]	B0[5]	x
Data[2]	B0[4]	x
Data[1]	B0[3]	B0[1]
Data[0]	B0[2]	B0[0]

CONTROL[24-23] TDMCYCLEFORMAT = 0x1
DATA_CYCLE0 = 0x00000010
DATA_CYCLE1 = 0x00000002

	16-bpp
	1st cycle
Data[15]	R0[4]
Data[14]	R0[3]
Data[13]	R0[2]
Data[12]	R0[1]
Data[11]	R0[0]
Data[10]	G0[5]
Data[9]	G0[4]
Data[8]	G0[3]
Data[7]	G0[2]
Data[6]	G0[1]
Data[5]	G0[0]
Data[4]	B0[4]
Data[3]	B0[3]
Data[2]	B0[2]
Data[1]	B0[1]
Data[0]	B0[0]

CONTROL[24-23] TDMCYCLEFORMAT = 0x0
DATA_CYCLE0 = 0x00000010

	12-bpp
	1st cycle
Data[15]	x
Data[14]	x
Data[13]	x
Data[12]	x
Data[11]	R0[3]
Data[10]	R0[2]
Data[9]	R0[1]
Data[8]	R0[0]
Data[7]	G0[3]
Data[6]	G0[2]
Data[5]	G0[1]
Data[4]	G0[0]
Data[3]	B0[3]
Data[2]	B0[2]
Data[1]	B0[1]
Data[0]	B0[0]

CONTROL[24-23] TDMCYCLEFORMAT = 0x0
DATA_CYCLE0 = 0x0000000C

dispc-060

Figure 12-366. DISPC VP TDM 16-Bit Interface Settings

12.6.3.11.7 DISPC VP Stall Mode

The VP output can be programmed to operate in Stall Mode, where there is a stall handshake mechanism (back-pressure) with the display peripheral to which the VP is connected. No sync signals (HS, VS) are generated in Stall Mode of operation. Data and Enable (DE) signals are provided only when the stall signal is de-asserted (1'b0) by the peripheral. The Stall Mode can be enabled via the DSS0_VP_CONTROL[11] STALLMODE register bit.

Copyright © 2024 Texas Instruments Incorporated

Note

In this SoC, the Stall Mode of operation is supported only on the VP outputs connected to the DSI peripheral module.

There are two types of data transfer, if Stall Mode is enabled, that can be selected by the DSS0_VP_CONTROL[12] STALLMODETYPE register bit:

- In DSI Video Mode, multiple frames are sent out on VP until the VP is disabled. A FRAMEDONE_IRQ interrupt is generated at the end of each frame. No VSYNC_IRQ interrupt is generated.
- In DSI Command Mode, a single frame is sent out on VP and the VP is auto disabled. A FRAMEDONE_IRQ interrupt is generated at the end of frame. To transfer another frame in Command Mode, the user needs to re-enable the VP. No VSYNC_IRQ interrupt is generated.

12.6.3.11.8 DISPC VP Timing Generator and Display Panel Settings

Each video port output has a dedicated timing generator supporting progressive and interlaced modes. It is clocked using the pixel clock and is configured to generate the video data and sync signals to match the desired video display standard timings.

Two-level configuration for enabling the video ports exists:

- Via the idividual DSS0_VP_CONTROL[0] ENABLE register bit for each VP. It allows each VP to be independently controlled by two different hosts.
- Via the common DSS0_COMMON_DISPC_GLOBAL_OUTPUT_ENABLE[2-0] VP_ENABLE register field. It allows two or more VP timing generators to be in sync by enabling them simultaneously with a single register write.

Both ENABLE and VP_ENABLE register fields must be set in order for a VP (timing generator) to start.

Figure 12-367 shows the timing generator display parameters.

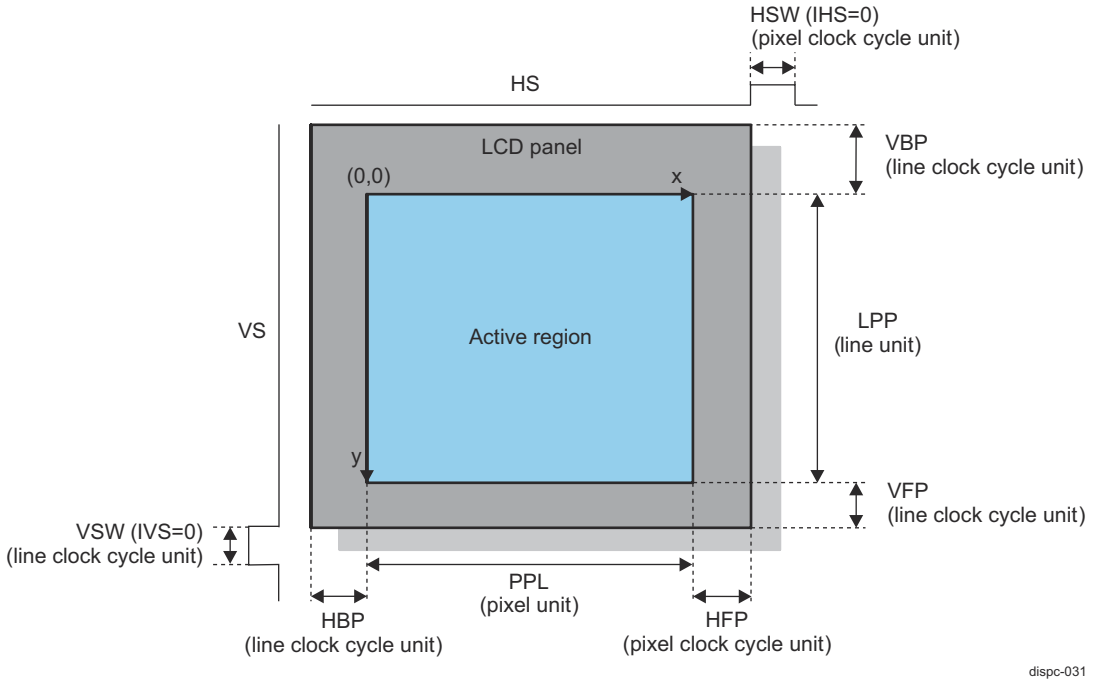


Figure 12-367. DISPC VP Display Timing Parameters

The width of VP output data bus is configured in the DSS0_VP_CONTROL[10-8] DATALINES register field, when TDM feature is disabled. When TDM is enabled, the width of the output data bus is defined according to Section 12.6.3.11.6, *DISPC VP Multiple Cycle Output Format (TDM)*.

The size of the display panel is defined by:

Copyright © 2024 Texas Instruments Incorporated

- Number of lines per frame, DSS0_VP_SIZE_SCREEN[29-16] LPP bit field
- Number of pixels per line, DSS0_VP_SIZE_SCREEN[13-0] PPL bit field

Standard HSYNC/VSYNC timing generation is programmable as follows:

- Horizontal front porch is set in the DSS0_VP_TIMING_H[19-8] HFP bit field.
- Horizontal back porch is set in the DSS0_VP_TIMING_H[31-20] HBP bit field.
- Horizontal synchronization pulse width is set in the DSS0_VP_TIMING_H[7-0] HSW bit field.
- Vertical front porch is set in the DSS0_VP_TIMING_V[19-8] VFP bit field.
- Vertical back porch is set in the DSS0_VP_TIMING_V[31-20] VBP bit field.
- Vertical synchronization pulse width is set in the DSS0_VP_TIMING_V[7-0] VSW bit field.

When the output is in BT.1120 or BT.656 mode, the following timing constants are mapped onto the DSS0_VP_TIMING_H and DSS0_VP_TIMING_V registers:

- Progressive mode:
  - Horizontal blanking (12 bits) is set in the {DSS0_VP_TIMING_V[3-0] VSW, DSS0_VP_TIMING_H[7-0] HSW} register fields (up to 2048 bytes of horizontal blanking supported).
  - Vertical frame blanking No 1 is set in the DSS0_VP_TIMING_V[19-8] VFP bit field.
  - Vertical frame blanking No 2 is set in the DSS0_VP_TIMING_V[31-20] VBP bit field.
  - Number of lines per frame is set in the DSS0_VP_SIZE_SCREEN[29-16] LPP bit field.
  - Number of pixels per line is set in the DSS0_VP_SIZE_SCREEN[13-0] PPL bit field.
- Interlaced mode:
  - Horizontal blanking (12 bits) is set in the {DSS0_VP_TIMING_V[3-0] VSW, DSS0_VP_TIMING_H[7-0] HSW} register fields (up to 2048 bytes of horizontal blanking supported).
  - Vertical field blanking No.1 for Even Field is set in the DSS0_VP_TIMING_H[19-8] HFP bit field.
  - Vertical field blanking No.2 for Even Field is set in the DSS0_VP_TIMING_H[31-20] HBP bit field.
  - Vertical field blanking No.1 for Odd Field is set in the DSS0_VP_TIMING_V[19-8] VFP bit field.
  - Vertical field blanking No.2 for Odd Field is set in the DSS0_VP_TIMING_V[31-20] VBP bit field.
  - Number of lines per field (even) is set in the DSS0_VP_SIZE_SCREEN[29-16] LPP bit field.
  - Delta number of odd field compared to even field (in a single line) is set in the DSS0_VP_SIZE_SCREEN[15-14] DELTA_LPP bit field. The DELTA_LPP field only controls the output channel and not the size of the field fetched from the frame buffer in memory. This field must be set to zero for YUV420 format.
  - Number of pixels per line is set in the DSS0_VP_SIZE_SCREEN[13-0] PPL bit field.

Horizontal/vertical synchronization and output enable signals polarity are programmable by setting the DSS0_VP_POL_FREQ[12] IVS, [13] IHS, and [15] IEO register bits. These signals can be gated by setting the DSS0_VP_CONFIG[7] VSYNCGATED and [6] HSYNCGATED register bits. In addition, the alignment between VSYNC and HSYNC signals can be programmed via the DSS0_VP_POL_FREQ[18] ALIGN register bit.

The latch of data can be driven on the rising or falling edge of the pixel clock by setting the DSS0_VP_POL_FREQ[14] IPC register bit. The drive of the SYNC and VSYNC signals in the function of the pixel clock is done by setting the DSS0_VP_POL_FREQ[16] RF bit.

Each VP output can be configured in progressive output mode or interlaced output mode. The selection is done by writing into the bit-field DSS0_VP_CONFIG[22] OUTPUTMODEENABLE register bit. The default setting is for progressive mode. The selection can be changed only when the corresponding VP output is disabled. The configuration is independent for each VP output. It is possible to change the configuration of one of the VP outputs while the other VP is enabled.

The pixel clock for each VP output can be gated by setting the DSS0_VP_CONFIG[5] PIXELCLOCKGATED register bit.

The hold time of the pixels on the data bus is determined in clock cycles by the DSS0_VP_CONTROL[16-14] HT register field.

12.6.3.11.9 DISPC VP Merge-Split-Sync (MSS) Module

In order to interface with peripherals which can support a different data rate than the VP output, a Merge-Split-Sync (MSS) module is introduced between the VP outputs and DISPC final output. When interfacing with a peripheral which can support a higher data rate than the VP, two VP outputs can be merged to generate one

Copyright © 2024 Texas Instruments Incorporated

high data rate stream on the DISPC output (MERGE mode). When interfacing with a peripheral which requires dual physical links to support a high resolution video, one VP output can be split into two half data rate pixel streams on two separate DISPC outputs (SPLIT mode).

Both left/right (L/R) or odd/even (O/E) pixel selection types are supported.

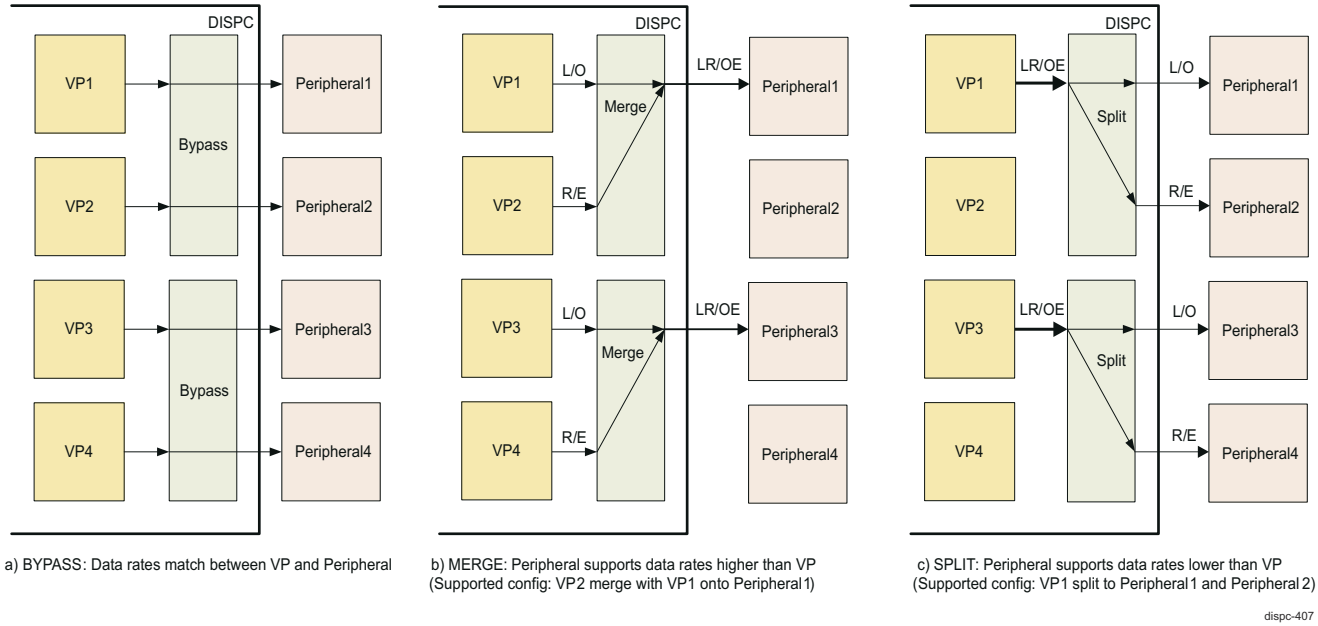


Figure 12-368. DISPC VP Merge-Split Scheme

Additionally, the MSS module can also operate in a SYNC mode where two VP outputs get the same source pixel clock. The SYNC mode only affects the clock muxing and does not manipulate the data going out of each VP output.

Only the following configurations, as shown in Figure 12-368, are supported:

- MERGE Mode
  - VP2 (Right/Even stream) merged with VP1 (Left/Odd stream) output onto Peripheral #1 (P1)
  - VP4 (Right/Even stream) merged with VP3 (Left/Odd stream) output onto Peripheral #3 (P3)
- SPLIT Mode
  - VP1 split into Peripheral #1 (P1)(Left/Odd stream) and Peripheral #2 (P2)(Right/Even stream)
  - VP3 split into Peripheral #3 (P3)(Left/Odd stream) and Peripheral #4 (P4)(Right/Even stream)
- SYNC Mode
  - VP1 and VP2 in sync and get the same pixel clock
  - VP3 and VP4 in sync and get the same pixel clock

The DSS0_COMMON_DISPC_MSS_VP1 and DSS0_COMMON_DISPC_MSS_VP3 registers control the MSS operation.

The MSS line buffers are sized to support up to 6K RGB30 video data. Therefore, in MERGE use-case, two 3K VP outputs can be merged to create a 6K output at DISPC. In SPLIT use-case, one 6K output can be split into two 3K outputs.

In both, MERGE and SPLIT use-cases, all the control signals (VS, HS, DE) are re-generated so as to match with the timing of the new merged frame or split frames.

Copyright © 2024 Texas Instruments Incorporated

Note

For Merge-Split-Sync mode of operation, the VP outputs need to be programmed to be active-high for the different signals (DE, VS, HS). For SPLIT mode of operation, it is required that the total number of pixels in a line is a multiple of 2. For both SPLIT mode and SYNC mode of operation, both the VP blocks need to be enabled at the same cycle. This is achieved by making a single write to the global enable register (DSS0_COMMON_DISPC_GLOBAL_OUTPUT_ENABLE[3-0] VP_ENABLE field).

When the DISPC is interfaced with the DP/eDP peripheral, the SPLIT mode corresponds to the Split (single stream transport) mode of operation of the DSC (Digital Stream Compression) sub-module within the EDP module. In this case, the two streams at half the pixel clock rate are first provided to DSC_ENC0 and DSC_ENC1 within the EDP. Then, the combined encoded stream is provided to the EDP MHDPTX Controller VIF0 port. The SYNC mode corresponds to the Dual (multi stream transport) mode of operation of the EDP module. In this case, the two streams at full pixel clock rate are provided to the EDP VIF0 and VIF1 ports (the EDP DSC has to be bypassed in this mode). For more details, see the EDP Section 12.6.5.2.1, *Video Stream Clock/Data Muxing*.

12.6.3.11.9.1 MSS Clocking Scheme

The pixel clocks going to the VPn as well as the pixel clocks going to the peripherals (Pn) will be different under different cases, depending on whether the output is in BYPASS, SYNC, MERGE or SPLIT mode. The different combinations of the clocks are as shown in Figure 12-369 and Figure 12-370.

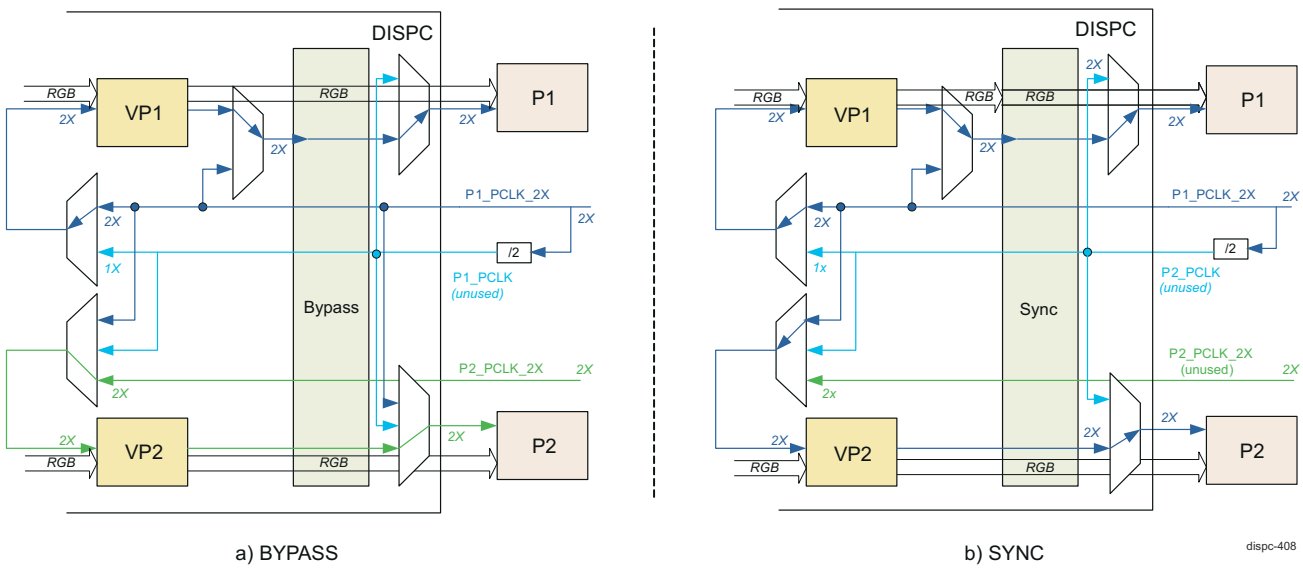


Figure 12-369. DISPC VP Clocking Scheme for MSS BYPASS and SYNC Modes

Copyright © 2024 Texas Instruments Incorporated

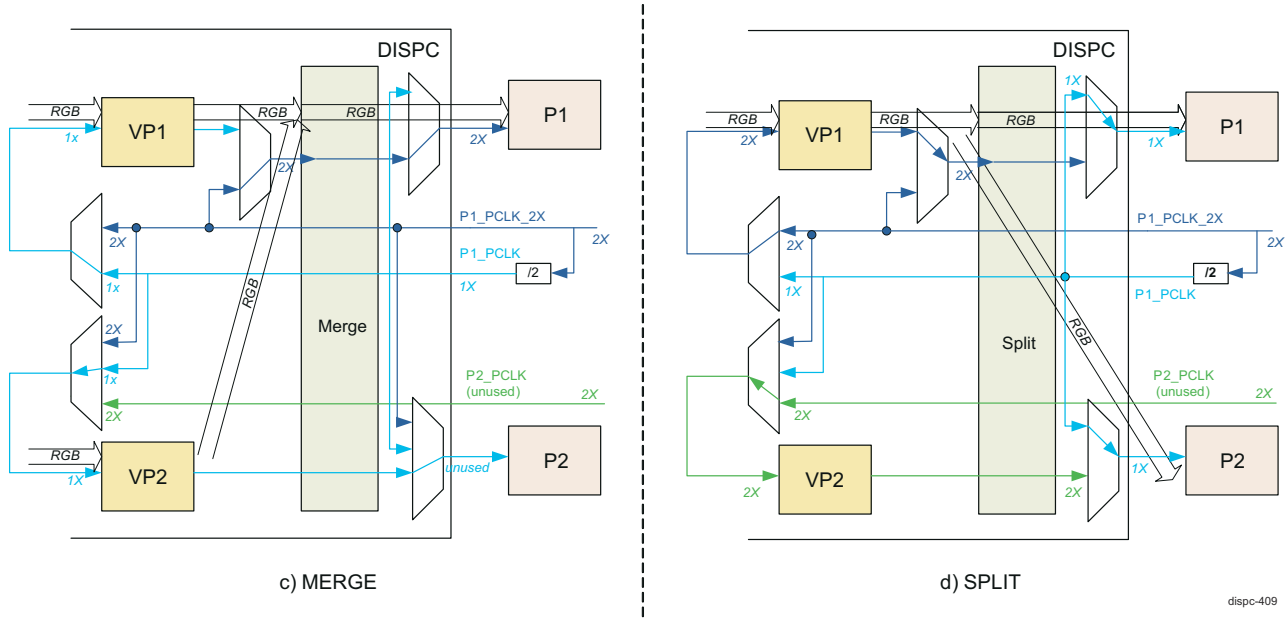


Figure 12-370. DISPC VP Clocking Scheme for MSS MERGE and SPLIT Modes

Figure 12-370 show VP1 and VP2. Same clock scheme is applicable to VP3 and VP4.

For all cases in Figure 12-369 and Figure 12-370 there is no async interaction between P1_PCLKs and P2_PCLKs.

12.6.3.11.9.2 MSS Merge with Scaling

Merge operation with scaling enabled in the VID pipelines can result in visual artifacts at the merged boundary. The artifacts can be avoided by scaling each half to a larger window size and then using the crop feature of the VID pipelines to get to the required size for each half.

12.6.3.12 DISPC Internal Diagnostic Features

The DSS supports the following internal diagnostic features in DISPC hardware:

- Data correctness check: To verify intended data is shown correctly on the display.
- Freeze frame detection: To notify a possible frame freeze, when there is no change in the display frame over a multiple frame periods.

These features are implemented by collecting signatures from user-defined regions within each pipeline output frame and the final display output frame, and comparing them to reference signatures provided by the user (software) and/or to previously saved signatures. The signature from each region is generated by using a MISR (Multiple Input Signature Register) module.

Copyright © 2024 Texas Instruments Incorporated

12.6.3.12.1 Internal Diagnostic Check Regions

The DSS supports the following internal diagnostic check regions:

- Video pipelines: One internal diagnostic check region at the output of each video pipeline.
- Video port outputs: Up to eight internal diagnostic sub-regions within the active video output area of the final display output of each video port.

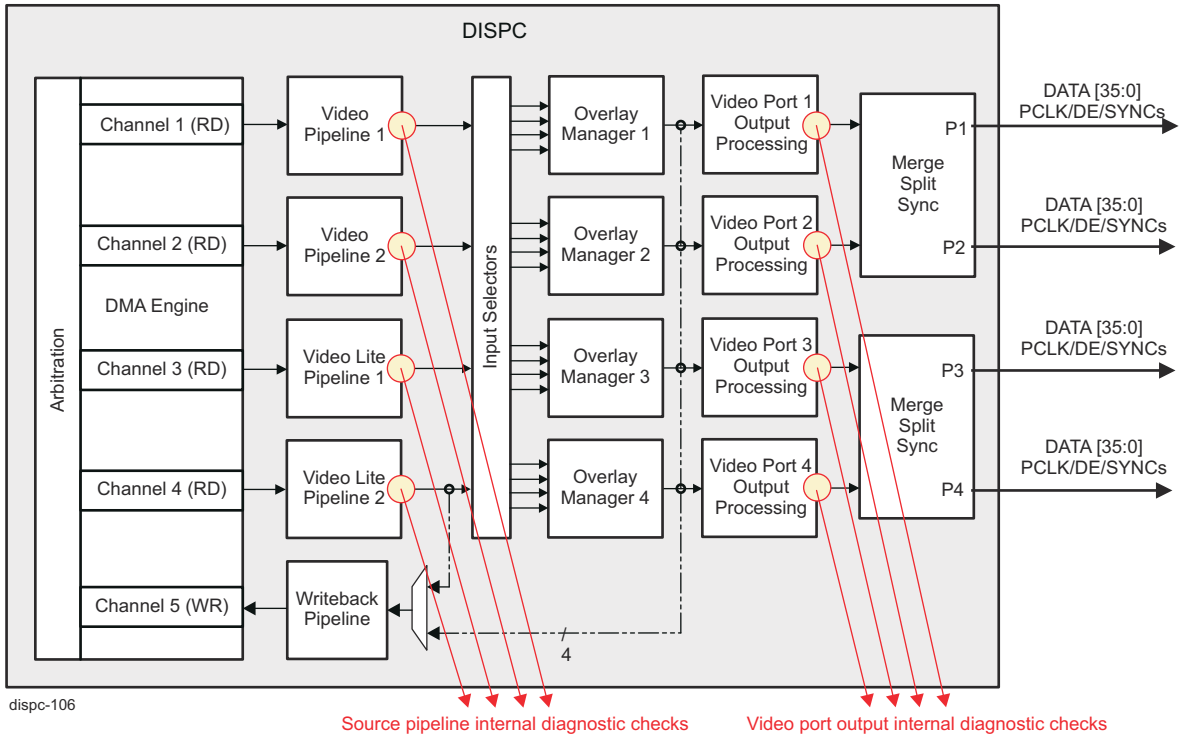


Figure 12-371. DISPC Internal Diagnostic Check Regions

Table 12-353 lists the parameters supported by each of the internal diagnostic check regions, together with the associated register control fields.

Table 12-353. DISPC Internal Diagnostic Check Regions Parameters

Internal Diagnostic Region Parameter	Video Pipeline Control Register	Video Port Sub-region m (m = 0 to 7) Control Register
X position	DSS0_VID_SAFETY_POSITION[11-0] POSX	DSS0_VP_SAFETY_POSITION_m[11-0] POSX
Y position	DSS0_VID_SAFETY_POSITION[27-16] POSY	DSS0_VP_SAFETY_POSITION_m[27-16] POSY
Width	DSS0_VID_SAFETY_SIZE[11-0] SIZEX	DSS0_VP_SAFETY_SIZE_m[11-0] SIZEX
Height	DSS0_VID_SAFETY_SIZE[27-16] SIZEY	DSS0_VP_SAFETY_SIZE_m[27-16] SIZEY

Copyright © 2024 Texas Instruments Incorporated

Table 12-353. DISPC Internal Diagnostic Check Regions Parameters (continued)

Internal Diagnostic Region Parameter	Video Pipeline Control Register	Video Port Sub-region m (m = 0 to 7) Control Register
Data Correctness Check Mode Enable	DSS0_VID_SAFETY_ATTRIBUTES[1] CAPTUREMODE	DSS0_VP_SAFETY_ATTRIBUTES_m[1] CAPTUREMODE
Freeze Frame Detection Mode Enable	DSS0_VID_SAFETY_ATTRIBUTES[1] CAPTUREMODE	DSS0_VP_SAFETY_ATTRIBUTES_m[1] CAPTUREMODE
Region Internal Diagnostic Check Enable	DSS0_VID_SAFETY_ATTRIBUTES[0] ENABLE	DSS0_VP_SAFETY_ATTRIBUTES_m[0] ENABLE
Freeze Frame Detection Threshold	DSS0_VID_SAFETY_ATTRIBUTES[10-3] THRESHOLD	DSS0_VP_SAFETY_ATTRIBUTES_m[10-3] THRESHOLD
Frames to Skip	DSS0_VID_SAFETY_ATTRIBUTES[12-11] FRAMESKIP	DSS0_VP_SAFETY_ATTRIBUTES_m[12-11] FRAMESKIP
Reference Signature	DSS0_VID_SAFETY_REF_SIGNATURE[31-0] SIGNATURE	DSS0_VP_SAFETY_REF_SIGNATURE_m[31-0] SIGNATURE
MISR Seed	DSS0_VID_SAFETY_LFSR_SEED[31-0] SEED	DSS0_VP_SAFETY_LFSR_SEED[31-0] SEED
MISR Seed Selection	DSS0_VID_SAFETY_ATTRIBUTES[2] SEEDSELECT	DSS0_VP_SAFETY_ATTRIBUTES_m[2] SEEDSELECT
MISR Captured Signature	DSS0_VID_SAFETY_CAPT_SIGNATURE[31-0] SIGNATURE	DSS0_VP_SAFETY_CAPT_SIGNATURE_m[31-0] SIGNATURE

Copyright © 2024 Texas Instruments Incorporated

Figure 12-372 shows examples of one internal diagnostic region in the video pipeline output stage and up to 8 possible regions in the final video port output stage.

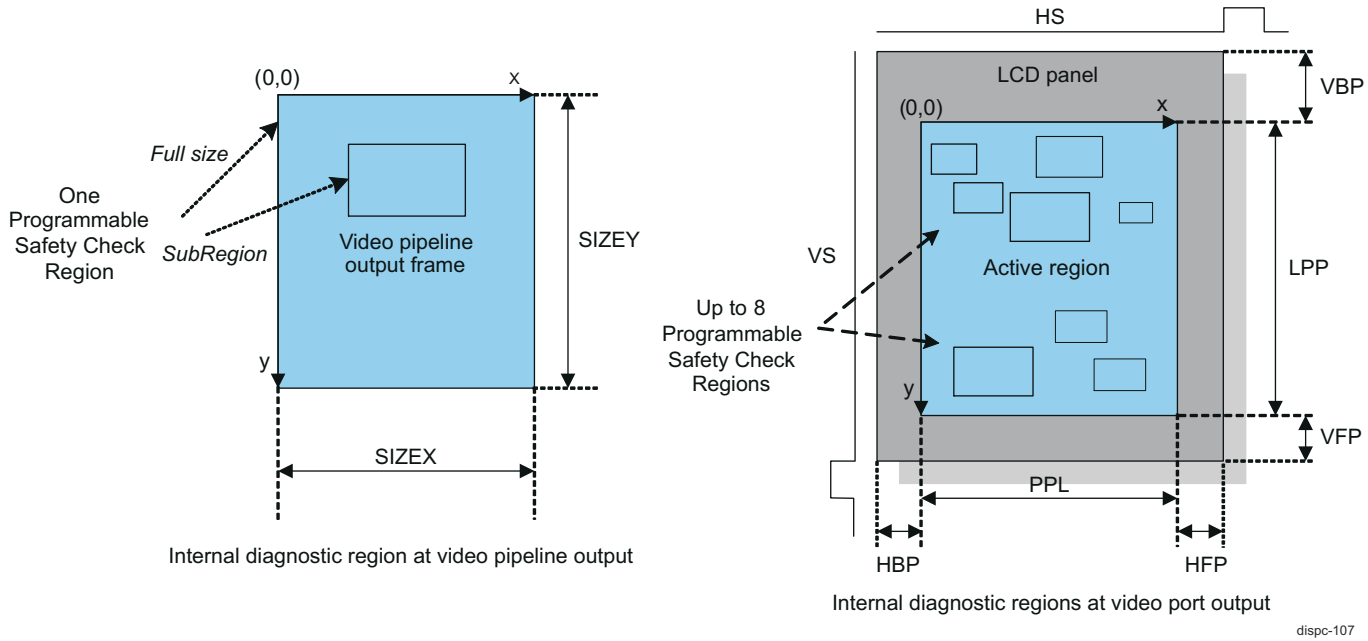


Figure 12-372. DISPC Internal Diagnostic Check Region Examples

The internal diagnostic region in the video pipeline captures data, only if the embedded alpha data is not equal to 0 (that is, non-transparent pixels). The internal diagnostic regions in the display video port output captures all active video pixels within the region boundary. The regions (up to eight) in the display output should be typically non-overlapping areas of the screen, but the hardware does not restrict them to be non-overlapping.

Figure 12-373 shows the locations within DISPC data path where the data is analyzed.

Copyright © 2024 Texas Instruments Incorporated

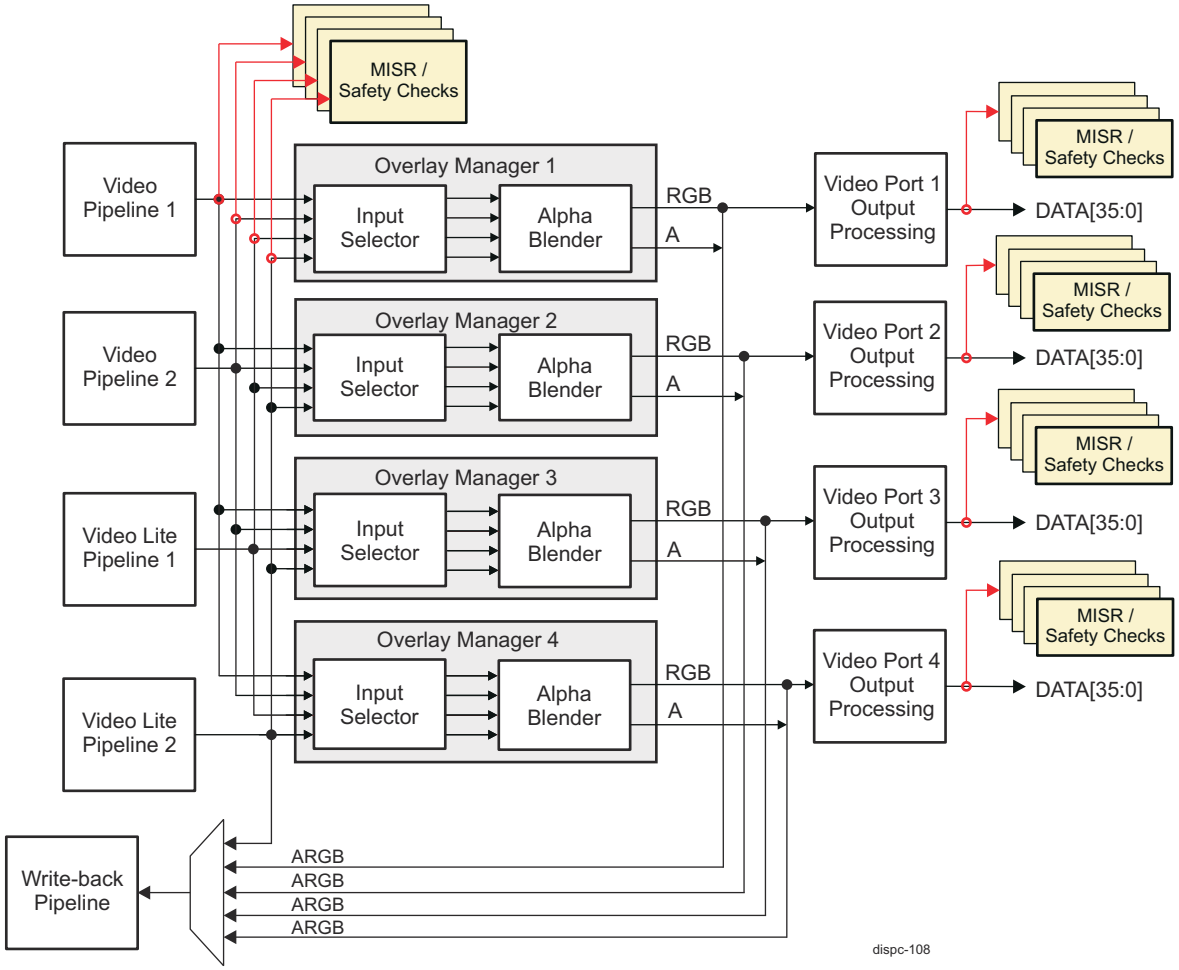


Figure 12-373. DISPC Internal Diagnostic Check Locations

Copyright © 2024 Texas Instruments Incorporated

12.6.3.12.2 Internal Diagnostic Signature Generator Using MISR

A 32-bit multiple input signature register (MISR) with 32-bit Galois LFSR polynomial, P(x)=X32+X22+X2+x+1, is used to capture a signature of active video pixel data (the 10 MSB bits of each color component) for each internal diagnostic region, as shown in Figure 12-374 and the algorithm below it.

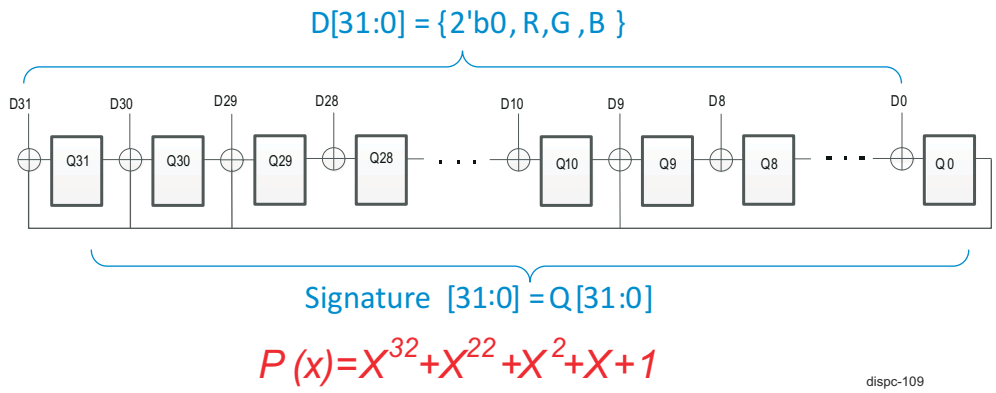


Figure 12-374. DISPC Internal Diagnostic 32-bit MISR Implementation

The MISR (32-bit Galois LFSR based) signature generation algorithm is as follows:

Tap _polynomial = 32'hE000_0200

lfsr_q = seed // initial LFSR value - any none zero value should work

For each (pixel data in the internal diagnostic region) {

data_in[31:0] = { 2'b0, r,g,b } // 10 MSB bits of R,G,B components

lfsr_d = (data_in$^{(lfsr_q >> 1)}$)

If (lfsr_q[0]) lfsr_d = lfsr_d^{Tap_polynomial}

lsfr_q = lfsr_d

}

signature = lfsr_q

All MISRs are initialized at the beginning of each frame with a non-zero "seed" value either with the default constant value (0xFFFF_FFFF) or with the programmable seed configuration value in SAFETY_LFSR_SEED register for each group of regions (refer to Table 12-353, *DISPC Internal Diagnostic Check Regions Parameters*). Note that a same seed must be used to compare signatures between two frames. If different than the default seed value is desired, then a SEEDSELECT parameter must be set before the corresponding SAFETY_LFSR_SEED register is configured with the customer seed value.

All captured signatures from the internal diagnostic regions are memory mapped to read-only SAFETY_CAPT_SIGNATURE registers for each video port (refer to Table 12-353, *DISPC Internal Diagnostic Check Regions Parameters*). A SAFETY_CAPT_SIGNATURE register is updated with the signature from each sub-region at the end of every frame. This register returns the signature of the last frame data when the MISR is enabled. When MISR is disabled, this register is cleared.

The MISR aliasing (faulty signature matches fault-free signature) probability is:

$(2^{(L-n)}-1)/(2^L-1) = \sim 2^{(L-n)}/2^L = 2^{-n}$ for large L, where

n = length of signature register

L = length of input sequence

Using the above approximation, the result is:

n=4, aliasing probability = 6.25%

Copyright © 2024 Texas Instruments Incorporated

n=16, aliasing probability = 0.0015%

...

n=32 (as in DISPC MISR), aliasing probability is ~2^{-32} (negligible)

12.6.3.12.3 Internal Diagnostic Checks

If the data correctness check is enabled (see Table 12-353, *DISPC Internal Diagnostic Check Regions Parameters*):

- When a new signature is generated, it is compared against the reference signature provided by the software. A SAFETY_REF_SIGNATURE register must be configured with a reference signature data, see Table 12-353, *DISPC Internal Diagnostic Check Regions Parameters*.
- If signatures do not match, an interrupt event is generated to indicate data mismatch.

If the freeze frame detection is enabled (see Table 12-353, *DISPC Internal Diagnostic Check Regions Parameters*):

- When a new signature is generated, it is first compared against the saved signature (from previous frame).
  - There exists a capability within the frame freeze detection logic to skip alternate frames during comparison. This feature is useful while handling ping-pong buffers or interlaced input video. This feature is controlled via the [12-11] FRAMESKIP field within a SAFETY_ATTRIBUTES register.
- If signatures match, an internal counter used to keep track of the number of frames with no data change (for the region) is incremented.
- If signatures do not match, the counter is cleared.
- If the counter value is greater than the user programmed freeze frame detection threshold value (see Table 12-353, *DISPC Internal Diagnostic Check Regions Parameters*), an interrupt event (VIDSAFETYREGION_IRQ or VPSAFETYREGION_IRQ, see Section 12.6.3.5, *DISPC Interrupt Requests*) is generated to indicate a possible freeze frame detection. The threshold value must be configured with the maximum number of identical successive frames allowed before an interrupt is generated.
- After the comparison, the signature is saved as the previous signature.
- The counter is cleared when the interrupt event is generated or when freeze frame detection check is disabled.

This frame freeze is different from the display getting frozen due to pipeline lock up. In that case, the DISPC will generate an SYNCLOST_IRQ and/or DMA VIDBUFFERUNDERFLOW_IRQ interrupt. The freeze frame detection is for source data getting frozen while the DSS is working normally.

The two internal diagnostic checks are continuously performed over multiple frames as long as their mode enable register bits are set.

12.6.3.12.4 Internal Diagnostic Check Limitations

The internal diagnostic check is only be available when the DISPC is outputing RGB/YUV component data with separate sync signals. The internal diagnostic functions are not available for the following output modes:

- YUV422 embedded sync modes (BT.656 and BT.1120)
- RGB TDM (Time Division Multiplex) mode

12.6.3.13 DISPC Security Management

12.6.3.13.1 Security Implementation

DSS supports the following security features:

- Secure mode configuration
- Illegal connection prevention

Secure Mode

The DISPC supports secure mode configuration registers (DSS0_VID_SECURE, DSS0_WB_SECURE, DSS0_OVR_SECURE and DSS0_VP_SECURE) which defines the "secure mode" attribute of each pipeline, overlay manager, and video port instance in DISPC. These registers can only be modified by a host with an appropriate secure privilege (MReqSecure=1). When the SECURE bit corresponding to an instance is set by a secure host, the instance is deemed to be in "secure mode" and the DISPC hardware prevents the output of the

Copyright © 2024 Texas Instruments Incorporated

instance getting connected to a non-secure downstream module. Also, any DMA transfer initiated by a secure pipeline will have its OCP in-band signal MReqSecure set to HIGH to indicate that is a secure mode transaction request.

By default, all pipelines, overlay managers, and video ports are in a "non-secure mode". The corresponding SECURE bits in the DSS0_VID_SECURE, DSS0_WB_SECURE, DSS0_OVR_SECURE and DSS0_VP_SECURE registers are active, only when the DSS0_COMMON_DISPC_SECURE_DISABLE[0] SECURE_DISABLE register bit is configured properly to 0x0. When the SECURE_DISABLE bit is set to 0x1, the SECURE register bits are non-active and DISPC will behave as non-secure module.

Illegal Connection Prevention

The DISPC hardware enforces the following rules to prevent an illegal connection:

- Secure input pipeline can only connect to a secure overlay manager: If any host (secure or non-secure) configures an overlay manager input selection to connect a secure input pipeline to a non-secure overlay manager, then the DISPC hardware will block the connection and issue a "security violation" interrupt (SECURITYVIOLATION_IRQ) to alert the host.
- Non-Secure WB pipeline can take its input only from a non-secure input pipeline or a non-secure overlay manager output. Otherwise, the DISPC hardware will block the data transfer through WB and issue a "security violation" interrupt to alert the host.
- Non-Secure video port can only display data from a non-secure overlay manager output. Otherwise, the DISPC hardware will block the data display on the video port and issue a "security violation" interrupt to alert the host.

Copyright © 2024 Texas Instruments Incorporated

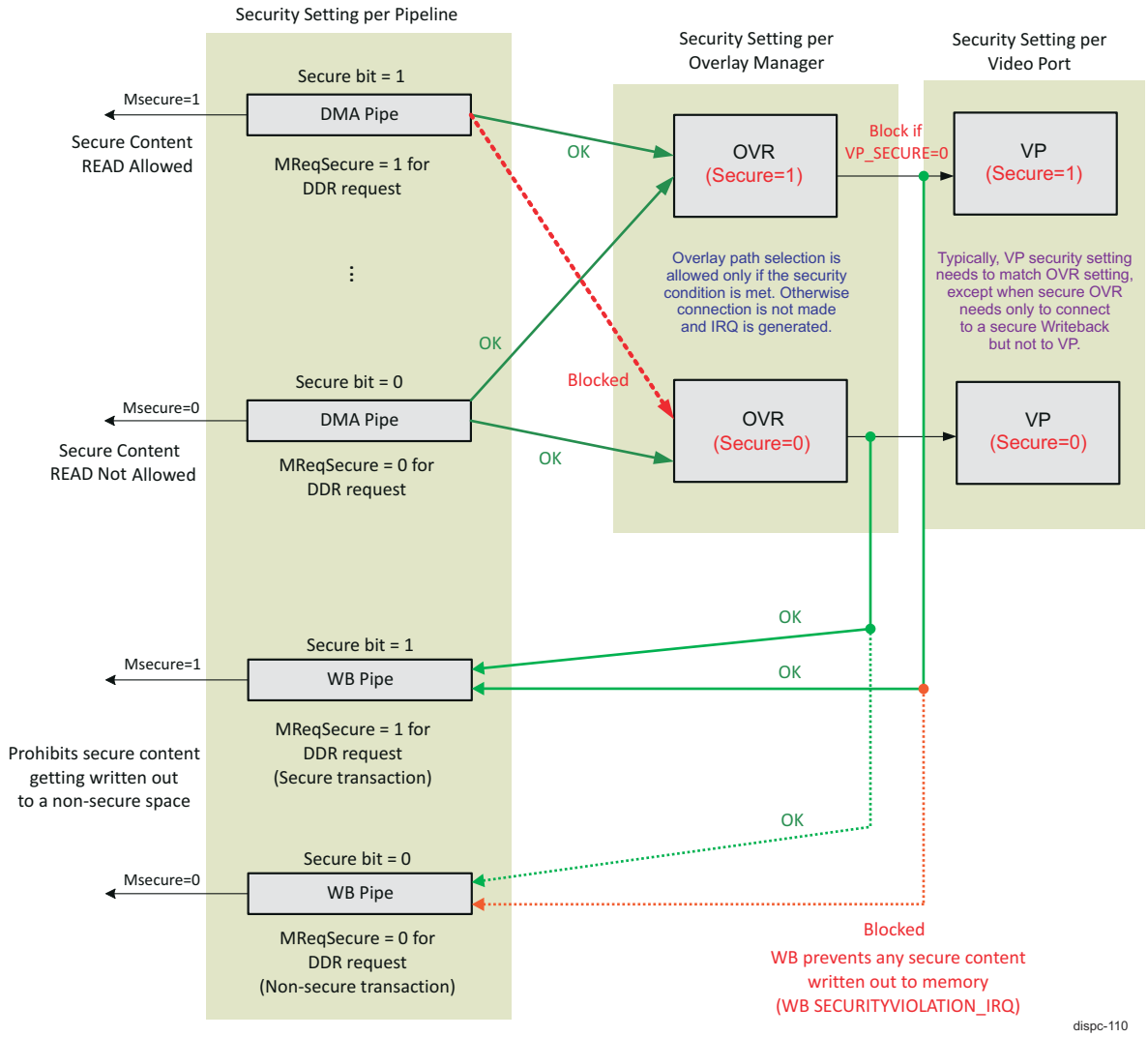


Figure 12-375. DISPC Secure Bit Setting and Illegal Connection Block

12.6.3.13.2 Secure Mode Configuration

The SECURE bit in DSS0_VID_SECURE, DSS0_WB_SECURE, DSS0_OVR_SECURE and DSS0_VP_SECURE registers is set/reset by a secure transaction. When the SECURE bit has been set, the software in "secure mode" is responsible for checking the DISPC configuration. The SECURE bit is propagated by DISPC to the system Interconnect in order to qualify all DISPC requests as secure or non-secure requests, based on the secure bits defined in the control register.

When DISPC accesses the frame buffer, in the case the SECURE bit has been reset and the frame buffer has been set secure, the DISPC will receive an "error" in response of non-secure requests.

12.6.3.14 DISPC Resources Sharing

For resource sharing of sub-components across different hosts processors or virtual machines, the DISPC implements the following features:

- Register region in memory per sub-component
- Interrupt duplication
- Independent context update for pipelines
- CHANNELID support

Copyright © 2024 Texas Instruments Incorporated

12.6.3.14.1 Register Region per Sub-component

The DISPC register regions in memory are partitioned such that each DISPC sub-component has its own region as shown in Table 12-354. This allows the SoC infrastructure to firewall the register memory regions in such a way that a particular processor host or virtual machine can only access the sub-component(s) allocated to it.

Table 12-354. DISPC Register Memory Region Partitioning

Region Name	Description	Byte Offset
DISPC_0_COMMON_M	COMMON Region for Master Host (Control Processor)	0x0
DISPC_0_COMMON_S0	COMMON Region for Slave Host-0	0x10000
VIDL1	Video Lite 1 Pipeline	0x20000
VIDL2	Video Lite 2 Pipeline	0x30000
VID1	Video 1 Pipeline	0x50000
VID2	Video 2 Pipeline	0x60000
OVR1	Overlay 1	0x70000
VP1	Video Port 1	0x80000
OVR2	Overlay 2	0x90000
VP2	Video Port 2	0xA0000
OVR3	Overlay 3	0xB0000
VP3	Video Port 3	0xC0000
OVR4	Overlay 4	0xD0000
VP4	Video Port 4	0xE0000
WB	Write-Back Pipeline	0xF0000
DISPC_0_COMMON_S1	COMMON Region for Slave Host-1	0x100000
DISPC_0_COMMON_S2	COMMON Region for Slave Host-2	0x110000

12.6.3.14.2 Interrupt Duplication

The DISPC supports multiple interrupt outputs, each of which can be set up independently of the other. There are also separate interrupt lines for different category of events such as functional, internal diagnostic error, and secure. This allows fully independent monitoring/control of the interrupt events by multiple hosts (up to 4). For more details on iterrupts, see Section 12.6.3.5, *DISPC Interrupt Requests*.

12.6.3.14.3 Independent Context Update for Pipelines

The DISPC supports use-cases where portions of a single display (driven out of single OVR and VP) are controlled by multiple host processors (or virtual machines). In such a case, having a single GO bit per VP (display output) would result in partial updates of a particular pipeline register configuration on a frame boundary. The DISPC provides the additional capability to sync register configurations of each pipeline independently to a chosen frame boundary. This is achieved by setting the <PIPE> GO bit.

When the DSS0_COMMON_GLOBAL_GOBITMODE[0] MODE register bit is set, the individual pipeline register configurations are synced (shadow to work copy) only on frame boundaries where the DSS0_VID_PIPE_GO[0] GOBIT register bit of each used pipeline is set. The scheme is as shown in Figure 12-376.

Copyright © 2024 Texas Instruments Incorporated

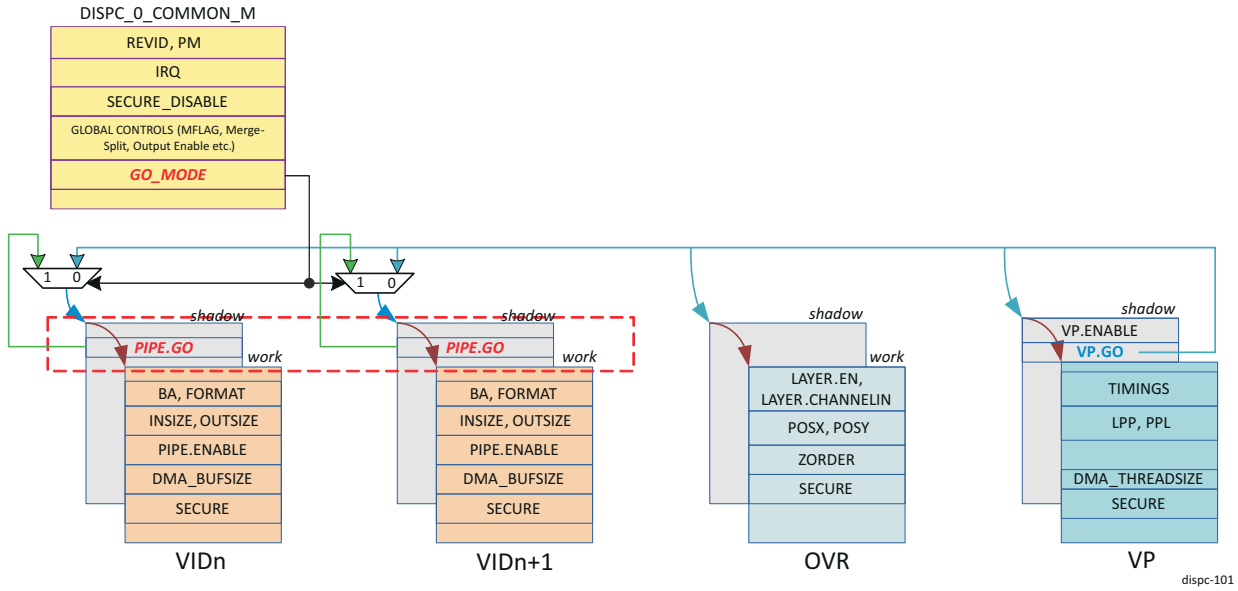


Figure 12-376. DISPC PIPE GO Bit Implementation

12.6.3.14.4 CHANNELID Support

In order to support IO virtualization that allows control of different DISPC pipes by different hosts, the DISPC pipelines must be identifiable on the SoC system interconnect with different channel IDs. To achieve this, every read and write transaction out of the DISPC DMA is tagged with a CHANNELID. The SoC infrastructure then uses the CHANNELID to set the various parameters (address translation, priority, security, etc.) on a per-channel basis.

The LSB bit of the CHANNELID (that is, CHANNELID[0]) is always used to differentiate between the two planes when a pipe is fetching a 2-plane pixel format (for example, YUV420-NV12). When the pipeline is fetching a 1-plane pixel format, the LSB bit is always '0'. The mapping of the MSB bits of the CHANNELID is as shown in Table 12-355.

Table 12-355. DISPC CHANNELID Mapping

CHANNELID[m:1]	Pipeline Mapping
0	VID1
1	VIDL1
2	VID2
3	VIDL2
4	WB

12.6.3.15 DISPC Shadow Mechanism for Registers

Some DISPC registers are termed *shadow registers*. The shadow registers allow the software to modify them at any time, without direct effect on the DISPC hardware configuration. When all the values for a given configuration are written into the shadow registers, software must set only one register bit to validate the configuration. When the hardware reaches the end of the current frame and sees that the bit has been set by software, the new configuration is now the configuration used by the hardware.

The DSS0_VP_CONTROL[5] GOBIT bit enables the hardware to use the new configuration, for all shadow registers associated with each VP output.

The registers are statically associated to a particular output (for example, timing registers) or dynamically associated to one output at a time (for example, video registers).

Copyright © 2024 Texas Instruments Incorporated

12.6.4 MIPI Display Serial Interface (DSI) Controller

12.6.4.1 DSI Block Diagram

The MIPI DSI v1.3.1 Transmitter Controller (DSITX) provides an interface that receives data and control from the host processor display system using either the DPI or SDI input bus interfaces. The DSITX will translate the incoming pixel information and control signals into an internal packed byte format, in the case of DPI, or pass in the prepacked SDI byte format, before the internal byte format data is packetized and sent to the MIPI DSI Compatible display via the MIPI D-PHY physical interface. It supports video and command mode displays and can work in multi-display mode using virtual channel identification on the packets.

The DSITX controller provides two interfaces for connection to a display panel. Normal operation for a panel supporting DCS commands can be done using either SDI interface, or using the APB (Advanced Peripheral Bus) access to DIRECT_CMD registers. Video streaming applications can only use the SDI or DPI interface. The DSI controller supports flow control using the SDI video interface.

The DSITX implements the stream arbitration and low-level protocol layer functionalities required by MIPI DSI specification v1.3. It supports up to 4 x 2.5 Gbps D-PHY data lanes in a single-link configuration and handles the byte lane mapping per use case (1, 2, 3, or 4-lanes).

The DSITX controller block diagram is illustrated in Figure 12-377.

Copyright © 2024 Texas Instruments Incorporated

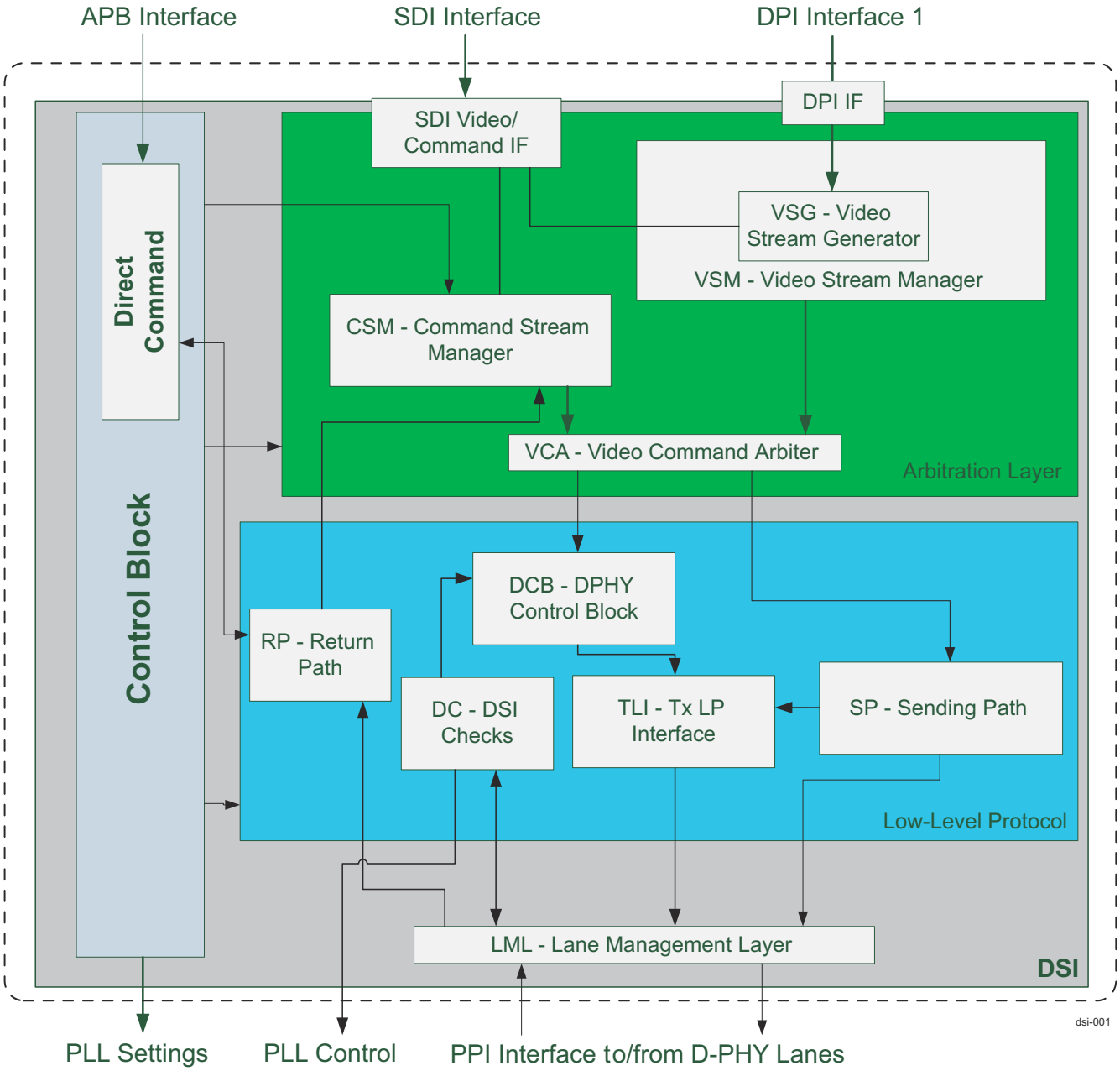


Figure 12-377. DSITX Controller Block Architecture

12.6.4.2 DSI Clocking

The DSI receives all the clocks from DISPC or DPHY_TX modules. There are no other dedicated PLLs. For more details on DSI clocks mapping at DSS and SoC level, see *DSI Integration*.

The DSI requires multiple clocks running asynchronously, that are shown in Table 12-356.

Table 12-356. DSI Clocking

Clock	Direction	Min	Max	Constraints	Description
dpi_0_clk (Async to all other clocks)	Input	7 MHz	425 MHz	Fpixel_clk = Ftx_byte_hs_clk × active_lanes × 8/(bits_per_pixel)	Used for DPI interface only. Frequency range depends on expected data rate with respect of number of data lanes, data lane frequency and frame rate

Copyright © 2024 Texas Instruments Incorporated

Table 12-356. DSI Clocking (continued)

Clock	Direction	Min	Max	Constraints	Description
sys_clk (Async to all other clocks)	Input	7 MHz	250 MHz	1. Cannot be slower than tx_byte_hs_clk x datapath_size/ if_datasize (risk of underrun) in SDI mode. 2. Must be greater than: rx_esc_clk x 8. 3. Speed should be sufficient to provide enough data in SDI operation. 4. Must be faster than the tx_byte_hs_clk for Direct command operation.	Main functional clock Also used for the VBUS/APB interface
dphy_0_tx_byte_hs_clk (Async to all other clocks)	Input	10 MHz	312.5 MHz	The max value in SDI interface operation depends on sys_clk: can't exceed sys_clk freq x if_datasize datapath_size Otherwise 312.5 MHz to match the 2.5Gbps limit on DPHY v1.3 specification	DPHY PPI Byte Clock Frequency set by the DPHY and its High Speed input clock (tx_byte_hs_clk = dphy bit rate /8) Maximum limit due to risk of underrun
dphy_0_tx_esc_clk (Async to all other clocks)	Input	1 MHz	20 MHz	No minimum limit is given by the DSI itself - while regular function mode max limit is 20 MHz	TX Escape Clock
dphy_0_rx_esc_clk (Async to all other clocks)	Input	1 MHz	10 MHz	No minimum limit is given by the DSI itself - while regular function mode max limit is 10 MHz	RX Escape Clock

Figure 12-378 describes the clock scheme and domains of the DSI_TX.

Copyright © 2024 Texas Instruments Incorporated

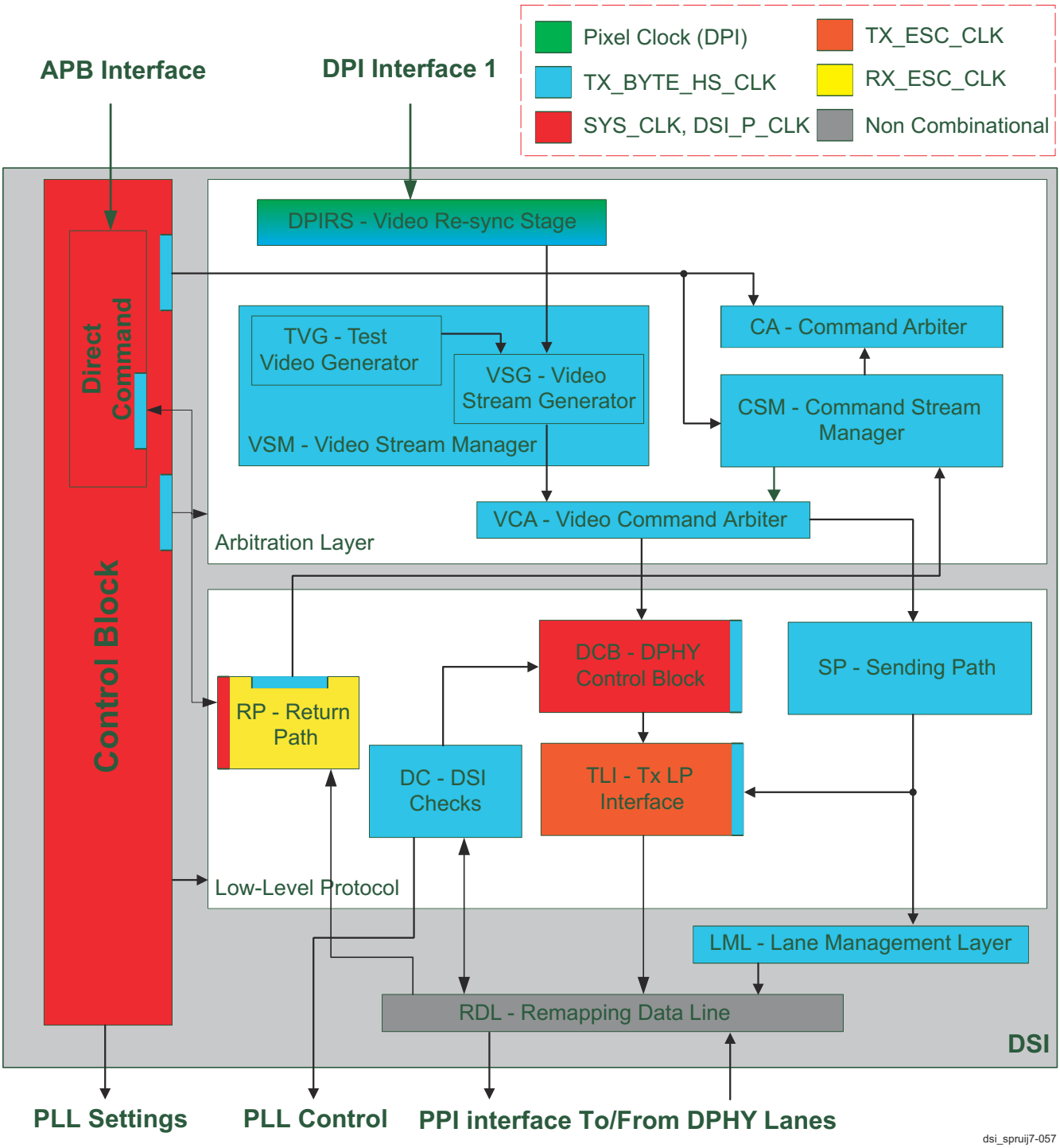


Figure 12-378. DSITX Clock Scheme and Domains

12.6.4.3 DSI Reset

The DSI has one active low reset input. The entire internal logic is reset by this reset. Internal resets (synchronous and asynchronous) for different clock domains are generated internally (synchronized to the respective clock domain).

12.6.4.4 DSI Power Management

The DSI implements a power management protocol to interface to a PSC (Power and Sleep Controller) module SoC level.

Copyright © 2024 Texas Instruments Incorporated

Figure 12-379 shows the expected sequence from SW while performing a *clkstop_req* to the DSI.

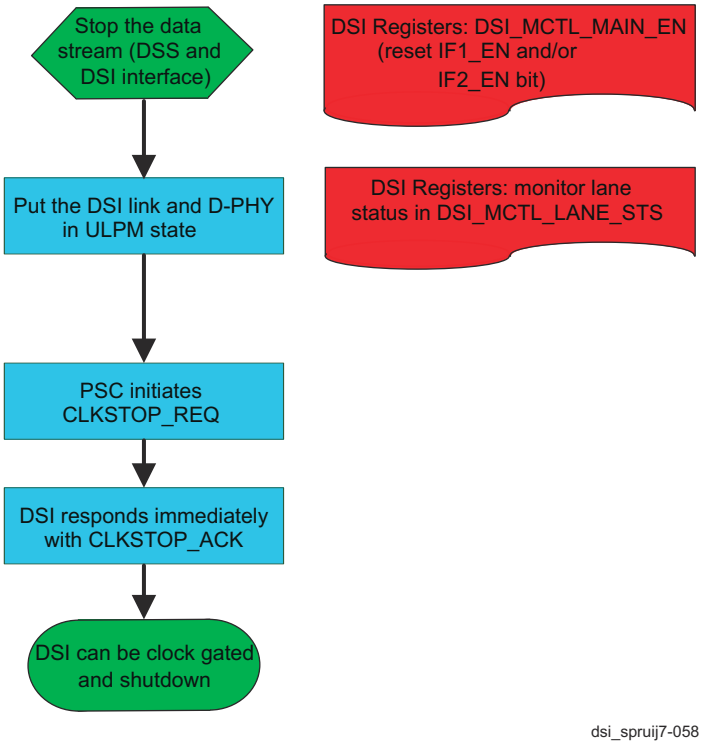


Figure 12-379. DSI Clock Gate / Power Off Procedure

12.6.4.5 DSI Interrupts

Table 12-357 shows the interrupts that are generated by the DSI module.

Table 12-357. DSI Interrupts

Interrupt (n = 0)	Description	Source Module	Source Signal Name
dsi_#n_func_intr	DSI-#n Func Interrupt	DSI Controller	dsi_irq_n
dsi_#n_safety_error_fatal_intr	DSI #n Internal Diagnostic Fatal interrupt	DSI Controller	asf_int_fatal
dsi_#n_safety_error_nonfatal_intr	DSI #n Internal Diagnostic Non-fatal interrupt	DSI Controller	asf_int_nonfatal
ecc_intr_uncorr_level_sys_intr	ECC aggr uncorrectable error	-	-

12.6.4.6 DSI Internal Interfaces

12.6.4.6.1 Video Input Interfaces

DSITX supports following video streaming interfaces:

- DPI (Uncompressed video stream)
- SDI (Serial Display Interface)

The DSI does not require DSI sub-link support. Therefore, only one video mode source is supported.

12.6.4.6.1.1 Pixel Mapping

The pixel bit color mapping for the DSITX input interfaces are as shown in Figure 12-380. This matches with the pixel bit mapping on the output of DISPC for the supported formats.

Copyright © 2024 Texas Instruments Incorporated

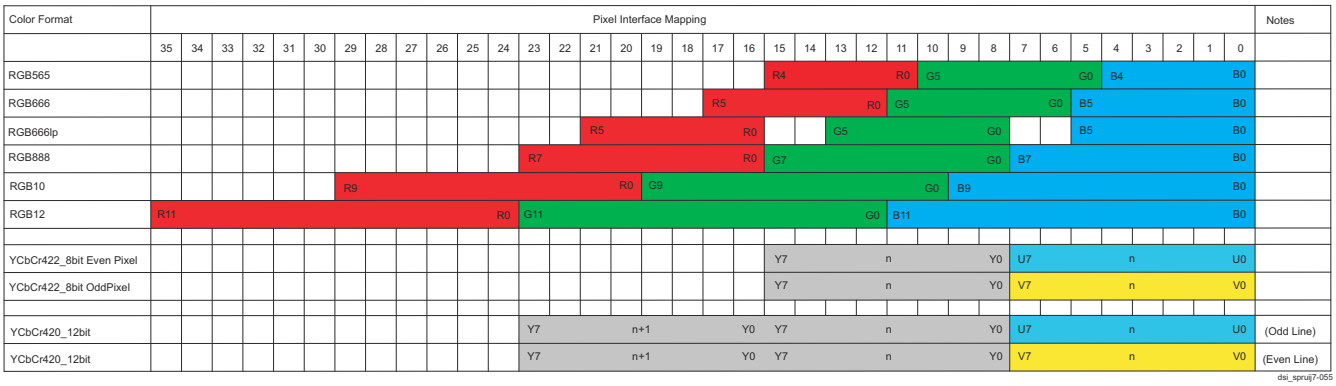


Figure 12-380. DSI Pixel Mapping On The DSI-DPI Interface

Table 12-358. DSI Pixel Format Interoperability between DISPC and DSITX

DSITX DPI Pixel Format	DISPC VPOUT Pixel Format[(1)] DSS0_VP_CONTROL[10-8] DATALINES Register Field Setting	DPI Interface Support	SDI Interface Support
RGB565	16-bit (DATALINES = 0x1)	Supported	Supported
RGB666	18-bit (DATALINES = 0x2)	Supported	Supported
RGB666lp	N/A	Not Supported	Not Supported
RGB888	24-bit (DATALINES = 0x3)	Supported	Supported
RGB10	30-bit (DATALINES = 0x4)	Supported	Not Supported
RGB12	36-bit (DATALINES = 0x5)	Supported	Not Supported
YcbCr422 8-bit	N/A	Not Supported	Not Supported
YCbCr420 12-bit	N/A	Not Supported	Not Supported

(1) Output aligned on the LSB of the pixel data interface.

12.6.4.6.2 DPI (Pixel Stream Interface)

- One uncompressed video stream input
- CEA-861 signaling/timing compatible interface
- Pixel clock rate range from 7 MHz - 425 MHz
- Frame Resolution – 4MPix@60fps equivalent frame size – supporting various frame resolutions and aspect ratios (with pixel clock freq < MAX_dsi_pixel rate = 425 MHz)
- Progressive video timing only
- Maximum Width - 8K (should only be constrained by the throughput)
- RGB format only –color space conversion or chroma up-sampling NOT supported

12.6.4.6.2.1 Signals

DPI interface supports the following set of DPI standard signals:

- VSYNC – Video Vertical Blanking indication signal as defined in the above standards
- HSYNC - Video Horizontal Blanking indication signal as defined in the above standards
- DATA ENABLE – Video data enable indication signal as defined in the above standards
- DATA - 36 bits of data for supporting up to 12bpp
- Clock – As defined for each format from 7 MHz - 425 MHz

12.6.4.6.3 SDI (Serial Data Interface)

DSITX uses a SDI interface (32-bit) to receive active video data from an internal frame memory using a DMA or other data pull mechanism. The interface uses req/ready handshake signaling to control the data flow.

The DSITX SDI Interface supports one of the following modes:

- Command-only

SDI-DSI Interface requires following signals:

- dsi_if_valid

Copyright © 2024 Texas Instruments Incorporated

- dsi_if_stall
- dsi_if_start (Line Start)
- dsi_if_frame_sync (frame_sync)
- dsi_if_data

The SDI interface of DSI_TX receives data directly from DSS VP output and is functional when the user configures the DSS to operate in COMMAND_STALLMODE (DSS0_VP_CONTROL[11] STALLMODE = '1' and DSS0_VP_CONTROL[12] STALLMODETYPE = '0').

12.6.4.6.3.1 Secure Display Support

Only secure peripherals can be connected to a secure DPI (VP) output of DSS. In order to support this requirement, the DSS exports a secure qualifier to the DSI_TX. Inside the DSI a SECURE register is implemented (DSI_WRAP_DPI_SECURE), which contains SECURE bits, [0] DPI_0_SECURE and [1] DPI_0_SECURE_VIOLATION, corresponding to the DPI port. These SECURE bits can only be set or reset by a secure host (vbusp transactions with secure qualifier set).

The behavior of DSI module for different settings of SECURE register bit and SECURE qualifier from DSS is as shown below:

Table 12-359. Secure Display Support

SECURE Register Bit (DSI)	SECURE mqualifier (DSS)	Security Violation Status	Comments
0	0	0	Data is passed (non-secure DSS data through non-secure DSI)
1	0	0	Data is passed (non-secure DSS data through secure DSI)
1	1	0	Data is passed (secure DSS data through secure DSI)
0	1	1	**Security Violation**. Data is blocked (secure DSS data through non-secure DSI)

Once a security violation is detected, it is captured in the DSI_WRAP_DPI_SECURE[1] DPI_0_SECURE_VIOLATION register status bit.

12.6.4.7 DSI Programming Guide

This section describes the programming guidelines for the DSI Controller and D-PHY.

The goal of this section is to present more in detail how the DSI link should be used at application level plus some scenarios of use of the DSI link (start-up, display switch on/off, dual-display).

12.6.4.7.1 Application Guidelines

The purpose of this section is to provide guidelines on how to drive the DSI host controller, provide general sequence of operations and some typical application notes.

12.6.4.7.1.1 Overview of a Display Subsystem

Figure 12-381 shows an overview of a display subsystem. It covers the point to point interfaces between the DPI/SDI and DSI blocks. Other blocks and interfaces are included to give a better understanding of the environment for this interface.

A DSI interface is made up from one DSI controller and one D-PHY block. D-PHY is the physical layer of the DSI interface. The DSI block handles protocol. The DSI interfaces support displays that comply with the MIPI DSI standard.

The displays can be of either command mode type or video mode type. Command mode displays have a frame memory that can hold the entire image. The display is refreshed from this memory and only changes of the display content are sent over the display interface. Video mode displays do not have this memory and thus all data for every refresh must be transferred in real time.

Copyright © 2024 Texas Instruments Incorporated

The SDI block can support one display in command mode. By using command mode displays and rerouting of connections, between display updates, more than two displays can be connected and active.

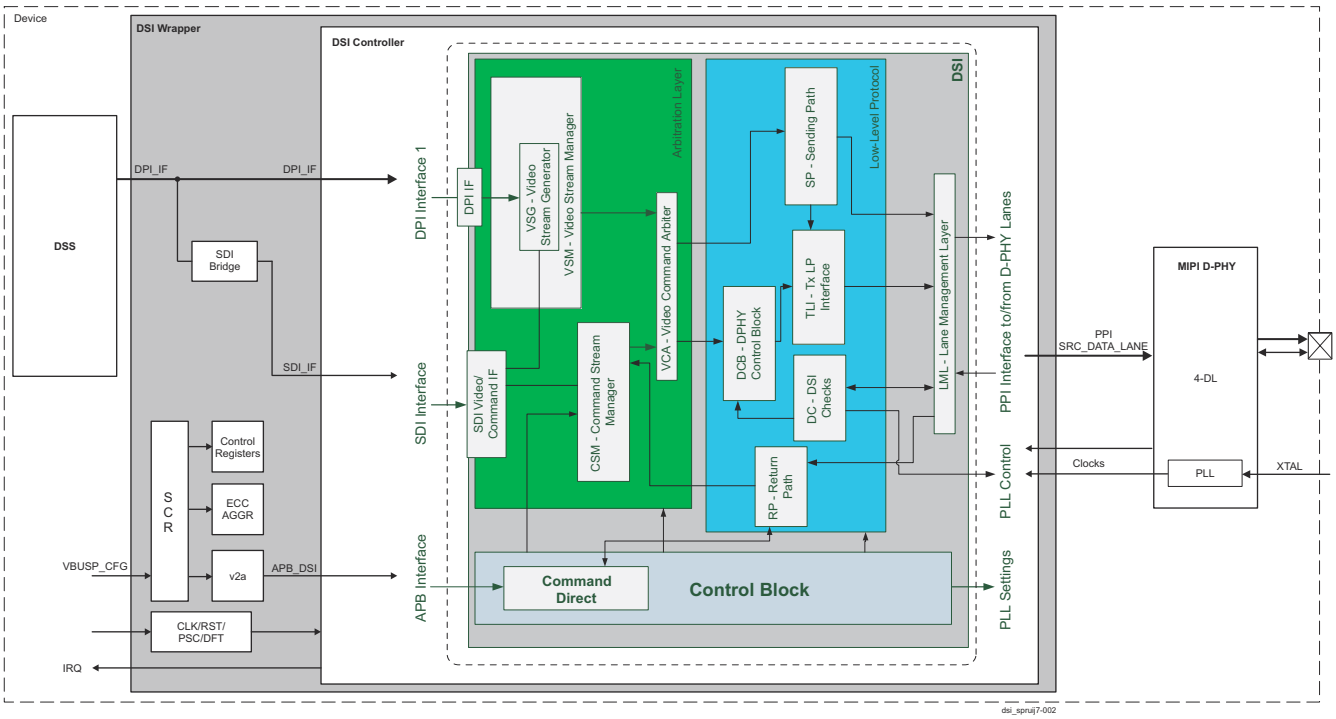


Figure 12-381. Example Display Subsystem Showing Controller and PHY Blocks

12.6.4.7.1.2 D-PHY And DSI Configuration

The DPHY and DSI controller must be configured before any panel video display can be used. All accesses to the DPHY to configure it for use with the DSI Controller should be performed through the DPHY own APB interface .

12.6.4.7.1.3 DSI Controller Initialization

The DSI controller will always require the DPHY and data path configuration to be set after power up and any hard reset. The minimum sequence should define the data path lane configuration; mappings, available lanes, DPHY power, resets and lane swap. All these register values should be set before the DSI controller is enabled for functional operation.

The DSI registers are accessed through the APB interface. The programming sequence for command, video and controller configuration must follow a sequence of accesses to ensure the clock gate control updates the internal nodes before the link is enabled.

All DSI_VID_xxx registers must be programmed, if required, before the write to the DSI_VID_MAIN_CTL register.

All DSI_MCTL_DPHY_xxx registers must be programmed, if required, before the write to the DSI_MCTL_MAIN_DATA_CTL register to set the [0] LINK_EN bit.

12.6.4.7.1.4 Panel Configuration Using Command Mode

The DSI controller can provide DCS command access using either the SDI interface or through the APB interface control of the DIRECT COMMAND registers (see Table 12-360). The panel registers will be programmed using the LP mode before the high-speed video is enabled. Commands can either be sent using the command interface or though the Command FIFO using the APB registers.

The command FIFO access is done though the APB Command mode registers by forming packets of DCS commands. The DCS commands can be found in the MIPI DCS specification or the display panel data sheet.

Copyright © 2024 Texas Instruments Incorporated

DCS commands are essentially register write or read transfers to panel registers. The initialization of a panel will require APB register writes to fill the FIFO with the DCS commands. The commands can be grouped into a long packet by sending the packet split into 4 bytes to fill the FIFO using a write to the DSI_DIRECT_CMD_WRDAT register.

Table 12-360. DSI Direct Command Mode Registers

DSI Register	Description
DSI_DIRECT_CMD_SEND	Direct Command - trigger the direct command sending - write only
DSI_DIRECT_CMD_MAIN_SETTINGS	Direct Command - main settings
DSI_DIRECT_CMD_STS	Direct Command - status - read only
DSI_DIRECT_CMD_RD_INIT	Direct Command - stop read operation
DSI_DIRECT_CMD_WRDAT	Direct Command - data to write byte 0 to 3
DSI_DIRECT_CMD_FIFO_RST	Direct Command- reset the write FIFO pointer

Each DCS command packet to be sent can be built by first selecting the packet configuration using the DSI_DIRECT_CMD_MAIN_SETTINGS. The DCS Command data types are defined in the DSI specification based on the number of parameters that are expected to be sent. Configuration of a panel display will normally use DCS write data types.

Table 12-361 shows the DSI_DIRECT_CMD_MAIN_SETTINGS register bit description.

Table 12-361. DSI Main Settings Register Description

DSI_DIRECT_CMD_MAIN_SETTINGS Register Bit	Description			
[24] CMD_LP_EN	Enables LP sending for the command request.			
[23:16] CMD_SIZE	Size of the command in case of write command - when written value is bigger than 0x10, the value is rounded to 0x10 for write - when size is bigger to 0x2 for read it is rounded to 0x2.			
[15:14] CMD_ID	In case of read/write command, Virtual Channel of the command.			
[13:8] CMD_HEAD	In case of read/write command, data type of the command:			
	0x05	DCS Write	0 parameters	Short Packet
	0x15	DCS Write	1 parameter	Short packet
	0x39	DCS Long Write	N parameters	Long Packet
[3] CMD_LONGNOTSHORT	This bit must be tied to one.			
[2:0] CMD_NAT	Nature of the direct command: 000: write command.			

Once the packet is loaded, issue a DIRECT_CMD_SEND (via the DSI_DIRECT_CMD_SEND register) and the data will be sent in LP mode.

12.6.4.7.1.5 VIDEO Interface Configuration

The DSI video operation requires the configuration of the VSG and TVG registers to match the panel configuration. The two video interface options will also require careful selection of the clock and registers to achieve error free video streaming.

The DSITX controller supports the mechanism for initial skew calibration for D-PHY data rates greater than 1.5 Gbps.

DPI Video Interface Operation

The time taken to output a frame on the PPI needs to match what is coming in on the DPI. This is best achieved by using the recommended clock ratio between the pixel and byte clocks. If this is the case, then the blanking and active data periods must be matched up. The DSI controller will slave its frame timing to the incoming DPI **VSYNC**, if it is programmed to generate a frame of slightly less than the same size when the VFP blanking is considered.

Copyright © 2024 Texas Instruments Incorporated

The controller works by transitioning to LP during the last programmed line of VFP. It will then remain in LP until the start of the next frame. So, programming the controller to match the DPI, but with at least one fewer line of VFP should result in a reliable configuration.

Program the DSI vertical size registers as follows:

- VSA = DPI_VSI (lines, minimum of 2)
- VBP = DPI_VBP (lines, minimum of 0)
- VACT = DPI_VACT (lines)
- VFP < DPI_VFP (minimum of 1)

The timing should also be matched per line, therefore the blanking and active periods should match. DPI horizontal timing is measured in pixels, whereas the DSI controller uses bytes. If any of the DPI related interrupts are triggered, then this highlights that the FIFO depth and/or the vsync_delay settings require to be tuned to the current configuration. Simulating the core operation with the expected clocks is the best way to ensure the FIFO depth and vsync_delay is suitable.

The relationship therefore depends upon the pixel format, which could be 24, 18 or 16 bpp. Additionally, the PPI short packets and packet headers that are inserted by the controller must be accounted for. HSA should be reduced by 14 bytes to account for the HSS short packet (4 bytes), the long blanking packet header and footer (4 + 2 bytes) and the HSE short packet (4 bytes). HBP should be reduced by 12 to account for the header and footer on the blanking packet (6 bytes) plus the header on the active data packet (6 bytes). HFP should be reduced by 6 bytes to account for the long packet header and footer. Finally, for lines with no active data, the BLKLINE_PULSE_PCK is the total size minus 20 bytes (14 for HSA, 6 for the remaining blacking which is all combined into a single packet).

Program the DSI horizontal size registers as follows:

- HSA = (DPI_HSA × bpp/8) - 14
- HBP = (DPI_HBP × bpp/8) - 12
- HACT = (DPI_HACT × bpp/8) NOTE: (DPI_HACT × bpp/32) must be an integer.
- HFP = (DPI_HFP × bpp/8)-6
- BLKLINE_PULSE_PCK = ((DPI_HSA + DPI_HBP + DPI_HACT + DPI_HFP) × bpp/8) - 20;

TVG Generator

The TVG oversees generating dummy data (to display something even if there is no video stream running). It is also able to ease verification or validation of the DSI without having a complete environment (application or verification test bench). The controller has a Test Video Generator that can be programmed to generate a set of test colour patterns based on the display panel that will be used. The panel parameters for horizontal and vertical resolution along with the frame rate and pixel colour depth need to be set based on the datasheet information for the panel.

Figure 12-382 presents TVG MODE patterns.

Copyright © 2024 Texas Instruments Incorporated

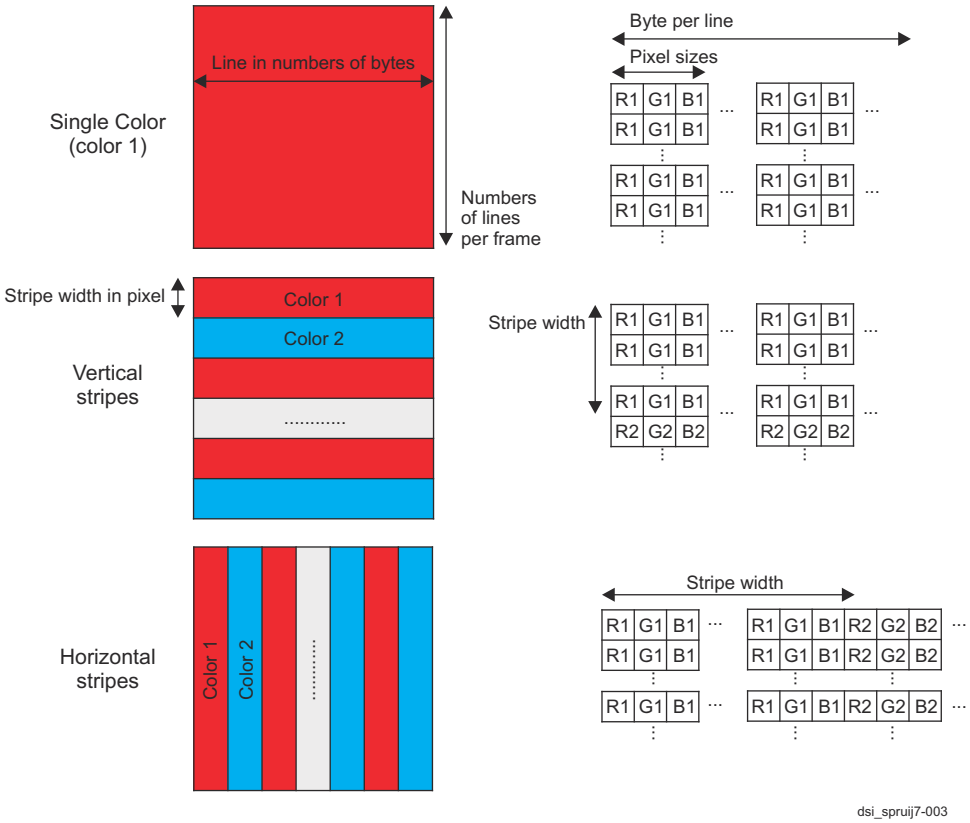


Figure 12-382. TVG MODE Patterns

The TVG generates a data stream in the same format than the one specified for the normal functional video stream (as set in VSG register). The content of that flow is specified by registers. Those registers specify the following information: Image size: Number of bytes per line (see TVG_LINE_SIZE[14:0] register) and number of lines per frame (TVG_NBLINE[12:0]).

Image kind: Single color, vertical stripes or horizontal stripes (see TVG_MODE[1:0] register). Stripe width: Significant when a stripe mode is enabled. Possible values are 1, 2, 4, 8, 16, 32, 64 and 128 (see TVG_STRIPE_SIZE[2:0] register).

Pixel kind: 16 bits RGB, 18 bits RGB, 18 bits loosely RGB, 24 bits RGB, 30 bits RGB or 36 bits RGB (see VID_PIXEL_MODE[3:0] register).

Color one: Color used in single color mode or first color when in stripe mode. 12 bits per component R/G/B. For formats using fewer bits than 12, the least significant ones are the ones considered (see COL1_GREEN[11:0], COL1_RED[11:0], COL1_BLUE[11:0] registers).

Color two: Color two when in stripe mode. 12 bits per component R/G/B. For formats using fewer bits than 12, the least significant are the one considered. (see COL2_GREEN[11:0], COL2_RED[11:0], COL2_BLUE[11:0] registers). Start pulse and stop handshake (+ stop mode) (see TVG_CTL register).

An example of the sequence is as follows:

- Select TVG for video stream generation instead of SDI 1 interface:
  - IF1_EN bit to 0 (stall signal at 1) in MCTL_MAIN_EN.
- Select video mode for interface 1 in MCTL_MAIN_DATA_CTL:
  - TVG_SEL bit to 1 in MCTL_MAIN_DATA_CTL register to select stream from TVG.
- Select generated frame format in TVG_CTL register:
  - Image format (single color, H stripes, V stripes) in TVG_MODE field.
  - Image color selection in TVG_COLOR1, TVG_COLOR1_BIS, TVG_COLOR2, VG_COLOR2_BIS.
  - Stripes size in TVG_STRIPE_SIZE field.

Copyright © 2024 Texas Instruments Incorporated

– Image size in TVG_IMG_SIZE register.

Note: The number of lines per frame and number of bytes per line. It is required that TVG settings on active area match VSG settings on active area. Any mismatch will create an error that is detected in the VSG and forces recovery mode in TVG, stopping test frame generation.

- Select TVG stop mode with TVG_STOPMODE field of TVG_CTL:
  – TVG video stream generation start/stop controlled by the TVG_RUN bit in TVG_CTL register. Polling TVG run status from the TVG_RUNNING bit in TVG_STS register is useful when setting TVG_RUN to 0. TVG video stream generation does not stop instantaneously (depends on TVG stop mode), so this should be checked to confirm the TVG has stopped.

12.6.4.7.2 Application Considerations

The following sections outline the normal operation of the DSITX controller and the programming and sequences require to operate the DSITX controller for normal video and command operation.

12.6.4.7.2.1 D-PHY Timings Control

Several DPHY timing constraints must be respected by the system, which are described in this section. When a switch in ULP is requested, the application should be aware that the D-PHY needs 1 ms to leave this state (this timing is guaranteed by the DSI itself). It makes no sense to start an ULP request if the expected time in ULP is short.

When PLL power down is asserted, the system should continue to assert for a specific minimum period of time (PLL internal requirement – please refer to DPHY documentation). Please note that shutting down the PLL implies that no clock is present on tx_byte_hs_clk input and thus the DSI link is stopped; even if the system-side interface clocks remain active, no more HS transfers can be done.

Force_stop, ppi_c_force_tx_stop (clock lane) and ppi_d_force_tx_stop (1 per data lane), can be useful to resolve certain deadlock situations, for example: DSI slave does not give back control to master, or critical DPHY error. When a 'force stop mode' is issued by the application, the system should maintain this state on the bus to ensure the request is correctly considered by the D-PHY cells and the **stop_state** is asserted and direction signal is deasserted.

reg_wait_burst_time (for the LML) must be programmed to ensure that two HS bursts are separated by at least 100 ns. The value will be based on the period of the tx_byte_clk cycle.

VCA: When trying to interleave commands to video stream, it is assumed that only write/read commands and BTA request are sent (no TE). The reason is that such transfers are slow and difficult to predict in term of duration. They could take longer than the slots available for video. The read commands are unpredictable and may cause some break in the video stream, however, they can still be used, if an appropriate response time can be guaranteed. It is the responsibility of the system integrator and the application to plan and implement recovery procedures when the video stream is stalled due to a read that takes too long.

12.6.4.7.2.2 Control Block

Most of the register contents must be resynchronized against a DSI internal clock (tx_hs_byte_clk). This means there is a certain time to pass data from one clock domain to the other. If two writes in the same register are too close, the register itself is updated but the synchronization may fail.

The formula to calculate the number of system clock periods between two writes is:.

nb_cycle(dsi_p_clk) = 6*($f_{dsi_p_clk}/f_{tx_byte_hs_clk}$ + 1)

The application must respect a period of TX ns between two write accesses to the same register (there is no issue writing different registers back-to-back).

Table 12-362 shows an example of required time between two write accesses.

Table 12-362. Example of Required Time between Two Write Accesses

$f_{dsi_p_clk}$	$f_{tx_byte_hs_clk}$	TX minimum time between 2 writes accesses
200 MHz	106 MHz	~ 90 ns
160 MHz	10 MHz	~ 650 ns

Copyright © 2024 Texas Instruments Incorporated

Table 12-362. Example of Required Time between Two Write Accesses (continued)

$f_{dsi_p_clk}$	$f_{tx_byte_hs_clk}$	TX minimum time between 2 writes accesses
200 MHz	10 MHz	~ 650 ns

Another place where asynchronous behaviour may interfere with programming in the CB is the direct command status register DIRECT_CMD_STS. Some of the bits of this register are set when a pulse (that lasts one clock period) coming from tx_hs_byte_clk domain is observed (after resynchronization on dsi_p_clk) and is reset when the clear bit is written. In use case where tx_byte_hs_clk is slow (10 MHz range), the bit can be read and a clear attempted before the end of the source pulse. This can result in the re-assertion of the bit immediately after it is cleared. For that reason, it is recommended to wait for a while between reading the bits of the DIRECT_CMD_STS register and clearing them.

Application Issues

Some register fields cannot take all the possible values but are restricted to a certain number of combinations (mode control). The DSI controller does not check that all the fields match a valid setup so it is the responsibility of the system integrator and the application to check that the written values are amongst the permitted combinations. Amongst the register fields that fall in to this category are most of the mode settings (stop mode, recovery mode, direct command type, image sizes, etc.)

Programming Coherency

The application level must ensure that the register configurations are valid for the system to operate correctly. There are several possible combinations are possible in the CB registers however should be avoided. A few examples:

- No HS transfer should be enabled when PLL is shut-off.
- TVG should not be run when the video interface is running.
- No access (from interface or from register) should be attempted while TBG is active.
- Prior to shut-down of the DPHY PLL, application should verify that the DSI link is in proper state. If the bit that enables BTA is not set and that any operation implying a BTA is sent, the system is stalled. Then the bit that enables BTA must be aligned with application needs. Moreover, read enable or TE enable cannot be set if bta_en is not set (these operations cannot be enabled when BTA is not enabled).
- TE feature must not be enabled in SDI interface.

12.6.4.7.2.3 Video Coherency

In video mode, there are a lot of sizes defined for the video stream generation. This data corresponds to the payload of the various generated packets and not to their duration. However, they are closely linked. It is the responsibility of the system integrator and the application to use correct sizes to ensure the correct video timing and behaviour, because the system may use 1 to 4 active data lanes with a fixed number of bytes for the header/checksum overhead. In the same way, when programming D-PHY time (to switch from LP to HS), the application must consider D-PHY time plus overhead due to LLP and LML crossing time (see Section 12.6.4.7.7.2, *Video stream settings (VSG)*).

VSG Control

Start and stop sequences can take a long time to be performed. If the requests are not maintained up to the time the status bit are indicating effective start or stop, they can be ignored and the VSG may not have started or stopped as expected by application. It is then up to the application to carefully manage the request and to verify that requests are being processed before changing the state of the VSG.

Test Generator:

The DSITX provides test generator to provide a video data stream. When using TVG, all sources can be used (provided VCA permits it) however the SDI interface must not be programmed to send video data and must be restricted to command mode.

The pixel modes supported are: RGB 16-, 18-, 24-, 30- and 36-bit; YCbCr422 16-bit; and YCbCr420 12-bit (note that additional YCbCr422 20-bit and YCbCr420 24-bit may be supported using SDI interface, but are restricted to command mode only).

Copyright © 2024 Texas Instruments Incorporated

The supported display sizes are listed below:

- QQVGA (160 x 120) - 15/20/30/60 fps - RGB 16-18-24 (-30 and 36) bits per pixels
- QCIF (176x144), QCIF + (176x208 and 176x220) - 15/20/30/60 fps - RGB 16-18 and 24 bits per pixels
- QVGA (320x240) - 15/20/30/60 fps - RGB R16-18 and 24 bits per pixels
- CIF (352x288), CIF + (352x416 or 352x440) - 15/20/30/60 fps - RGB 16-18 and 24 bits per pixels
- 1/2 VGA (320x480) and 2/3 VGA (640x320) - 15/20/30/60 fps - RGB 16-18 and 24 bits per pixels
- VGA (640x480) - 15/20/30/60 fps - RGB 16-18-24(-30 and 36) bits per pixels
- WVGA (800x480 - 848x480 -854x480 - 852x480) - RGB 15/20/30/60 fps - 16-18 and 24 bits per pixels
- SVGA (800x600) - 15/20/30/60 fps - RGB 16-18 and 24 bits per pixels
- QHD (960x540) - 15/20/30/60 fps - RGB 16-18 and 24 bits per pixels
- XVGA (1024x768) - 15/20/30/60 fps - 16-18 and 24 bits per pixels
- Full HD (1920x1080) - 15/20/30/60 fps - 16-18 and 24 bits per pixels
- WUXGA(1920x1200) - 15/20/30/60/(120 fps – 16 bpp only) 16, 18 and 24 bits per pixels
- 4Kx4K(4096x4096) - 15/20/30 fps - 16-18 and 24 bits per pixels

12.6.4.7.3 Start-up Procedure

The start-up sequence described in Figure 12-383 assumes that the rest of the chip is ready (all clocks are present except tx_byte_hs_clk, application it is ready...) and that the DSI is ready (reset de-asserted).

The following is the most common start up sequence:

Firstly, application layer programs and enables the PLL, then waits for a period for the PLL to lock (minimum time will be defined by the PHY documentation). At the end of the minimum wait period, the application layer should poll the PLL lock status register periodically to confirm lock status, repeating until confirmed. At the end of this step, it is assumed that a clock is present on the tx_byte_hs_clk input (this step may be replaced by an interrupt-based alternative if available from the selected DPHY or provided in the system integration logic).

During the time taken by the PLL to lock, the application can program the configuration registers and prepare the DSI link. This can also be done after the PLL is locked. It should at least configure enough to be able to use LP mode to send direct commands.

Enable clock and data lane(s) per the needs. This action is only concerned with programming registers that control the D-PHY lane enable signal. The active lane configuration with the DSI_MCTL_MAIN_PHY_CTL and DSI_MCTL_MAIN_EN registers must be programmed to match, i.e. for two data lanes (0 and1).

- DSI_MCTL_MAIN_PHY_CTL (0x08) register – LANE3/4_EN - bits[2:1] set to zero to disable lanes 2/3.
- DSI_MCTL_MAIN_EN (0x24) register – DAT3/4_EN fields - bits[6:5] set to zero to disable lanes 2/3.

Start the lane in the DCB. The DCB state machine puts the lanes in the correct state to be ready.

At this step, the DSI link should be able to access to display - at least in LP mode. If the system (DSI link and display) is fully programmed, code to display image can be started. If display characteristics must be read from the display and the system is programmed in LP mode, this is time to read the displays parameters to allow correct programming of the DSI. A new round of programming can then take place to set-up the system, based on the parameters which have been read fro the display.

The DSI internal setting is now completed and it is ready to accept data. The application can now enable the interface so that the stall signal is removed and that DSI link can accept data.

When the procedure described above is terminated, it does not imply that display itself is ready. At this stage, the D-PHY and DSI link are ready to send/receive data to/from display and thus application may still need to initialize the display itself.

Copyright © 2024 Texas Instruments Incorporated

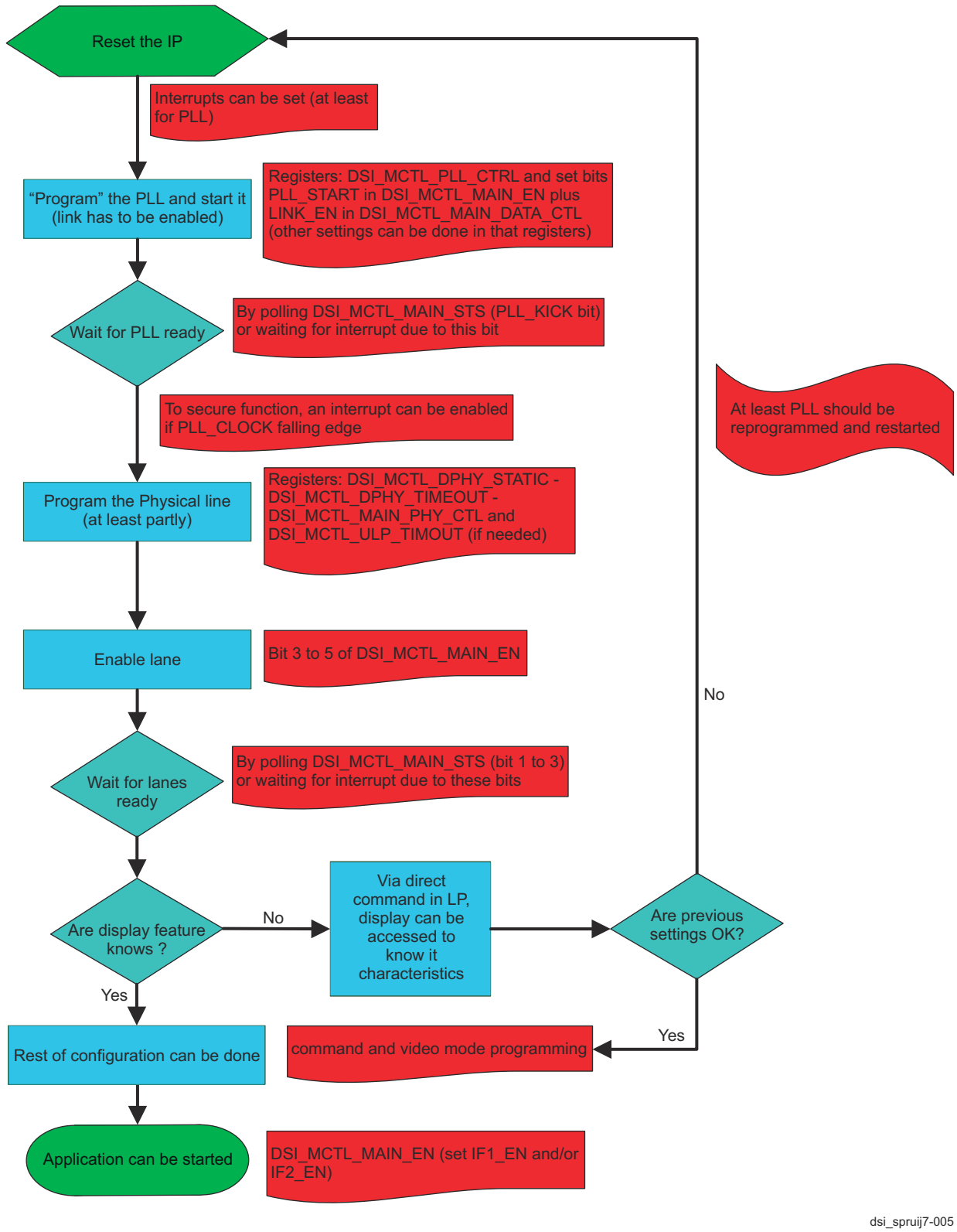


Figure 12-383. Start-up Procedure Summary

12.6.4.7.4 Interrupt Management

There are numerous error and status conditions that can be monitored via the interrupt mechanism. The edge on which these conditions are generated is programmable.

Copyright © 2024 Texas Instruments Incorporated

Each interrupt source has four associated registers:

1. The status register itself "<sts_reg>"
2. a control register "<sts_reg>_ctl" to control corresponding interrupt behaviour
3. a flag register "<sts_reg>_flag" to observe when interrupt/event has been triggered
4. A clear register "<sts_reg>_clr" to clear the event flag register.

The decision of which status bits (<sts_bit>) can generate interrupt is taken when the corresponding enable bit "<sts_bit>_en" is set in the status control register. In the same register the status bit edge "<sts_bit>_edge" states on which edge of the status bit the interrupt is triggered. When the selected edge is observed, the flag bit "<sts_bit>_flag" is set. The interrupt signal is an OR of all flag bits that are enabled. When the register "<sts_reg>_clr "is written with the bit "<sts_bit>_clr ", the " <sts_bit>_flag" is cleared.

12.6.4.7.4.1 Error and Status Registers

The error register bank handles storing all error flags that are identified. It also stores some status bits that must be observable and detectable by the application. These status and error registers can generate an interrupt on the rising or falling edge if this generation is enabled.

This mechanism is different depending on the nature of the status or error bits. For those that are generated in the DSI block (basically all status and error bits except direct command), the status bits cannot be easily controlled and toggle according to the internal status meaning only the current value is observed in the status/error register itself.

If the interrupt generation is enabled with the associated enable bit, a rising edge (associated edge bit set to 0) or a falling edge (associated edge bit set to 1) of that status bit toggles the corresponding flag bit. At the end, all the flag bits are put together with an OR to generate the interrupt signal. The flag register can be reset by writing in the clear register.

The described behavior can be summarized by the following pseudo-HDL. The signal named signal_sts_bit is the bit that is observable by reading status bit.

- reg_sts_q is the status register (accessible via APB)
- reg_flag_sts_q is the flag register
- clear_the_flag is set when the register clear is written
- edge_sts_ctl is the edge bit
- enable_sts_ctl is the enable bit

reg_sts_d <= **signal_sts_bit**;

reg_flag_sts_d <= '0' WHEN clear_the_flag = '1' ELSE
– 1' WHEN (reg_sts_q = '0' AND reg_sts_d = '1' AND edge_sts_ctl = '0') ELSE – detect rising
– '1' WHEN (reg_sts_q = '1' AND reg_sts_d = '0' AND edge_sts_ctl = '1') ELSE – detect falling
– reg_flag_sts_q;

irq_n <= NOT (OR_OF_ALL(reg_flag_sts_q AND enable_sts_ctl));

The code is slightly different in cases where the observed bit is a generated condition (in the control block), as is the case for the direct command status and error flags. The status/error information is automatically generated but is cleared only when writing in the corresponding clear bit. The interrupt generation behaves similarly to the error condition cases, however, as the status falling edge is observed only when the bit is cleared, it is not possible to use the falling edge detection on flag (however the code is kept as-is to provide a standard implementation method). This leads to the following code:

- reg_sts_d <= '0' WHEN clear_the_flag = '1' ELSE
  – '1' WHEN the_sts_bit (Note 1) = '1' AND reg_sts_q = '0' ELSE -- (see Note)
  – reg_sts_q;
- reg_flag_sts_d <= '0' WHEN clear_the_flag = '1' ELSE
  – '1' WHEN (reg_sts_q = '0' AND reg_sts_d = '1' AND edge_sts_ctl = '0') ELSE – detect rising
  – '1' WHEN (reg_sts_q = '1' AND reg_sts_d = '0' AND edge_sts_ctl = '1') ELSE – detect falling
  – reg_flag_sts_q;
- int <= NOT(OR_OF_ALL(reg_flag_sts_q AND enable_sts_ctl));

Copyright © 2024 Texas Instruments Incorporated

Note: In some case (pulse generated), the rising edge detection is not done and the code becomes simply '1' WHEN signal_sts_bit = '1' ELSE...

12.6.4.7.4.2 Interrupt Management for Direct Command Registers

The following direct command status bits/registers are only set when error is generated and only cleared when the clear bit is written:

- all the status bits of the register DSI_DIRECT_CMD_RD_STS
- all the status bits of the register DSI_DIRECT_CMD_STS
- all the status bits except bit 0 (cmd_transmission) of the register DSI_DIRECT_CMD_STS

As these registers are reset only when the clear register is written, it is not possible to detect the falling edge on them to generate interrupts. Only the rising edges can generate the interrupt.

Note: There can be issues with detection when using the bits for all the signals that are a pulse generated in the tx_byte_hs_clk domain. When the speed of that clock is slower in relation to dsi_p_clk, it is possible to have the interrupt set, the bit read and the clear attempted before the end of the tx_byte_hs_clk pulse, after the clear, the bit is set again.

12.6.4.7.5 Direct Command Usage

The direct command mechanism is a way to send commands (with a maximum size of direct_cmd_fifodepth bytes) by writing and reading registers. It allows commands to be sent that are not allowed through the SDI-DSI interface (DSI commands, triggers), to perform read operations or to send DCS / generic write / generic read operations when the DSI link is not fully programmed and cannot use the SDI interface for commands.

Figure 12-384 shows direct command management sequence.

Basically, the procedure to do so is as follows:

- Clear the status bits (when reading the status bits, only the DSI reads the current access status).
- Program the various registers that define the command to send.
- Write the direct_cmd_send register. Writing in that register (any value) triggers the command sending mechanism.

  The command should be started only if the command request is amongst the defined ones for DCS or Generic Write or Read with 0, 1, or more parameters:
  - DCS Write codes 0x05, 0x15, 0x39
  - DCS Read codes 0x06
  - Generic Write codes 0x03, 0x13, 0x23, 0x29
  - Generic Read codes 0x04, 0x14, 0x24

  Note: The direct_cmd_main_settings register can use other data type values for cmd_head to support future types and no check is performed for invalid types.
- Wait for the assertion of the command_completed bit in the direct_cmd_sts register (or any of the relevant bits that indicate a specific command termination). Instead of polling the registers, it is possible to enable the corresponding interrupts and wait for its detection.
- Read the received data and (if any) the different status bits.

Copyright © 2024 Texas Instruments Incorporated

Figure 12-384. Direct Command Management

There are several checks performed when sending a direct command to verify that requested command sizes are supported and to check that the requested command exists (otherwise the request to send the command is ignored).

- For Short packets commands the DCS and Generic Read and Write the data size is rounded to maximum of 2.
- For Long packet DCS and Generic Read and Write the data size is rounded to the maximum size of the FIFO.
- No checks are made for the write FIFO having the expected number of bytes programmed into the CMD_SIZE in the dir_cmd_main register. In this case the data written out will loop back through the value in the FIFO.

At completion of a read command, registers can be read (they contain read data and/or errors detected during the communication) and trigger events are sent to the application to announce the request completion (using the status and error register bank).

When a read request is performed, the read data is put in the RP FIFO. If more bytes are sent, they are ignored and errors are set accordingly. The received read can then be accessed by reading several times the register that gives an access to the FIFO.

Copyright © 2024 Texas Instruments Incorporated

For every new command, several status bits are present: one signals that a command is being transmitted (cmd_transmission). When the command is complete, several bits can be toggled to signal it (write_completed, trigger_completed, te_received, read_completed and read_completed_with_err).

Some complex commands provide intermediate information. For BTA requests, there is a bit that signals when the BTA request has been sent (BTA_completed) and another one that signals when the DSI is master of the D-PHY interface again (BTA_finished). For commands that imply a BTA, some other bits specify whether specific data has been received (such as trigger, acknowledge with or without error).

12.6.4.7.5.1 Trigger Mapping Information

Triggers are one of the groups of direct commands that can be sent and received using the DPHY in Low Power escape mode. The trigger_val[3:0] bits (in the registers DIRECT_CMD_MAIN_SETTINGS and DIRECT_CMD_STS) are used to define which of the four possible trigger entry codes has to be sent or has been received. The register fields described above are copied in signal ppi_d1_tx_trigger_esc[3:0] (for trigger_val[3:0] of DIRECT_CMD_MAIN_SETTINGS) or are a copy of ppi_d1_rx_trigger_esc[3:0] (for trigger_val[3:0] of DIRECT_CMD_STS). Only one bit out of the four should be set else the D-PHY behavior is not very precisely defined. Table 12-363 gives the trigger mapping in the system.

Table 12-363. Trigger Mapping

Trigger name	Trigger entry code	trigger_val	meaning in TX direction	meaning in RX direction
trigger 0 - reset	01100010	1b0001	Reset	Not affected by DSI spec
trigger 1 - unknown 3	01011101	1b0010	Not affected by DSI spec	TE response(1)
trigger 2- unknown 4	00100001	1b0100	Not affected by DSI spec	Acknowledge with no error(1)
trigger 3 - unknown 5	10100000	1b1000	Not affected by DSI spec	Not affected by DSI spec

(1) These triggers are not observed in the trigger_val field of the direct_cmd_sts register because they are used by the DSI link for a specific purpose.

Trigger Reset – This will request from the display and expects an immediate reset of the DSI command/video mode with all pending requests in TX sending path FIFO for transfer to be discarded.

Acknowledge – This is a response to DSITX controller. The host processor may request a command acknowledge and error information related to any transmission by asserting Bus Turnaround with the transmission. The peripheral shall respond with ACK Trigger Message if there are no errors and with Acknowledge and Error Report packet if any errors were detected in previous transmissions. Appropriate flags shall be set to indicate what errors were detected on the preceding transmissions.

If the transmission was a Read request, the peripheral shall return READ data without issuing additional ACK Trigger Message or an Acknowledge and Error Report packet if no errors were detected. If there was an error in the Read request, the peripheral shall return the appropriate Acknowledge and Error Report unless the error was a single-bit correctable error. In that case, the peripheral shall return the requested READ data packet followed by Acknowledge and Error Report packet with appropriate error bits set.

Tearing Effect - The Tearing effect on the display is avoided by having synchronization information from the display. It is used only in command mode. Users are responsible for selecting the command mode for the VC using the TE feature and selecting the polling or automatic mechanism.

The user is responsible for ensuring that there is a delay after a trigger is issued by the host to the panel, before any other action, message or trigger is performed. When the host issues a reset trigger, the system will expect the controller to stop and re initialise the clock and data links. All other triggers issued by the host will be application specific and are outside the scope of this document.

12.6.4.7.5.2 Command Mode Settings

The command mode is enabled on the SDI interface. This is controlled using the register mctl_main_en. Other settings can be set using the registers: mctl_main_data_ctrl (to decide about TE usage, read enable...); and cmd_mode_ctl (Virtual Channel of the command packets, arbitration between requests of the two interfaces, possibility to use LPDT, TE timer programming and padding value in case of an error).

Copyright © 2024 Texas Instruments Incorporated

When programming Direct READ commands, a BTA request is sent automatically following the read transmission, for the peripheral to respond – it is therefore not necessary to explicitly send a BTA request at that time. The system must allow the BTA request to be completed and the bus returned to the host before issuing any new read or write command. The system must either: poll the read_completed (and read_completed_with_err) status bits; or, wait for an interrupt caused by these bits before issuing any new command.

When programming Direct WRITE commands, it is advisable to ensure that each transaction can complete successfully before moving on to a subsequent command. There are two recommended methods to achieve this, either: explicitly request a BTA between write transactions while checking for the associated interrupt or polling the appropriate status flag (this method also provides the added security of an ACK response from the peripheral); or allowing a gap between commands sufficient to allow completion of the first before commencing with the second. The recommended approximate gap time required to ensure that all commands complete successfully is equivalent to 100 TX ESC byte periods. If required (for instance, to minimize the start-up time), it is also possible to calculate the minimum delay time required for each transaction, based on the TX escape clock frequency used in the application and the length of each command (no active video transmission should be enabled at the time – a typical use case would be peripheral parameter configuration). As with read operations, any write operation followed by an explicit BTA request must allow the BTA to be completed and the bus returned to the host before issuing any new read or write command.

For single parameter commands, the upper bytes of the 32-bit value written to the Direct Command write data register (direct_cmd_wrdat) should be masked to zero to comply with the DSI specification requirement that unused parameter locations are set to zero. For zero parameter writes, assuming there have been previous write commands sent, it is necessary to clear the sending path FIFO by writing to the direct_cmd_fifo_rst register to clear the data path, then write a zero value to the direct_cmd_wrdat register to send the zero parameter write command.

Note: Failure to follow these procedures may result in non-zero data in the unused parameter locations; this may or may not affect the peripheral, however it is recommended to use the process described to ensure compliance with the DSI specification in this regard.

If STOP mode is forced during a command mode read or write transaction, all outstanding transactions should be considered lost and all status and error flags should be disregarded and cleared. Similarly, if an LPDT read data operation is stopped by writing to the STOP_READ_OPERATION field of the DIRECT_CMD_RD_INIT register, the status for the aborted read operation should be disregarded and cleared before resuming normal operation.

12.6.4.7.5.3 Bus Turnaround Sequence

The command operation can request a read or an acknowledge response from the panel after a long sequence of packets is sent. The Bus turnaround is also used to control the Tearing Effect control.

When a Bus Turnaround is requested from the command stream manager, the DPHY will be given a turn request. The request will make the DPHY interface perform a sequence of LP states (LP11-10-00-10-00) to indicate that it is releasing control of the DP/DN signals. The DPHY Direction signal will change once it detects the Panel side driving the LP00-10-11 states. The panel will then be able to send any response packets (see section 0) and release the bus by issuing a bus turnaround sequence to the controller DPHY. The DSITX controller DPHY will detect the sequence and change the direction signal once the pattern is sent and LP00 state is valid for 2-3 TX_ESC_CLK cycles.

Figure 12-385 shows Bus turnaround timing sequence from controller to panel and panel to controller.

Copyright © 2024 Texas Instruments Incorporated

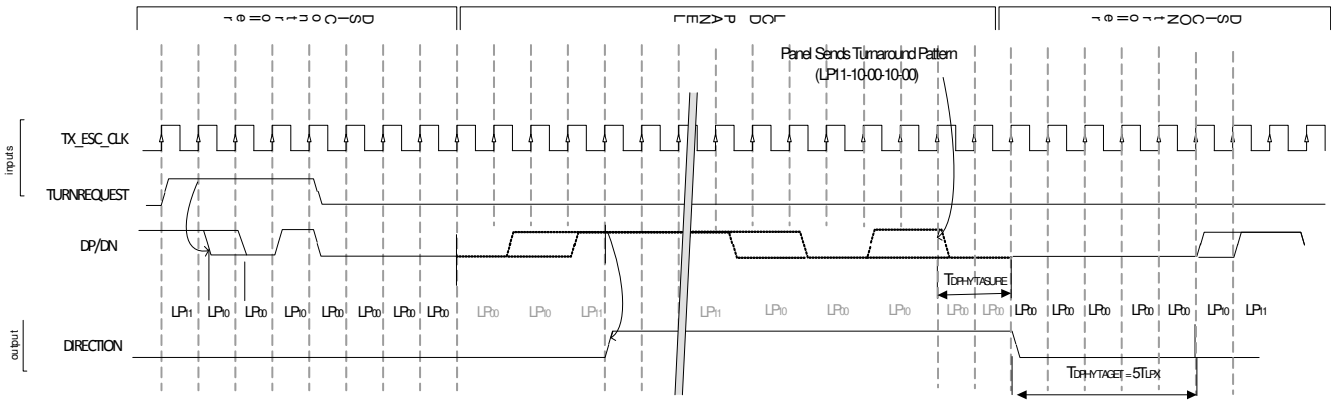


Figure 12-385. Bus Turnaround Timing Sequence from Controller to Panel and Panel to Controller

12.6.4.7.5.4 Tearing Effect Control

The DSI command control for a tearing effect request can be performed using the polling method or automatically. The enable bits for BTA and TE must be enabled for the mechanism to be performed with either the SDI or DIRCMD interface. For automatic operation, the DSI will send a first BTA, and then waits for BTA reception and if no TE has been received during the first BTA [(1)] , it sends another BTA and then waits for TE. There are three possible error conditions that can happen (but only the two first are detected by the DSITX controller and passed to registers):

- If the display answers to the second BTA with another BTA and thus does not sent the TE - this means that the display does not support TE generation OR that TE generation mechanism has not been enabled (reg_err_no_te).
- If the TE has not arrived within a given programmable period – this means that the TE request arrived too late and that the Display Application Processor is not synchronized with the display. The error reported is reg_err_te_miss but the system continues to wait until the TE is received. [(2)]
- If a time-out has been detected by the DSI showing that a problem happened during the BTA and that the system is forced back to idle without BTA – in this case, a BTA without TE is seen and the CSM generates a reg_err_no_te (as for first case).

For the polling method to request a tearing effect (TE polling mode is enabled with te_hw_polling_en = 1) the DSI sends a first BTA, waits for peripheral TE + BTA, if only BTA has been received after the first BTA, it keeps sending BTAs until peripheral responds to the BTA by a TE + BTA instead of BTA only; in this specific mode reg_err_no_te will never be asserted. To stop sending BTA and waiting for TE a force stop [(3)]) is needed.

Note 1: TE received is checked on the first BTA in case a BTA was the last command issued by the DSI link (BTA or read command sent via direct command interface).

Note 2: Again to support the cases where a BTA was emitted just before the TE request, the TE windows is counted on the first BTA (in case the display understands this BTA as the TE request and thus may emits too late the TE), and is restarted after the second BTA.

Note 3: The time to wait before forcing a stop should be around the duration of a frame. The reason is that if you miss the TE time at the end of the current frame, you've to wait till the end of the next frame before new TE can be sent by display.

The following table describes the TE timeout counter operation, Programming of the IP. The counter value should be calculated based on the tx_byte_clk period.

Table 12-364. TE Timeout Programming

te_timeout(11)	te_timeout(10)	timeout value
0	0	256 × te_timeout<9:0>
0	1	512 × te_timeout<9:0>
1	0	1024 × te_timeout<9:0>
1	1	2048 × te_timeout<9:0>

Copyright © 2024 Texas Instruments Incorporated

12.6.4.7.5.5 Tearing Effect Control on Panels with Frame Buffer

Display panels will use command mode sequencing when an on-board frame buffer is used to store the video image rather than using the video streaming and synchronisation packets to control the update of the image information. This means that the host display driver must know when the panel is processing the image and avoid changing pixel information before it is updated. The mechanism is known as the tearing effect. The DSITX controller can support two schemes as outlined in the MIPI DSI specification.

TE control using the Polling Method

For polling to the display module, the host processor shall detect the current scan line information with a DCS command such as **get_scan_line** to avoid Tearing Effects. DSITX controller will issue a READ using this command until it gets to the expected value for the last line.

DCS Command - get_scanline (45h)

Command

Direction H->D

Hex Code 0 1 0 0 0 1 0 1 45h

Parameter 1

Direction D->H

Hex Code N15 N14 N13 N12 N11 N10 N9 N8 XXh

Parameter 2

Direction D->H

Hex Code N7 N6 N5 N4 N3 N2 N1 N0 XXh

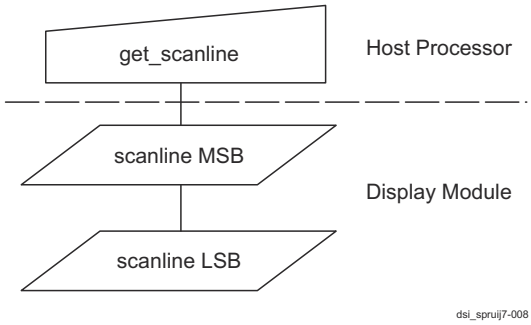


Figure 12-386. get_scan_line Command

Description

The display module returns the current scanline, N, used to update the display device. The total number of scanlines on a display device is defined as VSYNC + VBP + VACT + VFP. The first scanline is defined as the first line of V Sync and is denoted as Line 0.

In Sleep Mode, the value returned by get_scanline is undefined. See [MIPI-DPI] for definitions of VSYNC, VBP, VACT, and VFP.

- In 2D mode, the scanline value of the display memory and the display panel is the same.
- In 3D Mode, the scanline value of the display memory and the display panel can be different; get_scanline shall return the current scanline of the display panel.

Restrictions

None

TE control using the Automatic Method

Copyright © 2024 Texas Instruments Incorporated

For TE-reporting from the display module, the TE-reporting function is enabled and disabled by three DCS commands to the display module controller: set_tear_on, set_tear_scanline, and set_tear_off. See [MIPI-DCS] for details.

set_tear_on and set_tear_scanline are sent to the display module as DSI Data Type 0x15 (DCS Short Write, one parameter) and DSI Data Type 0x39 (DCS Long Write/write_LUT), respectively. The host processor ends the transmission with Bus Turn-Around asserted, giving bus possession to the display module.

Since the display module DSI Protocol layer does not interpret DCS commands, but only passes them through to the display controller, it responds with a normal Acknowledge and returns bus possession to the host processor. In this state, the display module cannot report TE events to the host processor since it does not have bus possession.

To enable TE-reporting, the host processor shall give bus possession to the display module without an accompanying DSI command transmission after TE reporting has been enabled. This is accomplished by the host processor protocol logic asserting (internal) Bus Turn-Around signal to its D-PHY functional block. The PHY layer will then initiate a Bus Turn-Around sequence in LP mode, which gives bus possession to the display module.

Since the timing of a TE event is unknown to the host processor, the host processor shall give bus possession to the display module and then wait for up to one video frame period for the TE response. During this time, the host processor cannot send new commands, or requests to the display module, because it does not have bus possession.

DCS Command - set_tear_on (35h)

Command

Direction H->D

Hex Code 35h

Parameter

Direction H->D

Hex Code X X X X X X X M XXh

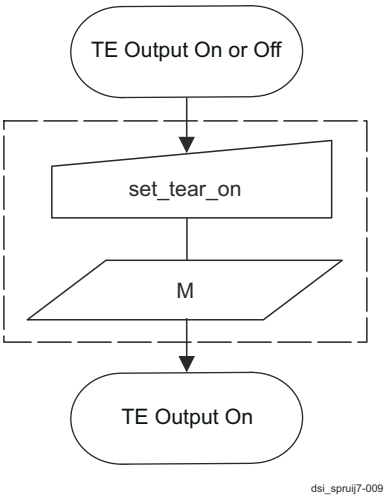


Figure 12-387. set_tear_on Command - 1

Description

This command turns on the display module Tearing Effect output signal on the TE signal line. The TE signal is not affected by changing set_address_mode bit B4. set_tear_on has one parameter that describes the Tearing Effect Output Line mode.

Copyright © 2024 Texas Instruments Incorporated

If M = 0 (Mode 0):

The Tearing Effect Output line consists of V-Blanking information only.

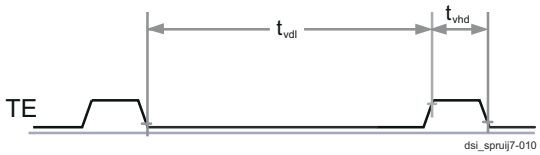


Figure 12-388. set_tear_on Command - 2

If M = 1 (Mode 1):

The Tearing Effect Output Line consists of both V-Blanking and H-Blanking information.

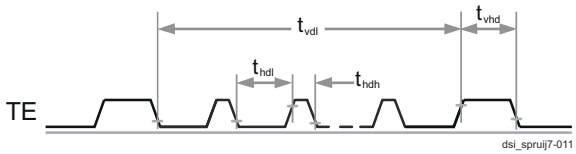


Figure 12-389. set_tear_on Command - 3

The Tearing Effect Output line shall be active low when the display module is in Sleep mode. See [MIPI- DPI] for definitions of tvdl, tvdh, thdl and thdh.

Restrictions

This command takes effect on the frame following the current frame. Therefore, if the Tearing Effect (TE) output is already ON, the TE output shall continue to operate as programmed by the previous set_tear_on, or set_tear_scanline, command until the end of the frame.

DCS Command - set_tear_scanline (44h)

Command

Direction H->D

Hex Code 0 1 0 0 0 1 0 0 44h

Parameter 1

Direction H->D

Hex Code N15 N14 N13 N12 N11 N10 N9 N8 XXh

Parameter 2

Direction H->D

Hex Code N7 N6 N5 N4 N3 N2 N1 N0 XXh

Copyright © 2024 Texas Instruments Incorporated

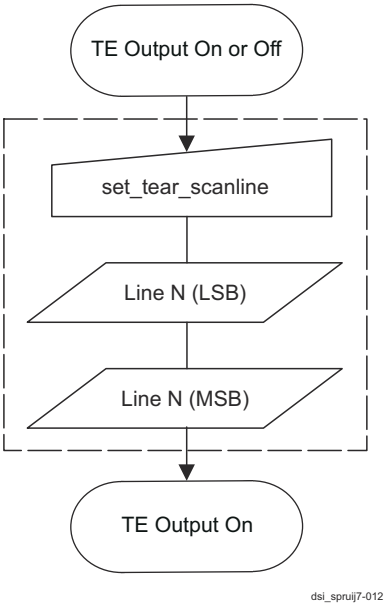


Figure 12-390. set_tear_scanline Command - 1

Description

This command turns on the display module Tearing Effect output signal on the TE signal line when the display module reaches line N. The TE signal is not affected by changing set_address_mode bit B4.

The Tearing Effect Line On has one parameter that describes the Tearing Effect Output Line mode.

After issuing a set_tear_scanline command to the display module, the Tearing Effect output signal, e.g. as in DBI- 2 systems, shall be a delayed version of V-Blanking information as illustrated in Figure 12-391.

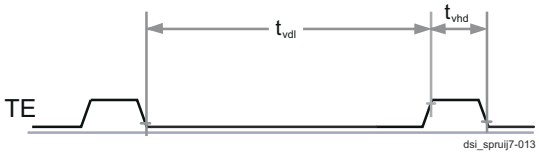


Figure 12-391. set_tear_scanline Command - 2

Note that set_tear_scanline with N = 0 is equivalent to set_tear_on with M = 0. The Tearing Effect Output line shall be active low when the display module is in Sleep mode. See [MIPI-DBI] for definitions of tvdl and tvdh and [MIPI-DSI] for definition of display module line numbers. In 2D mode, the scanline value of the display memory and the display panel is the same.

Restrictions

This command takes effect on the frame following the current frame. Therefore, if the Tear Effect (TE) output is already ON, the TE output shall continue to operate as programmed by the previous set_tear_on, or set_tear_scanline, command until the end of the frame.

DCS Command - set_tear_off (34h)

Command

Direction H->D

Hex Code 0 0 1 1 0 1 0 0 34h

Parameters None

Description

This command turns off the display module Tearing Effect output signal on the TE signal line.

Copyright © 2024 Texas Instruments Incorporated

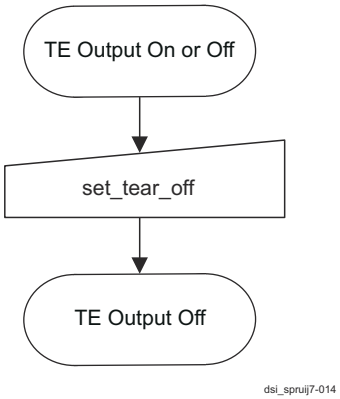


Figure 12-392. set_tear_off Command

Restrictions

This command has no effect when the Tearing Effect output is already off.

12.6.4.7.5.6 Return Path Operation

The DSI Return Path behaves like a slave, by collecting data from the DPHY receive interface when the DPHY is configured to accept a transmission from the peripheral. The RP receives 6 signals from the DPHY (see Table 12-365).

The data (and trigger) reception is inactive if the DPHY PPI direction signal is at low level. When it is active, data is received using the RX ESC_CLK clock (the clock is transmitted with the data itself - see D-PHY protocol description from MIPI) and may stop if there is a pause in the data transmission but data is generated on clock falling edge. New data is available on rx_data_esc on a rising edge of the clock, if both rx_lpdt_esc and rx_valid_esc are asserted.

Table 12-365. Interface Between Return Path and DPHY

Signal Name	I/O	From/To	Description
rx_lpdt_esc	I	DPHY	Escape Low Power data receive mode This active high signal is asserted to indicate that the lane module is in low-power data receive mode. While in this mode, received data bytes are driven onto the rx_data_esc output when rx_valid_esc is active. The lane module remains in this mode with rx_lpdt_esc asserted until a stop state is detected on the lane interconnect.
rx_trigger_esc(3:0)	I	DPHY	Escape trigger received These signals indicate that an escape trigger command has been received. Only one of these signals will be asserted at any time and the signal will remain asserted until the lane module returns to stop state.
rx_data_esc(7:0)	I	DPHY	Escape receive data For Low Power data receptions, this eight-bit value is driven by the D-PHY and is valid on rising edges of rx_clk_esc with rx_valid_esc asserted. The bit connected to rx_data_esc[0] was received first.
rx_valid_esc	I	DPHY	Escape receive data valid This signal indicates that the lane module is driving data on rx_data_esc and expects the protocol layers to take the data at the current rising edge of rx_clk_esc. There is no "ready" signal to throttle the receive data.
stop_state_dl1	I	DPHY	Lane is in stop state This signal indicates that the lane module is in STOP state. Note that this signal is asynchronous to any clock in the protocol interface.
direction	I	DPHY	Lane direction. When this signal is high, the lane module is in receive mode. When direction is low, the lane module is in transmit mode. (Mnemonic: 1 = Input, 0 = Output.)

Copyright © 2024 Texas Instruments Incorporated

The received data messages are identified as one of two possible types:

- Trigger
- Read packet

Trigger messages are passed almost directly to the register (reg_req = reg_trigger = 1 and reg_rd_data<3:0> equals to trigger value during one clock cycle). However, triggers are decoded to see if they are a TE trigger or an acknowledge with no error.

- The TE data is passed separately to the CSM using csm_te_received pin. It needs to be resynchronized with tx_byte_clk byte clock because of the CSM working clock domain.
- The acknowledge in only passed to register using reg_req and reg_ack pins. All other trigger values are passed undecoded to the registers without any processing. It is the responsibility of the application to decode and decide what to do with them.

Read Packet: When data is received, the system waits for the 4 bytes of the **HEADER**, and performs an ECC correction (if it is enabled) and then decodes the packets.

The following cases are considered when the system performs the ECC check:

- If the received command is not in the list of legal display opcodes, (i.e. errors in the packet header), it discards that packet and all further incoming bytes, even if they are legal packets (until the next BTA - i.e. direction change). It sends errors to the register bank (err_undecodable and uncorrectable_err). This may also result in an EOT error being reported as no further bytes have been decoded. But even if ECC correction shows an uncorrected error, decoding is attempted (if the errors are in the ECC byte or in the payload bytes, the opcode can be understood and data correctly recovered). In this case only err_uncorrectable error bit is sent to register and it falls in the situation of regular commands (acknowledge with error or read).
- If "an acknowledge with error", the two status bytes are sent to the registers (reg_req = 1 and reg_ack_err = 1 and reg_rd_data<15:0> contains the correct value).
- If the message type is a short read (either DCS or generic), data is passed to the registers (2 bytes without any specific decoding). Interface signals must be set accordingly and the two bytes are sent to registers (reg_req = reg_start = reg_end = reg_read = 1 with data and dscnotgen set accordingly). Size information is decoded in the first byte of the received packet depending on the data type information and then sent to register using reg_size port.
- If a long read, the size is sent to the register using reg_size. The system goes to **LONG state**. All data received are sent to register using reg_data and are used for checksum detection. If during receive, return packet fifo (in control block) becomes full (rd_data_fifo_full), the error flag err_oversize must be set. All remaining data is used for checksum calculation but are not transferred to registers. When the number of received bytes is equal to the packet size, the system will check the **CHECKSUM**. The two checksum bytes are never transferred to register. They are used to make checksum detection and indicate err_checkum status.
- If the transfer is shorter than the size specified in the header, the error err_wrong_length is issued along with any subsequent errors such as err_missing_eot (if needed). The case where it is longer drives to a tentative checksum decoding that generates an error and by a tentative to decode a new header (that is in fact a part of the previous packet) that drives to an error err_undecodable and the throwing of the rest of the received bytes (with corresponding errors such as err_missing_eot).
- If it is an EoT packet, it is not passed to the control block. The following data is ignored (if any). If the system is waiting for **EoT**, this is the only action. If system is not waiting for EoT packet, the err_eot_with_err is sent.
- If after an "acknowledge with error", the system waits for EoT packet and counts 4 bytes but is not able to decode them, it emits an err_eot_with_err and err_undecodable.
- If direction is removed at a "wrong" time (i.e before the fourth byte of a small packet, the system stops to works but emit errors to the register: err_receive with eventually err_missing_eot.

The DSI is designed to support multi-peripheral integration, the virtual channel of the received packet is also stored in register using reg_vc port.

The application FW layer is responsible of all read actions, it is assumed that once a read is requested, application reads return packets before requesting other read actions. Based on this assumption, the receive FIFO implemented in the register block is cleared each time a read action is started. All data not read before the new read command are lost.

Copyright © 2024 Texas Instruments Incorporated

A re-initialization of the return path is also performed using register access on reg_init_rp.

All errors related to short or long received data packets should be captured in the register when the direction changes (to capture all errors at the same time and not generate several errors and possibly several interrupts on the same reverse transaction). As soon as an error is detected on a packet, the error must be recorded and maintained up to the end of the packet transfer.

The maximum packet size supported by the DSI return path is 16 bytes. If a longer packet is received, only the first 16 bytes are available (by register way) for application.

12.6.4.7.5.7 EoT Packet Management

The DSI protocol states that, in the sending path, only the HS transmission requires use of EoT packet. For transfer from display to host, it is not recommended to use EoT but it is possible. The host needs to be able to detect the EoT packet and then behaves accordingly.

In case the display makes use of the EoT packet, the bit disp_eot_gen must be set to one. In that case, if after the ECC correction, the packet header is 0x08, we've detected an EoT packet and all bytes that may arrive between this packet and the direction change must be ignored.

If additional data is received between the EOT packet and the next BTA (end of read process), or if an unwanted EOT packet is detected, the unwanted bytes must be rejected and err_receive is asserted.

If the EoT packet is not detected although it should be present, the error err_missing_eot is set. If after an "acknowledge with error", 4 bytes are received before BTA but are not identified as EoT (because of errors), it is assumed these 4 bytes are the EoT. In that case, the error err_eot_with_err has to be set with err_undecodable.

12.6.4.7.5.8 ECC Correction

The DSITX controller uses the packet header ECC information to detect and correct one error, and can detect two or more errors without correction.

12.6.4.7.5.9 LP Transmission and BTA

The DSI host may send a command sequence and then request an acknowledge message from the panel. The host will send the command and release the bus by performing a BTA sequence. The panel will then respond with an acknowledge message (or error response) followed by a BTA to return the bus to the host. This sequence is illustrated in Overview of a Display Subsystem.

Overview of a Display Subsystem.

Figure 12-393 shows LP transmission timing diagram.

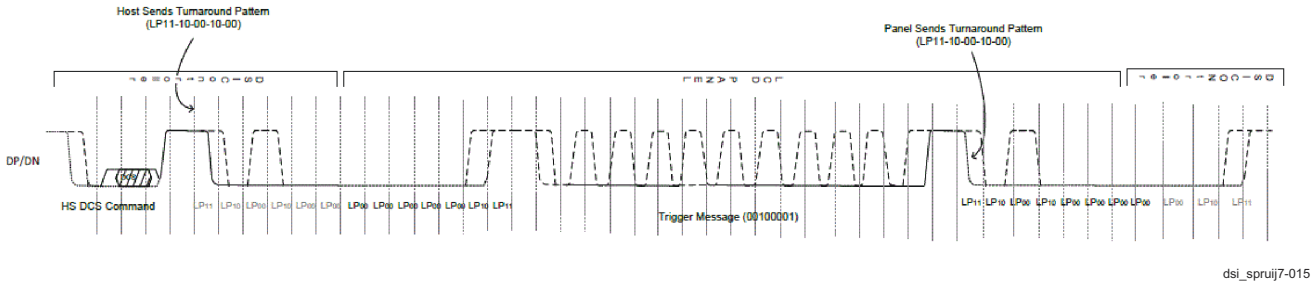


Figure 12-393. LP Transmission Timing Diagram for Host Read and Trigger ACK

The Host will not be able to transmit any other packets once the BTA is sent to the panel and will expect either an ACK trigger or response LPDT message with a payload. The system must ensure there is sufficient time to send any command and issue the BTA, receive the response (Trigger, Short or long packet) and accept the BTA from the panel, if this request is made in the line blanking intervals.

Figure 12-394 presents panel read response sequences.

Copyright © 2024 Texas Instruments Incorporated

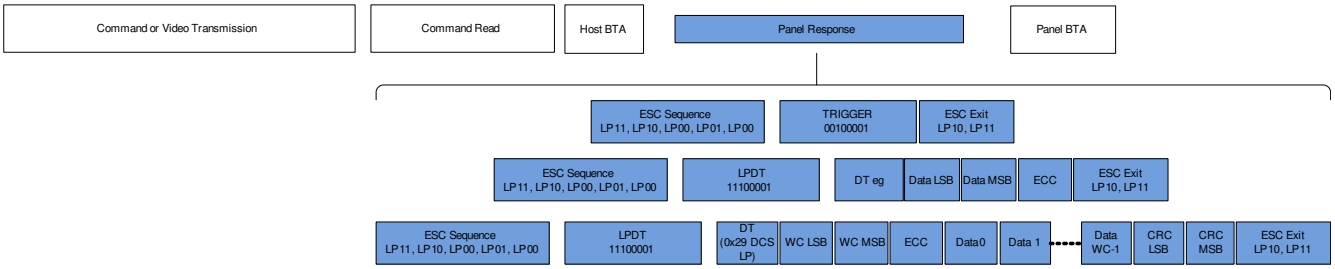


Figure 12-394. Panel Read Response Sequences

The panel response time will be based on the tx_esc_clk used inside the panel and the expect number of bytes for the payload. The expected reposes are illustrated in the blocks in the MIPI DSI standard.

12.6.4.7.6 Low-power Management

There are several low-power scenarios which consist of switching off various parts of the design:

1. Switch D-PHY cells in ULP mode when there are is no data to be sent (bits clklane_ulpm_en, dat1_ulpm_en, dat2_ulpm_en, dat3_ulpm_en and dat4_ulpm_en in register mctl_main_en set to one with ULP mode enabled in register mctl_main_phy_ctl). Please, note that the time to leave ULP state is 1 ms and needs to be set in clock cycles in the register mctl_ulpout_time.
2. Switch the DPHY byte/bit clock PLL off (refer to DPHY documentation for control sequence). In this case, the D-PHY cell can only transmit data in LPDT thus maximum bandwidth is 10 Mbits/s).
3. More drastic methods are to shut-down some parts of the design. The D-PHY and the DSI themselves can be shut down.

When D-PHY and DSI are both shut-down and the display is powered down, the restart is a normal start procedure as described in Section 12.6.4.7.3, *Start-up procedure*.

When only the D-PHY is stopped, it is placed in either the stop or ULP state. Restarting the D-PHY is similar to the regular start-up procedure, however, after the lanes are enabled by the DSI Controller (clklane_en, dat1_en, dat2_en, dat3_en and dat4_en in register mctl_main_en are set), the D-PHY generates a rising edge on the stopstate signal to indicate to the DSI Controller that the enable request has been effective within the D-PHY. After that the start-up procedure may continue up to the point that the DSI link is ready to begin transmission.

Note that when setting the D-PHY into the ULP state, the application needs to wait enough time between the time it sets the ULP request in register and the time the D-PHY is disabled – this ensures that the ULP request is detected on the D-PHY interface.

12.6.4.7.7 Video Mode Settings

All values contained in the Control Block (CB) registers must be set with meaningful values and are expressed with a range of different units: clock cycles, bytes, number of lines.

12.6.4.7.7.1 Video Stream Presentation

In Figure 12-395, it shows the list of packets that can be part of a video stream and the associated register fields. In the four following figures (Figure 12-396, Figure 12-397, Figure 12-398, Figure 12-399), different video streams that can be generated are presented.

Copyright © 2024 Texas Instruments Incorporated

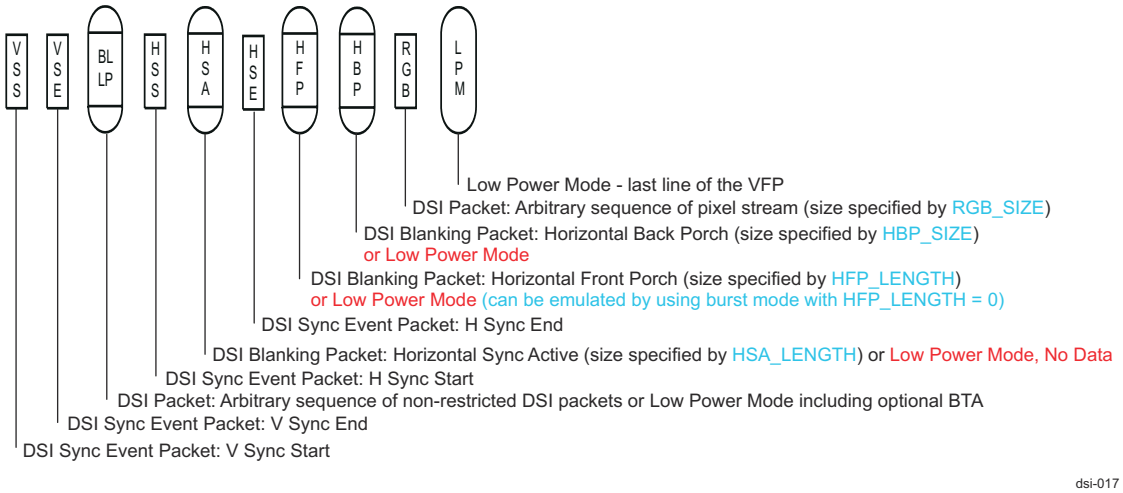


Figure 12-395. DSI Packet Terminology

Red text - behaviour not possible.

Blue text - register fields used to specify the packet size.

The BLLP (Blanking – Low Power) periods allow the DPHY link to perform the following sequences:

- Host Transmit blanking packets in HS.
- Host Transmit command packets in HS or LP (Escape mode).
- Host Transmit packets using interleaving to a different virtual Channel in HS or LP (Escape mode).
- Host performs BTA and waits for response packets from the panel using Escape Mode followed by a panel BTA.

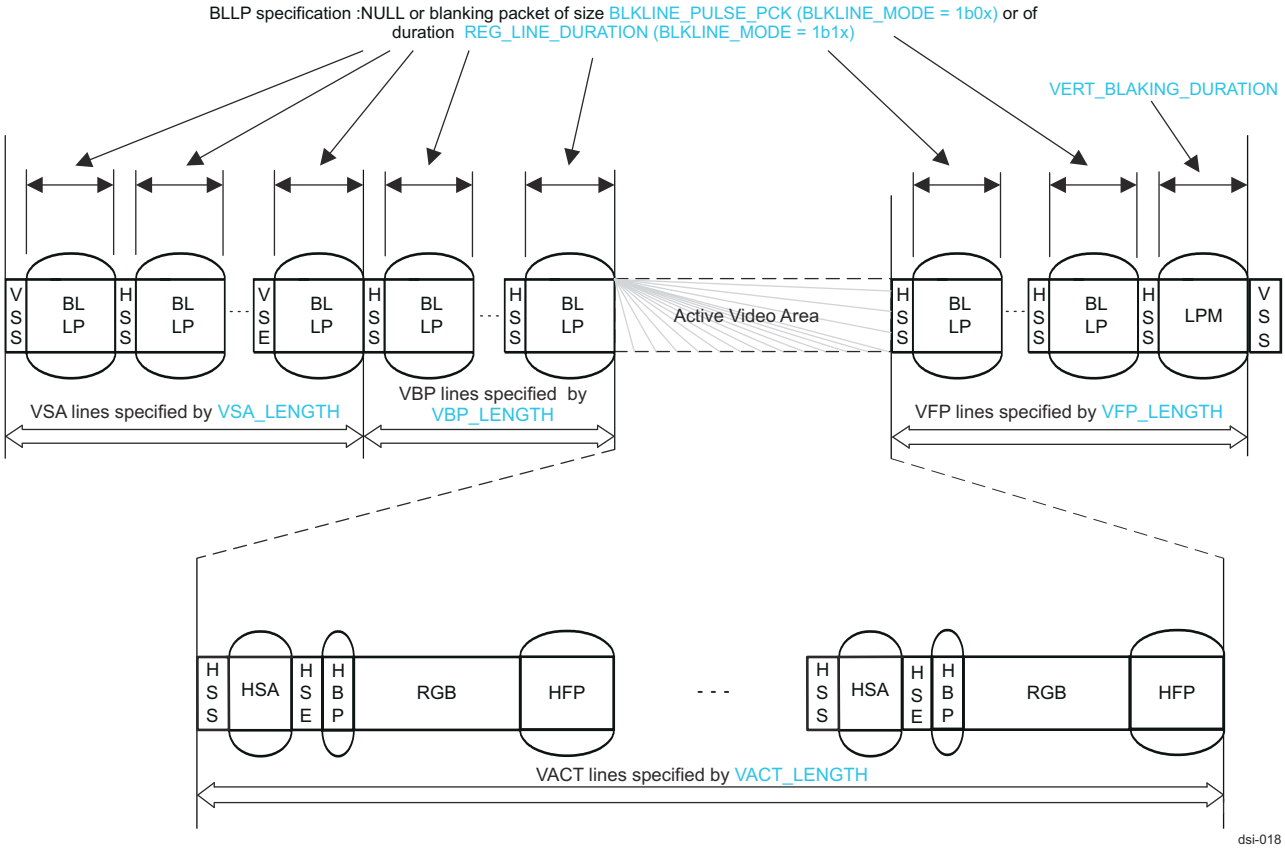


Figure 12-396. Video Pulse Mode (Non-Burst) Timing Diagram

Copyright © 2024 Texas Instruments Incorporated

Note

This sequence is no longer specified in the MIPI spec Version 1.02.00 28-Jun-2010 but supported by the DSI-controller.

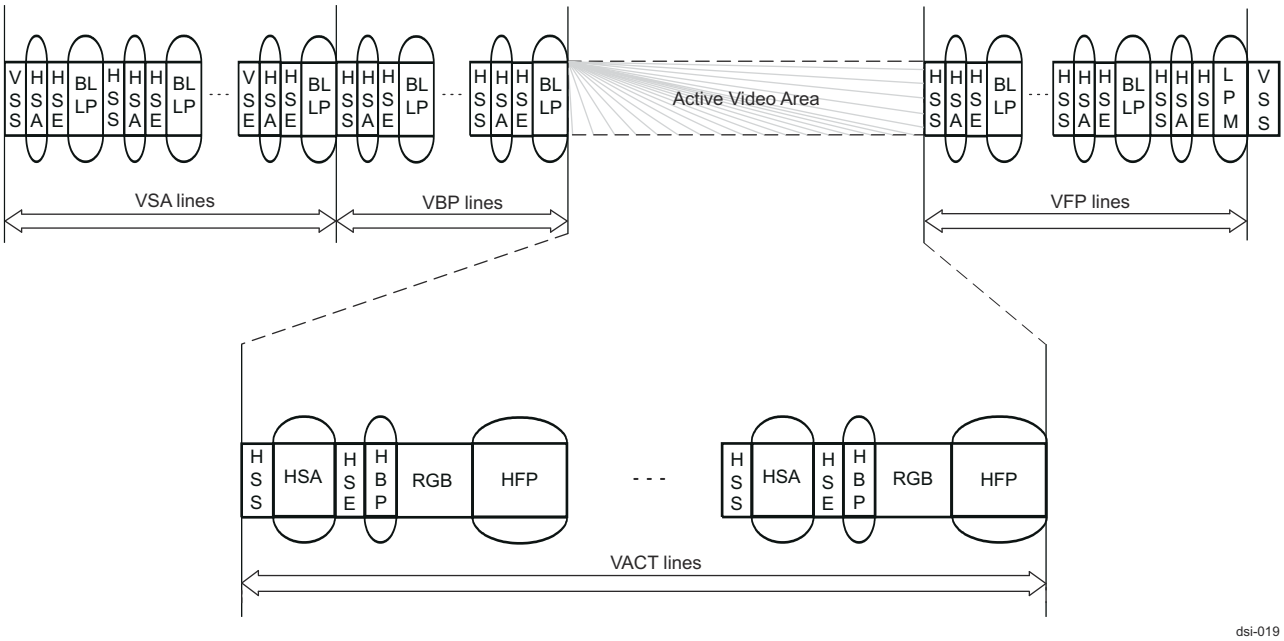


Figure 12-397. Video Pulse Mode (Non-Burst)

Note

In the MIPI spec 1.02.00 June 2010, the line (VSE/HSA/HSE/BLLP) has been moved from VSA lines part to the VBP lines part. The sequence is the same but in the design the VSA lines has one more line and the VBP lines has one line less. This must be considered during the VSA and VBP register programming if the DSI MIPI spec 1.02.00 is used as reference.

Copyright © 2024 Texas Instruments Incorporated

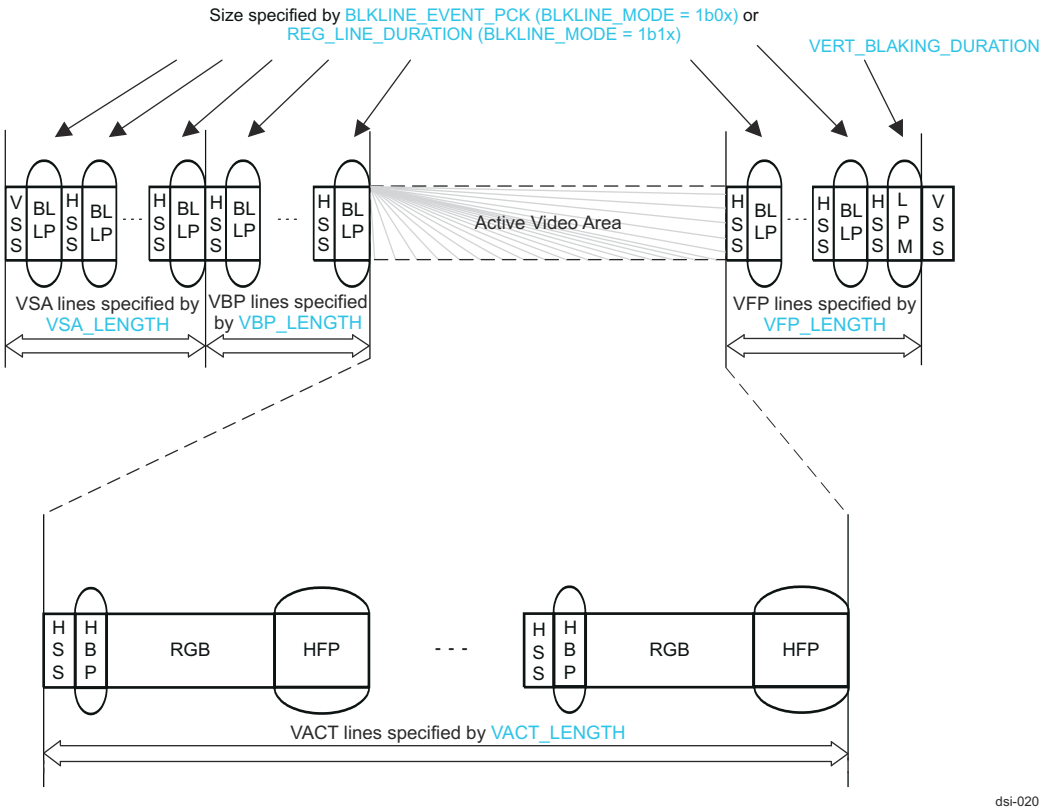


Figure 12-398. Video Event Mode (Non-Burst)

burst_mode = sync_pulse_active = sync_pulse_horizontal = 0 - all spec versions.

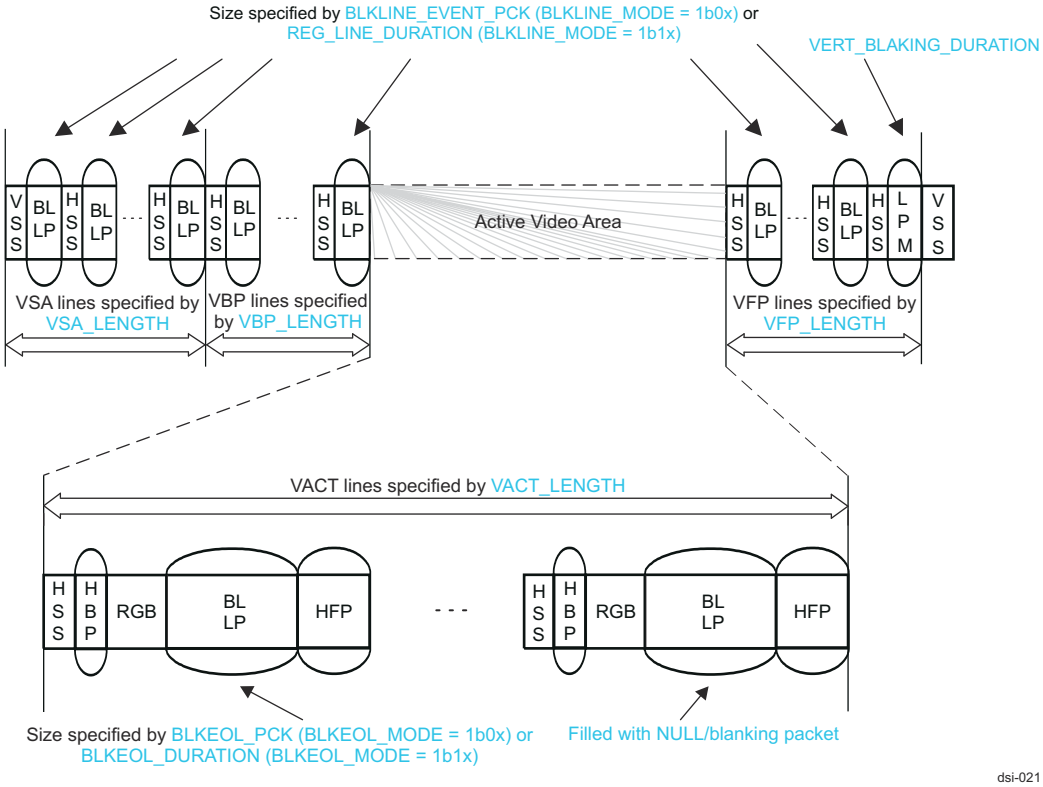


Figure 12-399. Video Sync Event in Burst Mode

Copyright © 2024 Texas Instruments Incorporated

burst_mode = 1 and sync_pulse_active = sync_pulse_horizontal = 0 - all spec versions.

Burst Mode operation requires the tx_byte_clk to be set to double the bit rate compared to the non-burst case. The calculations for the video parameters must then be increased to match the higher rate. The DPI FIFO must also be large enough to store at least half of the active line bytes before the output transmission begins otherwise it will underrun.

12.6.4.7.7.2 Video Stream Settings (VSG)

The following is all the registers that need to be set before using the video in SDI or DPI interface operation.

- vid_vsize1, 2
- vid_hsize1, 2
- vid_blksize1, 2
- vid_pck_time
- vid_dphy_time
- vid_error_color1, 2
- vid_vca_setting1, 2
- vid_main_ctl – programmed last in sequence of vid registers to apply to DSI registers

This section addresses the dependencies between all video registers, and details how the TVG and VCA registers must be set per VSG registers.

vid_main_ctl

- header information must correspond with the vid_pixel_mode used to control the generation of the recovery streams. The header information is used to be enable management of future cases where the DSI link could be used to pass streams with formats that differ feom those currently supported directly.
- sync_pulse_horizontal can only be asserted if sync_pulse_active = 1.

  When sync_pulse_active is set to one, the HSYNC pulses are emulated only during the active part of the frame. When both sync_pulse_horizontal and sync_pulse_active are set, the HSYNC pulses are present on all lines of the frame.

vid_vsize - the variables are expressed in line numbers in this register.

- vsa_length should be at least one (in order VSG can insert at least the vertical synchronization start VSS line) and greater or equal to 2 when pulse mode (sync_pulse_active = 1).
- vbp_length 0 to 63.
- vfp_length must be greater than 0.
- vact_length must be greater than 0.
- vid_hsize1: - the numbers are expressed in number of bytes (the fields are specifying the payload).

vid_hsize1 - the numbers are expressed in number of bytes (the fields are specifying the payload of the corresponding packet).

- hsa_length need to be at least set to one if pulse mode is enabled. The expected normal operating range is 1 to 255, large values for the HAS length will impact on the size of the DPI FIFO required.
- hbp_length can have any value. However, when set to 0, a HBP packet is generated with 0 payload.

vid_hsize2 - again the fields are specifying the payload of the corresponding packet (in byte).

- hfp_length can have any value. However, when set to 0, no HFP blanking packet is generated.
- rgb_size must match the number of pixel per line of the display multiplied by the number of byte per pixel.

vid_blksize1 - still specifying size of the payload in byte.

- blkline_event_pck defines the payload length of the blanking or NULL packet to be inserted in blanking line (with sync event). This value is used when reg_blkline_mode = 0b01 or 0b00 and when sync_pulse_active = 0. It is also needed by the VSG when reg_blkline_mode = 0b1x. Its value depends on the hbp, rgb, blkeol, and hfp packet length.
- blkline_event_pck defines the payload length of the blanking or NULL packet to be inserted in blanking line (with sync event). This value is used when reg_blkline_mode = 0b01 or 0b00 and when sync_pulse_active = 0. It is also needed by the VSG when reg_blkline_mode = 0b1x. Its value depends on the hbp, rgb, blkeol, and hfp packet length.

Copyright © 2024 Texas Instruments Incorporated

-- 32767 = 0x7FFF = max value on 15 bits and -100 to take

margin blkeol_pck < 32767 - hfp_length - hbp_length - hsa_length - rgb_size - 100;

The blkeol_pck (and blkeol_duration) must be adapted to the reg_wakeup_time in case VCA is authorized to go back in LP.

vid_blksize2 - still specifying size of the payload in byte.

- blkline_pulse_pck has almost the same definition than blk_event_pck but it concerns blanking lines with sync pulse.

vid_pck_time - specifies duration in clock cycles.

- blkeol_duration: blkeol_duration = div_round_up [(1)] ((blkeol_pck + 6), lane_nb);
- vert_blanking_duration is no longer used and can always be set to 0. It remains present in the design for legacy reason.

Note 1: The function div_round_up is a function that performs a division then round the result to the first entire number superior or equal to the division result.

vid_dphy_time: specifies duration in clock cycles. These are values that have external dependencies and thus need to be calculated first (and rest of VSG programming is calculated from these values).

reg_line_duration: depends on display type and size and of its programming (in some display, the blanking sizes can be adapted to provide a given number of frame per second with one of the D-PHY possible clock frequency (as PLL can only generate a given number of discrete frequencies).

if (pulse mode) {

line_length = (blkline_pulse_pck + 6);

}

else (event mode) {

line_length = (blkline_event_pck + 6);

if (burst_mode and burst_lp and lane_nb == 2 and

hsa_length[0:0] == 1 and hfp_length[0:0] == 1)

line_length = line_length - 1;

}

result = div_round_up(line_length, lane_nb);

Note 1: The function div_round_up is a function that performs a division then round the result to the first entire number superior or equal to the division result.

Additional Notes about the line duration:

- If line length (in bytes) is a multiple of the number of lanes enabled, DSI will never face any timing jitter except due to the resynchronization in SP/DCB.
- If line length (in bytes) is NOT a multiple of the number of lanes enabled For Non LP operation, there should not have any problem since the line is once longer, once shorter than the "normal" value. For example: if line length is 491 bytes, it will be 245 then 246 in clock cycles with 2 lanes for an average value of 245.5.

  For LP enabled operation, the controller cannot allow underflow of the line and frame timing. With the previous example, the LP line length in clock cycles will be 246 since we cannot start the HS transmission in the middle of a clock cycle. And there is no mechanism to have one line in 245 and the next one in 246 cycles. For LP see section 0.

reg_wakeup_time must be shorter than line duration and depends on the D-PHY cell plus some pipelines delays inserted by the DSI link. This value strongly depends on the DPHY PLL programming and configuration, and is a mix of both internal and external analog and digital timing factors, therefore it is very difficult to provide an exact formula. The recommended approach to selecting a suitable value for this parameter is to characterize

Copyright © 2024 Texas Instruments Incorporated

behaviour in at the system level environment using simulation, emulation (e.g. Palladium) or validation (e.g. FPGA).

The DPHY needs 100 ns + time for THXS-EXIT to go to Low Power mode.

The timing for the clock lane TXRequestHS going active to the TXReadyHS will be determined by the DPHY DDR bit rate clock frequency. The timing will be the total number of DDR bit clk cycles for the TXRequestHS to be detected and the clock lane to transition to High Speed, so TLPX + TCLK-PREPARE + TCLK_ZERO. Additional time is required between the clock lane driving HS clocks and the Data Lanes being activate, TCLK-PRE. All of these parameters can be calculated from the DPHY datasheet.

Figure 12-400 shows timing diagram for clock lane activation/deactivation.

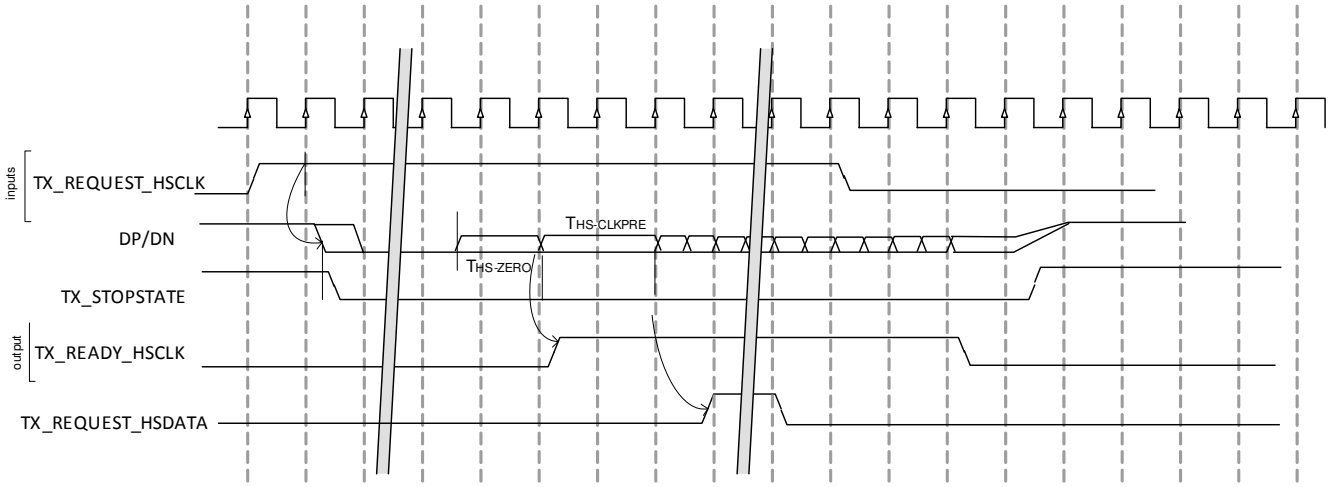


Figure 12-400. Timing Diagram for Clock Lane Activation/Deactivation

The data lane request follows a similar timing sequence from the TX_Request_HS to TX_Ready_HS, TLPX + THS-PREPARE + THS_ZERO. The data lane requests will be activated once the clock lane is ready, i.e TX_Ready_HSCLK is high.

Figure 12-401 shows timing diagram for data lane activation/deactivation.

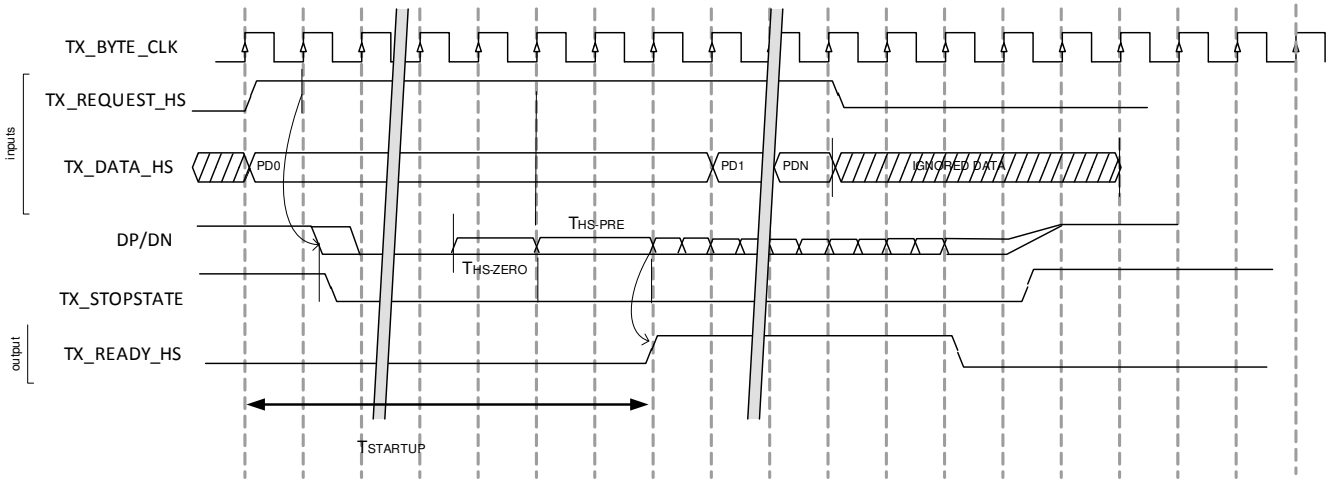


Figure 12-401. Timing Diagram for Data Lane Activation/Deactivation

Table 12-366. Timing Parameters

Parameter	Description	Min	Max	Example 650 Mbps (UI = 1.538 ns)
TLPX	Transmitted length of any Low- Power state period.	50 ns		50 ns

Copyright © 2024 Texas Instruments Incorporated

Table 12-366. Timing Parameters (continued)

Parameter	Description	Min	Max	Example 650 Mbps (UI = 1.538 ns)
TCLK-PREPARE	Time that the transmitter drives the Clock Lane LP-00 Line state immediately before the HS-0 Line state starting the HS transmission.	38 ns	95 ns	38-95 ns
TCLK-PREPARE + TCLK-ZERO	TCLK-PREPARE + time that the transmitter drives the HS-0 state prior to starting the Clock.	300 ns		300 ns
Wakeup Time CL				388-445 ns
TCLK-PRE	Time that the HS clock shall be driven by the transmitter prior to any associated Data Lane beginning the transition from LP to HS mode.	8 UI		12.3 ns
THS-PREPARE	Time that the transmitter drives the Data Lane LP-00 Line state immediately before the HS-0 Line state starting the HS transmission.	40 ns + 4 × UI	85 ns + 6 × UI	~ 46-95 ns
THS-PREPARE + THS-ZERO	THS-PREPARE + time that the transmitter drives the HS-0 state prior to transmitting the Sync sequence.	145 ns + 10 × UI		160 ns
Wakeup Time DL				~ 270 ns

So the wakeup time values expressed as tx_byte_clk (12.3ns) cycles, CL = 36 and CLKPRE + DL = 22.

Figure 12-402 shows wakeup time timing diagram.

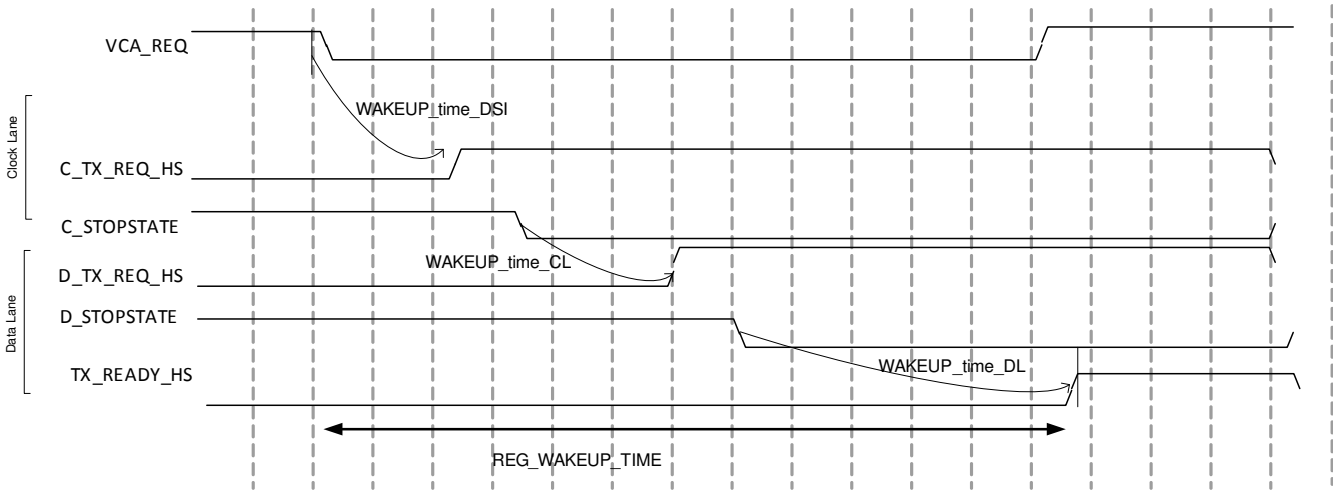


Figure 12-402. Wakeup Time Timing Diagram

Here is the example of the calculation done for the system running at **650 MHz**.

if (clk_continuous == TRUE) {

wakeup_time_cl = 0x1;

} else {

wakeup_time_cl = 0x24;

};

wakeup_time_dsi = 0xA; -- from request on VSG to request HS on DL1 (+ 1 for VSG internal cycle)

wakeup_time_dl = 0x14; -- from stop state falling edge to tx_ready rising edge

reg_wakeup_time = wakeup_time_dsi + wakeup_time_cl + wakeup_time_dl + (hs_host_eot × 4 / lane_nb)

Note: It is not necessary to program all the register values every time because some of them are used only in some modes. For instance, blkline_event_pck and blkline_pulse_pck are used depending on the way SYNC is generated, and then in pulse mode, there is no need to program blkline_event_pck and vice-versa.

Copyright © 2024 Texas Instruments Incorporated

12.6.4.7.7.3 VCA Configuration

The VCA setting done in two registers: vid_vca_setting_1 and vid_vca_setting_2. Their programming must respect the following rules:

- vid_vca_setting_1:
  - max_burst_limit specifies the size of the bigger packet that generates the following sequence RGB + packet + NULL packet + BLK packet; it must be linked with blkeol_pck (same size minus the smaller NULL packet). It is equal to blkeol_pck - 6.
    - burst_lp can be set to 1'b1 if blkeol_pck > 2 × reg_wakeup_time × lane_number (reg_wakeup_time is only specifying the time to go from LP to HS thus need to consider the time to go from HS to LP too).
- vid_vca_setting_2:
  - exact_burst_limit: as it specifies (in byte) the payload of the packet that generates a sequence RGB + packet + HFP, it must have the same value as of blkeol_pck when the DSI link is in burst mode.
  - max_line_limit specifies the maximum size of a packet that generates HSS + (HSA + HSE) + packet + NULL packet. It then depends on line_duration, hsa_length regardless if the system is using pulse or event synchronizing. Its size is linked to vert_blanking_duration.

if (sync_pulse_active == 0) {

-- size of the payload of the inserted packet followed by a NULL pkt

max_line_limit = (blkline_event_pck-6);

};

if (sync_pulse_active == 1) {

-- size of the payload of the inserted packet followed by a NULL pkt

max_line_limit = gen_blkline_pulse_pck-6;

Another constraint to respect when using VCA is to ensure that only supported operations are sent from command side when video is active. For instance, no TE should be attempted, because it could break the video stream (TE completion time can be long). Command messages can be inserted into the LP states provided there is sufficient space for the message bytes and any associate bytes for the header and CRC, if a long packet type.

12.6.4.7.7.4 TVG Configuration

The TVG has a single register: tvg_img_size that should complies with the following rules against VSG settings:

tvg_line_size: must be equal to rgb_size set for VSG

tvg_nbline: must be equal to vact_length set for VSG

The TVG generates a data stream in the same format than the one specified at the output of the VRS. The content of that flow is specified by the following registers:

- Image size - TVG_LINE_SIZE[14:0] and TVG_NBLINE[12:0] registers: Number of bytes per line and number of lines per frame.
- Image kind - TVG_MODE[1:0] register: Single color, vertical stripes or horizontal stripes.
- Stripe width - TVG_STRIPE_SIZE[2:0] register: Significant when a stripe mode is enabled. Possible values are 1, 2, 4, 8, 16, 32, 64 and 128.
- Pixel kind VID_PIXEL_MODE[3:0] register: 16 bits RGB, 18 bits RGB, 18 bits loosely RGB, 24 bits RGB, 30 bits RGB or 36 bits RGB.
- Color one - COL1_GREEN[11:0], COL1_RED[11:0], COL1_BLUE[11:0] registers: Color used in single color mode or first color when in stripe mode. 12 bits per component R/G/B. For formats using fewer bits than 12, the least significant ones are the ones considered.
- Color two - COL2_GREEN[11:0], COL2_RED[11:0], COL2_BLUE[11:0] registers: Color two when in stripe mode. 12 bits per component R/G/B. For formats using fewer bits than 12, the least significant are the one considered.
- Start pulse and stop handshake (+ stop mode) (see TVG_CTL register).

Copyright © 2024 Texas Instruments Incorporated

The Test Video Generator can be programmed to generate a set of test colour patterns based on the display panel that will be used. The panel parameters for horizontal and vertical resolution along with the frame rate and pixel colour depth need to be set based on the datasheet information for the panel.

Figure 12-403 presents TVG MODE patterns.

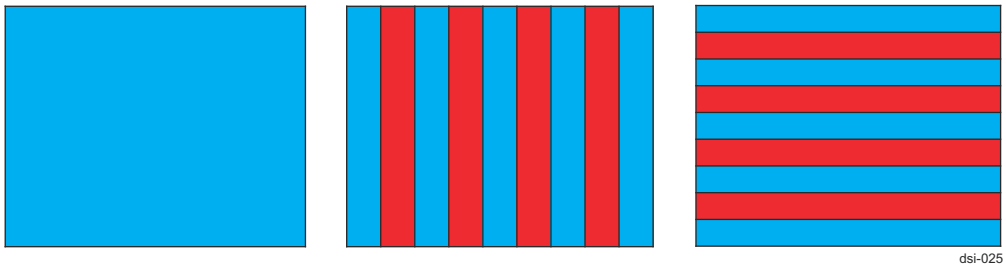


Figure 12-403. TVG MODE Patterns

An example of the sequence is as follows:

- Select TVG for video stream generation instead of DPI/SDI 1 interface.
  - IF1_EN bit to 0 (SDI stall signal at 1) in MCTL_MAIN_EN.
- Select video mode for interface 1 in MCTL_MAIN_DATA_CTL.
  - TVG_SEL bit to 1 in MCTL_MAIN_DATA_CTL register to select stream from TVG.
- Select generated frame format in TVG_CTL register.
  - Image format (single color, H stripes, V stripes) in TVG_MODE field.
  - Image color selection in TVG_COLOR1, TVG_COLOR1_BIS, TVG_COLOR2, VG_COLOR2_BIS.
  - Stripes size in TVG_STRIPE_SIZE field.
  - Image size in TVG_IMG_SIZE register.

  Note: The TVG settings on active area for the number of lines per frame and number of bytes per line must match VSG settings on active area. Any mismatch will create an error that is detected in the VSG and forces recovery mode in TVG, stopping test frame generation.
- Select TVG stop mode with TVG_STOPMODE field of TVG_CTL.
  - TVG video stream generation start/stop controlled by the TVG_RUN bit in TVG_CTL register.
  - Polling TVG run status from the TVG_RUNNING bit in TVG_STS register is useful when setting TVG_RUN to 0. TVG video stream generation does not stop the instantaneously (depends on TVG stop mode), so this should be checked to confirm the TVG has stopped.

12.6.4.7.8 DPI To DSI Programming

The DPI interface will normally be driven from a graphics driver that will be configured to match a frame geometry and refresh rate for a panel display. The DPI driver will use the expected refresh rate and geometry for the active and blanking parts of a frame to determine the pixel rate clock. The DSITX controller will then translate this to match the incoming byte information to send across the DSI DPHY link.

A system will always expect the bandwidth of the incoming pixels × BPP to match the transmission of bytes from the DSI DPHY based on the number of active lanes. The relationship of DPI clock to TX byte clock must ideally hold with the following formula:

$F_{pixel_clk} = F_{tx_byte_clk} \times active_lanes \times 8 \div bits_per_pixel;$

The DSI will supply packets to the DPHY interface using the internal sequencing and counters clocked from the tx_byte clock. The normal system operation will mean that the DSI must always have the correct number of tx_byte clock cycles to match the number of pixel clock cycles. This means that all the events used for the packet generation will be directly impacted by the delay that exists in the DPHY High Speed request to ready.

The DSITX controller relies upon the DPHY PLL being programmed and locked before starting the DPI video. The DSITX controller can then be configured and enabled (clock lane and data lanes) before starting the video transmission.

Copyright © 2024 Texas Instruments Incorporated

The DSITX controller will begin data transmission with a High Speed request after it receives the falling edge of the VSYNC on the DPI interface. The controller will then begin to build the bytes for the VSYNC packets (VSS, HSA, Blanking packets) during the delay period before the ready is returned from the DPHY. This delay will add an interval effect on all the packets that follow.

The DPI FIFO size must be sufficient to buffer all the incoming pixel data during the active line stage while the DSITX controller continues to send the previous lines packet information due to this interval effect.

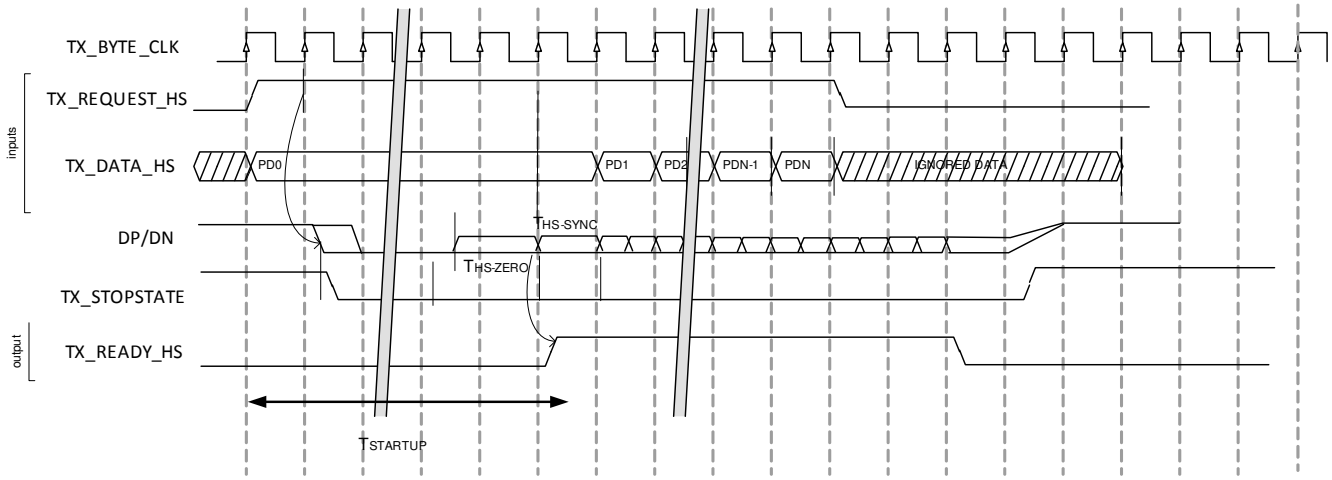


Figure 12-404. Timing Diagram

The DPI and DSI system may not be able to guarantee the exact frequency on the pixel clock input or tx_byte clock from the DPHY. The DSI configuration can be adjusted to support a small deviation in frequencies using the size of the HFP packet so that it changes the alignment of packets on the active lanes to absorb the different pixel clock cycles during the line. More details are given in section0.

12.6.4.7.8.1 DSI and DPHY Operation

The DPI FIFO is used to buffer all the active pixel data from the DPI interface during the active line. The depth of the FIFO must be selected to allow the pixel to be stored after the DPHY request to ready delay is introduced at the start of every frame. The FIFO will fill to a depth based on the tx_byte clock cycles used from the point where VSYNC is identified to the response from the DPHY link after it has exited Low Power and entered the zeroes state on the active data lanes.

Figure 12-405 presents DPI interface to DSI DPHY timing diagram.

Copyright © 2024 Texas Instruments Incorporated

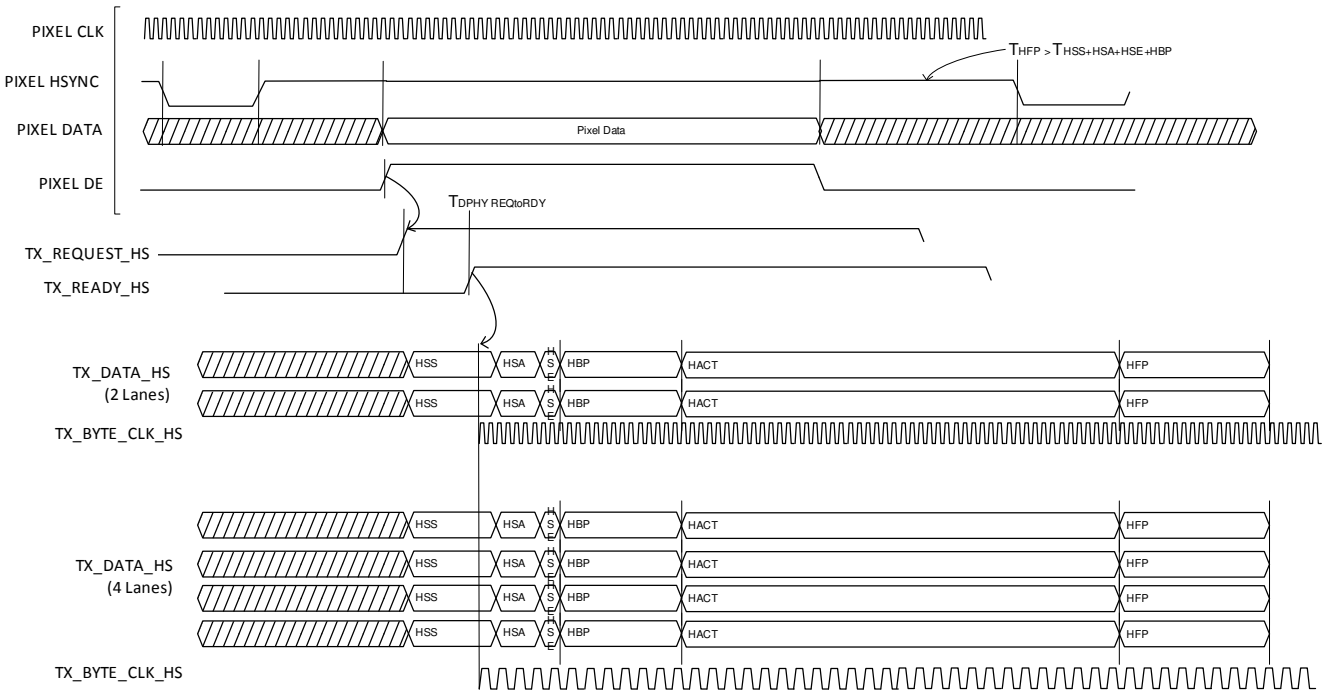


Figure 12-405. DPI Interface to DSI DPHY Timing Diagram

The timing diagram above illustrates how the packets are generated for a DPHY with two and with four lanes activated. The diagram shows that the tx_byte clock changes based on the number of lanes, and that the configuration register values for the number of bytes used for HSA, HBP and HFP will remain the same.

The register values calculated for each of the line synchronization stages needs to take the number of bytes used to form the packet into account so that the timing for each stage aligns to the timing of the DPI input and most importantly that the total number of bytes exactly matches the number of byte clock cycles. The values used for the Horizontal Front porch must be larger than the combined horizontal sync and back porch to allow the DPI FIFO to empty.

The basic rule is that the values used for the DSI will be based on the DPI × BPP/8.

The DPI graphics driver must be configured with horizontal values that can be directly translated to DSI packet payloads.

- The minimum values for the DPI HSA must convert to be greater than:
  - Non-Burst Sync Pulses – 15 bytes
  - Non-Burst Sync Events – 11 bytes
- The minimum values for the DPI HBP must convert to be greater than:
  - Non-Burst Sync Pulses – 7 bytes
  - Non-Burst Sync Events – 7 bytes

These values will then form the number of tx_byte_clk cycles used to send the bytes over the number of active lanes. So, for an RGB888, DSI configuration the minimum values will be:

- HSA = roundup(15/3) = 5
- HBP = roundup(11/3) = 4

The diagram below, D-PHY Timings Control (see Figure 12-406), illustrates the byte mapping for a four lane DPHY link. In this example, the calculation for the horizontal sync stage aligns exactly across the four byte lanes.

- HSS – Four Bytes
- HSA – Blank Packet with Header (4) + HSA_Count + CRC(2)
- HSE – Four Bytes

The next stage is the horizontal back porch:

- HBP – Blank Packet with Header (4) + HBP_Count + CRC(2)

Copyright © 2024 Texas Instruments Incorporated

The value used in the HBP count can be used to account for the Header bytes used in the active data packet in next stage, although it is best to add this to the HFP stage calculation and allow time to empty the FIFO.

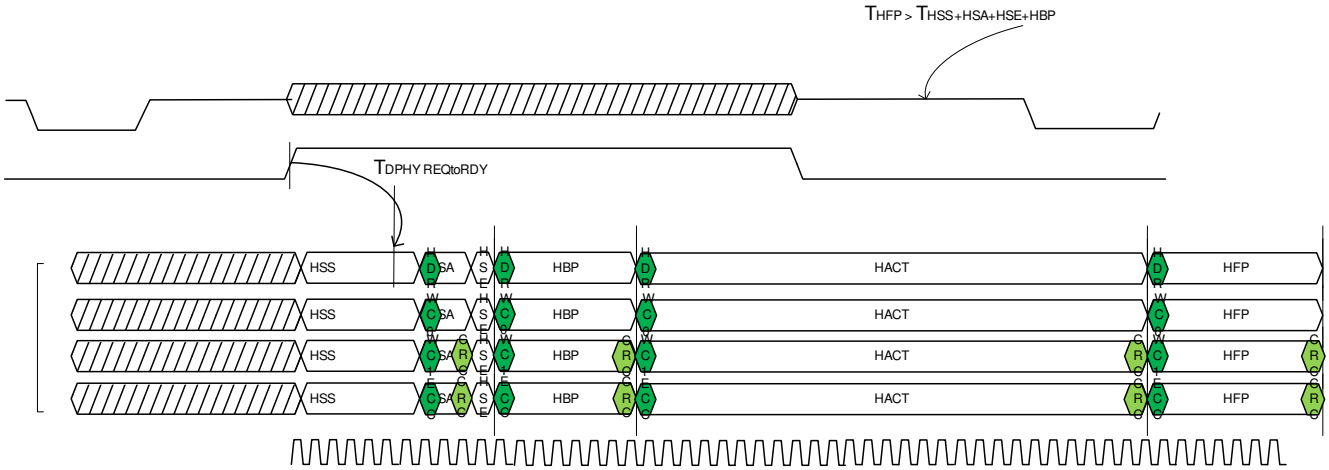


Figure 12-406. Horizontal Line Packet Timing Diagram

The Active part of the line will be a fixed size based on the line size and BPP, and include the six bytes for the packet header and CRC. The final stage is the horizontal front porch.

- HFP – Blank Packet with Header (4) + HFP_Count + CRC(2)

The HFP value should be adjusted to ensure that the number of tx_byte clock cycles for the horizontal line exactly matches the pixel clock cycles for the line times the BPP.

The timing diagram below, Control Block, illustrates the byte alignment changing when the HBP calculated is two bytes shorter than the previous case. This leads to the active and front porch packet headers beginning two bytes earlier, however the HFP count is adjusted to keep the end of the line on exactly the same tx_byte_clk cycle count.

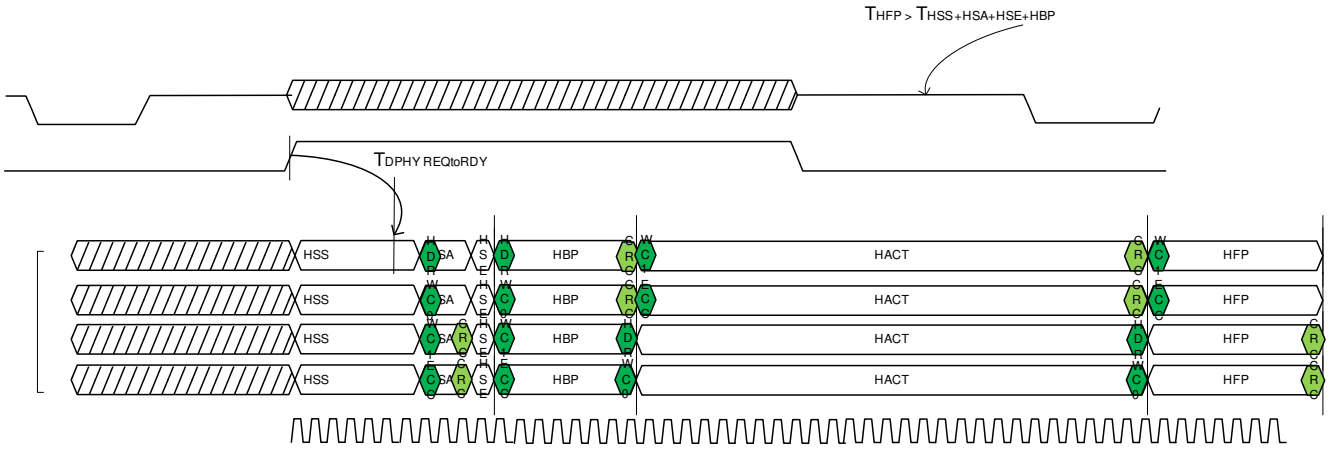


Figure 12-407. Horizontal Line Packet - Shorter HBP - Longer HFP Timing Diagram

The recommendation is to ensure that the total bytes for each horizontal line results in an exact integer tx_byte clock cycles (Total_bytes MOD num_of_lanes = zero), except for the case where the clocks are not exactly matched.

12.6.4.7.8.2 Pixel Clock to TX_BYTE_CLK Variation

The selection of the Pixel clock rate and DPHY Bit clock rate should ideally match based on the ratio of bpp on the pixel side compared to the tx_byte_clk. The timing for the horizontal sync, back and front porch will also be configured to allow the pixel data to be transferred without any over or underflow of the DPI FIFO.

Copyright © 2024 Texas Instruments Incorporated

The DPI FIFO fill level register (DPI_CFG) can be used to track that the parameters are sufficient and that any clock variation can still be handled by the FIFO depth. This register will show the pixel data held in the buffer, so during the HSA and HBP packet generation stages DPI pixel data will be stored and this value will increment. Once the data packet is being generated, the DPI_CFG level will remain constant, as the buffer is emptied at the same bandwidth as it is filled. Finally, the value will reduce to zero during the HFP stage.

The packet length of the HFP can be adjusted to help absorb small differences in the pixel_clk to tx_byte_clk relationship. The DSITX controller will concatenate all of the packets during High Speed transmission, so for example adding to the byte value of the HFP will make the bytes at the end of the packet move to align on another lane.

The packets that follow will then be concatenated and aligned to the next available lane, so if the tx_byte_clk is faster than the expected pixel clock times bpp the extra byte will delay the start of the new line tx_byte_clk × extra_byte/lanes.

Figure 12-408, Video coherency, illustrates the impact of increasing the HFP by one byte so that the next line begins ¼ tx_byte clock later. This allows averaging out across the line length of small differences in the pixel to tx_byte clocks when the tx_byte clock is not ideally matched to the expected pixel clock multiplied by bpp.

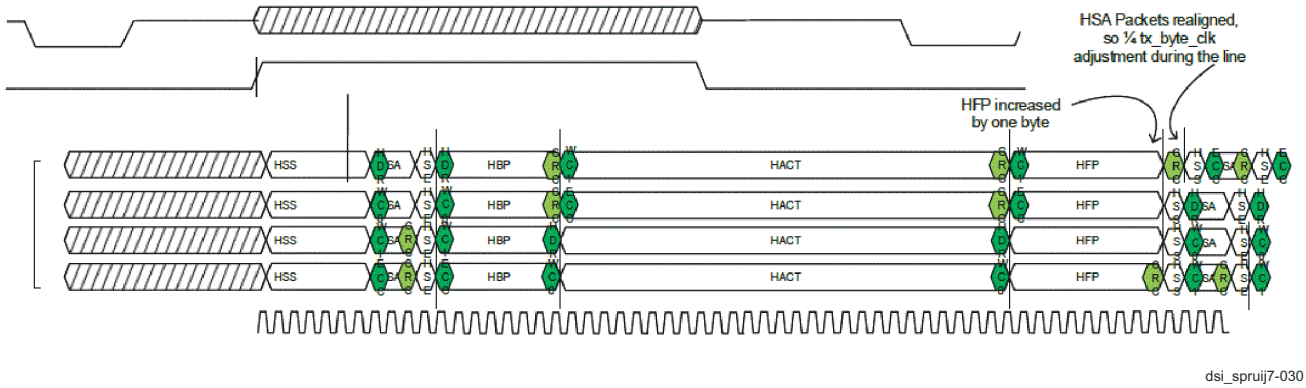


Figure 12-408. Packet Alignment Control Using HFP Byte Increase - tx_byte_clk Faster than Ideal Rate

The DPI_CFG register will show any clock misalignment as an increasing/decreasing value for the FIFO fill level on each line of pixels (must be read mid-line for an accurate reading of the level). When the byte count adjustment is made, the DPI_CFG register will return to the original fill level value every four lines.

12.6.4.7.8.3 LP Operation

The DSITX controller can be configured to perform a transition between LP state and HS state during the horizontal lines which have long periods of blanking. The registers controlling the timing for the DPHY wakeup time and the reg_line_duration must be programmed to exactly match the DPI cycles of the horizontal line to the tx_byte_clk cycles required for the number of active lanes selected.

Copyright © 2024 Texas Instruments Incorporated

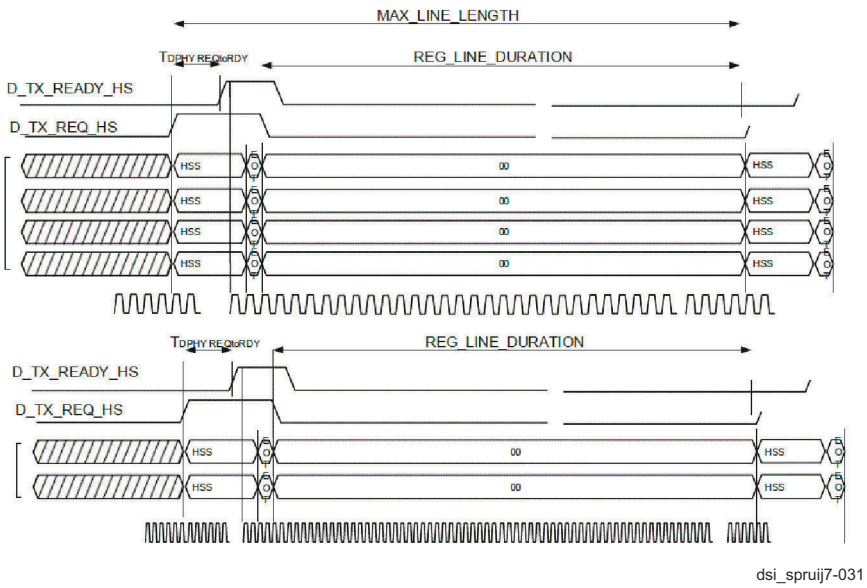


Figure 12-409. Low Power Operation with Four and Two Active Lanes

The LP operation for each vertical blanking line will require the reg_line_duration to be configured to match the following:

- Max_line_length (in tx_byte_clk cycles see 2.2.4) minus Eot (if enabled), also minus further 10 if CLOCK lane is non-continuous.

Note REG_WAKEUP_TIME is used internally to adjust the cycles for each vertical LP line.

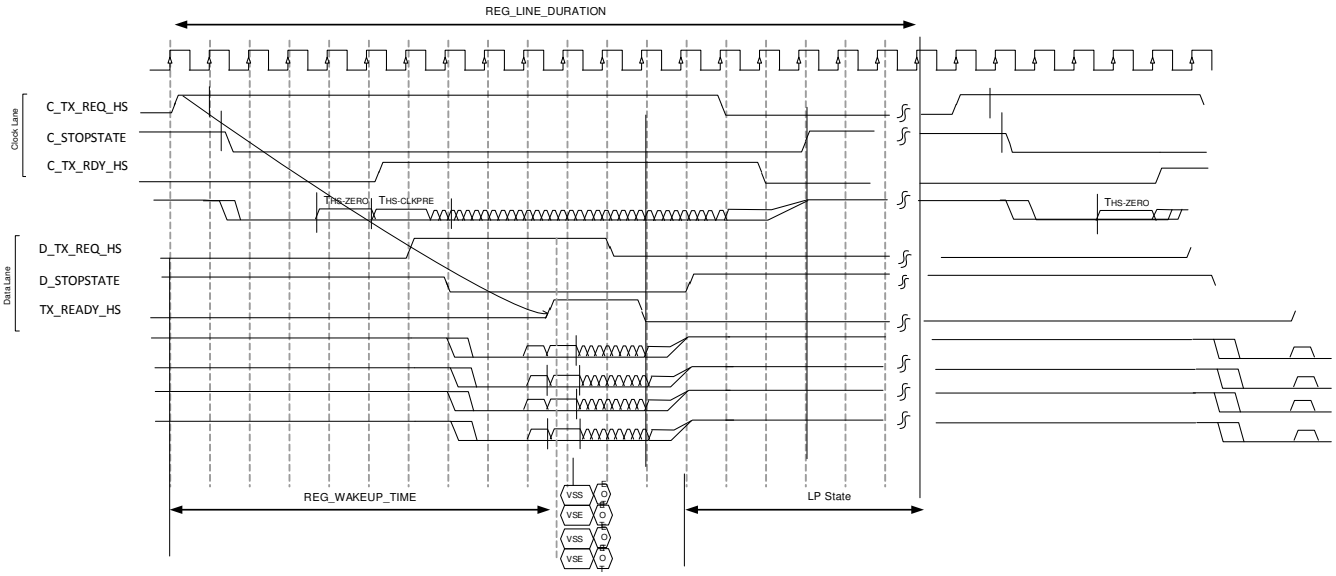


Figure 12-410. REG_LINE_DURATION Timing Example for LP Operation

12.6.4.7.8.4 DPI Interface Burst Operation

The DSITX controller can support burst operation on the DPI interface provided the DPI FIFO size is large enough to store the line pixel data with enough data to not underflow. The burst operation will use a tx_byte_clk that is faster than normal event based operation, so the DPI FIFO size will be impacted by the ratio of the pixel_clk and BPP input to the tx_byte_clk and number of lanes on the output.

The recommended operation is to set the DSI HSA and DSI HFP registers to zero, and adjust the DSI HBP size to control the number of bytes prefilled in the DPI FIFO before the burst is sent.

Copyright © 2024 Texas Instruments Incorporated

Figure 12-411 shows DPI operation with event burst mode.

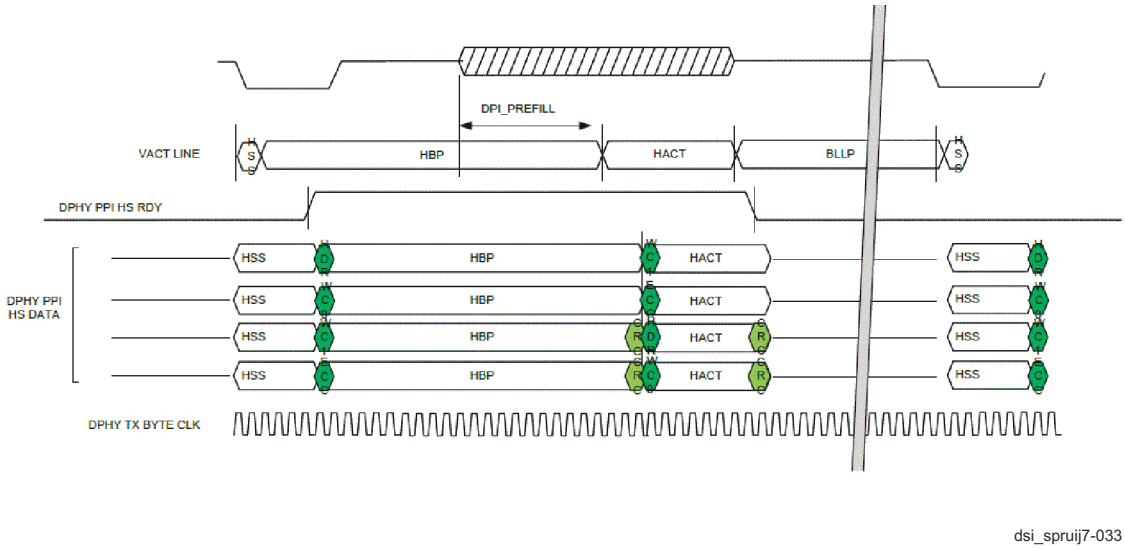


Figure 12-411. DPI Operation with Event Burst Mode

12.6.4.7.9 Programming the DSITX Controller to Match the Incoming DPI Stream

The time taken to output a frame on the PPI needs to match what is coming in on the DPI. This is best achieved by using the recommended clock ratio between the pixel and byte clocks. Assuming that is the case, then the blanking and active data periods must be matched up.

For each frame of video the time will be:

$(V_{total}*H_{total})_{@pixel_clk} = (V_{total}*H_{total})_{@tx_byte_clk} \times bits_per_pixel/8;$

So for example, the time for a full HD frame 1920x1080 with reduced blanking in RGB565 will be:

2200x1200 pixels = 2200x1200 tx_bytes × 16/8;

The DSITX controller will slave its frame timing to the incoming DPI vsync, as long as it is programmed to generate a frame of slightly less than the same size when the VFP blanking is considered. The controller works by transitioning to LP during the last programmed line of VFP. It will then remain in LP until the start of the next frame. So programming the controller to match the DPI, but with at least one fewer line of VFP should result in a reliable configuration.

12.6.4.7.9.1 Vertical Timing

The DSITX controller will generate vertical packets for each part of the frame beginning with the VSS and VSE combined with blanking packets to fill the VSA. The blanking packets for the pulse mode program the DSI vertical size registers as follows:

- VSA = DPI_VSI (lines, minimum of 2 –> VSS and VSE)
- VBP = DPI_VBP (lines, minimum of 0)
- VACT = DPI_VACT (lines)
- VFP < DPI_VFP (minimum of 1)

DPI horizontal timing is measured in pixel clocks, whereas the DSITX controller uses byte clocks. The horizontal timing should also be matched per line; therefore the blanking and active periods of each line should match. The relationship therefore depends upon the pixel format, which could be 24, 18 or 16 bpp. The timing for a horizontal line for the DPI driver side will be the number of pixel cycles for the **HLINE** = (HSA + HBP + HACT + HFP) and this will form **HLINE** × bpp/8 bytes for the controller. These bytes will then be sent using 1, 2, 3 or 4 data lanes, so the DSITX controller will required 1, ½, 1/3, ¼ this number of tx_byte_clk cycles respectively.

e.g a DPI with HSA = 12, HBP = 12, HACT = 1920, HFP = 24 and 16bpp will be 1968 pixel clocks for each horizontal line.

Copyright © 2024 Texas Instruments Incorporated

This gives the DSI horizontal lines 3936 bytes to transmit and would use 984 tx_byte_clks on a four lane system, or 1312 on a three lane system etc..

The horizontal configuration uses different registers depending on the Mode (Pulse or Event) and the control of the BLKLINE_MODE register bit.

- Non-Burst Mode with Sync Pulses – enables the peripheral to accurately reconstruct original video timing, including sync pulse widths.

  Note that for accurate reconstruction of timing, packet overhead including Data ID, ECC, and Checksum bytes should be taken into consideration.
- Non-Burst Mode with Sync Events – similar to above, but accurate reconstruction of sync pulse widths is not required, so a single Sync Event is substituted.
- Burst mode – RGB pixel packets (active video) portion are time-compressed, leaving more time during a scan line for LP mode (saving power) or for multiplexing other transmissions onto the DSI link.

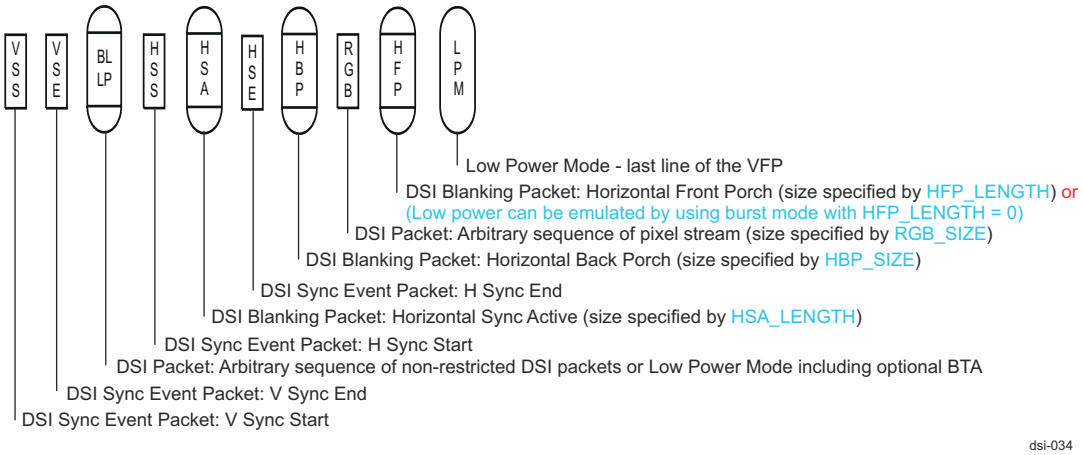


Figure 12-412. Vertical Timing

12.6.4.7.9.2 Horizontal Timing for Non-Burst Mode with Sync Pulses

Sync Pulse Mode uses the Short packet and Long packet structures to accurately track the DPI interface timing. Figure 12-413 illustrates the packet sequence for a single frame.

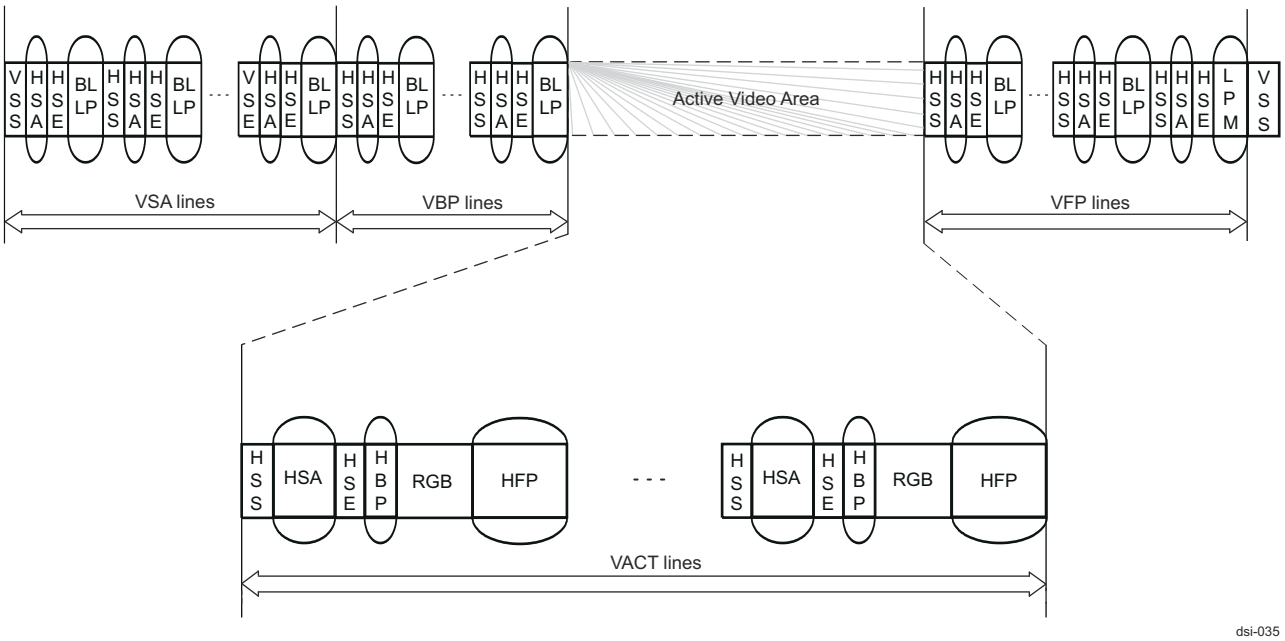


Figure 12-413. Vertical Timing

Copyright © 2024 Texas Instruments Incorporated

The DSITX controller registers for this mode are used to generate the exact number of bytes required for each horizontal line based on the DPI line configuration and the BPP for the colour pixel data.

The packet structure for each type of line during the frame is shown below, Trigger Mapping Information. The calculation of the total number of bytes in any horizontal line must always be HLINE × bpp/8 bytes.

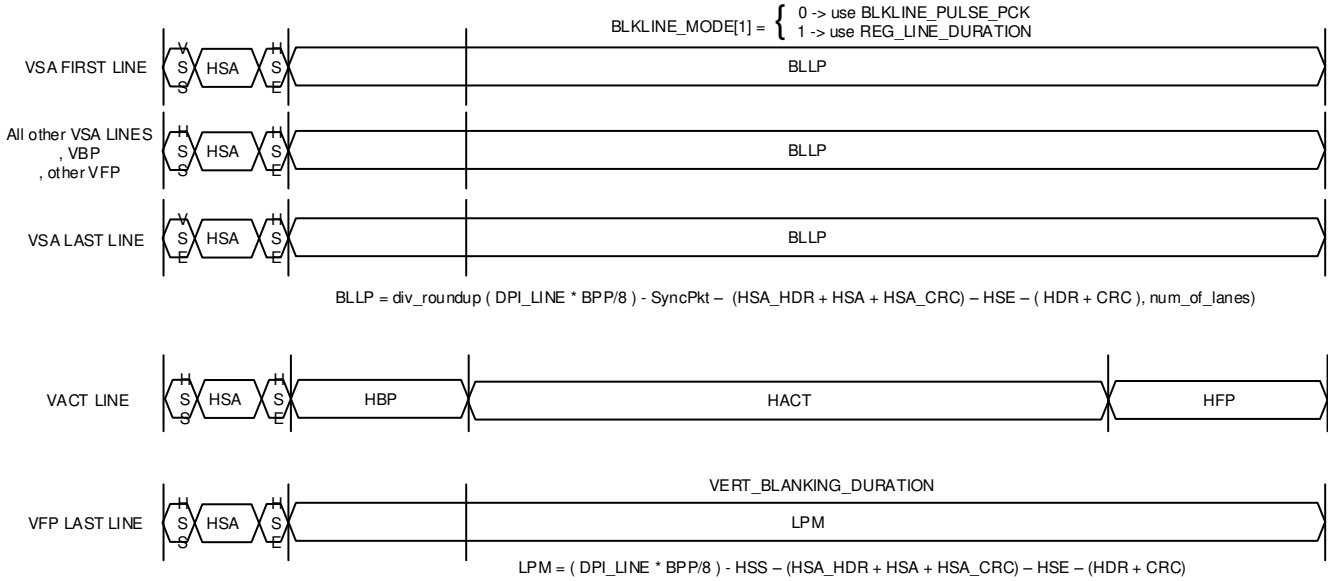


Figure 12-414. Non-Burst Mode With Sync Pulse Line Structures

The DSI short packets and packet headers that are inserted by the controller must be accounted for:

- HSA should be reduced by 14 bytes to account for the HSS short packet (4 bytes), the long blanking packet header and CRC footer (4 + 2 bytes) and the HSE short packet (4 bytes).
- HBP should be reduced by 12 to account for the header and footer on the blanking packet (6 bytes) plus the header/footer on the active data packet (6 bytes).
- HFP should be reduced by 6 bytes to account for the long packet header and CRC footer.

Finally for lines with no active data, the controller will use either:

- BLKLINE_PULSE_PCK: Total line size (HLINE × bpp/8) in bytes minus the HSA 20 bytes (14 for HSA header and CRC, 6 for the remaining blanking which is all combined into a single packet).
- REG_LINE_DURATION: Total line size (HLINE × bpp/8) in tx_byte_clk cycles minus the calculated HSA in tx_byte_clk cycles (HSA × bpp/8 minus 14 bytes (14 for HSA header and CRC) minus the EOT packet (4 bytes) if required, divided by number of lanes). This should be aligned to the number of active lane so rounding the value so that REG_LINE_DURATION MOD Lanes equals zero.
- VERT_BLANKING_DURATION: Total line size (HLINE × bpp/8) in tx_byte_clk cycles minus the calculated HSA in tx_byte_clk cycles (HSA × bpp/8 minus 14 bytes (14 for HSA header and CRC) divided by number of lanes).

Program the DSI horizontal size registers as follows:

burst_mode = 0 and sync_pulse_active = sync_pulse_horizontal = 1;

HSA = (DPI_HSA × bpp/8)-14 – DPI HSA min value 5 for RGB888

HBP = (DPI_HBP × bpp/8)-12– DPI HBP min value 5 for RGB888

HACT = (DPI_HACT × bpp/8)

HFP = (DPI_HFP × bpp/8)-6 – DPI HFP min value 10 for RGB888

Note: (DPI_HACT × bpp/32) must be an integer. Total Line Length = div_roundup((HLINE × bpp/8), num of lanes);

(BLKLINE_PULSE_PCK) bytes} = (HLINE × bpp/8)-20-HSA;

Copyright © 2024 Texas Instruments Incorporated

(REG_LINE_DURATION)tx_{byte}clks} = Total Line Length - div_roundup((HSA × bpp/8) - 14, number of lanes);

(VERT_BLANKING_DURATION)tx_{byte}clks} = Total Line Length - div_roundup((HSA × bpp/8) - 14, number of lanes);

For example; a DPI with HSA = 12, HBP = 12, HACT = 1920, HFP = 24 and 16bpp will be 1968 pixel clocks for each horizontal line. This give each DSI HLINE of 3936 bytes. So for 4 lanes, 984 tx_byte_clk cycles;

HSA = (12 × 16/8) -14 = 10

HBP = (12 × 16/8) -12 = 12

HACT = (1920 × 16/8) = 3840

HFP = (24 × 16/8) -6 = 42

Total Line = div_roundup((12 + 12 + 1920 + 24) × 16/8, 4) = 984 tx_byte_clk cycles = 3936 bytes;

BLKLINE_PULSE_PCK = (3936) -20 -10 = 3906

REG_LINE_DURATION = 984 – div_roundup(12 × 2, 4) = 978

VERT_BLANKING_DURATION = 984 – div_roundup(12 × 2, 4) = 978

Confirming the calculation for each line with BLKLINE_MODE = 0;

VSS = VSSPkt + HSAPkt + HSEPkt + BLK_LINE_PULSE_PCKPkt = 4 + (4 + 10 + 2) +4 + (4 + 3906 + 2) = 3936;

VSALine = HSSPkt + HSAPkt + HSEPkt + BLK_LINE_PULSE_PCKPkt = 4 + (4 + 10 + 2) + 4 + (4 + 3906 + 2) = 3936;

VSE = VSEPkt + HSAPkt + HSEPkt + BLK_LINE_PULSE_PCKPkt = 4 + (4 + 10 + 2) + 4 + (4 + 3906 + 2) = 3936;

VACT = HSSPkt + HSAPkt + HSEPkt + HBPPkt + HACTPkt + HFPPkt = 4 + (4 +10 + 2) +4 + (4 + 12 + 2)+ (4 + 3840 +2) + (4 + 42 + 2) = 3936;

VFP = HSSPkt + HSAPkt + HSEPkt + BLK_LINE_PULSE_PCKPkt = 4 + (4 + 10 + 2) + 4 + (4 + 3906 + 2) = 3936;

VFPLast = HSSPkt + HSAPkt + HSEPkt + VERT_BLANKING_DURATION = 4 + (4 + 10 + 2) + 4 + (978 × 4) = 3936;

Confirming the calculation for each line with BLKLINE_MODE = 1 matches the tx_byte cycles;

VSS = VSSPkt + HSAPkt + HSEPkt + REG_LINE_DURATION = div_roundup((4 + (4 +10 + 2) + 4), 4) + 978 = 984;

VSALine = HSSPkt + HSAPkt + HSEPkt + REG_LINE_DURATION = div_roundup((4 + (4 +10 + 2) + 4), 4) + 978 = 984;

VSE = VSEPkt + HSAPkt + HSEPkt + REG_LINE_DURATION = div_roundup((4 + (4 + 10 + 2) + 4), 4) + 978 = 984;

VACT = HSSPkt + HSAPkt + HSEPkt + HBPPkt + HACTPkt + HFPPkt = div_roundup((4 + (4 + 10 + 2) +4 + (4 + 12 + 2)+ (4 + 3840 +2) + (4 + 42 + 2) = div_roundup((3936, 4) = 984;

VFP = HSSPkt + HSAPkt + HSEPkt + REG_LINE_DURATION = div_roundup((4 + (4 + 10 + 2) + 4), 4) + 978 = 984;

VFPLast = HSSPkt + HSAPkt + HSEPkt + VERT_BLANKING_DURATION = div_roundup((4 + (4 + 10 + 2) + 4 + (3912), 4) = 984;

12.6.4.7.9.3 Event Mode Horizontal Timing

Non Burst Event Mode uses the Short packet and Long packet structures to track the DPI interface timing without accurate reconstruction of sync pulse widths, so a single Sync Event is substituted. Figure 12-415 illustrates the packet sequence for a single frame.

Copyright © 2024 Texas Instruments Incorporated

The DSITX controller registers for this mode are used to generate the exact number of bytes required for each horizontal.

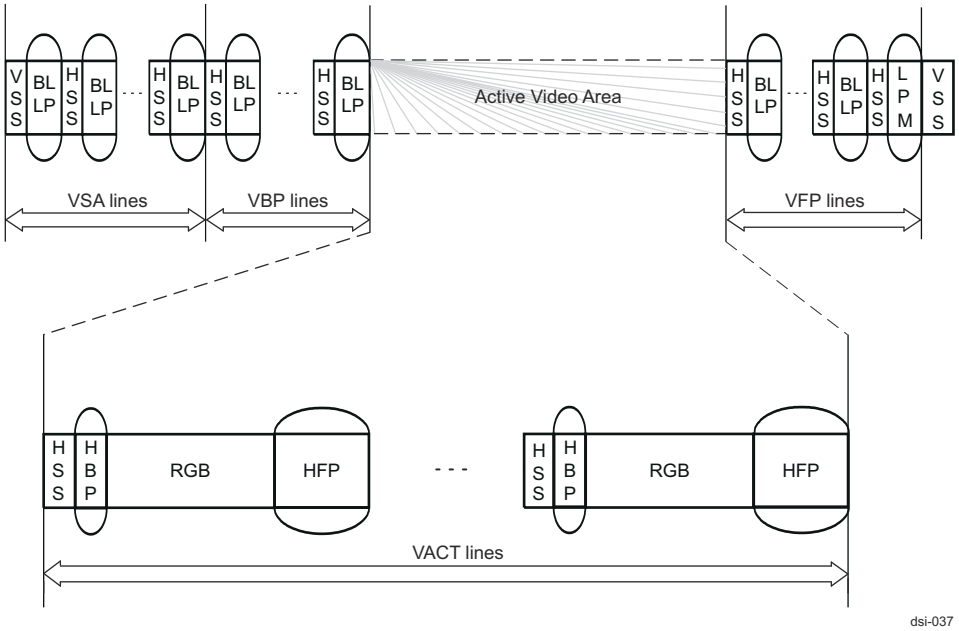


Figure 12-415. Horizontal Timing - 1

line based on the DPI line configuration and the BPP for the colour pixel data. The packet structure for each type of line during the frame is shown in Figure 12-416:

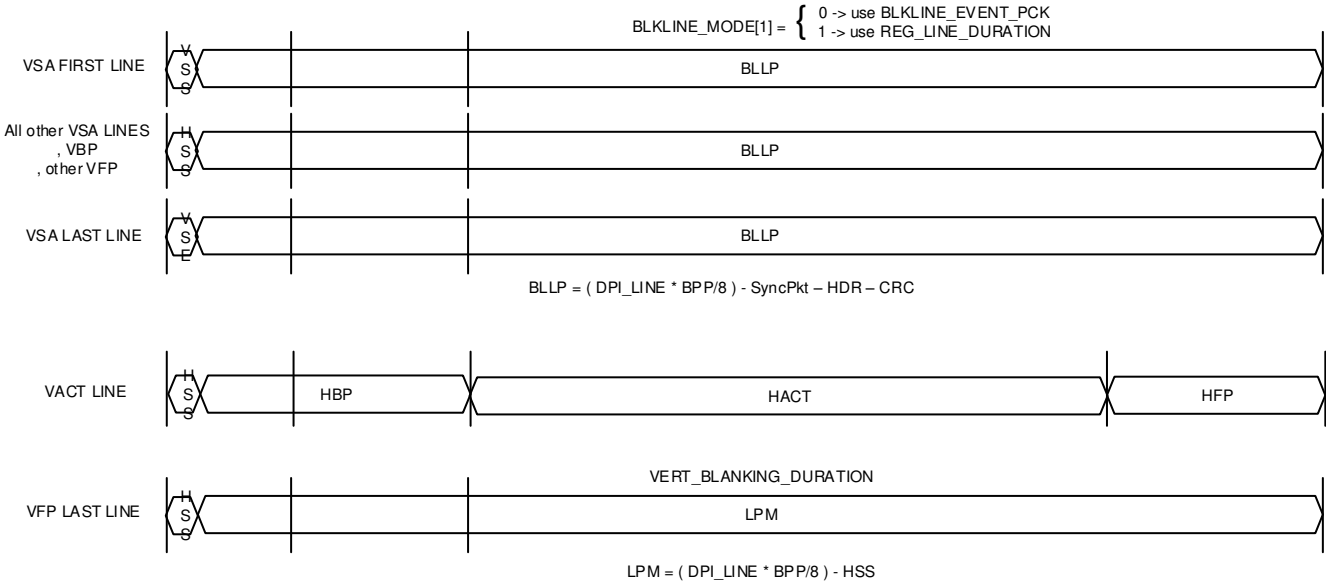


Figure 12-416. Horizontal Timing - 1

The DSI short packets and packet headers that are inserted by the controller must be accounted for:

- VSS/VSE/HSS short packet (4 bytes).
- HBP should account for the header and footer on the blanking packet (6 bytes) plus the header on the active data packet (2 bytes). This will match the DPI_HSA + DPI_HBP minus the Sync Short packet.
- HFP should be reduced by 6 bytes to account for the long packet header and CRC footer and footer of the active data.

Copyright © 2024 Texas Instruments Incorporated

Finally, for lines with no active data, the controller will use either:

- BLKLINE_EVENT_PCK: Total line size (HLINE × bpp/8) in bytes minus 10 bytes (4 for VSS/VSE/HSS, 6 for the remaining blanking header/footer which is all combined into a single packet).
- REG_LINE_DURATION: Total line size div_roundup(HLINE × bpp/8, num_of_lanes) in tx_byte_clk cycles minus tx_byte cycles for the 8 bytes (4 for VSS/VSE/HSS, and 4 for EOT). This should be aligned to the number of active lane so round the value so that REG_LINE_DURATION MOD Lanes equals zero.
- VERT_BLANKING_DURATION: Total line size div_roundup (HLINE × bpp/8, num_of_lanes) in tx_byte_clk cycles minus tx_byte cycles for the 4 bytes (4 for HSS).

Program the DSI horizontal size registers as follows:

burst_mode = 0 and sync_pulse_active = sync_pulse_horizontal = 0;

- HSA = 0
- HBP = ((DPI_HSA + DPI_HBP) × bpp/8)-12 – DPI HSA + HBP min value 5 for RGB888
- HACT = (DPI_HACT × bpp/8)
- HFP = (DPI_HFP × bpp/8)-6 – DPI HFP min value 10 for RGB888

Note: (DPI_HACT × bpp/32) must be an integer. Total Line Length = div_roundup((HLINE × bpp/8), num of lanes).

(BLKLINE_EVENT_PCK) bytes} = (HLINE × bpp/8) - 10

(REG_LINE_DURATION) tx_{byte}clks} = Total Line Length - div_roundup(8, number of lanes)

(VERT_BLANKING_DURATION) tx_{byte}clks} = Total Line Length - div_roundup(4, number of lanes)

For example: a DPI with HSA = 12, HBP = 12, HACT = 1920, HFP = 24 and 16bpp will be 1968 pixel clocks for each horizontal line. This give each DSI HLINE of 3936 bytes.

HSA = 0

HBP = ((12 + 12) × 16/8) -12 = 24

HACT = (1920 ×16/8) = 3840

HFP = (24 ×16/8) -6 = 42

Total Line = div_roundup((12 + 12+1920 + 24) × 16/8, 4) = 984

BLKLINE_EVENT_PCK = (HLINE × bpp/8) – 10 = 981

REG_LINE_DURATION = 984 – div_roundup(8, 4) = 982

VERT_BLANKING_DURATION = 984 – div_roundup(4, 4) = 983

Confirming the calculation for each line:

VSS = VSSPkt + BLK_LINE_EVENT_PCKPkt = 4 + (4 + 3926 + 2) = 3936

VSALine = HSSPkt + BLK_LINE_EVENT_PCKPkt = 4 + (4 + 3926 + 2) = 3936

VSE = VSEPkt + BLK_LINE_EVENT_PCKPkt = 4 + (4 + 3926 + 2) = 3936

VACT = HSSPkt + HBPPkt + HACTPkt + HFPPkt = 4 + (4 + 24 + 2) + (4 + 3840 +2) + (4 + 42 + 2) = 3936

VFP = HSSPkt + BLK_LINE_EVENT_PCKPkt = 4 + (4 + 3926 +2) = 3936

VFPLast = HSSPkt + VERT_BLANKING_DURATION = 4 + (983 × 4) = 3936

12.6.4.7.9.4 Burst Event Mode Horizontal Timing

Burst Event Mode uses the Short packet and Long packet structures to transmit the video information in short bursts and allow the system to use the Low Power states to save active power. This can only be supported with DSITX controllers able to store a horizontal line in the DPI FIFO. The burst mode will expect the tx_byte_clk to be twice the rate of the non burst event mode, and the register calculation will expect the total bytes for each line to double. Figure 12-417 illustrates the packet sequence for a single frame.

Copyright © 2024 Texas Instruments Incorporated

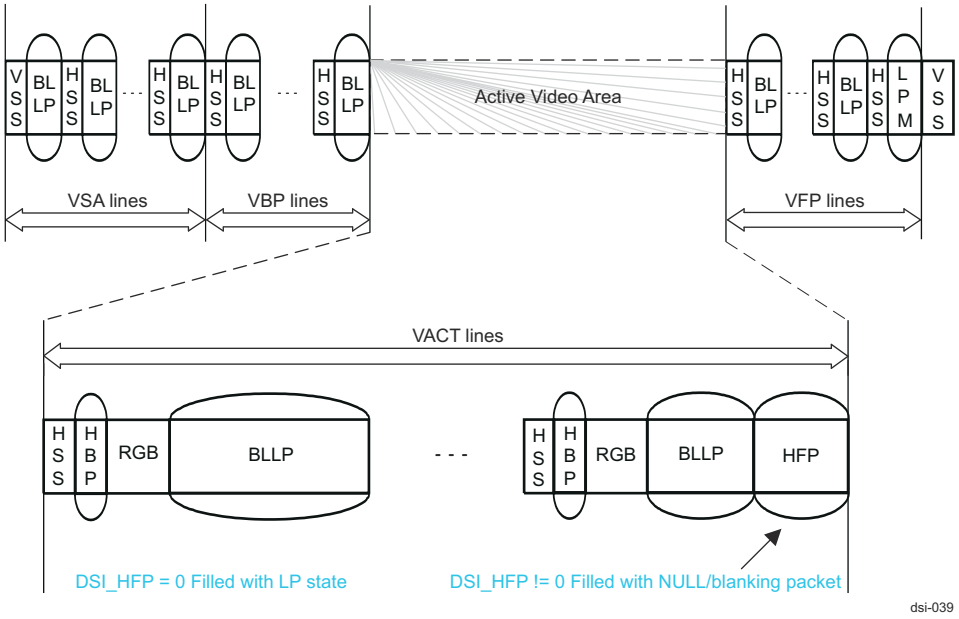


Figure 12-417. Burst Event Mode Horizontal Timing - 1

The DSITX controller registers for this mode are used to generate the exact number of bytes required for each horizontal line based on the DPI line configuration and the BPP for the colour pixel data.

The packet structure for each type of line during the frame is shown below, with the DSI HFP = 0 to select BLLP operation.

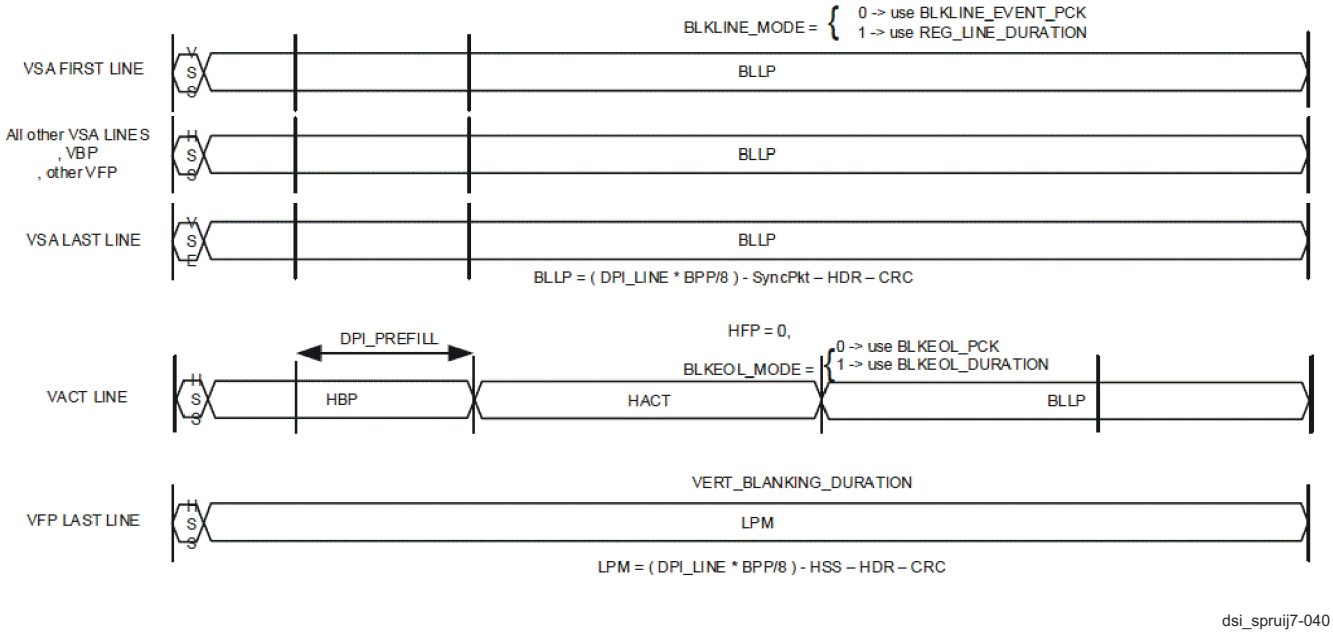


Figure 12-418. Burst Event Mode Horizontal Timing - 2

The DSI short packets and packet headers that are inserted by the controller for active lines must be accounted for:

- VSS/VSE/HSS short packet (4 bytes).

Copyright © 2024 Texas Instruments Incorporated

- HBP should account for the header and footer on the blanking packet (6 bytes) plus the header/footer on the active data packet (6 bytes). This will match the DPI_HSA + DPI_HBP minus the Sync Short packet. In this case the DPI must also reduce the HSA and HFP timing.
- BLLP uses BLKEOL_PCK or BLKEOL_DURATION and will change depending on the HFP use case:
  - HFP can be zero to extend the BLKEOL_x values(recommended).
  - or a non-zero value to transmit a Blanking Packet instead of LP state. The Blanking Packet HFP should be reduced by 6 bytes to account for the long packet header and CRC footer.
  - The host_eot state will cause the BLLP to either add and EOT cycle or not. The BLKEOL_PCK or BLKEOL_DURATION value will need to be adjusted to include this extra cycle if host_eot is disabled.

Finally for lines with no active data, the controller will use either:

- BLKLINE_EVENT_PCK in bytes, excluding the header and crc.
- REG_LINE_DURATION or VERT_BLANKING_DURATION.
- Both are the total expected tx_byte_clk cycle for the line minus the number of cycles required to send the header packet bytes (4 for VSS/VSE/HSS) over the active lanes.

Program the DSI horizontal size registers as follows:

burst_mode = 1 and sync_pulse_active = sync_pulse_horizontal = 0;

- HSA = 0
- HBP = (2 × (DPI_HSA + DPI_HBP) × bpp/8)-12 + DPI FIFO Prefill
- HACT = (DPI_HACT × bpp/8) NOTE: (DPI_HACT × bpp/32) must be an integer.
- HFP = Zero

Total Line Length = div_roundup((HLINE × bpp/8), num of lanes);

(BLKLINE_EVENT_PCK)} = (HLINE × bpp/8) × 2 - 4

(REG_LINE_DURATION)} = Total Line Length × 2 - div_roundup(4, number of lanes)

(VERT_BLANKING_DURATION)} = Total Line Length × 2 - div_roundup(4, number of lanes)

(BLKEOL_DURATION)} = Total Line Length × 2 - TX_BYTE_CYCLES

(BLKEOL_PCK) bytes} = Total Line Length × 2 - (HACT + HBP + HSS)

Note: TX_BYTE_CYCLES = TX Byte clock cycles to send Active Line bytes on the number of active lanes = div_roundup (HBP + HACT, number of lanes).

Burst Operation Frame Configuration

Burst mode operation can be used to save power provided the system configuration can support the higher tx_byte clock rate. The user should consider if the power saving during LP states is more than reducing the number of active lanes and running in non-burst mode.

The burst mode can only be used if the controller configuration can support it; either using SDI video interface, of DPI interface with a buffer size large enough to prefill the data. The video driver timing from the DPI side will also need to be changed.

Copyright © 2024 Texas Instruments Incorporated

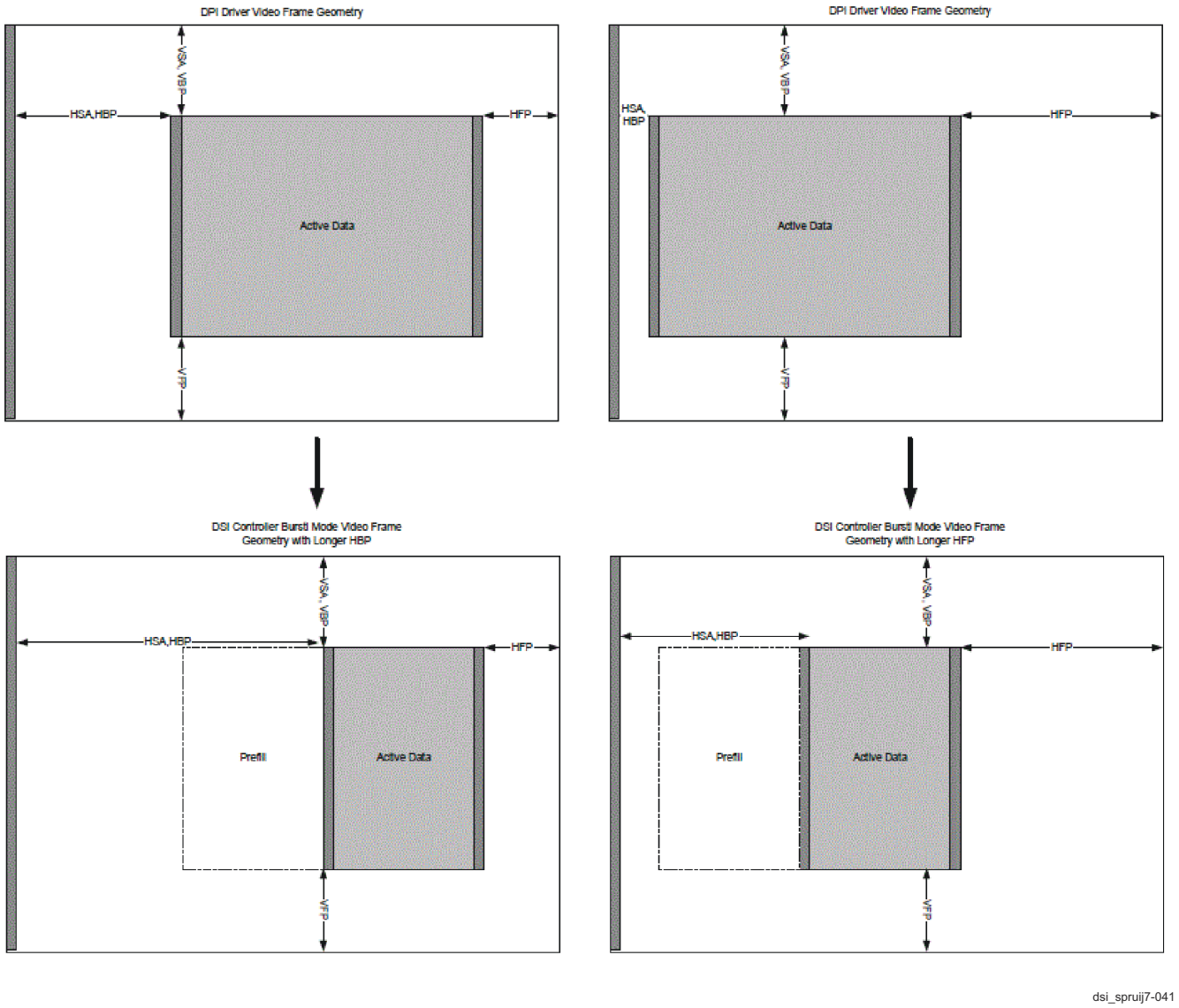


Figure 12-419. DSITX Controller Video Frames for Burst Mode Compared with DPI Driver Side

For example:

The VESA configuration for a 1920x1200 frame at 60-frames per second with normal blanking would have a pixel clock at 193.25 MHz, HSA 200 pixels, HBP 336 pixels, HFP 136 pixels and total line 2592 pixels. This would need to be reconfigured to move the active area earlier in the horizontal line for the DSITX controller, so could be changed to; HSA 20 pixels, HBP 30 pixels, HFP 622 pixels. This would allow the HFP LP state to be almost ¼ of the line (622/2592).

12.6.4.7.9.5 Burst Mode Operation

The DSI IP offers different options for behavior in the BLLP areas:

- Insertion of blanking or null packets.
- Switch to LP (power saving).
- Insertion of commands (if any) + blanking or null packet.

Figure 12-420 shows burst mode operation.

Copyright © 2024 Texas Instruments Incorporated

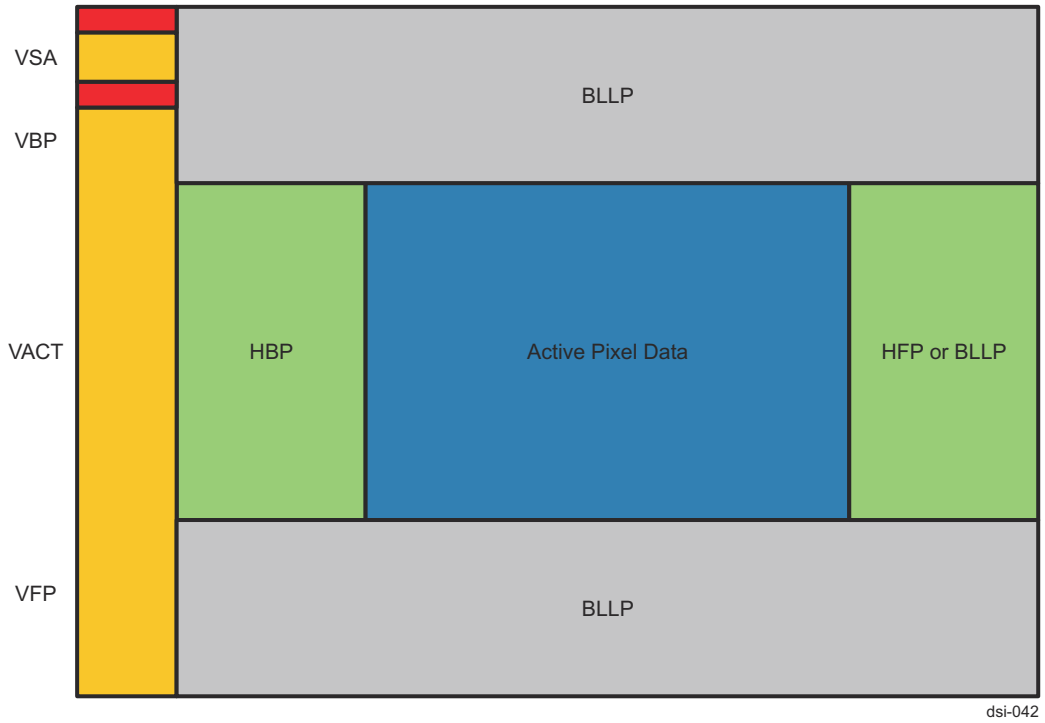


Figure 12-420. Burst Mode Operation

The user should balance the increase in the clock frequencies for transmission of the active display, against the power saving by using BLLP(Null) or LP state.

For Burst with the LP state the register fields must be programmed for burst_lp = 1 and reg_blkeol_mode = 2'b2.

Command Insertion Operation

The DSITX can use the burst operation to allow the blanking/LP stages to be used for DCS/GEN command insertion by programming the size of the space available after the sync of each line.

Only one command packet can be passed within each video line. The decision to insert a command in a video line is done at the beginning of the slot just after the previous video packet. If a command arrives in the middle of the slot, it is not processed in that line, the decision to accept the command will be taken in the next line.

If the packet does not fit in a short slot (size bigger than reg_max_burst_limit and not equal to reg_exact_burst_limit), it is delayed up to next long slot. The signal reg_err_burstwrite is flagged (it is only an information that can help application software control). If its length is longer than reg_max_line_limit, it is discarded and the error reg_err_linewrite is generated.

Read commands and BTA requests are only passed during long slots, as there is no way to calculate their duration exactly. In the case where their duration takes longer time than the vertical blanking period, the error reg_err_longread is flagged.

Copyright © 2024 Texas Instruments Incorporated

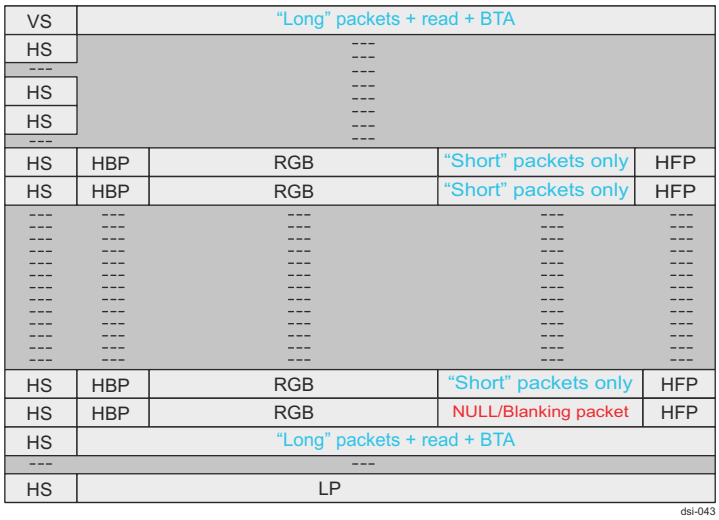
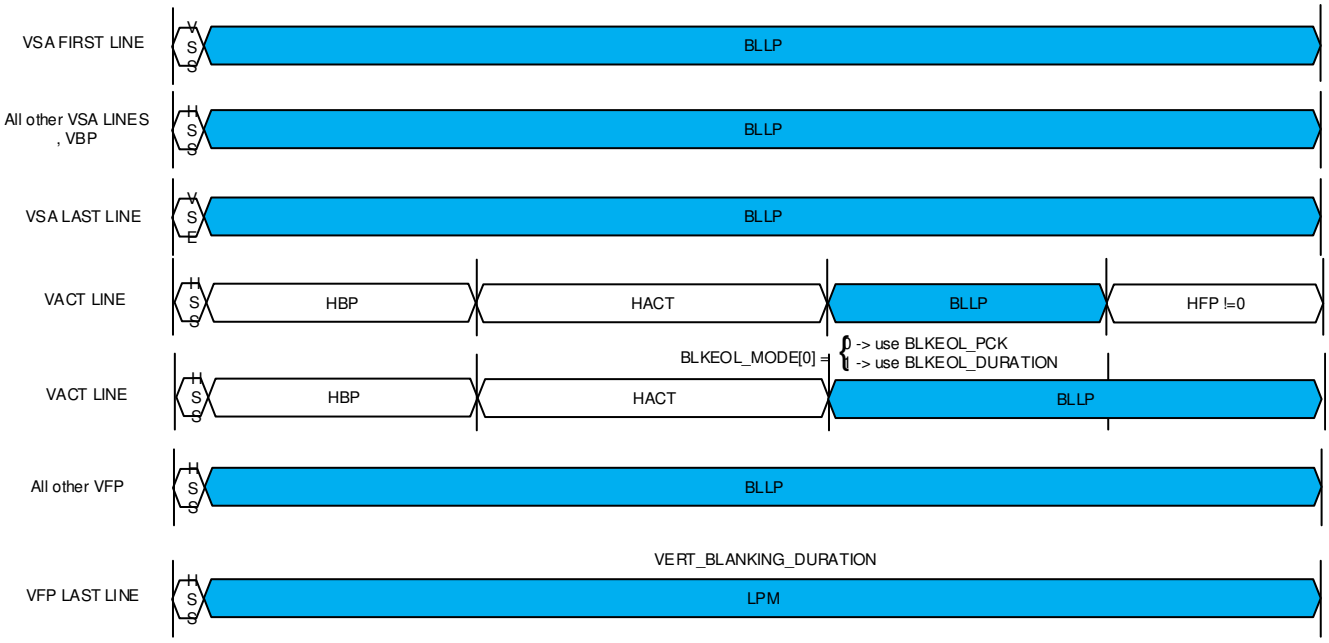


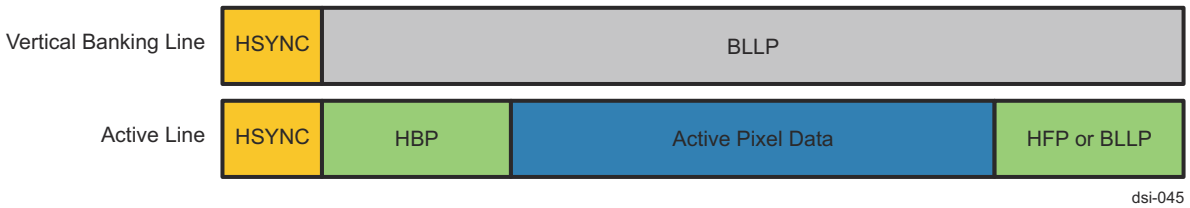
Figure 12-421. Burst Mode Command Locations During Video Frame

Command Insertion Registers in Burst Operation

The burst operation requires the configuraton of the registers to control the size of the packets that will fit in the space available on each type of horizontal line.



In VSA, VBP or VFP: Insertion of short and long command packets, read or BTA are allowed.

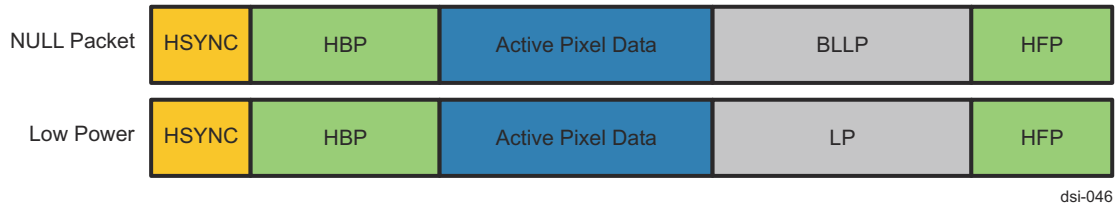


- In VACT: Only short packets can be inserted.
- Command, if inserted, is inserted right after the HSA or HSE or the video packet.
- Only one command can be inserted by line.
- When a command is inserted on a video line, no switch to LP but Null packet insertion.
- Trigger and TE insertion not supported.

Copyright © 2024 Texas Instruments Incorporated

VID_VCA_SETTING1 register:

[16] BURST_LP: After an active line, in burst mode, the system can switch in LP (1) or should complete the line with NULL packet (0).

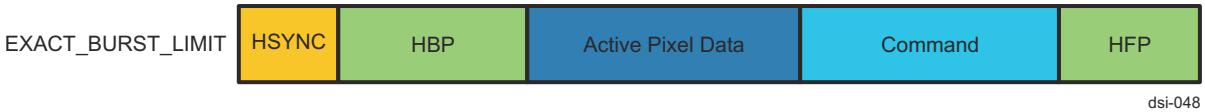


VID_VCA_SETTING1 register:

[15:0] MAX_BURST_LIMIT: Maximum length of a packet (number of bytes) that can be passed during the blanking period at the end of an active line (in burst mode) followed by a NULL packet with no data (6 bytes).

VID_VCA_SETTING2 register:

[15:0] EXACT_BURST_LIMIT:Exact maximum size of the burst packet (packet that fits after pixel data in burst mode), that is with no Null packet insertion afterwards.



VID_VCA_SETTING2 register:

[31:16] MAX_LINE_LIMIT: The "maximum" packet (number of bytes) that can be passed during the vertical blanking period.

If the packet does not fit in a short slot (size bigger than MAX_BURST_LIMIT and not equal to EXACT_BURST_LIMIT), it is delayed up to next long slot.

- If the packet length is longer than MAX_LINE_LIMIT, it is thrown away (an error is asserted).

Table 12-367 outlines the expected behaviour of each packet type during the video frame.

Table 12-367. Behavior of the DSITX Transmission when Both Video and Command are Running

Packet type	Packet Size	Behavior	Error Flagged
VACT Active Line - Read and BTA do not pass			
No packet	-	LP mode null packet sent	
Short packet		Packet passed followed by a null packet	
Long packet	size <reg_max_burst_limit	Packet passed followed by a null packet	
Long packet	size = reg_exact_burst_limit	Packet passed	
Long packet	reg_max_burst_limit < size < reg_exact_burst_limit	Packet delayed, LP mode packet delayed, null packet sent	Burstwrite Burstwrite

Copyright © 2024 Texas Instruments Incorporated

Table 12-367. Behavior of the DSITX Transmission when Both Video and Command are Running (continued)

Packet type	Packet Size	Behavior	Error Flagged
Long packet	reg_exact_burst_limit < size <= reg_max_line_limit	packet is delayed to non-active part of frame: Packet delayed, LP mode packet delayed, null packet sent	Burstwrite Burstwrite
Packet type	Packet Size	Behavior	Error Flagged
Long packet	size > reg_max_line_limit	Packet thrown, LP mode packet thrown, null packet sent	Linewrite Linewrite
Read packet	-	Packet delayed, LP mode Packet delayed, null packet sent	-
BTA request	-	Packet delayed, LP mode Packet delayed, null packet sent	-
VSA, VBP, VFP - Long packet, read and BTA can be passed			
No packet	-	LP mode	-
Short packet	-	Packet passed followed by a null packet	-
Long packet	size < reg_max_burst_limit	Packet passed followed by a null packet	
Long packet	size >reg_exact_burst_limit	packet thrown, LP mode	Linewrite
Read packet	read completed (direction) before end of slot	Packet passed	
Read packet	read not completed (direction) before end of slot	Packet passed	Longread If operation lasts too long
BTA request	-	Packet passed	Longread If operation lasts too long

Note that for burst operation with Low Power, the command will be sent as a high-speed packet even if the request is made to send as Low Power. Also, the next active line will return to using LP during the HFP phase if no other commands are requested.

12.6.4.7.9.6 Example Configurations

1. Here is an example for a VGA frame with 24bpp @60fps using non burst pulse sync mode: With blanking, total size 500 lines, each line is 800 pixels.

400000 DPI cycles/frame @ 24 MHz pixel clock <-> 300000 DSI cycles/frame @ 18 MHz byte clock;

800 DPI cycles/line @ 24 MHz <-> 600 DSI cycles/line @ 18 MHz;

Table 12-368. DPI and DSI Parameters - 1

Parameter	DPI	DSI
VSA	5	5
VBP	5	5
VACT	480	480
VFP	10	5
HSA	40	106 (120-14)
HBP	40	108 (120-12)
HACT	640	1920
HFP	80	234 (240-6)

Copyright © 2024 Texas Instruments Incorporated

Table 12-368. DPI and DSI Parameters - 1 (continued)

Parameter	DPI	DSI
HTOTAL	800 pixel cycles	2400 bytes

burst_mode = 0

sync_pulse_active = 1

sync_pulse_horizontal = 1

BLKLINE_PULSE_PCK = 2380 (2400-20)

REG_LINE_DURATION = 2380

VERT_BLANKING_DURATION = 2384

Note: The controller will send 4 VFP lines then transition to LP for the duration equivalent to 6 lines.

2. Here are some example values for a 24fps UHD frame size with 24bpp using non-burst event mode: With blanking, total size 2250 lines, each line is 4000 pixels.

9000000 DPI cycles/frame @ 216 MHz pixel clock <-> 6750000 DSI cycles/frame @ 162 MHz byte clock

4000 DPI cycles/line @ 216 MHz <-> 3000 DSI cycles/line @ 162 MHz

Table 12-369. DPI and DSI Parameters - 2

Parameter	DPI	DSI
VSA	2	2
VBP	0	0
VACT	2160	2160
VFP	40	1
HSA	40	0
HBP	40	228 (120 + 120 - 12)
HACT	3840	11520
HFP	80	234 (240 - 6)
HTOTAL	4000	12000

burst_mode = 0

sync_pulse_active = 0

sync_pulse_horizontal = 0

BLKLINE_EVENT_PCK = 11990 (12000-10)

REG_LINE_DURATION = 11990

VERT_BLANKING_DURATION = 11996

Note: After the active data the controller will transition immediately to LP for the duration equivalent to 88 lines, giving the maximum power saving between lines.

If any of the DPI related interrupts are triggered, then this highlights that the FIFO depth and/or the vsync_delay settings require to be tuned to the current configuration. Simulating the core operation with the expected clocks is the best way to ensure.

12.6.4.7.9.7 Stereoscopic Video Support

The DSI controller can support stereoscopic display format (SDF) using both command and video modes.

- Command Mode
  - APB Direct Command 3D Commands – Use get_3D_control with Read
  - SDI Command Mode – DCS Writes

Copyright © 2024 Texas Instruments Incorporated

- Video Mode – Single Link Operation
  - VSYNC Parameter1 Definition (L/R, Mode, Format)
  - SDI Video Mode
  - DPI Video Mode – Video stream will expect to follow expected format; frame, line, pixel
    - 3DVSYNC = 0 not currently supported.

The DSI must be configured to enable the 3D operation and identify the format that is being used using the register control to enable the 3D features and the configuration that matches the video stream from the system.

The SDF format requires the system to provide frame and synchronisation information using the two parameter fields within the vertical sync short packet (VSS).

Configured the DSI mctl_3dvideo_ctl register to match format – L/R Pixel, L/R Line, L/R Frame Drive video stream based on orientation and format.

Note: DSI Controller will not perform any on-the-fly pixel manipulation for L-R ordering.

Example Video Operation

For a 3D video stream where the DSI video is given the combined images as one continuous large frame, the VSYNC packet would identify that the left image is sent first, it is a line based format and in landscape mode (see Table 12-370).

Table 12-370. Example Video Operation

Position	D7	D6	D5	D4	D3	D2	D1	D0
Value	0	0	0	0	0	0	1	0
Description	Rsvd	Rsvd	Left First	No Sync	Line-based		Landscape	

mctrl_3dvideo_ctl (0x20)

Table 12-371. VSYNC Parameter 1

[7]	vid_vsync_3d_en	1
[5]	vid_vsync_3d_lr	0
[4]	vid_vsync_3d_second_en	1
[3:2]	vid_vsync_3dformat	00
[1:0]	vid_vsync_3dmode	10

The video stream would then be configured in the DSI to be twice the line length of the normal image.

Copyright © 2024 Texas Instruments Incorporated

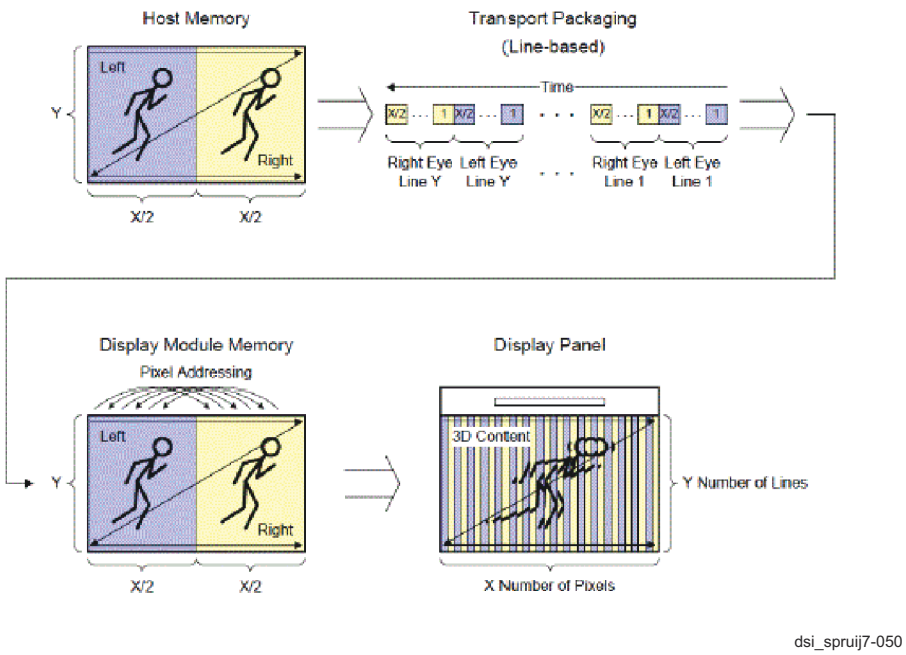


Figure 12-422. Example DPI Image Format for SDF Combining Left And Right Pixels

Note: The vertical operation DSI controller will expect the system to provide the left and right as two individual frames with SYNC to allow the blanking to be generated. The DSI does not insert the vertical blanking area when VSYNC = 0.

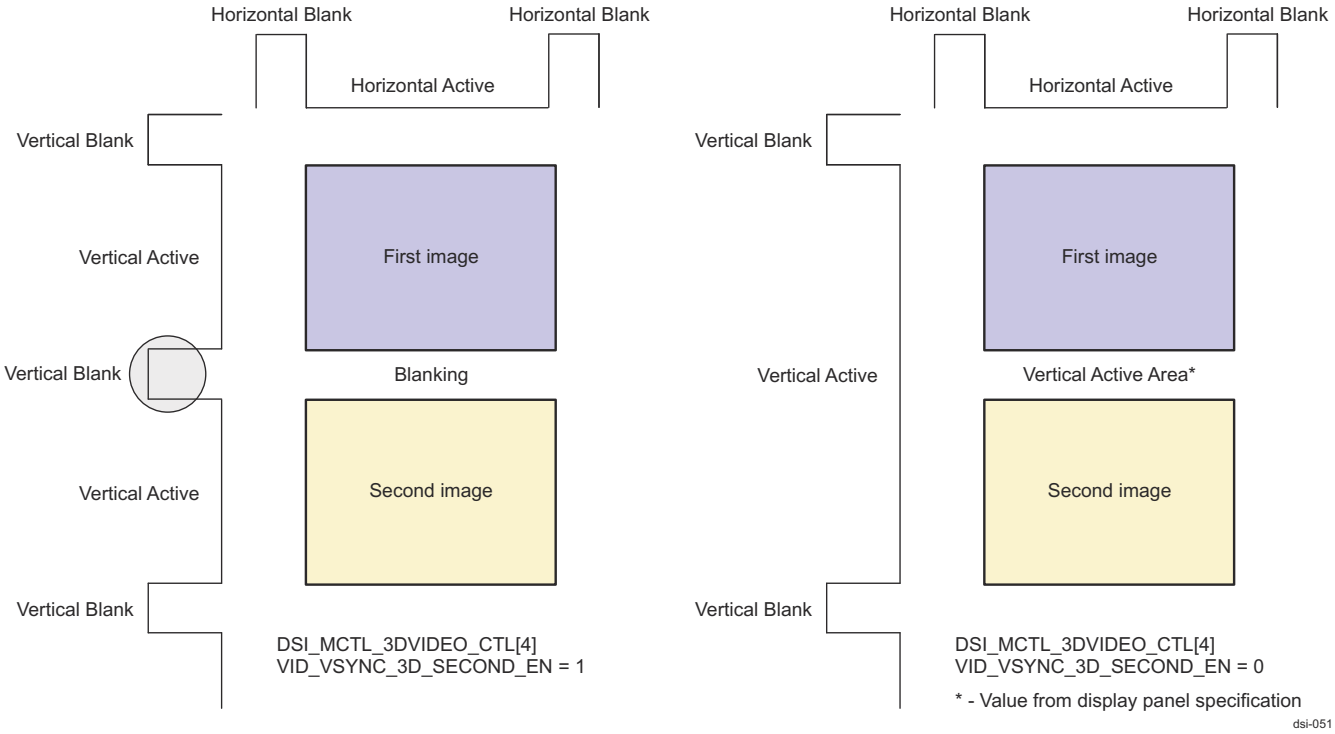


Figure 12-423. 3D Image format for 3DVSYNC splitting first and second images

Copyright © 2024 Texas Instruments Incorporated

12.6.4.7.10 DSITX Video Stream Variable Refresh

The DSITX Controller can be programmed to allow the video stream sequence from a video source to halt between frames and enter Low Power state. The DPI video stream can halt before the new VSYNC is sent and remain in this state until the host video system wants to begin transmitting the new frame.

The DSITX will not perform the normal frame recovery mechanism when this is enabled, however it will continue to recover from line errors until the end of the frame.

Table 12-372. DSITX Video Stream Variable Refresh

Variable refresh rate operation enable control	Configures how the video generation timing is controlled.0, 1 DPI mode operation supports = 1 (enabled). For SDI operation, this parameter can optionally be set to FALSE (= 0) when the system clock and tx_byte_clk are generated from the same reference clock source, or TRUE to allow variable refresh rate control.

Copyright © 2024 Texas Instruments Incorporated

12.6.5 Embedded DisplayPort (eDP) Transmitter

12.6.5.1 EDP Block Diagram

Figure 12-424 shows the high level architecture of EDP and system integration.

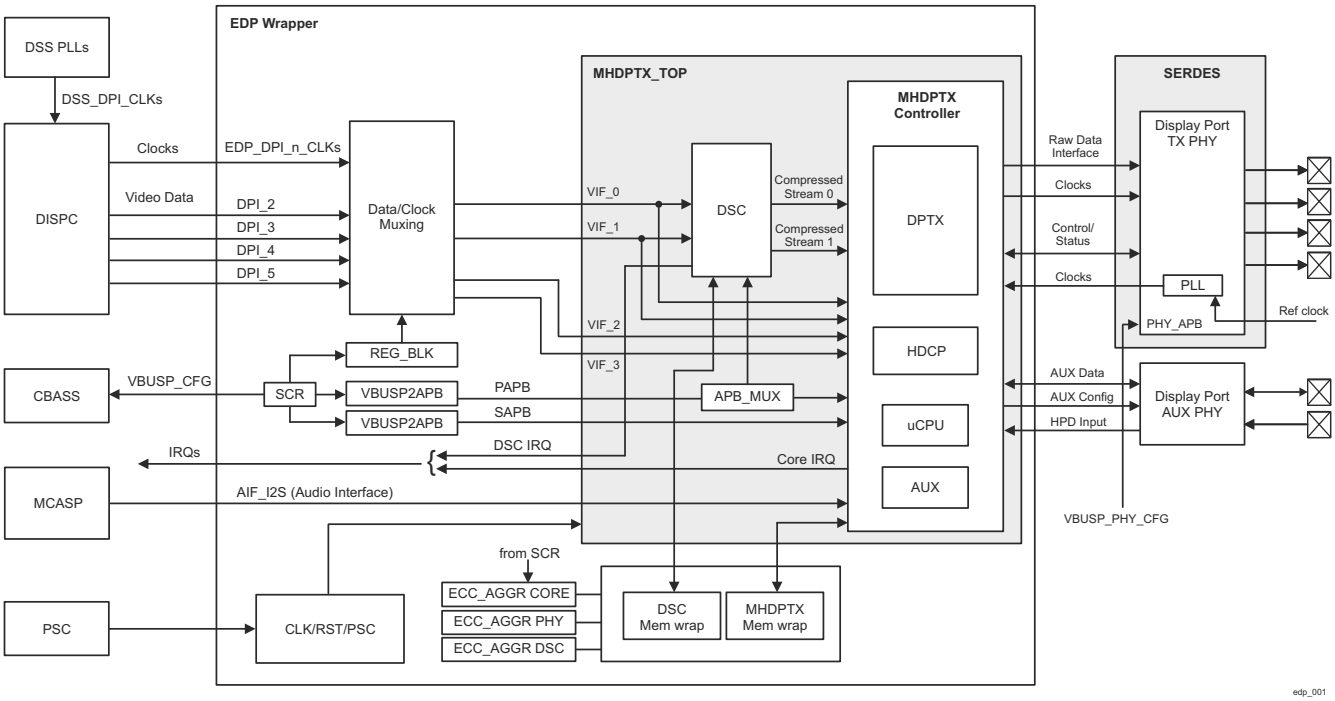


Figure 12-424. EDP System Integration Block Diagram

12.6.5.2 EDP Wrapper Functions

The EDP Wrapper provides logics for following interfaces:

- Interface Bridges - Map VBUS_CFG bus to APB interfaces via CBASS switch and VBUSP2APB bridges
- Video Stream Clock/Data Muxing - Muxes the DPI (from DSS) streams and maps the selected streams to MHDPTX_TOP video interface (VIF) inputs
- SERDES PHY Interface Mapping
- Clock/Reset/Power Management
- Configurations – Provides a register bank to configure/control the top level EDP wrapper functions

12.6.5.2.1 Video Stream Clock/Data Muxing

The EDP wrapper supports up to 6 DPI inputs and internally maps 4 of them to the MHDPTX module, which can support up to 4-stream transmission in Multiple Stream Transport (MST) mode.

Each DPI interface supports the following:

- CEA-861 signaling/timing compatible interface
- Pixel clock rate range from 25MHz to 600MHz
- Frame Resolution - 4K@60Hz equivalent frame size – supporting various frame resolutions and aspect ratios that are equivalent to 4Kx2K@60Hz UHD (Ultra High Definition – 4096x2160 pixels)
- Progressive video timing only
- Maximum Width - 8x1024 pixels wide
- RGB (or YUV444) format only

Table 12-373 describes the DPI interface signals at EDP Wrapper boundary.

Table 12-373. EDP Wrapper DPI Interface Signals

Signal Name[(1)]	Direction	Default Value	Description
dpi_n_data[47:0]	Input	48'd0	Pixel Data

Copyright © 2024 Texas Instruments Incorporated

Table 12-373. EDP Wrapper DPI Interface Signals (continued)

Signal Name[(1)]	Direction	Default Value	Description
dpi_n_m_mdata[47:0]	Input	48'd0	Master Meta Data Only bit [10] secure mode bit is used
dpi_n_de	Input	1'd0	Data Enable (active high)
dpi_n_vs	Input	1'd0	Start of frame or Vsync (active low)
dpi_n_hs	Input	1'd0	Start of line or Hsync (active low)

(1) n = 0 to 5

Figure 12-425 shows the DPI data mapping to VIF ports via muxing in the EDP wrapper.

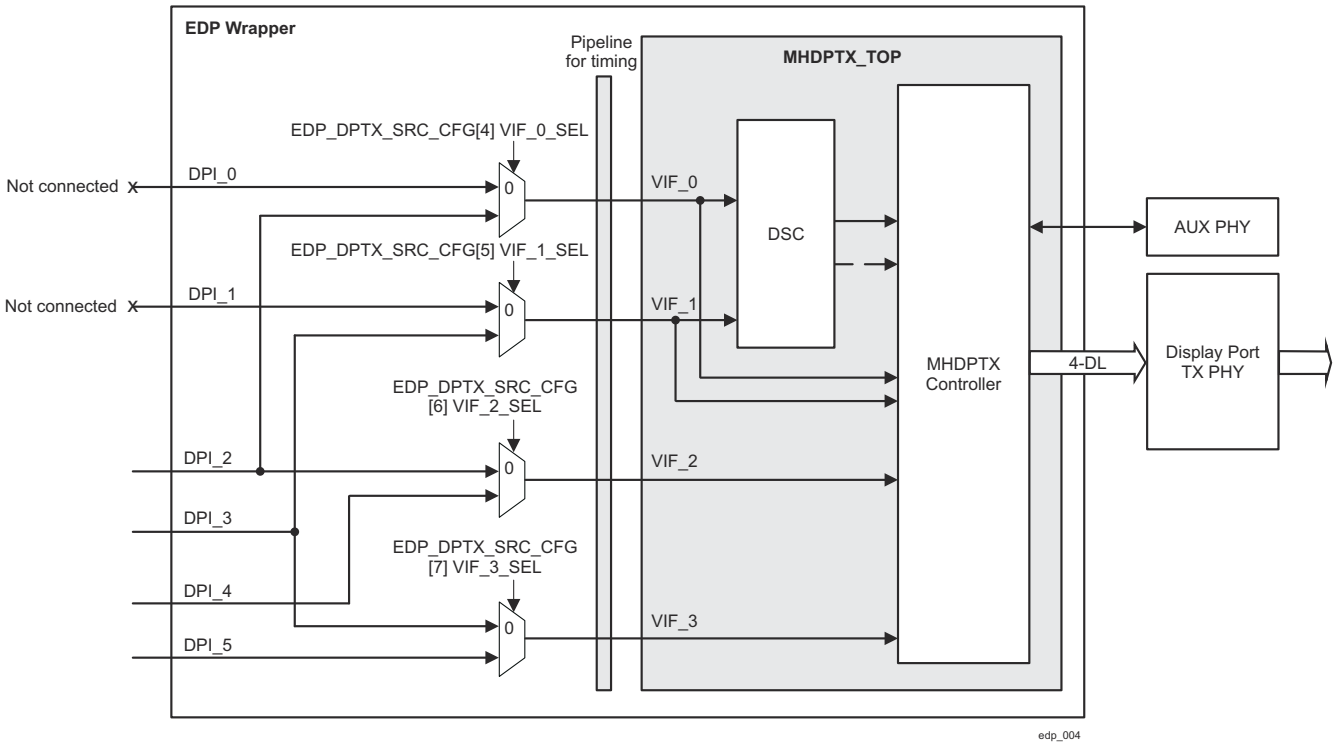


Figure 12-425. DP Wrapper DPI Data Muxing and VIF Mapping

The following data stream transport modes are supported by the DPI data mapping to VIF ports:

SST Mode (Single Stream Transport) – VIF0 used

- DPI_2 - Single Panel Input (optional DSC compression) mapped to VIF0
- DPI_2/DPI_3 - Split Panel Dual Inputs (for DSC compression only) mapped to VIF0 (via DSC) (see Note 1)

MST Mode (Multi-Stream Transport) – All VIFs used

- DPI_2 - Single Panel Input (optional DSC compression) mapped to VIF0
- DPI_3 - Single Panel Input (optional DSC compression in dual panel mode) mapped to VIF1 (see Note 2 and Note 3)
- DPI_2/DPI_3 - Split Panel Dual Inputs (for DSC compression only - two L/R streams compressed separately but outputs combined as one stream) with compressed output mapped to VIF0 (via DSC)
- DPI_2 or DPI_4 - Single Panel Input (no DSC compression) mapped to VIF2
- DPI_3 or DPI_5 - Single Panel Input (no DSC compression) mapped to VIF3

Notes:

1. DSC can be enabled to perform compression on a single input stream as Left and Right panel data for high resolution video (pixel clock > ~400 MHz) that cannot be compressed with a single DSC encoder. In this mode, two input streams share the same video timing (vs/hs/de) and identical input pixel clocks. This mode is referred to as "**split panel mode**".

Copyright © 2024 Texas Instruments Incorporated

2. DSC can also be enabled to perform compression of two unrelated input streams of the same resolution. These streams must share the same pixel input clock since the DSC encoders require two encoder clocks to be synchronous. But, in this case, two streams can be asynchronous in terms of video timing (vs/hs/de). This mode is also referred to as "**dual panel mode**".
3. When DSC is using only one encoder, then it is required to be on Enc0 input (meaning the stream to be compressed must be on DPI_2). In MST mode (where this constraint is applicable), the single stream that is to be compressed must be placed on DPI_2 and therefore the DP sink device (that receives the compressed stream) will be assigned to receive the STREAM_0.

When DSC is in split or dual panel mode, the DSS uses a common clock to send two streams. In the EDP, two DSC input clocks are sourced from the same DPI_2 source clock to keep these clocks to be the same (per DSC requirement) in order to minimize the skew between two DSC encoder clocks.

For detailed description of the clock diagrams, including the clock muxes per data selection, refer to Section 12.6.5.5.1, *Clock Diagram*.

12.6.5.2.2 Secure Video Content Protection

Each DPI_n input port includes a 48-bit DPI_n_M_MDATA bus from DSS (DISPC) which carries attribute sideband signals associated with the DPI interface. The EDP sets the security status (DPISECURE) of the incoming DPI interface using the DPI_n_M_MDATA[10] bit as shown in Table 12-374.

Table 12-374. EDP Secure Video Contention Protection

Sideband bit(s)	Name	Description
DPI_n_M_MDATA[10]	DPISECURE	DPI-np secure bit 1: DPI-np is secure 0: DPI-np is not secure

The EDP performs a simple security check on the selected DPI connection by comparing the value of the selected DPISECURE bit (per EDP_DPTX_SRC_CFG[7-4] VIF_n_SEL bits (where n = 0 to 3)) to the EDP_DPTX_VIF_SECURE_MODE_CFG[3-0] VIF_n bits.

If the selected DPISECURE bit is set to 1, the video content is considered to be secure and the connection (transmission via DPTX) is only made if the EDP_DPTX_VIF_SECURE_MODE_CFG[3-0] VIF_n bits are also set to 1 (indicating the display is also secure). Otherwise, the EDP forces the DSS_DPI_DATA to be 0h while leaving the sync signals alone – in effect, causing the video to go black. If the DPISECURE bit is set to 0, then the connection is also enabled.

12.6.5.2.3 DPI_DATA Input Pixel Format Supported

Figure 12-426 shows the supported by the EDP pixel packing formats in the DSS_DPI_DATA.

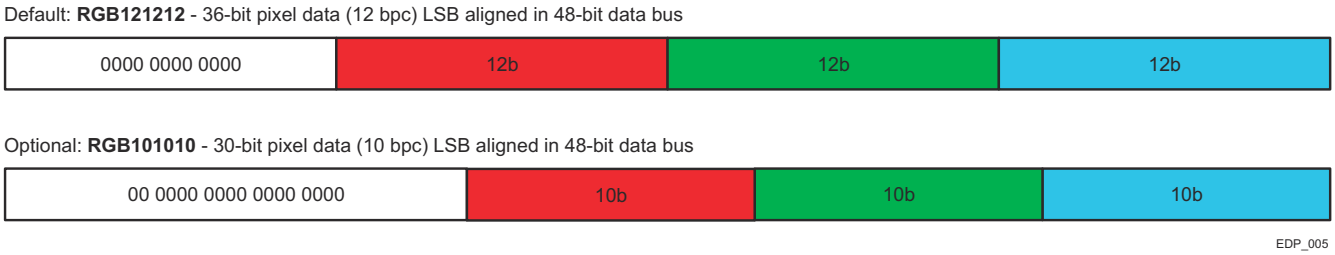


Figure 12-426. EDP Wrapper Supported Input DSS_DPI_DATA Pixel Formats

The selection between these two formats is made per VIF_n source by configuration EDP_DPTX_SRC_CFG[11-8] VIF_n_IN30B parameters. The default is 36-bit format (RGB121212). Optionally, the 30-bit format (which is the pixel format when the DPI_data is generated from the DSS's merge/split module) can be selected to match the DSS configuration.

Internally, the EDP wrapper performs following pixel data/component realignments to match the input pixel data alingment required by the MHDPTX core (MSB alignment) and DSC (native LSB alignment).

Copyright © 2024 Texas Instruments Incorporated

Figure 12-427 shows the internal mapping from the LSB packed pixel bus to MSB-aligned container packed bus: the 30 or 36-bit pixel data remapped to 48-bit (16 bpc (bits per component) MSB aligned within color component). The MHDPTX core extracts MSB 8/10/12-bits from each 16-bit color component to include in the display port video transport packet (see Table 12-375, *EDP Video Interface Pixel Mapping (MSB Mapping)*).

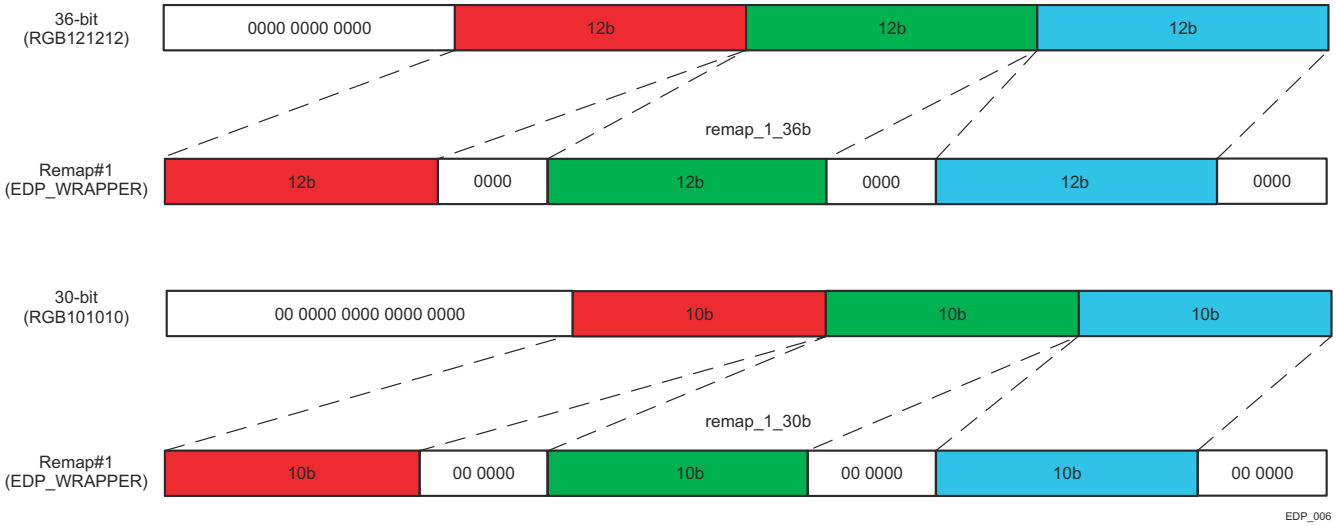


Figure 12-427. EDP Wrapper Internal Pixel Bus Remapping (First Stage)

DSC Input Data Re-alignment (Optional for DSC enabled VIF_0 and VIF_1) – From the 48-bit aligned data bus, the EDP wrapper performs additional re-alignement (LSB aligned component data per DSC requirement) and pixel data alignment per DSC input processing bit size configuration.

For this LSB alignment, EDP_DPTX_DSC_CFG[6-5] DSC_1/0_10BPC bits must be configured to extract significant 8 or 10-bit component data from the input pixel bus, as shown on Figure 12-428.
The parameters in these bits must match the parameters in the EDP_CORE_ENC0_MAIN_CONF_P/ EDP_CORE_ENC1_MAIN_CONF_P[1-0] IPUT_BPC bitfields.

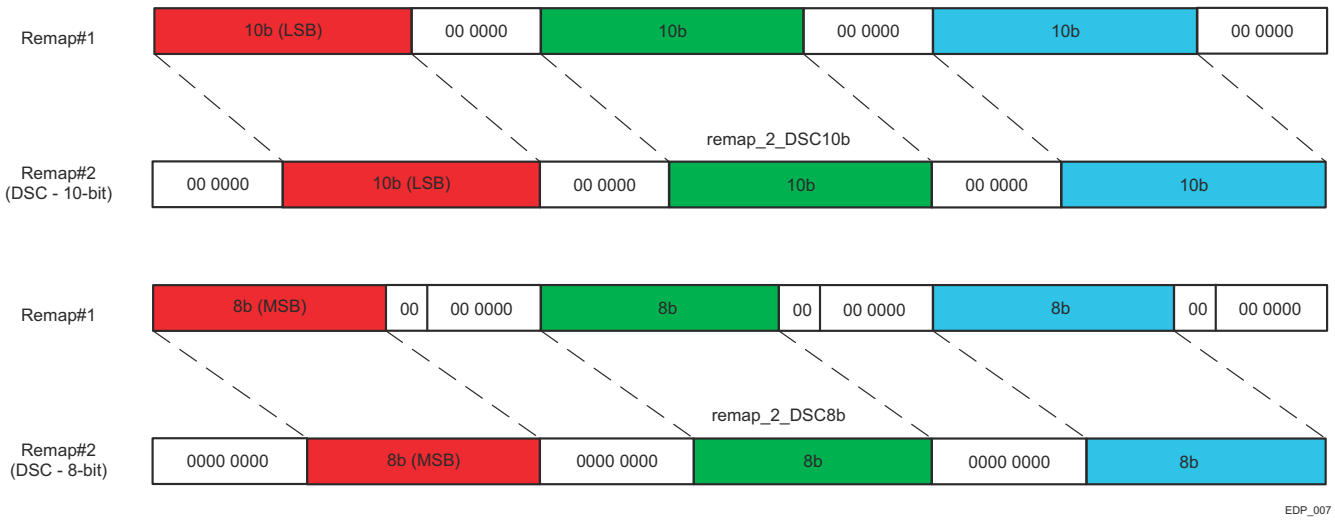


Figure 12-428. EDP Wrapper Internal Pixel Remapping (Additional for DSC Bound Pixel Data Bus)

Pixel data alignment within the MHDPTX_TOP should be set to be MSB always to enable the pixel mapping, as shown in Table 12-375. DSC bound pixel data bus is to be LSB aligned and requires no other MHDPTX_TOP configuration.

Copyright © 2024 Texas Instruments Incorporated

Table 12-375. EDP Wrapper Video Interface Pixel Mapping (MSB Mapping)

Video Bus	Source RGB/YCbCr444 (C of YCbCr422[2])				
	Bit Width	8-bit	10-bit	12-bit	16-bit[1]
Channel 2	[47:40]	R/Cr[15:8] / C[15:8]	R/Cr[15:6] / C[15:6]	R/Cr[15:4] / C[15:4]	R/Cr[15:0] / C[15:0]
	[39:38]				
	[37:36]				
	[35:32]				
Channel 1	[31:24]	G/Y[15:8]	G/Y[15:6]	G/Y[15:4]	G/Y[15:0]
	[23:22]				
	[21:20]				
	[19:16]				
Channel 0	[15:8]	B/Cb[15:8]	B/Cb[15:6]	B/Cb[15:4]	B/Cb[15:0]
	[7:6]				
	[5:4]				
	[3:0]				

(1) 16-bit source format is not supported. When selected, lower 4 bits are set to 0x0.
(2) YCbCr422 format is locally generated from YCbCr444 input from DSS (by a simple chroma decimation).

12.6.5.2.4 Audio Input Interface

The audio input interface provides the following features:

- One I2S interface configured to support multiple physical audio channels (up to 8-channels)
- Frame Format – TDM mode
  - Multiple Physical Channels N = 1, 2, or 4
  - Variable TDM time slots M = 2 or 8
  - Time slot size: 32, 24, or 16
  - Variable word length (28, 24, 20, 16) configured as left or right justified
- I2S_DATA - 4-bits wide
- Supports data rate up to 8-channel L-PCM @192 kHz (Buffer dependent)
- SPDIF mode is not supported

12.6.5.2.4.1 Audio I2S Signals/Timing

Table 12-376 shows the I2S signals and timing.

Table 12-376. EDP Audio I2S Signals/Timing

Signal Name	Direction	Default Value	Description
AIF_I2S_CLK	Input	1'd0	I2S clock
AIF_I2S_DATA[3:0]	Input	4'd0	I2S data for up to 4*M logical channels. Number of physical channels: 1, 2, or 4 M – number of time slots: 2 or 8
AIF_I2S_WS	Input	1'd0	I2S word-select indication. In TDM mode, WS = 0 indicates channel 0.

Figure 12-429 and Figure 12-430 show the I2S timing.

Copyright © 2024 Texas Instruments Incorporated

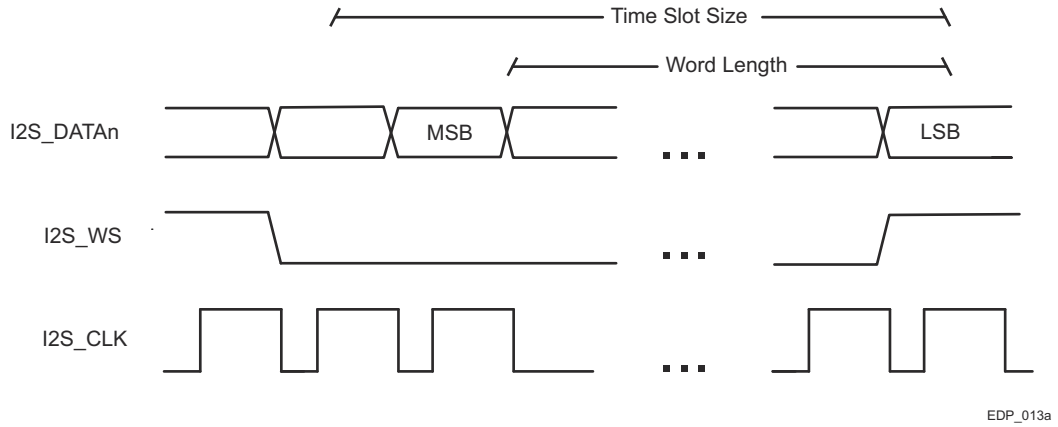


Figure 12-429. EDP Audio I2S Timing - Bit Allocation (Right Justification)

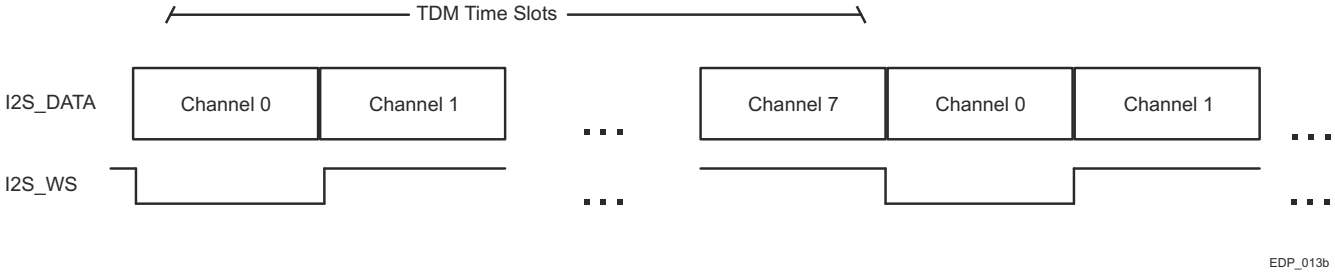


Figure 12-430. EDP Audio I2S Timing - TDM Time Slote Allocation (M = 8)

12.6.5.2.4.2 Audio I2S Clock Frequency

The bit clock frequency is defined as below:

- Bit clk freq = Sampling_rate * Number_of_bits_per_channel * Number_of_channels

For 16-bit/44.1 kHz stereo channels:

- I2S_CLK (freq) = 44.1 kHz x 16 x 2 = 1.4112 MHz

For 24-bit/192 kHz stereo channels:

- I2S_CLK (freq) = 192 kHz x 24 x 2 = 9.216 MHz

12.6.5.3 EDP Transmitter Controller Subsystem (MHDPTX_TOP)

The EDP Transmitter Controller Subsystem (MHDPTX_TOP) integrates the following major components:

- Display Stream Compression (DSC) v1.1 Encoder – VESA Compression Engine
- Display Port TX Controller (MHDPTX Controller) - EDP/DP Transmitter Controller

12.6.5.3.1 Display Stream Compression Encoder (DSC)

This module performs the VESA Digital Stream Compression encoding for the Display Port TX interface. The DSC contains two parallel Hard Slice Encoders, which support compression for VESA Display Port transmitter.

12.6.5.3.1.1 DSC Encoder Features

The DSC Encoder supports the following main features:

- Maximum image width and height per encoder slice can be configured with up to 8K
  - Programmable slice width and height
  - All slices must be the same height, including the final slice of the picture
- 8 bits/component (24 bits/pixel)
- 10 bits/component (30 bits/pixel)
  - An encoder built for 10 bits/component will be able to run in either 8 or 10 bits/component mode based on a register value

Copyright © 2024 Texas Instruments Incorporated

- RGB/YCbCr with 4:4:4 sampling support only on the input streams
- Single or Dual Pixel input(s)
  - Single pixel input (one encoder slice processing only)
  - Dual pixel inputs
    - Split Panel (Left/Right) streams – requires low skew between streams
- Support for all DSC 1.1 encoding mechanisms
  - MMAP, BP, MPP predictions and ICH
  - Flatness detection and signaling
- Supports CBR (Constant Bit Rate) mode only - Configurable target bpp (bits per pixel), non-integer values supported. The following modes are available:
  - 8 bpp and 12 bpp compressed bit rates for 8 bits/component - Corresponds to 3:1 and 2:1 compression
  - 10 bpp and 15 bpp compressed bit rates for 10 bits/component - Corresponds to 3:1 and 2:1 compression
  - 12 bpp and 18 bpp compressed bit rates for 12 bits/component
- Each Hard Slice encoder instance can be configured (with the parameter NB_SS_ENC) to support the following number of slices per line
  - Supports 1 or 2 slices (soft slices) per line
    - When configured with support for 2 slices per line, the encoder support dynamically the configuration of both 1 and 2 slices per line.
  - Thus a total of up to 4 slices per line can be supported with two Hard Slice Encoders
- Encoder data flow supported:
  - Synchronous Streaming mode – no back pressure support
- Implements the following active internal diagnostic mechanisms:
  - Self-Checking during VBLANK
  - Control output diagnostics
  - SRAM Protection
  - Configuration and Status Register (CSR) Protection
  - Fault avoidance mechanism

12.6.5.3.1.2 Usage Models for EDP

Table 12-377 summarizes the usage models supported by DSC instantiated in the MHDPTX_TOP subsystem.

Table 12-377. EDP DSC Usage Models

	Usage Models		
	Single Panel Mode (SST/MST)	Split Panel Mode (SST/MST)	Dual Panel Mode (Only in MST)
ENC0	Stream 0 Full Frame	Stream 0 Left Frame	Stream 0
ENC1	Not Used	Stream 0 Right Frame	Stream 1

The shaded boxes in Table 12-377 indicate the encoder slice enabled for EDP usage. The active paths for these two modes are shown on Figure 12-431.

Copyright © 2024 Texas Instruments Incorporated

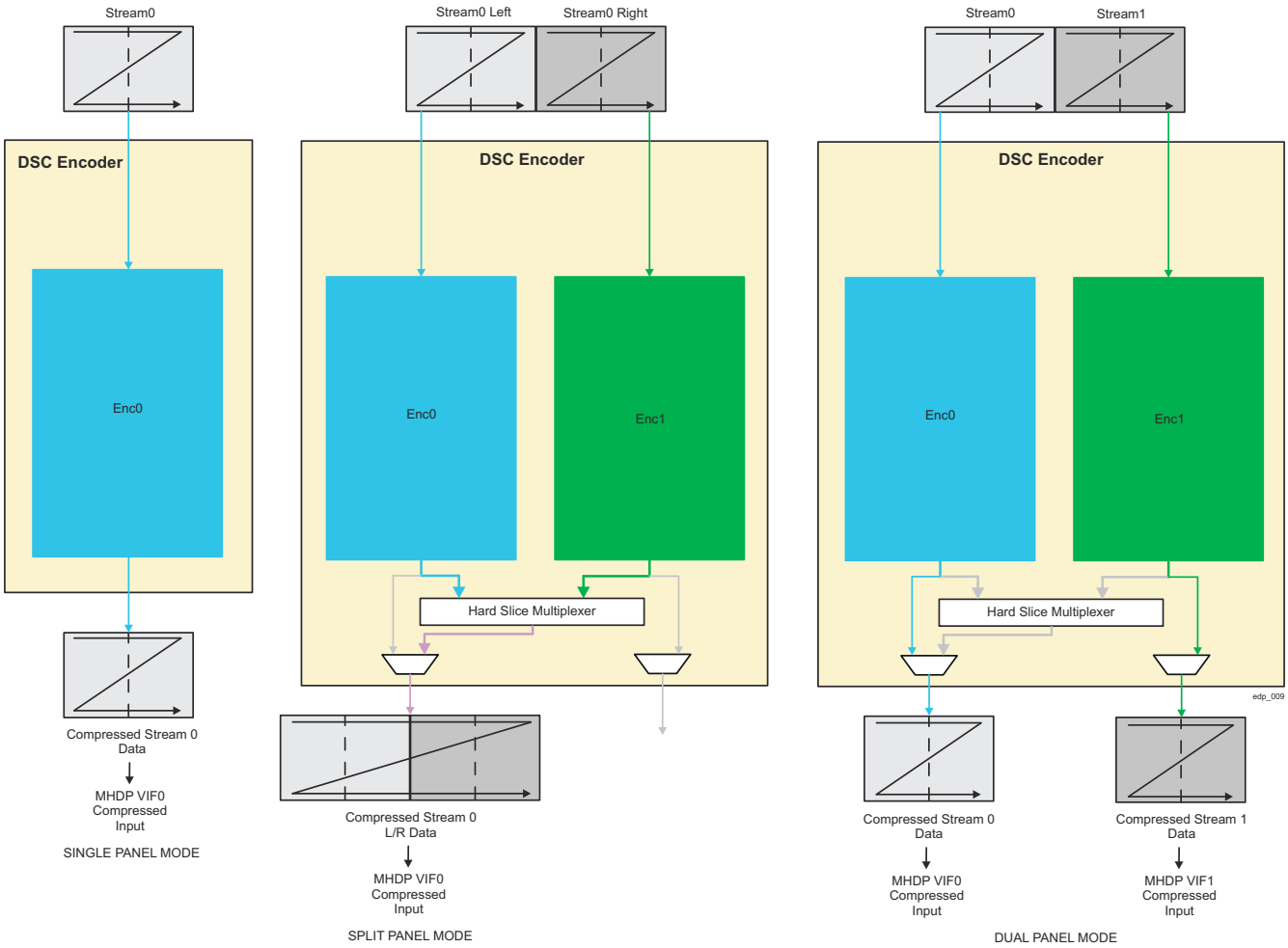


Figure 12-431. EDP DSC Single Input/1-Transport Link Mode and Split Panel/Dual Panel Mode

12.6.5.3.2 Display Port Transmitter Controller (MHDPTX Controller)

Figure 12-432 shows a simplified block diagram of the EDP Display Port Transmitter Controller (MHDPTX) that is included in the EDP subsystem.

Copyright © 2024 Texas Instruments Incorporated

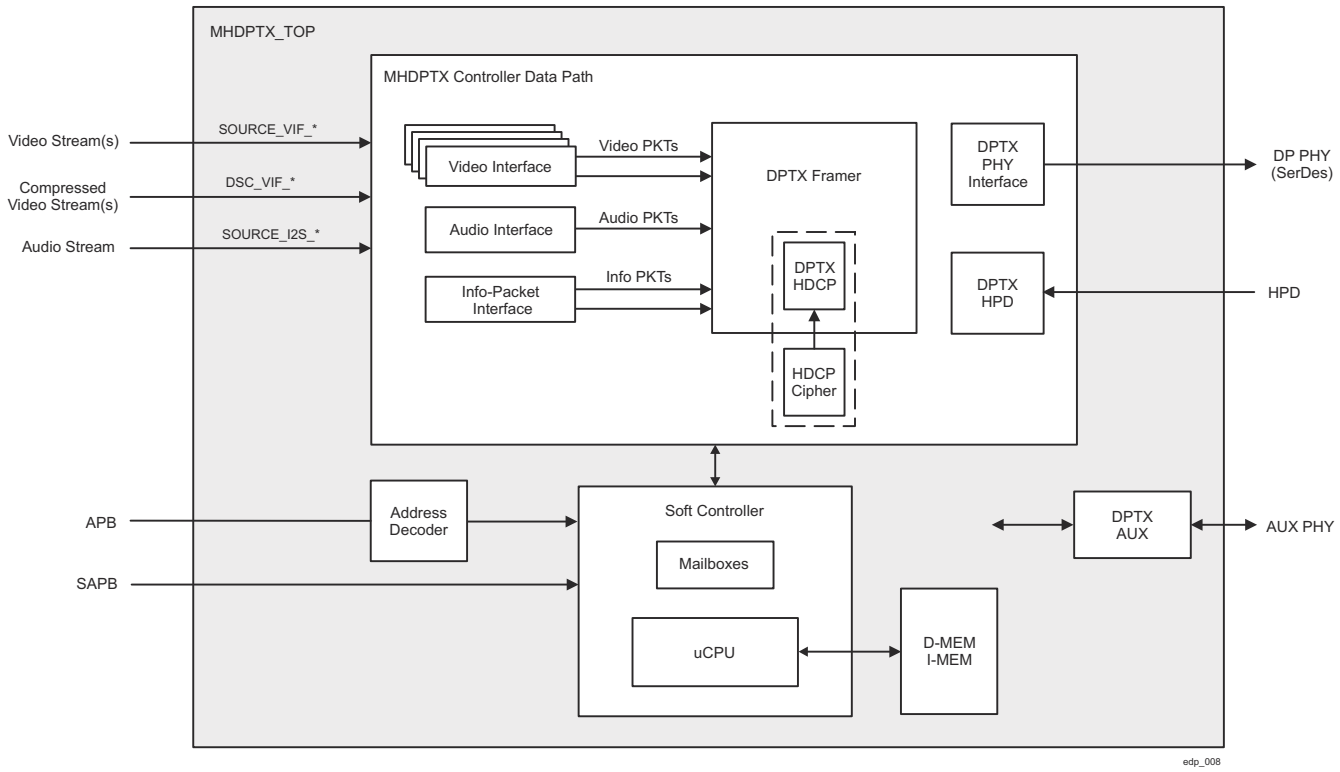


Figure 12-432. EDP Display Port Transmitter Controller Functional Block Diagram

n = 0 to 3

12.6.5.3.2.1 EDP Transmitter Controller Mode Configurations

This section describes the configuration mode input signals of the MHDPTX_TOP module.

SOURCE_SECURE_MODE

This mode configuration signal sets the "source_secure_mode" of the MHDPTX's APB interface to provide additional security control on the host interface. The main APB interface in "debug" mode can be used to access almost all internal registers as well as IRAM/DRAM of the internal uCPU. This signal must be set to 0 to load FW during boot time. By setting SOURCE_SECURE_MODE to 1, this functionality can be disabled to only allow mailbox and basic control register accesses.

In EDP, this signal is mapped to a wrapper level configuration register within the EDP_CFG region: EDP_DPTX_IPCFG[0] APB_SECURE_REG_BLOCK_EN. If tempering of this register should be blocked, the EDP_CFG region should be firewalled to grant access only to a secure host only.

If the DP firmware regions are needed to be protected (to prevent either corruption or altering of the firmware), a secure host can change this setting to 1 after initializing the firmware memories. Then, a non-secure host can come and take the CPU out of reset and enable the DP function without an ability to alter the firmware.

SOURCE_CRYPTO_DIS

This mode configuration signal is used to completely disable HDCP functionality by disabling Crypto module (module responsible for keys encryption during authentication) . Refer to *EDP eFuse Tie-Off*, for eFuse tie-off information.

12.6.5.4 EDP AUX_PHY Interface

The AUX PHY module is required for each Display Port interface. The AUX_PHY module is integrated with the EDP controller as shown on Figure 12-433. The AUX PHY has no configuration interface. All configuration and control signals are supplied from EDP_CORE_APB memory-mapped registers.

Copyright © 2024 Texas Instruments Incorporated

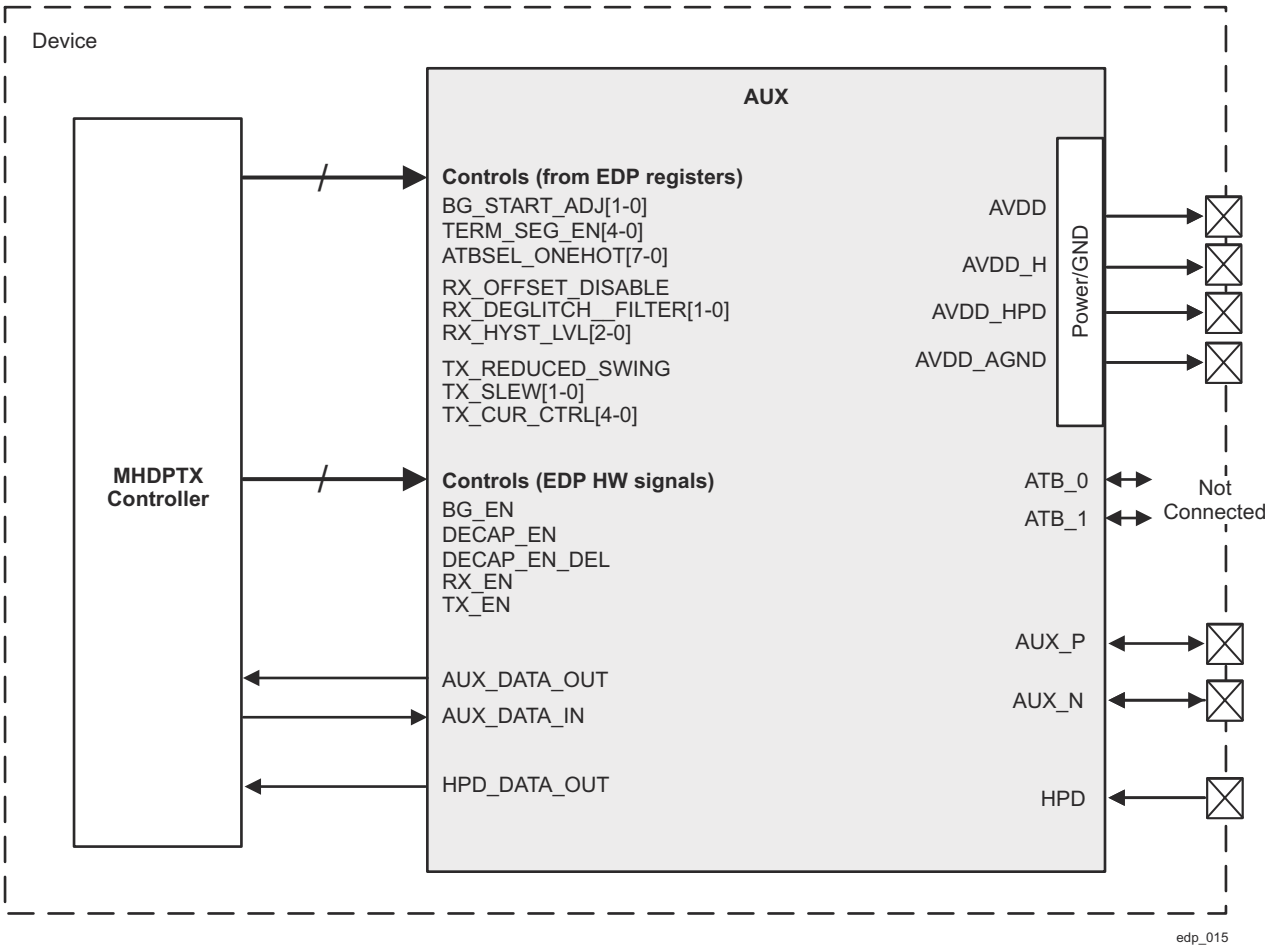


Figure 12-433. EDP Interface to AUX PHY

The configurations and controls in Figure 12-433 are specified through EDP_CORE_AUX_CONFIG_P and EDP_CORE_AUX_CTRL_P registers.

12.6.5.5 EDP Clocks

12.6.5.5.1 Clock Diagram

Figure 12-434 shows the input clocks of eDP and muxing on the DPI input clocks to match the pixel data selection.

Copyright © 2024 Texas Instruments Incorporated

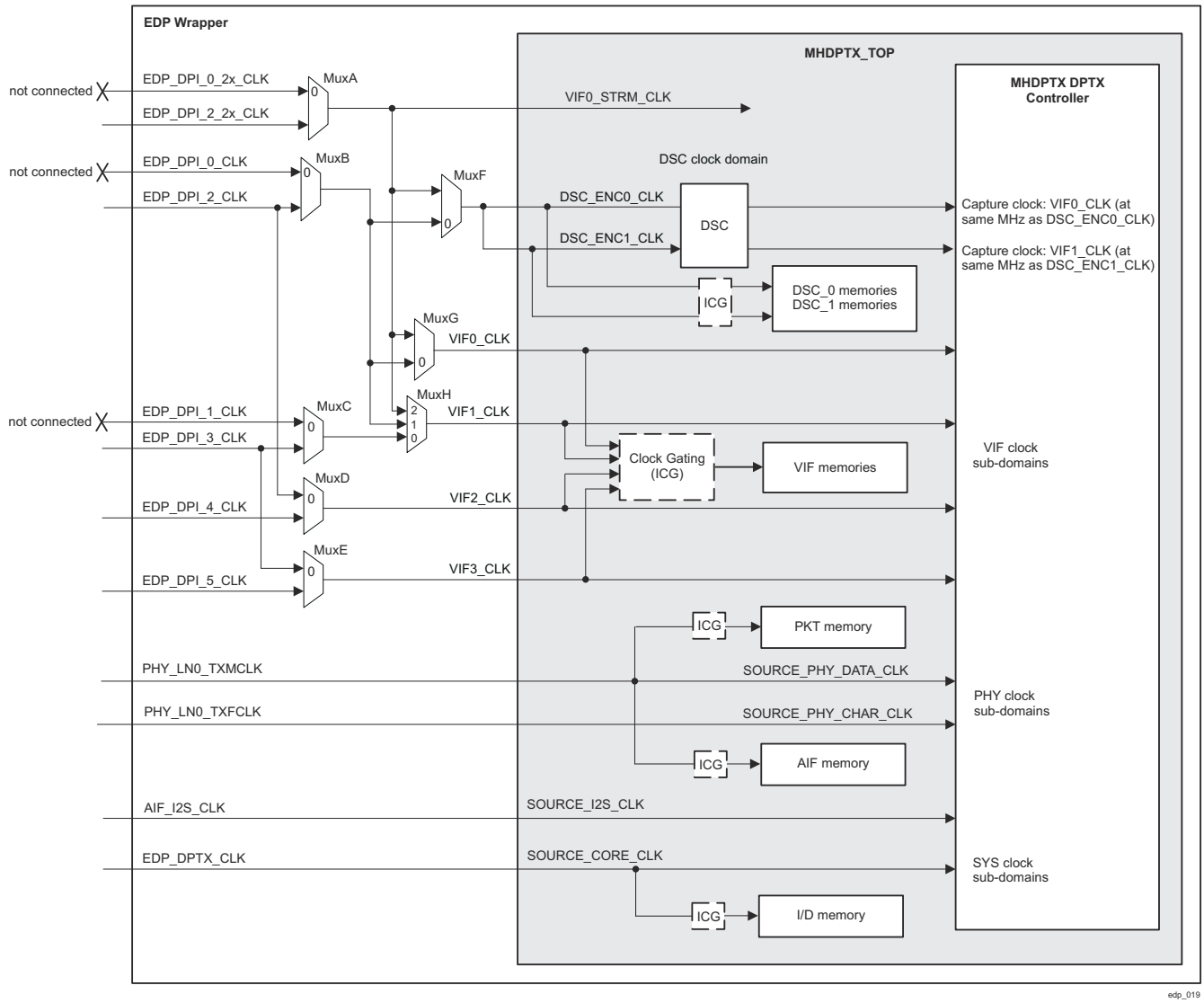


Figure 12-434. EDP Clock Diagram

12.6.5.5.1.1 DPI Interface Clock Sourcing

In default (non-DSC enabled) modes, the EDP performs the matching pixel data clock selection (MuxA - MuxE) based only on the VIF input source selection (EDP_DTPX_SRC_CFG[7-4] VIF_n_SEL register fields) settings.

For VIF0_CLK, the source selection in non-DSC enabled mode comes from the output of MuxA (EDP_DPI_0_2x_CLK or EDP_DPI_2_2x_CLK). These clocks are full pixel data frequency clocks.

When DSC is enabled, the DSC_ENC0_CLK and DSC_ENC1_CLK clocks are sourced (MuxF) from:

- Full pixel data frequency clock (MuxA selected clock) - when DSC is configured in a non-split panel mode (EDP_DPTX_DSC_CFG[1-0] MODE_SEL != 0h AND EDP_DPTX_DSC_CFG[2] SPLIT_PANEL_EN = 0h)
- Half pixel data frequency clock (MuxB selected clock) – when DSC is configured in the split panel mode (EDP_DPTX_DSC_CFG[1-0] MODE_SEL = 2h AND eDP_DPTX_DSC_CFG[2] SPLIT_PANEL_EN = 1h)

VIF0_CLK and VIF1_CLK selection goes through additional muxing (MuxG/MuxH) to match the DSC encoder clocks.

12.6.5.5.1.2 Memory Clock Gating

Memory clock gating is handled at the wrapper based on the scheme described below.

Functional mode clock enable (functional clock gating – for memory):

Copyright © 2024 Texas Instruments Incorporated

- DPTX firmware memory (I/D mem) clocks are enabled when EDP_DPTX_IPCFG[1] FW_MEM_CLK_EN = 1.
- Memory clocks for all DPTX data memories in the normal functional mode are only enabled when the corresponding source (video/audio/pif) is enabled per DPTX_SRC_CFG.
  - Clocks for the video stream memories (vif_mem) are enabled whenever the stream is enabled (EDP_DPTX_SRC_CFG[3-0] VIF_n_EN).
  - Clocks for the packet memories (pkt_mem) are enabled whenever the packet stream interface is enabled (EDP_DPTX_SRC_CFG[3-0] VIF_n_EN).
  - Similarly clock for the audio buffer (aif_mem) is enabled whenever the audio stream is enabled (EDP_DPTX_SRC_CFG[16] AIF_EN).
- Clocks for the DSC memories (DSC_ENC0 and DSC_ENC1 mem) are enabled based on the ENC0/ENC1 usage (EDP_DPTX_DSC_CFG[1-0] MODE_SEL). When EDP_DPTX_DSC_CFG[1-0] MODE_SEL is not 0 (that is, at least one DSC encoder is enabled), the wrapper enables VIF_n_MEM clock based on the DSC usage

Functional mode clock enable override:

- Clocks for all ECC memories can be forced to turn on (that is, functional clock gating bypassed) during ECC diagnostic access mode by setting EDP_ECC_MEM_CFG[0] CLK_EN = 1.

12.6.5.5.1.3 PHY Clock Connections

There are multiple clocks from/to the physical interface. Figure 12-435 shows clocks used for the PHY clock domain logic (TXMCLK/TXFCLK) and for the return data clock (TXCLK). Not shown in the diagram are RX mode input clocks unused by EDP (TX only mode) – PHY_INn_RXCLK/RXFCLK/REFCLK.

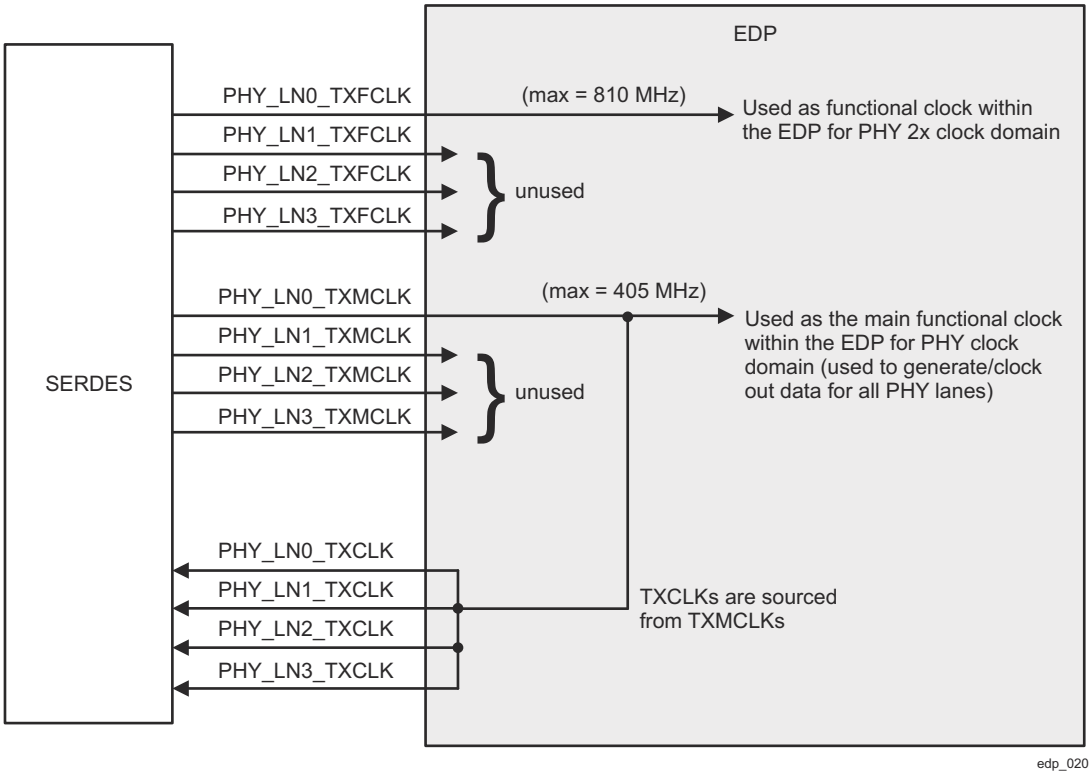


Figure 12-435. EDP PHY Clock Connections

12.6.5.5.2 Clock Groups

Following are the major groups of clocks in the EDP subsystem:

- System clock – This is the VBUSP configuration clock. It is used to source the clock for all internal configuration interfaces (VBUSP and PAB) and the core logic of the MHDPTX controller.
- DPI clocks - These are video stream source clocks. These can be sourced either directly from PLL or DSS (that is, DPI_CLK outputs, DSS sourced mode).

Copyright © 2024 Texas Instruments Incorporated

- PHY I/O clocks - These are clocks used to transmit display port raw data to the SERDES or the clock used to receive audio data from MCASP.

Table 12-378 describes clocks at the EDP boundary.

Table 12-378. EDP Clocks

Clock	Freq Min	Freq Max	Source	Description
EDP_DPTX_CLK	125 MHz	125 MHz	System PLL	Functional clock (all VBUSP/APB/DPTX_core logics are clocked by this clock) Valid Freq: 100 – 200 MHz (125 MHz chosen to be 1/4x of CBASS 500 MHz clock)
EDP_DPI_2_2x_CLK	25 MHz	600 MHz	Video PLL or DSS EDP_DPI_2x_CLK	DPI Streaming Clock – Typically 1x of the corresponding EDP_DPI_2_CLK. But, in DSC split-panel mode, 2x of the EDP_DPI_2_CLK
EDP_DPI_2/3/4/5_CLK	25 MHz	600 MHz	Video PLL or DSS EDP_DPI_CLK	DPI Clock – pixel clock of the EDP_DPI_DATA bus. In DSS-sourced mode, EDP_DPI_n_CLK is the pixel clock of the EDP_DPI_n_DATA bus. In PLL-sourced mode, the pixel clock needs to be internally muxed between EDP_DPI_2_2x_CLK and EDP_DPI_2_CLK depending on the EDP_DPI_DATA type (split panel data or not).
EDP_AIF_I2S_CLK	1 MHz	125 MHz	MCASP	Audio I2S Clock (192 kHz clock)
PHY_LN0_TXMCLK (SOURCE_PHY_DATA_CLK)	81 MHz	405 MHz	DP-SERDES (PHY) PLL	PHY Data clock (1/2 Char Clock)
PHY_LN0_TXFCLK (SOURCE_PHY_CHAR_CLK)	162 MHz	810 MHz	DP-SERDES (PHY) PLL	PHY Char Clock

12.6.5.6 EDP Resets

The EDP subsystem has one synchronous active low reset input. The entire subsystem is reset by this reset. All resets for different clock domains (both synchronous and asynchronous) are generated internally using this reset.

All resets within the MHDPTX_TOP are asynchronous. Each clock's sub-domain resets are synchronized with respective software reset.

12.6.5.7 EDP Interrupt Requests

The interrupt signals generated by the EDP subsystem and propagated to SoC level are shown in Table 12-379.

Table 12-379. EDP Subsystem Interrupts

Interrupt	Description
EDP_INTR[3:0]	EDP Controller Interrupt Event (Functional)
EDP_INTR_ASF[6:0]	EDP Controller ASF Error Interrupt

12.6.5.7.1 EDP_INTR Interrupt Description

Table 12-380 provides details on the EDP_INTR[3:0] interrupt lines. Each of the interrupts listed in Table 12-380 has a corresponding set of memory-mapped interrupt registers (status/mask/clear).

Table 12-380. EDP_INTR Interrupts

Interrupt Bit	Source	Description	Status Register	Mask Register	Clear Register
[3]	DSC	DSC Enc1 interrupt event detected	EDP_CORE_ENC1_INT_STAT_P	EDP_CORE_ENC1_INT_MASK_P	EDP_CORE_ENC1_INT_CLR_P
[2]		DSC Enc0 interrupt event detected	EDP_CORE_ENC0_INT_STAT_P	EDP_CORE_ENC0_INT_MASK_P	EDP_CORE_ENC0_INT_CLR_P

Copyright © 2024 Texas Instruments Incorporated

Table 12-380. EDP_INTR Interrupts (continued)

Interrupt Bit	Source	Description	Status Register	Mask Register	Clear Register
[1]	MHDPTX Controller	DPTX SIRQ - Secure APB domain interrupt event. Set when an interrupt in the status register is reported.	EDP_CORE_APB_STATUS_S	EDP_CORE_APB_INT_MASK_S	-
[0]		DPTX PIRQ - General DPTX interrupt event. Set when an interrupt in the status register is reported.	EDP_CORE_APB_INT_STATUS_P	EDP_CORE_APB_INT_MASK_P	-

12.6.5.7.2 EDP_INTR_ASF Interrupt Description

Table 12-381 provides details on the EDP_INTR_ASF[6:0] interrupts lines. Each of the interrupts listed in Table 12-381 has corresponding set of memory-mapped interrupt registers (status/mask/clear).

Table 12-381. EDP_INTR_ASF Interrupts

Interrupt Bit	Source	Description	Status Register	Mask Register	Clear Register
[6]	ECC_AGGR_DSC	ECC Aggregator Uncorrected Error Interrupt	EDP_ECC_DSC_DED_STATUS_REG0	EDP_ECC_DSC_DED_ENABLE_SET_REG0	EDP_ECC_DSC_DED_ENABLE_CLR_REG0
[5]	ECC_AGGR_PHY	ECC Aggregator Uncorrected Error Interrupt	EDP_ECC_PHY_DED_STATUS_REG0	EDP_ECC_PHY_DED_ENABLE_SET_REG0	EDP_ECC_PHY_DED_ENABLE_CLR_REG0
[4]	ECC_AGGR_CORE	ECC Aggregator Uncorrected Error Interrupt	EDP_ECC_CORE_DED_STATUS_REG0	EDP_ECC_CORE_DED_ENABLE_SET_REG0	EDP_ECC_CORE_DED_ENABLE_CLR_REG0
[3]	DSC	ASF Corrected interrupt detected in DSC	EDP_CORE_ENC_ASF_INT_STAT_P	EDP_CORE_ENC_ASF_INT_MASK_P	EDP_CORE_ENC_ASF_INT_CLR_P
[2]		ASF Un-Corrected interrupt detected in DSC			
[1]	MHDPTX Controller	ASF Corrected (NonFatal) event detected in MHDPTX Controller. Set if non-fatal error occurs.	EDP_CORE_ASF_INT_STATUS(1)	EDP_CORE_ASF_INT_MASK(1)	-
[0]		ASF Un-corrected (Fatal) event detected in MHDPTX Controller. Set when fatal error occurs.			

(1) Each interrupt can be individually defined as fatal or non-fatal in the EDP_CORE_ASF_FATAL_NONFATAL_SELECT register.

12.6.5.8 EDP Embedded Memories

12.6.5.8.1 MHDPTX Controller Memories

Table 12-382. EDP MHDPTX Controller Memories

Name	ECC	Description
IRAM	Yes	uCPU Instruction Memory. Contains uCPU FW, loaded by host during boot time.
DRAM	Yes	uCPU DATA Memory. Contains uCPU data-memory, mailbox for communication between uCPU and host processor and EDID segment. In DisplayPort mode, it contains DPCD registers and AUX mailbox. In HDMI mode, it contains SCDC standard registers. When HDCP is supported, it contains keys and protected information.
PKT_MEM_n (n = 0 to 3)	Yes	Source Packet Memory. Contains up to 16 info-frames. Accessed by host processor (over APB) for write and by HD Display TX Controller for read.

Copyright © 2024 Texas Instruments Incorporated

Table 12-382. EDP MHDPTX Controller Memories (continued)

Name	ECC	Description
VIF_MEM_n (n = 0 to 3)	No (No ECC for the video buffer)	Video Memory. Elastic buffer, compensating the difference between the pixel clock and the line clock.
AIF_MEM	Yes	Audio Memory. Buffer audio samples during H active period.

12.6.5.8.2 DSC Memories

Table 12-383. EDP DSC Memories

Name	ECC	Description
ENC0/1_LB	Yes	Line Buffer RAM. Stores the reconstructed pixels that wil be re-used for next line processing.
ENC0/1_OB0	Yes	Output Buffer RAM. Holds (as FIFO) results of encoder slices.
ENC0/1_SSM_S	Yes	Sub-stream Mux Size RAM. Holds context information required to generate the proper order of the mux words.
ENC0/1_SSM_D	Yes	Sub-stream mux balance FIFO RAM. Holds stream data for re-ordering.

12.6.5.8.3 ECC Aggregation

The tasks of ECC detection/correction of 'ECC enabled' memories are handled within the MHDPTX Controller. The EDP wrapper provides 'ECC error injection' using a combination of ECC_Wrapper (to access the memory) and ECC_Aggregator (to allow system to access the ECC injection logic). An ECC_Aggregator can access multiple ECC_Wrappers of the same clock group. Table 12-384 shows how the ECC memories (and the EDC_CTRL for Parity Inv) in the EDP are grouped to three ECC aggregators.

Table 12-384. EDP ECC_Aggregator (Based on Clock Group)

Memory / Logic	ECC Vector_ID	ECC_Aggregator
IRAM	0	ECC_AGGR_CORE
DRAM	1	
EDC_CTRL (Parity)	2	
PKT_MEM_0	0	ECC_AGGR_PHY
PKT_MEM_1	1	
PKT_MEM_2	2	
PKT_MEM_3	3	
AIF_MEM	4	
VIF_MEM_0	-	N/A (No ECC for the video buffer)
VIF_MEM_1	-	
VIF_MEM_2	-	
VIF_MEM_3	-	
ENC0_LB	0	ECC_AGGR_DSC
ENC0_SSM_S	1	
ENC0_SSM_D	2	
ENC0_OB0	3	
ENC1_LB	4	
ENC1_SSM_S	5	
ENC1_SSM_D	6	
ENC1_OB0	7	

Copyright © 2024 Texas Instruments Incorporated

Furthermore, each ECC_Aggregator is connected to the VBUSP_CFG via the main CBASS SCR (with asynchronous bridge). All ECC aggregator bound VBUSP_CFG transactions are mapped to a separate RSEL (in EDP case, RSEL=3), see Figure 12-436.

Each ECC_Aggregator and the asynchronous VBUSP interface of the aggregator are clocked using the read clocks used to read the memory, as follows:

- ECC_AGGR_CORE: EDP_DPTX_CLK
- ECC_AGGR_PHY: SOURCE_PHY_DATA_CLK (PHY_LN0_TXMCLK)
- ECC_AGGR_DSC: EDP_DPI_0_CLK or EDP_DPI_2_CLK (selected by mux)

The clocks to some ECC memories may be disabled when the associated video or audio channel is not enabled. If none of the memories for an ECC aggregator is enabled (for example, DSC memory clocks are disabled, if the DSC is not enabled), then the clock for the aggregator is also disabled and the aggregator is disconnected from the CBASS (resulting in null return for any VBUSP access to the aggregator)

But, if any of the ECC memories connected to an ECC aggregator is enabled, the clock for the ECC aggregator must also be enabled. Any access to the ECC_CTRL of the disabled memory will result in TIMEOUT error (interrupt generated) since the memory clock is not running.

If ECC diagnostics/access are needed for all ECC memories whether the memories are enabled or not, the ECC software can set the clock enable override configuration register - EDP_ECC_MEM_CFG[0] CLK_EN to bypass the clock gating.

Even with this bit set, the ECC_CTRL connected to the ECC_AGGR_PHY aggregator may not get any clock if the PHY is not enabled and properly configured. The EDP_PHY_CLK_STATUS[0] VALID status register tells whether the PHY data clock is running or not.

The DSC clock is sourced from the system video PLL and therefore it should be running. Therefore, the ECC software just need to ensure the clock gating is not active and/or set the ECC CLK_EN override bit to get the clock to the memory.

The ECC_CORE Aggregator clock should be running whenever the EDP is enabled. There is no clock gating to the IRAM/DRAM memories.

Copyright © 2024 Texas Instruments Incorporated

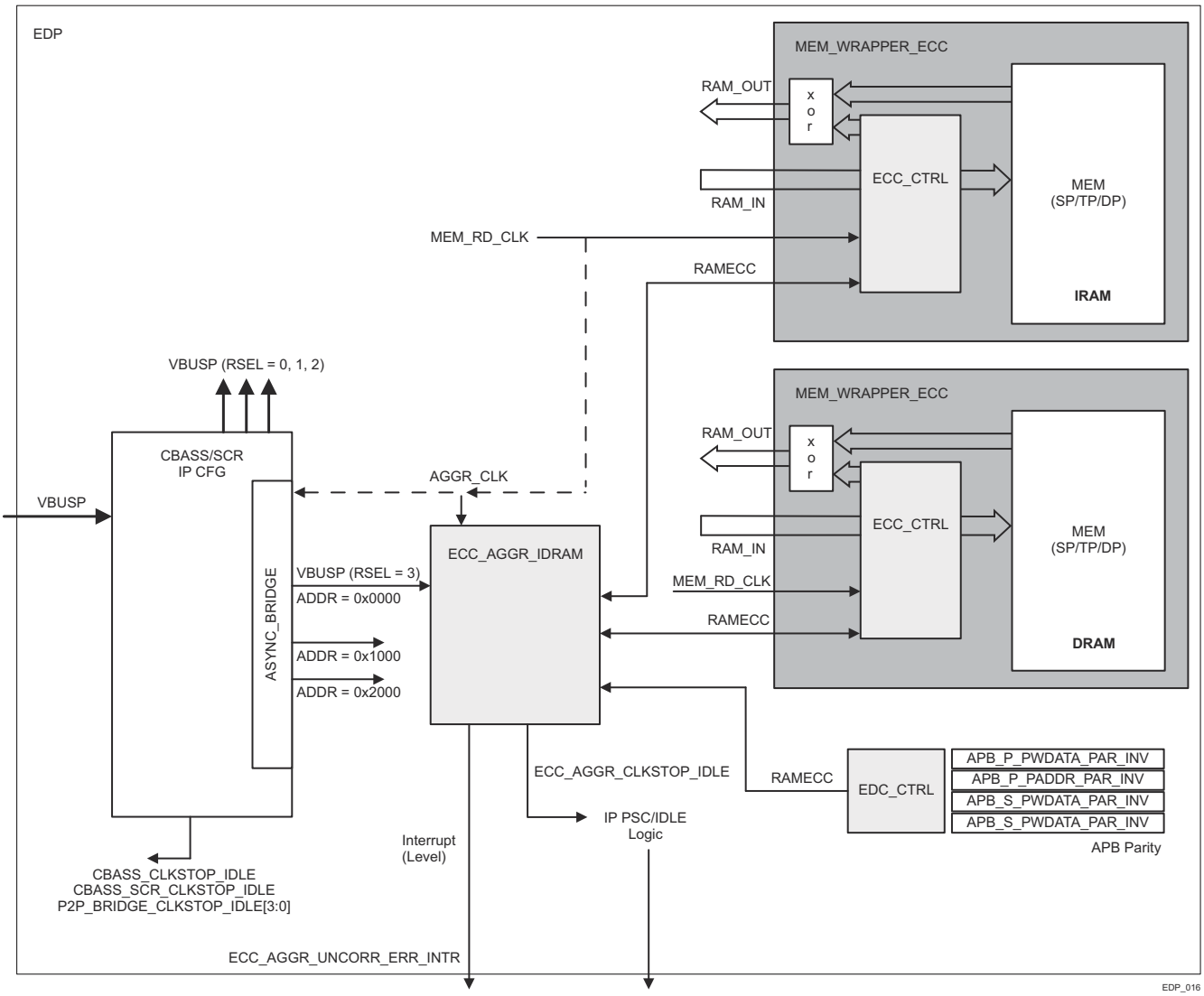


Figure 12-436. EDP ECC_WRAP/ECC_AGGR and PARITY_INV Logic Connection

12.6.5.9 EDP Programmer's Guide

This chapter describes the programming guidelines for the MDPTX Controller and EDP PHY.

12.6.5.9.1 EDP Controller Programming

12.6.5.9.1.1 MHDPTX Register/Memory Regions

Table 12-385 provides a summary of the bus interfaces and the MMR regions mapped to each interface in this module.

Table 12-385. EDP Memory Map Region - Firewall Secure Mode Settings

Region Name	Firewall Secure Mode Settings[(1) (2)]
INTG_CFG_VP	Should be Secure
V2A_CORE_VP_REGS_APB	Secure (for Secure display) / Non-secure
V2A_S_CORE_VP_REGS_SAPB	Must be Secure always (HDCP configuration)
MHDPTX_WRAPPER_ECC_AGGR_CORE_CFG	Secure (for Secure display) / Non-secure
MHDPTX_WRAPPER_ECC_AGGR_PHY_CFG	Secure (for Secure display) / Non-secure
MHDPTX_WRAPPER_ECC_AGGR_DSC_CFG	Secure (for Secure display) / Non-secure

(1) Firewall secure mode settings:

Copyright © 2024 Texas Instruments Incorporated

- The V2A_S_CORE_VP_REGS_SAPB (secure APB) region must be restricted to secure host access only (for HDCP key protection)
- The INTG_CFG_VP region should be set to secure host access only since this region controls the security of the DPI ports and controls the clock enables.
- Other regions should be set to secure, if the display connected is to be secure.

(2) Word-access only for RSEL (1 and 2) – accessing APB and SAPB regions of MHDTPX core. Any non-word access request will result in either rstatus=1 or sstatus=1.

Table 12-386 shows the MHDPTX Controller register/memory regions that are accessible during various operational modes:

Table 12-386. EDP MHDPTX Controller APB/SAPB Memory Map and Access Modes

Description	Address Base [19:0]	Register Bank	Direct access (x= APB (x= APB /SAPB)	Mailbox access (x= APB (x= APB /SAPB)	FW (uCPU) access	APB debug (Bootmode) access
Main configuration and Mailbox control registers. Each APB interface has its own set of these registers.	0x00000	APB_CFG	x			x
Source digital PHY control	0x00800	SOURCE_PHY		x	x	x
Clocks and Reset	0x00900	SOURCE_CAR		x	x	x
Clock Meters	0x00a00	CLOCK_METERS		x	x	x
Video Interface 0 control	0x00b00	SOURCE_VIF		x	x	x
Video Interface 1 control	0x00b20	SOURCE_VIF		x	x	x
Video Interface 2 control	0x00b40	SOURCE_VIF		x	x	x
Video Interface 3 control	0x00b60	SOURCE_VIF		x	x	x
DPTX Digital PHY	0x02000	DPTX_PHY		x	x	x
DPTX HPD	0x02100	DPTX_HPD		x	x	x
DPTX Framer	0x02200	DPTX_FRAMER		x	x	x
DPTX Stream	0x02200	DPTX_STREAM		x	x	x
DPTX Main control	0x02300	DPTX_GLBL		x	x	x
DPTX HDCP SM	0x02400	DPTX_HDCP		x	x	x
DPTX Auxiliary	0x02800	DP_AUX		x	x	x
DPTX Stream 0	0x03000	DPTX_STREAM		x	X	x
DPTX Stream 1	0x03080	DPTX_STREAM		x	X	x
DPTX Stream 2	0x03100	DPTX_STREAM		x	X	x
DPTX Stream 3	0x03080	DPTX_STREAM		x	X	x
HDCP Crypto	0x04000	Crypto		x	x	-
HDCP Cipher	0x05000	Cipher Accessible only through SAPB mailbox		x (SAPB)	x	-
memory	0x10000 … 0x1ff00	IMEM			x	x
Data memory	0x20000 … 0x2ff00	DMEM			x	x
Audio decoder	0x30000	SOURCE_AIF_DECODER	APB			x
SDP control stream 0	0x30800	SOURCE_PIF	APB			x
SDP control stream 1	0x30840	SOURCE_PIF	APB			x
SDP control stream 2	0x30880	SOURCE_PIF	APB			x
SDP control stream 3	0x308c0	SOURCE_PIF	APB			x
Registers related with IPS configuration	0x30A00	IPS_REGS	APB			x
Fault reporting module	0x30B00	ASF	APB			x

Copyright © 2024 Texas Instruments Incorporated

Table 12-386. EDP MHDPTX Controller APB/SAPB Memory Map and Access Modes (continued)

Description	Address Base [19:0]	Register Bank	Direct access (x= APB (x= APB /SAPB)	Mailbox access (x= APB (x= APB /SAPB)	FW (uCPU) access	APB debug (Bootmode) access
DSC encoder	0x30C00 - 0x30F00	DSC	APB			x

Access Modes:

- APB Debug (Boot Mode) Access - After a hardware reset, MHDPTX comes up in this mode – allowing APB access to all regions including IMEM/DMEM (for FW download)
- Direct Access - Once the boot mode initialization is complete, APB_CTRL register is cleared. This takes it out of the Boot mode and into the direct access mode in which only the mailbox control (APB_CFG space) and regions marked as APB in the direct access column.
- Mailbox Access - While in Direct Access mode, all configurations of the DPTX for DP operation are done through the APB mailbox accesses. Secure transactions (secure refers to transactions to protect "secure" content) are done through the secure mailbox access via SAPB interface. The SAPB access is made through the secure EDP vbusp region (must be configured as such in the cfg Firewall)
- FW (uCPU) Access – This column indicates register/memory bank accessible by FW access.

12.6.5.9.1.2 Boot Sequence

After reset, the uCPU is disabled. The full MHDPTX Controller address space is accessible for the host processor for debugging purposes, and the I-MEM and D-MEM.

The host processor must perform the following:

- Load the FW into the memories.
- Enable the uCPU.
- Verify that FW is running as expected.

Copyright © 2024 Texas Instruments Incorporated

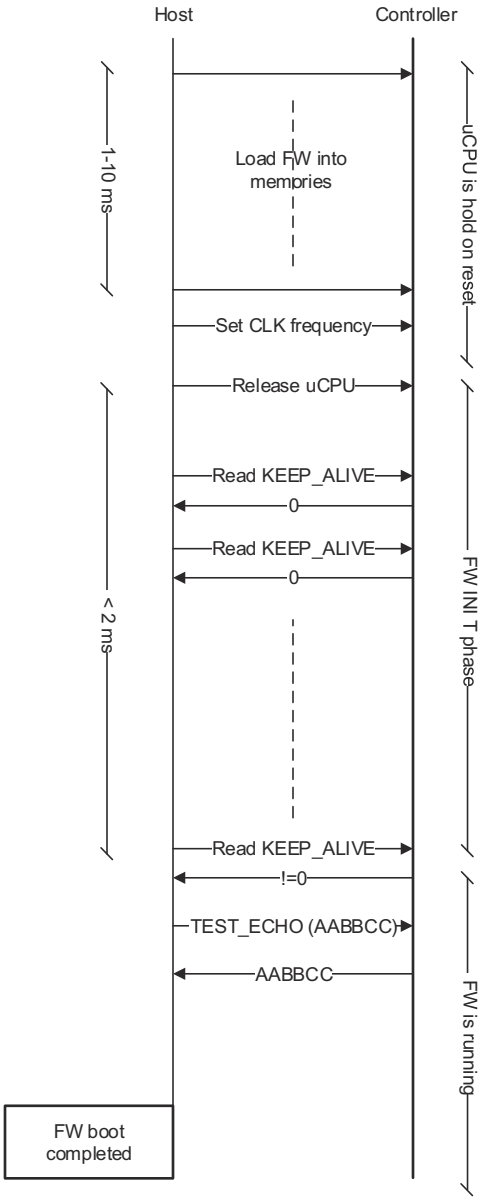


Figure 12-437. EDP MHDPTX Controller Boot Sequence

12.6.5.9.1.3 Setting Core Clock Frequency

The FW relies on a known core_clock frequency for configuring time HW dividers and timeout counters.

Therefore, core_clock frequency (SW_CLK_H, SW_CLK_I) must be configured by the host processor, prior to booting the FW.

12.6.5.9.1.4 Loading Firmware

The FW binaries are provided in the hexadecimal format that is composed from two files IRAM0.DATA and DRAM0.DATA.

The IRAM0.DATA contains the instruction memory and the DRAM0.DATA contains the variables initialization.

The external host processor should load the FW into the I-MEM and D-MEM, and enable the uCPU. Accessing the I-MEM, D-MEM is enabled only after reset (after reset, the system is going into debug mode, after loading FW and set EDP_CORE_APB_CTRL_P register to 0 system is going out from debug mode, and only reset can bring the system back to debug mode).

Copyright © 2024 Texas Instruments Incorporated

After loading the memories, the host processor should select the uCPU to be the master of the I-MEM and D-MEM memories, and enable the uCPU by writing 0x00 to the EDP_CORE_APB_CTRL_P register.

12.6.5.9.1.5 FW Running indication

After loading the FW, the embedded FW takes approximately 2ms to complete the INIT phase and start running. The EDP_CORE_KEEP_ALIVE_P register specifies whether the FW is running.

KEEP_ALIVE – The EDP_CORE_KEEP_ALIVE_P register is a counter initialized to 0 after reset and incremented by the FW on any scheduler loop.

When a valid FW is running, this register will be changed on every FW Scheduler loop (< 2ms). The host processor may use the mailbox channel immediately after reset, FW will response after the completion of INIT phase.

Operational mode

After completion of boot sequence, the FW set the MHDPTX Controller in a standby mode. In the standby mode, it is ensured that no transaction is executed over the external interfaces (that is, AUX, PHY, Video).

Switching into active mode is done by executing the GENERAL_MAIN_CONTROL command.

12.6.5.9.1.6 Software Events Handling

Software events are captured by the MHDPTX Controller FW, where the Event Value is stored in a designated area (in the D-MEM) and a bit is set by the FW in EDP_CORE_SW_EVENTS0_P register, see Table 12-387.

The host processor should poll the EDP_CORE_SW_EVENTS0_P register (cleared after read) and following a detection of event(s) the host processor should call the relevant command which returns the Event Value.

The Event Value holds the latest value associated with the event.

The EDP_CORE_SW_EVENTS0_P register is not cleared by the FW. Therefore, it is recommended to read this register (dummy read) to initialize the events to zero.

While any bit in EDP_CORE_SW_EVENTS0_P register is set, the MHDPTX Controller raises a hardware interrupt.

Table 12-387. EDP MHDPTX Controller Software Events

Bit	Event	Description
31:8	RESERVED	-
7	HDCP_TX_IS_RECEIVER_ID_VALID	IP has an ID to check if it is valid, HDCP_TX_IS_RECEIVER_ID_VALID_REQ needs to be called.
6	HDCP2_TX_STORE_KM	IP has Km to store, HDCP2_TX_STORE_KM_REQ needs to be called.
5	HDCP2_TX_IS_KM_STORED	IP need to check if Km is stored, HDCP2_TX_IS_KM_STORED_REQ needs to be called.
4	HDCP_TX_STATUS	HDCP TX was changed, HDCP_TX_STATUS_REQ needs to be called.
3:1	RESERVED	-
0	DPTX_HPD	HPD was changed, DPTX_READ_EVENT_REQUEST needs to be called.

12.6.5.9.1.7 DisplayPort Source (TX) Sequence

Figure 12-438 shows the expected host processor and uCPU tasks executed to start a video/audio transmission.

Copyright © 2024 Texas Instruments Incorporated

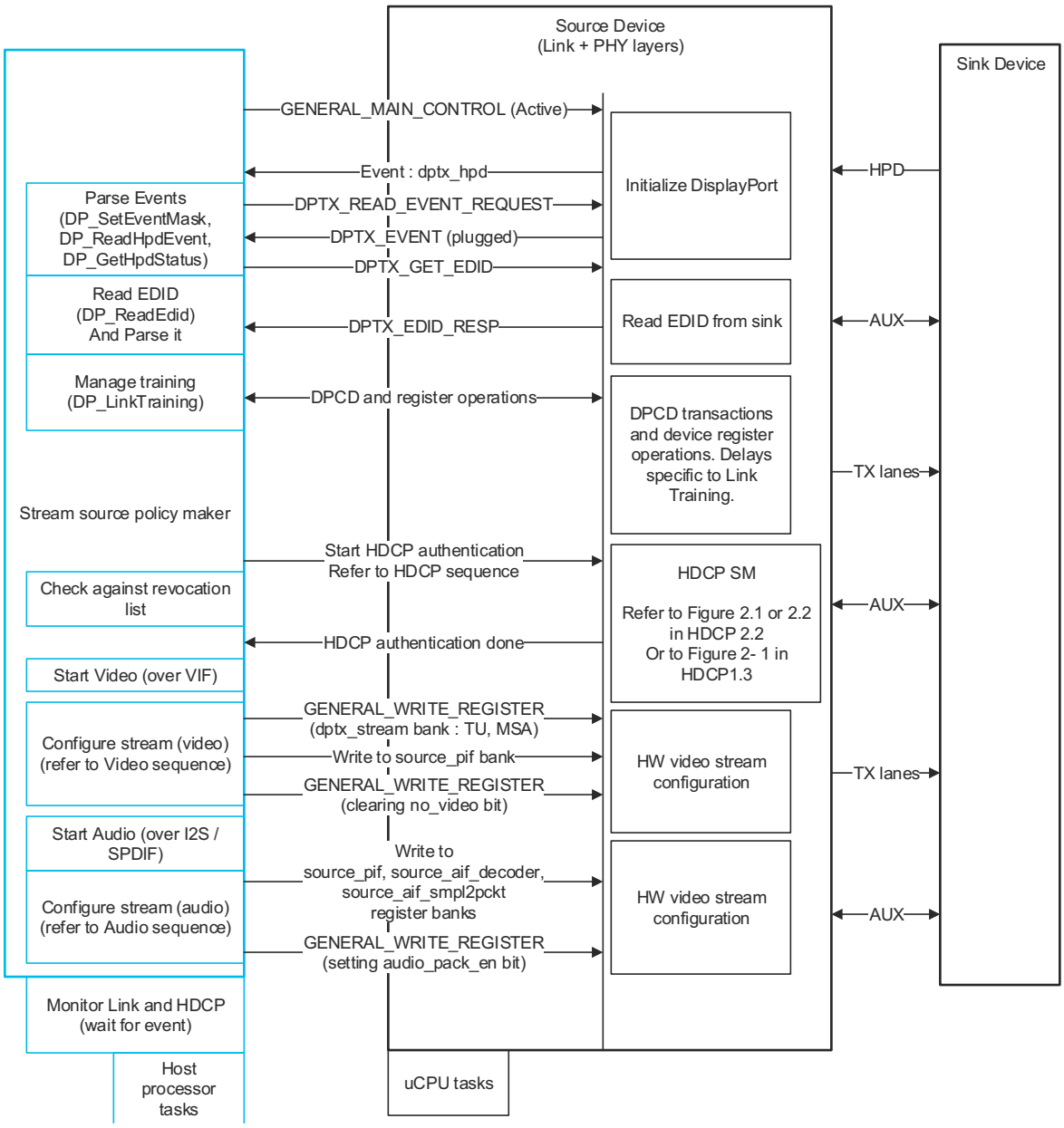


Figure 12-438. EDP Operation Sequence – DisplayPort

Figure 12-439 shows the expected host processor and uCPU tasks executed for updating a video format and Figure 12-440 - the audio format.

Copyright © 2024 Texas Instruments Incorporated

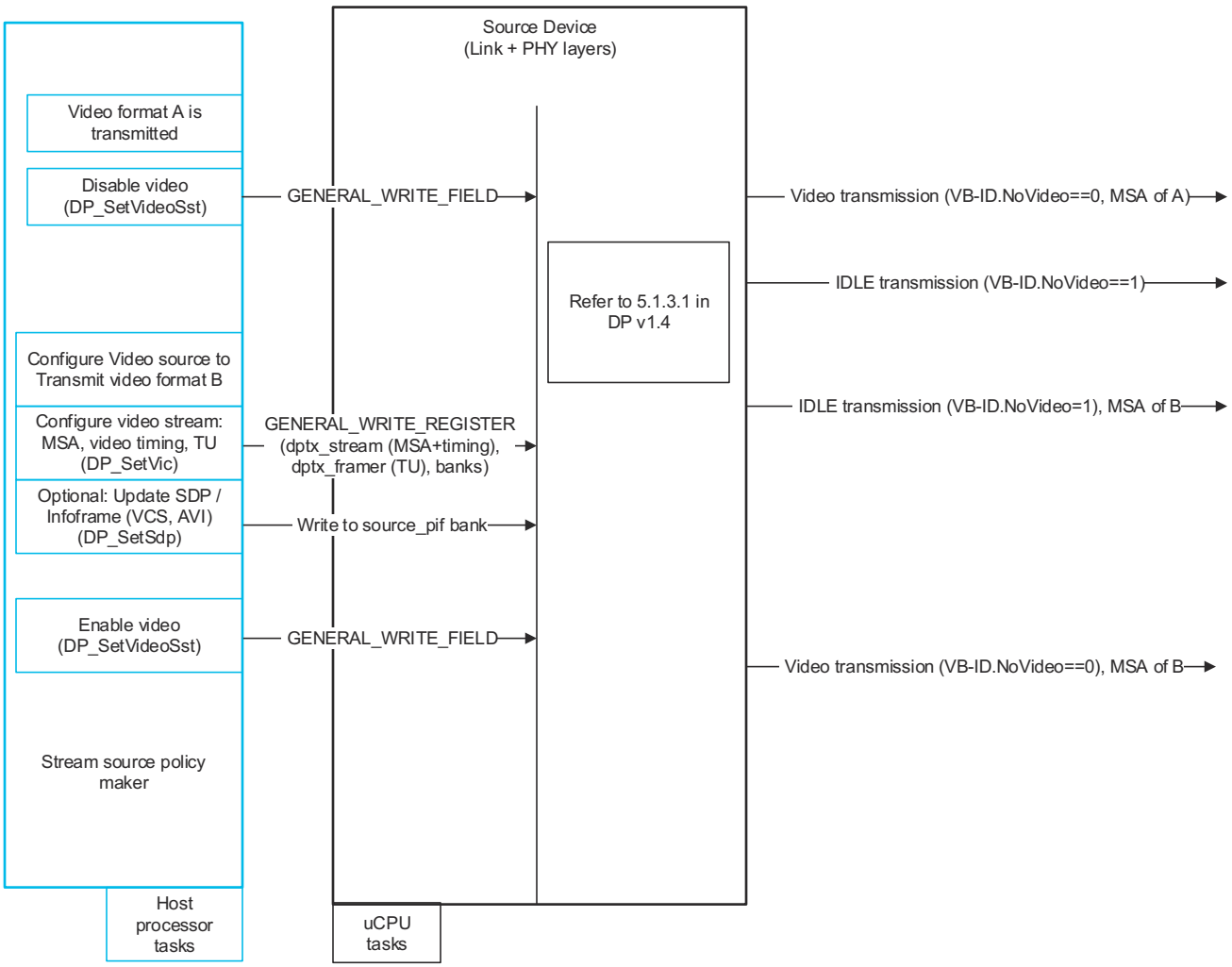


Figure 12-439. EDP Operation Sequence – DisplayPort Video Format Change

Copyright © 2024 Texas Instruments Incorporated

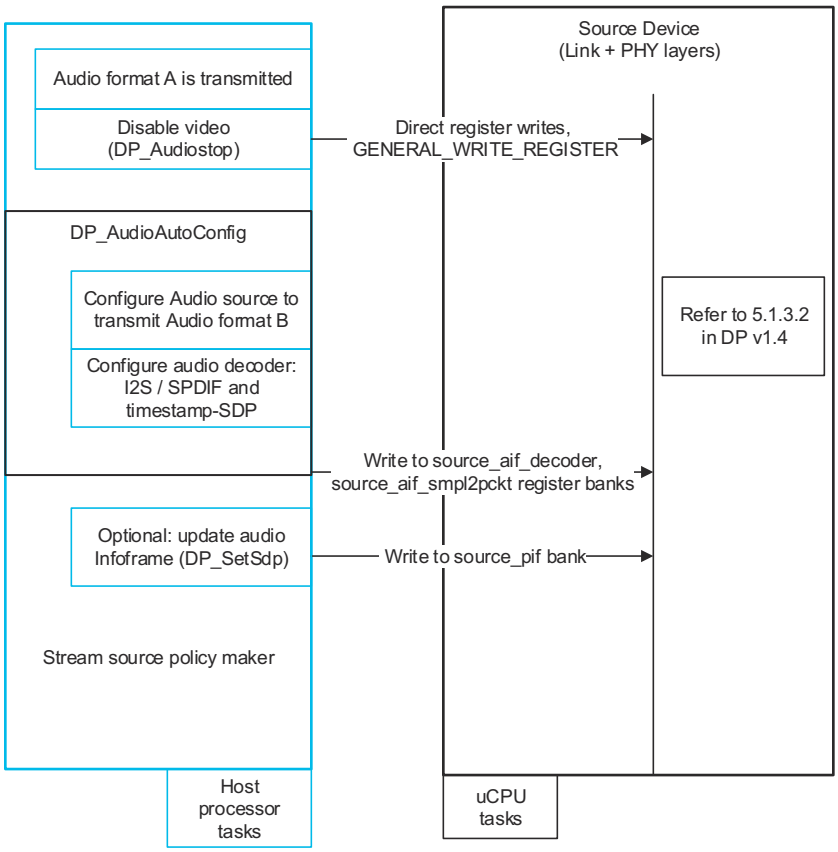


Figure 12-440. EDP Operation Sequence – DisplayPort Audio Format Change

12.6.5.9.1.8 HDCP

The FW supports embedded HDCP crypto, with optional Km-key encryption.

12.6.5.9.1.8.1 Embedded HDCP Crypto

In this mode, the HDCP functionality is fully supported by the MHDPTX Controller. The AKE phase and the corresponding cryptographic operations are handled by the FW.

Following the AKE phase, the FW initializes the stream cipher with the session key.

The host processor communicates with the MHDPTX Controller for:

- Configuring the capabilities
- Loading the HDCP keys (or certificate)
- Monitoring authentication status.

Note

Commands that carry secret information must be protected from tampering. Therefore, SAPB must be used to carry secured commands over mailbox.

Table 12-388. EDP HDCP Commands With Secret Information

HDCP command	Description
HDCP_GENERAL_2_SET_LC	Refer to HDCP 2.2 Appendix A
HDCP_GENERAL_SET_SEED	External TRNG must be used for generating a true random seed number. Therefore, it is expected that the seed number will meet the high entropy level.
HDCP2_TX_RESPOND_KM	Refer to HDCP 2.2 Appendix A
HDCP1_TX_SEND_KEYS	Refer to HDCP 1.4 Appendix B
HDCP2_TX_STORE_KM	Refer to HDCP 2.2 Appendix A

Copyright © 2024 Texas Instruments Incorporated

Figure 12-441 and Figure 12-442 show the operation sequences for an embedded HDCP 2.2 and HDCP 1.4

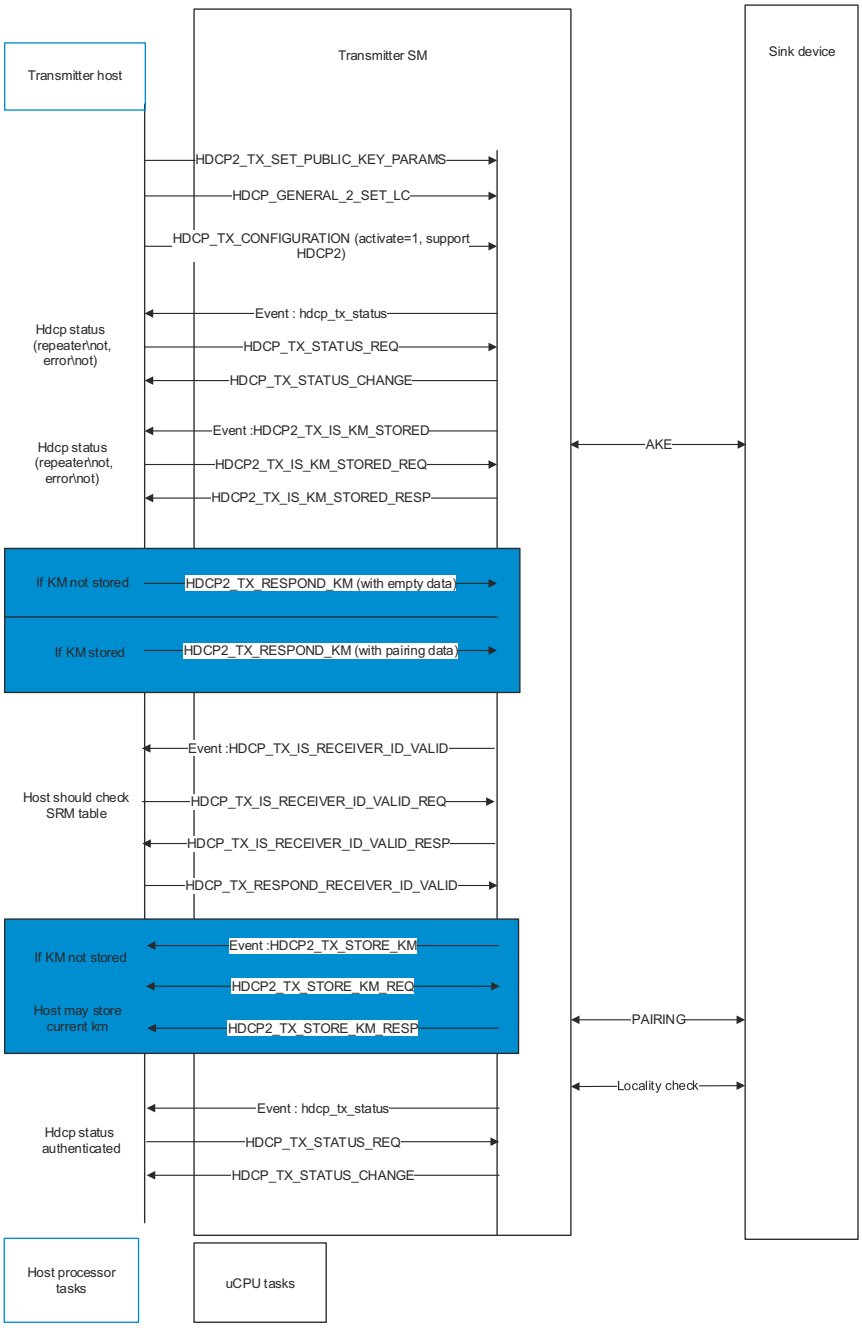


Figure 12-441. EDP Operation Sequence – Embedded HDCP 2.2

Copyright © 2024 Texas Instruments Incorporated

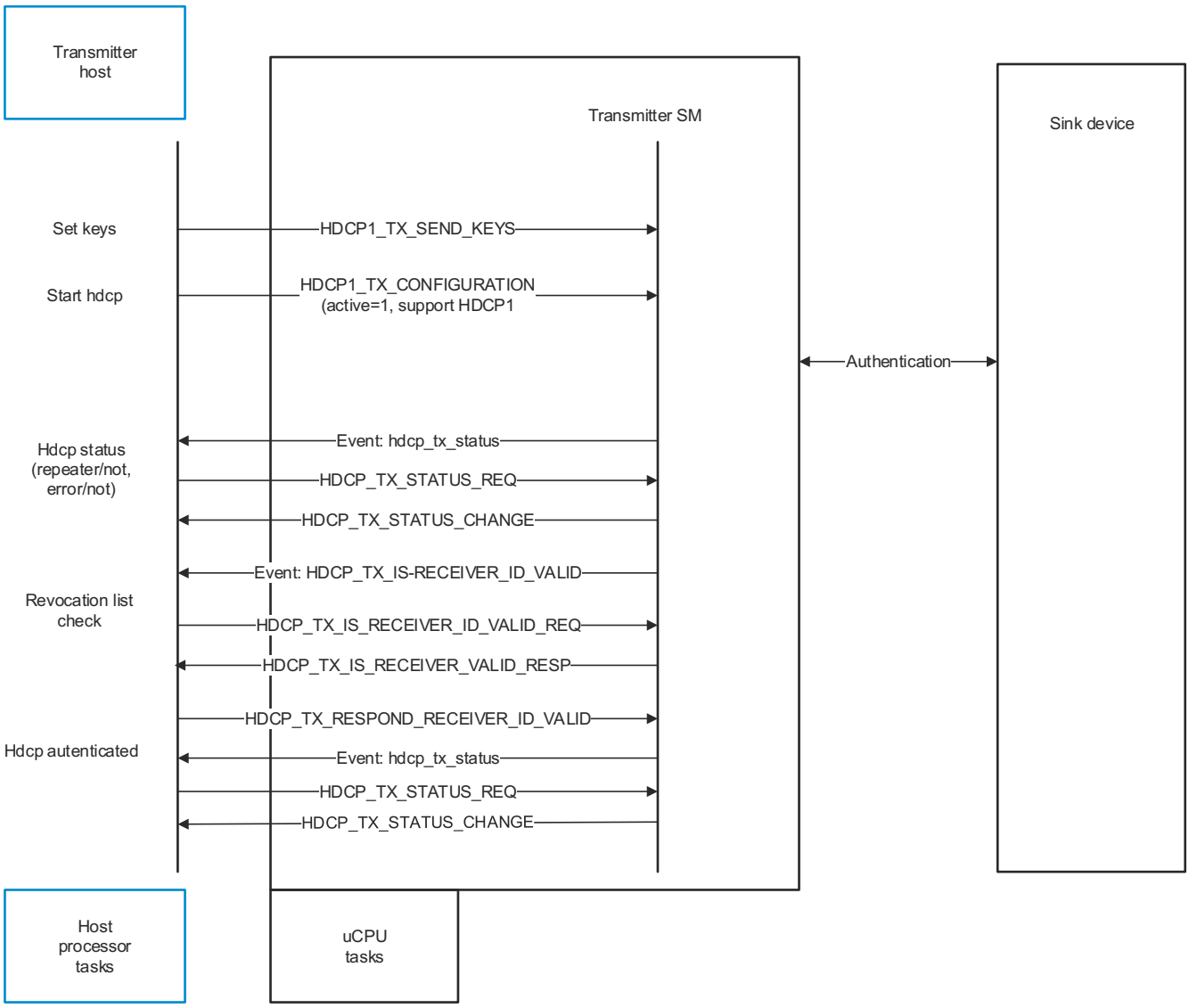


Figure 12-442. EDP Operation Sequence – Embedded HDCP 1.4

12.6.5.9.1.8.2 Additional Security Features

12.6.5.9.1.8.2.1 KM-Key Encryption

In order to provide higher security level an optional Km-key encryption is provided. If Km-key encryption is enabled while sending HDCP_TX_CONFIGURATION command, then HDCP controller will expect Master Key, 1.4 Device Keys and Global Constant. These keys will be decrypted using Km-key (loaded using HDCP2_TX_SET_KM_KEY_PARAMS command) by the controller's internal Firmware and used during Authentication and Key Exchange procedure. The Km-key encryption will also assure that when host reads Master Key it will receive it in encrypted form.

Enabling Km-key encryption affects the following commands or their parameters:

- HDCP2_TX_RESPOND_KM/Km
- HDCP1_TX_SEND_KEYS/Transmitter Device Key
- HDCP2_TX_STORE_KM_RESP/Km
- HDCP_GENERAL_2_SET_LC

Copyright © 2024 Texas Instruments Incorporated

12.6.5.9.1.8.2.2 Cyphertext Stealing

In order to use Km-key encryption for the 1.4 Device Keys and avoid changes in the API a Cyphertext Stealing technique is used. This technique allows to apply AES-ECB in Km-key encryption while 1.4 Device Key does not have size equal to multiple of 16.

12.6.5.9.1.9 HD Display TX Controller

12.6.5.9.1.9.1 Info-Frame Handling

Info-frames, which include meta-data (that is, auto type or video format), are initialized by the host processor.

The host processor directly initializes the PKT-MEM (which stores the info-frames). Initialization includes loading info-frame into PKT-MEM and updating the PKT LUT. Refer to DP_SetSdp, DP_RemoveSdp.

12.6.5.9.1.9.1.1 EDID Handling

The MHDPTX Controller issues command for reading EDID data from the remote side (sink). The EDID is sent to the host processor in a single block, that is, each block in one message. The MHDPTX Controller does not parse the EDID blocks. Refer to DP_ReadEdid.

12.6.5.9.1.9.1.2 Audio Control

When changing an audio stream format, the host processor must follow a specific programing sequence by configuring the I2S decoder and audio info frame packet. Refer to DP_AudioAutoConfig.

12.6.5.9.1.9.1.3 Video Control

The host processor need to configure the video timing before enabling the video stream over the VIF interface.

12.6.5.9.1.10 DPTX TX Controller

12.6.5.9.1.10.1 Protocol over Auxiliary

The firmware provides an interface for the AUX and I2C-over-AUX data transactions. The host processor may use it for direct DPCD (including the HDCP address range) or EDID access.

12.6.5.9.1.10.2 PHY (Physical layer) Handling

Separate PHY driver is used to initialize PHY.

DPTX PHY is handled by Core Driver, during link training (that is, Lane count, Link Rate, Pre-emphasis and Voltage Swing control).

12.6.5.9.2 EDP PHY Wrapper Initialization

EDP connects to the SERDES PHY wrapper (WIZ) using the RAW data mode. To enable this connection, the WIZ configuration registers in the SERDES PHY must be configured. For details on the SERDES PHY wrapper configuration, see Section *10-G SerDes Programming Guide*.

The following is an example configuration for enabling 4-data lane RAW data mode for Display Port connection.

// SERDES_TOP_CTRL register bit-field settings:

// PMA_CMN_REF_CLK_MODE = 2'b10, PMA_CMN_REF_CLK_INT_MODE = 2'b10, PMA_CMN_REFCLK_DIG_DIV = 0, PMA_SUSPEND_OVERRIDE = 0

SERDES_TOP_CTRL register value = 32'h30000000

// SERDES_RST register bit-field settings:

// PHY_RESET_N = 0 (PHY in reset)

// PHY_EN_REFCLK = 0 (PHY reference clock output disabled)

// PLL1/0_REFCLK_SEL = 0 (for cmn_refclk_<p/m> select), 1 (for pma_cmn_refclk_int select)

// REFCLK_TERM_DIS = 1 (termination disabled)

// REFCLK_DIG_SEL = 1 (PMA common reference clock select: 0 for cmn_refclk_<p/m>, 1 – pma_cmn_refclk_int)

Copyright © 2024 Texas Instruments Incorporated

// If the reference clock is sourced from internal digital input:

// PLL1_REFCLK_SEL = 1, PLL0_REFCLK_SEL = 1, REFCLK_TERM_DIS = 1, REFCLK_DIG_SEL = 1

SERDES_RST register value = 32'h39000000

// Alternatively, if the reference clock is sourced from <p/m> inputs:

// PLL1_REFCLK_SEL = 0, PLL0_REFCLK_SEL = 0, REFCLK_TERM_DIS = 0, REFCLK_DIG_SEL = 0

SERDES_RST alternative register value = 32'h00000000

// LANECTLx register bit-field settings:

// P0_ENABLE = 0, P0_FORCE_ENABLE = 1, P0_ALIGN = 1, P0_RAW_AUTO_START = 1, P0_STANDARD_MODE = 0, P0_FULLRT_DIV = 0

LANECTL0 register value = 32'h70000000 // for lane 0

// P0_ENABLE = 1, P0_FORCE_ENABLE = 0, P0_ALIGN = 0, P0_RAW_AUTO_START = 0, P0_STANDARD_MODE = 0, P0_FULLRT_DIV = 0

LANECTL1 register value = 32'h80000000 // for lane 1

LANECTL2 register value = 32'h80000000 // for lane 2

LANECTL3 register value = 32'h80000000 // for lane 3

// LANEDIVx register bit-field settings:

// P0_MAC_DIV_SEL0 = 7'b1, P0_MAC_DIV_SEL1 = 9'b1

LANEDIV0 register value = 32'h00010001 // for lane 0

LANEDIV1 register value = 32'h00010001 // for lane 1

LANEDIV2 register value = 32'h00010001 // for lane 2

LANEDIV3 register value = 32'h00010001 // for lane 3

// Set DIAG_REG register to 0

DIAG_TEST register value = 32'h0

Note

The SERDES PHY reset is not driven by the EDP controller (that is, by writing to the EDP_CORE_PHY_RESET_P[8] PHY_RESET register bit. Instead, the SERDES PHY reset is mapped to the SERDES_RST[31] PHY_RESET_N register bit in the SERDES PHY wrapper. But, the EDP_CORE_PHY_RESET[8] PHY_RESET bit is still used in the EDP controller to disable the controller's output to the SERDES. Therefore, after reset, the EDP_CORE_PHY_RESET[8] PHY_RESET bit must be set to 1 (reset off). After this, the SERDES reset can be controlled strictly with the SERDES_RST[31] PHY_RESET_N bit.

12.6.5.9.3 EDP PHY Programming

Table 12-389 shows information on how to program the EDP PHY (SERDES).

Table 12-389. EDP PHY (SERDES) Programming Details

General Information	
Standards / links / lanes supported	Display port and embedded display port / 2 links / 1, 2, or 4 lanes per link
Reference clock information and setup	
External / internal reference clock selection	Set up external or internal reference clock.
Reference clock frequency selection	Set up the reference clock programming for the selected reference clock frequency
PLL and high speed clocking information and setup	

Copyright © 2024 Texas Instruments Incorporated

Table 12-389. EDP PHY (SERDES) Programming Details (continued)

General Information	
PLL 0, Mode 0	Display port link 0. Note, if only link 1 is used, PLL 0 must be programmed in the same was as PLL 1.
PLL 0, Mode 1	Unused
PLL 1, Mode 0	Display port link 1. Note, if only link 0 is used, PLL 1 must be programmed in the same was as PLL 0.
PMA common full rate and data rate clocks	Unused
PMA transceiver full rate and data rate clocks	Full rate and data rate clocks are used as specified in the PMA spec table Clock rates for supported standards.
Top level pins	
PMA: cmn_pll0_mode_sel (Driven by PHY internal logic)	1'b0
PMA: cmn_pll1_mode_sel (Driven by PHY internal logic)	1'b0
PHY: phy_l{nn:00}_mode[1:0]	2'b11
PHY: phy_link_cfg_ln_{n:1}	1'b0 for slave lanes of each link 1'b1 for the master lane of each link
PHY configuration specific registers	
SERDES register PHY_AUTO_CFG_SPDUP, bit-fields [1] PHY_PLL_CFG_1 and [0] PHY_PLL_CFG_0	16'h0000 for single DP link configuration 16'h0002 for 2 DP link configuration
Set PLL analog clock divider values	
SERDES register CMN_PDIAG_PLL0_CLK_SEL_M0__CMN_PDIAG_PLL0_CTRL_M0	Link 0: Program it as described
SERDES register CMN_PDIAG_PLL1_CLK_SEL_M0__CMN_PDIAG_PLL1_CTRL_M0	Link 1: Program it as described
Select analog high speed clock, and transceiver clock divider values	
SERDES register XCVR_DIAG_HSCLK_DIV__XCVR_DIAG_HSCLK_SEL_j, bits [15:0] Link 0 Link 1	16'h0000 16'h0001
SERDES register XCVR_DIAG_HSCLK_DIV__XCVR_DIAG_HSCLK_SEL_j, bits [31:16]	Program it as described
Select digital PLL clock and data rate divider values	
SERDES register XCVR_DIAG_PLLDRC_CTRL__XCVR_DIAG_XDP_PWRI_STAT_j Link 0 Link 1	16'h0001 16'h0009
Protocol specific setup	
Display port and embedded display	Program it as described

Copyright © 2024 Texas Instruments Incorporated

12.7 Camera Subsystem

This section describes the Camera Subsystem in the device.

Copyright © 2024 Texas Instruments Incorporated

12.7.1 Camera Streaming Interface Receiver (CSI_RX_IF)

The following sections describe the camera streaming receiver interface (CSI_RX_IF) modules in the device.

12.7.1.1 CSI_RX_IF Overview

The integration of the CSI_RX_IF module allows the device to stream video inputs from multiple cameras to the image processing accelerator (VPAC) or to internal memory. The video input may also be retransmitted via the transmitter CSI (CSI_TX_IF) for debug and test purposes.

12.7.1.1.1 CSI_RX_IF Features

The CSI_RX_IF module supports the following features:

- Compliant to MIPI CSI v1.3
- Supports up to 16 virtual channels per input (partial MIPI CSI v2.0 feature)
- Data rate up to 2.5 Gbps per lane (wire rate)
- Supports 1, 2, 3, or 4 Data Lane connection to DPHY_RX
- Programmable formats including YUV420, YUV422, RGB, Raw, and User Defined (over 25 different formats supported)
- Four independent (simultaneous) output streams:
  - Two VP 32-bit streams to VISS inputs of VPAC image processing accelerator:
    - 2x 16-bit pixels per clock cycle
    - One virtual channel and data type per port
    - Raw format only (8-16 bits)
    - 32bit, 2 pixels wide, elastic buffer mode
      - Data[15:0]: Pixel n (MSB zero padding)
      - Data[31:0]: Pixel n+1 (MSB zero padding)
      - Internal full flag (FF) based FIFO (2048x32)
      - VP clock asynchronous to CSI_RX_IF main clock. Crossing done internally.
  - One (up to 4 channels) PPI 16-bit pixel retransmission interface to CSI_TX_IF:
    - 2x 16-bit pixels per clock cycle
    - 32bit retransmission width
    - No external buffer
    - Raw format only (8-20 bits, partial MIPI CSI v2.0 feature)
    - CSI_RX_IF and CSI_TX_IF main clocks must be running at the same frequency and synchronous.
  - One (up to 32 Channels) DMA interface through a 128-bit PSI_L connection to NAVSS for transfers to memory:
    - Byte packed (32x4) format, elastic buffer mode
    - Max rate 1 data cycle every 4 main clocks
    - ByteValid per byte in Last Data Phase (LDP)
    - 32 thread ID’s supported (virtual channel & data type combinations); Flexible number of threads (32 Max)
    - Virtual channels and data types mapped via mmr to PSI_L thread ID’s
    - Internal FF based FIFO; RAM based buffer (2kx128)
- Functional and data path error interrupts
- ECC support

Unsupported Features:

- See the Module Integration section of this document for a list of module features not supported by the integration on this Device.

Copyright © 2024 Texas Instruments Incorporated

12.7.1.1.2 CSI_RX_IF Ports

This section describes the CSI_RX_IF ports related to clocks, resets, and hardware requests.

Table 12-390. CSI_RX_IF Clocks and Resets

Clocks	
Module Clock Input	**Description**
CSI_RX_MAIN_CLK	Main functional clock.
CSI_RX_VBUS_CLK	The VBUS clock runs at always half the speed of the CSI_RX_MAIN_CLK.
CSI_RX_VP_CLK	Video port interface clock. It must run at the same speed or higher than the CSI_RX_MAIN_CLK. It can be async to the CSI_RX_MAIN_CLK clock. However, it must be sync to VPAC video clock.
CSI_RX_BYTE_CLK	The byte clock is the clock supplied by the DPHY_RX.
Resets	
Module Reset Input	**Description**
CSI_RX_RST	Asynchronous module global reset, driving all collater asynchronous resets of the 4 clock domains to the low state.

Table 12-391. CSI_RX_IF Hardware Requests

Interrupt Requests		
Module Interrupt Signal	**Description**	**Type**
CSI_RX_IF_CSI_ERR_IRQ_0	Stream error detected. The CSI_RX_IF will detect the error condition and capture the associated VC, DT and WC information for the packet header before generating an error interrupt flag on CSI_RX_CSI_ERR_IRQ.	Level
CSI_RX_IF_CSI_IRQ_0	Global interrupt that various resynchronized sources converge into interrupt generation.	Level
CSI_RX_IF_CSI_LEVEL_0	PSI_L fifo overflow or VP0/VP1 frame/line mismatch	Level
CSI_RX_IF_CORR_LEVEL_0	This interrupt is for checking the interface signals of the CSI_RX_IF controller for parity.	Level
CSI_RX_IF_CSI_FATAL_0	ASF port fatal interrupt. Level sensitive.	Level
CSI_RX_IF_CSI_NONFATAL_0	ASF port non-fatal interrupt. Level sensitive.	Level
CSI_RX_IF_UNCORR_LEVEL_0	This interrupt is for checking the interface signals of the CSI_RX_IF controller for parity.	Level

12.7.1.2 CSI_RX_IF Environment

The CSI_RX_IF has no dedicated pins. At the device level, video inputs come from the DPHY_RX; see Section 12.8.2.

Copyright © 2024 Texas Instruments Incorporated

12.7.1.3 CSI_RX_IF Functional Description

12.7.1.3.1 CSI_RX_IF Block Diagram

Figure 12-443 depicts a simplified internal block diagram of the CSI_RX_IF and its surroundings.

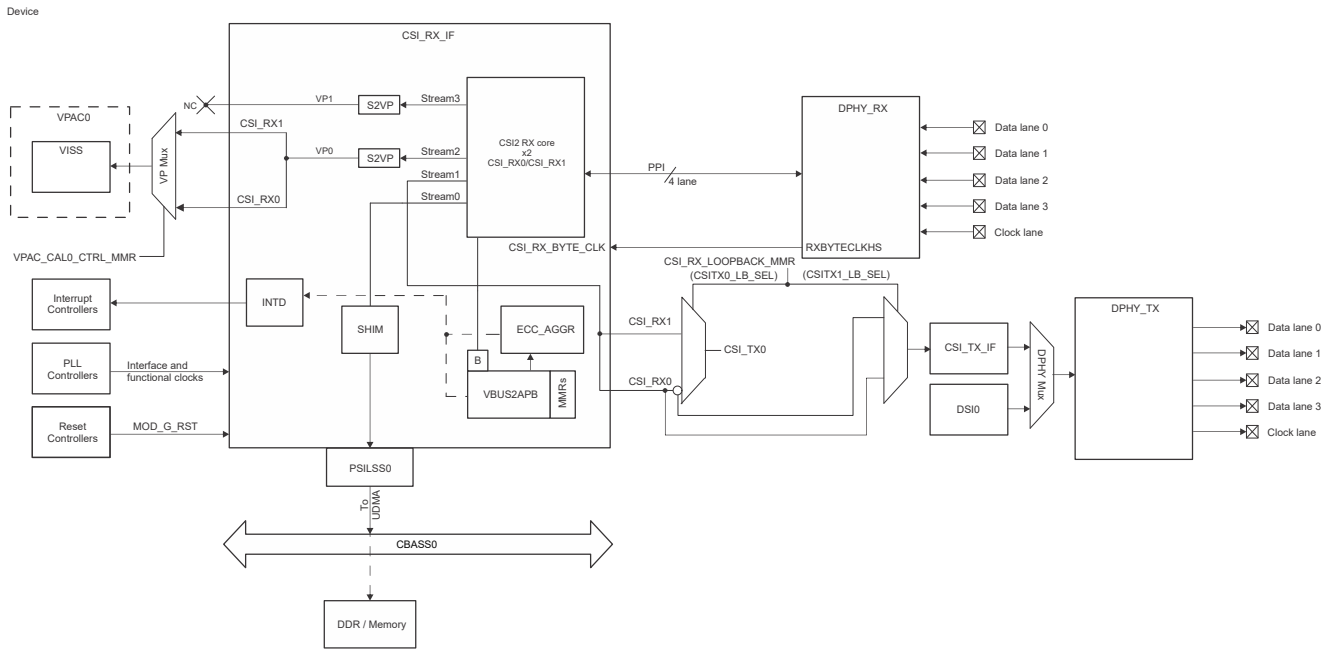


Figure 12-443. CSI_RX_IF Block Diagram

The CSI2 core streams 4 channels of data to the several export paths.

Stream0 (high BW DMA up to 32 channel contexts) streams 32-bits worth of data, that is re-formatted and sent as pixel format into 128 bits as PSIL data. There are 32 sets of MMRs/Registers that map virtual channels and data type to PSIL threads. Data is sent out PSIL in FIFO order as it is received. There is no priority. Data re-formatting is done to end up with correct data organization in memory so that other ip can use the data. The data type must be configured in MMRs to ensure proper reformatting. See Section 12.7.1.3.5. The large buffer of PSIL data (2048x128) has ECC protection, see Section 12.7.1.3.7

Stream1 (unlimited channel contexts, minimal buffering) streams an optional loopback output to CSI_TX_IF controller for diagnostic check.

12.7.1.3.2 CSI_RX_IF Hardware and Software Reset

An active low asynchronous hardware reset is provided to CSI_RX_IF by device LPSC. It is internally resynchronized to the functional clock domain.

A software reset is triggered by configuring the CSI_RX_IF_VBUS2APB_SOFT_RESET bit-fields for protocol reset and/or module reset.

12.7.1.3.3 CSI_RX_IF Clock Configuration

There are four clock domain in the CSI_RX_IF.

1. The CSI_RX_MAIN_CLK clock runs the CSI2 core, PSIL DMA, and retransmitts to CSI_TX_IF. The minimum clock rate for the CSI_RX_MAIN_CLK is 312.5MHz when DPHY is at max data rate (slower clock permissible when DPHY is at lower data rates). When CSI_RX_MAIN_CLK is operating lower than 312.5MHz, then the clock is essentially limiting the clock rate of the DPHY_RX. Said another way, CSI_RX_MAIN_CLK must be at least the CSI_RX_BYTE_CLK rate, else FIFOs will overflow.
   The maximum clock rate is 500MHz 16ffc, 650 Max
   f(CSI_RX_BYTE_CLK) × nb_lanes / 4; where f(CSI_RX_BYTE_CLK) is the minimum frequency of CSI_RX_BYTE_CLK.

Copyright © 2024 Texas Instruments Incorporated

2. The CSI_RX_VBUS_CLK is the interface configuration clock that runs at half the speed of the CSI_RX_MAIN_CLK.
3. The CSI_RX_VP_CLK is the video port interface clock. It must run at the same speed or higher than CSI_RX_MAIN_CLK. It can be async to CSI_RX_MAIN_CLK clock. It also must be sync to VPAC video clock. Up to 720MHz max rate
4. The CSI_RX_BYTE_CLK is the clock supplied by the DPHY_RX PLL and is divided down to byte clock. The DPHY_RX is designed for max of 10gbps. This translates to a max byte clock of 312.5MHz. The clock is inactive when DPHY_RX is not in HS operation.

Table 12-392 shows the CSI_RX_IF and DPHY_RX inter-clock dependencies.

Table 12-392. CSI_RX_IF Inter-clock Dependencies

	CSI_RX_MAIN_CLK	CSI_RX_VBUS_CLK	CSI_RX_VP_CLK	CSI_RX_BYTE_CLK
Min freq	CSI_RX_BYTE_CLK freq	CSI_RX_MAIN_CLK / 2 freq	CSI_RX_BYTE_CLK freq	N/A
Max freq	500MHz	CSI_RX_MAIN_CLK / 2 freq	720MHz	312.5MHz

12.7.1.3.4 CSI_RX_IF Interrupt Events

This section describes the register configuration of the interrupt events that can trigger the several CSI_RX_IF interrupt signals. For detailed description and mapping of the interrupt ot the device interrupt processors see *CSI_RX_IF Integration*.

The interrupts are generally handled wihtin the INTD module of the CSI_RX_IF, although there are several interrupt registers in the ECC_AGGR for ECC errors and in the VBUS2APB for stream monitoring errors/flags.

Table 12-393 lists the event generation and corresponding registers of the CSI_RX_IF controller.

Table 12-393. CSI_RX_IF Interrupt Events Cross Table

Event	Mask Regsiter	Status Register	Description
CSI_RX_CSI_ERR_IRQ	CSI_RX_IF_VBUS2APB_ERROR_IRQS_MASK_CFG	CSI_RX_IF_VBUS2APB_ERROR_IRQS	Stream error detected. The CSI_RX_IF0 will detect the error condition and capture the associated VC, DT and WC information for the packet header before generating an error interrupt flag on CSI_RX_CSI_ERR_IRQ. Read the status regsiter bitfields to trace the source of the event.
CSI_RX_CSI_IRQ	CSI_RX_IF_CP_INTD_ENABLE_REG_LEVEL_0 CSI_RX_IF_VBUS2APB_ERROR_IRQS_MASK_CFG	CSI_RX_IF_CP_INTD_STATUS_REG_LEVEL_0 CSI_RX_IF_VBUS2APB_ERROR_IRQS	Global functional interrupt that various resynchronized sources converge into interrupt generation.
CSI_RX_CSI_LEVEL	CSI_RX_IF_VBUS2APB_MONITOR_IRQS_MASK_CFG	CSI_RX_IF_VBUS2APB_STREAM0_FIFO_FILL_LVL CSI_RX_IF_VBUS2APB_STREAM1_FIFO_FILL_LVL CSI_RX_IF_VBUS2APB_STREAM2_FIFO_FILL_LVL CSI_RX_IF_VBUS2APB_STREAM3_FIFO_FILL_LVL CSI_RX_IF_VBUS2APB_STREAM0_FIFO_FILL_LVL	PSI_L fifo overflow or VP0/VP1 frame/ line mismatch (at the minimum an error interrupt will occur at the end of frame but may be issued within the frame as well). Read the status regsiter bitfields to trace the source of the event.
CSI_RX_CORR_LEVEL	CSI_RX_IF_VBUS2APB_ASF_INT_MASK	CSI_RX_IF_VBUS2APB_ASF_INT_STATUS	This interrupt is for checking the interface signals of the CSI_RX_IF0 controller for parity.
CSI_RX_UNCORR_LEVEL	CSI_RX_IF_VBUS2APB_ASF_INT_MASK	CSI_RX_IF_VBUS2APB_ASF_INT_STATUS	This interrupt is for checking the interface signals of the CSI_RX_IF0 controller for parity.

Copyright © 2024 Texas Instruments Incorporated

Table 12-393. CSI_RX_IF Interrupt Events Cross Table (continued)

Event	Mask Regsiter	Status Register	Description
CSI_RX_CSI_FATAL	CSI_RX_IF_VBUS2APB_ASF_INT_MASK	CSI_RX_IF_VBUS2APB_ASF_INT_STATUS	ASF port fatal interrupt. Level sensitive. Set CSI_RX_IF_VBUS2APB_ASF_FATAL_NONFATAL_SELECT for whether fatal or non-fatal ASF interrupt is triggered. If any of the CSI_RX_IF_VBUS2APB_ASF_INT_STATUS bit is set, the CSI_RX_CSI_FATAL or CSI_RX_CSI_NONFATAL event signal is asserted.
CSI_RX_CSI_NONFATAL	CSI_RX_IF_VBUS2APB_ASF_INT_MASK	CSI_RX_IF_VBUS2APB_ASF_INT_STATUS	ASF port non-fatal interrupt. Level sensitive. Set CSI_RX_IF_VBUS2APB_ASF_FATAL_NONFATAL_SELECT to whether fatal or non-fatal ASF interrupt is triggered. If any of the CSI_RX_IF_VBUS2APB_ASF_INT_STATUS bit is set, the CSI_RX_CSI_FATAL or CSI_RX_CSI_NONFATAL event signal is asserted.

12.7.1.3.5 CSI_RX_IF Data Memory Organization Details

Figure 12-444 shows the YUV422-8 data organization in memory.

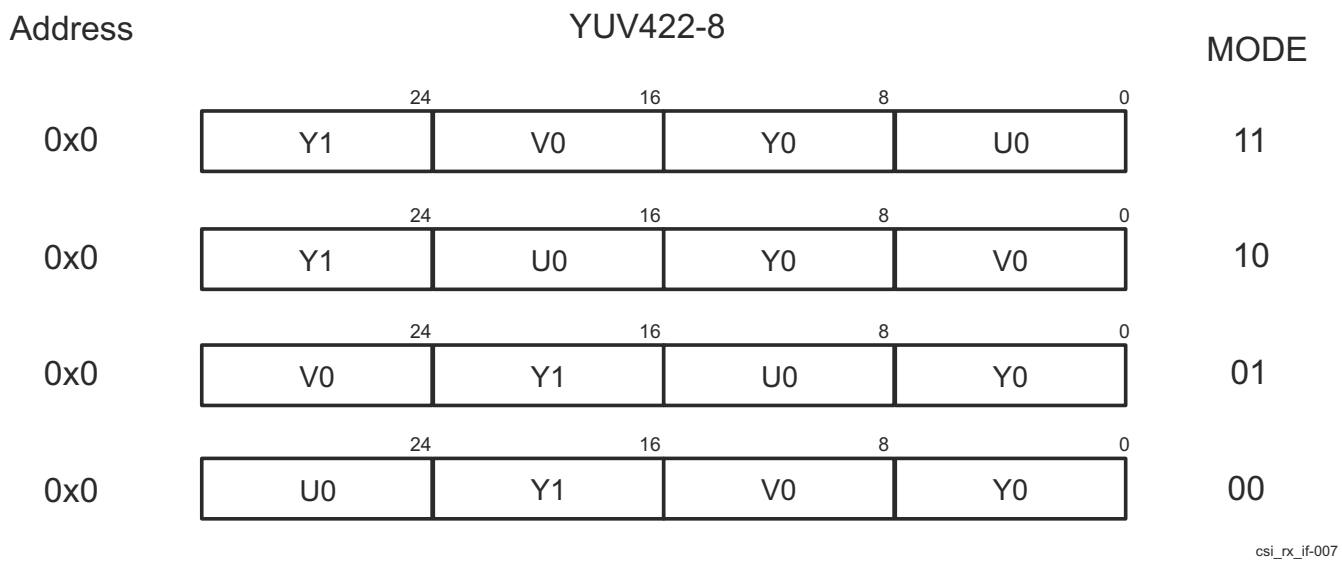


Figure 12-444. CSI_RX_IF YUV422-8 Memory Data Organization

Figure 12-445 shows the YUV422-10 data organization in memory.

Copyright © 2024 Texas Instruments Incorporated

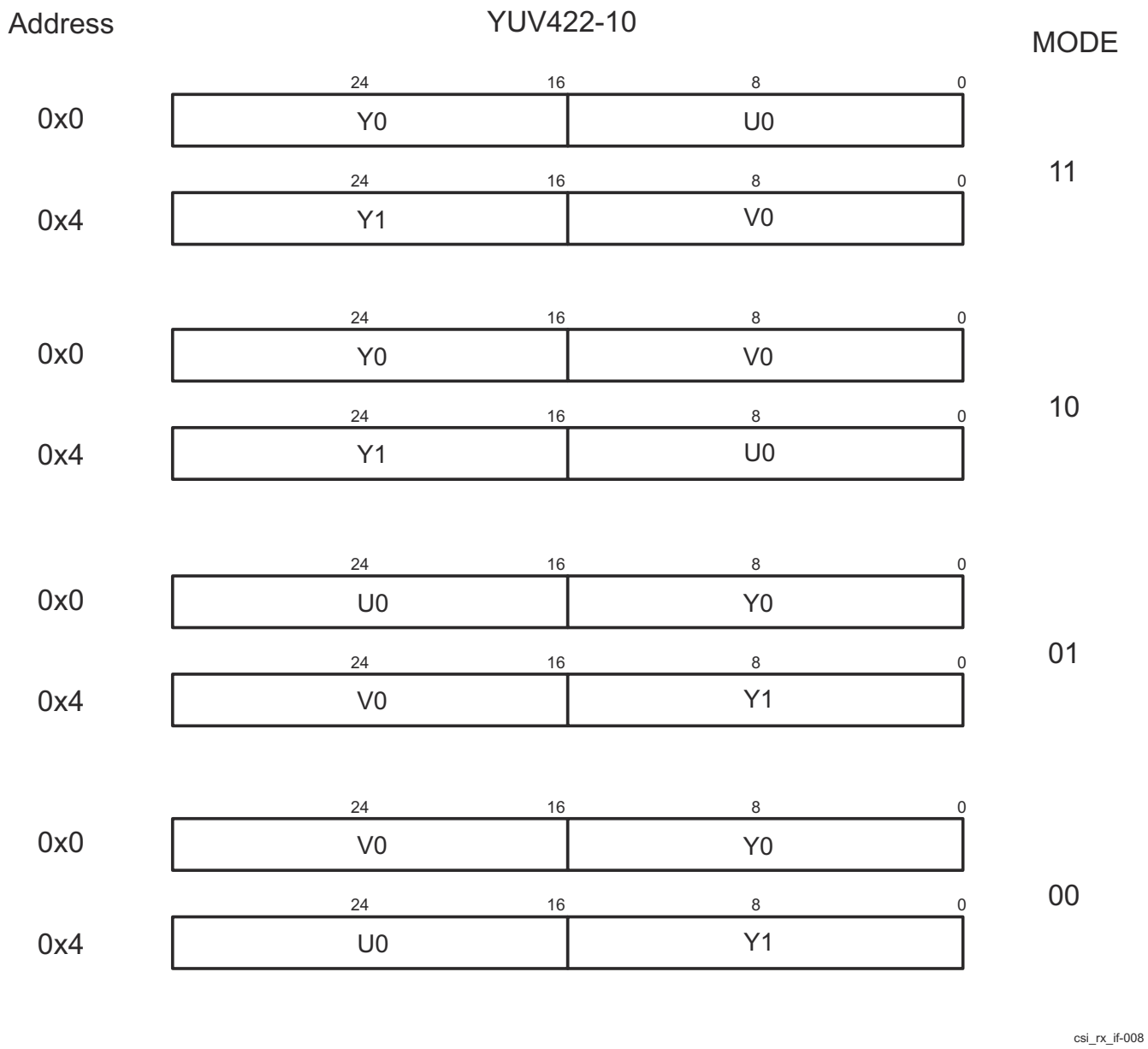


Figure 12-445. CSI_RX_IF YUV422-10 memory data organization

Figure 12-446 shows the YUV420-8 data organization in memory.

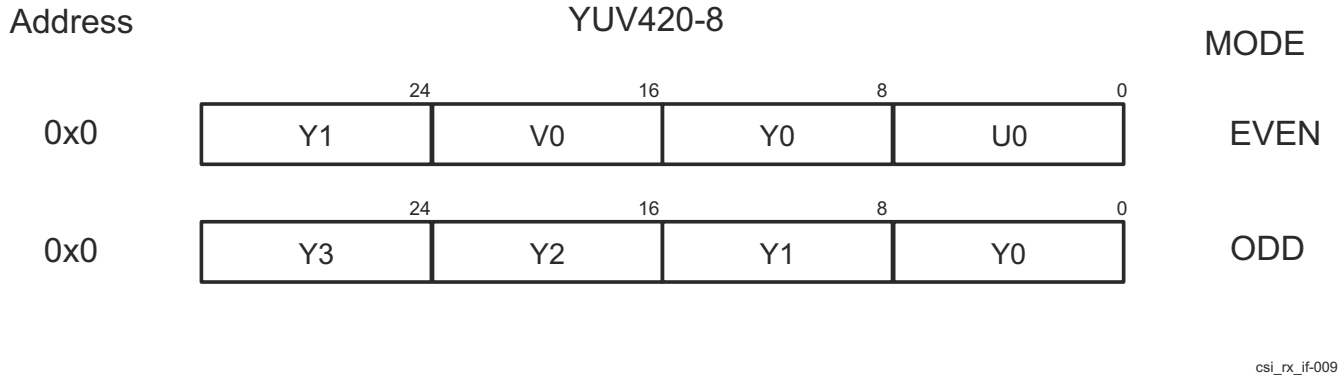


Figure 12-446. CSI_RX_IF YUV420-8 memory data organization

Copyright © 2024 Texas Instruments Incorporated

Figure 12-447 shows the YUV420-10 data organization in memory.

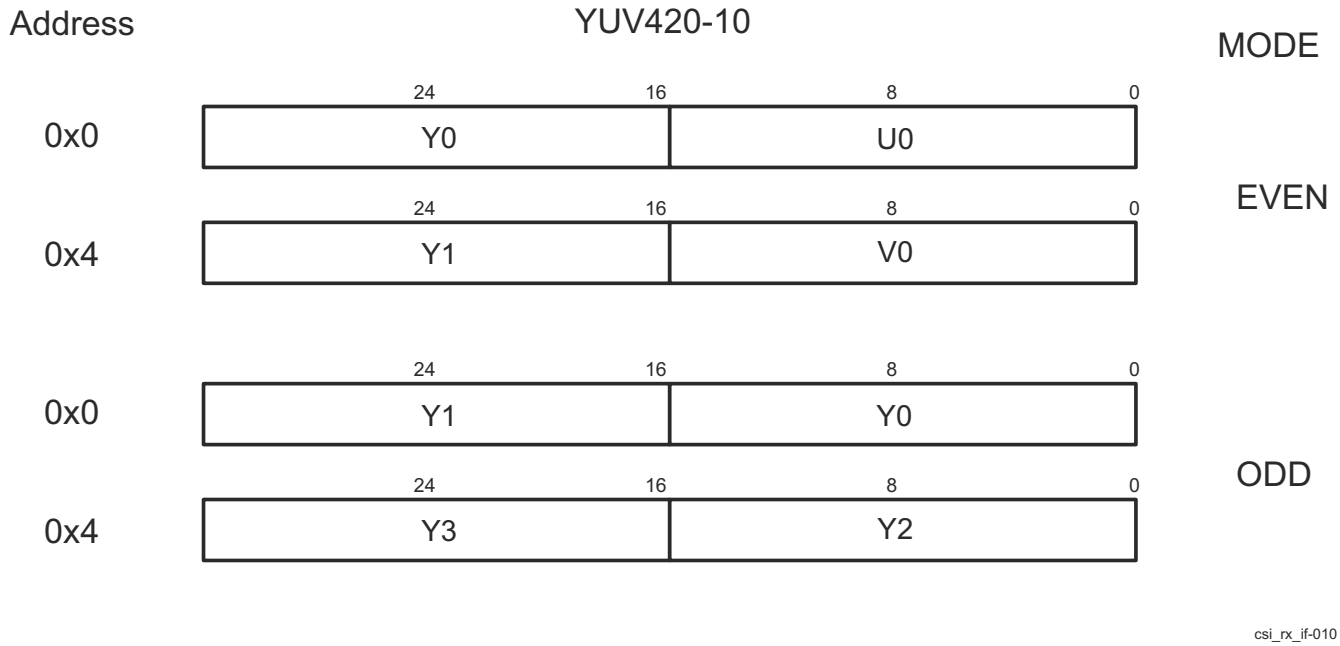


Figure 12-447. CSI_RX_IF YUV420-10 memory data organization

Figure 12-448 shows the RAW data organization in memory.

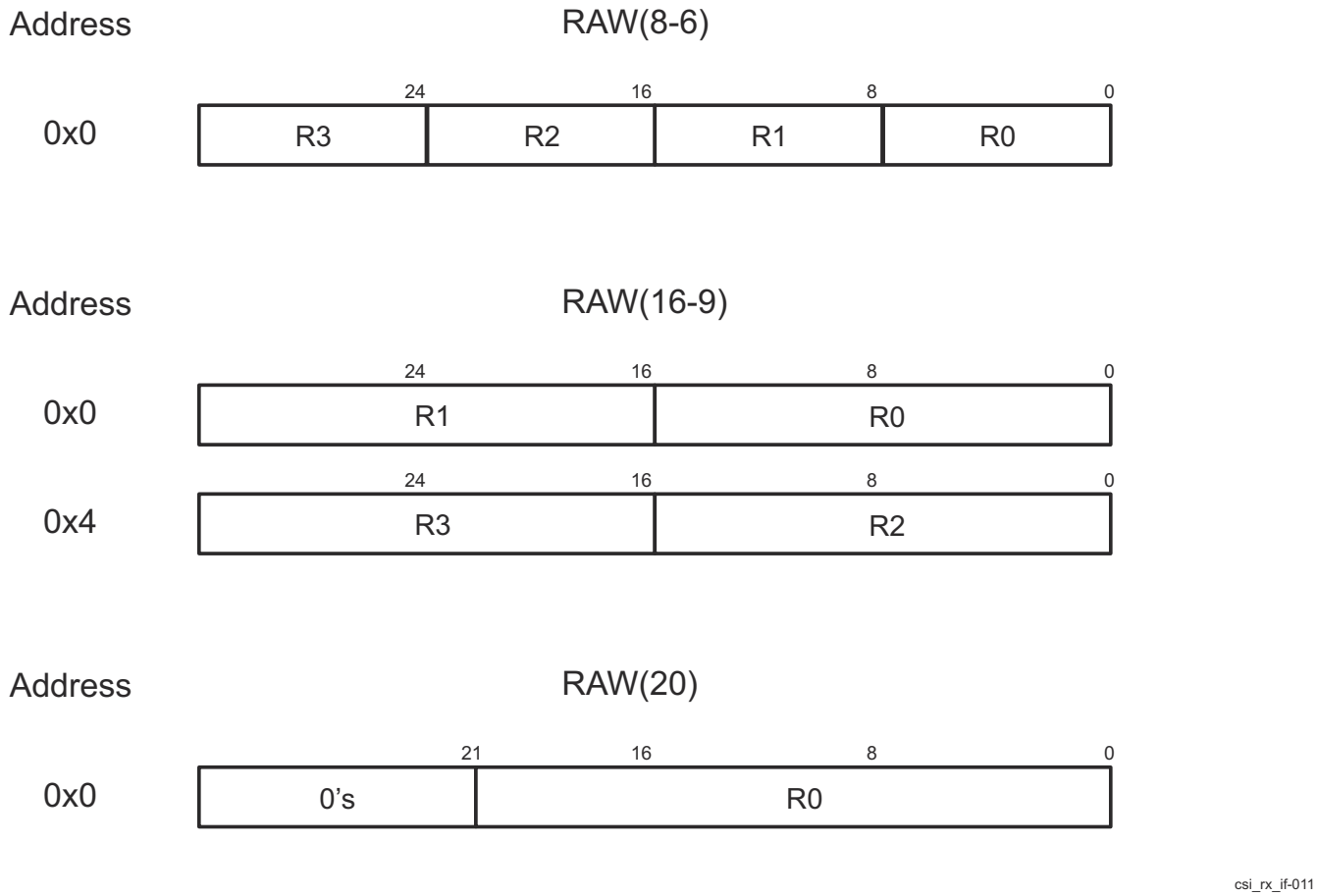


Figure 12-448. CSI_RX_IF RAW (UNPACKED) memory data organization

Copyright © 2024 Texas Instruments Incorporated

Figure 12-449 shows the RGB data organization in memory.

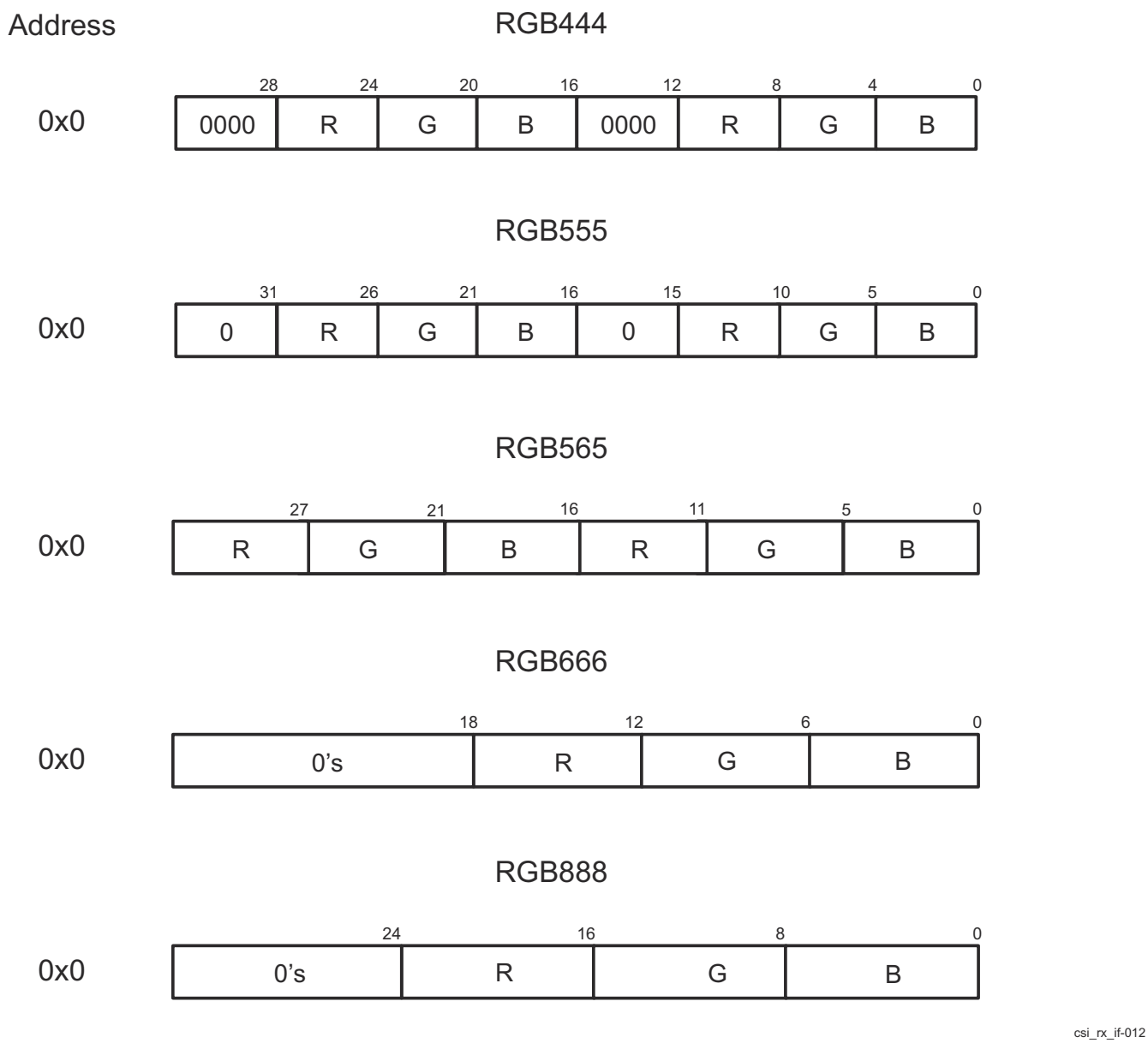
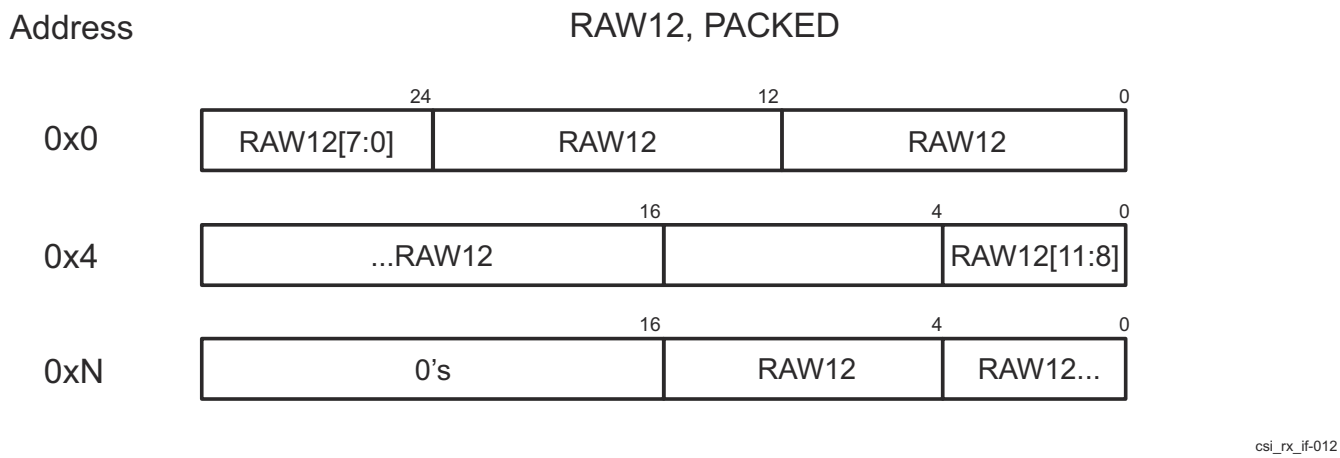


Figure 12-449. CSI_RX_IF RGB memory data organization

Figure 12-450 shows the RAW12 (PACKED) data organization in memory.

Copyright © 2024 Texas Instruments Incorporated



A. Where data does not fill out to a byte boundary it is zero extended.

Figure 12-450. CSI_RX_IF RAW12 (PACKED) memory data organization

12.7.1.3.6 CSI_RX_IF PSI_L (DMA) Interface

Note

This section describes CSI_RX_IF PSI_L specific functionality related to the camera interface. The general PSI_L functional description and registers are described in the PSI_L specific chapter (see *PSIL Subsystem (PSILSS)*).

12.7.1.3.6.1 PSI_L DMA framing

The CSI_RX_IF stream interface provides the usual VSYNC/HSYNC signals. The signals are per virtual channel where the transition from 1-to-0 or 0-to-1 represents SOF/EOF or SOL/EOL. VSYNC/HSYNC transition is usually not coincident with a valid data phase, though there are use cases where it can be. Because of the nature of CSI protocol, the VSYNC and HSYNC transactions require at least a single clock cycle and can only transition sequentially (important fact in handling of VSYNC events).

There is a rarely used mode of CSI where an entire frame is passed in a single packets without line breaks. In this mode, there are no HSYNC transitions at line starts/ends.

Lastly, there is redundant information provided with each CSI packet where the packet length is supplied in bytes.

To overcome these framing restrictions, the CSI_RX_IF PSILSS0 passes metadata through the DMA FIFO. Anytime metadata is inserted into the FIFO, the CSI_RX_IF stream is stalled for a clock cycle. The forms of supported meta tags are:

- SOF for a given virtChan
- EOF for a given virtChan
- Packet length in bytes for a given dmaCntx

The PSILSS0 will disregard the HSYNC signal entirely. The long packet beginning is used as SOL and the EOL will be "counted" using the packet length provided. In the special case where packets are of FRAME length, the SOL/EOL handling is identical.

The PSILSS0 solution for VSYNC handling is to create the concept of metadata FIFO entry which is indicated by a metadata bit. The SOF or EOF VSYNC information plus the virtualChan index is fed into the FIFO. On the output of the FIFO, the SOF is saved for each context of that virtualChan type. The very next data phase of that channel context will be marked as SOF. Additionally a state bit per context is maintained indicating the channel is MOP (middle of packet). Once the EOF arrives at the FIFO output, all contexts of that type virtual channel (and currently in MOP state) will receive a PSI_EOP with zero active bytes of data.

Copyright © 2024 Texas Instruments Incorporated

CAUTION

The line and frame size is not known outside the CSI core (i.e. not passed to the DMA interface). Therefore any mismatches at system level programming will be unknown. If the DMA line/frame size does not match the CSI_RX_IF line/frame size, it is assumed the DMA will not result in any adverse side affects as a result of not enough data or too much data.

12.7.1.3.6.2 PSI_L DMA error handling due to FIFO overflow

The DMA error handling is also called a PSI_L protocol enforcer. It is intended to prevent hang of the PSILSS0. Unnatural packet size should not cause hang so only SOP/SOL/EOL/EOP framing is enforced. The following list highlights the error handling mechanism:

- For context cleanup the protocol enforcer:
  - cycle through each context index checking if context is in MOL or MOP
  - close out MOP with EOP and MOL&MOP with EOL&EOP
- dropOnFloor SOP if currently in MOPstate
- dropOnFloor EOP if not currently in MOPstate
- dropOnFloor FIFO data
- EOL context if EOP and MOLstate
- After closing out all open contexts the PSILSS0 logic will then wait till end of frame per virtual channel. Once a new frame starts it will then start sending out data from that new frame

12.7.1.3.7 CSI_RX_IF ECC Protection Support

ECC is a mechanism for providing increased system reliability (via reduction of memory soft errors) by allowing single bit errors to be detected and corrected and double bit errors to be detected.

The ECC protection on the CSI_RX_IF RAM provides Single Error Correction and Double Error Detection (SEC/DED). This logic detects and corrects a single bit error (1 bit error per ECC word or per ECC data segment). For memories that contain critical and/or persistent data, automatic (immediate or delayed) write-back of the corrected data to the corresponding memory address is supported. In addition, the ECC also supports multiple options for partial word writes, such as read-modify-write or multiple ECC code segments per word.

The ECC protection also provides Double Error Detection (DED). This logic only detects (does not correct) double errors (2 bit errors per ECC word or per ECC data segment).

The ECC aggregator in the CSI_RX_IF subsystem level consolidates the ECC configuration and status bits for all the ECC supported memories in the subsystem. It provides a single EOI-handshake based interrupt to the interrupt processors (for both single and double error detections) and a standard 32-bit VBUSP interface for configuring and querying the ECC register set, see *CSI__RX_IF_ECC Registers*. For complete details on the features and functions of the ECC aggregator, see chapter ,*ECC Aggregator*.

12.7.1.3.8 CSI_RX_IF Programming Guide

12.7.1.3.8.1 Overview

This section details specific steps on how to program the CSI_RX_IF controller.

12.7.1.3.8.2 Controller Configuration

The CSI_RX_IF streams will default to accept all virtual channels and all data types after reset, so the user must program the stream configuration and pixel interface to match the system use case.

Note

The CSI_RX_IF controller can be configured so that the reset values can adopt the users Power_On state. This can reduce the programming steps required by the system.

The CSI_RX_IF controller is designed to operate all the defined streams with all virtual channels and all data types passed to the pixel interface. Also, the connection to the front interface adopts reset values that will allow the connection of the lanes to expect no remapping.

Copyright © 2024 Texas Instruments Incorporated

The basic system configuration steps are then programming the number of enabled data lanes and starting the streams.

The system can also decrease the virtual channel and data type processed by the stream by configuring the data config (CSI_RX_IF_VBUS2APB_STREAM0_DATA_CFG - CSI_RX_IF_VBUS2APB_STREAM3_DATA_CFG) registers.

The user must perform a read from the stream config register (CSI_RX_IF_VBUS2APB_STREAM0_CFG - CSI_RX_IF_VBUS2APB_STREAM3_CFG) at power up to identify the number of streams and pixel interface mode. The stream FIFO depth must be determined from the system configuration information defined at build time to ensure that any programmed [31-16] FIFO_FILL level is valid for the available FIFO depth.

12.7.1.3.8.3 Power on Configuration

The CSI_RX_IF requires the system to perform the configuration of the DPHY_RX interface before starting the streams of the controller. The default configuration of the controller and each stream can be identified by reading the device ID CSI_RX_IF_VBUS2APB_DEVICE_CONFIG register.

The FW must read the CSI_RX_IF_VBUS2APB_DEVICE_CONFIG register at power up. This will provide the FW the number of streams that will need to be configured, and all the default system information for the FIFO structures in the available streams.

Table 12-394. CSI_RX_IF_VBUS2APB_DEVICE_CONFIG bitfield details

STREAMx_NUM_PIXELS	The width of the pixel interface and the bits per pixel for the selected datatype will determine how many pixels can be output in a single cycle. Default will be 1 pixel per clock. 00 -> 1 pixel per clock 01 -> 2 pixels per clock 10 -> 4 pixels per clock
DATAPATH_SIZE	Internal Datapath width 00 - 32 bit, all other values are reserved.
NUM_STREAMS	Number of Stream interfaces (1-4) = (value+1)
MAX_LANE_NB	Max Number of Lanes (1-4) = (value+1)

Figure 12-451 shows the minimal sequence of registers that will configure the DPHY_RX and then start the stream; this will output all the pixel information for all virtual channels and all data types detected in the link data stream.

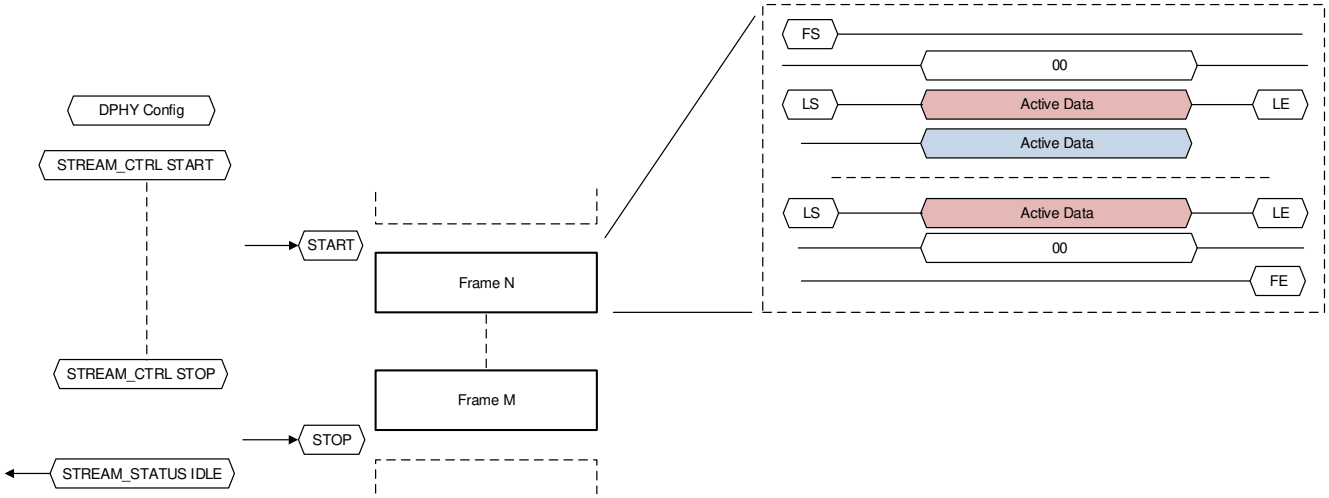


Figure 12-451. Minimal Stream Control - Start and Stop

12.7.1.3.8.4 Stream Start and Stop

The CSI_RX_IF will perform management of the stop and halt control to the pixel stream during functional operation to allow any stream to change the virtual channel or data type information that the stream will process.

Copyright © 2024 Texas Instruments Incorporated

- The FW will use the stream control register (CSI_RX_IF_VBUS2APB_STREAM0_CTRL - CSI_RX_IF_VBUS2APB_STREAM3_CTRL) to perform a stop (set bit 1) and wait for the end of any current frame, and wait for the stream to return its state to IDLE, which can be read from stream status register register (CSI_RX_IF_VBUS2APB_STREAM0_STATUS - CSI_RX_IF_VBUS2APB_STREAM3_STATUS)[31] RUNNING = 1.
- The FW will use the stream control register to perform an abort and wait for the end of any current line, and wait for the stream to return its state to IDLE, which can be read from register STREAM_STATUS[31] RUNNING = 1.

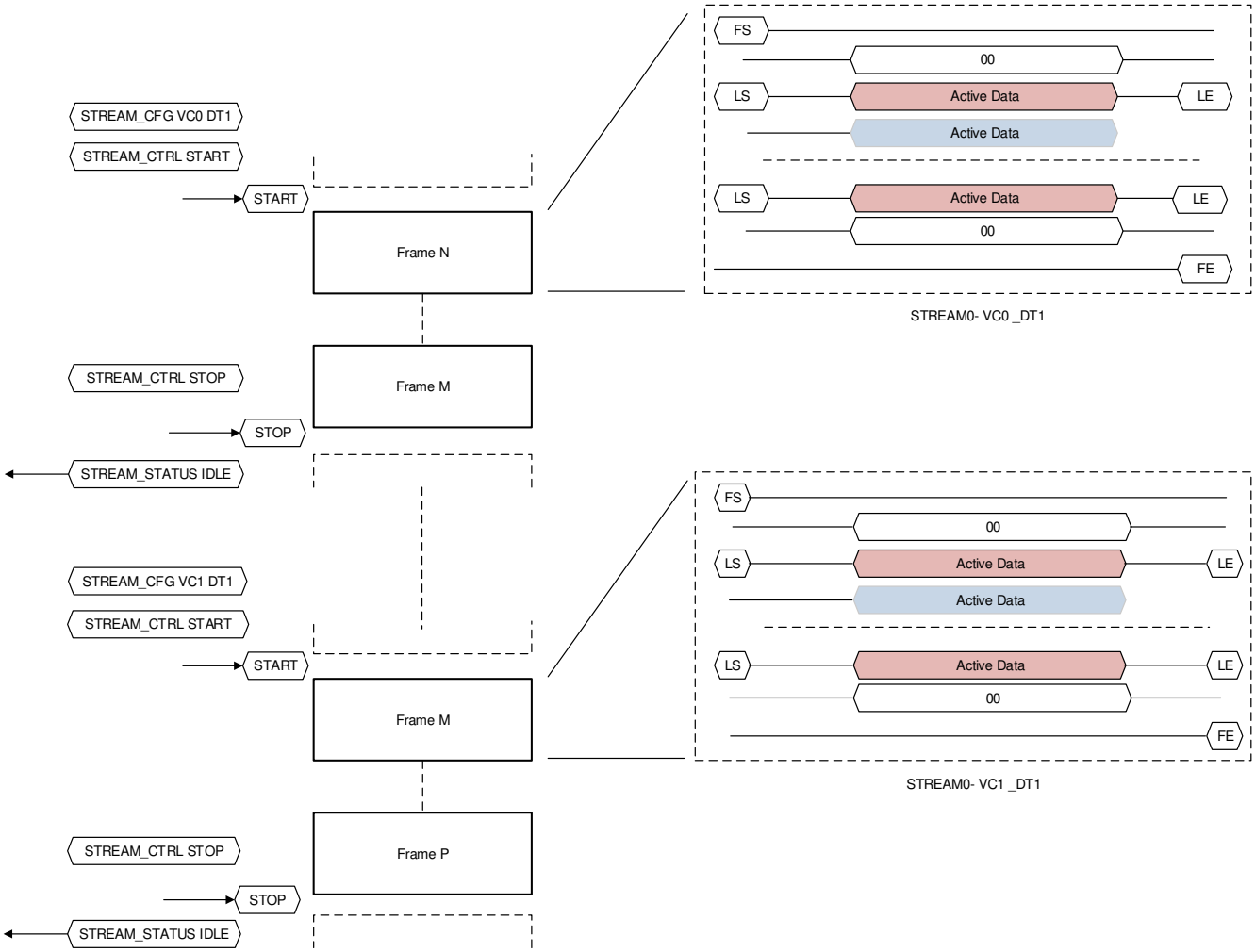


Figure 12-452. Stream Reconfiguration Using Start Stop Start Flow

The CSI_RX_IF will use the monitor control or STOP mechanism to stop the stream processing at the end of the active frame. The stream monitor, configuration and interrupt registers can be redefined and then the stream restarted. The pixel interface will begin processing at the next frame start.

Copyright © 2024 Texas Instruments Incorporated

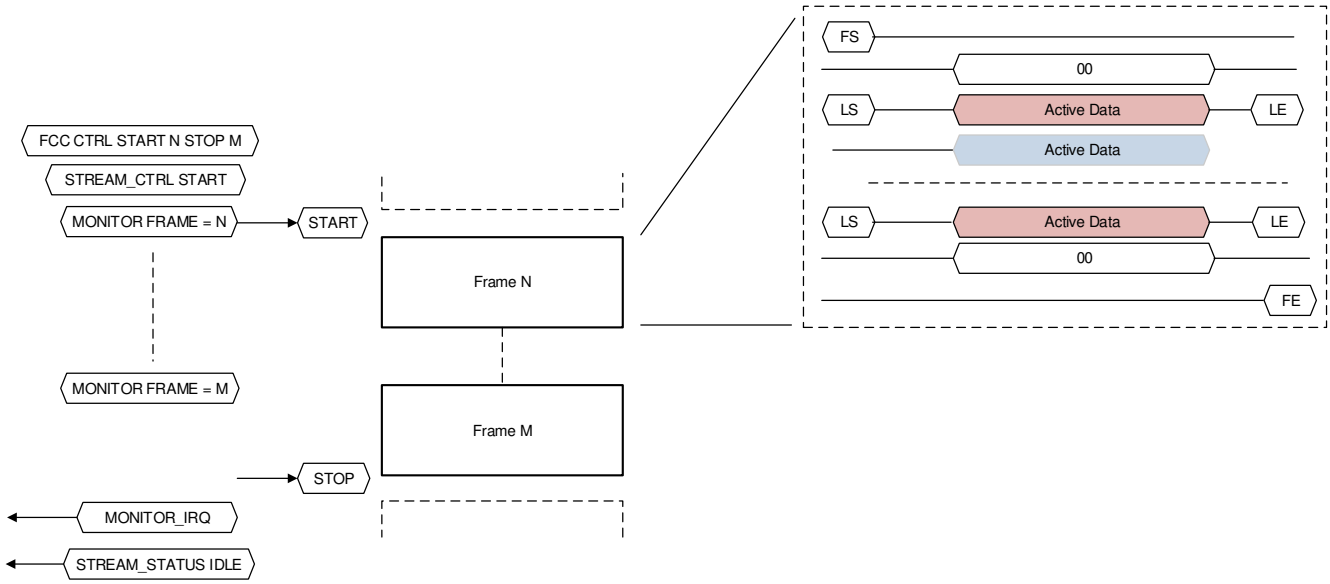


Figure 12-453. Stream Start and Stop Using Monitor Control

12.7.1.3.8.5 Error Control With Soft Resets

The CSI_RX_IF will perform control of the soft reset either for error event recovery or to clear a stream FIFO or internal state machine.

- The FRONT block can be soft reset if the DPHY_RX becomes unresponsive and the controller wishes to maintain its configuration. In this case the DPHY_RX resets can be applied and the DPHY_RX enabled to begin the transfer again.
- The Protocol block can be soft reset if the FRONT soft reset is required, and the protocol is not in the IDLE state.
- The stream soft resets (CSI_RX_IF_VBUS2APB_STREAM0_CTRL - CSI_RX_IF_VBUS2APB_STREAM3_CTRL)[4] SOFT_RST can be used to clear the stream to the stop state and reset all the stream state machines and FIFO. If the system has a failure and wishes to clear the stream FIFO and return to a safe state on the pixel interface, the stream soft reset should be asserted.

12.7.1.3.8.6 Stream Error Detected – No Error Bypass Mode

The CSI_RX_IF will default to stop processing the frame whenever a header Reserved DT or ECC error is detected, or when a long packet payload CRC is identified.

The CSI_RX_IF will detect the error condition and capture the associated VC, DT and WC information for the packet header before generating an error interrupt flag on CSI_RX_IF_VBUS2APB_ERROR_IRQS.

The CSI_RX_IF will stop processing the current frame and stop the stream.

Copyright © 2024 Texas Instruments Incorporated

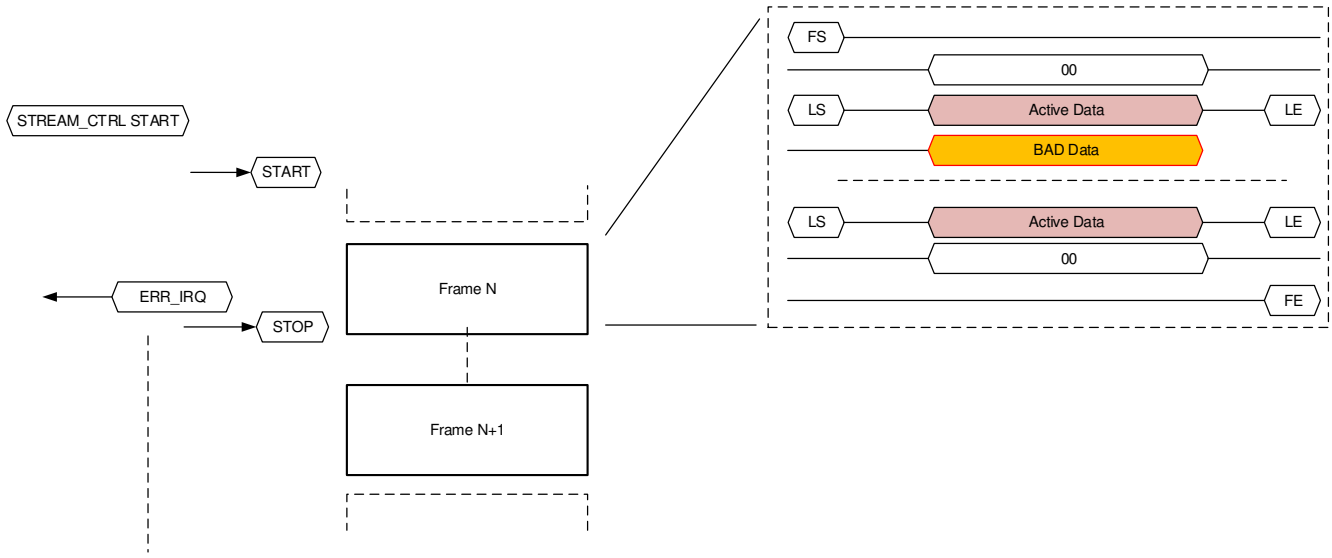


Figure 12-454. Stream Error Detection Causing Stop

12.7.1.3.8.7 Stream Error Detected – Error Bypass Mode

The CSI_RX_IF will detect the error condition and capture the associated VC, DT and WC information for the packet header before generating an error interrupt flag on CSI_RX_IF_VBUS2APB_ERROR_IRQS.

The CSI_RX_IF will continue processing the current frame and continue the stream when a header Reserved DT or a long packet payload CRC error is detected.

The CSI_RX_IF will stop processing the current frame and continue the stream when a packet header ECC error is detected, as the current frame synchronisation cannot be maintained.

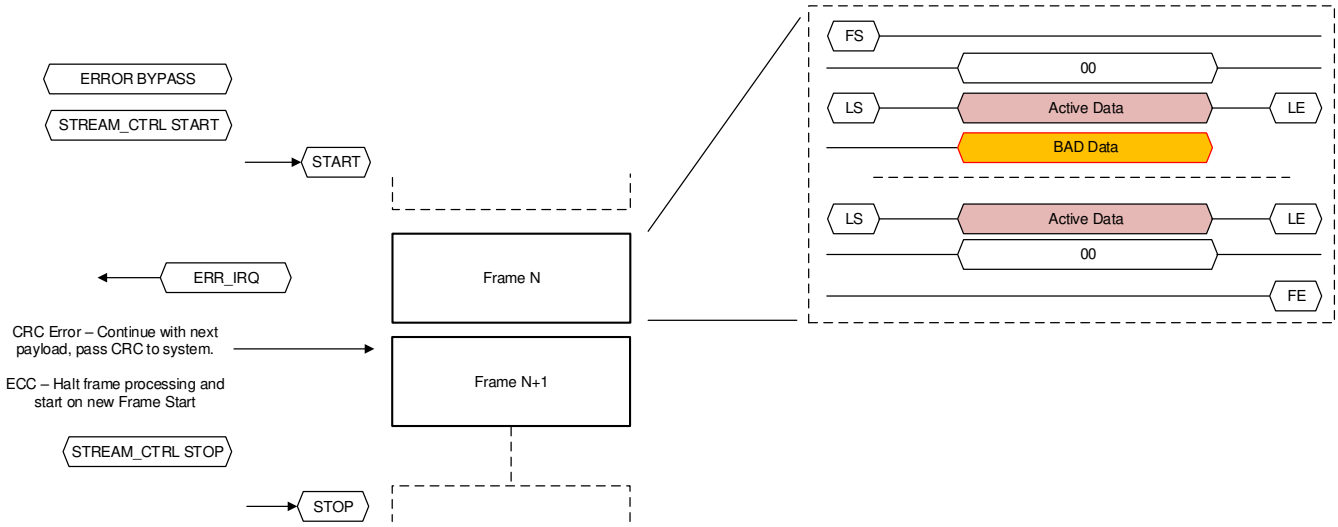


Figure 12-455. Stream Error Bypass - CRC Error During Payload

12.7.1.3.8.8 Stream Error Detected – Soft Reset Recovery

The CSI_RX_IF will stop processing the current frame and continue the stream when a packet header ECC error is detected, as the current frame synchronisation cannot be maintained.

The system can perform a soft reset on the individual stream to clear the pixel interface and payload FIFO and allow the system to restart the stream from the next frame start.

Copyright © 2024 Texas Instruments Incorporated

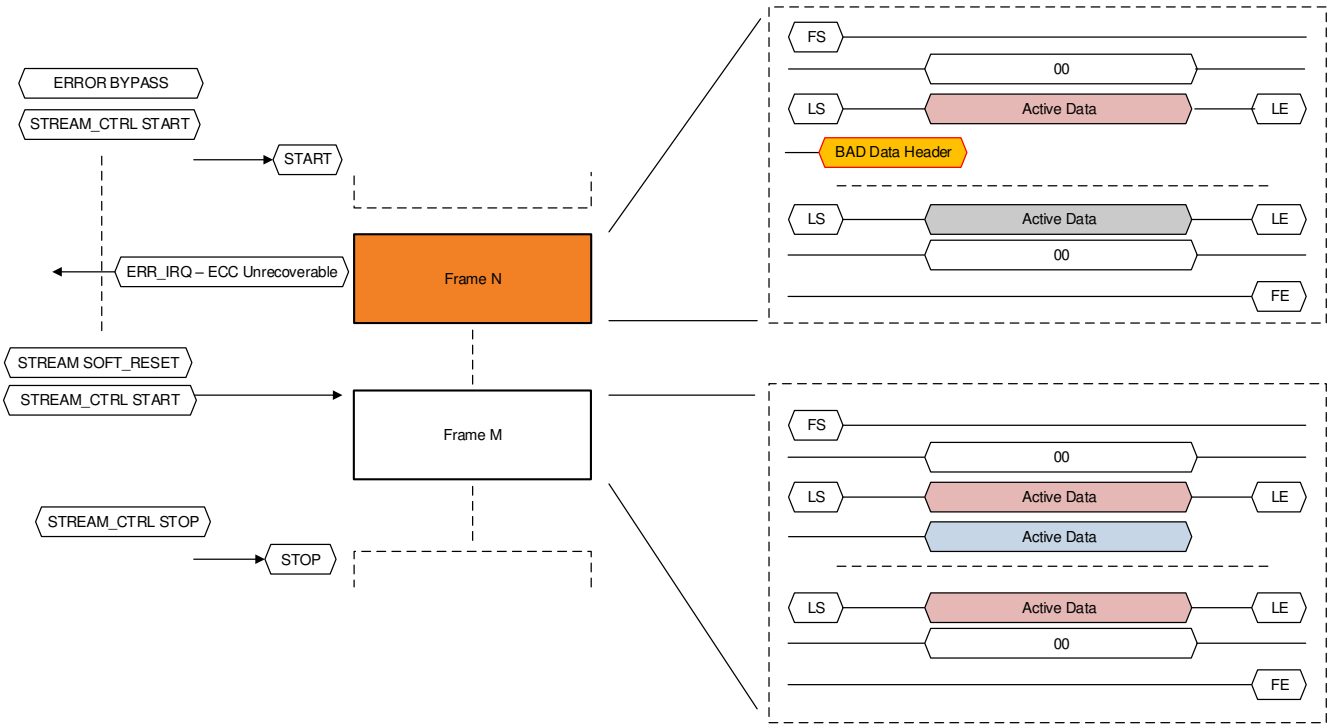


Figure 12-456. Stream Soft Reset After ECC Non-Recoverable Error

12.7.1.3.8.9 Stream Monitor Configuration

The CSI_RX_IF will use the monitor control (CSI_RX_IF_VBUS2APB_STREAM0_MONITOR_CTRL - CSI_RX_IF_VBUS2APB_STREAM3_MONITOR_CTRL) or STOP mechanism to stop the stream processing at the end of the active frame. The stream monitor, configuration and interrupt registers can be redefined and then the stream restarted. The pixel interface will begin processing at the next frame start.

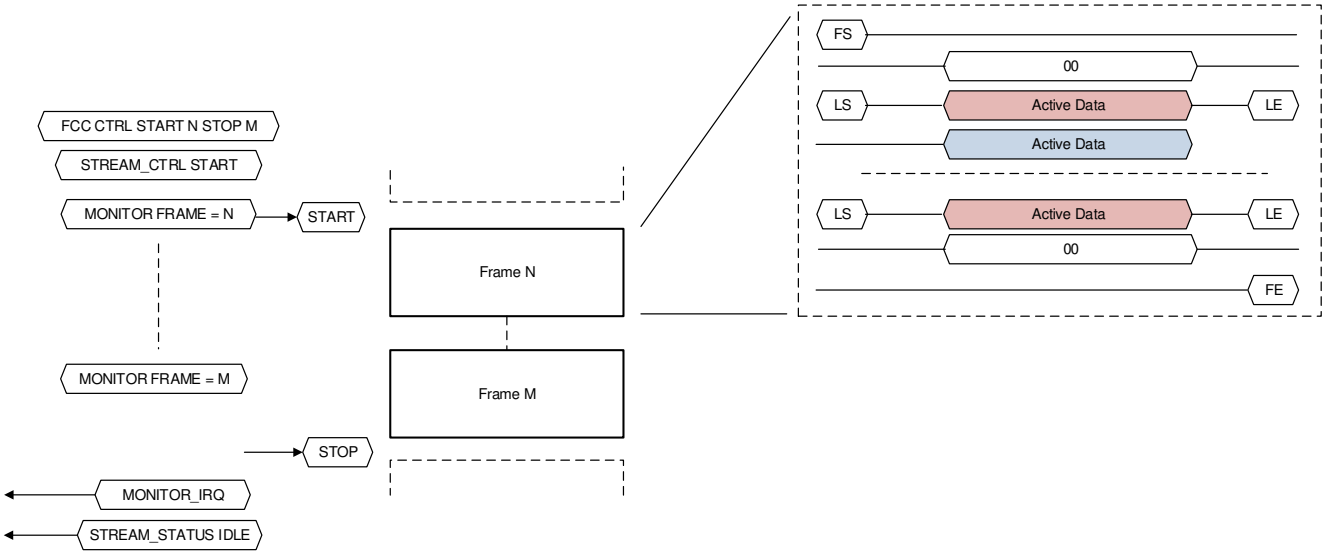


Figure 12-457. Stream Start and Stop Using Monitor Control

The FW will use the FCC config control register (CSI_RX_IF_VBUS2APB_STREAM0_FCC_CFG - CSI_RX_IF_VBUS2APB_STREAM3_FCC_CFG) to perform a start and stop as soon as a frame count is detected. The FCC config control register will define the start and stop frame count and the stream will identify when these values are matched. The stream output will begin when start frame is detected and then stop once the stop frame is reached. The virtual channel can be defined

Copyright © 2024 Texas Instruments Incorporated

in CSI_RX_IF_VBUS2APB_STREAM0_FCC_CTRL - CSI_RX_IF_VBUS2APB_STREAM3_FCC_CTRL[4-1] FCC_VC, and must match the virtual channels that are available to the stream in the CSI_RX_IF_VBUS2APB_STREAM0_DATA_CFG - CSI_RX_IF_VBUS2APB_STREAM3_DATA_CFG register.

The frame capture is enabled when CSI_RX_IF_VBUS2APB_STREAM0_FCC_CTRL - CSI_RX_IF_VBUS2APB_STREAM3_FCC_CTRL[0] FCC_EN is set ‘1’

The FW can check the current frame counter value from the CSI_RX_IF_VBUS2APB_STREAM0_FCC_CTRL - CSI_RX_IF_VBUS2APB_STREAM3_FCC_CTRL[31:16] FRAME_COUNTER

12.7.1.3.8.10 Stream Monitor Frame Capture Control

To use Monitor Frame Capture Control Start and Stop operations:

1. Set Stop and Start values in config register (CSI_RX_IF_VBUS2APB_STREAM0_FCC_CFG - CSI_RX_IF_VBUS2APB_STREAM3_FCC_CFG). Set to ‘0’ for free-running.
   a. [31:16] FRAME_COUNT_STOP -> define desired value
   b. [15:0] FRAME_COUNT_START -> define desired value
2. Set the Virtual Channel to the one that is enabled and the enable in register CSI_RX_IF_VBUS2APB_STREAM0_FCC_CTRL - CSI_RX_IF_VBUS2APB_STREAM3_FCC_CTRL.
   a. [4:1] FCC_VC -> define VC used
   b. [0] FCC_EN -> set to ‘1’

This will allow visual checking, by setting the Start/Stop as the following examples, this is the behaviour to be observed:

- Stop = 1 / Start = 1: Output (i.e. display) will show a Frame of the Input (i.e. camera)
- Stop = F / Start = 1: Output will show live image from the input during 14 frames and then will freeze in the last one.

Optionally software can read back the frame count value from register (CSI_RX_IF_VBUS2APB_STREAM0_FCC_CTRL - CSI_RX_IF_VBUS2APB_STREAM3_FCC_CTRL).

In order to do that, software will first need to enable the monitor control register.

1. Define the same Virtual Channel as above and enable the monitor control register (CSI_RX_IF_VBUS2APB_STREAM0_MONITOR_CTRL - CSI_RX_IF_VBUS2APB_STREAM3_MONITOR_CTRL).
   a. [15] FRAME_MON_EN -> set to ‘1’
   b. [14:11] FRAME_MON_VC -> define VC used
2. Read frame_counter containing the current frame number being processed from CSI_RX_IF_VBUS2APB_STREAM0_FCC_CTRL - CSI_RX_IF_VBUS2APB_STREAM3_FCC_CTRL (only when frame numbers are coming from the protocol, not when internal counter)
   a. [31:16] FRAME_COUNTER -> read frame number value

The easiest way to check that the frame number matches the Start and/or Stop values defined, is to use the interrupts in CSI_RX_IF_VBUS2APB_MONITOR_IRQS register. To be able to do that you will need to allow those interrupts through the mask register CSI_RX_IF_VBUS2APB_MONITOR_IRQS_MASK_CFG.

1. By default, the CSI_RX_IF_VBUS2APB_MONITOR_IRQS_MASK_CFG register is set to all ‘0’s, meaning all interrupts are disabled.
   a. [4] STREAM0_FCC_STOP_IRQM -> set to ‘1’
   b. [3] STREAM0_FCC_START_IRQM -> set to ‘1’
2. Create an interrupt handler to read from CSI_RX_IF_VBUS2APB_MONITOR_IRQS.
   a. [4] STREAM0_FCC_STOP_IRQ -> read, if ‘1’, Stop interrupt triggered
   b. [3] STREAM0_FCC_START_IRQ -> read, if ‘1’, Start interrupt triggered

To ensure correct functionality, it is highly recommended that the input device (i.e. camera) should be configured after the CSI_RX_IF has been configured and enabled if software is using Frame Counter Control features.

Software must make sure all the Virtual Channel fields in the registers explained above are set to the correct VC being used by the stream.

Copyright © 2024 Texas Instruments Incorporated

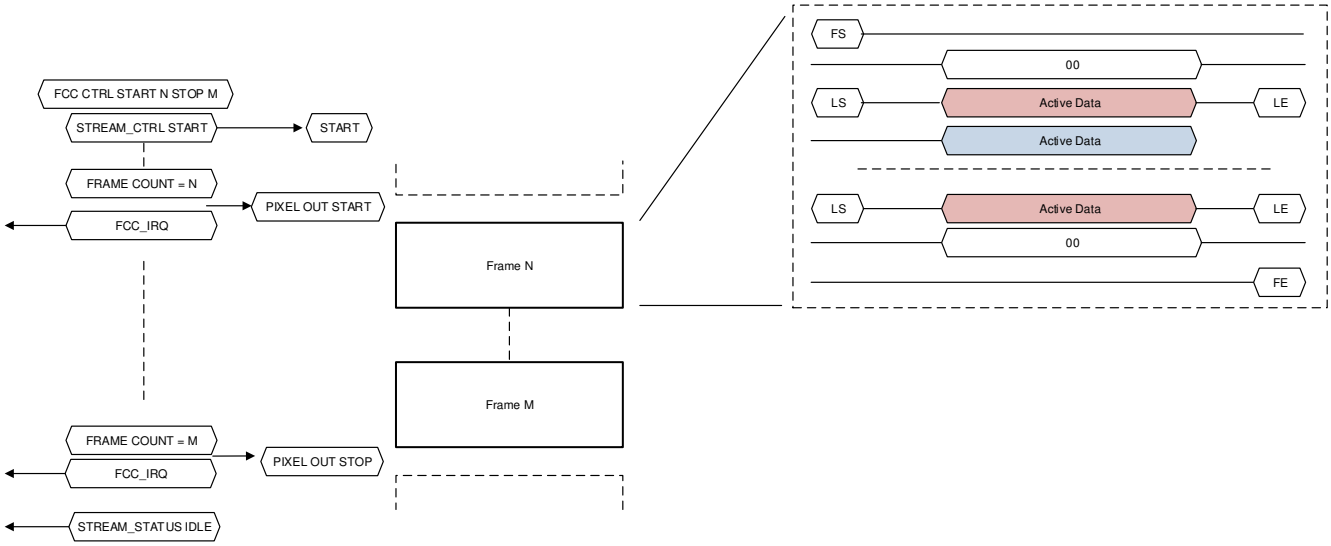


Figure 12-458. Stream Frame Capture Control Flow Diagram

12.7.1.3.8.11 Stream Monitor Timer interrupt

To use the Stream Monitor Timer operations:

1. Set Count value in timer register (CSI_RX_IF_VBUS2APB_STREAM0_TIMER - CSI_RX_IF_VBUS2APB_STREAM3_TIMER). If set to 0 won't count but interrupt will still trigger every EOF or SOF.
   a. [24-0] COUNT -> define desired value
2. Set the Virtual Channel to the one that is enabled, define timer_eof and set enable in the monitor controll register (CSI_RX_IF_VBUS2APB_STREAM0_MONITOR_CTRL - CSI_RX_IF_VBUS2APB_STREAM3_MONITOR_CTRL).
   a. [15] FRAME_MON_EN -> set to '1'
   b. [14:11] FRAME_MON_VC -> define VC used
   c. [10] TIMER_EOF -> set to '1' to count from EOF, set to '0' to count from SOF
   d. [9] TIMER_EN -> set to '1'
   e. [8:5] TIMER_VC -> define VC used

Software should now use the interrupts in register CSI_RX_IF_VBUS2APB_MONITOR_IRQS. To be able to do that you will need to allow those interrupts through the mask register CSI_RX_IF_VBUS2APB_MONITOR_IRQS_MASK_CFG.

1. By default, the CSI_RX_IF_VBUS2APB_MONITOR_IRQS_MASK_CFG register is set to all '0', meaning all interrupts are disabled.
   a. [0] STREAM0_TIMER_IRQM -> set to '1'
2. Create an interrupt handler to read from monitor_irqs.
   a. [0] STREAM0_TIMER_IRQ -> read, if '1', timer interrupt is triggered

To ensure correct functionality, it is highly recommended that the input device (i.e. camera) should be configured after the CSI_RX_IF has been configured and enabled.

Software must make sure all the Virtual Channel fields in the registers explained above are set to the correct VC being used by the stream.

Copyright © 2024 Texas Instruments Incorporated

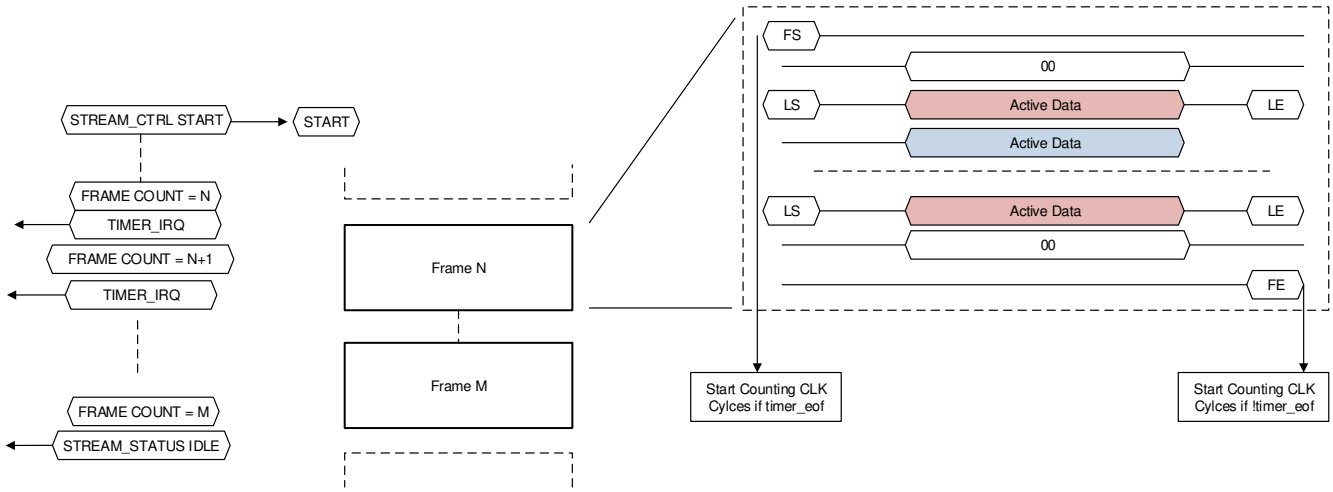


Figure 12-459. Timer Interrupt Flow Diagram

12.7.1.3.8.12 Stream Monitor Line/Byte Counters Interrupt

To use the stream Monitor Line and Byte counters operations:

1. Set LineCount and ByteCount values in register (CSI_RX_IF_VBUS2APB_STREAM0_MONITOR_LB - CSI_RX_IF_VBUS2APB_STREAM3_MONITOR_LB).
   a. [31:16] LINE_COUNT -> define desired value
   b. [15:0] BYTE_COUNT -> define desired value
2. Set the Virtual Channel to the one that is enabled, and set enable in register SI_RX_IF_VBUS2APB_STREAM0_MONITOR_CTRL.
   a. [15] FRAME_MON_EN -> set to ‘1’
   b. [14:11] FRAME_MON_VC -> define VC used
   c. [4] LB_EN -> set to ‘1’
   d. [3:0] LB_VC -> set to VC used

Software should now use the interrupts in register CSI_RX_IF_VBUS2APB_MONITOR_IRQS. To be able to do that you will need to allow those interrupts through the mask register CSI_RX_IF_VBUS2APB_MONITOR_IRQS_MASK_CFG.

1. By default, the CSI_RX_IF_VBUS2APB_MONITOR_IRQS_MASK_CFG register is set to all ‘0’, meaning all interrupts are disabled.
   a. [7] STREAM0_LINE_CNT_E RROR_IRQM -> set to ‘1’
   b. STREAM0_LB_IRQM [1] -> set to ‘1’
2. Create an interrupt handler to read from CSI_RX_IF_VBUS2APB_MONITOR_IRQS.
   a. [7] STREAM0_LINE_CNT_E RROR_IRQ -> read, if ‘1’, line count error interrupt is triggered
   b. [1] STREAM0_LB_IRQ -> read, if ‘1’, line/byte counter interrupt is triggered

Note that the interrupt will trigger on each frame when it reaches byte number BYTE_COUNT in line number LINE_COUNT.

To ensure correct functionality, it is highly recommended that the input device (i.e. camera) should be configured after the CSI_RX_IF has been configured and enabled.

Software must make sure all the Virtual Channel fields in the registers explained above are set to the correct VC being used by the stream.

Copyright © 2024 Texas Instruments Incorporated

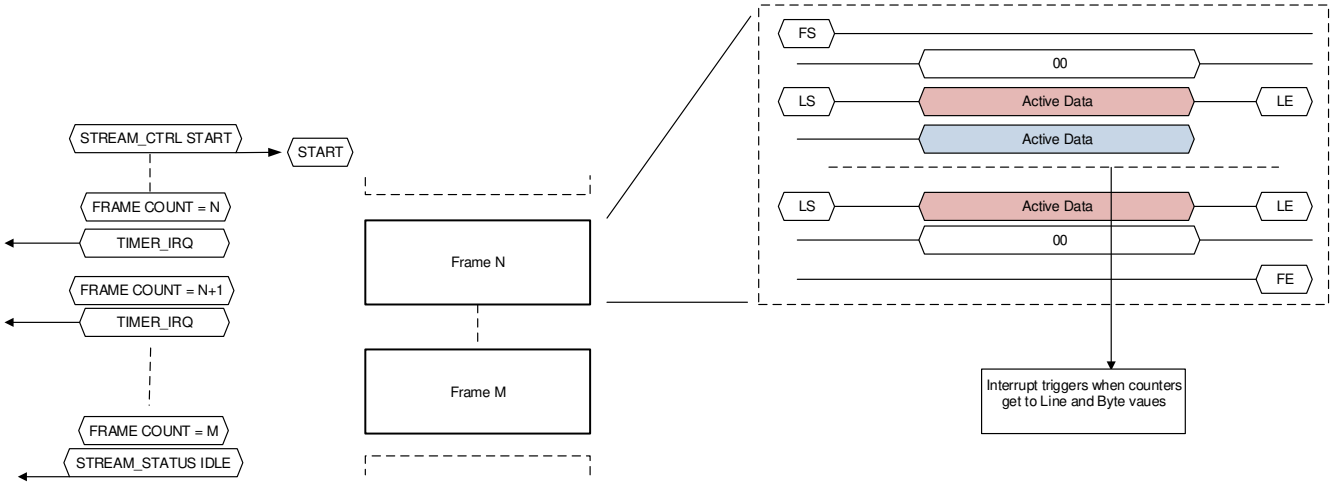


Figure 12-460. Line/Byte Counters Interrupt Flow Diagram

12.7.1.3.8.13 Example Controller Programming Sequence (Single Stream Operation)

The single stream operation will provide the smallest multi-lane IP configuration with the reduced registers configuration removing some control and status registers. The stream will not require a large FIFO configuration and the clock rates will be matched to simplify the implementation to single pixel transfers using all the available interface bits, (i.e. up to 32).

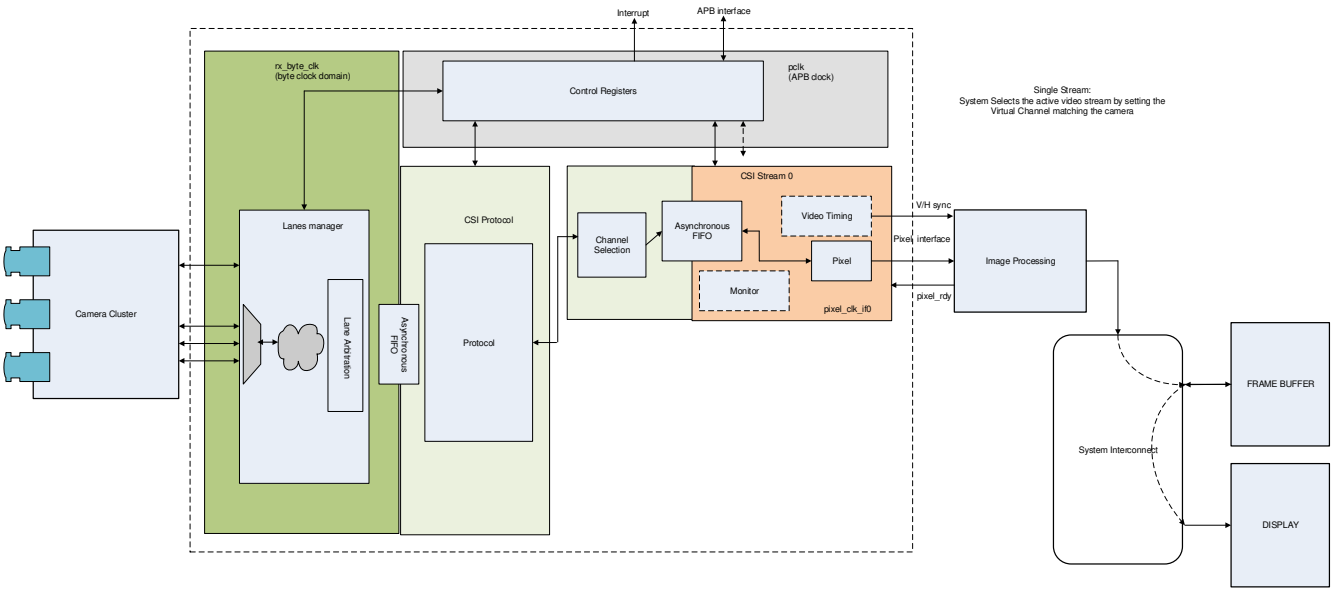


Figure 12-461. Basic Single Stream Use Case Illustration

For single pixel stream configuration, elastic buffer, RAW8 Data type on VC2, one pixel per cycle:

1. Configure the number of DPHY_RX lanes:
   a. See CSI_RX_IF_VBUS2APB_STATIC_CFG register
2. Set Error Interrupt mask:
   a. See CSI_RX_IF_VBUS2APB_ERROR_IRQS_MASK_CFG register
3. Set the Pixel Interface and FIFO configuration. See CSI_RX_IF_VBUS2APB_STREAM0_CFG - CSI_RX_IF_VBUS2APB_STREAM3_CFG.
   a. Set [9-8] FIFO_MODE to select the elastic buffer configuration -> ‘1’
   b. Set the FIFO fill level that will determines the FIFO depth that must be reached before data is output -> 0x0000

Copyright © 2024 Texas Instruments Incorporated

   c. Select the number of pixels to be output on each cycle -> ‘0’.
   d. Set the type of stream interface to “pixel” mode -> ‘0’
4. Select the virtual channel and data types to be processed. See CSI_RX_IF_VBUS2APB_STREAM0_DATA_CFG - CSI_RX_IF_VBUS2APB_STREAM3_DATA_CFG register.
   a. Set [31-16] VC_SELECT and virtual channel bits if not supporting all virtual channels -> 1h, 2h
   b. Set [7] ENABLE_DT0 and [5-0] DATATYPE_SELECT0 values for one or both data types (default all DT) -> 0h, 1h, 2Ah
5. Enable the stream to begin processing the incoming data from the DPHY_RX. See CSI_RX_IF_VBUS2APB_STREAM0_CTRL - CSI_RX_IF_VBUS2APB_STREAM3_CTRL.
   a. Set [0] START bit -> ‘1’

The stream will begin to pass pixel data matching the configuration on the next frame start.

12.7.1.3.8.14 CSI_RX_IF Programming Restrictions

The following CSI_RX_IF programming restrictions apply:

- Full line buffer mode is not supported.
- LS/LE detection has a restriction to be disabled
- Only use Pixel transmit mode can be used, never packet mode
  – In pixel mode only single, dual, or quad mode can be used. It is not allowed to mix these based on data format or virtual channels.
- LSB alignment only
- In CSI_RX_IF to CSI_TX_IF loop back, the stream needs to have the same mode on both controllers(single, dual, quad). Can not mix dual on CSI_RX_IF and single on CSI_TX_IF.
- Video port(VP) stream interfaces must be programmed in the CSI_RX_IF to meet the following restrictions:
  – Raw 8-16 formats only
  – Dual pixel mode
  – Filtering for single virtual channel and single data type

Note

CSI_RX_IF has a limitation that any line must fully be exported on its stream interface before a new line or even end of frame is sent on the D-PHY interface. To meet this requirement, software will need to take into account export rates on stream interface(single, dual, quad modes) at 500MHz versus D-PHY 32-bits @ byte clock frequency to know when a new line or end of frame can be sent in after last line was sent over D-PHY interface.

Table 12-395 shows the CSI_RX_IF programming requirements for pixel modes and data sizes.

Table 12-395. CSI_RX_IF Programming requirements for pixel modes and data sizes

Format	Pixel transmit mode used	Size	Details
RAW(6-8), user defined 8-bit	Single	0	
RAW(6-8), user defined 8-bit	Dual	1	
RAW(6-8), user defined 8-bit	Quad	2	
RAW(10-16), user defined 16-bit	Single	1	
RAW(10-16), user defined 16-bit	Dual	2	
RAW20	Single	2	
YUV422-8	Single	0	must set dual mode=0
YUV422-8	Dual	0	must set dual mode=1
YUV422-10	Single	1	
YUV420-8	Single	0	must set dual mode=0
YUV420-8	Dual	0	must set dual mode=1
YUV420-10	Single	1	

Copyright © 2024 Texas Instruments Incorporated

12.7.1.3.8.15 CSI_RX_IF Real-time operating requirements

Care must be taken on blanking requirements for next line, end of frame, or start of frame. The previous packets(long or short) must be fully consumed before any new line, end of frame, or start frame can be sent over the CSI interface. Software will get various error conditions if these are not met. Below are some sample equations to determine this blanking time. Note that this is not blanking data. High level details are that the internal FIFO must be completely empty prior to sending any thing new into it. below are some sample equations to determine how much time is needed to empty the FIFO and blanking time needed. Byte clock rates, pixel data type, and export modes play into the calculation as well.

$$\text{Byte_clock_rate} \times 8 \times \#\text{lanes} \times \text{Tcsi} = \text{pixel_clock_rate} \times \text{BPP} \times \text{Tpix} = \text{total bits} \quad (27)$$

BPP = Bits Per Pixel. This number is how many bits are sent per pixel clock cycle is based on single, dual, quad modes and data type.

Tcsi = Time for csi interface to complete 1 line
Tpix = Time for pixel interface to complete 1 line
Tpix16 = Tpix+16/pixel_clock_rate

(Tpix16-Tcsi) = required blanking time on CSI interface
Tpix16-Tcsi <= 0 implies 0

Copyright © 2024 Texas Instruments Incorporated

12.7.2 MIPI D-PHY Receiver (DPHY_RX)

The following sections describe the MIPI D-PHY receiver (DPHY_RX) modules in the device.

12.7.2.1 DPHY_RX Overview

The integration of the DPHY_RX camera physical port module allows the device to grab video streams from external sensor cameras and other CSI2 compliant sources.

12.7.2.1.1 DPHY_RX Features

The DPHY_RX module supports the following features:

- Compliant to MIPI D-PHY standard v1.2
- Supports up to 4 data and 1 clock lanes
- Supports up to 2.5 Gbps (with deskew) and 1.5 Gbps (without deskew) per data lane
- Clock lane control / interface logic type is CIL-SCNN for HS and low power receiving:
  1. **S**lave
  2. **C**lock
  3. **N**/A forward, **N**/A reverse escape mode features
- Data lane control / interface logic type is CIL-SFAN for HS and low power receiving:
  1. **S**lave
  2. **F**orward direction only for high speed mode
  3. **A**ll forward direction escape mode features are supported
  4. **N**o reverse direction escape mode features are supported
- Data lanes can be independently operated in HS or ULP mode
- Swapping of DP/DN signals within each clock/data pair (Facilitated by CSI_RX_IF controller)

Unsupported Features:

- See the Module Integration section of this document for a list of module features not supported by the integration on this Device.

Copyright © 2024 Texas Instruments Incorporated

12.7.2.1.2 DPHY_RX Ports

This section describes the DPHY_RX ports related to clocks, resets, and hardware requests.

Table 12-396. DPHY_RX Clocks and Resets

Clocks	
Module Clock Input	**Description**
DPHY_RX_MAIN_CLK	Main functional clock.
CSI_RX_BYTE_CLK	The byte clock is the clock supplied by the DPHY_RX.
Resets	
Module Reset Input	**Description**
DPHY_RX_RST	Asynchronous module global reset.

12.7.2.2 DPHY_RX Environment

This section describes the DPHY_RX application fields from an environment point of view (external connections).

Table 12-397 describes the external signals of the DPHY_RX.

Table 12-397. DPHY_RX I/O Signals

Device Level Signal	I/O	Description
CSI0_RXN0	I	Lane 0 Receive Differential Data (Negative)
CSI0_RXP0	I	Lane 0 Receive Differential Data (Positive)
CSI0_RXN1	I	Lane 1 Receive Differential Data (Negative)
CSI0_RXP1	I	Lane 1 Receive Differential Data (Positive)
CSI0_RXN2	I	Lane 2 Receive Differential Data (Negative)
CSI0_RXP2	I	Lane 2 Receive Differential Data (Positive)
CSI0_RXN3	I	Lane 3 Receive Differential Data (Negative)
CSI0_RXP3	I	Lane 3 Receive Differential Data (Positive)
CSI0_RXCLKN	I	Lane 3 Receive Differential Clock (Negative)
CSI0_RXCLKP	I	Lane 3 Receive Differential Clock (Positive)
CSI0_RXRCALIB	A	Pin for external calibration resistor. An external resistor must be connected between this pin and package ground. Refer to the device-specific Datasheet for a recommended resistor value.
CSI1_RXN0	I	Lane 0 Receive Differential Data (Negative)
CSI1_RXP0	I	Lane 0 Receive Differential Data (Positive)
CSI1_RXN1	I	Lane 1 Receive Differential Data (Negative)
CSI1_RXP1	I	Lane 1 Receive Differential Data (Positive)
CSI1_RXN2	I	Lane 2 Receive Differential Data (Negative)
CSI1_RXP2	I	Lane 2 Receive Differential Data (Positive)
CSI1_RXN3	I	Lane 3 Receive Differential Data (Negative)
CSI1_RXP3	I	Lane 3 Receive Differential Data (Positive)
CSI1_RXCLKN	I	Lane 3 Receive Differential Clock (Negative)
CSI1_RXCLKP	I	Lane 3 Receive Differential Clock (Positive)
CSI1_RXRCALIB	A	Pin for external calibration resistor. An external resistor must be connected between this pin and package ground. Refer to the device-specific Datasheet for a recommended resistor value.

Copyright © 2024 Texas Instruments Incorporated

12.7.2.3 DPHY_RX Functional Description

12.7.2.3.1 DPHY_RX Programming Guide

12.7.2.3.1.1 Overview

This section details specific steps on how to program the DPHY_RX

12.7.2.3.1.2 Initial Configuration Programming

The DPHY_RX operation shows the registers that are written during the startup sequence.

This section provides description and timing diagrams for DPHY_RX operation. Unless otherwise described in this subsection, the DPHY_RX PPI interface is compliant with the timing diagrams described in the MIPI D-PHY v1.2 specification.

12.7.2.3.1.2.1 Start-up Sequence Timing Diagram

The common configuration pins noted in Figure 12-462 are ipconfig_cmn, pll_ipdiv, pll_fbdiv, pll_opdiv, psm_clock_freq_cmn. Drive these pins as per the pin description given in PHY pin list.

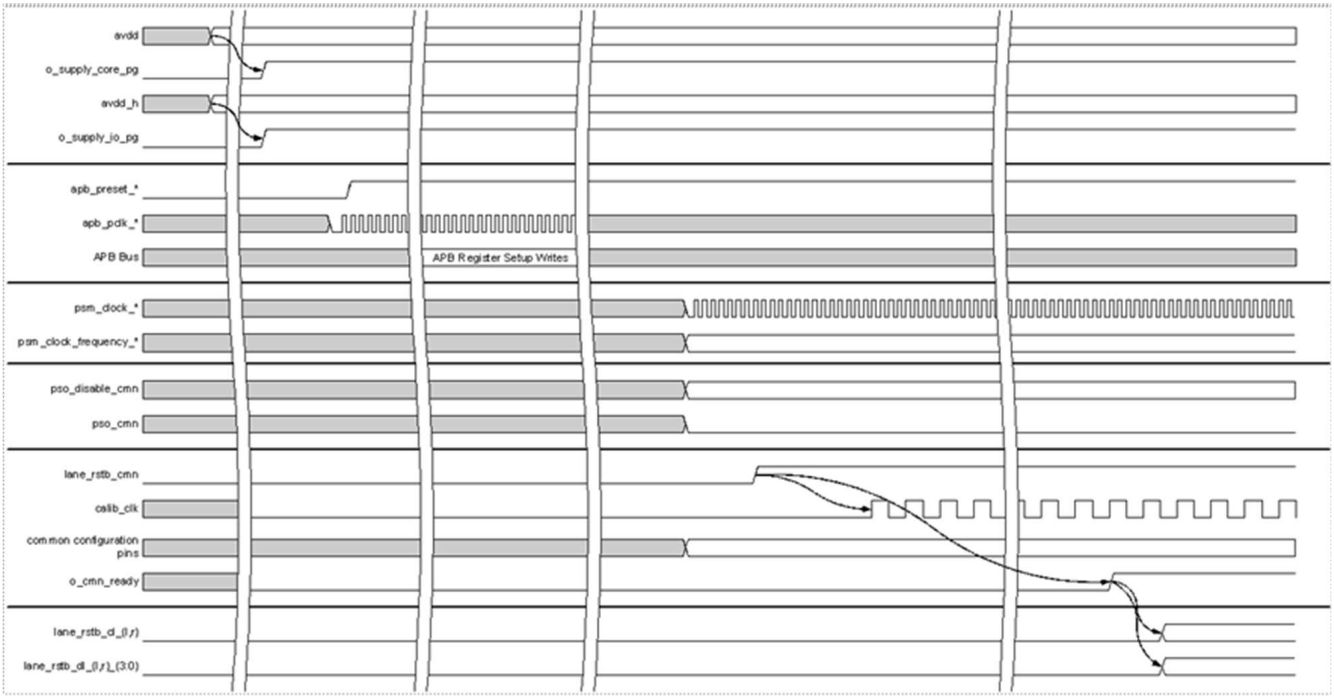


Figure 12-462. Common Power Up and Initialization Timing Diagram

Copyright © 2024 Texas Instruments Incorporated

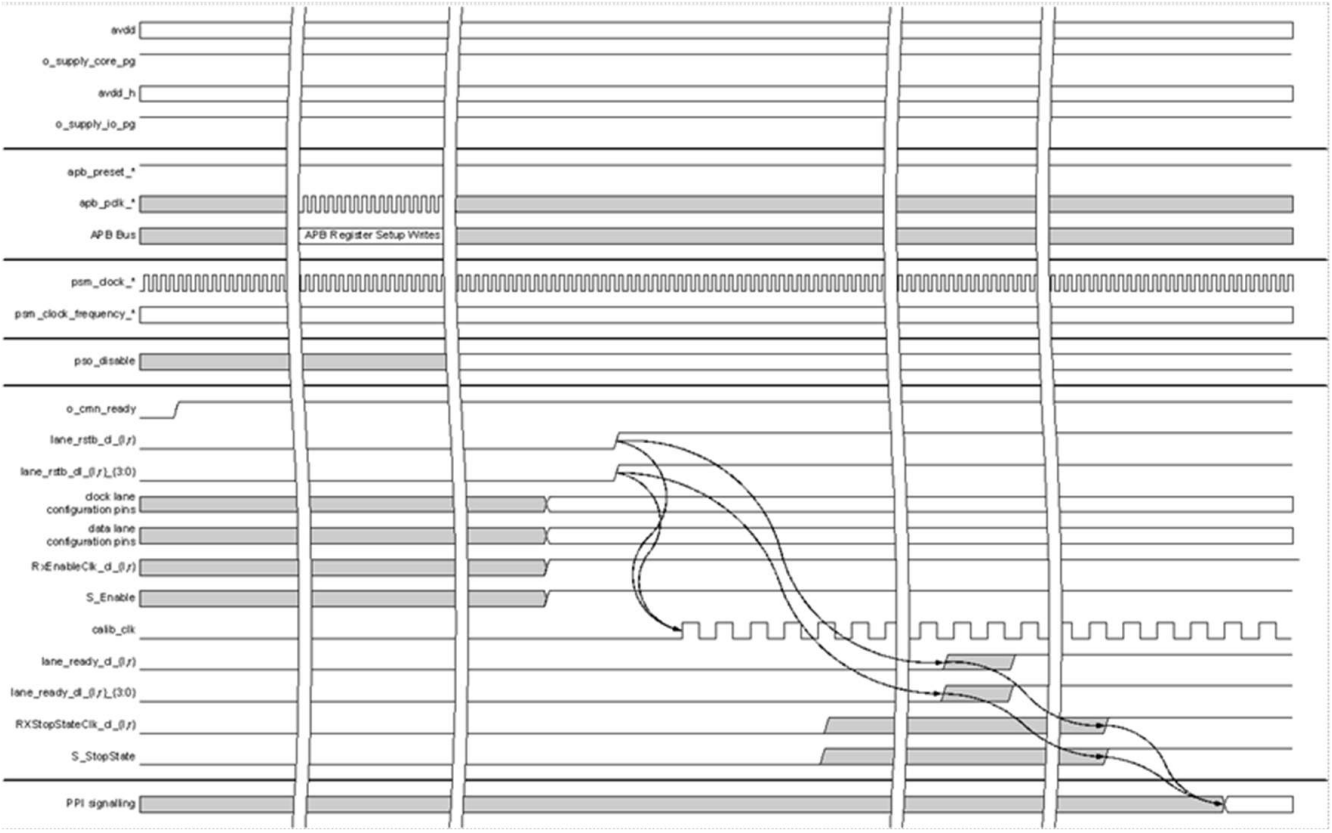


Figure 12-463. Lane Power Up and Initialization Timing Diagram

Copyright © 2024 Texas Instruments Incorporated

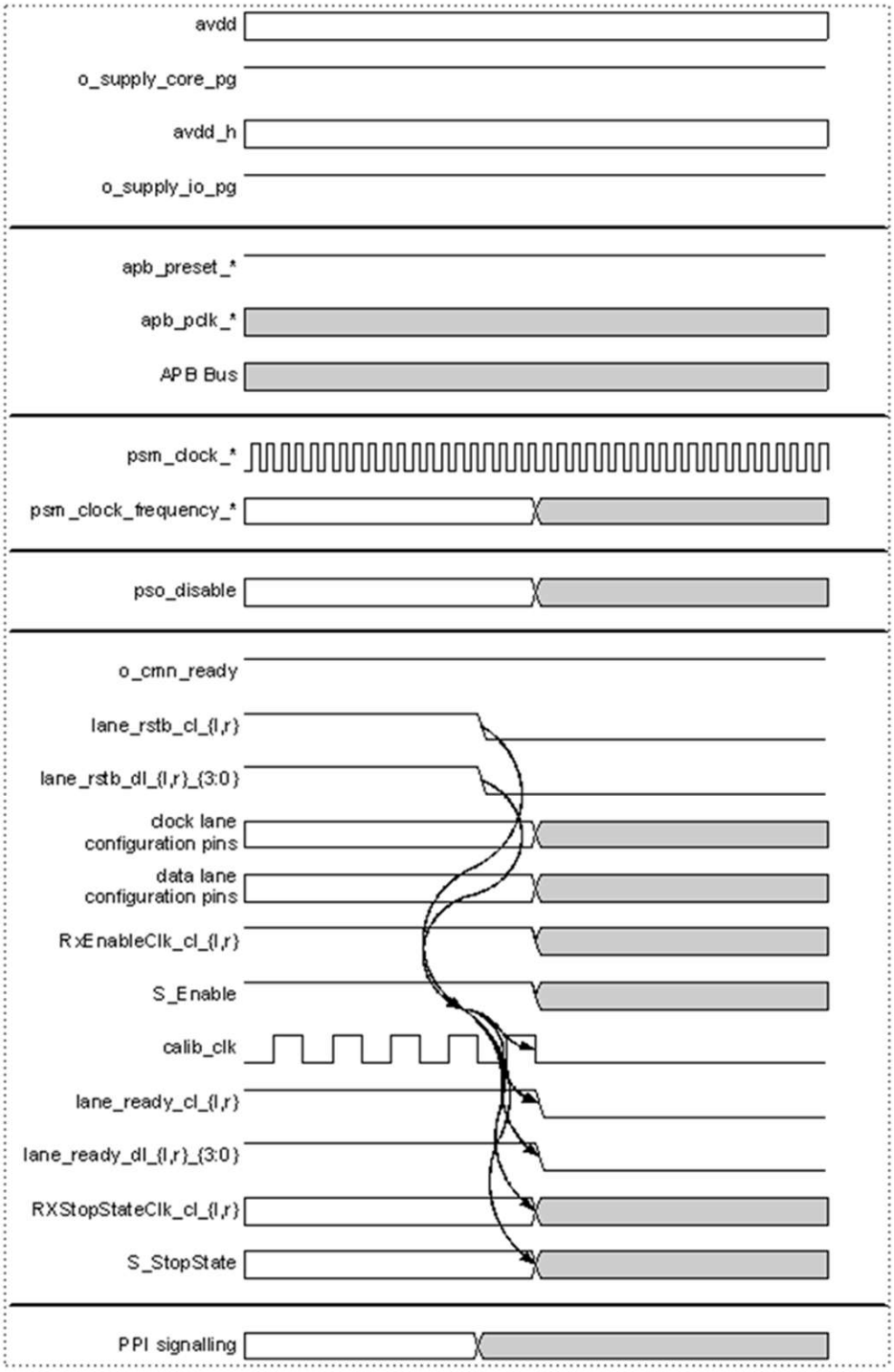


Figure 12-464. PHY Disable Timing Diagram

For initial set up, the DPHY_RX must be configured/set (registers and configuration input pins) for the common module prior to releasing it from reset, and the lane modules prior to releasing them from reset. Registers shall be configured (as required) between releasing the APB from reset and releasing the common / lanes from reset.

Note that in some cases, the option exists to configure a function using either a pin or a register. In such cases, both options will be specified and the you can select the preferred option.

Copyright © 2024 Texas Instruments Incorporated

12.7.2.3.1.3 Common Configuration

Table 12-398. Common Configuration-Related Setup

Configuration Register / Pin	Configuration Requirement
PHY Pin: psm_clock_freq PHY Register: DPHY_RX_VBUS2APB_PCS_TX_DIG_TBIT1	Set the PMA state machine clock frequency divider. Set either the pin or the register, as specified in the respective description, for the required PMA state machine clock.
PHY Pin: ipconfig_cmn PHY Register: DPHY_RX_MMR_SLV_LANE[11-9] IPCONFIG_CMN	Set the clock lane configuration. Set as specified in the description for the required clock lane configuration.
PMA Register: DPHY_RX_VBUS2APB_CMN0_CMN_DIG_TBIT2[0] O_CMN_SSM_EN, DPHY_RX_VBUS2APB_CMN0_CMN_DIG_TBIT2[10] O_CMN_RX_MODE_EN	Enable the startup state machines for TX mode of operation. Set both register bits to 1'b1.
PMA Register: DPHY_RX_VBUS2APB_CMN0_CMN_DIG_TBIT35	Set the RX oscillator calibration feedback clock counter start values. Set the register fields, as specified in the description for the required PMA state machine clock and oscillator clock.
PHY Register: DPHY_RX_VBUS2APB_PCS_TX_DIG_TBIT2	Set the required power island phase 2 time. Set the register to 32'hAAAAAAAA.
PHY Register: DPHY_RX_VBUS2APB_PCS_TX_DIG_TBIT3[7:4] POWER_SW_2_TIME_CL_R, DPHY_RX_VBUS2APB_PCS_TX_DIG_TBIT3[3:0] POWER_SW_2_TIME_CL_L	Set the required power island phase 2 time. Set the register to 8'hAA

12.7.2.3.1.4 Lane Configuration

Table 12-399. Lane Configuration-Related Setup

Configuration Register / Pin	Configuration Requirement
PHY Register: DPHY_RX_VBUS2APB_PCS_TX_DIG_TBIT0	Set the lane band control value. Set the register fields, as specified in the register description for the required data rate.
PMA Register (clock lanes): DPHY_RX_VBUS2APB_CLK0_RX_DIG_TBIT2[18:15] RXDA_FREQ_BAND_SEL1 PMA Register (clock lanes): DPHY_RX_VBUS2APB_CLK0_RX_DIG_TBIT2[13:10] RXDA_FREQ_BAND_SEL2	Set the analog frequency band selects, as specified in the register description for the required data rate.
PMA Register (data lanes): DPHY_RX_VBUS2APB_DL0_RX_DIG_TBIT0 - DPHY_RX_VBUS2APB_DL3_RX_DIG_TBIT0[21] TM_1P5TO2P5G_MODE_EN	Set the analog data rate select, as specified in the register description for the required data rate.
PMA Register (data lanes): DPHY_RX_VBUS2APB_DL0_RX_DIG_TBIT0 - DPHY_RX_VBUS2APB_DL3_RX_DIG_TBIT0[16] TM_SETTLE_COUNT_SEL and [15-9] TM_SETTLE_COUNT	Set the HS-settle counter, as specified in the register description for the required data rate.
PMA Register (data lanes): DPHY_RX_VBUS2APB_DL0_RX_DIG_TBIT3 - DPHY_RX_VBUS2APB_DL3_RX_DIG_TBIT3	Set the iteration and initialization wait timer values to create a delay of 500 nSec as a function of the PSM clock rate.
PMA Register (data lanes): DPHY_RX_VBUS2APB_DL0_RX_DIG_TBIT5 - DPHY_RX_VBUS2APB_DL3_RX_DIG_TBIT5	Set the iteration and initialization wait timer values to create a delay of 500 nSec as a function of the PSM clock rate.

Copyright © 2024 Texas Instruments Incorporated

Table 12-399. Lane Configuration-Related Setup (continued)

Configuration Register / Pin	Configuration Requirement
PMA Register (data lanes): DPHY_RX_VBUS2APB_DL0_RX_DIG_TBIT7 - DPHY_RX_VBUS2APB_DL3_RX_DIG_TBIT7	Set the iteration and initialization wait timer values to create a delay of 500 nSec as a function of the PSM clock rate.
PMA Register (data lanes): DPHY_RX_VBUS2APB_DL0_RX_DIG_TBIT9 - DPHY_RX_VBUS2APB_DL3_RX_DIG_TBIT9	Set the iteration and initialization wait timer values to create a delay of 500 nSec as a function of the PSM clock rate.
PMA Register (data lanes): DPHY_RX_VBUS2APB_DL0_RX_DIG_TBIT12 - DPHY_RX_VBUS2APB_DL3_RX_DIG_TBIT12	Set the iteration and initialization wait timer values to create a delay of 500 nSec as a function of the PSM clock rate.

12.7.2.3.1.5 Procedure: Clock Lane Low Power Analog Receiver Functions Test

12.7.2.3.1.5.1 Description of Procedure

This procedure describes how the low power analog receiver functions (LPRX, and ULPRX) can be tested. The process used in this test is to enable test MUXes for the low power receiver functions, then write registers in common to drive LP signaling states to the lanes, and read registers in the lanes to detect the current state of the analog functions in the lane.

12.7.2.3.1.5.2 Details of the Procedure

1. Follow the procedure described in section Section 12.7.2.3.1.2, but do not release the common or lanes from reset.
2. Write the following registers.
   - PMA Register (cmn): DPHY_RX_VBUS2APB_CMN0_CMN_DIG_TBIT21[0] O_RX_DIG_BIST_EN = 1’b1, to enable the analog BIST power island in common.
   - PMA Register (cmn): DPHY_RX_VBUS2APB_CMN0_CMN_DIG_TBIT28[23:20] = 4’b1111, to enable the clock lane diagnostic low power override functions, and drive LP11 from common to the clock lanes.
   - PMA Register (clock lanes): DPHY_RX_VBUS2APB_DL0_RX_DIG_TBIT2[24] = 1’b1, to enable the diagnostic low power override MUXes in the analog.
   - PMA Register (clock lanes): DPHY_RX_VBUS2APB_DL0_RX_DIG_TBIT2[1:0] = 2’b11, to force the analog ULPS receiver to be enabled.
3. Release the common from reset, and wait for DPHY_RX_VBUS2APB_ISO_PHY_ISO_CMN_CTRL[5] O_CMN_READY to be driven to 1’b1.
4. Release the clock and data lanes from reset, and wait for DPHY_RX_VBUS2APB_ISO_PHY_ISO_CMN_CTRL[8] LANE_READY_CMN to be driven to 1’b1.
5. Poll PMA Register (CMN): DPHY_RX_VBUS2APB_CMN0_CMN_DIG_TBIT56[20-11] I_CMN_RX_SSM_STATE until it is set to 9’b100000000.
6. Repeat the following for each of the following values, to drive LP values from common and test the analog functions in the clock lanes: 2’b01, 2’b00, 2’b10, 2’b11.
   a. Write the PMA Register (cmn): DPHY_RX_VBUS2APB_CMN0_CMN_DIG_TBIT28[22,20] = value.
   b. Read and confirm the PMA Register (clock lanes): DPHY_RX_VBUS2APB_CLK0_RX_DIG_TBIT4[13:12] = value.
   c. Read and confirm the PMA Register (clock lanes): DPHY_RX_VBUS2APB_CLK0_RX_DIG_TBIT4[9:8] = value.

12.7.2.3.1.6 Procedure: Data Lane Low Power Analog Receiver Functions Test

12.7.2.3.1.6.1 Description of Procedure

This procedure describes how the low power analog receiver functions (LPRX, and ULPRX) can be tested. The process used in this test is to enable test MUXes for the low power receiver functions, then write registers in common to drive LP signaling states to the lanes, and read registers in the lanes to detect the current state of the analog functions in the lane.

Copyright © 2024 Texas Instruments Incorporated

12.7.2.3.1.6.2 Details of the Procedure

1. Follow the procedure described in section Section 12.7.2.3.1.2, but do not release the common or lanes from reset.
2. Write the following registers.
   - PMA Register (cmn): DPHY_RX_VBUS2APB_CMN0_CMN_DIG_TBIT21[0] O_RX_DIG_BIST_EN = 1'b1, to enable the analog BIST power island in common.
   - PMA Register (cmn): DPHY_RX_VBUS2APB_CMN0_CMN_DIG_TBIT28[19:16] = 4'b1111, to enable the data lane diagnostic low power override functions, and drive LP11 from common to the data lanes.
   - PMA Register (data lanes): DPHY_RX_VBUS2APB_DL0_RX_DIG_TBIT30[1] TM_LPRX_BIST_EN = 1'b1, to enable the diagnostic low power override MUXes in the analog.
   - PMA Register (data lanes): DPHY_RX_VBUS2APB_DL0_RX_DIG_TBIT1[9:8] = 2'b11, to force the analog ULPS receiver to be enabled.
3. Release the common from reset, and wait for DPHY_RX_VBUS2APB_ISO_PHY_ISO_CMN_CTRL[5] O_CMN_READY to be driven to 1'b1.
4. Release the clock and data lanes from reset, and wait for DPHY_RX_VBUS2APB_ISO_PHY_ISO_CMN_CTRL[8] LANE_READY_CMN to be driven to 1'b1.
5. Poll PMA Register (CMN): DPHY_RX_VBUS2APB_CMN0_CMN_DIG_TBIT56[20-11] I_CMN_RX_SSM_STATE until it is set to 9'b100000000.
6. Repeat the following for each of the following values, to drive LP values from common and test the analog functions in the data lanes: 2'b01, 2'b00, 2'b10, 2'b11.
   a. Write the PMA Register (cmn): DPHY_RX_VBUS2APB_CMN0_CMN_DIG_TBIT28[18,16] = value.
   b. Read and confirm the PMA Register (data lanes): DPHY_RX_VBUS2APB_DL0_RX_DIG_TBIT34[13:12] = value.
   c. Read and confirm the PMA Register (data lanes): DPHY_RX_VBUS2APB_DL0_RX_DIG_TBIT34[9:8] = value.

12.7.2.3.1.7 Procedure: Clock and Data Lane High Speed Receiver BIST Functions Test

12.7.2.3.1.7.1 Description of Procedure

This procedure describes how to use the high speed data BIST generation and checking functions to test the high speed data receiver functions. The process used in this test is to enable test MUXes for the low power and high speed data receiver functions, then write registers to setup, run and check the results of the BIST functions.

12.7.2.3.1.7.2 Details of the Procedure

1. Set up IP for BIST operations.
   a. Follow the procedure described in section Section 12.7.2.3.1.2, but do not release the common or lanes from reset.
   b. Write the following registers.
      - PMA Register (cmn): DPHY_RX_VBUS2APB_CMN0_CMN_DIG_TBIT21[0] O_RX_DIG_BIST_EN = 1'b1, to enable the analog BIST power island in common.
      - PMA Register (clock lanes): DPHY_RX_VBUS2APB_DL0_RX_DIG_TBIT2 - DPHY_RX_VBUS2APB_DL3_RX_DIG_TBIT2 [24] = 1'b1, to enable the diagnostic low power override MUXes in the analog.
      - PMA Register (clock lanes): DPHY_RX_VBUS2APB_DL0_RX_DIG_TBIT2 - DPHY_RX_VBUS2APB_DL3_RX_DIG_TBIT2 [21:20] = 2'b11, to enable the diagnostic high speed override MUXes in the analog.
      - PMA Register (data lanes): DPHY_RX_VBUS2APB_DL0_RX_DIG_TBIT30 - DPHY_RX_VBUS2APB_DL3_RX_DIG_TBIT30[1:0] = 2'b11, to enable the diagnostic low power and high speed override MUXes in the analog.
   c. Release the common from reset, and wait for DPHY_RX_VBUS2APB_ISO_PHY_ISO_CMN_CTRL[5] O_CMN_READY to be driven to 1'b1.
   d. Release the clock and data lanes from reset, and wait for DPHY_RX_VBUS2APB_ISO_PHY_ISO_CMN_CTRL[8] LANE_READY_CMN to be driven to 1'b1.
   e. Poll PMA Register (cmn): DPHY_RX_VBUS2APB_CMN0_CMN_DIG_TBIT56[20-11] I_CMN_RX_SSM_STATE until it is set to 9'b100000000.
   f. When running BIST at 1.5 Gbps, do the following, otherwise skip this if running BIST at 2.5 Gbps.

Copyright © 2024 Texas Instruments Incorporated

   i. Write PMA Register (cmn): DPHY_RX_VBUS2APB_CMN0_CMN_DIG_TBIT36[1] O_RX_TM_SEL_1P5G_M ODE= 1’b1.
   ii. Wait 5 uSec.
2. Run BIST operations.
   a. At this point, the BIST generator and checker registers can be written to the values required to generate the desired data patterns. One should see the BIST register descriptions for information about programming the various fields for creating different data patterns. If no registers are written, the BIST default pattern will be generated and checked.
   b. If running infinite BIST mode is desired, write the PMA Register (cmn): DPHY_RX_VBUS2APB_CMN0_CMN_DIG_TBIT24[15] BIST_INF_MODE = 1’b1, to enable infinite BIST mode.
3. Write PMA Register (data lanes): DPHY_RX_VBUS2APB_DL0_RX_DIG_TBIT27[0] = 1’b1, to enable the pattern checkers in the data lanes.
4. Write PMA Register (cmn): CMN_DIG_TBIT22[0] TM_BIST_EN = 1’b1, to enable the pattern generator in the common.
5. If infinite BIST mode was enabled, do the following.
   a. Allow the BIST to run as long as required.
   b. Write the PMA Register (cmn lanes): DPHY_RX_VBUS2APB_CMN0_CMN_DIG_TBIT24[15] BIST_INF_MODE = 1’b0, to disable infinite BIST mode.
6. Wait for a minimum of 100 nSec.
7. Poll PMA Register (cmn): DPHY_RX_VBUS2APB_CMN0_CMN_DIG_TBIT50[1] BIST_COMPLETE until it is set to 1’b1, indicating that the BIST pattern generation process is complete.
8. Wait 1 uSec.
9. Read and confirm the PMA Register (data lanes): DPHY_RX_VBUS2APB_DL0_RX_DIG_TBIT48 - DPHY_RX_VBUS2APB_DL3_RX_DIG_TBIT48[0] W_DRX_BIST_PASS = 1’b1, indicating the BIST checker has indicated a pass condition.
10. Read and confirm the PMA Register (data lanes): DPHY_RX_VBUS2APB_DL0_RX_DIG_TBIT48 - DPHY_RX_VBUS2APB_DL3_RX_DIG_TBIT48[1] R_PAT_CHE_SYNC = 1’b1, indicating the BIST checker has observed a marker symbol.
11. Read and confirm the PMA Register (data lanes): DPHY_RX_VBUS2APB_DL0_RX_DIG_TBIT48 - DPHY_RX_VBUS2APB_DL3_RX_DIG_TBIT48[2] W_BIST_ERROR = 1’b0, indicating the BIST checker has not detected any errors.
12. Read and confirm the PMA Register (data lanes): DPHY_RX_VBUS2APB_DL0_RX_DIG_TBIT47 - DPHY_RX_VBUS2APB_DL3_RX_DIG_TBIT47[15:0] W_PAT_CHE_PKT_COUNT = the generated number of packets (15 by default).
13. Read and confirm the PMA Register (data lanes): DPHY_RX_VBUS2APB_DL0_RX_DIG_TBIT47 - DPHY_RX_VBUS2APB_DL3_RX_DIG_TBIT47[31:16] W_PAT_CHE_ERROR_COUNT = 16’h0000, indicating the BIST checker error count is 0.
14. Clear the internal state of the BIST pattern checkers by doing the following:
   a. Read PMA Register (data lanes): DPHY_RX_VBUS2APB_DL0_RX_DIG_TBIT29 - DPHY_RX_VBUS2APB_DL3_RX_DIG_TBIT29[28] TM_CLEAR_BIST.
   b. Write PMA Register (data lanes): DPHY_RX_VBUS2APB_DL0_RX_DIG_TBIT29 - DPHY_RX_VBUS2APB_DL3_RX_DIG_TBIT29[28] TM_CLEAR_BIST = inverted value of what was read from this bit in the previous step.
15. Write PMA Register (data lanes): DPHY_RX_VBUS2APB_DL0_RX_DIG_TBIT27 - DPHY_RX_VBUS2APB_DL3_RX_DIG_TBIT27[0] TM_BIST_EN = 1’b0, to disable the pattern checkers in the data lanes.
16. Clear the internal state of the BIST pattern generator by doing the following:
   a. Write the PMA Register (cmn): DPHY_RX_VBUS2APB_CMN0_CMN_DIG_TBIT24[6] = 1’b1,
   b. Write the PMA Register (cmn): CMN_DIG_TBIT24[6] BIST_CLEAR = 1’b0.
17. Write the PMA Register (cmn): DPHY_RX_VBUS2APB_CMN0_CMN_DIG_TBIT22[0] BIST_CONTROLLER_EN = 1’b0, to disable the BIST pattern generator.
18. If desired, the run BIST operations can be repeated without needing to run set up the IP for BIST operations again.

Copyright © 2024 Texas Instruments Incorporated

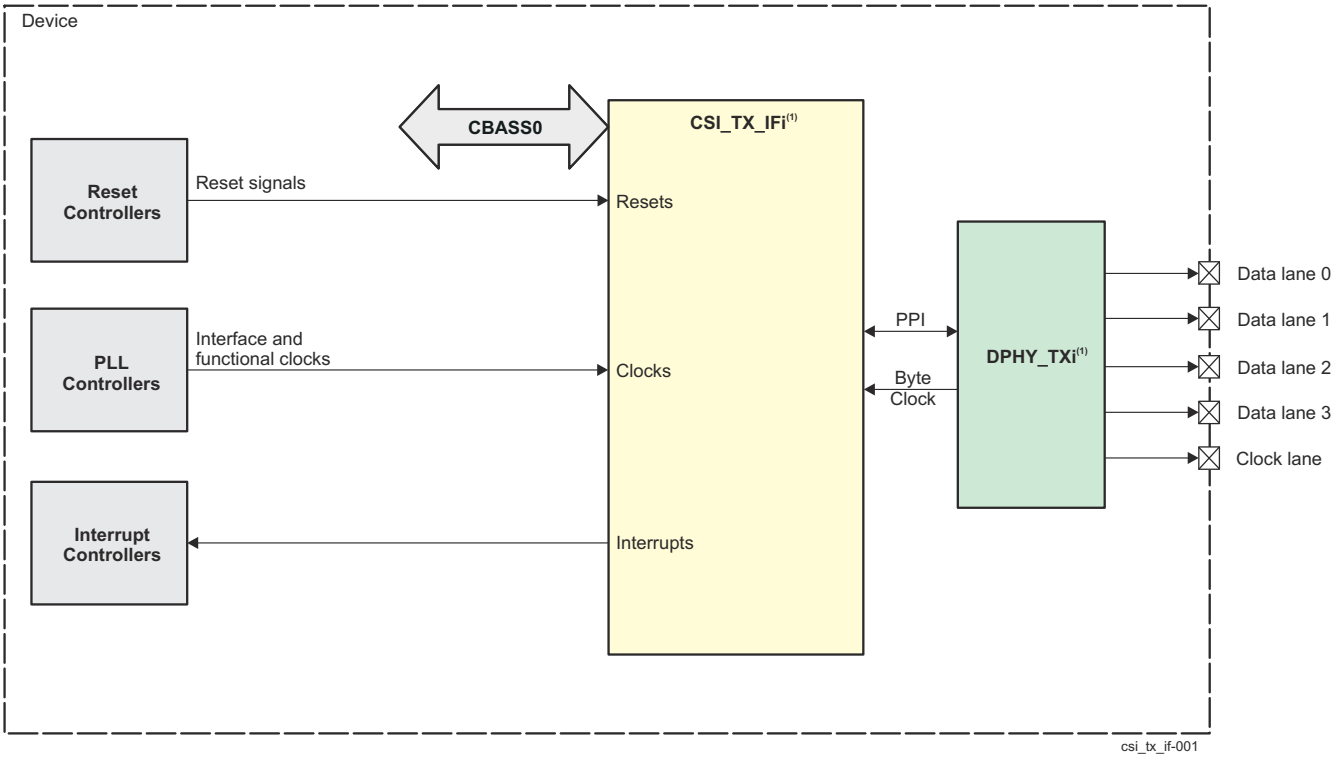
12.7.3 Camera Streaming Interface Transmitter (CSI_TX_IF)

The following sections describe the camera streaming transmitter interface (CSI_TX_IF) module in the device.

12.7.3.1 CSI_TX_IF Overview

The integration of the CSI_TX_IF module allows the device to stream out video data from memory, or retransmit from the CSI receivers as an optional loopback output for diagnostics, debug, and test purposes.

Figure 12-465 shows the CSI_TX_IF module overview.



A. i = 0 to 1

Figure 12-465. CSI_TX_IF Module Overview

12.7.3.1.1 CSI_TX_IF Ports

This section describes the CSI_TX_IF ports related to clocks, resets, and hardware requests.

Table 12-400. CSI_TX_IF Clocks and Resets

Clocks	
Module Clock Input	**Description**
CSI_TX_MAIN_CLK	Main functional(sometimes referred to as pixel clock) clock.
CSI_TX_VBUS_CLK	The VBUS clock runs at always half the speed of the CSI_TX_MAIN_CLK.
CSI_TX_ESC_CLK	20 MHz max clock input for low speed data transmission and some control signals.
DPHY_TXBYTECLKHS	The byte clock is the clock supplied by the DPHY_TX.
Resets	
Module Reset Input	**Description**
CSI_TX_RST	Asynchronous module global reset, driving all collateral asynchronous resets of the 4 clock domains to the low state.

Table 12-401. CSI_TX_IF Hardware Requests

Interrupt Requests		
Module Interrupt Signal	**Description**	**Type**

Copyright © 2024 Texas Instruments Incorporated

Table 12-401. CSI_TX_IF Hardware Requests (continued)

CSI_TX_IF_CSI_INTERRUPT_0	Global interrupt that various re-synchronized sources converge into interrupt generation.	Level
CSI_TX_IF_CSI_LEVEL_0	Error interrupt that is generated under the following conditions: • Retransmit error on Stream2 (packet error on the stream interface) • Retransmit error on Stream3 (packet error on the stream interface)	Level
CSI_TX_IF_CSI_FATAL_0	ASF port fatal interrupt. Level sensitive.	Level
CSI_TX_IF_CSI_NONFATAL_0	ASF port non-fatal interrupt. Level sensitive.	Level
CSI_TX_IF_CDNS_RAM_CORR_LEVEL_0	Interrupt on internal FIFO RAM	Level
CSI_TX_IF_CDNS_RAM_UNCORR_LEVEL_0	Interrupt on internal FIFO RAM	Level
CSI_TX_IF_CORR_LEVEL_0	Interrupt on internal FIFO RAM	Level
CSI_TX_IF_UNCORR_LEVEL_0	Interrupt on internal FIFO RAM	Level

12.7.3.2 CSI_TX_IF Features

The CSI_TX_IF module supports the following features:

- Compliant to MIPI CSI v1.3, MIPI CSI v2.1, and MIPI D-PHY v2.1
- Data rate up to 2.5 Gbps per lane (wire rate)
- Supports 1, 2, 3, or 4 Data Lane connection to DPHY_TX
- Programmable formats including YUV422, RGB, Raw, and User Defined (over 25 different formats supported); Limitations apply depending on the stream used
- 16 virtual channel support
- Configurable input streams supported:
  - **Stream0**: DMA interface through a 128-bit PSI_L connection for transfers from memory **OR** CSI_RX retransmit:
    - 128bit wide pixel data with bursting
    - ByteValid per byte in Last Data Phase (LDP)
    - 32 thread IDs supported (virtual channel & data type combinations); Flexible number of threads (32 Max).
    - Unpacking PSI_L data and converting to CSI video format
    - Internal FF based FIFO and external RAM based buffer
  - **Stream1**: Color bar video data generator
    - 2 pixel wide
    - YUV422 8-bit format support only
    - Configurable frame/line size via registers
    - 1 programmable virtual channel
    - Internal FF based FIFO; No external buffer.
- Functional and data path error interrupts
- ECC support on external RAMs

Unsupported Features:

- See the Module Integration section of this document for a list of module features not supported by the integration on this Device.

12.7.3.2.1 CSI_TX_IF Legacy Compatibility

The following lists changes that must be considered compared to previous generation CSI_TX_IF:

- YUV420 is not supported, YUV420 configuration registers/bitfields are not used
- Word count must be programmed in the CSI_TX_IF, not in the TX Core.
- CSI_TX_IF compatibility mode must be used in Stream0 when retransmitting from CSI_RX_IF

Copyright © 2024 Texas Instruments Incorporated

12.7.3.3 CSI_TX_IF Environment

The CSI_TX_IF has no dedicated pins. At the device level, the CSI_TX_IF video output goes through the shared(with DSS_DSI) DPHY_TX. See chapter *Shared MIPI D-PHY Transmitter (DPHY_TX)*.

12.7.3.4 CSI_TX_IF Functional Description

12.7.3.4.1 CSI_TX_IF Block Diagram

Figure 12-466 depicts a simplified internal block diagram of the CSI_TX_IF and its surroundings.

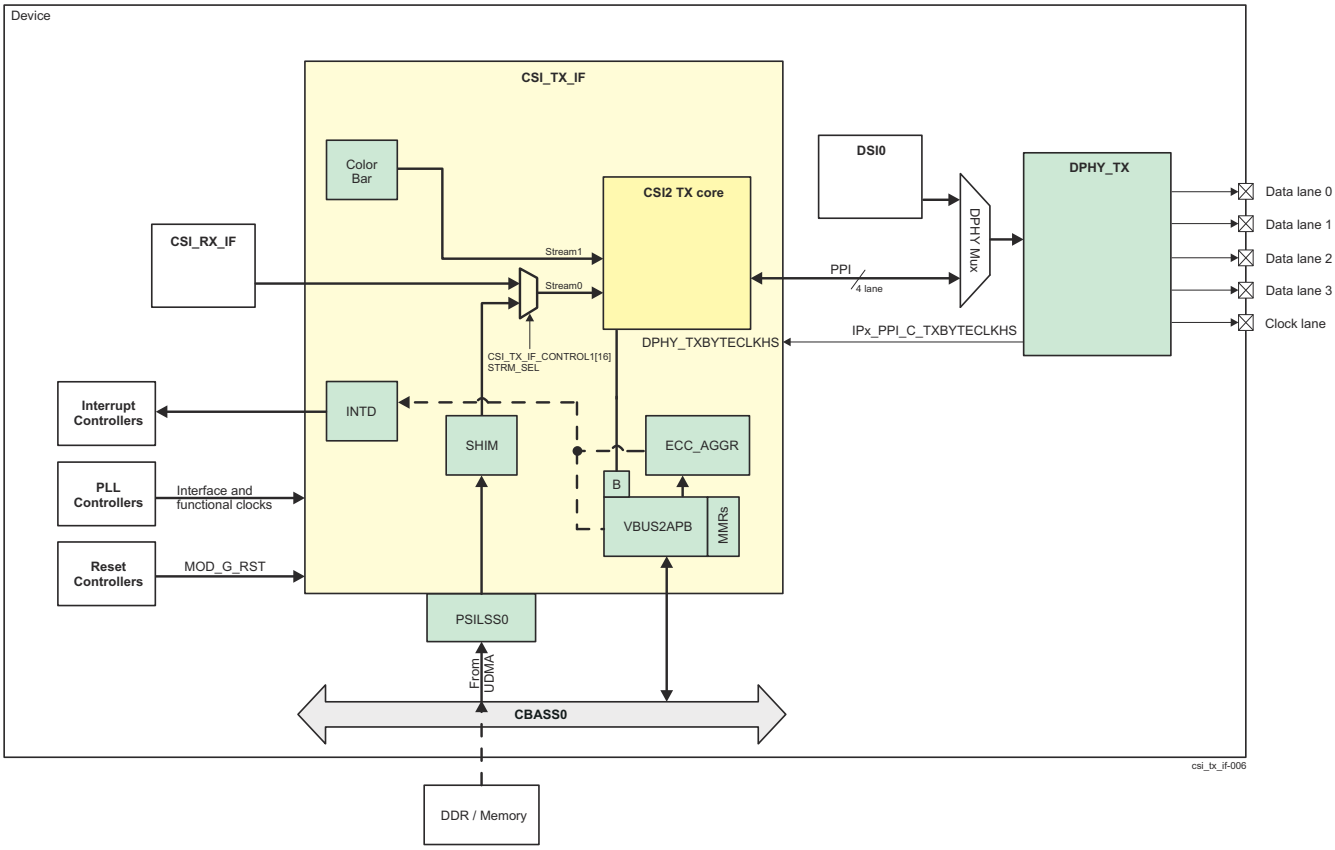


Figure 12-466. CSI_TX_IF Block Diagram

The CSI2 core streams 4 channels of data to the several export paths.

Stream0 (from PSI_L) goes through the PSI_L that converts 128(4x32) bits pixel data and converts it to single 32-bit CSI streaming protocol of pixel data. Each 128 bit set of PSI_L data is split into N M-bit data transfers based on the SIZE_CFG bitfield setting in the CSI_TX_IF_DMACNTX_j register for a given thread. See memory organization details in *CSI_TX_IF Data Memory Organization Details*. Via register mapping in the CSI_TX_IF_DMACNTX_j register the PSI_L thread ID is mapped to a virtual channel and a data type. The CSI_TX_IF_DMACNTX_j register is the thread filter. There are 32 dma context registers. Figure 12-467 shows the stream to PSI_L thread mapping.

Copyright © 2024 Texas Instruments Incorporated

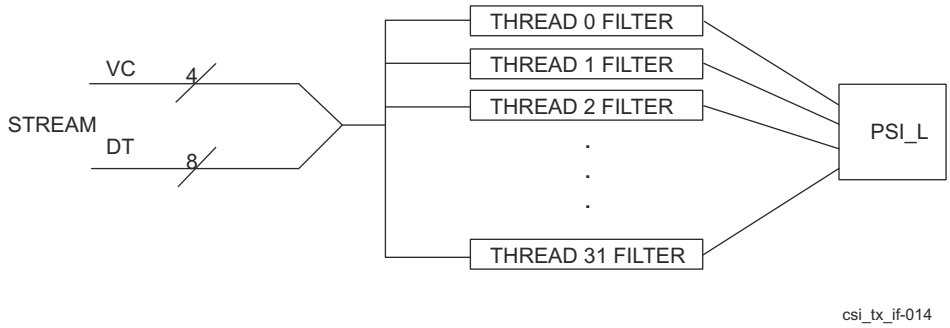


Figure 12-467. CSI_TX_IF PSI_L thread mapping

Software must take the following notes in consideration

- Interleaved data formats on single virtual channel and frame are supported.
- All PSI_L threads that open a frame must be closed before a new PSI_L thread can start a new frame. Until all channels close the frame other threads will be stalled.
- DATA is assumed in a specific organization for CSI_TX_IF to export data correctly. See memory organization details in *CSI_TX_IF Data Memory Organization Details*.
- Threads can only switch after full lines are received. PSI_L pushback will occur on all other threads.
- Thread priority is 0 - 31 and after a line is received will be re-arbitrated in that order for the next thread to send.

Registers CSI_TX_IF_L2L_DELAY_j control the line start to next line start. Also register CSI_TX_IF_L2L_DELAY_j control the last line of frame to next start of frame first line start of next frame. Buffering of PSI_L data is 2k×128 RAM.

Stream0 (from CSI_RX_IF) is a loopback stream from the CSI receiver muxed with PSI_L data stream.

Stream1 streams data from the Color Bar generator. It supports YUV422 8bit format only. CSI_TX_IF_COLOR_PARAM register is used to control height and width information. Width is divided by 8 for each color section. Colors are defined below. Transfers are done on the Stream1 interface in 32-bit packed 8-bit pixel format(required programming in CSI_TX_IF core). Virtual channel and data type is configurable in CSI_TX_IF_COLOR_CNTL register. Data selection inside CSI_TX_IF controller maps data type information. CSI_TX_IF_COLOR_START_DELAY register can configure delays from enabled to start. CSI_TX_IF_COLOR_LINE_DELAY, and CSI_TX_IF_COLOR_FRAME_DELAY register can delay last line to first line of new frame. Once enabled, it will continuously send out frames until disabled. Once disabled, it will stop at end of current frame. Table 12-402 shows the Color Bar format.

Table 12-402. CSI_TX_IF Color Bar format details

Color	Y	CB	CR
White	235	128	128
Yellow	162	44	142
Cyan	131	156	44
Green	112	72	58
Magenta	84	184	198
Red	65	100	212
Blue	35	212	114
Black	16	128	128

Copyright © 2024 Texas Instruments Incorporated

Figure 12-468. CSI_TX_IF PSI_L Color Bar sample output

Note

It is a requirement that only PSI_L/DMA, color bar, or CSI_RX_IF loopback streams are not active at the same point in time. The intended use model is not to have multiple combinations active at any given time. The main issue with this is that RAMs are not sized for concurrent operation and will likely overflow resulting in data loss.

12.7.3.4.2 CSI_TX_IF Hardware and Software Reset

An active low asynchronous hardware reset is provided to CSI_TX_IF by device LPSC. It is internally re-synchronized to the functional clock domain.

A software reset is triggered by configuring the CSI_TX_IF_TX_CONF[1] SOFT_RESET_REQUEST bit-field for protocol reset and/or module reset.

12.7.3.4.3 CSI_TX_IF Clock Configuration

There are four clock domain in the CSI_TX_IF.

1. The CSI_TX_MAIN_CLK(or also referred to as pixel clock) runs most of the logic. It needs to be same frequency as CSI_RX_MAIN_CLK main clock (500MHz). When CSI_TX_MAIN_CLK is operating lower than 312.5MHz, then the clock is essentially limiting the clock rate of the DPHY_TX. Said another way, CSI_TX_MAIN_CLK must be at least the DPHY_TXBYTECLKHS rate else FIFOs will overflow, crashing the module.
2. The CSI_TX_VBUS_CLK is the interface configuration clock that runs at half the speed of the CSI_TX_MAIN_CLK (250MHz).
3. The DPHY_TXBYTECLKHS is the clock supplied by the DPHY_TX PLL and is divided down to byte clock. The DPHY_TX is designed for max of 10gbps. This translates to a max byte clock of 312.5MHz. The clock is inactive when DPHY_TX is not in HS operation.
4. The CSI_TX_ESC_CLK escape clock runs at 20MHz. CSI_TX_ESC_CLK is rarely used by the CSI_TX_IF, usually is only to clock in some low speed control signals from DPHY_TX. The ESC interface is a low speed DPHY common link to the camera/sensor.

Table 12-403 shows the CSI_TX_IF and DPHY_TX inter-clock dependencies.

Table 12-403. CSI_TX_IF Inter-clock Dependencies

	CSI_TX_MAIN_CLK	CSI_TX_VBUS_CLK	DPHY_TXBYTECLKHS	CSI_TX_ESC_CLK
Min freq	DPHY_TXBYTECLKHS freq	CSI_TX_MAIN_CLK / 2 freq	N/A	N/A
Max freq	500MHz	CSI_TX_MAIN_CLK / 2 freq	312.5MHz	20MHz

Copyright © 2024 Texas Instruments Incorporated

12.7.3.4.4 CSI_TX_IF Interrupt Events

12.7.3.4.4.1 CSI_TX_IF Interrupt Events

This section describes the register configuration of the interrupt events that can trigger the several CSI_TX_IF interrupt signals. For detailed description and mapping of the interrupt to the device interrupt processors see *CSI_TX_IF Integration*.

The interrupts are generally handled within the INTD module of the CSI_TX_IF, although there are several interrupt registers in the ECC_AGGR for ECC errors and in the VBUS2APB for stream monitoring errors/flags.

Table 12-404 lists the event generation and corresponding registers of the CSI_TX_IF controller.

Table 12-404. CSI_TX_IF Interrupt Events Cross Table

Event	Mask Register	Status Register	Description
CSI_TX_IF_MAIN_0_CSI_INTERRUPT	CSI_TX_IF_IRQ_MASK CSI_TX_IF_DPHY_IRQ_MASK	CSI_TX_IF_IRQ CSI_TX_IF_DPHY_IRQ_MASK	Global interrupt that various re-synchronized sources converge into interrupt generation. Read the status register bitfields to trace the source of the event.
CSI_TX_IF_MAIN_0_CSI_LEVEL	CSI_TX_IF_IRQ_MASK CSI_TX_IF_DPHY_IRQ_MASK	CSI_TX_IF_IRQ CSI_TX_IF_DPHY_IRQ_MASK	Error interrupt that is generated under the following conditions: • Retransmit error • Unselected stream error. An unselected DMA path or CSI_RX_IF loopback path is sending data when not selected Read the status register bitfields to trace the source of the event.
CSI_TX_IF_MAIN_0_CORR_LEVEL	CSI_TX_IF_ASF_INT_MASK	CSI_TX_IF_ASF_SRAM_CORR_FAULT_STATUS	Interrupt on internal FIFO RAM. Read the status register bitfields to trace the source of the event.
CSI_TX_IF_MAIN_0_UNCORR_LEVEL	CSI_TX_IF_ASF_INT_MASK	CSI_TX_IF_ASF_SRAM_UNCORR_FAULT_STATUS	Interrupt on internal FIFO RAM. Read the status register bitfields to trace the source of the event.
CSI_TX_IF_MAIN_0_CSI_FATAL	CSI_TX_IF_ASF_INT_MASK	CSI_TX_IF_ASF_INT_STATUS	ASF port fatal interrupt. Level sensitive. Set CSI_TX_IF_ASF_FATAL_NONFATAL_SELECT for whether fatal or non-fatal ASF interrupt is triggered. If any of the CSI_TX_IF_ASF_INT_STATUS bit is set, the CSI_RX_CSI_FATAL or CSI_RX_CSI_NONFATAL event signal is asserted.
CSI_RX_CSI_NONFATAL	CSI_TX_IF_ASF_INT_MASK	CSI_TX_IF_ASF_INT_STATUS	ASF port fatal interrupt. Level sensitive. Set CSI_TX_IF_ASF_FATAL_NONFATAL_SELECT for whether fatal or non-fatal ASF interrupt is triggered. If any of the CSI_TX_IF_ASF_INT_STATUS bit is set, the CSI_RX_CSI_FATAL or CSI_RX_CSI_NONFATAL event signal is asserted.

12.7.3.4.5 CSI_TX_IF Data Memory Organization Details

Figure 12-469 shows the YUV422-8 data organization in memory.

Copyright © 2024 Texas Instruments Incorporated

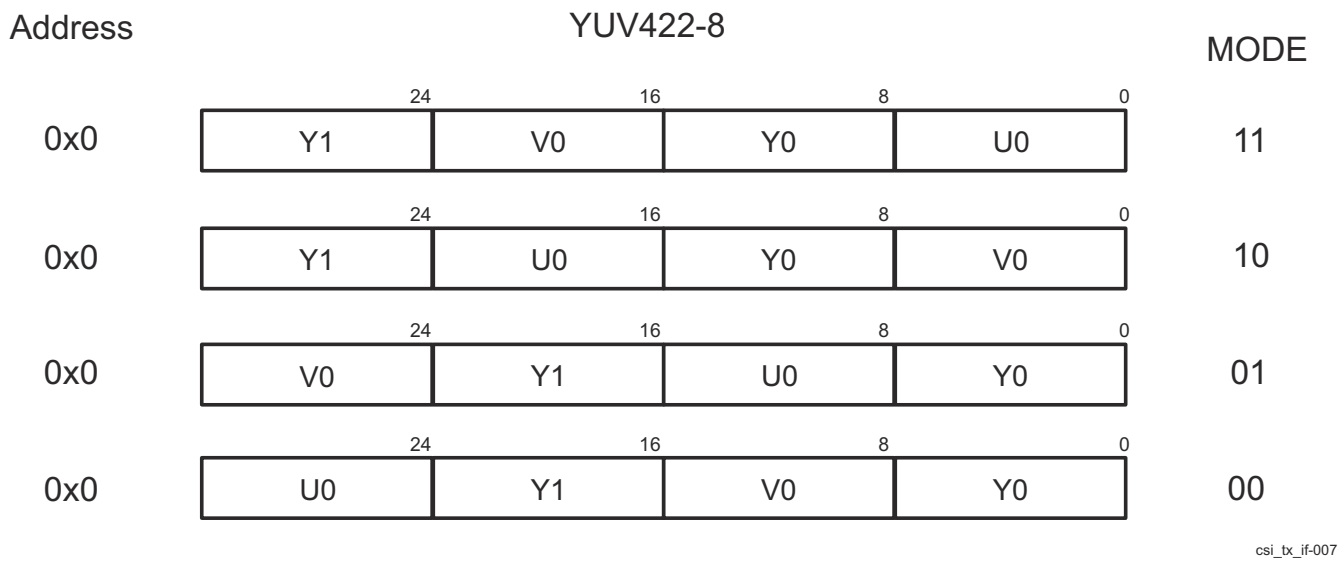


Figure 12-469. CSI_TX_IF YUV422-8 Memory Data Organization

Figure 12-470 shows the YUV422-10 data organization in memory.

Copyright © 2024 Texas Instruments Incorporated

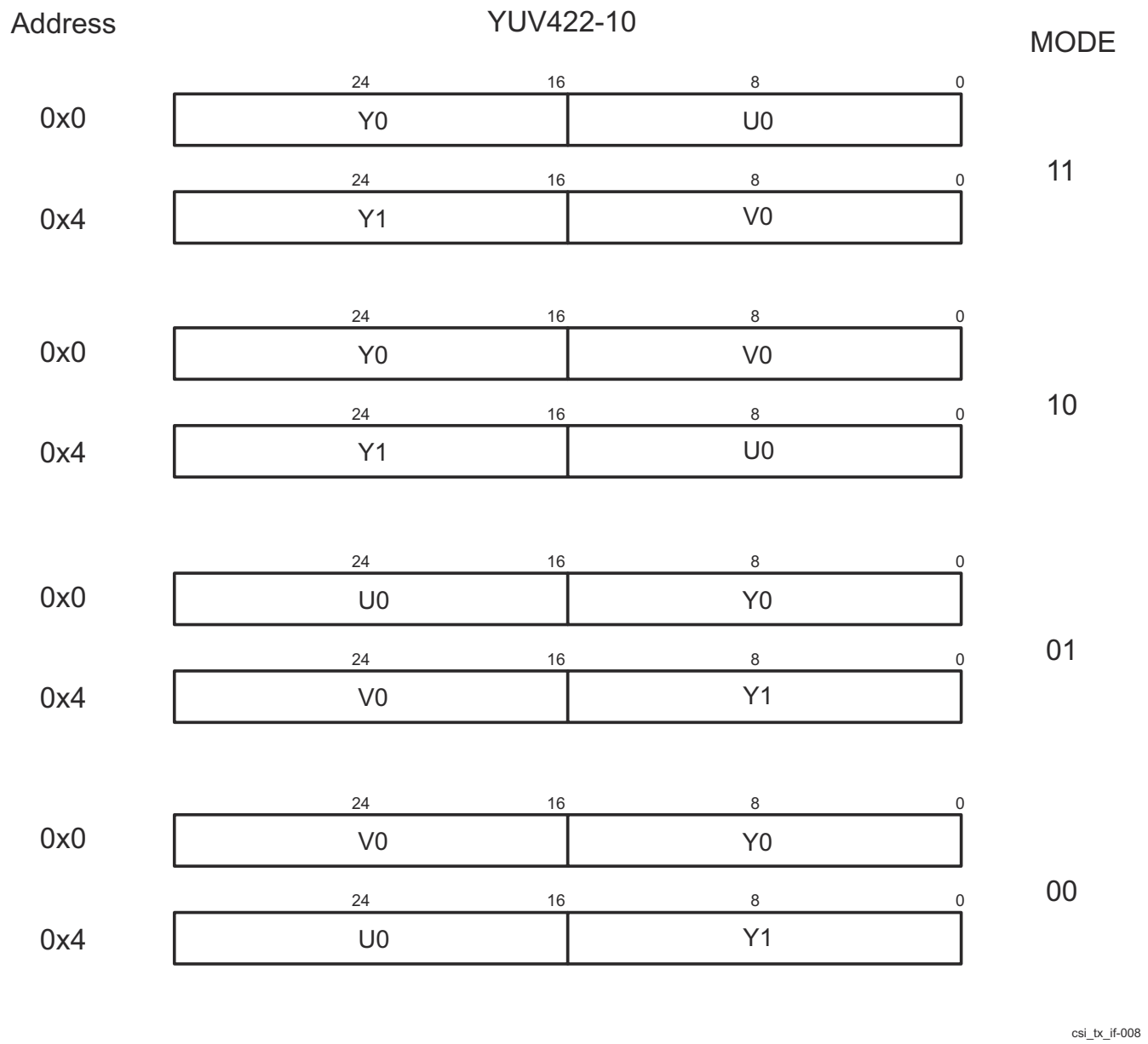


Figure 12-470. CSI_TX_IF YUV422-10 memory data organization

Figure 12-471 shows the RAW data organization in memory. User defined data types behave the same as RAW data types.

Copyright © 2024 Texas Instruments Incorporated

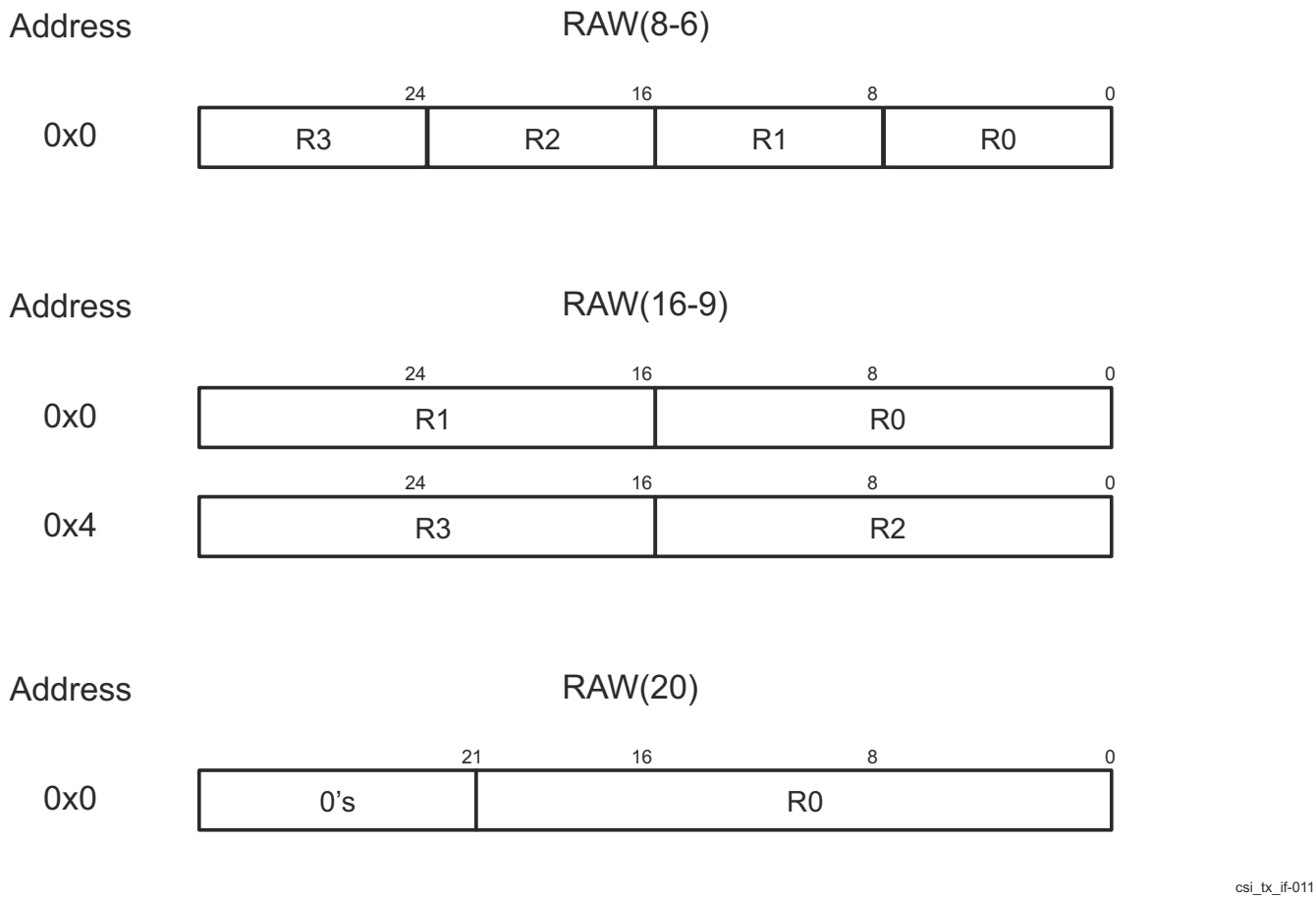


Figure 12-471. CSI_TX_IF RAW (UNPACKED) memory data organization

Figure 12-472 shows the RGB data organization in memory.

Copyright © 2024 Texas Instruments Incorporated

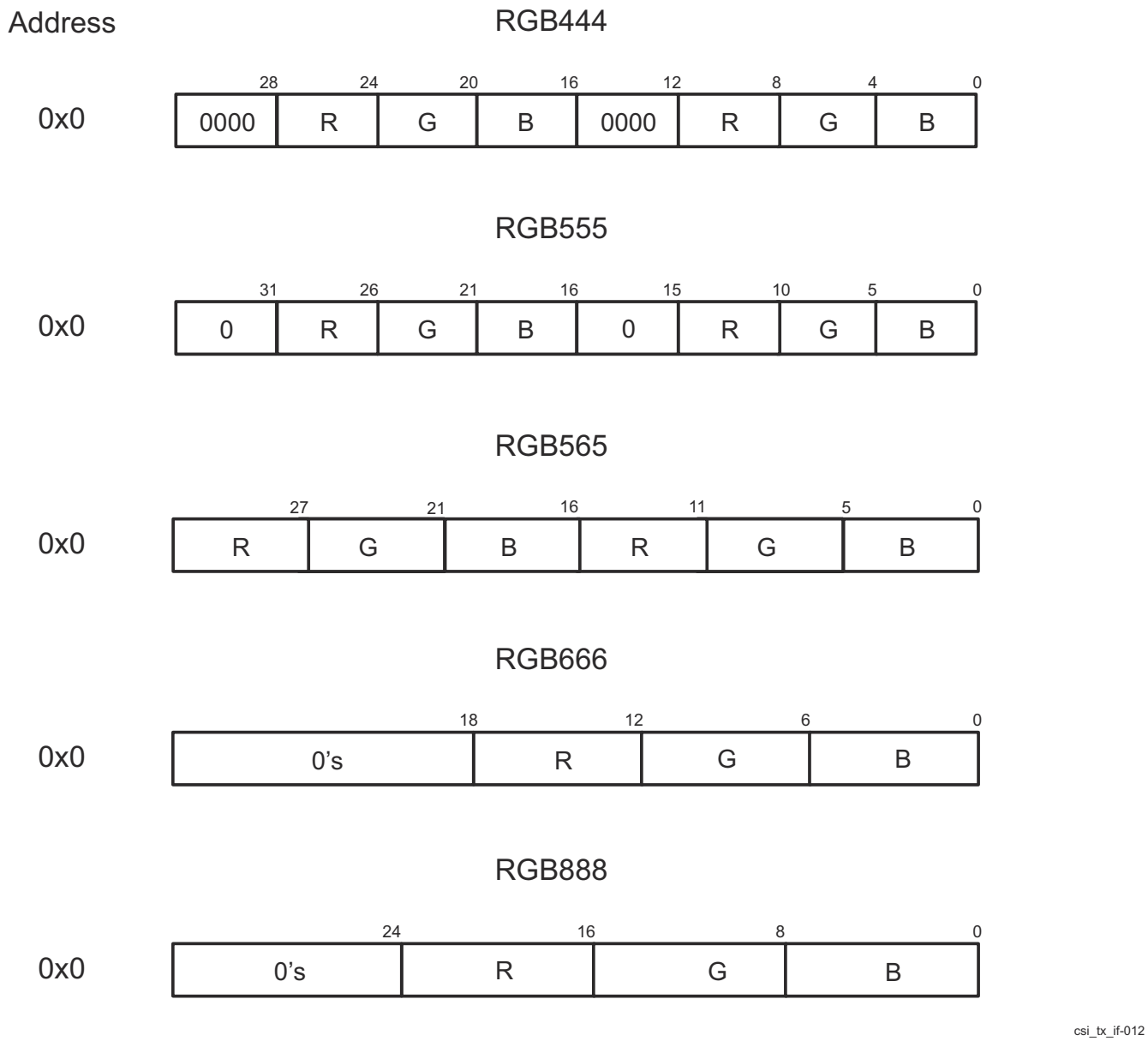
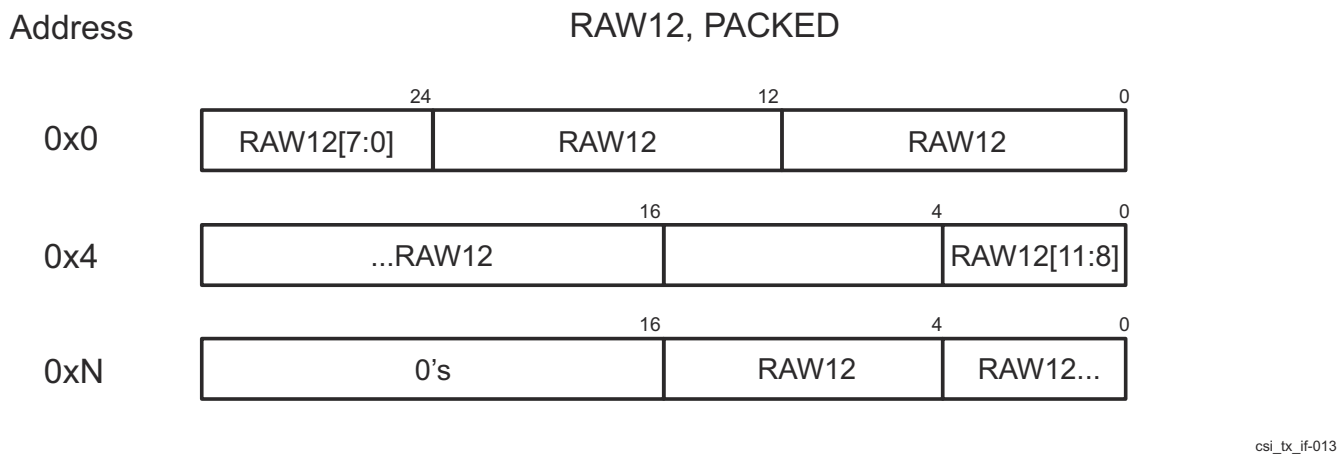


Figure 12-472. CSI_TX_IF RGB memory data organization

Figure 12-473 shows the RAW12 (PACKED) data organization in memory.

Copyright © 2024 Texas Instruments Incorporated



A. Where data does not fill out to a byte boundary, it is zero extended.

Figure 12-473. CSI_TX_IF RAW12 (PACKED) memory data organization

12.7.3.4.6 CSI_TX_IF PSI_L (DMA) Interface

The PSI_L (DMA) interfaces is common for both CSI_RX_IF and CSI_TX_IF, see chapter *Camera Streaming Interface Receiver (CSI_RX_IF) and MIPI DPHY Receiver (DPHY_RX)*, subsection *CSI_RX_IF PSI_L (DMA) Interface*.

12.7.3.4.7 CSI_TX_IF ECC Protection Support

The CSI_TX_IF has two ECC aggregators, one for byte clock (DPHY_TXBYTECLKHS) and the other one for all other clocks. The ECC is a mechanism for providing increased system reliability (via reduction of memory soft errors) by allowing single bit errors to be detected and corrected and double bit errors to be detected.

The ECC protection on the CSI_TX_IF RAM provides Single Error Correction and Double Error Detection (SEC/DED). This logic detects and corrects a single bit error (1 bit error per ECC word or per ECC data segment). For memories that contain critical and/or persistent data, automatic (immediate or delayed) write-back of the corrected data to the corresponding memory address is supported. In addition, the ECC also supports multiple options for partial word writes, such as read-modify-write or multiple ECC code segments per word.

The ECC protection also provides Double Error Detection (DED). This logic only detects (does not correct) double errors (2 bit errors per ECC word or per ECC data segment).

The ECC aggregators on the CSI_TX_IF subsystem level consolidates the ECC configuration and status bits for all the ECC supported memories in the subsystem. They provide a single EOI-handshake based interrupt to the interrupt processors (for both single and double error detections) and a standard 32-bit VBUS interface for configuring and querying the ECC register set, see *CSI_RX_IF_ECC Registers*. For complete details on the features and functions of the ECC aggregator, see *ECC Aggregator*.

12.7.3.5 CSI_TX_IF Programming Guide

Note

Software must keep the CSI_TX_IF_CONTROL1[0] PIXEL_RESET register in the asserted state in the control register until all CSI_TX_IF controller registers are programmed. Only after all CSI_TX_IF controller programming can software de-assert CSI_TX_IF_CONTROL1 register. If software has to reprogram any CSI_TX_IF controller registers it MUST have the control CSI_TX_IF_CONTROL1 register reset asserted.

It is a restriction that LS/LE detection is disabled.

12.7.3.5.1 CSI_TX_IF Programming (Configuration Mode)

Although it is always possible to read/write all control registers, it strongly recommended writing to the control registers only while configuration mode is active. To enter configuration mode, the CSI_TX_IF_TX_CONF[2]

Copyright © 2024 Texas Instruments Incorporated

CONFIGURATION_REQUEST bit must be set. CSI_TX_IF enters configuration mode when transmission of any ongoing frame has ended and the configuration request bit is set. Configuration mode is the default state after reset. While configuration mode is active, the CSI_TX_IF_STATUS[2] CONFIGURATION_ACTIVE bit is set. Traffic at the pixel interface is ignored during configuration mode, and it is safe to write to control registers and change configuration.

To exit configuration mode, the CONFIGURATION_REQUEST bit in the CSI_TX_IF_TX_CONF register must be cleared. The CSI_TX_IF then starts normal operation, with first new frame incoming after configuration mode exit.

During configuration mode, all interrupts are disabled.

Note

A mandatory assertion of soft reset occurs following the de-assertion of CONFIGURATION_REQUEST. This ensures the controller is in a known state and all the parameters set in the configuration registers have been loaded correctly into the internal state. The DPHY_TXBYTECLKHS must be active before the configuration request is completed.

12.7.3.5.2 CSI_TX_IF System Initialization Programming

The power up/reset release of the system will clear all the register values to their reset conditions. Software must perform programming of the virtual channel and data type registers to match the system operation if the reset conditions do not match the required values. Software must ensure that all the data type select registers are programmed with valid values for the pixel_dt_sel_if parameter used.

The CSI_TX_IF registers for all the DPHY_TX related control and delays must be set before starting the system. The DPHY_TX delays must be calculated using the system clock periods associated with CSI_TX_ESC_CLK and DPHY_TXBYTECLKHS. The wait burst time must be calculated using the relevant DPHY_TX data sheet to ensure that all the lanes begin new requests at the same point.

Software must configure the FIFO fill level control registers, if required to suit clock ratios used and the desired payload control configurations. When all registers are programmed the configuration can be made active. The pixel streams can then start requests.

Note

The CSI_TX_IF is configured at design time, such that the reset values adopt a generic preferred Power_On state. This reduces the programming steps required by the system for the general use case scenario.

12.7.3.5.3 CSI_TX_IF Lane Control Programming

The CSI_TX_IF links to the DPHY_TX clock and data lanes. After power up, software must identify that the DPHY_TX is powered on and the DPHY_TX PLL has locked. The DPHY_TX controls for swapping the DP/DN control can be modified using the CSI_TX_IF_DPHY_CFG1 register if required.

Software must then program the enable for the clock lane and each data lane that is required. The DPHY_TX lane signals can be checked using the CSI_TX_IF_DPHY_STATUS register to identify that the lane is in STOPSTATE and is ready for High Speed transmission. This register can also be used to identify when ULPS is active.

12.7.3.5.4 CSI_TX_IF Virtual Channel and Data Type Management

The following section provides information on how the configuration of virtual channels and data types may be used in conjunction with one or more stream inputs to meet the system requirements.

12.7.3.5.4.1 CSI_TX_IF Data Type Interleaving

The CSI_TX_IF can support interleaving of data types by configuring the pixel_dt_sel_if inputs at the pixel interface to select the required data type and pixel count values.

Copyright © 2024 Texas Instruments Incorporated

12.7.3.5.4.2 CSI_TX_IF Data Type Interleaving with Multiple Interfaces

The CSI_TX_IF will perform data type interleaving with one, two, or more pixel interfaces. In this scenario, Stream0 will be the master and one or more streams may be ganged together as slaves to the master stream. Therefore, for all the ganged streams, the virtual channel signals must be configured to the same value and the pixel streams must have the same active frame valid cycle (that is the virtual_channel_if* and frame_valid_if* must be wired in parallel). The STREAM_IF_*_SLAVE_MODE bit in the STREAM_IF_*_CFG registers must also be set for all slaved streams. This ensures that only one frame start and frame end are generated for all the ganged streams. Note that the pixel 0 interface is always the master and slave bits exist for the other streams, in the control registers, to determine if they are to operate in master or slave mode.

In the case where streams are ganged together as master and slave(s), all ganged interfaces must have the same virtual channel value, but the pixel_dt_sel_if inputs must be set to different values. Again, each of the interfaces can have the pixel_dt_sel_if input change on a line by line basis, so that interleaving is done with all the ganged streams.

Note

Each stream must have exclusive use of the data types allocated to it.

Data type interleaving is only done when the frame is valid (CSI_TX_IF_DEBUG_PROT*_FSM[11] FRAME_VALID_IF* bitfield input is asserted). The virtual channel will be sampled on the first cycle after the rising edge of the FRAME_VALID_IF input and this virtual channel will be used for all the packets originating from that stream during that frame. The FRAME_VALID_IF input must be the same on all ganged stream interfaces.

12.7.3.5.4.3 CSI_TX_IF Virtual Channel Interleaving

The CSI_TX_IF can support interleaving of virtual channels by using two or more pixel interfaces and configuring the value of the virtual channels signal differently on each interface. The data streams can then be controlled using the pixel_dt_sel_if inputs to select the data type for the lines. In this scenario, each pixel interface must have exclusive use of the virtual channel. Each stream will generate its own frame start and frame end synchronisation packets. In this case, all the stream protocol modules employing virtual channel interleaving must be configured as masters by clearing the STREAM_IF_*_SLAVE_MODE bit in the STREAM_IF_*_CFG registers.

Note that the virtual channel control input is sampled at the rising edge of the frame_valid_if input, and thereafter remains fixed for that frame, therefore the virtual channel cannot be switched on a line-by-line basis for any single pixel interface. While it is possible to switch the Virtual Channel control input from frame to frame, the timing-critical switching required here would indicate a requirement for hardware rather than software control.

12.7.3.5.4.4 CSI_TX_IF Virtual Channel and Data Type Interleaving

The CSI_TX_IF may be configured to perform a combination of the scenarios outlined in the previous sections if sufficient streams are available. One or more streams may be ganged with Stream0 to perform data type interleaving on the same virtual channel, whilst the remaining streams may be left autonomous to perform virtual channel interleaving.

In this case, the same rules governing the use of the virtual channels and data types must be observed. In this scenario, the virtual channel signals will be configured to the single value required for the ganged group (as they need to be controlled within the same frame valid active signal) and different values for those left to be autonomous (as they may generate their own frame start and frame end events).The slave stream for each virtual channel can have a different data type selected, and this may change on a line-by-line (or packet-by-packet) basis.

12.7.3.5.5 CSI_TX_IF Line Control

The line control block uses the DPHY_TXBYTECLKHS from the DPHY_TX PPI interface to take the line data from the packet interface FIFO and transfer the bytes to the active lanes of the DPHY_TX. The block detects when the DPHY_TX is ready to transmit high speed data and the pixel stream has packets available.

Copyright © 2024 Texas Instruments Incorporated

The line control block performs pixel stream arbitration using the line control arbitration block when more than one pixel stream is generated in the configuration. The block performs all the clock and data lane control required to activate the high-speed transmission and return the lane to ULPS state as required.

12.7.3.5.5.1 CSI_TX_IF Line Control Arbitration

The CSI_TX_IF controller uses a simple arbitration scheme for configurations with more than one pixel interface.

The arbitration has 2 levels; the first is a round-robin scheme and if no streams are granted by this a priority based scheme is used. The arbitration will select the first stream making a request if there is only one stream waiting or the first in the round robin after a stream has completed transmission. This means that there may be one cycle where the round-robin selector moves across a stream position that is not requesting to transmit.

The re-arbitration boundary is between packets except where line sync is used. In this case the line start packet, long packet and line end are grouped together.

The selected stream will pass the line data from the stream FIFO to the line control block for distribution to the active PPI lanes of the DPHY_TX.

12.7.3.5.6 CSI_TX_IF Lane Manager FSM

Data distribution to the available lanes is controlled via the lane manager FSM.

The state descriptions can be found in Table 12-405.

Table 12-405. CSI_TX_IF Lane Manager FSM State Description

State (line_fsm_st_r)	Description
LINE_FSM_IDLE	Wait for new transmission to be ready. When start_hs_transmission_c is asserted then go to the transmission state per enabled lanes (lanes_enable_r)
LINE_FSM_BURST_1L	Transmit burst data over lane 0 until end of the burst. When end then go to the LINE_FSM_BURST_END state if EPD is not enabled, otherwise go to the LINE_FSM_LRTE_EPD_SPACER state.
LINE_FSM_BURST_2L	Transmit burst data over lanes 0 and 1 until end of the burst. When end then go to the LINE_FSM_BURST_END state if EPD is not enabled, otherwise go to the LINE_FSM_LRTE_EPD_SPACER state.
LINE_FSM_BURST_4L	Transmit burst data over all lanes until end of the burst. When end then go to the LINE_FSM_BURST_END state if EPD is not enabled, otherwise go to the LINE_FSM_LRTE_EPD_SPACER state.
LINE_FSM_LRTE_EPD_SPACER	Insert spacer packets across each active lane. When finished, if EPD Option 1 then go to the LINE_FSM_LRTE_EPD_PDQ state, otherwise go to the LINE_FSM_IDLE state.
LINE_FSM_LRTE_EPD_PDQ	Tell D-PHY to initiate HS-IDLE state and insert PDQ.
LINE_FSM_BURST_END	Wait for being ready for new burst transmission. In this state counter counts until WAIT_BURST_TIME value in the register is reached.

12.7.3.5.7 CSI_TX_IF Data Lane Control FSM

Data lanes control uses a simple FSM to sequence entry and exit of Ultra Low Power mode. This FSM observes inputs from the control registers and based on these inputs it generates PPI outputs for ULP mode.

Table 12-406 shows the state descriptions.

Table 12-406. CSI_TX_IF Data Lanes Control FSM State Description

State (ulps_data_fsm_st_r)	Description
DATA_LN_IDLE	Idle state. In this state HS transmission can be issued if the Clock Lane is in HS mode.
DATA_LN_ULPS_REQ	Request to enter ULP state. Waiting for activation ULP confirmed by the ppi_ulps_active_not_dl input.
DATA_LN_ULPS_ACTIVE	ULPS for Data Lanes is active. Waiting for the ulps_req_sync signal being low to exit ULPS.
DATA_LN_ULPS_EXIT	ULPS exiting, waiting for the time defined in the CSI_TX_IF_DPHY_ULPS_WAKEUP register.

Copyright © 2024 Texas Instruments Incorporated

12.7.3.5.8 CSI_TX_IF Application Examples

This section details specific steps on how to program the CSI_TX_IF.

12.7.3.5.8.1 CSI_TX_IF D-PHY Control and Configuration

The DPHY_TX will use a PLL to provide the bit clock for transmission. The CSI_TX_IF controller uses the associated byte clock as a primary clock source and this must be active and stable before the CSI_TX_IF is ready to transmit high speed data.

Software must perform the programming of the PLL dividers and monitor the lock status. Software must also ensure the correct sequence of programming the PLL, DPHY_TX and CSI_TX_IF to guarantee the clocks are active and resets are released, following the start-up sequence described in DPHY_TX chapter (See chapter *Shared MIPI D-PHY Transmitter (DPHY_TX)*).

12.7.3.5.8.2 CSI_TX_IF Clock and Data Lane Enable

The DPHY_TX lanes are controlled using the CSI_TX_IF_DPHY_CFG register. The clock enable will be asserted once the DPHY_TX is configured, PLL is programmed and locked and the lanes are enabled. Enabling the clock lane will then make the TX clock module stay in stop state with the DP/DN pads in LP11. The TX clock high speed ready state will be detected once the TX clock lane has completed the preamble and begun to drive the high speed clock.

The configured number of data lanes will be enabled at the same time as enabling the TX clock. The data lane will remain in LP until the high-speed clock is stable and the CSI_TX_IF makes the request for the data lane transitions from stop state to active (LP11-LP01-LP00).

12.7.3.5.8.3 CSI_TX_IF DP/DN Signal Swap

The DPHY_TX may have the capability to swap the data pin polarity or invert the clock lane polarity - the CSI_TX_IF_DPHY_CFG1 register is available within the controller for driving these pad level controls (using the DPHY_DIFF_INVERT_* fields). The DPHY_TX will identify any contention issues with the transmitter and flag this using the contention signals and error interrupt - this may be used by software to determine if the data pin or clock lane polarity switching is required or not (by optimizing to the minimum, preferably zero, rate of occurrence of contention issues).

Int the DPHY_TX, the IO pads for the clock and data lanes support swapping of the polarity of the signals DP-DN. All dphy_differential_invert_* outputs should be treated as asynchronous. Note that, if these signals need to be asserted, they should be asserted before releasing the DPHY_TX reset.

Each lane can be controlled independently with the CSI_TX_IF_DPHY_CFG and CSI_TX_IF_DPHY_CFG1 registers.

12.7.3.5.9 CSI_TX_IF DPHY_TX Status

The status of clock and data lanes is available in the read only CSI_TX_IF_DPHY_STATUS register. The clock lane fields show the condition of the ppi_tx_ulps_active_not_cl and ppi_stopstate_cl PPI inputs. The data lane fields show the ppi_ulps_active_not_dl* and ppi_stopstate_dl* PPI inputs.

All the status signals are based on the active enabled lane configuration.

The DPHY_TX can signal error conditions for each lane for contention in low power transitions and signal errors at the start of high speed transitions, using the CSI_TX_IF_DPHY_IRQ register. The signals can be used to generate an error interrupt event if the relevant bits are set in the CSI_TX_IF_DPHY_IRQ_MASK register. The flag can be cleared by writing to the CSI_TX_IF_DPHY_IRQ register.

12.7.3.5.10 CSI_TX_IF ULPS Operation

Each lane can be put in ultra-low power state (ULPS) by software configuration. The ULPS mode requires all the following conditions:

- The lane must be in stop state.
- For data lanes, no data must be pending in the CSI_TX_IF module.

Copyright © 2024 Texas Instruments Incorporated

12.7.3.5.11 CSI_TX_IF System Frame Rate Measurement

The CSI_TX_IF provides an interrupt event to signal when the frame start (FS) and frame end (FE) packets are sent to the DPHY_TX. These events are generated on a stream basis for each virtual channel and allow a system configured to perform virtual channel interleaving to capture the events, and to derive the frame rate achieved based on the flow control and clock configuration.

The system performance will be impacted by the FIFO sizing and any flow control mechanism used by the pixel streams, so this provides a mechanism for checking that the system is scaled correctly.

12.7.3.5.12 CSI_TX_IF Configuration for PSI_L

Table 12-407 lists the settings that must be programed before enabling CSI_TX_IF controller and sending PSI_L data.

Table 12-407. CSI_TX_IF Configuration table for PSI_L

Format	Pixel transmit mode used	Size	Details
YUV420-8	single	0	Must set YUV420 mode in CSI_TX_IF_DMACNTX_j
YUV420-10	single	1	Must set YUV420 mode in CSI_TX_IF_DMACNTX_j
YUV422-8	single	0	Must set YUV422 mode in CSI_TX_IF_DMACNTX_j
YUV422-10	single	1	Must set YUV422 mode in CSI_TX_IF_DMACNTX_j
YUV422-8	packed	2	Must set YUV422 mode in CSI_TX_IF_DMACNTX_j
RGB888, RGB666	single	2	
RGB565, RGB555, RGB444	single	1	
RAW6-7(1)	single	0	
RAW8(1)	dual	1	
RAW8(1)	quad	2	
RAW10(1)	single	1	
RAW10(1)	dual	2	
RAW12(1)	single	1	No packing
RAW12(1)	dual	2	No packing
RAW12(packed)(1)	single	0	Must set CSI_TX_IF_DMACNTX_j[23] PACK12_CFG = 1
RAW12(packed)(1)	dual	1	Must set CSI_TX_IF_DMACNTX_j[23] PACK12_CFG = 1
RAW14(1)	single	1	
RAW14(1)	dual	2	
RAW16(1)	single	1	
RAW20(1)	single	2	

(1) User defined/blanking/generic data types are programmed the same way as RAW data types.

12.7.3.5.13 CSI_TX_IF Configuration for Color Bar

The following lists the Color Bar configuration requirements:

- Color bar must be set to packed YUV422-8 format, this is an extended datatype in CSI_TX_IF controller.
- Software must program CSI_TX_IF controller before enabling the color bar from the CSI_TX_IF_COLOR_CNTL[0] EN bit-field.
- Program CSI_TX_IF_COLOR_LINE_DELAY, CSI_TX_IF_COLOR_FRAME_DELAY, CSI_TX_IF_COLOR_START_DELAY, and CSI_TX_IF_COLOR_PARAM registers with appropriate configurations. Update CSI_TX_IF_COLOR_CNTL with virtual channel and data type desired and enable it.

Copyright © 2024 Texas Instruments Incorporated

12.7.3.5.14 CSI_TX_IF Error Recovery

When an underflow error occurs, software must reset the entire CSI_TX_IF by SoC PSC reset and restart over.

12.7.3.5.15 CSI_TX_IF Power up/down Sequence

The following lists the power up/down sequence requirements:

1. Software must ensure no more PSI_L traffic is directed toward the CSI_TX_IF module.
2. Software must then go through a tear down process on PSI_L gasket inside CSI_TX_IF wrapper. Refer to chapter *PSI_L* for tear down process.
3. Software must then check the stream idle values so that they are idle in the CSI_TX_IF_CONTROL1 register.
4. Software must then stop all traffic inside the CSI_TX_IF controller and wait till it is idle.
5. After that software can then proceed to do PSC power down routine.

Copyright © 2024 Texas Instruments Incorporated

12.8 Shared MIPI D-PHY Transmitter (DPHY_TX)

This section describes the features and functions of the shared MIPI D-PHY Transmitter (DPHY_TX).

12.8.1 DPHY_TX Subsystem Overview

The DPHY_TX module provides one option for video output interfacing by implementing a four-lane MIPI D-PHY Transmitter.

The device includes of DPHY_TX module.

12.8.1.1 DPHY_TX Features

The DPHY_TX module supports the following main features:

- Compliance with MIPI D-PHY 1.2 physical layer interface specification and features
- 1, 2 or 4 data lanes, in addition to clock signaling
- Maximum data rate up to 2.5 Gbps per data lane (with deskew) and 1.5 Gbps (without deskew)
- Protocol Peripheral Interface (PPI)
- HS (High-Speed) continuous and burst modes
- LP (Low-Power), ULPM (Ultra-Lower Power), and Shutdown modes
- Data lanes can be independently operated in HS or ULP mode.
- Forward direction and reverse direction escape modes (only on Lane-0)
- Automatic termination control in both high-speed and low-power modes
- Fault detection:
  - Contention detection
  - Sequence error detection (corruption on lanes)

Unsupported Features:

- See the Module Integration section of this document for a list of module features not supported by the integration on this Device.

Copyright © 2024 Texas Instruments Incorporated

12.8.1.2 DPHY_TX Ports

This section describes the DPHY_TX ports related to clocks and resets.

Table 12-408. DPHY_TX Clocks and Resets

Clocks	
Module Clock Input	**Description**
DPHY_TX_CLK	DPHY_TX interface clock
PSM_CLK	DPHY_TX reference clock
IP1_PPI_K_M_TXCLKESC	DPHY_TX escape mode clock
IP1_PPI_K_LN0_M_TXCLKESC	DPHY_TX escape mode clock for lane 0 for IP1_PPI
IP1_PPI_K_LN1_M_TXCLKESC	DPHY_TX escape mode clock for lane 1 for IP1_PPI
IP1_PPI_K_LN2_M_TXCLKESC	DPHY_TX escape mode clock for lane 2 for IP1_PPI
IP1_PPI_K_LN3_M_TXCLKESC	DPHY_TX escape mode clock for lane 3 for IP1_PPI
DPHY_REF_CLK	DPHY_TX reference clock.
PPI_0_TXBYTECLKHS	DPHY_TX byte clock for IP1_PPI
DPHY_0_RXCLKESC	DPHY_TX reverse escape mode recovered clock from lane 0
DPHY_TXBYTECLKHS	DPHY_TX byte clock for IP2_PPI
Resets	
Module Reset Input	**Description**
DPHY_TX_RST	DPHY_TX reset

12.8.2 DPHY_TX Environment

This section describes the DPHY_TX application fields from an environment point of view (external connections).

Table 12-409 describes the DPHY_TX signals when transmitting data from a DSI module, and Table 12-410 describes the DPHY_TX signals when transmitting data from a CSI_TX_IF module.

Table 12-409. DPHY_TX I/O Signals in DSI Mode

Device Level Signal	I/O[(1)]	Description
DSI_TXN0	I/O	DPHY_TX0 Lane 0 Transmit Differential Data (Negative)
DSI_TXP0	I/O	DPHY_TX0 Lane 0 Transmit Differential Data (Positive)
DSI_TXN1	O	DPHY_TX0 Lane 1 Transmit Differential Data (Negative)
DSI_TXP1	O	DPHY_TX0 Lane 1 Transmit Differential Data (Positive)
DSI_TXN2	O	DPHY_TX0 Lane 2 Transmit Differential Data (Negative)
DSI_TXP2	O	DPHY_TX0 Lane 2 Transmit Differential Data (Positive)
DSI_TXN3	O	DPHY_TX0 Lane 3 Transmit Differential Data (Negative)
DSI_TXP3	O	DPHY_TX0 Lane 3 Transmit Differential Data (Positive)
DSI_TXCLKN	O	DPHY_TX0 Transmit Differential Clock Lane (Negative)
DSI_TXCLKP	O	DPHY_TX0 Transmit Differential Clock Lane (Positive)
DSI_TXRCALIB	A	DPHY_TX0 pin for external calibration resistor. Refer to the device-specific Datasheet for a recommended resistor value.
DSI_ATB_0_H	I/O	DPHY_TX0 Analog Test Bus 0
DSI_ATB_1_H	I/O	DPHY_TX0 Analog Test Bus 1

(1) I = Input; O = Output.

Table 12-410. DPHY_TX I/O Signals in CSI Mode

Device Level Signal	I/O	Description
CSI0_TXN0	O	DPHY_TX0 Lane 0 Transmit Differential Data (Negative)
CSI0_TXP0	O	DPHY_TX0 Lane 0 Transmit Differential Data (Positive)
CSI0_TXN1	O	DPHY_TX0 Lane 1 Transmit Differential Data (Negative)
CSI0_TXP1	O	DPHY_TX0 Lane 1 Transmit Differential Data (Positive)
CSI0_TXN2	O	DPHY_TX0 Lane 2 Transmit Differential Data (Negative)

Copyright © 2024 Texas Instruments Incorporated

Table 12-410. DPHY_TX I/O Signals in CSI Mode (continued)

Device Level Signal	I/O	Description
CSI0_TXP2	O	DPHY_TX0 Lane 2 Transmit Differential Data (Positive)
CSI0_TXN3	O	DPHY_TX0 Lane 3 Transmit Differential Data (Negative)
CSI0_TXP3	O	DPHY_TX0 Lane 3 Transmit Differential Data (Positive)
CSI0_TXCLKN	O	DPHY_TX0 Transmit Differential Clock Lane (Negative)
CSI0_TXCLKP	O	DPHY_TX0 Transmit Differential Clock Lane (Positive)

Copyright © 2024 Texas Instruments Incorporated

12.9 Timer Modules

This section describes the timer modules in the device.

12.9.1 Global Timebase Counter (GTC)

This section describes the global timebase counter (GTC) in the device.

12.9.1.1 GTC Overview

The GTC module provides a continuous running counter that can be used for time synchronization and debug trace time stamping.

The device includes one GTC instance.

12.9.1.1.1 GTC Features

The GTC supports the following features:

- 64-bit up counter
- No rollover during the lifetime of the device
- Compatible with ARMv8 system counter requirements:
  - Disabled at power-up
  - Register definition and memory map aligned to ARMv8 definition
  - Implements memory-mapped counter control and status frames
- Outputs reflected binary (Gray) encoded timer value for system timer bus distribution to other modules
- Selectable counter bit output as a push event that can be used by CPTS modules, timers or interface protocols

12.9.1.1.2 GTC Ports

Table 12-411. GTC Clocks and Resets

Clocks	
Module Clock Input	**Description**
GTC_CLK	Module functional (counter) clock.
GTC_VBUSP_CLK	Module interface clock.
Resets	
Module Reset Input	**Description**
GTC_POR_RST	Module reset. Affects counter and MMR logic.
GTC_SYS_RST	POR that is unstretched and not delayed by any sequential logic. Used for boot config to sample device pins.

Table 12-412. GTC Hardware Requests

Time Sync Events		
Module Sync Output	**Description**	**Type**
GTC_GTC_PUSH_EVENT_0	GTC hardware push event	Pulse

Copyright © 2024 Texas Instruments Incorporated

12.9.1.2 GTC Functional Description

12.9.1.2.1 GTC Block Diagram

The GTC is essentially comprised of a 64-bit up counter, a Gray encoder, a 64-bit multiplexer, and memory-mapped control registers. Figure 12-474 shows a high-level block diagram of the GTC.

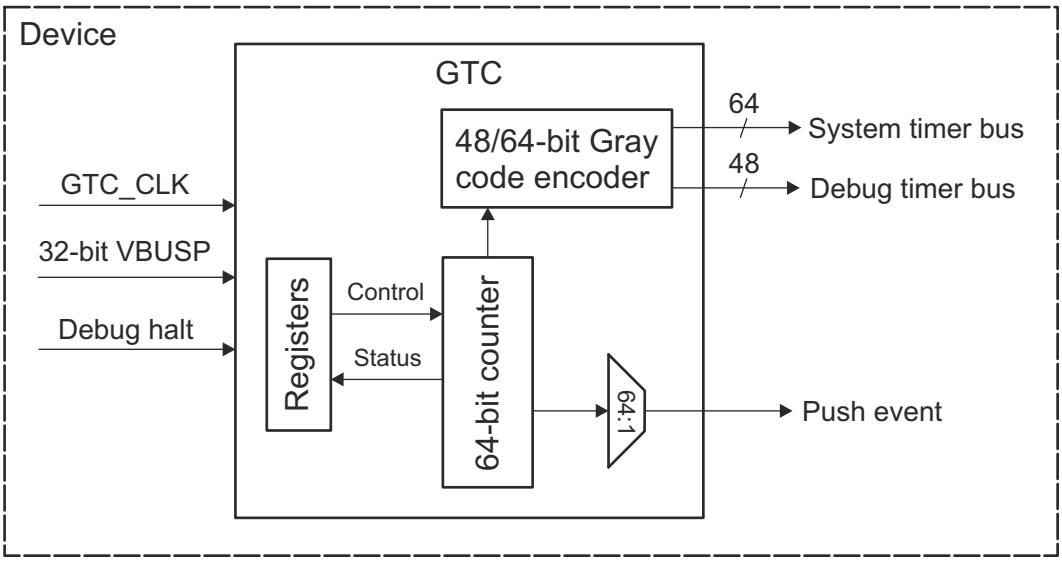


Figure 12-474. GTC Block Diagram

12.9.1.2.2 GTC Counter

The counter is a 64-bit binary up counter that increments on every GTC_CLK rising edge. The source for GTC_CLK is selected through a mux which is controlled via the CTRLMMR_GTC_CLKSEL[2-0] CLK_SEL register bit field.

The counter is disabled by default (at power-on-reset) and increments only when enabled by setting the GTC_CNTCR[0] EN bit to '1'. The current counter value is readable via the combined COUNTVALUE bit field of both GTC_CNTCV_HI (upper 32 bits) and GTC_CNTCV_LO (lower 32 bits) registers.

When counting is disabled, a new counter value can be loaded via a software write to the combined COUNTVALUE bit field. In this case, when the counter gets re-enabled, it will start counting from the last value written to this field.

The counter can be optionally configured to stop incrementing when a debug halt signal is issued, by writing a '1' to the GTC_CNTCR[1] HDBG bit. This condition is indicated through the GTC_CNTCR[1] DBGH status bit.

12.9.1.2.3 GTC Gray Encoder

The Gray encoder performs a realtime conversion of the binary counter value into a 64-bit Gray code. This value is exported as the system timer bus for distributing the counter value to:

- (2x) A72SS global timestamp input bus
- (1x) GPU timestamp input bus

A second realtime conversion of the 48 LSBs of the binary counter value into a 48-bit Gray code value is also performed and exported to:

- (2x) A72SS debug timestamp input bus
- (1x) DMPAC timestamp input bus
- (2x) VPAC timestamp input bus
- (1x) DMSC timestamp input bus

Copyright © 2024 Texas Instruments Incorporated

12.9.1.2.4 GTC Push Event Generation

The GTC can generate periodic push events for global time synchronization at up to half the GTC_CLK frequency. The frequency of the events is determined by selecting one of the counter bits through the internal [64:1] mux, which is controlled through the GTC_PUSHEVT[5-0] EXPBIT_SEL bit field.

12.9.1.2.5 GTC Register Partitioning

The ARMv8 architecture spec requires specific system-level components, each with one or two register frames. To accommodate this, the GTC memory-mapped registers (MMRs) are divided into four separate regions (4KB each):

- Peripheral MMRs (GTC0_GTC_CFG0): This region includes standard peripheral identification registers and any other control registers not associated with the ARMv8 system timer functions.
- Counter control MMRs (GTC0_GTC_CFG1): This region includes registers that provide control over the operation of the system timer.
- Counter status MMRs (GTC0_GTC_CFG2): This region includes registers that provide a mechanism for reading the status of the system timer.
- Timer control MMRs (GTC0_GTC_CFG3): This region includes registers that identify and provide a control mechanism for memory-mapped timer implementations. For this device, no memory-mapped timers are implemented and only the minimum registers required to indicate the presence of no timers are necessary.

Copyright © 2024 Texas Instruments Incorporated

12.9.2 Windowed Watchdog Timer (WWDT)

This section describes the Windowed Watchdog Timer (WWDT), implemented by using the Digital Windowed Watchdog (DWWD) function of the Real Time Interrupt (RTI) module in the device.

> **CAUTION**
>
> The RTI module shall be used to serve only as a Digital Windowed Watchdog (DWWD).

The Real Time Interrupt module provides timer functionality for operating systems and for benchmarking code. The module incorporates several counters, which define the timebases needed for scheduling in the operating system.

This module is specifically designed to fulfill the requirements for OSEK (“Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug”; “Open Systems and the Corresponding Interfaces for Automotive Electronics”) as well as OSEK/Time compliant operating systems.

The timers also provide the ability to benchmark certain areas of code by reading the counter contents at the beginning and the end of the desired code range and calculating the difference between the values.

12.9.2.1 RTI Overview

12.9.2.1.1 RTI Features

The RTI modules include the following main features:

- Windowed Watchdog Timer (WWDT) feature.
- Two independent 64 bit counter blocks (counter block0 or counter block1). Each block consists of
  - One 32 bit up counter
  - One 32 bit free running counter
  - Two capture registers for capturing the prescale and free running counter on a special event.
- Free running counter 0 can be incremented by either the internal prescale counter or by an external event.
- Four configurable compare registers for generating operating system ticks . Each event can be driven by either counter block0 or counter block1.
- Fast enabling/disabling of events.
- RTI clock input derived from any of the available clock sources, selectable in the System Module
- Optional capability to drive a pulse-width modulated signal out on an interrupt line.

12.9.2.1.2 RTI Not Supported Features

See the Module Integration section of this document for a list of module features not supported by the integration on this Device.

12.9.2.1.3 RTI Ports

Table 12-413. RTI Clocks and Resets

Clocks	
Module Clock Input	**Description**
RTI_ICLK	RTI Interface Clock
RTI_FCLK	RTI Functional Clock.
Resets	
Module Reset Input	**Description**
RTI_RST	RTI Asynchronous Reset
RTI_POR_RST	RTI Power-On Reset

Copyright © 2024 Texas Instruments Incorporated

Table 12-414. RTI Hardware Requests

Interrupt Requests		
Module Interrupt Signal	**Description**	**Type**
RTI_INTR_WWD_0	RTI window watchdog violation interrupt	Pulse

Copyright © 2024 Texas Instruments Incorporated

12.9.2.2 RTI Functional Description

The MCU_RTI[1-0] and RTIi (where i =) modules are hereinafter referred to as RTI module.

12.9.2.2.1 RTI Counter Operation

Note

Some of the RTI features described in this section may not be supported on this family of devices. For more information, see *RTI Not Supported Features*.

Figure 12-475 shows the RTI module counter blocks. The RTI module supports two counter blocks.

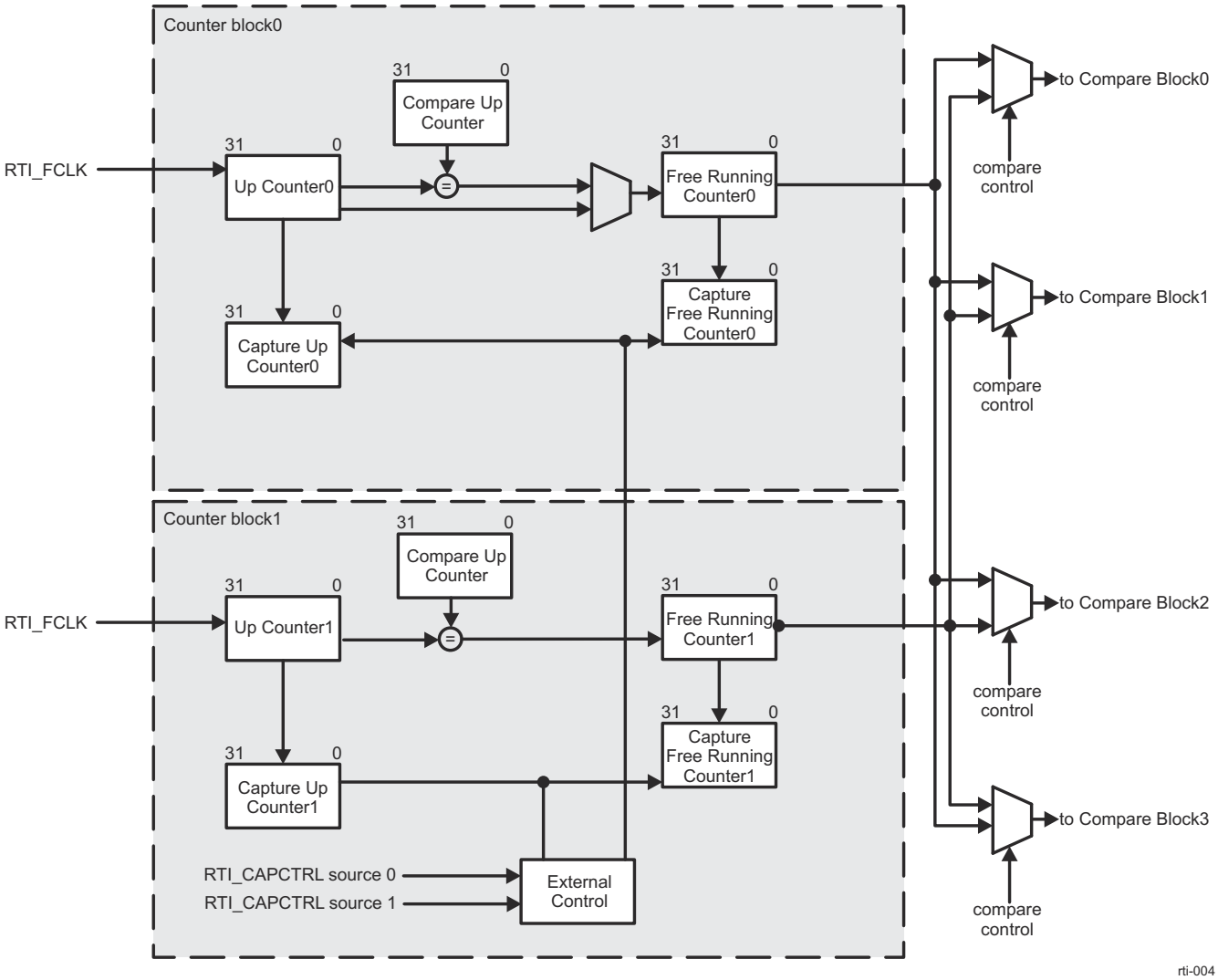


Figure 12-475. RTI Counters Block Diagram

Each block consists of two 32 bit up counters - Up Counter (UC) and Free Running Counter (FRC). The Up Counter (RTI_UC0 or RTI_UC1 register) is driven by the RTI_FCLK and counts up until the compare value in the Compare Up Counter register (RTI_CPUC0 or RTI_CPUC1) is reached. When the compare matches, the second counter (RTI_FRC0 or RTI_FRC1 register), which is a free running counter, is incremented. At the same time UCx is reset to zero.

To ensure the consistency of the counters, when both counter value have to be determined, the Free Running Counter has to be read first. This will ensure that at the CPU read cycle, the Up Counter value is stored in the

Copyright © 2024 Texas Instruments Incorporated

counter register. The second read is done on the Up Counter register, which holds then the value of the counter cycle of the previous read on the Free Running Counter register.

Both blocks provide also a capture feature on external events. Two capture sources can trigger the capture event. Which event triggers block 0 or block 1 is configurable from the RTI_CAPCTRL register. The sources are coming from the interrupt manager, in order to be able to generate a capture event when one of the peripheral modules has generated an interrupt. The peripheral, which can generate an event is configured in the interrupt manager. When the event is detected, UCx and FRCx are stored in Capture Up Counter (RTI_CAUC0 or RTI_CAUC1) and Capture Free Running Counter (RTI_CAFRC0 or RTI_CAFRC1) registers. The read order of the captured values has to be like the order of the actual counters. So the CAFRCx has to be read first and the CAUCx registers has to be read after the CAFRCx value was determined. While the CAFRCx is read the CAUCx value is loaded into a shadow register to ensure data consistency, if during the two reads of the captured data another capture event happens. If the application fails to read the two registers before a second capture event happens, the previous data will be overwritten.

Figure 12-476 shows the block diagram for one compare block. The RTI module supports four compare blocks.

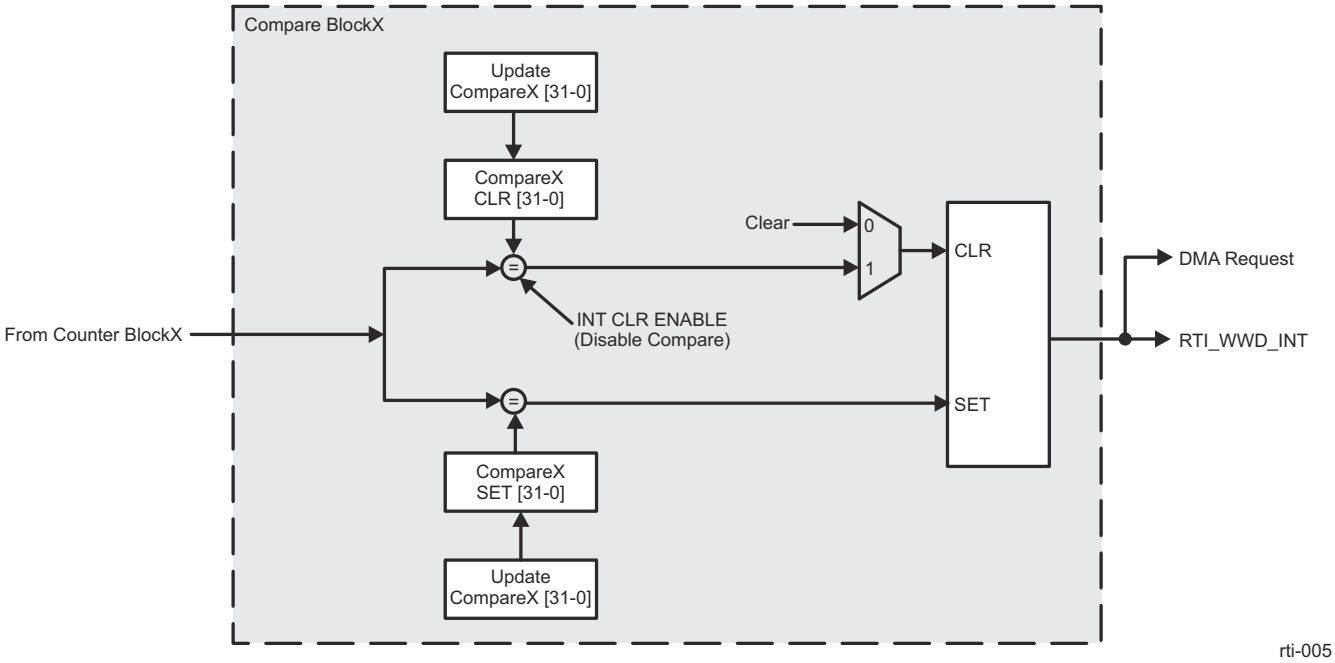


Figure 12-476. RTI Compare Block Diagram

In order to generate interrupt requests to the interrupt manager, there are four compare registers (RTI_COMP0, RTI_COMP1, RTI_COMP2, and RTI_COMP3). Each of the compare registers can be configured to work either on FRC0 (Counter block0) or FRC1 (Counter block1). When the counter value matches the compare value, an interrupt is generated. This sets an interrupt request line to the interrupt manager. The compare value gets updated automatically with the value stored in Update Compare (RTI_UDCP0, RTI_UDCP1, RTI_UDCP2, and RTI_UDCP3) registers when the compare matches. This gives the ability to generate periodic interrupts/DMA requests without having to update the compare value by software.

An optional feature allows an application to program another compare value which is then used to clear the interrupt request line. This feature is supported by four compare clear registers (RTI_COMP0CLR, RTI_COMP1CLR, RTI_COMP2CLR, and RTI_COMP3CLR). When the counter value matches the compare clear value, the interrupt line is cleared. This clears the interrupt request line to the interrupt manager. The compare clear value gets updated automatically with the value stored in Update Compare (RTI_UDCPx) registers when the compare matches.

Copyright © 2024 Texas Instruments Incorporated

12.9.2.2.2 RTI Digital Watchdog

Note

Some of the RTI features described in this section may not be supported on this family of devices. For more information, see *RTI Not Supported Features*.

Some applications might use a digital watchdog (DWD) integrated in the RTI module. The digital watchdog generates resets after a programmable period, if no correct key sequence is written to the RTI_WDKEY register. Figure 12-477 shows the digital watchdog functional block.

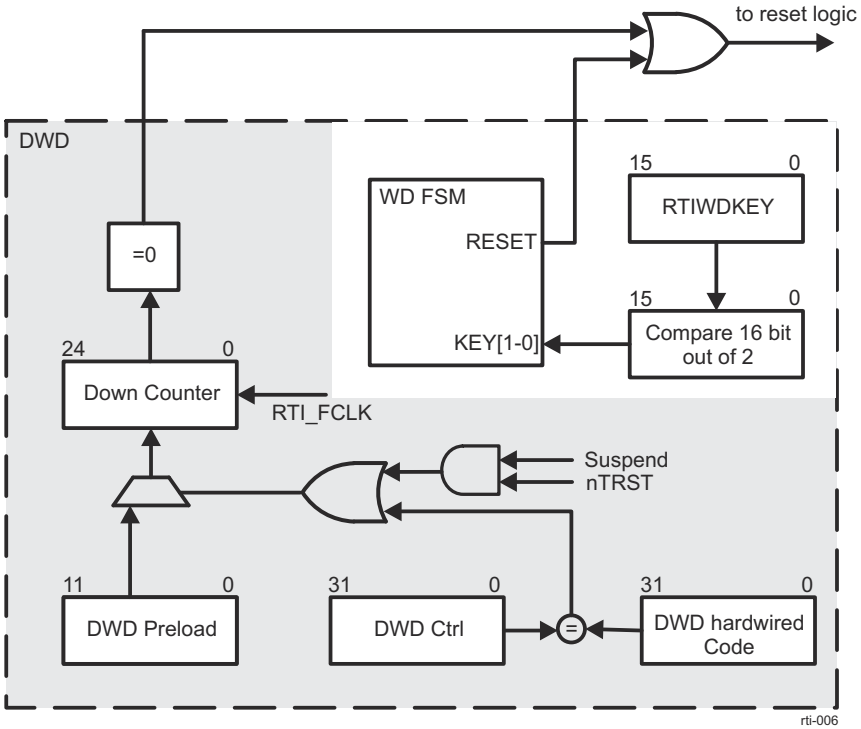


Figure 12-477. RTI Digital Watchdog Functional Block Diagram

The digital watchdog functionality is implemented such that it can be enabled by software.

The DWD starts counting down from the reset value of the RTI_DWDCNTR (DWD Counter Register). The DWD preload register can be configured at any time by the application according to the desired time-out period.

When enabled by software, the digital watchdog is disabled after system reset. If it should be used, it has to be enabled by writing A98559DAh to the RTI_DWDCTRL register. The DWD timeout period must be configured using the DWD preload register before the DWD is enabled. The DWD cannot be disabled by the application once it is enabled.

Note

When the DWD is enabled by software, any system reset will disable the DWD. This reset could have been generated by the watchdog itself.

If the correct key sequence is written to the RTI_WDKEY register (E51Ah followed by A35Ch), the 25-bit DWD Down Counter is reloaded with the 12-bit preload value stored in RTI_DWDPRLD register. If any incorrect value is written to the RTI_WDKEY register, a watchdog reset will occur immediately. A reset will also be generated, when the DWD Down Counter is decremented to 0.

The user has to take into account that the write to the RTI_WDKEY register takes 3 RTI_ICLK cycles. This needs to be considered for the DWD expiration calculation.

Copyright © 2024 Texas Instruments Incorporated

The DWD Down Counter will be decremented with RTI_FCLK frequency. If the RTI_FCLK is switched off via the disable registers of the Clock management, the DWD counter stops decrementing. The DWD module cannot generate a reset under this condition.

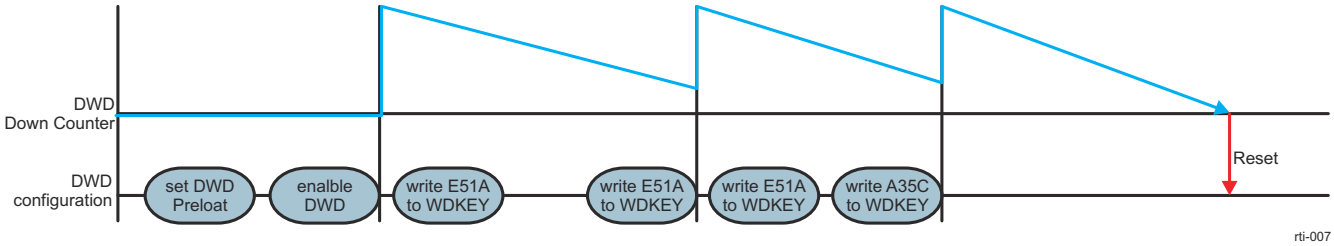


Figure 12-478. RTI Digital Watchdog Operation

The expiration time of the DWD Down Counter can be determined with following equation:

$$t_{exp} = (RTI_DWDPRLD + 1) \times 2^{13} / RTI_FCLK \quad (28)$$

where RTI_DWDPRLD = 0...4095

12.9.2.2.3 RTI Digital Windowed Watchdog

Note

Some of the RTI features described in this section may not be supported on this family of devices. For more information, see *RTI Not Supported Features*.

Note

Digital windowed watchdog (DWWD) timer is implemented using the digital windowed watchdog function of the RTI modules. Real time interrupt functionality is not supported. In this mode, the timer should default to disabled and user can adjust the period as desired before enabling the watchdog.

In addition to the time-out boundary configurable via the digital watchdog (DWD), some applications may also want to configure the start-time boundary of the watchdog. This is enabled by the digital windowed watchdog (DWWD) feature.

Functional Behavior

The DWWD opens a configurable time window in which the watchdog must be serviced. Any attempt to service the watchdog outside this time window, or a failure to service the watchdog in this time window, will cause the watchdog to generate either a reset or a non-maskable interrupt to the CPU. This is controlled by configuring the RTI_WWDRXNCTRL register. As stated earlier, when the watchdog needs to be enabled by software, the watchdog counter is disabled on a system reset. When the DWWD is configured to generate a non-maskable interrupt on a window violation, the watchdog counter continues to count down. The RTI_INTR_WWD interrupt handler needs to clear the watchdog violation status flag(s) and then service the watchdog by writing the correct sequence in the watchdog key RTI_WDKEY register. This service will cause the watchdog counter to get reloaded from the preload value and start counting down. If the RTI_INTR_WWD handler does not service the watchdog in time, it could count down all the way to zero and wrap around. No second exception for a time out is generated in this case.

Configuration of DWWD

The DWWD preload value (same as DWD preload) can only be configured when the DWWD counter is disabled. The window size and watchdog reaction to a violation can be configured even after the watchdog has been enabled. Any changes to the window size and watchdog reaction configurations will only take effect after the next servicing of the DWWD.

Copyright © 2024 Texas Instruments Incorporated

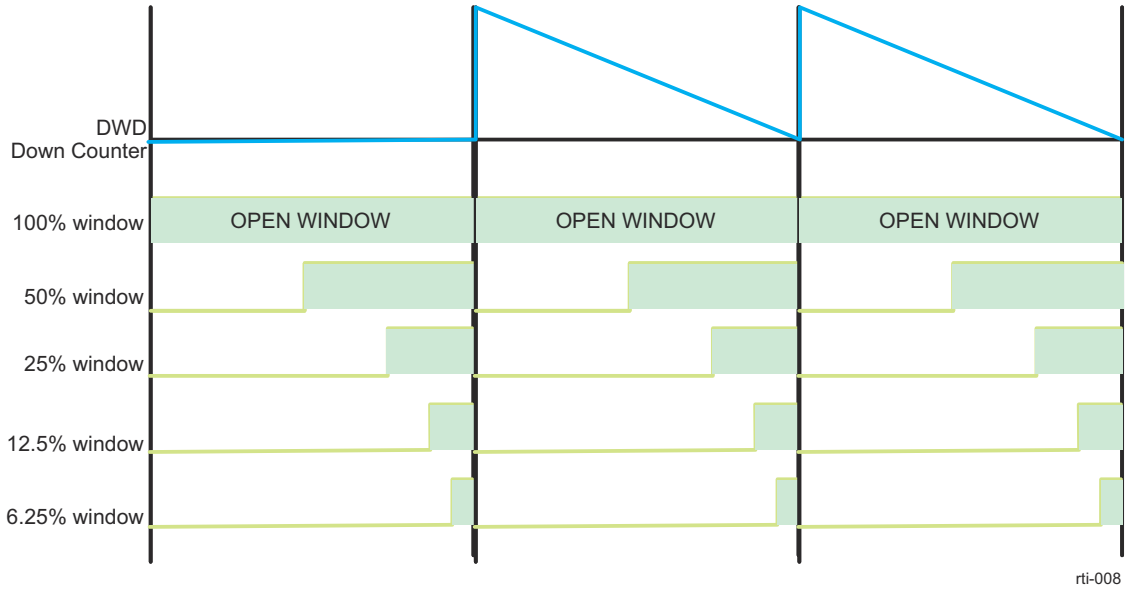


Figure 12-479. RTI Digital Windowed Watchdog Timing Example

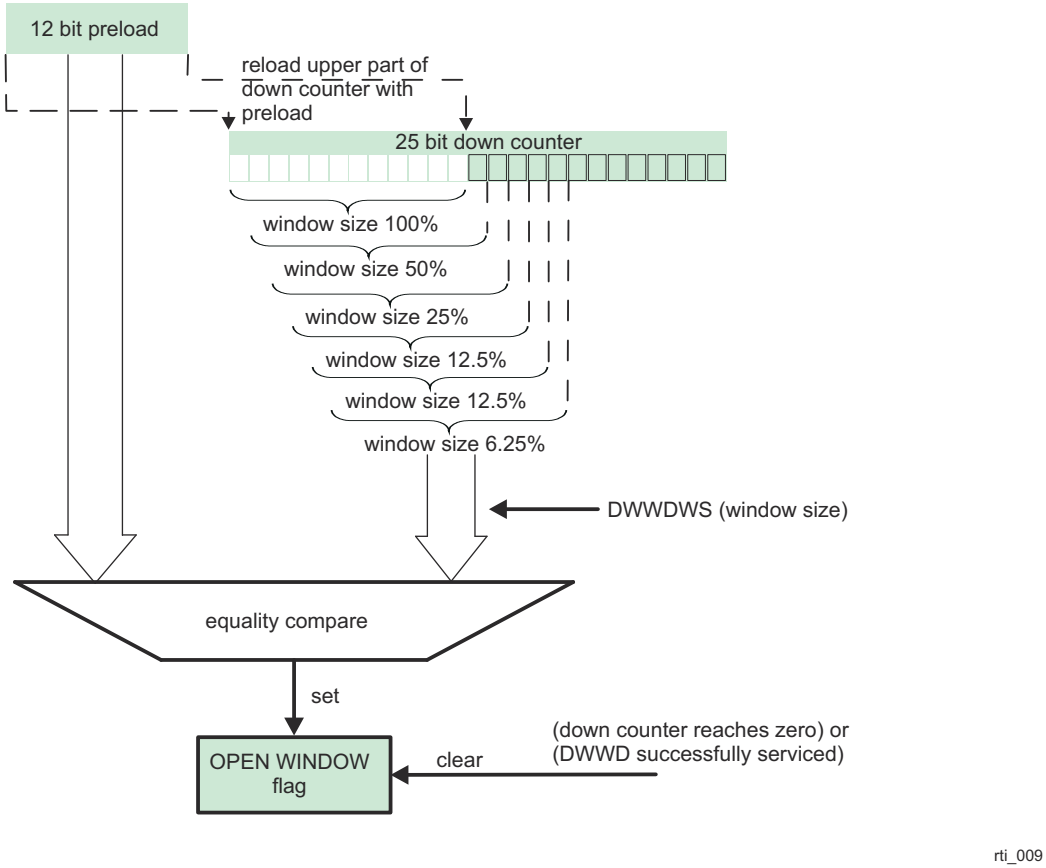


Figure 12-480. RTI Digital Windowed Watchdog Operation Block Diagram

12.9.2.2.4 RTI Low Power Mode Operation

The operation of the RTI module is guaranteed in run, doze and snooze mode. In sleep or hibernate mode all clocks will be switched off and the RTI module will not work.

In doze and snooze modes all parts of the RTI are active, since it has to be able to wake up the device with compare and timebases interrupts. Capturing events generated by the interrupt module is also possible since in

Copyright © 2024 Texas Instruments Incorporated

both modes the peripheral modules are able to generate interrupts, which can trigger capture events. The RTI module will generate compare and timebases interrupts. The compare interrupts will periodically wake up the device.

Note

In the special case of doze mode with DPLL off, RTI_FCLK might have a different period than with DPLL enabled, since RTI_FCLK will be derived from the oscillator output. It has to be ensured that the RTI_ICLK to RTI_FCLK ratio is at least 3:1.

The DWD/DWWD remains active when the device enters low power mode as long as the RTI_FCLK is kept active.

Whenever the LPSC that controls an RTI is in any state other than Enable (see *Module States*), the RTI cannot count or generate interrupts. For more information, see *Power Control Modules*.

During standard SoC warm reset the RTI and the rest of the SoC are reset. In this case, the RTI counters are stopped and interrupts are not issued. During reset-isolated warm reset, if an R5FSS is put in reset-isolation, then the associated RTI also becomes reset isolated. As such, the R5FSS and the RTI are not reset, but RTI stops counting and cannot generate interrupts until R5FSS is taken out of CLKSTOP (brought to Enable state).

12.9.2.2.5 RTI Debug Mode Behavior

Note

Some of the RTI features described in this section may not be supported on this family of devices. For more information, see *RTI Not Supported Features*.

Once the system enters debug mode, the behavior of the RTI depends on the RTI_GCTRL[15] COS bit. If the bit is cleared and debug mode is active, all counters will stop operation. If the bit is set to one, all counters will be clocked normally and the RTI will work like in normal mode.

The DWD counter will not decrement in debug mode and will hold its current value, regardless of the RTI_GCTRL[15] COS bit.

Note

The user must not service the watchdog while in debug mode.

Copyright © 2024 Texas Instruments Incorporated

12.9.3 Timers

This section describes the Timer modules for the device.

12.9.3.1 Timers Overview

There are thirty timer modules in the device.

All timers include specific functions to generate accurate tick interrupts to the operating system.

Each timer can be clocked from several different independent clocks. The selection of clock source is made from registers in the MCU_CTRL_MMR0/CTRL_MMR0.

In the MCU domain the device provides 10 timer pins to be used as MCU Timer Capture inputs or as MCU Timer PWM outputs. In order to provide maximum flexibility, these 10 pins may be used with any of MCU_TIMER0 through MCU_TIMER9 instances. System level muxes are used to control the capture source pin for each MCU_TIMER[9-0] and the MCU_TIMER[9-0] source for each MCU_TIMER_IO[9-0] PWM output.

In the MAIN domain the device provides 8 timer pins to be used as Timer Capture inputs or as Timer PWM outputs. For maximum flexibility, these 8 pins may be used with any of TIMER0 through TIMER19 instances. System level muxes are used to control the capture source pin for each TIMER[19-0] and the TIMER[19-0] source for each TIMER_IO[7-0] PWM output.

Each odd numbered timer instance from each of the domains may be optionally cascaded with the previous even numbered timer instance from the same domain to form up to a 64-bit timer. For example, TIMER1 may be cascaded to TIMER0, MCU_TIMER1 may be cascaded to MCU_TIMER0, etc.

When cascaded, TIMERi acts as a 32-bit prescaler to TIMERi+1, as well as MCU_TIMERn acts as a 32-bit prescaler to MCU_TIMERn+1. TIMERi / MCU_TIMERn must be configured to generate a PWM output edge at the desired rate to increment the TIMERi+1/ MCU_TIMERn+1 counter.

12.9.3.1.1 Timers Features

The following are the main features of the timer controllers:

- Slave interface supports:
  - 32-bit data bus width
  - 32-bit access supported
  - 10-bit address bus width
  - Write nonposted transaction mode supported
- Interrupts generated on overflow, compare, and capture
- Free-running 32-bit upward counter
- Compare and capture modes
- Autoreload mode
- Start/stop mode
- Programmable divider clock source (2^n, where n = [0-8])
- Dedicated input trigger for capture mode and dedicated output trigger/PWM signal
- On-the-fly read/write register (while counting)
- Generates a 1-ms tick clock when functional clock is 32.768 kHz

Unsupported Features:

- See the Module Integration section of this document for a list of module features not supported by the integration on this Device.

12.9.3.1.2 Timers Ports

Table 12-415. TIMER Clocks and Resets

Clocks	
Module Clock Input	**Description**
TIMER_ICLK	TIMER Interface Clock
TIMER_FCLK	TIMER Functional Clock.
Resets	

Copyright © 2024 Texas Instruments Incorporated

Table 12-415. TIMER Clocks and Resets (continued)

Module Reset Input	Description
TIMER_RST	Asyncronous Reset to TIMER

Copyright © 2024 Texas Instruments Incorporated

Table 12-416. TIMER Hardware Requests

Interrupt Requests		
Module Interrupt Signal	**Description**	**Type**
TIMER_INTR_PEND_0	TIMER Interrupt Request	Level
TIMER_TIMER_PWM_0	TIMER Timesync Event	Level

12.9.3.2 Timers Environment

This section describes the timers external connections (environment).

Table 12-417. Timer I/O Signals

Module Pin	I/O[(1)]	Description
TIMER_IO[9-0]	I/O	TIMER[9-0] trigger input or TIMER[9-0] output

(1) When configured for that function; I = Input, O = Output

Copyright © 2024 Texas Instruments Incorporated

12.9.3.3 Timers Functional Description

Each timer contains a free-running upward counter with autoreload capability on overflow. The timer counter can be read and written on-the-fly (while counting). Each timer includes compare logic to allow an interrupt event on a programmable counter matching value. A dedicated output signal can be pulsed or toggled on either an overflow or a match event. This offers time-stamp trigger signaling or PWM signal sources. A dedicated input signal can be used to trigger an automatic timer counter capture or an interrupt event on a programmable input signal transition. A programmable clock divider (prescaler) allows reduction of the timer input clock frequency. All internal timer interrupt sources are merged into one module interrupt line and one wake-up line.

Each internal interrupt source can be independently enabled and disabled by a dedicated bit in the TIMER_IRQSTATUS_SET and TIMER_IRQSTATUS_CLR register for the interrupt features, and a dedicated bit of the TIMER_IRQWAKEEN register for a wake-up. In addition, timers have a mechanism implemented to generate an accurate tick interrupt.

For each timer implemented in the device, there are two possible clock sources:

- 32-kHz clock
- System clock

For more information of the selection of the input clock source, see *Clocking*.

Each timer supports three functional modes:

- Timer mode
- Capture mode
- Compare mode

The capture and compare modes are disabled by default after core reset.

12.9.3.3.1 Timer Block Diagram

Figure 12-481 is a block diagram of the timers.

Copyright © 2024 Texas Instruments Incorporated

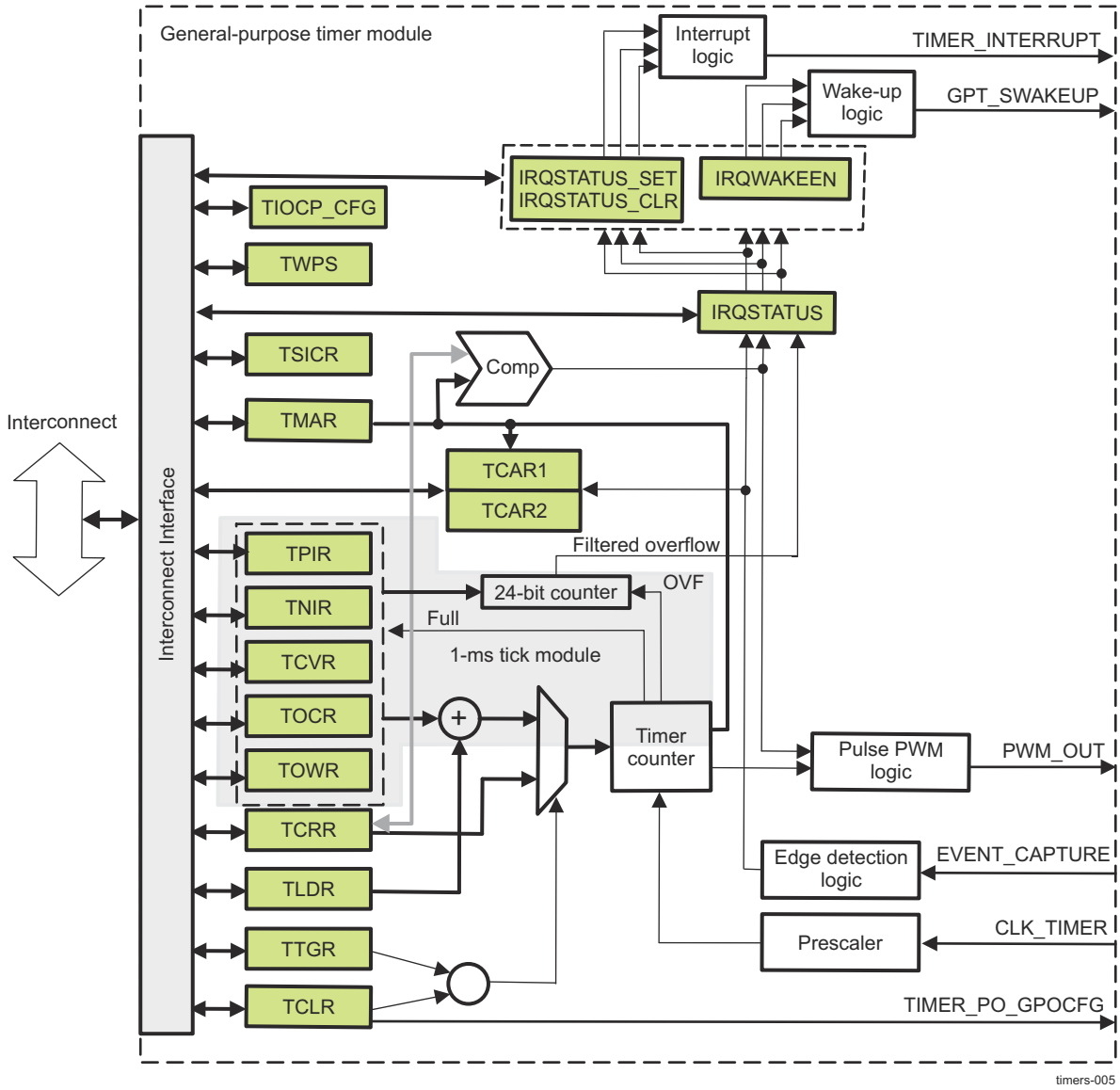


Figure 12-481. Timer Block Diagram

12.9.3.3.2 Timer Power Management

Note

Some of the timers features described in this section may not be supported on this family of devices. For more information, see *Timers Not Supported Features*.

The way the timer acknowledges the LPSC clock stop request is configurable through the TIMER_TIOCP_CFG[3-2] IDLEMODE bit field.

Table 12-418 lists the IDLEMODE settings and the related acknowledgment modes.

Table 12-418. IDLEMODE Settings

IDLEMODE Value	Selected Mode	Description
00	Force-idle	The clock stop request is unconditionally acknowledged from the timer, regardless of its internal operations. This mode must be used carefully, because it does not prevent the loss of data when the clock is switched off.
01	No-idle	The clock stop request is never acknowledged from the timer. This mode is safe from a module point of view but is not efficient from a power-saving perspective because the clocks remain active.

Copyright © 2024 Texas Instruments Incorporated

Table 12-418. IDLEMODE Settings (continued)

IDLEMODE Value	Selected Mode	Description
10	Smart-idle	The timer acknowledges the clock stop request, basing its decision on its internal activity. The acknowledge signal is asserted only when all pending transactions and interrupt requests are treated. This is the best approach to efficient system power management.
11	Smart-idleWakeup	The module behaves like in Smart-idle mode, with the exception, that it can issue a wake-up request in sleep mode, if the functional clock is not cut off.

12.9.3.3.2.1 Wake-Up Capability

Note

Some of the timers features described in this section may not be supported on this family of devices. For more information, see *Timers Not Supported Features*.

If the TIMER_TIOCP_CFG[3-2] IDLEMODE bit field sets the smart-idle mode or smart-idle with wake-up mode, the timer evaluates its internal capability to have the interface clock switched off. When there is no further internal activity (no pending interrupt sources: match, overflow, or timer capture events), the clock stop acknowledge signal is asserted and the timer enters sleep mode, ready to issue a wake-up request if configured in smart-idle with wake-up mode.

Figure 12-482 shows the wake-up request generation.

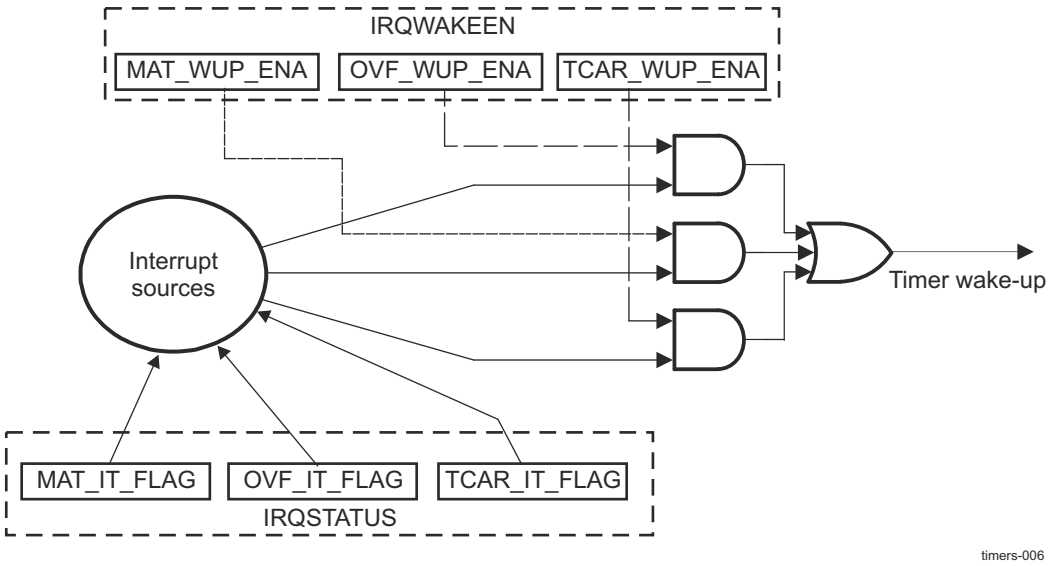


Figure 12-482. Wake-Up Request Generation

The timer wake-up-enable register allows masking of the expected source of the wake-up event that generates a wake-up request. The register is synchronously programmed with the interface clock before the Clock magagemet sends a clock stop request. The expected source of the wake-up event is an overflow (TIMER_TCRR), a timer match (the compare result of TIMER_TCRR and TIMER_TMAR matches the counter value), and a timer capture (detection of an external pulse transition of the correct polarity on the TIMER_PIEVENTCAPT).

When the wake-up event is issued, the associated interrupt status bit is set in the timer status register (TIMER_IRQSTATUS). The pending wake-up event is reset when the set status bit is overwritten with 1.

Note

The status bit must be reset to re-enter idle mode.

Copyright © 2024 Texas Instruments Incorporated

12.9.3.3.3 Timer Software Reset

TIMER_TIOCP_CFG[0] SOFTRESET bit can initiate a software reset of the timer. This bit is autocleared to 0 when the reset is complete.

Before accessing or using the timer, the local host must ensure that internal reset is released by reading the TIMER_TIOCP_CFG[0] SOFTRESET bit. This bit monitors the internal reset status.

12.9.3.3.4 Timer Interrupts

The timer can issue an overflow interrupt, a timer match interrupt, and a timer capture interrupt. Each internal interrupt source can be independently enabled and disabled in the interrupt-enable register (TIMER_IRQSTATUS_SET) and disabled in the interrupt-disable register (TIMER_IRQSTATUS_CLR). When the interrupt event is issued, the associated interrupt status bit is set in the timer status register (TIMER_IRQSTATUS).

12.9.3.3.5 Timer Mode Functionality

The timer is an upward counter that can be started and stopped at any time through the timer control register (the TIMER_TCLR[0] ST bit). The timer counter register (TIMER_TCRR) can be loaded when stopped or on-the-fly (while counting). TIMER_TCRR can be loaded directly by a TIMER_TCRR write access with a new timer value. TIMER_TCRR can also be loaded with the value held in the timer load register (TIMER_TLDR) by a trigger register (TIMER_TTGR) write access. The loading of TIMER_TCRR is done regardless of the written value of TIMER_TTGR. The value of TIMER_TCRR can be read when stopped or captured on-the-fly by a TIMER_TCRR read access. The timer is stopped and the counter value is set to 0 when the module reset is asserted. The timer is maintained at stop after the reset is released.

In one-shot mode (the TIMER_TCLR[1] AR bit is set to 0), the counter is stopped after counting overflow occurs (the counter value remains at 0).

When the autoreload mode is enabled (the TIMER_TCLR[1] AR bit is set to 1), TIMER_TCRR is reloaded with the value of TIMER_TLDR after a counting overflow occurs.

> **CAUTION**
>
> Do not put the overflow value (0xFFFF FFFF) in the TIMER_TLDR register because it can lead to undesirable results.

An interrupt can be issued on overflow if the overflow interrupt-enable bit is set in the timer interrupt-enable register (the TIMER_IRQSTATUS_SET[1] OVF_EN_FLAG bit is set to 1). A dedicated output pin (POTIMERPWM) can be programmed in the TIMER_TCLR[12] PT bit through the TIMER_TCLR[11-10] (PT and TRG bits) to generate one positive pulse (prescaler duration) or to invert the current value (toggle mode) when an overflow occurs. The TIMER_TCLR[12] PT bit selects pulse/toggle modulation (the TIMER_TCLR[11-10] TRG bit field selects trigger mode).

Figure 12-483 shows the TIMER_TCRR timing value.

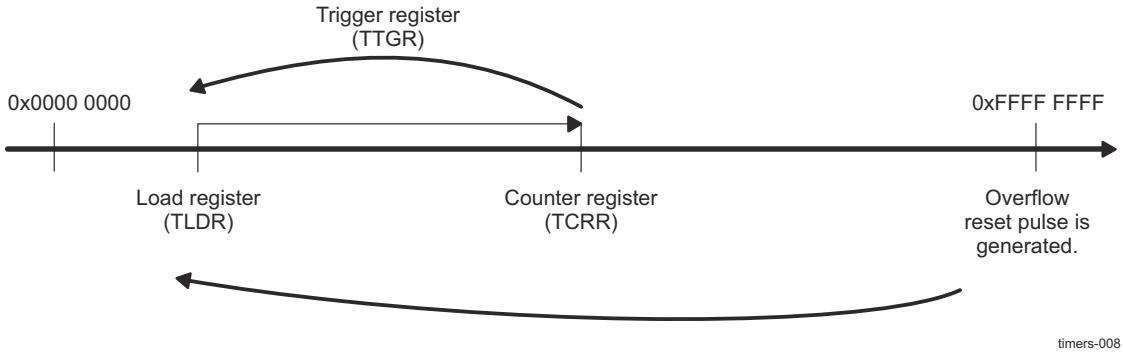


Figure 12-483. TIMER_TCRR Timing Value

Copyright © 2024 Texas Instruments Incorporated

12.9.3.3.5.1 1-ms Tick Generation

The interrupt period is not exactly 1 ms, because the timer input clock is 32.768 kHz. If the clock counts up to 32, it obtains a 0.977-ms period; if it counts up to 33, it obtains a 1.007-ms period. For large granularity, the error is cumulative and can generate important deviations from the standard value.

To minimize the error between a true 1-ms tick and the tick generated by the 32.768-kHz timer, the sequencing of periods less than 1 ms and periods greater than 1 ms must be shuffled. An additional 1-ms block is used to correct this error. See Figure 12-484.

In this implementation, the increment sequencing is automatically managed by the timer to minimize the error. The user must define only the value of the timer positive increment register (the TIMER_TPIR[31-0] POSITIVE_INC_VALUE bit field) and the timer negative increment register (the TIMER_TNIR[31-0] NEGATIVE_INC_VALUE bit field). An automatic adaptation mechanism is used to simplify the programming model.

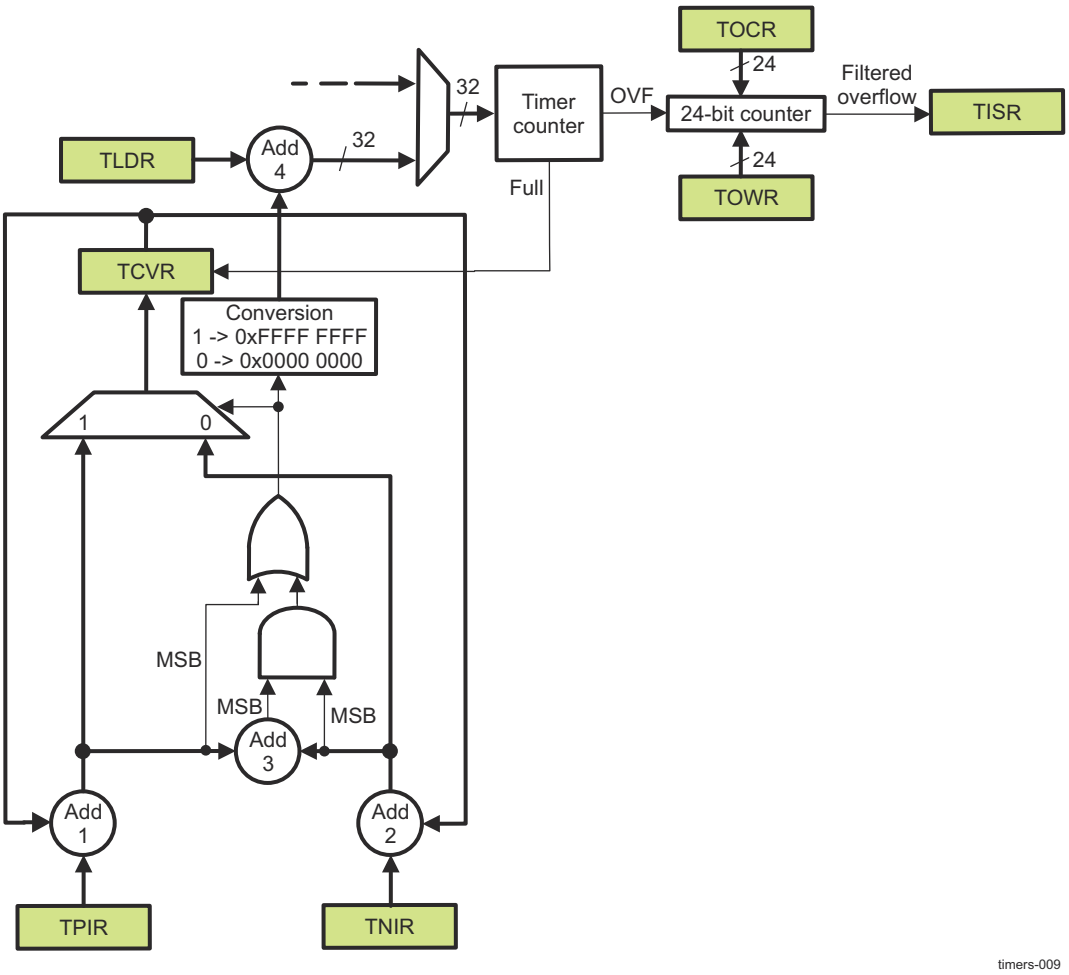


Figure 12-484. Block Diagram of the 1-ms Tick Module

The TIMER_TPIR, TIMER_TNIR, and TIMER_TCVR registers and adders Add1, Add2, and Add3 are used to define whether the next value loaded in the timer counter register (the TIMER_TCRR[31-0] TIMER_COUNTER bit field) is the value of the TIMER_TLDR[31-0] LOAD_VALUE bit field (period less than 1 ms) or the value of TIMER_TLDR[31-0] LOAD_VALUE –1 (period greater than 1 ms).

Table 12-419 lists the value loaded in the TIMER_TCRR according to the sign of the result of Add1, Add2, and Add3.

MSB = 0: Positive value; MSB = 1: Negative value

Copyright © 2024 Texas Instruments Incorporated

Table 12-419. Value Loaded in TIMER_TCRR to Generate 1-ms Tick

Add1 MSB	Add2 MSB	Add3 MSB	Value of TIMER_TCRR Register
0	0	0	TIMER_TLDR[31-0] LOAD_VALUE bit field
0	0	1	TIMER_TLDR[31-0] LOAD_VALUE bit field
0	1	0	TIMER_TLDR[31-0] LOAD_VALUE bit field
0	1	1	TIMER_TLDR[31-0] LOAD_VALUE –1
1	0	0	N/A
1	0	1	N/A
1	1	0	TIMER_TLDR[31-0] LOAD_VALUE –1
1	1	1	TIMER_TLDR[31-0] LOAD_VALUE –1

The values of the TIMER_TPIR and TIMER_TNIR registers are calculated using the following formulas:

- Positive increment value = (INTEGER[F_{clk} × T_{tick}] + 1) × 1e6) – (F_{clk} × T_{tick} × 1e6)
- Negative increment value = (INTEGER[F_{clk} × T_{tick}] × 1e6) – (F_{clk} × T_{tick} × 1e6)

Note

F_{clk} clock frequency (kHz)

T_{tick} tick period (ms)

The timer overflow counter register (TIMER_TOCR) and the timer overflow wrapping register (TIMER_TOWR) are used to filter interrupts. When the timer overflows, it increments the 24-bit TIMER_TOCR. When the values in the 24-bit TIMER_TOCR match the values in the 24-bit TIMER_TOWR and the timer overflow is asserted, the TIMER_TOCR is reset and an interrupt is generated to the TIMER_IRQSTATUS register.

Note

TIMER_TOWR has to be set to requested value. For example, if no interrupt needs to be masked TIMER_TOWR must be set to 0, if one interrupt needs to be masked TIMER_TOWR must be set to 1, if two interrupts need to be masked TIMER_TOWR must be set to 2 and so on.

It is important to have in mind that the case when FFFFFF interrupts need to be masked is not possible.

With the conversion block in reset state (the positive increment register, negative increment register, and counter value register are zeroed), the programming model and the behavior of timers remain unchanged.

For 1-ms tick with a 32.768-kHz clock:

- TIMER_TPIR[31-0] POSITIVE_INC_VALUE = 232,000
- TIMER_TNIR[31-0] NEGATIVE_INC_VALUE = –768,000
- TIMER_TLDR[31-0] LOAD_VALUE = 0xFFFF FFE0

Note

Any value of the tick period can be generated with the appropriate value of the TIMER_TPIR, TIMER_TNIR, and TIMER_TLDR.

By default, the TIMER_TPIR, TIMER_TNIR, TIMER_TCVR, TIMER_TOCR, and TIMER_TOWR and the associated logic are in reset mode (all 0s) and have no effect on the programming model.

12.9.3.3.6 Timer Capture Mode Functionality

When a transition is detected on the module input pin (PIEVENTCAPT), the timer value in the TIMER_TCRR can be captured and saved in the TIMER_TCAR1 or TIMER_TCAR2 register function of the mode selected in the TIMER_TCLR[13] CAPT_MODE bit. The edge detection circuitry monitors transitions on the input pin (PIEVENTCAPT).

Copyright © 2024 Texas Instruments Incorporated

The rising edge, falling edge, or both, can be selected in the TIMER_TCLR[9-8] TCM bit field to trigger the timer counter capture. The module sets the TIMER_IRQSTATUS[2] TCAR_IT_FLAG bit when an active edge is detected, and at the same time, the counter value TIMER_TCRR is stored in timer capture register TIMER_TCAR1 or TIMER_TCAR2, as follows:

- If the TIMER_TCLR[13] CAPT_MODE bit is 0, on the first enabled capture event, the value of the counter register is saved in the TIMER_TCAR1 register, and the next events are ignored (no update on the TIMER_TCAR1 register and no interrupt triggering) until the detection logic is reset or the TIMER_IRQSTATUS[2] TCAR_IT_FLAG is cleared by writing 1 to it.
- If the TIMER_TCLR[13] CAPT_MODE bit is 1, on the first enabled capture event, the value of the counter register is saved in the TIMER_TCAR1 register, and on the second enabled capture event, the value of the counter register is saved in the TIMER_TCAR2 register. If a capture interrupt is enabled, the interrupt triggers on the second event capture. All other events are ignored (no update on TIMER_TCAR1/TIMER_TCAR2 and no interrupt triggering) until the detection logic is reset or the TIMER_IRQSTATUS[2] TCAR_IT_FLAG bit is cleared by writing 1 to it. This mechanism is useful for period calculation of a clock, if that clock is connected to the PIEVENTCAPT input pin.

The edge detection logic is reset (a new capture is enabled) when the active capture interrupt is served. The TIMER_IRQSTATUS[2] TCAR_IT_FLAG bit is cleared by writing 1 to it or when the edge detection mode bits (the TIMER_TCLR[9-8] TCM bit field) are changed from no-capture mode detection to any other mode. The timer functional clock (input to prescaler) is used to sample the input pin (PIEVENTCAPT). A negative or positive pulse input can be detected when the pulse time is greater than the functional clock period. An interrupt is issued on edge detection if the capture interrupt-enable bit is set in the TIMER_IRQSTATUS_SET[2] TCAR_EN_FLAG bit. See the examples in Figure 12-485 and Figure 12-486.

In Figure 12-485, the value of the TIMER_TCLR[9-8] TCM bit field is 1h, and the TIMER_TCLR[13] CAPT_MODE bit is 0. Only the rising edge of PIEVENTCAPT triggers a capture in the TIMER_TCAR1 and TIMER_TCAR2 registers, and only the TIMER_TCAR1 register updates.

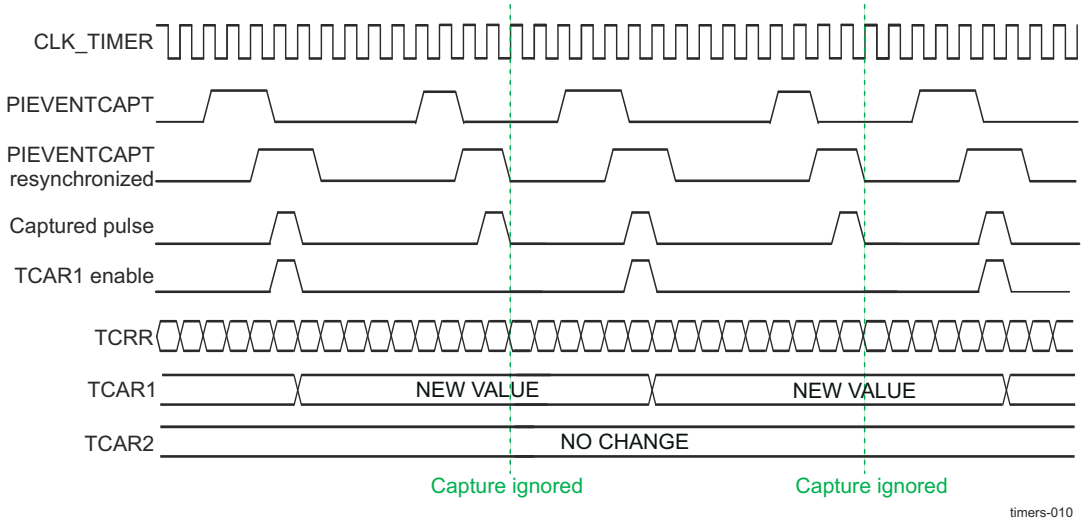


Figure 12-485. Capture Wave Example for TIMER_TCLR[13] CAPT_MODE = 0

In Figure 12-486, the value of the TIMER_TCLR[9-8] TCM bit field is 1h, and the TIMER_TCLR[13] CAPT_MODE bit is 1. Only the rising edge of PIEVENTCAPT triggers a capture in the TIMER_TCAR1 register on the first enabled event, and the TIMER_TCAR2 register updates on the second enabled event.

Copyright © 2024 Texas Instruments Incorporated

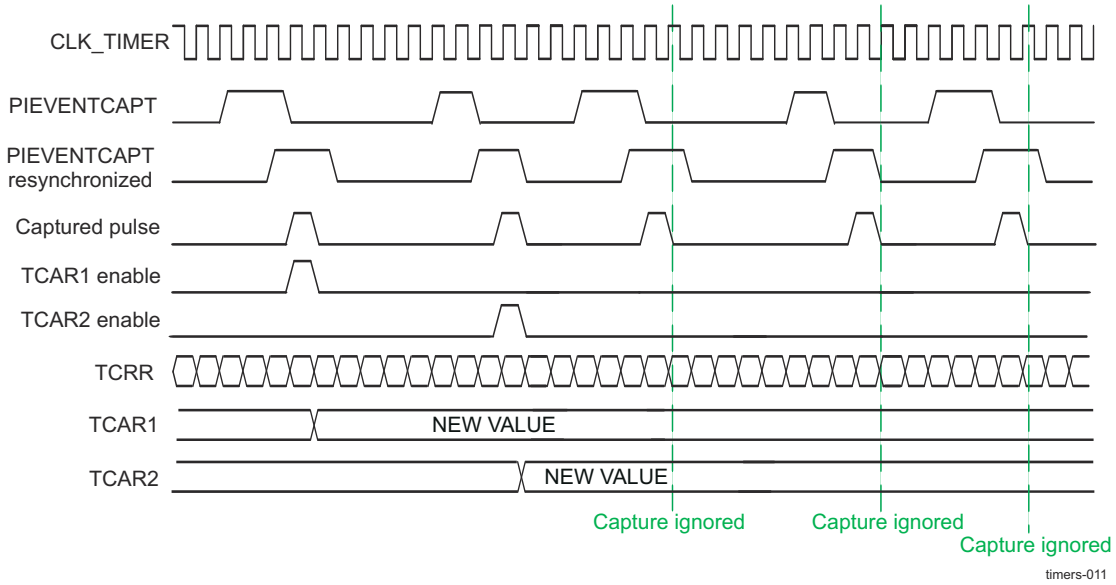


Figure 12-486. Capture Wave Example for TIMER_TCLR[13] CAPT_MODE = 1

12.9.3.3.7 Timer Compare Mode Functionality

When the compare-enable register TIMER_TCLR[6] CE bit is set to 1, the timer value (the TIMER_TCRR[31-0] TIMER_COUNTER bit field) is continuously compared to the value held in the timer match register (TIMER_TMAR). The value of the TIMER_TMAR[31-0] COMPARE_VALUE bit field can be loaded at any time (timer counting or stopped). When the TIMER_TCRR and the TIMER_TMAR values match, an interrupt is issued, if the TIMER_IRQSTATUS_SET[0] MAT_EN_FLAG bit is set.

To prevent any unwanted interrupts due to reset value matching effect, write a compare value to the TIMER_TMAR before setting the TIMER_TCLR[6] CE bit.

The dedicated output pin (POTIMERPWM) can be programmed in the TIMER_TCLR[12] PT bit through the TIMER_TCLR[11-10] TRG bit field to generate one positive pulse (timer clock duration) or to invert the current value (toggle mode) when an overflow or a match occurs.

12.9.3.3.8 Timer Prescaler Functionality

A prescaler can be used to divide the timer counter input clock frequency. The prescaler is enabled when the TIMER_TCLR[5] PRE bit is set. The TIMER_TCLR[4-2] PTV bit field sets the 2^n division ratio (prescaler value is $2^{(PTV + 1)}$. The prescaler counter is reset when the timer counter is stopped or reloaded on-the-fly.

Table 12-420 lists the prescaler/timer reload values versus contexts.

Table 12-420. Prescaler/Timer Reload Values Versus Contexts

Context	Prescaler	Timer Counter
Overflow (when autoreload is on)	Reset	TIMER_TLDR[31-0]
TIMER_TCRR write	Reset	TIMER_TCRR[31-0]
TIMER_TTGR write	Reset	TIMER_TLDR[31-0]
Stop	Reset	Frozen

12.9.3.3.9 Timer Pulse-Width Modulation

The timer can be configured to provide a programmable PWM output. The timer PWM (POTIMERPWM) output pin can be configured to toggle on an event. The TIMER_TCLR[11-10] TRG bit field determines on which register value the PWM pin toggles. Either overflow or both overflow and match can be selected to toggle the timer PWM pin when a compare condition occurs.

Copyright © 2024 Texas Instruments Incorporated

Note

In toggle mode, when TIMER_TCLR[11-10] TRG = 0x2 (overflow and match), the first event that toggles the PWM line is an overflow event.

The TIMER_TCLR[7] SCPWM bit can be programmed to set or clear the timer PWM output signal only while the counter is stopped or the trigger is off. This allows setting the output pin to a known state before modulation starts. Modulation synchronously stops when the TIMER_TCLR[11-10] TRG bit field is cleared and overflow occurs. This allows fixing a deterministic state of the output pin when modulation stops.

In Figure 12-487, the internal overflow pulse is set each time the (0xFFFF FFFF – TIMER_TLDR[31-0] LOAD_VALUE + 1) value is reached, and the internal match pulse is set when the counter reaches the value of TIMER_TMAR. Depending on the value of the TIMER_TCLR[12] PT bit and TIMER_TCLR[11-10] TRG bit field, the timer provides pulse or PWM event on the output pin (POTIMERPWM).

The TIMER_TLDR and TIMER_TMAR must keep values below the overflow value (0xFFFF FFFF) by at least two units. If the PWM trigger events are both overflow and match, the difference between the values kept in the TIMER_TMAR and the value in the TIMER_TLDR must be at least two units. When match event is used, the compare mode TIMER_TCLR[6] CE bit must be set.

In Figure 12-487, the TIMER_TCLR[7] SCPWM bit is set to 0. In Figure 12-488, the TIMER_TCLR[7] SCPWM bit is set to 1. To obtain the desired wave form, start the counter at 0xFFFF FFFE value (to ensure an overflow first) or adjust the line polarity (TIMER_TCLR[7] SCPWM bit).

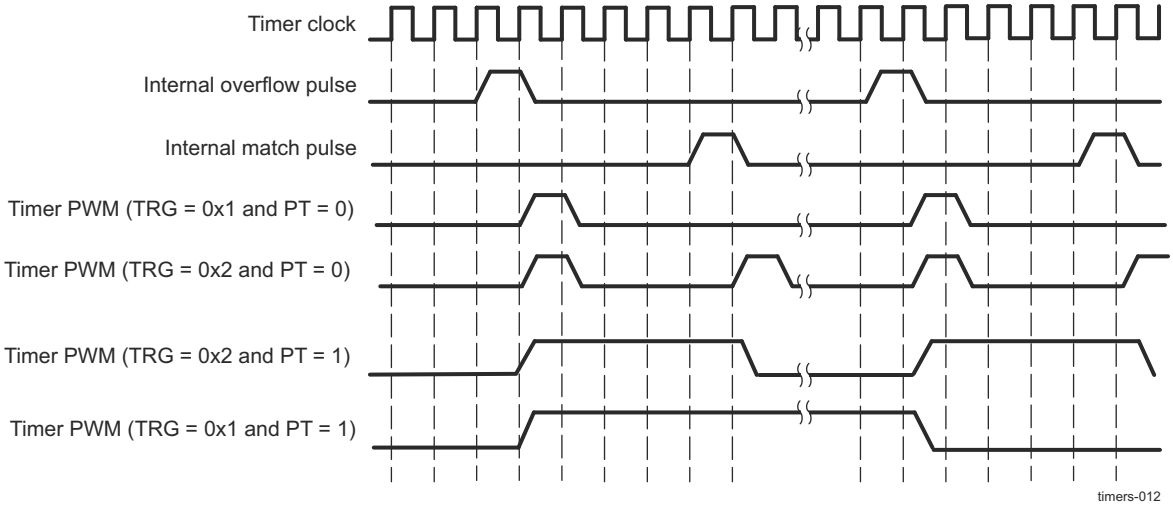


Figure 12-487. Timing Diagram of PWM With TIMER_TCLR[7] SCPWM Bit = 0

Copyright © 2024 Texas Instruments Incorporated

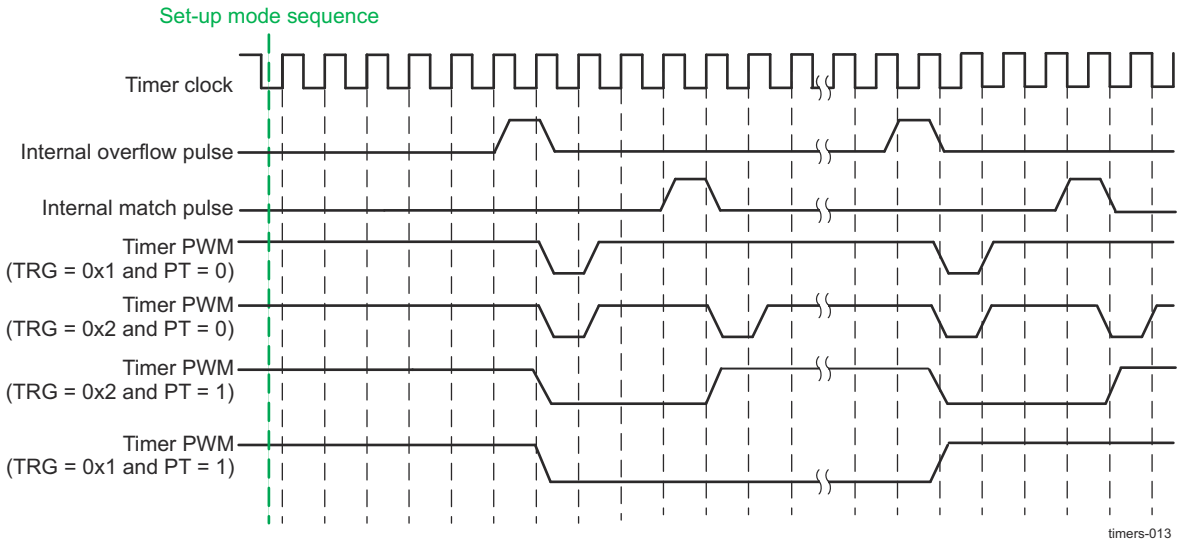


Figure 12-488. Timing Diagram of PWM With TIMER_TCLR[7] SCPWM Bit = 1

12.9.3.3.10 Timer Counting Rate

The timer rate is defined by the following values:

- Value of the prescaler fields (the TIMER_TCLR[5] PRE bit and TIMER_TCLR[4-2] PTV bit field)
- Value loaded into the TIMER_TLDR

Table 12-421 lists the prescaler clock ratio values.

Table 12-421. Prescaler Clock Ratio Values

TIMER_TCLR[5] PRE	TIMER_TCLR[4-2] PTV	Divisor (PS)
0	X	1
1	0	2
1	1	4
1	2	8
1	3	16
1	4	32
1	5	64
1	6	128
1	7	256

Thus, the timer overflow rate is expressed as:

OVF_Rate = (0xFFFF FFFF – TIMER_TLDR + 1) × (timer-functional clock period) × PS

With (timer-functional clock period) = 1/(timer-functional clock frequency) and PS = $2^{(PTV + 1)}$ if prescaler is enabled, or PS = 1 if prescaler is disabled.

CAUTION

Internal resynchronization causes any write to the TIMER_TCLR[1] ST bit to have some latency before the register is updated:

2.5 × functional clock cycles write_TIMER_TCLR_latency 3.5 × functional clock cycles

Remember to consider this latency whenever the timer must be started or stopped by a software change to the TIMER_TCLR[1] ST bit.

Copyright © 2024 Texas Instruments Incorporated

CAUTION

- In non-PWM mode, TIMER_TLDR must be maintained at less than or equal to 0xFFFF FFFE.
- In PWM mode, TIMER_TLDR must be maintained at less than or equal to 0xFFFF FFFD.

For example, with a timer clock input of 32 kHz and the TIMER_TCLR[5] PRE bit set to 0, the timer output period is as listed in Table 12-422.

Table 12-422. Value and Corresponding Interrupt Period

TIMER_TLDR[31-0] LOAD_VALUE	Interrupt Period
0x0000 0000	37 h
0xFFFF 0000	2 s
0xFFFF FFF0	500 µs
0xFFFF FFFE	62.5 µs

12.9.3.3.11 Timer Under Emulation

During emulation mode, the timer continues to run according to the value of the TIMER_TIOCP_CFG[1] EMUFREE bit.

If the TIMER_TIOCP_CFG[1] EMUFREE bit is set to 1, timer execution is not stopped in emulation mode and the interrupt is still generated when overflow or match is reached.

If the TIMER_TIOCP_CFG[1] EMUFREE bit is set to 0, the prescaler and timer are frozen and both resume on exit from emulation mode. The asynchronous external input pin (MCU_TIMER_IO[9-0] or TIMER_IO[7-0]) is internally synchronized on two timer-clock rising edges.

12.9.3.3.12 Accessing Timer Registers

All accesses are posted mode, under the assumption that freq(timer clock) < freq(interface clock)/4, until software reconfiguration. In addition, it is not recommended to access the timer registers prior to the PLLs generating the timer and interface clocks have been configured and the clocks have stabilized. All registers are 32 bits wide, accessible through the configuration interface with 32-bit access (read/write).

Write operations to the following functional registers must be complete (the MSB must be written even if the MSB data is not used):

- TIMER_TCLR
- TIMER_TCRR
- TIMER_TLDR
- TIMER_TTGR
- TIMER_TMAR
- TIMER_TPIR
- TIMER_TNIR
- TIMER_TCVR
- TIMER_TOCR
- TIMER_TOWR

The following registers are not affected by the posted/nonposted mode selection; the write/read operation is effective and acknowledged (command accepted) after one interface clock cycle from command assertion:

- TIMER_TIDR
- TIMER_TIOCP_CFG
- TIMER_IRQSTATUS
- TIMER_IRQSTATUS_RAW
- TIMER_IRQSTATUS_SET
- TIMER_IRQSTATUS_CLR
- TIMER_IRQWAKEEN
- TIMER_TWPS
- TIMER_TSICR

Copyright © 2024 Texas Instruments Incorporated

12.9.3.3.12.1 Writing to Timer Registers

The host uses the configuration interface to write to the following registers synchronously with the timer interface clock:

- TIMER_TLDR
- TIMER_TCRR
- TIMER_TCLR
- TIMER_TIOCP_CFG
- TIMER_IRQSTATUS
- TIMER_IRQSTATUS_SET
- TIMER_IRQSTATUS_CLR
- TIMER_IRQWAKEEN
- TIMER_TTGR
- TIMER_TSICR
- TIMER_TMAR
- TIMER_TPIR
- TIMER_TNIR
- TIMER_TCVR
- TIMER_TOCR
- TIMER_TOWR

12.9.3.3.12.1.1 Write Posting Synchronization Mode

This mode is used if the TIMER_TSICR[2] POSTED bit is set to 1 (default value).

This mode uses a posted write scheme to update any internal register (TIMER_TCLR, TIMER_TCRR, TIMER_TLDR, TIMER_TTGR, TIMER_TMAR, TIMER_TPIR, TIMER_TNIR, TIMER_TCVR, TIMER_TOCR, and TIMER_TOWR). Therefore, the write transaction is immediately acknowledged on the configuration interface, although the effective write operation occurs later because of a resynchronization in the timer clock domain. The advantage is that neither the interconnect, nor the device that requested the write transaction is stalled.

For each register, a status bit is provided in the timer write-posted status (TIMER_TWPS) register. In this mode, it is mandatory that software check this status bit before any write access. If a write is attempted to a register with a previous access pending, the previous access is discarded without notice.

The timer module updates the value of the timer counter register synchronously with the interface clock. Consequently, any read access to TIMER_TCRR does not add any resynchronization latency; the current value is always available.

Note

Because the overflow IRQ is generated when the value of TIMER_TCRR reaches 0xFFFF FFFF, and not when it changes its value to the value after overflow, it is necessary to wait a delay of (1 × PS × timer functional clock period) before any read access to TIMER_TCRR to ensure a correct reading of its content.

Note

If TIMER_TTGR register is written during a posted write to TIMER_TCRR, the value to be written to TIMER_TCRR will be discarded.

If a posted write to TIMER_TCVR is started, the user must not write to TIMER_TPIR or TIMER_TNIR before the TIMER_TCVR write is finished, because the value of TIMER_TCVR is re-evaluated, so both the value to be written, and the recalculated value will be discarded.

If a write access is pending for a register, reading from this register does not yield a correct result. Software synchronization must be used to avoid incorrect results.

Functional frequency range: freq(timer clock) < freq(interface clock)/4.

Copyright © 2024 Texas Instruments Incorporated

12.9.3.3.12.1.2 Write Nonposting Synchronization Mode

This mode is used if the TIMER_TSICR[2] POSTED bit is set to 0. It uses a nonposted write scheme to update any internal register. Therefore, the write transaction is not acknowledged on the configuration interface until the effective write operation occurs after the resynchronization in the timer functional clock domain. The drawback is that the interconnect and the device that requested the write transaction are stalled during this period.

The same full resynchronization scheme is used for a read transaction, and the same stall period applies. A register read following a write to the same register is always coherent.

This mode is functional regardless of the ratio between the configuration interface frequency and the timer clock frequency.

12.9.3.3.12.2 Reading From Timer Counter Registers

Note

LSB/MSB accesses cannot be interleaved (that is, the sequence LSB register 1, LSB register 2, MSB register 1, MSB register 2 is not supported).

The TIMER_TCRR is a 32-bit “atomic datum” and its 16-bit capture is done on the 16-bit LSB first to allow atomic LSB16 + MSB16 capture. This capture scheme is also performed for the TIMER_TCAR1 and TIMER_TCAR2 registers as they can be changed due to internal processes too.

12.9.3.3.12.2.1 Read Posted

This mode is functional regardless of the ratio between the configuration interface frequency and the functional clock frequency. The recommended functional frequency range is freq(timer) < freq(interface clock)/4.

Read posted mode is used if TIMER_TSICR[2] POSTED = 0x1 or TIMER_TSICR[3] READ_MODE is set to 0. This mode uses a posted-read scheme for reading any internal timer register. The read transaction is immediately acknowledged on the configuration interface, prior to the value to be read has been resynchronized. With this method, neither the interconnect nor the device that requested the read transaction are stalled.

Read posted mode applies to TIMER_TCRR, TIMER_TCAR1, TIMER_TCAR2, TIMER_TCVR, and TIMER_TOWR, which needs resynchronization from functional to interface clock domains.

Note that in Posted mode, if the TIMER_TCRR is read immediately after wake-up and the interface clock is off during idle state, then it is possible to get an old value from just before going to idle state due to the fact the interface clock is needed for synchronization.

In order to avoid this situation, another synchronization mechanism is used for the first read operation after idle state. The TIMER_TSICR[4] READ_AFTER_IDLE bit is used to enable/disable the mechanism.

When the synchronization mechanism is disabled (READ_AFTER_IDLE bit is set to 1), first read transaction takes only 2 interface clock cycles, but the read value of TIMER_TCRR could be wrong.

When the synchronization mechanism is enabled (READ_AFTER_IDLE bit is set to 0), first read value of TIMER_TCRR is correct, but the read transaction takes more than 2 interface clock cycles.

12.9.3.3.12.2.2 Read Non-Posted

This mode is functional regardless of the ratio between the configuration interface frequency and the functional clock frequency. Recommended functional frequency range is freq(timer) >= freq(interface clock)/4.

Read non-posted mode is used if TIMER_TSICR[2] POSTED = 0x0 and TIMER_TSICR[3] READ_MODE = 0x1. This mode uses a non-posted read scheme for reading internal timer registers. The read transaction is not acknowledged on the configuration interface until the effective read operation occurs, after the resynchronization in the timer clock domain. The result is that both the interconnect and the device that requested the read transaction are stalled during this period.

This mode applies to TIMER_TCRR, TIMER_TCAR1, TIMER_TCAR2, TIMER_TCVR, and TIMER_TOWR, which need resynchronization from functional to interface clock domains.

Copyright © 2024 Texas Instruments Incorporated

12.9.3.3.13 Timer Posted Mode Selection

A choice between two synchronization modes is made taking into account the frequency ratio and the stall periods that can be supported by the system, without impacting the global performance.

The posted mode selection applies only to registers that require synchronization on or from the timer clock domain. For write operation, the registers affected by posted and non-posted selection are TIMER_TCLR, TIMER_TLDR, TIMER_TCRR, TIMER_TTGR, TIMER_TMAR, TIMER_TPIR, TIMER_TNIR, TIMER_TCVR, TIMER_TOCR, and TIMER_TOWR. For read operation, the registers affected by this selection are: TIMER_TCRR, TIMER_TCAR1, TIMER_TCAR2, TIMER_TCVR, and TIMER_TOWR.

The interface clock domain synchronous registers TIMER_TIDR, TIMER_TIOCP_CFG, TIMER_IRQSTATUS, TIMER_IRQSTATUS_SET, TIMER_IRQWAKEEN, TIMER_TWPS, and TIMER_TSICR are not affected by posted and non-posted mode selection. The operation (read or write) is effective and acknowledged after one interface clock cycle from the command assertion.

The configuration of posted or non-posted mode can be changed (overwritten) by software by writing in TIMER_TSICR[2] POSTED bit. The TIMER_TSICR[3] READ_MODE defines how the read operation is performed when the module is configured in non-posted mode (see TIMER_TSICR). The following cases are possible:

- TIMER_TSICR[2] POSTED = 0x1 and TIMER_TSICR[3] READ_MODE = x (don’t care): read and write operations are expected in posted mode.
- TIMER_TSICR[2] POSTED = 0x0 and TIMER_TSICR[3] READ_MODE = 0x0: the write operation is executed in non-posted mode and read is executed in posted mode.
- TIMER_TSICR[2]POSTED = 0x0 and TIMER_TSICR[3] READ_MODE = 0x1: write is executed in non-posted mode and read is executed in non-posted mode.

Copyright © 2024 Texas Instruments Incorporated

12.9.3.4 Timers Low-Level Programming Models

This section describes the low-level hardware programming sequences for the configuration and use of the module.

12.9.3.4.1 Timer Operational Mode Configuration

12.9.3.4.1.1 Timer Mode

12.9.3.4.1.1.1 Main Sequence – Timer Mode Configuration

Table 12-423 lists the steps in the timer mode configuration.

Table 12-423. Timer Mode Configuration

Step	Register/Bit Field/Programming Model	Value
Select autoreload mode.	TIMER_TCLR[1] AR	0x-
Set prescale timer value.	TIMER_TCLR[4-2] PTV	0x-
Enable prescaler.	TIMER_TCLR[5] PRE	0x1
Enable overflow interrupt.	TIMER_IRQSTATUS_SET[1] OVF_EN_FLAG	0x1
Load timer counter value.	TIMER_TCRR	0x-
Load timer load value.	TIMER_TLDR	0x-
Start the timer.	TIMER_TCLR[0] ST	0x1

12.9.3.4.1.2 Timer Compare Mode

12.9.3.4.1.2.1 Main Sequence – Timer Compare Mode Configuration

Table 12-424 lists the steps in the timer compare mode configuration.

Table 12-424. Timer Compare Mode Configuration

Step	Register/Bit Field/Programming Model	Value
Select autoreload mode.	TIMER_TCLR[1] AR	0x-
Set prescale timer value.	TIMER_TCLR[4-2] PTV	0x-
Enable prescaler.	TIMER_TCLR[5] PRE	0x1
Enable match interrupt.	TIMER_IRQSTATUS_SET[0] MAT_EN_FLAG	0x1
Load timer counter value.	TIMER_TCRR	0x-
Load timer compare value.	TIMER_TMAR	0x-
Enable compare mode.	TIMER_TCLR[6] CE	0x1
Start the timer.	TIMER_TCLR[0] ST	0x1

12.9.3.4.1.3 Timer Capture Mode

12.9.3.4.1.3.1 Main Sequence – Timer Capture Mode Configuration

Table 12-425 lists the steps in the timer capture mode configuration.

Table 12-425. Timer Capture Mode Configuration

Step	Register/Bit Field/Programming Model	Value
Initialize capture mode.	See Section 12.9.3.4.1.3.2.	
Enable capture interrupt.	TIMER_IRQSTATUS_SET[2] TCAR_EN_FLAG	0x1
Start the timer.	TIMER_TCLR[0] ST	0x1
Detect event.	See Section 12.9.3.4.1.3.3.	

12.9.3.4.1.3.2 Subsequence – Initialize Capture Mode

Table 12-426 lists the steps to initialize capture mode.

Table 12-426. Initialize Capture Mode

Step	Register/Bit Field/Programming Model	Value
Select autoreload mode.	TIMER_TCLR[1] AR	0x-
Set prescale timer value.	TIMER_TCLR[4-2] PTV	0x-

Copyright © 2024 Texas Instruments Incorporated

Table 12-426. Initialize Capture Mode (continued)

Step	Register/Bit Field/Programming Model	Value
Enable prescaler.	TIMER_TCLR[5] PRE	0x1
Select TIMER[19-0] or MCU_TIMER[9-0] Capture input at device pins TIMER_IO[7-0] for TIMER[19-0] or at pins MCU_TIMER_IO[9-0] for MCU_TIMER[9-0].	TIMER_TCLR[14] GPO_CFG	0x1
Select single or second event capture.	TIMER_TCLR[13] CAPT_MODE	0x-
Select transition capture mode.	TIMER_TCLR[9-8] TCM	0x-

12.9.3.4.1.3.3 Subsequence – Detect Event

Table 12-427 lists the steps in detecting an event.

Table 12-427. Detect Event

Step	Register/Bit Field/Programming Model	Value
Wait until event detected?	TIMER_IRQSTATUS[2] TCAR_IT_FLAG	= 0x1
Read timer capture value.	TIMER_TCAR1 and/or TIMER_TCAR2	
Clear capture interrupt request.	TIMER_IRQSTATUS[2] TCAR_IT_FLAG	0x1

12.9.3.4.1.4 Timer PWM Mode

12.9.3.4.1.4.1 Main Sequence – Timer PWM Mode Configuration

Table 12-428 lists the steps in the timer PWM mode configuration.

Table 12-428. Timer PWM Mode Configuration

Step	Register/Bit Field/Programming Model	Value
Select autoreload mode.	TIMER_TCLR[1] AR	0x-
Set prescale timer value.	TIMER_TCLR[4-2] PTV	0x-
Enable prescaler.	TIMER_TCLR[5] PRE	0x1
Select trigger output mode.	TIMER_TCLR[11-10] TRG	0x-
Select pulse or toggle modulation PWM mode.	TIMER_TCLR[12] PT	0x-
Select TIMER[19-0] or MCU_TIMER[9-0] PWM output at device pins TIMER_IO[7-0] for TIMER[19-0] or at pins MCU_TIMER_IO[9-0] for MCU_TIMER[9-0].	TIMER_TCLR[14] GPO_CFG	0x0
Configure PWM output pin default value.	TIMER_TCLR[7] SCPWM	0x-
Load timer load value.	TIMER_TLDR	0x-
Load timer compare value.	TIMER_TMAR	0x-
Enable compare.	TIMER_TCLR[6] CE	0x1
Start the timer.	TIMER_TCLR[0] ST	0x1

Copyright © 2024 Texas Instruments Incorporated

12.10 Internal Diagnostics Modules

This section describes the internal diagnostics modules in the device.

12.10.1 Dual Clock Comparator (DCC)

This section describes the Dual Clock Comparator (DCC) modules in the device.

12.10.1.1 DCC Overview

The Dual Clock Comparator (DCC) is used to determine the accuracy of a clock signal during the time execution of an application. Specifically, the DCC is designed to detect drifts from the expected clock frequency. The desired accuracy can be programed based on calculation for each application. The DCC measures the frequency of a selectable clock source using another input clock as a reference.

12.10.1.1.1 DCC Features

The DCC uses two independent clock sources to detect when one is out of specification. Each DCC module implements the following features:

- Two independent counter blocks count clock pulses from each clock source
- Each counter block is programmable, however, for proper operation the counters must be programmed with seed values that respect the ratio of the two clock frequencies
- Configurable timebase for error signal
- Error signal generation when one of the clocks is out of specification
- Clock frequency measurement

Unsupported Features:

- See the Module Integration section of this document for a list of module features not supported by the integration on this Device.

12.10.1.1.2 DCC Ports

Table 12-429. DCC Clocks and Resets

Clocks	
Module Clock Input	**Description**
DCC_FICLK	DCC interface and functional clock
Resets	
Module Reset Input	**Description**
DCC_RST	DCC asynchronous module reset

Table 12-430. DCC Hardware Requests

Interrupt Requests		
Module Interrupt Signal	**Description**	**Type**
DCC_INTR_DONE_LEVEL_0	DCC one-shot mode complete interrupt	Level
DCC_INTR_ERR_LEVEL_0	DCC error interrupt	Level

Copyright © 2024 Texas Instruments Incorporated

12.10.1.2 DCC Functional Description

The DCC module supports two selectable clock sources for Counter0 and Counter1. Figure 12-489 shows the DCC functional block diagram.

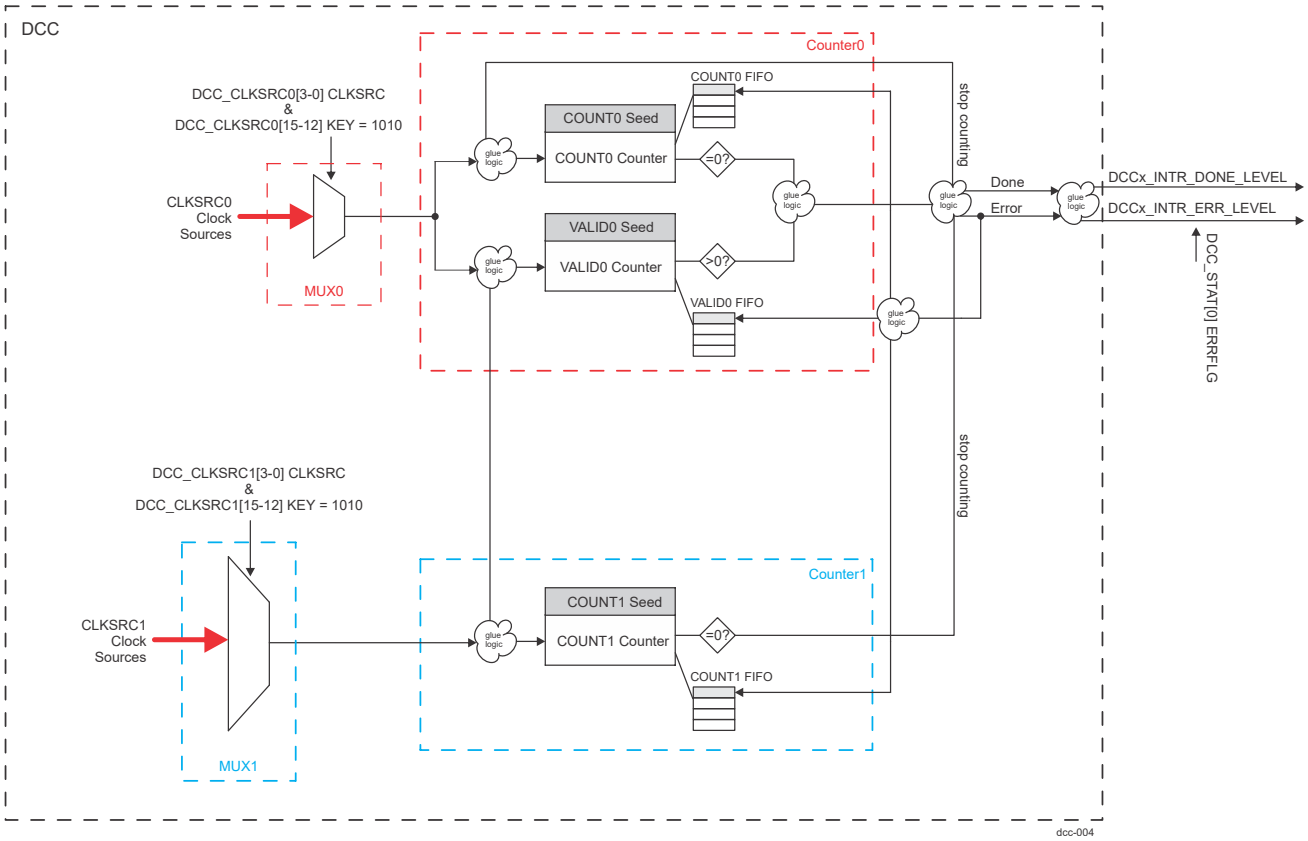


Figure 12-489. DCC Functional Block Diagram

12.10.1.2.1 DCC Counter Operation

Note

For detailed DCC compute calculations, refer to Continuous Monitor of the PLL Frequency With the DCC App Note.

The DCC has two parallel counters that count clock pulses for two independent clock sources:

- Counter0 generates a fixed-width counting window (VALID0) after a pre-programmed number of pulses (COUNT0). The values for VALID0 and COUNT0 can be programmed in DCC_VALIDSEED0 and DCC_CNTSEED0 registers respectively.
- Counter1 generates a fixed-width pulse (1 cycle) after a pre-programmed number of pulses (COUNT1). This pulse sets an error signal if Counter1 does not reach 0 during the time when VALID0 is running. The seed value for COUNT1 can be programmed in DCC_CNTSEED1 register.

The error signal is generated by any one of the following conditions:

- Clock1 expires before the COUNT0 reaches 0.
- Clock1 expires after both COUNT0 and VALID0 reach 0.
- Clock1 not present.
- Clock0 not present.

Copyright © 2024 Texas Instruments Incorporated

Any of these errors causes the counters to stop counting. An application must then read out the counter values to determine what caused the error. Once the error is detected, the counters are stopped after 3 FICLK and 2 source clock cycles due to the cross clock domain synchronisation.

Reloads or restarts occur under two conditions:

- The module is reset or restarted through software (that is, software starts the module after reset, or software checks an error condition and decides to restart the module).
- COUNT0, COUNT1, and VALID0 all reach 0 without error.

CAUTION

The DCC module does not check jitter for Clock0 or Clock1.

As the counter preset signal is synchronized to either of the source clock domains, the counters begin downcounting after two corresponding source clock cycles.

The error signal is captured to the FICLK domain. There is 1 FICLK period uncertainty on either side of the fixed width counting window (VALID0) in generating the error signal since the counters work in different clock domains. This should be accounted for when setting the count value for VALID0.

Figure 12-490 through Figure 12-494 shows examples of counters relationship and error generation.

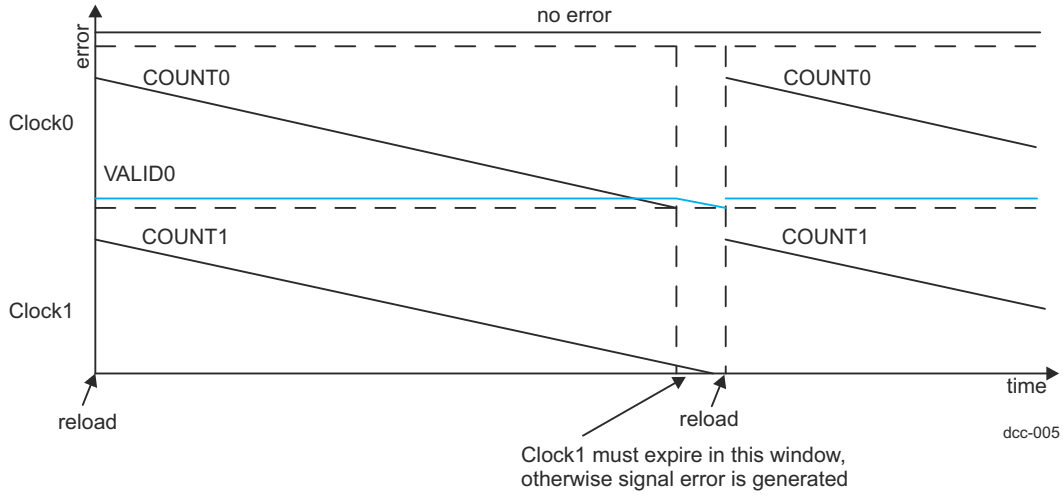


Figure 12-490. DCC Clock0 and Clock1 With no Error

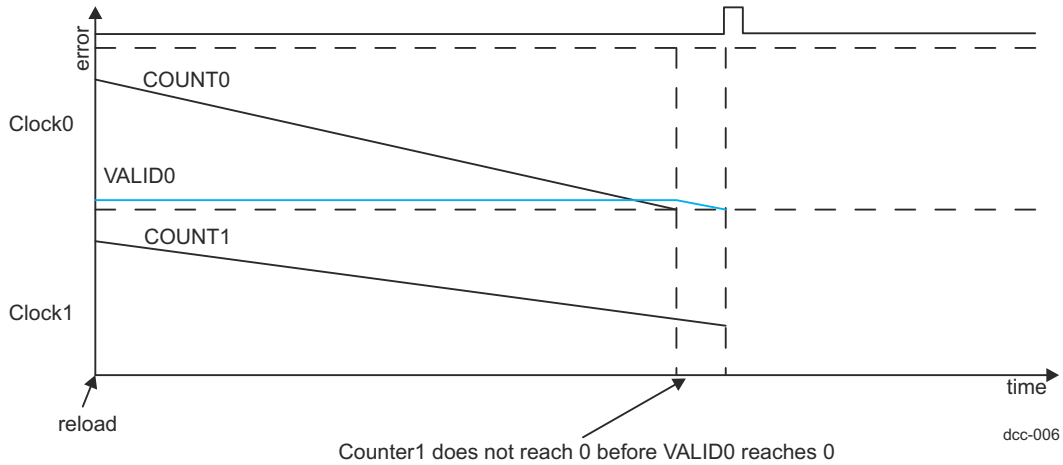


Figure 12-491. DCC Clock1 slower than Clock0 results in an error and stops counting

Copyright © 2024 Texas Instruments Incorporated

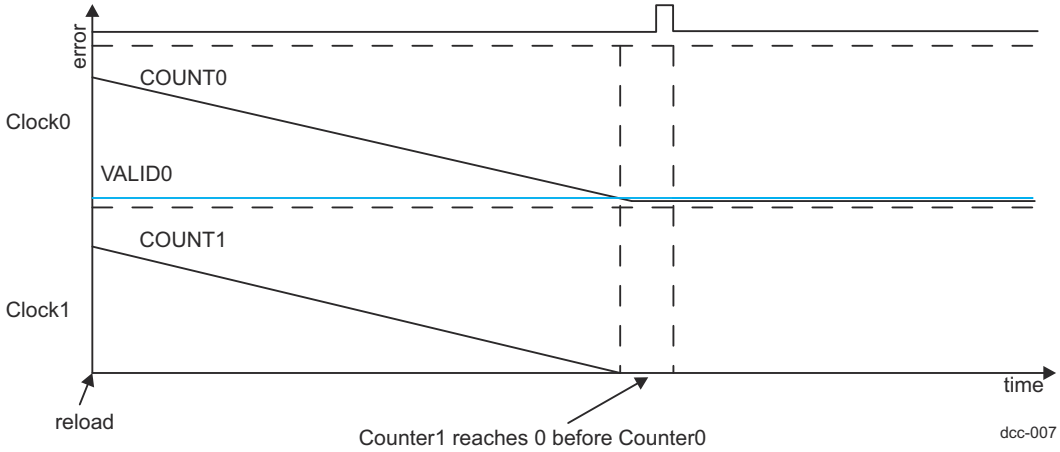


Figure 12-492. DCC Clock1 faster than Clock0 results in an error and stops counting

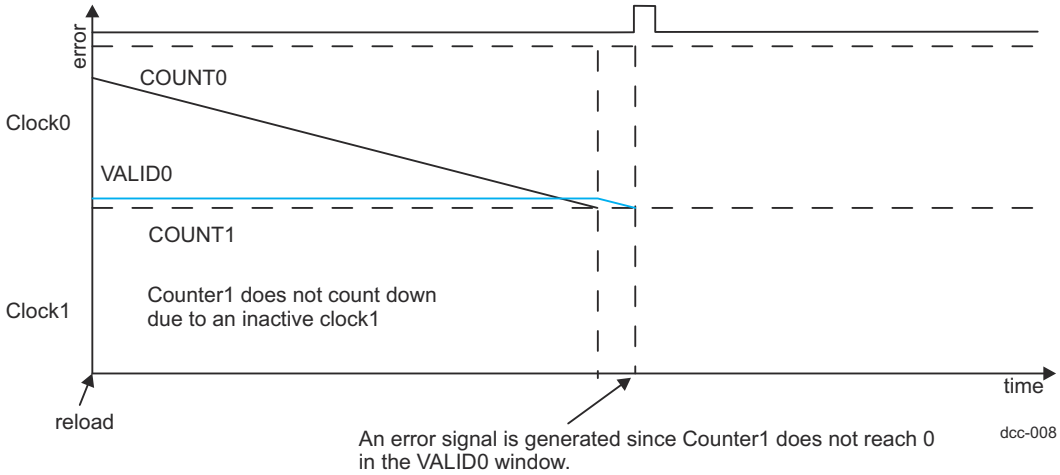


Figure 12-493. DCC Clock1 not present results in an error and stops counting

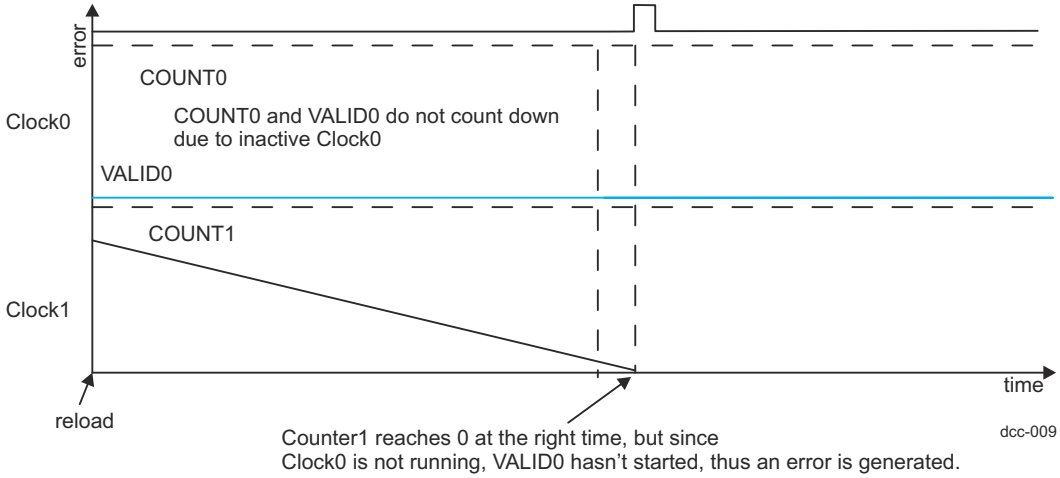


Figure 12-494. DCC Clock0 not present results in an error and stops counting

12.10.1.2.2 DCC Low Power Mode Operation

The DCC module does not function in Low Power Mode. It is the responsibility of software to stop the module before entering low power mode and to restart the module after exiting low power mode.

Copyright © 2024 Texas Instruments Incorporated

12.10.1.2.3 DCC Suspend Mode Behavior

The DCC module will continue running regardless of the state of emulation. All registers are readable via emulation reads.

12.10.1.2.4 DCC Single-Shot Mode

The DCC can be programmed to count down one time using single-shot mode. In this mode, the DCC stops operation when both COUNT0 and VALID0 reach 0.

At the end of one sequence in single-shot mode the DCC_GCTRL[3-0] DCCENA bitfield is set to disabled, which stops further counting. Single-shot mode is enabled from the DCC_GCTRL[11-8] SINGLESHOT bitfield.

At the end of one sequence in single-shot mode, if there is no error which stops counting, then the done status bit is set in the DCC_STAT[1] DONEFLG bitfield and a done interrupt DCCx_INTR_DONE_LEVEL is generated. Software must clear the done bit before restarting the counting.

12.10.1.2.5 DCC Continuous mode

When DCC runs in continuous mode both the counts shall get reloaded with seed value upon completion of counts without error. If the counts end in error DCC stops the operation and counts are not reloaded.

12.10.1.2.5.1 DCC Continue on Error

During debug, if there are events which are causing clocks to be anomalous over short period covering more than one evaluation window then it would be important to capture trajectory of error event and period around such event. To allow capturing the successive error events DCC can be programmed to continue after error. DCC_GCTRL2 [3-0] CONT_ON_ERR shall be set to value other than "0101" to enable this mode. It is recommended to write "1010" to avoid single soft errors.

12.10.1.2.5.2 DCC Error Count

DCC also counts the number of error pulses generated since reset or since last time the error count is cleared. This is read/write register for CPU to clear when new trace of number of errors is required to be maintained.

12.10.1.2.6 DCC Control and count hand-off across clock domains

As the counters run in two different selectable clock domains and the register interface runs on the fixed bus clock domain, control signals and counter value hand-off have synchronizers implemented. These add to the margins of error while comparing the counts.

1. With all three counts synchronized to FICLK domain for comparison error detection there is delay in terms of FICLK domain.
2. Based on the error signal, the enable for the counters is synchronized back to the respective clock domains of the counters, which adds latency in terms of clock periods which are different, this would create a skew between start of count. Depending upon frequency ratios of the clocks used, the difference between two could vary.

Application needs to consider the worst case delay differences while measuring the clocks.

12.10.1.2.7 DCC Error Trajectory record

Once the clock errors out, the host can read the counter values to determine the extent of error to analyze type of failure. For short window comparisons this would become difficult, specially if there are back to back errors due to some transient event. Secondly, for random events which can cause an interrupt during the critical phase of application running, then event if not recorded may get overwritten and also not provide meaningful trace of error.

12.10.1.2.7.1 DCC FIFO capturing for Errors

DCC provides the FIFO for capturing COUNT0, VALID0, and COUNT1 information which captures all three counts upon "Error" event. For "Done" event no results are captured by default.

12.10.1.2.7.2 DCC FIFO in continuous capture mode

To track the VALID0 counter values regardless of "Error" or not, FIFOs can be configured to capture the count for each compare window. This is useful in validation and characterization exercise. DCC_GCTRL2[11-7]

Copyright © 2024 Texas Instruments Incorporated

FIFO_NONERR control when set to value other than "0101" this mode is set; it is recommended to write "1010" to avoid single soft errors. Note, this capture is applicable only in continuous mode and not in single shot mode.

12.10.1.2.7.3 DCC FIFO Details

The FIFO is 4 deep for each count and updates new count information for all the non-full FIFOs. Information is updated on every configured trigger of error or cycle completion. If full, the next values are not written till at-least one entry is read. Application owns responsibility to read the FIFOs uniformly to keep synchronisation between three entries of the FIFO. Both empty and full indications for individual FIFOs is provided through the DCC_STATUS2.

12.10.1.2.7.4 DCC FIFO Debug mode behavior

Upon debug access, the FIFO pointers should not advance hence not impacting the functional behavior. This requirement is same as other functional blocks in the device.

12.10.1.2.8 DCC Count read registers

DCC has provision to read the counts during operation. This is performed using DCC_CNT0, DCC_VALID0, and DCC_CNT1 registers. Read from these registers in default mode allows reading the present value of count. This is useful when in single shot mode or mode where DCC stops upon error.

These registers can be used to read the FIFO through the DCC_GCTRL2[7-4] FIFO_READ configuration. Reads on the empty FIFO shall provide the contents of last pointed location. Application shall track the empty/full conditions of the FIFOs to track the count records consistently.

Regardless of FIFO_READ configuration, the FIFO internally keeps updating records based on configured triggers till full.

Copyright © 2024 Texas Instruments Incorporated

12.10.2 Error Signaling Module (ESM)

This section describes the Error Signaling Module (ESM) in the device.

12.10.2.1 ESM Overview

The Error Signaling Module (ESM) aggregates events and/or errors from throughout the device into one location. It can signal both low and high priority interrupts to a processor to deal with an event and/or manipulate an I/O error pin to signal an external hardware that an error has occurred. Therefore an external controller is able to reset the device or keep the system in a safe, known state.

12.10.2.1.1 ESM Features

Each ESM module implements the following features:

- Up to 1024 error event inputs
  - Implemented in groups of 32 events
  - Level or Pulse inputs (Pulse inputs are triple redundant)
- Selectable low and high priority interrupt error pin prioritization of each error event
- Error pin to signal severe device failure
  - Support of level or PWM modes
- Configurable timebase for error signal
- Error forcing capability
- Internal redundant flops on critical fields

Unsupported Features:

- See the Module Integration section of this document for a list of module features not supported by the integration on this Device.

12.10.2.1.2 ESM Ports

Table 12-431. ESM Clocks and Resets

Clocks	
Module Clock Input	**Description**
ESM_FICLK	ESM Interface and Functional clock
Resets	
Module Reset Input	**Description**
ESM_RST	ESM Asynchronous module reset
ESM_POR_RST	ESM Power-on module reset

Table 12-432. ESM Hardware Requests

Interrupt Requests		
Module Interrupt Signal	**Description**	**Type**
ESM_ESM_INT_CFG_LVL_0	ESM configuration error interrupt	Level
ESM_ESM_INT_LOW_LVL_0	ESM low priority interrupt	Level
ESM_ESM_INT_HI_LVL_0	ESM high priority interrupt	Level

12.10.2.2 ESM Environment

This section describes the ESM external connections (environment).

Table 12-433 describes the ESM I/O signals.

Table 12-433. ESM I/O Signals

Module Pin	I/O[(1)]	Description
ERR_O	O	Active low error output signal.

(1) O = Output;

Copyright © 2024 Texas Instruments Incorporated

12.10.2.3 ESM Functional Description

The Error Signaling Module (ESM) centralizes fault reports. It provides mechanisms to classify errors by severity and to provide programmable error response. The error classification in the ESM is determined by programmed configuration for each individual error input. For each individual error input the configuration can be set to assert an output error pin, or generate an interrupt to a CPU, or both. When an individual error input is configured to generate an interrupt, the configuration will also select whether the interrupt that is generated will be one of high priority or low priority.

By reporting the faults in a central location, the system may determine what caused the fault and what action can be taken. In general, the faults can be split into two categories:

- Corrected faults
- Non-corrected faults

The ESM reports errors in two ways:

- An interrupt to a processor in the device. This allows the device to analyze and try to recover from an error.
- An external ERROR pin. This allows the system outside of the SoC to monitor for potentially fatal errors(errors that the device cannot self-recover from). Moreover, the external I/O (ERROR pin) can operate in level or PWM modes. In level mode, the output will remain asserted (active low) for a minimum period of time. After that period of time, if the error has been cleared by an internal processor, the pin will go inactive (high). If it does not go inactive in that time, then an external agent should intervene, as there may be an unrecoverable error. In PWM mode, the error will cause the output pin to maintain its value for a minimum period of time. After that period of time, if the error has been cleared by an internal processor, the pin will continue the PWM pattern. If it does not go inactive in that time, then an external agent should intervene, as there may be an unrecoverable error.

Both mechanisms can be used at the same time for the same fault, signaling both an interrupt and the external ERROR pin. This allows the device to attempt to recover, but if it fails, then the external system is still alerted. If it succeeds, then it can remove the ERROR pin assertion so that the external system knows that a potentially unsafe condition was avoided.

Lastly, the ESM does not specify any methods of intervention, only the process of alerting internal CPUs and external monitor(s) of an existing error event.

Figure 12-495 shows the ESM module block diagram. Note that not all instances may be pinned out in the device. For more information, see *ESM Environment*.

Copyright © 2024 Texas Instruments Incorporated

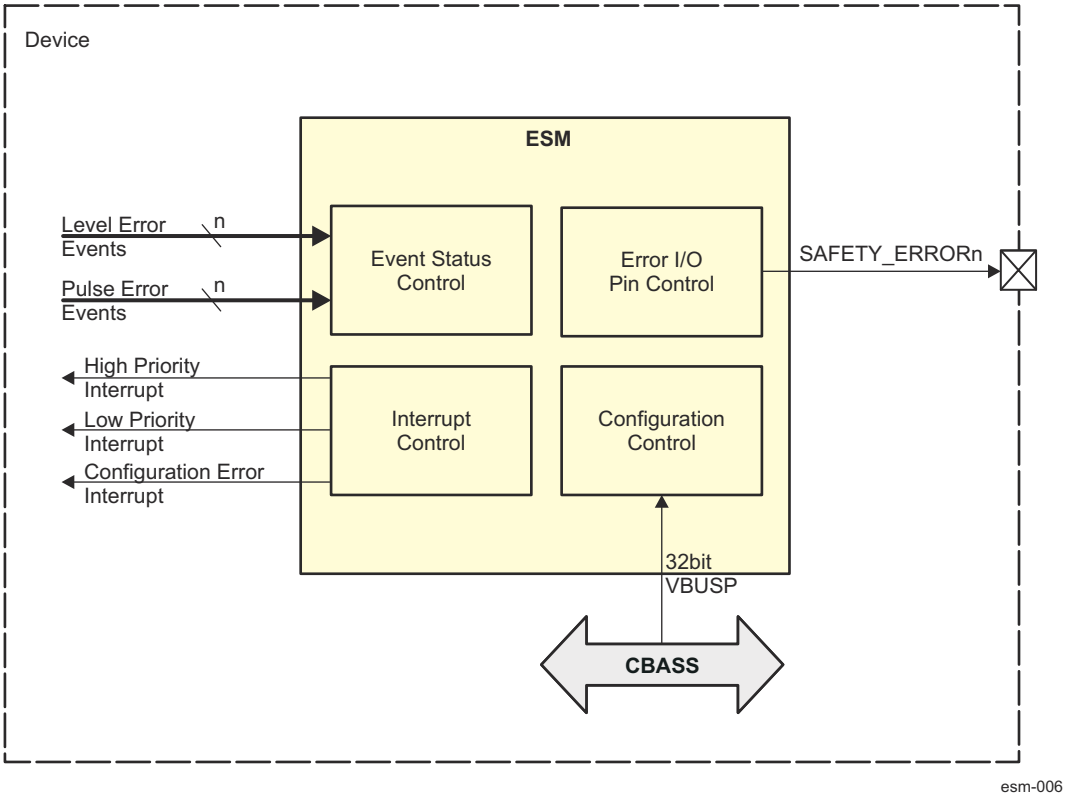


Figure 12-495. ESM Block Diagram

12.10.2.3.1 ESM Interrupt Requests

The ESM module generates three output interrupts to the device interrupt controllers:

- Configuration error interrupt (see Section 12.10.2.3.1.1)
- High priority error interrupt (see Section 12.10.2.3.1.2)
- Low priority error interrupt (see Section 12.10.2.3.1.3)

The error interrupt outputs are provided so that a processor in the device can be signaled to intervene when an error event occurs. Each error event input can be enabled, via software, to cause an error interrupt to occur (via the ESM_INTR_EN_SET_j register). Additionally, each error event input can be programmed to influence either the low priority (default) interrupt or the high priority interrupt (via the ESM_INT_PRIO_j). The low priority interrupt is intended for events that are of interest, but do not require immediate intervention. For example, an indication that there was a single bit error that was corrected may signal a low priority interrupt, so that information can be collected for statistical purposes. A high priority interrupt is intended for events that need immediate attention. For example, an indication that there was an uncorrected two-bit error may be signaled as a high priority interrupt.

12.10.2.3.1.1 ESM Configuration Error Interrupt

The configuration error interrupt (ESM_INT_CFG_LVL_0) indicates that there is an inconsistency in the configuration of one (or more) error group *j* registers (MMRs).

In such inconsistency in the internal copies of any of the MMRs caused by fault associated with error group *j*, the corresponding raw status will be set in the ESM_ERR_RAW register. If the corresponding bit is enabled in the ESM_ERR_EN_SET register, a configuration error interrupt will be triggered.

When a configuration error interrupt is received, the acting processor must perform the following steps:

1. Read the ESM_ERR_STS register to determine which group has a configuration error
2. Write the correct values to the following group registers:

Copyright © 2024 Texas Instruments Incorporated

Note

If there has been a configuration error, the values in the below registers cannot be guaranteed to be correct anymore. Therefore, software should maintain a copy of the correct values prior to an error to ensure that they can be re-programmed with the correct configuration.

a. ESM_INTR_EN_SET_j
b. ESM_INTR_EN_CLR_j
c. ESM_INT_PRIO_j
d. ESM_PIN_EN_SET_j
e. ESM_PIN_EN_CLR_j

3. Service any pending interrupts in steps 4, 5, and 6 below

Note

The raw status of any pending interrupts may be inconsistent. Servicing the interrupt will return it to consistency via the error group ESM_RAW_j register.

4. Write 0x1 to the appropriate bits in the ESM_ERR_STS register. This step will clear the raw status
   a. If the error event is still asserted (or re-asserted) the raw status will be set back to 0x1
   b. If there are no additional errors, the level interrupt will go low
5. Write the end of interrupt vector to the ESM_EOI register
   a. If there are additional configuration error interrupts pending, then a new pulse will be generated. The level interrupt will remain asserted and is unaffected by EOI.
   b. If there are no additional low priority error interrupts pending, there will be no new pulse.

12.10.2.3.1.2 ESM Low Priority Error Interrupt

Events mapped to the low priority error interrupt (ESM_INT_LOW_LVL_0) are intended to be events of interest that should be addressed eventually, not events that require immediate attention. An example would be an event indicating a corrected error. The system may want to track this for statistical purposes, but it does not require immediate attention.

Any error event can be mapped to the low priority error interrupt. A low priority error interrupt will be generated when an event is enabled to cause an interrupt (via the ESM_INTR_EN_SET_j register) and mapped to the low priority error interrupt (via ESM_INT_PRIO_j register) and the raw status is set (via the ESM_RAW_j register).

When a low priority error interrupt is received, the acting processor must perform the following steps:

1. Read the ESM_LOW_PRI register
   a. If both [31-16]PLS and [15-0]LVL bit fields are equal to 0xFFFF, then interrupt is no longer asserted and the interrupt routine has ended.
   b. If either [31-16]PLS or [15-0]LVL bit fields are not equal to 0xFFFF, software has two options for determining what event to service:
      i. First option: Record the value in [31-16]PLS and [15-0]LVL bit fields. Determine which is higher priority. This is based on the global event map number of the highest priority low priority error event

      Note

      The global event map is neither defined by nor maintain in ESM. The global event map must be defined and maintained by software using software determined memory resources. It is conceivable that the global event map will be predefined and loaded during the execution of a secondary boot loader.

      ii. Second option:
         1. Read the ESM_LOW register to determine which event group(s) have pending low priority error interrupts
         2. Read the desired error group *j* ESM_STS_j register
         3. Identify which low priority interrupt to service
2. Determine, based on the global event map for the device, where the error event came from

Copyright © 2024 Texas Instruments Incorporated

3. Service the error event based on the IP's specification:
   a. The system may take several actions including (but not limited to):
      i. Fixing the error
      ii. Resetting the errored IP peripheral
      iii. Resetting the device
      iv. Communicating outside the device via the error pin for outside intervention

The rest of the steps assume that the error has been handled and the system wants to clear the error event. Clearing the error event depends on whether the event is a level (see Section 12.10.2.3.1.2.1) or pulse (see Section 12.10.2.3.1.2.2) event.

12.10.2.3.1.2.1 ESM Low Priority Error Level Event

When a low priority error level event has to be cleared, the acting processor must perform the following steps:

1. Clear the low priority error level event at the source
2. Write 0x1 to the appropriate bit in the error group *j* of the ESM_STS_j register. This step will clear the raw status
   a. If the error event is still asserted (or re-asserted) the raw status will be set back to 0x1
   b. If there are no error events, the level will de-assert

Note

There is a possible software race condition if software manages to write to the Clear register before the de-asserted level from the source has been synchronized to the ESM clock. If this is an issue, software may perform a read-back at the source IP before writing the clear register to insure order.

3. Write the end of interrupt vector to the ESM_EOI interrupt register
   a. If there are low priority error level events enabled and pending, then a new pulse will be generated
   b. If there are no additional low priority error level events enabled and pending, there will be no new pulse
   c. The level interrupt output is unaffected by EOI
4. Write a CLEAR (0x5) to the ESM_PIN_CTRL register. This step is optional if the event is not enabled to influence the error pin (error group *j* ESM_PIN_EN_SET_j register), but may be done regardless as an extra CLEAR is not harmful.

12.10.2.3.1.2.2 ESM Low Priority Error Pulse Event

When a low priority error pulse event has to be cleared, the acting processor must perform the following steps:

1. Write 0x1 to the appropriate bit in the error group *j* of the ESM_STS_j register. This will clear the raw status and will de-assert the level interrupt.
2. Write the end of interrupt vector to the ESM_EOI interrupt register
   a. If there are additional low priority error pulse events enabled and pending, then a new pulse will be generated. The level interrupt will remain asserted and is unaffected by EOI.
   b. If there are no additional low priority error pulse events enabled and pending, there will be no new pulse
   c. Note – a pulse error event input that stays asserted will be treated as a "new" pulse event when it is cleared in the ESM.
   d. The level interrupt will remain asserted and is unaffected by EOI.
3. Clear the error event at the source. The source may generate a new pulse which will show up as a new error event at the ESM
4. Write a CLEAR (0x5) to the ESM_PIN_CTRL register. This step is optional if the event is not enabled to influence the error pin (error group *j* ESM_PIN_EN_SET_j register), but may be done regardless as an extra CLEAR is not harmful.

12.10.2.3.1.3 ESM High Priority Error Interrupt

Events mapped to the high priority error interrupt are intended to be events that require immediate intervention from the system because a potentially dangerous error has occurred. An example would be an event indicating an uncorrected error. The system will want to diagnose the issue and intervene to ensure there are no violations.

Copyright © 2024 Texas Instruments Incorporated

Any error event can be mapped to the high priority error interrupt. A high priority error interrupt will be generated when an event is enabled to cause an interrupt (via the ESM_INTR_EN_SET_j) register and mapped to the high priority interrupt (via the ESM_INT_PRIO_j) register and the raw status is set (via the ESM_RAW_j).

When a high priority error interrupt is received, the acting processor must perform the following steps:

1. Read the ESM_HI_PRI register
   a. If both [31-16] PLS and [15-0] LVL bit fields are equal to 0xFFFF, then interrupt is no longer asserted and the interrupt routine has ended.
   b. If either [31-16] PLS or [15-0] LVL bit fields are not equal to 0xFFFF, software has two options for determining what event to service:
      i. First option: Record the value in [31-16] PLS and [15-0] LVL bit fields. Determine which is higher priority. This is based on the global event map number of the highest priority high priority error event.

      Note

      The global event map is neither defined by nor maintain in ESM. The global event map must be defined and maintained by software using software determined memory resources. It is conceivable that the global event map will be predefined and loaded during the execution of a secondary boot loader.

      ii. Second option:
         1. Read the ESM_HI register to determine which event group(s) have pending high priority error interrupts
         2. Read the desired error group *j* ESM_STS_j register
         3. Identify which high priority error interrupt to service
2. Determine, based on the global event map for the device, where the error event came from
3. Service the error event based on the IP's specification:
   a. The system may take several actions including (but not limited to):
      i. Fixing the error
      ii. Resetting the errored IP peripheral
      iii. Resetting the device
      iv. Communicating outside the device via the error pin for outside intervention

The rest of the steps assume that the error has been handled and the system wants to clear the error event. Clearing the error event depends on whether the event is a level (see Section 12.10.2.3.1.3.1) or pulse (see Section 12.10.2.3.1.3.2) event.

12.10.2.3.1.3.1 ESM High Priority Error Level Event

When a high priority error level event has to be cleared, the acting processor must perform the following steps:

1. Clear the error event at the source
2. Write 0x1 to the appropriate bit in the error group *j* of the ESM_STS_j register. This step will clear the raw status
   a. If the error event is still asserted (or re-asserted) the raw status will be set back to 0x1
   b. If there are no error events, the level will de-assert

   Note

   There is a possible software race condition if software manages to write to the Clear register before the de-asserted level from the source has been synchronized to the ESM clock. If this is an issue, software may perform a read-back at the source IP before writing the clear register to insure order.

3. Write the end of interrupt vector to the ESM_EOI interrupt register
   a. If there are high priority error level events enabled and pending, then a new pulse will be generated
   b. If there are no additional high priority error level events enabled and pending, there will be no new pulse
   c. The level interrupt output is unaffected by EOI.

Copyright © 2024 Texas Instruments Incorporated

4. Write a CLEAR (0x5) to the ESM_PIN_CTRL register. This step is optional if the event is not enabled to influence the error pin (error group *j* ESM_PIN_EN_SET_j register, but may be done regardless as an extra CLEAR is not harmful.

12.10.2.3.1.3.2 ESM High Priority Error Pulse Event

When a high priority error pulse event has to be cleared, the acting processor must perform the following steps:

1. Write 0x1 to the appropriate bit in the error group *j* of the ESM_STS_j register. This will clear the raw status and will de-assert the level interrupt.
2. Write the end of interrupt vector to the ESM_EOI interrupt register
   a. If there are high priority error pulse events enabled and pending, then a new pulse will be generated and the level interrupt will remain asserted
   b. If there are no additional high priority error pulse events enabled and pending, there will be no new pulse
3. Clear the error event at the source. The source may generate a new pulse which will show up as a new error event at the ESM
4. Write a CLEAR (0x5) to the ESM_PIN_CTRL register. This step is optional if the event is not enabled to influence the error pin (error group *j* ESM_PIN_EN_SET_j register), but may be done regardless as an extra CLEAR is not harmful.

12.10.2.3.2 ESM Error Event Inputs

The ESM can have up to 1024 error event inputs, configurable by groups of 32. For the mapping of system interrupt error events to ESM interrupt inputs, see *Interrupt Sources*.

Level error events (active high) are synchronized to the ESM clock. This synchronized value is captured in to a flop.

Pulse error events use rising edge detection. Each pulse error event has 3 redundant inputs. Each input has its own edge detection logic. Multiple transmission protects against Single Event Upsets (SEUs, transient errors) causing a pulse to be lost during transmission and against failure of the edge detection logic. Once an edge has been detected on any of the three inputs, the ESM_RAW_j status is set. Subsequent pulses are likely to come concurrently or quickly enough that software will not have reacted yet. This logic is intentionally biased against false negatives and towards false positives. An SEU that causes an event where none actually occurred will cause software to be called in to action. Software must observe that there is no real error and clear the false status.

12.10.2.3.3 ESM Error Pin Output

The error pin output (ERR_O) is used to signal an external agent that it needs to (or may need to) intervene because of an error. Each error event input can be programmed, via software, to influence the error pin output (via the ESM_PIN_EN_SET_j register). The error pin output is active low or PWM based on the ESM_EN[7-4] PWM_EN field. This bit field should only be modified when the ESM is disabled, based on the ESM_EN register.

During Power-On Reset (POR), the error pin is active (asserted low) and the device drives this via a weak internal pull-down. The I/O is under the control of the device. When POR is removed from the ESM, it will be driving the error pin so the device can hand over control to the ESM. The user may also add an external pull-down that is only active when the device is in reset.

During a warm reset the state of the error pin is unchanged, that is the error pin logic is only reset by a POR. The device leaves the I/O active during a warm reset.

The ESM has also a software error forcing capability on the error pin. That is, a force error can be set via the ESM_EN[3-0] KEY bit field.

CAUTION

The isolation value for the ERR_O output of ESM is active (0). This is intended and supposed to protect against an accidental transition to a low power state. If the actual low power mode transition is intended, the PMIC should be made aware by software to ignore the ESM error signal. Otherwise device resets will be asserted from companion chip.

Copyright © 2024 Texas Instruments Incorporated

Figure 12-496 describes the behavior of the error pin. Not shown is that a reset (Power-On-Reset only) will immediately transition the error pin to the ESM_RESET state and a Global Soft Reset will immediately transition the error pin to the ESM_IDLE state. A pending error interrupt event is any error event with the raw state set and the error pin Influence enabled. There are two types of "clear" events associated with servicing the error pin. The first is to clear the status of the pending event (see Section 12.10.2.3.1 for how to clear level and pulse pending events). The second is the CLEAR event meant to de-assert the error pin.

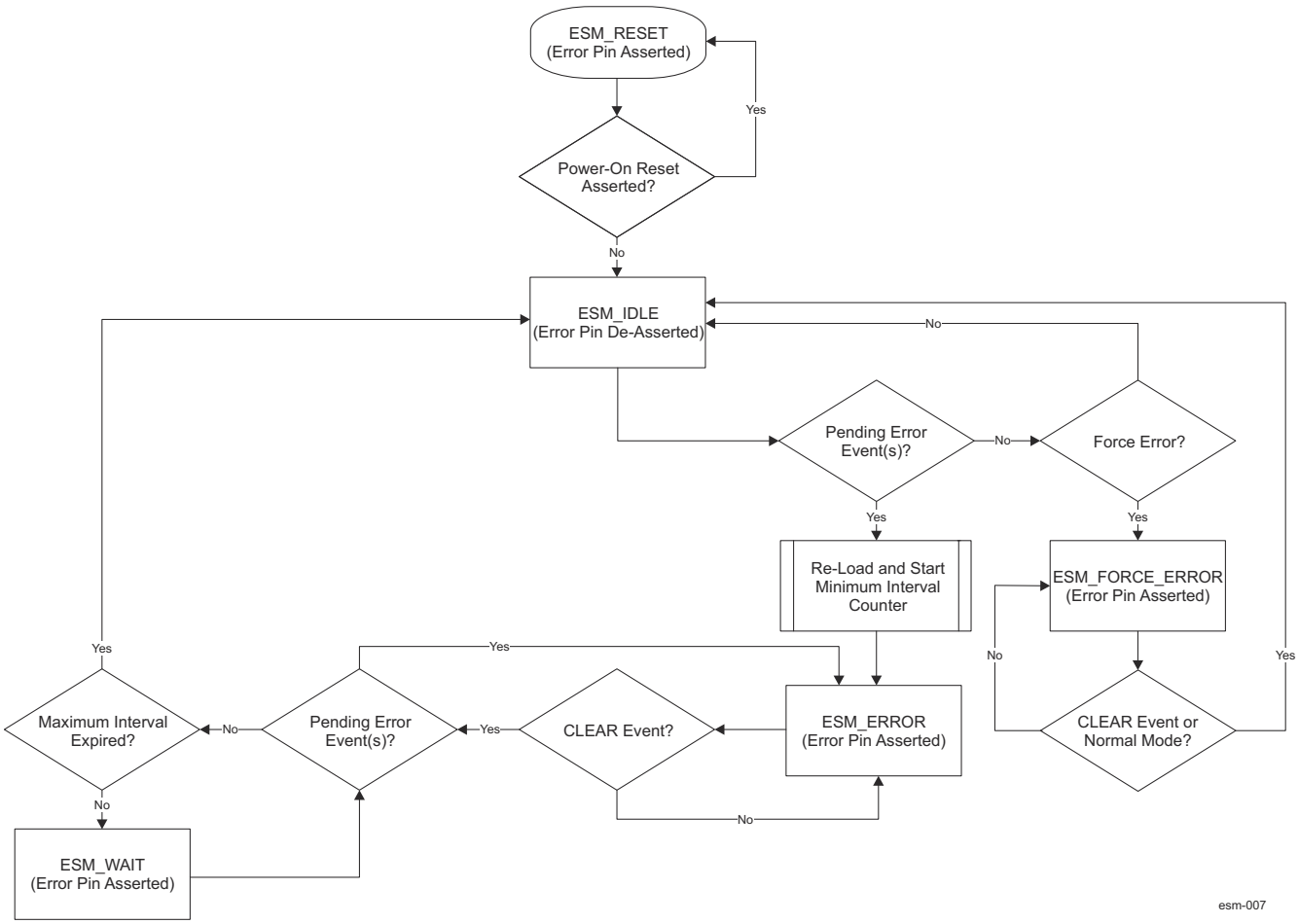


Figure 12-496. ESM Error Pin State Flowchart

If an error event happens that has been programmed to influence the error pin, the error pin will assert (active low) for a minimum time (as programmed by the ESM_PIN_CNTR_PRE register). In order for the error pin to de-assert, the following 3 things must happen:

1. The minimum time interval must expire
2. The event that caused the error pin to assert must be cleared (see Section 12.10.2.3.1)
3. A CLEAR (0x5) must be written to the ESM_PIN_CTRL register.

Note

Step 3 should happen after step 2, but either (or both) of these steps may happen before or after step 1.

Figure 12-497 shows a typical error pin assertion.

Copyright © 2024 Texas Instruments Incorporated

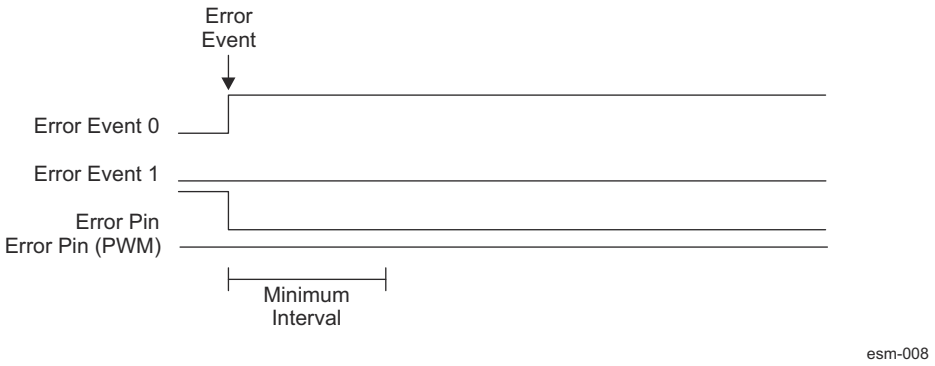


Figure 12-497. ESM Error Pin Assertion

If, during the minimum time, CLEAR is written to the error key, then the error pin will de-assert after the minimum time interval, as shown in Figure 12-498.

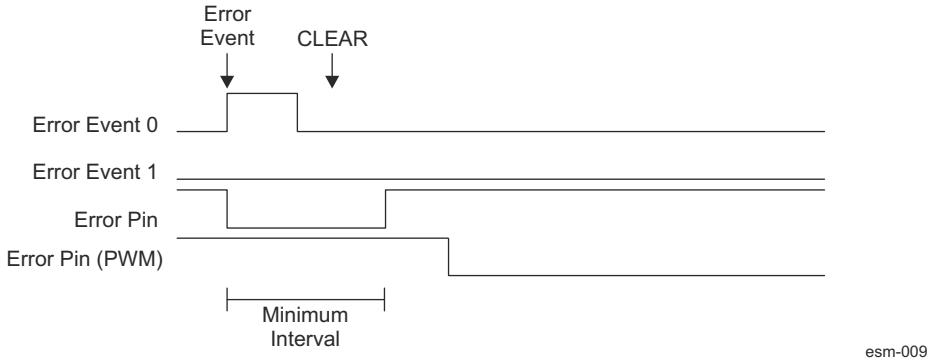


Figure 12-498. ESM Error Pin Assertion with CLEAR during Minimum Interval

If CLEAR is not written till after the minimum time interval, the error pin will de-assert when CLEAR is written. This is regardless of whether the error interrupt event itself is removed before or after the minimum time interval, as shown by the dotted line in Figure 12-499.

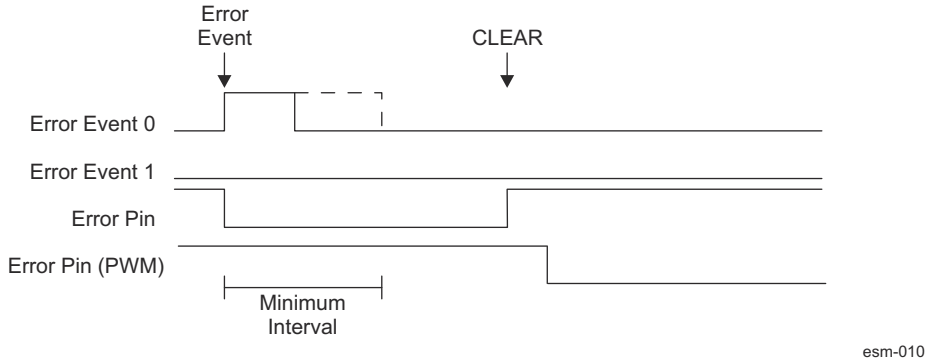


Figure 12-499. ESM Error Pin Asserting with CLEAR after Minimum Interval

When in the ESM_ERROR state and a CLEAR event happen, if there are still pending error events, the ESM stays in the ESM_ERROR state with the error pin asserted. Multiple error events when in the ESM_ERROR state do not reset the minimum time interval counter as shown in Figure 12-500.

Copyright © 2024 Texas Instruments Incorporated

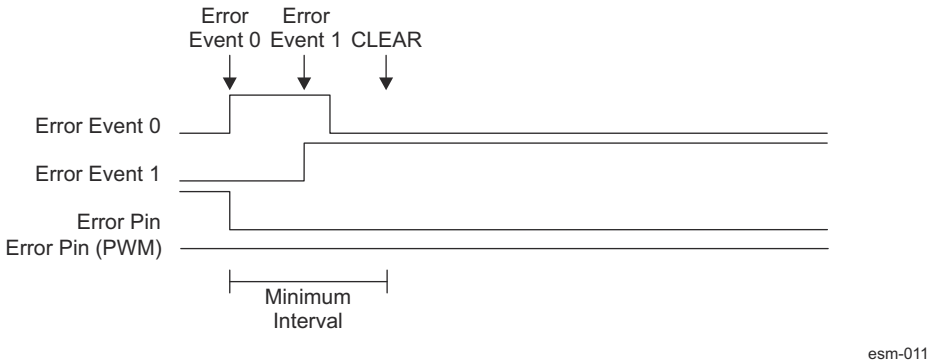


Figure 12-500. ESM Error Pin Asserting with Interval Reset by Additional Error Event(s)

A CLEAR event causes a re-evaluation of whether there are any pending error events. As such, a single CLEAR can be used to clear the error pin after multiple error events. Multiple CLEAR events can occur (such as the one with the dotted arrow shown in Figure 12-501), but are not necessary. No matter how many error events occur nor when (or how many) CLEAR events occur, the error pin will always be asserted for at least the minimum time interval

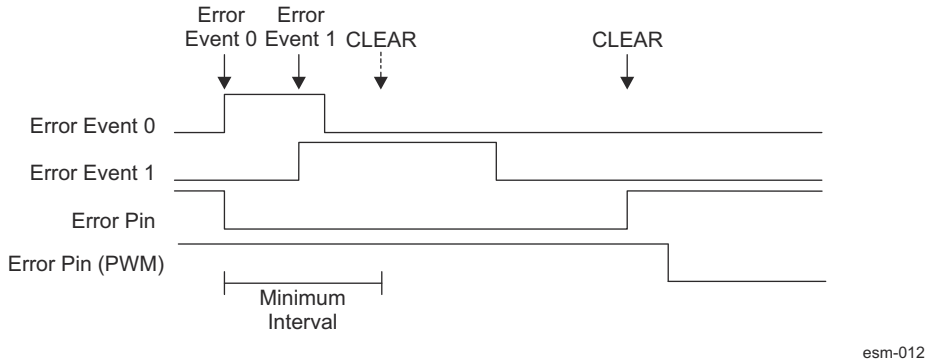


Figure 12-501. ESM Error Pin Asserting with Single CLEAR for Multiple Events

If all error events are cleared and the ESM is in the ESM_WAIT state, waiting for the minimum time interval to expire, and a new error interrupt event occurs, the ESM will go back to the ESM_ERROR state. The minimum time interval will not reset, but a new CLEAR event will be required as shown in Figure 12-502.

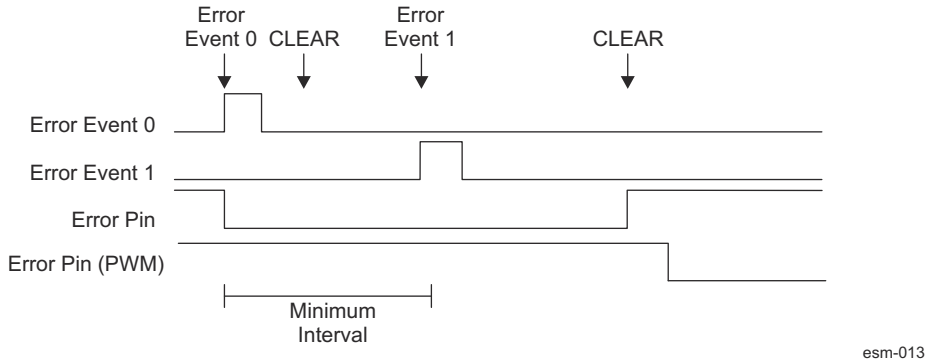


Figure 12-502. ESM Error Pin Asserting with New Error During Minimum Time Interval

Table 12-434 shows some common scenarios of how the error pin as well as the two associated registers (ESM_PIN_CTRL and ESM_PIN_STS) will be set.

Copyright © 2024 Texas Instruments Incorporated

Table 12-434. ESM Error Pin Scenarios

Scenario	Error Pin State Value	ESM_PIN_CTRL[3-0] KEY	ESM_PIN_STS[0] VAL status value	Additional Notes
POR Asserted	0	N/A	N/A	Registers are inaccessible. Device disables the I/O and pulls down internally.
After de-assertion of POR	1	0x0 (Normal Mode)	0x0	-
After de-assertion of Warm Reset (error was not asserted when reset asserted)	1	0x0 (Normal Mode)	0x0	-
After de-assertion of Warm Reset (error was asserted when reset asserted)	0	0x0 (Normal Mode)	0x1	-
Force error pin	0	0xA (Force Error Mode)	0x0	Forcing error on the pin via software.

12.10.2.3.4 PWM Mode

If the error output pin is in PWM mode then when no error is detected it will toggle according to programmable MMR widths for high and low periods. When an error occurs, the error pin stops toggling and remains constant until the error is cleared. An external PMIC that is detecting the PWM toggles can identify the error if the pin stops toggling. The periods should be programmed such that they fit within the expectation of the external PMIC.

12.10.2.3.5 ESM Minimum Time Interval

The minimum time interval is the minimum amount of time that the error pin will be asserted (active low) when an enabled error interrupt event happens. This value is system dependent, but should be enough time so that the external monitoring agent can always see the error pin asserted, but short enough so that if all of the error events are cleared, then the error pin can be de-asserted before the external agent decides to intervene. This is highly dependent on the application and the Fault Tolerant Time Interval.

The Minimum Time Interval counter is clock cycle based, therefore the time of the interval is a combination of the value in the ESM_PIN_CNTR_PRE register and the clock frequency of the ESM. Software must calculate the value accordingly. The Minimum Time Interval should be set according to the needs of the application.

12.10.2.3.6 ESM Protection for Registers

The configuration for each error group *j* of registers are backed up by 3 flops in order to protect against single or double-bit errors. When written, all 3 bits are set to the same value. When read (and for functioning of the internal state machines) the value is the OR of all 3 bits. Whenever any of the bits disagree, the Configuration Error interrupt is asserted (if enabled). The registers covered by this mechanism are:

- ESM_ERR_RAW
- ESM_ERR_STS
- ESM_ERR_EN_SET
- ESM_ERR_EN_CLR
- ESM_RAW_j
- ESM_STS_j
- ESM_INTR_EN_SET_j
- ESM_INTR_EN_CLR_j
- ESM_INT_PRIO_j
- ESM_PIN_EN_SET_j
- ESM_PIN_EN_CLR_j

The error pin control register ESM_PIN_CTRL contains a multi-bit field KEY. The key value ensures normal operation on the error pin and that an error even will be generated if one occurs. Software should periodically read check the KEY bit field value and make sure it is 0x0. If the value is not 0x0, software must re-write it to this key value (unless in test mode forcing an error on the pin) to ensure the normal operation. Table 12-435 lists the KEY values and their respective meaning.

Copyright © 2024 Texas Instruments Incorporated

Table 12-435. ESM Error Pin Control Values

ESM_PIN_CTRL[3-0] KEY	Description
N/A	Registers are inaccessible. Device disables the I/O and pulls down internally.
0x0 (Normal)	Normal operation mode - Error pin will activate when an enabled error event occurs.
0xA (Force Error)	Force error mode - Forces the error pin active. To clear the error pin (return to the ESM_IDLE state) write this field back to normal mode (writing a CLEAR event will also work). Force error mode must be set only while in IDLE. Attempting force error while in another state will have no effect.
0x5 (CLEAR)	CLEAR Event - generates a CLEAR event to the ESM state machine. KEY will return to normal mode (0x0) on the next cycle.
Other Values	All other values - Normal mode. Writing any of these values will have no effect. When reading any of these values indicates that one or more bits have experienced a single event upset, software should write the field back to 0x0. The ESM will continue to operate in normal mode.

The error pin counter pre-load register ESM_PIN_CNTR_PRE should also be read and checked periodically by software.

12.10.2.3.7 ESM Clock Stop

The ESM can only be put in to clock stop if all of the internal state machines are idle and the [3-0] KEY bit field for the global enable command is cleared in the ESM_EN register.

Copyright © 2024 Texas Instruments Incorporated

12.10.3 Memory Cyclic Redundancy Check (MCRC) Controller

This chapter describes the Memory Cyclic Redundancy Check (MCRC) controller in the device.

12.10.3.1 MCRC Overview

VBUSM CRC controller is a module which is used to perform CRC (Cyclic Redundancy Check) to verify the integrity of a memory system. A signature representing the contents of the memory is obtained when the contents of the memory are read into MCRC Controller. The responsibility of MCRC controller is to calculate the signature for a set of data and then compare the calculated signature value against a pre-determined good signature value. MCRC controller provides four channels to perform CRC calculation on multiple memories in parallel and can be used on any memory system.

12.10.3.1.1 MCRC Features

MCRC has the following features:

- Four channels to perform background signature verification on any memory subsystem
- Data compression on 8-, 16-, 32-, and 64-bit data size
- Maximum-length PSA (Parallel Signature Analysis) register constructed based on 64-bit primitive polynomial
- Each channel has a CRC Value Register which contains the pre-determined CRC value
- Use timed base event trigger from timer to initiate DMA data transfer
- Programmable 20-bit pattern counter per channel to count the number of data patterns for compression
- Three modes of operation:
  - Auto
  - Semi-CPU
  - Full-CPU
- For each channel, CRC can be performed either by MCRC Controller or by CPU
- Automatically performs signature verification without CPU intervention in AUTO mode
- Generates interrupt to CPU in Semi-CPU mode to allow CPU to perform signature verification itself
- Generates CRC fail interrupt in AUTO mode if signature verification fails
- Generates Timeout interrupt if CRC is not performed within the time limit
- Generates DMA request per channel to initiate CRC value transfer
- An 128-byte block burst address for the PSA register to DMA without constant mode bus attribute

12.10.3.1.2 MCRC Ports

Table 12-436. MCRC Clocks and Resets

Clocks	
Module Clock Input	**Description**
MCRC_FICLK	MCRC clock. This clock is used for all interface and functional operations.
Resets	
Module Reset Input	**Description**
MCRC_RST	MCRC hardware reset

Table 12-437. MCRC Hardware Requests

Interrupt Requests		
Module Interrupt Signal	**Description**	**Type**
INT_MCRC_INTR	MCRC Interrupt signal	Level
DMA Events		
Module DMA Event	**Description**	**Type**
DMA_EVENT_INTR0	MCRC DMA event for channel 1	Pulse
DMA_EVENT_INTR1	MCRC DMA event for channel 2	Pulse
DMA_EVENT_INTR2	MCRC DMA event for channel 3	Pulse
DMA_EVENT_INTR3	MCRC DMA event for channel 4	Pulse

Copyright © 2024 Texas Instruments Incorporated

12.10.3.2 MCRC Functional Description

12.10.3.2.1 MCRC Block Diagram

Figure 12-503 shows the MCRC internal blocks.

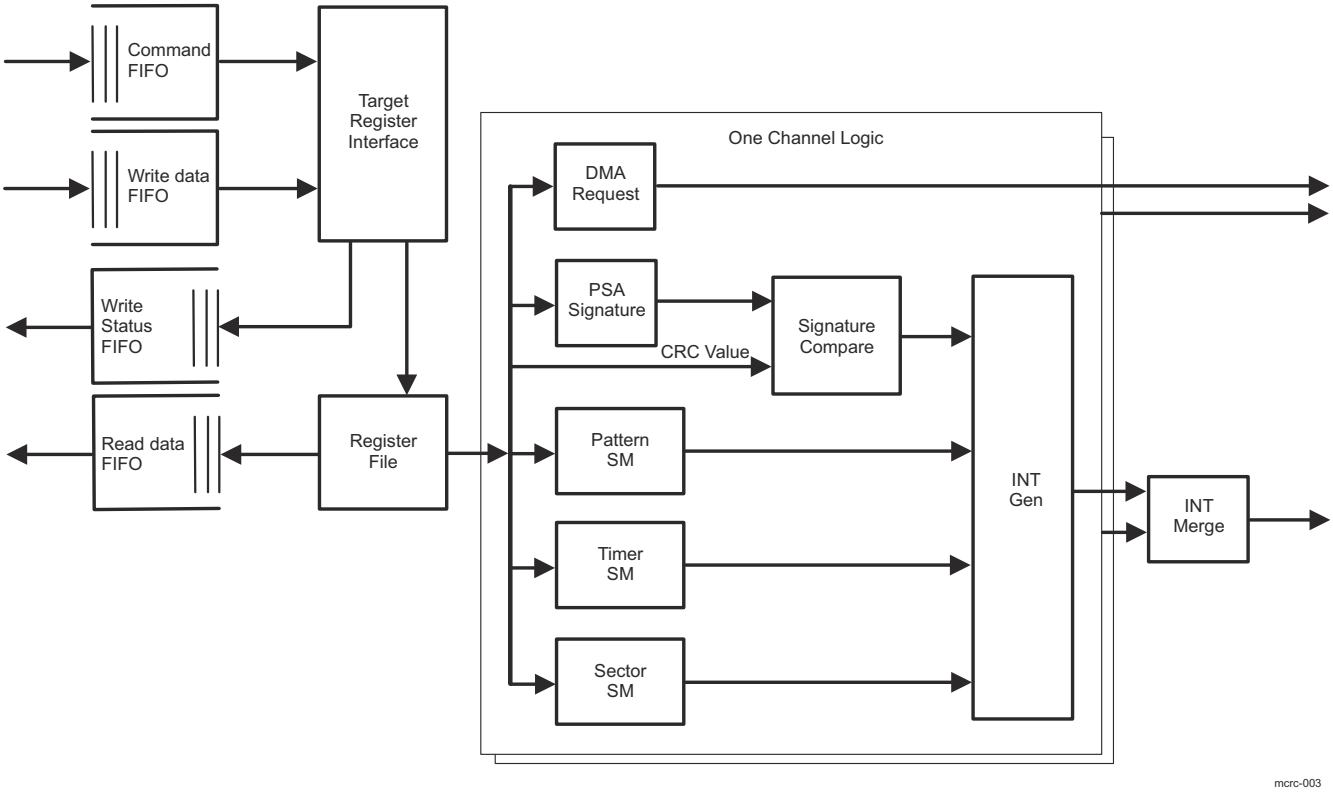


Figure 12-503. MCRC Block Diagram

- **Command FIFO**: The Command FIFO pipelines the commands to the target register interface. The Command and Write FIFOs allow the data to be coincident for processing. If there is no space for writes in the Write Status FIFO or no space in the Read Data FIFO for reads, the command processing will be halted until there is space in the appropriate FIFO. This FIFO is 4 elements deep.
- **Write FIFO**: The Write FIFO pipelines the write data to the target register interface. The Command and Write FIFOs allow the data to be coincident for processing. If there is no space for writes in the Write Status FIFO or no space in the Read Data FIFO for reads, the command processing will be halted until there is space in the appropriate FIFO. This FIFO is 2 elements deep.
- **Write Status FIFO**: The Write Status FIFO pipelines the write status back to the VBUSM. A write status will be issued on the final data phase of a write command. This FIFO is 2 elements deep.
- **Read Data FIFO**: The Read Data FIFO pipelines the read data back to the VBUSM. This FIFO is 3 elements deep.
- **Target Register Interface**: The Target Register Interface directs the written data to the register file.
- **PSA Signature**: The PSA Signature creates the signature of the data written. This data will then be compared to the CRC Value or read by software to determine goodness.
- **Pattern State Machine**: The Pattern State Machine determines when a block of data has been serviced.
- **Timer State Machine**: The Timer State Machine determines when overrun and under-run events are detected.
- **Sector State Machine**: The Sector State Machine determines when a sector error should be captured so the software can determine the errant block of data.
- **Signature Compare**: The Signature Compare block compares the current signature to the CRC Value register and sends the result to the Interrupt Generation block.

Copyright © 2024 Texas Instruments Incorporated

12.10.3.2.2 MCRC General Operation

There are four channels in MCRC controller and for each channel there is a PSA (Parallel Signature Analysis) Signature Register (MCRC_PSA_SIGREGL1-4) and a CRC (Cyclic Redundancy Check) Value Register (MCRC_CRC_REGL1-4).

A memory can be organized into multiple sectors with each sector consisting of multiple data patterns. A data pattern can be a 8-, 16-, 32-, or 64-bit data. MCRC module performs the signature calculation and compares the signature to a pre-determined value. The PSA Signature Register compresses an incoming data pattern into a signature when it is written. When one sector of data patterns are written into PSA Signature Register, a final signature corresponding to the sector is obtained. CRC Value Register stores the pre-determined signature corresponding to one sector of data patterns. The calculated signature and the pre-determined signature are then compared to each other for signature verification. To minimize CPU's involvement, data patterns transfer can be carried out at the background of CPU using DMA controller. DMA is setup to transfer data from memory of which the contents to be verified to the memory mapped PSA Signature Register. When DMA transfers data to the memory mapped PSA Signature Register, a signature is generated.

A programmable 20-bit data pattern counter is used for each channel to define the number of data patterns to calculate for each sector. Signature verification can be performed automatically by MCRC controller in AUTO mode or by CPU itself in Semi-CPU or Full-CPU mode. In AUTO mode, a self-sustained CRC signature calculation can be achieved without any CPU intervention.

12.10.3.2.3 MCRC Modes of Operation

MCRC Controller can operate in AUTO, Semi-CPU, and Full-CPU modes.

12.10.3.2.3.1 AUTO Mode

In AUTO mode, MCRC Controller in conjunction with DMA controller can perform CRC without CPU intervention. A sustained transfer of data to both the PSA Signature Register (MCRC_PSA_SIGREGL1-4) and a CRC (Cyclic Redundancy Check) Value Register (MCRC_CRC_REGL1-4) are performed in the background of CPU. When a mismatch is detected, an interrupt is generated to the CPU. A 16-bit, current sector ID register (MCRC_CRC_CURSEC_REG1-4) is provided to identify which sector causes a CRC failure.

12.10.3.2.3.2 Semi-CPU Mode

In Semi-CPU mode, DMA controller is also utilized to perform data patterns transfer to PSA Signature Register (MCRC_PSA_SIGREGL1-4). Instead of performing signature verification automatically, the CRC controller generates an compression complete interrupt to CPU after each sector is compressed. Upon responding to the interrupt the CPU performs the signature verification by reading the calculated signature stored at the PSA Sector Signature Register (MCRC_PSA_SECSIGREGL1-4) and compare it to a pre-determined CRC value.

12.10.3.2.3.3 Full-CPU Mode

In Full-CPU mode, the CPU does the data patterns transfer and signature verification all by itself. When CPU has enough throughput, it can perform data patterns transfer by reading data from the memory system to the PSA Signature Register (MCRC_PSA_SIGREGL1). After certain number of data patterns are compressed, the CPU can read from the PSA Signature Register and compare the calculated signature to the pre-determined CRC signature value. In Full-CPU mode, neither interrupt nor DMA request is generated. All counters are also disabled.

12.10.3.2.4 PSA Signature Register

The 64-bit PSA Signature Register (MCRC_PSA_SIGREGL1-4 and MCRC_PSA_SIGREGH1-4) is based on one of the selected CRC polynomials to produce the maximum length LFSR (Linear Feedback Shift Register).

$$\text{CRC16: } f(x) = x^{16} + x^{12} + x^{5} + 1 \quad (29)$$

$$\text{CRC32: } f(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + x + 1 \quad (30)$$

$$\text{CRC64: } f(x) = x^{64} + x^{4} + x^{3} + x + 1 \quad (31)$$

$$\text{SAE J1850 CRC8: } f(x) = x^{8} + x^{4} + x^{3} + x^{2} + 1 \quad (32)$$

Copyright © 2024 Texas Instruments Incorporated

H2F Autosar 4.0: $f(x) = x^5 + x^3 + x^2 + x + 1$ (33)

CASTAGNOLI, iSCSI: $f(x) = x^{28} + x^{27} + x^{26} + x^{25} + x^{23} + x^{22} + x^{20} + x^{19} + x^{18} + x^{14} + x^{13} + x^{11} + x^{10} + x^9 + x^8 + x^6 + 1$ (34)

E2E Profile 4: $f(x) = x^{31} + x^{30} + x^{29} + x^{28} + x^{26} + x^{23} + x^{21} + x^{19} + x^{18} + x^{15} + x^{14} + x^{13} + x^{12} + x^{11} + x^9 + x^8 + x^4 + x + 1$ (35)

A. More details of the 64-bit primitive polynomial can be found at W. Stahnke, Primitive binary polynomials, Math. Comp. 27 (1973), 977--980.

There is one PSA Signature Register per CRC channel. PSA Signature Register can be both read and written. When it is written, it can either compress the data or just capture the data depending on the state of CHi_MODE bits (where i = 1 to 4). If CHi_MODE=0x0 (Data Capture), a seed value can be planted in the PSA Signature Register without compression. Other modes other than Data Capture will result with the data compressed by PSA Signature Register when it is written. Each channel can be planted with different seed value before compression starts. When PSA Signature Register is read, it gives the calculated signature.

MCRC Controller should be used in conjunction with the on-chip DMA controller to produce optimal system performance. The incoming data pattern to PSA Signature Register is typically initiated by the DMA controller. When DMA is properly setup, it would read data from the pre-determined memory system and write them to the memory mapped PSA Signature Register. Each time PSA Signature Register is written a signature is generated. CPU itself can also perform data transfer by reading from the memory system and perform write operation to PSA Signature Register if CPU has enough throughput to handle data patterns transfer.

After a system reset and when AUTO mode is enabled, MCRC Controller automatically generates a DMA request to request the pre-determined CRC value corresponding to the first sector of memory to be checked.

In AUTO mode, when one sector of data patterns is compressed, the signature stored at the PSA Signature Register is first copied to the PSA Sector Signature Register (MCRC_PSA_SECSIGREGL1-4) and PSA Signature Register is then cleared out to all zeros. An automatic signature verification is then performed by comparing the signature stored at the PSA Sector Signature Register to the CRC Value Register (MCRC_CRC_REGL1-4). After the comparison the MCRC Controller can generate a DMA request. Upon receiving the DMA request the DMA controller will update the CRC Value Register by transferring the next pre-determined signature value associated with the next sector of memory system. If the signature verification fails then MCRC Controller can generate a CRC fail interrupt.

In Full-CPU mode, no DMA request and interrupt are generated at all. The number of data patterns to be compressed is determined by CPU itself. Full-CPU mode is useful when DMA controller is not available to perform background data patterns transfer. Software can periodically generate a software interrupt to CPU and use CPU to accomplish data transfer and signature verification.

MCRC Controller supports double word, word, half word and byte access to the PSA Signature Register. During a non-doubleword write access, all unwritten byte lanes are padded with zeros before compression. Note that comparison between PSA Sector Signature Register and CRC Value Register is always in 64 bits because a compressed value is always expressed in 64 bits.

There is a software reset per channel for PSA Signature Register. When set, the PSA Signature Register is reset to all zeros.

PSA Signature Register is reset to zero under the following conditions:

- System reset
- PSA Software reset
- One sector of data patterns is compressed

12.10.3.2.5 PSA Sector Signature Register

After one sector of data is compressed, the final resulting signature calculated by PSA Signature Register is transferred to the 64-bit PSA Sector Signature Register (MCRC_PSA_SECSIGREGL1-4 and MCRC_PSA_SECSIGREGH1-4). PSA Signature Register is a read-only register. During Semi-CPU mode, the host CPU must read from the PSA Sector Signature Register instead of reading from PSA Signature Register

Copyright © 2024 Texas Instruments Incorporated

for signature verification to avoid data coherency issue. The PSA Signature Register can be updated with new signature before the host CPU is able to retrieve it.

In Semi-CPU mode, no DMA request is generated. When one sector of data patterns is compressed, CRC controller first generates a compression complete interrupt. Responding to the interrupt, CPU will read the PSA Sector Signature Register and compare it to the known good signature or write the signature value to another memory location to build a signature file. In Semi-CPU mode, CPU must perform the signature verification in a manner to prevent any overrun condition. The overrun condition occurs when the compression complete interrupt is generated after one sector of data patterns is compressed and CPU has not read from the PSA Sector Signature Register to perform necessary signature verification before PSA Sector Signature Register is overridden with a new value. An overrun interrupt can be enabled to generate when overrun condition occurs.

12.10.3.2.6 CRC Value Register

Associated with each channel there is a 64-bit CRC Value Register (MCRC_CRC_REGL1-4 and MCRC_CRC_REGH1-4).

The CRC Value Register stores the pre-determined CRC value. After one sector of data patterns is compressed by PSA Signature Register, MCRC Controller can automatically compare the resulting signature stored at the PSA Sector Signature Register with the pre-determined value stored at the CRC Value Register if AUTO mode is enabled. If the signature verification fails, MCRC Controller can be enabled to generate an CRC fail interrupt. When the channel is set up for Semi-CPU mode, CRC controller first generates a compression complete interrupt to CPU. Upon servicing the interrupt, CPU will then read the PSA Sector Signature Register and then read the corresponding CRC value stored at another location and compare them. CPU must not read from the CRC Value Register during Semi-CPU or Full-CPU mode because the CRC Value Register is not updated during these two modes.

In AUTO mode, for first sector's signature, DMA request is generated when mode is programmed to AUTO. For subsequent sectors, DMA request is generated after each sector is compressed. Responding to the DMA request, DMA controller reloads the CRC Value Register for the next sector of memory system to be checked. The user software needs to configure the DMA to ensure that the DMA first writes to the MCRC_CRC_REGL1 followed by the MCRC_CRC_REGH1 register in during the AUTO Mode.

When CRC Value Register is updated with a new CRC value, an internal flag is set to indicate that CRC Value Register contains the most current value. This flag is cleared when CRC comparison is performed. Each time at the end of the final data pattern compression of a sector, MCRC Controller first checks to see if the corresponding CRC Value Register has the most current CRC value stored in it by polling the flag. If the flag is set then the CRC comparison can be performed. If the flag is not set then it means the CRC Value Register contains stale information. A CRC underrun interrupt is generated. When an underrun condition is detected, signature verification is not performed.

MCRC Controller supports double word, word, half word and byte access to the CRC Value Register. As noted before comparison between PSA Sector Signature Register and CRC Value Register during AUTO mode is carried out in 64 bits.

12.10.3.2.7 Raw Data Register

The raw or un-compressed data written to the PSA Signature Register is also saved in the 64-bit Raw Data Register (MCRC_RAW_DATAREGL1-4 and MCRC_RAW_DATAREGH1-4). This register is read only.

12.10.3.2.8 Example DMA Controller Setup

DMA controller needs to be setup properly in either AUTO or Semi-CPU mode as DMA controller is used to transfer data patterns. Hardware or a combination of hardware and software DMA triggering are supported.

12.10.3.2.8.1 AUTO Mode Using Hardware Timer Trigger

There are two DMA channels associated with each CRC channel when in AUTO mode. One DMA channel is setup to transfer data patterns from the source memory to the PSA Signature Register (MCRC_PSA_SIGREGL1-4). The second DMA channel is setup to transfer the pre-determined signature to the CRC Value Register (MCRC_CRC_REGL1-4). The trigger source for the first DMA channel can be either by hardware or by software. As illustrated in Figure 12-504, a timer can be used to trigger a DMA request to

Copyright © 2024 Texas Instruments Incorporated

initiate transfer from the source memory system to PSA Signature Register. In AUTO mode, MCRC Controller also generates DMA request after one sector of data patterns is compressed to initiate transfer of the next CRC value corresponding to the next sector of memory. Thus a new CRC value is always updated in the CRC Value Register (MCRC_CRC_REGL1-4)by DMA synchronized to each sector of memory.

A block of memory system is usually divided into many sectors. All sectors are the same size. The sector size is programmed in the MCRC_CRC_PCOUNT_REG1-4 and the number of sectors in one block is programmed in the MCRC_CRC_SCOUNT_REG1-4 of the respective channel. MCRC_CRC_PCOUNT_REG1-4 multiplies MCRC_CRC_SCOUNT_REG1-4 and multiplies transfer size of each data pattern should give the total block size in number of bytes.

The total size of the memory system to be examined is also programmed in the respective transfer count register inside DMA module. The DMA transfer count register is divided into two parts. They are element count and frame count. Note that a hardware DMA request can be programmed to trigger either one frame or one entire block transfer. In Figure 12-504, a hardware DMA request from a timer is used as a trigger source to initiate DMA transfer. If all four CRC channels are active in AUTO mode then a total of four DMA requests would be generated by MCRC Controller.

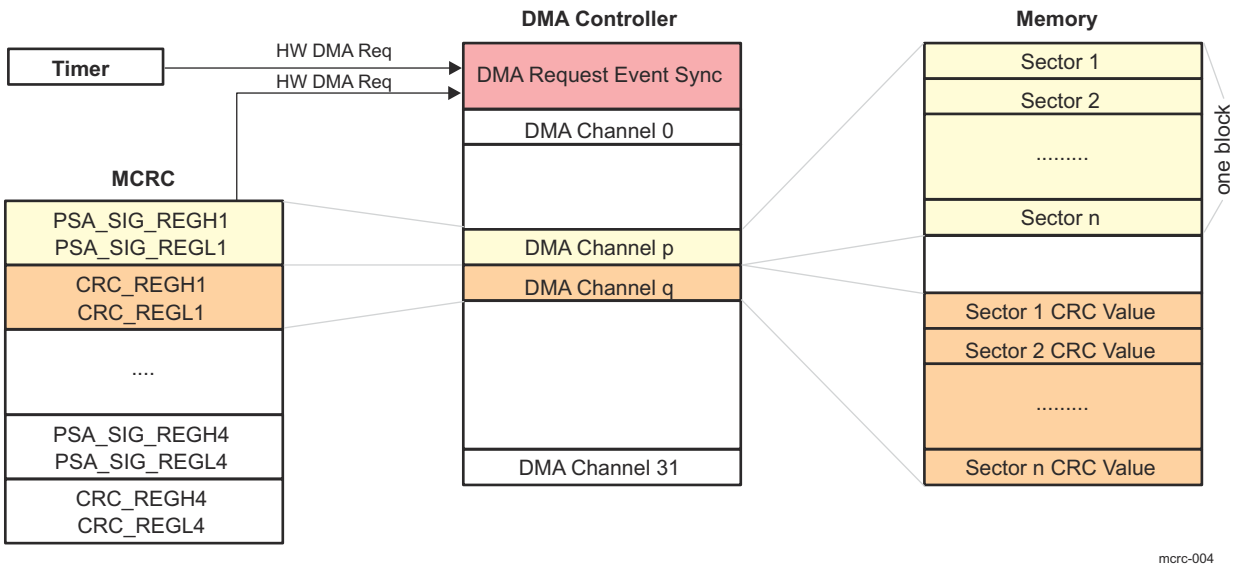


Figure 12-504. AUTO Mode Using Hardware Timer Trigger

12.10.3.2.8.2 AUTO Mode Using Software Trigger

The data patterns transfer can also be initiated by software. CPU can generate a software DMA request to activate the DMA channel to transfer data patterns from source memory system to the PSA Signature Register. To generate a software DMA request CPU needs to set the corresponding DMA channel in the DMA software trigger register. Note that just one software DMA request from CPU is enough to complete the entire data patterns transfer for all sectors. Please see AUTO Mode With Software CPU Trigger for illustration.

Copyright © 2024 Texas Instruments Incorporated

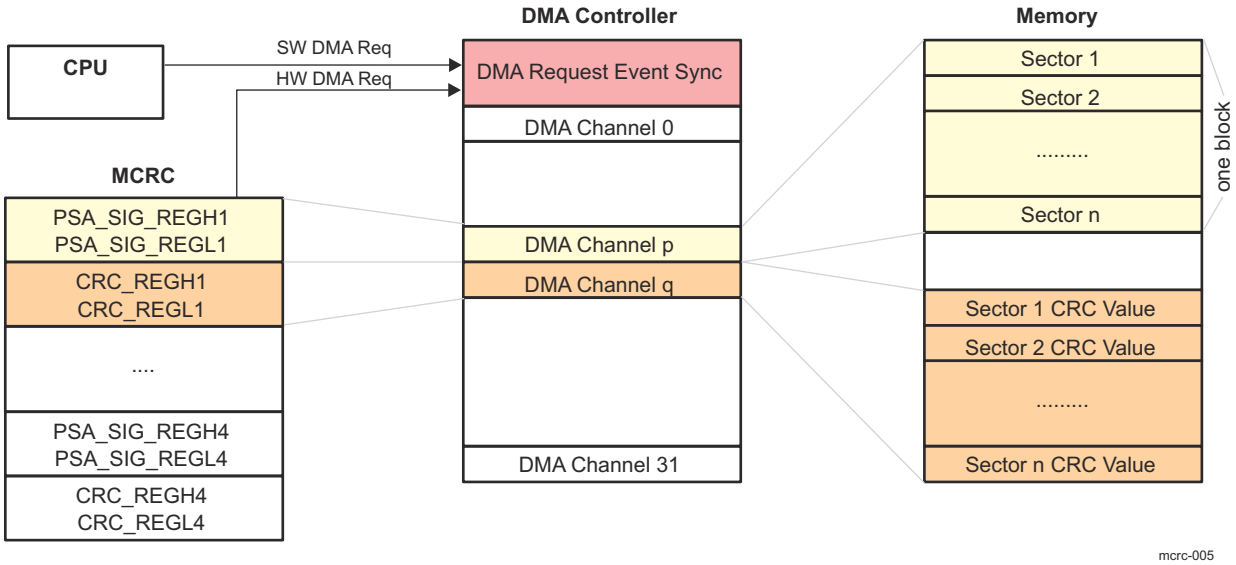


Figure 12-505. AUTO Mode With Software CPU Trigger

12.10.3.2.8.3 Semi-CPU Mode Using Hardware Timer Trigger

During Semi-CPU mode, no DMA request is generated by CRC controller. Therefore, no DMA channel is allocated to update CRC Value Register. CPU should not read from CRC Value Register in semi-CPU mode as it contains stale value. Note that no signature verification is performed at all during this mode. Similar to AUTO mode, either by hardware or by software DMA request can be used as a trigger for data patterns transfer. Figure 12-506 illustrates the DMA setup using semi-CPU mode with hardware timer trigger.

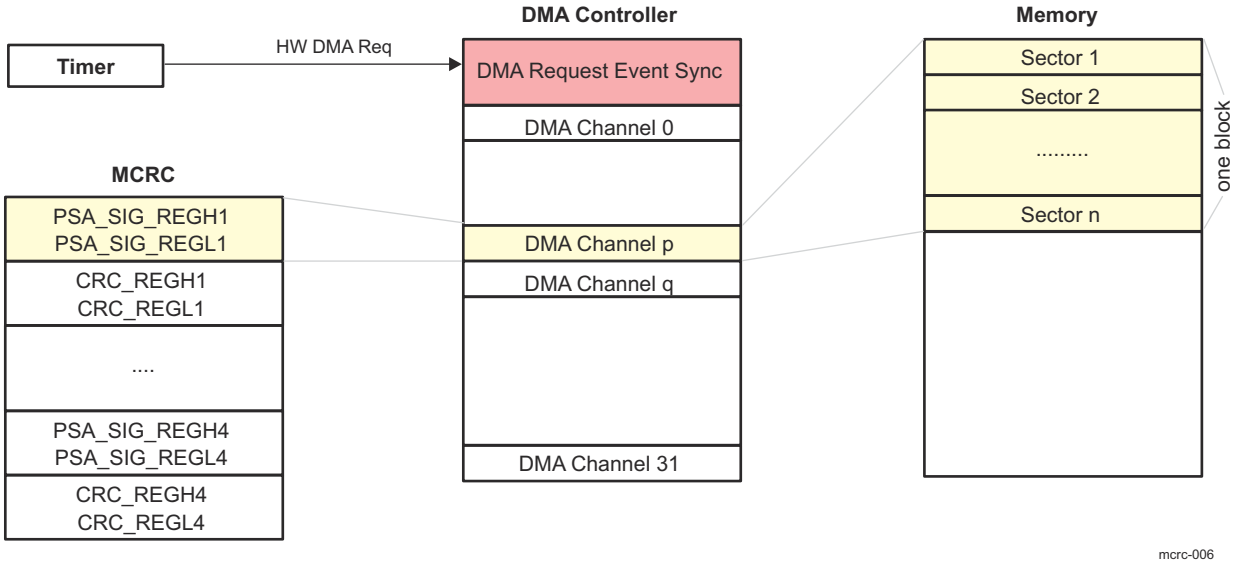


Figure 12-506. Semi-CPU Mode With Hardware Timer Trigger

Table 12-438. DMA Request and Counter Logic Operation According to CRC Mode

CRC Mode	DMA Request	Pattern Counter	Sector Counter	Timeout Counter
AUTO	Active	Active	Active	Active
Semi-CPU	Inactive	Active	Active	Active
Full CPU	Inactive	Inactive	Inactive	Inactive

Copyright © 2024 Texas Instruments Incorporated

12.10.3.2.9 Pattern Count Register

There is a 20-bit data pattern counter for every CRC channel. The data pattern counter is a down counter and can be pre-loaded with a programmable value stored in the Pattern Count Register (MCRC_CRC_PCOUNT_REG1-4). When the data pattern counter reaches zero, a compression complete interrupt is generated in Semi-CPU mode and an automatic signature verification is performed in AUTO mode. In AUTO only, DMA request is generated to trigger the DMA controller to update the CRC Value Register.

Note

The data pattern count must be divisible by the total transfer count as programmed in DMA controller. The total transfer count is the product of element count and frame count.

12.10.3.2.10 Sector Count Register/Current Sector Register

Each channel contains a 16-bit sector counter. The sector count register stores the number of sectors. Sector counter is a free running counter and is incremented by one each time when one sector of data patterns is compressed. When the signature verification fails, the current value stored in the sector counter is saved into current sector register (MCRC_CRC_CURSEC_REG1-4). If signature verification fails, CPU can read from the current sector register to identify the sector which causes the CRC mismatch. To aid and facilitate the CPU in determining the cause of a CRC failure it is advisable to use the following equation during CRC and DMA setup.

$$\text{CRC Pattern Count} \times \text{CRC Sector Count} = \text{DMA Element Count} \times \text{DMA Frame Count} \quad (36)$$

The current sector register is frozen from being updated until both the current sector register is read and CRC fail (CHi_CRC_FAIL) status bit is cleared by CPU. If CPU does not respond to the CRC failure in a timely manner before another sector produces a signature verification failure, the current sector register is not updated with the new sector number. An overrun interrupt is generate instead. If current sector register is already frozen with an erroneous sector and emulation is entered with SUSPEND signal goes to high then the register still remains frozen even it is read.

In Semi-CPU mode, the current sector register is used to indicate the sector for which the compression complete has last happened.

Current sector register is reset when the PSA software reset is enabled.

Note

Both data pattern count and sector count registers must be greater than or equal to one for the counters to count. After reset, pattern count and sector count registers default to zero and the associated counters are inactive.

12.10.3.2.11 Interrupts

CRC generate several types of interrupts per channel. Associated with each interrupt there is a interrupt enable bit (see MCRC_CRC_INTS). No interrupt is generated in Full-CPU mode.

- Compression complete interrupt
- CRC fail interrupt
- Overrun interrupt
- Underrun interrupt
- Timeout interrupt

Table 12-439. Interrupt Conditions Per CRC Mode

CRC Mode	Compression Complete	CRC Fail	Overrun	Underrun	Timeout
AUTO	No	Yes	Yes	Yes	Yes
Semi-CPU	Yes	No	Yes	No	Yes
Full-CPU	No	No	No	No	No

Copyright © 2024 Texas Instruments Incorporated

12.10.3.2.11.1 Compression Complete Interrupt

Compression complete interrupt is generated in Semi-CPU mode only. When the data pattern counter reaches zero, the compression complete flag is set and the interrupt is generated

12.10.3.2.11.2 CRC Fail Interrupt

CRC fail interrupt is generated in AUTO mode only. When the signature verification fails, the CRC fail flag is set, and CPU must take action to address the fail condition and clear the CRC fail flag after it resolves the CRC mismatch.

12.10.3.2.11.3 Overrun Interrupt

Overrun Interrupt is generated in either AUTO or Semi-CPU mode. During AUTO mode, if a CRC fail is detected then the current sector number is recorded in the current sector register (MCRC_CRC_CURSEC_REG1-4). If CRC fail status bit is not cleared and current sector register is not read by the host CPU before another CRC fail is detected for another sector then an overrun interrupt is generated. During Semi-CPU mode, when the data pattern counter finishes counting, it generates a compression complete interrupt. At the same time the signature is copied into the PSA Sector Signature Register (MCRC_PSA_SECSIGREGL1-4). If the host CPU does not read the signature from PSA Sector Signature Register before it is updated again with a new signature value then an overrun interrupt is generated.

12.10.3.2.11.4 Underrun Interrupt

Underrun interrupt only occurs in AUTO mode. The interrupt is generated when the CRC Value Register is not updated with the corresponding signature when the data pattern counter finishes counting. During AUTO mode, MCRC Controller generates DMA request to update CRC Value Register in synchronization to the corresponding sector of the memory. Signature verification is also performed if underrun condition is detected. And CRC fail interrupt is generated at the same time as the underrun interrupt.

12.10.3.2.11.5 Timeout Interrupt

To ensure that the memory system is examined within a pre-defined time frame and no loss of incoming data there is a 24-bit timeout counter per CRC channel. The timeout counter can be pre-loaded with two different pre-load values, watchdog timeout pre-load value (MCRC_CRC_WDTOPLD1-4) and block complete timeout pre-load value (MCRC_CRC_BCTOPLD1-4). The timeout counter is clocked by a prescaler clock which is permanently running at division 64 of FICLK clock.

Watchdog timeout pre-load register (MCRC_CRC_WDTOPLD1-4) is used to check if DMA does supply a block of data responding to a request in a given time frame. Block complete timeout pre-load register (MCRC_CRC_BCTOPLD1-4) is used to check if one complete block of data patterns are compressed within a specific time frame. The timeout counter is first pre-loaded with MCRC_CRC_WDTOPLD1-4 after either AUTO or Semi-CPU mode is selected and starts to down count. If the timeout counter expires before DMA transfers any data pattern to PSA Signature Register then a timeout interrupt is generated. An incoming data pattern before the timeout counter expires will automatically pre-load the timeout counter with MCRC_CRC_BCTOPLD1-4 the block complete timeout pre-load value.

Block complete timeout pre-load value is used to check it one block of data patterns are compressed within a given time limit. If the timeout counter pre-loaded with MCRC_CRC_BCTOPLD1-4 value expires before one block of data patterns are compressed a timeout interrupt is generated. When one block (pattern count × sector count) of data patterns are compressed before the counter has expired, the counter is pre-loaded with MCRC_CRC_WDTOPLD1-4 value again. If the timeout counter is pre-loaded with zero then the counter is disable and no timeout interrupt is generated.

12.10.3.2.11.6 Interrupt Offset Register

MCRC Controller only generates one interrupt request to interrupt manager. An interrupt offset register (MCRC_CRC_INT_OFFSET_REG) is provided to indicate the source of the pending interrupt with highest priority. Table 12-440 shows the offset interrupt vector address of each interrupt in an ascending order of priority.

Table 12-440. Interrupt Offset Mapping

Interrupt Condition	Offset Value
Phantom	0x0

Copyright © 2024 Texas Instruments Incorporated

Table 12-440. Interrupt Offset Mapping (continued)

Interrupt Condition	Offset Value
Ch1 CRC Fail	0x1
Ch2 CRC Fail	0x2
Ch3 CRC Fail	0x3
Ch4 CRC Fail	0x4
reserved	0x5-0x8
Ch1 Compression Complete	0x9
Ch2 Compression Complete	0xA
Ch3 Compression Complete	0xB
Ch4 Compression Complete	0xC
reserved	0xD-0x10
Ch1 Overrun	0x11
Ch2 Overrun	0x12
Ch3 Overrun	0x13
Ch4 Overrun	0x14
reserved	0x15-0x18
Ch1 Underrun	0x19
Ch2 Underrun	0x1A
Ch3 Underrun	0x1B
Ch4 Underrun	0x1C
reserved	0x1D-0x20
Ch1 Timeout	0x21
Ch2 Timeout	0x22
Ch3 Timeout	0x23
Ch4 Timeout	0x24

12.10.3.2.11.7 Error Handling

When an interrupt is generated, host CPU must take appropriate actions to identify the source of error and restart the respective channel in DMA and MCRC module. To restart a CRC channel user must perform the following steps in the ISR:

1. Write to software reset bit in MCRC_CRC_CTRL0 register to reset the respective PSA Signature Register
2. Reset the CHi_MODE bits to 0 in MCRC_CRC_CTRL2 register as Data capture mode
3. Set the CHi_MODE bits in MCRC_CRC_CTRL2 register to desired new mode again
4. Release software reset

The host CPU must use byte write to restart each individual channel.

12.10.3.2.12 Power Down Mode

MCRC module can be put into power down mode when the power down control bit MCRC_CRC_CTRL1[0] PWDN is set. The module wakes up when the PWDN bit is cleared. When MCRC controller is in power down mode, no data tracing alone will happen.

12.10.3.2.13 Emulation

A read access from a register in functional mode can sometimes trigger a certain internal event to follow. For example, reading interrupt offset register triggers an event to clear the corresponding interrupt status flag. During emulation when SUSPEND signal is high, a read access from any register should only return the register contents to the bus and should not trigger or mask any event as it would have in functional mode. This is to prevent debugger from reading the interrupt offset register, that is, during refreshing screen and cause the corresponding interrupt status flag to get cleared. Timeout counters are stopped to generate timeout interrupts in emulation mode. No VBUSM bus error will be generated if reading from the unimplemented locations. .

Copyright © 2024 Texas Instruments Incorporated

CEMUDBG is the VBUSM suspend signal which need not explicitly indicate that whether CPU is in suspend mode or not. In data trace mode, a separate suspend signal CPU_EMUSP, is used to indicate MCRC controller that the CPU is in suspend mode or not.

Copyright © 2024 Texas Instruments Incorporated

12.10.3.3 MCRC Programming Examples

12.10.3.3.1 Example: Auto Mode Using Time Based Event Triggering

A large memory area with 2 Mbyte (256 k × 32-bit [doubleword]) is to be checked in the background of CPU. CRC is to be performed every 1 Kbyte (128 doubleword). Therefore there will be 2048 pre-recorded CRC values. For illustration purpose, we map MCRC_CRC_REGL1 register to DMA channel 1 and MCRC_PSA_SIGREGL1 register to DMA channel 2. Let's assume all DMA transfers are carried out in 64-bit transfer size.

12.10.3.3.1.1 DMA Setup

- Setup DMA channel 1 with the starting address from which the pre-determined CRC values are stored. Setup the destination address to the MCRC_CRC_REGL1. Put the source address at post increment addressing mode and put the destination address at constant addressing mode. Use hardware DMA request for channel 1 to trigger a frame transfer.
- Setup DMA channel 2 with the source address from which the contents of memory to be verified. Setup the destination address to the MCRC_PSA_SIGREGL1. Program the element transfer count to 128 and the frame transfer count to 2048. Program the read and write element size to 64 bits. Put the source address at post increment addressing mode and put the destination address at constant address mode. Use hardware DMA request for channel 2 to trigger an entire block transfer.

12.10.3.3.1.2 Timer Setup

The timer can be any general purpose timer which is capable of generating a time based DMA request.

- Setup Timer to generate DMA request associated with DMA channel 2. For example, software can setup the timer to generate a DMA request every 10 ms.

12.10.3.3.1.3 CRC Setup

- Program the pattern count MCRC_CRC_PCOUNT_REG1 to 128
- Program the sector count MCRC_CRC_SCOUNT_REG1 to 2048
- For example, we want the entire 2 Mbytes to be compressed within 5 ms. We can program the block complete timeout pre-load (MCRC_CRC_BCTOPLD1-4) value to 15625 (5 ms / (1 FICLK period × 64)) if CRC is operating at 200 MHz.
- Enable AUTO mode and all interrupts.

After AUTO mode is selected MCRC Controller automatically generates a DMA request on channel 1. Around the same time the timer module also generates a DMA request on DMA channel 2. When the first incoming data pattern arrives at the MCRC_PSA_SIGREGL1/H1, the MCRC Controller will compress it. After some time, the DMA controller would update the CMCRC_CRC_REGL1/H1 with a pre-determined value matching the calculated signature for the first sector of 128 64-bit data patterns. After one sector of data patterns are compressed, the MCRC Controller generate a CRC fail interrupt if signature stored at the MCRC_PSA_SECSIGREGL1/H1 does not match the MCRC_CRC_REGL1/H1 Register. MCRC Controller generates a DMA request on DMA channel 1 when one sector of data patterns are compressed. This routine will continue until the entire 2 Mbytes are consumed. If the timeout counter reached zero before the entire 2 Mbytes are compressed, a timeout interrupt is generated. After 2Mbytes are transferred, the DMA can generate an interrupt to CPU. The entire operation will continue again when DMA responds to the DMA request from both the timer and MCRC Controller. The CRC is performed totally without any CPU intervention.

12.10.3.3.2 Example: Auto Mode Without Using Time Based Triggering

A small but highly secured memory area with 1 Kbytes is to be checked in the background of CPU. CRC is to be performed every 1 Kbytes. Therefore, there is only one pre-recorded CRC value. For illustration purpose, we map channel 1 MCRC_CRC_REGL1/H1 to DMA channel 1 and channel 1 MCRC_PSA_SIGREGL1/H1 to DMA channel 2. Assume all transfers carried out by DMA are in 64-bit transfer size.

12.10.3.3.2.1 DMA Setup

- Setup DMA channel 1 with the source address from which the pre-determined CRC value is stored. Setup the destination address to the MCRC_CRC_REGL1 Register. Put the source address at constant addressing mode and put the destination address at constant addressing mode. Use hardware DMA request for channel 1.

Copyright © 2024 Texas Instruments Incorporated

- Setup DMA channel 2 with the source address from which the memory area to be verified. Setup the destination address to the MCRC_PSA_SIGREGL1/H1. Program the element transfer count to 128 and the frame transfer count to 1. Put the source address at post increment addressing mode and put the destination address at constant address mode. Generate a software DMA request on channel 2 after CRC has completed its setup. Enable autoinitiation for DMA channel 2.

12.10.3.3.2.2 CRC Setup

- Program the MCRC_CRC_PCOUNT_REG1 to 128
- Program the MCRC_CRC_SCOUNT_REG1 to 1
- Leaving the timeout count MCRC_CRC_BCTOPLD1 register with the reset value of zero means no timeout interrupt is generated
- Enable AUTO mode and all interrupts

After AUTO mode is selected the MCRC Controller automatically generates a DMA request on channel 1. At the same time the CPU generates a software DMA request on DMA channel 2. When the first incoming data pattern arrives at the MCRC_PSA_SIGREGL1/H1, the MCRC Controller will compress it. After some time, the DMA controller would update the MCRC_CRC_REGL1/H1 with a pre-determined value matching the calculated signature for the first sector of 128 64-bit data patterns. After one sector of data patterns are compressed, the MCRC Controller generate a CRC fail interrupt if signature stored at the MCRC_PSA_SECSIGREGL1/H1 does not match the MCRC_CRC_REGL1/H1. MCRC Controller generate a DMA request on DMA channel 1 again after one sector is compressed. After 1 Kbytes are transferred, the DMA can generate an interrupt to CPU. Responding to the DMA interrupt CPU can restart the CRC routine by generating a software DMA request onto channel 2 again.

12.10.3.3.3 Example: Semi-CPU Mode

If DMA controller is available in a system the CRC module can also operate in semi-CPU mode. This means that CPU can still make use of the DMA to perform data patterns transfer to CRC controller in the background. The difference between semi-CPU mode and AUTO mode is that CRC controller does not automatically perform the signature verification. CRC controllers generates a compression complete interrupt to CPU when the one sector of data patterns are compressed. CPU needs to perform the signature verification itself.

A memory area with 2 Mbytes is to be verified with the help of the CPU. CRC operation is to be performed every 1 Kbyte. Since there are 2 Mbytes (256 K doublewords) of memory to be check and we want to perform a CRC every 1 Kbytes (128 doublewords) and therefore there must be 2048 pre-recorded CRC values. In Semi-CPU mode, the MCRC_CRC_REGL1/H1 is not updated and contains indeterminate data.

12.10.3.3.3.1 DMA Setup

- Setup DMA channel 1 with the source address from which the memory area to be verified are mapped. Setup the destination address to the MCRC_PSA_SIGREGL1/H1. Put the starting address at post increment addressing mode and put the destination address at constant address mode. Use hardware DMA request to trigger an entire block transfer for channel 1. Disable autoinitiation for DMA channel 1.

12.10.3.3.3.2 Timer Setup

The timer can be any general purpose timer which is capable of generating a time-based DMA request.

- Setup Timer to generate DMA request associated with DMA channel 1. For example, software can setup the timer to generate a DMA request every 10 ms.

12.10.3.3.3.3 CRC Setup

- Program the MCRC_CRC_PCOUNT_REG1 to 128
- Program the MCRC_CRC_SCOUNT_REG1 to 2048
- For example, we want the entire 2 Mbytes to be compressed within 5 ms. We can program the block complete timeout pre-load value MCRC_CRC_BCTOPLD1 to 15625 (5 ms / (1 FICLK period × 64)) if CRC is operating at 200 MHz
- Enable AUTO mode and all interrupts.

The timer module first generates a DMA request on DMA channel 1 when it is enabled. When the first incoming data pattern arrives at the MCRC_PSA_SIGREGL1/H1, the CRC controller will compress it. After one sector of

Copyright © 2024 Texas Instruments Incorporated

data patterns are compressed, the CRC controller generate a compression complete interrupt. Upon responding to the interrupt the CPU would read from the MCRC_PSA_SECSIGREGL1/H1. It is up to the CPU on how to deal with the PSA value just read. It can compare it to a known signature value or it can write it to another memory location to build a signature file or even transfer the signature out of the device via SCI or SPI. This routine will continue until the entire 2 Mbytes are consumed. The latency of the interrupt response from CPU can cause overrun condition. If CPU does not read from MCRC_PSA_SECSIGREGL1 before the PSA value is overridden with the signature of the next sector of memory, an overrun interrupt will be generated by CRC controller.

12.10.3.3.4 Example: Full-CPU Mode

In a system without the availability of DMA controller, the CRC routine can be operated by CPU provided the CPU has enough throughput. CPU needs to read from the memory area from which CRC is to be performed.

A memory area with 2 Mbytes is to be checked with the help of the CPU. CRC operation is to be performed every 1 Kbyte. In CPU mode, the MCRC_CRC_REGL1/H1 is not updated and contains indeterminate data.

12.10.3.3.4.1 CRC Setup

- All control registers can be left in their reset state. Only enable Full-CPU mode.

CPU itself reads from the memory and write the data to the MCRC_PSA_SIGREGL1/MCRC_PSA_SIGREGH1 inside MCRC Controller. When the first incoming data pattern arrives at the MCRC_PSA_SIGREGL1/MCRC_PSA_SIGREGH1, the MCRC Controller will compress it. After n data patterns are compressed, CPU can read from the MCRC_PSA_SIGREGL1/MCRC_PSA_SIGREGH1. It is up to the CPU on how to deal with the PSA signature value just read. It can compare it to a known signature value stored at another memory location.

Copyright © 2024 Texas Instruments Incorporated

12.10.4 ECC Aggregator

This section describes the common ECC aggregator functionality.

12.10.4.1 ECC Aggregator Overview

To increase functional and system reliability the memories (for example, FIFOs, queues, SRAMs and others) in many device modules and subsystems are protected by error correcting code (ECC). This is accomplished through an ECC aggregator and ECC wrapper. The ECC aggregator is connected to these memories (hereinafter ECC RAMs) and involved in the ECC process. Each memory is surrounded by an ECC wrapper which performs the ECC detection and correction. The wrapper communicates via serial interface with the aggregator which has memory mapped configuration interface.

The ECC aggregator is also connected to interconnect ECC components that protect the command, address and data buses of the system interconnect. ECC is calculated for the data bus and parity and redundancy for the command and address buses. Each interconnect ECC component has the same serial interface for communication with the aggregator as the ECC wrapper. An ECC aggregator may be connected to both endpoints - the ECC wrapper and interconnect ECC component.

The ECC aggregator, ECC wrapper and interconnect ECC component are considered as single entity and are hereinafter referred to as ECC aggregator unless otherwise explicitly specified.

Table 12-441 lists the device modules and subsystems which have ECC aggregator.

Table 12-441. Device Modules and Subsystems with ECC Aggregator

Module Instance	Domain		
	WKUP	MCU	MAIN
WKUP_CBASS0	✓	-	-
WKUP_VTM0	✓	-	-
MCU_ADC0	-	✓	-
MCU_ADC1	-	✓	-
MCU_CBASS0	-	✓	-
MCU_CPSW0	-	✓	-
MCU_FSS0_HPB0	-	✓	-
MCU_FSS0_OSPI0	-	✓	-
MCU_FSS0_OSPI1	-	✓	-
MCU_I3C0	-	✓	-
MCU_I3C1	-	✓	-
MCU_MCAN0	-	✓	-
MCU_MCAN1	-	✓	-
MCU_MSRAM_1MB0	-	✓	-
MCU_NAVSS0	-	✓	-
A72SS0	-	-	✓
C71SS0	-	-	✓
CBASS0	-	-	✓
COMPUTE_CLUSTER0	-	-	✓
CPSW0	-	-	✓
CSI_RX_IF0	-	-	✓
CSI_RX_IF1	-	-	✓
CSI_TX_IF0	-	-	✓
DDRSS0	-	-	✓
DMPAC0	-	-	✓
DSS0	-	-	✓
GIC0	-	-	✓
I3C0	-	-	✓

Copyright © 2024 Texas Instruments Incorporated

Table 12-441. Device Modules and Subsystems with ECC Aggregator (continued)

Module Instance	Domain		
	WKUP	MCU	MAIN
MCAN0 to MCAN13	-	-	✓
MLBSS0	-	-	✓
MMCSD0	-	-	✓
MMCSD1	-	-	✓
MMCSD2	-	-	✓
MSRAM16KX256	-	-	✓
NAVSS0	-	-	✓
PCIE0	-	-	✓
PCIE1	-	-	✓
PCIE2	-	-	✓
PCIE3	-	-	✓
PRU_ICSSG0	-	-	✓
PRU_ICSSG1	-	-	✓
PSRAM2KECC0	-	-	✓
PSRAMECC0	-	-	✓
UFS0	-	-	✓
USB3SS0	-	-	✓
USB3SS1	-	-	✓
VPAC0	-	-	✓

Copyright © 2024 Texas Instruments Incorporated

12.10.4.1.1 ECC Aggregator Features

The ECC aggregator has the following features:

- Reduces memory software errors via single error correction (SEC) and double error detection (DED)
- Provides a mechanism to control and monitor the ECC protected memories in a module or subsystem
- SEC and DED over the system interconnect data bus and parity and redundancy for the system interconnect command and address buses
- Generates an interrupt for correctable error
- Generates an interrupt for non-correctable error
- Supports inject only mode for diagnostic purposes
- Supports software readable status for single and double-bit ECC errors and associated information such as row address where error has occurred and data bits that have been flipped
- Supports up to 256 ECC endpoints. An ECC endpoint can be either ECC RAM or interconnect ECC component.
- Detects single bit error via parity checking on:
  - Memory mapped configuration interface FIFO
  - Serial interface FIFO
  - Serial interface transaction
- Single bit error detection via parity checking results in a non-correctable error interrupt
- Supports timeout mechanism on transactions over the ECC serial interface. Timeout occurrence results in a non-correctable error interrupt.
- Certain control bits have redundancy and if a bit flips an interrupt is generated

12.10.4.1.2 ECC Aggregator Ports

Table 12-442. ECC Aggregator Clocks and Resets

Clocks	
Module Clock Input	**Description**
ECC_CLK	ECC aggregator clock
Resets	
Module Reset Input	**Description**
ECC_RST	ECC aggregator reset

Table 12-443. ECC Aggregator Hardware Requests

Interrupt Requests		
Module Interrupt Signal	**Description**	**Type**
ECC_SEC_INT	Interrupt for correctable error (SEC)	Level
ECC_DED_INT	Interrupt for non-correctable error (DED, parity, redundancy, timeout)	Level
DMA Events		
Module DMA Event	**Description**	**Type**
-	-	-

Copyright © 2024 Texas Instruments Incorporated

12.10.4.2 ECC Aggregator Functional Description

This section describes the architecture and functional details of the ECC aggregator.

Note

See Appendix Spreadsheet for ECC Aggregators Interconnect details.

12.10.4.2.1 ECC Aggregator Block Diagram

Figure 12-507 shows the ECC aggregator block diagram.

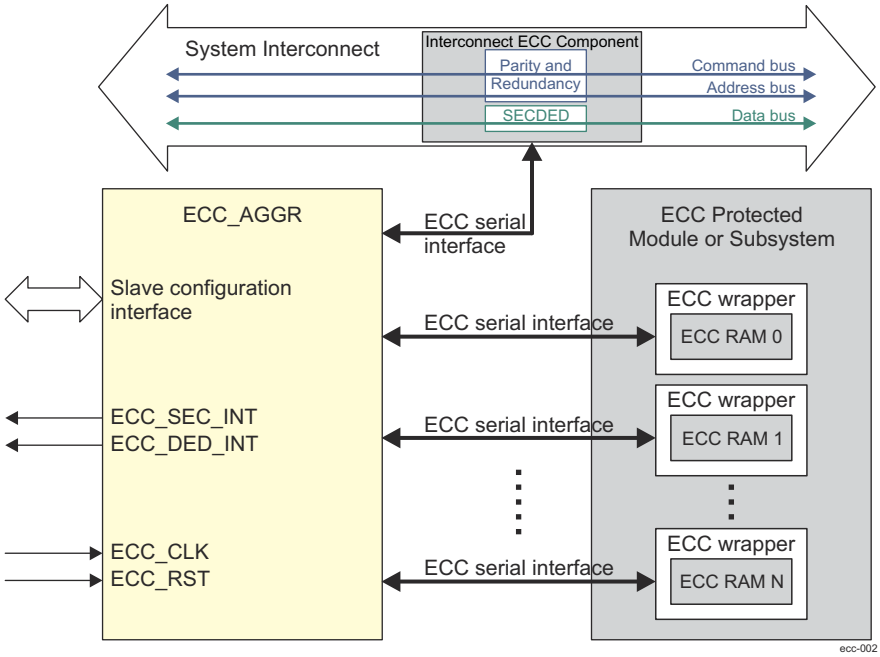


Figure 12-507. ECC Aggregator Block Diagram

The ECC aggregator is connected to one or more ECC endpoints each of which has assigned a unique ID used when the endpoint is accessed for status information or configuration. The ECC aggregator provides software access to all ECC related registers through its memory mapped slave configuration interface while the serial interface is used to communicate with the ECC endpoints. Upon detection of single or double-bit error the corresponding interrupt line is asserted.

12.10.4.2.2 ECC Aggregator Register Groups

The ECC aggregator has ECC control, status and interrupt registers for each ECC endpoint in a module or subsystem. These registers are memory mapped and occupy 1 KB address space although part of it may contain reserved locations. The registers are split in the following types:

- **Global registers**. They are common to all ECC endpoints associated with the ECC aggregator and include the ECC_VECTOR and ECC_REV registers. Each ECC endpoint has assigned a unique ID. When this ID is written to the ECC_VECTOR[10-0] ECC_VECTOR field the corresponding endpoint is selected either for control or for status reading.
- **ECC control and status registers**. These registers are specific to each ECC endpoint and reside in the range from address offset 0x10 to 0x28, if the endpoint is ECC RAM or from 0x10 to 0x24, if the endpoint is interconnect ECC component. They are memory mapped but are accessed through the ECC serial interface. They are also selected by the ECC endpoint ID written to the ECC_VECTOR[10-0] ECC_VECTOR field. Because of latency on the serial interface the ECC control and status registers are read by performing special sequence as described in Section 12.10.4.2.3. These registers have also different functionality for both types of endpoints - ECC RAM and interconnect ECC component.
- **Interrupt registers**. They include interrupt status, interrupt enable, interrupt disable, and EOI registers. For more information, see *Interrupts*.

Copyright © 2024 Texas Instruments Incorporated

12.10.4.2.3 Read Access to the ECC Control and Status Registers

Read accesses to the ECC control and status registers for each ECC endpoint represent read operations over the ECC serial interface and are triggered by performing the following sequence:

1. Software writes the following in the ECC_VECTOR register:
   - The ECC endpoint ID in the ECC_VECTOR[10-0] ECC_VECTOR field to select particular ECC endpoint.
   - The register read address in the ECC_VECTOR[23-16] RD_SVBUS_ADDRESS field to select which register has to be read through the ECC serial interface.
   - A value of 0x1 in the ECC_VECTOR[15] RD_SVBUS bit to trigger read operation through the ECC serial interface.
2. Software polls the ECC_VECTOR[24] RD_SVBUS_DONE bit to check if it is 0x1. This indicates that the read operation on the ECC serial interface has completed.
3. Software reads the data from the register previously selected by the ECC_VECTOR[23-16] RD_SVBUS_ADDRESS field.

The following is an example for serial read operation:

1. Write 0x0010 8005 to the ECC_VECTOR register. This sends read request to the ECC_WRAP_REV or ECC_CBASS_REV register (address = 0x10) associated with ECC endpoint with ID = 5.
2. Poll the ECC_VECTOR[24] RD_SVBUS_DONE bit until value of 0x1 is read.
3. Read the ECC_WRAP_REV or ECC_CBASS_REV register to get its value.

12.10.4.2.4 Serial Write Operation

Write operations over the ECC serial interface are performed as follows:

1. Software specifies the ECC endpoint ID in the ECC_VECTOR[10-0] ECC_VECTOR field. The ECC_VECTOR[23-16] RD_SVBUS_ADDRESS field is a don't care but the ECC_VECTOR[15] RD_SVBUS bit must be set to 0x0.
2. Software performs regular write operation to the desired address. If the ECC endpoint ID has already been specified, step 1 can be skipped. Unlike serial read operations it is not necessary to always specify the endpoint ID before performing serial write operation.

The following is an example for serial write operation:

1. Write 0x0000 0008 to the ECC_VECTOR register.
2. Write 0x0000 000F to the ECC_CTRL register. This sends write request with data 0x0000 000F to the ECC_CTRL register associated with ECC RAM with ID = 8.

12.10.4.2.5 Interrupts

The ECC aggregator generates the following interrupts:

- Correctable interrupt (ECC_SEC_INT) where hardware can correct the error but notifies the system in case of SEC.
- Non-correctable interrupt (ECC_DED_INT) where hardware cannot correct the error in cases of DED, parity check, redundancy check or timeout occurrence.

The following is the sequence for servicing interrupts:

- Software enables the interrupts for an ECC endpoint by writing 0x1 to the corresponding bit of the following interrupt enable registers:
  - ECC_SEC_ENABLE_SET_REG0 through ECC_SEC_ENABLE_SET_REG7 for the correctable interrupt
  - ECC_DED_ENABLE_SET_REG0 through ECC_DED_ENABLE_SET_REG7 for the non-correctable interrupt
- On receiving an interrupt, software checks which ECC endpoint has caused the error by reading the following interrupt status registers:
  - ECC_SEC_STATUS_REG0 through ECC_SEC_STATUS_REG7 for the correctable interrupt
  - ECC_DED_STATUS_REG0 through ECC_DED_STATUS_REG7 for the non-correctable interrupt
- Software performs serial read operations as described in Section 12.10.4.2.3 to read the following status registers that contain details about the error:
  - If the endpoint is ECC RAM:

Copyright © 2024 Texas Instruments Incorporated

    - ECC_ERR_STAT1
    - ECC_ERR_STAT2
    - ECC_ERR_STAT3
  - If the endpoint is interconnect ECC component:
    - ECC_CBASS_ERR_STAT1
    - ECC_CBASS_ERR_STAT2
- After the interrupt has been serviced, depending on the error type, software should clear the corresponding status bits in the ECC_ERR_STAT1 and ECC_ERR_STAT3 registers or in the ECC_CBASS_ERR_STAT1 register. Software has to poll these registers to guarantee that status bits are cleared as there is no other indication for write completion over the ECC serial interface.
  The value of the *_PEND_CLR fields in the ECC_CBASS_ERR_STAT1 register must be read and then written back to decrement the count of each field back to 0x0. A further error capture into the ECC_CBASS_ERR_STAT1 register does not occur unless all its fields are 0x0. The decrement value should not be larger than the read value. If a field in the ECC_CBASS_ERR_STAT1 register should not be modified, write a value of 0x0 to that field.
- Software writes 0x1 to the corresponding end of interrupt register to clear the interrupt:
  - ECC_SEC_EOI_REG for the correctable interrupt
  - ECC_DED_EOI_REG for the non-correctable interrupt

12.10.4.2.6 Inject Only Mode

There are modules that already perform the ECC generation and checking as part of their data path. In this case, the ECC wrapper may be configured in inject only mode, if needed. In this mode the ECC wrapper does not perform ECC detection and correction. The inject only mode allows users to inject single or double-bit errors so that the module logic can be tested for diagnostic purposes.

Note

There is no software control to enable inject only mode. It is configured via tie-off value. Inject only and ECC modes are mutually exclusive.

The interconnect ECC component also supports error injection mode. There is error injection logic for testing of the error checking logic (checkers). The injection logic can be configured to inject either single or double bit error and what data pattern to be used for injection (ECC_CBASS_CTRL[11-8] ECC_PATTERN). The ECC_CBASS_ERR_CTRL1 and ECC_CBASS_ERR_CTRL2 registers should be written first to setup the injection. Then, either the ECC_CBASS_CTRL[3] FORCE_SE or the ECC_CBASS_CTRL[4] FORCE_DE bit must be set to 0x1 to start the injection. Both bits must not be set at the same time. If the injection should continue in incrementing mode, then the ECC_CBASS_CTRL[5] FORCE_N_BIT bit should be set to 0x1. Once the FORCE_N_BIT is set, then each successive injection can simply write the ECC_CBASS_CTRL register to set the FORCE_SE or FORCE_DE again. Reading 0x0 from either the FORCE_SE or the FORCE_DE bit indicates that the injection has completed, as these bits automatically clear when the checker indicates that it has performed the injection. The time for an injection to complete is not guaranteed, so some delay is needed between successive injections.

12.10.4.2.7 ECC Error Injection Sequence

The ECC aggregator is a critical component for enhancing system reliability by protecting memories in various device modules and subsystems. The ECC Aggregator is connected to these memory and interconnect components which have the ECC, to provide access to control and monitor the ECC protected memories in a module or subsystem.

The ECC aggregator supports error injection to aid in diagnostic purposes and to evaluate robustness. The Error Injection feature supported by the ECC aggregator allows users to deliberately introduce errors into their system to test the resilience and effectiveness of error correction mechanisms. This feature helps in evaluating the robustness of the ECC aggregator by simulating real-world error scenarios, enabling users to identify potential vulnerabilities and improve their error handling strategies.

This feature includes configurable parameters such as ECC aggregator type, subtype, error type, and injection location, allowing users to customize their testing scenarios based on their specific requirements.

Copyright © 2024 Texas Instruments Incorporated

12.10.4.2.7.1 Types of Error Injection

ECC is a mechanism for providing increased protection against soft errors by allowing single bit errors to be detected and corrected and double bit errors to be detected. The ECC Control Register can be used to inject single bit or double bit errors. For more information on this register, refer to Section 12.10.4.2.2. These errors are then reported through the Error Signal Module (ESM).

- Single Error Correction (SEC): In the event of single bit errors, the ECC aggregator has the ability to both detect and correct these errors, ensuring the integrity of the data. Single bit error injection is supported for all endpoints and checkers. Single bit error correction is not supported for parity and redundancy type of checkers.
- Double Error Detection (DED): For double bit errors, the ECC aggregator can detect these errors. However, it does not have the capability to correct them. Instead, it signals the presence of these errors to the Central Processing Unit (CPU) via ESM interrupts. This is supported for all RAM wrapper types and interconnect EDC checker type of endpoints. Parity and Redundant checker types do not support double bit ECC error injection.

The ECC aggregator plays a crucial role in maintaining data accuracy by correcting single bit errors and detecting double bit errors, although it cannot correct the latter. This process helps in enhancing the overall system reliability. Each ECC aggregator connects to two kinds of ESM events – Correctable error events (for SEC events) and Un-correctable error events (for DED events and parity and redundancy checker types). ESM is used for notification of events from endpoints which are not Inject-Only. Inject-Only endpoints notify errors through IP-specific mechanisms. Sometimes a separate ESM event (not via ECC aggregator) is available. Please refer to the individual IP sections for more details on the ESM events generated.

The ECC aggregator provides support for injecting single and double bit errors. In the case of RAM wrapper type endpoints, errors can be injected to a specific row or can be requested to be injected on the next row access (n_row). Errors can be requested to be injected once or on repeated reads. For Interconnect endpoint types, single and double bit errors (depending on the checker type) can be injected.

12.10.4.2.7.1.1 ECC Wrapper Type Endpoints

Each memory is surrounded by an ECC wrapper which performs the ECC detection and correction. The wrapper communicates via serial interface with the aggregator which has memory a mapped configuration interface. For RAM wrapper types, after injecting the error, the RAM location where the error was injected needs to be accessed in order to trigger the error interrupt. If the “force_n_row” option is selected during error injection, then any RAM access will trigger the interrupt and the specific row is not needed.

12.10.4.2.7.1.2 EDC Interconnect Type Endpoints

The ECC aggregator is also connected to interconnect ECC components that protect the command, address and data buses of the system interconnect. ECC is calculated for the data bus and parity and redundancy for the command and address buses. For the interconnect types, the type of checker must be verified to determine if they are parity or redundant types.

12.10.4.2.7.1.3 Inject Only ECC Endpoints

Some modules may already do the generation of and checking of ECC as part of their data path. In this case, the ECC wrapper does not have to perform ECC detection and correction and is configured in inject only mode. The wrapper will not add extra bits to the ram, generate, nor check ECC. It will, however, allow the user to inject single or double-bit errors so that the IP’s logic can be tested for diagnostic purposes. Detection and correction occurs in the IP itself. In some cases, the error may also be routed to ESM, but not to the same ESM event as the ECC aggregator. For more information on these memories, refer to Section 12.10.4.2.6.

12.10.4.2.7.1.4 Full Functionality Error Capture ECC Endpoints

These ECC aggregators possess the ability to perform Error Injection alongside Error Capture. The wrapper can be utilized to introduce either single or double-bit errors, while also having the capability to detect and correct errors.

For more information regarding the ECC endpoint types, refer to the Appendix spreadsheet.

Copyright © 2024 Texas Instruments Incorporated

12.10.4.2.7.2 Types of ECC Checkers

There are various checkers for bus signals, MMRs, or internal registers in the interconnect.

12.10.4.2.7.2.1 Parity Type Checkers

The most common checker type is parity. The parity type refers to checkers where a parity bit is added to cover the entire signal. The parity bit is the ~^signal, where the signal width can be any number of bits.

12.10.4.2.7.2.2 Redundant Type Checkers

The redundant type refers to checkers for critical signals (like request). The redundant signal is the opposite of the signal it is protecting.

12.10.4.2.7.2.3 Error Detect and Correct (EDC) Type Checkers

The EDC type refers to SECDED error-correcting code that can correct single bit error, and detect a two bit error.

For more information regarding the ECC checker type for each aggregator, refer to the Appendix spreadsheet.

12.10.4.2.7.3 Recommendations for Testing Error Injection

The following are the general recommendations on when the error injection on ECC aggregators must be performed:

- The ECC error injection and monitoring of the error response should be run at ‘startup’ where possible.
- An alternative recommended approach is to run ECC error injection at ‘shutdown’.

When running of ECC error injection during ‘runtime’, the system integrator must be aware of the below:

For interconnect endpoints:

- The ECC SEC error injection runs during idle cycles, so can be run during ‘runtime’ for checkers of all types.
  - Parity and redundant bit error injections are non-destructive and can be done at runtime.
- The ECC DED error injection runs during idle cycles, so can be run during ‘runtime’.
  - Only single bit error injections are to be done on parity/redundant checker types. This is because these protect odd number of injections. A 2 bit injection on parity will not cause a failure.

For RAM memories:

- ECC self-test for SEC on RAM memories, can be run at runtime:
  - The ECC self-test for SEC injects a single-bit error that is corrected in-line.
- ECC DED error injection during ‘runtime’ for RAM memories, should only be done for memories with direct read access
  - Two examples of direct read access memories are MSMC SRAM and MCU SRAM
  - Double-bit error injection on all other embedded SRAMS could possibly result in the downstream IP reading bad data if run during ‘runtime’
- CC error injection on inject-only memories, should be run at ‘startup’, they are not recommended to be run during ‘runtime’. For more information on these memories, refer to Section 12.10.4.2.6.

12.10.4.2.7.4 Error Injection Programming Sequence

12.10.4.2.7.4.1 ESM Initialization

The ESM instance must beinitializedwith the following sequence. This initialization will allow the application to specify for each event whether the interrupt is enabled or disabled, the priority of the event, and whether the nErrorPin assertion is enabled or disabled for the event. This needs to be done only once. All the required ESM events can be initialized at the same time.

For more information on the ESM, refer to Section 12.10.2.

To initialize the ESM module for a specified instance:

1. Clear all raw status and enable bits by setting ESM_SFT_RST Register bits [3:0] to 0Fh, to reset the ESM
2. Write 0x1 to the appropriate bits in the ESM_ERR_STS register. This step will clear the configuration interrupt raw status

Copyright © 2024 Texas Instruments Incorporated

3. Write 0x1 to the appropriate bits in the ESM_ERR_EN_SET register, to n enable the configuration error interrupt as described in Section 12.10.2.3.1.1.
4. Write 0x1 to the appropriate bits in the error group j of the ESM_STS_j register. This step will clear the level interrupt raw status.
5. Write 0x1 to the appropriate bits in the error group j of the ESM_INTR_EN_SET_j register, to enable the level error interrupt.
6. Write 0x1 to the appropriate bits in the error group j of the ESM_INT_PRIO_j register, to indicate which interrupt the corresponding event influences (if enabled) for event group j.
7. If required, write 0x1 to the appropriate bits in the error group j of the ESM_PIN_EN_SET_j register, to set error output corresponding to the desired events.
8. Enable the global interrupts by setting ESM_EN Register bits [3:0] to 0Fh

12.10.4.2.7.4.2 ECC Initialization

1. Setting the timeout bit is required to enable notification for ECC serial interface timeout errors. The parity bit must be set to enable notification for parity errors. To enable the timeout (bit 1) and parity (bit 0) bits, write 0x1 to the ECC_AGGR_ENABLE_SET register.
2. Write 0x1 to the ECC_SEC_ENABLE_SET_REGx (to enable correctable SEC error interrupts) and ECC_DED_ENABLE_SET_REGx (to enable uncorrectable DED error interrupts) registers for required endpoints.

The following register configurations are per endpoint and done through the SVBUS programming interface. Refer to Section 12.10.4.2.4 for more information.

1. Write 0x1 to the ECC_CHECK field of the ECC_CBASS_CTRL register to enable ECC check for interconnect endpoints.
2. Write 0x1 to the ENABLE_RMW, ECC_CHECK and ECC_ENABLE bit fields of the ECC_CTRL register to enable ECC check for wrapper type endpoints.

12.10.4.2.7.4.3 Error Injection

For ECC Wrapper Type Endpoints:

1. Write to the ECC_ERR_CTRL1 Register to set the row address where FORCE_SEC or FORCE_DED needs to be applied. This is ignored if FORCE_N_ROW is set.
2. Configure the ECC_RAM_CTRL bits[6:3] register based on the type of error injection.
3. Perform a read back of the ECC_RAM_CTRL register to confirm error injection. The read value may not be same as what is written as some fields in the register are not writable or can self-clear.
4. For RAM wrapper types, after injecting the error, the RAM needs to be accessed in the row where the error was injected (if FORCE_N_ROW is used, any row can be accessed) in order to trigger the error interrupt.
5. For Inject-only Wrapper endpoints, only error injection is possible via the ECC aggregator, the detection and correction is be done by the IP itself.

Field	Name	Type	Reset	Description
0	ecc_enable	r/w	1	Enable ECC generation. **Reset to 1 with inject only configuration, but any write to this register byte clears this bit to 0. Resets to 1 in all other configurations.**
1	ecc_check	r/w	1/0	Enable ECC check. ECC is completely bypassed if both ecc_enable and ecc_check are ‘0’. **Reset to 0 with inject only configuration, and not writable. Reset to 1 in all other configurations.**
2	enable_rmw	r/w	1/0	Enable read-modify-write on partial word writes. **If disabled ecc detection and correction will no longer work and if re-enabled the ram contents must all be rewritten to correct ecc codes. Reset to 0 with inject only configuration, and not writable. Reset to 1 in all other configurations.**

Copyright © 2024 Texas Instruments Incorporated

Field	Name	Type	Reset	Description
3	force_sec	r/w	0	Force single-bit error. Cleared the cycle following the error if error_once is asserted. for write through mode this applies to writes as well as reads.
4	force_ded	r/w	0	Force double-bit error. Cleared the cycle following the error if error_once is asserted. for write through mode this applies to writes as well as reads.
5	force_n_row	r/w	0	Force single/double-bit error on the next RAM access. for write through mode this applies to writes as well as reads.
6	error_once	r/w	0	If this bit is set, the force_sec/force_ded will inject an error to the specified row only once. The force_sec bit will be cleared the cycle after the error is generated. For double-bit errors, the force_ded bit will be cleared the cycle following the double-bit error. In either case force_n_row will be cleared as well. Any subsequent reads will not force an error.
7	check_parity	r/w	1	Enables parity checking on internal data
8	Check svbus timeout	r/w	1	Enable svbus timeout mechanism
9-31	Reserved			

For ECC Interconnect Type Endpoints:

1. Clear any pending error injection by writing 0x0 to the bit fields of the ECC_CBASS_CTRL register
2. Configure the ECC_CBASS_ERR_CTRL1 and ECC_CBASS_ERR_CTRL2 registers based on the error group and type of error injection.
3. Write to the ECC_CBASS_CTRL register to inject the error.
4. For interconnect types, the error injection happens on idle cycles, so no additional access of the memory is required.

12.10.4.2.7.4.4 Error Handling

1. When an error event occurs, the user can query and collect the information about the endpoint and offset information for the location relative to the start of the associated RAM.
2. To clear the ECC error event, clear the pending status of the ECC error events by configuring the ECC_RAM_ERR_STAT1 (for Wrapper type endpoints) and SDL_EDC_CTL_ERR_STATUS1(for Interconnect type endpoints) register.
3. The ECC_SEC_EOI_REG register is used to re-trigger the pulse interrupt signal to ensure that any nested interrupt events are serviced. The software interrupt handler must write to the ECC_SEC_EOI_REG register at the end of the current interrupt processing routine, so that new events can re-trigger the pulse interrupt signal again. For level interrupt signals the ECC_SEC_EOI_REG register is not functional and must not be used.
4. Clear the ESM error interrupt using the ESM registers.

Copyright © 2024 Texas Instruments Incorporated

12.10.4.3 ECC Aggregator Registers

Table 12-444 lists the memory-mapped registers for an ECC Aggregator (ECC_AGGR).

Table 12-444. ECC_AGGR Registers

Offset	Acronym	Register Name
ECC Wrapper Registers		
10h	Section 12.10.4.3.1	ECC Wrapper Revision Register
14h	Section 12.10.4.3.2	ECC RAM Control Register
18h	Section 12.10.4.3.3	ECC RAM Error Control 1 Register
1Ch	Section 12.10.4.3.4	ECC RAM Error Control 2 Register
20h	Section 12.10.4.3.5	ECC RAM Error Status 1 Register
24h	Section 12.10.4.3.6	ECC RAM Error Status 2 Register
28h	Section 12.10.4.3.7	ECC RAM Error Status 3 Register
Interconnect ECC Component Registers		
10h	Section 12.10.4.3.8	Interconnect ECC Component Revision Register
14h	Section 12.10.4.3.9	Interconnect ECC Component Control Register
18h	Section 12.10.4.3.10	Interconnect ECC Component Error Control 1 Register
1Ch	Section 12.10.4.3.11	Interconnect ECC Component Error Control 2 Register
20h	Section 12.10.4.3.12	Interconnect ECC Component Error Status 1 Register
24h	Section 12.10.4.3.13	Interconnect ECC Component Error Status 2 Register

Copyright © 2024 Texas Instruments Incorporated

12.10.4.3.1 ECC_WRAP_REV Register (Offset = 10h) [reset = X]

ECC_WRAP_REV is shown in Figure 12-508 and described in Table 12-445.

Return to Summary Table.

ECC Wrapper Revision Register

Revision parameters.

Figure 12-508. ECC_WRAP_REV Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
REV																															
R-66A49A02h																															

LEGEND: R = Read Only; *-n* = value after reset

Table 12-445. ECC_WRAP_REV Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	REV	R	66A49A02h	TI internal data.

Copyright © 2024 Texas Instruments Incorporated

12.10.4.3.2 ECC_CTRL Register (Offset = 14h) [reset = X]

ECC_CTRL is shown in Figure 12-509 and described in Table 12-446.

Return to Summary Table.

ECC RAM Control Register

Figure 12-509. ECC_CTRL Register

31	30	29	28	27	26	25	24
RESERVED							
R-0h							
23	22	21	20	19	18	17	16
RESERVED							
R-0h							
15	14	13	12	11	10	9	8
RESERVED							CHECK_SVBUS_TIMEOUT
R-0h							R/W-1h
7	6	5	4	3	2	1	0
CHECK_PARITY	ERROR_ONCE	FORCE_N_ROW	FORCE_DED	FORCE_SEC	ENABLE_RMW	ECC_CHECK	ECC_ENABLE
R/W-1h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-1h	R/W-1h	R/W-1h

LEGEND: R = Read Only; R/W = Read/Write; *-n* = value after reset

Table 12-446. ECC_CTRL Register Field Descriptions

Bit	Field	Type	Reset	Description
31-9	RESERVED	R	0	Reserved
8	CHECK_SVBUS_TIMEOUT	R/W	1h	Enable ECC serial interface timeout mechanism 0h - Timeout mechanism disabled 1h - Timeout mechanism enabled
7	CHECK_PARITY	R/W	1h	Enables parity checking on internal data 0h - Parity checking disabled 1h - Parity checking enabled
6	ERROR_ONCE	R/W	0h	Force error only once. If this bit is set to 1h, the FORCE_SEC/FORCE_DED injects an error to the specified row only once. The FORCE_SEC bit is cleared the cycle after the error is generated. For double-bit errors, the FORCE_DED bit is cleared the cycle following the double-bit error. Any subsequent reads do not force an error.
5	FORCE_N_ROW	R/W	0h	Force error on any RAM read Force single or double-bit error on the next RAM access. For write through mode this applies to writes as well as reads.
4	FORCE_DED	R/W	0h	Force double-bit error. Cleared the cycle following the error if ERROR_ONCE is 1h. For write through mode this applies to writes as well as reads.
3	FORCE_SEC	R/W	0h	Force single-bit error. Cleared the cycle following the error if ERROR_ONCE is 1h. For write through mode this applies to writes as well as reads.

Copyright © 2024 Texas Instruments Incorporated

Table 12-446. ECC_CTRL Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
2	ENABLE_RMW	R/W	1h	Enable read-modify-write on partial word writes. 0h - Read-modify-write disabled 1h - Read-modify-write enabled **Note** **NOTE:** If disabled, ECC detection and correction does no longer work and if re-enabled the RAM contents must all be rewritten to correct ECC codes. The reset value of this bit is 0h in inject only mode and 1h in ECC mode.
1	ECC_CHECK	R/W	1h	Enable ECC check. 0h - ECC check disabled 1h - ECC check enabled **Note** **NOTE:** ECC is completely bypassed if both ECC_ENABLE and ECC_CHECK are 0h. The reset value of this bit is 0h in inject only mode and 1h in ECC mode.
0	ECC_ENABLE	R/W	1h	Enable ECC generation. 0h - ECC generation disabled 1h - ECC generation enabled

Copyright © 2024 Texas Instruments Incorporated

12.10.4.3.3 ECC_ERR_CTRL1 Register (Offset = 18h) [reset = X]

ECC_ERR_CTRL1 is shown in Figure 12-510 and described in Table 12-447.

Return to Summary Table.

ECC RAM Error Control 1 Register

Figure 12-510. ECC_ERR_CTRL1 Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ECC_ROW																															
R/W-0h																															

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 12-447. ECC_ERR_CTRL1 Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	ECC_ROW	R/W	0h	Row address where single or double-bit error needs to be applied. This is ignored if ECC_CTRL[5] FORCE_N_ROW bit is set to 1h.

Copyright © 2024 Texas Instruments Incorporated

12.10.4.3.4 ECC_ERR_CTRL2 Register (Offset = 1Ch) [reset = X]

ECC_ERR_CTRL2 is shown in Figure 12-511 and described in Table 12-448.

Return to Summary Table.

ECC RAM Error Control 2 Register

Figure 12-511. ECC_ERR_CTRL2 Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ECC_BIT2																ECC_BIT1															
R/W-0h																R/W-0h															

LEGEND: R/W = Read/Write; *-n* = value after reset

Table 12-448. ECC_ERR_CTRL2 Register Field Descriptions

Bit	Field	Type	Reset	Description
31-16	ECC_BIT2	R/W	0h	Data bit that needs to be flipped if double-bit error has to be forced. The ECC_CTRL[4] FORCE_DED bit must be set to 1h for these values to take affect.
15-0	ECC_BIT1	R/W	0h	Data bit that needs to be flipped if single-bit error has to be forced. The ECC_CTRL[3] FORCE_SEC bit must be set to 1h for these values to take affect.

Copyright © 2024 Texas Instruments Incorporated

12.10.4.3.5 ECC_ERR_STAT1 Register (Offset = 20h) [reset = X]

ECC_ERR_STAT1 is shown in Figure 12-512 and described in Table 12-449.

Return to Summary Table.

ECC RAM Error Status 1 Register

Figure 12-512. ECC_ERR_STAT1 Register

31	30	29	28	27	26	25	24
ECC_BIT1							
R-0h							
23	**22**	**21**	**20**	**19**	**18**	**17**	**16**
ECC_BIT1							
R-0h							
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**
CLR_CTRL_REG_ERR	CLR_PARITY_ERR		CLR_ECC_OTHER	CLR_ECC_DED		CLR_ECC_SEC	
R/W1C-0h	R/W-0h		R/W1C-0h	R/W-0h		R/W-0h	
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
CTR_REG_ERR	PARITY_ERR		ECC_OTHER	ECC_DED		ECC_SEC	
R/W1S-0h	R/W1S-0h		R/W1S-0h	R/W-0h		R/W-0h	

LEGEND: R = Read Only; R/W = Read/Write; R/W1C = Read/Write 1 to Clear Bit; R/W1S = Read/Write 1 to Set Bit; -*n* = value after reset

Table 12-449. ECC_ERR_STAT1 Register Field Descriptions

Bit	Field	Type	Reset	Description
31-16	ECC_BIT1	R	0h	Indicates the bit position in the RAM data that is in error. For example, a value of 1h indicates that bit [1] in the RAM data is in error. This is valid only for single-bit errors. **Note** **NOTE:** Not used in inject only mode. Always read as 0h.
15	CLR_CTRL_REG_ERR	R/W1C	0h	Clear the CTR_REG_ERR bit. A write of 1h clears this bit and the CTR_REG_ERR bit, but if the redundancy protected bits in the ECC_CTRL register have not been written to a known state to correct the error, this flag is immediately set again.
14-13	CLR_PARITY_ERR	R/W	0h	A write of a non-zero value to this field decrements the CLR_ECC_DED and ECC_DED fields by that value. If the value written is less than the current one, the non-correctable interrupt (ECC_DED_INT) stays asserted. If the value to decrement is more than the current value, the result is 0. 0h - No parity errors have occurred 1h - 1 parity error has occurred 2h - 2 parity errors have occurred 3h - 3 or more parity errors have occurred
12	CLR_ECC_OTHER	R/W1C	0h	Clear other error status. 1h indicates a successive single-bit error. Writing 1h clears the status bit.

Copyright © 2024 Texas Instruments Incorporated

Table 12-449. ECC_ERR_STAT1 Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
11-10	CLR_ECC_DED	R/W	0h	A write of a non-zero value to this field decrements it and the ECC_DED field by that value. If the value written is less than the current one, the non-correctable interrupt (ECC_DED_INT) stays asserted. If the value to decrement is more than the current value, the result is 0. 0h - No double-bit errors have occurred 1h - 1 double-bit has error occurred 2h - 2 double-bit have errors occurred 3h - 3 or more double-bit errors have occurred **Note** **NOTE:** Not used in inject only mode. Always read as 0h.
9-8	CLR_ECC_SEC	R/W	0h	A write of a non-zero value to this field decrements it and the ECC_SEC field by that value. If the value written is less than the current one, the correctable interrupt (ECC_SEC_INT) stays asserted. If the value to decrement is more than the current value, the result is 0. 0h - No single-bit errors have occurred 1h - 1 single-bit has occurred 2h - 2 single-bit have occurred 3h - 3 or more single-bit have occurred **Note** **NOTE:** Not used in inject only mode. Always read as 0h.
7	CTR_REG_ERR	R/W1S	0h	Indicates that a redundancy protected bit in the ECC_CTRL register has been flipped. This means that the redundancy logic have detected a state where not all values are the same and has defaulted to the reset state. Software needs to re-write these registers to a known state. A write of 1h sets this bit. 0h - Bit not flipped 1h - Bit flipped
6-5	PARITY_ERR	R/W1S	0h	2-bit saturating counter for the number of parity errors that have occurred since last cleared. This is also a status set register and a non-zero value sets the level interrupt. Software can also write a value to the CLR_PARITY_ERR field to decrement this counter. 0h - No parity errors have occurred 1h - 1 parity error has occurred 2h - 2 parity errors have occurred 3h - 3 or more parity errors have occurred
4	ECC_OTHER	R/W1S	0h	1h - Indicates that successive single-bit errors have occurred while a write-back is still pending. Software can also write 1h to set the pending status and write 1h to the corresponding clear bit to clear the status. **Note** **NOTE:** Not used in inject only mode. Always read as 0h.

Copyright © 2024 Texas Instruments Incorporated

Table 12-449. ECC_ERR_STAT1 Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
3-2	ECC_DED	R/W	0h	2-bit saturating counter for the number of double-bit errors that have occurred since last cleared. This is also a status set register and a non-zero value sets the level interrupt. Software can also write a value to the CLR_ECC_SEC field to decrement this counter. 0h - No double-bit errors have occurred 1h - 1 double-bit has error occurred 2h - 2 double-bit have errors occurred 3h - 3 or more double-bit errors have occurred **Note** **NOTE:** Not used in inject only mode. Always read as 0h.
1-0	ECC_SEC	R/W	0h	2-bit saturating counter for the number of single-bit errors that have occurred since last cleared. This is also a status set register and a non-zero value sets the level interrupt. Software can also write a value to the CLR_ECC_SEC field to decrement this counter. 0h - No single-bit errors have occurred 1h - 1 single-bit has occurred 2h - 2 single-bit have occurred 3h - 3 or more single-bit have occurred **Note** **NOTE:** Not used in inject only mode. Always read as 0h.

Copyright © 2024 Texas Instruments Incorporated

12.10.4.3.6 ECC_ERR_STAT2 Register (Offset = 24h) [reset = X]

ECC_ERR_STAT2 is shown in Figure 12-513 and described in Table 12-450.

Return to Summary Table.

ECC RAM Error Status 2 Register

Figure 12-513. ECC_ERR_STAT2 Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ECC_ROW																															
R-0h																															

LEGEND: R = Read Only; *-n* = value after reset

Table 12-450. ECC_ERR_STAT2 Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	ECC_ROW	R	0h	Row address where the single or double-bit error has occurred. **Note** **NOTE:** Not used in inject only mode. Always read as 0h.

Copyright © 2024 Texas Instruments Incorporated

12.10.4.3.7 ECC_ERR_STAT3 Register (Offset = 28h) [reset = X]

ECC_ERR_STAT3 is shown in Figure 12-514 and described in Table 12-451.

Return to Summary Table.

ECC RAM Error Status 3 Register

Figure 12-514. ECC_ERR_STAT3 Register

31	30	29	28	27	26	25	24
RESERVED							
R-0h							
23	22	21	20	19	18	17	16
RESERVED							
R-0h							
15	14	13	12	11	10	9	8
RESERVED						CLR_SVBUS_TIMEOUT_ERR	RESERVED
R-0h						R/W1C-0h	R-0h
7	6	5	4	3	2	1	0
RESERVED						SVBUS_TIMEOUT_ERR	WB_PEND
R-0h						R/W1S-0h	R-0h

LEGEND: R = Read Only; R/W1C = Read/Write 1 to Clear Bit; R/W1S = Read/Write 1 to Set Bit; -*n* = value after reset

Table 12-451. ECC_ERR_STAT3 Register Field Descriptions

Bit	Field	Type	Reset	Description
31-10	RESERVED	R	0	Reserved
9	CLR_SVBUS_TIMEOUT_ERR	R/W1C	0h	Clear ECC serial interface timeout error status 0h - No effect 1h - Clears this bit and the SVBUS_TIMEOUT_ERR bit
8-2	RESERVED	R	0	Reserved
1	SVBUS_TIMEOUT_ERR	R/W1S	0h	ECC serial interface timeout error. Write a 1h to set the flag 0h - No timeout error 1h - Timeout error
0	WB_PEND	R	0h	Delayed write-back pending status. 0h - An ECC data correction write-back is not pending 1h - An ECC data correction write-back is pending **Note** **NOTE:** Not used in inject only mode. Always read as 0h.

Copyright © 2024 Texas Instruments Incorporated

12.10.4.3.8 ECC_CBASS_REV Register (Offset = 10h) [reset = X]

ECC_CBASS_REV is shown in Figure 12-515 and described in Table 12-452.

Return to Summary Table.

Interconnect ECC Component Revision Register.

Figure 12-515. ECC_CBASS_REV Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
REV																															
R-Xh																															

LEGEND: R = Read Only; *-n* = value after reset

Table 12-452. ECC_CBASS_REV Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	REV	R	Xh	TI internal data.

Copyright © 2024 Texas Instruments Incorporated

12.10.4.3.9 ECC_CBASS_CTRL Register (Offset = 14h) [reset = X]

ECC_CBASS_CTRL is shown in Figure 12-516 and described in Table 12-453.

Return to Summary Table.

Interconnect ECC Component Control Register.

Figure 12-516. ECC_CBASS_CTRL Register

31	30	29	28	27	26	25	24
RESERVED							
R-0h							
23	22	21	20	19	18	17	16
RESERVED							
R-0h							
15	14	13	12	11	10	9	8
RESERVED				ECC_PATTERN			
R-0h				R/W-0h			
7	6	5	4	3	2	1	0
RESERVED		FORCE_N_BIT	FORCE_DE	FORCE_SE	RESERVED	ECC_CHECK	RESERVED
R-0h		R/W-0h	R/W-0h	R/W-0h	R-0h	R/W-1h	R-0h

LEGEND: R = Read Only; R/W = Read/Write; *-n* = value after reset

Table 12-453. ECC_CBASS_CTRL Register Field Descriptions

Bit	Field	Type	Reset	Description
31-12	RESERVED	R	0h	Reserved
11-8	ECC_PATTERN	R/W	0h	Data pattern to be used for injection. 0h = 0s 1h = Fs 2h = As 3h = 5s
7-6	RESERVED	R	0h	Reserved
5	FORCE_N_BIT	R/W	0h	Update injection fields after the injection to setup for the next incremental injection. 0h = Keep current settings after injection 1h = Increment to next bit or group after injection
4	FORCE_DE	R/W	0h	Inject a double bit error when set. Automatically cleared when injection completes.
3	FORCE_SE	R/W	0h	Inject a single bit error when set. Automatically cleared when injection completes.
2	RESERVED	R	0h	Reserved
1	ECC_CHECK	R/W	1h	Enable checkers. 0h = Disabled 1h = Enabled
0	RESERVED	R	0h	Reserved

Copyright © 2024 Texas Instruments Incorporated

12.10.4.3.10 ECC_CBASS_ERR_CTRL1 Register (Offset = 18h) [reset = X]

ECC_CBASS_ERR_CTRL1 is shown inFigure 12-517 and described in Table 12-454.

Return to Summary Table.

Interconnect ECC Component Error Control 1 Register.

This register allows setting the injection data.

Figure 12-517. ECC_CBASS_ERR_CTRL1 Register

31	30	29	28	27	26	25	24
RESERVED							ECC_BIT1
R-0h							R/W-0h
23	22	21	20	19	18	17	16
ECC_BIT1							
R/W-0h							
15	14	13	12	11	10	9	8
ECC_GRP							
R/W-0h							
7	6	5	4	3	2	1	0
ECC_GRP							
R/W-0h							

LEGEND: R = Read Only; R/W = Read/Write; *-n* = value after reset

Table 12-454. ECC_CBASS_ERR_CTRL1 Register Field Descriptions

Bit	Field	Type	Reset	Description
31-25	RESERVED	R	0h	Reserved
24-16	ECC_BIT1	R/W	0h	First bit to inject an error.
15-0	ECC_GRP	R/W	0h	Group of checker to inject.

Copyright © 2024 Texas Instruments Incorporated

12.10.4.3.11 ECC_CBASS_ERR_CTRL2 Register (Offset = 1Ch) [reset = X]

ECC_CBASS_ERR_CTRL2 is shown in Figure 12-518 and described in Table 12-455.

Return to Summary Table.

Interconnect ECC Component Error Control 2 Register.

This register allows setting the injection data.

Figure 12-518. ECC_CBASS_ERR_CTRL2 Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																							ECC_BIT2								
R-0h																							R/W-0h								

LEGEND: R = Read Only; R/W = Read/Write; *-n* = value after reset

Table 12-455. ECC_CBASS_ERR_CTRL2 Register Field Descriptions

Bit	Field	Type	Reset	Description
31-9	RESERVED	R	0h	Reserved
8-0	ECC_BIT2	R/W	0h	Second bit to inject an error. Only valid if ECC_CBASS_CTRL[4] FORCE_DE is set.

Copyright © 2024 Texas Instruments Incorporated

12.10.4.3.12 ECC_CBASS_ERR_STAT1 Register (Offset = 20h) [reset = X]

ECC_CBASS_ERR_STAT1 is shown in Figure 12-519 and described in Table 12-456.

Return to Summary Table.

Interconnect ECC Component Error Status 1 Register.

This register allows reading the captured error data.

Figure 12-519. ECC_CBASS_ERR_STAT1 Register

31	30	29	28	27	26	25	24
ERR_GRP							
R-0h							
23	22	21	20	19	18	17	16
ERR_GRP							
R-0h							
15	14	13	12	11	10	9	8
INJ_UNC_PEND_CLR		INJ_COR_PEND_CLR		UNC_PEND_CLR		COR_PEND_CLR	
R/WD-0h		R/WD-0h		R/WD-0h		R/WD-0h	
7	6	5	4	3	2	1	0
INJ_UNC_PEND		INJ_COR_PEND		UNC_PEND		COR_PEND	
R/WI-0h		R/WI-0h		R/WI-0h		R/WI-0h	

LEGEND: R = Read Only; R/WD = Read/Write to Decrement Field; R/WI = Read/Write to Increment Field; *-n* = value after reset

Table 12-456. ECC_CBASS_ERR_STAT1 Register Field Descriptions

Bit	Field	Type	Reset	Description
31-16	ERR_GRP	R	0h	Specific checker that reported the error.
15-14	INJ_UNC_PEND_CLR	R/WD	0h	Number of injected uncorrected pending interrupts (same value as UNC_PEND). Writing decrements INJ_UNC_PEND by that value.
13-12	INJ_COR_PEND_CLR	R/WD	0h	Number of injected corrected pending interrupts (same value as COR_PEND). Writing decrements INJ_COR_PEND by that value.
11-10	UNC_PEND_CLR	R/WD	0h	Number of uncorrected pending interrupts (same value as UNC_PEND). Writing decrements UNC_PEND by that value.
9-8	COR_PEND_CLR	R/WD	0h	Number of corrected pending interrupts (same value as COR_PEND). Writing decrements COR_PEND by that value.
7-6	INJ_UNC_PEND	R/WI	0h	Number of injected uncorrected pending interrupts. Writing increments by that value.
5-4	INJ_COR_PEND	R/WI	0h	Number of injected corrected pending interrupts. Writing increments by that value.
3-2	UNC_PEND	R/WI	0h	Number of uncorrected pending interrupts. Writing increments by that value.
1-0	COR_PEND	R/WI	0h	Number of corrected pending interrupts. Writing increments by that value.

Copyright © 2024 Texas Instruments Incorporated

12.10.4.3.13 ECC_CBASS_ERR_STAT2 Register (Offset = 24h) [reset = X]

ECC_CBASS_ERR_STAT2 is shown in Figure 12-520 and described in Table 12-457.

Return to Summary Table.

Interconnect ECC Component Error Status 2 Register.

This register allows reading the captured error data.

Figure 12-520. ECC_CBASS_ERR_STAT2 Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ERR_TYPE																ERR_BIT															
R-0h																R-0h															

LEGEND: R = Read Only; *-n* = value after reset

Table 12-457. ECC_CBASS_ERR_STAT2 Register Field Descriptions

Bit	Field	Type	Reset	Description
31-16	ERR_TYPE	R	0h	This field is not supported and can read any value and be ignored.
15-0	ERR_BIT	R	0h	Bit that caused the error. Always valid for EDC single bit corrected errors or redundant errors. Identifies parity segment number (but not the exact bit) with a parity error. This field is not valid for EDC double bit errors.

13 On-Chip Debug

This chapter describes the on-chip debug support.

Note

This chapter is under development and is included as a placeholder for future updates.

14 Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from July 31, 2024 to September 30, 2024 (from Revision D (July 2024) to Revision E (September 2024)) **Page**

- Updated Device Top-level Block Diagram..... 8
- Updated X.509 Certificate information from Optional..... 210
- Updated Internal RC Oscillator section..... 333
- Added R5FSS_RAT Registers section..... 458
- Added Main Navigator Subsystem (NAVSS) Ports section..... 845
- Added MCU NAVSS Ports section..... 853
- Added Unified DMA Controller (UDMA) Ports section..... 858
- Added Ring Accelerator (RINGACC) Ports section..... 870
- Added Proxy Ports section..... 880
- Added Secure Proxy Ports section..... 883
- Added Interrupt Aggregator (INTR_AGGR) Ports section..... 889
- Added Peripheral DMA (PDMA) Ports section..... 907
- Added Data Routing Unit (DRU) Ports section..... 968
- Update CPTS_TS_COMP_LEN_REG from 24 to 32 bits..... 983
- Update CPTS_TS_COMP_LEN_REG from 24 to 32 bits..... 983

Copyright © 2024 Texas Instruments Incorporated

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2024, Texas Instruments Incorporated