Technical Reference Manual

AM263P Technical Reference Manual

Table of Contents

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.

Glossary

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

Trademarks

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.

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.

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.

Release History

The following table summarizes the AM263Px Technical Reference Manual (TRM) and associated Register Addendum (RA) release versions.

Release	Date	TRM Version	RA Version
*	September 2023	SPRUJ55 SPRUJ56	SPRUJ57

1 Introduction

This chapter introduces the features, subsystems, and architecture of the AM263Px Sitara MCU Processor Platform high-performance System-on-Chip (SoC).

Note

This document describes the superset architecture, processors, and peripherals of the AM263Px Family of SoCs, which are part of the Sitara MCU Processors Multicore SoC architecture platform. Not all features are available on each family of devices. The superset AM263Px device will be available for preproduction software development. Software should constrain the features used to match the intended production device. For more information on the specific modules and features available on a particular device, refer to the device comparison table in the corresponding device-specific Datasheet.

The AM263Px Sitara Processor Platform is hereinafter commonly referred to as *AM263Px*, *platform*, *device*, *chip*, or *SoC*.

1.1 Overview

The AM263Px Sitara Arm® Microcontrollers are built to meet the complex real-time processing and control needs of next generation industrial and automotive embedded projects. AM263Px uniquely combines advanced compute with industry leading real-time control peripherals to meet the growing performance needs of applications such as HEV/EV (traction inverters, on-board chargers, and DC-DC converters), motor drives, renewable energy, energy storage, and other general real-time constrained systems. AM263Px combines up to four Cortex-R5F MCUs, a real-time control subsystem (CONTROLSS), a Hardware Security Module (HSM), and one instance of Sitara's PRU-ICSS, making AM263Px designed for advanced motor control and digital power control applications.

For multicore AM263Px devices, the R5F cores are arranged in clusters of two Cortex-R5F cores per cluster. Each Cortex-R5F core has 256KB of shared tightly coupled memory (TCM). AM263Px has 3MB of shared SRAM spread across 6 banks of 512kB each. The multiple Arm® cores are configured to be in lockstep mode after device reset. They can be optionally programmed by the bootlooder to run in dual core mode instead. Extensive ECC is included with the on-chip memory, peripherals, and interconnect for enhanced reliability. The HSM on AM263Px provides cryptographic acceleration, secure boot, and manages granular firewalls, enabling developers to design the most secure systems. AM263Px also includes a Trigonometric Math Unit (TMU) on each R5F core for accelerating trigonometric functions.

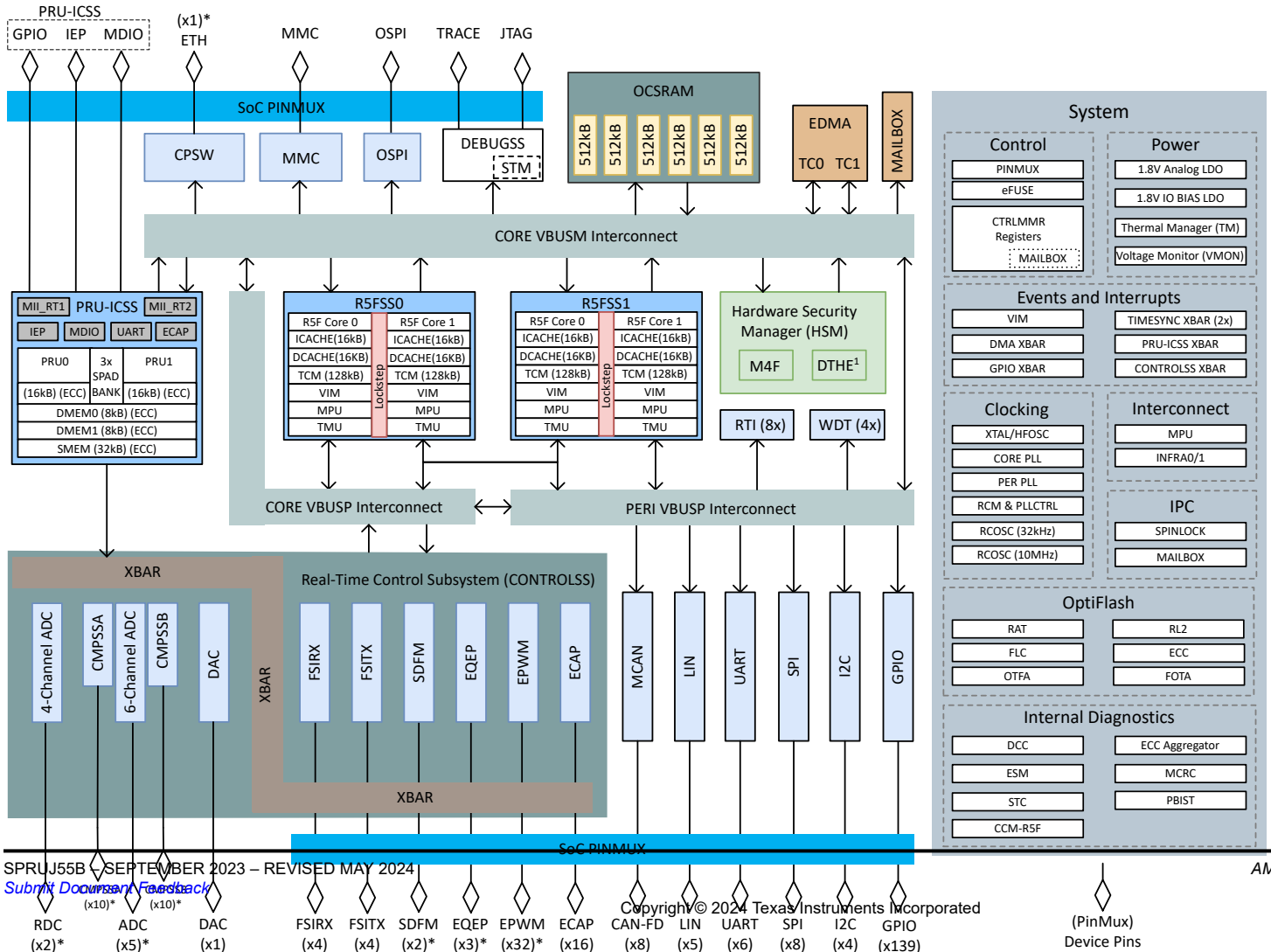
The Real-Time Control Subsystem (CONTROLSS) is a revolutionary subsystem integrated into the device. CONTROLSS contains multiple digital and analog control peripherals including: ADC, Resolver-ADC, CMPSS, EPWM, ECAP, and EQEP, among others to enable efficient execution of critical sense/process/actuate real-time signal chain control loops. The integrated crossbar (XBAR) infrastructure enables flexible configuration and routing of external signals to internal ports and internal signals to external pins.

The Flash Subsystem (FSS) provides access to external Flash devices via Octal Serial Peripheral Interface (OSPI). The OpTIflash module features Region Based Address Translation (RAT), Fast Local Copy (FLC), and Remote L2 cache (RL2) functions that greatly improve Flash performance on the device.

The PRU-ICSS in AM263Px provides the flexible industrial communications capability necessary to run advanced Ethernet protocols such as EtherCAT®, PROFINET®, and Ethernet/IP™, or the PRU-ICSS can be used for standard Ethernet connectivity and custom I/O interfacing. The PRU-ICSS supports two Ethernet Ports at 10/100 Mbit operation. It also enables additional interfaces in the SoC including sigma delta decimation filters and absolute encoder interfaces. In addition to the PRU-ICSS, the Common Platform Switch (CPSW) interface provides up to two Ethernet ports that can support up to 10/100/1000 Mbit operation and supports standard Ethernet connectivity.

TI provides a complete set of microcontroller software and development tools for the AM263Px family of microcontrollers in addition to multiple pin-to-pin compatible devices for scalability and ease of use.

1.2 Device Block Diagram



Note

*See the AM263Px Device Comparison table for specific peripheral instance counts.

1.3 Module Allocation and Instances

Module Abbreviation	Module Full Name	Device Instances
SOC Modules		
R5FSS	Dual Core Arm Cortex-R5F Subsystem	2 dual core R5FSS, total of 4 cores
PRU-ICSS	Programmable Real-time Unit Subsystem	1
HSM	Hardware Security Manager (M4F-based Subsystem)	1
SPINLOCK	Interprocessor Communication - Spinlock	1
MAILBOX	Interprocessor Communcation - Mailbox	1
EDMA	Enhanced DMA	1x (2x TC + 1x CC)
DEBUGSS	On-Chip Debug	1
General Connectivity Peripherals		
GPIO	General Purpose Input/Output	4 (1 per Cortex-R5F) 139x Total GPIO Pins
I2C	Inter-Integrated Circuit	4
SPI	Serial Peripheral Interface	8
UART	Universal Asynchronous Receiver/Transmitter	6
High-speed Serial Interfaces		
CPSW	2x External Port Gigabit Ethernet Switch	1
Industrial and Control Interfaces		
MCAN	Controller Area Network Interface	8
LIN	Local Interconnect Network	5
Memory Interfaces		
OSPI	Octal Serial Peripheral Interface	1
OCSRAM	On-Chip Static Random Access Memory	1
MMC	Multi-Media Card/Secure Digital (4-bit) Interface	1
Timer Modules		
WWDT	Real Time Interrupt/Windowed WatchDog Timer	4 (1 per Cortex-R5F)
RTI	Real Time Interrupt Timer	8
Internal Diagnostics Modules		
DCC	Dual Clock Comparator	4
ESM	Error Signaling Module	1
MCRC	Memory Cyclic Redundancy Check Controller	1
CCM-R5F	CPU Compare Module for Cortex-R5F	2
STC	Self-Test Controller	2
PBIST	Programmable Built-In Self Test	1
ECC	ECC Aggregator	1x-SoC 4x-R5FSS 1x-ICSSM 4x-MCAN 1x-CPSW3G
Real-time Control Subsystem (CONTROLSS)		
Analog Control Peripherals		
ADC	Analog to Digital Converter	5 (6 Channels per ADC)

Module Abbreviation		Module Full Name	Device Instances
Resolver [1] (ADC12B3M)	RDC	Resolver to Digital Converter	2
	ADC	General Purpose Analog to Digital Converter	2 (4 Channels per ADC)
CMPSSA		Comparator Subsystem A	10 (2x/ADC)
CMPSSB		Comparator Subsystem B	10 (2x/ADC)
DAC		Buffered Digital to Analog Converter	1
Digital Control Peripherals			
EPWM		Enhanced Pulse Width Modulation Module	32
EQEP		Enhanced Quadrature Encoder Pulse Module	3
ECAP		Enhanced Capture Module	16
SDFM		Sigma-Delta Filter Module	2
FSI		Fast Serial Interface (RX/TX)	4x RX 4x TX
Crossbar Modules			
INPUTXBAR		Flexible Signal Multiplex Input Crossbar	1
OUTPUTXBAR		Flexible Signal Multiplex Output Crossbar	1
DMAXBAR		EDMA Data Movement Architecture Crossbar	1
PWMXBAR		PWM Signal Crossbar	1
PWMSYNCOUTXBAR		PWM Sync Output Crossbar	1
MDLXBAR		Minimum Dead-band Logic (MDL) Crossbar	1
DELXBAR		Diode Emulation Logic (DEL) Crossbar	1
ICLXBAR		Illegal Combo Logic (ICL) Crossbar	1
INTXBAR		Peripheral Interrupt Crossbar	1

(1) ZCZ-S, ZCZ-F package only. Resolver may only be used as ADC or RDC. If ADC is not used in resolver mode, it can be used as General Purpose ADC.

AM263Px Register Addendum Link

A Register Addendum PDF has been created in order to make the Technical Reference Manual a more effective and size-efficient collateral document, the AM263Px Register Addendum can be downloaded at https://www.ti.com/lit/pdf/SPRUJ57.

1.4 Device Modules

This section describes the modules integrated in the device.

Arm Cortex-R5F Processor (R5FSS)

The ARM Dual-Core Cortex-R5F processor subsystem (R5FSS) supports the following main features:

- Armv7-R architecture
- Supported modes of operation (boot-time configurable):
  - Dual Core mode: two independent free-operating cores (Asymmetric Multi-Processing, no coherence)
  - Single Core mode: one free-operating core and one non-operating core
  - Lockstep mode: one free-operating core and a lockstep core for safety-enabled applications
    - There is a two clock cycle delay between CORE0 and CORE1 in lockstep mode. Any errors are routed to the Error Signal Module (ESM) which in turn is routed as an interrupt to the CPU. The ESM is also available as an I/O pin which can be used for external monitoring. See the *Error Signal Module* chapter for more details.
- R5FSS Memory System

  - 16KB per CPU Instruction Cache
    - 4x4KB ways
    - SECDED ECC protected per 64 bits
  - 16KB per CPU Data Cache
    - 4x4KB ways
    - SECDED ECC protected per 32 bits
  - 256KB tightly-coupled memory (TCM) per CPU
    - SECDED ECC protected per 32 bits
    - TCM hard error cache Implemented in CPU
    - Readable/writable from system
    - Configurable reset initialization values through the CTRLMMR
    - 64KB TCMA (ATCM) in lock-step and single core mode
    - 192KB TCMB (BTCM) in lock-step and single core mode
      - TCMB is split equally between B0 and B1 interleaved banks
    - 32KB TCMA (ATCM) for each core in split-core mode
    - 96KB TCMB (BTCM) for each core in split-core mode
      - TCMB is split equally between B0 and B1 interleaved banks
- Full-precision Floating Point (VFPv3)
- 4/8/16-region Memory Protection Unit (MPU)
- 8 breakpoints, 8 watch points
- CoreSight Debug Access Port (DAP)
- CoreSight ETM-R5 interface (CTI, ETM, ATB)
- Performance Monitoring Unit (PMU)
- Integrated Vectored Interrupt Manager (VIM) per core with 256 Interrupt Inputs each
  - Programmable interrupt priority (4-bit)
  - Programmable interrupt enable mask
  - Software-generated interrupts
  - Synchronous clock domain crossing on all core interfaces

Note

The operating cores can be configured to use the full TCM memory space available to both cores.

In Dual Core mode, CORE0 and CORE1 each have 128KB of TCM:
- 32KB TCMA
- 48KB TCMB0 + 48KB TCMB1

In Single Core and Lockstep mode, CORE0 has 256KB of TCM :
- 64KB TCMA
- 192KB TCMB (96KB TCMB0 + 96KB TCMB1)

- Trigonometric Math Unit (TMU)
  - Each R5F subsystem contains two TMU modules
  - Speeds up the execution of common trigonometric and arithmetic operations
  - Minimum frequency: 200MHz
  - Operate in lock-step or dial core mode
- Region Address Translator (RAT)
  - Minimum granularity of 4KB
- Fast Local Copy (FLC) Engine
  - Accelerate boot-up
  - Support up to four flash regions per FLC with 64KB granularity

Note

These details describe a superset of the R5FSS memory configuration. For additional details on device memory availability, please refer to the device-specific Datasheet.

1.4.1 Programmable Real-Time Unit and Industrial Communication Subsystem (PRU-ICSS)

One instance of the Programmable Real-Time Unit and Industrial Communication Subsystem (PRU-ICSS) allows implementation of various high-performance industrial control algorithms and industrial interface standards such as PROFINET™ and EtherCAT®.

The PRU-ICSS subsystem supports the following main features, among others:

- One Programmable Real-time Unit Subsystems (PRUSS):
  - 2x PRU (PRU0/PRU1)
- 32KB shared general purpose RAM with ECC
- Two 8KB data memories with ECC
- Up to two 10/100 Ethernet Ports
- One Industrial Ethernet Peripheral (IEP) module to manage/generate Industrial Ethernet functions
- One 16550-compatible UART module, with a dedicated 192 MHz to support 12 Mbps PROFIBUS®
- One Industrial Ethernet 64-bit timer, with 10 capture and 16 compare events, along with slow and fast compensation
- One Enhanced Capture (ECAP) module
- One interrupt controller (INTC) with 160 input events supported – 96 external, 64 internal
- ECC support for all internal memories

Among the interfaces supported by the PRU-ICSS are real-time industrial protocols used in Controller and Peripheral mode, such as:

- EtherCAT®
- PROFINET™
- EtherNet/IP™
- PROFIBUS®

Note

See device-specific datasheet for more details related to industrial protocol support.

1.4.2 Hardware Security Module (HSM)

One Hardware Security Module (HSM) to facilitate the device security-related functionality:

- Arm Cortex M4F Core (200 MHz)
- 1x Real-time Interrupt (RTI) module
- 1x RTI/WWDT module used as watchdog (WD mode)
- 2x Timers
  - 32-bit up counter
  - Cascading mode support for 2x 64 bit counters
- HSM Mailbox for Messaging between HSM and host processors.
- Designated HSM DMA to fetch and store the data for cryptography services.
- Hardware Security Accelerators (200MHz)
  - Symmetric Encryption/Decryption
    - AES: 128, 192 and 256-bits key
    - Cipher modes ECB, CTR, CBC, GCM
    - CBC-MAC, CMAC based on AES
  - Asymmetric Cryptography
    - High-performance PKA (public key engine) for large vector math/modulus operation
    - RSA2048, RSA3092, RSA4096
    - ECC Secp/NIST Curves: Curve25519, X25519, secP256r1, secP256k1, secP384r1, secP384k1, Brain pool, and others.
  - Hashing HMAC

    - SHA2 – 256, 384 and 512-bit support
    - HMAC-SHA256, HMAC-SHA512 – Keyed Hashing
  - Random Number Generator
    - Deterministic Random Bit Generator (DRBG) with Pseudo and True Random Number Generation (PRNG / TRNG) support
    - Capability to seed the PRNG with TRNG seed

1.4.3 Real-time Control Subsystem (CONTROLSS)

The integrated real-time Control Subsystem (CONTROLSS) enables closed loop control systems with flexible interconnection between data acquisition, actuator modules, and other control signal resources. The CONTROLSS module consists of the following control peripherals:

Analog Control Peripherals

- 5x Analog to Digital Converter (ADC) modules
  - 12-bit resolution with 4MSPS sample rate
  - Programmable 6x single-ended or 3x differential channels
  - 3.2V full scale voltage range with 1.8V reference (32/18 internal input scaling)
  - Support for internal or external 1.8V ADC VREF reference voltage (2% internal reference accuracy error)
  - Two common external calibration pins for all ADCs
  - 4x Post-processing blocks per ADC
  - 12x Simultaneous Compare Blocks (ADC Safety Tiles)
  - Multiple ADC trigger sources including CPU timers, GPIO/Input XBAR, and EPWM SOCa/SOCb signals.
- 1x Resolver with 2x dedicated SAR ADCs configurable in the following modes:
  - 2x motor position sensing units
  - 2x General Purpose ADCs with 4x channels, 12-bit resolution with 3MSPS sample rate
- 1x Buffered Digital to Analog (DAC) module
  - 12-bit resolution
  - Support for internal or external 1.8V DAC VREF reference voltage (2% internal reference accuracy error)
- 10x Comparator Subsystem A (CMPSSA)
  - Each instance has 2 comparators + 2 DACs
  - Each instance supports the window comparison of one input (uses both comparators) OR
  - Compare two inputs OR
  - Single threshold compare of a single input
- 10x Comparator Subsystem B (CMPSSB)
  - Each instance has 2 comparators + 2 DACs
  - Each instance supports the window comparison of one input (uses both comparators) OR
  - Single threshold compare of a single input

Digital Control Peripherals

- 32x Enhanced Pulse-width Modulation (EPWM) modules
- 16x Enhanced Capture (ECAP) modules
- 2x Sigma-Delta Filter (SDFM) modules
- 3x Enhanced Quadrature Encoder Pulse (EQEP) modules
- 4x Fast Serial Interface Transmitter (FSITX) modules
- 4x Fast Serial Interface Receiver (FSIRX) modules

1.4.4 Spinlock (SPINLOCK)

One Spinlock module with (256 hardware semaphores) for synchronizing the processes running on multiple cores in the device.

1.4.5 Enhanced Data Movement Architecture (EDMA)

One Enhanced Data Movement Architecture (EDMA) module can be used for efficient transfer of data and support between software, firmware, and hardware in all combinations. The EDMA consists of a single Channel Controller (TPCC) and two Transfer Controllers (TPTC) to enable various data movement requirements.

The **TPCC** is a high flexible channel controller that serves as both a user interface and an event interface for the EDMA controller. The EDMA_TPCC serves to prioritize incoming software requests or events from peripherals, and submits transfer requests (TRs) to the transfer controller.

The **TPTC** performs read and write transfers by EDMA ports to the target peripherals, as programmed in the Active and Pending set of the registers. The transfer controllers are responsible for data movement, and issue read/write commands to the source and destination addresses programmed for a given transfer in the EDMA_TPCC.

The **EDMA_TPCC** channel controller has the following features:

- Fully orthogonal transfer description:
  - Three transfer dimensions
  - A-synchronized transfers: one dimension serviced per event
  - AB-synchronized transfers: two dimensions serviced per event
  - Independent indexes on source and destination
  - Chaining feature allowing a 3-D transfer based on a single event.
- Flexible transfer definition:
  - Increment or FIFO transfer addressing modes
  - Linking mechanism allows automatic PaRAM set update
  - Chaining allows multiple transfers to execute with one event
- Interrupt generation for the following:
  - Transfer completion
  - Error conditions
- Debug visibility:
  - Queue water marking/threshold
  - Error and status recording to facilitate debug
- 64 DMA request channels:
  - Event synchronization
  - Manual synchronization (CPUs write to event set registers EDMA_TPCC_ESR and EDMA_TPCC_ESRH).
  - Chain synchronization (completion of one transfer triggers another transfer).
- Eight QDMA channels:
  - QDMA channels trigger automatically upon writing to a parameter RAM (PaRAM) set entry.
  - Support for programmable QDMA channel to PaRAM mapping.
- Each PaRAM set can be used for a DMA channel, QDMA channel, or link set.
- Multiple transfer controllers/event queues.
- 16 event entries per event queue.

The **EDMA_TPTC** transfer controller has the following features:

- 128-bit wide read and write ports per TC
- Supports two-dimensional transfers with independent indexes on source and destination (EDMA_TPCC manages the third dimension)
- Support for increment or constant addressing mode transfers
- Interrupt and error support
- Memory-Mapped Register (MMR) bit fields are fixed position in 32-bit MMR regardless of endianness

1.4.6 General Purpose Input/Output Interface (GPIO)

Four General Purpose Input/Output (GPIO) modules, each dedicated to a specific R5FSS core. These provide dedicated general-purpose pins that can be configured as either inputs or outputs. The GPIO module main features include:

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

  - Each processor core has a separate module for controlling GPO pins and observing GPI pins
  - IOMUX CTRLMMR register-based 4:1 multiplexer to individually assign GPO pin control to a specific processor core
  - GPI pins are observable by all processor cores
- Support for GPI signal conditioning chain
  - Invert/Non-invert
  - Signal Qualification
    - Asynchronous input
    - Synchronise to SYSCLK
    - Qualification using sampling window
- Software-based tristate control to emulate open-drain IO mode

1.4.7 Inter-Integrated Circuit Interface (I2C)

Four instances of the multi-controller Inter-Integrated Circuit (I2C) interface module, each with the following main features:

- 1x Instances with open-drain voltage buffers in compliance with 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
- 8-bit-wide data access
- Support of multi-controller transmitter/peripheral receiver and receiver/peripheral transmitter modes
- Built-in FIFOs with programmable size of 8 to 64 bytes for buffered read or write

1.4.8 Serial Peripheral Interface (SPI)

Eight instances of the Serial Peripheral Interface (SPI) module with 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 two channels in controller mode, or single channel in receiver mode
- Support for various controller 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

1.4.9 Universal Asynchronous Receiver/Transmitter (UART)

Six instances of the configurable Universal Asynchronous Receiver/Transmitter (UART) interface module with the following main features:

- 16C750-compatible interface
- 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-rate from 300 bits/s up to 3.6864 Mbits/s 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

Note

Only one UART instance has support for support full modem control functions. All other UART instances will support only the TX, RX, RTS, and CTS signals.

1.4.10 3-port Gigabit Ethernet Switch (CPSW)

One instance of the 3-port Gigabit Ethernet Switch (CPSW) subsystem provides Ethernet packet communication for the device. The CPSW subsystem provides the following main features:

- Two Ethernet ports (Port 1/Port 2) with selectable MII, RMII, and RGMII interfaces and a single internal Communications Port Programming Interface (CPPI) port (Port 0)
- Synchronous 10/100/1000 Mbit operation with Flexible logical FIFO-based packet buffer structure
  - Full duplex mode supported in 10/100/1000 Mbps modes
  - Half-duplex mode supported in 10/100 Mbps modes only
- Maximum frame size of 3024 bytes
- Management Data Input/Output (MDIO) module for PHY Management with Clause 45 support
- Programmable interrupt control with selected interrupt pacing
- One CPDMA CPPI 3.0 DMA Host Interface (Port 0)
- Emulation Mode, Digital loopback, and FIFO loopback modes supported
- RAM Error Detection and Correction (SECDED)
- 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)
- Non-Blocking switch fabric with Flow Control Support (802.3x) and Wire rate switching (802.1d)
- Time Sensitive Network (TSN) Support
  - IEEE 802.1Qbv Enhancements for Scheduled Traffic
- Address Lookup Engine (ALE) with 512 ALE table entries
- EtherStats and 802.3 Stats Remote Network Monitoring (RMON) statistics gathering (per port statistics)
- Support for Ethernet MAC transmit to MAC receive digital loopback mode

1.4.11 Octal Serial Peripheral Interface (OSPI)

One instance of the Octal Serial Peripheral Interface (OSPI) with support for the following main features:

- Support for Single, Dual, Quad, or Octal SPI mode of operation
- Eleven pin interface for single OSPI device: DQS, DQ0, DQ1, DQ2, DQ3, DQ4, DQ5, DQ6, DQ7, CSn0, CLK
  - Optional pins: RESET_OUT0, ECC_FAIL, LBCLKO
  - Support for multiple OSPI devices: CSn1, RESET_OUT1
- Up to 133MHz SDR frequency and 133MHz DDR frequency
- Prefetcher support
- Memory mapped direct mode of operation for flash data transfer and XIP from flash
- Software triggered indirect mode of operation
- 256B buffer for indirect transfers
- Full XIP support with no limitation on R5 Cache miss wrapping burst
- Two Chip Select signals to connect up to two external flash devices
- PHY and Tap Mode support
- Support for on the fly encryption and authentication(OTFA). OTFA enables external memory IP protection at runtime during XIP. Option to disable OTFA. Support for 4Byte and 8 Byte MAC per 32 Byte data for increased security
- Support concurrent direct read and write requests to enable Firmware over the air update(FOTA) during XIP.

1.4.12 Multi-Media Card/Secure Digital Interface (MMCSD)

One Multi-Media Card/Secure Digital (MMCSD) controller module with the following features:

- One controller with 4-bit wide data bus
- Support of MMC 4.3 Host Specification
- Support of SD Host Controller Standard Specification - SDIO 2.00
- Multi-Media card features:
  - 3.3v legacy modes with 1-bit single data rate (0-24MHz clock)
  - 3.3v HS-SDR with 4-bit bus width (0-48MHz Clock)
- SD card support:
  - DS mode (1/4-bit, 3.3V): up to 12 MBps (24 MHz clock)
  - HS mode (1/4-bit, 3.3V): up to 24 MBps (48 MHz clock)
- Supports Card Detect (SDCD) and Write Protect (SDWP)

1.4.13 Controller Area Network (MCAN)

Eight Controller Area Network interfaces (MCAN) with support for classic CAN and CAN FD (CAN with Flexible Data-Rate) specifications. The MCAN module consists of 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
- AUTOSAR and SAE J1939 support
- Loopback mode for self-test
- 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
- Two interrupt lines with support for maskable interrupts
- Timestamp Counter

1.4.14 Local Interconnect Network (LIN)

Ten instances of the configurable Local Interconnect Network (LIN) interface module with the following main features:

- 16C750-compatible
- Compatibility with LIN 1.3, 2.0, and 2.1 protocols
- Enhanced Baud Rate Generated configurable up to 20 kpbs
  - 2^{31} programmable transmission rates with 7 fractional bits
- Two external pins: LINRX and LINTX.
- Multi-buffered receive and transmit units
- Automatic wake-up support and bus idle detection
- Support for common Error Detection methods

1.4.15 Timers

Two sets of timer modules are instantiated in the device:

- Eight RTI Timer instances, implemented by the Real-time Interrupt function of the RTI/WWDT module.
- Four Windowed Watchdog Timer (WWDT) instances, implemented by the Digital Windowed Watchdog (DWWD) function of the RTI/WWDT module
- The RTI/WWDT provides 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 pre-scale counter or by an external event
  - Selectable RTI clock input (derived from any of the available clock sources)
  - Fast enabling/disabling of events

1.4.16 Internal Diagnostics Modules

Instantiated in the device are various internal diagnostics modules which provide on-chip monitoring and diagnostic functions required to achieve certain safety compliance levels:

- Four 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 spec
  - Clock frequency measurement
- One Memory Cyclic Redundancy Check (MCRC) module to enable hardware-based CRC calculations.
- Integrated on-die temperature monitor (+/- 8º C temperature accuracy)

- One instance of 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 signal routed out of device through MCU_ESM error signal
  - Configurable time base for error signal
  - Error forcing capability
  - Internal redundant flops on safety critical fields
- Multiple 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
  - 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)

1.5 Device Identification

The device part number identification data can be read in the TOP_CTRL.EFUSE_JTAG_USERCODE_ID register. See Table 1-1 for more information.

Table 1-1. Device Part Number Identifier

TOP_CTRL.EFUSE_JTAG_USERCODE_ID Register Field	Value and Description	Comment
[31-13] DEVICE_ID	Base Part Number	Refer to the Device Comparison section for the DEVICE_ID value of a given part number.
[12] SAFETY	0 = Non Functional Safety 1 = Functional Safety	
[11] PACKAGE	Package 0x06 = ZCZ Others = Reserved	
[10-6] SPEED	Device Speed Grade 0x0E (Grade N): 400 MHz R5F 2MB (Full speed and MIN memory) 0x0F (Grade O): 400 MHz R5F 3MB (Full speed and full memory) 0x10 (Grade P): 200 MHz R5F 3MB (Half speed and full memory)	Refer to the Operating Performance Points section for the supported speed grades and the definitions for a given device.
[5-3] TEMP	Temperature Grade 0x05 = -40°C to 125°C 0x07 = -40°C to 150°C Others = Reserved	Operating junction temperature range.
[2-0] FEATURE	Package Feature 0x01 = AM263x compatible package 0x02 = Sensor package + FLASH-in-Package 0x05 = Sensor package	

The manufacturer identity, the boundary scan part number, and the silicon revision of the device can be read from the configuration port via JTAG.

2 Memory Map

This chapter summarizes the memory map address regions for the device.

2.1 Device Memory Map

This section describes the device memory map.

Note

The memory locations not shown are either unallocated or reserved and not used.

Accesses to these locations are not recommended and must be avoided.

Region Name	Start Address	End Address	Size
Core-specific Internal Memory Map[1]	0h0000 0000	0h1FFF FFFF	537 MB
MCRC0	0h3500 0000	0h3500 0FFF	4 KB
STM_STIM0	0h3900 0000	0h39FF FFFF	16 MB
MPU_L2OCRAM_BANK0	0h4002 0000	0h4002 03FF	1 KB
MPU_L2OCRAM_BANK1	0h4004 0000	0h4004 03FF	1 KB
MPU_L2OCRAM_BANK2	0h4006 0000	0h4006 03FF	1 KB
MPU_L2OCRAM_BANK3	0h4008 0000	0h4008 03FF	1 KB
MPU_R5SS0_CORE0_AXIS	0h400A 0000	0h400A 03FF	1 KB
MPU_R5SS0_CORE1_AXIS	0h400C 0000	0h400C 03FF	1 KB
MPU_R5SS1_CORE0_AXIS	0h400E 0000	0h400E 03FF	1 KB
MPU_R5SS1_CORE1_AXIS	0h4010 0000	0h4010 03FF	1 KB
MPU_MBOX_SRAM	0h4014 0000	0h4014 03FF	1 KB
MPU_FSS_DATA	0h4016 0000	0h4016 03FF	1 KB
MPU_SCRM2SCRP0	0h4018 0000	0h4018 03FF	1 KB
MPU_SCRM2SCRP1	0h401A 0000	0h401A 03FF	1 KB
MPU_R5SS0_CORE0_AHB	0h401C 0000	0h401C 03FF	1 KB
MPU_R5SS0_CORE1_AHB	0h401E 0000	0h401E 03FF	1 KB
MPU_R5SS1_CORE0_AHB	0h4020 0000	0h4020 03FF	1 KB
MPU_R5SS1_CORE1_AHB	0h4022 0000	0h4022 03FF	1 KB
MPU_FSS_CONFIG	0h4026 0000	0h4026 03FF	1 KB
MPU_R5SS0	0h4028 0000	0h4028 03FF	1 KB
MPU_R5SS1	0h402A 0000	0h402A 03FF	1 KB
MPU_L2OCRAM_BANK4	0h402C 0000	0h402C 03FF	1 KB
MPU_L2OCRAM_BANK5	0h402E 0000	0h402E 03FF	1 KB
ICSSM0_DRAM0_SLV_RAM	0h4800 0000	0h4800 1FFF	8 KB
ICSSM0_DRAM1_SLV_RAM	0h4800 2000	0h4800 3FFF	8 KB
ICSSM0_RAT_SLICE0_CFG	0h4800 8000	0h4800 8FFF	4 KB
ICSSM0_RAT_SLICE1_CFG	0h4800 9000	0h4800 9FFF	4 KB
ICSSM0_RAM_SLV_RAM	0h4801 0000	0h4801 7FFF	32 KB
ICSSM0_PR1_ICSS_INTC_INTC_SLV	0h4802 0000	0h4802 1FFF	8 KB
ICSSM0_PR1_PDSP0_IRAM	0h4802 2000	0h4802 20FF	256 B
ICSSM0_PR1_PDSP0_IRAM_DEBUG	0h4802 2400	0h4802 24FF	256 B
ICSSM0_PR1_PDSP1_IRAM	0h4802 4000	0h4802 40FF	256 B
ICSSM0_PR1_PDSP1_IRAM_DEBUG	0h4802 4400	0h4802 44FF	256 B
ICSSM0_PR1_PROT_SLV	0h4802 4C00	0h4802 4CFF	256 B
ICSSM0_PR1_CFG_SLV	0h4802 6000	0h4802 61FF	512 B
ICSSM0_PR1_ICSS_UART_UART_SLV	0h4802 8000	0h4802 803F	64 B
ICSSM0_IEP0	0h4802 E000	0h4802 EFFF	4 KB
ICSSM0_PR1_ICSS_ECAP0_ECAP_SLV	0h4803 0000	0h4803 00FF	256 B
ICSSM0_PR1_MII_RT_PR1_MII_RT_CFG	0h4803 2000	0h4803 20FF	256 B

Region Name	Start Address	End Address	Size
ICSSM0_PR1_MDIO_V1P7_MDIO	0h4803 2400	0h4803 24FF	256 B
ICSSM0_PR1_MII_RT_PR1_MII_RT_G_CFG_REGS_G	0h4803 3000	0h4803 3FFF	4 KB
ICSSM0_PR1_PDSP0_IRAM_RAM	0h4803 4000	0h4803 7FFF	16 KB
ICSSM0_PR1_PDSP1_IRAM_RAM	0h4803 8000	0h4803 BFFF	16 KB
ICSSM0_ECC_AGGR	0h4810 0000	0h4810 03FF	1 KB
MMC0	0h4830 0000	0h4830 1FFF	8 KB
CONTROLSS_EPWM0_G0	0h5000 0000	0h5000 0FFF	4 KB
CONTROLSS_EPWM1_G0	0h5000 1000	0h5000 1FFF	4 KB
CONTROLSS_EPWM2_G0	0h5000 2000	0h5000 2FFF	4 KB
CONTROLSS_EPWM3_G0	0h5000 3000	0h5000 3FFF	4 KB
CONTROLSS_EPWM4_G0	0h5000 4000	0h5000 4FFF	4 KB
CONTROLSS_EPWM5_G0	0h5000 5000	0h5000 5FFF	4 KB
CONTROLSS_EPWM6_G0	0h5000 6000	0h5000 6FFF	4 KB
CONTROLSS_EPWM7_G0	0h5000 7000	0h5000 7FFF	4 KB
CONTROLSS_EPWM8_G0	0h5000 8000	0h5000 8FFF	4 KB
CONTROLSS_EPWM9_G0	0h5000 9000	0h5000 9FFF	4 KB
CONTROLSS_EPWM10_G0	0h5000 A000	0h5000 AFFF	4 KB
CONTROLSS_EPWM11_G0	0h5000 B000	0h5000 BFFF	4 KB
CONTROLSS_EPWM12_G0	0h5000 C000	0h5000 CFFF	4 KB
CONTROLSS_EPWM13_G0	0h5000 D000	0h5000 DFFF	4 KB
CONTROLSS_EPWM14_G0	0h5000 E000	0h5000 EFFF	4 KB
CONTROLSS_EPWM15_G0	0h5000 F000	0h5000 FFFF	4 KB
CONTROLSS_EPWM16_G0	0h5001 0000	0h5001 0FFF	4 KB
CONTROLSS_EPWM17_G0	0h5001 1000	0h5001 1FFF	4 KB
CONTROLSS_EPWM18_G0	0h5001 2000	0h5001 2FFF	4 KB
CONTROLSS_EPWM19_G0	0h5001 3000	0h5001 3FFF	4 KB
CONTROLSS_EPWM20_G0	0h5001 4000	0h5001 4FFF	4 KB
CONTROLSS_EPWM21_G0	0h5001 5000	0h5001 5FFF	4 KB
CONTROLSS_EPWM22_G0	0h5001 6000	0h5001 6FFF	4 KB
CONTROLSS_EPWM23_G0	0h5001 7000	0h5001 7FFF	4 KB
CONTROLSS_EPWM24_G0	0h5001 8000	0h5001 8FFF	4 KB
CONTROLSS_EPWM25_G0	0h5001 9000	0h5001 9FFF	4 KB
CONTROLSS_EPWM26_G0	0h5001 A000	0h5001 AFFF	4 KB
CONTROLSS_EPWM27_G0	0h5001 B000	0h5001 BFFF	4 KB
CONTROLSS_EPWM28_G0	0h5001 C000	0h5001 CFFF	4 KB
CONTROLSS_EPWM29_G0	0h5001 D000	0h5001 DFFF	4 KB
CONTROLSS_EPWM30_G0	0h5001 E000	0h5001 EFFF	4 KB
CONTROLSS_EPWM31_G0	0h5001 F000	0h5001 FFFF	4 KB
CONTROLSS_EPWM_WLINK_G0	0h5002 0000	0h5002 0FFF	4 KB
CONTROLSS_EPWM0_G1	0h5004 0000	0h5004 0FFF	4 KB
CONTROLSS_EPWM1_G1	0h5004 1000	0h5004 1FFF	4 KB
CONTROLSS_EPWM2_G1	0h5004 2000	0h5004 2FFF	4 KB
CONTROLSS_EPWM3_G1	0h5004 3000	0h5004 3FFF	4 KB
CONTROLSS_EPWM4_G1	0h5004 4000	0h5004 4FFF	4 KB
CONTROLSS_EPWM5_G1	0h5004 5000	0h5004 5FFF	4 KB
CONTROLSS_EPWM6_G1	0h5004 6000	0h5004 6FFF	4 KB
CONTROLSS_EPWM7_G1	0h5004 7000	0h5004 7FFF	4 KB

Region Name	Start Address	End Address	Size
CONTROLSS_EPWM8_G1	0h5004 8000	0h5004 8FFF	4 KB
CONTROLSS_EPWM9_G1	0h5004 9000	0h5004 9FFF	4 KB
CONTROLSS_EPWM10_G1	0h5004 A000	0h5004 AFFF	4 KB
CONTROLSS_EPWM11_G1	0h5004 B000	0h5004 BFFF	4 KB
CONTROLSS_EPWM12_G1	0h5004 C000	0h5004 CFFF	4 KB
CONTROLSS_EPWM13_G1	0h5004 D000	0h5004 DFFF	4 KB
CONTROLSS_EPWM14_G1	0h5004 E000	0h5004 EFFF	4 KB
CONTROLSS_EPWM15_G1	0h5004 F000	0h5004 FFFF	4 KB
CONTROLSS_EPWM16_G1	0h5005 0000	0h5005 0FFF	4 KB
CONTROLSS_EPWM17_G1	0h5005 1000	0h5005 1FFF	4 KB
CONTROLSS_EPWM18_G1	0h5005 2000	0h5005 2FFF	4 KB
CONTROLSS_EPWM19_G1	0h5005 3000	0h5005 3FFF	4 KB
CONTROLSS_EPWM20_G1	0h5005 4000	0h5005 4FFF	4 KB
CONTROLSS_EPWM21_G1	0h5005 5000	0h5005 5FFF	4 KB
CONTROLSS_EPWM22_G1	0h5005 6000	0h5005 6FFF	4 KB
CONTROLSS_EPWM23_G1	0h5005 7000	0h5005 7FFF	4 KB
CONTROLSS_EPWM24_G1	0h5005 8000	0h5005 8FFF	4 KB
CONTROLSS_EPWM25_G1	0h5005 9000	0h5005 9FFF	4 KB
CONTROLSS_EPWM26_G1	0h5005 A000	0h5005 AFFF	4 KB
CONTROLSS_EPWM27_G1	0h5005 B000	0h5005 BFFF	4 KB
CONTROLSS_EPWM28_G1	0h5005 C000	0h5005 CFFF	4 KB
CONTROLSS_EPWM29_G1	0h5005 D000	0h5005 DFFF	4 KB
CONTROLSS_EPWM30_G1	0h5005 E000	0h5005 EFFF	4 KB
CONTROLSS_EPWM31_G1	0h5005 F000	0h5005 FFFF	4 KB
CONTROLSS_EPWM_WLINK_G1	0h5006 0000	0h5006 0FFF	4 KB
CONTROLSS_EPWM0_G2	0h5008 0000	0h5008 0FFF	4 KB
CONTROLSS_EPWM1_G2	0h5008 1000	0h5008 1FFF	4 KB
CONTROLSS_EPWM2_G2	0h5008 2000	0h5008 2FFF	4 KB
CONTROLSS_EPWM3_G2	0h5008 3000	0h5008 3FFF	4 KB
CONTROLSS_EPWM4_G2	0h5008 4000	0h5008 4FFF	4 KB
CONTROLSS_EPWM5_G2	0h5008 5000	0h5008 5FFF	4 KB
CONTROLSS_EPWM6_G2	0h5008 6000	0h5008 6FFF	4 KB
CONTROLSS_EPWM7_G2	0h5008 7000	0h5008 7FFF	4 KB
CONTROLSS_EPWM8_G2	0h5008 8000	0h5008 8FFF	4 KB
CONTROLSS_EPWM9_G2	0h5008 9000	0h5008 9FFF	4 KB
CONTROLSS_EPWM10_G2	0h5008 A000	0h5008 AFFF	4 KB
CONTROLSS_EPWM11_G2	0h5008 B000	0h5008 BFFF	4 KB
CONTROLSS_EPWM12_G2	0h5008 C000	0h5008 CFFF	4 KB
CONTROLSS_EPWM13_G2	0h5008 D000	0h5008 DFFF	4 KB
CONTROLSS_EPWM14_G2	0h5008 E000	0h5008 EFFF	4 KB
CONTROLSS_EPWM15_G2	0h5008 F000	0h5008 FFFF	4 KB
CONTROLSS_EPWM16_G2	0h5009 0000	0h5009 0FFF	4 KB
CONTROLSS_EPWM17_G2	0h5009 1000	0h5009 1FFF	4 KB
CONTROLSS_EPWM18_G2	0h5009 2000	0h5009 2FFF	4 KB
CONTROLSS_EPWM19_G2	0h5009 3000	0h5009 3FFF	4 KB
CONTROLSS_EPWM20_G2	0h5009 4000	0h5009 4FFF	4 KB
CONTROLSS_EPWM21_G2	0h5009 5000	0h5009 5FFF	4 KB

Region Name	Start Address	End Address	Size
CONTROLSS_EPWM22_G2	0h5009 6000	0h5009 6FFF	4 KB
CONTROLSS_EPWM23_G2	0h5009 7000	0h5009 7FFF	4 KB
CONTROLSS_EPWM24_G2	0h5009 8000	0h5009 8FFF	4 KB
CONTROLSS_EPWM25_G2	0h5009 9000	0h5009 9FFF	4 KB
CONTROLSS_EPWM26_G2	0h5009 A000	0h5009 AFFF	4 KB
CONTROLSS_EPWM27_G2	0h5009 B000	0h5009 BFFF	4 KB
CONTROLSS_EPWM28_G2	0h5009 C000	0h5009 CFFF	4 KB
CONTROLSS_EPWM29_G2	0h5009 D000	0h5009 DFFF	4 KB
CONTROLSS_EPWM30_G2	0h5009 E000	0h5009 EFFF	4 KB
CONTROLSS_EPWM31_G2	0h5009 F000	0h5009 FFFF	4 KB
CONTROLSS_EPWM_WLINK_G2	0h500A 0000	0h500A 0FFF	4 KB
CONTROLSS_EPWM0_G3	0h500C 0000	0h500C 0FFF	4 KB
CONTROLSS_EPWM1_G3	0h500C 1000	0h500C 1FFF	4 KB
CONTROLSS_EPWM2_G3	0h500C 2000	0h500C 2FFF	4 KB
CONTROLSS_EPWM3_G3	0h500C 3000	0h500C 3FFF	4 KB
CONTROLSS_EPWM4_G3	0h500C 4000	0h500C 4FFF	4 KB
CONTROLSS_EPWM5_G3	0h500C 5000	0h500C 5FFF	4 KB
CONTROLSS_EPWM6_G3	0h500C 6000	0h500C 6FFF	4 KB
CONTROLSS_EPWM7_G3	0h500C 7000	0h500C 7FFF	4 KB
CONTROLSS_EPWM8_G3	0h500C 8000	0h500C 8FFF	4 KB
CONTROLSS_EPWM9_G3	0h500C 9000	0h500C 9FFF	4 KB
CONTROLSS_EPWM10_G3	0h500C A000	0h500C AFFF	4 KB
CONTROLSS_EPWM11_G3	0h500C B000	0h500C BFFF	4 KB
CONTROLSS_EPWM12_G3	0h500C C000	0h500C CFFF	4 KB
CONTROLSS_EPWM13_G3	0h500C D000	0h500C DFFF	4 KB
CONTROLSS_EPWM14_G3	0h500C E000	0h500C EFFF	4 KB
CONTROLSS_EPWM15_G3	0h500C F000	0h500C FFFF	4 KB
CONTROLSS_EPWM16_G3	0h500D 0000	0h500D 0FFF	4 KB
CONTROLSS_EPWM17_G3	0h500D 1000	0h500D 1FFF	4 KB
CONTROLSS_EPWM18_G3	0h500D 2000	0h500D 2FFF	4 KB
CONTROLSS_EPWM19_G3	0h500D 3000	0h500D 3FFF	4 KB
CONTROLSS_EPWM20_G3	0h500D 4000	0h500D 4FFF	4 KB
CONTROLSS_EPWM21_G3	0h500D 5000	0h500D 5FFF	4 KB
CONTROLSS_EPWM22_G3	0h500D 6000	0h500D 6FFF	4 KB
CONTROLSS_EPWM23_G3	0h500D 7000	0h500D 7FFF	4 KB
CONTROLSS_EPWM24_G3	0h500D 8000	0h500D 8FFF	4 KB
CONTROLSS_EPWM25_G3	0h500D 9000	0h500D 9FFF	4 KB
CONTROLSS_EPWM26_G3	0h500D A000	0h500D AFFF	4 KB
CONTROLSS_EPWM27_G3	0h500D B000	0h500D BFFF	4 KB
CONTROLSS_EPWM28_G3	0h500D C000	0h500D CFFF	4 KB
CONTROLSS_EPWM29_G3	0h500D D000	0h500D DFFF	4 KB
CONTROLSS_EPWM30_G3	0h500D E000	0h500D EFFF	4 KB
CONTROLSS_EPWM31_G3	0h500D F000	0h500D FFFF	4 KB
CONTROLSS_EPWM_WLINK_G3	0h500E 0000	0h500E 0FFF	4 KB
CONTROLSS_ADC0_RESULTS	0h5010 0000	0h5010 0FFF	4 KB
CONTROLSS_ADC1_RESULTS	0h5010 1000	0h5010 1FFF	4 KB
CONTROLSS_ADC2_RESULTS	0h5010 2000	0h5010 2FFF	4 KB

Region Name	Start Address	End Address	Size
CONTROLSS_ADC3_RESULTS	0h5010 3000	0h5010 3FFF	4 KB
CONTROLSS_ADC4_RESULTS	0h5010 4000	0h5010 4FFF	4 KB
CONTROLSS_CMPSSA0	0h5020 0000	0h5020 0FFF	4 KB
CONTROLSS_CMPSSA1	0h5020 1000	0h5020 1FFF	4 KB
CONTROLSS_CMPSSA2	0h5020 2000	0h5020 2FFF	4 KB
CONTROLSS_CMPSSA3	0h5020 3000	0h5020 3FFF	4 KB
CONTROLSS_CMPSSA4	0h5020 4000	0h5020 4FFF	4 KB
CONTROLSS_CMPSSA5	0h5020 5000	0h5020 5FFF	4 KB
CONTROLSS_CMPSSA6	0h5020 6000	0h5020 6FFF	4 KB
CONTROLSS_CMPSSA7	0h5020 7000	0h5020 7FFF	4 KB
CONTROLSS_CMPSSA8	0h5020 8000	0h5020 8FFF	4 KB
CONTROLSS_CMPSSA9	0h5020 9000	0h5020 9FFF	4 KB
CONTROLSS_CMPSSB0	0h5022 0000	0h5022 0FFF	4 KB
CONTROLSS_CMPSSB1	0h5022 1000	0h5022 1FFF	4 KB
CONTROLSS_CMPSSB2	0h5022 2000	0h5022 2FFF	4 KB
CONTROLSS_CMPSSB3	0h5022 3000	0h5022 3FFF	4 KB
CONTROLSS_CMPSSB4	0h5022 4000	0h5022 4FFF	4 KB
CONTROLSS_CMPSSB5	0h5022 5000	0h5022 5FFF	4 KB
CONTROLSS_CMPSSB6	0h5022 6000	0h5022 6FFF	4 KB
CONTROLSS_CMPSSB7	0h5022 7000	0h5022 7FFF	4 KB
CONTROLSS_CMPSSB8	0h5022 8000	0h5022 8FFF	4 KB
CONTROLSS_CMPSSB9	0h5022 9000	0h5022 9FFF	4 KB
CONTROLSS_ECAP0	0h5024 0000	0h5024 0FFF	4 KB
CONTROLSS_ECAP1	0h5024 1000	0h5024 1FFF	4 KB
CONTROLSS_ECAP2	0h5024 2000	0h5024 2FFF	4 KB
CONTROLSS_ECAP3	0h5024 3000	0h5024 3FFF	4 KB
CONTROLSS_ECAP4	0h5024 4000	0h5024 4FFF	4 KB
CONTROLSS_ECAP5	0h5024 5000	0h5024 5FFF	4 KB
CONTROLSS_ECAP6	0h5024 6000	0h5024 6FFF	4 KB
CONTROLSS_ECAP7	0h5024 7000	0h5024 7FFF	4 KB
CONTROLSS_ECAP8	0h5024 8000	0h5024 8FFF	4 KB
CONTROLSS_ECAP9	0h5024 9000	0h5024 9FFF	4 KB
CONTROLSS_ECAP10	0h5024 A000	0h5024 AFFF	4 KB
CONTROLSS_ECAP11	0h5024 B000	0h5024 BFFF	4 KB
CONTROLSS_ECAP12	0h5024 C000	0h5024 CFFF	4 KB
CONTROLSS_ECAP13	0h5024 D000	0h5024 DFFF	4 KB
CONTROLSS_ECAP14	0h5024 E000	0h5024 EFFF	4 KB
CONTROLSS_ECAP15	0h5024 F000	0h5025 FFFF	4 KB
CONTROLSS_DAC0	0h5026 0000	0h5026 0FFF	4 KB
CONTROLSS_SDFM0	0h5026 8000	0h5026 8FFF	4 KB
CONTROLSS_SDFM1	0h5026 9000	0h5026 9FFF	4 KB
CONTROLSS_EQEP0	0h5027 0000	0h5027 0FFF	4 KB
CONTROLSS_EQEP1	0h5027 1000	0h5027 1FFF	4 KB
CONTROLSS_EQEP2	0h5027 2000	0h5027 2FFF	4 KB
CONTROLSS_FSI_TX0	0h5028 0000	0h5028 0FFF	4 KB
CONTROLSS_FSI_TX1	0h5028 1000	0h5028 1FFF	4 KB
CONTROLSS_FSI_RX0	0h5029 0000	0h5029 1FFF	4 KB

Region Name	Start Address	End Address	Size
CONTROLSS_FSI_RX1	0h5029 1000	0h5029 1FFF	4 KB
CONTROLSS_FSI_TX2	0h502A 0000	0h502A 0FFF	4 KB
CONTROLSS_FSI_TX3	0h502A 1000	0h502A 1FFF	4 KB
CONTROLSS_FSI_RX2	0h502B 0000	0h502B 0FFF	4 KB
CONTROLSS_FSI_RX3	0h502B 1000	0h502B 1FFF	4 KB
CONTROLSS_ADC0_CFG	0h502C 0000	0h502C 0FFF	4 KB
CONTROLSS_ADC1_CFG	0h502C 1000	0h502C 1FFF	4 KB
CONTROLSS_ADC2_CFG	0h502C 2000	0h502C 2FFF	4 KB
CONTROLSS_ADC3_CFG	0h502C 3000	0h502C 3FFF	4 KB
CONTROLSS_ADC4_CFG	0h502C 4000	0h502C FFF	4 KB
CONTROLSS_ADCR0_CFG	0h502C 5000	0h502C 51FF	512 B
CONTROLSS_ADCR0_RESULTS	0h502C 6000	0h502C 60FF	256 B
CONTROLSS_ADCR1_CFG	0h502C 7000	0h502C 71FF	512 B
CONTROLSS_ADCR1_RESULTS	0h502C 8000	0h502C 80FF	256 B
CONTROLSS_RESOLVER0	0h502C B000	0h502C B3FF	1 KB
CONTROLSS_ADC_SAFETY0	0h502C B400	0h502C B8FF	1 KB
CONTROLSS_ADC_SAFETY1	0h502C B800	0h502C BBFF	1 KB
CONTROLSS_ADC_SAFETY2	0h502C BC00	0h502C BFFF	1 KB
CONTROLSS_ADC_SAFETY3	0h502C C000	0h502C C3FF	1 KB
CONTROLSS_ADC_SAFETY4	0h502C C400	0h502C C7FF	1 KB
CONTROLSS_ADC_SAFETY5	0h502C C800	0h502C CBFF	1 KB
CONTROLSS_ADC_SAFETY6	0h502C CC00	0h502C CFFF	1 KB
CONTROLSS_ADC_SAFETY7	0h502C D000	0h502C D3FF	1 KB
CONTROLSS_ADC_SAFETY8	0h502C D400	0h502C D7FF	1 KB
CONTROLSS_ADC_SAFETY9	0h502C D800	0h502C DBFF	1 KB
CONTROLSS_ADC_SAFETY10	0h502C DC00	0h502C DFFF	1 KB
CONTROLSS_ADC_SAFETY11	0h502C E000	0h502C E3FFF	1 KB
CONTROLSS_ADC_SAFETY_AGGR	0h502C EC00	0h502C EFFF	1 KB
CONTROLSS_INPUTXBAR	0h502D 0000	0h502D 0FFF	4 KB
CONTROLSS_PWMXBAR	0h502D 1000	0h502D 1FFF	4 KB
CONTROLSS_PWMSYNCOUTXBAR	0h502D 2000	0h502D 23FF	1 KB
CONTROLSS_MDLXBAR	0h502D 3000	0h502D 37FF	2 KB
CONTROLSS_ICLXBAR	0h502D 4000	0h502D 47FF	2 KB
CONTROLSS_INTXBAR	0h502D 5000	0h502D 5FFF	4 KB
CONTROLSS_DMAXBAR	0h502D 6000	0h502D 67FF	2 KB
CONTROLSS_OUTPUTXBAR	0h502D 8000	0h502D 87FF	2 KB
CONTROLSS_OTTOCAL0	0h502E 0000	0h502E 0FFF	4 KB
CONTROLSS_OTTOCAL1	0h502E 1000	0h502E 1FFF	4 KB
CONTROLSS_OTTOCAL2	0h502E 2000	0h502E 2FFF	4 KB
CONTROLSS_OTTOCAL3	0h502E 3000	0h502E 3FFF	4 KB
CONTROLSS_GLOBAL_CTRL	0h502F 0000	0h502F 1FFF	8 KB
DEBUGSS	0h5080 0000	0h5087 FFFF	512 KB
MSS_CTRL	0h50D0 0000	0h50D1 FFFF	128 KB
TOP_CTRL	0h50D8 0000	0h50D8 1FFF	8 KB
SPINLOCK	0h50E0 0000	0h50E0 7FFF	32 KB
VIM	0h50F0 0000	0h50F0 3FFF	16 KB
GPIO0	0h5200 0000	0h5200 00FF	256 B

Region Name	Start Address	End Address	Size
GPIO1	0h5200 1000	0h5200 10FF	256 B
GPIO2	0h5200 2000	0h5200 20FF	256 B
GPIO3	0h5200 3000	0h5200 30FF	256 B
WDT0	0h5210 0000	0h5210 00FF	256 B
WDT1	0h5210 1000	0h5210 10FF	256 B
WDT2	0h5210 2000	0h5210 20FF	256 B
WDT3	0h5210 3000	0h5210 30FF	256 B
RTI0	0h5218 0000	0h5218 00FF	256 B
RTI1	0h5218 1000	0h5218 10FF	256 B
RTI2	0h5218 2000	0h5218 20FF	256 B
RTI3	0h5218 3000	0h5218 30FF	256 B
RTI4	0h5218 4000	0h5218 40FF	256 B
RTI5	0h5218 5000	0h5218 50FF	256 B
RTI6	0h5218 6000	0h5218 60FF	256 B
RTI7	0h5218 7000	0h5218 70FF	256 B
SPI0	0h5220 0000	0h5220 03FF	1 KB
SPI1	0h5220 1000	0h5220 13FF	1 KB
SPI2	0h5220 2000	0h5220 23FF	1 KB
SPI3	0h5220 3000	0h5220 33FF	1 KB
SPI4	0h5220 4000	0h5220 43FF	1 KB
SPI5	0h5220 5000	0h5220 53FF	1 KB
SPI6	0h5220 6000	0h5220 63FF	1 KB
SPI7	0h5220 7000	0h5220 73FF	1 KB
UART0	0h5230 0000	0h5230 01FF	512 B
UART1	0h5230 1000	0h5230 11FF	512 B
UART2	0h5230 2000	0h5230 21FF	512 B
UART3	0h5230 3000	0h5230 31FF	512 B
UART4	0h5230 4000	0h5230 41FF	512 B
UART5	0h5230 5000	0h5230 51FF	512 B
LIN0	0h5240 0000	0h5240 00FF	256 B
LIN1	0h5240 1000	0h5240 10FF	256 B
LIN2	0h5240 2000	0h5240 20FF	256 B
LIN3	0h5240 3000	0h5240 30FF	256 B
LIN4	0h5240 4000	0h5240 40FF	256 B
I2C0	0h5250 0000	0h5250 007F	128 B
I2C1	0h5250 1000	0h5250 107F	128 B
I2C2	0h5250 2000	0h5250 207F	128 B
I2C3	0h5250 3000	0h5250 307F	128 B
MCAN0_MSG_RAM	0h5260 0000	0h5260 7FFF	32 KB
MCAN0_CFG	0h5260 8000	0h5260 83FF	1 KB
MCAN1_MSG_RAM	0h5261 0000	0h5261 7FFF	32 KB
MCAN1_CFG	0h5261 8000	0h5261 83FF	1 KB
MCAN2_MSG_RAM	0h5262 0000	0h5262 7FFF	32 KB
MCAN2_CFG	0h5262 8000	0h5262 83FF	1 KB
MCAN3_MSG_RAM	0h5263 0000	0h5263 7FFF	32 KB
MCAN3_CFG	0h5263 8000	0h5263 83FF	1 KB
MCAN4_MSG_RAM	0h5264 0000	0h5264 7FFF	32 KB

Region Name	Start Address	End Address	Size
MCAN4_CFG	0h5264 8000	0h5264 83FF	1 KB
MCAN5_MSG_RAM	0h5265 0000	0h5265 7FFF	32 KB
MCAN5_CFG	0h5265 8000	0h5265 83FF	1 KB
MCAN6_MSG_RAM	0h5266 0000	0h5266 7FFF	32 KB
MCAN6_CFG	0h5266 8000	0h5266 83FF	1 KB
MCAN7_MSG_RAM	0h5267 0000	0h5267 7FFF	32 KB
MCAN7_CFG	0h5267 8000	0h5267 83FF	1 KB
MCAN0_ECC	0h5270 0000	0h5270 03FF	1 KB
MCAN1_ECC	0h5270 1000	0h5270 13FF	1 KB
MCAN2_ECC	0h5270 2000	0h5270 23FF	1 KB
MCAN3_ECC	0h5270 3000	0h5270 33FF	1 KB
MCAN4_ECC	0h5270 4000	0h5270 43FF	1 KB
MCAN5_ECC	0h5270 5000	0h5270 53FF	1 KB
MCAN6_ECC	0h5270 6000	0h5270 63FF	1 KB
MCAN7_ECC	0h5270 7000	0h5270 73FF	1 KB
CPSW	0h5280 0000	0h529FF FFFF	2 MB
EDMA_TPCC	0h52A0 0000	0h52A0 7FFF	32 KB
EDMA_TPTC0	0h52A4 0000	0h52A4 0FFF	4 KB
EDMA_TPTC1	0h52A6 0000	0h52A6 0FFF	4 KB
DCC0	0h52B0 0000	0h52B0 003F	64 B
DCC1	0h52B0 1000	0h52B0 103F	64 B
DCC2	0h52B0 2000	0h52B0 203F	64 B
DCC3	0h52B0 3000	0h52B0 303F	64 B
ESM0_CFG	0h52D0 0000	0h52D0 0FFF	4 KB
SOC_TIMESYNC_XBAR0	0h52E0 0000	0h52E0 03FF	1 KB
EDMA_TRIGGER_XBAR0	0h52E0 1000	0h52E0 17FF	2 KB
GPIO_XBAR0	0h52E0 2000	0h52E0 23FF	1 KB
ICSSM_XBAR0	0h52E0 3000	0h52E0 31FF	512 B
SOC_TIMESYNC_XBAR1	0h52E0 4000	0h52E0 47FF	2 KB
R5SS0_CORE0_ECC_AGG	0h5300 0000	0h5300 03FF	1 KB
R5SS0_CORE1_ECC_AGG	0h5300 3000	0h5300 33FF	1 KB
R5SS1_CORE0_ECC_AGG	0h5300 4000	0h5300 43FF	1 KB
R5SS1_CORE1_ECC_AGG	0h5300 7000	0h5300 73FF	1 KB
ECC_AGG_TOP	0h5301 0000	0h5301 03FF	1 KB
R5SS0_CORE0_TMU_ROM	0h5302 0000	0h5302 1FFF	8 KB
R5SS0_CORE1_TMU_ROM	0h5302 4000	0h5302 5FFF	8 KB
R5SS1_CORE0_TMU_ROM	0h5302 8000	0h5302 9FFF	8 KB
R5SS1_CORE1_TMU_ROM	0h5302 C000	0h5302 DFFF	8 KB
IOMUX	0h5310 0000	0h5310 0FFF	4 KB
TOP_RCM	0h5320 0000	0h5320 1FFF	8 KB
MSS_RCM	0h5320 8000	0h5320 9FFF	8 KB
R5SS0_CCMR	0h5321 0000	0h5321 0FFF	4 KB
R5SS1_CCMR	0h5321 1000	0h5321 1FFF	4 KB
R5SS0_CORE0_RL2	0h5321 2000	0h5321 23FF	1 KB
R5SS0_CORE1_RL2	0h5321 3000	0h5321 33FF	1 KB
R5SS1_CORE0_RL2	0h5321 4000	0h5321 43FF	1 KB
R5SS1_CORE1_RL2	0h5321 5000	0h5321 53FF	1 KB

Region Name	Start Address	End Address	Size
PBIST0	0h5330 0200	0h5330 03FF	512 B
FSS_GASKET_CFG	0h5340 0000	0h5340 03FF	1 KB
R5SS0_STC	0h5350 0000	0h5350 01FF	512 B
R5SS0_STC	0h5351 0000	0h5351 01FF	512 B
EFUSE0	0h5360 0000	0h5360 03FF	1 KB
FSS_CFG (FLASH_CONFIG_REG0)	0h5380 0000	0h5380 00FF	256 B
FSAS_CFG (FLASH_CONFIG_REG1)	0h5380 1000	0h5380 10FF	256 B
FSAS_OTFA_CFG (FLASH_CONFIG_REG2)	0h5380 2000	0h5380 2FFF	4 KB
OSPI_CFG (FLASH_CONFIG_REG6)	0h5380 6000	0h5380 61FF	512 B
OSPI0_ECC_AGG (FLASH_CONFIG_REG7)	0h5380 7000	0h5380 73FF	1 KB
OSPI0_FLASH_APB (FLASH_CONFIG_REG8)	0h5380 8000	0h5380 80FF	256 B
FOTA_MMR_CFG (FLASH_CONFIG_REG11)	0h5380 B000	0h5380 B0FF	256 B
FOTA_PDMEM_CFG (FLASH_CONFIG_REG12)	0h5380 C000	0h5380 C7FF	2 KB
FOTA_IMEM_CFG (FLASH_CONFIG_REG13)	0h5380 D000	0h5380 D0FF	256 B
FOTA_WBUF_CFG (FLASH_CONFIG_REG14)	0h5380 E000	0h5380 E1FF	512 B
FSAS_ECC_AGGR (FLASH_CONFIG_REG15)	0h5380 F000	0h5380 F3FF	1 KB
FSS_DATA_REG0	0h6000 0000	0h67FF FFFF	128 MB
L2OCRAM_BANK0	0h7000 0000	0h7007 FFFF	512 KB
L2OCRAM_BANK1	0h7008 0000	0h700F FFFF	512 KB
L2OCRAM_BANK2	0h7010 0000	0h7017 FFFF	512 KB
L2OCRAM_BANK3	0h7018 0000	0h701F FFFF	512 KB
L2OCRAM_BANK4	0h7020 0000	0h7027 FFFF	512 KB
L2OCRAM_BANK5	0h7028 0000	0h702F FFFF	512 KB
SOC_MAILBOX_RAM0	0h7200 0000	0h7200 3FFF	16 KB
R5SS0_CORE0_ICACHE[4]	0h7400 0000	0h747F FFFF	16 KB (8 MB)[5]
R5SS0_CORE0_DCACHE[4]	0h7480 0000	0h74FF FFFF	16 KB (8 MB)[5]
R5SS0_CORE1_ICACHE[2 4]	0h7500 0000	0h757F FFFF	16 KB (8 MB)[5]
R5SS0_CORE1_DCACHE[2 4]	0h7580 0000	0h75FF FFFF	16 KB (8 MB)[5]
R5SS1_CORE0_ICACHE[4]	0h7600 0000	0h767F FFFF	16 KB (8 MB)[5]
R5SS1_CORE0_DCACHE[4]	0h7680 0000	0h76FF FFFF	16 KB (8 MB)[5]
R5SS1_CORE1_ICACHE[2 4]	0h7700 0000	0h777F FFFF	16 KB (8 MB)[5]
R5SS1_CORE1_DCACHE[2 4]	0h7780 0000	0h77FF FFFF	16 KB (8 MB)[5]
R5SS0_CORE0_TCMA[3 4]	0h7800 0000	0h7800 FFFF	64 KB
R5SS0_CORE0_TMU_EXT	0h7806 0000	0h7806 03FF	1 KB
R5SS0_CORE0_TCMB[3 4]	0h7810 0000	0h7810 FFFF	192 KB
R5SS0_CORE1_TCMA[2 4]	0h7820 0000	0h7820 7FFF	32 KB
R5SS0_CORE1_TMU_EXT	0h7826 0000	0h7826 03FFF	1 KB
R5SS0_CORE1_TCMB[2 4]	0h7830 0000	0h7830 7FFF	96 KB
R5SS1_CORE0_TCMA[3 4]	0h7840 0000	0h7840 FFFF	64 KB
R5SS1_CORE0_TMU_EXT	0h7846 0000	0h7846 03FF	1 KB
R5SS1_CORE0_TCMB[3 4]	0h7850 0000	0h7850 FFFF	192 KB
R5SS1_CORE1_TCMA[2 4]	0h7860 0000	0h7860 7FFF	32 KB
R5SS1_CORE1_TMU_EXT	0h7866 0000	0h7866 03FF	1 KB
R5SS1_CORE1_TCMB[2 4]	0h7870 0000	0h7870 7FFF	96 KB
FSS_DATA_REG1	0h8000 0000	0h87FF FFFF	128 MB
FSS_DATA_REG3	0h8800 0000	0h88FF FFFF	128 MB

1. See core-specific tables for the internal memory map.

2. In Lockstep mode, the R5FSS*x*_CORE**1** memory region is not accessible.
3. The size of these memories changes based on Dual-Core vs Lockstep operation.
   For more information about Dual-Core and Lockstep modes, see the *R5FSS* chapter.
   For more information about ATCM and BTCM, see the *Tightly-Coupled Memories (TCM)* section within the *R5FSS* chapter.
4. This memory region is used by each CPU core to access the TCM/Cache memory space of other CPU cores.
5. Each R5FSS contains 16 KB i-cache and 16 KB d-cache. However, the system interconnect sees an 8 MB address range at ICACHE/DCACHE. Any core attempting to access more than 16 KB results in a wrap around and accesing the same cache multiple times.

2.2 R5FSS Memory Map

Table 2-1. R5FSS0-0 Memory Map

Region Name	Start Address	End Address	Size
R5SS0_CORE0_TCMA_ROM	0h0000 0000	0h0001 FFFF	128 KB
R5SS0_CORE0_TCMA_RAM	0h0002 0000	0h0003 FFFF	32 KB
R5SS0_CORE0_TMU	0h0006 0000	0h0007 FFFF	131 KB
R5SS0_CORE0_TCMB_RAM	0h0008 0000	0h0009 FFFF	96 KB
R5SS0_CORE0_VIM	0h50F0 0000	0h51FF FFFF	16 KB
R5SS0_CORE0_WWDT (WDT0)	0h5210 0000	0h5210 00FF	256 Bytes
R5SS0_CORE0_RTI (RTI0)	0h5218 0000	0h5218 0FFF	1 KB

Note

At device boot, the R5SS0_CORE0 executes ROM from address 0h0. After device boot, the R5SS0_CORE0_TCMA_RAM takes the base address and executes from address 0h0.

Table 2-2. R5FSS0-1 Memory Map

Region Name	Start Address	End Address	Size
R5SS0_CORE1_TCMA_RAM	0h0000 0000	0h0000 7FFF	32 KB
R5SS0_CORE1_TMU	0h0006 0000	0h0007 FFFF	131 KB
R5SS0_CORE1_TCMB_RAM	0h0008 0000	0h0008 7FFF	32 KB
R5SS0_CORE1_VIM	0h50F0 0000	0h50F0 3FFF	16 KB
R5SS0_CORE1_WWDT (WDT1)	0h5210 1000	0h5210 10FF	256 Bytes
R5SS0_CORE1_RTI (RTI1)	0h5218 1000	0h5218 13FF	1 KB

Table 2-3. R5FSS1-0 Memory Map

Region Name	Start Address	End Address	Size
R5SS1_CORE0_TCMA_RAM	0h0000 0000	0h0000 7FFF	32 KB
R5SS1_CORE0_TMU	0h0006 0000	0h0007 FFFF	131 KB
R5SS1_CORE0_TCMB_RAM	0h0008 0000	0h0008 7FFF	32 KB
R5SS1_CORE0_VIM	0h50F0 0000	0h50F0 3FFF	16 KB
R5SS1_CORE0_WWDT (WDT2)	0h5210 2000	0h5210 20FF	256 Bytes
R5SS1_CORE0_RTI (RTI2)	0h5218 2000	0h5218 23FF	1 KB

Table 2-4. R5FSS1-1 Memory Map

Region Name	Start Address	End Address	Size
R5SS1_CORE1_TCMA_RAM	0h0000 0000	0h0000 7FFF	32 KB
R5SS1_CORE1_TMU	0h0006 0000	0h0007 FFFF	131 KB
R5SS1_CORE1_TCMB_RAM	0h0008 0000	0h0009 FFFF	32 KB
R5SS1_CORE1_VIM	0h50F0 0000	0h50F0 3FFF	16 KB
R5SS1_CORE1_WWDT (WDT3)	0h5210 3000	0h5210 30FF	256 Bytes
R5SS1_CORE1_RTI (RTI3)	0h5218 3000	0h5218 33FF	1 KB

2.3 PRU-ICSS Memory Map

Region Name	Start Address	End Address	Size
PRU-ICSS Data RAM0 (DRAM0)	0h0000 0000	0h0000 1FFF	8 KB
PRU-ICSS Data RAM1 (DRAM1)	0h0000 2000	0h0000 3FFF	8 KB
PRU-ICSS Data RAM2 (Shared DRAM2)	0h0001 0000	0h0001 FFFF	64 KB
PRU-ICSS INTC	0h0002 0000	0h0002 1FFF	8 KB
PRU-ICSS PRU0 Control	0h0002 2000	0h0002 23FF	1 KB
PRU-ICSS PRU0 Debug	0h0002 2400	0h0002 3FFF	7 KB
PRU-ICSS PRU1 Control	0h0002 4000	0h0002 43FF	1 KB
PRU-ICSS PRU1 Debug	0h0002 4400	0h0002 5FFF	7 KB
PRU-ICSS CFG	0h0002 6000	0h0002 6FFF	4 KB
PRU-ICSS ECC_CFG	0h0002 7000	0h0002 7FFF	4 KB
PRU-ICSS UART0	0h0002 8000	0h0002 9FFF	8 KB
PRU-ICSS Reserved	0h0002 A000	0h0002 BFFF	8 KB
PRU-ICSS Reserved	0h0002 C000	0h0002 DFFF	8 KB
PRU-ICSS IEP	0h0002 E000	0h0002 EFFF	8 KB
PRU-ICSS ECAP0	0h0003 0000	0h0003 1FFF	8 KB
PRU-ICSS MII_RT_CFG	0h0003 2000	0h0003 23FF	1 KB
PRU-ICSS MII_MDIO	0h0003 2400	0h0003 3FFF	7 KB
PRU-ICSS PRU0 IRAM	0h0003 4000	0h0003 7FFF	16 KB
PRU-ICSS PRU1 IRAM	0h0003 8000	0h0003 BFFF	16 KB

3 System Interconnect

This chapter describes the device system interconnect.

System interconnect provides a multi-layered crossbar network among initiators and targets within SoC. This mutli-layered crossbar network supports multiple in-flight transactions to improve both latency and throughput

3.1 System Interconnect Overview

The device implements a system interconnect using TI's Common Bus Architecture (CBA), composed of the VBUSM and VBUSP protocols.

The system is based on a multi-layered interconnect approach designed to meet high-performance system requirements. The core interconnect structure consists of a full crossbar implementation, where every initiator has an independent communication path with every target. In other words, any initiator can access any target on the interconnect while another initiator can access a different target **simultaneously without any contention,**such that, transactions from each initiator has access to full interconnect bandwidth. Arbitration will only happen at the target end point (when the same target is accessed by two or more initiators) with round-robin prioritization. Targets cannot generate read/write requests directly. However, they can respond to these requests by generating error events (as defined by the CBA protocol), interrupts, and DMA requests.

The device interconnect is partitioned into the following sections:

- CORE VBUSM Interconnect
- CORE VBUSP Interconnect
- R5SS0 VBUSM Interconnect
- R5SS1 VBUSM Interconnect
- PERI VBUSP Interconnect
- INFRA0 VBUSP Interconnect
- INFRA1 VBUSP Interconnect
- R5SS0 CONFIG TARGET Interconnect
- R5SS1 CONFIG TARGET Interconnect
- CONTROLSS VBUSP Interconnect

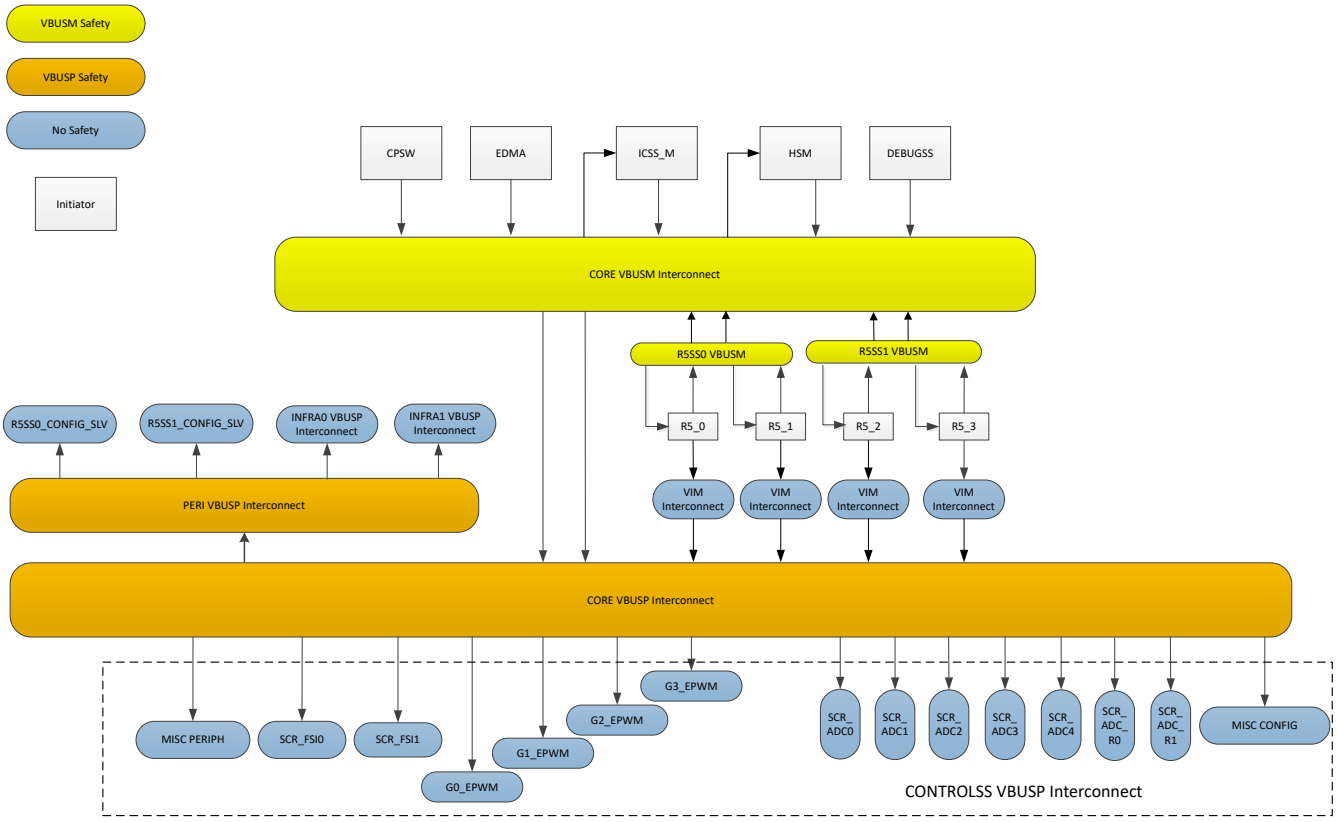


Figure 3-1. Top-Level System Interconnect

Note

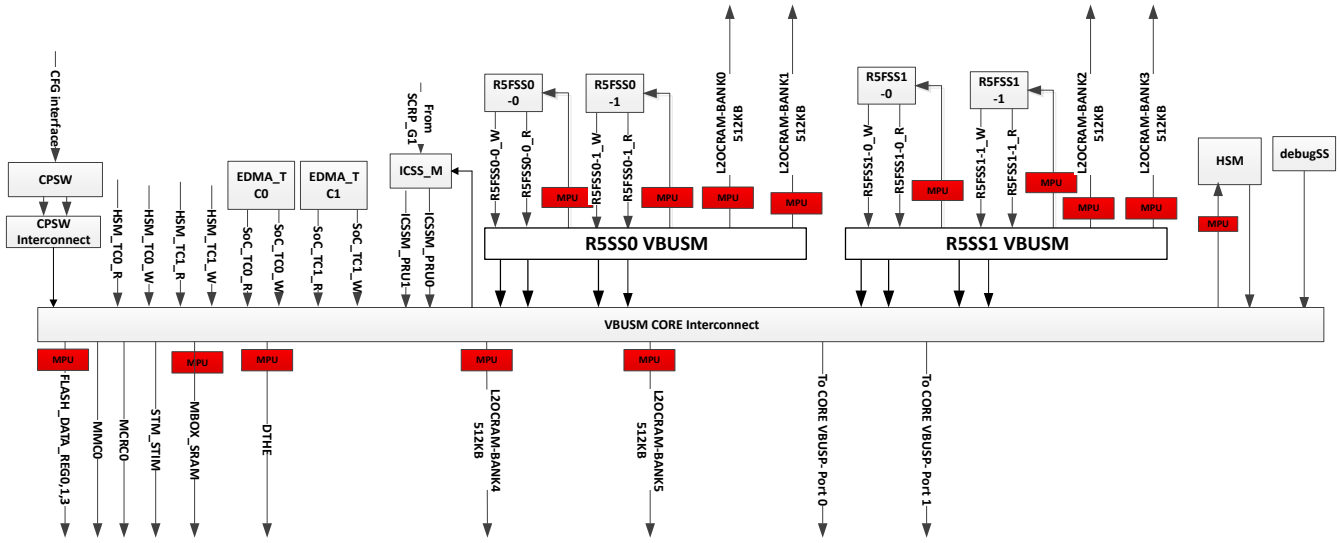
CORE VBUSM, R5SS0 VBUSM and R5SS1 VBUSM Interconnects are 64-bit wide interconnect (i.e. 64-bit data bus width). Rest of the above interconnects are 32-bit wide (i.e. 32-bit data bus width).

There are multiple targets for each of the above interconnects, these are detailed in later sections of the chapter.

3.2 CORE VBUSM Interconnect

The device Core Interconnect (CORE VBUSM) utilizes the VBUSM architecture to enable extensive transaction pipelining configuration along with support for multiple outstanding transactions; this dramatically increases system performance at the cost of higher complexity and additional logic. The diagram below shows the device peripherals with Core Interconnect target ports.

Figure 3-2. Core Interconnect Diagram



The red blocks in the diagram above indicate designated MPU (Memory protection units) on the associated target ports. The device MPUs allow for up to 8 programmable regions. Additional details related the Memory Protection Unit, can be found in the device Section 3.12 chapter.

Note

The placement of 6 L2OCRAM Banks across the 3 interconnects (R5SS0 VBUSM, R5SS1 VBUSM and VBUSM CORE Interconnect) has been done such that cores in a cluster can have faster access (lesser latency) to the banks closer to that particular cluster. In other words, R5SS0_Core0 and R5SS0_Core1 cores will have faster access latency to its near L2OCSRAM banks (BANK0 and BANK1) placed on R5SS0 VBUSM interconnect. Similarly, R5SS1_Core0 and R5SS1_Core1 cores will have faster access latency to its near L2OCSRAM banks (BANK2 and BANK3) placed on R5SS1 VBUSM interconnect. All the 4 cores, will have the same but slower access latency to the common L2OCSRAM banks (BANK4 and BANK5) as compared to their near banks. Furthermore, all the 4 cores will have slower access latency to their far L2OCSRAM banks (BANK2 and BANK3 for cluster R5SS0 and BANK0 and BANK1 for cluster R5SS1) as compared to common banks.

To summarize, for particular cores in a cluster, below is the L2OCRAM Bank access latency comparison:

Access latency of near banks < Access latency of common banks < Access latency of far banks

Table 3-1. CORE VBUSM Initiator-Target Table

This table lists initiator and target end point connections for the CORE VBUSM Interconnect. A cell can contain one of the following:

- Y – Connection **does** exist between initiator and target.
- N – Connection **does NOT** exist between initiator and target.

Targets	Initiators												
	R5FSS 0-0*	R5FSS 0-1*	R5FSS 1-0*	R5FSS 1-1*	HSM	HSM_TC0 R/W*	HSM_TC1 R/W*	SoC_TC0 R/W*	SoC_TC1 R/W*	DEBUGSS	ICSSM PRU0	ICSSM PRU1	CPSW3G
R5FSS0-0	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS0-1^	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS1-0	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
R5FSS1-1^	Y	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	Y
OCSRAM (BANK0)	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
OCSRAM (BANK1)	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
OCSRAM (BANK2)	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
OCSRAM (BANK3)	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
OCSRAM (BANK4)	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
OCSRAM (BANK5)	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
FLASH_DATA_REG0,1,3	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
MBOX_SRAM	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
HSM	Y	Y	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y
DTHE	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
FSS/OSPI	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
ICSSM	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	N	Y
MMC0	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
STM_STIM	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
MCRC	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
CORE VBUSP (Port0)	N	N	N	N	N	Y	N	Y	N	Y	Y	N	N
CORE VBUSP (Port1)	N	N	N	N	Y	N	Y	N	Y	N	N	Y	N

Note

* These initiators have separate read and write ports.

Note

^ Accessible only with LOCKSTEP mode disabled. Any access with LOCKSTEP mode enabled results in an error response.

3.3 CORE VBUSP Interconnect

VBUSP is a very simple and easy to implement protocol that is pended such that only a single transaction can be outstanding at any given time. VBUSP protocol is classified as a point-to-point, pended interface protocol. The design is split into multi layers of VBUSP interconnect for performance requirements. The diagram below shows the peripherals which are target ports for the CORE VBUSP interconnect.

VIM interconnect is a local VBUSP interconnect which allows a low latency path to the dedicated VIM from each R5SS. Since this is locally connected before the CORE VBUSP interconnect, access is restricted only from each R5SS core to its own VIM module.

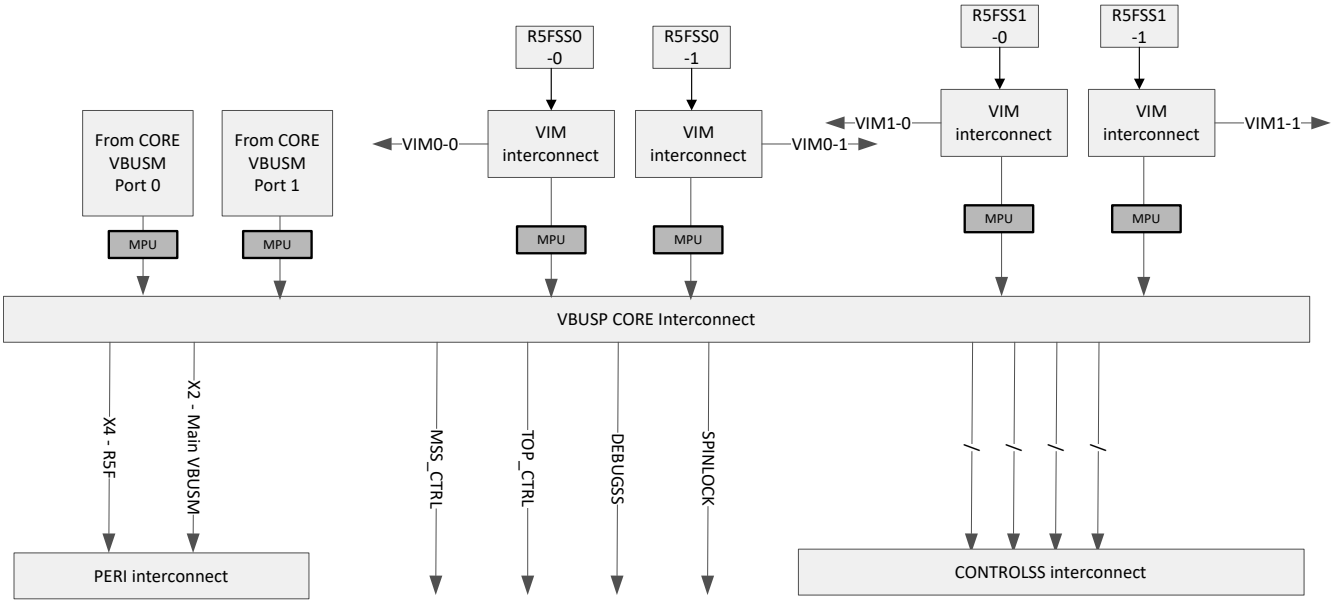


Figure 3-3. CORE VBUSP Interconnect Diagram

The grey blocks are MPU (Memory Protection Units) on the target ports. These are used to protect data and configuration spaces by managing the accesses to these memory regions. The MPUs above can have up to 16 programmable regions. For more details on MPU, please refer to Section 3.12.

Table 3-2. CORE VBUSP Initiator-Target Table

This table lists the initiator and target end point connections for the CORE VBUSP Interconnect. A cell can contain one of the following:

- Y – Connection **does** exist between initiator and target.
- N – Connection **does NOT** exist between initiator and target.

Targets	Initiators					
	R5FSS 0-0_AHB	R5FSS 0-1_AHB	R5FSS 1-0_AHB	R5FSS 1-1_AHB	CORE VBUSM (Port0)	CORE VBUSM (Port1)
EPWM_G0	Y	Y	Y	Y	Y	Y
EPWM_G1	Y	Y	Y	Y	Y	Y
EPWM_G2	Y	Y	Y	Y	Y	Y
EPWM_G3	Y	Y	Y	Y	Y	Y
EPWM_G0_WLINK	Y	Y	Y	Y	Y	Y
EPWM_G1_WLINK	Y	Y	Y	Y	Y	Y
EPWM_G2_WLINK	Y	Y	Y	Y	Y	Y
EPWM_G3_WLINK	Y	Y	Y	Y	Y	Y
SCR_ADC_0	Y	N	N	N	N	N
SCR_ADC_1	N	Y	N	N	N	N
SCR_ADC_2	N	N	Y	N	N	N
SCR_ADC_3	N	N	N	Y	N	N
SCR_ADC_4	N	N	N	N	Y	N
SCR_ADC_5	N	N	N	N	N	Y
MISC PERIPH	Y	Y	Y	Y	Y	Y
SCR_FSI_0	Y	Y	Y	Y	Y	Y
SCR_FSI_1	Y	Y	Y	Y	Y	Y
MISC CONFIG0	Y	Y	Y	Y	Y	Y
MISC CONFIG1	Y	Y	Y	Y	Y	Y
PERI_R5FSS0-0*	Y	N	N	N	N	N
PERI_R5FSS0-1*	N	Y	N	N	N	N
PERI_R5FSS1-0*	N	N	Y	N	N	N
PERI_R5FSS1-1*	N	N	N	Y	N	N
PERI VBUSP (Port0)*	N	N	N	N	Y	N
PERI VBUSP (Port1)*	N	N	N	N	N	Y
SPINLOCK	Y	Y	Y	Y	Y	Y
DEBUGSS	Y	Y	Y	Y	Y	Y
MSS_CTRL	Y	Y	Y	Y	Y	Y
TOP_CTRL	Y	Y	Y	Y	Y	Y
VIM0-0	Y	N	N	N	N	N
VIM0-1	N	Y	N	N	N	N
VIM1-0	N	N	Y	N	N	N
VIM1-1	N	N	N	Y	N	N

Note

*These targets connect to initiator ports on the PERI interconnect.

3.4 PERI VBUSP Interconnect

PERI VBUSP interconnect connects with the CORE VBUSP interconnect through the target ports each dedicated for individual initiators on the CORE VBUSP. The diagram below shows the peripherals which are target ports for the CORE VBUSP interconnect.

Figure 3-4. PERI VBUSP Interconnect Diagram

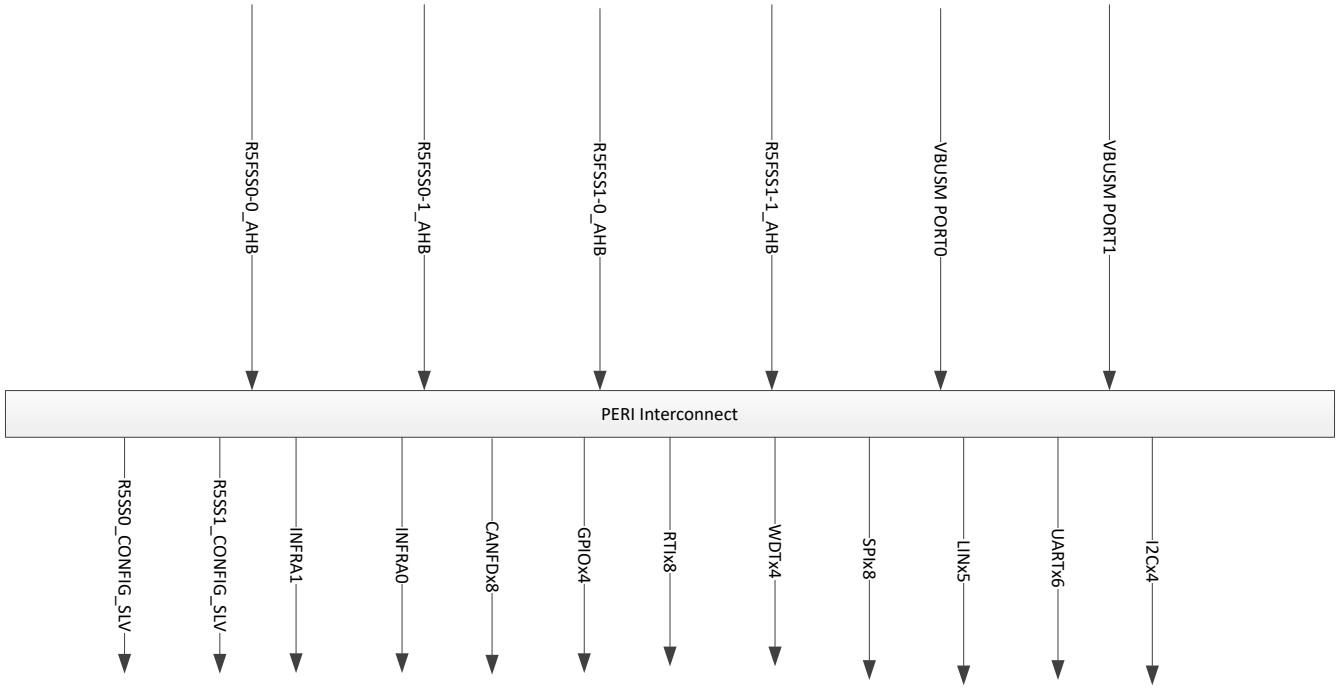


Table 3-3. PERI VBUSP Initiator-Target Table

This table lists initiator and target end point connections for the PERI VBUSP Interconnect. A cell may contain one of the following:

- Y – Connection **does** exist between initiator and target.
- N – Connection **does NOT** exist between initiator and target.

Targets	Initiators					
	R5FSS 0-0_AHB	R5FSS 0-1_AHB	R5FSS 1-0_AHB	R5FSS 1-1_AHB	PERI VBUSP (Port0)	PERI VBUSP (Port1)
GPIO0	Y	N	N	N	Y	Y
GPIO1	N	Y	N	N	Y	Y
GPIO2	N	N	Y	N	Y	Y
GPIO3	N	N	N	Y	Y	Y
WDT0	Y	N	N	N	Y	Y
WDT1	N	Y	N	N	Y	Y
WDT2	N	N	Y	N	Y	Y
WDT3	N	N	N	Y	Y	Y
SPI0	Y	Y	Y	Y	Y	Y
SPI1	Y	Y	Y	Y	Y	Y
SPI2	Y	Y	Y	Y	Y	Y
SPI3	Y	Y	Y	Y	Y	Y
SPI4	Y	Y	Y	Y	Y	Y
SPI5	Y	Y	Y	Y	Y	Y
SPI6	Y	Y	Y	Y	Y	Y
SPI7	Y	Y	Y	Y	Y	Y
UART0	Y	Y	Y	Y	Y	Y
UART1	Y	Y	Y	Y	Y	Y
UART2	Y	Y	Y	Y	Y	Y
UART3	Y	Y	Y	Y	Y	Y
UART4	Y	Y	Y	Y	Y	Y
UART5	Y	Y	Y	Y	Y	Y
LIN0	Y	Y	Y	Y	Y	Y
LIN1	Y	Y	Y	Y	Y	Y
LIN2	Y	Y	Y	Y	Y	Y
LIN3	Y	Y	Y	Y	Y	Y
LIN4	Y	Y	Y	Y	Y	Y
I2C0	Y	Y	Y	Y	Y	Y
I2C1	Y	Y	Y	Y	Y	Y
I2C2	Y	Y	Y	Y	Y	Y
I2C3	Y	Y	Y	Y	Y	Y
RTI0	Y	Y	Y	Y	Y	Y
RTI1	Y	Y	Y	Y	Y	Y
RTI2	Y	Y	Y	Y	Y	Y
RTI3	Y	Y	Y	Y	Y	Y
RTI4	Y	Y	Y	Y	Y	Y
RTI5	Y	Y	Y	Y	Y	Y
RTI6	Y	Y	Y	Y	Y	Y
RTI7	Y	Y	Y	Y	Y	Y
CANFD0	Y	Y	Y	Y	Y	Y
CANFD1	Y	Y	Y	Y	Y	Y
CANFD2	Y	Y	Y	Y	Y	Y
CANFD3	Y	Y	Y	Y	Y	Y
CANFD4	Y	Y	Y	Y	Y	Y
CANFD5	Y	Y	Y	Y	Y	Y

Table 3-3. PERI VBUSP Initiator-Target Table (continued)

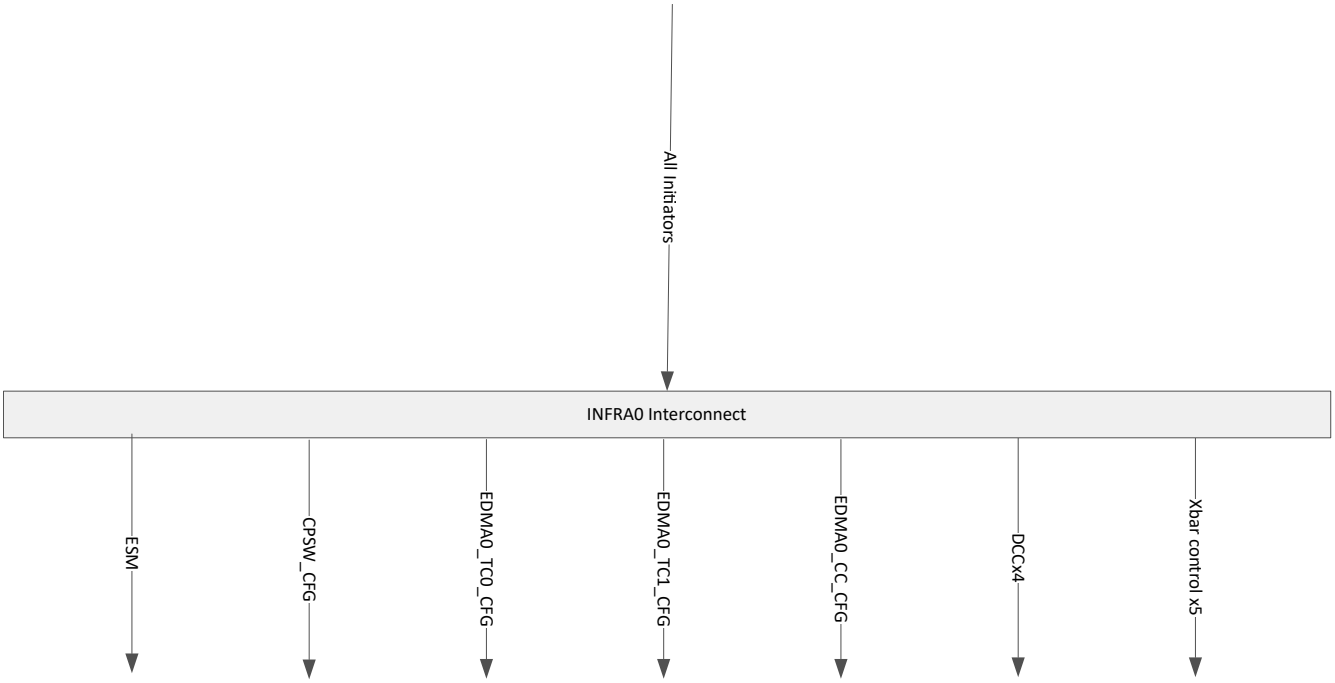
This table lists initiator and target end point connections for the PERI VBUSP Interconnect. A cell may contain one of the following:

- Y – Connection **does** exist between initiator and target.
- N – Connection **does NOT** exist between initiator and target.

Targets	Initiators					
	R5FSS 0-0_AHB	R5FSS 0-1_AHB	R5FSS 1-0_AHB	R5FSS 1-1_AHB	PERI VBUSP (Port0)	PERI VBUSP (Port1)
CANFD6	Y	Y	Y	Y	Y	Y
CANFD7	Y	Y	Y	Y	Y	Y
INFRA0	Y	Y	Y	Y	Y	Y
INFRA1	Y	Y	Y	Y	Y	Y

3.5 INFRA0 VBUSP Interconnect

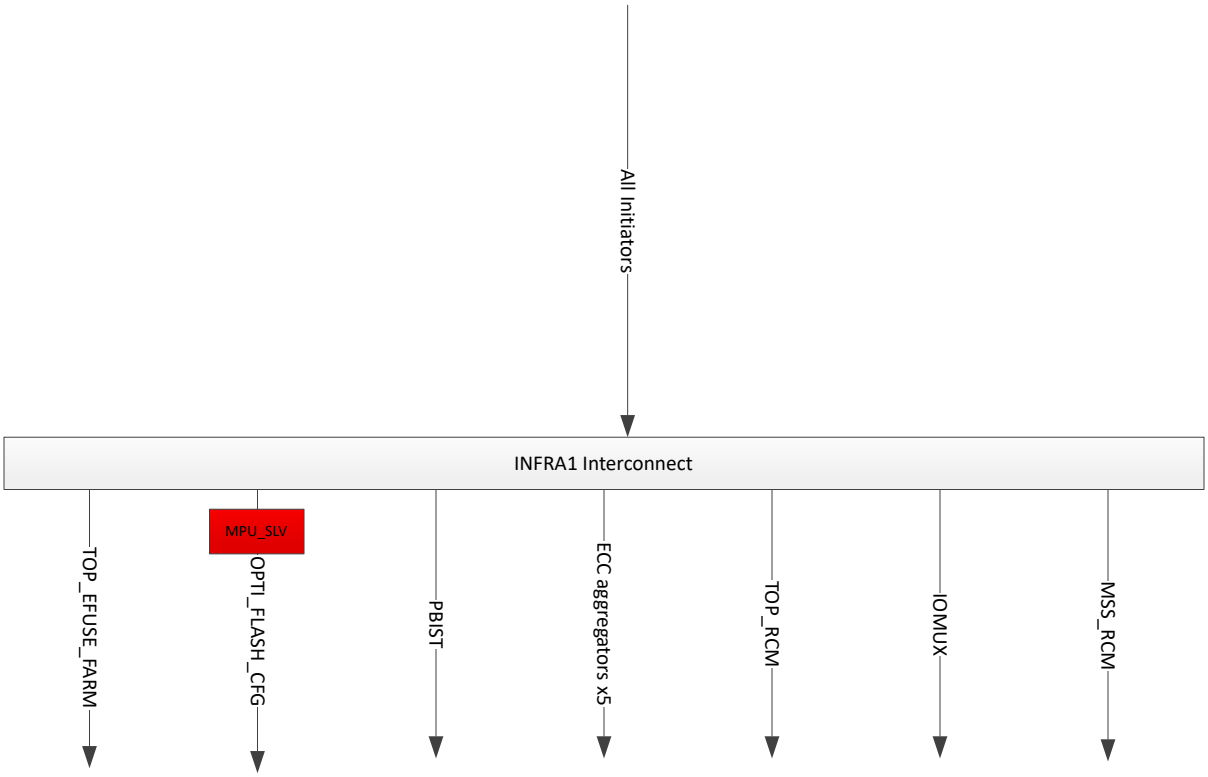
INFRA0 VBUSP interconnect connects with the PERI VBUSP interconnect through a single target port catering to all initiators on the PERI VBUSP. Accessing a particular target by multiple initiators at the same time will be arbitrated in this interconnect. The diagram below shows the peripherals which are target ports for the INFRA0 VBUSP interconnect.



There is no access restriction since its a single initiator, multiple target interconnect.

3.6 INFRA1 VBUSP Interconnect

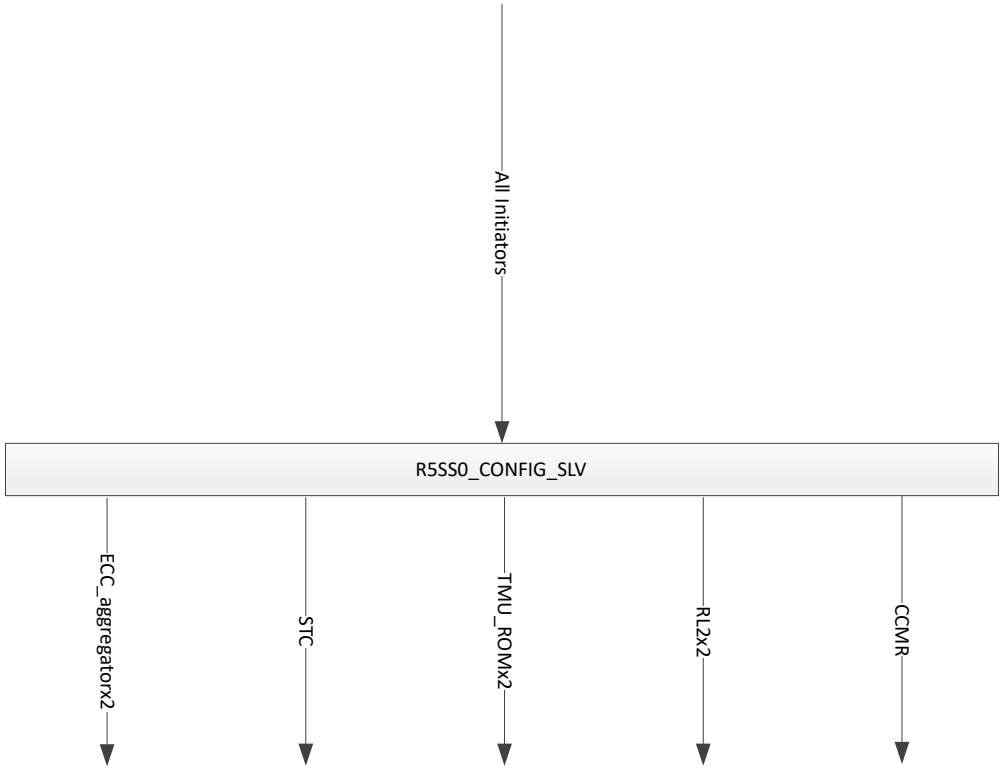
INFRA1 VBUSP interconnect connects with the PERI VBUSP interconnect through a single target port catering to all initiators on the PERI VBUSP. Accessing a particular target by multiple initiators at the same time will be arbitrated in this interconnect. The diagram below shows the peripherals which are target ports for the INFRA1 VBUSP interconnect.



There is no access restriction since its a single initiator, multiple target interconnect.

3.7 R5SS0 CONFIG SLV Interconnect

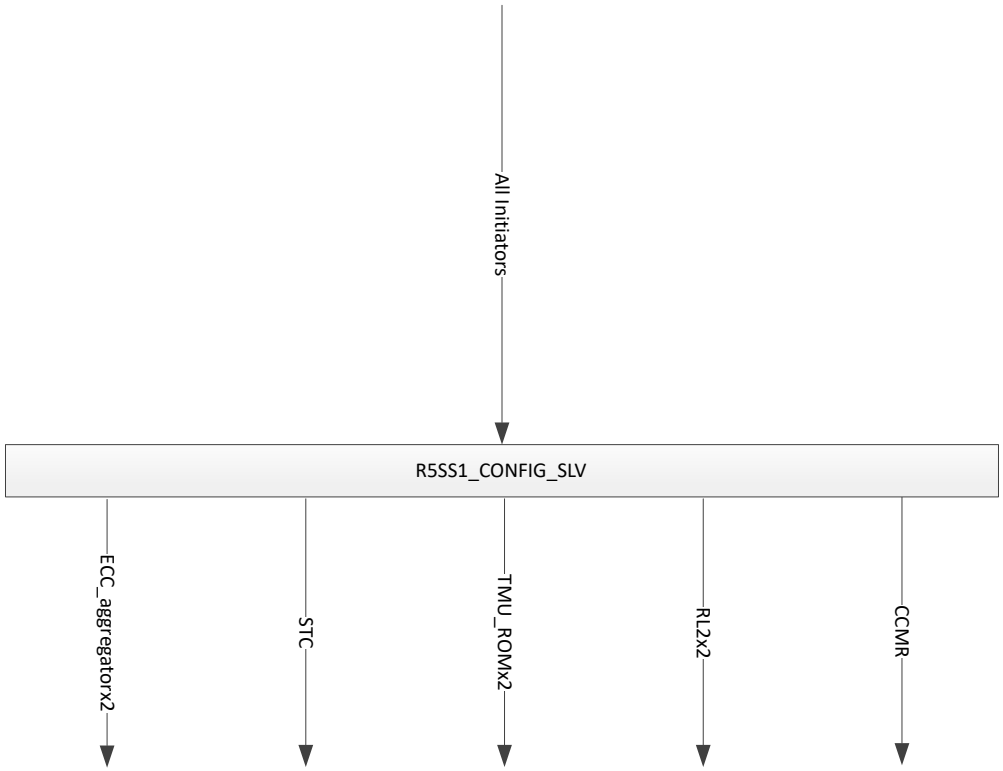
The diagram below shows the peripherals which are target ports for the R5SS0 CONFIG SLV interconnect.



There is no access restriction since its a single initiator, multiple target interconnect.

3.8 R5SS1 CONFIG SLV Interconnect

The diagram below shows the peripherals which are target ports for the R5SS1 CONFIG SLV interconnect.

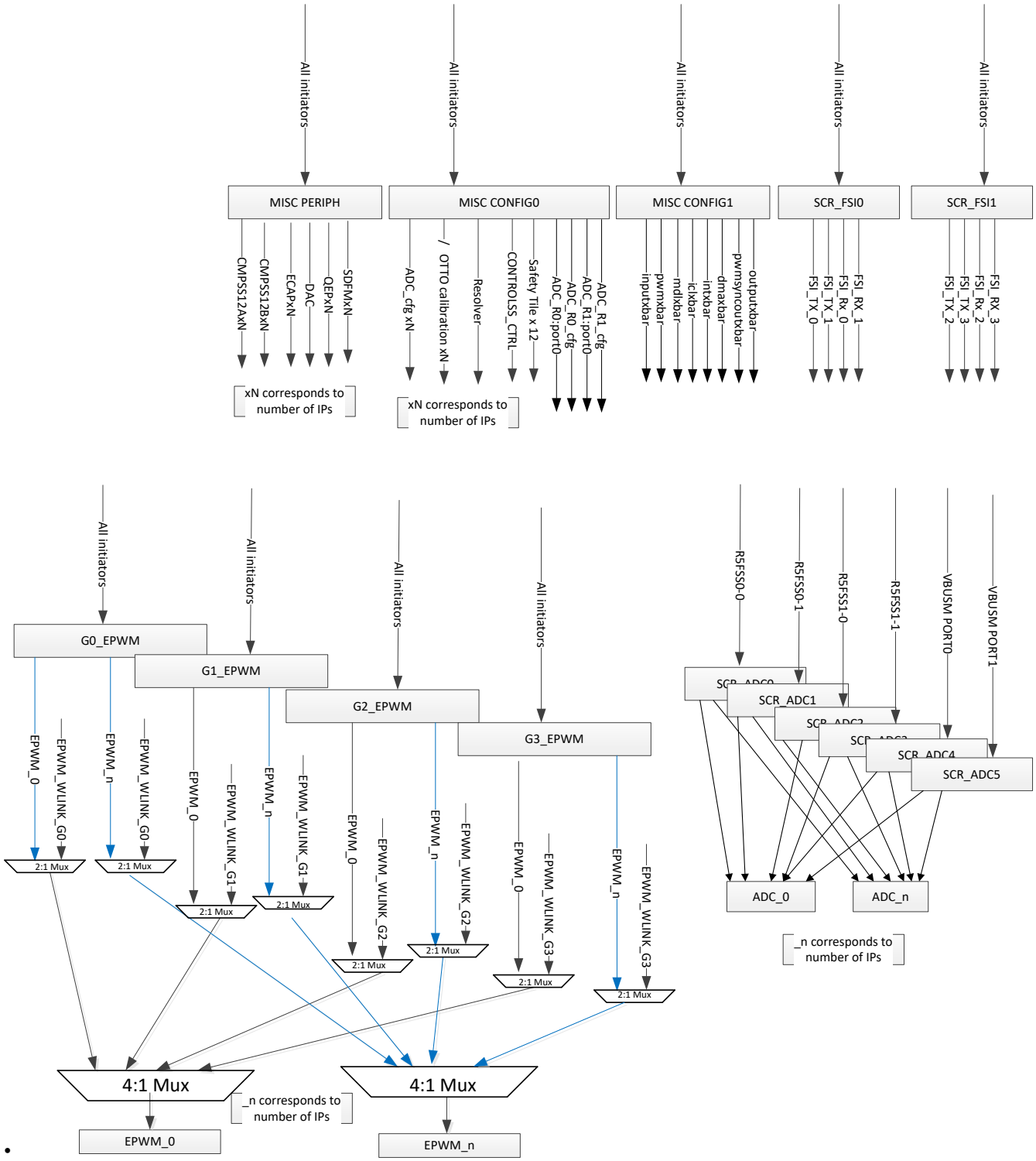


There is no access restriction since its a single initiator, multiple target interconnect.

3.9 CONTROLSS Interconnect

CONTROLSS interconnect is divided into below list of separate interconnect connected to the CORE VBUSP interconnect individually. Since these are connected to the CORE VBUSP interconnect separately, each of this interconnect can be accessed in parallel by different initiators without any arbitration. Accessing a single CONTROLSS interconnect by multiple initiators at the same time will be arbitrated.

- MISC PERIPH
- MISC CONFIG0
- MISC CONFIG1
- SC_FSI0 (FSITX[0:1] and FSIRX[0:1])
- SCR_FSI1 (FSITX[2:3] and FSIRX[2:3])
- G0_EPWM, G1_EPWM, G2_EPWM, G3_EPWM
- ADC0, ADC1, ADC2, ADC3, ADC4, ADC5



-
- MISC PERIPH, MISC CONFIG0, MISC CONFIG1, SCR_FSI0 and SCR_FSI1 are single initiator, multiple targets as shown in the diagram.
- EPWM interconnect are divided into 4 groups G0_EPWM, G1_EPWM, G2_EPWM and G3_EPWM accessed using different address regions in the memory map. Any initiator can access an EPWM group while another initiator is accessing a different EPWM group simultaneously. Each interconnect has n target ports depending on number of EPWM in the design. After the interconnect, a 4:1 Static

Mux can be configured per EPWM using CONTROLSS_GLOBAL_CTRL.EPWM_STATICXBAR_SEL0 & CONTROLSS_GLOBAL_CTRL.EPWM_STATICXBAR_SEL1 register, which statically assigns that EPWM to any of the selection groups – G0 to G3.

- Additionally, for each EPWM group, there is a 4KB write only alias region added in the device memory map (CONTROLSS_G0_EPWM_WLINK to CONTROLSS_G3_EPWM_WLINK), using which selected EPWM instances of that particular group can be written together in a single write. The registers CONTROLSS_GLOBAL_CTRL.CONTROLSS_G0_EPWM_WLINK to CONTROLSS_GLOBAL_CTRL.CONTROLSS_G3_EPWM_WLINK contains EPWM enables to select which instances can be written together.

SCR_ADC0, SCR_ADC1,.. SCR_ADCn are different interconnect per intiator (R5FSS0-0_AHB, R5FSS0-1_AHB,R5FSS1-0_AHB, R5FSS1-1_AHB, CORE VBUSP (Port0), and CORE VBUSP (Port1)). The target ports are based on number of ADCs in the design. Each initiator can independently access any ADC register without any arbitration. In other words, the same ADC result register can be accessed by multiple initiators simultaneously without contention.

3.10 Interconnect Safety

In order to ensure the safety of data through the interconnect, redundancy has been implemented in VBUSM and VBUSP interconnect. For VBUSP, data and control signals are passed through a redundant interconnect and compared. For VBUSM, ECC of the data is generated and is passed through redundant interconnect. The comparison will happen for ECC of the data. The control signals are directly compared without any ECC generation. The status of comparison from Main and Redundant interconnect are available in MSS_CTRL MMR.

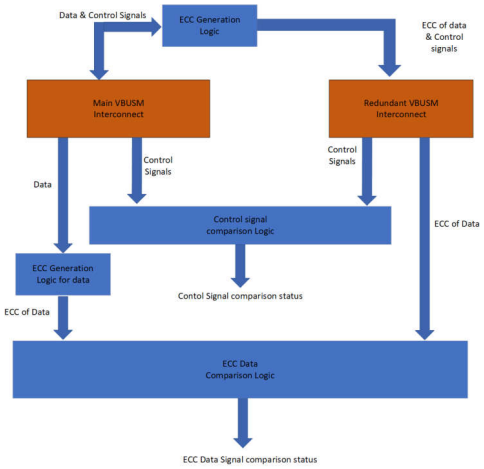


Figure 3-5. VBUSM Interconnect

The following interconnects are safety compliant:

1. CORE VBUSM
2. CORE VBUSP
3. PERI VBUSP
4. R5SS0 VBUSM
5. R5SS1 VBUSM

The VBUSM Interconnect follows the ECC based VBUSM safety architecture, CORE VBUSP and PERI VBUSP follows VBUSP Safety architecture as discussed above. All the Initiators/Targets of these Interconnects are safety compliant.

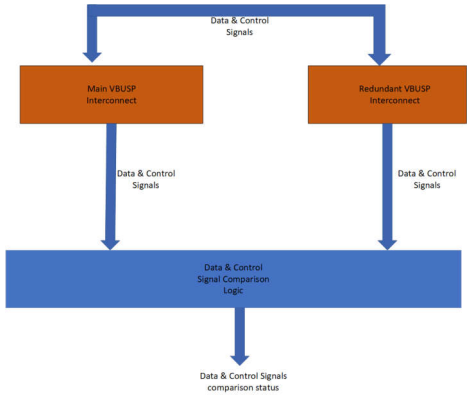


Figure 3-6. VBUSP Interconnect

3.11 Bus Safety Errors

3.11.1 Error Signaling Integration

The bus safety errors which gets generated from VBUSP and VBUSM Interconnects will get aggregated and are available as status registers in the MSS_CTRL. The Registers which contain various error status are:

1. ***_INTAGG_STATUS_RAW** – These Registers capture raw error status for each safety compliant Initiator/ Target.
2. ***_INTAGG_STATUS** – These Registers capture masked error status for each Initiator/Target which are safety compliant. The masking is done by programming the register - ***_INTAGG_MASK** with appropriate value. Masking will override the corresponding bit to be default value irrespective of raw error status.
3. ***_RD_BUS_SAFETY_ERR** – This Register contains more information such as Single error, Double Error that had occurred in the data. Additionally, it contains if an error occurred in command bus, write bus, write status, or read bus of the Target/Slave Port.

The Masked errors from various Targets/Slaves are aggregated and sent to ESM. There are three such signals : Aggregated_VBUSP_error_H, Aggregated_VBUSM_error_H and Aggregated_VBUSM_error_L. The Initiators/ Targets errors which are aggregated and used for generation of these signals are given in the below table.

Table 3-4. Initiators/Targets errors aggregated and sent to ESM GROUP0

MSS ESM GROUP0 Channel No.	Description	Comments
31	Aggregated_VBUSP_error_H • R5SS0_0_AHB • R5SS0_1_AHB • R5SS1_0_AHB • R5SS1_1_AHB • MAIN_VBUSP (Aggregated error for all VBUSP Initiators and Targets) • PERI_VBUSP (Aggregated error for all VBUSP Initiators and Targets)	Aggregated High interrupt line for VBUSP slaves. Only compare error is mapped to this line.

Table 3-5. Initiators/Targets errors aggregated and sent to ESM GROUP1

MSS ESM GROUP1 Channel No.	Description	Comment
1	Aggregated_VBUSM_error_H • R5SS0_0_RD • R5SS0_1_RD • R5SS0_0_WR • R5SS0_1_WR • R5SS0_0_S • R5SS0_1_S • R5SS1_0_RD • R5SS1_1_RD • R5SS1_0_WR • R5SS1_1_WR • R5SS1_0_S • R5SS1_1_S • Debugss • HSM_M • CPSW • OCSRAM(Bank0) • OCSRAM(Bank1) • OCSRAM(Bank2) • OCSRAM(Bank3) • OCSRAM(Bank4) • OCSRAM(Bank5) • SoC_TC_0_RD • SoC_TC_1_RD • SoC_TC_0_WR • SoC_TC_1_WR • HSM_TC_0_RD • HSM_TC_1_RD • HSM_TC_0_WR • HSM_TC_1_WR • ICSSM0_PRU0 • ICSSM0_PRU1 • OSPI • MCRC • DTHE • SCRP0 • SCRP1 • HSM	Aggregated High interrupt line for VBUSM peripherals. DED(Double Error Detection) of data and compare errors of control signals are mapped to this line.

Table 3-5. Initiators/Targets errors aggregated and sent to ESM GROUP1 (continued)

MSS ESM GROUP1 Channel No.	Description	Comment
2	Aggregated_VBUSM_error_L • R5SS0_0_RD • R5SS0_1_RD • R5SS0_0_WR • R5SS0_1_WR • R5SS0_0_S • R5SS0_1_S • R5SS1_0_RD • R5SS1_1_RD • R5SS1_0_WR • R5SS1_1_WR • R5SS1_0_S • R5SS1_1_S • Debugss • HSM_M • MSS_CPSW • OCSRAM(Bank0) • OCSRAM(Bank1) • OCSRAM(Bank2) • OCSRAM(Bank3) • OCSRAM(Bank4) • OCSRAM(Bank5) • SoC_TC_0_RD • SoC_TC_1_RD • SoC_TC_0_WR • SoC_TC_1_WR • HSM_TC_0_RD • HSM_TC_0_WR • HSM_TC_1_RD • HSM_TC_1_WR • ICSSM0_PRU0 • ICSSM0_PRU1 • OSPI • MCRC • DTHE • CORE VBUSP(Port0) • CORE VBUSP(Port1) • HSM_S • ICSSM • MBOX_SRAM • STM_STIM • MMC	Aggregated Low interrupt line for VBUSM slaves. SEC (Single Error Correction) error is mapped to this line.

3.11.2 Programming sequence

Bus Infrastructure Safety is disabled by default.

- The first step is to enable Bus Safety for the MSS subsystem globally. For this, write 0x7 to **MSS_CTRL.MSS_BUS_SAFETY_CTRL.MSS_BUS_SAFETY_CTRL_ENABLE**.
- User can enable safety on a node mentioned in above table by writing to the multibit field – **MSS_CTRL.<NODE>_BUS_SAFETY_CTRL_ENABLE**

- Write 0x7: To enable safety for the Node
- Write 0x0: To disable safety for the Node

3.11.3 Diagnostic Check Mechanism

1. Enable MSS bus safety errors.
   - **MSS_CTRL.MSS_BUS_SAFETY_CTRL.MSS_BUS_SAFETY_CTRL _ENABLE** = 0x7
2. Enable bus safety for each interface. Taking MSS L2 Bank A VBUSM interface as a reference.
   - Set the mask bit for the respective source in **MSS_CTRL.MSS_VBUSM_SAFETY_x_ERRAGG_MASK** register. In this example,

     MSS_CTRL.MSS_VBUSM_SAFETY_H0_ERRAGG_MASK.MSS_VBUSM_SAFETY_H0_ERRAGG_

     MASK_L2RAM0_VBUSM_ERRH = 1

     Note

     x can be a high or low priority setting for the corresponding source.
   - **MSS_CTRL.MSS_L2_A_BUS_SAFETY_CTRL.MSS_L2_A_BUS_SAFETY_CTRL_ENABLE** = 0x7
3. For double/single error injection on data,
   - **MSS_CTRL.MSS_L2_A_BUS_SAFETY_FI.MSS_L2_A_BUS_SAFETY_FI_DED** = 0x1; (For Double Error Detection)

     MSS_CTRL.MSS_L2_A_BUS_SAFETY_FI.MSS_L2_A_BUS_SAFETY_FI_SEC = 0x1; (For Single Error Correction)
   - **MSS_CTRL.MSS_L2_A_BUS_SAFETY_FI.MSS_L2_A_BUS_SAFETY_FI_DATA** = 0x1<<i;
     - Double/single errors can be injected only on 32-bit segments of data at a time.
     - i=0 for data[31:0]

       i=1 for data[63:32]

       i=2 for data[95:64] and so on.
     - For controller interfaces, it will be read data to which error will be injected, and for target interfaces it will be write data to which error will get injected.
   - The write access is to be followed by a read to the endpoint of the bus interface. The address should be selected based on the FI_DATA value.
     - **WR_MEM_32(MSS_L2_U_BASE + 0x2000 + i*0x4, wr_data)**; // sufficient for targets like MSS_L2
     - **rd_data = RD_MEM_32(MSS_L2_U_BASE + 0x2000 + i*0x4)**; // sufficient for controllers like CR5A_AXI_READ
   - Upon detection of DED/SEC error on the interface, an ESM error gets triggered and the following sequence needs to be executed by the ISR to clear the error.
     - **MSS_CTRL.MSS_L2_A_BUS_SAFETY_CTRL.MSS_L2_A_BUS_SAFETY_CTRL_ERR_CLEAR** = 0x1;
     - Once the error at the interface is cleared, clear the ESM status register.
     - Register **MSS_CTRL.MSS_L2_A_BUS_SAFETY_ERR** is read to confirm whether the ERROR is SEC/DED.
   - Before Exiting the ISR need to do the below setting to ensure the fault injection is removed.
     - **MSS_CTRL.MSS_L2_A_BUS_SAFETY_FI.MSS_L2_A_BUS_SAFETY_FI_SEC** = 0x0
     - **MSS_CTRL.MSS_L2_A_BUS_SAFETY_FI.MSS_L2_A_BUS_SAFETY_FI_DED** = 0x0
   - All the Bus-Safety SEC errors in MSS are aggregated to a single ESM line. So, before exiting the ISR, the corresponding bit in the aggregated registers **MSS_CTRL.MSS_VBUSM/P_x_ERRAGG_STATUS** should be written 1 to clear the status.
4. For redundancy on the bus interface signals,
   - – **MSS_CTRL.MSS_L2_A_BUS_SAFETY_FI.MSS_L2_A_BUS_SAFETY_FI_MAIN** = 0x1<<i;
     - i=0 checks redundancy on the main command interface
     - i=1 checks redundancy on the main write interface

    - i=2 checks redundancy on the main write status interface
    - i=3 checks redundancy on the main read interface
    - For vbusp interfaces, only the main command interface is checked.
  - **MSS_CTRL_Ptr.MSS_L2_A_BUS_SAFETY_FI.MSS_L2_A_BUS_SAFETY_FI_SAFE** = 0x1<<i;
    - i=0 checks redundancy on the safe command interface
    - i=1 checks redundancy on the safe write interface
    - i=2 checks redundancy on the safe write status interface
    - i=3 checks redundancy on the safe read interface
    - For vbusp interfaces, only the safe command interface is checked.
  - To inject redundancy errors on all the command, read, write and write status interface signals simultaneously, the following sequence is executed,
    - **MSS_CTRL.MSS_L2_A_BUS_SAFETY_FI.MSS_L2_A_BUS_SAFETY_FI_GLOBAL_MAIN** = 0x1; // for the main interface
    - **MSS_CTRL.MSS_L2_A_BUS_SAFETY_FI.MSS_L2_A_BUS_SAFETY_FI_GLOBAL_SAFE**= 0x1; // for the safe interface
- Upon detection of a redundancy error on the interface, an ESM error gets triggered and the following sequence needs to be executed by the ISR to clear the error.
  - **MSS_CTRL.MSS_L2_A_BUS_SAFETY_CTRL.MSS_L2_A_BUS_SAFETY_CTRL_ERR_CLEAR** = 0x1;
  - Once the error at the interface is cleared, clear the ESM status register.
- Before Exiting the ISR, one needs to do the below setting to ensure the fault injection is removed.
  - **MSS_CTRL.MSS_L2_A_BUS_SAFETY_FI_ST. MSS_L2_A_BUS_SAFETY_FI_MAIN** = 0x0
  - **MSS_CTRL.MSS_L2_A_BUS_SAFETY_FI_ST. MSS_L2_A_BUS_SAFETY_FI_SAFE** = 0x0
  - **MSS_CTRL.MSS_L2_A_BUS_SAFETY_FI_ST.MSS_L2_A_BUS_SAFETY_FI_GLOBAL_MAIN** = 0x0
  - **MSS_CTRL.MSS_L2_A_BUS_SAFETY_FI_ST.MSS_L2_A_BUS_SAFETY_FI_GLOBAL_SAFE**= 0x0

3.12 System Memory Protection Unit (MPU)/Firewalls

The device incorporates multiple system Memory Protection Units (MPU) aka Firewall in the interconnect for security purpose or to ensure freedom from interference (FFI) in safety application. The choice of using the MPU for security or for FFI is an application level decision.

The MPU works by allowing access to the underlying memory map (Peripheral or Memory) for authorized initiators and disallowing access to other initiators.

3.12.1 MPU Overview

The MPU has the following features:

- Supports multiple programmable address ranges
- Supports secure and debug access privileges
- Supports read, write, and execute access privileges
- Distinguishes access from different initiators based on an Identifier (Privilege ID)
- Generates an interrupt when there is addressing or protection violation

3.12.2 MPU Instances

There are 23 MPU firewall instances in the device placed at various points in the interconnect topology. The firewalls are referred to as "Initiator side firewall" (firewall is located right at the initiator port) or "Target side firewall" (firewall is located right before the target port), depending on where the firewalls are present in the topology. All MPUs are identical from application perspective. However, all the target side MPUs have 8 Regions and Initiator side MPUs have 16 regions (initiator MPUs provide more regions to handle peripheral spaces) As evident from Figure3-1, the Initiator side firewalls are intended to protect the peripheral space while the Target side firewalls protect individual Target memory space (memory bank or a unique target space).

Initiator side MPUs

The Initiator side firewalls as shown in Figure 3-3 (CORE VBUSP Interconnect Diagram) are listed below.

- SCRM2SCRP0
- SCRM2SCRP1
- R5SS0_CORE0_AHB_MST
- R5SS0_CORE1_AHB_MST
- R5SS1_CORE0_AHB_MST
- R5SS1_CORE1_AHB_MST

Target side MPUs

The Target side firewalls as shown in Figure 3-2 (Core Interconnect Diagram) are listed below.

- R5SS0_CORE0_AXIS_SLV
- R5SS0_CORE1_AXIS_SLV
- R5SS1_CORE0_AXIS_SLV
- R5SS1_CORE1_AXIS_SLV
- L2OCRAM_BANK0_SLV
- L2OCRAM_BANK1_SLV
- L2OCRAM_BANK2_SLV
- L2OCRAM_BANK3_SLV
- L2OCRAM_BANK4_SLV
- L2OCRAM_BANK5_SLV
- FSS_CONFIG_SLV
- MBOX_RAM_SLV
- HSM_SLV
- DTHE_SLV
- FSS/OSPI_SLV
- R5SS0_CONFIG_SLV
- R5SS1_CONFIG_SLV

3.12.3 MPU Functional Description

3.12.3.1 Functional Operation

The MPU performs access permission check for each Bus access reaching the MPU and decides to allow the access to pass unmodified further to the target memory if it passes the permission check OR disallow the access and fault the access back to the initiator if it fails the permission check.

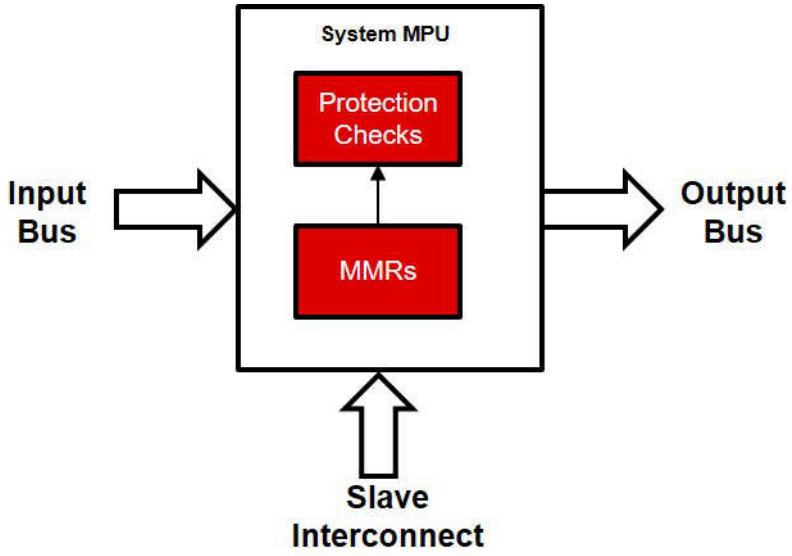


Figure 3-7. MPU Top Level Diagram

Privilege ID (PrivID)

Every initiator is associated with an Identifier referred to as Priv ID. See section **ISC (Initiator-side Security Control)** for how to assign a PrivID to each initiator. This PrivID identifies the controller for privilege purposes and accompanies all bus accesses made on behalf of that controller. That is, when a controller triggers a bus access command, the PrivID is carried alongside the command.

Privilege Level (Priv)

Every initiator access on the input bus is associated with a privilege level. Two privilege levels are supported: supervisor and user. The privilege level is inherited from the code running on the corresponding processor. For example, ARM processor has User mode and Supervisor Mode.

Secure/Non Secure Access

Every initiator is associated with a security level identifier Secure or Non Secure. When an initiator triggers a bus access command, the security level is carried alongside the command. See section **ISC (Initiator-side Security Control)** for how to assign a security level to each initiator.

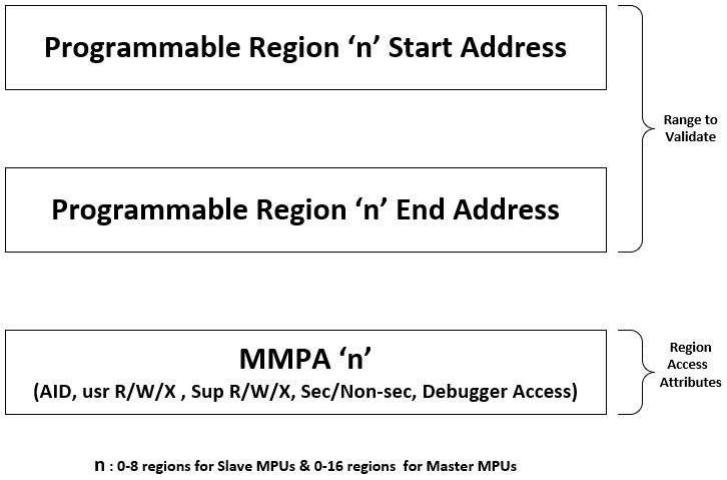
Debugger Access

When a JTAG based debugger makes access to a peripheral or Memory through the AHB-Ap port, such an access is qualified by a EMU (Emulator) signal.

MPU Region

Each MPU is associated with multiple MPU regions. Each region of the MPU is programmable. The programming specifies which Initiators are allowed access, the address range where the access is allowed and additional access attributes such as read/Write/execute etc.

A high level view of each MPU region is shown below:



• The start address PROGRAMMABLE_n_START_ADDRESS specifies the start address of the nemory protection region 'n'.

• The End address PROGRAMMABLE_n_END_ADDRESS specifies the end address of the memory protection region 'n'.

Note

The granularity of the MPU in this device is 1KB. The lower 10 bits of the Programmable start and end address are a don't care.

Actual MPU_start_address[31:0] = PROGRAMMABLE_n_START_ADDRESS[31:10] : 10'b0

Actual MPU_end_address[31:0] = PROGRAMMABLE_n_END_ADDRESS[31:10]:10'b1111111111

• The MPPA(Memory Protection Permission Attribute) register PROGRAMMABLE_n_MPPA specifies the permission attributes for region 'n'. AID15_0 field (Register bits 25-10) specifies the privID's for which the rule of this region applies. There is an AID register bit for each possible privID (0 to 15) and an AIDX that covers privIDs not configured. The other bits specify the access attributes such as User Read/Write/Execute or Supervisor ReadWrite/Execute as well as Non secure access and Emulation/Debugger access.

Rule

1. The MPU works by first checking the transfer's privID against the AID settings. The privID is used to lookup the associated AID bit. If the AID bit is 0, then the range does not cover that Initiator/ID and the range is not checked (although other ranges with different AID setting will) for this transfer. If the AID bit is 1, then the range does cover that Initiator/PrivID and the permissions are checked.

2. The transfer secure and debug parameters are checked against the MPPA values to detect an allowed access. The two bits (NS and EMU) provide 3 permission levels.

- If the NS is set, the range is non-secure and any security or debug initiator may access the range.
- If the NS is not set, the range is secure only and only secure level accesses are allowed.
- If Emulation(debugger) access is happening then the permission check is only two bits EMU and NS
  - If EMU is set then the region allows access to debugger (does not check for R/W/PRiv permissions)
  - If NS is set then region allows access to debugger (does not check for R/W/Priv permissions)

3. For Non Debugger(Regular Initiator access from within the Device) the read, write and execute permissions are also checked.

Table 3-6. Protection Levels for Secure and Debug attributes

NS	EMU	Description
0	0	Region *x* is secure without debug: Only secure accesses are allowed. Debug accesses are not allowed. Non-secure accesses are also not allowed
0	1	Region *x* is secure with debug: Only secure and debug accesses are allowed. Non-secure accesses are not allowed
1	-	Region *x* is non-secure: All accesses (non-secure, secure and debug) are allowed.

4. There is a set of permissions for supervisor mode and another for user mode. The “priv” attribute of the transfer determines the mode of access.

- If priv = 1, the supervisor rwx bits are checked
- If priv = 0, the user rwx bits are checked against the same attributes

The Priv attribute signal on the bus depends on the mode of the CPU making the bus access.

Table 3-7. Request Type Access Controls

Bit	Description
PROGRAMMABLE_x_MPPA[5] SR	Supervisor may read
PROGRAMMABLE_x_MPPA[4] SW	Supervisor may write
PROGRAMMABLE_x_MPPA[3] SX	Supervisor may execute
PROGRAMMABLE_x_MPPA[2] UR	User may read
PROGRAMMABLE_x_MPPA[1] UW	User may write
PROGRAMMABLE_x_MPPA[0] UX	User may execute

For each bit, a value of 1 permits the access type, and 0 denies it. So, setting the UX bit to 1 means that a controller in user mode may execute from corresponding region. The MPU allows the programmer to specify each of these 6 bits separately. Thus 64 different combinations are possible but programs might not use all of them .

5. Each region outputs whether the transfer is allowed or disallowed or don’t care.

- If the AIDs match and the transfer is within the address range and the permissions match, the region indicates access allowed.
- If the AIDs match and the transfer is within the address range and the permissions don’t match , the region indicates access disallowed.
- In all other cases the region is a don’t care.

The region outputs are aggregated to decide if the access is allowed or disallowed.

The MPU configuration used in this device does not allow access by default (Blocking by default).

- If none of the region allow access, the access is not allowed
- In case of overlapping regions, If any of the region does not allow access, the access is not allowed.
- The access is allowed only if one or more regions allow access and none of the regions disallow access.

In other words the final permission is the lowest of each type of permission from any hit range. (So If a transfer hits 2 regions , one that is rw and another r, the final permission is just r).

Note

Due to MPU architecture limitation, in case of a Cacheable access from R5 CPU, if the cache line(32Byte) access falls in the last 32Bytes of the MPU region, the MPU incorrectly indicates an access fault. Hence it is recommended that the application does not perform a cacheable access on the last 32Bytes of an MPU region. This limitation does not exist for non Cacheable access from R5 or any access from non R5 initiators.

3.12.3.2 Protection of the MPU Configuration Registers

Accesses to the PROGRAMMABLE_x_START_ADDRESS, PROGRAMMABLE_x_END_ADDRESS and PROGRAMMABLE_x_MPPA registers are also protected. All non-debug writes must be by a supervisor controller. If the PROGRAMMABLE_x_MPPA[7] NS bit is 0, then all writes must be by a secure controller. In addition, the NS bit can be modified only by a secure controller. A register write with invalid permissions results in protection fault and interrupt generation.

A debug write is only allowed if NS = 1 or the EMU = 1 regardless of the secure or privilege attributes. Neither faults are recorded nor interrupts are generated for debug accesses.

3.12.3.3 MPU Interrupt Requests

The MPU module generates the following interrupts when there is any kind of MPU violation:

Table 3-8. MPU Interrupts

Interrupt	Description
mpu_addr_err_intr	Addressing violation interrupt
mpu_prot_err_intr	Protection violation interrupt.

The **mpu_addr_err_intr** interrupt occurs when a read or write access is made to a non-existent register address in the MPU configuration space.

The **mpu_prot_err_intr** interrupt occurs when there is a protection violation. Two kinds of protection violation is possible.

1. When the access on the input bus violates the MPU rules as defined in Functional Operation section or
2. When the access violates the protection of MPU configuration registers as defined in Protection of the MPU Configuration Registers section.

The transfer parameters that caused the above violations are saved in **MPU.FAULT_ADDRESS** and **MPU.FAULT_STATUS** registers. This violation status MMRs can be cleared by writing to **MPU.FAULT_CLEAR** register.

The above interrupts can be enabled by writing to **MPU.INTERRUPT_ENABLE** register. The register **MPU.INTERRUPT_RAW_STATUSSET** register can be read to know the raw interrupt status. The register **MPU.INTERRUPT_ENABLED_STATUSCLEAR** can be read to know the enabled interrupt status. The interrupt can be cleared by writing ‘1’ to **MPU.INTERRUPT_ENABLED_STATUSCLEAR** register.

MPU Interrupt Aggregation

The error Interrupts from all MPUs in the device are aggregated and provided to each R5SS core as **R5FSSx_COREy_INTR_MPU_ADDR_ERRAGG (#69**) and **R5FSSx_COREy_INTR_MPU_PROT_ERRAGG(#70)** interrupts.

This aggregated address error interrupt can be controlled by the MMR **MSS_CTRL.MPU_ADDR_ERRAGG_R5SSx_CPUy_MASK**. There is one register per associated R5SS Core. Each bit represents one MPU which can be masked or enabled to generate the aggregated interrupt. The status of the Address error interrupt can be read from the MMRs **MSS_CTRL.MPU_ADDR_ERRAGG_R5SSx_CPUy_STATUS** and the raw status can be read from **MSS_CTRL.MPU_ADDR_ERRAGG_R5SSx_CPUy_STATUS_RAW .** The aggregated interrupt can be cleared

by writing ‘1’ to the **MSS_CTRL.MPU_ADDR_ERRAGG_R5SSx_CPUy_STATUS** register. The raw status can be cleared by writing ‘1’ to the **MSS_CTRL.MPU_ADDR_ERRAGG_R5SSx_CPUy_STATUS_RAW** register.

Note: To clear the aggregated status, the source MPU error interrupt must be cleared first followed by clearing the aggregated interrupt STATUS register.

Similarly the aggregated protection error interrupt is associated with the registers **MSS_CTRL.MPU_PROT_ERRAGG_R5SSx_CPUy_MASK**, **MSS_CTRL.MPU_PROT_ERRAGG_R5SSx_CPUy_STATUS** and **MSS_CTRL.MPU_PROT_ERRAGG_R5SSx_CPUy_STATUS_RAW.**

Similar to the above mechanism, the interrupts from all the MPUs in the device are aggregated and provided to HSM-ESM as **HSM_MPU_AGGR_ADDR_ERR**(#22) and **HSM_MPU_AGGR_PROT_ERR**(#23) Error.

The relevant MMRs to mask/enable individual MPU errors is **HSM_SOC_CTRL.HSM_MPU_ERRAGG_MASK0 and HSM_SOC_CTRL. HSM_MPU_ERRAGG_MASK1 respectively.**

Note: The MPU source interrupt can be cleared only by the entity who has access to the respective MPU config space (Typically HSM. However, other cores can be given access to MPU by opening up the HSM_SLV MPU). The aggregated interrupt can be cleared by the respective R5 Core themselves, by writing to the respective aggregated status register once the source interrupt is cleared.

CPU Behavior when its access is faulted by MPU

When a violation is triggered in a MPU, the corresponding R5 CPU whose access caused this violation will receive a suitable response from the Bus interconnect.

1. When a MPU present on CORE VBUSM interconnect violates, both Read or Write transaction causing the violation will result in the corresponding R5 Core taking an Abort exception.
2. When a MPU present on CORE VBUSP interconnect violates during a Read transaction, the corresponding R5 Core will take an Abort exception.
3. When a MPU present on the CORE VBUSP interconnect violates during a Write transaction the corresponding R5 Core will get an interrupt on the interrupt line **R5FSSx_COREy_INTR_AHB_WRITE_ERR(#135).** It will not take an Abort exception.

3.12.4 MPU Parameters

The position of the MPU with respect to the interconnect topology (Fig 3-2 and Fig 3-3) decides what the MPU is responsible for protecting and which initiators can perform access through a given MPU.

For example : Notice that for MPU R5SS0_CORE_AHB_MST, R5SS0_CORE0 is the only initiator. Hence any regions configured inside this MPU only refer to the R5SS0_COR0 AID and not bother about other AIDs.

Another example is MPU SCRM2SCRP0 where the R5SS cores do not have access (Refer to the Interconnect Initiator-Target Table). The access is possible from HSM EDMA or SOC EDMA or ICSS or Debugger.

There are 23 MPUs in this device. The parameters of each MPU and the memory regions associated with each MPU is listed in Table 6-15 .

Table 3-9. MPU Parameters Table

MPU	Controller/ Target	ID	MPU Config Addr	Num of MPU Regions	Memory/Peripheral space Protected by the MPU			
					Num of protected segments*	Segment Num	Segment Start Address	Segment Size
R5SS0_CORE0_AXIS_SLV	Target	0	0x400A0000	8	5	0	0x78000000	64*1024
						1	0x78100000	64*1024
						2	0x74000000	8*1024*1024
						3	0x74800000	8*1024*1024
						4	0x78060000	944

Table 3-9. MPU Parameters Table (continued)

MPU	Controller/ Target	ID	MPU Config Addr	Num of MPU Regions	Memory/Peripheral space Protected by the MPU			
					Num of protected segments*	Segment Num	Segment Start Address	Segment Size
R5SS0_CORE1_AXIS_SLV	Target	1	0x400C0000	8	5	0	0x78200000	32*1024
						1	0x78300000	32*1024
						2	0x75000000	8*1024*1024
						3	0x75800000	8*1024*1024
						4	0x78260000	944
R5SS1_CORE0_AXIS_SLV	Target	2	0x400E0000	8	5	0	0x78400000	64*1024
						1	0x78500000	64*1024
						2	0x76000000	8*1024*1024
						3	0x76800000	8*1024*1024
						4	0x78460000	944
R5SS1_CORE1_AXIS_SLV	Target	3	0x40100000	8	5	0	0x78600000	32*1024
						1	0x78700000	32*1024
						2	0x77000000	8*1024*1024
						3	0x77800000	8*1024*1024
						4	0x78660000	944
L2OCRAM_BANK0_SLV	Target	4	0x40020000	8	1	0	0x70000000	512*1024
L2OCRAM_BANK1_SLV	Target	5	0x40040000	8	1	0	0x70080000	512*1024
L2OCRAM_BANK2_SLV	Target	6	0x40060000	8	1	0	0x70100000	512*1024
L2OCRAM_BANK3_SLV	Target	7	0x40080000	8	1	0	0x70180000	512*1024
L2OCRAM_BANK4_SLV	Target	8	0x402C0000	8	1	0	0x70200000	512*1024
L2OCRAM_BANK5_SLV	Target	9	0x402E0000	8	1	0	0x70280000	512*1024
FSS/ OSPI_SLV	Target	10	0x40160000	8	3	0	0x60000000	128*1024*1024
						1	0x80000000	128*1024*1024
						2	0x88000000	128*1024*1024
MBOX_RAM_SLV	Target	11	0x40140000	8	1	0	0x72000000	16*1024
HSM_SLV	Target	12	0x40240000	8	2	0	0x20000000	128*1024*1024
						1	0x40000000	128*1024*1024

Table 3-9. MPU Parameters Table (continued)

MPU	Controller/ Target	ID	MPU Config Addr	Num of MPU Regions	Memory/Peripheral space Protected by the MPU			
					Num of protected segments*	Segment Num	Segment Start Address	Segment Size
DTHE_SLV	Target	13	0x40120000	8	1	0	0xCE000000	16*1024*1024
FSS_CONFIG_SLV	Target	14	0x40260000	4	11	0	0x53800000	4*1024
						1	0x53801000	4*1024
						2	0x53802000	4*1024
						3	0x53806000	4*1024
						4	0x53807000	4*1024
						5	0x53808000	4*1024
						6	0x5380B000	4*1024
						7	0x5380C000	4*1024
						8	0x5380D000	4*1024
						9	0x5380E000	4*1024
						10	0x5380F000	4*1024
R5SS0_CONFIG_SLV	Target	15	0x40280000	4	8	0	0x53000000	528
						1	0x53003000	528
						2	0x53210000	60
						3	0x53500000	172
						4	0x53020000	8*1024
						5	0x53024000	8*1024
						6	0x53212000	1024
						7	0x53213000	1024
R5SS1_CONFIG_SLV	Target	16	0x402A0000	4	8	0	0x53004000	528
						1	0x53007000	528
						2	0x53211000	60
						3	0x53510000	172
						4	0x53028000	8*1024
						5	0x5302C000	8*1024
						6	0x53214000	1024
						7	0x53215000	1024
SCRM2SCRP0	Controller	17	0x40180000	16	1	0	0x50000000	256*1024*1024
SCRM2SCRP1	Controller	18	0x401A0000	16	1	0	0x50000000	256*1024*1024
R5SS0_CORE0_AHB_MST	Controller	19	0x401C0000	16	1	0	0x50000000	256*1024*1024
R5SS0_CORE1_AHB_MST	Controller	20	0x401E0000	16	1	0	0x50000000	256*1024*1024
R5SS1_CORE0_AHB_MST	Controller	21	0x40200000	16	1	0	0x50000000	256*1024*1024
R5SS1_CORE1_AHB_MST	Controller	22	0x40220000	16	1	0	0x50000000	256*1024*1024

Table 3-9. MPU Parameters Table (continued)

MPU	Controller/ Target	ID	MPU Config Addr	Num of MPU Regions	Memory/Peripheral space Protected by the MPU			
					Num of protected segments*	Segment Num	Segment Start Address	Segment Size
* - Each segment is a contiguous address range in the memory map of the device which the corresponding MPU is responsible for protecting. The Segment start address and Segment size columns lists these segments.								

3.12.5 MPU Default HW Configuration

Each MPU region has a default value based on the Device type. The default value of MPU region based on device type is captured in table below

Table 3-10. Default Hardware MPU Configurations: HSFS Device

MPU Instance/Region#	PROGRAMMABLESTART_ADDRESS	PROGRAMMABLEEND_ADDRESS	PROGRAMMABLEMPPA	Priv IDsAllowed	Initiator access Allowance**
R5SS0_CORE0_AXIS_SLV/ Region1	0x00000000	0xFFFFFFFF	0x03FFFFFF	0 to 15	Open to All Initiators
R5SS0_CORE1_AXIS_SLV/ Region1	0x00000000	0xFFFFFFFF	0x03FFFFFF	0 to 15	Open to All Initiators
R5SS1_CORE0_AXIS_SLV/ Region1	0x00000000	0xFFFFFFFF	0x03FFFFFF	0 to 15	Open to All Initiators
R5SS1_CORE1_AXIS_SLV/ Region1	0x00000000	0xFFFFFFFF	0x03FFFFFF	0 to 15	Open to All Initiators
L2OCRAM_BANK0_SLV/ Region1	0x00000000	0xFFFFFFFF	0x03FFFFFF	0 to 15	Open to All Initiators
L2OCRAM_BANK1_SLV/ Region1	0x00000000	0xFFFFFFFF	0x03FFFFFF	0 to 15	Open to All Initiators
L2OCRAM_BANK2_SLV/ Region1	0x00000000	0xFFFFFFFF	0x03FFFFFF	0 to 15	Open to All Initiators
L2OCRAM_BANK3_SLV/ Region1	0x00000000	0xFFFFFFFF	0x03FFFFFF	0 to 15	Open to All Initiators
MBOX_RAM_SLV/ Region1	0x00000000	0xFFFFFFFF	0x03FFFFFF	0 to 15	Open to All Initiators
HSM_SLV/ Region1	0x00000000	0xFFFFFFFF	0x00000838	1	HSM Only
HSM_SLV/Region2	0x 44000000	0x440007FF	0x03FFFFFF	0-15	Open to All Initiators
DTHE_SLV/ Region1	0x00000000	0xFFFFFFFF	0x00000838	1	HSM Only
L2OCRAM_BANK4_SLV/ Region1	0x00000000	0xFFFFFFFF	0x03FFFFFF	0-15	Open to All Initiators
SCRM2SCRP0 SCRM2SCRP1 R5SS0_CORE0_MST R5SS0_CORE1_MST R5SS1_CORE0_MST R5SS1_CORE1_MST (Regions 1 to 5)	0x50000000	0x535FFFFF	0x0000087F	0-15	Open to All Initiators*
	0x53600000	0x53600FFF	0x0000087F	1	HSM*
	0x53601000	0x53801FFF	0x0000087F	0-15	Open to All Initiators*
	0x53802000	0x53802FFF	0x0000087F	1	HSM*
	0x53803000	0x5FFFFFFF	0x0000087F	0-15	Open to All Initiators*
L2OCRAM_BANK5_SLV/ Region1	0x00000000	0xFFFFFFFF	0x03FFFFFF	0-15	Open to All Initiators
FSS_DATA_SLV/ Region1	0x00000000	0xFFFFFFFF	0x03FFFFFF	0-15	Open to All Initiators
FSS_CONFIG_SLV/ Region1	0x00000000	0xFFFFFFFF	0x03FFFFFF	0-15	Open to All Initiators
R5SS0_CONFIG_SLV/ Region1	0x00000000	0xFFFFFFFF	0x03FFFFFF	0-15	Open to All Initiators
R5SS1_CONFIG_SLV/ Region1	0x00000000	0xFFFFFFFF	0x03FFFFFF	0-15	Open to All Initiators
*Modifed by ROM					
**Access allowance interpretation based on the default ISC Configuration Table for PrivID-Initiator Mapping					

Table 3-11. Default Hardware/ROM MPU Configurations: HSSE Device

MPU Instance/Region#	PROGRAMMABLE_START_ADDRESS	PROGRAMMABLE_END_ADDRESS	PROGRAMMABLE_MPPA	Priv IDs Allowed	Priv ID Enabled **
R5SS0_CORE0_AXIS_SLV/ Region1	0x00000000	0xFFFFFFFF	0x00000838	1	HSM Only

Table 3-11. Default Hardware/ROM MPU Configurations: HSSE Device (continued)

MPU Instance/Region#	PROGRAMMABLE_START_ADDRESS	PROGRAMMABLE_END_ADDRESS	PROGRAMMABLE_MPPA	Priv IDs Allowed	Priv ID Enabled **
R5SS0_CORE1_AXIS_SLV/ Region1	0x00000000	0xFFFFFFFF	0x00000838	1	HSM Only
R5SS1_CORE0_AXIS_SLV/ Region1	0x00000000	0xFFFFFFFF	0x00000838	1	HSM Only
R5SS1_CORE1_AXIS_SLV/ Region1	0x00000000	0xFFFFFFFF	0x00000838	1	HSM Only
L2OCRAM_BANK0_SLV/ Region1	0x70000000	0x7007FFFF	0x0000487F	1,4	HSM*,R5CORE0*, EDMA-TC0 (MSS TPTC0)*, EDMA-TC1 (MSS TPTC1)*
L2OCRAM_BANK1_SLV/ Region1	0x70080000	0x700FFFFF	0x0000487F	1,4	HSM*,R5CORE0*, EDMA-TC0 (MSS TPTC0)*, EDMA-TC1 (MSS TPTC1)*
L2OCRAM_BANK2_SLV/ Region1	0x70100000	0x7017FFFF	0x0000487F	1,4	HSM*,R5CORE0*, EDMA-TC0 (MSS TPTC0)*, EDMA-TC1 (MSS TPTC1)*
L2OCRAM_BANK3_SLV/ Region1	0x70180000	0x701FFFFF	0x0000487F	1,4	HSM*,R5CORE0*, EDMA-TC0 (MSS TPTC0)*, EDMA-TC1 (MSS TPTC1)*
MBOX_RAM_SLV/ Region1	0x72000000	0x72003FFF	0x0000487F	1,4	HSM*,R5CORE0*
HSM_SLV/Region1	0x00000000	0xFFFFFFFF	0x00000838	1	HSM Only
HSM_SLV/Region2	0x 44000000	0x440007FF	0x03FFFFFF	1	HSM Only
DTHE_SLV/ Region1	0x00000000	0xFFFFFFFF	0x00000838	1	HSM Only
SCRM2SCRP0 SCRM2SCRP1 R5SS0_CORE0_MST R5SS0_CORE1_MST R5SS1_CORE0_MST R5SS1_CORE1_MST (Regions 1 to 5)	0x50000000	0x535FFFFF	0x0000487F	1,4	HSM*,R5CORE0*, EDMA-TC0 (MSS TPTC0)*, EDMA-TC1 (MSS TPTC1)*
	0x53600000	0x53600FFF	0x0000087F	1	HSM*
	0x53601000	0x53801FFF	0x0000487F	1,4	HSM*,R5CORE0*, EDMA-TC0 (MSS TPTC0)*, EDMA-TC1 (MSS TPTC1)*
	0x53802000	0x53802FFF	0x0000087F	1	HSM*
	0x53803000	0x5FFFFFFF	0x0000487F	1,4	HSM*,R5CORE0*, EDMA-TC0 (MSS TPTC0)*, EDMA-TC1 (MSS TPTC1)*
L2OCRAM_BANK4_SLV/ Region1	0x70200000	0x7027FFFF	0x0000487F	1,4	HSM*,R5CORE0*, EDMA-TC0 (MSS TPTC0)*, EDMA-TC1 (MSS TPTC1)*
L2OCRAM_BANK5_SLV/ Region1	0x70280000	0x702FFFFF	0x0000487F	1,4	HSM*,R5CORE0*, EDMA-TC0 (MSS TPTC0)*, EDMA-TC1 (MSS TPTC1)*
FSS_DATA_SLV/ Region1	0x60000000	0x61FFFFFF	0x0000487F	1,4	HSM*,R5CORE0*, EDMA-TC0 (MSS TPTC0)*, EDMA-TC1 (MSS TPTC1)*

Table 3-11. Default Hardware/ROM MPU Configurations: HSSE Device (continued)

MPU Instance/Region#	PROGRAMMABLE_START_ADDRESS	PROGRAMMABLE_END_ADDRESS	PROGRAMMABLE_MPPA	Priv IDs Allowed	Priv ID Enabled **
FSS_CONFIG_SLV/ Region1	0x53800000	0x5387FFFF	0x0000487F	1,4	HSM*,R5CORE0*
R5SS0_CONFIG_SLV/ Region1	0x00000000	0xFFFFFFFF	0x00000838	1	HSM Only
R5SS1_CONFIG_SLV/ Region1	0x00000000	0xFFFFFFFF	0x00000838	1	HSM Only
*Modified by ROM					
**Access allowance interpretation based on the default ISC Configuration Table for PrivID-Initiator Mapping					

3.12.6 ISC (Initiator-side Security Control)

The Initiator side Security Control (ISC) module is responsible for assigning the PrivID values to the Initiators. Each initiator is associated with a ID allocation register which assigns the Priv ID to the corresponding initiator.

Allocation of PrivIDs to all initiators based on the ID Allocation register must be done under the control of HSM.

The default Hardware ISC configuration is shown in below table.

Table 3-12. Default Hardware ISC Configurations

ISC Config Addr	Config Address	Priv ID at Reset
ISC_CTRL_REG_HSM_CM4	0x4000 0400	0x1
ISC_CTRL_REG_HSM_TPTC_A0	0x4000 0404	0x1
ISC_CTRL_REG_HSM_TPTC_A1	0x4000 0408	0x1
ISC_CTRL_REG_MSS_R5FA0_AXI	0x4000 0800	0x4
ISC_CTRL_REG_MSS_R5FB0_AXI	0x4000 0804	0x5
ISC_CTRL_REG_MSS_R5FA1_AXI	0x4000 0808	0x6
ISC_CTRL_REG_MSS_R5FB1_AXI	0x4000 080C	0x7
ISC_CTRL_REG_MSS_TPTC_A0	0x4000 0810	0x4
ISC_CTRL_REG_MSS_TPTC_A1	0x4000 0814	0x4
ISC_CTRL_REG_MSS_ETHERNET_DMA	0x4000 0818	0xA
ISC_CTRL_REG_DBG_JTAG	0x4000 081C	0xB
ISC_CTRL_REG_ICSSM0_PDSP0	0x4000 0820	0x9
ISC_CTRL_REG_ICSSM0_PDSP1	0x4000 0824	0x9

Note

The PrivID for the JTAG debugger is determined by the DOM signal from HSM

Note

It is recommended to have only the HSM PrivID to be 0x1.

Note

The ISC has bypass control which when set, will mean the Initiator will drive the Priv ID instead of being assigned from ISC ID allocation register. This Bypass needs to be set for EDMA initiators since they are capable of inheriting the PrivID from the CPU programing the DMA transfer task.

3.12.6.1 ID Allocation

The general format of the ID allocation register ISC_CTRL_REG_<INITIATOR> is shown in **ISC ID allocation Register** below:

3.12.6.1.1

Table 3-13. ISC ID allocation Register

Bit	Name	Type	Reset	Description	In the device	Routed to IP
31:22	reserved	r	0x0	Reserved	Reserved	N
21	PASS	rw	0	No privID replacement. A value of 1 will pass through privid value. A value of 0 will replace privid with priv_id field value	Yes	N

Table 3-13. ISC ID allocation Register (continued)

Bit	Name	Type	Reset	Description	In the device	Routed to IP
20	NONSEC	rw	0	Make outgoing non-secure. A value of 1 forces secure clear, others do nothing. Do not set both sec and nonsec.	Yes	Y
19	reserved	r	0x0	Reserved	Reserved	N
18:16	SEC	rw	0x0	Make outgoing secure. A value of 3&APOS;B111 forces secure set, others do nothing. Do not set both sec and nonsec.	Yes	Y
15:12	reserved	r	0x0	Reserved	Reserved	N
11:08	PRIVID	rw	0x1	Privilege ID configuration	Yes	Y
07:00	reserved	r	0x0	Reserved	Reserved	N

4 Module Integration

This chapter describes the integration details for each module in the device, including information about clocks, resets, interrupts, DMA events and other hardware requests.

4.1 ADC Integration

There are 5x Analog-to-Digital Converter (ADC) modules integrated in the device.

Note

For each ADC[0:4]:

- Analog input channels ADCIN[0:5] have dedicated pins.
- Analog input channels ADCIN[6:7] are tied to shared ADC_CAL[0:1] pins, respectively.
- Refer to *ADC Triggers* for the full list of ADC sample trigger options.

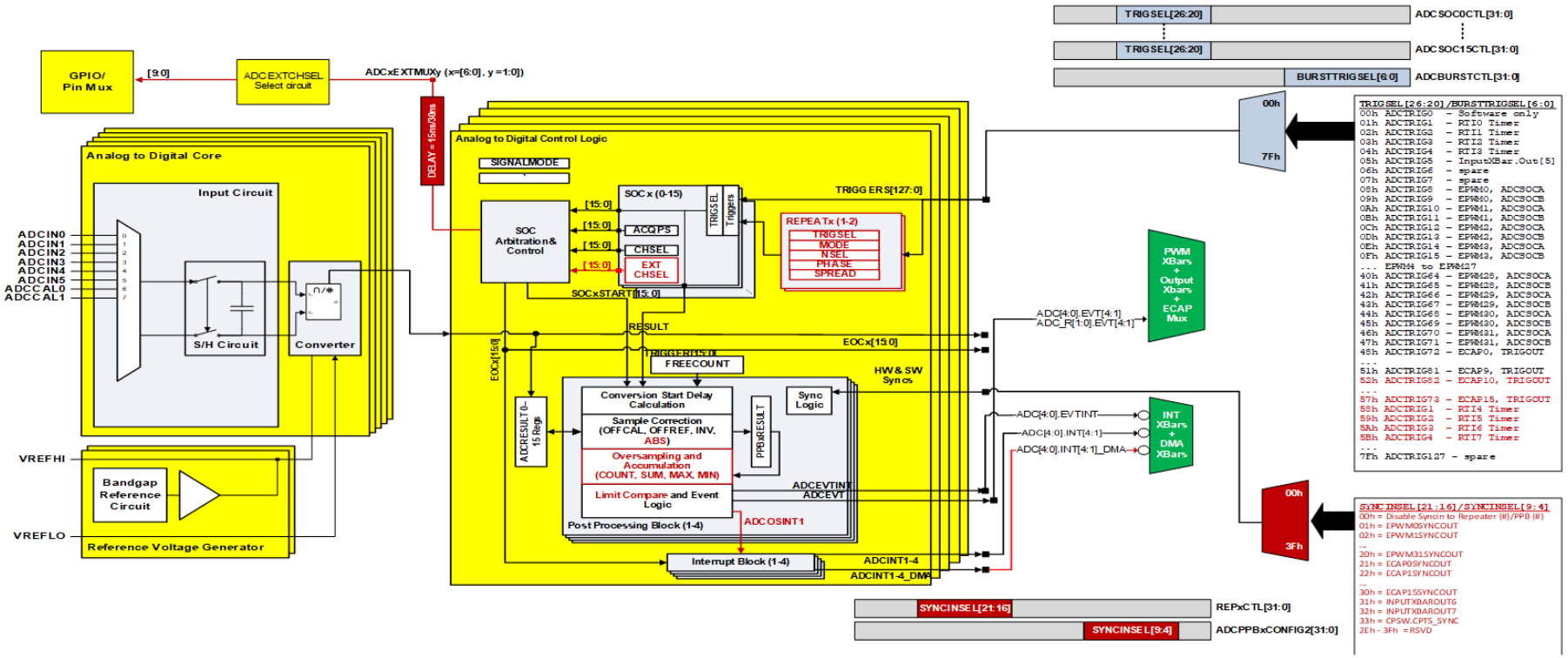


Figure 4-1. ADC Integration Diagram

4.2 Resolver to Digital Convertor Integration

There are 2x Resolver-to-Digital Converter (RDC) modules integrated in the device as shown in Figure 4-2.

Note

For each RDC Module:

- Resolver Module input channels have ADC_R0_AIN[3:0] and ADC_R1_AIN[3:0] dedicated pins.
- ADC_VREFLO_G3 and ADC_VREFHI_G3 are dedicated voltage reference pins for Resolver modules.
- Refer to Figure 7-117 for full list of all RDC pins.

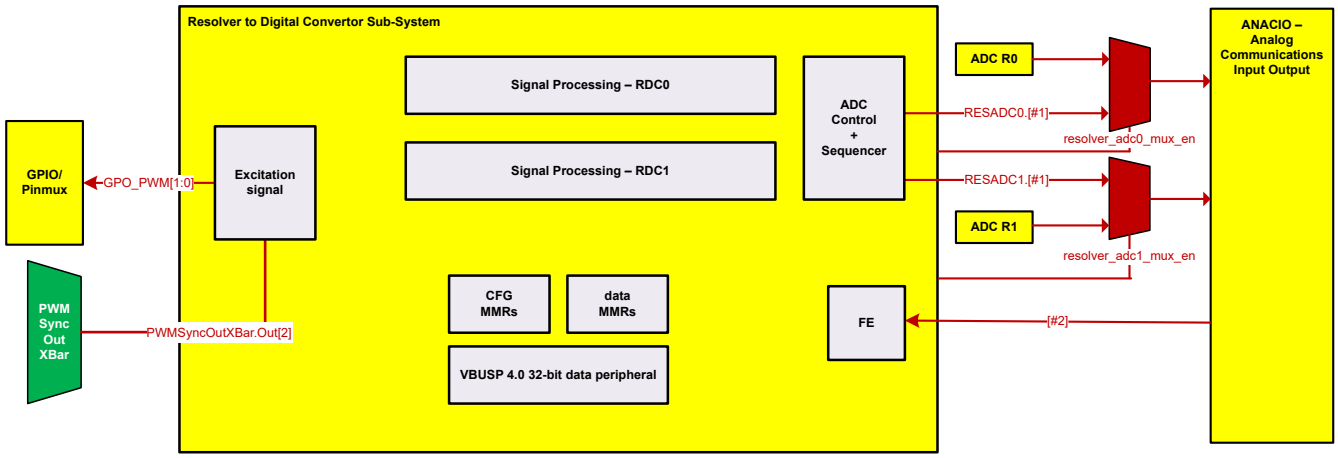


Figure 4-2. RDC Integration Diagram

4.3 Resolver Integration

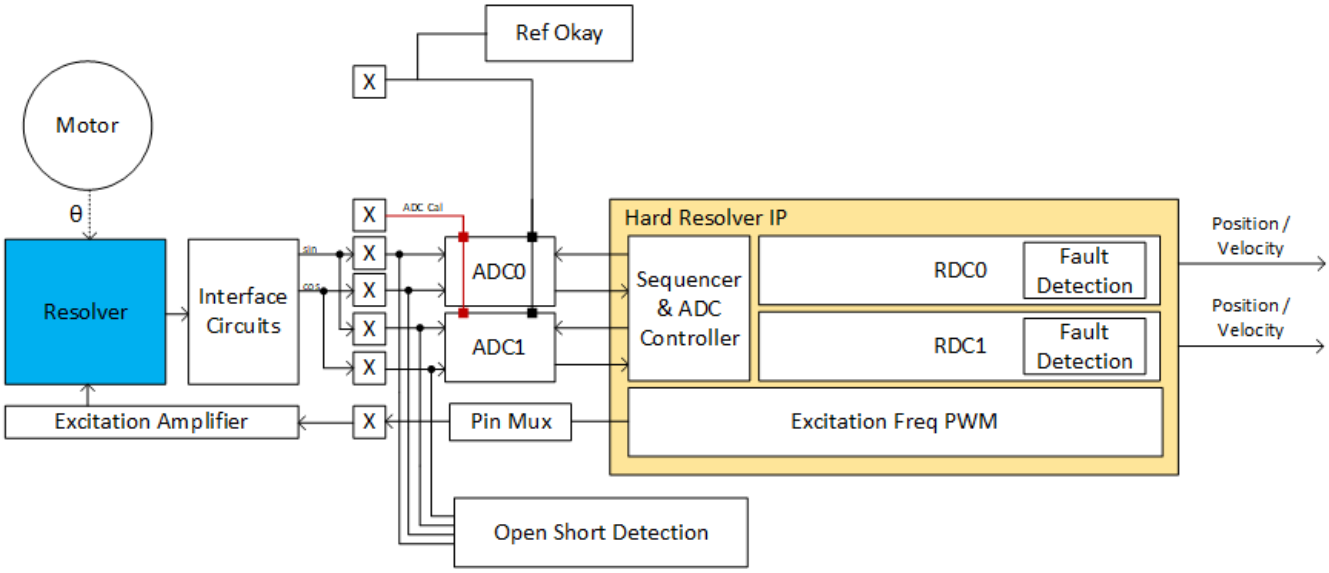
There are 1x Resolver with up to 2x motor position sensing modules integrated in the device. Motor position sensing utilizes two dedicated SAR ADC's.

Note

For each Resolver

- Resolver ADC's support 12 bit/14 bit
- Can be used for General Purpose if required

Figure 4-3. Resolver Integration Diagram



4.4 DAC Integration

There is 1x DAC module integrated in the device. The diagram below provides a visual representation of the device integration details.

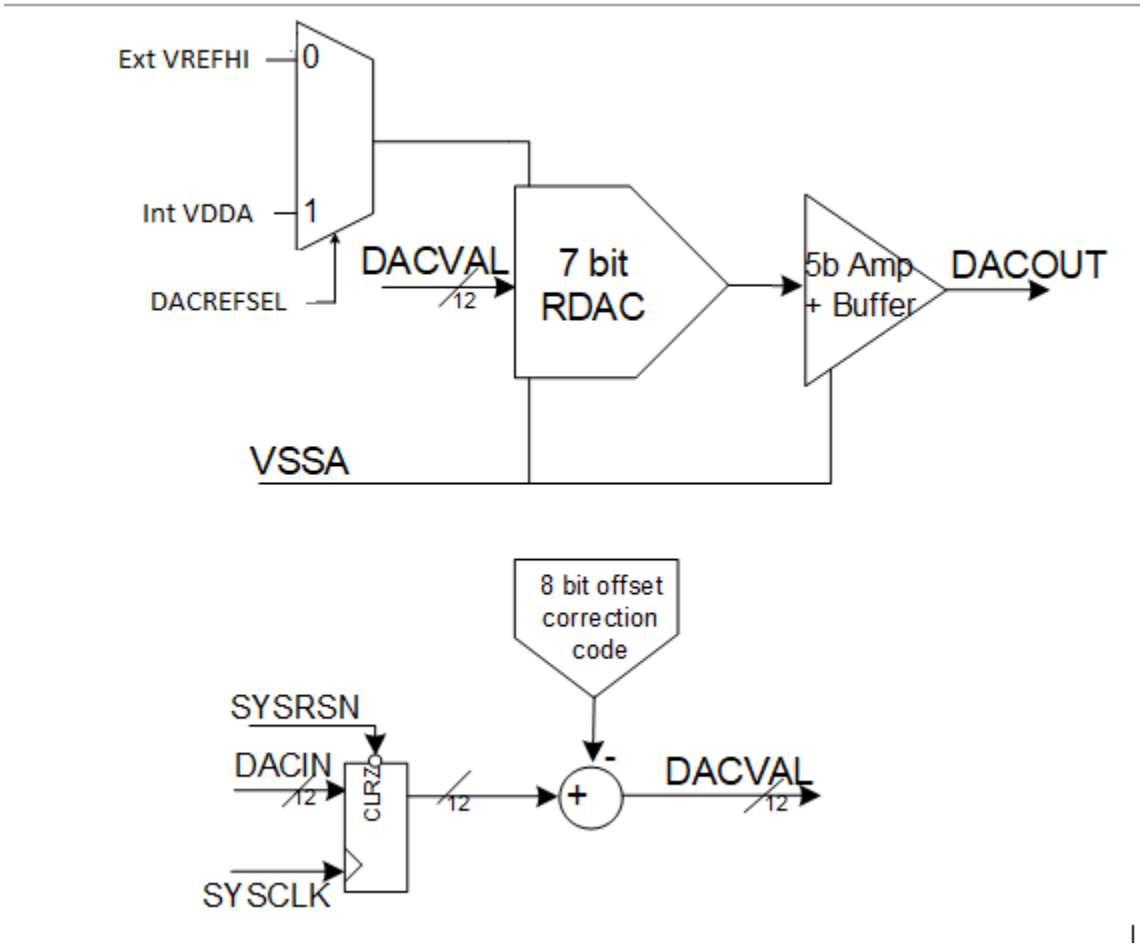


Figure 4-4. DAC Integration Diagram

4.5 ECAP Integration

There are 16x ECAP modules integrated in the device. The diagram below provides a visual representation of the device integration details.

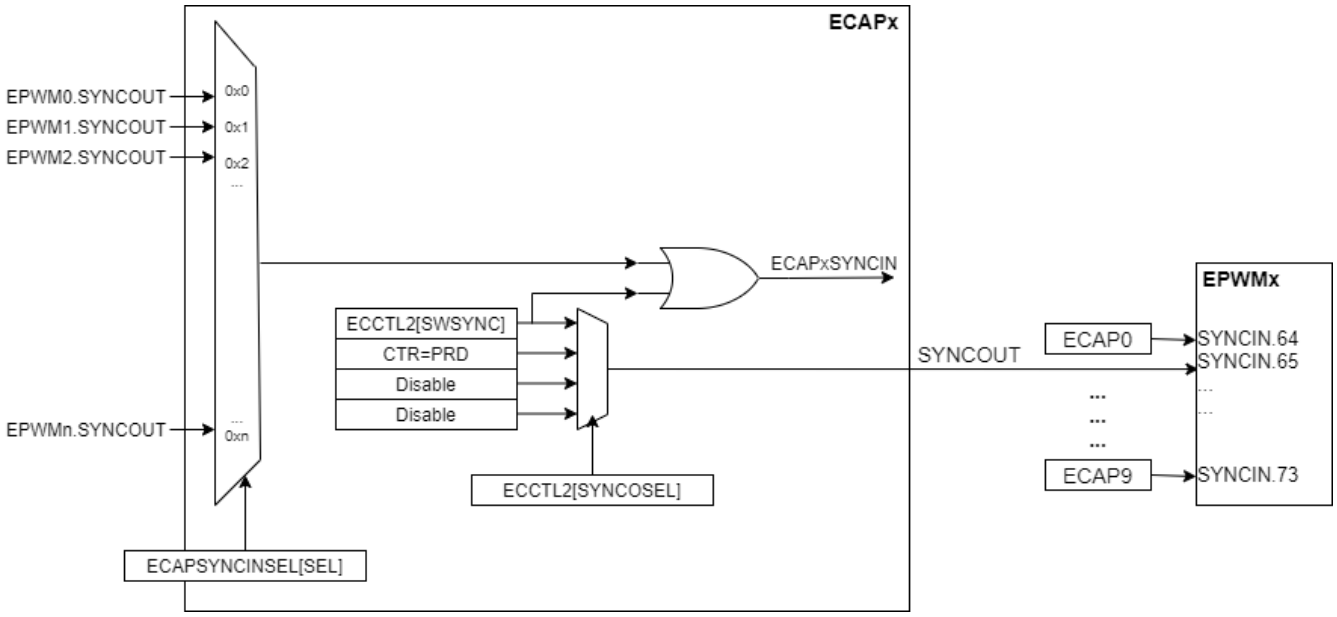


Figure 4-5. ECAP Integration Diagram

4.6 EPWM Integration

There are 32x EPWM modules integrated in the device. The diagram below provides a visual representation of the device integration details.

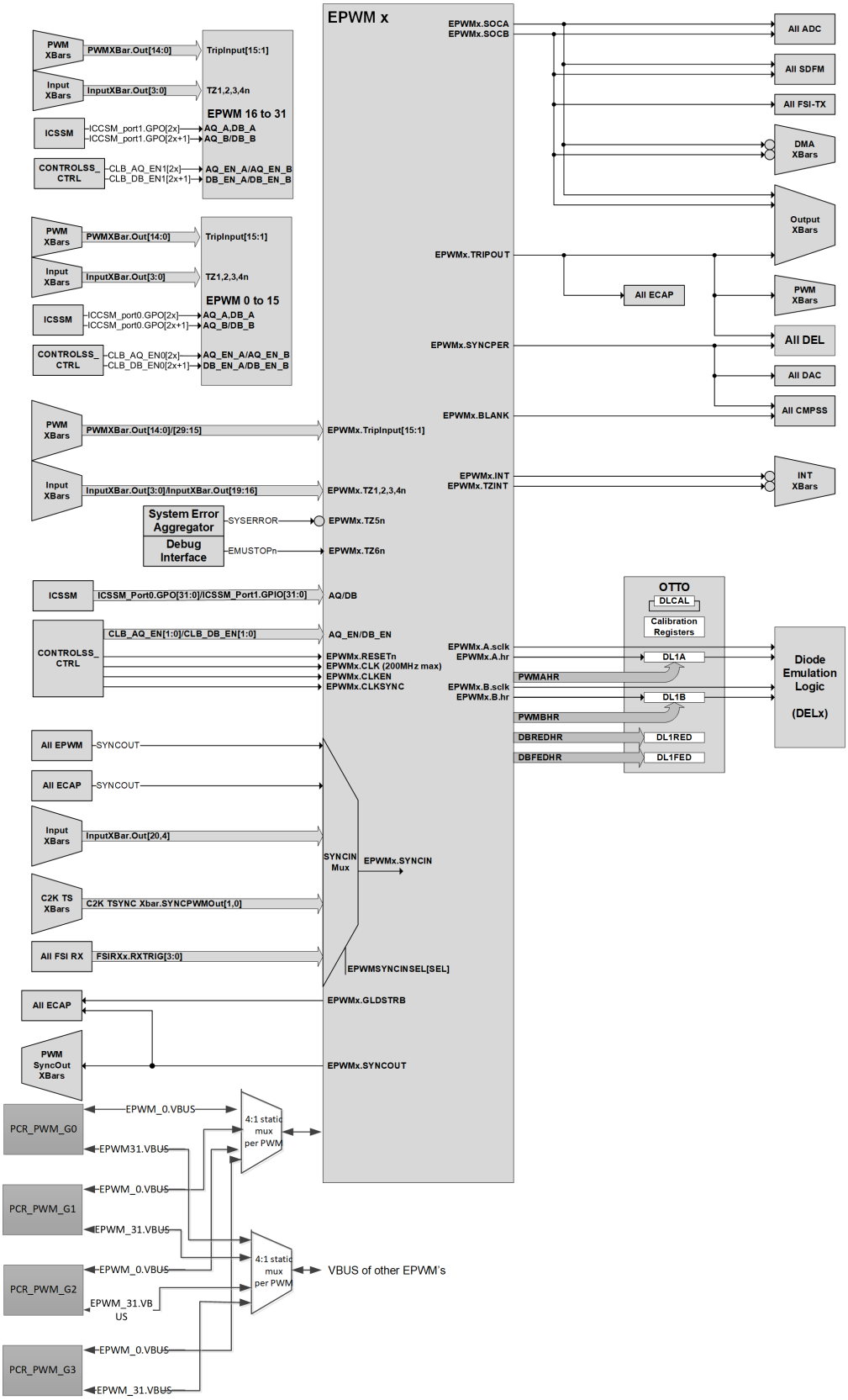


Figure 4-6. EPWM Integration Diagram

4.7 EQEP Integration

There are 3x EQEP modules integrated in the device. The diagram below provides a visual representation of the device integration details.

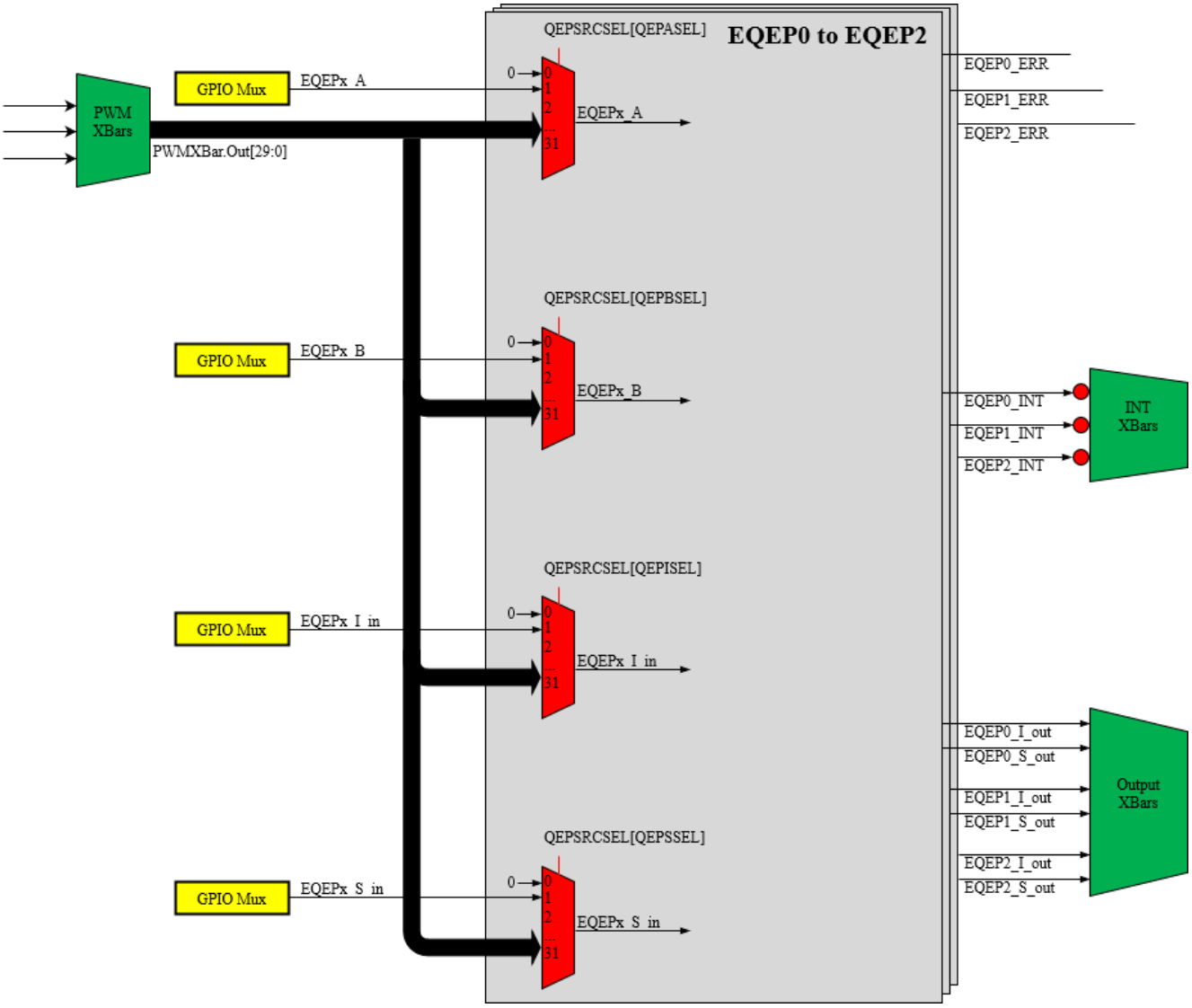


Figure 4-7. EQEP Integration Diagram

4.8 FSI Integration

There are 4x FSI modules integrated in the CONTROLSS. The diagram below provides a visual representation of the device integration details.

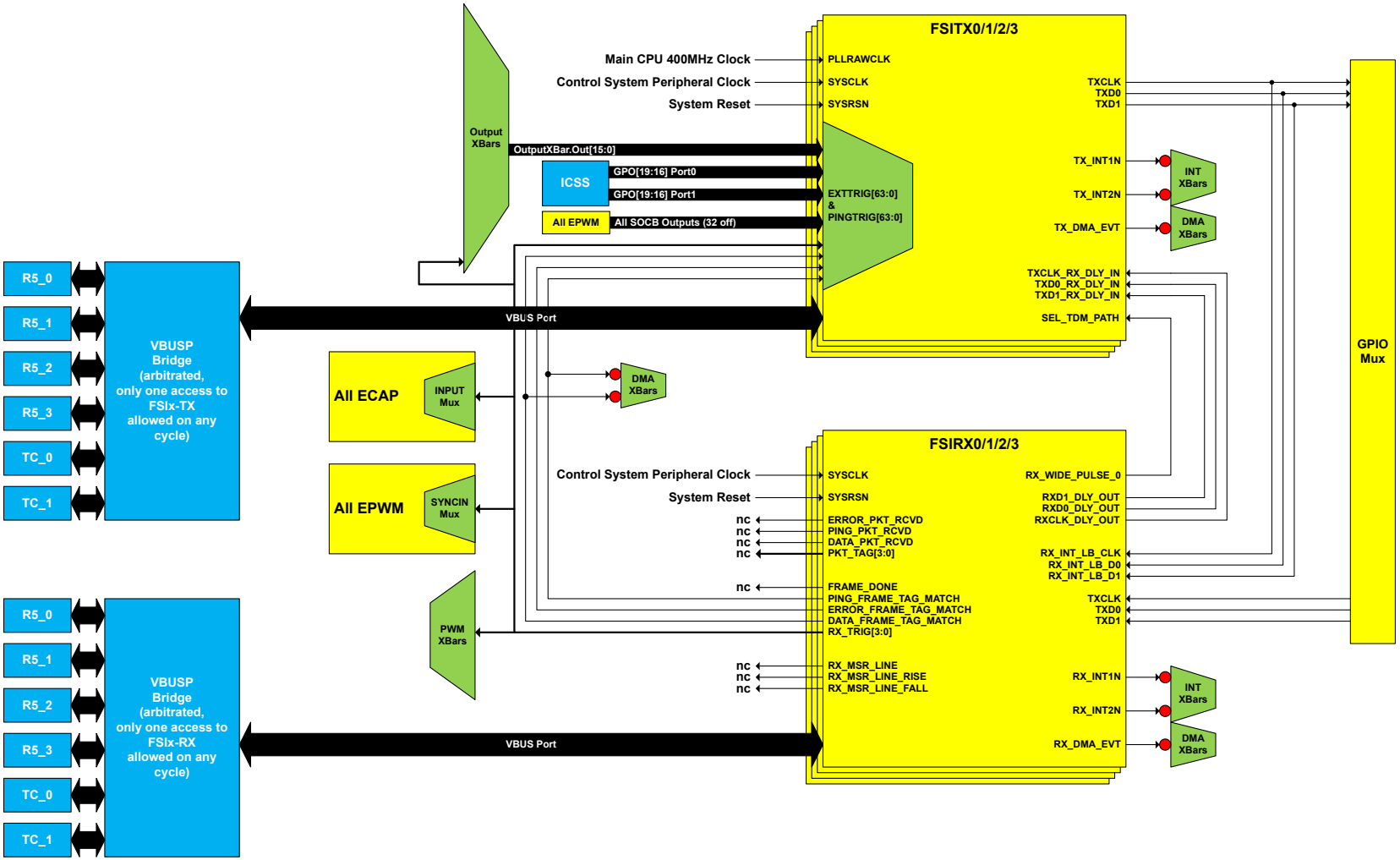


Figure 4-8. FSI Integration Diagram

4.9 SDFM Integration

There are 2x SDFM modules integrated in the CONTROLSS. The diagrams and table below provides a visual representation of the device integration details.

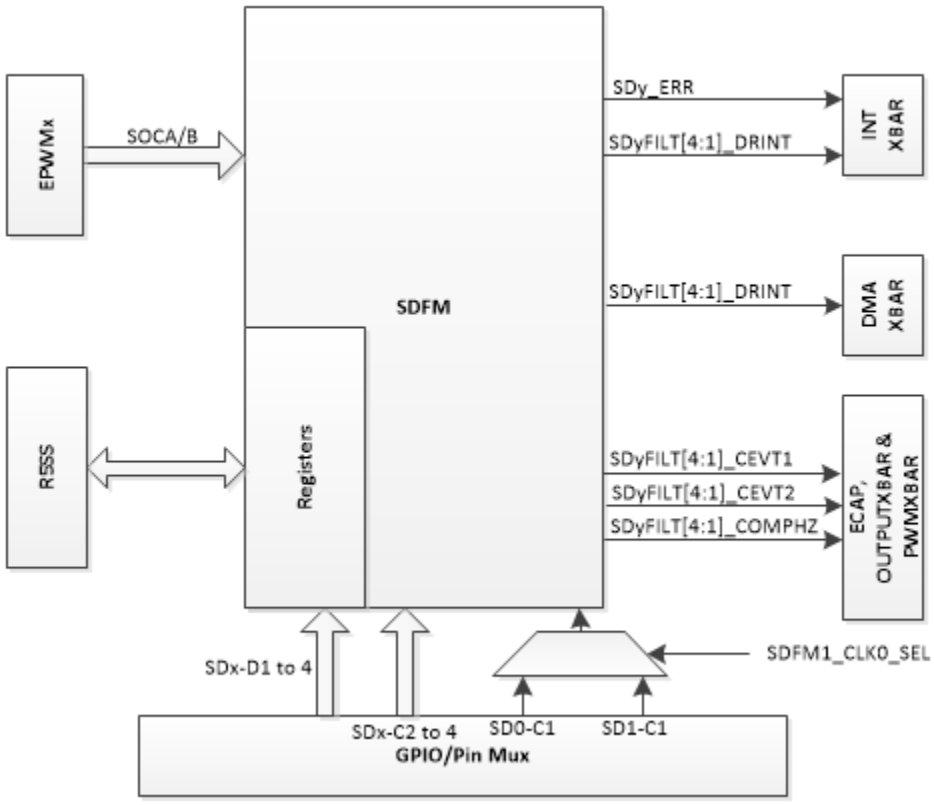


Figure 4-9. SDFM Integration Diagram (Simple)

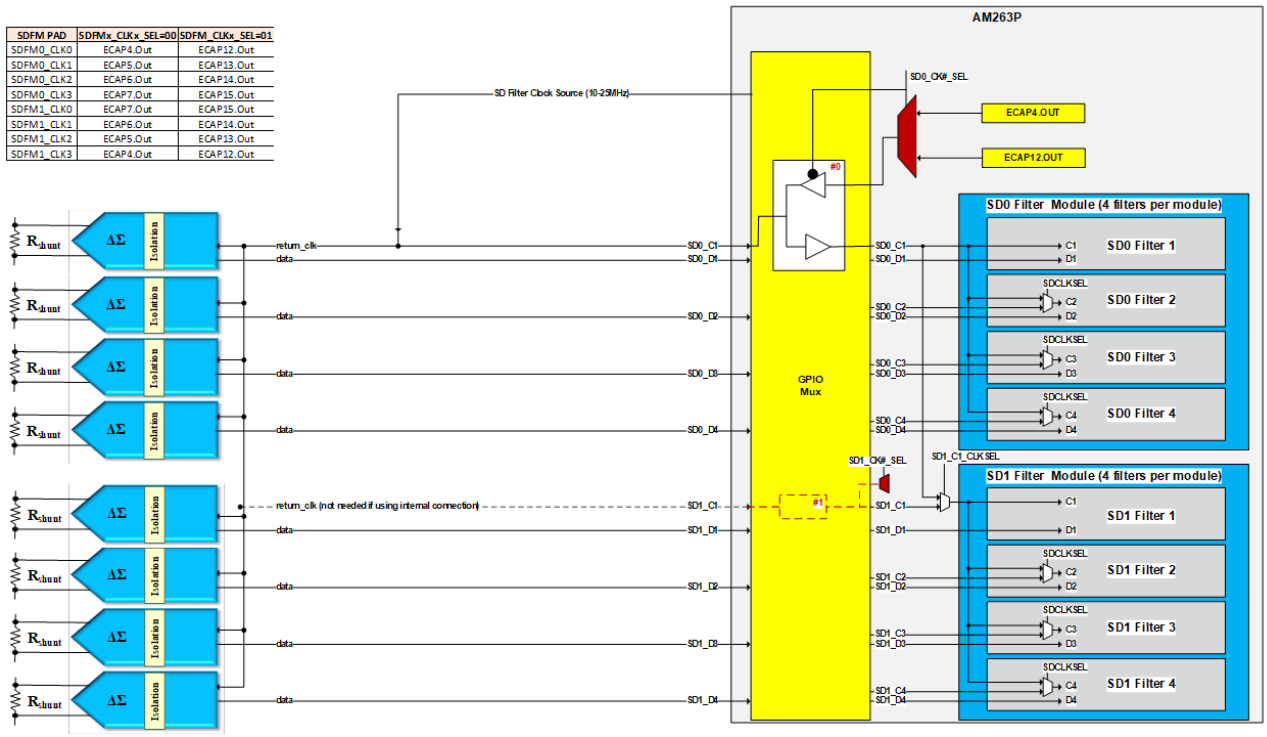


Figure 4-10. SDFM Integration Diagram (Detailed)

Table 4-1. SDFM Integration Diagram Detailed

SDFM PAD	SDFMx_CLKx_SEL = 00	SDFM_CLKx_SEL = 01
SDFM0_CLK0	ECAP4.Out	ECAP12.Out
SDFM0_CLK1	ECAP5.Out	ECAP13.Out
SDFM0_CLK2	ECAP6.Out	ECAP14.Out
SDFM0_CLK3	ECAP7.Out	ECAP15.Out
SDFM1_CLK0	ECAP7.Out	ECAP15.Out
SDFM1_CLK1	ECAP6.Out	ECAP14.Out
SDFM1_CLK2	ECAP5.Out	ECAP13.Out
SDFM1_CLK3	ECAP4.Out	ECAP12.Out

4.10 SOC_TIMESYNC_XBAR0 Integration

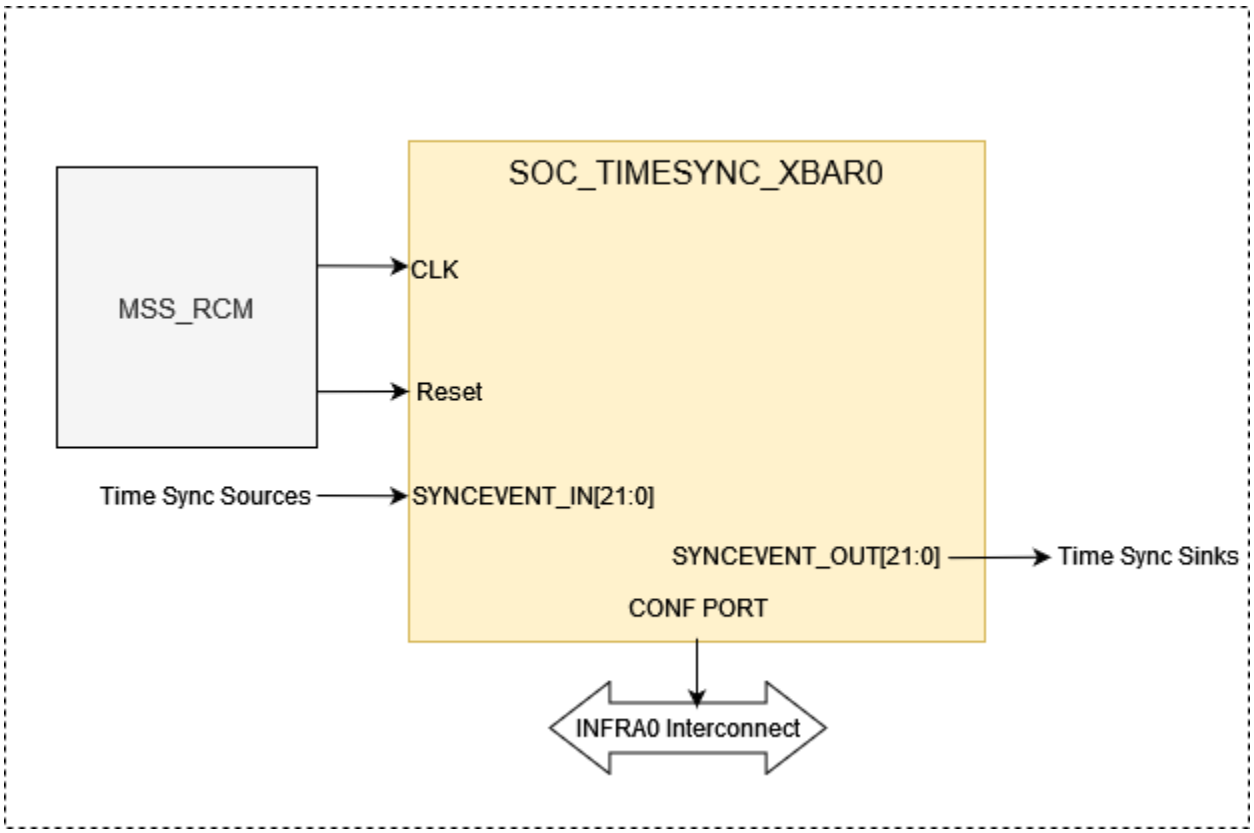


Figure 4-11. SOC_TIMESYNC_XBAR0 Integration

Table 4-2. SOC_TIMESYNC_XBAR0 Device Integration

Module Instance	Device Allocation	SoC Interconnect
SOC_TIMESYNC_XBAR0	✓	VBUSP INFRA0 Interconnect

Table 4-3. SOC_TIMESYNC_XBAR0 Clocks

Module Instance	Module Clock Input	Source Clock Signal	Source	Default Freq	Description
SOC_TIMESYNC_XBAR0	CLK	SYSCLK	MSS_RCM	200 MHz	SOC_TIMESYNC_XBAR0 Functional and Interface clock

Table 4-4. SOC_TIMESYNC_XBAR0 Resets

Module Instance	Module Reset Input	Source Reset Signal	Source	Description
SOC_TIMESYNC_XBAR0	RST	SYS_RST	RCM + Warm Reset Sources	SOC_TIMESYNC_XBAR0 Reset

Table 4-5. SOC_TIMESYNC_XBAR0 Time Sync Output Events

Module Instance	Module Sync Output	Destination Sync Signal	Destination	Type	Description
SOC_TIMESYNC_XBAR0	SYNCEVENT_OUT0	EPWMx_SYNCIN58	EPWMx	Edge	Selectable sync event 0
	SYNCEVENT_OUT1	EPWMx_SYNCIN59	EPWMx		Selectable sync event 1
	SYNCEVENT_OUT2	CAPEVT0	RTI0,WDT0		Selectable sync event 2
	SYNCEVENT_OUT3	CAPEVT1	RTI0,WDT0		Selectable sync event 3
	SYNCEVENT_OUT4	CAPEVT0	RTI1,WDT1		Selectable sync event 4
	SYNCEVENT_OUT5	CAPEVT1	RTI1,WDT1		Selectable sync event 5
	SYNCEVENT_OUT6	CAPEVT0	RTI2,WDT2		Selectable sync event 6

Module Instance	Module Sync Input	TimeSync Event Sources
SOC_TIMESYNC_XBAR0	SYNCEVENT_IN[21:0]	See SOC_TIMESYNC_XBAR0 Event Map table for time sync event mapping.

4.11 SOC_TIMESYNC_XBAR1 Integration

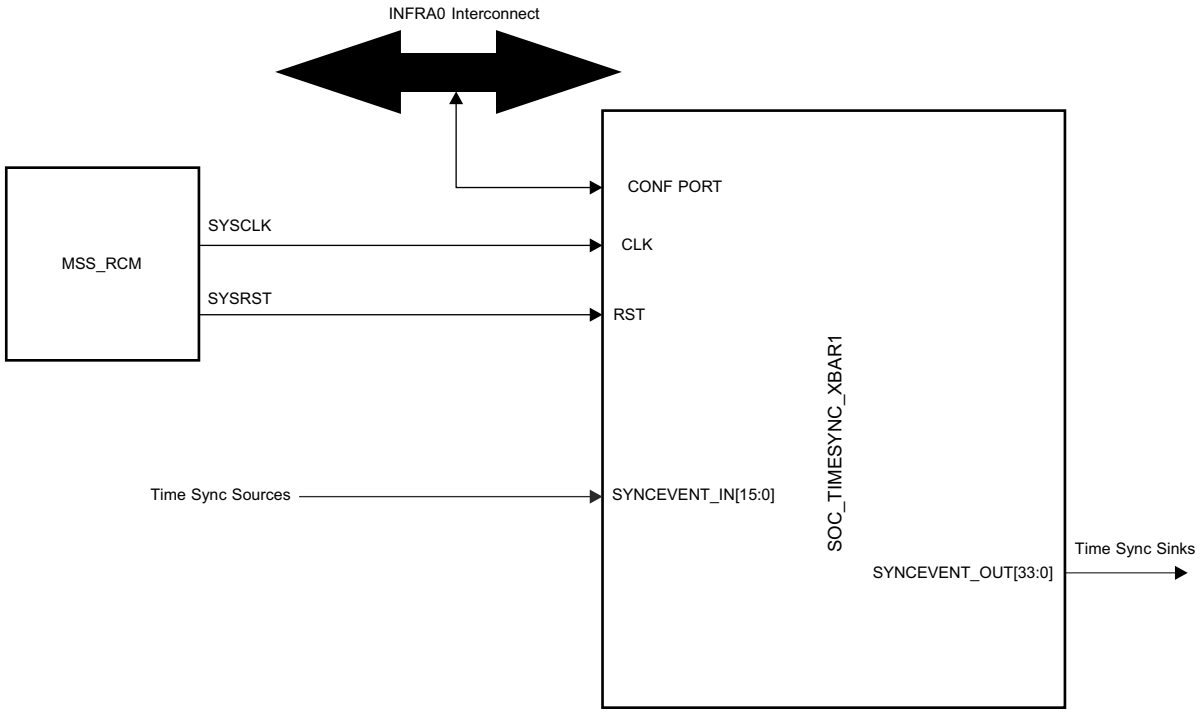


Figure 4-12. SOC_TIMESYNC_XBAR1 Integration

Table 4-6. SOC_TIMESYNC_XBAR1 Device Integration

Module Instance	Device Allocation	SoC Interconnect
SOC_TIMESYNC_XBAR1	✓	VBUSP INFRA Interconnect

Table 4-7. SOC_TIMESYNC_XBAR1 Clocks

Module Instance	Module Clock Input	Source Clock Signal	Source	Default Freq	Description
SOC_TIMESYNC_XBAR1	CLK	SYSCLK	MSS_RCM	200 MHz	SOC_TIMESYNC_XBAR1 Functional and Interface clock

Table 4-8. SOC_TIMESYNC_XBAR1 Resets

Module Instance	Module Reset Input	Source Reset Signal	Source	Description
SOC_TIMESYNC_XBAR1	RST	SYS_RST	RCM + Warm Reset Sources	SOC_TIMESYNC_XBAR1 Reset

Table 4-9. SOC_TIMESYNC_XBAR1 Time Sync Output Events

Module Instance	Module Sync Output	Destination Sync Signal	Destination	Type	Description
SOC_TIMESYNC_XBAR1	SYNCEVENT_OUT0	EDMA_TRIGGERXBAR_IN111	EDMA_TRIGGERXBAR	Edge	Selectablesync event 0
	SYNCEVENT_OUT1	EDMA_TRIGGERXBAR_IN112	EDMA_TRIGGERXBAR		Selectablesync event 1
	SYNCEVENT_OUT2	R5SS0_CORE0_INTR138	R5SS0_CORE0_VIM		Selectablesync event 2
	SYNCEVENT_OUT3	R5SS0_CORE0_INTR139	R5SS0_CORE0_VIM		Selectablesync event 3
	SYNCEVENT_OUT4	R5SS0_CORE0_INTR140	R5SS0_CORE0_VIM		Selectablesync event 4
	SYNCEVENT_OUT5	R5SS0_CORE0_INTR141	R5SS0_CORE0_VIM		Selectablesync event 5
	SYNCEVENT_OUT6	R5SS0_CORE1_INTR138	R5SS0_CORE1_VIM		Selectablesync event 6
	SYNCEVENT_OUT7	R5SS0_CORE1_INTR139	R5SS0_CORE1_VIM		Selectablesync event 7
	SYNCEVENT_OUT8	R5SS0_CORE1_INTR140	R5SS0_CORE1_VIM		Selectablesync event 8
	SYNCEVENT_OUT9	R5SS0_CORE1_INTR141	R5SS0_CORE1_VIM		Selectablesync event 9
	SYNCEVENT_OUT10	ICSSM0_EDC_LATCH0_IN	PRU_ICSSM0		Selectablesync event 10
	SYNCEVENT_OUT11	ICSSM0_EDC_LATCH1_IN	PRU_ICSSM0		Selectablesync event 11
	SYNCEVENT_OUT12	ICSSM0_IEP_CAP_INT R0	PRU_ICSSM0		Selectablesync event 12
	SYNCEVENT_OUT13	ICSSM0_IEP_CAP_INT R1	PRU_ICSSM0		Selectablesync event 13
	SYNCEVENT_OUT14	ICSSM0_IEP_CAP_INT R2	PRU_ICSSM0		Selectablesync event 14
	SYNCEVENT_OUT15	ICSSM0_IEP_CAP_INT R3	PRU_ICSSM0		Selectablesync event 15
	SYNCEVENT_OUT16	ICSSM0_IEP_CAP_INT R4	PRU_ICSSM0		Selectablesync event 16

Table 4-9. SOC_TIMESYNC_XBAR1 Time Sync Output Events (continued)

Module Instance	Module Sync Output	Destination Sync Signal	Destination	Type	Description
SOC_TIMESYNC_XBAR1	SYNCEVENT_OUT17	ICSSM0_IEP_CAP_INT R5	PRU_ICSSM0	Edge	Selectablesync event 17
	SYNCEVENT_OUT18	CPTS_HW1_TS_PUSH	CPSW0_CPTS		Selectablesync event 18
	SYNCEVENT_OUT19	CPTS_HW2_TS_PUSH	CPSW0_CPTS		Selectablesync event 19
	SYNCEVENT_OUT20	CPTS_HW3_TS_PUSH	CPSW0_CPTS		Selectablesync event 20
	SYNCEVENT_OUT21	CPTS_HW4_TS_PUSH	CPSW0_CPTS		Selectablesync event 21
	SYNCEVENT_OUT22	CPTS_HW5_TS_PUSH	CPSW0_CPTS		Selectablesync event 22
	SYNCEVENT_OUT23	CPTS_HW6_TS_PUSH	CPSW0_CPTS		Selectablesync event 23
	SYNCEVENT_OUT24	CPTS_HW7_TS_PUSH	CPSW0_CPTS		Selectablesync event 24
	SYNCEVENT_OUT25	CPTS_HW8_TS_PUSH	CPSW0_CPTS		Selectablesync event 25
	SYNCEVENT_OUT26	R5SS1_CORE0_INTR138	R5SS1_CORE0_VIM		Selectablesync event 26
	SYNCEVENT_OUT27	R5SS1_CORE0_INTR139	R5SS1_CORE0_VIM		Selectablesync event 27
	SYNCEVENT_OUT28	R5SS1_CORE0_INTR140	R5SS1_CORE0_VIM		Selectablesync event 28
	SYNCEVENT_OUT29	R5SS1_CORE0_INTR141	R5SS1_CORE0_VIM		Selectablesync event 29
	SYNCEVENT_OUT30	R5SS1_CORE1_INTR138	R5SS1_CORE1_VIM		Selectablesync event 30
	SYNCEVENT_OUT31	R5SS1_CORE1_INTR139	R5SS1_CORE1_VIM		Selectablesync event 31
	SYNCEVENT_OUT32	R5SS1_CORE1_INTR140	R5SS1_CORE1_VIM		Selectablesync event 32
	SYNCEVENT_OUT33	R5SS1_CORE1_INTR141	R5SS1_CORE1_VIM		Selectablesync event 33

Table 4-10. SOC_TIMESYNC_XBAR1 Time Sync Input Events

Module Instance	Module Sync Input	TimeSync Event Sources
SOC_TIMESYNC_XBAR1	SYNCEVENT_IN[15:0]	See SOC_TIMESYNC_XBAR1 Event Map table for time sync event mapping.

4.12 GPIO Integration

There are 4x GPIO modules integrated in the device. The diagram below provides a visual representation of the device integration details.

Note

There is a designated GPIO module per R5FSS core. Each R5FSS core has access to all GPI signals at all times. The GPO signals are assigned to a specific R5FSS core by configuring the MSS_IOMUX.*PAD*_CFG_REG.GPIO_SEL[17:16] of the associated IOMUX Pad Configuration register.

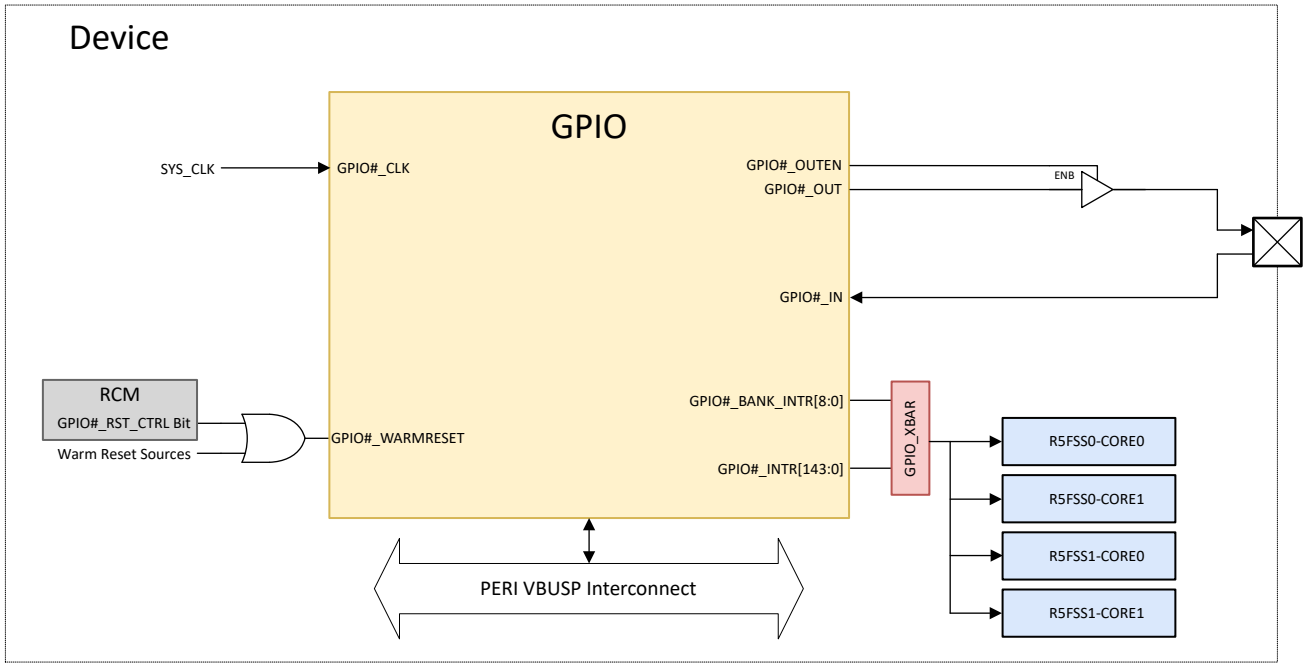


Figure 4-13. GPIO Integration Diagram

This diagram describes the GPIO multiplexor connectivity.

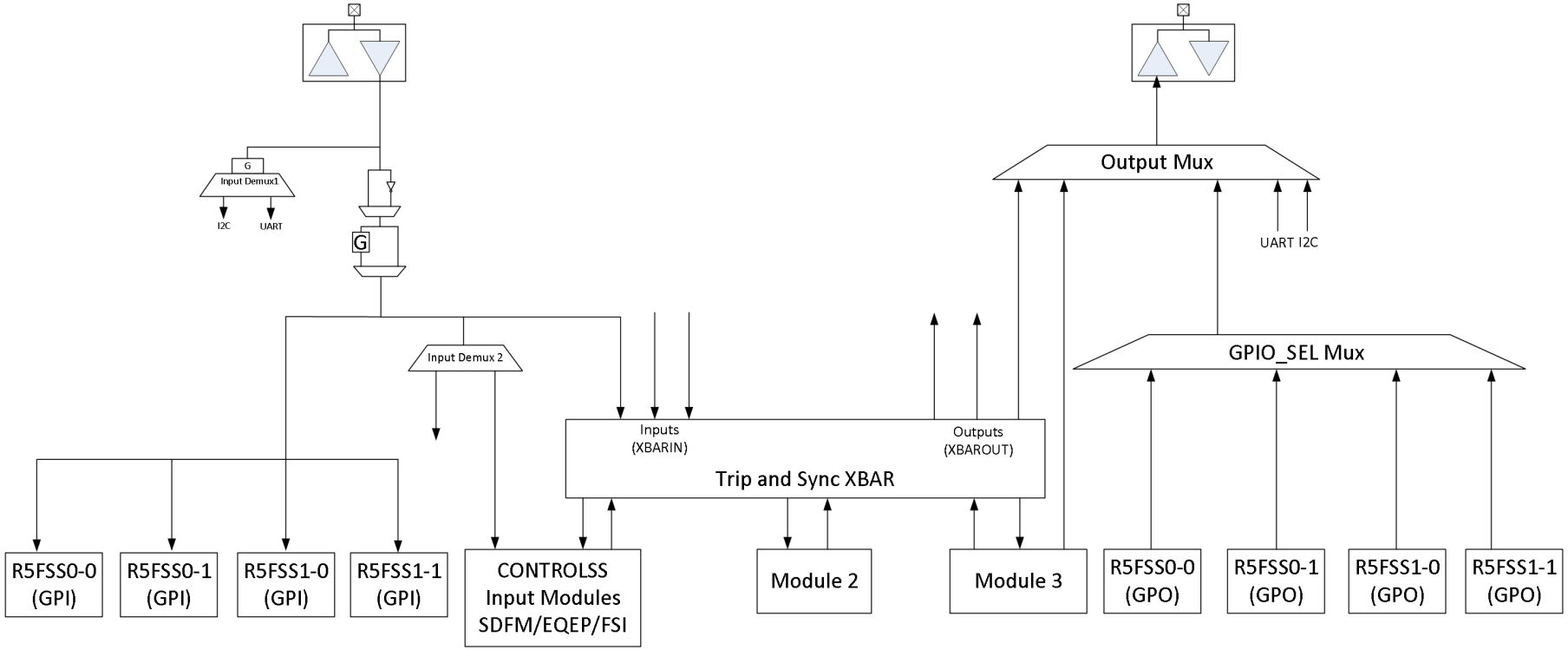


Figure 4-14. GPIO Mux Diagram

The tables below summarize the device integration details for the GPIO.

Table 4-11. *GPIO* Device Integration

This table describes the module device integration details.

Module Instance	Device Allocation	SoC Interconnect
GPIO0	✓	PERI VBUSP Interconnect
GPIO1	✓	PERI VBUSP Interconnect
GPIO2	✓	PERI VBUSP Interconnect
GPIO3	✓	PERI VBUSP Interconnect

Table 4-12. *GPIO* Clocks

This table describes the module clocking signals.

Module Instance	Module Clock Input	Source Clock Signal	Source	Default Freq	Description
GPIO0	GPIO0_VBUS_FICLK	SYS_CLK	PLL_CORE_CLK:HSDIV0_CLKOUT 0	200 MHz	GPIO0 Functional and Interface Clock
GPIO1	GPIO1_VBUS_FICLK	SYS_CLK	PLL_CORE_CLK:HSDIV0_CLKOUT 0	200 MHz	GPIO1 Functional and Interface Clock
GPIO2	GPIO2_VBUS_FICLK	SYS_CLK	PLL_CORE_CLK:HSDIV0_CLKOUT 0	200 MHz	GPIO2 Functional and Interface Clock
GPIO3	GPIO3_VBUS_FICLK	SYS_CLK	PLL_CORE_CLK:HSDIV0_CLKOUT 0	200 MHz	GPIO3 Functional and Interface Clock

Table 4-13. *GPIO* Resets

This table describes the module reset signals.

Module Instance	Module Reset Input	Source Reset Signal	Source	Description
GPIO0	GPIO0_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	GPIO0 Reset
GPIO1	GPIO1_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	GPIO1 Reset
GPIO2	GPIO2_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	GPIO2 Reset
GPIO3	GPIO3_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	GPIO3 Reset

Table 4-14. *GPIO* Interrupt Requests

This table describes the module interrupt requests.

Note

Where GPIO# = GPIO[0:3]

Module Instance	Module Interrupt Signal	Destination Interrupt Input	Destination	Type	Description
GPIO#	GPIO#_[0:137]	Programmable via GPIO_XBAR_INTR0	GPIO_XBAR_INTR0	Pulse	GPIO#_[0:137] interrupt request

Table 4-14. *GPIO* Interrupt Requests (continued)

This table describes the module interrupt requests.

Note

Where GPIO# = GPIO[0:3]

Module Instance	Module Interrupt Signal	Destination Interrupt Input	Destination	Type	Description
GPIO#	GPIO#_BANK0_INT	Programmable via GPIO_XBAR_INTR0	GPIO_XBAR_INTR0	Pulse	GPIO# BANK0 interrupt request
GPIO#	GPIO#_BANK1_INT	Programmable via GPIO_XBAR_INTR0	GPIO_XBAR_INTR0	Pulse	GPIO# BANK1 interrupt request
GPIO#	GPIO#_BANK2_INT	Programmable via GPIO_XBAR_INTR0	GPIO_XBAR_INTR0	Pulse	GPIO# BANK2 interrupt request
GPIO#	GPIO#_BANK3_INT	Programmable via GPIO_XBAR_INTR0	GPIO_XBAR_INTR0	Pulse	GPIO# BANK3 interrupt request
GPIO#	GPIO#_BANK4_INT	Programmable via GPIO_XBAR_INTR0	GPIO_XBAR_INTR0	Pulse	GPIO# BANK4 interrupt request
GPIO#	GPIO#_BANK5_INT	Programmable via GPIO_XBAR_INTR0	GPIO_XBAR_INTR0	Pulse	GPIO# BANK5 interrupt request
GPIO#	GPIO#_BANK6_INT	Programmable via GPIO_XBAR_INTR0	GPIO_XBAR_INTR0	Pulse	GPIO# BANK6 interrupt request
GPIO#	GPIO#_BANK7_INT	Programmable via GPIO_XBAR_INTR0	GPIO_XBAR_INTR0	Pulse	GPIO# BANK7 interrupt request
GPIO#	GPIO#_BANK8_INT	Programmable via GPIO_XBAR_INTR0	GPIO_XBAR_INTR0	Pulse	GPIO# BANK8 interrupt request

Note

For more information on the interconnects, see the *System Interconnect* chapter.

For more information on power, reset, and clock management, see the corresponding sections within the *Device Configuration* chapter.

For more information on the device interrupt controllers, see the *Interrupt Controllers* chapter.

4.13 I2C Integration

There are 4x I2C module integrated in the device. The diagram below provides a visual representation of the device integration details.

Note

Only the I2C0 instance is a true I2C Open Drain buffer. I2C[1-3] are implemented with the typical LVCMOS voltage buffer and should be properly configured to operate as an Input/Output Open Drain signal type.

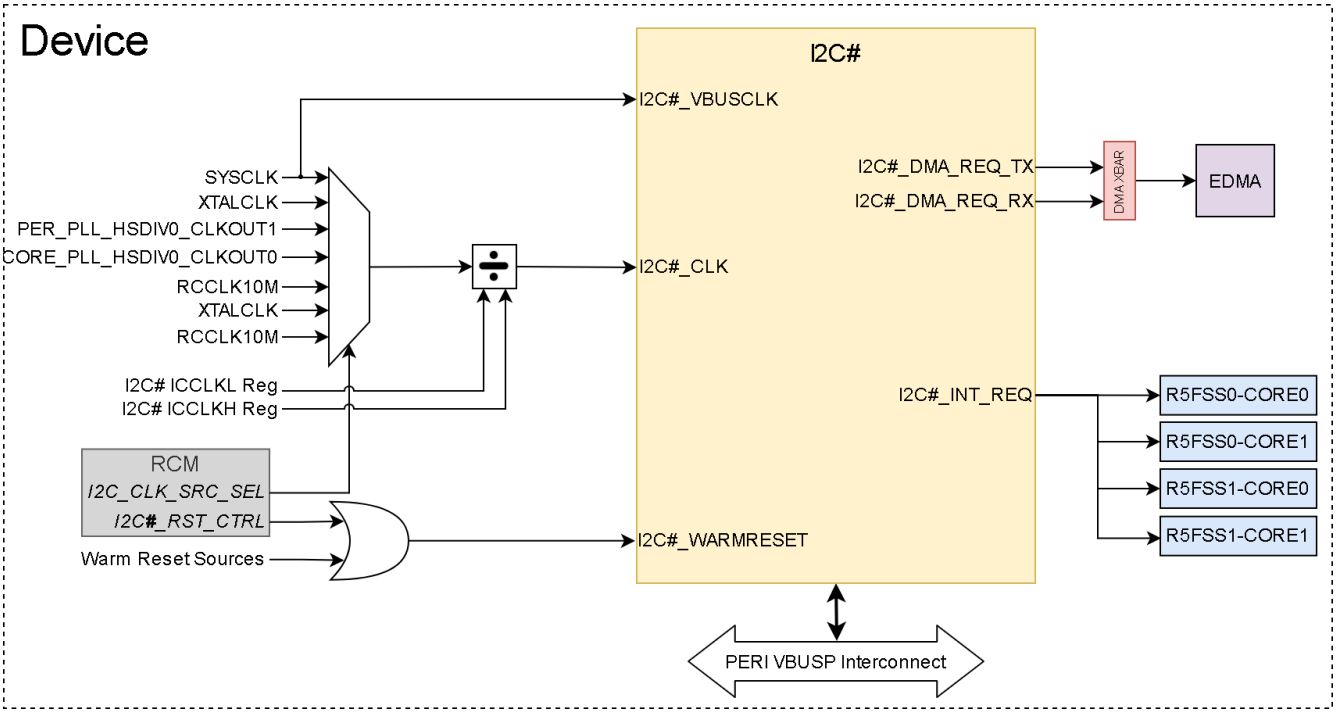


Figure 4-15. I2C Integration

The tables below summarize the device integration details of I2C# (where # = 0, 1, 2, 3).

Table 4-15. *I2C* Device Integration

This table describes the module device integration details.

Module Instance	Device Allocation	SoC Interconnect
I2C0	✓	PERI VBUSP Interconnect
I2C1	✓	PERI VBUSP Interconnect
I2C2	✓	PERI VBUSP Interconnect
I2C3	✓	PERI VBUSP Interconnect

Table 4-16. *I2C* Clocks

This table describes the module clocking signals.

Module Instance	Module Clock Input	Source Clock Signal	Source	Default Freq	Description
I2C[0:3]	I2C[0:3]_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	I2C[0:3] VBUS Clock
	I2C[0:3]_FCLK (I2C_CLK)	XTALCLK	External XTAL	25 MHz	I2C[0:3] Interface Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK: HSDIV0_CLKOUT0	500 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	

Table 4-17. *I2C* Resets

This table describes the module reset signals.

Module Instance	Module Reset Input	Source Reset Signal	Source	Description
I2C0	I2C0_RST(VBUSP_RSTn)	Warm Reset (SYS_NPRST)	RCM + Warm Reset Sources	I2C0 Asynchronous Reset
I2C1	I2C1_RST	Warm Reset (SYS_NPRST)	RCM + Warm Reset Sources	I2C1 Asynchronous Reset
I2C2	I2C2_RST	Warm Reset (SYS_NPRST)	RCM + Warm Reset Sources	I2C2 Asynchronous Reset
I2C3	I2C3_RST	Warm Reset (SYS_NPRST)	RCM + Warm Reset Sources	I2C3 Asynchronous Reset

Table 4-18. *I2C* Interrupt Requests

This table describes the module interrupt requests.

Module Instance	Module Interrupt Signal	Destination Interrupt Input	Destination	Type	Description
I2C0	i2c0_int_req	i2c0_int_req	ALL R5FSS Cores ICSSM Core	Pulse	I2C0 Status Event Interrupt
I2C1	i2c1_int_req	i2c1_int_req	ALL R5FSS Cores ICSSM Core	Pulse	I2C1 Status Event Interrupt
I2C2	i2c2_int_req	i2c2_int_req	ALL R5FSS Cores ICSSM Core	Pulse	I2C2 Status Event Interrupt
I2C3	i2c3_int_req	i2c3_int_req	ALL R5FSS Cores ICSSM Core	Pulse	I2C3 Status Event Interrupt

Table 4-19. *I2C* DMA Requests

This table describes the module DMA requests.

Module Instance	Module DMA Event	Destination DMA Event Input	Destination	Type	Description
I2C0	I2C0_TX	i2c0_dma_req_tx	EDMA Crossbar (EDMA_XBAR)	Pulse	I2C0 DMA Transmit Request
	I2C0_RX	i2c0_dma_req_rx			I2C0 DMA Receive Request
I2C1	I2C1_TX	i2c1_dma_req_tx	EDMA Crossbar (EDMA_XBAR)	Pulse	I2C1 DMA Transmit Request
	I2C1_RX	i2c1_dma_req_rx			I2C1 DMA Receive Request
IC2	I2C2_TX	i2c2_dma_req_tx	EDMA Crossbar (EDMA_XBAR)	Pulse	I2C2 DMA Transmit Request
	I2C2_RX	i2c2_dma_req_rx			I2C2 DMA Receive Request
I2C3	I2C3_TX	i2c3_dma_req_tx	EDMA Crossbar (EDMA_XBAR)	Pulse	I2C3 DMA Transmit Request
	I2C3_RX	i2c3_dma_req_rx			I2C3 DMA Receive Request

Note

For more information on the interconnects, see the *System Interconnect* chapter.

For more information on power, reset, and clock management, see the corresponding sections within the *Device Configuration* chapter.

For more information on the device interrupt controllers, see the *Interrupt Controllers* chapter.

4.14 SPI Integration

There are 8x SPI modules integrated in the device. The diagram below provides a visual representation of the device integration details.

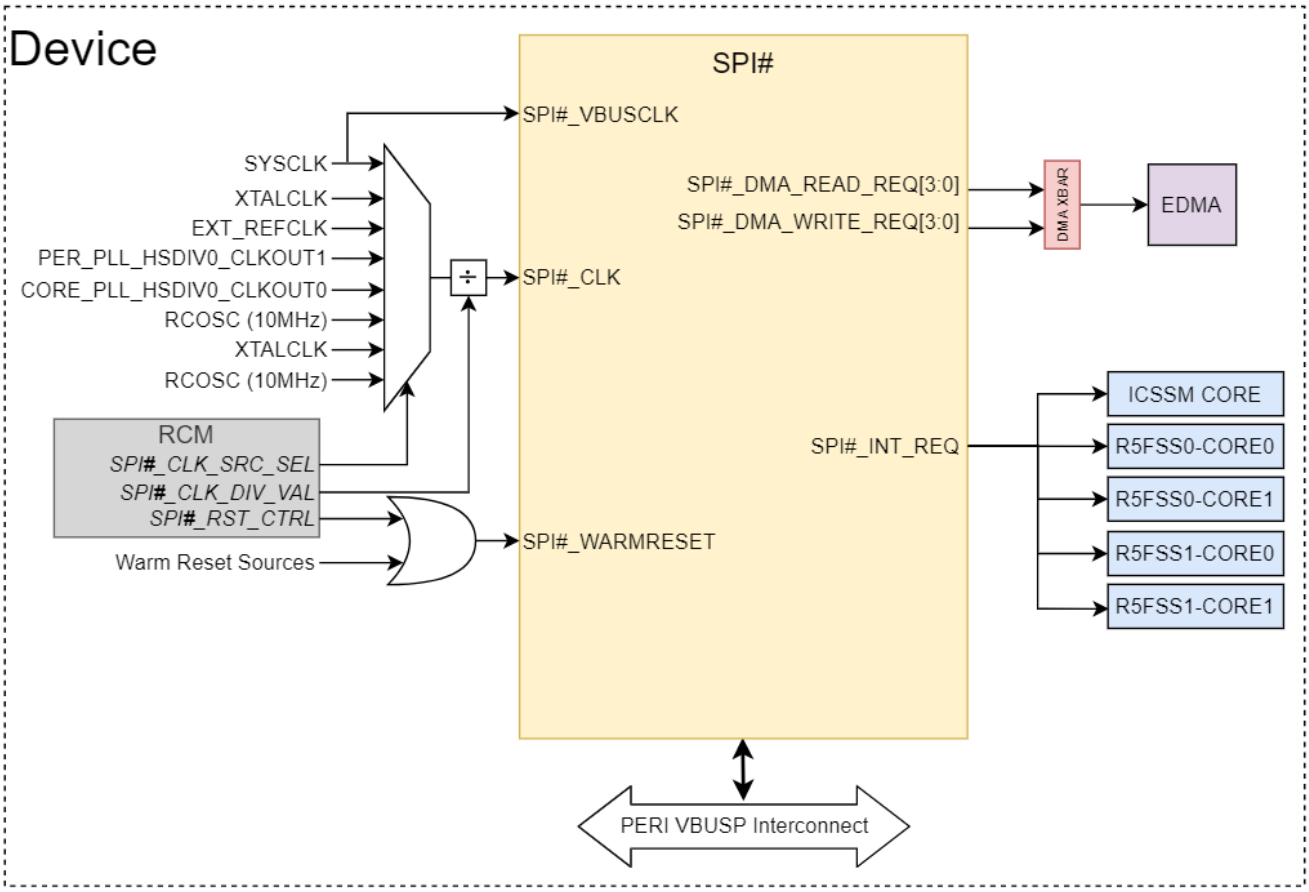


Figure 4-16. SPI Integration

The tables below summarize the device integration details of SPI# (where # = 0 to 7).

Table 4-20. *SPI* Device Integration

This table describes the module device integration details.

Module Instance	Device Allocation	SoC Interconnect
SPI0	✓	PERI VBUSP Interconnect
SPI1	✓	PERI VBUSP Interconnect
SPI2	✓	PERI VBUSP Interconnect
SPI3	✓	PERI VBUSP Interconnect
SPI4	✓	PERI VBUSP Interconnect
SPI5	✓	PERI VBUSP Interconnect
SPI6	✓	PERI VBUSP Interconnect
SPI7	✓	PERI VBUSP Interconnect

Table 4-21. *SPI* Clocks

This table describes the module clocking signals.

Module Instance	Module Clock Input	Source Clock Signal	Source	Default Freq	Description
SPI0	SPI0_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	SPI0 VBUS Clock
	SPI0_FCLK (SPI_CLK)	XTALCLK	External XTAL	25 MHz	SPI0 Interface Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK: HSDIV0_CLKOUT1	400 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	

Table 4-21. *SPI* Clocks (continued)

This table describes the module clocking signals.

Module Instance	Module Clock Input	Source Clock Signal	Source	Default Freq	Description
SPI1	SPI1_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	SPI1 VBUS Clock
	SPI1_FCLK (SPI_CLK)	XTALCLK	External XTAL	25 MHz	SPI1 Interface Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK: HSDIV0_CLKOUT1	400 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
SPI2	SPI2_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	SPI2 VBUS Clock
	SPI2_FCLK (SPI_CLK)	XTALCLK	External XTAL	25 MHz	SPI2 Interface Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK: HSDIV0_CLKOUT1	400 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	

Table 4-21. *SPI* Clocks (continued)

This table describes the module clocking signals.

Module Instance	Module Clock Input	Source Clock Signal	Source	Default Freq	Description
SPI3	SPI3_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	SPI3 VBUS Clock
	SPI3_FCLK (SPI_CLK)	XTALCLK	External XTAL	25 MHz	SPI3 Interface Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK: HSDIV0_CLKOUT1	400 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
SPI4	SPI4_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	SPI4 VBUS Clock
	SPI4_FCLK (SPI_CLK)	XTALCLK	External XTAL	25 MHz	SPI4 Interface Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK: HSDIV0_CLKOUT1	400 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	

Table 4-21. *SPI* Clocks (continued)

This table describes the module clocking signals.

Module Instance	Module Clock Input	Source Clock Signal	Source	Default Freq	Description
SPI5	SPI5_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	SPI5 VBUS Clock
	SPI5_FCLK (SPI_CLK)	XTALCLK	External XTAL	25 MHz	SPI5 Interface Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK: HSDIV0_CLKOUT1	400 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
SPI6	SPI6_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	SPI6 VBUS Clock
	SPI6_FCLK (SPI_CLK)	XTALCLK	External XTAL	25 MHz	SPI6 Interface Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK: HSDIV0_CLKOUT1	400 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	

Table 4-21. *SPI* Clocks (continued)

This table describes the module clocking signals.

Module Instance	Module Clock Input	Source Clock Signal	Source	Default Freq	Description
SPI7	SPI7_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	SPI7 VBUS Clock
	SPI7_FCLK (SPI_CLK)	XTALCLK	External XTAL	25 MHz	SPI7 Interface Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK: HSDIV0_CLKOUT1	400 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	

Table 4-22. *SPI* Resets

This table describes the module reset signals.

Module Instance	Module Reset Input	Source Reset Signal	Source	Description
SPI0	SPI0_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	SPI0 Asynchronous Reset
SPI1	SPI1_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	SPI1 Asynchronous Reset
SPI2	SPI2_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	SPI2 Asynchronous Reset
SPI3	SPI3_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	SPI3 Asynchronous Reset
SPI4	SPI4_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	SPI4 Asynchronous Reset
SPI5	SPI5_RST	Warm Reset (MOD_G_RST)	RCM+ Warm Reset Sources	SPI5 Asynchronous Reset
SPI6	SPI6_RST	Warm Reset (MOD_G_RST)	RCM+ Warm Reset Sources	SPI6 Asynchronous Reset
SPI7	SPI7_RST	Warm Reset (MOD_G_RST)	RCM+ Warm Reset Sources	SPI7 Asynchronous Reset

Table 4-23. *SPI* Interrupt Requests

This table describes the module interrupt requests.

Module Instance	Module Interrupt Signal	Destination Interrupt Input	Destination	Type	Description
SPI0	spi0_int_req	spi0_int_req	ALL R5FSS Cores PRU-ICSS Core	Level	SPI0 IP Status Information
SPI1	spi1_int_req	spi1_int_req	ALL R5FSS Cores PRU-ICSS Core	Level	SPI1 IP Status Information

Table 4-23. *SPI* Interrupt Requests (continued)

This table describes the module interrupt requests.

Module Instance	Module Interrupt Signal	Destination Interrupt Input	Destination	Type	Description
SPI2	spi2_int_req	spi2_int_req	ALL R5FSS Cores PRU-ICSS Core	Level	SPI2 IP Status Information
SPI3	spi3_int_req	spi3_int_req	ALL R5FSS Cores PRU-ICSS Core	Level	SPI3 IP Status Information
SPI4	spi4_int_req	spi4_int_req	ALL R5FSS Cores PRU-ICSS Core	Level	SPI4 IP Status Information
SPI5	spi5_int_req	spi5_int_req	ALL R5FSS Cores PRU-ICSS Core	Level	SPI5 IP Status Information
SPI6	spi6_int_req	spi6_int_req	ALL R5FSS Cores PRU-ICSS Core	Level	SPI6 IP Status Information
SPI7	spi7_int_req	spi7_int_req	ALL R5FSS Cores PRU-ICSS Core	Level	SPI7 IP Status Information

Table 4-24. *SPI* DMA Requests

This table describes the module DMA requests.

Module Instance	Module DMA Event	Destination DMA Event Input	Destination	Type	Description
SPI0	SPI0_DMA_READ_0	spi0_dma_read_req[0]	EDMA Crossbar (EDMA_XBAR)	Pulse	SPI0 DMA Read Request
	SPI0_DMA_READ_1	spi0_dma_read_req[1]			
	SPI0_DMA_READ_2	spi0_dma_read_req[2]			
	SPI0_DMA_READ_3	spi0_dma_read_req[3]			
	SPI0_DMA_WRITE_0	spi0_dma_write_req[0]			SPI0 DMA Write Request
	SPI0_DMA_WRITE_1	spi0_dma_write_req[1]			
	SPI0_DMA_WRITE_2	spi0_dma_write_req[2]			
	SPI0_DMA_WRITE_3	spi0_dma_write_req[3]			
SPI1	SPI1_DMA_READ_0	spi1_dma_read_req[0]	EDMA Crossbar (EDMA_XBAR)	Pulse	SPI1 DMA Read Request
	SPI1_DMA_READ_1	spi1_dma_read_req[1]			
	SPI1_DMA_READ_2	spi1_dma_read_req[2]			
	SPI1_DMA_READ_3	spi1_dma_read_req[3]			
	SPI1_DMA_WRITE_0	spi1_dma_write_req[0]			SPI1 DMA Write Request
	SPI1_DMA_WRITE_1	spi1_dma_write_req[1]			
	SPI1_DMA_WRITE_2	spi1_dma_write_req[2]			
	SPI1_DMA_WRITE_3	spi1_dma_write_req[3]			
SPI2	SPI2_DMA_READ_0	spi2_dma_read_req[0]	EDMA Crossbar (EDMA_XBAR)	Pulse	SPI2 DMA Read Request
	SPI2_DMA_READ_1	spi2_dma_read_req[1]			
	SPI2_DMA_READ_2	spi2_dma_read_req[2]			
	SPI2_DMA_READ_3	spi2_dma_read_req[3]			
	SPI2_DMA_WRITE_0	spi2_dma_write_req[0]			SPI2 DMA Write Request
	SPI2_DMA_WRITE_1	spi2_dma_write_req[1]			
	SPI2_DMA_WRITE_2	spi2_dma_write_req[2]			
	SPI2_DMA_WRITE_3	spi2_dma_write_req[3]			

Table 4-24. *SPI* DMA Requests (continued)

This table describes the module DMA requests.

Module Instance	Module DMA Event	Destination DMA Event Input	Destination	Type	Description
SPI3	SPI3_DMA_READ_0	spi3_dma_read_req[0]	EDMA Crossbar (EDMA_XBAR)	Pulse	SPI3 DMA Read Request
	SPI3_DMA_READ_1	spi3_dma_read_req[1]			
	SPI3_DMA_READ_2	spi3_dma_read_req[2]			
	SPI3_DMA_READ_3	spi3_dma_read_req[3]			
	SPI3_DMA_WRITE_0	spi3_dma_write_req[0]			SPI3 DMA Write Request
	SPI3_DMA_WRITE_1	spi3_dma_write_req[1]			
	SPI3_DMA_WRITE_2	spi3_dma_write_req[2]			
	SPI3_DMA_WRITE_3	spi3_dma_write_req[3]			
SPI4	SPI4_DMA_READ_0	spi4_dma_read_req[0]	EDMA Crossbar (EDMA_XBAR)	Pulse	SPI4 DMA Read Request
	SPI4_DMA_READ_1	spi4_dma_read_req[1]			
	SPI4_DMA_READ_2	spi4_dma_read_req[2]			
	SPI4_DMA_READ_3	spi4_dma_read_req[3]			
	SPI4_DMA_WRITE_0	spi4_dma_write_req[0]			SPI4 DMA Write Request
	SPI4_DMA_WRITE_1	spi4_dma_write_req[1]			
	SPI4_DMA_WRITE_2	spi4_dma_write_req[2]			
	SPI4_DMA_WRITE_3	spi4_dma_write_req[3]			
SPI5	SPI5_DMA_READ_0	spi5_dma_read_req[0]	EDMA Crossbar (EDMA_XBAR)	Pulse	SPI5 DMA Read Request
	SPI5_DMA_READ_1	spi5_dma_read_req[1]			
	SPI5_DMA_READ_2	spi5_dma_read_req[2]			
	SPI5_DMA_READ_3	spi5_dma_read_req[3]			
	SPI5_DMA_WRITE_0	spi5_dma_write_req[0]			SPI5 DMA Write Request
	SPI5_DMA_WRITE_1	spi5_dma_write_req[1]			
	SPI5_DMA_WRITE_2	spi5_dma_write_req[2]			
	SPI5_DMA_WRITE_3	spi5_dma_write_req[3]			
SPI6	SPI6_DMA_READ_0	spi6_dma_read_req[0]	EDMA Crossbar (EDMA_XBAR)	Pulse	SPI6 DMA Read Request
	SPI6_DMA_READ_1	spi6_dma_read_req[1]			
	SPI6_DMA_READ_2	spi6_dma_read_req[2]			
	SPI6_DMA_READ_3	spi6_dma_read_req[3]			
	SPI6_DMA_WRITE_0	spi6_dma_write_req[0]			SPI6 DMA Write Request
	SPI6_DMA_WRITE_1	spi6_dma_write_req[1]			
	SPI6_DMA_WRITE_2	spi6_dma_write_req[2]			
	SPI6_DMA_WRITE_3	spi6_dma_write_req[3]			
SPI7	SPI7_DMA_READ_0	spi7_dma_read_req[0]	EDMA Crossbar (EDMA_XBAR)	Pulse	SPI7 DMA Read Request
	SPI7_DMA_READ_1	spi7_dma_read_req[1]			
	SPI7_DMA_READ_2	spi7_dma_read_req[2]			
	SPI7_DMA_READ_3	spi7_dma_read_req[3]			
	SPI7_DMA_WRITE_0	spi7_dma_write_req[0]			SPI7 DMA Write Request
	SPI7_DMA_WRITE_1	spi7_dma_write_req[1]			
	SPI7_DMA_WRITE_2	spi7_dma_write_req[2]			
	SPI7_DMA_WRITE_3	spi7_dma_write_req[3]			

Note

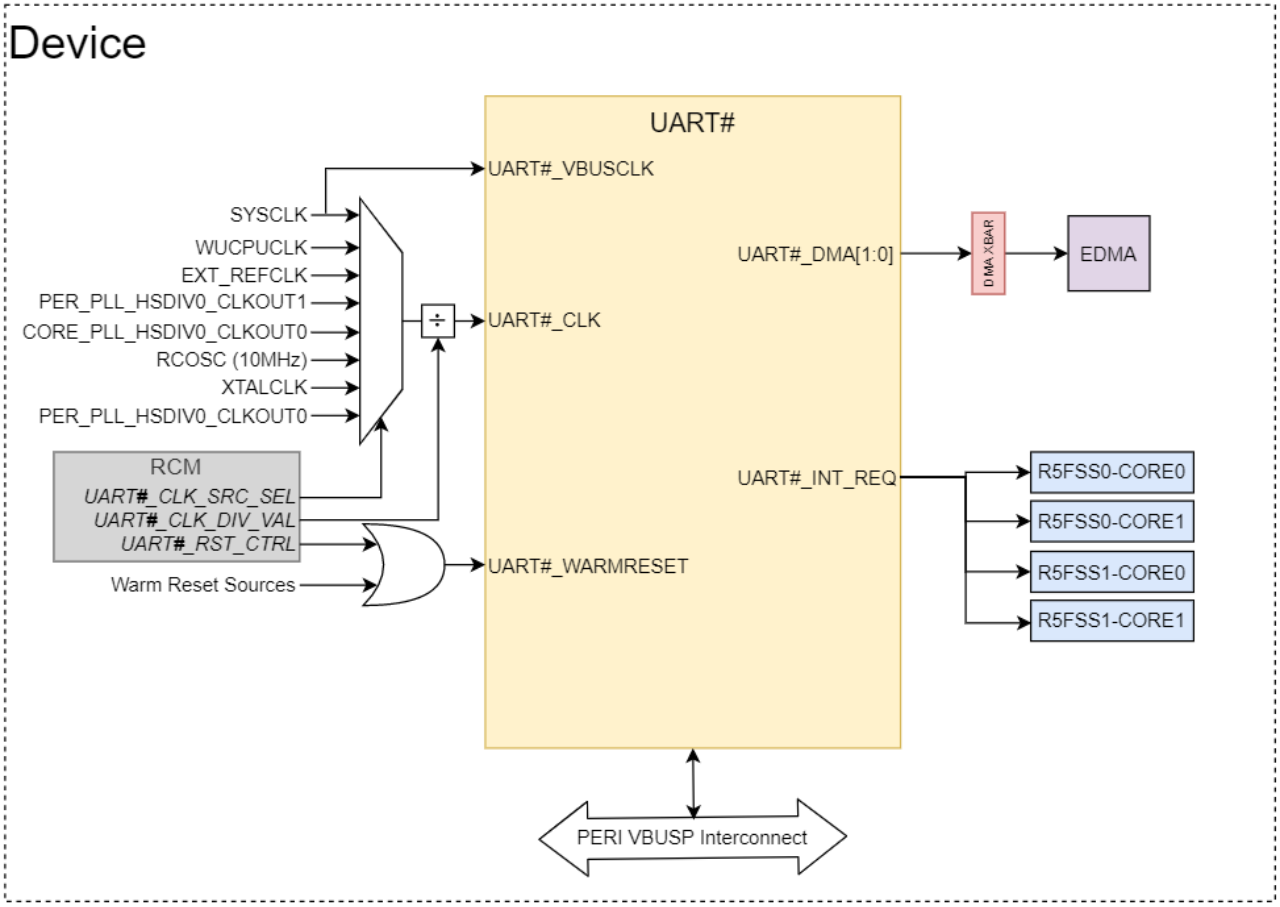
For more information on the interconnects, see the *System Interconnect* chapter.

For more information on power, reset, and clock management, see the corresponding sections within the *Device Configuration* chapter.

For more information on the device interrupt controllers, see the *Interrupt Controllers* chapter.

4.15 UART Integration

There are 6x UART modules integrated in the device. The diagram below provides a visual representation of the device integration details.



= 0, 1, 2, 3, 4, 5

Figure 4-17. UART Integration

The tables below summarize the device integration details of UART# (where # = 0, 1, 2, 3, 4, 5) in the device.

Table 4-25. *UART* Device Integration

This table describes the module device integration details.

Module Instance	Device Allocation	SoC Interconnect
UART0	✓	PERI VBUSP Interconnect
UART1	✓	PERI VBUSP Interconnect
UART2	✓	PERI VBUSP Interconnect
UART3	✓	PERI VBUSP Interconnect
UART4	✓	PERI VBUSP Interconnect
UART5	✓	PERI VBUSP Interconnect

Table 4-26. *UART* Clocks

This table describes the module clocking signals.

Module Instance	Module Clock Input	Source Clock Signal	Source	Default Freq	Description
UART0	UART0_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	UART0 VBUS Clock
	UART0_FCLK (UART_CLK)	XTALCLK	External XTAL	25 MHz	UART0 Interface Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK: HSDIV0_CLKOUT0	400 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
		DPLL_PER_HSDIV0_CLKOUT0	PLL_PER_CLK:HSDIV0_CLKOUT0	160 MHz	
UART1	UART1_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	UART1 VBUS Clock
	UART1_FCLK (UART_CLK)	XTALCLK	External XTAL	25 MHz	UART1 Interface Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK: HSDIV0_CLKOUT0	400 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
		DPLL_PER_HSDIV0_CLKOUT0	PLL_PER_CLK:HSDIV0_CLKOUT0	160 MHz	

Table 4-26. *UART* Clocks (continued)

This table describes the module clocking signals.

Module Instance	Module Clock Input	Source Clock Signal	Source	Default Freq	Description
UART2	UART2_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	UART2 VBUS Clock
	UART2_FCLK (UART_CLK)	XTALCLK	External XTAL	25 MHz	UART2 Interface Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK: HSDIV0_CLKOUT0	400 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
		DPLL_PER_HSDIV0_CLKOUT0	PLL_PER_CLK:HSDIV0_CLKOUT0	160 MHz	
UART3	UART3_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	UART3 VBUS Clock
	UART3_FCLK (UART_CLK)	XTALCLK	External XTAL	25 MHz	UART3 Interface Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK: HSDIV0_CLKOUT0	400 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
		DPLL_PER_HSDIV0_CLKOUT0	PLL_PER_CLK:HSDIV0_CLKOUT0	160 MHz	

Table 4-26. *UART* Clocks (continued)

This table describes the module clocking signals.

Module Instance	Module Clock Input	Source Clock Signal	Source	Default Freq	Description
UART4	UART4_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	UART4 VBUS Clock
	UART4_FCLK (UART_CLK)	XTALCLK	External XTAL	25 MHz	UART4 Interface Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK: HSDIV0_CLKOUT0	400 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
		DPLL_PER_HSDIV0_CLKOUT0	PLL_PER_CLK:HSDIV0_CLKOUT0	160 MHz	
UART5	UART5_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	UART5 VBUS Clock
	UART5_FCLK (UART_CLK)	XTALCLK	External XTAL	25 MHz	UART5 Interface Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK: HSDIV0_CLKOUT0	400 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
		DPLL_PER_HSDIV0_CLKOUT0	PLL_PER_CLK:HSDIV0_CLKOUT0	160 MHz	

Table 4-27. *UART* Resets

This table describes the module reset signals.

Module Instance	Module Reset Input	Source Reset Signal	Source	Description
UART0	UART0_RST(VBUSP_RSTn)	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	UART0 Asynchronous Reset
UART1	UART1_RST(VBUSP_RSTn)	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	UART1 Asynchronous Reset
UART2	UART2_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	UART2 Asynchronous Reset

Table 4-27. *UART* Resets (continued)

This table describes the module reset signals.

Module Instance	Module Reset Input	Source Reset Signal	Source	Description
UART3	UART3_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	UART3 Asynchronous Reset
UART4	UART4_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	UART4 Asynchronous Reset
UART5	UART5_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	UART5 Asynchronous Reset

Table 4-28. *UART* Interrupt Requests

This table describes the module interrupt requests.

Module Instance	Module Interrupt Signal	Destination Interrupt Input	Destination	Type	Description
UART0	uart0_int_req	uart0_int_req	ALL R5FSS Cores PRU-ICSS Core	Level	UART0 IP Status Information
UART1	uart1_int_req	uart1_int_req	ALL R5FSS Cores PRU-ICSS Core		UART1 IP Status Information
UART2	uart2_int_req	uart2_int_req	ALL R5FSS Cores PRU-ICSS Core		UART2 IP Status Information
UART3	uart3_int_req	uart4_int_req	ALL R5FSS Cores PRU-ICSS Core		UART3 IP Status Information
UART4	uart4_int_req	uart4_int_req	ALL R5FSS Cores PRU-ICSS Core		UART4 IP Status Information
UART5	uart5_int_req	uart5_int_req	ALL R5FSS Cores PRU-ICSS Core		UART5 IP Status Information

Table 4-29. *UART* DMA Requests

This table describes the module DMA requests.

Module Instance	Module DMA Event	Destination DMA Event Input	Destination	Type	Description
UART0	UART0_DMA_0	UART0_dma_req[0]	EDMA Crossbar (DMA_XBAR)	Level	UART0 DMA Request
	UART0_DMA_1	UART0_dma_req[1]			
UART1	UART1_DMA_0	UART1_dma_req[0]	EDMA Crossbar (DMA_XBAR)		UART1 DMA Request
	UART1_DMA_1	UART1_dma_req[1]			
UART2	UART2_DMA_0	UART2_dma_req[0]	EDMA Crossbar (DMA_XBAR)		UART2 DMA Request
	UART2_DMA_1	UART2_dma_req[1]			
UART3	UART3_DMA_0	UART3_dma_req[0]	EDMA Crossbar (DMA_XBAR)		UART3 DMA Request
	UART3_DMA_1	UART3_dma_req[1]			
UART4	UART4_DMA_0	UART4_dma_req[0]	EDMA Crossbar (DMA_XBAR)		UART4 DMA Request
	UART4_DMA_1	UART4_dma_req[1]			
UART5	UART5_DMA_0	UART5_dma_req[0]	EDMA Crossbar (DMA_XBAR)		UART5 DMA Request
	UART5_DMA_1	UART5_dma_req[1]			

Note

For more information on the interconnects, see the *System Interconnect* chapter.

For more information on power, reset, and clock management, see the corresponding sections within the *Device Configuration* chapter.

For more information on the device interrupt controllers, see the *Interrupt Controllers* chapter.

4.16 CPSW3G Integration

There is 1x CPSW3G module integrated in the device. The diagram below provides a visual representation of the device integration details.

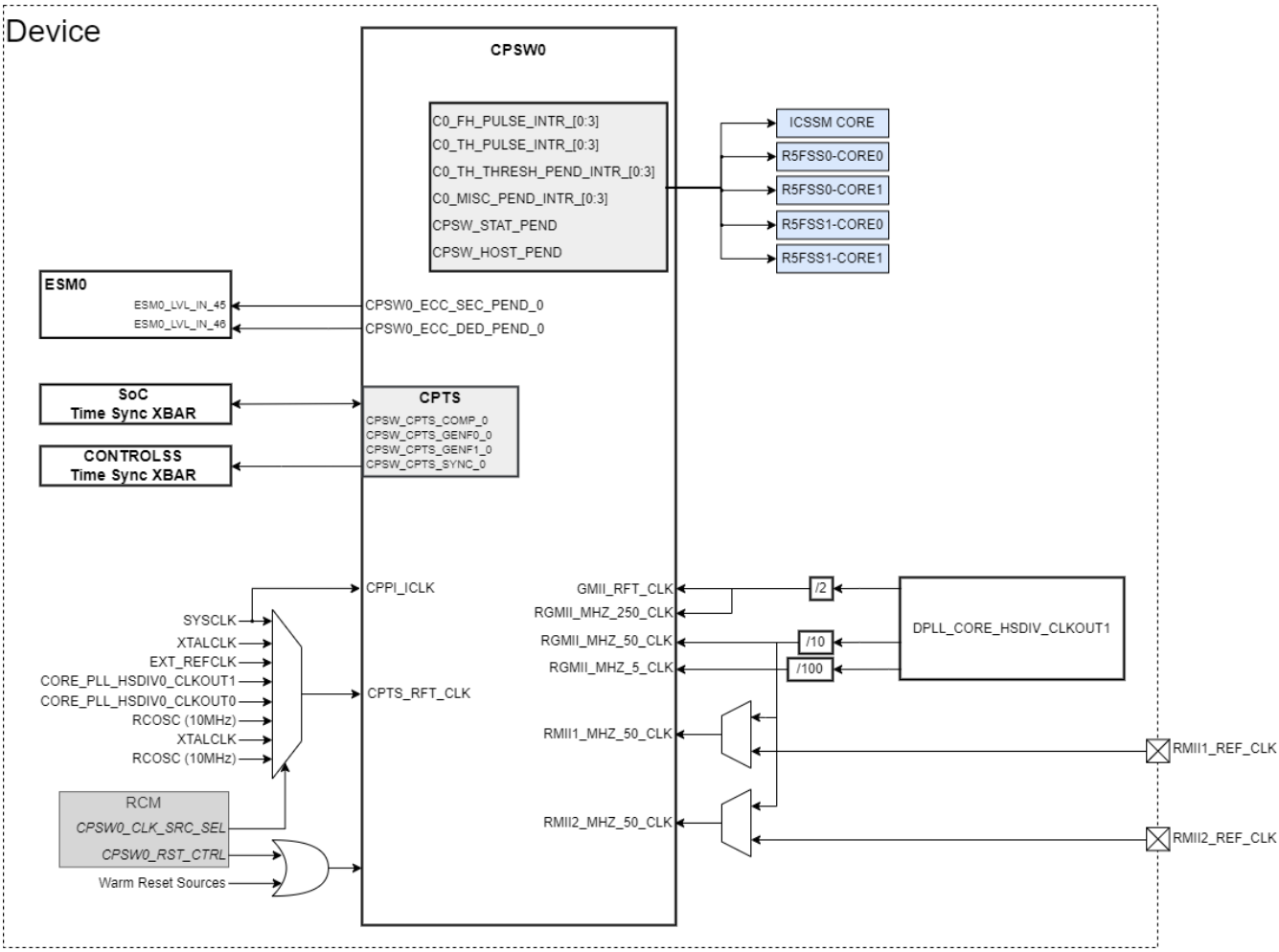


Figure 4-18. CPSW3G Integration Diagram

The tables below summarize the device integration details of CPSW0.

Table 4-30. *CPSW0* Device Integration

This table describes the module device integration details.

Module Instance	Device Allocation	SoC Interconnect
CPSW0	✓	INFRA0 VBUSP Interconnect

Table 4-31. *CPSW0* Clocks

This table describes the module clocking signals.

Module Instance	Module Clock Input	Source Clock Signal	Source	Default Freq	Description
CPSW0	CPPI_ICLK	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	CPSW0 Interface Clock
	CPTS_RFT_CLK	XTACLK	External XTAL	25 MHz	CPSW0 Interface Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	CPSW0 Interface Clock
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	CPSW0 Interface Clock
		DPLL_CORE_HSDIV0_CLKOUT1	PLL_CORE_CLK: HSDIV0_CLKOUT1	500 MHz	CPSW0 Interface Clock
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK: HSDIV0_CLKOUT0	400 MHz	CPSW0 Interface Clock
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	CPSW0 Interface Clock
		XTALCLK	External XTAL	25 MHz	CPSW0 Interface Clock
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	CPSW0 Interface Clock
	GMII_RFT_CLK	RGMII_250_CLK	RGMII 250 MHz Clock	250 MHz	CPSW0 Interface Clock
	RMII1_MHZ_50_CLK	RGMII_50_CLK	RGMII 50 MHz Clock	50 MHz	CPSW0 Interface Clock
		RMII1_REF_CLK	RMII1 Reference Clock	50 MHz[1]	CPSW0 Interface Clock
	RMII2_MHZ_50_CLK	RGMII_50_CLK	RGMII 50 MHz Clock	50 MHz	CPSW0 Interface Clock
		RMII2_REF_CLK	RMII2 Reference Clock	50 MHz[1]	CPSW0 Interface Clock
	RGMII_MHZ_50_CLK	RGMII_50_CLK	RGMII 50 MHz Clock	50 MHz	CPSW0 Interface Clock
	RGMII_MHZ_5_CLK	RGMII_5_CLK	RGMII 5 MHz Clock	5 MHz	CPSW0 Interface Clock
	RGMII_MHZ_250_CLK	RGMII_250_CLK	RGMII 250 MHz Clock	250 MHz	CPSW0 Interface Clock

Note

[1]The RMIIx_REF_CLK input pin can be drive by an external clock reference source. 50 MHz is required for proper operation.

Table 4-32. *CPSW0* Resets

This table describes the module reset signals.

Module Instance	Module Reset Input	Source Reset Signal	Source	Description
CPSW0	CPSW_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	CPSW0 Asynchronous Reset

Table 4-33. *CPSW0* Interrupt Requests

This table describes the module interrupt requests.

Module Instance	Module Interrupt Signal	Destination Interrupt Input	Destination	Type	Description
CPSW0	C0_FH_PULSE_INTR_[0:3]	C0_FH_PULSE_INTR	All R5FSS Cores ICSSM Core	Level	FHost (from host to Ethernet) paced pulse interrupt
	C0_TH_PULSE_INTR_[0:3]	C0_TH_PULSE_INTR	All R5FSS Cores ICSSM Core	Level	THost (from Ethernet to host) paced pulse interrupt
	C0_TH_THRESH_PEND_INTR_[0:3]	C0_TH_THRESH_PEND_INTR	All R5FSS Cores ICSSM Core	Level	THost (from Ethernet to host) non-paced pulse interrupt
	C0_MISC_PEND_INTR_[0:3]	C0_MISC_PEND_INTR	All R5FSS Cores ICSSM Core	Level	Miscellaneous non-paced pulse interrupt
	CPSW_STAT_PEND	STAT_PEND	All R5FSS Cores ICSSM Core	Level	Statistics level interrup
	CPSW_HOST_PEND	HOST_PEND	All R5FSS Cores ICSSM Core	Level	CPDMA host error level interrupt
	CPSW_ECC_SEC_PEND_INTR	ECC_SEC_PEND_INTR	ESM	Level	ECC SEC pulse interrupt – output from CPSW ECC module.
	CPSW_ECC_DED_PEND_INTR	ECC_DED_PEND_INTR	ESM	Level	ECC DED pulse interrupt – output from CPSW ECC module.

Table 4-34. *CPSW0* Time Sync and Compare Event

This table describes the module capture event inputs.

Module Instance	Module Event	Destination Event Input	Destination	Type	Description
CPSW0	CPSW0_CPTS_COMP	SoC_TimeSyncXBAR[0:3]	CPSW0_CPTS_COMP_INTR	Level	CPSW0 Compare Event Interrupt
		C2K_TimeSyncXBAR[0:3]			
	CPSW0_CPTS_GENF0	SoC_TimeSyncXBAR[0:3]	CPSW0_CPTS_GENF0_INTR	Level	CPSW0 CPTS generator function event interrupt 0
		C2K_TimeSyncXBAR[0:3]			
	CPSW0_CPTS_GENF1	SoC_TimeSyncXBAR[0:3]	CPSW0_CPTS_GENF1_INTR	Level	CPSW0 CPTS generator function event interrupt 1
		C2K_TimeSyncXBAR[0:3]			
	CPSW0_CPTS_SYNC	SoC_TimeSyncXBAR[0:3]	CPSW0_CPTS_SYNC_INTR	Level	CPSW0 CPTS Sync Event Interrupt
		C2K_TimeSyncXBAR[0:3]			

Note

For more information on the interconnects, see the *System Interconnect* chapter.

For more information on power, reset, and clock management, see the corresponding sections within the *Device Configuration* chapter.

For more information on the device interrupt controllers, see the *Interrupt Controllers* chapter.

For pin information on RGMII_ID_MODE and RGMII_REFCLK_SEL, see Register information and the corresponding section within the *Device Configuration* chapter

4.17 MMCSD Integration

There is 1x MMCSD integrated in the device. The diagram below provides a visual representation of the device integration details.

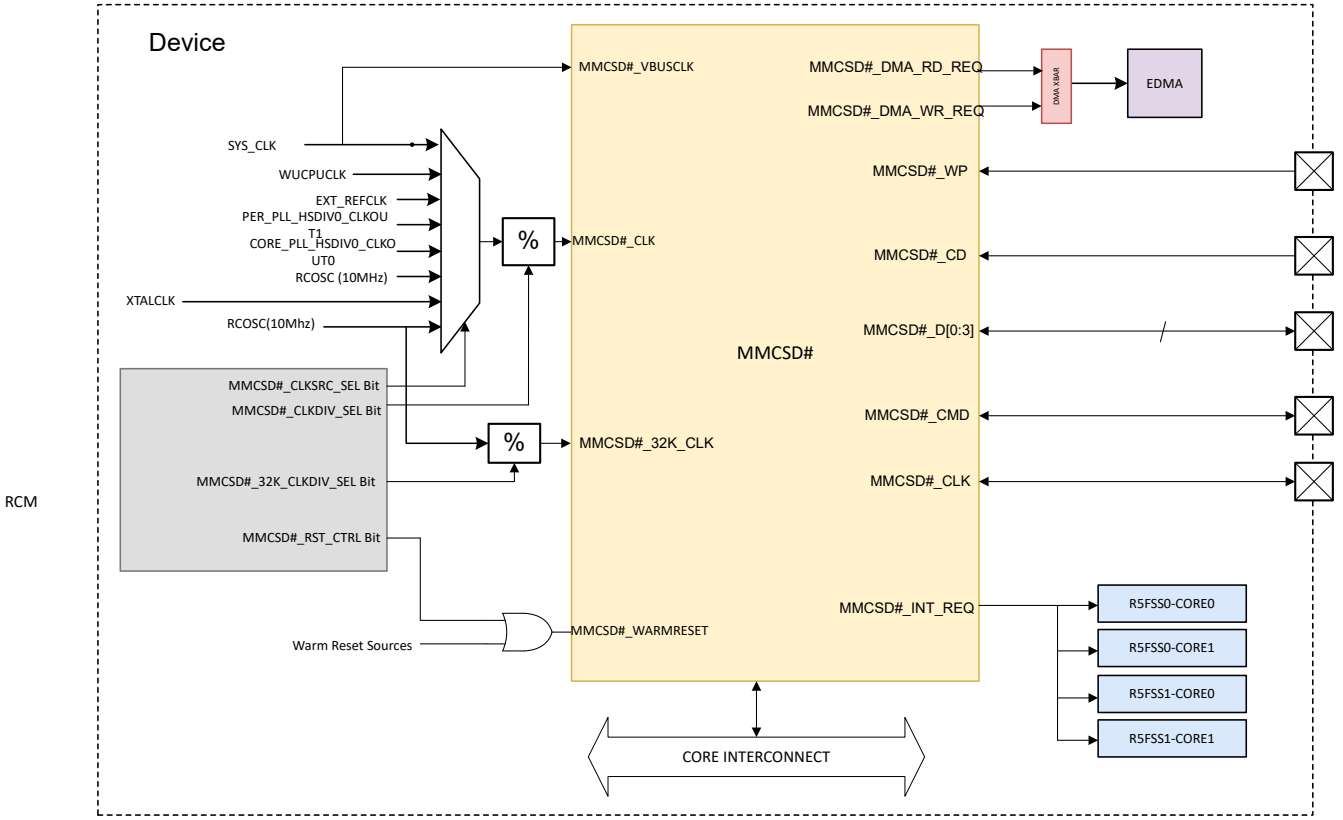


Figure 4-19. MMCSD Integration

The tables below summarize the device integration details of the MMC/SD module.

Table 4-35. *MMCSD* Device Integration

This table describes the module integration details.

MMCSD Instance	Device Allocation	SoC Interconnect
MMCSD0	✓	CORE VBUSM Interconnect

Table 4-36. *MMCSD* Clocks

This table describes the module clocking signals.

MMCSD Instance	MMCSD Clock Input	Source Clock Signal	Source	Default Freq	Description
MMCSD0	MMCSD0_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	MMC/SD Interface Clock
	MMCSD0_32K_CLK	MMCSD0_32K_CLK	XTALCLK	32 KHz	MMC/SD Debounce Clock
	MMCSD0_FCLK (MMCSD_CLK)	XTALCLK	External XTAL	25 MHz	MMC/SD Interface Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK: HSDIV0_CLKOUT0	400 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	

Table 4-37. *MMCSD* Resets

This table describes the module reset signals.

MMCSD Instance	MMCSD Reset Input	Source Reset Signal	Source	Description
MMCSD0	MMCSD0_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	MMCSD0 Asynchronous Reset

Table 4-38. *MMCSD* Interrupt Requests

This table describes the module interrupt requests.

MMCSD Instance	MMCSD Interrupt Signal	Destination Interrupt Input	Destination	Type	Description
MMCSD0	MMCSD0_INT_req_0	MMCSD0_INT_req_0	ALL R5FSS Cores	Level	MMC/SD Interrupt

Table 4-39. *MMCSD* DMA Requests

This table describes the module DMA requests.

MMCSD Instance	MMCSD DMA Event	Destination DMA Event Input	Destination	Type	Description
MMCSD0	MMCSD0_DMA_RD_REQ	MMCSD0_DMA_RD_REQ	EDMA Crossbar (DMA_XBAR)	Level	MMC/SD DMA Read Request
	MMCSD0_DMA_WR_REQ	MMCSD0_DMA_WR_REQ			MMC/SD DMA Write Request

Note

For more information on the interconnects, see the *System Interconnect* chapter.

For more information on power, reset, and clock management, see the corresponding sections within the *Device Configuration* chapter.

For more information on the device interrupt controllers, see the *Interrupt Controllers* chapter.

4.18 OSPI Integration

There is 1x OSPI module integrated in the device. The diagram below provides a visual representation of the device integration details.

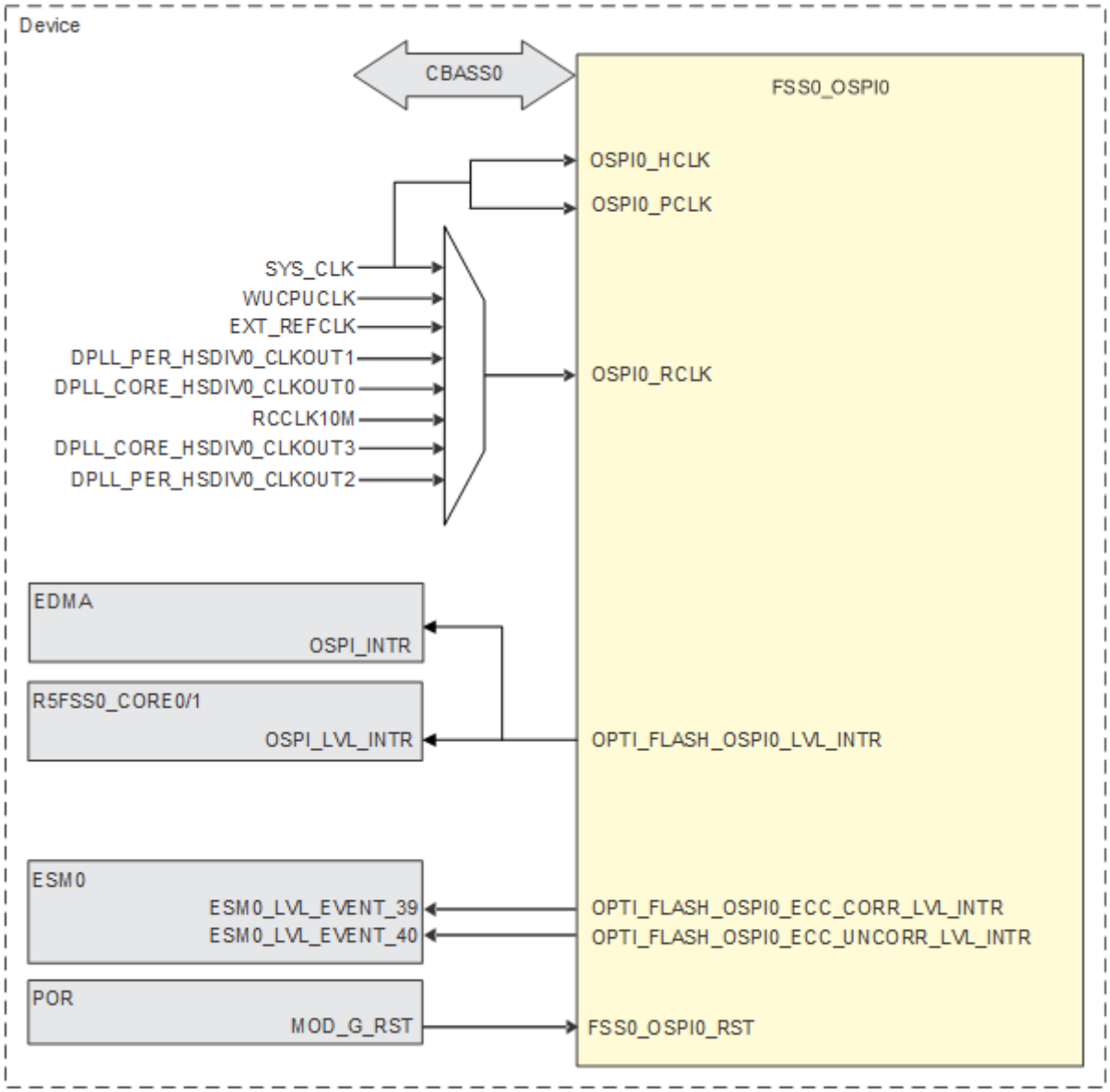


Figure 4-20. OSPI Integration Diagram

The tables below summarize the device integration details of OSPI.

Table 4-40. *OSPI* Device Integration

This table describes the module device integration details.

Module Instance	Device Allocation	SoC Interconnect
OSPI0	✓	CORE VBUSM Interconnect

Table 4-41. FSS0_OSPI Clocks

Module Instance	Module Clock Input	Source Clock Signal	Source	Description
FSS0_OSPI0	OSPI0_HCLK	SYS_CLK	SYS_CLK	FSS0_OSPI0 data transfer clock
	OSPI0_PCLK	SYS_CLK	SYS_CLK	FSS0_OSPI0 configuration clock

Table 4-41. FSS0_OSPI Clocks (continued)

Module Instance	Module Clock Input	Source Clock Signal	Source	Description
	OSPI0_RCLK	OSPI_CLK	WUCPUCLK	FSS0_OSPI0 Reference clock. Mux controlled by *MSS_RCM:OSPI0_CLK_SRC_SEL*
			EXT_REFCLK	
			SYS_CLK	
			DPLL_PER_HSDIV0_CLKOUT1	
			DPLL_CORE_HSDIV0_CLKOUT0	
			RCCLK10M	
			DPLL_CORE_HSDIV0_CLKOUT3	
			DPLL_PER_HSDIV0_CLKOUT2	

Table 4-42. FSS0_OSPI Resets

Module Instance	Module Reset Input	Source Reset Signal	Source	Description
FSS0_OSPI0	FSS0_OSPI0_RST	MOD_G_RST	POR	FSS0_OSPI0 reset

Table 4-43. FSS0_OSPI Interrupt Requests

Module Instance	Module Interrupt Signal	Destination Interrupt Input	Destination	Description	Type
FSS0_OSPI0	OPTI_FLASH_OSPI0_LVL_INTR	OSPI0_LVL_INTR	All R5FSS Cores ICSSM Core	FSS0_OSPI0 interrupt	Level
	OPTI_FLASH_OSPI0_ECC_CORR_LVL_INTR	ESM0_LVL_EVENT_39	OPTI_FLASH	FSS0_OSPI0 ECC Aggregator correctable error interrupt	Level
	OPTI_FLASH_OSPI0_ECC_UNCORR_LVL_INTR	ESM0_LVL_EVENT_40	OPTI_FLASH	FSS0_OSPI0 ECC Aggregator uncorrectable error interrupt	Level

Table 4-44. *OSPI* DMA Requests

This table describes the module DMA requests.

Module Instance	Module DMA Event	Destination DMA Event Input	Destination	Type	Description
OSPI0	OPTI_FLASH	OSPI_INTR	OPTI_FLASH_OSPI0_LVL_INTR	Pulse	OSPI0 DMA Event Request

Note

For more information on the interconnects, see the *System Interconnect* chapter.

For more information on power, reset, and clock management, see the corresponding sections within the *Device Configuration* chapter.

For more information on the device interrupt controllers, see the *Interrupt Controllers* chapter.

4.19 MCAN Integration

There are 8x MCAN modules integrated in the device. The diagram below provides a visual representation of the device integration details.

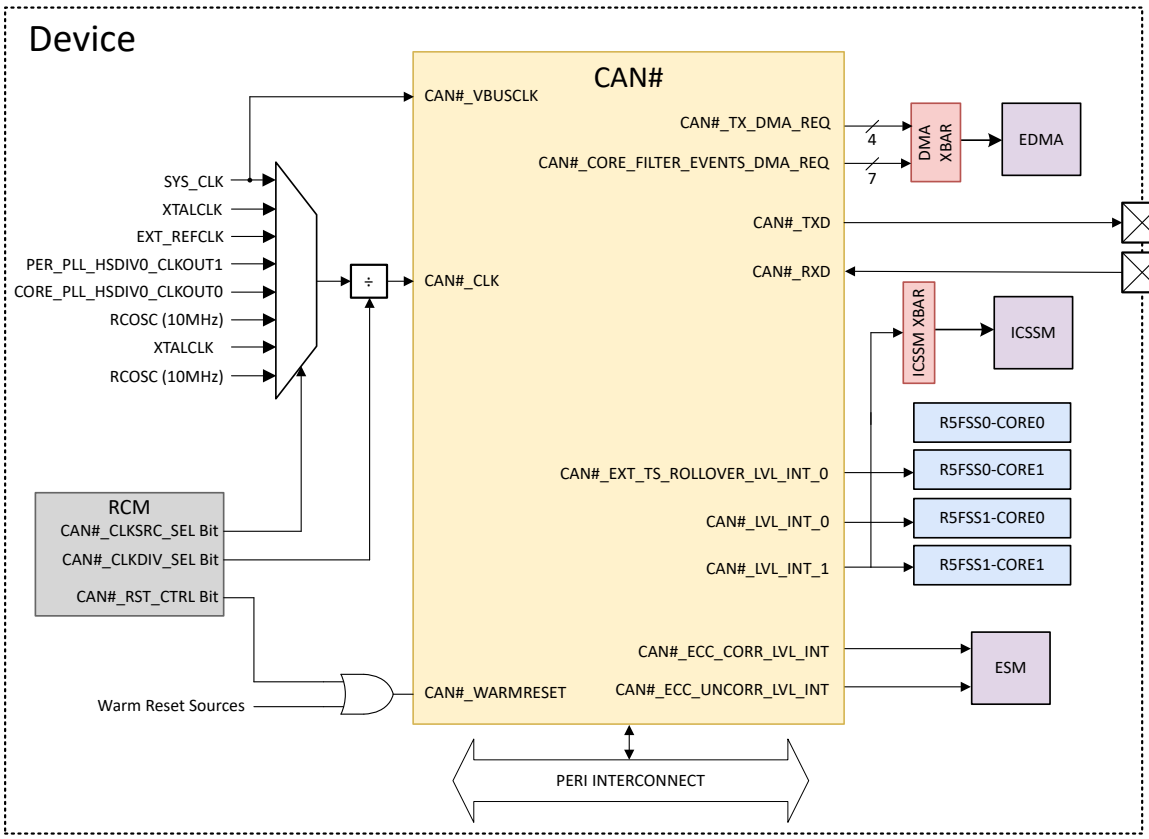


Figure 4-21. MCAN Integration Diagram

The tables below summarize the device integration details of MCAN# (where # = 0 to 7).

Table 4-45. *MCAN* Device Integration

This table describes the module device integration details.

Module Instance	Device Allocation	SoC Interconnect
MCAN0	✓	Peripheral VBUSP Interconnect
MCAN1	✓	Peripheral VBUSP Interconnect
MCAN2	✓	Peripheral VBUSP Interconnect
MCAN3	✓	Peripheral VBUSP Interconnect
MCAN4	✓	Peripheral VBUSP Interconnect
MCAN5	✓	Peripheral VBUSP Interconnect
MCAN6	✓	Peripheral VBUSP Interconnect
MCAN7	✓	Peripheral VBUSP Interconnect

Table 4-46. *MCAN* Clocks

This table describes the module clocking signals.

Module Instance	Module Clock Input	Source Clock Signal	Source	Default Freq	Description
MCAN0	MCAN0_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK:HSDIV0_CLKOUT0	200 MHz	MCAN0 Interface Clock
	MCAN0_FCLK (CAN_CLK)	XTALCLK	External Crystal (XTAL)	25 MHz	MCAN0 Functional Clock
		EXT_REFCLK	External Reference Clock(EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK:HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK:HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK:HSDIV0_CLKOUT1	500 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator(RCCLK10M)	10 MHz	
		XTALCLK	External Crystal (XTAL)	25 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator(RCCLK10M)	10 MHz	
MCAN1	MCAN1_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK:HSDIV0_CLKOUT0	200 MHz	MCAN1 Interface Clock
	MCAN1_FCLK (CAN_CLK)	XTALCLK	External Crystal (XTAL)	25 MHz	MCAN1 Functional Clock
		EXT_REFCLK	External Reference Clock(EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK:HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK:HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK:HSDIV0_CLKOUT1	500 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator(RCCLK10M)	10 MHz	
		XTALCLK	External Crystal (XTAL)	25 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator(RCCLK10M)	10 MHz	

Table 4-46. *MCAN* Clocks (continued)

This table describes the module clocking signals.

Module Instance	Module Clock Input	Source Clock Signal	Source	Default Freq	Description
MCAN2	MCAN2_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK:HSDIV0_CLKOUT0	200 MHz	MCAN2 Interface Clock
	MCAN2_FCLK (CAN_CLK)	XTALCLK	External Crystal (XTAL)	25 MHz	MCAN2 Functional Clock
		EXT_REFCLK	External Reference Clock(EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK:HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK:HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK:HSDIV0_CLKOUT1	500 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator(RCCLK10M)	10 MHz	
		XTALCLK	External Crystal (XTAL)	25 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator(RCCLK10M)	10 MHz	
MCAN3	MCAN3_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK:HSDIV0_CLKOUT0	200 MHz	MCAN3 Interface Clock
	MCAN3_FCLK (CAN_CLK)	XTALCLK	External Crystal (XTAL)	25 MHz	MCAN3 Functional Clock
		EXT_REFCLK	External Reference Clock(EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK:HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK:HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK:HSDIV0_CLKOUT1	500 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator(RCCLK10M)	10 MHz	
		XTALCLK	External Crystal (XTAL)	25 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator(RCCLK10M)	10 MHz	
MCAN4	MCAN4_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK:HSDIV0_CLKOUT0	200 MHz	MCAN4 Interface Clock
	MCAN4_FCLK (CAN_CLK)	XTALCLK	External Crystal (XTAL)	25 MHz	MCAN4 Functional Clock
		EXT_REFCLK	External Reference Clock(EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK:HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK:HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK:HSDIV0_CLKOUT1	500 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator(RCCLK10M)	10 MHz	
		XTALCLK	External Crystal (XTAL)	25 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator(RCCLK10M)	10 MHz	

Table 4-46. *MCAN* Clocks (continued)

This table describes the module clocking signals.

Module Instance	Module Clock Input	Source Clock Signal	Source	Default Freq	Description
MCAN5	MCAN5_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK:HSDIV0_CLKOUT0	200 MHz	MCAN5 Interface Clock
	MCAN5_FCLK (CAN_CLK)	XTALCLK	External Crystal (XTAL)	25 MHz	MCAN5 Functional Clock
		EXT_REFCLK	External Reference Clock(EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK:HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK:HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK:HSDIV0_CLKOUT1	500 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator(RCCLK10M)	10 MHz	
		XTALCLK	External Crystal (XTAL)	25 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator(RCCLK10M)	10 MHz	
MCAN6	MCAN6_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK:HSDIV0_CLKOUT0	200 MHz	MCAN6 Interface Clock
	MCAN6_FCLK (CAN_CLK)	XTALCLK	External Crystal (XTAL)	25 MHz	MCAN6 Functional Clock
		EXT_REFCLK	External Reference Clock(EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK:HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK:HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK:HSDIV0_CLKOUT1	500 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator(RCCLK10M)	10 MHz	
		XTALCLK	External Crystal (XTAL)	25 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator(RCCLK10M)	10 MHz	
MCAN7	MCAN7_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK:HSDIV0_CLKOUT0	200 MHz	MCAN7 Interface Clock
	MCAN7_FCLK (CAN_CLK)	XTALCLK	External Crystal (XTAL)	25 MHz	MCAN7 Functional Clock
		EXT_REFCLK	External Reference Clock(EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK:HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK:HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK:HSDIV0_CLKOUT1	500 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator(RCCLK10M)	10 MHz	
		XTALCLK	External Crystal (XTAL)	25 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator(RCCLK10M)	10 MHz	

Table 4-47. *MCAN* Resets

This table describes the module reset signals.

Module Instance	Module Reset Input	Source Reset Signal	Source	Description
MCAN0	MCAN0_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	Asynchronous MCAN0 Module Reset
MCAN1	MCAN1_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	Asynchronous MCAN1 Module Reset
MCAN2	MCAN2_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	Asynchronous MCAN2 Module Reset
MCAN3	MCAN3_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	Asynchronous MCAN3 Module Reset
MCAN4	MCAN4_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	Asynchronous MCAN4 Module Reset
MCAN5	MCAN5_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	Asynchronous MCAN5 Module Reset
MCAN6	MCAN6_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	Asynchronous MCAN6 Module Reset
MCAN7	MCAN7_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	Asynchronous MCAN7 Module Reset

Table 4-48. *MCAN* Interrupt Requests

This table describes the module interrupt requests.

Module Instance	Module Interrupt Signal	Destination Interrupt Input	Destination	Type	Description
MCAN0	MCAN0_INT_0	R5FSS0_CORE0_INTR_IN_27	R5FSS0-0	Level	MCAN0 Line 0 Interrupt Request
		R5FSS0_CORE1_INTR_IN_27	R5FSS0-1		
		R5FSS1_CORE0_INTR_IN_27	R5FSS1-0		
		R5FSS1_CORE1_INTR_IN_27	R5FSS1-1		
		PRU_ICSS0_INTR_IN_40	PRU_ICSS		
	MCAN0_INT_1	R5FSS0_CORE0_INTR_IN_28	R5FSS0-0	Level	MCAN0 Line 1 Interrupt Request
		R5FSS0_CORE1_INTR_IN_28	R5FSS0-1		
		R5FSS1_CORE0_INTR_IN_28	R5FSS1-0		
		R5FSS1_CORE1_INTR_IN_28	R5FSS1-1		
		PRU_ICSS0_INTR_IN_41	PRU_ICSS		
	MCAN0_EXT_TS_ROLLOVER_INT_0	R5FSS0_CORE0_INTR_IN_26	R5FSS0-0	Level	MCAN0 External TimeStamp Counter Rollover Interrupt
		R5FSS0_CORE1_INTR_IN_26	R5FSS0-1		
		R5FSS1_CORE0_INTR_IN_26	R5FSS1-0		
		R5FSS1_CORE1_INTR_IN_26	R5FSS1-1		
		PRU_ICSS0_INTR_IN_39	PRU_ICSS0		
	MCAN0_ECC_CORR_LVL_INT_0	ESM0_LVL_IN_2	ESM0	Level	MCAN0 ECC Correctable Error Interrupt
	MCAN0_ECC_UNCORR_LVL_INT_0	ESM0_LVL_IN_3	ESM0	Level	MCAN0 ECC Uncorrectable Error Interrupt

Table 4-48. *MCAN* Interrupt Requests (continued)

This table describes the module interrupt requests.

Module Instance	Module Interrupt Signal	Destination Interrupt Input	Destination	Type	Description
MCAN1	MCAN1_INT_0	R5FSS0_CORE0_INTR_IN_30	R5FSS0-0	Level	MCAN1 Line 0 Interrupt Request
		R5FSS0_CORE1_INTR_IN_30	R5FSS0-1		
		R5FSS1_CORE0_INTR_IN_30	R5FSS1-0		
		R5FSS1_CORE1_INTR_IN_30	R5FSS1-1		
		PRU_ICSS0_INTR_IN_43	PRU_ICSS		
	MCAN1_INT_1	R5FSS0_CORE0_INTR_IN_31	R5FSS0-0	Level	MCAN1 Line 1 Interrupt Request
		R5FSS0_CORE1_INTR_IN_31	R5FSS0-1		
		R5FSS1_CORE0_INTR_IN_31	R5FSS1-0		
		R5FSS1_CORE1_INTR_IN_31	R5FSS1-1		
		PRU_ICSS0_INTR_IN_44	PRU_ICSS		
	MCAN1_EXT_TS_ROLLOVER_INT_0	R5FSS0_CORE0_INTR_IN_29	R5FSS0-0	Level	MCAN1 External TimeStamp Counter Rollover Interrupt
		R5FSS0_CORE1_INTR_IN_29	R5FSS0-1		
		R5FSS1_CORE0_INTR_IN_29	R5FSS1-0		
		R5FSS1_CORE1_INTR_IN_29	R5FSS1-1		
		PRU_ICSS0_INTR_IN_42	PRU_ICSS		
	MCAN1_ECC_CORR_LVL_INT_0	ESM0_LVL_IN_4	ESM0	Level	MCAN1 ECC Correctable Error Interrupt
	MCAN1_ECC_UNCORR_LVL_INT_0	ESM0_LVL_IN_5	ESM0	Level	MCAN1 ECC Uncorrectable Error Interrupt
MCAN2	MCAN2_INT_0	R5FSS0_CORE1_INTR_IN_33	R5FSS0-0	Level	MCAN1 Line 0 Interrupt Request
		R5FSS0_CORE1_INTR_IN_33	R5FSS0-1		
		R5FSS1_CORE0_INTR_IN_33	R5FSS1-0		
		R5FSS1_CORE1_INTR_IN_33	R5FSS1-1		
		PRU_ICSS0_INTR_IN_46	PRU_ICSS		
	MCAN2_INT_1	R5FSS0_CORE0_INTR_IN_34	R5FSS0-0	Level	MCAN2 Line 1 Interrupt Request
		R5FSS0_CORE1_INTR_IN_34	R5FSS0-1		
		R5FSS1_CORE0_INTR_IN_34	R5FSS1-0		
		R5FSS1_CORE1_INTR_IN_34	R5FSS1-1		
		PRU_ICSS0_INTR_IN_47	PRU_ICSS		
	MCAN2_EXT_TS_ROLLOVER_INT_0	R5FSS0_CORE0_INTR_IN_32	R5FSS0-0	Level	MCAN2 External TimeStamp Counter Rollover Interrupt
		R5FSS0_CORE1_INTR_IN_32	R5FSS0-1		
		R5FSS1_CORE0_INTR_IN_32	R5FSS1-0		
		R5FSS1_CORE1_INTR_IN_32	R5FSS1-1		
		PRU_ICSS0_INTR_IN_45	PRU_ICSS		
	MCAN2_ECC_CORR_LVL_INT_0	ESM0_LVL_IN_6	ESM0	Level	MCAN2 ECC Correctable Error Interrupt
	MCAN2_ECC_UNCORR_LVL_INT_0	ESM0_LVL_IN_7	ESM0	Level	MCAN2 ECC Uncorrectable Error Interrupt

Table 4-48. *MCAN* Interrupt Requests (continued)

This table describes the module interrupt requests.

Module Instance	Module Interrupt Signal	Destination Interrupt Input	Destination	Type	Description
MCAN3	MCAN3_INT_0	R5FSS0_CORE0_INTR_IN_36	R5FSS0-0	Level	MCAN3 Line 0 Interrupt Request
		R5FSS0_CORE1_INTR_IN_36	R5FSS0-1		
		R5FSS1_CORE0_INTR_IN_36	R5FSS1-0		
		R5FSS1_CORE1_INTR_IN_36	R5FSS1-1		
		PRU_ICSS0_INTR_IN_49	PRU_ICSS		
	MCAN3_INT_1	R5FSS0_CORE0_INTR_IN_37	R5FSS0-0	Level	MCAN3 Line 1 Interrupt Request
		R5FSS0_CORE1_INTR_IN_37	R5FSS0-1		
		R5FSS1_CORE0_INTR_IN_37	R5FSS1-0		
		R5FSS1_CORE1_INTR_IN_37	R5FSS1-1		
		PRU_ICSS0_INTR_IN_50	PRU_ICSS		
	MCAN3_EXT_TS_ROLLOVER_INT_0	R5FSS0_CORE0_INTR_IN_35	R5FSS0-0	Level	MCAN3 External TimeStamp Counter Rollover Interrupt
		R5FSS0_CORE1_INTR_IN_35	R5FSS0-1		
		R5FSS1_CORE0_INTR_IN_35	R5FSS1-0		
		R5FSS1_CORE1_INTR_IN_35	R5FSS1-1		
		PRU_ICSS0_INTR_IN_48	PRU_ICSS		
	MCAN3_ECC_CORR_LVL_INT_0	ESM0_LVL_IN_8	ESM0	Level	MCAN3 ECC Correctable Error Interrupt
	MCAN3_ECC_UNCORR_LVL_INT_0	ESM0_LVL_IN_9	ESM0	Level	MCAN3 ECC Uncorrectable Error Interrupt
MCAN4	MCAN4_INT_0	R5FSS0_CORE0_INTR_IN_198	R5FSS0-0	Level	MCAN4 Line 0 Interrupt Request
		R5FSS0_CORE1_INTR_IN_198	R5FSS0-1		
		R5FSS1_CORE0_INTR_IN_198	R5FSS1-0		
		R5FSS1_CORE1_INTR_IN_198	R5FSS1-1		
		PRU_ICSS0_INTR_IN_60	PRU_ICSS		
	MCAN4_INT_1	R5FSS0_CORE0_INTR_IN_199	R5FSS0-0	Level	MCAN4 Line 1 Interrupt Request
		R5FSS0_CORE1_INTR_IN_199	R5FSS0-1		
		R5FSS1_CORE0_INTR_IN_199	R5FSS1-0		
		R5FSS1_CORE1_INTR_IN_199	R5FSS1-1		
		PRU_ICSS0_INTR_IN_61	PRU_ICSS		
	MCAN4_EXT_TS_ROLLOVER_INT_0	R5FSS0_CORE0_INTR_IN_197	R5FSS0-0	Level	MCAN4 External TimeStamp Counter Rollover Interrupt
		R5FSS0_CORE1_INTR_IN_197	R5FSS0-1		
		R5FSS1_CORE0_INTR_IN_197	R5FSS1-0		
		R5FSS1_CORE1_INTR_IN_197	R5FSS1-1		
		PRU_ICSS0_INTR_IN_59	PRU_ICSS		
	MCAN4_ECC_CORR_LVL_INT_0	ESM0_LVL_IN_39	ESM0	Level	MCAN4 ECC Correctable Error Interrupt
	MCAN4_ECC_UNCORR_LVL_INT_0	ESM0_LVL_IN_40	ESM0	Level	MCAN4 ECC Uncorrectable Error Interrupt

Table 4-48. *MCAN* Interrupt Requests (continued)

This table describes the module interrupt requests.

Module Instance	Module Interrupt Signal	Destination Interrupt Input	Destination	Type	Description
MCAN5	MCAN5_INT_0	R5FSS0_CORE0_INTR_IN_201	R5FSS0-0	Level	MCAN5 Line 0 Interrupt Request
		R5FSS0_CORE1_INTR_IN_201	R5FSS0-1		
		R5FSS1_CORE0_INTR_IN_201	R5FSS1-0		
		R5FSS1_CORE1_INTR_IN_201	R5FSS1-1		
		PRU_ICSS0_INTR_IN_63	PRU_ICSS		
	MCAN5_INT_1	R5FSS0_CORE0_INTR_IN_202	R5FSS0-0	Level	MCAN5 Line 1 Interrupt Request
		R5FSS0_CORE1_INTR_IN_202	R5FSS0-1		
		R5FSS1_CORE0_INTR_IN_202	R5FSS1-0		
		R5FSS1_CORE1_INTR_IN_202	R5FSS1-1		
		PRU_ICSS0_INTR_IN_64	PRU_ICSS		
	MCAN5_EXT_TS_ROLLOVER_INT_0	R5FSS0_CORE0_INTR_IN_200	R5FSS0-0	Level	MCAN5 External TimeStamp Counter Rollover Interrupt
		R5FSS0_CORE1_INTR_IN_200	R5FSS0-1		
		R5FSS1_CORE0_INTR_IN_200	R5FSS1-0		
		R5FSS1_CORE1_INTR_IN_200	R5FSS1-1		
		PRU_ICSS0_INTR_IN_62	PRU_ICSS		
	MCAN5_ECC_CORR_LVL_INT_0	ESM0_LVL_IN_41	ESM0	Level	MCAN5 ECC Correctable Error Interrupt
	MCAN5_ECC_UNCORR_LVL_INT_0	ESM0_LVL_IN_42	ESM0	Level	MCAN5 ECC Uncorrectable Error Interrupt
MCAN6	MCAN6_INT_0	R5FSS0_CORE0_INTR_IN_204	R5FSS0-0	Level	MCAN6 Line 0 Interrupt Request
		R5FSS0_CORE1_INTR_IN_204	R5FSS0-1		
		R5FSS1_CORE0_INTR_IN_204	R5FSS1-0		
		R5FSS1_CORE1_INTR_IN_204	R5FSS1-1		
		PRU_ICSS0_INTR_IN_66	PRU_ICSS		
	MCAN6_INT_1	R5FSS0_CORE0_INTR_IN_205	R5FSS0-0	Level	MCAN6 Line 1 Interrupt Request
		R5FSS0_CORE1_INTR_IN_205	R5FSS0-1		
		R5FSS1_CORE0_INTR_IN_205	R5FSS1-0		
		R5FSS1_CORE1_INTR_IN_205	R5FSS1-1		
		PRU_ICSS0_INTR_IN_67	PRU_ICSS		
	MCAN6_EXT_TS_ROLLOVER_INT_0	R5FSS0_CORE0_INTR_IN_203	R5FSS0-0	Level	MCAN6 External TimeStamp Counter Rollover Interrupt
		R5FSS0_CORE1_INTR_IN_203	R5FSS0-1		
		R5FSS1_CORE0_INTR_IN_203	R5FSS1-0		
		R5FSS1_CORE1_INTR_IN_203	R5FSS1-1		
		PRU_ICSS0_INTR_IN_65	PRU_ICSS		
	MCAN6_ECC_CORR_LVL_INT_0	ESM0_LVL_IN_43	ESM0	Level	MCAN6 ECC Correctable Error Interrupt
	MCAN6_ECC_UNCORR_LVL_INT_0	ESM0_LVL_IN_44	ESM0	Level	MCAN6 ECC Uncorrectable Error Interrupt

Table 4-48. *MCAN* Interrupt Requests (continued)

This table describes the module interrupt requests.

Module Instance	Module Interrupt Signal	Destination Interrupt Input	Destination	Type	Description
MCAN7	MCAN7_INT_0	R5FSS0_CORE0_INTR_IN_207	R5FSS0-0	Level	MCAN7 Line 0 Interrupt Request
		R5FSS0_CORE1_INTR_IN_207	R5FSS0-1		
		R5FSS1_CORE0_INTR_IN_207	R5FSS1-0		
		R5FSS1_CORE1_INTR_IN_207	R5FSS1-1		
		PRU_ICSS0_INTR_IN_69	PRU_ICSS		
	MCAN7_INT_1	R5FSS0_CORE0_INTR_IN_208	R5FSS0-0	Level	MCAN7 Line 1 Interrupt Request
		R5FSS0_CORE1_INTR_IN_208	R5FSS0-1		
		R5FSS1_CORE0_INTR_IN_208	R5FSS1-0		
		R5FSS1_CORE1_INTR_IN_208	R5FSS1-1		
		PRU_ICSS0_INTR_IN_70	PRU_ICSS		
	MCAN7_EXT_TS_ROLLOVER_INT_0	R5FSS0_CORE0_INTR_IN_206	R5FSS0-0	Level	MCAN7 External TimeStamp Counter Rollover Interrupt
		R5FSS0_CORE1_INTR_IN_206	R5FSS0-1		
		R5FSS1_CORE0_INTR_IN_206	R5FSS1-0		
		R5FSS1_CORE1_INTR_IN_206	R5FSS1-1		
		PRU_ICSS0_INTR_IN_68	PRU_ICSS		
	MCAN7_ECC_CORR_LVL_INT_0	ESM0_LVL_IN_45	ESM0	Level	MCAN7 ECC Correctable Error Interrupt
	MCAN7_ECC_UNCORR_LVL_INT_0	ESM0_LVL_IN_46	ESM0	Level	MCAN7 ECC Uncorrectable Error Interrupt

Table 4-49. *MCAN* DMA Requests

This table describes the module DMA requests.

Module Instance	Module DMA Event	Destination DMA Event Input	Destination	Type	Description
MCAN0	MCAN0_FE_INTR_0	EDMA_XBAR_147	EDMA	Pulse	MCAN0 Receive Filter Event 0 DMA Request
	MCAN0_FE_INTR_1	EDMA_XBAR_148	EDMA	Pulse	MCAN0 Receive Filter Event 1 DMA Request
	MCAN0_FE_INTR_2	EDMA_XBAR_149	EDMA	Pulse	MCAN0 Receive Filter Event 2 DMA Request
	MCAN0_FE_INTR_3	EDMA_XBAR_150	EDMA	Pulse	MCAN0 Receive Filter Event 3 DMA Request
	MCAN0_FE_INTR_4	EDMA_XBAR_151	EDMA	Pulse	MCAN0 Receive Filter Event 4 DMA Request
	MCAN0_FE_INTR_5	EDMA_XBAR_152	EDMA	Pulse	MCAN0 Receive Filter Event 5 DMA Request
	MCAN0_FE_INTR_6	EDMA_XBAR_153	EDMA	Pulse	MCAN0 Receive Filter Event 6 DMA Request
	MCAN0_TXDMA_0	EDMA_XBAR_74	EDMA	Pulse	MCAN0 Transmit Core DMA Request 0
	MCAN0_TXDMA_1	EDMA_XBAR_75	EDMA	Pulse	MCAN0 Transmit Core DMA Request 1
	MCAN0_TXDMA_2	EDMA_XBAR_76	EDMA	Pulse	MCAN0 Transmit Core DMA Request 2
	MCAN0_TXDMA_3	EDMA_XBAR_77	EDMA	Pulse	MCAN0 Transmit Core DMA Request 3
MCAN1	MCAN1_FE_INTR_0	EDMA_XBAR_154	EDMA	Pulse	MCAN1 Receive Filter Event 0 DMA Request
	MCAN1_FE_INTR_1	EDMA_XBAR_155	EDMA	Pulse	MCAN1 Receive Filter Event 1 DMA Request
	MCAN1_FE_INTR_2	EDMA_XBAR_156	EDMA	Pulse	MCAN1 Receive Filter Event 2 DMA Request
	MCAN1_FE_INTR_3	EDMA_XBAR_157	EDMA	Pulse	MCAN1 Receive Filter Event 3 DMA Request
	MCAN1_FE_INTR_4	EDMA_XBAR_158	EDMA	Pulse	MCAN1 Receive Filter Event 4 DMA Request
	MCAN1_FE_INTR_5	EDMA_XBAR_159	EDMA	Pulse	MCAN1 Receive Filter Event 5 DMA Request
	MCAN1_FE_INTR_6	EDMA_XBAR_160	EDMA	Pulse	MCAN1 Receive Filter Event 6 DMA Request
	MCAN1_TXDMA_0	EDMA_XBAR_78	EDMA	Pulse	MCAN1 Transmit Core DMA Request 0
	MCAN1_TXDMA_1	EDMA_XBAR_79	EDMA	Pulse	MCAN1 Transmit Core DMA Request 1
	MCAN1_TXDMA_2	EDMA_XBAR_80	EDMA	Pulse	MCAN1 Transmit Core DMA Request 2
	MCAN1_TXDMA_3	EDMA_XBAR_81	EDMA	Pulse	MCAN1 Transmit Core DMA Request 3

Table 4-49. *MCAN* DMA Requests (continued)

This table describes the module DMA requests.

Module Instance	Module DMA Event	Destination DMA Event Input	Destination	Type	Description
MCAN2	MCAN2_FE_INTR_0	EDMA_XBAR_161	EDMA	Pulse	MCAN2 Receive Filter Event 0 DMA Request
	MCAN2_FE_INTR_1	EDMA_XBAR_162	EDMA	Pulse	MCAN2 Receive Filter Event 1 DMA Request
	MCAN2_FE_INTR_2	EDMA_XBAR_163	EDMA	Pulse	MCAN2 Receive Filter Event 2 DMA Request
	MCAN2_FE_INTR_3	EDMA_XBAR_164	EDMA	Pulse	MCAN2 Receive Filter Event 3 DMA Request
	MCAN2_FE_INTR_4	EDMA_XBAR_165	EDMA	Pulse	MCAN2 Receive Core Filter Event 4 DMA Request
	MCAN2_FE_INTR_5	EDMA_XBAR_166	EDMA	Pulse	MCAN2 Receive Filter Event 5 DMA Request
	MCAN2_FE_INTR_6	EDMA_XBAR_167	EDMA	Pulse	MCAN2 Receiver Filter Event 6 DMA Request
	MCAN2_TXDMA_0	EDMA_XBAR_82	EDMA	Pulse	MCAN2 Transmit Core DMA Request 0
	MCAN2_TXDMA_1	EDMA_XBAR_83	EDMA	Pulse	MCAN2 Transmit Core DMA Request 1
	MCAN2_TXDMA_2	EDMA_XBAR_84	EDMA	Pulse	MCAN2 Transmit Core DMA Request 2
	MCAN2_TXDMA_3	EDMA_XBAR_85	EDMA	Pulse	MCAN2 Transmit Core DMA Request 3
MCAN3	MCAN3_FE_INTR_0	EDMA_XBAR_168	EDMA	Pulse	MCAN3 Receive Filter Event 0 DMA Request
	MCAN3_FE_INTR_1	EDMA_XBAR_169	EDMA	Pulse	MCAN3 Receive Filter Event 1 DMA Request
	MCAN3_FE_INTR_2	EDMA_XBAR_170	EDMA	Pulse	MCAN3 Receive Filter Event 2 DMA Request
	MCAN3_FE_INTR_3	EDMA_XBAR_171	EDMA	Pulse	MCAN3 Receive Filter Event 3 DMA Request
	MCAN3_FE_INTR_4	EDMA_XBAR_172	EDMA	Pulse	MCAN3 Receive Filter Event 4 DMA Request
	MCAN3_FE_INTR_5	EDMA_XBAR_173	EDMA	Pulse	MCAN3 Receive Filter Event 5 DMA Request
	MCAN3_FE_INTR_6	EDMA_XBAR_174	EDMA	Pulse	MCAN3 Receive Filter Event 6 DMA Request
	MCAN3_TXDMA_0	EDMA_XBAR_86	EDMA	Pulse	MCAN3 Transmit Core DMA Request 0
	MCAN3_TXDMA_1	EDMA_XBAR_87	EDMA	Pulse	MCAN3 Transmit Core DMA Request 1
	MCAN3_TXDMA_2	EDMA_XBAR_88	EDMA	Pulse	MCAN3 Transmit Core DMA Request 2
	MCAN3_TXDMA_3	EDMA_XBAR_89	EDMA	Pulse	MCAN3 Transmit Core DMA Request 3

Table 4-49. *MCAN* DMA Requests (continued)

This table describes the module DMA requests.

Module Instance	Module DMA Event	Destination DMA Event Input	Destination	Type	Description
MCAN4	MCAN4_FE_INTR_0	EDMA_XBAR_192	EDMA	Pulse	MCAN4 Receive Filter Event 0 DMA Request
	MCAN4_FE_INTR_1	EDMA_XBAR_193	EDMA	Pulse	MCAN4 Receive Filter Event 1 DMA Request
	MCAN4_FE_INTR_2	EDMA_XBAR_194	EDMA	Pulse	MCAN4 Receive Filter Event 2 DMA Request
	MCAN4_FE_INTR_3	EDMA_XBAR_195	EDMA	Pulse	MCAN4 Receive Filter Event 3 DMA Request
	MCAN4_FE_INTR_4	EDMA_XBAR_196	EDMA	Pulse	MCAN4 Receive Filter Event 4 DMA Request
	MCAN4_FE_INTR_5	EDMA_XBAR_197	EDMA	Pulse	MCAN4 Receive Filter Event 5 DMA Request
	MCAN4_FE_INTR_6	EDMA_XBAR_198	EDMA	Pulse	MCAN4 Receive Filter Event 6 DMA Request
	MCAN4_TXDMA_0	EDMA_XBAR_176	EDMA	Pulse	MCAN4 Transmit Core DMA Request 0
	MCAN4_TXDMA_1	EDMA_XBAR_177	EDMA	Pulse	MCAN4 Transmit Core DMA Request 1
	MCAN4_TXDMA_2	EDMA_XBAR_178	EDMA	Pulse	MCAN4 Transmit Core DMA Request 2
	MCAN4_TXDMA_3	EDMA_XBAR_179	EDMA	Pulse	MCAN4 Transmit Core DMA Request 3
MCAN5	MCAN5_FE_INTR_0	EDMA_XBAR_199	EDMA	Pulse	MCAN5 Receive Filter Event 0 DMA Request
	MCAN5_FE_INTR_1	EDMA_XBAR_200	EDMA	Pulse	MCAN5 Receive Filter Event 1 DMA Request
	MCAN5_FE_INTR_2	EDMA_XBAR_201	EDMA	Pulse	MCAN5 Receive Filter Event 2 DMA Request
	MCAN5_FE_INTR_3	EDMA_XBAR_202	EDMA	Pulse	MCAN5 Receive Filter Event 3 DMA Request
	MCAN5_FE_INTR_4	EDMA_XBAR_203	EDMA	Pulse	MCAN5 Receive Filter Event 4 DMA Request
	MCAN5_FE_INTR_5	EDMA_XBAR_204	EDMA	Pulse	MCAN5 Receive Filter Event 5 DMA Request
	MCAN5_FE_INTR_6	EDMA_XBAR_205	EDMA	Pulse	MCAN5 Receive Filter Event 6 DMA Request
	MCAN5_TXDMA_0	EDMA_XBAR_180	EDMA	Pulse	MCAN5 Transmit Core DMA Request 0
	MCAN5_TXDMA_1	EDMA_XBAR_181	EDMA	Pulse	MCAN5 Transmit Core DMA Request 1
	MCAN5_TXDMA_2	EDMA_XBAR_182	EDMA	Pulse	MCAN5 Transmit Core DMA Request 2
	MCAN5_TXDMA_3	EDMA_XBAR_183	EDMA	Pulse	MCAN5 Transmit Core DMA Request 3

Table 4-49. *MCAN* DMA Requests (continued)

This table describes the module DMA requests.

Module Instance	Module DMA Event	Destination DMA Event Input	Destination	Type	Description
MCAN6	MCAN6_FE_INTR_0	EDMA_XBAR_206	EDMA	Pulse	MCAN6 Receive Filter Event 0 DMA Request
	MCAN6_FE_INTR_1	EDMA_XBAR_207	EDMA	Pulse	MCAN6 Receive Filter Event 1 DMA Request
	MCAN6_FE_INTR_2	EDMA_XBAR_208	EDMA	Pulse	MCAN6 Receive Filter Event 2 DMA Request
	MCAN6_FE_INTR_3	EDMA_XBAR_209	EDMA	Pulse	MCAN6 Receive Filter Event 3 DMA Request
	MCAN6_FE_INTR_4	EDMA_XBAR_210	EDMA	Pulse	MCAN6 Receive Filter Event 4 DMA Request
	MCAN6_FE_INTR_5	EDMA_XBAR_211	EDMA	Pulse	MCAN6 Receive Filter Event 5 DMA Request
	MCAN6_FE_INTR_6	EDMA_XBAR_212	EDMA	Pulse	MCAN6 Receive Filter Event 6 DMA Request
	MCAN6_TXDMA_0	EDMA_XBAR_184	EDMA	Pulse	MCAN6 Transmit Core DMA Request 0
	MCAN6_TXDMA_1	EDMA_XBAR_185	EDMA	Pulse	MCAN6 Transmit Core DMA Request 1
	MCAN6_TXDMA_2	EDMA_XBAR_186	EDMA	Pulse	MCAN6 Transmit Core DMA Request 2
	MCAN6_TXDMA_3	EDMA_XBAR_187	EDMA	Pulse	MCAN6 Transmit Core DMA Request 3

Table 4-49. *MCAN* DMA Requests (continued)

This table describes the module DMA requests.

Module Instance	Module DMA Event	Destination DMA Event Input	Destination	Type	Description
MCAN7	MCAN7_FE_INTR_0	EDMA_XBAR_213	EDMA	Pulse	MCAN7 Receive Filter Event 0 DMA Request
	MCAN7_FE_INTR_1	EDMA_XBAR_214	EDMA	Pulse	MCAN7 Receive Filter Event 1 DMA Request
	MCAN7_FE_INTR_2	EDMA_XBAR_215	EDMA	Pulse	MCAN7 Receive Filter Event 2 DMA Request
	MCAN7_FE_INTR_3	EDMA_XBAR_216	EDMA	Pulse	MCAN7 Receive Filter Event 3 DMA Request
	MCAN7_FE_INTR_4	EDMA_XBAR_217	EDMA	Pulse	MCAN7 Receive Filter Event 4 DMA Request
	MCAN7_FE_INTR_5	EDMA_XBAR_218	EDMA	Pulse	MCAN7 Receive Filter Event 5 DMA Request
	MCAN7_FE_INTR_6	EDMA_XBAR_219	EDMA	Pulse	MCAN7 Receive Filter Event 6 DMA Request
	MCAN7_TXDMA_0	EDMA_XBAR_188	EDMA	Pulse	MCAN7 Transmit Core DMA Request 0
	MCAN7_TXDMA_1	EDMA_XBAR_189	EDMA	Pulse	MCAN7 Transmit Core DMA Request 1
	MCAN7_TXDMA_2	EDMA_XBAR_190	EDMA	Pulse	MCAN7 Transmit Core DMA Request 2
	MCAN7_TXDMA_3	EDMA_XBAR_191	EDMA	Pulse	MCAN7 Transmit Core DMA Request 3

Note

For more information on the interconnects, see the *System Interconnect* chapter.

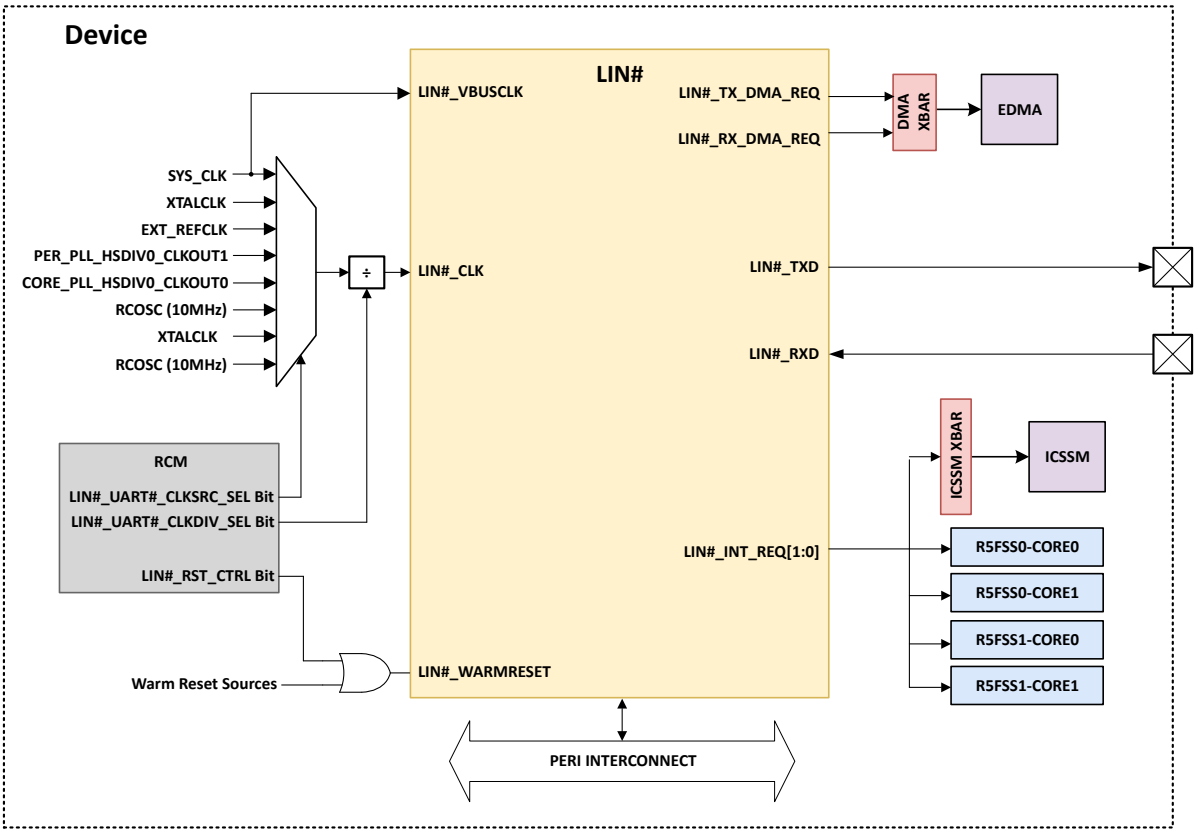
For more information on power, reset, and clock management, see the corresponding sections within the *Device Configuration* chapter.

For more information on the device interrupt controllers, see the *Interrupt Controllers* chapter.

There are 5x LIN modules integrated in the device. The diagram below provides a visual representation of the device integration details.

= 0 to 4

Figure 4-22. LIN Integration



The tables below summarize the device integration details of LIN#.

Table 4-50. *LIN* Device Integration

This table describes the LIN device integration details.

LIN Instance	Device Allocation	SoC Interconnect
LIN0	✓	Peripheral VBUSP Interconnect
LIN1	✓	Peripheral VBUSP Interconnect
LIN2	✓	Peripheral VBUSP Interconnect
LIN3	✓	Peripheral VBUSP Interconnect
LIN4	✓	Peripheral VBUSP Interconnect

Table 4-51. *LIN* Clocks

This table describes the LIN clocking signals.

LIN Instance	LIN Clock Input	Source Clock Signal	Source	Default Freq	Description
LIN0	LIN0_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	LIN0 Interface Clock (LIN0_CLK must be running for register access)
	LIN0_FCLK (LIN_CLK)	WUCPUCLK	Wakeup CPU Clock	25 MHz	LIN0 Functional Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK: HSDIV0_CLKOUT0	400 MHz	
		RCCLK10M	Internal 10MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
		DPLL_PER_HSDIV0_CLKOUT0	PLL_PER_CLK:HSDIV0_CLKOUT0	160 MHz	
LIN1	LIN1_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	LIN1 Interface Clock (LIN1_CLK must be running for register access)
	LIN1_FCLK (LIN_CLK)	WUCPUCLK	Wakeup CPU Clock	25 MHz	LIN1 Functional Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK: HSDIV0_CLKOUT0	400 MHz	
		RCCLK10M	Internal 10MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
		DPLL_PER_HSDIV0_CLKOUT0	PLL_PER_CLK:HSDIV0_CLKOUT0	160 MHz	

Table 4-51. *LIN* Clocks (continued)

This table describes the LIN clocking signals.

LIN Instance	LIN Clock Input	Source Clock Signal	Source	Default Freq	Description
LIN2	LIN2_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	LIN2 Interface Clock (LIN2_CLK must be running for register access)
	LIN2_FCLK (LIN_CLK)	WUCPUCLK	Wakeup CPU Clock	25 MHz	LIN2 Functional Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK: HSDIV0_CLKOUT0	400 MHz	
		RCCLK10M	Internal 10MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
		DPLL_PER_HSDIV0_CLKOUT0	PLL_PER_CLK:HSDIV0_CLKOUT0	160 MHz	
LIN3	LIN3_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	LIN3 Interface Clock (LIN3_CLK must be running for register access)
	LIN3_FCLK (LIN_CLK)	WUCPUCLK	Wakeup CPU Clock	25 MHz	LIN3 Functional Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK: HSDIV0_CLKOUT0	400 MHz	
		RCCLK10M	Internal 10MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
		DPLL_PER_HSDIV0_CLKOUT0	PLL_PER_CLK:HSDIV0_CLKOUT0	160 MHz	

Table 4-51. *LIN* Clocks (continued)

This table describes the LIN clocking signals.

LIN Instance	LIN Clock Input	Source Clock Signal	Source	Default Freq	Description
LIN4	LIN4_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	LIN4 Interface Clock (LIN4_CLK must be running for register access)
	LIN4_FCLK (LIN_CLK)	WUCPUCLK	Wakeup CPU Clock	25 MHz	LIN4 Functional Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK: HSDIV0_CLKOUT0	400 MHz	
		RCCLK10M	Internal 10MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
		DPLL_PER_HSDIV0_CLKOUT0	PLL_PER_CLK:HSDIV0_CLKOUT0	160 MHz	

Table 4-52. *LIN* Resets

This table describes the LIN reset signals.

LIN Instance	LIN Reset Input	Source Reset Signal	Source	Description
LIN0	LIN0_RST (VBUSP_RSTn)	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	LIN0 Asynchronous Reset
LIN1	LIN1_RST (VBUSP_RSTn)	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	LIN1 Asynchronous Reset
LIN2	LIN2_RST (VBUSP_RSTn)	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	LIN2 Asynchronous Reset
LIN3	LIN3_RST (VBUSP_RSTn)	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	LIN3 Asynchronous Reset
LIN4	LIN4_RST (VBUSP_RSTn)	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	LIN4 Asynchronous Reset

Table 4-53. *LIN* Interrupt Requests

This table describes the LIN interrupt requests.

LIN Instance	LIN Interrupt Signal	Destination Interrupt Input	Destination	Type	Description
LIN0	LIN0_INT_req_0	LIN0_INT_req_0	ALL R5FSS Cores, ICSSMXBAR	Pulse	LIN0 Event Interrupts
	LIN0_INT_req_1	LIN0_INT_req_1			
LIN1	LIN1_INT_req_0	LIN1_INT_req_0	ALL R5FSS Cores, ICSSMXBAR	Pulse	LIN1 Event Interrupts
	LIN1_INT_req_1	LIN1_INT_req_1			
LIN2	LIN2_INT_req_0	LIN2_INT_req_0	ALL R5FSS Cores, ICSSMXBAR	Pulse	LIN2 Event Interrupts
	LIN2_INT_req_1	LIN2_INT_req_1			
LIN3	LIN3_INT_req_0	LIN3_INT_req_0	ALL R5FSS Cores, ICSSMXBAR	Pulse	LIN3 Event Interrupts
	LIN3_INT_req_1	LIN3_INT_req_1			

Table 4-53. *LIN* Interrupt Requests (continued)

This table describes the LIN interrupt requests.

LIN Instance	LIN Interrupt Signal	Destination Interrupt Input	Destination	Type	Description
LIN4	LIN4_INT_req_0	LIN4_INT_req_0	ALL R5FSS Cores, ICSSMXBAR	Pulse	LIN4 Event Interrupts
	LIN4_INT_req_1	LIN4_INT_req_1			

Table 4-54. *LIN* DMA Requests

This table describes the LIN DMA requests.

LIN Instance	LIN DMA Event	Destination DMA Event Input	Destination	Type	Description
LIN0	LIN0_TX_DMA_REQ	LIN0_tx_dma_req	EDMA Crossbar (DMA_XBAR)	Pulse	LIN0 TX DMA Request
	LIN0_RX_DMA_REQ	LIN0_rx_dma_req			LIN0 RX DMA Request
LIN1	LIN1_TX_DMA_REQ	LIN1_tx_dma_req	EDMA Crossbar (DMA_XBAR)	Pulse	LIN1 TX DMA Request
	LIN1_RX_DMA_REQ	LIN1_rx_dma_req			LIN1 RX DMA Request
LIN2	LIN2_TX_DMA_REQ	LIN2_tx_dma_req	EDMA Crossbar (DMA_XBAR)	Pulse	LIN2 TX DMA Request
	LIN2_RX_DMA_REQ	LIN2_rx_dma_req			LIN2 RX DMA Request
LIN3	LIN3_TX_DMA_REQ	LIN3_tx_dma_req	EDMA Crossbar (DMA_XBAR)	Pulse	LIN3 TX DMA Request
	LIN3_RX_DMA_REQ	LIN3_rx_dma_req			LIN3 RX DMA Request
LIN4	LIN4_TX_DMA_REQ	LIN4_tx_dma_req	EDMA Crossbar (DMA_XBAR)	Pulse	LIN4 TX DMA Request
	LIN4_RX_DMA_REQ	LIN4_rx_dma_req			LIN4 RX DMA Request

Note

For more information on the interconnects, see the *System Interconnect* chapter.

For more information on power, reset, and clock management, see the corresponding sections within the *Device Configuration* chapter.

For more information on the device interrupt controllers, see the *Interrupt Controllers* chapter.

4.20 RTI Integration

There are 8x RTI modules integrated in the device. The diagram and tables below show the device integration details.

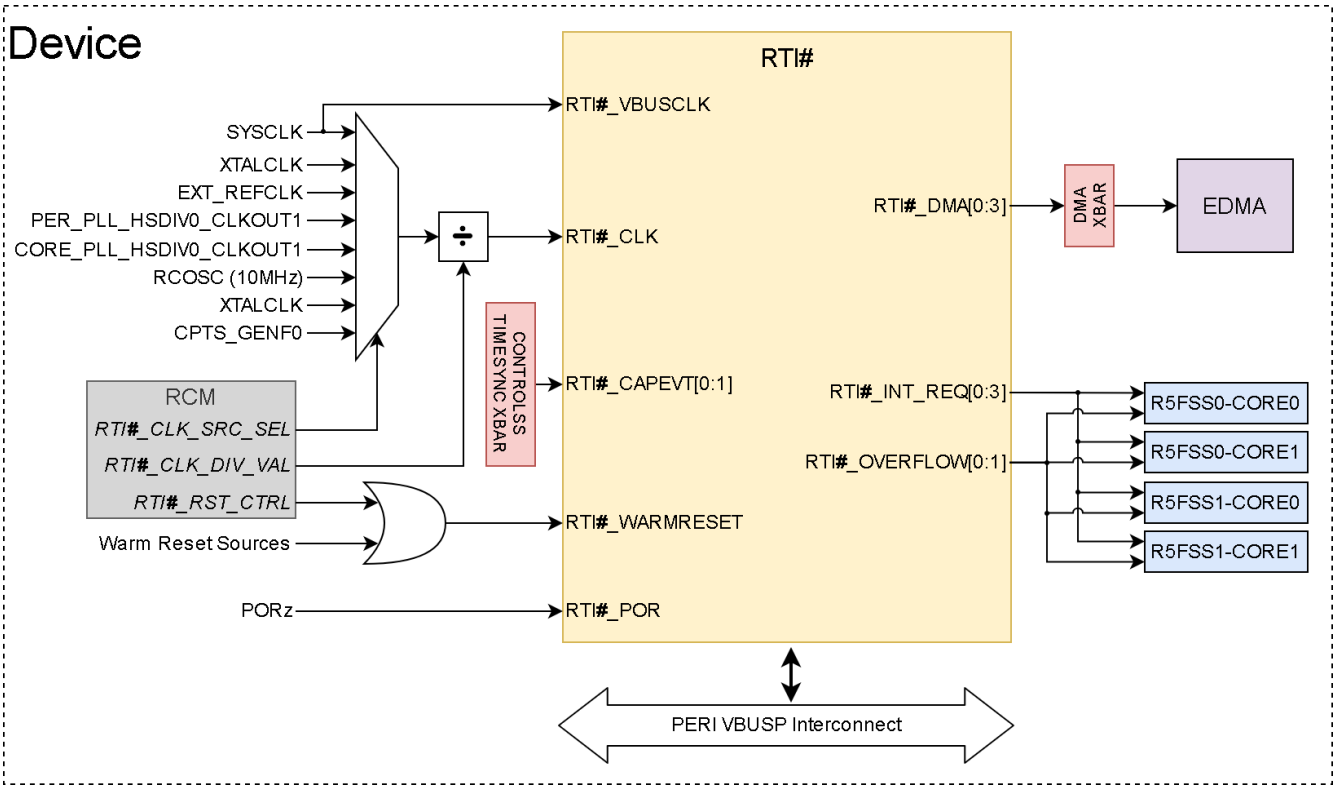


Figure 4-23. RTI Integration

The tables below summarize the integration of RTI# (where # = 0, 1, 2, 3, 4, 5, 6, 7) in the device.

Each RTI# instance is supplied by dedicated RTICLK# mux.

Table 4-55. *RTI* Device Integration

This table describes the module device integration details.

Module Instance	Device Allocation	SoC Interconnect
RTI0	✓	VBUSP CORE Interconnect
RTI1	✓	VBUSP CORE Interconnect
RTI2	✓	VBUSP CORE Interconnect
RTI3	✓	VBUSP CORE Interconnect
RTI4	✓	VBUSP CORE Interconnect
RTI5	✓	VBUSP CORE Interconnect
RTI6	✓	VBUSP CORE Interconnect
RTI7	✓	VBUSP CORE Interconnect

Table 4-56. *RTI* Clocks

This table table describes the module clocking signals.

Module Instance	Module Clock Input	Source Clock Signal	Source	Default Freq	Description
RTI0	RTI0_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	RTI0 VBUSP Interface Clock
	RTI0_FCLK (RTI_CLK)	XTALCLK	External XTAL	25 MHz	RTI0 Functional Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT1	PLL_CORE_CLK: HSDIV0_CLKOUT1	500 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
		CTPS_GENF0	CPSW3G CPTS GENF0 Clock	50 MHz	
RTI1	RTI1_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	RTI1 VBUSP Interface Clock
	RTI1_FCLK (RTI_CLK)	XTALCLK	External XTAL	25 MHz	RTI1 Functional Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT1	PLL_CORE_CLK: HSDIV0_CLKOUT1	500 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
		CTPS_GENF0	CPSW3G CPTS GENF0 Clock	50 MHz	

Table 4-56. *RTI* Clocks (continued)

This table table describes the module clocking signals.

Module Instance	Module Clock Input	Source Clock Signal	Source	Default Freq	Description
RTI2	RTI2_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	RTI2 VBUSP Interface Clock
	RTI2_FCLK (RTI_CLK)	XTALCLK	External XTAL	25 MHz	RTI2 Functional Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT1	PLL_CORE_CLK: HSDIV0_CLKOUT1	500 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
		CTPS_GENF0	CPSW3G CPTS GENF0 Clock	50 MHz	
RTI3	RTI3_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	RTI3 VBUSP Interface Clock
	RTI3_FCLK (RTI_CLK)	XTALCLK	External XTAL	25 MHz	RTI3 Functional Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT1	PLL_CORE_CLK: HSDIV0_CLKOUT1	500 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
		CTPS_GENF0	CPSW3G CPTS GENF0 Clock	50 MHz	

Table 4-56. *RTI* Clocks (continued)

This table table describes the module clocking signals.

Module Instance	Module Clock Input	Source Clock Signal	Source	Default Freq	Description
RTI4	RTI4_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	RTI4 VBUSP Interface Clock
	RTI4_FCLK (RTI_CLK)	XTALCLK	External XTAL	25 MHz	RTI4 Functional Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT1	PLL_CORE_CLK: HSDIV0_CLKOUT1	500 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
RTI5	RTI5_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	RTI5 VBUSP Interface Clock
	RTI5_FCLK (RTI_CLK)	XTALCLK	External XTAL	25 MHz	RTI5 Functional Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT1	PLL_CORE_CLK: HSDIV0_CLKOUT1	500 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
RTI6	RTI6_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	RTI6 VBUSP Interface Clock
	RTI6_FCLK (RTI_CLK)	XTALCLK	External XTAL	25 MHz	RTI6 Functional Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT1	PLL_CORE_CLK: HSDIV0_CLKOUT1	500 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	

Table 4-56. *RTI* Clocks (continued)

This table table describes the module clocking signals.

Module Instance	Module Clock Input	Source Clock Signal	Source	Default Freq	Description
RTI7	RTI7_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	RTI7 VBUSP Interface Clock
	RTI7_FCLK (RTI_CLK)	XTALCLK	External XTAL	25 MHz	RTI7 Functional Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT1	PLL_CORE_CLK: HSDIV0_CLKOUT1	500 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	

Table 4-57. *RTI* Resets

This table describes the module reset signals.

Module Instance	Module Reset Input	Source Reset Signal	Source	Description
RTI0	RTI0_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	RTI0 Asynchronous Reset
	RTI0_POR_RST	POR Reset (MOD_POR_RST)	Device Power-On Reset	RTI0 Power-On Reset
RTI1	RTI1_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	RTI1 Asynchronous Reset
	RTI1_POR_RST	POR Reset (MOD_POR_RST)	Device Power-On Reset	RTI1 Power-On Reset
RTI2	RTI2_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	RTI2 Asynchronous Reset
	RTI2_POR_RST	POR Reset (MOD_POR_RST)	Device Power-On Reset	RTI2 Power-On Reset
RTI3	RTI3_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	RTI3 Asynchronous Reset
	RTI3_POR_RST	POR Reset (MOD_POR_RST)	Device Power-On Reset	RTI3 Power-On Reset
RTI4	RTI4_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	RTI4 Asynchronous Reset
	RTI4_POR_RST	POR Reset (MOD_POR_RST)	Device Power-On Reset	RTI4 Power-On Reset
RTI5	RTI5_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	RTI5 Asynchronous Reset
	RTI5_POR_RST	POR Reset (MOD_POR_RST)	Device Power-On Reset	RTI5 Power-On Reset
RTI6	RTI6_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	RTI6 Asynchronous Reset
	RTI6_POR_RST	POR Reset (MOD_POR_RST)	Device Power-On Reset	RTI6 Power-On Reset

Table 4-57. *RTI* Resets (continued)

This table describes the module reset signals.

Module Instance	Module Reset Input	Source Reset Signal	Source	Description
RTI7	RTI7_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	RTI7 Asynchronous Reset
	RTI7_POR_RST	POR Reset (MOD_POR_RST)	Device Power-On Reset	RTI7 Power-On Reset

Table 4-58. *RTI* Interrupt Requests

This table describes the module interrupt requests.

Module Instance	Module Interrupt Signal	Destination Interrupt Input	Destination	Type	Description
RTI0	RTI0_INT_REQ_0	RTI0_INT_REQ_0	ALL R5FSS Cores	Pulse	RTI0 Status Event Interrupt
	RTI0_INT_REQ_1	RTI0_INT_REQ_1			
	RTI0_INT_REQ_2	RTI0_INT_REQ_2			
	RTI0_INT_REQ_3	RTI0_INT_REQ_3			
	RTI0_OVL_REQ_0	RTI0_OVERFLOW_LEVEL_0			RTI0 Counter Overflow Event Interrupt
	RTI0_OVL_REQ_1	RTI0_OVERFLOW_LEVEL_1			
RTI1	RTI1_INT_REQ_0	RTI1_INT_REQ_0	ALL R5FSS Cores	Pulse	RTI1 Status Event Interrupt
	RTI1_INT_REQ_1	RTI1_INT_REQ_1			
	RTI1_INT_REQ_2	RTI1_INT_REQ_2			
	RTI1_INT_REQ_3	RTI1_INT_REQ_3			
	RTI1_OVL_REQ_0	RTI1_OVERFLOW_LEVEL_0			RTI1 Counter Overflow Event Interrupt
	RTI1_OVL_REQ_1	RTI1_OVERFLOW_LEVEL_1			
RTI2	RTI2_INT_REQ_0	RTI2_INT_REQ_0	ALL R5FSS Cores	Pulse	RTI2 Status Event Interrupt
	RTI2_INT_REQ_1	RTI2_INT_REQ_1			
	RTI2_INT_REQ_2	RTI2_INT_REQ_2			
	RTI2_INT_REQ_3	RTI2_INT_REQ_3			
	RTI2_OVL_REQ_0	RTI2_OVERFLOW_LEVEL_0			RTI2 Counter Overflow Event Interrupt
	RTI2_OVL_REQ_1	RTI2_OVERFLOW_LEVEL_1			
RTI3	RTI3_INT_REQ_0	RTI3_INT_REQ_0	ALL R5FSS Cores	Pulse	RTI3 Status Event Interrupt
	RTI3_INT_REQ_1	RTI3_INT_REQ_1			
	RTI3_INT_REQ_2	RTI3_INT_REQ_2			
	RTI3_INT_REQ_3	RTI3_INT_REQ_3			
	RTI3_OVL_REQ_0	RTI3_OVERFLOW_LEVEL_0			RTI3 Counter Overflow Event Interrupt
	RTI3_OVL_REQ_1	RTI3_OVERFLOW_LEVEL_1			
RTI4	RTI4_INT_REQ_0	RTI4_INT_REQ_0	ALL R5FSS Cores	Pulse	RTI4 Status Event Interrupt
	RTI4_INT_REQ_1	RTI4_INT_REQ_1			
	RTI4_INT_REQ_2	RTI4_INT_REQ_2			
	RTI4_INT_REQ_3	RTI4_INT_REQ_3			
	RTI4_OVL_REQ_0	RTI4_OVERFLOW_LEVEL_0			RTI4 Counter Overflow Event Interrupt
	RTI4_OVL_REQ_1	RTI4_OVERFLOW_LEVEL_1			

Table 4-58. *RTI* Interrupt Requests (continued)

This table describes the module interrupt requests.

Module Instance	Module Interrupt Signal	Destination Interrupt Input	Destination	Type	Description
RTI5	RTI5_INT_REQ_0	RTI5_INT_REQ_0	ALL R5FSS Cores	Pulse	RTI5 Status Event Interrupt
	RTI5_INT_REQ_1	RTI5_INT_REQ_1			
	RTI5_INT_REQ_2	RTI5_INT_REQ_2			
	RTI5_INT_REQ_3	RTI5_INT_REQ_3			
	RTI5_OVL_REQ_0	RTI5_OVERFLOW_LEVEL_0			RTI5 Counter Overflow Event Interrupt
	RTI5_OVL_REQ_1	RTI5_OVERFLOW_LEVEL_1			
RTI6	RTI6_INT_REQ_0	RTI6_INT_REQ_0	ALL R5FSS Cores	Pulse	RTI6 Status Event Interrupt
	RTI6_INT_REQ_1	RTI6_INT_REQ_1			
	RTI6_INT_REQ_2	RTI6_INT_REQ_2			
	RTI6_INT_REQ_3	RTI6_INT_REQ_3			
	RTI6_OVL_REQ_0	RTI6_OVERFLOW_LEVEL_0			RTI6 Counter Overflow Event Interrupt
	RTI6_OVL_REQ_1	RTI6_OVERFLOW_LEVEL_1			
RTI7	RTI7_INT_REQ_0	RTI7_INT_REQ_0	ALL R5FSS Cores	Pulse	RTI7 Status Event Interrupt
	RTI7_INT_REQ_1	RTI7_INT_REQ_1			
	RTI7_INT_REQ_2	RTI7_INT_REQ_2			
	RTI7_INT_REQ_3	RTI7_INT_REQ_3			
	RTI7_OVL_REQ_0	RTI7_OVERFLOW_LEVEL_0			RTI7 Counter Overflow Event Interrupt
	RTI7_OVL_REQ_1	RTI7_OVERFLOW_LEVEL_1			

Table 4-59. *RTI* DMA Requests

This table describes the module DMA requests.

Module Instance	Module DMA Event	Destination DMA Event Input	Destination	Type	Description
RTI0	RTI0_DMA_0	RTI0_DMA_REQ_0	EDMA Crossbar (EDMA_XBAR)	Pulse	RTI0 DMA Request
	RTI0_DMA_1	RTI0_DMA_REQ_1			
	RTI0_DMA_2	RTI0_DMA_REQ_2			
	RTI0_DMA_3	RTI0_DMA_REQ_3			
RTI1	RTI1_DMA_0	RTI1_DMA_REQ_0			RTI1 DMA Request
	RTI1_DMA_1	RTI1_DMA_REQ_1			
	RTI1_DMA_2	RTI1_DMA_REQ_2			
	RTI1_DMA_3	RTI1_DMA_REQ_3			
RTI2	RTI2_DMA_0	RTI2_DMA_REQ_0			RTI2 DMA Request
	RTI2_DMA_1	RTI2_DMA_REQ_1			
	RTI2_DMA_2	RTI2_DMA_REQ_2			
	RTI2_DMA_3	RTI2_DMA_REQ_3			
RTI3	RTI3_DMA_0	RTI3_DMA_REQ_0			RTI3 DMA Request
	RTI3_DMA_1	RTI3_DMA_REQ_1			
	RTI3_DMA_2	RTI3_DMA_REQ_2			
	RTI3_DMA_3	RTI3_DMA_REQ_3			
RTI4	RTI4_DMA_0	RTI4_DMA_REQ_0			RTI4 DMA Request
	RTI4_DMA_1	RTI4_DMA_REQ_1			
	RTI4_DMA_2	RTI4_DMA_REQ_2			
	RTI4_DMA_3	RTI4_DMA_REQ_3			
RTI5	RTI5_DMA_0	RTI5_DMA_REQ_0			RTI5 DMA Request
	RTI5_DMA_1	RTI5_DMA_REQ_1			
	RTI5_DMA_2	RTI5_DMA_REQ_2			
	RTI5_DMA_3	RTI5_DMA_REQ_3			
RTI6	RTI6_DMA_0	RTI6_DMA_REQ_0			RTI6 DMA Request
	RTI6_DMA_1	RTI6_DMA_REQ_1			
	RTI6_DMA_2	RTI6_DMA_REQ_2			
	RTI6_DMA_3	RTI6_DMA_REQ_3			
RTI7	RTI7_DMA_0	RTI7_DMA_REQ_0			RTI7 DMA Request
	RTI7_DMA_1	RTI7_DMA_REQ_1			
	RTI7_DMA_2	RTI7_DMA_REQ_2			
	RTI7_DMA_3	RTI7_DMA_REQ_3			

Table 4-60. *RTI* Capture Events

This table describes the module capture events.

Module Instance	Module Capture Event Input	Capture Event Source Signal	Source	Type	Description
RTI0	RTI0_CAPEVT_0	SoC_TIMESYNC_XBAROUT_2	SoC Time Sync Crossbar (TIMESYNC_XBAR)	Pulse	RTI0 Counter Capture Input Event
	RTI0_CAPEVT_1	SoC_TIMESYNC_XBAROUT_3			
RTI1	RTI1_CAPEVT_0	SoC_TIMESYNC_XBAROUT_4			RTI1 Counter Capture Input Event
	RTI1_CAPEVT_1	SoC_TIMESYNC_XBAROUT_5			
RTI2	RTI2_CAPEVT_0	SoC_TIMESYNC_XBAROUT_6			RTI2 Counter Capture Input Event
	RTI2_CAPEVT_1	SoC_TIMESYNC_XBAROUT_7			
RTI3	RTI3_CAPEVT_0	SoC_TIMESYNC_XBAROUT_8			RTI3 Counter Capture Input Event
	RTI3_CAPEVT_1	SoC_TIMESYNC_XBAROUT_9			
RT4	RTI3_CAPEVT_0	SoC_TIMESYNC_XBAROUT_12			RTI4 Counter Capture Input Event
	RTI3_CAPEVT_1	SoC_TIMESYNC_XBAROUT_13			
RTI5	RTI3_CAPEVT_0	SoC_TIMESYNC_XBAROUT_14			RTI5 Counter Capture Input Event
	RTI3_CAPEVT_1	SoC_TIMESYNC_XBAROUT_15			
RTI6	RTI3_CAPEVT_0	SoC_TIMESYNC_XBAROUT_16			RTI6 Counter Capture Input Event
	RTI3_CAPEVT_1	SoC_TIMESYNC_XBAROUT_17			
RTI7	RTI3_CAPEVT_0	SoC_TIMESYNC_XBAROUT_18			RTI7 Counter Capture Input Event
	RTI3_CAPEVT_1	SoC_TIMESYNC_XBAROUT_19			

Note

For more information on the interconnects, see the *System Interconnect* chapter.

For more information on the power, reset and clock management, see the corresponding sections within the *Device Configuration* chapter.

For more information on the device interrupt controllers, see the *Interrupt Controllers* chapter.

4.21 WWDT Integration

There are 4x WWDT modules integrated in the device. The diagram and tables below show the device integration details.

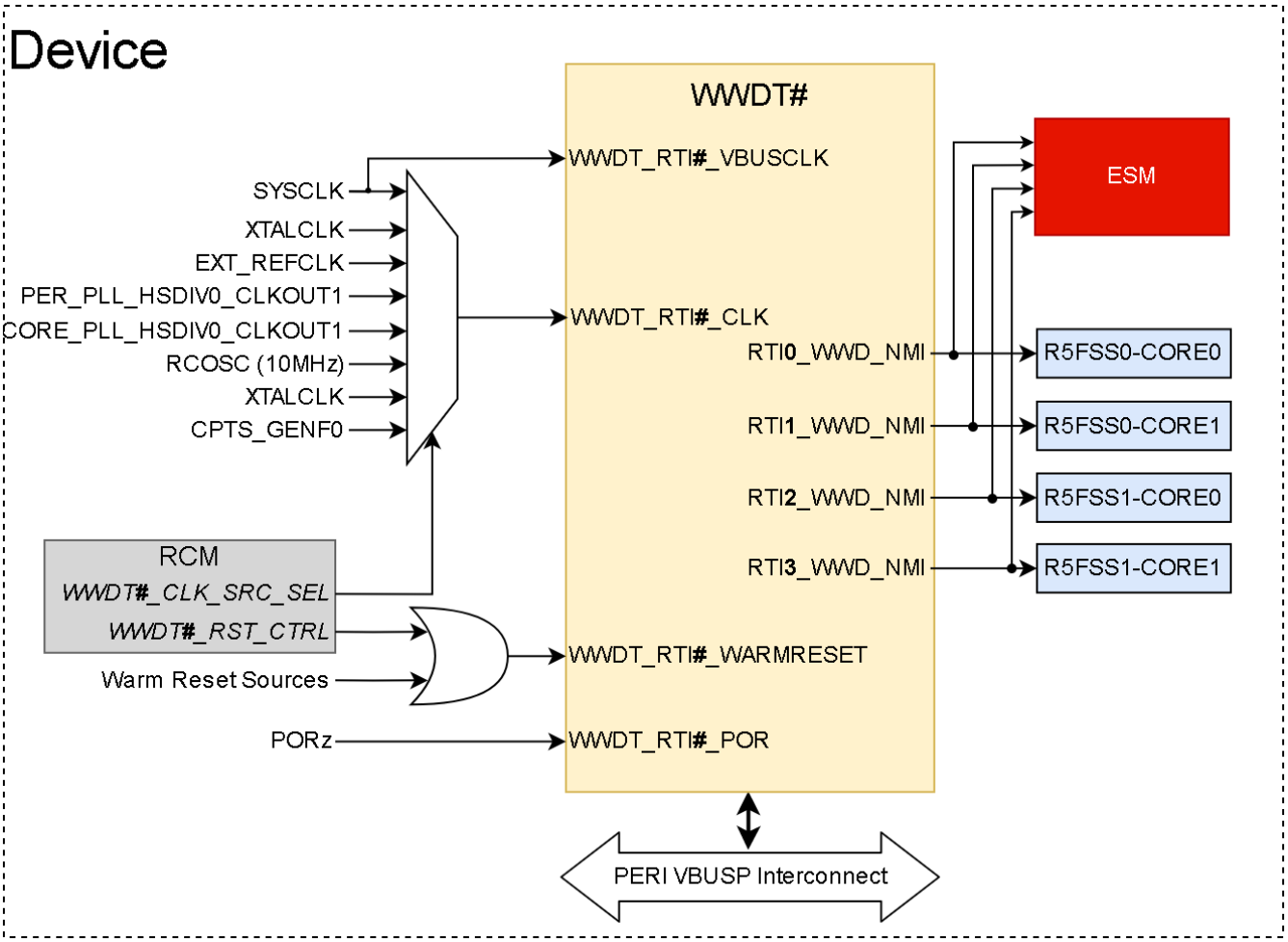


Figure 4-24. WWDT Integration

The tables below summarize the integration of WWDT# (where # = 0, 1, 2, 3) in the device.

Each WWDT# instance is supplied by dedicated WWDTCLK# mux.

Table 4-61. *WWDT* Device Integration

This table describes the module device integration details.

Module Instance	Device Allocation	SoC Interconnect
WWDT0	✓	VBUSP CORE Interconnect
WWDT1	✓	VBUSP CORE Interconnect
WWDT2	✓	VBUSP CORE Interconnect
WWDT3	✓	VBUSP CORE Interconnect

Table 4-62. *WWDT* Clocks

This table describes the module clocking signals.

Module Instance	Module Clock Input	Source Clock Signal	Source	Default Freq	Description
WWDT0	WWDT0_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	WWDT0 VBUSP Interface Clock
	WWDT0_FCLK (WWDT_CLK)	XTALCLK	External Crystal (XTAL)	25 MHz	WWDT0 Functional Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT1	PLL_CORE_CLK: HSDIV0_CLKOUT1	500 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External Crystal (XTAL)	25 MHz	
		CTPS_GENF0	CPSW3G CPTS GENF0 Clock	50 MHz	
WWDT1	WWDT1_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	WWDT1 VBUSP Interface Clock
	WWDT1_FCLK (WWDT_CLK)	XTALCLK	External Crystal (XTAL)	25 MHz	WWDT1 Functional Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT1	PLL_CORE_CLK: HSDIV0_CLKOUT1	500 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External Crystal (XTAL)	25 MHz	
		CTPS_GENF0	CPSW3G CPTS GENF0 Clock	50 MHz	

Table 4-62. *WWDT* Clocks (continued)

This table describes the module clocking signals.

Module Instance	Module Clock Input	Source Clock Signal	Source	Default Freq	Description
WWDT2	WWDT2_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	WWDT2 VBUSP Interface Clock
	WWDT2_FCLK (WWDT_CLK)	XTALCLK	External Crystal (XTAL)	25 MHz	WWDT2 Functional Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT1	PLL_CORE_CLK: HSDIV0_CLKOUT1	500 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External Crystal (XTAL)	25 MHz	
		CTPS_GENF0	CPSW3G CPTS GENF0 Clock	50 MHz	
WWDT3	WWDT3_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	WWDT3 VBUSP Interface Clock
	WWDT3_FCLK (WWDT_CLK)	XTALCLK	External Crystal (XTAL)	25 MHz	WWDT3 Functional Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT1	PLL_CORE_CLK: HSDIV0_CLKOUT1	500 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External Crystal (XTAL)	25 MHz	
		CTPS_GENF0	CPSW3G CPTS GENF0 Clock	50 MHz	

Table 4-63. *WWDT* Resets

This table describes the module reset signals.

Module Instance	Module Reset Input	Source Reset Signal	Source	Description
WWDT0	WWDT0_RST	Warm Reset (MOD_G_RST)	RCM Reset Control Register + Warm Reset Sources	WWDT0 Asynchronous Reset
	WWDT0_POR_RST	POR Reset (MOD_POR_RST)	Device Power-On Reset	WWDT0 Power-On Reset

Table 4-63. *WWDT* Resets (continued)

This table describes the module reset signals.

Module Instance	Module Reset Input	Source Reset Signal	Source	Description
WWDT1	WWDT1_RST	Warm Reset (MOD_G_RST)	RCM Reset Control Register + Warm Reset Sources	WWDT1 Asynchronous Reset
	WWDT1_POR_RST	POR Reset (MOD_POR_RST)	Device Power-On Reset	WWDT1 Power-On Reset
WWDT2	WWDT2_RST	Warm Reset (MOD_G_RST)	RCM Reset Control Register + Warm Reset Sources	WWDT2 Asynchronous Reset
	WWDT2_POR_RST	POR Reset (MOD_POR_RST)	Device Power-On Reset	WWDT2 Power-On Reset
WWDT3	WWDT3_RST	Warm Reset (MOD_G_RST)	RCM Reset Control Register + Warm Reset Sources	WWDT3 Asynchronous Reset
	WWDT3_POR_RST	POR Reset (MOD_POR_RST)	Device Power-On Reset	WWDT3 Power-On Reset

Table 4-64. *WWDT* Interrupt Requests

This table describes the module interrupt requests.

Module Instance	Module Interrupt Signal	Destination Interrupt Input	Destination	Type	Description
WWDT0	WWDT0_NMI_REQ	ESM0_PLS_IN_0	ESM0	Pulse	WWDT0 Window Watchdog Violation Non-Maskable Interrupt (NMI) Event
		R5FSS0_0_VIM_128	R5FSS0_CORE0		
WWDT1	WWDT1_NMI_REQ	ESM0_PLS_IN_1	ESM0	Pulse	WWDT1 Non-Maskable Interrupt (NMI) Event
		R5FSS0_1_VIM_128	R5FSS0_CORE1		
WWDT2	WWDT2_NMI_REQ	ESM0_PLS_IN_2	ESM0	Pulse	WWDT2 Non-Maskable Interrupt (NMI) Event
		R5FSS1_0_VIM_128	R5FSS1_CORE0		
WWDT3	WWDT3_NMI_REQ	ESM0_PLS_IN_3	ESM0	Pulse	WWDT3 Non-Maskable Interrupt (NMI) Event
		R5FSS1_1_VIM_128	R5FSS1_CORE1		

Note

For more information on the interconnects, see the *System Interconnect* chapter.

For more information on the power, reset and clock management, see the corresponding sections within the *Device Configuration* chapter.

For more information on the device interrupt controllers, see the *Interrupt Controllers* chapter.

4.22 DCC Integration

There are 4x DCC modules integrated in the device. The diagram below provides a visual representation of the device integration details.

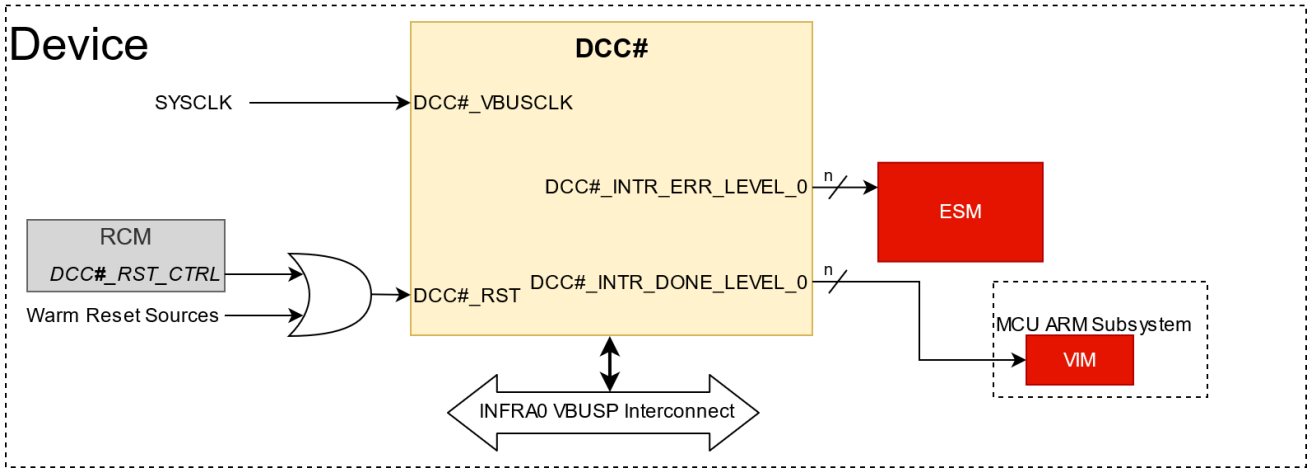


Figure 4-25. DCC Integration Diagram

The tables below summarize the device integration details of DCC.

Table 4-65. DCC Device Integration

This table describes the module device integration details.

Module Instance	Device Allocation	SoC Interconnect
MAIN_DCC0	✓	INFRA0 VBUSP Interconnect
MAIN_DCC1	✓	INFRA0 VBUSP Interconnect
MAIN_DCC2	✓	INFRA0 VBUSP Interconnect
MAIN_DCC3	✓	INFRA0 VBUSP Interconnect

Table 4-66. DCC Clocks

This table describes the module clocking signals.

Module Instance	Module Clock Input	Source Clock Signal	Source	Default Freq	Description
MAIN_DCC0	MAIN_DCC0_CLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	MAIN_DCC0 Interface Clock
MAIN_DCC1	MAIN_DCC1_CLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	MAIN_DCC1 Interface Clock
MAIN_DCC2	MAIN_DCC2_CLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	MAIN_DCC2 Interface Clock
MAIN_DCC3	MAIN_DCC3_CLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	MAIN_DCC3 Interface Clock

Table 4-67. DCC Resets

This table describes the module reset signals.

Module Instance	Module Reset Input	Source Reset Signal	Source	Description
MAIN_DCC0	MAIN_DCC0_RST	Warm Reset (SYNC_RST_N)	RCM + Warm Reset Sources	Synchronous Assertion Reset, Active Low

Table 4-67. DCC Resets (continued)

This table describes the module reset signals.

Module Instance	Module Reset Input	Source Reset Signal	Source	Description
MAIN_DCC1	MAIN_DCC1_RST	Warm Reset (SYNC_RST_N)	RCM + Warm Reset Sources	Synchronous Assertion Reset, Active Low
MAIN_DCC2	MAIN_DCC2_RST	Warm Reset (SYNC_RST_N)	RCM + Warm Reset Sources	Synchronous Assertion Reset, Active Low
MAIN_DCC3	MAIN_DCC3_RST	Warm Reset (SYNC_RST_N)	RCM + Warm Reset Sources	Synchronous Assertion Reset, Active Low

Table 4-68. DCC Interrupt Requests

This table describes the module interrupt requests.

Module Instance	Module Interrupt Signal	Destination Interrupt Input	Destination	Type	Description
MAIN_DCC0	DCC0_done	DCC0_done	ALL R5FSS Cores	Level	DCC Done Interrupt
	DCC0_errror	DCC0_errror	ESM	Level	DCC Error Interrupt
MAIN_DCC1	DCC1_done	DCC1_done	ALL R5FSS Cores	Level	DCC Done Interrupt
	DCC1_error	DCC1_errror	ESM	Level	DCC Error Interrupt
MAIN_DCC2	DCC2_done	DCC2_done	ALL R5FSS Cores	Level	DCC Done Interrupt
	DCC2_errror	DCC2_errror	ESM	Level	DCC Error Interrupt
MAIN_DCC3	DCC3_done	DCC3_done	ALL R5FSS Cores	Level	DCC Done Interrupt
	DCC3_errror	DCC3_errror	ESM	Level	DCC Error Interrupt

Note

For more information on the interconnects, see the *System Interconnect* chapter.

For more information on power, reset, and clock management, see the corresponding sections within the *Device Configuration* chapter.

For more information on the device interrupt controllers, see the *Interrupt Controllers* chapter.

Table 4-69 and *Table 4-70* summarizes the DCC input source clocks in the device.

Table 4-69. DCC0 - DCC1 Input Source Clock Mapping

		MAIN_DCC0												MAIN_DCC1											
		Input0				Input1								Input0				Input1							
		MUX0				MUX1								MUX0				MUX1							
DCC_CLKSRC0 / DCC_CLKSRC1 value:		0	1	2	3	0	1	2	3	4	5	6	7	0	1	2	3	0	1	2	3	4	5	6	7
Clock Source	Input:	CLOCK0[0]	CLOCK0[1]	CLOCK0[2]	CLOCK0[3]	CLOCK1[0]	CLOCK1[1]	CLOCK1[2]	CLOCK1[3]	CLOCK1[4]	CLOCK1[5]	CLOCK1[6]	CLOCK1[7]	CLOCK0[0]	CLOCK0[1]	CLOCK0[2]	CLOCK0[3]	CLOCK1[0]	CLOCK1[1]	CLOCK1[2]	CLOCK1[3]	CLOCK1[4]	CLOCK1[5]	CLOCK1[6]	CLOCK1[7]
XTALCLK	Crystal Clock	✓							✓					✓											
RCCLK10M	Internal 10 MHz RC Oscilator. Always on			✓												✓					✓				
EXT_REFCLK	External Ref Clock		✓							✓					✓										
RCCLK32K	32 KHz RC Clock				✓						✓						✓								
PLL_CORE_CLKOUT (PLL_CORE)																									

Table 4-69. DCC0 - DCC1 Input Source Clock Mapping (continued)

		MAIN_DCC0												MAIN_DCC1											
		Input0				Input1								Input0				Input1							
		MUX0				MUX1								MUX0				MUX1							
DCC_CLKSRC0 / DCC_CLKSRC1 value:		0	1	2	3	0	1	2	3	4	5	6	7	0	1	2	3	0	1	2	3	4	5	6	7
Clock Source	**Input:**	CLOCK0[0]	CLOCK0[1]	CLOCK0[2]	CLOCK0[3]	CLOCK1[0]	CLOCK1[1]	CLOCK1[2]	CLOCK1[3]	CLOCK1[4]	CLOCK1[5]	CLOCK1[6]	CLOCK1[7]	CLOCK0[0]	CLOCK0[1]	CLOCK0[2]	CLOCK0[3]	CLOCK1[0]	CLOCK1[1]	CLOCK1[2]	CLOCK1[3]	CLOCK1[4]	CLOCK1[5]	CLOCK1[6]	CLOCK1[7]
DPLL_CORE_HSDIV0_CLKOUT0	Root clock for Processor SS and Interconnect (Not Mapped to DCC - coverd by SYS_CLK below)																								
DPLL_CORE_HSDIV0_CLKOUT1	CPSW2G/ICSSG RGMII/GMII Clock																		✓						
PLL_PER_CLKOUT (PLL_PER)																									
DPLL_PER_HSDIV0_CLKOUT0	UART 5mbps Clocking																	✓							
DPLL_PER_HSDIV0_CLKOUT1	Peripheral Clocking																			✓					
Other IP Clocks																									
R5SS0_CLK	R5 Cluster 0 Clock						✓																		
R5SS1_CLK	R5 Cluster 1 Clock							✓																	
SYS_CLK	Interconnect System Clock					✓																			
WDT0_CLK	Watch Dog Timer																								
WDT1_CLK	Watch Dog Timer																								
WDT2_CLK	Watch Dog Timer																								
WDT3_CLK	Watch Dog Timer																								
MCAN0_CLK	MCAN Clock																								
MCAN1_CLK	MCAN Clock																								
TEMPSENSE_32K_CLK	32 KHz Clock (divided down from XTALCLK)																								
RMII1_REFCLK	IO Reference Clock Input																								
RMII2_REFCLK	IO Reference Clock Input																								
RGMII1_RXC	IO Receive clock input																								
RGMII2_RXC	IO Receive clock input																								
MII1_RXCLK	IO Receive clock input																								
MII2_RXCLK	IO Receive clock input																								
PR0_MII0_RXCLK	IO Receive clock input																								
PR0_MII1_RXCLK	IO Receive clock input																								
FSI0_RX_CLK	IO Receive clock input																					✓			
FSI1_RX_CLK	IO Receive clock input																						✓		
FSI2_RX_CLK	IO Receive clock input																							✓	
FSI3_RX_CLK	IO Receive clock input																								✓

Table 4-70. DCC2 - DCC3 Input Source Clock Mapping

		MAIN_DCC2											MAIN_DCC3									
		Input0			Input1								Input0			Input1						
		MUX0			MUX1								MUX0			MUX1						
DCC_CLKSRC0 / DCC_CLKSRC1 value:		[0, 3-F]	1	2	0	1	2	3	4	5	6	7	[0, 3-F]	1	2	0	1	2	3	4	5	6	7
Clock Source	Input:	CLOCK0[0]	CLOCK0[1]	CLOCK0[2]	CLOCK1[0]	CLOCK1[1]	CLOCK1[2]	CLOCK1[3]	CLOCK1[4]	CLOCK1[5]	CLOCK1[6]	CLOCK1[7]	CLOCK0[0]	CLOCK0[1]	CLOCK0[2]	CLOCK1[0]	CLOCK1[1]	CLOCK1[2]	CLOCK1[3]	CLOCK1[4]	CLOCK1[5]	CLOCK1[6]	CLOCK1[7]
XTALCLK	Crystal Clock	✓											✓										
RCCLK10M	Internal 10 MHz RC Oscilator. Always on			✓											✓								
EXT_REFCLK	External Ref Clock		✓											✓									
RCCLK32K	32 KHz RC Clock																						
PLL_CORE_CLKOUT (PLL_CORE)																							
DPLL_CORE_HSDIV0_CLKOUT 0	Root clock for Processor SS and Interconnect (Not Mapped to DCC - coverd by SYS_CLK below)																						
DPLL_CORE_HSDIV0_CLKOUT 1	CPSW2G/ICSSG RGMII/GMII Clock																						
PLL_PER_CLKOUT (PLL_PER)																							
DPLL_PER_HSDIV0_CLKOUT0	UART 5mbps Clocking																						
DPLL_PER_HSDIV0_CLKOUT1	Peripheral Clocking																						
Other IP Clocks																							
R5SS0_CLK	R5 Cluster 0 Clock																						
R5SS1_CLK	R5 Cluster 1 Clock																						
SYS_CLK	Interconnect System Clock				✓																		
WDT0_CLK	Watch Dog Timer					✓																	
WDT1_CLK	Watch Dog Timer						✓																
WDT2_CLK	Watch Dog Timer							✓															
WDT3_CLK	Watch Dog Timer								✓														
MCAN0_CLK	MCAN Clock									✓													
MCAN1_CLK	MCAN Clock										✓												
TEMPSENSE_32K_CLK	32 KHz Clock (divided down from XTALCLK)											✓											
RMII1_REFCLK	IO Reference Clock Input															✓							
RMII2_REFCLK	IO Reference Clock Input																✓						
RGMII1_RXC	IO Receive clock input																	✓					
RGMII2_RXC	IO Receive clock input																		✓				
MII1_RXCLK	IO Receive clock input																			✓			
MII2_RXCLK	IO Receive clock input																				✓		
PR0_MII0_RXCLK	IO Receive clock input																					✓	
PR0_MII1_RXCLK	IO Receive clock input																						✓
FSI0_RX_CLK	IO Receive clock input																						
FSI1_RX_CLK	IO Receive clock input																						
FSI2_RX_CLK	IO Receive clock input																						

Table 4-70. DCC2 - DCC3 Input Source Clock Mapping (continued)

		MAIN_DCC2											MAIN_DCC3										
		Input0			Input1								Input0			Input1							
		MUX0			MUX1								MUX0			MUX1							
DCC_CLKSRC0 / DCC_CLKSRC1 value:		[0, 3-F]	1	2	0	1	2	3	4	5	6	7	[0, 3-F]	1	2	0	1	2	3	4	5	6	7
Clock Source	Input:	CLOCK0[0]	CLOCK0[1]	CLOCK0[2]	CLOCK1[0]	CLOCK1[1]	CLOCK1[2]	CLOCK1[3]	CLOCK1[4]	CLOCK1[5]	CLOCK1[6]	CLOCK1[7]	CLOCK0[0]	CLOCK0[1]	CLOCK0[2]	CLOCK1[0]	CLOCK1[1]	CLOCK1[2]	CLOCK1[3]	CLOCK1[4]	CLOCK1[5]	CLOCK1[6]	CLOCK1[7]
FSI3_RX_CLK	IO Receive clock input																						

4.23 ESM Integration

There is 1x ESM integrated in the device. The diagram below provides a visual representation of the device integration details.

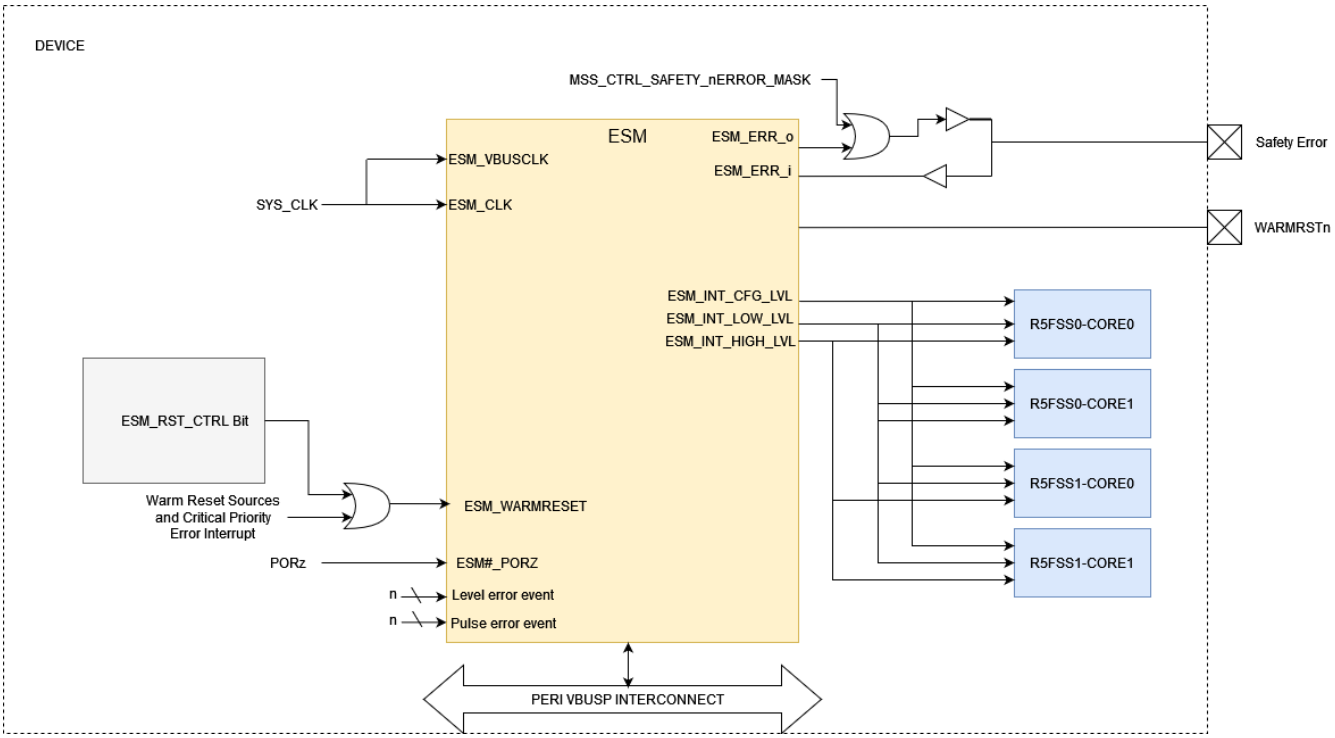


Figure 4-26. ESM Integration Diagram

The tables below summarize the device integration details of ESM.

Table 4-71. *ESM* Device Integration

This table describes the ESM device integration details.

ESM Instance	Device Allocation	SoC Interconnect
ESM0	✓	INFRA0 VBUSP Interconnect

Table 4-72. *ESM* Clocks

This table describes the ESM clocking signals.

ESM Instance	ESM Clock Input	Source Clock Signal	Source	Default Freq	Description
ESM0	ESM0_VBUSCLK	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	ESM0 VBUSP Interface Clock
	ESM0_CLK				ESM0 Functional Clock

Table 4-73. *ESM* Resets

This table describes the ESM reset signals.

ESM Instance	ESM Reset Input	Source Reset Signal	Source	Description
ESM0	ESM0_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	ESM0 Asynchronous Reset
	ESM0_POR_RST	POR Reset (MOD_POR_RST)	Device Power-On Reset	ESM0 Power-On Reset

Table 4-74. *ESM* Interrupt Requests

This table describes the ESM interrupt requests.

ESM Instance	ESM Interrupt Signal	Destination Interrupt Input	Destination	Type	Description
ESM0	ESM0_INT_CFG_LVL_0	ESM0_INT_CFG_LVL	ALL R5FSS Cores	Level	ESM0 Configuration Error Interrupt
	ESM0_INT_LOW_LVL_0	ESM0_INT_LOW_LVL			ESM0 Low Priority Interrupt
	ESM0_INT_HIGH_LVL_0	ESM0_INT_HIGH_LVL			ESM0 High Priority Interrupt
	crit_pri_lvl_intr	crit_pri_lvl_intr			SoC Level Warm Reset (Critical Priority Interrupt)

Note

For more information on the interconnects, see the *System Interconnect* chapter.

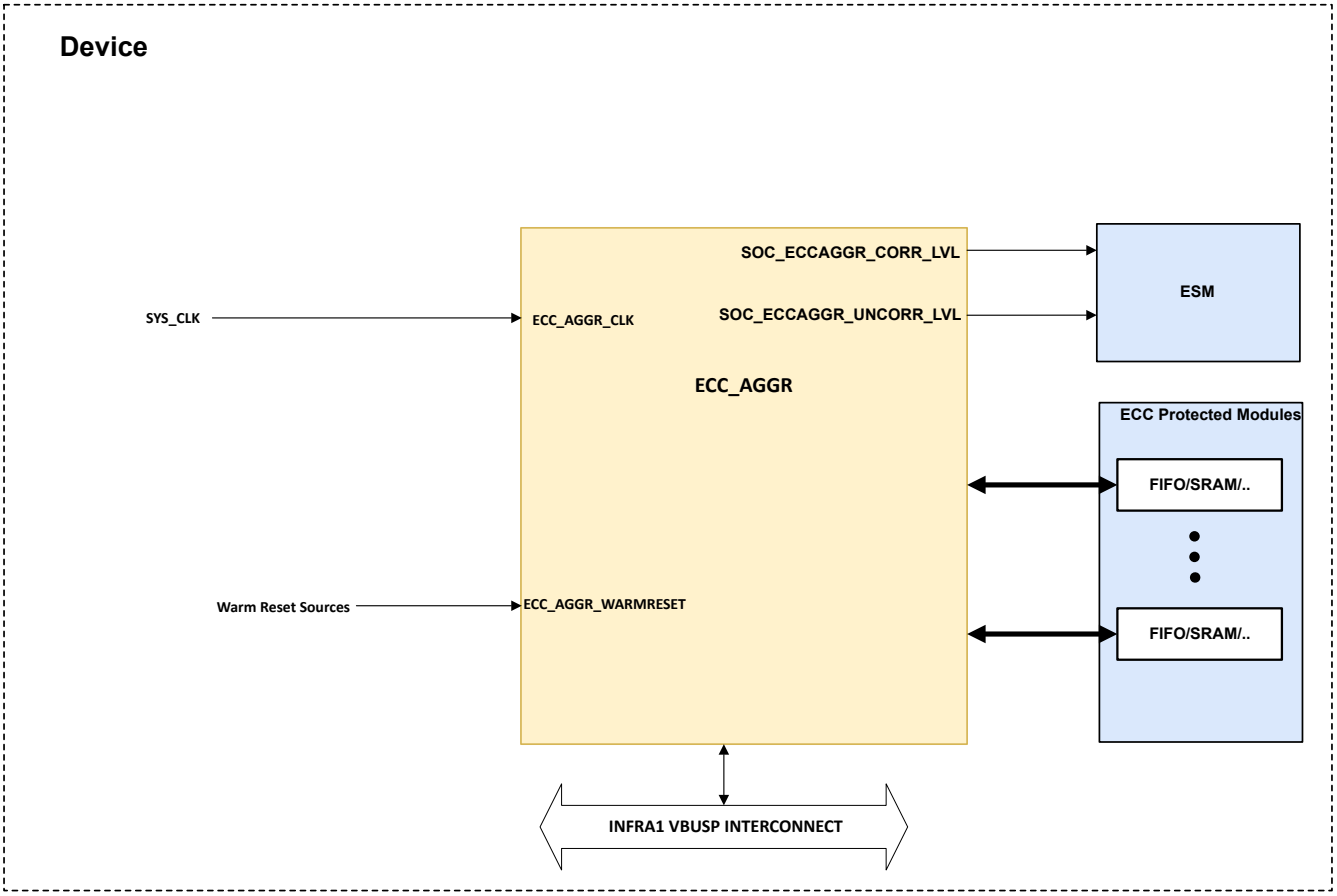
For more information on power, reset, and clock management, see the corresponding sections within the *Device Configuration* chapter.

For more information on the device interrupt controllers, see the *Interrupt Controllers* chapter.

4.24 ECC Aggregator Integration

There is 1x ECC Aggregator integrated in the device. The diagram below provides a visual representation of the device integration details.

Figure 4-27. ECC Aggregator Integration



The tables below summarize the device integration details of ECC Aggregator.

Table 4-75. *ECC Aggregator* Device Integration

This table describes the ECC Aggregator device integration details.

ECC Aggregator Instance	Device Allocation	SoC Interconnect
ECC Aggregator0	✓	INFRA1 VBUSP Interconnect

Table 4-76. *ECC Aggregator* Clocks

This table describes the ECC Aggregator clocking signals.

ECC Aggregator Instance	ECC Aggregator Clock Input	Source Clock Signal	Source	Default Freq	Description
ECC Aggregator0	ECC_AGGR_CLK	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	ECC Aggregator Interface Clock

Table 4-77. *ECC Aggregator* Resets

This table describes the ECC Aggregator reset signals.

ECC Aggregator Instance	ECC Aggregator Reset Input	Source Reset Signal	Source	Description
ECC Aggregator 0	ECC_AGGR_WARMRESET(VBUSP_RSTn)	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	ECC Aggregator0 Asynchronous Reset

Table 4-78. *ECC Aggregator* Event Requests

This table describes the ECC Aggregator interrupt requests.

ECC Aggregator Instance	ECC Aggregator Interrupt Signal	Destination Interrupt Input	Destination	Type	Description
ECC Aggregator0	SOC_ECCAGGR_UNCORR_LVL_0	SOC_ECCAGGR_UNCORR_LVL_0	ESM	Level	ECC Aggregator0 uncorrectable error event
	SOC_ECCAGGR_CORR_LVL_0	SOC_ECCAGGR_CORR_LVL_0			ECC Aggregator0 correctable error event

Table 4-79. *Device modules with ECC Aggregator*

This table describes the ECC Aggregator interrupt requests.

ECC Aggregator	ECC Aggregator Module instances
ECC Aggregator0	L2OCRAM_BANK0
	L2OCRAM_BANK1
	L2OCRAM_BANK2
	L2OCRAM_BANK3
	L2OCRAM_BANK4
	L2OCRAM_BANK5
	MBOX_SRAM
	TPTC00
	TPTC01

Note

For more information on the interconnects, see the *System Interconnect* chapter.

For more information on power, reset, and clock management, see the corresponding sections within the *Device Configuration* chapter.

For more information on the device interrupt controllers, see the *Interrupt Controllers* chapter.

4.25 MCRC Integration

There is 1x MCRC integrated in the device. The diagram below provides a visual representation of the device integration details.

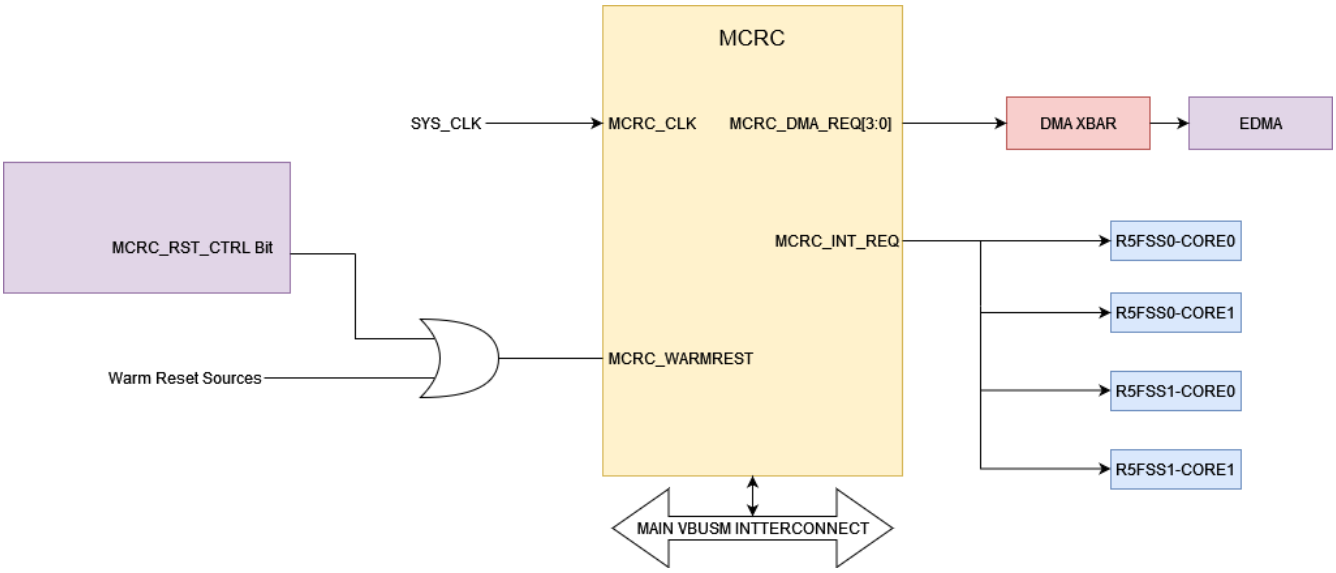


Figure 4-28. MCRC Integration

The tables below summarize the device integration details of MCRC# (where # = 1).

Table 4-80. *MCRC* Device Integration

This table describes the MCRC device integration details.

MCRC Instance	Device Allocation	SoC Interconnect
MCRC0	✓	CORE VBUSM Interconnect

Table 4-81. *MCRC* Clocks

This table describes the MCRC clocking signals.

MCRC Instance	MCRC Clock Input	Source Clock Signal	Source	Default Freq	Description
MCRC0	MCRC_CLK	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	MCRC0 Interface Clock

Table 4-82. *MCRC* Resets

This table describes the MCRC reset signals.

MCRC Instance	MCRC Reset Input	Source Reset Signal	Source	Description
MCRC0	MCRC0_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	MCRC0 Asynchronous Reset

Table 4-83. *MCRC* Interrupt Requests

This table describes the MCRC interrupt requests.

MCRC Instance	MCRC Interrupt Signal	Destination Interrupt Input	Destination	Type	Description
MCRC0	MCRC0_INT_req	MCRC0_INT_req	ALL R5FSS Cores	Level	MCRC0 Event Interrupt

Table 4-84. *MCRC* DMA Requests

This table describes the MCRC DMA requests.

MCRC Instance	MCRC DMA Event	Destination DMA Event Input	Destination	Type	Description
MCRC0	MCRC0_DMA_0	MCRC0_dma_req[0]	EDMA Crossbar (DMA_XBAR)	Pulse	MCRC0 DMA Request
	MCRC0_DMA_1	MCRC0_dma_req[1]			
	MCRC0_DMA_2	MCRC0_dma_req[2]			
	MCRC0_DMA_3	MCRC0_dma_req[3]			

Note

For more information on the interconnects, see the *System Interconnect* chapter.

For more information on power, reset, and clock management, see the corresponding sections within the *Device Configuration* chapter.

For more information on the device interrupt controllers, see the *Interrupt Controllers* chapter.

5 Initialization

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

5.1 Initialization Overview

This section describes different stages involved in initialization, starting from SoC power-on to loading and running an application. Following are the stages involved as shown in the Figure 5-1:

- Hardware Startup Process
- RBL Process
- SBL Process

- **Hardware Startup Process** :
  - **Preinitialization**: Must provide necessary hardware inputs for the device to function i.e., power, clock, control connections and the boot configuration pins. All the control and 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)

Hardware Startup requires an understanding of the process of configuring system interface pins i.e., pads 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 for the proper initialization of the device.

- **RBL (ROM Bootloader) Process**:
  - **R5F ROM**: ROM code running on R5F0 core is responsible for identifying the boot interface, downloading, and executing the Secondary Boot Loader (SBL) software.
  - **HSM ROM**: HSM ROM code runs on M4 core performs image integrity/authentication and it allows or forbids the initial software (SBL) execution.

R5F ROM and HSM ROM primarily focuses on executing the SBL.

ROM Code Overview describes RBL process in detail.

- **SBL (Secondary Bootloader) Process :**
  - **Initial software or SBL:** Primary software responsible for configuring SoC, that loads and passes control to the application software.
  - **HSM RunTime or TIFS-MCU:** Firmware running on secure island i.e Cortex M4. This will enable security services as needed by the application software.
  - **R5F Runtime or Application:** FreeRTOS/ NO RTOS or bare-metal application which runs on main processor(s).

HSM ROM and SBL collectively boot the HSM RunTime, and the SBL and HSM RunTime collectively boot the R5F Run Time/ Application

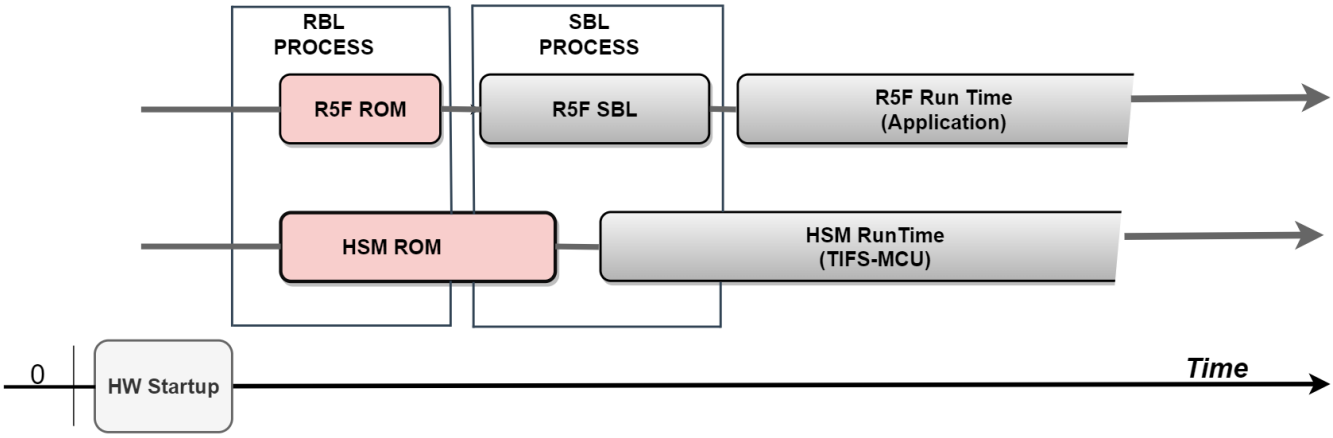


Figure 5-1. Initialization and Boot Process

5.1.1 ROM Code Overview

ROM bootloader (or ROM Code) is a multi-core software that resides in a on-chip read-only memory (ROM) to assist the customer in transferring and executing their SBL and application code. The device has two ROM codes operating in tandem – the Public ROM code (run on R5F core), and the HSM ROM code (run on M4 core).

Figure below gives a pictorial representation of the various stages of the Boot flow. The HSM ROM starts after the power-on sequence where PORz/RSTz is provided cleanly i.e. without any glitches on these pads. HSM ROM assumes R5 core is out of reset and halted. HSM clears R5SS0_COREA_HALT register to un-halt R5. IPC between R5 and HSM is established using messages through dedicated Mailbox RAM , Write/Read and ACK are interrupt based.

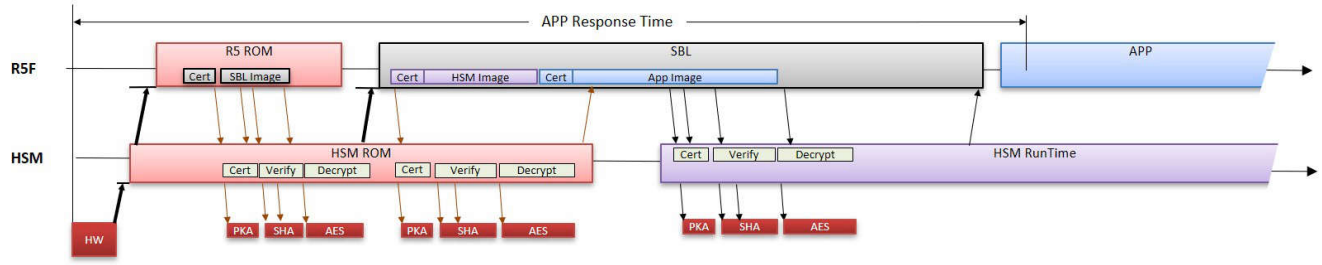


Figure 5-2. Boot Flow

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 UART interface. ROM Code receives the application code on the UART interface and stores it in the internal L2 memory.

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

HSM and R5F_0 will collectively download the SBL image to internal L2 RAM from the external QSPI flash (incase of QSPI boot mode) or the external PC (incase of UART boot mode).

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. HSM 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 which is used later to initialize and startup the PLLs and boot peripherals.

5.1.2 Bootloader Modes

Table 5-1 shows the boot modes supported by ROM code.

Table 5-1. ROM Code Boot Modes

Boot Mode/Peripheral	Boot Media/Host	Notes
OSPI (4S) - Quad Read Mode	OSPI Flash	Download and boot SBL from OSPI flash in quad read mode. Attempt Primary SBL, followed by redundant SBLs if primary loading fails.It supports UART fallback boot mode if redundant SBLs also fail.
UART	External Host	Download and boot SBL from UART interface via XMODEM protocol at 115200bps BaudRate.
OSPI (1S) - Single Read Mode	OSPI Flash	Download and boot SBL from OSPI flash in single read mode. Attempt Primary SBL,followed by redundant SBLs if primary loading fails. It supports UART fallback boot mode if redundant SBLs also fail.
OSPI (8S) - Octal Read Mode	OSPI Flash	Download and boot SBL from OSPI flash in octal read mode. Attempt Primary SBL, followed by redudant SBLs if primary loading fails. It supports UART fallback boot mode if redundant SBLs also fail.
xSPI 8D (SFDP)	OSPI Flash	Read SFDP table for read command, download and boot SBL from OSPI flash in 8D (DDR) mode. Attempt Primary SBL, followed by redundant SBLs if primary loading fails. It supports UART fallback boot mode if redundant SBLs also fail.
DevBoot	N/A	This mode is used for SBL development and JTAG based KeyWriter provision. In this mode, R5 ROM is eclipsed , PLLs are not initialized, PBIST and memory initialization is not done for TCMA,TCMB and L2.

5.1.3 Boot 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.
- **Downloaded software:** Initial software downloaded into on-chip RAM 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 and programs an image in external memories.
- **HS device:** HS-Security device (SoC)
- **HS-FS device:** (HS-Field Securable) - This is the HS device state before the customer keys are provisioned in the device (the state at which HS device leaves TI factory). In this state, secure features are not available and the device protects the ROM code, TI keys and certain security peripherals. In this state, the device does not force authentication for booting.
- **HS-SE device:** (HS-Security Enforced) - This is the HS device state after the customer keys are successfully provisioned in the device. In an HS-SE device, all security features are enabled, all secrets within the device are fully protected, all of the security goals are fully enforced, debug override sequence is supported and the device forces secure booting.
- **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 downloading initial software from external memory to OCSRAM and executing.
- **Controller CPU:** The Arm® Cortex® CPU for which CPU-ID is 0. This core configures the multicore platform and starts the ROM code to boot device from a mass storage memory (memory booting) or a peripheral interface (peripheral booting).
- **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.
- **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.

5.2 Boot Process

5.2.1 Public ROM Code Architecture

The Public ROM code (run on the R5 core) has the following components and is described in the Figure 5-3 diagram:

- Public ROM Entry
- Boot loop
- Modules
- Drivers
- IPC
- Logger

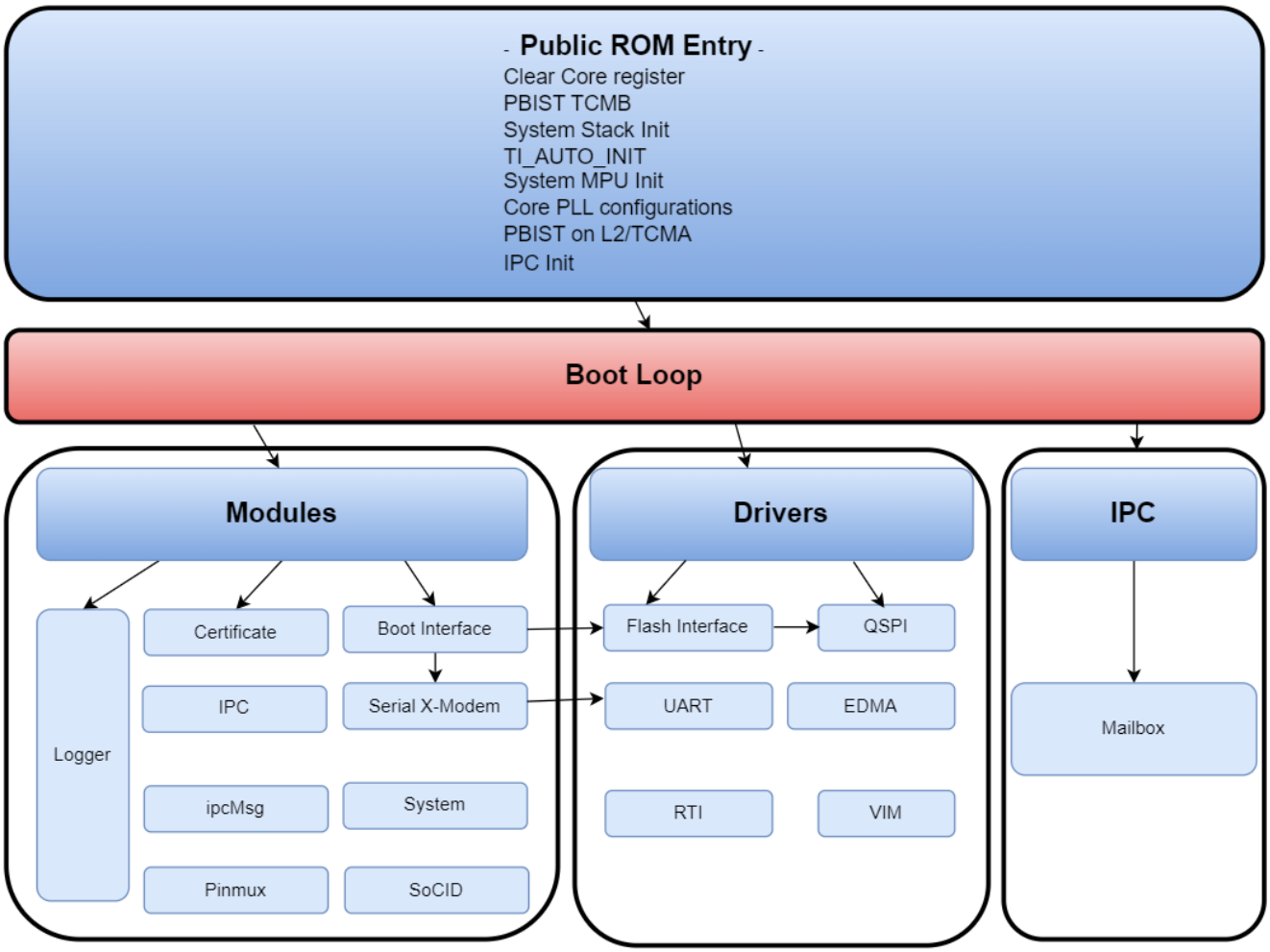


Figure 5-3. Public ROM Code Architecture

5.2.1.1 Public ROM Entry

After the HSM unhalts the R5 core, execution starts at this entry point with the following sequence:

1. Clears core registers
2. Performs PBIST on TCMB
3. Sets up exception and main stack
4. Performs TI auto Init
5. Branch to main()

5.2.1.2 Main Module

The Main module performs the following configurations required for the boot on R5 core before entering the boot loop and then enters the boot loop.

1. System MPU initialization
2. Core PLL initialization (ROM uses only core PLL)
3. Logger module Initialization
4. System Initialization
   - VIM module Initialization
   - RTIA Initialization
5. Performs PBIST on TCMA and L2
6. Performs TCMA and L2 memory initialization
7. Initializes the IPC module (Mbox RAM memInit is done part of it)
8. R5 sends ‘Hello message to HSM’ (R5 core indicates HSM that it is ready for boot)

5.2.1.3 Boot Loop

Boot loop starts with the identification of the boot interface by reading boot-strap pins. The device supports two boot interfaces i.e OSPI and UART. Boot parameters are initialized for the identified interface

OSPI :

- Clock mode : **mode3**
- Clock Frequency : Depends on 1S or 4S or 8S/8D (See the respective section explained later in the chapter)
- Primary flash image address : **0x0** (SeeSection 5.5 in case of redundant SBL image boot)
- Interface support : Supports **fast single, Quad and Octal (SDR and DDR)** read modes only with separate boot pin configuration

UART :

- Baud rate : **115200 bps**
- Parity : **None**
- Data bits : **8**
- Stop bits : **1**
- Flow control : **None**

5.2.1.4 Modules

Modules are the interface between main module and the drivers. Following are the modules present.

- **Boot interface**: Reads the boot mode and identifies the boot interface i.e., UART or OSPI
- **Certificate**: Reads the length of the certificate and image load address
- **Serial x-modem**: Handles x-modem protocol needs while receiving image via UART host
- **System**: Handles VIM and RTIA initializations, provides APIs for timeout handling and interrupt handling
- **ipcMsg**: The IPC Message Layer is used to exchange messages between the R5 and HSM RBL
- **SoCID**: Describes the SOC Identifier data which is exported by the R5 Boot ROM over the supported peripherals
- **Pinmux**: This module is used to configure the peripheral IOs to function for the boot interface
- **Logger**: To log boot info, warnings and errors

5.2.1.5 Drivers

Drivers are the software components which configure the various blocks present in the SoC as per the boot interface selected by the user.

Following are the drivers used:

- **QSPI:** ROM configures QSPI to support fast single Read and Quad Read modes.
- **Flash interface:** Handles flash read APIs for device info and high level APIs for data read
- **UART:** Handles APIs for UART FIFO Write/Read in interrupt mode
- **EDMA:** DMA module to transfer image from flash to internal L2 memory

- **RTI:** Timer module to handle timeouts and logger timestamp
- **VIM:** Vector interrupt module to handle interrupt routines and APIs to configure VIM

5.2.1.6 IPC

R5F boot rom and HSM boot rom communicate via IPC (Inter processor communication) using shared mailbox RAM. The Mailbox architecture is a distributed architecture with the Mailbox memory present in the Receiving processors Subsystem.

Following is the processor numbering:

Processor	Number
R5FSS0 Core0	0
HSM M4	6

Following is the Tx and Rx mailbox addressing:

Mailbox	Address
R5 Tx Mailbox	0x44000000
R5 Rx Mailbox	0x72000000

Following are the mailbox interrupts:

Interrupt Type	Interrupt Line
R5 Mailbox Read Request	136
R5 Mailbox Read Done Acknowledge	137
HSM Mailbox Read Request	0
HSM Mailbox Read Done Acknowledge	40

Mailbox message scheme:

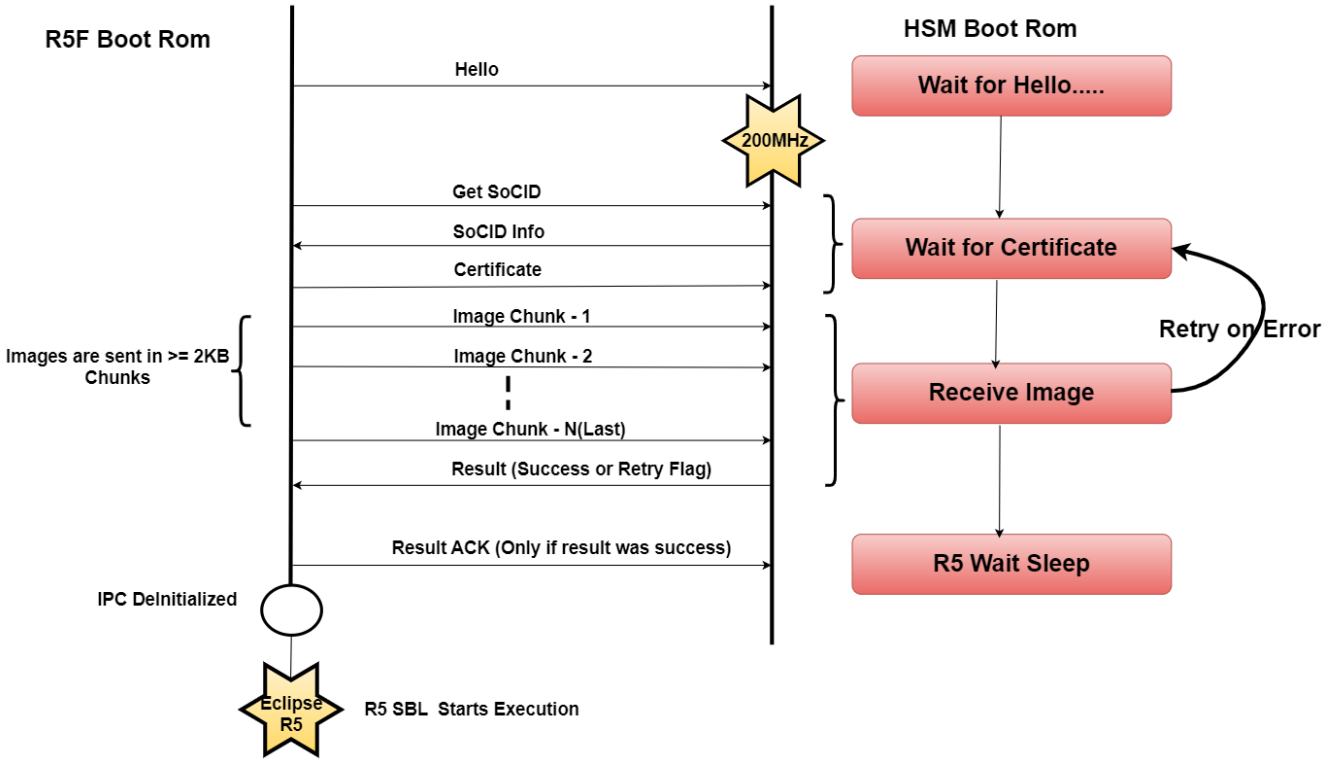
1. PROC_WRITE writes the message in the PROC_READ mailbox
2. PROC_WRITE triggers an interrupt to PROC_READ by writing 1 to **<PROC_WRITE_SS>_CTRL: <PROC_WRITE>_MBOX_WRITE_DONE [PROC_READ]**. Note. It is writing to its own CTRL space.
3. PROC_READ gets a single interrupt for all inter processor communication which is an aggregated interrupt. PROC_READ Reads the register **<PROC_READ_SS>_CTRL::<PROC_READ>_MBOX_READ_REQ** and sees bit [PROC_WRITE] is 0x1
4. PROC_READ Writes to 0x1 to **<PROC_READ_SS>>_CTRL:: <PROC_READ>_MBOX_READ_REQ [PROC_WRITE]** to clear the interrupt.
5. PROC_READ Reads the Message
6. PROC_READ Writes to 0x1 to **<PROC_READ_SS>>_CTRL:: <PROC_READ>_MBOX_READ_DONE_ACK[PROC_WRITE]** to generate an acknowledgement interrupt to PROC_WRITE.
7. PROC_WRITE gets a single interrupt for all inter processor communication which is an aggregated ACK interrupt. PROC_WRITE reads the register **<PROC_WRITE_SS>_CTRL: <PROC_WRITE>_MBOX_READ_DONE** and sees bit [PROC_READ] is 0x1
8. PROC_WRITE writes 0x1 to **<PROC_WRITE_SS>_CTRL: <PROC_WRITE>_MBOX_READ_DONE [PROC_READ]** to clear the interrupt.

The supported messages are as follows:

- IPC_MsgType_HELLO : It's a hello message from R5 to HSM.
- IPC_MsgType_CERT : It's a message type of certificate from R5 to HSM.
- IPC_MsgType_IMAGE : It's a message type of image from R5 to HSM.
- IPC_MsgType_GET_SOC_ID : SOCID message from R5 to HSM for asking SOCID.
- IPC_MsgType_RESULT_ACK : Result acknowledge message from R5 to HSM.

- IPC_MsgType_CANCEL : It's a cancel message from R5 to HSM.
- IPC_MsgType_SOC_ID : SOCID message from HSM to R5 for providing SOCID.
- IPC_MsgType_RESULT : It's a result message from HSM to R5.
- IPC_MsgType_CANCEL_ACK : It's a cancel acknowledge message from HSM to R5.

The message flow between HSM and R5 as follows:



HSM State machine:

- **Wait for Hello…**: After unhalting R5 core, HSM ROM waits for 'Hello…' message from R5 core. R5 ROM starts execution and initializes core PLL and other necessary modules, configures clocks i.e., R5 Core@400MHz and HSM Core@200MHz and then sends the message **IPC_MsgType_HELLO.**
- **Wait for Certificate**: R5 core downloads certificate from the identified boot interface and sends message to HSM i.e**., IPC_MsgType_CERT**. HSM validates the certificate based on the device type.

All the certificate extensions are validated against the above table.

- **Receive Image**: R5 core updates SBL image information in chunks to HSM, chunk size is >= 2KB.

HSM performs the following two operations on the image:

- – SHA512 of the image
  - Image hash is calculated on the chunks received, and after receiving entire image the computed HASH is compared with hash present in the certificate
  - – Image Decryption
    - Decryption of the image is optional. If certificate is enabled with decryption, decryption will start only after certificate verification and image integrity checks are passed.
- **R5 wait Sleep** :
  - – HSM checks for the valid certificate and the image
  - – On successful validation of the certificate and the image, HSM ROM will eclipse R5 ROM and issues R5 core reset, then **SBL starts execution from 0x0**
  - – In case of any failures observed with the certificate or image validation , HSM retries the boot, state machine jumps to Wait for certificate state.

Note : Refer to section R5 SBL Handoff for more details

5.3 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 *Boot Parameter Tables* for table list and description.

Boot mode pins can be divided into the following categories:

- **BOOTMODE[3:0]** – Select the requested boot (primary) mode after POR, that is, the peripheral/memory to boot from.

Note

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

5.3.1 BOOTMODE Pin Mapping

The ROM execution is directed through the main boot mode pins. This provides flexibility through additional booting peripherals. The device must be powered and functional.

Main boot mode pins are shown in Table 5-2.

Any Bootmode pins marked as Reserved or not used must be tied high or low with pull resistors. They should not be left floating.

Table 5-2. BOOTMODE Pin Mapping

Boot Mode	SPI0_D0_pad (SOP3)	SPI0_CLK_pad (SOP2)	QSPI_D1 (SOP1)	QSPI_D0 (SOP0)
OSPI (4S) - Quad Read Mode	0	0	0	0
UART	0	0	0	1
OSPI (1S) - Single Read Mode	0	0	1	0
OSPI (8S) - Octal Read Mode	0	0	1	1
DevBoot	1	0	1	1
xSPI 8D (SFDP)	1	1	0	0
Unsupported Boot Mode	All other combinations not defined above			

5.4 Boot Modes

5.4.1 OSPI Boot

The following apply to all or multiple boot modes that are OSPI-related.

Note

When using a OSPI\SPI flash device greater than 128 Mb, a flash device package with a RESET signal must be used. The reason is that the ROM only uses 3 byte addressing mode (address is 24-bits). To address the full memory address range, software typically switches to 4-byte addressing mode. If a reset to the processor occurs (for example, due to a warm reset), the ROM executes expecting 3-byte addressing mode, but the flash will have been left in 4-byte addressing mode. For the flash device to return to 3-byte addressing mode, it must be reset using this signal. This typically can be achieved by using the RESET signal on the flash memory device. ROM code does not issue a software reset command.

Refer to the AM263x QSPI Flash Selection Guide for additional details.

5.4.1.1 OSPI (8S) and xSPI (8D)

Refer to Section 5.4.1 for more information about all OSPI boot modes.

Table 5-3 summarizes the QSPI pin configuration done by ROM code for OSPI boot device on port 0.

Table 5-3. OSPI (8S) and xSPI (8D) Boot Pinmux

Package Name	Function Name	Pad Num	Input Override	Input override control	Output override	Output override control	PinMux Mode #	PI	PU/PD Sel	SC1	Gpio Sel	Qual Sel	Input Invert Sel	Safety Override sel	HS Mode	HS Controller
QSPI0_CSn0	OSPI0_CSn0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0
QSPI0_CLK0	OSPI0_CLK	2	0	0	0	0	0	1	0	1	0	0	0	0	0	0
QSPI0_D0	OSPI0_D0	3	0	0	0	0	0	1	0	1	0	0	0	0	0	0
QSPI0_D1	OSPI0_D1	4	0	0	0	0	0	1	0	1	0	0	0	0	0	0
QSPI0_D2	OSPI0_D2	5	0	0	0	0	0	1	0	1	0	0	0	0	0	0
QSPI0_D3	OSPI0_D3	6	0	0	0	0	0	1	0	1	0	0	0	0	0	0
MCAN0_RX	OSPI0_D4	7	0	0	0	0	2	1	0	1	0	0	0	0	0	0
MCAN0_TX	OSPI0_D5	8	0	0	0	0	2	1	0	1	0	0	0	0	0	0
MCAN1_RX	OSPI0_D6	9	0	0	0	0	2	1	0	1	0	0	0	0	0	0
MCAN1_TX	OSPI0_D7	10	0	0	0	0	2	1	0	1	0	0	0	0	0	0

Note

All signals in the table are configured, even though some may not be used by this particular boot mode.

5.4.1.1.1 OSPI (8S) Bootloader Operation

The OSPI protocol is described according to bit-width (1 or 8) and data rate (Single Data Rate (S) or Double Data rate (D)) for the Command/Address/Data segments of the protocol. Device supports the 1S-1S-8S mode of OSPI configuration. The Command and Address issued are 8 bits and 24 bits respectively followed by 8 dummy cycles. The frequency of operation supported is 33 MHz.

OSPI (8S) Module Configuration:

- OSPI has two associated memory regions, the first memory region is dedicated to the configuration port i.e all internal registers can be programmed and serial transfers made from the supported external OSPI flash devices. Configuration region is available at 0x5380_8000 in the SoC address map.
- The second memory region is associated mainly with the memory-mapped port and is used for communication directly with external flash devices, the memory region starts at 0x6000_0000. Code will be copied from this region to internal RAM and then execution starts.
- Serial data clock is derived from the clock source “SYS_CLK” (200MHz). This clock is divided by a factor 6 and results in a 33MHz interface clock.
- MODE3 of OSPI clock mode is used, Clock phase and polarity are set to 1, when data is not being transferred SCK=1, data shifted on falling edge and input on rising edge.

ROM Sequence:

- Command issued by ROM in this mode is 0x8B.
- RBL looks for SBL image at address 0x0000_0000, in case of boot failures due to corrupted image or any other reason, RBL tries to boot with redundant images as explained in Section 5.5 .

Flash dependency:

- RBL does not perform any specific action to detect, reset, or power up the OSPI device. OSPI is assumed to be properly powered and reset completed before every attempt to boot by RBL.

5.4.1.1.1.1 OSPI (8S) Loading Process

OSPI (8S) boot mode is not eXecute-In-Place (XIP). ROM code first copies boot image into on-chip RAM and then executes it.

5.4.1.2 OSPI (4S)

Refer to Section 5.4.1 section for more information about all SPI boot modes.

Table 5-4 summarizes the OSPI pin configuration done by ROM code for OSPI boot device on port 0.

Table 5-4. OSPI (4S) Boot Pinmux

Package Name	Function Name	Ball #	Input Override	Input Override Control	Output Override	Output Override Control	PinMux Mode #	PI	PU/PD Sel	SC1	GPIO Sel	Qual Sel	Input Invert Sel	Safety Override Sel	HS Mode	HS Controller
OSPI0_CSn0	OSPI0_CSn0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0
OSPI0_CLK0	OSPI0_CLK	2	0	0	0	0	0	1	0	1	0	0	0	0	0	0
OSPI0_D0	OSPI0_D0	3	0	0	0	0	0	1	0	1	0	0	0	0	0	0
OSPI0_D1	OSPI0_D1	4	0	0	0	0	0	1	0	1	0	0	0	0	0	0
OSPI0_D2	OSPI0_D2	5	0	0	0	0	0	1	0	1	0	0	0	0	0	0
OSPI0_D3	OSPI0_D3	6	0	0	0	0	0	1	0	1	0	0	0	0	0	0
OSPI_CLKLB	OSPI_CLKLB	145	0	0	0	0	0	1	0	1	0	0	0	0	0	0

Note

All signals in the table will be configured even though some may not be used by this particular boot mode.

5.4.1.2.1 OSPI (4S) Bootloader Operation

Device supports 1S-1S-4S mode of OSPI configuration for fast read operation. This means that command and address are issued in single bit transfer mode and data access occurs in quad bit mode. The Command and Address issued are 8 bits and 24 bits followed by 8 dummy cycles. 50 MHz is the supported frequency of operation.

OSPI (4S) Module Configuration:

- OSPI has two associated memory regions, the first memory region is dedicated to the configuration port i.e all internal registers can be programmed and serial transfers made from the supported external OSPI flash devices. Configuration region is available at 0x5380_8000 in the SoC address map.
- The second memory region is associated mainly with the memory-mapped port and is used for communication directly with external flash devices, the memory region starts at 0x6000_0000. Code will be copied from this region to internal RAM and then execution starts.
- Serial data clock is derived from the clock source “SYS_CLK” (200MHz). This clock is divided by a factor 4 and results in a 50MHz interface clock.
- MODE3 of OSPI clock mode is used, Clock phase and polarity are set to 1, when data is not being transferred SCK=1, data shifted on falling edge and input on rising edge.

ROM Sequence:

- Command issued by ROM in this mode is 0x6B.
- RBL looks for SBL image at address 0x0000_0000, in case of boot failures due to corrupted image or any other reason, RBL tries to boot with redundant image as explained in the section Section 5.5 .

Flash dependency:

- RBL does not perform any specific action to detect, reset, or power up the OSPI device. OSPI is assumed to be properly powered and reset completed before every attempt to boot by RBL.
- RBL also expects the QE bit is SET in non-volatile configuration so that flash is active in quad mode by default after POR.

5.4.1.2.1.1 OSPI (4S) Loading Process

OSPI (4S) boot mode is not eXecute-In-Place (XIP). ROM code first copies boot image into on-chip RAM and then executes it.

5.4.1.3 OSPI (1S)

Table 5-5 summarizes the OSPI pin configuration done by ROM code for OSPI (1S) boot device on port 0.

Table 5-5. OSPI (1S) Boot Pinmux

Package Name	Function Name	Ball #	Input Override	Input Override Control	Output Override	Output Override Control	PinMux Mode #	PI	PU/PD Sel	SC1	GPIO Sel	Qual Sel	Input Invert Sel	Safety Override Sel	HS Mode	HS Controller
OSPI0_CSn0	OSPI0_CSn0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0
OSPI0_CLK0	OSPI0_CLK	2	0	0	0	0	0	1	0	1	0	0	0	0	0	0
OSPI0_D0	OSPI0_D0	3	0	0	0	0	0	1	0	1	0	0	0	0	0	0
OSPI0_D1	OSPI0_D1	4	0	0	0	0	0	1	0	1	0	0	0	0	0	0
OSPI_CLKLB	OSPI_CLKLB	145	0	0	0	0	0	1	0	1	0	0	0	0	0	0

Note

All signals in the table will be configured even though some may not be used by this particular boot mode.

5.4.1.3.1 OSPI (1S) Bootloader Operation

Device supports the 1S-1S-1S mode of OPSI configuration. This means that command, address, and data access are in single bit mode. The Command and Address issued are 8 bits and 24 bits followed by 8 dummy cycles. 50 MHz is the supported frequency of operation.

OSPI (1S) Module Configuration:

- OSPI has two associated memory regions, the first memory region is dedicated to the configuration port i.e all internal registers can be programmed and serial transfers made from the supported external OSPI flash devices. Configuration region is available at 0x5380_8000 in the SoC address map.
- The second memory region is associated mainly with the memory-mapped port and is used for communication directly with external flash devices, the memory region starts at 0x6000_0000. Code will be copied from this region to internal RAM and then execution starts.
- Serial data clock is derived from the clock source “SYS_CLK” (200MHz). This clock is divided by a factor 4 and results in a 50MHz interface clock.
- MODE3 of OSPI clock mode is used, Clock phase and polarity are set to 1, when data is not being transferred SCK=1, data shifted on falling edge and input on rising edge.

ROM Sequence:

- Command issued by ROM in this mode is 0x0B.
- RBL looks for SBL image at address 0x0000_0000, in case of boot failures due to corrupted image or any other reason, RBL tries to boot with redundant image placed as explained in the section Section 5.5 .

Flash Dependency:

- RBL does not perform any specific action to detect, reset, or power up the OSPI device. OSPI is assumed to be properly powered and reset completed before every attempt to boot by RBL.

5.4.1.3.1.1 OSPI (1S) Loading Process

OSPI (1S) boot mode is not eXecute-In-Place (XIP). ROM code first copies boot image into on-chip RAM and then executes it.

5.4.2 UART Boot

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 5-6 summarizes the UART pin configuration done by ROM code for UART host on port 0.

Table 5-6. UART Boot Pinmux

Function Name	Pad Num	Input Override	Input Override Control	Output Override	Output Override Control	Pinmux Sel	PI	PUPD Sel	SC1	Gpio Sel	Qual Sel	Inp Inv Sel	HS Mode	HS Controller
UART0_TXD	28	0	0	0	0	0	1	0	1	0	0	0	0	0
UART0_RXD	27	0	0	0	0	0	1	0	1	0	0	0	0	0

5.4.2.1 UART Bootloader Operation

5.4.2.1.1 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 *UART Boot Device Configuration* and *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. Both 128 and 1024 byte block sizes are supported.

5.4.2.1.2 UART Loading Process

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

5.4.2.1.2.1 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 5-7 and Table 5-8.

Table 5-7. 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 5-8. XMODEM Data Frame Fields

Field	Value	Description
STX	0x02	The start character for 1024-byte CRC data blocks
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 5-9.

Table 5-9. Example of XMODEM Transfer protocol

Transmitter Sends		Receiver Sends
	←	Ping (‘C’)
Frame 1	→	
	←	ACK (or NACK)
Frame 2	→	
	←	ACK (or NACK)

Table 5-9. Example of XMODEM Transfer protocol (continued)

Transmitter Sends		Receiver Sends
EOT	→	
	←	ACK (or NACK)

5.4.2.1.3 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.

5.4.3 DevBoot

This boot mode is useful for the development of SBL and for the JTAG based KeyWriter.

5.5 Redundant boot support

Redundant boot is supported on OSPI flash boot modes. ROM tries to boot SBL at the Region1 address and if it fails to boot, then ROM tries to boot from the locations in the order Region1 → Region2 → Region3 → Region4 until boot is successful.

Table 5-10.

Region	Address
1	**0x0_0000**
2	**0x2_0000**
3	**0x4_0000**
4	**0x6_0000**

Following are the failures which can lead to redundant boot:

1. Certificate corruption, ex. Image size, hash of the image, extension IDs, signature etc.
2. Image corruption, ex. bit corruption, byte corruption due to aging of the flash, external interferences etc.

Note

AM263Px supports max image size of 256KB with redundant image support and 512KB without redundant image support. Successfully booted flash Image offset Address is available at the address **0000x84100**

5.6 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.

Note

This device requires 25MHz XTAL clock source.

The Public ROM code configures PLLs which are required during boot. ROM configures only core PLL during boot. The ROM Core PLL configuration details is as follows:

- InputClockDiv (N) = 0xB
- Multiplier (M) = 0x180
- Divider (N2) = 0x1
- Post-Divider (M2) = 0X1
- Fractional Multiplier (Frac) = 0x0

Using the above values, the PLL output frequency is computed as follows:

- XTAL_IN/(N + 1) = 25/12 = 2.0833 MHz
- (XTAL_IN * M) /(N + 1) = 2.08333 * 384 = 800MHz
- (XTAL_IN * M) /[(N2 +1) * (N + 1)] = 800 / 2 = 400MHz
- 400/ M2 = 400MHz

Note

See ADPLLLJ Module section in the Clocking section of Device configuration chapter for more details on PLL configuration sequence and the PLL output frequency equation.

Where, XTAL_IN is the XTAL Clock source frequency (25MHz)

This Core PLL output (ADPLL0) is used to configure R5 clock = 400MHz and SysClk = 200MHz

5.7 Secure Boot Flow

5.7.1 Overview

The secure boot flow is as depicted in Figure 5-4. The ROM-based secure boot is realized by interactions between the MSS R5F ROM and the HSM ROM. When the secondary bootloader (SBL) and the HSM runtime firmware is brought into the respective (MSS R5F and HSM CM4) cores, it is then the responsibility of the SBL to download the further application images. First, the ROM bootloader downloads the SBL. The SBL begins execution (MSS R5F ROM is eclipsed at this point and ROM services are not available anymore) and in turn invokes an API into HSM ROM to download the HSM runtime firmware. When the HSM runtime firmware is downloaded, the HSM ROM is eclipsed and ROM services not available anymore.

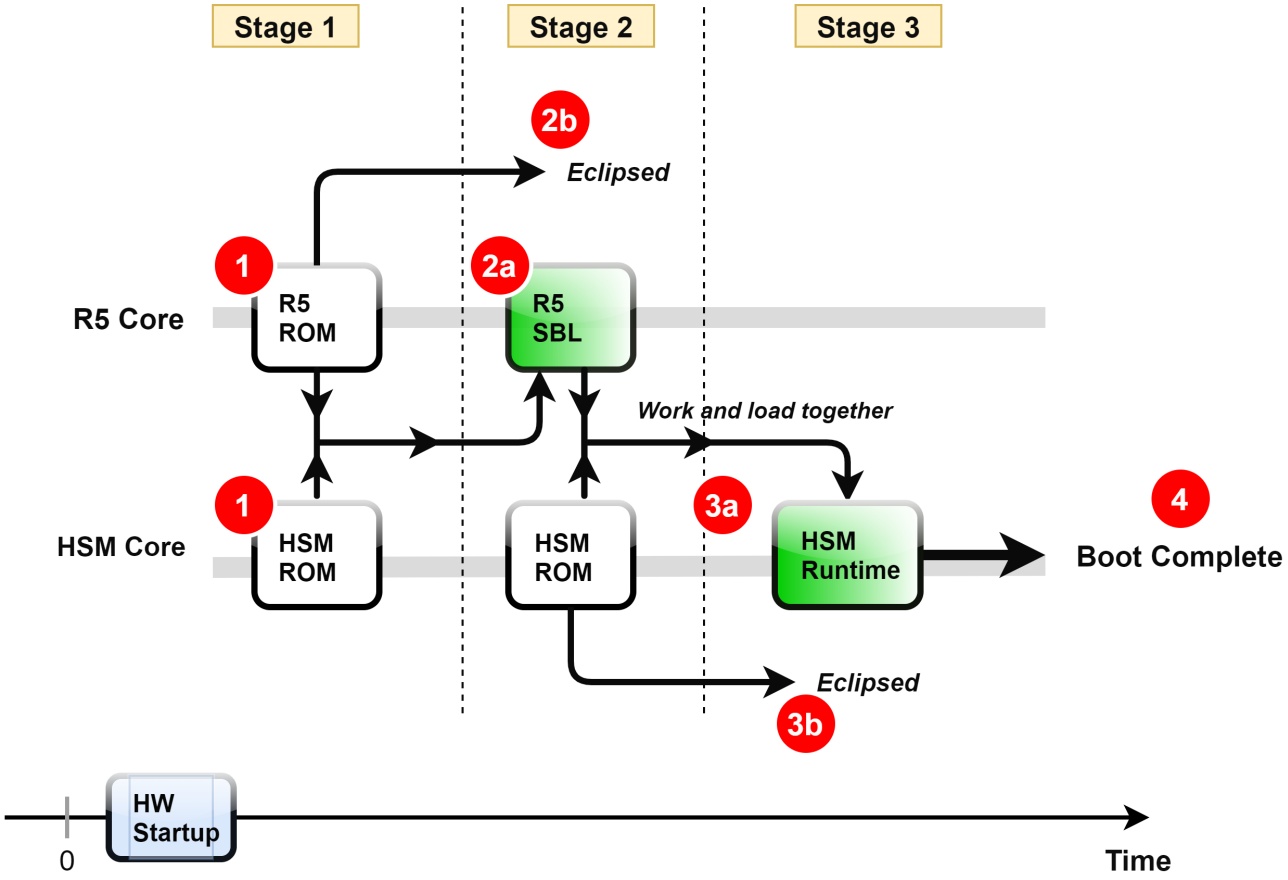


Figure 5-4. Secure Boot Flow

5.7.2 x509 Certificate Structure

The X.509 certificate is defined in Annex A of ITU-T X.509 specification [1]. Certificate is encoded using ASN.1 encoding [2] with DER (Distinguished Encoding Rules) [3]. The main body of the certificate is illustrated below. Essentially, the signed certificate is the concatenation of the certificate itself and the signature. Certificate includes mandatory and optional fields. The supported version shall be v2. V3 and higher versions support is desired but not guaranteed. Various fields of the x509 certificate are as shown below.

```
SIGNATURE ::= SEQUENCE {
            algorithmIdentifier AlgorithmIdentifier{{SupportedAlgorithms}},
            signature BIT STRING,
            ... }
SIGNED{ToBeSigned} ::= SEQUENCE {
            toBeSigned ToBeSigned,
            COMPONENTS OF SIGNATURE,
            ... }
Certificate ::= SIGNED{TBSCertificate}
TBSCertificate ::= SEQUENCE {
            version [0] Version DEFAULT v1,
            serialNumber CertificateSerialNumber,
            signature AlgorithmIdentifier{{SupportedAlgorithms}},
            issuer Name,
            validity Validity,
            subject Name,
            subjectPublicKeyInfo SubjectPublicKeyInfo,
            issuerUniqueIdentifier [1] IMPLICIT UniqueIdentifier OPTIONAL,
            ...,
            [[2: -- if present, version shall be v2 or v3
            subjectUniqueIdentifier [2] IMPLICIT UniqueIdentifier OPTIONAL]],
            [[3: -- if present, version shall be v2 or v3
            extensions [3] Extensions OPTIONAL]]
            -- If present, version shall be v3]]
} (CONSTRAINED BY { -- shall be DER encoded -- } )
```

In order to meet the security goals, the R5F SBL and the HSM runtime image needs to have an X.509 certificate attached to the binary images. The Boot-ROM will only load images which have a valid X.509 certificate attached to them.

5.7.3 Certificate expectations

ROM expectations from the certificate for HS-FS and HS-SE devices is as follows:

Device Type	Validation requirements for SBL			Validation requirements for HSM RT		
	Certificate Verification	Image Integrity	Image Decryption	Certificate Verification	Image Integrity	Image Decryption
HSFS	No authentication, only Dummy certificate for metadata	It's supported, but not mandatory, SBL can boot with or without image integrity. Based on the certificate extension Image integrity will be carried out. SHA512 only supported.	Not supported on HS-FS devices for SBL. Boot fails if encrypted images are loaded.	Authentication is must and it's with TI root of trust (RoT). RSA4K only supported.	It's mandatory, ensure that certificate extension is present. SHA512 only supported.	It's optional. HSMRt can boot without image decryption. Certificate extension for Image decryption will decide this feature. AES256-CBC only supported.

HSSE	Authentication is must and it's with Customer root of trust (RoT). RSA4K only supported.	It's mandatory, ensure that certificate extension is present. SHA512 only supported.	It's optional. SBL can boot without image decryption. Certificate extension for Image decryption will decide this feature. AES256-CBC only supported.	Authentication is must and it's with Customer root of trust (RoT). RSA4K only supported.	It's mandatory, ensure that certificate extension is present. SHA512 only supported.	It's optional. HSMRt can boot without image decryption. Certificate extension for Image decryption will decide this feature. AES256-CBC only supported.

5.7.4 Object Identifiers

Oject Identifiers or OIDs are an identifier mechanism standardized by ITU and ISO/IEC for naming any object, concept, or "thing" with a globally unambiguous persistent name.

An OID corresponds to a node in the "OID tree" or hierarchy, which is formally defined using the ITU's OID standard, X.660.

OID denoting 1.3.6.1.4.1.294.1 is used and followings are the OID Tree.

- 1 ISO,
- 1.3 identified-organization,
- 1.3.6 dod,
- 1.3.6.1 internet,
- 1.3.6.1.4 private,
- 1.3.6.1.4.1 enterprise,
- 1.3.6.1.4.1.294 Texas Instruments,
- 1.3.6.1.4.1.294.1 Device-Boot

5.7.4.1 Boot Information OID (1.3.6.1.4.1.294.1.1)

The Boot Information Object Identifier has the following format:-

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

DESCRIPTION

The Boot Information Object identifier provides information about the image which is being loaded. This information is mandatory and needs to be present in the all the X.509 certificates else the image boot will fail.

OPTIONS

Certificate Type: The certificate type defines the type of the image which is being loaded by the Boot-ROM. The following table illustrates the supported values.

Value	Description
0x1	R5 SBL Boot Image
0x2	HSM Runtime Image

Boot core: The boot core identifies the core on which the image will be executing.

Value	Description
0x0	HSM Core
0x10	R5 Core

Core Options: The core options are documented in the table below.

Value	Description
0x0	Lock Step Mode
Non-Zero	Dual Core Mode

The core options work in conjunction with the following EFUSE configurations:-

1. DUAL_CORE_BOOT_ENABLE
2. DUAL_CORE_SWITCH_DISABLE

These will determine the final operational mode in which the R5 will be executed. The following table summarizes the operation:

Dual Core Boot Enable	Dual Core Switch Disable	Core Options	Description
0	1	0	**Case1:** Executing in Lock Step Mode Switching to dual boot is *Disabled* Certificate requests to execute in Lock Step Mode **Result:** R5 will be started in Lock Step Mode.
0	1	1	**Case2:** Executing in Lock Step Mode Switching to dual boot is *Disabled* Certificate requests to execute in Dual Core Mode **Result:** Error: Dual Boot Switching is disabled
0	0	0	**Case3:** Executing in Lock Step Mode Switching to dual boot is *Enabled* Certificate requests to execute in Lock Step Mode **Result:** R5 will continue to execute in Lock Step Mode

0	0	1	**Case4:** Executing in Lock Step Mode Switching to dual boot is *Enabled* Certificate requests to execute in Dual Core Mode **Result:** R5 will switch from Lock Step to Dual Core Mode.
1	1	0	**Case5:** Executing in Dual Core Mode Switching to dual boot is *Disabled* Certificate requests to execute in Lock Step Mode **Result:** Error: Switching from Dual Core to Lock Step is not allowed.
1	1	1	**Case6:** Executing in Dual Core Mode Switching to dual boot is *Disabled* Certificate requests to execute in Dual Core Mode **Result:** R5 will continue to execute in Dual Core Mode.
1	0	0	**Case7:** Executing in Dual Core Mode Switching to dual boot is *Enabled* Certificate requests to execute in Lock Step Mode **Result:** Error: Switching from Dual Core to Lock Step is not allowed.
1	0	1	**Case8:** Executing in Dual Core Mode Switching to dual boot is *Enabled* Certificate requests to execute in Dual Core Mode **Result:** R5 will continue to execute in Dual Core Mode.

Core options are applicable only for the R5 SBL Images and will be ignored for HSM Runtime certificates.

Load Address: The load address will be the address in the system where the image will be loaded. This information is provided and the R5 SBL and HSM Runtime developers need to ensure that the images account for this.

Value	Description
0x70002000	R5 SBL Load Address
0x0	HSM Runtime Load Address

Image Size: This is the size in bytes of the R5 SBL or HSM Runtime Image to which the certificate has been attached.

5.7.4.2 Software Revision OID (1.3.6.1.4.1.294.1.3)

The Software Revision Object Identifier has the following format: -

```
softwareRevision: = SEQUENCE {
   revision:    INTEGER -- Software revision
}
```

DESCRIPTION

The information in the software revision is used to indicate the version of the image which is being loaded.

revision:

This is the version number. This will be matched to the EFUSE programmed version to indicate if the image loading should be done or not.

Note

This is applicable only for HSSE devices.
HSM Runtime + R5 SBL

The following table summarizes the behavior:

EFUSE Revision	Certificate Revision	Description
0	0	Ignore the revision checking. Images will *always* be loaded
0	>0	Device does not mandate revision checking. Images will be loaded
>0	0	EFUSE Version > Certificate Version. Image will *never* be loaded.
>0	>0	Image will be loaded only if the Certificate revision >= EFUSE revision

Revision Information is read from the following EFUSE fields.

EFUSE	Description
SWRV_SBL	This is used to perform the revision checking while loading the R5 SBL
SWRV_HSM	This is used to perform the revision checking while loading the HSM Runtime

The number of bits for each of the EFUSE in the table above is 64bits. The revision EFUSE supports dual redundancy; this implies that a maximum of 32 revisions can be supported.

Note: The EFUSE SWRV_APP is read and is passed as is by the HSM Boot ROM to the application via the Asset Interface. This EFUSE has a length of 192bits and the interpretation of this is left to the HSM Runtime developers.

5.7.4.3 Image Integrity OID (1.3.6.1.4.1.294.1.2)

The Image Integrity Object Identifier has the following format: -

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

DESCRIPTION

If the X.509 certificate provides the image integrity boot extension the Boot-ROM will perform the SHA-512 on the entire image and will verify the computed hash with the hash provided in the boot extension. In the case of a mismatch the boot will fail.

SHA Type: The Boot-ROM only supports SHA-512.

Value	Description
2.16.840.1.101.3.4.2.3	SHA-512 Object Identifier

Please refer to the Section 2.4 of the RFC-5754 for the SHA-512 Object Identifier.

Hash: This is SHA-512 hash which is calculated over the image (R5 SBL/HSM Runtime)

5.7.4.4 Image Encryption OID (1.3.6.1.4.1.294.1.4)

The Image Encryption Object Identifier has the following format: -

```
imageEncryption ::= SEQUENCE {
   iv:        OCTET STRING -- The initialization vector
   rs:          OCTET STRING -- Random string
   iter:        INTEGER      -- Iteration count
   salt:        OCTET STRING -- encryption salt value
}
```

DESCRIPTION

The Boot-ROM only supports AES-CBC mode with 256bit keys. The information in the image encryption object identifier is used to decrypt the image.

IV:

The initialization vector is used during the AES-CBC decryption procedure. The initialization vector needs to be 16bytes.

rs:

This is the random string which is 32bytes long and is added by the X.509 certificate generator at the end of the image. The Boot-ROM will decrypt the image and will perform a random string comparison to determine if the decryption was successful.

iter:

Iteration Count which is used to determine if the HKDF needs to be performed and key derivation needs to be done. If the iteration count is 0 then the key from the e-fuse is used as is for the decryption. If the iteration count is non-zero then the Boot-ROM will perform the HKDF key derivation using the salt. The derived key is then used for the decryption operation.

salt:

The salt is used only if the iteration count is non-zero and key derivation is being done. The salt is fed to the HKDF module to derive the key. The salt fields should be 32bytes.

5.7.4.5 Derivation OID (1.3.6.1.4.1.294.1.5)

The Derivation Object Identifier has the following format:-

```
derivationKey ::= SEQUENCE {
   salt:      OCTET STRING -- encryption salt value
   info:      OCTET STRING -- [optional]information
}
```

DESCRIPTION

The Boot-ROM will leave a derived key in the assets interface for the HSM Runtime. The key is derived using HKDF from the parameters specified here.

salt: The salt is limited to be 32bytes and is used for key derivation

info: The information is optional in which case the size of the information is set to 0 but if specified is limited to 32bytes.

Note

- If this extension is not present, derived key will be the same across SBL/hsmRT and application
- If this extension is present, derived key will not be the same as SBL/hsmRT

5.7.4.6 Debug OID (1.3.6.1.4.1.294.1.8)

The Debug Object Identifier has the following format:-

```
Debug::= SEQUENCE {
      uid          OCTET STRING     -- Device unique ID
      debugType    INTEGER          -- Debug type
      coreDbgEn    INTEGER          -- Enable core debug mask
      secCoreDbgEn INTEGER          -- Enable secure core debug mask
}
```

DESCRIPTION

The debug object identifier if specified allows the debug ports to be enabled for a specific device. It also can be used to specify the Key protections.

OPTIONS

UID: This is the unique identifier associated with the device. Device specific unique identifiers can be retrieved using the following: -

1. SOC Identifier while operating in a peripheral boot mode

2. Assets on the successful load of the HSM Runtime

The UID field of all 0's is considered to be a wildcard.

Debug Type:

The debug type is described as follows:-

31	18	16	15			0
Reserved		CUST	Debug Type			

Key Protections:

Key Protection	Value	Description
CUST	0	Do not disable access to customer keys
	1	Disable access to customer keys

Debug Type:

Value	Description
0	Disable debug
1	Preserve debug state
2	Enable non-secure debug (Public Debug)
3	Reserved

4	Enable secure and non-secure debug (Full Debug)
5-65535	Reserved

coreDbgEn and secCoreDbgEn: These fields are not used and will be ignored.

Note

R5 SBL Image:	• Optional. • Wildcard UID is allowed. • Key Protections are ignored • Debug Type = 0, 1 and 2 for HS-SE devices • Debug Type = 0 and 1 for HS-FS devices since R5 JTAG is opened by default. • Debug Type = 4 is Invalid
HSM Runtime Image:	• Debug OID is not applicable and is ignored

Outer certificate	• Certificate verification is done with the TI Public Key • Debug Boot Extension is **mandatory**. • UID in the debug extension could either be wild-carded or match the device • Key protection is ignored • Debug Type shall be 4 since we need to unlock the JTAG to debug the HSM Boot-ROM

5.7.5 Binary Image Creation

For secure devices, the process is illustrated in Figure 5-5, and includes the following steps:

1. Create X.509 certificate (1a).
2. Populate certificate extension fields: write image load address and value of the Magic Number from the unencrypted image (1b).
3. Populate image SW version (1c).
4. Encrypt (AES-256-CBC) binary image using derived 256-bit Symmetric Key (2).
5. Compute hash (SHA-512) of encrypted image (3a), and write the digest value to the certificate (3b).
6. Public key is written into the certificate. This could be RSA based public key information.
7. Whole certificate is hashed (SHA-512) (4a), encrypted with private key (4b) using RSA and signature is inserted back into certificate (4c).

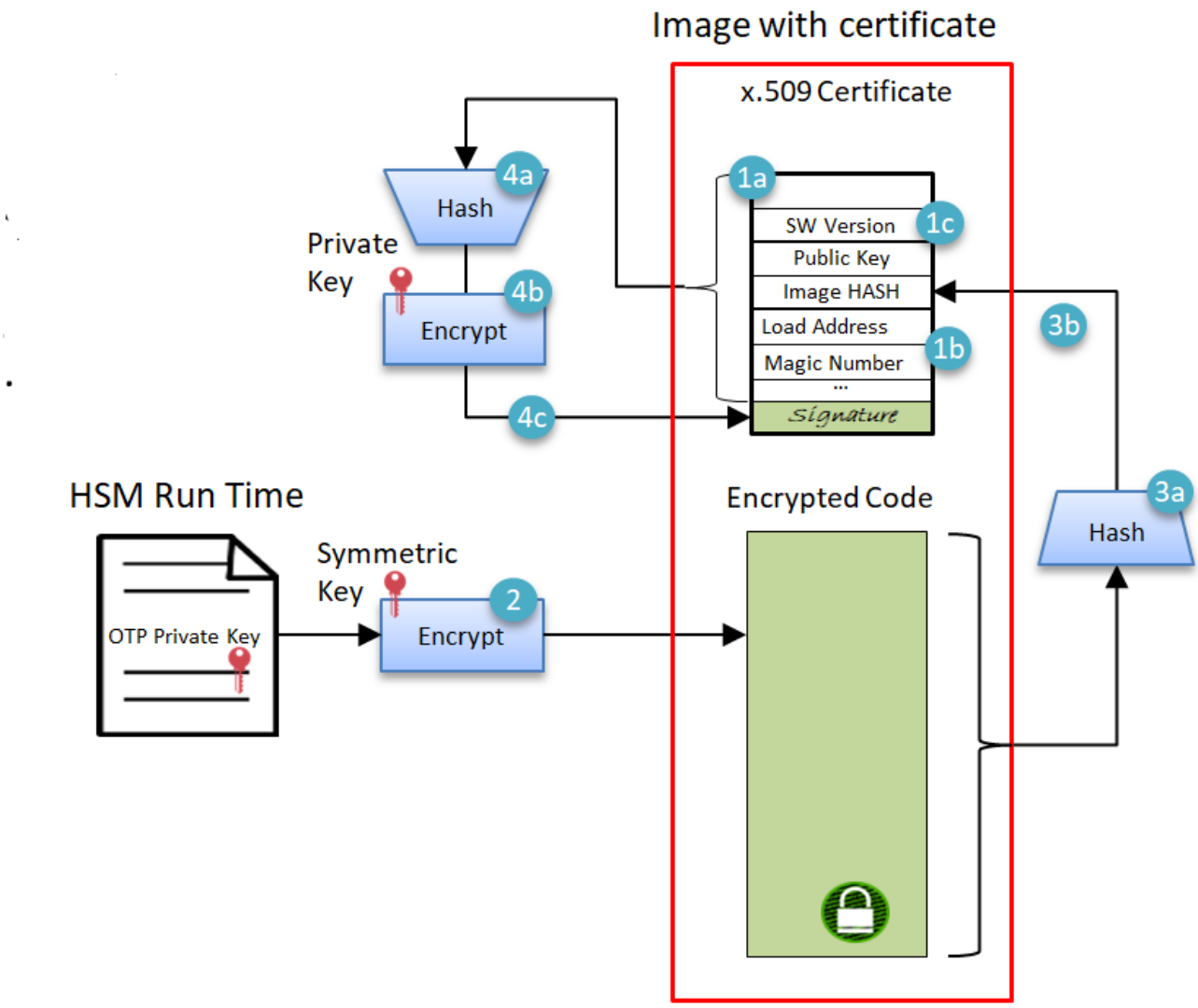


Figure 5-5. Binary Image Creation

Note

Binary Image Creation for non-secure devices, only step 1 is required. Optionally, binary image hashing (step 5) can be performed to verify image integrity.

Note

ROM bootloader supports only RSA4K, SHA512 and AES 256 CBC

Note

TI provides reference scripts and tools for certificate generation and boot image creation in the HSM/ Security software package shared via MSS.

5.7.6 Binary Image Verification

Binary image is verified by the secure device; the process includes the following steps:

1. Compute hash (SHA-512) of the public key in certificate (1a) and compare with the stored public key hash value (1b).
2. Hash (SHA-512) the certificate (2a), decrypt the signature using the public key (2b), and verify they match (2c).
3. Compare whether SW Revision is allowed (3).
4. Read the binary image load address from the certificate.
5. Compute hash (SHA-512) of the encrypted image (4a), and compare with the code hash value from the certificate (4b).
6. Decrypt code (AES-256-CBC) using the 256-bit key derived from the symmetric key (5).
7. Verify the value of the magic number from the certificate and from clear text binary image.

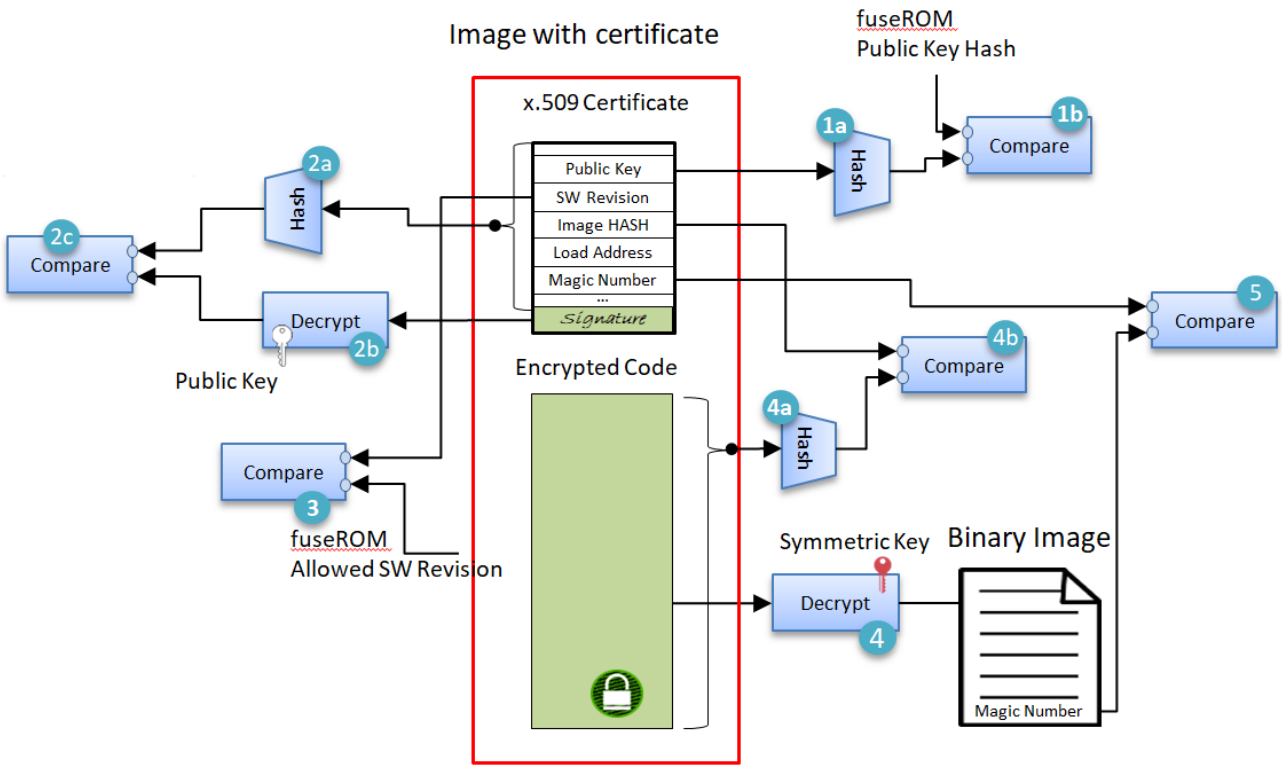
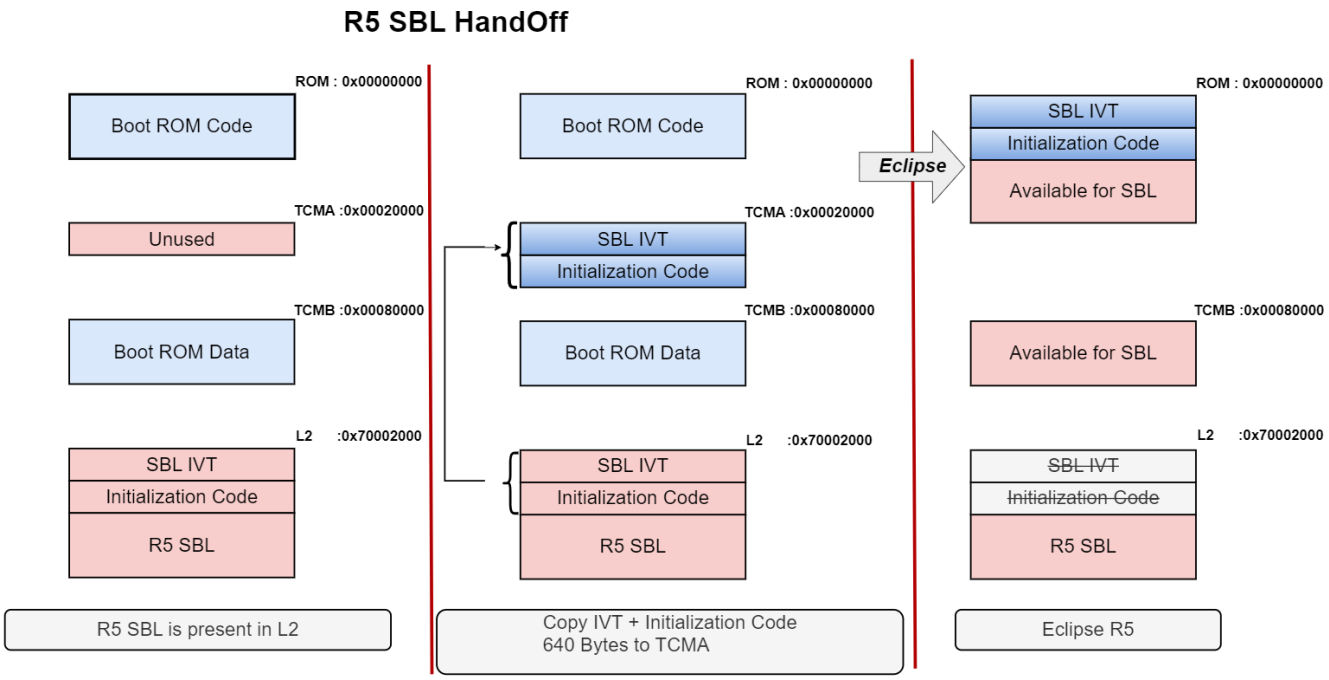


Figure 5-6. Binary Image Verification

5.7.7 R5 SBL Handoff

Figure 5-7 shows the different stages involved after successful validation of the certificate and the image of the SBL:

1. R5 SBL available at L2 address 0x70002000
2. HSM copies 640 bytes from the address 0x70002000 to TCMA start address 0x20000. These 640 bytes consists of IVT and initialization code
3. After the copy, R5 ROM eclipse process is initiated which involves masking R5 ROM and mapping TCMA start address to 0x0 address
4. R5 core reset is issued
5. R5 starts execution from 0x0



5.7.8 HSM RunTime Handoff

Figure 5-8 shows different stages involved in the HSM RunTime boot

1. HSM Runtime available at L2 address
2. SBL sends '**LoadHSMRt**' message to HSM ROM , message will have L2 address pointing to hsmRT image
3. HSM ROM validates the certificate
4. On successful validation of the certificate, HSM ROM copies entire binary from L2 to IRAM address 0x20000
5. After the binary is copied, HSM ROM validates the image against integrity followed by image decryption (image decryption is optional).
6. HSM ROM eclipse process is initiated after image validation is success. This involves masking HSM ROM and mapping IRAM start address to 0x0 address
7. HSM core reset is issued
8. HSMRt starts execution from 0x0

When HSM gets eclipsed, the IRAM RAM address region is mapped to ROM address region. Address mapping during normal and ROM Eclipse Mode is captured in the below tables.

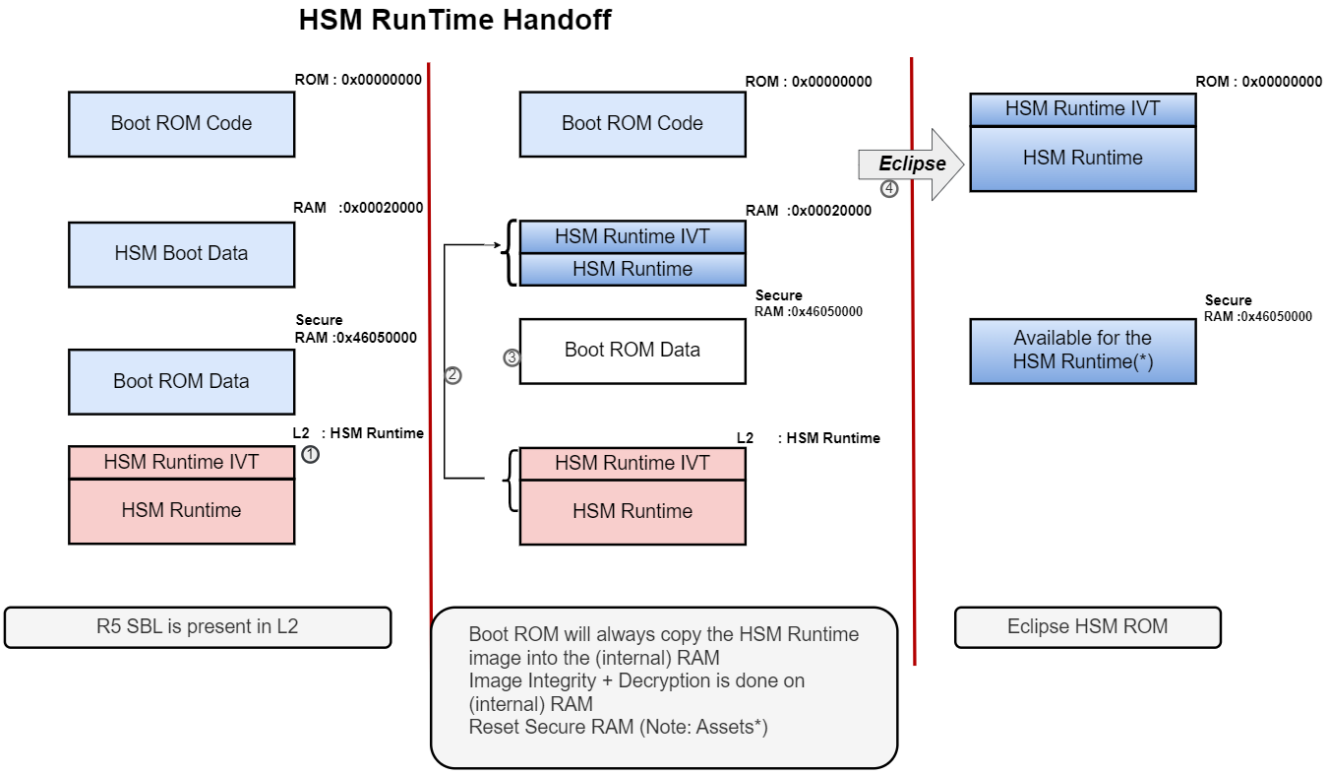


Table 5-11. Address Mapping when HSM ROM is not eclipsed

M4 Address	SCR Hardware Address Translation	Size(KB)	Category
0x0000 0000	0x2000 0000	48	Non-secure ROM
…	…		
0x0000 BFFF	0x2000 BFFF		
0x0001 0000	0x2001 0000	48	Secure ROM
…	…		
0x0001 BFFF	0x2001 BFFF		

Table 5-11. Address Mapping when HSM ROM is not eclipsed (continued)

0x0002 0000	0x2002 0000	256	IRAM
…	…		
0x0000 7FFF	0x2000 7FFF		
0x0002 8000	0x2002 8000		
…	…		
0x0002 FFFF	0x2002 FFFF		
0x0003 0000	0x2003 0000		
…	…		
0x0003 FFFF	0x2003 FFFF		
0x0004 0000	0x2004 0000		
…	…		
0x0004 FFFF	0x2004 FFFF		
0x0005 0000	0x2005 0000		
…	…		
0x0005 FFFF	0x2005 FFFF		

Table 5-12. Address Mapping when HSM ROM is eclipsed

M4 Address	SCR + Eclipse Hardware Address Translation	Size	Category
0x0000 0000	0x2002 0000	256KB	RAM
…	…		
…	…		
0x0003 FFFF	0x2005 FFFF		
0x0004 0000	0x2000 0000	64KB	Reserved space
…	…		
0x0004 FFFF	0x2000 FFFF		
0x0005 0000	0x2001 0000	64KB	Reserved space
…	…		
0x0005 FFFF	0x2001 FFFF		

5.7.9 Post Boot Status

5.7.9.1 R5

5.7.9.1.1 Memory

Memory used by R5 Boot-Rom and their status is shown in Table 5-13. Memory not used by R5 is untouched by Boot-Rom.

Table 5-13. R5 Memory

Memory type	Status
TCMA	SBL IVT and init code runs from TCMA have been copied to TCMA. The maximum size of code in TCMA is 640 bytes (refer to example SBL linker command file for more details).
TCMB	Open to be used by SBL
L2	Contains SBL image and certificate

5.7.9.1.2 Clock

Table 5-14. R5 Clock

Clock type	Status
R5 PLL_CORE_CLK	Running at 400 MHz
R5 VCLK	Running at 200 MHz
DPLL_CORE_HSDIV0_CLKOUT0	Running at 400 MHz

5.7.9.1.3 IP Blocks

This is the state where the ROM Bootloader hands over the control to the Secondary BootLoader (SBL)

Table 5-15. R5 IP Blocks

IP	Status
Timer	Disabled
VIM	All interrupts are disabled VIM memory is cleared
Mail Box	Memory cleared
QSPI(QSPI boot)	QSPI clock is disabled QSPI is set to “Force idle”
EDMA(QSPI boot)	EDMA channel is disabled paramSet memory is cleared Channel to paramSet mapping is cleared
UART(UART boot)	SCIA is reset through IP’s soft reset

5.7.9.1.4 Pinmux Settings

Pinmux settings are left with the settings used for the boot mode.

For OSPI flash boot, OSPI interface pins are left configured as OSPI boot. UART pins are NOT touched in this boot mode.

For UART boot, UART pins are left configured as UART boot. OSPI pins are NOT touched in this boot mode.

Table 5-16. R5 Pinmux Settings

Boot Mode	OSPI Pin Status	UART Pin Status
OSPI	MCAN1_TX=>MCAN1_TX(Default Pull) MCAN1_RX=>MCAN1_RX (Default Pull) MCAN0_TX=>MCAN0_TX(Default Pull) MCAN0_RX=>MCAN0_RX(Default Pull) QSPI0_D3=>QSPI_D3(Default Pull) QSPI0_D2=>QSPI_D2 (Default Pull) QSPI0_D1=>QSPI_D1(Default Pull) QSPI0_D0=>QSPI_D0(Default Pull) QSPI0_CLK0=>QSPI_CLK (Default Pull) QSPI0_CSn0=>QSPI_CS (Default Pull) QSPI_CLKLB =>QSPI_CLKLB(Default Pull)	Same as reset
UART	Same as reset	UART0_TXD=>UART0_TX D(Default Pull) UART0_RXD=>UART0_RXD (Default Pull)

5.7.9.1.5 PBIST

BootRom executes the PBIST test for the following memory groups used by ROM during boot:

Table 5-17. R5 PBIST

Memory Group Number	Memory Group Description
4	MEM_TOP_PBISTROM

Table 5-17. R5 PBIST (continued)

Memory Group Number	Memory Group Description
6	MEM_MSS_CA_ATCM
7	MEM_MSS_CA_BTCM
8	MEM_MSS_CB_ATCM
9	MEM_MSS_CB_BTCM
15	MEM_MSS_L2_0
16	MEM_MSS_L2_1

Note

ROM does not perform LBIST

5.7.9.2 Assets

Once the HSM Boot-ROM has loaded the HSM Runtime, it will leave behind Assets which are located at the beginning of the SECURE RAM. This information is made available for the development of the HSM Runtime. Please refer to *ROM External Interface documentation* in HSM/Security software package which describes this asset structure details and the start address.

Assets recorded are as follows:

1. HSM Boot ROM Version
2. Device Type (HS-FS, HS-SE etc.)
3. Key revision and count
4. Derived Key (Using the Derivation Object Id)
5. Public Key
6. Unique Device Identifier, etc.

5.8 Boot Image Format

5.8.1 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)

5.8.2 Generating X.509 Certificates

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

5.8.2.1 Key Generation

The SBL must always be signed with a given OpenSSL key. Secure devices must have encryption and 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.

5.8.2.1.1 RSA Key Generation

$$\text{Signature} = \text{digest}^{\text{privExp}} \bmod \text{n} \quad (1)$$

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 4096-bit is chosen.

Note

AM263Px Supports the following parameters:

- Public Key Length: RSA4K
- Decryption: AES-256 CBC
- Hashing: SHA512

The following sequence is used to generate degenerate RSA keys:

1. Create a random RSA key:

```
openssl genrsa –out key.pm 4096
```

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, concatenate the lines and add a preceding 0x.

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.

5.8.2.2 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
1.3.6.1.4.1.294.1.8 = ANSI:SEQUENCE:debug
[ boot_seq ]
certType =INTEGER:1
bootCore = INTEGER:16
bootArchWidth = INTEGER:32
destAddr = FORMAT:HEX,OCT:bc934b00
imageSize = INTEGER:0x00004860
[ image_integrity ]
shaType = OID:2.16.840.1.101.3.4.2.3
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 ]
[ 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.

5.8.2.3 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.

5.9 Boot Memory Maps

5.9.1 Memory Layout/MPU

Table 5-18 shows an overview of the MPU configuration. In the R5FSS MPU, higher numbered regions have priority, therefore, where two regions overlap, the right-most region column defines the memory attributes in the table.

Table 5-18. Memory Layout/MPU

Memory Address	Regions	
0x0000_0000	Region 1 - Non-executable Full Access	Region 2 - ROM Read-only Exec
0x0001_FFFF		
…….		
0x0002_0000		Region 3 - TCM User Access
0x0002_3FFF		
…….		
0x0008_0000		Region 4 - ROM User Access
0x0008_3FFF		
…….		
0x4400_0000		Region 7 - TX Mailbox RAM
0x440F_FFFF		
…….		
0x5380_0000		Region 10 - OSPI Config Space
0x5380_FFFF		
…….		
0x6000_0000		Region 9 - OSPI Memory
0x67FF_FFFF		
…….		
0x7000_0000		Region 5 - All OCSRAM
0x703F_FFFF		
…….		
0x7200_0000		Region 8 - RX Mailbox RAM
0x720F_FFFF		
…….		
0xFFFF_FFFF		

5.9.2 Logger

The ROM code uses logger module for debug information. They are shown in the below table:

Table 5-19. Global Memory Addresses

Group	Address	Size (bytes)	Content
Infor/Warning/Error Logs	0x0008_2800	4096	Log Entry size: 8 words (128 bits) Word 1: Log_type - 0xABCD001 - Info 0xABCD002 - Warning 0xABCD003 - Error 0xABCD004 - Critical 2nd word: FileName - source file name 3rd word: Line number - line at which log reported in the source file 4th word: Value1 - Debug word1 5th word: Value2 - Debug word2 6th word: Timer count (lower) - lower 32-bit timer count 7th word: Timer count (upper) - upper 32-bit timer count

Failure and recovery

Any failures detected by R5 or HSM while booting, will lead to SoC warm reset issued by WDT (watchdog timer) after 180 sec. From the ROM perspective, cold reset and warm reset are the same as far as boot flow is considered.

The ROM code version information is a structure shown in Table 5-20

Table 5-20. ROM Code Version

Field	Address	Size (bytes)	Value
Version Number	0x4605_0940	4	"0x0001_0000" (1.0.0)
Device Name	0x4605_0944	12	"am263Px"

6 Device Configuration

This chapter describes the device configuration details including information related to Control MMR's, Power, Reset, and Clocking.

6.1 Control Module

6.1.1 Control Overview

The Control module is the main controller for top-level device behavior in various states. This module contains registers for configuration, bootstrap (SOP) signals, I/O terminal pad multiplexing, clock selection, and many other device-level configuration options. MMR (Memory Mapped Registers) are used by software to program the hardware. The register is directly accessible from software because it is mapped into a memory location of the memory-map, such that writing to and reading from that memory location corresponds to writing to and reading from the hardware register. There are various Control Module or CTRLMMR modules defined in this device:

General SoC Control Modules

- TOP_CTRL (CTRLMMR0): SoC-level configuration registers
- MSS_CTRL (CTRLMMR1): SoC and peripheral-level configuration registers
- CONTROLSS_CTRL (CTRLMMR2): CONTROLSS-level configuration registers including general control, reset, and clocking-related functions for the real time control subsystem (CONTROLSS))

Pad Configuration Control Modules

- IOMUX (PADCFG_CTRLMMR0): SoC-level terminal configuration control registers

Reset and Clocking Control Modules

- TOPRCM (RCM_CTRLMMR0): SoC-level Clock and Reset control registers
- MSS_RCM (RCM_CTRLMMR1): SoC and Peripheral-level Clock and Reset control registers

6.1.1.1 MMR Write Protection

All Control Module MMR have a protection mechanism which prevents spurious writes from changing register values. LOCK0_KICK0 and LOCK0_KICK1 registers are used for this purpose. The sequence to unlock these MMR is as follows:

1. Write exact unlock value (Table 6-1) to <Control Module>LOCK0_KICK0:KEY field
2. Write exact unlock value (Table 6-1) to <Control Module>LOCK0_KICK1:KEY field

The sequence to lock the MMR is as follows:

1. Write zero (or anyother value other than the unlock value)Table 6-1) to <Control Module>LOCK0_KICK1:KEY field
2. Write zero (or anyother value other than the unlock value)Table 6-1) to <Control Module>LOCK0_KICK0:KEY field

- **Note**

  If the above sequence for locking the IOMUX is not followed, an AHB_WRITE_ERROR interrupt will occur (if enabled).

For example, to unlock Control Module MSS_CTRL the sequence is as below:

1. Write 0x01234567 to MSS_CTRL.LOCK0_KICK0:KEY
2. Write 0xFEDCBA8 to MSS_CTRL.LOCK0_KICK1:KEY

To lock the Control Module MSS_CTRL the sequence is as below:

1. Write 0x0 to MSS_CTRL.LOCK0_KICK1:KEY
2. Write 0x0 to MSS_CTRL.LOCK0_KICK0:KEY

Any writes to locked memory region will result in assertion of the MMR_ACCESS_ERR_WR event by the respective control modules. This assertion can be enabled or disabled by writing the appropriate value to <Control Module>.INTR_ENABLE.KICK_ERR_EN field.

The table below shows the values that must be written to the LOCK0_KICK0 and LOCK0_KICK1 registers to unlock the various Control modules' MMR.

Table 6-1. Kick Protection Register Unlock Values

Protected Register	LockKick Register	Unlock Value
TOP_CTRL	LOCK0_KICK0	0x01234567
	LOCK0_KICK1	0xFEDCBA8
MSS_CTRL	LOCK0_KICK0	0x01234567
	LOCK0_KICK1	0xFEDCBA8
CONTROLSS_CTRL	LOCK0_KICK0	0x01234567
	LOCK0_KICK1	0xFEDCBA8
TOP_RCM	LOCK0_KICK0	0x01234567
	LOCK0_KICK1	0xFEDCBA8
MSS_RCM	LOCK0_KICK0	0x01234567
	LOCK0_KICK1	0xFEDCBA8
IOMUX	LOCK0_KICK0	0x83E70B13
	LOCK0_KICK1	0x95A4F1E0

Note

To ensure that all registers from a given partition are write protected, software must always re-lock the protection mechanism after completing the register writes.

The kick protection registers described in this section are an exception and are not write protected by the protection mechanism.

6.1.1.2 MMR Access Error Interrupt

The Control Modules can generate the access error interrupts MMR_ACCESS_ERR_WR and MMR_ACCESS_ERR_RD. The interrupts are asserted when one or more of the following accesses are made:

- (a) write access when MMR are locked
- (b) access to illegal address in the control module

The following registers are related to handling of these errors inside the respective Control Module.

- <Control Module>INTR_RAW_STATUS - Interrupt Raw Status/Set register
- <Control Module>INTR_ENABLED_STATUS_CLEAR - Interrupt Enabled Status/Clear register
- <Control Module>INTR_ENABLE - Interrupt Enable register
- <Control Module>INTR_ENABLE_CLEAR - Interrupt Enable Clear register

The following applies for the interrupt behavior of each Control Module:

- The Control Module only asserts the interrupt line if the interrupt is enabled.
  - Interrupts are **enabled** by setting the corresponding bits in the INTR_ENABLE register to 1h.
  - Interrupts are **disabled** by setting the corresponding bits in the INTR_ENABLE_CLEAR register to 1h.
- 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. The INTR_ENABLED_STATUS_CLEAR register is only set if the corresponding interrupt is enabled.
- An interrupt is generated by the control module if the relevant bit in the INTR_RAW_STATUS register is set to 1h and the interrupt is enabled through the INTR_ENABLE register. This feature is useful during user software debugging. In addition, even if interrupts are disabled, the corresponding raw flag in the INTR_RAW_STATUS register is set to 1h when an interrupt condition occurs.
- If interrupts are disabled, the corresponding raw flag in the INTR_RAW_STATUS register is set to 1h when an interrupt condition occurs. The INTR_RAW_STATUS can be cleared by setting the corresponding bit in the INTR_RAW_STATUS register to 1h.

The MSS_CTRL module aggregates the Control Module interrupts MMR_ACCESS_ERR_WR and MMR_ACCESS_ERR_RD and generates MMR_ACCESS_ERRAGGR to the R5 Cores (see Section 6.1.3.2.8).

Note

C2K_GLOBAL_CTRL is not aggregated into MSS_CTRL's MMR_ACCESS_ERR_WR and MMR_ACCESS_ERR_RD

Table 6-2 lists the interrupt events which can assert the MSS_CTRL Access Error.

Table 6-2. MSS_CTRL Access Error Interrupt Events

Event Name	Event Flag	Event Mask	Description
MMR_ACCESS_ERR_WR	INTR_RAW_STATUS.KICK_ERR	INTR_ENABLE.KICK_ERR_EN	Lock violation interrupt. Occurs when writing to a register in a locked control module.
MMR_ACCESS_ERR_RD	INTR_RAW_STATUS.ADDR_ERR	INTR_ENABLE.ADDR_ERR_EN	Read addressing violation interrupt. Occurs when reading an illegal address inside the control module.
MMR_ACCESS_ERR_WR	INTR_RAW_STATUS.ADDR_ERR	INTR_ENABLE.ADDR_ERR_EN	Write addressing violation interrupt. Occurs when writing an illegal address inside the control module.

When an error event as described in Table 6-2 above occurs, the associated error details are captured in the FAULT_ADDRESS, FAULT_TYPE_STATUS, and FAULT_ATTR_STATUS registers.

FAULT_ADDRESS contains the address of the first fault access. FAULT_TYPE_STATUS and FAULT_ATTR_STATUS 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.FAULT_CLR bit must be set to 1h.

6.1.2 TOP_CTRL

The TOP_CTRL (CTRLMMR0) module has MMR associated with the following functions:

- Section 6.2.2.1
- Section 6.2.2.3

6.1.2.1 TOP_CTRL Integration

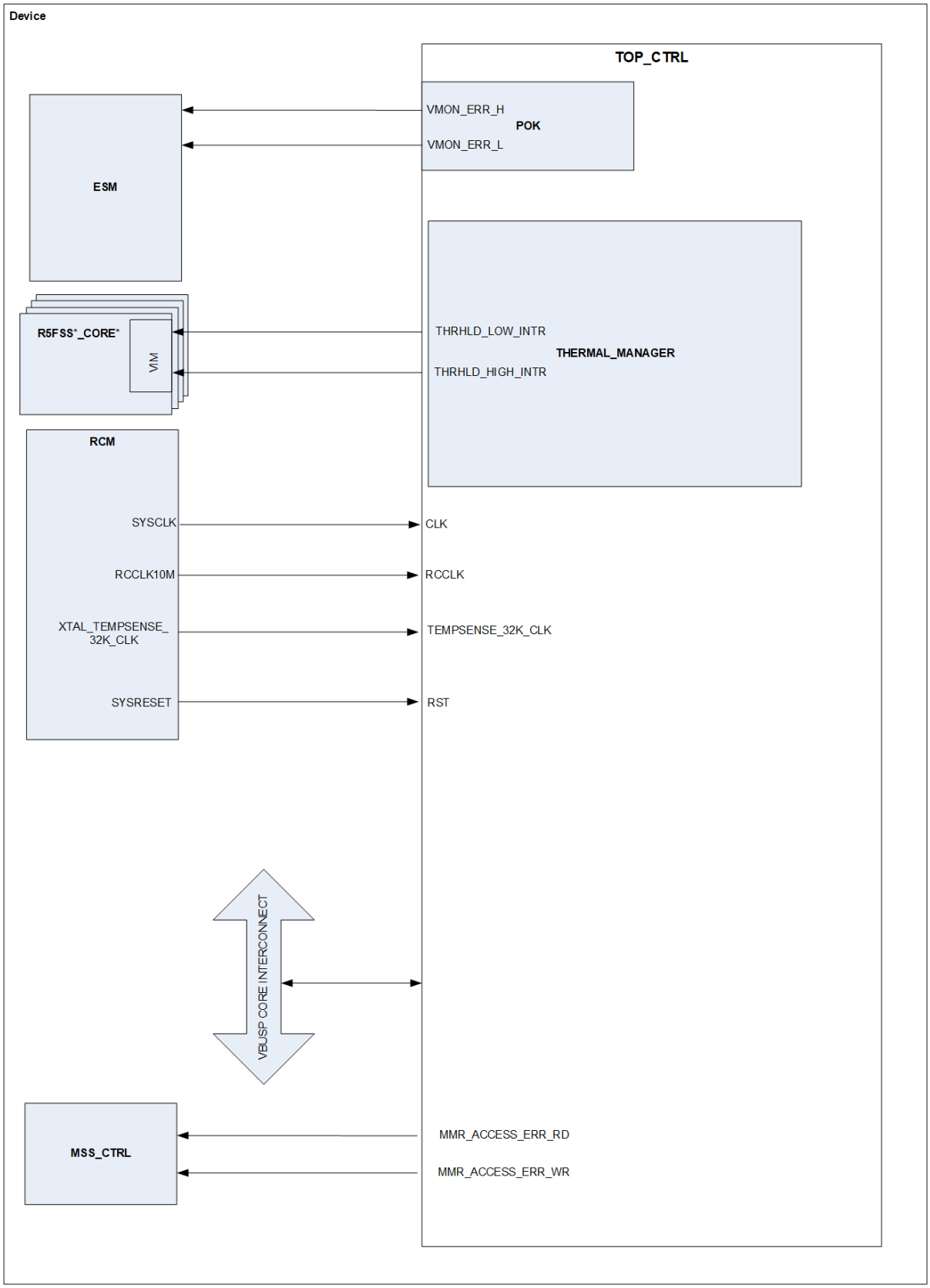


Figure 6-1. TOP_CTRL Integration Diagram

Table 6-3. TOP_CTRL Device Integration

Module Instance	Device Allocation	Interconnect
TOP_CTRL	✓	VBUSP CORE Interconnect

Table 6-4. TOP_CTRL Clocks and Resets

Clocks				
Module Instance	**Module Clock Input**	**Source Clock Signal**	**Source**	**Description**
TOP_CTRL	CLK	SYSCLK	TOP_RCM	Functional and Interface Clock
TOP_CTRL	RCCLK	RCCLK10M	TOP_RCM	POK Filter clock
TOP_CTRL	TEMPSENSE_32K_CLK	XTAL_TEMPSENSE_32K_CLK	TOP_RCM	Thermal Manager clock

Table 6-5. TOP_CTRL Resets

Resets				
Module Instance	**Module Input**	**Source Signal**	**Source**	**Description**
TOP_CTRL	RST	SYSRESET	TOP_RCM	TOP_CTRL Reset
TOP_CTRL	TSENSE_RESET	TSENSE_RESET	MSS_RCM	Thermal Manager Reset

Table 6-6. TOP_CTRL Interrupt Requests

Module Instance	Module Interrupt Signal	Destination Interrupt Input	Destination	Type	Description
TOP_CTRL	MMR_ACCESS_ERR_RD	TOP_CTRL_RD_ACCESS_ERR	MSS_CTRL	Level	Read Access Error Interrupt indicating protection, addressing, or lock violation
	MMR_ACCESS_ERR_WR	TOP_CTRL_WR_ACCESS_ERR			Write Access Error Interrupt indicating protection, addressing, or lock violation
	THRHLD_HIGH_INTR (INTR_TSENSE_H)	R5SS*_CORE*_INTR_IN_133	R5SS*_CORE*_VIM		Temperature High Threshold Interrupt
	THRHLD_LOW_INTR (INTR_TSENSE_L)	R5SS*_CORE*_INTR_IN_134			Temperature Low threshold Interrupt

Table 6-7. TOP_CTRL ESM Interrupts

Module Instance	Module Interrupt Signal	Destination Interrupt Input	Destination	Type	Description
TOP_CTRL	VMON_ERR_H	ESM_LVL_EVENT_41	ESM	Level	POK Voltage monitor error high interrupt
	VMON_ERR_L	ESM_LVL_EVENT_42			POK Voltage monitor error low interrupt

6.1.3 MSS_CTRL

The MSS_CTRL module has MMR associated with the following functions:

- R5FSS CPU Global Configuration and Control
- Memory Initialization
- EDMA Global Configuration and Event Aggregation
- CPSW Global Configuration
- GPMC Global Configuration
- MPU Interrupt Aggregator
- MMR Access Error Interrupt Aggregator
- Safety Registers
- Interconnect Safety
- MSS_CTRL MMR Kick Protection Registers
- MSS_CTRL MMR Access Error

6.1.3.1 MSS_CTRL Integration

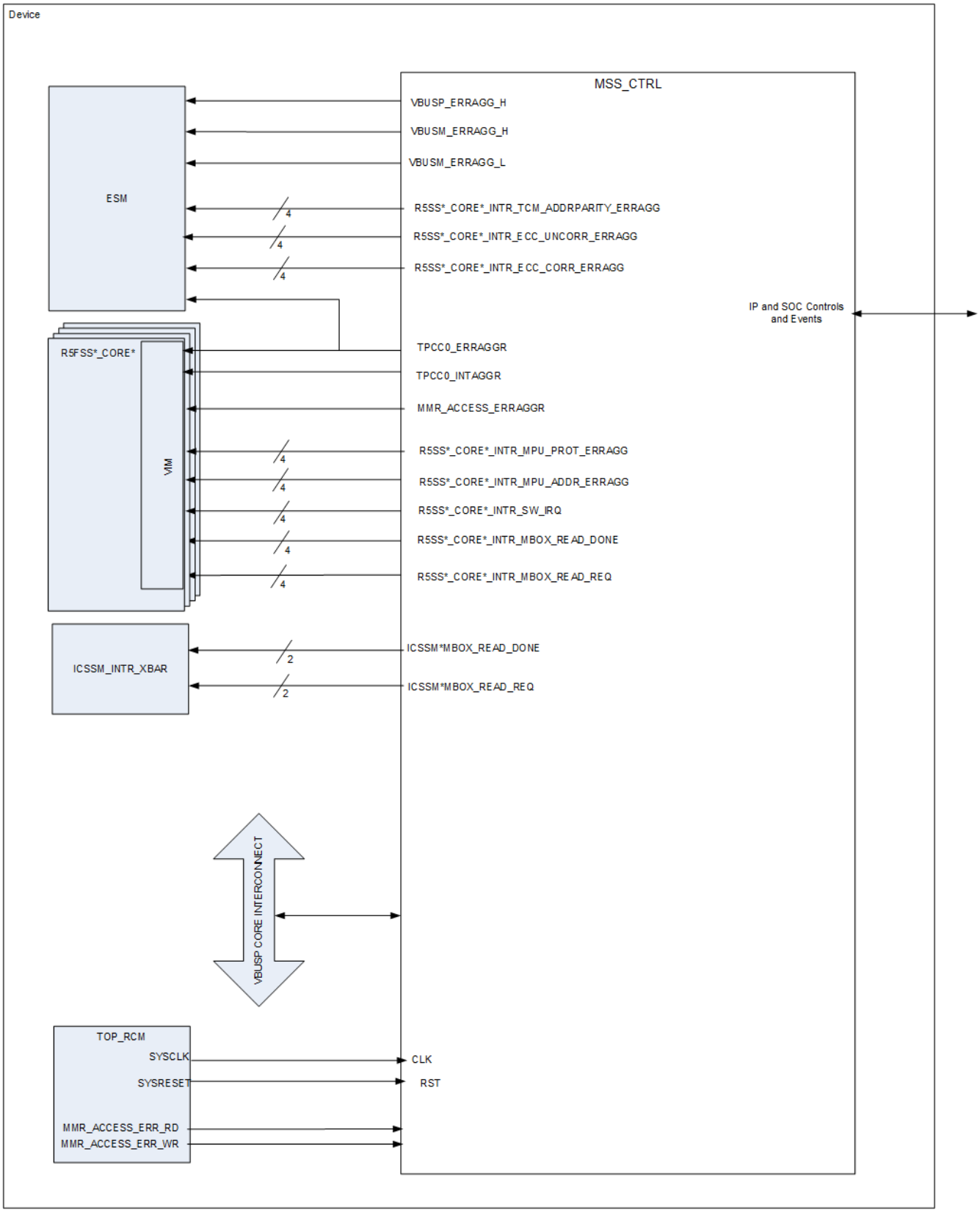


Figure 6-2. MSS_CTRL Integration Diagram

Table 6-8. MSS_CTRL Integration Attributes

Module Instance	Attributes
	Interconnect
MSS_CTRL	VBUSP CORE Interconnect

Table 6-9. MSS_CTRL Clocks and Resets

Module Instance	Module Clock Input	Source Clock Signal	Source	Description
Clocks				
MSS_CTRL	CLK	SYSCLK	TOP_RCM	Functional and Interface Clock
Resets				
MSS_CTRL	RST	SYSRESET	TOP_RCM	MSS_CTRL Reset

Table 6-10. MSS_CTRL Hardware Requests

Interrupt Requests					
Module Instance	**Module Interrupt Signal**	**Destination Interrupt Input**	**Destination**	**Description**	**Type**
MSS_CTRL	ICSSM_PRU0_MBOX_READ_REQ	IN_INTR51	PRU_ICSS_XBAR_INTR TR0	Interrupt indicating Mailbox Read Request to PRU0	Level
MSS_CTRL	ICSSM_PRU0_MBOX_READ_DONE	IN_INTR53	PRU_ICSS_XBAR_INTR TR0	Interrupt indicating Mailbox Read Done to PRU0	Level
MSS_CTRL	ICSSM_PRU1_MBOX_READ_REQ	IN_INTR52	PRU_ICSS_XBAR_INTR TR0	Interrupt indicating Mailbox Read Request to PRU0	Level
MSS_CTRL	ICSSM_PRU1_MBOX_READ_DONE	IN_INTR54	PRU_ICSS_XBAR_INTR TR0	Interrupt indicating Mailbox Read Done to PRU0	Level
MSS_CTRL	R5FSS0_CORE0_INTR_MBOX_READ_REQ	R5SS0_CORE0_INTR_IN_136	R5SS0_CORE0_VIM	Interrupt indicating Mailbox Read Request to R5SS0 CORE0	Level
MSS_CTRL	R5FSS0_CORE0_INTR_MBOX_READ_DONE	R5SS0_CORE0_INTR_IN_137	R5SS0_CORE0_VIM	Interrupt indicating Mailbox Read Done to R5SS0 CORE0	Level
MSS_CTRL	R5FSS0_CORE1_INTR_MBOX_READ_REQ	R5SS0_CORE1_INTR_IN_136	R5SS0_CORE1_VIM	Interrupt indicating Mailbox Read Request to R5SS0 CORE1	Level
MSS_CTRL	R5FSS0_CORE1_INTR_MBOX_READ_DONE	R5SS0_CORE1_INTR_IN_137	R5SS0_CORE1_VIM	Interrupt indicating Mailbox Read Done to R5SS0 CORE1	Level
MSS_CTRL	R5FSS1_CORE0_INTR_MBOX_READ_REQ	R5SS1_CORE0_INTR_IN_136	R5SS1_CORE0_VIM	Interrupt indicating Mailbox Read Request to R5SS1 CORE0	Level

Table 6-10. MSS_CTRL Hardware Requests (continued)

Interrupt Requests					
Module Instance	**Module Interrupt Signal**	**Destination Interrupt Input**	**Destination**	**Description**	**Type**
MSS_CTRL	R5FSS1_CORE0_INTR_MBOX_READ_DONE	R5SS1_CORE0_INTR_IN_137	R5SS1_CORE0_VIM	Interrupt indicating Mailbox Read Done to R5SS1 CORE0	Level
MSS_CTRL	R5FSS1_CORE1_INTR_MBOX_READ_REQ	R5SS1_CORE1_INTR_IN_136	R5SS1_CORE1_VIM	Interrupt indicating Mailbox Read Request to R5SS1 CORE1	Level
MSS_CTRL	R5FSS1_CORE1_INTR_MBOX_READ_DONE	R5SS1_CORE1_INTR_IN_137	R5SS1_CORE1_VIM	Interrupt indicating Mailbox Read Done to R5SS1 CORE1	Level
MSS_CTRL	R5FSS0_CORE0_INTR_SW_IRQ	R5SS0_CORE0_INTR_IN_129	R5SS0_CORE0_VIM	Interrupt indicating SW Interrupt to R5SS0 CORE0	Level
MSS_CTRL	R5FSS0_CORE1_INTR_SW_IRQ	R5SS0_CORE1_INTR_IN_129	R5SS0_CORE1_VIM	Interrupt indicating SW Interrupt to R5SS0 CORE1	Level
MSS_CTRL	R5FSS1_CORE0_INTR_SW_IRQ	R5SS1_CORE0_INTR_IN_129	R5SS1_CORE1_VIM	Interrupt indicating SW Interrupt to R5SS1 CORE0	Level
MSS_CTRL	R5FSS1_CORE1_INTR_SW_IRQ	R5SS1_CORE1_INTR_IN_129	R5SS1_CORE1_VIM	Interrupt indicating SW Interrupt to R5SS1 CORE1	Level
MSS_CTRL	R5FSS0_CORE0_INTR_MPU_PROT_ERRAGG	R5SS0_CORE0_INTR_IN _70	R5SS0_CORE0_VIM	Aggregated Interrupt indicating MPU Protection Error to R5SS0 CORE0	Level
MSS_CTRL	R5FSS0_CORE1_INTR_MPU_PROT_ERRAGG	R5SS0_CORE1_INTR_IN _70	R5SS0_CORE1_VIM	Aggregated Interrupt indicating MPU Protection Error to R5SS0 CORE1	Level
MSS_CTRL	R5FSS1_CORE0_INTR_MPU_PROT_ERRAGG	R5SS1_CORE0_INTR_IN _70	R5SS1_CORE0_VIM	Aggregated Interrupt indicating MPU Protection Error to R5SS1 CORE0	Level
MSS_CTRL	R5FSS1_CORE1_INTR_MPU_PROT_ERRAGG	R5SS1_CORE1_INTR_IN _70	R5SS1_CORE1_VIM	Aggregated Interrupt indicating MPU Protection Error to R5SS1 CORE1	Level

Table 6-10. MSS_CTRL Hardware Requests (continued)

Interrupt Requests					
Module Instance	**Module Interrupt Signal**	**Destination Interrupt Input**	**Destination**	**Description**	**Type**
MSS_CTRL	R5FSS0_CORE0_INTR_MPU_ADDR_ERRAGG	R5SS0_CORE0_INTR_IN _69	R5SS0_CORE0_VIM	Aggregated Interrupt indicating MPU Address Error to R5SS0 CORE0	Level
MSS_CTRL	R5FSS0_CORE1_INTR_MPU_ADDR_ERRAGG	R5SS0_CORE1_INTR_IN _69	R5SS0_CORE1_VIM	Aggregated Interrupt indicating MPU Address Error to R5SS0 CORE1	Level
MSS_CTRL	R5FSS1_CORE0_INTR_MPU_ADDR_ERRAGG	R5SS1_CORE0_INTR_IN _69	R5SS1_CORE0_VIM	Aggregated Interrupt indicating MPU Address Error to R5SS1 CORE0	Level
MSS_CTRL	R5FSS1_CORE1_INTR_MPU_ADDR_ERRAGG	R5SS1_CORE1_INTR_IN _69	R5SS1_CORE1_VIM	Aggregated Interrupt indicating MPU Address Error to R5SS1 CORE1	Level
MSS_CTRL	MMR_ACCESS_ERRAGGR	R5SS0_CORE0_INTR_IN_124	R5SS0_CORE0_VIM	Aggregated Interrupt indicating MMR Access Error	Level
		R5SS0_CORE1_INTR_IN_124	R5SS0_CORE1_VIM		
		R5SS0_CORE0_INTR_IN_124	R5SS0_CORE0_VIM		
		R5SS1_CORE1_INTR_IN_124	R5SS1_CORE1_VIM		
MSS_CTRL	TPCC0_INTAGGR	R5SS0_CORE0_INTR_IN _72	R5SS0_CORE0_VIM	Aggregated Interrupt from EDMA Interrupt sources	Level
		R5SS0_CORE1_INTR_IN _72	R5SS0_CORE1_VIM		
		R5SS0_CORE0_INTR_IN _72	R5SS0_CORE0_VIM		
		R5SS1_CORE1_INTR_IN _72	R5SS1_CORE1_VIM		
MSS_CTRL	TPCC0_ERRGGR	R5SS0_CORE0_INTR_IN _73	R5SS0_CORE0_VIM	Aggregated Interrupt from EDMA Error sources	Level
		R5SS0_CORE1_INTR_IN _73	R5SS0_CORE1_VIM		
		R5SS0_CORE0_INTR_IN _73	R5SS0_CORE0_VIM		
		R5SS1_CORE1_INTR_IN _73	R5SS1_CORE1_VIM		
ESM Events					
MSS_CTRL	TPCC0_ERRGGR	ESM_LVL_EVENT_63	ESM	Aggregated Error from EDMA Error sources	Level
MSS_CTRL	R5SS0_CORE0_CORR_ERRAGG	ESM_LVL_EVENT_47	ESM	Aggregated Correctable Memory ECC Error from R5SS0 CORE0	Level
MSS_CTRL	R5SS0_CORE1_CORR_ERRAGG	ESM_LVL_EVENT_49	ESM	Aggregated Correctable Memory ECC Error from R5SS0 CORE1	Level
MSS_CTRL	R5SS1_CORE0_CORR_ERRAGG	ESM_LVL_EVENT_55	ESM	Aggregated Correctable Memory ECC Error from R5SS1 CORE0	Level

Table 6-10. MSS_CTRL Hardware Requests (continued)

Interrupt Requests					
Module Instance	**Module Interrupt Signal**	**Destination Interrupt Input**	**Destination**	**Description**	**Type**
MSS_CTRL	R5SS1_CORE1_CORR_ERRAGG	ESM_LVL_EVENT_57	ESM	Aggregated Correctable Memory ECC Error from R5SS1 CORE1	Level
MSS_CTRL	R5SS0_CORE0_UNCORR_ERRAGG	ESM_LVL_EVENT_48	ESM	Aggregated Uncorrectable Memory ECC Error from R5SS0 CORE0	Level
MSS_CTRL	R5SS0_CORE1_UNCORR_ERRAGG	ESM_LVL_EVENT_50	ESM	Aggregated Uncorrectable Memory ECC Error from R5SS0 CORE1	Level
MSS_CTRL	R5SS1_CORE0_UNCORR_ERRAGG	ESM_LVL_EVENT_56	ESM	Aggregated Uncorrectable Memory ECC Error from R5SS1 CORE0	Level
MSS_CTRL	R5SS1_CORE1_UNCORR_ERRAGG	ESM_LVL_EVENT_58	ESM	Aggregated Uncorrectable Memory ECC Error from R5SS1 CORE1	Level
MSS_CTRL	R5SS0_CORE0_TCM_ADDRPARITY_ERRAGG	ESM_LVL_EVENT_14	ESM	Aggregated TCM Address parity Error from R5SS0 CORE0	Level
MSS_CTRL	R5SS0_CORE1_TCM_ADDRPARITY_ERRAGG	ESM_LVL_EVENT_15	ESM	Aggregated TCM Address parity Error from R5SS0 CORE1	Level
MSS_CTRL	R5SS1_CORE0_TCM_ADDRPARITY_ERRAGG	ESM_LVL_EVENT_16	ESM	Aggregated TCM Address parity Error from R5SS1 CORE0	Level
MSS_CTRL	R5SS1_CORE1_TCM_ADDRPARITY_ERRAGG	ESM_LVL_EVENT_17	ESM	Aggregated TCM Address parity Error from R5SS1 CORE1	Level
MSS_CTRL	VBUSM_ERRAGG_H	ESM_LVL_EVENT_33	ESM	Aggregated VBUSM Bus Safety Error High	Level
MSS_CTRL	VBUSM_ERRAGG_L	ESM_LVL_EVENT_34	ESM	Aggregated VBUSM Bus Safety Error Low	Level
MSS_CTRL	VBUSP_ERRAGG_H	ESM_LVL_EVENT_31	ESM	Aggregated VBUSP Bus Safety Error	Level

6.1.3.2 MSS_CTRL Functional Description

6.1.3.2.1 R5FSS CPU Global Configuration and Control

6.1.3.2.1.1 R5SS Lock Step/Dual Core Configuration

The R5SS*_CONTROL register configures the lockstep/dual core behavior of the R5SS*.

When R5SS*_CONTROL.LOCK_STEP is progammed to 0x7, it configures R5SS* to be in lockstep. When programmed to 0x0, it configures R5SS* to be in dual core mode.

A reset must be issued to R5SS* to switch between dual core and lockstep mode.

When R5SS*_CONTROL.RESET_FSM_TRIGGER is programmed to 0x7, it issues a reset to R5SS*.

Note

By default, after a device reset, both R5SS0 and R5SS1 are in lockstep. Each cluster can be independently configured to be in dual core by the application.

Note

Lockstep to dual core switch can be programmed only once, and cannot be reprogrammed until the device's next power on reset cycle.

R5SS*_STATUS_REG. LOCK_STEP bitfield indicates the mode of R5SS. LOCK_STEP = 0 indicates the corresponding R5SS is in dual core mode, and LOCK_STEP = 1 indicates the corresponding R5SS is in lockstep mode.

6.1.3.2.1.2 R5 Core Halting and Unhalting

The R5SS*_CORE*_HALT register halts and unhalts the respective R5 Cores. Programming R5SS*_CORE*_HALT.HALT bitfield to 0x7 halts the respective R5 Core. Programming the bitfield to 0x0 unhalts the respective R5 Core.

6.1.3.2.1.3 R5 Wait-For-Interrupt (WFI)

The R5SS*_CORE*_STAT register provides the Wait-For-Event (WFE) and Wait-For-Interrupt (WFI) status of the respective R5 Cores.

R5SS*_CORE*_STAT.WFI_STAT = 1 indicates the respective R5 core is in WFI.

R5SS*_CORE*_STAT.WFE_STAT = 1 indicates the respective R5 core is in WFE.

6.1.3.2.2 Memory Initialization

The SRAM memories in the device are protected by SECDED ECC for functional safety. At bootup, the memory content must be initialized for ECC. Memory initialization can be triggered by the memory initialization registers.

6.1.3.2.2.1 R5 TCM Memory Initialization

R5SS*_ATCM_MEM_INIT. MEM_INIT bitfield, when programmed to 1, initializes the ATCM memories of the corresponding R5SS.

R5SS*_ATCM_MEM_INIT_STATUS. MEM_STATUS shows the status of memory initialization. If the bitfield reads ‘1’, it indicates the memory initialization is in progress.

R5SS*_ATCM_MEM_INIT_DONE. MEM_INIT_DONE bitfield is set when the memory initialization is complete. Writing ‘1’ to this field clears the field.

R5SS*_BTCM_MEM_INIT , R5SS*_BTCM_MEM_INIT_STATUS and R5SS*_BTCM_MEM_INIT_DONE are associated with R5SS BTCM memory initialization.

6.1.3.2.2.2 L2 OCRAM and Mailbox RAM and EDMA RAM Memory Initialization

The L2OCRAM_MEM_INIT register triggers ECC initialization of L2OCRAM.

L2OCRAM is split into four banks.

L2OCRAM_MEM_INIT.PARTITION0, when programmed to 1, triggers the initialization of Bank0 of L2 OCRAM.

Similarly PARTITION1, 2, and 3 bit fields trigger initialization of Bank1, 2, 3 of L2 OCRAM, respectively.

The L2OCRAM_MEM_INIT_STATUS register shows the status of memory initialization. If the bitfield reads ‘1’, it indicates the memory initialization is in progress.

L2OCRAM_MEM_INIT_DONE.PARTITIONx bitfield is set when the memory initialization of corresponding bank of L2OCRAM is complete. Writing ‘1’ to this field clears the field.

The MAILBOXRAM_MEM_INIT, MAILBOXRAM_MEM_INIT_STATUS and MAILBOXRAM_MEM_INIT_DONE registers are associated with Mailbox RAM ECC Initialization

The TPCC_MEM_INIT, TPCC_MEMINIT_STATUS, and TPCC_MEM_INIT_DONE registers are associated with EDMA memory initialization.

6.1.3.2.3 EDMA Configuration

6.1.3.2.3.1 EDMA Global Configuration and Event Aggregation

The register TPTC_DBS_CONFIG configures the burst size of the DMA transfer. The bitfields TPTC_A0 and TPT_A1 configure the burst size of TPTC00 and TPTC01, respectively.

The registers TPCC0_INTAGG_MASK, TPCC0_INTAGG_STATUS, and TPCC0_INTAGG_STATUS_RAW are associated with the aggregated interrupt from EDMA TPCC0_INTAGGR. The TPCC0_INTAGG_MASK register can be configured to mask unwanted interrupt sources from EDMA from triggering the TPCC0_INTAGGR interrupt.

The TPCC0_INTAGG_STATUS register indicates the status of interrupt sources which caused the TPCC0_INTAGGR interrupt to occur. The TPCC0_INTAGG_STATUS_RAW register indicates the raw status of interrupt sources of the TPCC0_INTAGGR interrupt.

6.1.3.2.3.2 EDMA Error Aggregation

The registers TPCC0_ERRAGG_MASK, TPCC0_ERRAGG_STATUS, and TPCC0_ERRAGG_STATUS_RAW are associated with the aggregated interrupt from EDMA TPCC0_ERRAGGR. The TPCC0_ERRAGG_MASK register can be configured to mask unwanted interrupt sources from EDMA from triggering the TPCC0_ERRAGGR interrupt. The TPCC0_ERRAGG_STATUS register indicates the status of interrupt sources which caused the TPCC0_ERRAGGR interrupt to occur. The TPCC0_ERRAGG_STATUS_RAW register indicates the raw status of interrupt sources of the TPCC0_ERRAGGR interrupt.

6.1.3.2.4 CPSW Global Configuration

The CPSW_CONTROL register is used for global configuration of CPSW modes.

The CPSW_CONTROL.PORT*_MODE_SEL bitfield configures the Ethernet mode of the corresponding port of CPSW to be in either MII, RMII, or RGMII.

CPSW_CONTROL.RGMII*_ID_MODE, when set to 1, enables the internal delay mode for the transmit path of the corresponding RGMII port. This provides a phase shift of quarter cycle b/w clock and data.

CPSW_CONTROL.RMII*_REF_CLK_OE_N controls how the RMII REF_CLK is generated in the system.

As shown in Figure 6-3, the RMII*_REF_CLK can be generated from the device and fed to the CPSW and transmitted out to device pins. Alternately, the RMII*_REF_CLK can be sourced from external sources as an input to the device.

CPSW_CONTROL.RMII*_REF_CLK_SEL is used to select the RMII*_REF_CLK source, either from the IO pad (write 0x0) or from an internal source (write 0x1).

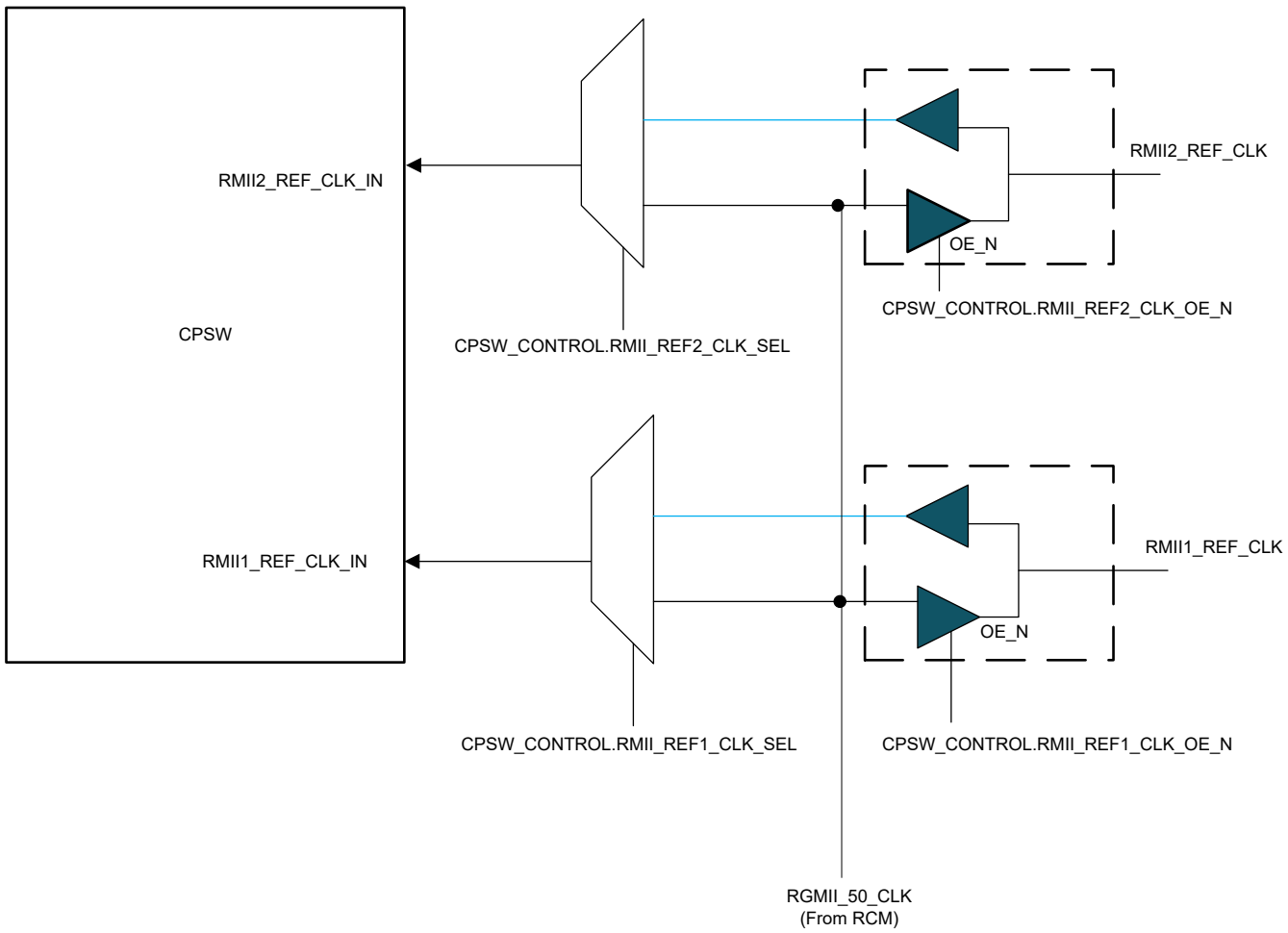


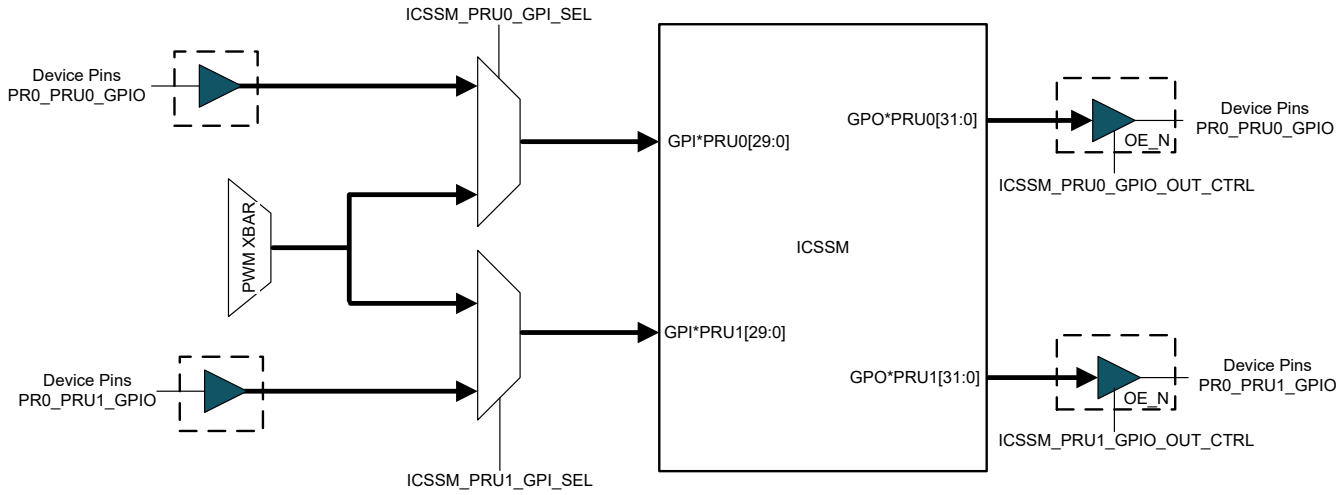
Figure 6-3. CPSW Configuration

6.1.3.2.5 ICSSM Global Configuration

The ICSSM*_IDLE_CONTROL register enables dynamic power saving in ICSSM*. When ICSSM*_IDLE_CONTROL.NOGATE is programmed to ‘0’, it enables auto clock gating in ICSSM* with increased access latency. When this bit is programmed to ‘1’, the clock is continuously active with low latency access.

The GPI signals of ICSSM* can be sourced either from device pins or from PWM_XBAR . This selection can be done on a per signal basis using the registers ICSSM*_PRU*_GPI_SEL, as shown in Figure 6-4.

When the pinmux is configured to choose ICSSM* GPIO function (PR0_PRU*_GPIO*), the control of the output buffer of the device pin can be done using the registers ICSSM*_PRU*_GPIO_OUT_CTRL, as shown in Figure 6-4.



Note: Not all GPI and GPO signals of ICSSM are available on Device spins. Refer to device specific data sheet for the signals available on pinmux.

Figure 6-4. ICSSM Configuration

6.1.3.2.6 GPMC Global Configuration

The register GPMC_CONTROL configures the GPMC I/O clock source and GPMC feedback clock source, as shown in Figure 6-5.

GPMC_CONTROL.CLKOUT_SEL, when programmed to 0, selects the GPMC_FUNC_CLK from RCM as the I/O clock GPMC_CLK at the device pin. If a free running clock is desired at the device pin, the application must program this bit field to 0.

GPMC_CONTROL.CLKOUT_SEL, when programmed to 1, selects the divided clock from GPMC module as the I/O clock GPMC_CLK. In this case, the I/O clock is not free running and is only active during GPMC I/O transfers.

GPMC_CONTROL.CLK_LB_SEL configures the source of I/O loop back clock. Based on the system, timing, and noise careabouts, the application can configure the I/O loop back clock.

GPMC_CONTROL. CLK_OE_N and GPMC_CONTROL. CLK_LB_OE_N enable the output I/O buffer of GPMC_CLK and GPMC_LB_CLK.

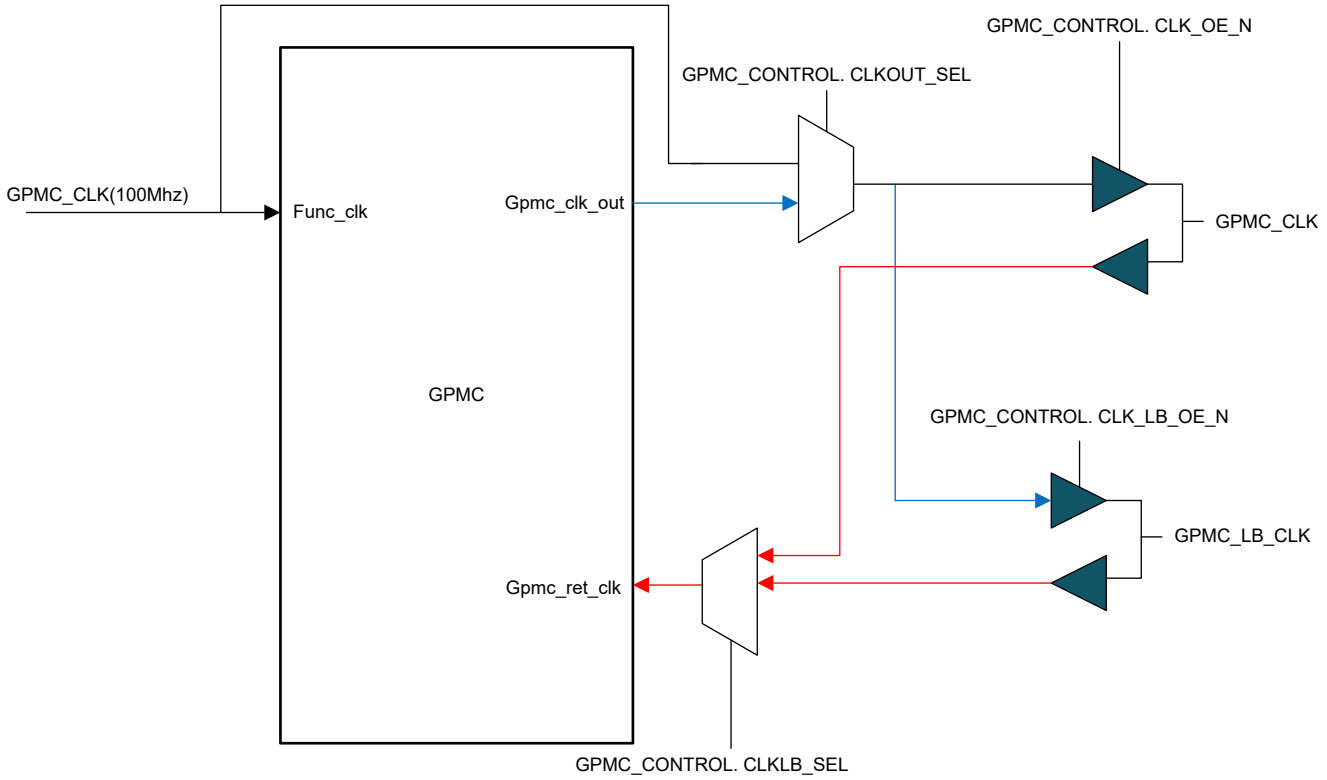


Figure 6-5. GPMC Configuration

6.1.3.2.7 MPU Interrupt Aggregator

The Memory Protection Units (MPU) are present on various module ports. Each MPU can generate two kinds of error types: an address error and a protection error. Refer to the Section 3.12.3.3 for a description of these errors.

The address errors from all MPU are aggregated and generate one interrupt R5SS*_CORE*_MPU_ADDR_ERRAGG to each R5 Core. Similarly, the protection errors from all MPU are aggregated and generate one interrupt R5SS*_CORE*_MPU_PROT_ERRAGG to each R5.

The interrupt to each R5 can be independently configured to select the MPU sources, which should generate the above interrupts.

The registers MPU_ADDR_ERRAGG_R5SS*_CPU*_MASK , MPU_ADDR_ERRAGG_R5SS*_CPU*_STATUS, and MPU_ADDR_ERRAGG_R5SS*_CPU*_STATUS_RAW are associated with R5SS*_CORE*_MPU_ADDR_ERRAGG interrupt to the respective R5F core.

The register MPU_ADDR_ERRAGG_R5SS*_CPU*_MASK configures interrupt sources which can generate the ADDR_ERR interrupt to the respective R5 core. MPU_ADDR_ERRAGG_R5SS*_CPU*_STATUS register indicates the status of the source which caused the ADDR_ERR interrupt to the respective R5 Core.

The MPU_ADDR_ERRAGG_R5SS*_CPU*_STATUS_RAW register indicates the raw status of all possible interrupt sources which can generate the ADDR_ERR interrupt.

The registers MPU_PROT_ERRAGG_R5SS0_CPU0_MASK, MPU_PROT_ERRAGG_R5SS0_CPU0_STATUS, and MPU_PROT_ERRAGG_R5SS0_CPU0_STATUS_RAW are associated with R5SS*_CORE*_MPU_PROT_ERRAGG interrupt to the respective CPU.

The register MPU_PROT_ERRAGG_R5SS0_CPU0_MASK configures interrupt sources, which can generate the PROT_ERR interrupt to the respective R5 core. MPU_PROT_ERRAGG_R5SS0_CPU0_STATUS register indicates the status of source which caused the PROT_ERR interrupt to the respective R5 Core.

The MPU_PROT_ERRAGG_R5SS0_CPU0_STATUS_RAW register indicates the raw status of all possible interrupt sources which can generate the PROT_ERR interrupt.

6.1.3.2.8 MMR Access Error Interrupt Aggregator

Some of the SoC Control Modules generate MMR access error interrupts (see Section 6.1.1.2) as shown in Figure 6-6.

All control modules' MMR access error interrupts are aggregated and generate a single interrupt MMR_ACCESS_ERRAGGR to the R5 cores. The registers MMR_ACCESS_ERRAGG_MASK0, MMR_ACCESS_ERRAGG_STATUS0, and MMR_ACCESS_ERRAGG_STATUS_RAW0 are associated with this interrupt.

The MMR_ACCESS_ERRAGG_MASK0 register selects the sources which can generate the MMR_ACCESS_ERRAGGR interrupt. The MMR_ACCESS_ERRAGG_STATUS0 register indicates the status of interrupt sources which caused the MMR_ACCESS_ERRAGGR interrupt to occur. The MMR_ACCESS_ERRAGG_STATUS_RAW0 register indicates the raw status of all interrupt sources which can cause MMR_ACCESS_ERRAGGR interrupt to occur.

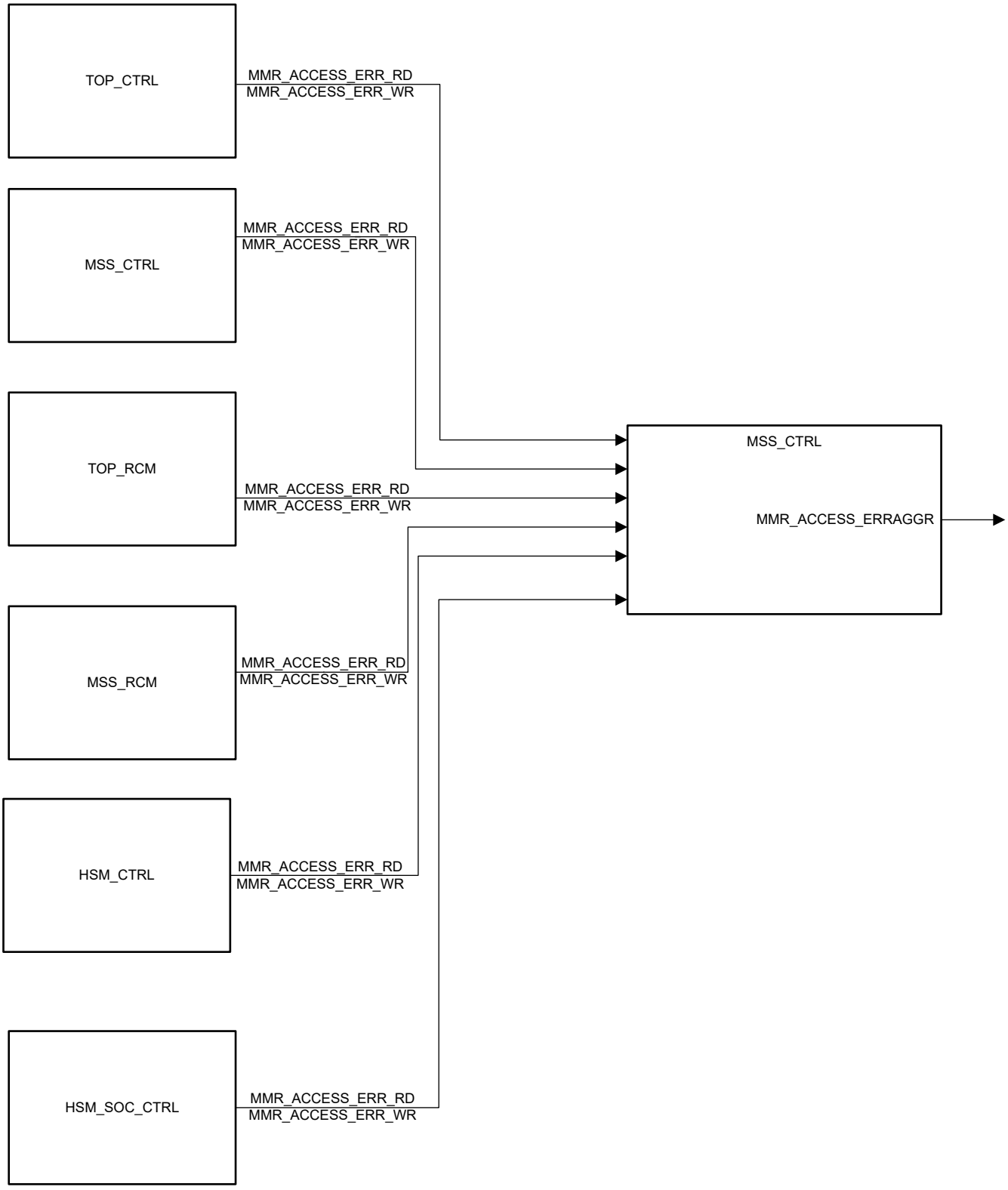


Figure 6-6. MMR Access Error Interrupt Aggregator

6.1.3.2.9 Safety Registers

The following sections detail the Safety Registers in the device.

6.1.3.2.9.1 R5 Memory ECC Error Aggregator

ARM R5 Cores support ECC error detection on the R5 associated memories – ICache, DCache, and TCM memories. These errors are visible on the ARM R5 Event bus EVNTBUS[*] (refer to ARM documentation for more details on the event bus interface). These errors are aggregated in the Control module and presented as two errors to ESM per R5 Core:

- R5SS*_CORE*_CORR_ERRAGG : Correctable ECC Errors, one per R5 Core
- R5SS*_CORE*_UNCORR_ERRAGG : Uncorrectable/Fatal ECC Errors, one per R5 Core

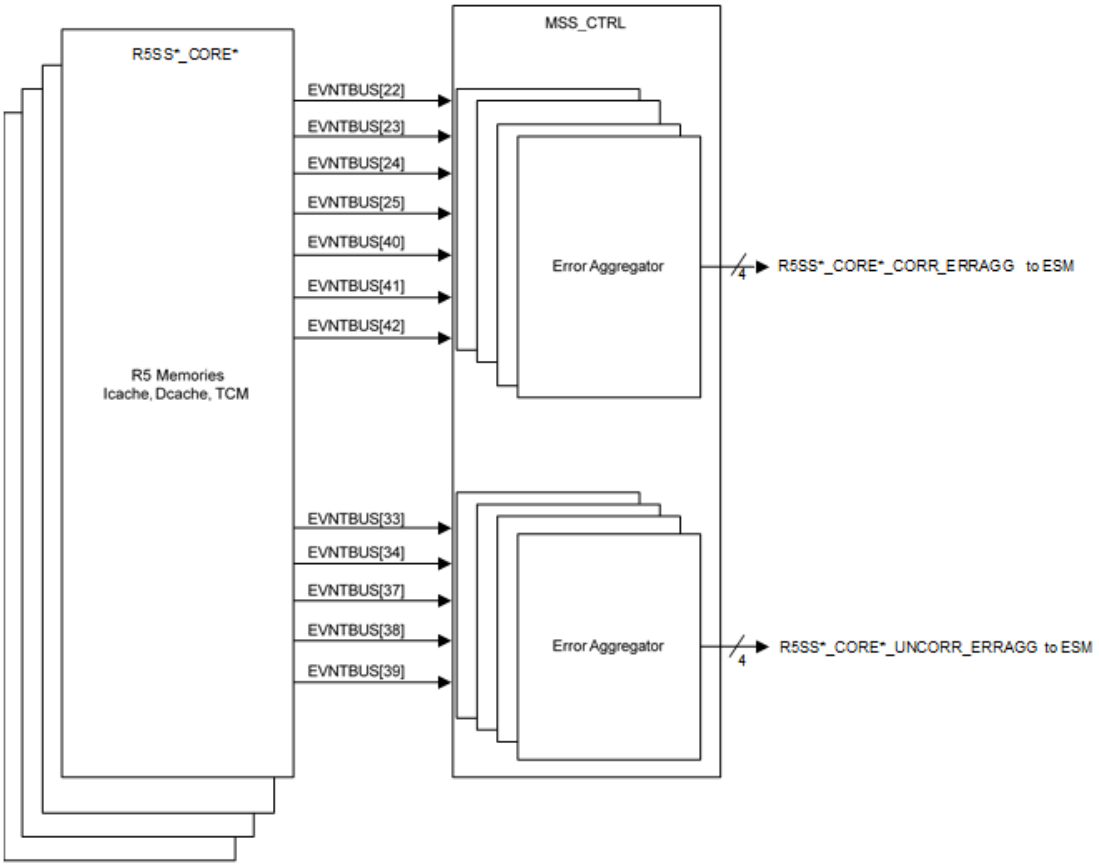


Figure 6-7. R5 Memory ECC Error Event Interrupt Aggregator

The following registers are associated with R5SS*_CORE*_CORR_ERRAGG:

- R5SS*_CPU*_ECC_CORR_ERRAGG_MASK – Error Mask Register
- R5SS*_CPU*_ECC_CORR_ERRAGG_STATUS – Error Status Register/Clear Register
- R5SS*_CPU*_ECC_CORR_ERRAGG_STATUS_RAW – Raw Error Status Register

Table 6-11 lists the register fields that control the generation of R5SS*_CORE*_CORR_ERRAGG Error.

Table 6-11. R5SS*_CORE*_CORR_ERRAGG Error Events

Event Flag	Event Mask	Description
R5SS*_CPU*_ECC_CORR_ERRAGG_STATUS[0]	R5SS*_CPU*_ECC_CORR_ERRAGG_MASK[0]	ATCM single-bit ECC error. From R5 event bus EVNTBUS[40] Register Field name - R5SS*_CPU*_ATCM_CORR_ERR
R5SS*_CPU*_ECC_CORR_ERRAGG_STATUS[1]	R5SS*_CPU*_ECC_CORR_ERRAGG_MASK[1]	B1TCM single-bit ECC error. From R5 event bus EVNTBUS[42] Register Field name - R5SS*_CPU*_B1TCM_CORR_ERR
R5SS*_CPU*_ECC_CORR_ERRAGG_STATUS[2]	R5SS*_CPU*_ECC_CORR_ERRAGG_MASK[2]	B0TCM single-bit ECC error. From R5 event bus EVNTBUS[41] Register Field name - R5SS*_CPU*_B0TCM_CORR_ERR
R5SS*_CPU*_ECC_CORR_ERRAGG_STATUS[3]	R5SS*_CPU*_ECC_CORR_ERRAGG_MASK[3]	Data cache tag or dirty RAM parity error or correctable ECC error. From R5 event bus EVNTBUS[24] Register Field name - R5SS*_CPU*_DTAG_CORR_ERR
R5SS*_CPU*_ECC_CORR_ERRAGG_STATUS[4]	R5SS*_CPU*_ECC_CORR_ERRAGG_MASK[4]	Data cache data RAM parity error or correctable ECC error. From R5 event bus EVNTBUS[25] Register Field name - R5SS*_CPU*_DDATA_CORR_ERR
R5SS*_CPU*_ECC_CORR_ERRAGG_STATUS[5]	R5SS*_CPU*_ECC_CORR_ERRAGG_MASK[5]	Instruction cache tag RAM parity or correctable ECC error. From R5 event bus EVNTBUS[22] Register Field name - R5SS*_CPU*_ITAG_CORR_ERR
R5SS*_CPU*_ECC_CORR_ERRAGG_STATUS[6]	R5SS*_CPU*_ECC_CORR_ERRAGG_MASK[6]	Instruction cache data RAM parity or correctable ECC error. From R5 event bus EVNTBUS[23] Register Field name - R5SS*_CPU*_IDATA_CORR_ERR

The following registers are associated with R5SS*_CORE*_UNCORR_ERRAGG:

- R5SS*_CPU*_ECC_UNCORR_ERRAGG_MASK – Error Mask Register
- R5SS*_CPU*_ECC_UNCORR_ERRAGG_STATUS – Error Status Register/Clear Register
- R5SS*_CPU*_ECC_UNCORR_ERRAGG_STATUS_RAW – Raw Error Status Register

Table 6-12 lists the register fields that control the generation of R5SS*_CORE*_UNCORR_ERRAGG Error.

Table 6-12. R5SS*_CORE*_UNCORR_ERRAGG Error Events

Event Flag	Event Mask	Description
R5SS*_CPU*_ECC_UNCORR_ERRAGG_STATUS[0]	R5SS*_CPU*_ECC_UNCORR_ERRAGG_MASK[0]	ATCM multi-bit ECC error. From R5 event bus EVNTBUS[37] Register Field name - R5SS*_CPU*_ATCM_UNCORR_ERR
R5SS*_CPU*_ECC_UNCORR_ERRAGG_STATUS[1]	R5SS*_CPU*_ECC_UNCORR_ERRAGG_MASK[1]	B1TCM multi-bit ECC error. From R5 event bus EVNTBUS[39] Register Field name - R5SS*_CPU*_B1TCM_UNCORR_ERR
R5SS*_CPU*_ECC_UNCORR_ERRAGG_STATUS[2]	R5SS*_CPU*_ECC_UNCORR_ERRAGG_MASK[2]	B0TCM multi-bit ECC error. From R5 event bus EVNTBUS[38] Register Field name - R5SS*_CPU*_B0TCM_UNCORR_ERR
R5SS*_CPU*_ECC_UNCORR_ERRAGG_STATUS[3]	R5SS*_CPU*_ECC_UNCORR_ERRAGG_MASK[3]	Data cache tag/dirty RAM fatal ECC error. From R5 event bus EVNTBUS[34] Register Field name - R5SS*_CPU*_DTAG_UNCORR_ERR
R5SS*_CPU*_ECC_UNCORR_ERRAGG_STATUS[4]	R5SS*_CPU*_ECC_UNCORR_ERRAGG_MASK[4]	Data cache data RAM fatal ECC error. From R5 event bus EVNTBUS[33] Register Field name - R5SS*_CPU*_DDATA_UNCORR_ERR

6.1.3.2.9.2 R5SS TCM Address Parity Error Aggregator

The R5_CORE can generate parity bits on the TCM address. R5SS implements a parity error detection mechanism and generates an error event if parity error is detected. These errors are aggregated in Control module and presented as one Error per R5_CORE to the ESM module - R5SS*_CORE*_TCM_ADDRPARITY_ERRAGG.

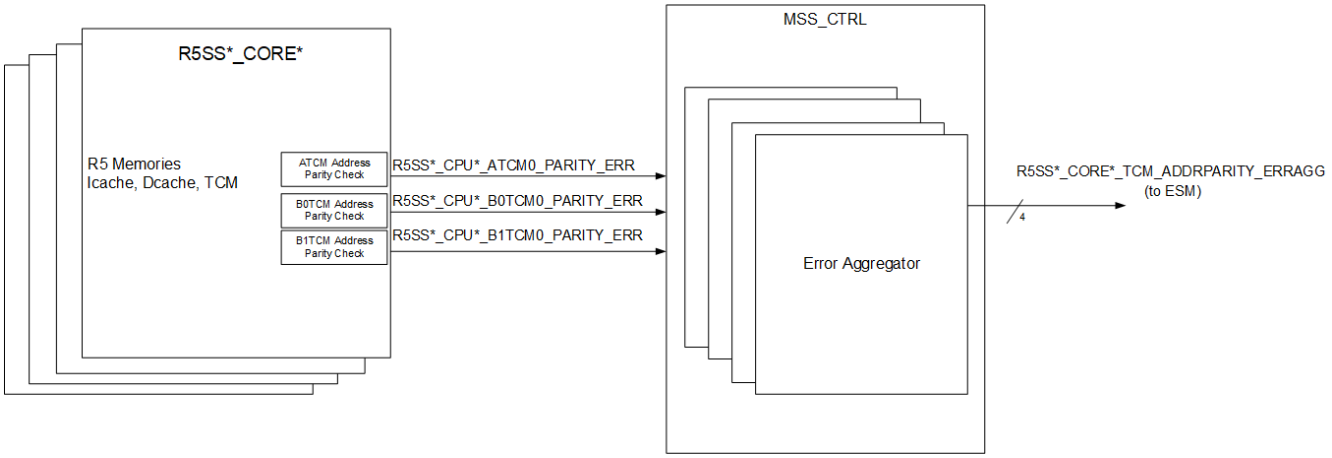


Figure 6-8. R5SS TCM Address Parity Error Aggregator

The following registers are associated with R5SS*_CORE*_TCM_ADDRPARITY_ERRAGG:

- R5SS*_CPU*_ECC_CORR_ERRAGG_MASK – Error Mask register
- R5SS*_CPU*_ECC_CORR_ERRAGG_STATUS - Error Status Register/Clear Register
- R5SS*_CPU*_ECC_CORR_ERRAGG_STATUS_RAW – Raw error status register
- R5SS*_CORE*_ADDRPARITY_ERR_ATCM – Latched Address of Parity Error location on ATCM Memory of respective R5 Core
- R5SS*_CORE*_ERR_ADDRPARITY_B0TCM - Latched Address of Parity Error location on B0TCM Memory of respective R5 Core
- R5SS*_CORE*_ERR_ADDRPARITY_B1TCM - Latched Address of Parity Error location on B1TCM Memory of respective R5 Core
- R5SS*_TCM_ADDRPARITY_CLR - Parity Error Address clear register for respective R5SS

Table 6-13 lists the register fields that control the generation of R5SS*_CORE*_TCM_ADDRPARITY_ERRAGG.

Table 6-13. R5SS*_CORE*_TCM_ADDRPARITY_ERRAGG Events

Event Flag	Event Mask	Description
R5SS*_CPU*_TCM_ADDRPARITY_ERRAGG_STATUS [0]	R5SS*_CPU*_TCM_ADDRPARITY_ERRAGG_MASK[0]	ATCM Address Parity Error. Register field - ATCM0_PARITY_ERR
R5SS*_CPU*_TCM_ADDRPARITY_ERRAGG_STATUS [1]	R5SS*_CPU*_TCM_ADDRPARITY_ERRAGG_MASK[1] B0TCM0_PARITY_ERR	B0TCM Address Parity Error. Register field - B0TCM0_PARITY_ERR
R5SS*_CPU*_TCM_ADDRPARITY_ERRAGG_STATUS [2]	R5SS*_CPU*_TCM_ADDRPARITY_ERRAGG_MASK[2]	B1TCM Address Parity Error. Register field - B1TCM0_PARITY_ERR

6.1.3.2.9.3 Interconnect Safety

Various MMR are present for detecting and injecting errors into VBUS interconnects.

- *_BUS_SAFETY_CTRL –
  - to enable the interconnect for safety
  - to clear the error status
  - top level idea of the available bus (cmd,wr,ws,rd) for the particular port and whether the port follows the VBUS protocol or not
- *_BUS_SAFETY_FI – To inject fault on

  - data bus - double error detection(ded) error
  - data bus - single error detection(sec) error
  - read, write, command and request bus on safe interconnect
  - read, write, command and request bus on main interconnect
- *_BUS_SAFETY_ERR – Error status register for sec, ded and comparison error. It also indicates if the fault injection has been do successfully done.
- *_BUS_SAFETY_ERR_STAT_* - Comparator status register for the respective bus

6.1.3.2.10 MSS_CTRL MMR Kick Protection Registers

The MSS_CTRL memory space is protected for writes using the kick registers as discussed in MMR Write Protection.

6.1.3.2.11 MSS_CTRL MMR Access Error Registers

The MSS_CTRL module can generate an Access Error interrupt which is associated with the following registers.

- INTR_RAW_STATUS - Interrupt Raw Status/Set register
- INTR_ENABLED_STATUS_CLEAR - Interrupt Enabled Status/Clear register
- INTR_ENABLE - Interrupt Enable register
- INTR_ENABLE_CLEAR - Interrupt Enable Clear register

See Section MMR Access Error Interrupt for details.

6.1.4 CONTROLSS_CTRL (CTRLMMR2)

This module consists of registers associated with the following functions:

- IP clock gating - Writing 3’b111 will gate clock for corresponding IP. Programmed as multibit.
- IP reset - Writing 3’b111 will generate reset for corresponding IP. Programmed as multibit.
- IP Halt
  - IP Halt disabled with corresponding CPU halt when programmed to 0
  - IP Halt enabled with corresponding CPU halt when programmed to 1

6.1.5 IOMUX (PADCFG_CTRLMMR0)

SoC-level terminal configuration control registers

Every device pinmux I/O pad is associated with a configuration MMR register <PAD_NAME>_CFG_REG. Table 6-14 describes each of these I/O pad configuration register fields.

Table 6-14. I/O Pad Configuration Register Fields

Register Field	Description
MSS_IOMUX..<PAD_NAME>_CFG_REG_func_sel	For selecting the input for the peripheral to pad mux or output of the pad to peripheral demux
MSS_IOMUX..<PAD_NAME>_CFG_REG_ie_override_ctrl	Active Low Input Override Control : Write 1 to select Active low Input Override value to control IOs IE_N/RXACTIVE_N instead of the control from hardware
MSS_IOMUX..<PAD_NAME>_CFG_REG_ie_override	Active Low Input Override
MSS_IOMUX..<PAD_NAME>_CFG_REG_oe_override_ctrl	Active Low Output Override Control : Write 1 to select Active low Output Override value to control IOs OE_N/GZ instead of the control from hardware
MSS_IOMUX..<PAD_NAME>_CFG_REG_oe_override	Active Low Output Override
MSS_IOMUX..<PAD_NAME>_CFG_REG_pupdsel	Pullup/PullDown Selection 0 -- Pull Down 1 - Pull Up
MSS_IOMUX..<PAD_NAME>_CFG_REG_pi	Pull Inhibit/Pull Disable 0 -- Enable 1- Disable
MSS_IOMUX..<PAD_NAME>_CFG_REG_sc1	Slew rate control : 0 : higher slew rate. 1: Lower slew rate.
MSS_IOMUX..<PAD_NAME>_CFG_REG_gpio_sel	R5F CPU ownership select for GPIO. 0 : GPO0, 1 :GPO1, 2 : GPO2, 3:GPO3
MSS_IOMUX..<PAD_NAME>_CFG_REG_qual_sel	select value for choosing input qualifer type for PAD. 00 : Sync, 01 : 3 Sample qual 10 : 6 Samples qual 11 : Async
MSS_IOMUX..<PAD_NAME>_CFG_REG_inp_inv_sel	select value for chosing inverted version of PAD input for chip: 0 : Non Inverted 1 : Inverted
MSS_IOMUX..<PAD_NAME>_CFG_REG_hsmode	MMR bits for HSMODE pin incase of true I2C pads
MSS_IOMUX..<PAD_NAME>_CFG_REG_hsmaster	MMR bits for HSMASTER pin incase of true I2C pads
MSS_IOMUX_QUAL_GRP_*_CFG_REG_qual_period_per_sample	MMR bits for programming the qualifier clock count per sample

6.1.6 TOPRCM (RCM_CTRLMMR0): SoC-level Clock and Reset control registers

The below Table 6-15 describes the SoC Level clock and Reset Control Registers. Refer to Section 6.3.2.2 for more information on Warm Reset.

Table 6-15. SoC Level Reset Registers

Control/Status Register	Description
TOP_RCM.WARM_RESET_CONFIG	Enable/disable individual warm reset sources
TOP_RCM.WARM_RESET_REQ	SW warm reset request
TOP_RCM.WARM_RST_CAUSE_CLR	Clear request for registered warm reset cause
TOP_RCM.WARM_RSTTIME1	When warm reset is triggered by internal warm reset sources, the time for which the warm reset pad pin has to be asserted low.
TOP_RCM.WARM_RSTTIME2	When warm reset is de-asserted externally, the time delay after which the external warm reset is de-asserted.
TOP_RCM.WARM_RSTTIME3	When warm reset is asserted externally, the time delay after which the external warm reset is asserted.
TOP_RCM.WARM_RST_CAUSE	Status register capturing which warm reset source caused the warm reset

The below Table 6-16 refers to the programmable values for WARM_RSTTIME1/2/3 that correspond to delays in the design.

Table 6-16. WARM_RSTTIMEx Programmable Delay Values

Programmable Value	Delay Value
0	500ns
1	1µs
2	2µs
3	4µs
4	8µs
5	16µs
6	32µs
7	64µs
8	128µs
9	256µs
10	512µs
11	1.024ms
12	2.048ms
13	4.096ms
14	8.192ms
15	16.384ms

Table 6-17. SoC Level Clock Registers

Control/Status Register	Description
TOP_RCM.x_CLK_SRC_SEL	Select line for selecting source clock for corresponding IP. Data should be loaded as multibit
TOP_RCM.x_CLK_DIV_VAL	Divider value for corresponding selected clock. Data should be loaded as multibit.
TOP_RCM.x_CLK_GATE	For gating the corresponding clock. writing '111' will gate clock for the IP
TOP_RCM.x_CLK_STATUS_clkinuse	Status shows the source clock selected for the corresponding clock
TOP_RCM.x_CLK_STATUS_currdivider	Status shows the current divider value chosen for the corresponding clock

6.1.7 MSS_RCM (RCM_CTRLMMR1): SoC and Peripheral-level Clock and Reset control registers

The below describes the SoC and Peripheral Clock and Reset Control Registers

Table 6-18. SoC and Preipheral Reset Registers

Control/Status Registers	Description
MSS_RCM.R5SSx_DBG_RST_EN	Controlsenable/disable of debug reset request to reset CORE0 and CORE1
MSS_RCM.R5SSx_RST_ASSERDLY	Controls the number of cycles reset should be kept asserted for R5SS resets.
MSS_RCM.R5SSx_RST2ASSERTDLY	Controls the number of cycles reset should be to wait before asserting R5SS resets.
MSS_RCM.R5SSx_RST_WFICHECK	Enable/disables if WFI is required before asserting R5SS resets.
MSS_RCM.R5SSx_RST_CAUSE_CLR	Clear the reset cause register
MSS_RCM.<IP>_RST_CTRL	Controlsthe individual IP reset generation
MSS_RCM.R5SSx_RST_STATUS	Statusregister capturing which event caused the corresponding R5SS reset

Table 6-19. SoC and Preipheral Clock Registers

Control/Status Registers	Description
MSS_RCM.x_CLK_SRC_SEL	Select line for selecting source clock for corresponding IP. Data should be loaded as multibit
MSS_RCM.x_CLK_DIV_VAL	Divider value for corresponding selected clock. Data should be loaded as multibit.
MSS_RCM.x_CLK_GATE	For gating the corresponding clock. writing '111' will gate clock for the IP
MSS_RCM.x_CLK_STATUS_clkinuse	Status shows the source clock selected for the corresponding clock
MSS_RCM.x_CLK_STATUS_currdivider	Status shows the current divider value chosen for the corresponding clock

6.2 Power

This chapter describes the power-management architecture implemented in the device.

The Power Management content is presented in several general sections:

- Power Management Subsystems (System and Control Modules)
- Device Power States
- Dynamic and Thermal Management

6.2.1 Power Management Overview

The chip consists of four voltage rails. There are two main external power supply for the device (3.3V and 1.2V). The external supply can be independently ramped up. Two additional 1.8V rails are generated by internal LDO.

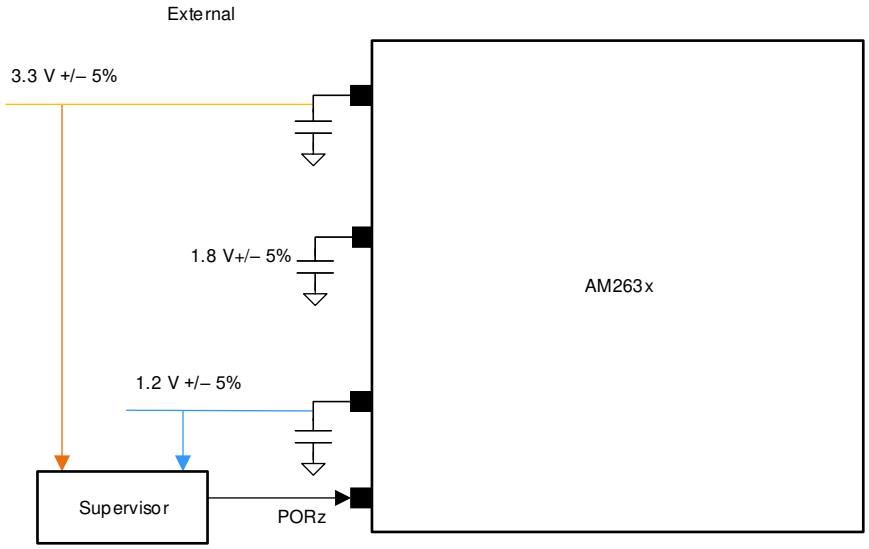


Figure 6-9. AM263Px Power Overview

The device consists of the following power supply modules

- BGAP – Reference voltage source for all voltage regulators and VREF's
- IO BIAS – To remove the requirement of supply sequence of IO by generating a 1.8V bias supply to IO from 3.3V
- 1.8V LDO – To generate 1.8V supply to analog modules from 3.3V
- Voltage Monitors – VMONs are implemented on following supplies:
  - VDD 1.2V
  - VDDA 1.8V
  - VDDS 3.3V
  - External supply monitor
  - IO BIAS
  - Reference Voltages
- POR – Delay the external input reset to the logic ult ill the VDDA 1.8V is within threshold. Generate isolation signals HHV.
- Thermal Manager (TM) module to monitor the device temperature.

The device supports clock control on all IP clocks. Unused clocks can be gated off by software to save power. See Section 6.4.1 for details

The On-Chip Static Random Access Memory (OCSRAM) banks can be powered down to save power. See section L2OCRAM power control.

Device power states are defined based on the state of external supply, as described in *Device Power States*.

Additional hardware modules for temperature sensing, thermal management, power monitoring, and reset functions related to power-on sequencing, are described in *Power System Modules*.

6.2.2 Power Management Unit

The Power Management Unit in AM263Px consists of a Reference system and a Safety system.

- **Reference System:**

  The Reference system generates internally used power supply and reference rails. It ensures reliable power on sequencing with the PMU and generates a reset signal based on coarse voltage level checks given

to the external power supplies. The Reference system also contains the 1.8-V LDO which generates a 1.8-V output on the VDDA18_LDO pin. The 1.8-V on VDDA18_LDO can be connected to VDDA18 and VDDA18_OSC_PLL on the board to provide the 1.8-V supply to the Analog Circuit and PLL.

- **Safety System:**

  The Safety System contains the safety comparators and temperature sensor to monitor the system supplies and temperature. The glitch filtered output of the voltage monitors are connected to the Error Signaling Module (ESM) and provide an error signal if the supply is not within the voltage thresholds set for the individual monitored supplies.

6.2.2.1 Power OK (POK) Modules

Voltage Comparator subsystem

POK modules are responsible for accurately detecting the voltage levels. Each module is trimmed to account for process and temperature variations. The trim values are provided by eFuse chains enabled by a POR module.

During POR, coarse monitors on supplies VDDA18, VDD12, 1.8V LDO, BGAP are enabled. The status of this can be monitored through TOP_CTRL.PMU_COARSE_STAT register during runtime.

The table below shows different voltage monitors available. Enabling/Disabling of this monitors can be controlled by TOP_CTRL.VMON_CTRL, TOP_CTRL.ADC_REF_COMP_CTRL register. There are corresponding status bits available in TOP_CTRL.VMON_STAT, TOP_CTRL:ADC_REF_GOOD_STATUS. The output of voltage monitors comparators are filtered using a configurable digital glitch filter module. The configuration of filter can be done using TOP_CTRL.VMON_FILTER_CTRL.SELECT_VALUE to select from no filtering option to max of 14.4us filtering of voltage monitor signals. The output of the voltage monitors are aggregated and the aggregated output is forward to ESM. Individual mask bits in TOP_CTRL.MASK_VMON_ERROR_ESM_L and TOP_CTRL.MASK_VMON_ERROR_ESM_H can be used to MASK the corresponding monitor to trigger ESM event.

POK is set to 1 when the voltage supply is within the range and goes to 0 when out of range. See device datasheet for POK tolerance.

The ESM event is generated only when POK signals out of range.

Voltage Comparator subsystem compares the sensed voltage level (VSENSE) to a reference voltage supply (VREF) against a reference voltage to generate a POWER good/OK (POK) signal. For all comparators, VSENSE is derived from the supply being monitored (VMON) through a resistor divider (Vsense=Vmon R2/(R1+R2))). Therefore, threshold value can be calculated as (Vth=Vref (R1+R2)/R2). For an under voltage comparator, if VMON>Vth then POK=1. Comparators have a decision range where this transition may occur. This decision range is influenced by the variation in reference voltage, resistor ratio and comparator offset due to the process variation and mismatch. Threshold should be set to a voltage level such that decision range of the comparator would be outside of the operating range of the supply.

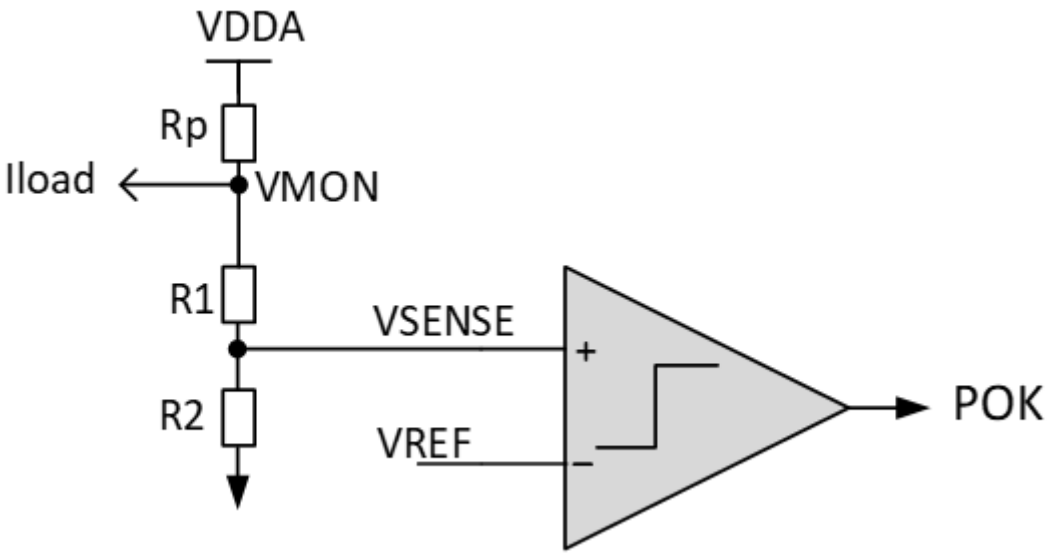


Figure 6-10. Voltage Comparator architecture

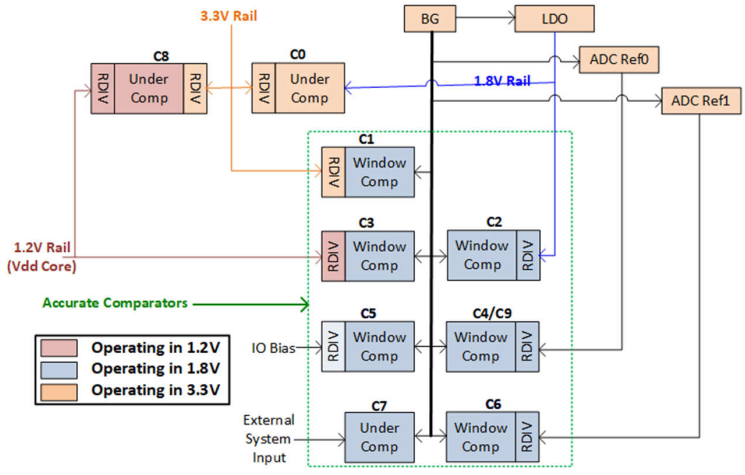


Figure 6-11. Voltage Comparator subsystem

Table 6-20. POK Module overview

Voltage Monitored	Comparator block	UV/OV[1]	Description
VDDA18	C0	UV	Voltage monitor for 1.8V LDO output using 3.3V as reference
VDDA18	C2	UV/OV	Voltage monitor for 1.8V LDO output using BGAP as reference
VBGAP09	C1	UV/OV	Voltage monitor for 0.9V bandgap.
VDD12	C3	UV/OV	Voltage monitor for 1.2V I/O supply
VDDSBIO	C5	UV/OV	Voltage monitor for 1.8V IO bias supply
VSYS_MON	C7	UV	Voltage monitor for external VSYS_MON

Table 6-20. POK Module overview (continued)

Voltage Monitored	Comparator block	UV/OV[1]	Description
VDDA33	C8	UV	Voltage monitor for 3.3V I/O supply
ADC0_REF	C4	UV/OV	Voltage monitor for ADC0_REF
ADC12_REF	C9	UV/OV	Voltage monitor for ADC12_REF
ADC34_REF	C6	UV/OV	Voltage monitor for ADC34_REF

6.2.2.2 Power on Reset module

The device relies on an external supervisor to ensure that the device external supplies (3.3V, 1.2V) are within range before releasing the reset to the device (PORz pin). Once the PORz is deasserted, the LDO to generate 1.8V analog is enabled.

The POR module monitors internally 3.3V, 1.2V external power supply as well internal 1.8V LDO and 0.9V bandgap voltages before internal reset is released to the system.

6.2.2.3 Thermal Manager

This section describes the Thermal Manager (TM) module in the device.

The TM module on the AM263Px enables thermal management of the device by providing control of on-chip temperature sensors.

The device has two temperature sensors, each located near critical hotspots in the device die. There are two additional temperature sensors at other locations in the device die.

Active temperature monitoring is available for two temperature sensors near the hotspots. The other two temperature sensors are only for temperature readout.

6.2.2.3.1 Thermal Manager Features

The Thermal Manager (TM) module supports the following features:

- Programming of temperature-crossing thresholds
- Signals when programed thresholds are exceeded (up to 3 alerts):
  - Temperature exceeding the TSENSE*_ALERT.ALERT_THRHLD_HOT for ALERT_HOT_INTR.
  - Temperature exceeding the TSENSE*_ALERT.ALERT_THRHLD_COLD for ALERT_LOW_THLD_BREACH_INTR.
  - Temperature below the TSENSE*_ALERT.ALERT_THRHLD_COLD for ALERT_HOT_INTR
- Supports up to 4 temperature monitors.
- Allows resolution of 2°C for temperature reading and threshold point temperature alert/interrupt generation.
- Maximum temperature alert.
- Supports one shot sampling mode for the sensors.
- Temp sense controller loops cyclically through each sensor and generates the results. Each sensor can be enabled/disabled independently.
- Provides register control and status for all 4 sensors. Interrupt generation, FIFO registers and alerts for 2 SOC temperature monitors.
- Default threshold are controlled through efuse values. This can be also controlled through programmable registers.
- There are four FIFOs used to store a brief history for the last few temperature measurements and are also dedicated to temperature time-stamping feature.
- Accumulator register for cumulative sum of past temperature measurements on 2 SOC temperature monitors.
- Warm reset generation when SOC temperature monitors exceed TSHUT_HOT.

6.2.2.3.2 Thermal Manager Functional Description

There are four temperature sensors on the device die. Each sensor is associated with one voltage domain and is also a part of a VBGAPTS cell. The VBGAPTS cell integrates bandgap voltage reference, temperature

sensor with ADC and thermal comparator shutdown. The 7-bit ADC produces a digital output, proportional to the temperature on SOC. Figure 6-12 shows the Thermal Management Functional Block Diagram.

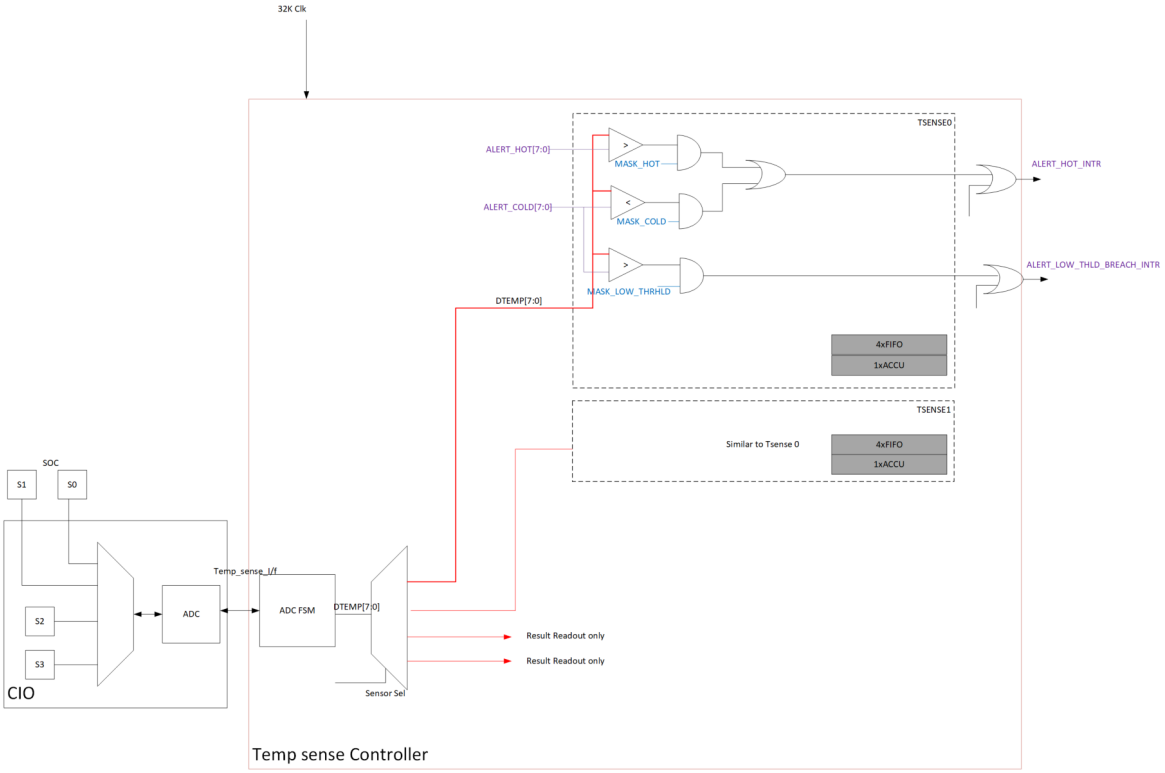


Figure 6-12. Thermal Management Functional Block Diagram

6.2.2.3.3 Thermal FSM

The Thermal FSM is clocked by the 32KHz clock. At reset the FSM is not enabled and can be enabled by configuring the TOP_CTRL.TSENSE_CFG register.

Software needs to configure the below register bits to enable the Temperature Sensor:

- TOP_CTRL.TSENSE_CFG.TMPSOFF – Temperature Sensor Controller OFF
- TOP_CTRL.TSENSE_CFG.BGROFF – Bandgap Reference OFF
- TOP_CTRL.TSENSE_CFG.AIPOFF - Temperature Sensor IP OFF
- TOP_CTRL.TSENSE_CFG.SNSR_MX_HIZ – Sensor mux select high impedence

By default these bits are set to 1, which disables the temperature sensor. To enable the temperature sensor these bits should be cleared (set to 0). Once the sensor is enabled, temperature measurement is initiated by enabling the FSM by writing 1 to TOP_CTRL.TSENSE_CFG.ENABLE.

Once enabled, the FSM will read out the temperature values from the sensors in a round robin fashion based on TOP_CTRL.TSENSE_CFG.SENSOR_SEL bitfield value. TOP_CTRL.TSENSE_CFG.SENSOR_SEL controls the enabling/disabling of individual sensors.

For each selected sensor, FSM requires anywhere between 51 to 54 clock cycles to start the sequence and register the result into TOP_CTRL.TSENSE*_RESULT.DTEMP register. TOP_CTRL.TSENSE_CFG.DELAY configures the number of clock cycles between end of result captured to FSM starting the sequence for the next enabled sensor. When the conversion is ongoing for a particular sensor, the corresponding TOP_CTRL.TSENSE*_RESULT.EOCZ status bits are set to 1. The EOCZ bit is reset to 0 again when the conversion completes. After this the valid temperature is written automatically by FSM in the TOP_CTRL.TSENSE*_RESULT.DTEMP bit fields, and then software is able to read it from the corresponding register. Figure 6-13 describes the sequence of sensor measurement based on SENSOR_SEL bits.

Note

Value 0 in TOP_CTRL.TSENSE_CFG.DELAY is not valid. A non-zero value should be programmed to this register.

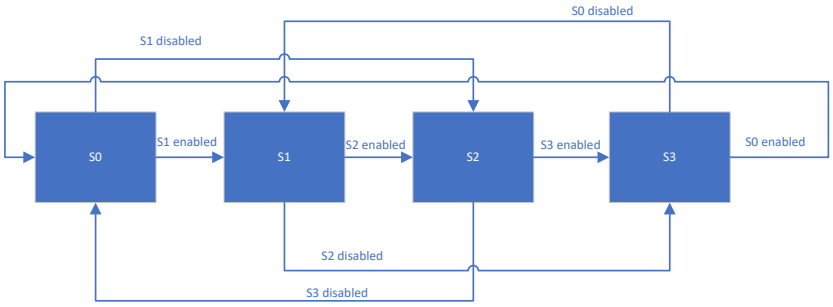


Figure 6-13. Sensor flow diagram

6.2.2.3.4 Thermal Alert Comparator

Thermal Comparators are implemented on the Temperature readouts to generate interrupts or ESM Errors. Alert indication generated by controller is shown in Figure 6-12.

Low and High Threshold Alert Mode:

This mode is used when two interrupts are needed at two temperature threshold points.

- When the temperature is less than TOP_CTRL.TSENSE*_ALERT.ALERT_THRHLD_COLD the interrupt ALERT_LOW_THLD_BREACH_INTR is triggered. The interrupt can be masked using TOP_CTRL.TSENSE*_CNTL.MASK_LOW_THRHLD which is set to 0 by default.
- When temperature is greater than TOP_CTRL.TSENSE*_ALERT.ALERT_THRHLD_HOT the interrupt ALERT_HOT_INTR is triggered. The interrupt is masked by default. It should be enabled by writing 0 to TOP_CTRL.TSENSE*_CNTL.MASK_HOT register bit.

In this mode of operation, TOP_CTRL.TSENSE*_CNTL.MASK_COLD should always be set to 1 by software, which is 0 by default, to disable cold interrupt condition.

The masked interrupt signals are also routed to the TOP_CTRL.TSENSE_STATUS register and the non-masked (raw) comparator outputs are available for reading through the corresponding bits in the TOP_CTRL.TSENSE_STATUS_RAW register.

Single Hot/Cold Alert mode:

This is an alternate mode of using the temperature sensor controller. In this mode ALERT_HOT_INTR is used for indicating the hot and subsequent cooldown condition.

In this mode TOP_CTRL.TSENSE*_CNTL.MASK_COLD should be set to 1 and TOP_CTRL.TSENSE*_CNTL.MASK_HOT should be set to 0. This will enable the interrupt for hot condition.

When the temperature exceeds the TOP_CTRL.TSENSE*_ALERT.ALERT_THRHLD_HOT register value, it triggers the ALERT_HOT_INTR interrupt. The interrupt will be asserted until masked by setting TOP_CTRL.TSENSE*_CNTL.MASK_HOT to 1. This will dessert the interrupt.

Software should additionally unmask the cold interrupt condition by setting register bit TOP_CTRL.TSENSE*_CNTL.MASK_COLD to 0. When the temperature cools down below the TOP_CTRL.TSENSE*_ALERT.ALERT_THRHLD_COLD register value, the interrupt ALERT_HOT_INTR is again triggered to indicate to the software that the device has cooled off sufficiently.

6.2.2.3.5 Thermal Shutdown Comparators

There is also a comparator block responsible for the thermal shutdown (TSHUT) function of the thermal management logic. This comparator block is also composed of two comparators. One dedicated to COLD TSHUT threshold and the other one to high TSHUT threshold.

The comparator outputs for the high TSHUT thresholds of each sensor are ORed and used to generate a single EVNT_TSENSE_CRITICAL error signal to ESM.The comparator outputs for the high and low TSHUT thresholds are used to generate TSHUT_RST0 pulse signal which asserts when high TSHUT is active and deasserts when low TSHUT is active. TSHUT_RST* signasl from both sensors are routed as warm reset source to reset the chip.

The TSHUT_HOT and TSHUT_COLD threshold values are by default controlled from EFUSE. TOP_CTRL.TSENSE*_TSHUT registers can be used to override the value from EFUSE and apply the same from the bit fields of TOP_CTRL.TSENSE*_TSHUT registers.

6.2.2.3.6 Temperature Timestamp Registers

Each time one of the TOP_CTRL.TSENSE*_RESULT.DTEMP bit fields is updated with new temperature value, this value is also automatically stored into a 4-level deep FIFO and a timestamp is registered too. There are four FIFOs used to store a brief history for the last few temperature measurements and are also dedicated to temperature timestamping feature. Each FIFO has two fields.

The first one is 8 bits wide, 4 levels deep, and is intended to store the temperature values for the last four measurements. The second field is 22 bits wide, 4 levels deep, and acts like a counter for the number of temperature measurements. Each FIFO is composed of the following registers:

- TOP_CTRL.TSENSE*_DATA0
- TOP_CTRL.TSENSE*_DATA1
- TOP_CTRL.TSENSE*_DATA2
- TOP_CTRL.TSENSE*_DATA3

6.2.2.3.7 FIFO Management

Software can stop a certain FIFO to update with new temperature and timestamp values by setting one of the FREEZE bits in the TOP_CTRL.TSENSE0_CNTL and TOP_CTRL.TSENSE1_CNTL registers to 1. These FIFO_FREEZE bits are automatically cleared by hardware after the FIFOs are cleared.

Each FIFO is cleared by setting to 1 one of the FIFO_CLEAR bits in the TOP_CTRL.TSENSE0_CNTL and TOP_CTRL.TSENSE1_CNTL registers. These FIFO_CLEAR bits are also automatically set by hardware to 0 after the FIFOs clearing procedure completes.

Additionally TOP_CTRL.TSENSE0_ACCU and TOP_CTRL.TSENSE1_ACCU registers store the accumulated temperature values. This are cleared by setting one of the ACCU_CLEAR bits to 1 in the TOP_CTRL.TSENSE0_CNTL and TOP_CTRL.TSENSE1_CNTL registers.

6.2.2.3.8 ADC Values Versus Temperature

Table 6-21 provides all the valid ADC values which correspond to the temperature measured which is read from the TOP_CTRL.TSENSE*_DATA*.DATA, TOP_CTRL.TSENSE*_RESULT.DTEMP bit fields. The table also provides the values for the temperature thresholds which are configurable through the TOP_CTRL.TSENSE*_ALERT.ALERT_THRHLD_COLD, TOP_CTRL.TSENSE*_ALERT.ALERT_THRHLD_HOT, TOP_CTRL.TSENSE*_TSHUT.TSHUT_THRHLD_COLD & TOP_CTRL.TSENSE*_TSHUT.TSHUT_THRHLD_HOT bit fields.

Table 6-21. ADC Values Versus Temperature

ADC code	Temperature	ADC code	Temperature	ADC code	Temperature
0-24	150	56	86	88	22
25	148	57	84	89	20
26	146	58	82	90	18
27	144	59	80	91	16
28	142	60	78	92	14
29	140	61	76	93	12
30	138	62	74	94	10
31	136	63	72	95	8
32	134	64	70	96	6
33	132	65	68	97	4
34	130	66	66	98	2
35	128	67	64	99	0
36	126	68	62	100	-2
37	124	69	60	101	-4
38	122	70	58	102	-6
39	120	71	56	103	-8
40	118	72	54	104	-10
41	116	73	52	105	-12
42	114	74	50	106	-14
43	112	75	48	107	-16
44	110	76	46	108	-18
45	108	77	44	109	-20
46	106	78	42	110	-22
47	104	79	40	111	-24
48	102	80	38	112	-26
49	100	81	36	113	-28
50	98	82	34	114	-30
51	96	83	32	115	-32
52	94	84	30	116	-34
53	92	85	28	117	-36
54	90	86	26	118	-38
55	88	87	24	119-128	-40

Note

Based on the characterization data, the Temperature mentioned in table can be offsetted by 8 degree C.

6.2.2.4 Power Control Modules

The Power Control Modules are divided into two section, dependent on their functionality - Clock ICG control section and L2OCRAM power control section.

6.2.2.4.1 Clock ICG controls

Clock ICG control manages the clock to each of the IPs using software control. For each module, there is a dedicated register <IP>_CLK_GATE part of MSS_RCM register space. By default all module clocks are enabled. Writing 0x7 into field GATED of corresponding <IP>_CLK_GATE will disable the clock to the IP.

Additionally, TOP_RCM host clock gating register R5SS0_CLK_GATE and R5SS1_CLK_GATE to disable core clock to individual R5SS. SYS_CLK_GATE disable SYS_CLK to the whole system. It is not recommended to gate R5SS_CLK and SYS_CLK as the system will hang and only option is to reset the whole system. TOP_RCM also allows clock gating for TRACE_CLK and CLKOUT0/CLKOUT1 ports.

6.2.2.4.2 L2OCRAM Power Control

There are 6 memory banks each of 512KB available as L2OCRAM.

By default all of this banks are in Power ON.

Individual L2OCRAM banks can be powered OFF using software writes to MSS_RCM.L2OCRAM_BANK*_PD_CTRL register and by observing status bits from MSS_RCM.L2OCRAM_BANK*_PD_STATUS

Sequence to power off each bank is captured below

1. Write 0x7 to ISO field of the register.
2. Write 0x0 to AONIN field of the register.
3. Wait till AONOUT status field is 0x0.
4. Write 0x0 to AGOODIN field of the register.
5. Wait till AGOODOUT status field is 0x0.

Sequence to Power On each bank is capture below

1. Write 0x7 to AONIN field of the register.
2. Wait till AONOUT status field is 0x1.
3. Write 0x7 to AGOODIN field of the register.
4. Wait till AGOODOUT field is 0x1.
5. Write 0x0 to ISO field of the register.

When a block is powered off, a bus error is generated on access.

Note

It is always safe to have decent delay between each step because memory might take some time before reaching to total power on state. So even though aonout is 1’b1 does not mean memory is ON.

6.2.2.5 Device Power States

6.2.2.5.1 Overview of Device Power Modes

The device supports multiple power modes described in the sections below. The lower power consumption comes at the cost of longer time for recovery to a running mode.

Figure 6-14 depicts the valid power modes for the device.

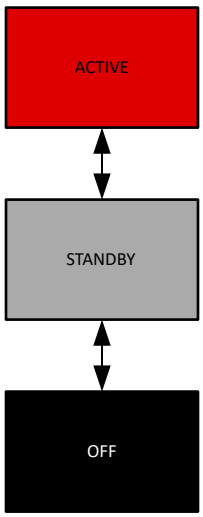


Figure 6-14. Power Modes

Device supports the following power states:

- ACTIVE: Main Status and Control registers are ON, processors and subsystems are ON, based on use cases.
- STANDBY: Main processor cores in WFI/WFE, rest of active subsystems idles.
- OFF: entire device is off by switching off voltage rails externally.

6.2.2.5.2 Device Power States and Transitions

The LPMs always transition from run state. If the device software decides to transition from one low power mode to another low power mode, it needs to go through the run state first.

Normally, the software initiates transition to low power mode and interrupts/events are the exit trigger.

Figure 6-15 shows the state diagram of transitions between LPMs.

This device supports three power states: OFF, ACTIVE, and STANDBY. STANDBY mode here is referred to for WFI/WFE execution from individual R5SS and clock gating enabled for all/required peripherals. Exit from Standby will restart the individual R5SS clocks but the IP clocks are not automatically switched ON. They need to be programmed in MSS_RCM.<IP>_CLK_GATE register to enable/disable. The transitioning schemes are as follows:

- OFF → ACTIVE
- STANDBY → ACTIVE
- ACTIVE → OFF
- ACTIVE → STANDBY

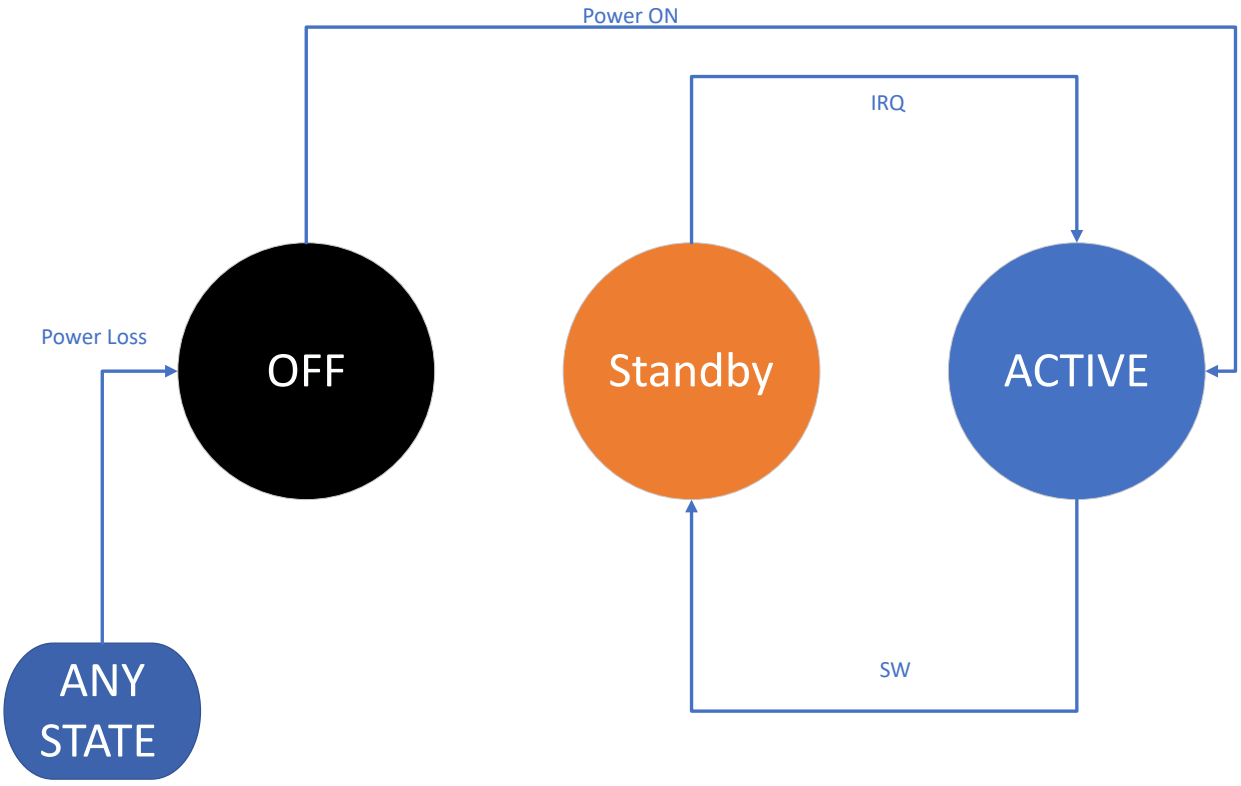


Figure 6-15. Transition between Power Modes

6.3 Reset

This chapter describes the device reset signals and contains details on reset management.

6.3.1 Overview

At a high-level, Resets are designed to bring a device or subsystem into a predetermined or known state. Resets are triggered in our device after power-up events, as well as upon various software and hardware reset requests. They are primarily used for system initialization, error detection, and debugging purposes. This chapter introduces the various reset capabilities available in the device and their functionality.

Reset Architecture Block Diagram

This is the device reset architecture block diagram. It represents the devices reset sources and critical internal signal connections each of which are discussed in the subsequent sections.

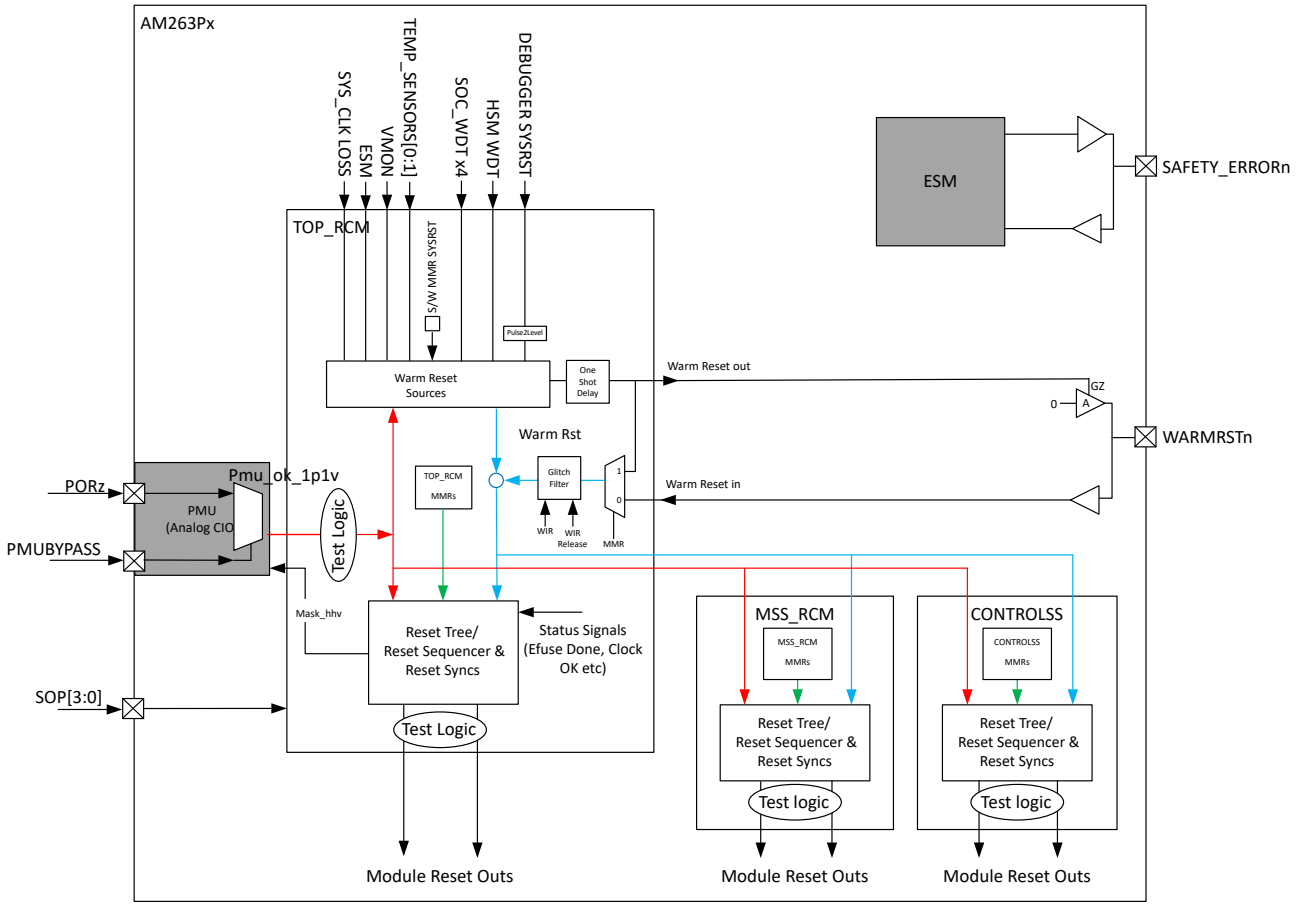


Figure 6-16. AM263Px Reset Architecture Block Diagram

Note

ESM (Error Signaling Module) aggregates all safety related events from throughout the SoC and gives out an error signal to the ‘SAFETY_ERRORn’ pin in the case of an error.

Note

‘Wait In Reset’ (WIR) signal from Debugss POWER AP extends the Warm Reset till the WIR signal is deasserted (Becomes HIGH). ‘Release from WIR’ signal deasserts the Warm Reset.

6.3.1.1 SoC Supported Resets

There are various resets supported by the SoC, each of which are explained below.

- **Power-On-Reset:**

  The device Power-on-Reset (POR) resets all the logic in the SoC without any exceptions. This reset is controlled by an external pin ***‘ PORz’*** which is driven by an external (off-chip) "Power-Good" Circuit or Power

Management IC (PMIC). The PORz pin should be held active LOW (0) until all power supplies are stable. It should also be driven low whenever the external PMIC detects that the 3.3V /1.2V supply is not in range. The system comes out of the reset only after an additional delay owing to efuse shifting and High Frequency Oscillator (XTAL) clock stabilization.

- **Warm Resets:**

  The device Warm Reset resets only the logic sensitive to warm reset and does not affect the logic that are ‘ Reset only on PORz**’** . Warm reset can be triggered by certain internal reset sources and also by asserting the ‘WARMRSTn’ pin externally. Additionally, the warm reset is brought out on ‘WARMRSTn’ pin to assert reset on external board components. When the pin is LOW, it indicates that the system is in a warm reset state. When HIGH, it indicates that the system is out of warm reset.

- **Local Module Resets:**

  These are module level resets programmed through software using the MMRs in RCM modules, only intended for debug purposes. They are uncontrolled resets and have potential side-effects (like pending interrupts, pending bus transactions, pending DMA triggers) that will impact the rest of the SOC. Hence, it is not recommended to use these resets in production and functional mode.

6.3.2 Reset Details

The Reset Details section breaks down the available device resets and also explains operating details such as timing diagrams.

This table summarizes the available reset sources that are supported by the device.

Table 6-22. Device Reset Sources

Reset Name	Reset Type	Sync/ Async	Pin/Register/ Internal	Details
PORz	Hardware POR	Async	PORz HW Pin	PORz Reset
WARMRSTn	Hardware Warm Reset	Async	WARMRSTn HW Pin	WARMRSTn
SW_WARMRSTn	Software Warm Reset	Async	TOP_RCM.WARM_RESET_REQ	SW_WARMRSTn
WDT Reset	WDT Reset	Async	Internal signal	WDT Resets
VMON Reset	VMON Reset	Async	Internal signal	Voltage Monitor Error Reset
ESM Reset	ESM Reset	Async	Internal signal	ESM Reset
SYS_CLK clock loss Reset	SYS_CLK clock loss Reset	Async	Internal signal	Section 6.3.2.2.2.2
Thermal Reset	Thermal Reset	Async	Internal signal	Thermal Alert Reset
Debugger Reset	Debugger Reset	Async	Internal signal	Debugger Reset
Local Resets	Software Reset	Async	RCM MMRs	Local Module Resets

6.3.2.1 PORz Reset

The external pin 'PORz' is the primary power on reset input (active LOW) to the entire device. When LOW, it performs a POR on the entire device and puts all IOs in a safe state (Reset/HHV state). Upon PORz deassertion, IOs will enter the default state defined in the Device Datasheet and the boot process will be initiated.

Timing sequence below shows the reset sequence for WARMRSTn pad and the Internal Reset to System during PORz deassertion.

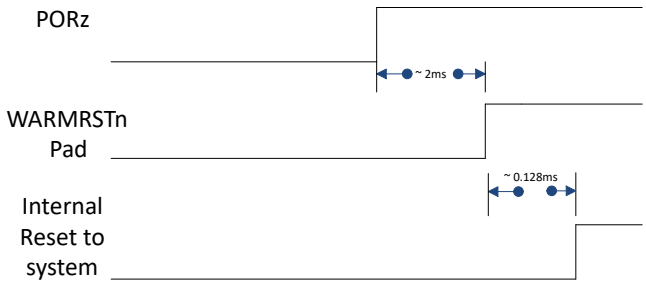


Figure 6-17. PORz timing sequence

Note

MMRs which get 'reset by PORz only' are captured in register description of those registers.

Note

SOP pin pull ups/pull downs which are needed to configure the boot mode should be held steady during the PORz assertion.

6.3.2.2 Warm Resets

When a warm reset LOW is detected, Reset Hardware generates an internal warm reset to the system logics working on warm reset (Except for logics which are reset only by PORz). All IO configurations are reset during warm reset assertion.

The following are the sources which can trigger a system warm reset in the device.

1. PORz (Reset by PORz Hardware Pin)
2. WARMRSTn (Reset by WARMRSTn Hardware Pin)
3. SW MMR in RCM (Reset by TOP_RCM.WARM_RESET_REQ)
4. WDT reset (Reset by 4x SoC WDTs or HSM WDT)
5. Debugger reset (Reset by 'SYSRESET' from debugger)
6. Thermal reset (Tempsensor[0:1] reset)
7. VMON reset
8. ESM reset
9. SYS_CLK clock loss reset

The cause for the warm reset is captured in TOP_RCM.WARM_RST_CAUSE register. Reset status bit reads active HIGH (1) when a particular reset is triggered. After reset is deasserted, device will boot-up and software can read the register to check the reset cause. TOP_RCM.WARM_RST_CAUSE_CLR should be written 3'b111 to clear the status bits. A PORz assertion also clears status register.

Except PORz source, all other sources can be enabled and disabled individually. The timing sequence for internal warm reset source, WARMRSTn Pad and internal system reset are discussed in the following sections.

6.3.2.2.1 Warm Reset by WARMRSTn HW Pin

This reset pin is the warm reset request (active LOW) given externally from the pad..

By default, the input path to trigger a warm reset from external pad is disabled. To enable, the TOP_RCM.WARM_RESET_CONFIG.PAD_BYPASS bit should be written 3'b000.

The timing diagram shows the reset sequence during WARMRSTn pad assertion and related timing for the internal system reset.

The input pad signal should remain LOW for at least 'TOP_RCM.WARM_RSTTIME3' time to register an assertion of WARMRSTn. Similarly,the signal should remain HIGH for at least 'TOP_RCM.WARM_RSTTIME2' continuously to register a deassertion of WARMRSTn. The glitch filter logic on 'Warm_Reset_in' filters out any input pad signal which is LOW for less than 'TOP_RCM.WARM_RSTTIME3' time and HIGH for less than 'TOP_RCM.WARM_RSTTIME2' time. The internal system reset gets asserted at time TOP_RCM.WARM_RSTTIME3 and deasserted at 'TOP_RCM.WARM_RSTTIME2' time relative to external WARMRSTn pad

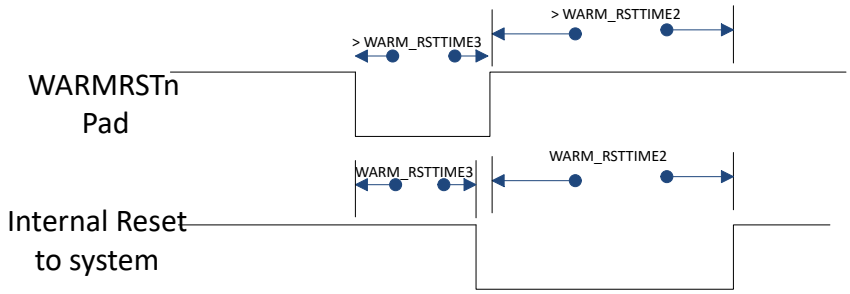


Figure 6-18. WARMRSTn Pad reset sequence

For more details on programmable values for WARM_RSTTIME1/2/3 Vs The Corresponding Delays, refer to Control Modules, MSS_RCM section.

6.3.2.2.2 Internal Warm Reset Sources

The different internal reset sources are:

- SW MMR in RCM (Reset by TOP_RCM.WARM_RESET_REQ)
- WDT Reset

- Thermal reset (Tempsensor[0:1] reset)
- VMON reset
- ESM reset
- SYS_CLK clock loss reset
- Debugger Reset

All of the warm reset sources have a corresponding enable bit in TOP_RCM.WARM_RESET_CONFIG register. Respective bits are configured for enabling the sources to trigger a warm reset.

The Internal System Reset and the External WARMRSTn pad assertion happen along with the assertion of Internal Reset Sources.

The external WARMRSTn pad deassertion is controlled by TOP_RCM.WARM_RSTTIME1 register. This is to enable sufficient reset assertion time for any external device relying on the reset signal. The internal system reset deassertion is relative to the deassertion of WARMRSTn pad and can be controlled by TOP_RCM.WARM_RSTTIME2.

The timing sequence below shows the overall sequence between assertion of Internal Reset Sources (Internal Reset Req) relative to Internal System Reset (Internal Reset to System) and WARMRSTn pad.

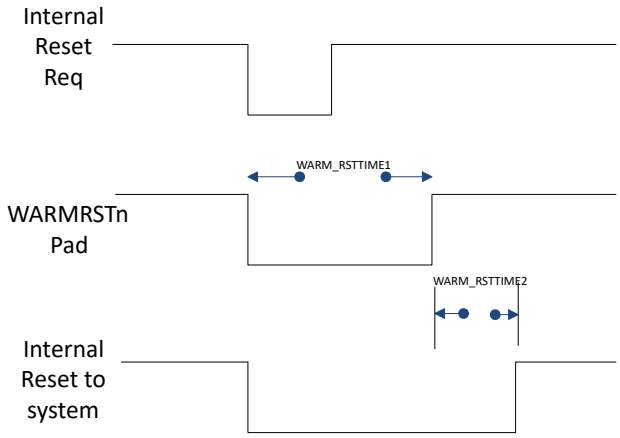


Figure 6-19. Internal Warm Reset Sequence

6.3.2.2.2.1 Thermal Alert Reset

Reset Overview

The Thermal manager module outputs an alert signal TSHUT_RST0 and TSHUT_RST1 for the two comparators in thermal manager, when the device temperature goes beyond a maximum threshold. This alert is used as a reset trigger for warm reset to the the system.

Thermal manager alert will be active as long as the error condition (Device Temperature > Maximum Temperature Threshold) is still TRUE.

When the error goes away, alert will be de-asserted and device will come out of warm reset.

For more details on Thermal manager shutdown comparator please refer to Section 6.2.2.3.5

6.3.2.2.2.2 SYS_CLK Clock Loss Reset

Reset Overview

This reset trigger for warm reset to the system is issued when the SYS_CLK clock loss event occurs and if reset is enabled using the register **WARM_RESET_CONFIG_MISC**.

6.3.2.2.2.3 Voltage Monitor Error Reset

Reset Overview

This reset trigger for warm reset to the system (if enabled using the registers **WARM_RESET_CONFIG_OV** and **WARM_RESET_CONFIG_UV**) is issued when there is a loss of power on any of the voltage rails, i.e., 3.3V, 1.8V or 1.2V, which can be monitored using the AM263P power safety monitors. These monitors can be enabled by writing into the corresponding registers as explained in the section **Power OK (POK) Modules**.

6.3.2.2.2.4 ESM Errors Reset

Reset Overview

ESM module monitors the SoC errors and can cause a system warm reset if enabled using the **WARM_RESET_CONFIG_MISC** register.

6.3.2.2.3 SW Warm Reset

This reset is triggered by a software controlled warm reset register TOP_RCM.WARM_RESET_REQ. The reset timing is the same as internal warm reset sources.

Any processor which needs to issue a warm reset to the system, should write 3'b000 into the TOP_RCM.WARM_RESET_REQ register.

6.3.2.3 Local Module Resets

The MSS_RCM.<IP>_RST_CTRL MMR's in the MSS_RCM module can be used by S/w to affect reset of individual modules. This feature is for debug purpose only. Software needs to ensure the state of the Device/IP before configuring.

6.3.2.4 R5FSS Reset

Transitions from Lockstep to Dual Core or vice-versa (on supported parts) requires a POR reset of the R5FSS. Moreover, POR reset of R5FSS is necessary for enforcing ROM eclipse.

The R5FSS POR reset can be triggered by writing to MSS_CTRL. R5SSx_CONTROL_RESET_FSM_TRIGGER (Note that this resets the full cluster). Triggering STC (Self Test Controller) also results in a POR reset of the R5FSS.

By default, R5FWFI (Wait for Interrupt) check is enabled by MSS_RCM.R5SSx_RST_WFICHECK register. The FSM checks if the CPU is in WFI state before propagating the reset.

Note

Disabling R5FWFI check is not recommended.

Delays for asserting the reset and holding the reset can be programmed in the MSS_RCM.R5SSx_RST_ASSERDLYand MSS_RCM.R5SSx_RST2ASSERTDLY registers.

Individual R5SS have their own status register MSS_RCM.R5SSx_RST_STATUS to capture the source of R5SS internal resets. Reset status bits are read active HIGH (1) when a particular reset is triggered. After reading this reset source register, software must clear the register. MSS_RCM.R5SSx_RST_CAUSE_CLR needs to be written 3'b111 to clear the status bits.

The following are the R5SS Reset sources:

- POR Reset
- Warm Reset (Also asserted during POR Reset)
- R5SSx STC Reset
- Reset for CORE0 and CORE0_VIM using MSS_RCM.R5SSx_CORE0_GRST_CTRL
- Reset for CORE1 and CORE1_VIM using MSS_RCM.R5SSx_CORE1_GRST_CTRL
- Reset for CORE0 only using MSS_RCM.R5SSx_CORE0_LRST_CTRL
- Reset for CORE1 only using MSS_RCM.R5SSx_CORE1_LRST_CTRL
- Reset for CORE0 and CORE0_VIM caused because of reset request by debugger in CORE0

- Reset for CORE1 and CORE1_VIM caused because of reset request by debugger in CORE1
- Reset for R5SSx by the RESET FSM using MSS_CTRL.R5SSx_CONTROL_RESET_FSM_TRIGGER
- Reset for R5SSx using MSS_RCM.R5SSx_POR_RST_CTRL0

Note

R5SSx refers to R5SS0 and R5SS1.

For additional details on R5SS Resets, refer to R5SS Chapter.

6.3.2.5 Reset - High Heating Value (HHV)

IOs support HHV mode during power up. HHV is defined as a state when PORz signal is driven LOW.

HHV is an IO Voltage Buffer feature that allows the IO cells to be tri-stated. All IO cells have HHV which is asserted (driven) by HHV generated during PORz assertion. There default pull values during this HHV/PORz assertion for each pin are specified in the device-specific datasheet. All HHV logic controlling the buffer high-impedance control and the associated default pull value will be asynchronous.

For more details on HHV signals, refer to *Device Configuration - Power Chapter* Device Configuration - Power Chapter

6.3.3 Core and Cluster Reset logic

Table 6-23. Reset effect on different R5FSS modules for different reset types

Modules	Both Core	Both Core	Both Core	Single Core	Single Core
	Device POR	**Device WARM RSTn**	**Cluster RSTn^**	**GRSTn^**	**LRSTn^**
R5F CPU	Yes	Yes	Yes	Yes	Yes
VIM	Yes	Yes	Yes	Yes	No
TCM Logic	Yes	Yes	Yes	Yes	No
VIM RAM	No	No	No	No	No
TCM RAM	No	No	No	No	No

^Cluster RSTn is the reset for R5SSx by the RESET FSM using MSS_CTRL.R5SSx_CONTROL_RESET_FSM_TRIGGER

^GRSTn is the reset for CORE0/1 and CORE0/1_VIM using MSS_RCM::R5SSx_CORE0/1_GRST_CTRL

^LRSTn is the reset for CORE0/1 only using MSS_RCM::R5SSx_CORE0/1_LRST_CTRL

6.3.4 Reset Status

This section summarizes the Reset Status functionality. The status of a specific individual reset is represented by an Output Pin or Software Bit.

Table 6-24. Reset Status Table

Reset Status Name	Reset Status Source	Reset Status Info	Signal Active Level	Reset Signal Details
TOP_RCM.WARM_RST_CAUSE Status Register	Register	Status register capturing which event caused the warm reset	Status bits read active HIGH (1) when a particular reset is asserted.	WARM_RST_CAUSE
MSS_RCM.R5SSx_RST_STATUS	Register	Status register capturing which event caused the corresponding R5SS reset	Status bits read active HIGH (1) when a particular reset is asserted.	R5SSx_RST_STATUS
WARMRSTn	Output Pin	On/Off pin status of warm reset	Active LOW (0)	WARMRSTn

6.3.5 Reset Registers

The reset control registers enable, disable, and adjust specific reset operations.

For additional details related to Reset Control registers, please refer to the Control Modules - MSS_RCM section.

6.3.6 Reset Power up Sequence

For additional details related to reset power up sequence, please refer to *Device Configuration - Power Chapter* Device Configuration - Power Chapter and the Power On and Reset Sequencing section of the device datasheet.

6.4 Clocking

This chapter describes the clock architecture of the device.

6.4.1 Overview

To satisfy the various subsystems requirements, the device features multiple clock sources and clock generators:

- External Crystal Driver
- Internal Oscillator
- Phase-Locked Loop circuits (PLLs)
- Dividers

Figure 6-20 shows a high-level overview of the device clock architecture. The figure captures the key clock sources and the configuration options available to select the appropriate clock source. The detailed structure is captured under each PLL clocking section.

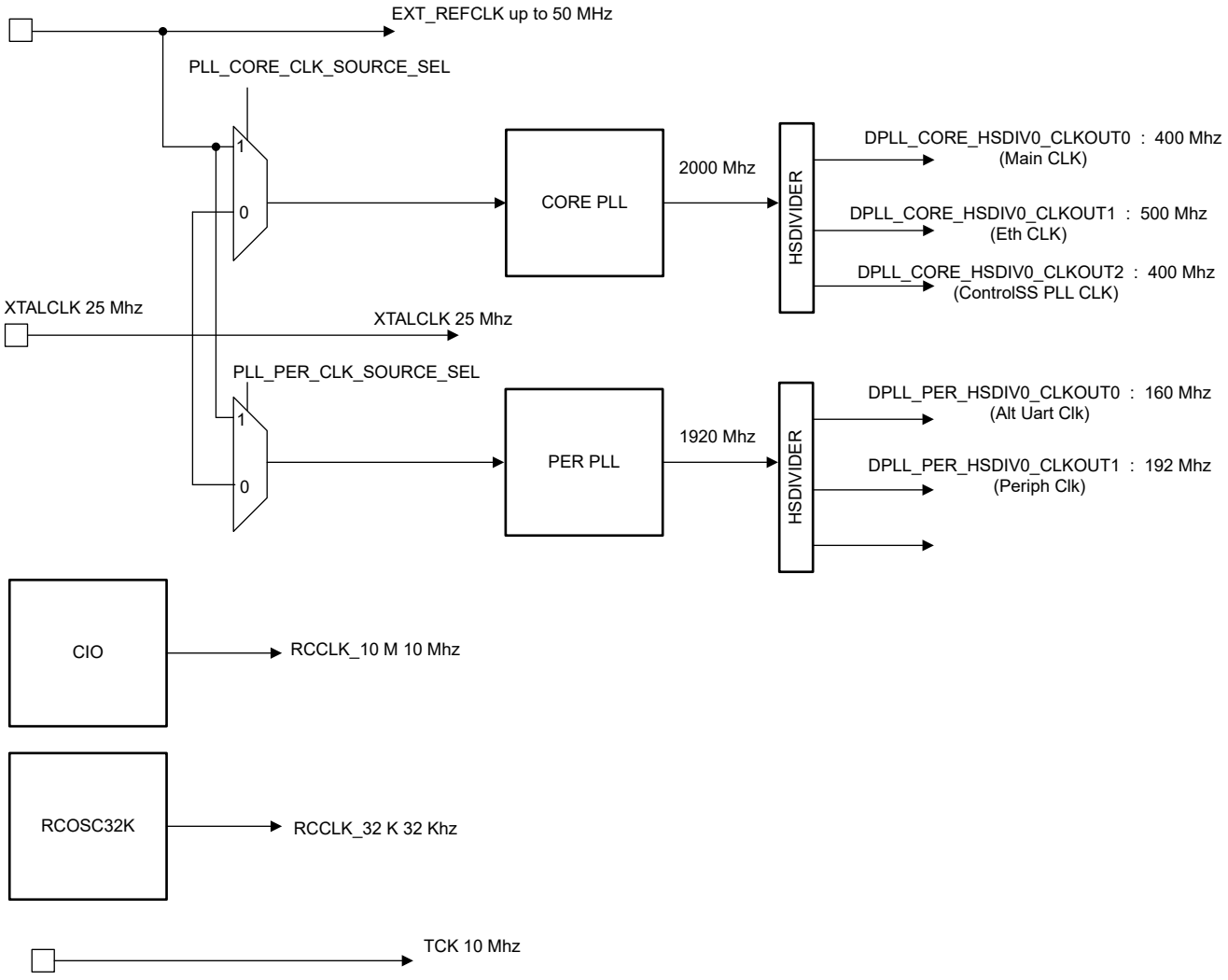


Figure 6-20. Root Clocks

The device's root clocks are depicted in the Table 6-25

Table 6-25. Root Clocks Table

Root Clocks	Frequency (MHz)
DPLL_CORE_HSDIV0_CLKOUT0	400
DPLL_CORE_HSDIV0_CLKOUT1	500
DPLL_CORE_HSDIV0_CLKOUT2	400
DPLL_CORE_HSDIV0_CLKOUT3	Unused

Table 6-25. Root Clocks Table (continued)

Root Clocks	Frequency (MHz)
DPLL_PER_HSDIV0_CLKOUT0	160
DPLL_PER_HSDIV0_CLKOUT1	192
DPLL_PER_HSDIV0_CLKOUT2	Unused
DPLL_PER_HSDIV0_CLKOUT3	Unused
RCCLK32K	0.032
RCCLK10M	10
XTALCLK	25

6.4.1.1 PLL Overview

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).

6.4.1.1.1 PLL Hookup

Bypass of HSDIVIDER will be by XTALCLK

PLL input pins are driven by TOPRCM:<Clock Instance>_SRC_SEL MMRs and the outputs are mapped as status on the TOPRCM:<Clock Instance>_STATUS MMRs.

The PHASELOCK output indicates phase tracking between output clocks (CLKOUT, CLKOUTLDO and CLKDCOLDO) and input clock (CLKINP). PHASELOCK is asserted when internally the phase difference between *FBCLK* and *REFCLK* is less than 6-12% of the *REFCLK* period for 96 continuous *REFCLKs.*

The PHASELOCK signal of CORE and PER PLL are inverted and connected as corresponding lock loss signal in ESM as shown in the following table:

Table 6-26. Lock Loss Event Mapping

Source	Event Mapping	Type	Polarity
PLL_CORE_LOCKLOSS	ESM_LVL_EVENT_25	Level	High
PLL_PER_LOCKLOSS	ESM_LVL_EVENT_26	Level	High

6.4.1.1.2 CORE PLL Overview

CORE_PLL is primarily responsible for the following IPs:

Description	Key Frequencies (MHz)
R5 Clock	400
Interconnect	200
Ethernet	250/50/5
QSPI, CANFD	80
HSM Clock	200
SPI Clock	50
FSI/SDFM PLL Clock	400

6.4.1.1.3 PER PLL Overview

PER_PLL is primarily responsible for the following IPs:

Description	Key Frequencies (MHz)
UART Clock	192

Description	Key Frequencies (MHz)
MMC Clock	50
SPI Clocks	48
I2C Clocks	48

6.4.1.2 Analog Modules

6.4.1.2.1 PLL Module

Clock Generator PLL (Phase-Locked Loop) circuits are used in the device to multiply a lower-frequency reference clock to the required operating frequency of the respective subsystem(s). The reference clock can either be external crystal driver provided through 'XTAL_XI' pad or external reference clock provided through 'EXT_REFCLK0' pad. This selection can be provided using the TOP_RCM.PLL_REF_CLK_SRC_SEL register

The low-jitter ADPLLLJ module is use as the Device CORE and PER PLL's. A high level block diagram of the ADPLLLJ is shown in Figure 6-21.

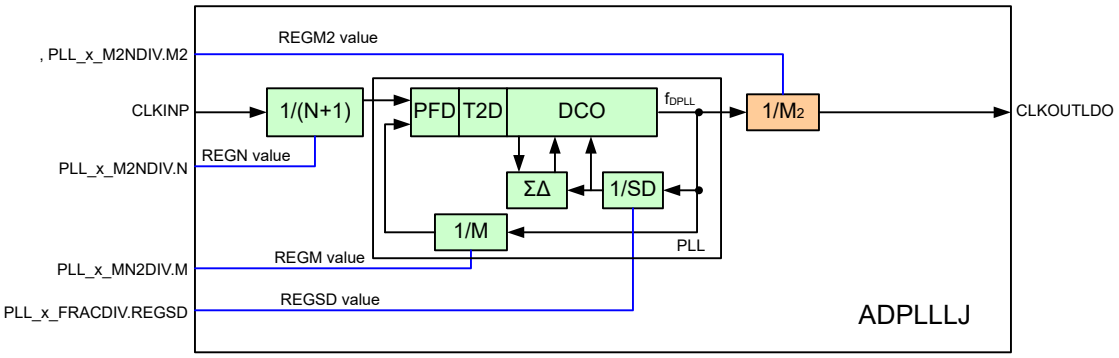


Figure 6-21. ADPLLLJ Architecture

The ADPLLLJ has the following input/output clocks

- CLKINP is the mandatory reference clock used to generate the synthesized clock. It can also be used to generate the bypass clock whenever the ADPLLLJ enters a bypass mode.
- CLKOUTLDO is the secondary output clock generated from the lock frequency of the PLL and post dividers. It does not have a bypass mode

The ADPLLLJ can be programmed to be locked at any frequency given by the following equation:

$$f_{DPLL} = \frac{M * CLKINP}{(N+1)M2}$$

Where:

- f_{DPLL} is the lock frequency.
- CLKINP is the reference system clock frequency.
- M is the 12-bit "multiplication ratio" binary value (2 – 4095). In Device it is S/W programmable via a dedicated PRCM register.
- N is the 8-bit "division ratio" binary value (0 – 255). In Device it is S/W programmable via a dedicated PRCM register.
- M2 is the 7-bit post divider binary value (1 – 127). In Device is S/W programmable via a dedicated PRCM register.

PLL input values and status ouptuts are routed to TOP_RCM MMRs

6.4.1.2.2 HSDIVIDER Module

The PLL can be coupled with an HSDIVIDER module to generate additional clocks which are divided down from the PLL lock frequency.

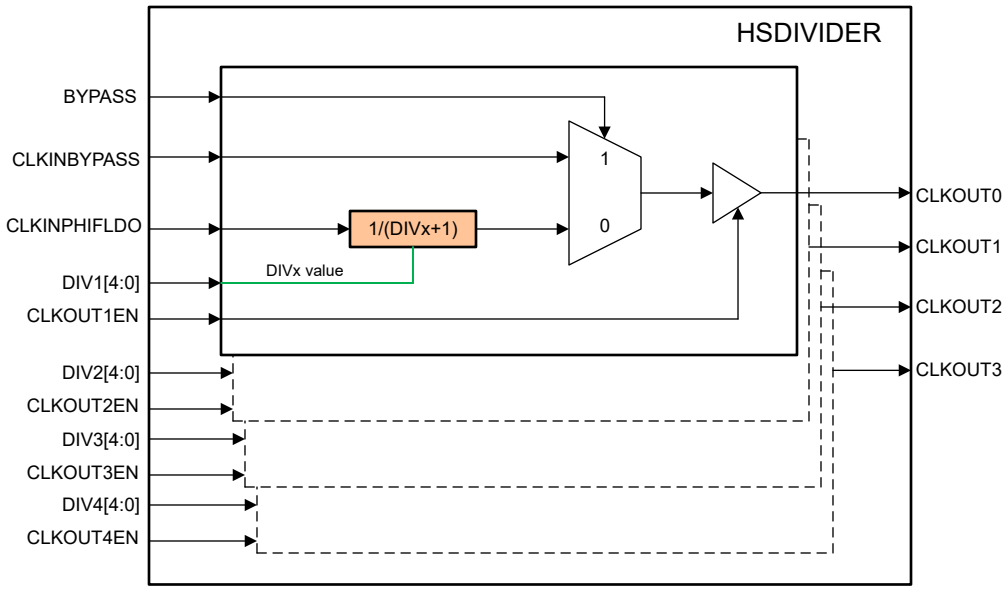


Figure 6-22. HSDIVIDER Architecture

The HSDIVIDER has two input clocks:

- CLKINPHIFLDO is the mandatory reference clock used to generate the divided clock outputs.
- CLKINBYPASSis an optional clock input and is used as the bypass clock.

The HSDIVIDER provides 4 post divider clocks whose frequency is given by:

$$CLKOUTx = \frac{CLKINPHIFL\ DO}{DIVx + 1}$$

Where:

- CLKINPHIFLDO is the input clock frequency.
- DIVx is the 5-bit divisor binary value (0-31) on the device, DIVx values are software programmable via dedicated TOP_RCM.PLL_CORE_HSDIVIDER_CLKOUTx.DIV and TOP_RCM.PLL_PER_HSDIVIDER_CLKOUTx.DIV registers

Note

The clocking subsytem provides registers to directly configure the final divide value of "DIVx+1". When specifying the desired HSDIV value to use, it should be specified as "DIVx-1".

Note

The "DIVx+1" reset value is 4.

6.4.1.2.3 PLL and Root Clocks Programming Guide

6.4.1.2.3.1 PLL Configurations

Note

This section describes the sequence in order to configure both the CORE_PLL and PER_PLL. For information on PLL configuration during boot, please refer to PLL Configuration.

6.4.1.2.3.1.1 Kick Protection Mechanism

The registers corresponding to the PLLs are present in TOP_RCM. Before accessing any register in MSS_RCM and TOP_RCM memory map, unlock the corresponding LOCK_KICK config registers with the following values:

1. LOCK0_KICK0.LOCK0_KICK0 = 0x01234567
2. LOCK0_KICK1.LOCK0_KICK1 = 0x0FEDCBA8

The above unlock procedure should be repeated for CONTROLSS_CTRL before configuring any MMRs in that region.

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.

Refer to the Control MMR chapter for more details on locking.

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.

6.4.1.2.3.1.2 Sequence to Configure the CORE PLL

1. Check for the CRYSTAL present status from TOP_RCM.CLK_LOSS_STATUS.CRYSTAL_CLOCK_LOSS register in TOP_RCM before proceeding further in configuring the *PLL*
2. If the CRYSTAL is not present then abort the *PLL* lock procedure and continue with RC_CLK for boot
3. Program the N divider of the *PLL* with the calculated value of 0x9 in register field in order to get *REF_CLK* suitable for *PLL* locking, TOP_RCM.PLL_CORE_M2NDIV.N = 0x09
4. Program the M2 divider with the value of 0x1 in the register field to get the desired frequency after PLL locking, TOP_RCM.PLL_CORE_M2NDIV.M2 = 0x1
5. Update the M divider setting of the *PLL* with the value which is derived from the above formula, TOP_RCM.PLL_CORE_MN2DIV.M = 0x360
6. Update the SELFREQDCO value based on the frequency of CLKDCOLDO
   a. TOP_RCM.PLL_CORE_CLKCTRL.SELFREQDCO = 010b, if DCOCLK range is from 500 MHz to 1000 MHz
   b. TOP_RCM.PLL_CORE_CLKCTRL.SELFREQDCO = 100b, if DCOCLK range is from 1000 MHz to 2000 MHz
7. Program the SD divider of the PLL with the value of 0x8 to get the optimum jitter performance,

   TOP_RCM.PLL_CORE_FRACDIV.REGSD = 0x8
8. Clear the IDLE bit from PLL_CORE_CLKCTRL register to make the *PLL* active for locking, TOP_RCM.PLL_CORE_CLKCTRL.IDLE= 0x0
9. Assert the TENABLE signal to make the M, N, SD divider and SELFREQDCO settings to get loaded into the *PLL* for locking, TOP_RCM.PLL_CORE_TENABLE.TENABLE = 0x1
10. Assert the TINTZ signal of the *PLL* to make the *PLL* out of SOFT reset, TOP_RCM.PLL_CORE_CLKCTRL.TINTZ = 0x1
11. De-assert the TENABLE signal by clearing the register with the value of 0x0, TOP_RCM.PLL_CORE_TENABLE.TENABLE = 0x0
12. Assert and de-assert the TENABLEDIV signal of the *PLL* by setting and clearing its corresponding register field,

    TOP_RCM.PLL_CORE_TENABLEDIV.TENABLEDIV = 0x1

    TOP_RCM.PLL_CORE_TENABLEDIV.TENABLEDIV = 0x0
13. Wait for the *PLL* to lock by polling the PHASELOCK bit to go high in the status register, TOP_RCM.PLL_CORE_STATUS.PHASELOCK = 0x1
14. Program the divider settings of the various PLL CORE HSDIVDER CLKOUT in their corresponding register field depending on the required output frequency,

    TOP_RCM.PLL_CORE_HSDIVIDER_CLKOUT0.DIV = 0x04 (i.e. 400 MHz)

    TOP_RCM.PLL_CORE_HSDIVIDER_CLKOUT1.DIV = 0x03 (i.e. 500 MHz)

    TOP_RCM.PLL_CORE_HSDIVIDER_CLKOUT2.DIV = 0x04 (i.e. 400 MHz)
15. Assert and de-assert the TENABLEDIV signal of the PLL CORE HSDIVER by setting and clearing the corresponding register field,

TOP_RCM.PLL_CORE_HSDIVIDER.TENABLEDIV = 0x1

TOP_RCM.PLL_CORE_HSDIVIDER.TENABLEDIV = 0x0

16. Un-gate the clocks from all CLKOUT of PLL CORE HSDIVDER with the following configuration,

    TOP_RCM.PLL_CORE_HSDIVIDER_CLKOUT0.GATE_CTRL = 0x1

    TOP_RCM.PLL_CORE_HSDIVIDER_CLKOUT1.GATE_CTRL = 0x1

    TOP_RCM.PLL_CORE_HSDIVIDER_CLKOUT2.GATE_CTRL = 0x1

Note

Note that PLL_CORE_HSDIVIDER.TENABLEDIV and PLL_CORE_TENABLE.TENABLE reference TENABLE fields in different registers. Make sure to address the correct registers when loading the M, N, SD dividers and SELFREQDCO settings and also when loading the HSDIVIDER values.

6.4.1.2.3.1.3 Sequence to Configure the PER PLL

The configuration sequence used for locking the CORE *PLL* (point 3 to 12) to be followed along with calculated values which is dependent on PER *PLL* lock frequency is programmed in the registers available for PER *PLL* inside TOP_RCM memory map for locking the PERIPHERAL PLL.

For PLL PER HSDIVDER settings follow the sequence below,

1. Program the divider settings of the various PLL PER HSDIVDER CLKOUT in their corresponding register field depending on the required output frequency,

   TOP_RCM.PLL_PER_HSDIVIDER_CLKOUT0.DIV= 0x0B (i.e. 160 MHz)

   TOP_RCM.PLL_PER_HSDIVIDER_CLKOUT1.DIV = 0x09 (i.e. 192 MHz)
2. Assert and de-assert the TENABLEDIV signal of the PLL PER HSDIVER by setting and clearing the

   TOP_RCM.PLL_PER_HSDIVIDER.TENABLEDIV register field,

   TOP_RCM.PLL_PER_HSDIVIDER.TENABLEDIV = 0x1

   TOP_RCM.PLL_PER_HSDIVIDER.TENABLEDIV = 0x0
3. Un-gate the clocks from all CLKOUT of PLL PER HSDIVDER with the following configuration,

   TOP_RCM.PLL_PER_HSDIVIDER_CLKOUT0.GATE_CTRL = 0x1

   TOP_RCM.PLL_PER_HSDIVIDER_CLKOUT1.GATE_CTRL = 0x1

Note

1. For faster PLL locking, configure the PLL settings (point 3 to 11) of both PLL's before polling for the lock of the corresponding PLL
2. For PLL lock using *EXT_REF* clock, configure the PLL_REF_CLK_SRC_SEL.PLL_CORE_REF_CLK_SRC_SEL (or)

   PLL_REF_CLK_SRC_SEL.PLL_PER_REF_CLK_SRC_SEL register fields in TOP_RCM before staring the PLL configurations
3. Configure the *DCC* with reference clock as CRYSTAL and compare clock as *PLL_CORE_CLKOUT1* to measure the frequency range before switching the *SYS_CLK* / R5 CLK to PLL clock. Refer *DCC* chapter for more information in its configuration and usage (Note - Optional configuration only used for safety purpose)

6.4.1.2.3.1.4 Sequence to Re-Configure the PLL

The following section provides details of steps involved in re-configuring the PLL with new frequency:

1. Switch all the peripheral clocks which are derived from PLL to WUCPU_CLK (XTAL_CLK) so that when PLL is unlocked other peripheral are in safe state. (Refer to the IP Clock Configurations section for programming.)

2. Change the CPU clock source to WUCPU_CLK (XTAL_CLK) by programming the R5SS GCM with the value of 0x0 and SYS_CLK GCD (optional) with the value of 0x0 so that CPU does not enter into dead lock condition. (Refer to the Root Clock Configurations for programming.)
3. Assert the TINTZ signal of the PLL to reset the internal FSM of PLL, TOP_RCM.PLL_CORE_CLKCTRL.TINTZ = 0x0.
4. Follow the steps mentioned in Sequence to Configure the CORE PLL from point 3 to 12 to re-configure the PLL CORE.

Note

Follow the above-mentioned steps except point (2) to re-configure the PLL PER.

6.4.1.2.3.2 Root Clock Configurations

6.4.1.2.3.2.1 Sequence for Programming SYS and R5 Clocks

1. Program SYS CLK *GCD* register with the value of 0x111 in-order to switch to a new desired frequency, TOP_RCM.SYS_CLK_DIV_VAL.CLKDIV = 0x111
2. Poll for the CURRDIVR field of corresponding status register to reflect its new frequency change, TOP_RCM.SYS_CLK_STATUS. CURRDIVIDER = 0x1
3. If the R5 clock frequency needs to be same as SYS clock frequency, then program the TOP_RCM.R5SS0_CLK_DIV_SEL.CLKDIVSEL = 0x7 (or / and) TOP_RCM.R5SS1_CLK_DIV_SEL.CLKDIVSEL = 0x7 register(s) as required or else leave with default value of 0x0 without any programming
4. After the divider configuration, update the R5SS *GCM* register with the value of 0x222 to select the PLL_CORE_CLOCKOUT0 as its source, TOP_RCM.R5SS_CLK_SRC_SEL.CLKSRCSEL= 0x222
5. Poll for the CLKINUSE field of corresponding status register to reflect its new frequency change, TOP_RCM.R5SS_CLK_STATUS.CLKINUSE = 0x04

6.4.1.2.3.2.2 Sequence for Programming TRACE Clock

1. Program TRCCLKOUT *GCD* register with the value of 0x111 in-order to switch to a new desired frequency, TOP_RCM.TRCCLKOUT_DIV_VAL.CLKDIV = 0x111
2. Poll for the CURRDIVR field of corresponding status register to reflect its new frequency change, TOP_RCM.TRCCLKOUT_CLK_STATUS.CURRDIVIDER = 0x01
3. Update the TRCCLKOUT *GCM* register with the value of 0x222 to select *PLL_CORE_CLKOUT1* as its source, TOP_RCM.TRCCLKOUT_CLK_SRC_SEL.CLKSRCSEL = 0x222
4. Poll for the CLKINUSE field of corresponding status register to reflect its new frequency change, TOP_RCM.TRCCLKOUT_CLK_STATUS.CLKINUSE = 0x04

6.4.1.2.3.2.3 Sequence for Programming CLKOUT Clock

1. Program CLKOUT0 *GCD* register with the value of 0x000 in-order to switch to a new desired frequency, TOP_RCM.CLKOUT0_DIV_VAL.CLKDIV = 0x111
2. Poll for the CURRDIVR field of corresponding status register to reflect its new frequency change, TOP_RCM.CLKOUT0_CLK_STATUS.CURRDIVIDER = 0x01
3. Update the CLKOUT0 *GCM* register with the value of 0x222 to select *PLL_CORE_CLKOUT1* as its source, TOP_RCM.CLKOUT0_CLK_SRC_SEL.CLKSRCSEL = 0x222
4. Poll for the CLKINUSE field of corresponding status register to reflect its new frequency change, TOP_RCM.CLKOUT0_CLK_STATUS.CLKINUSE = 0x04

6.4.2 Clock IO

6.4.2.1 Overview

Various external clock inputs are needed to drive the device, as well as there are external sources of the clock provisioned for certain peripherals. The clocks in the AM263Px device are as depicted in External Clocks

The device provides several system clock outputs. Summary of these output clock signals is as follows:

- R5FSS[1:0]_CLK
- SYS_CLK
- CLKOUT[1:0]
- IP Clocks

The IP Clocks are routed directly from subsystems to device pins, and they are described in the respective module chapter.

For more details on IP clocks generation, please refer to IP Clocking Section

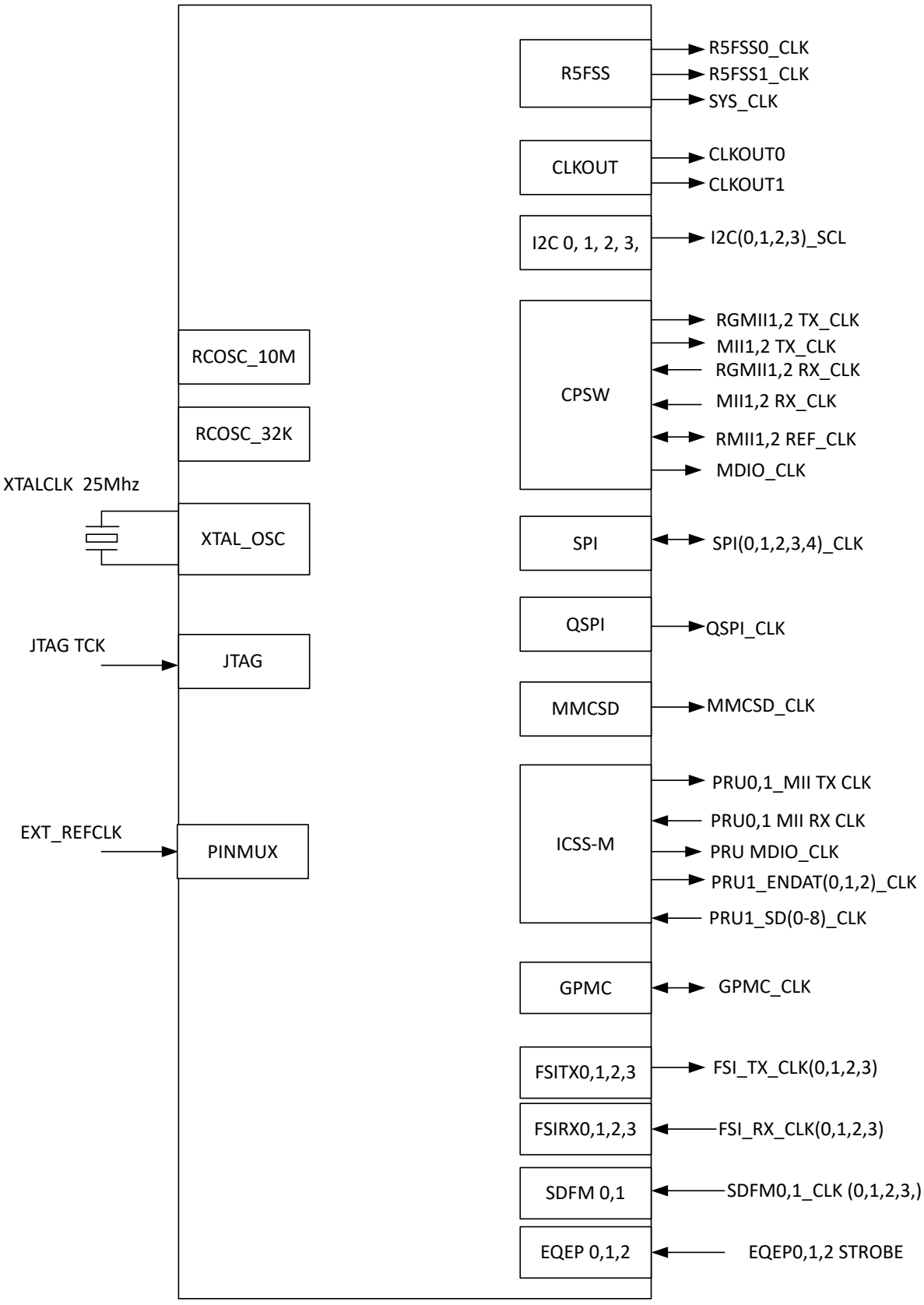


Figure 6-23. External Clocks

Note

While this device does not support a separate Observation Clock output signal, the system level functionality is recognized by utilizing the CLKOUT[1:0] signals

Note

CLKOUT0 will reflect RC clock after POK is asserted and switch to XTAL_CLK after Internal SYS_RST is released

6.4.2.2 Clock IO Mapping

Please refer to the *Terminal Configurations and Functions* section of the device-specific Datasheet.

6.4.3 IP Clocking

The required IP clocks for the device are generated using the Root clocks mentioned in Root clocks section.

To generate the IP clocks, the root clocks are muxed and divided using the GCM and GCD modules respectively.

The GCM module takes 8 clock sources as inputs and gives an output clock according to the select (MODULEx_CLK_SRC_SEL) provided. Additionally, one can gate the output clock using the clock gating input (MODULEx_CLK_GATE)

The GCM_Divider module takes in an input clock and divides it according to the divider value (MODULEx_CLK_DIV_VAL) provided. Note that to divide the input clock by 'DIV' value, the MMR value provided should be 'DIV-1'

6.4.3.1 R5SS and SYSCLK Clock Tree

The device's SYS_CLK is generated using GCM and GCM_Divider modules.

The GCM module takes 8 clock sources as inputs and gives an output clock according to the select (MODULEx_CLK_SRC_SEL) provided. Additionally, one can gate the output clock using the clock gating input (MODULEx_CLK_GATE)

The GCM_Divider module takes in an input clock and divides it according to the divider value (MODULEx_CLK_DIV_VAL)

R5SS/SYSCLK Clocking gives an overview of the R5 Subsystem and SYSCLK Clocking structure.

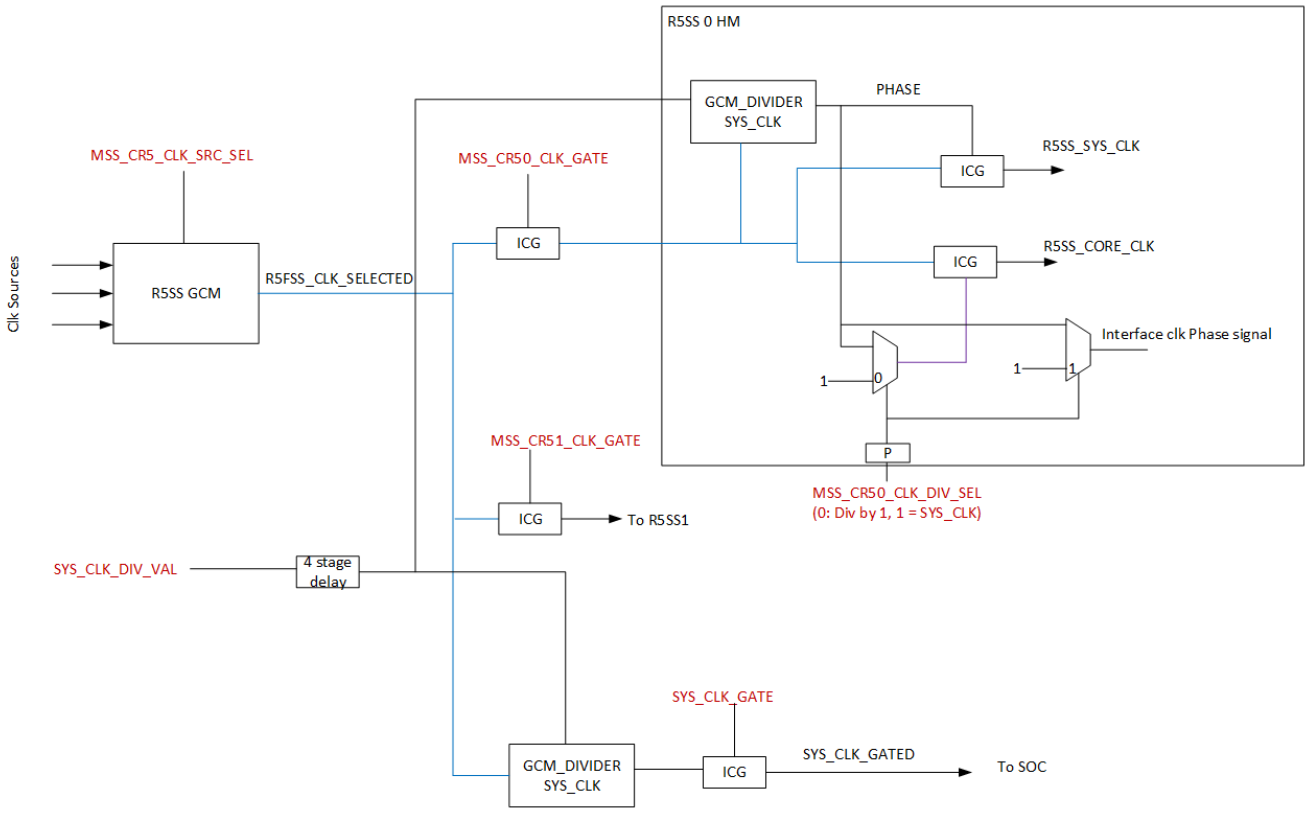


Figure 6-24. R5SS/SYSCLK Clocking

Tables *R5SS_CORE_CLK:SYSCLK* Achievable Ratio shows the different operation options concerning the ratio between R5SS_CORE_CLK and the SYSCLK.

Table 6-27. R5SS_CORE_CLK:SYSCLK Achievable Ratio

R5SS_CORE_CLK:SYS_CLK Ratio	Configuration	R5_CORE Frequency	SYS_CLK Frequency	Notes
1:1	R5FSS_CLK_SELECTED = 400MHz SYS_CLK_DIVIDER = Div by 2 MSS_CR5*_CLK_DIV_SEL = 1	200MHz	200MHz	This config is used for dynamic switching from 2:1 and 1:1. R5_CORE is 400MHz, only the DIV bit needs to be modified
2:1	R5FSS_CLK_SELECTED = 400MHz SYS_CLK_DIVIDER = Div by 2 MSS_CR5*_CLK_DIV_SEL = 0	400MHz	200MHz	

6.4.3.2 IP Clocks Having GCM

The structure is similar for all IP's having dedicated GCM's

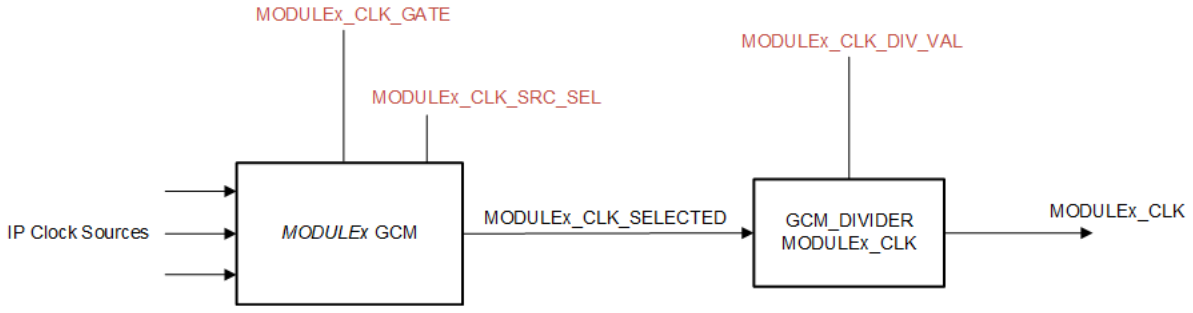


Figure 6-25. Generic IP clocking with GCM and Divider

Refer to the Clock Selection table for more details on MMRs present and clock sources for all the peripherals

6.4.3.3 IP Clocks working on SYS_CLK

Every IP working on SYS_CLK has a separate clock gate. In the case that clock gate is implemented in the IP, clock is routed directly to the IP with no clock gate inserted at the SOC-level. The diagram below shows the generic structure for all IP sourced from SYS_CLK.

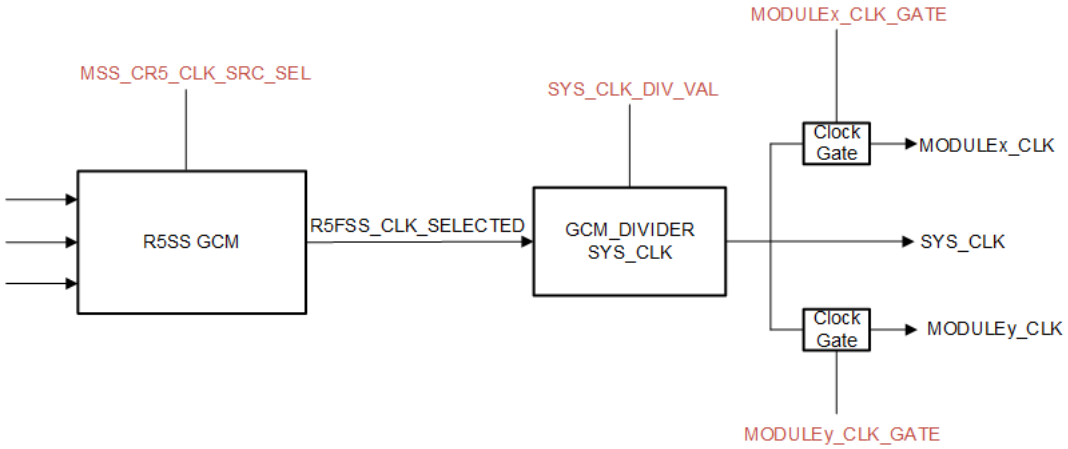


Figure 6-26. Generic IP clocking with SYS_CLK

Note

In the case that the peripherals implement clock gating internal to the IP, no additional ICG is provisioned in RCM.

6.4.3.4 Clock Selection

Table 6-28 lists the configuration options for the clock source, divider, and gating selections for different peripheral clocks.

Table 6-28. Configuration Options

Clock Muxes	Clock Sources		MMR Select	MMR Divider Select	MMR Clock Gate	IP's
R5FSS_CLK_MUX	0	XTALCLK	R5SS_CLK_SRC_SEL	R5SS0_CLK_DIV_SEL	R5SS0_CLK_GATE	R5SS0
	1	EXT_REFCLK				
	2	DPLL_CORE_HSDIV0_CLKOUT0				
	3	RCCLK10M				
	4	RCCLK10M		R5SS1_CLK_DIV_SEL	R5SS1_CLK_GATE	R5SS1
	5	RCCLK10M				
	6	XTALCLK				
	7	RCCLK10M				
TRC_CLKOUT_CLK_MUX	0	XTALCLK	TRCCLKOUT_CLK_SRC_SEL	TRCCLKOUT_DIV_VAL	TRCCLKOUT_CLK_GATE	Trace
	1	DPLL_CORE_HSDIV0_CLKOUT0				
	2	DPLL_CORE_HSDIV0_CLKOUT1				
	3	DPLL_PER_HSDIV0_CLKOUT0				
	4	DPLL_PER_HSDIV0_CLKOUT1				
	5	RCCLK10M				
	6	XTALCLK				
	7	RCCLK10M				
CLKOUT0_CLK_MUX	0	XTALCLK	CLKOUT0_CLK_SRC_SEL	CLKOUT0_DIV_VAL	CLKOUT0_CLK_GATE	CLKOUT0
	1	DPLL_CORE_HSDIV0_CLKOUT0				
	2	DPLL_CORE_HSDIV0_CLKOUT1				
	3	DPLL_PER_HSDIV0_CLKOUT0				
	4	DPLL_PER_HSDIV0_CLKOUT1				
	5	RCCLK10M				
	6	RCCLK32K				
	7	CTPS_GENF0				

Table 6-28. Configuration Options (continued)

Clock Muxes	Clock Sources		MMR Select	MMR Divider Select	MMR Clock Gate	IP's
CLKOUT1_CLK_MUX	0	XTALCLK	CLKOUT1_CLK_SRC_SEL	CLKOUT1_DIV_VAL	CLKOUT1_CLK_GATE	CLKOUT1
	1	DPLL_CORE_HSDIV0_CLKOUT0				
	2	DPLL_CORE_HSDIV0_CLKOUT1				
	3	DPLL_PER_HSDIV0_CLKOUT0				
	4	DPLL_PER_HSDIV0_CLKOUT1				
	5	RCCLK10M				
	6	RCCLK32K				
	7	CTPS_GENF0				
RTI0_CLK_MUX	0	XTALCLK	RTI0_CLK_SRC_SEL	RTI0_CLK_DIV_VAL	RTI0_CLK_GATE	RTI0
	1	EXT_REFCLK				
	2	SYS_CLK				
	3	DPLL_PER_HSDIV0_CLKOUT1				
	4	DPLL_CORE_HSDIV0_CLKOUT1				
	5	RCCLK10M				
	6	XTALCLK				
	7	CTPS_GENF0				
RTI1_CLK_MUX	0	XTALCLK	RTI1_CLK_SRC_SEL	RTI1_CLK_DIV_VAL	RTI1_CLK_GATE	RTI1
	1	EXT_REFCLK				
	2	SYS_CLK				
	3	DPLL_PER_HSDIV0_CLKOUT1				
	4	DPLL_CORE_HSDIV0_CLKOUT1				
	5	RCCLK10M				
	6	XTALCLK				
	7	CTPS_GENF0				

Table 6-28. Configuration Options (continued)

Clock Muxes	Clock Sources		MMR Select	MMR Divider Select	MMR Clock Gate	IP's
RTI2_CLK_MUX	0	XTALCLK	RTI2_CLK_SRC_SEL	RTI2_CLK_DIV_VAL	RTI2_CLK_GATE	RTI2
	1	EXT_REFCLK				
	2	SYS_CLK				
	3	DPLL_PER_HSDIV0_CLKOUT1				
	4	DPLL_CORE_HSDIV0_CLKOUT1				
	5	RCCLK10M				
	6	XTALCLK				
	7	CTPS_GENF0				
RTI3_CLK_MUX	0	XTALCLK	RTI3_CLK_SRC_SEL	RTI3_CLK_DIV_VAL	RTI3_CLK_GATE	RTI3
	1	EXT_REFCLK				
	2	SYS_CLK				
	3	DPLL_PER_HSDIV0_CLKOUT1				
	4	DPLL_CORE_HSDIV0_CLKOUT1				
	5	RCCLK10M				
	6	XTALCLK				
	7	CTPS_GENF0				
WDT0_CLK_MUX	0	XTALCLK	WDT0_CLK_SRC_SEL	WDT0_CLK_DIV_VAL	WDT0_CLK_GATE	WDT0
	1	RCCLK10M				
	2	SYS_CLK				
	3	DPLL_PER_HSDIV0_CLKOUT1				
	4	DPLL_CORE_HSDIV0_CLKOUT1				
	5	RCCLK10M				
	6	XTALCLK				
	7	RCCLK32K				

Table 6-28. Configuration Options (continued)

Clock Muxes	Clock Sources		MMR Select	MMR Divider Select	MMR Clock Gate	IP's
WDT1_CLK_MUX	0	XTALCLK	WDT1_CLK_SRC_SEL	WDT1_CLK_DIV_VAL	WDT1_CLK_GATE	WDT1
	1	RCCLK10M				
	2	SYS_CLK				
	3	DPLL_PER_HSDIV0_CLKOUT1				
	4	DPLL_CORE_HSDIV0_CLKOUT1				
	5	RCCLK10M				
	6	XTALCLK				
	7	RCCLK32K				
WDT2_CLK_MUX	0	XTALCLK	WDT2_CLK_SRC_SEL	WDT2_CLK_DIV_VAL	WDT2_CLK_GATE	WDT2
	1	RCCLK10M				
	2	SYS_CLK				
	3	DPLL_PER_HSDIV0_CLKOUT1				
	4	DPLL_CORE_HSDIV0_CLKOUT1				
	5	RCCLK10M				
	6	XTALCLK				
	7	RCCLK32K				
WDT3_CLK_MUX	0	XTALCLK	WDT3_CLK_SRC_SEL	WDT3_CLK_DIV_VAL	WDT3_CLK_GATE	WDT3
	1	RCCLK10M				
	2	SYS_CLK				
	3	DPLL_PER_HSDIV0_CLKOUT1				
	4	DPLL_CORE_HSDIV0_CLKOUT1				
	5	RCCLK10M				
	6	XTALCLK				
	7	RCCLK32K				

Table 6-28. Configuration Options (continued)

Clock Muxes	Clock Sources		MMR Select	MMR Divider Select	MMR Clock Gate	IP's
QSPI_CLK_MUX	0	XTALCLK	QSPI0_CLK_SRC_SEL	QSPI0_CLK_DIV_VAL	QSPI0_CLK_GATE	QSPI
	1	EXT_REFCLK				
	2	SYS_CLK				
	3	DPLL_PER_HSDIV0_CLKOUT1				
	4	DPLL_CORE_HSDIV0_CLKOUT0				
	5	RCCLK10M				
	6	XTALCLK				
	7	RCCLK10M				
SPI0_CLK_MUX	0	XTALCLK	MCSPI0_CLK_SRC_SEL	MCSPI0_CLK_DIV_VAL	MCSPI0_CLK_GATE	SPI0
	1	EXT_REFCLK				
	2	SYS_CLK				
	3	DPLL_PER_HSDIV0_CLKOUT1				
	4	DPLL_CORE_HSDIV0_CLKOUT0				
	5	RCCLK10M				
	6	XTALCLK				
	7	RCCLK10M				
SPI1_CLK_MUX	0	XTALCLK	MCSPI1_CLK_SRC_SEL	MCSPI1_CLK_DIV_VAL	MCSPI1_CLK_GATE	SPI1
	1	EXT_REFCLK				
	2	SYS_CLK				
	3	DPLL_PER_HSDIV0_CLKOUT1				
	4	DPLL_CORE_HSDIV0_CLKOUT0				
	5	RCCLK10M				
	6	XTALCLK				
	7	RCCLK10M				

Table 6-28. Configuration Options (continued)

Clock Muxes	Clock Sources		MMR Select	MMR Divider Select	MMR Clock Gate	IP's
SPI2_CLK_MUX	0	XTALCLK	MCSPI2_CLK_SRC_SEL	MCSPI2_CLK_DIV_VAL	MCSPI2_CLK_GATE	SPI2
	1	EXT_REFCLK				
	2	SYS_CLK				
	3	DPLL_PER_HSDIV0_CLKOUT1				
	4	DPLL_CORE_HSDIV0_CLKOUT0				
	5	RCCLK10M				
	6	XTALCLK				
	7	RCCLK10M				
SPI3_CLK_MUX	0	XTALCLK	MCSPI3_CLK_SRC_SEL	MCSPI3_CLK_DIV_VAL	MCSPI3_CLK_GATE	SPI3
	1	EXT_REFCLK				
	2	SYS_CLK				
	3	DPLL_PER_HSDIV0_CLKOUT1				
	4	DPLL_CORE_HSDIV0_CLKOUT0				
	5	RCCLK10M				
	6	XTALCLK				
	7	RCCLK10M				
SPI4_CLK_MUX	0	XTALCLK	MCSPI4_CLK_SRC_SEL	MCSPI4_CLK_DIV_VAL	MCSPI4_CLK_GATE	SPI4
	1	EXT_REFCLK				
	2	SYS_CLK				
	3	DPLL_PER_HSDIV0_CLKOUT1				
	4	DPLL_CORE_HSDIV0_CLKOUT0				
	5	RCCLK10M				
	6	XTALCLK				
	7	RCCLK10M				

Table 6-28. Configuration Options (continued)

Clock Muxes	Clock Sources		MMR Select	MMR Divider Select	MMR Clock Gate	IP's
I2C_CLK_MUX	0	XTALCLK	I2C_CLK_SRC_SEL	I2C_CLK_DIV_VAL	I2C0_CLK_GATE	I2C0
	1	EXT_REFCLK				
	2	SYS_CLK			I2C1_CLK_GATE	I2C1
	3	DPLL_PER_HSDIV0_CLKOUT1				
	4	DPLL_CORE_HSDIV0_CLKOUT0			I2C2_CLK_GATE	I2C2
	5	RCCLK10M				
	6	XTALCLK			I2C3_CLK_GATE	I2C3
	7	RCCLK10M				
UART0_CLK_MUX	0	XTALCLK	LIN0_UART0_CLK_SRC_SEL	LIN0_UART0_CLK_DIV_VAL	UART0_CLKGATE	UART0
	1	EXT_REFCLK				
	2	SYS_CLK				
	3	DPLL_PER_HSDIV0_CLKOUT1				
	4	DPLL_CORE_HSDIV0_CLKOUT0			LIN0_CLKGATE	LIN0
	5	RCCLK10M				
	6	XTALCLK				
	7	DPLL_PER_HSDIV0_CLKOUT0				
UART1_CLK_MUX	0	XTALCLK	LIN1_UART1_CLK_SRC_SEL	LIN1_UART1_CLK_DIV_VAL	UART1_CLKGATE	UART1
	1	EXT_REFCLK				
	2	SYS_CLK				
	3	DPLL_PER_HSDIV0_CLKOUT1				
	4	DPLL_CORE_HSDIV0_CLKOUT0			LIN1_CLKGATE	LIN1
	5	RCCLK10M				
	6	XTALCLK				
	7	DPLL_PER_HSDIV0_CLKOUT0				

Table 6-28. Configuration Options (continued)

Clock Muxes	Clock Sources		MMR Select	MMR Divider Select	MMR Clock Gate	IP's
UART2_CLK_MUX	0	XTALCLK	LIN2_UART2_CLK_SRC_SEL	LIN2_UART2_CLK_DIV_VAL	UART2_CLKGATE	UART2
	1	EXT_REFCLK				
	2	SYS_CLK				
	3	DPLL_PER_HSDIV0_CLKOUT1				
	4	DPLL_CORE_HSDIV0_CLKOUT0			LIN2_CLKGATE	LIN2
	5	RCCLK10M				
	6	XTALCLK				
	7	DPLL_PER_HSDIV0_CLKOUT0				
UART3_CLK_MUX	0	XTALCLK	LIN3_UART3_CLK_SRC_SEL	LIN3_UART3_CLK_DIV_VAL	UART3_CLKGATE	UART3
	1	EXT_REFCLK				
	2	SYS_CLK				
	3	DPLL_PER_HSDIV0_CLKOUT1				
	4	DPLL_CORE_HSDIV0_CLKOUT0			LIN3_CLKGATE	LIN3
	5	RCCLK10M				
	6	XTALCLK				
	7	DPLL_PER_HSDIV0_CLKOUT0				
UART4_CLK_MUX	0	XTALCLK	LIN4_UART4_CLK_SRC_SEL	LIN4_UART4_CLK_DIV_VAL	UART4_CLKGATE	UART4
	1	EXT_REFCLK				
	2	SYS_CLK				
	3	DPLL_PER_HSDIV0_CLKOUT1				
	4	DPLL_CORE_HSDIV0_CLKOUT0			LIN4_CLKGATE	LIN4
	5	RCCLK10M				
	6	XTALCLK				
	7	DPLL_PER_HSDIV0_CLKOUT0				

Table 6-28. Configuration Options (continued)

Clock Muxes	Clock Sources		MMR Select	MMR Divider Select	MMR Clock Gate	IP's
UART5_CLK_MUX	0	XTALCLK	LIN5_UART5_CLK_SRC_SEL	LIN5_UART5_CLK_DIV_VAL	UART5_CLKGATE	UART5
	1	EXT_REFCLK				
	2	SYS_CLK				
	3	DPLL_PER_HSDIV0_CLKOUT1				
	4	DPLL_CORE_HSDIV0_CLKOUT0				
	5	RCCLK10M				
	6	XTALCLK				
	7	DPLL_PER_HSDIV0_CLKOUT0				
ICSS_UART_CLK_MUX	0	XTALCLK	ICSSM0_UART0_CLK_SRC_SEL	ICSSM0_UART_CLK_DIV_VAL	ICSSM0_UART_CLK_GATE	ICSSM
	1	EXT_REFCLK				
	2	SYS_CLK				
	3	DPLL_PER_HSDIV0_CLKOUT1				
	4	DPLL_CORE_HSDIV0_CLKOUT0				
	5	RCCLK10M				
	6	XTALCLK				
	7	DPLL_PER_HSDIV0_CLKOUT0				
MCAN0_CLK_MUX	0	XTALCLK	MCAN0_CLK_SRC_SEL	MCAN0_CLK_DIV_VAL	MCAN0_CLK_GATE	MCAN0
	1	EXT_REFCLK				
	2	SYS_CLK				
	3	DPLL_PER_HSDIV0_CLKOUT1				
	4	DPLL_CORE_HSDIV0_CLKOUT0				
	5	RCCLK10M				
	6	XTALCLK				
	7	RCCLK10M				

Table 6-28. Configuration Options (continued)

Clock Muxes	Clock Sources		MMR Select	MMR Divider Select	MMR Clock Gate	IP's
MCAN1_CLK_MUX	0	XTALCLK	MCAN1_CLK_SRC_SEL	MCAN1_CLK_DIV_VAL	MCAN1_CLK_GATE	MCAN1
	1	EXT_REFCLK				
	2	SYS_CLK				
	3	DPLL_PER_HSDIV0_CLKOUT1				
	4	DPLL_CORE_HSDIV0_CLKOUT0				
	5	RCCLK10M				
	6	XTALCLK				
	7	RCCLK10M				
MCAN2_CLK_MUX	0	XTALCLK	MCAN2_CLK_SRC_SEL	MCAN2_CLK_DIV_VAL	MCAN2_CLK_GATE	MCAN2
	1	EXT_REFCLK				
	2	SYS_CLK				
	3	DPLL_PER_HSDIV0_CLKOUT1				
	4	DPLL_CORE_HSDIV0_CLKOUT0				
	5	RCCLK10M				
	6	XTALCLK				
	7	RCCLK10M				
MCAN3_CLK_MUX	0	XTALCLK	MCAN3_CLK_SRC_SEL	MCAN3_CLK_DIV_VAL	MCAN3_CLK_GATE	MCAN3
	1	EXT_REFCLK				
	2	SYS_CLK				
	3	DPLL_PER_HSDIV0_CLKOUT1				
	4	DPLL_CORE_HSDIV0_CLKOUT0				
	5	RCCLK10M				
	6	XTALCLK				
	7	RCCLK10M				

Table 6-28. Configuration Options (continued)

Clock Muxes	Clock Sources		MMR Select	MMR Divider Select	MMR Clock Gate	IP's
MMCSD_CLK_MUX	0	XTALCLK	MMC0_CLK_SRC_SEL	MMC0_CLK_DIV_VAL	MMC0_CLK_GATE	MMC
	1	EXT_REFCLK				
	2	SYS_CLK				
	3	DPLL_PER_HSDIV0_CLKOUT1				
	4	DPLL_CORE_HSDIV0_CLKOUT0				
	5	RCCLK10M				
	6	XTALCLK				
	7	RCCLK10M				
CPTS_CLK_MUX	0	XTALCLK	CPTS_CLK_SRC_SEL	CPTS_CLK_DIV_VAL	CPTS_CLK_GATE	CPSW
	1	EXT_REFCLK				
	2	SYS_CLK				
	3	DPLL_CORE_HSDIV0_CLKOUT1				
	4	DPLL_CORE_HSDIV0_CLKOUT0				
	5	RCCLK10M				
	6	XTALCLK				
	7	RCCLK10M				
HSM_RTI_CLK_MUX	0	XTALCLK	HSM_RTIA_CLK_SRC_SEL	HSM_RTI_CLK_DIV_VAL	HSM_RTI_CLK_GATE	RTI
	1	XTALCLK				
	2	SYS_CLK				
	3	DPLL_PER_HSDIV0_CLKOUT1				
	4	RCCLK10M				
	5	RCCLK10M				
	6	EXT_REFCLK				
	7	RCCLK32K				

Table 6-28. Configuration Options (continued)

Clock Muxes	Clock Sources		MMR Select	MMR Divider Select	MMR Clock Gate	IP's
HSM_WDT_CLK_MUX	0	XTALCLK	HSM_WDT_CLK_SRC_SEL	HSM_WDT_CLK_DIV_VAL	HSM_WDT_CLK_GATE	WDT
	1	XTALCLK				
	2	SYS_CLK				
	3	DPLL_PER_HSDIV0_CLKOUT1				
	4	RCCLK10M				
	5	RCCLK10M				
	6	EXT_REFCLK				
	7	RCCLK32K				
HSM_RTC_CLK_MUX	0	XTALCLK	HSM_RTC_CLK_SRC_SEL	HSM_RTC_CLK_DIV_VAL	HSM_RTC_CLK_GATE	RTC
	1	XTALCLK				
	2	SYS_CLK				
	3	DPLL_PER_HSDIV0_CLKOUT1				
	4	RCCLK10M				
	5	RCCLK10M				
	6	EXT_REFCLK				
	7	RCCLK32K				
HSM_DMTA_CLK_MUX	0	XTALCLK	HSM_DMTA_CLK_SRC_SEL	HSM_DMTA_CLK_DIV_VAL	HSM_DMTA_CLK_GATE	DMTA
	1	XTALCLK				
	2	SYS_CLK				
	3	DPLL_PER_HSDIV0_CLKOUT1				
	4	RCCLK10M				
	5	RCCLK10M				
	6	EXT_REFCLK				
	7	RCCLK32K				

Table 6-28. Configuration Options (continued)

Clock Muxes	Clock Sources		MMR Select	MMR Divider Select	MMR Clock Gate	IP's
HSM_DMTB_CLK_MUX	0	XTALCLK	HSM_DMTB_CLK_SRC_SEL	HSM_DMTB_CLK_DIV_VAL	HSM_DMTB_CLK_GATE	DMTB
	1	XTALCLK				
	2	SYS_CLK				
	3	DPLL_PER_HSDIV0_CLKOUT1				
	4	RCCLK10M				
	5	RCCLK10M				
	6	EXT_REFCLK				
	7	RCCLK32K				
CONTROLSS_PLL_CLK_MUX	0	XTALCLK	CONTROLSS_PLL_CLK_SRC_SEL	CONTROLSS_PLL_CLK_DIV_VAL	CONTROLSS_PLL_CLK_GATE	ControlSS
	1	EXT_REFCLK				
	2	DPLL_CORE_HSDIV0_CLKOUT2				
	3	DPLL_PER_HSDIV0_CLKOUT1				
	4	DPLL_CORE_HSDIV0_CLKOUT0				
	5	RCCLK10M				
	6	XTALCLK				
	7	RCCLK10M				
NA	DPLL_CORE_HSDIV0_CLKOUT1		NA	RGMII_250_CLK_DIV_VAL	RGMII_250_CLK_GATE	CPSW
NA	DPLL_CORE_HSDIV0_CLKOUT1		NA	RGMII_50_CLK_DIV_VAL	RGMII_50_CLK_GATE	CPSW
NA	DPLL_CORE_HSDIV0_CLKOUT1		NA	RGMII_5_CLK_DIV_VAL	RGMII_5_CLK_GATE	CPSW
NA	XTALCLK		NA	XTAL_MMC_32K_CLK_DIV_VAL	MMC0_32K_CLK_GATE	MMC 32K
NA	XTALCLK		NA	XTAL_TEMPSENSE_32K_CLK_DIV_VAL	TEMPSENSE_32K_CLK_GATE	Temp Sensor
NA	SYS_CLK		NA	MSS_ELM_CLK_DIV_VAL	MSS_ELM_CLK_GATE	MSS

6.4.3.5 IP Clock Configurations

<IP>_CLK_SRC_SEL controls the select pin of the corresponding clock GCM. The GCM can take several clock cycles before the clock switch is made. The status of the switch is available on <IP>_CLK_STATUS.CLKINUSE.

<IP>_CLK_DIV_VALcontrols the divider value of the Glitch free divider. The GCD takes several clock cycles before the division takes effect. The status can be observed at <IP>_CLK_STATUS. CURRDIVIDER. The status is reflected only if the clock input to the GCD is available.

Note

IP Clock configuration MMRs are present inside the MSS_RCM module.

6.4.3.5.1 RTI CLOCK

(FREQ = 200 MHz)

1. Program RTIx *GCD* register with the value of 0x000 in-order to switch to a new desired frequency, MSS_RCM.RTIx_CLK_DIV_VAL.CLKDIV = 0x000
2. Poll for the CURRDIVR field of corresponding status register to reflect its new frequency change, MSS_RCM.RTIx_CLK_STATUS.CURRDIVIDER = 0x00
3. Update the RTI0 CLK *GCM* register with the value of 0x222 to select *SYS_CLK* as its source, MSS_RCM.RTIx_CLK_SRC_SEL.CLKSRCSEL = 0x222
4. Poll for the CLKINUSE field of corresponding status register to reflect its new frequency change, MSS_RCM.RTIx_CLK_STATUS.CLKINUSE = 0x04

6.4.3.5.2 WDT CLOCK

(FREQ = 200 MHz)

1. Program WDTx *GCD* register with the value of 0x000 in-order to switch to a new desired frequency, MSS_RCM.WDTx_CLK_DIV_VAL.CLKDIV = 0x000
2. Poll for the CURRDIVR field of corresponding status register to reflect its new frequency change, MSS_RCM.WDTx_CLK_STATUS.CURRDIVIDER = 0x0
3. Update the MSS WDT0 *GCM* register with the value of 0x222 to select *SYS_CLK* as its source, MSS_RCM.WDTx_CLK_SRC_SEL.CLKSRCSEL = 0x222
4. Poll for the CLKINUSE field of corresponding status register to reflect its new frequency change, MSS_RCM.WDTx_CLK_STATUS.CLKINUSE = 0x04

6.4.3.5.3 QSPI CLOCK

(FREQ = 80 MHz, note – ROM is utilizing QSPI @ 40 MHz so program the *GCD* correspondingly)

1. Program QSPI *GCD* register with the value of 0x444 in-order to switch to a new desired frequency, MSS_RCM.QSPI_CLK_DIV_VAL.CLKDIV = 0x444
2. Poll for the CURRDIVR field of corresponding status register to reflect its new frequency change, MSS_RCM.QSPI_CLK_STATUS.CURRDIVIDER = 0x4
3. Update the QSPI *GCM* register with the value of 0x444 to select *PLL_CORE_CLKOUT0* clock as its source, MSS_RCM.QSPI_CLK_SRC_SEL.CLKSRCSEL = 0x444
4. Poll for the CLKINUSE field of corresponding status register to reflect its new frequency change, MSS_RCM.QSPI_CLK_STATUS.CLKINUSE = 0x10
5. Program DCLK_DIV field from the SPI_CLOCK_CNTRL register in MSS_QSPI memory map with the value of 0x00, MSS_QSPI.SPI_CLOCK_CNTRL.DCLK_DIV = 0x00

Baud rate relationship with QSPI functional clock frequency:

Baud rate = fQSPI / DCLK_DIV

Where fQSPI – QSPI Functional clock frequency
DCLK_DIV – Prescalar clock divider

6.4.3.5.4 MCSPI CLOCK

(FREQ = 50 MHz, Baud rate = 50Mbps)

1. Program MCSPIx *GCD* register with the value of 0x333 in-order to switch to a new desired frequency, MSS_RCM.MCSPIx_CLK_DIV_VAL.CLKDIV = 0x333
2. Poll for the CURRDIVR field of corresponding status register to reflect its new frequency change, MSS_RCM.MCSPIx_CLK_STATUS.CURRDIVIDER = 0x03
3. Update the MCSPI0 *GCM* register with the value of 0x444 to select *PLL_CORE_CLKOUT0* clock as its source, MSS_RCM.MCSPIx_CLK_SRC_SEL.CLKSRCSEL = 0x444
4. Poll for the CLKINUSE field of corresponding status register to reflect its new frequency change, MSS_RCM.MCSPIx_CLK_STATUS.CLKINUSE = 0x10
5. Program the CLKD field from the required channel config register of the corresponding instance with the value of 0x1, MCSPIx.CHxCONF.CLKD = 0x1

(FREQ = 48 MHz, Baud rate = 48Mbps)

1. Program MCSPIx *GCD* register with the value of 0x333 in-order to switch to a new desired frequency, MSS_RCM.MCSPIx_CLK_DIV_VAL.CLKDIV = 0x333
2. Poll for the CURRDIVR field of corresponding status register to reflect its new frequency change, MSS_RCM.MCSPIx_CLK_STATUS.CURRDIVIDER = 0x03
3. Update the MCSPI0 *GCM* register with the value of 0x333 to select *PLL_PER_CLKOUT1* clock as its source, MSS_RCM.MCSPIx_CLK_SRC_SEL.CLKSRCSEL = 0x333
4. Poll for the CLKINUSE field of corresponding status register to reflect its new frequency change, MSS_RCM.MCSPIx_CLK_STATUS.CLKINUSE = 0x08
5. Program the CLKD field from the required channel config register of the corresponding instance with the value of 0x1, MSS_RCM.MCSPIx.CHxCONF.CLKD = 0x1

Baud rate relationship with MCSPI functional clock frequency,

Baud rate = fSPI / CLKD

Where fSPI – SPI Functional clock frequency
CLKD – Prescalar clock divider

6.4.3.5.5 I2C CLOCK

(FREQ = 48 MHz, Baud rate = 400KHz)

1. Program I2C *GCD* register with the value of 0x333 in-order to switch to a new desired frequency, MSS_RCM.I2C_CLK_DIV_VAL.CLKDIV = 0x333
2. Poll for the CURRDIVR field of corresponding status register to reflect its new frequency change, MSS_RCM.I2C_CLK_STATUS.CURRDIVIDER = 0x03
3. Update the I2C *GCM* register with the value of 0x333 to select *PLL_PER_CLKOUT1* as its source, MSS_RCM.I2C_CLK_SRC_SEL.CLKSRCSEL = 0x333
4. Poll for the CLKINUSE field of corresponding status register to reflect its new frequency change, MSS_RCM.I2C_CLK_STATUS.CLKINUSE = 0x08
5. Program ICCL15_ICCL0 field of ICCLKL register of the corresponding I2C instance with calculated value of 0x35 to attain the 400KHz baud rate, MSS_I2Cx.ICCLKL. ICCL15_ICCL0 = 0x35
6. Program ICCL15_ICCH0 field of ICCLKH register of the corresponding I2C instance with calculated value of 0x35 to attain the 400KHz baud rate, MSS_I2Cx.ICCLKH. ICCL15_ICCH0 = 0x35

Baud rate relationship with I2C functional clock frequency,

SCL_LOW_PERIOD = [fI2C] * [IPSC+1] * [ICCLKL + d]

SCL_HIGH_PERIOD = [fI2C] * [IPSC+1] * [ICCLKH + d]

where fI2C – I2C functional clock frequency,
IPSC – Prescalar value
d – Constant w.r.t to prescaler (IPSC = 0,6 then d = 7, IPSC = 1,5 then d = 6)
ICCLKL – I2C clock low divider
ICCLKH – I2C clock high divider

6.4.3.5.6 LIN_UART CLOCK

(FREQ = 192 MHz)

1. Program LIN_UART *GCD* register with the value of 0x000 in-order to switch to a new desired frequency, MSS_RCM.LINx_UARTx_CLK_DIV_VAL.CLKDIV = 0x000
2. Poll for the CURRDIVR field of corresponding status register to reflect its new frequency change, MSS_RCM.LINx_UARTx_CLK_STATUS.CURRDIVIDER = 0x00
3. Update the LIN_UART *GCM* register with the value of 0x333 to select *PLL_PER_CLKOUT1* as its source, MSS_RCM.LINx_UARTx_CLK_SRC_SEL.CLKSRCSEL = 0x333
4. Poll for the CLKINUSE field of corresponding status register to reflect its new frequency change, MSS_RCM.LINx_UARTx_CLK_STATUS.CLKINUSE = 0x08

(FREQ = 160 MHz)

1. Program LIN_UART *GCD* register with the value of 0x000 in-order to switch to a new desired frequency, MSS_RCM.LINx_UARTx_CLK_DIV_VAL.CLKDIV = 0x000
2. Poll for the CURRDIVR field of corresponding status register to reflect its new frequency change, MSS_RCM.LINx_UARTx_CLK_STATUS.CURRDIVIDER = 0x00
3. Update the LIN_UART *GCM* register with the value of 0x777 to select *PLL_PER_CLKOUT0* as its source, MSS_RCM.LINx_UARTx_CLK_SRC_SEL.CLKSRCSEL = 0x777
4. Poll for the CLKINUSE field of corresponding status register to reflect its new frequency change, MSS_RCM.LINx_UARTx_CLK_STATUS.CLKINUSE = 0x80

For LIN, Baud rate = 20 Kbps then functional clock = 48MHz

1. Program SCI_LIN_PSL field of BRSR register of the corresponding LIN instance with calculated value of 0x95 to attain the required baud rate, MSS_LINx.BRSR.SCI_LIN_PSL = 0x95

Baud rate relationship with LIN functional clock frequency,

Baud rate = fLIN / [16 * (P + 1 + M/16)]

Where FLIN – LIN Functional clock
P – Prescalar to select baudrate
M – Prescalar for fine tuning of baudrate

For UART, Baud rate = 12 Mbps then functional clock = 192MHz,

Baud rate = 10 Mbps then functional clock = 160 MHz

1. Program CLOCK_LSB field of DLL register of the corresponding UART instance with calculated value of 0x1 to attain the required baud rate, MSS_UARTx.DLL.CLOCK_LSB = 0x1
2. Program CLOCK_MSB field of DLH register of the corresponding UART instance with calculated value of 0x0 to attain the required baud rate, MSS_UARTx.DLH.CLOCK_MSB = 0x0

Baud rate relationship with LIN functional clock frequency,

Baud rate = fUART / [16 * DIV]

Where FUART – UART Functional clock
DIV – Prescalar clock divider

6.4.3.5.7 ICSSM UART CLOCK

(FREQ = 192 MHz)

1. Program ICSSM UART *GCD* register with the value of 0x000 in-order to switch to a new desired frequency, MSS_RCM.ICSSMx_UARTx_CLK_DIV_VAL.CLKDIV = 0x000
2. Poll for the CURRDIVR field of corresponding status register to reflect its new frequency change, MSS_RCM.ICSSMx_UARTx_CLK_STATUS.CURRDIVIDER = 0x00
3. Update the ICSSM UART *GCM* register with the value of 0x333 to select *PLL_PER_CLKOUT1* as its source, MSS_RCM.ICSSMx_UARTx_CLK_SRC_SEL.CLKSRCSEL = 0x333

4. Poll for the CLKINUSE field of corresponding status register to reflect its new frequency change, MSS_RCM.ICSSMx_UARTx_CLK_STATUS.CLKINUSE = 0x08

6.4.3.5.8 MCAN CLOCK

(FREQ = 80 MHz, Baud Rate = 8 Mbps)

1. Program MCAN *GCD* register with the value of 0x444 in-order to switch to a new desired frequency, MSS_RCM.MCANx_CLK_DIV_VAL.CLKDIV = 0x444
2. Poll for the CURRDIVR field of corresponding status register to reflect its new frequency change, MSS_RCM.MCANx_CLK_STATUS.CURRDIVIDER = 0x04
3. Update the MCANx *GCM* register with the value of 0x444 to select *PLL_CORE_CLKOUT0* as its source, MSS_RCM.MCANx_CLK_SRC_SEL.CLKSRCSEL = 0x444
4. Poll for the CLKINUSE field of corresponding status register to reflect its new frequency change, MSS_RCM.MCANx_CLK_STATUS.CLKINUSE = 0x10
5. Program NBRP field of the NBTP register of the corresponding CAN instance with the calculated value of 0x0 to achieve the required baud rate of 8Mbps, MSS_MCANx.NBTP. NBRP = 0x0
6. Program NTSEG1 field of the NBTP register of the corresponding CAN instance with the value calculated from the formula to achieve the required baud rate, MSS_MCANx.NTSEG1.NBRP = 0x1
7. Program NTSEG2 field of the NBTP register of the corresponding CAN instance with the value calculated from the formula to achieve the required baud rate, MSS_MCANx.NTSEG2.NBRP = 0x1

Baud rate relationship with CAN functional clock frequency,

Baud rate = fCAN / [2 * (BRP +1) * (3 + TSEG1 + TSEG2)]

Where fCAN – MCAN Functional clock frequency
BRP – Baudrate prescaler
TSEG1 – Time segment before sample point
TSEG2 – Time segment after sample point

6.4.3.5.9 MMCx CLOCK

(FREQ = 50 MHz)

1. Program MMCSD *GCD* register with the value of 0x333 in-order to switch to a new desired frequency, MSS_RCM.MMCx_CLK_DIV_VAL.CLKDIV = 0x333
2. Poll for the CURRDIVR field of corresponding status register to reflect its new frequency change, MSS_RCM.MMCx_CLK_STATUS.CURRDIVIDER = 0x03
3. Update the MMCx *GCM* register with the value of 0x222 to select *PLL_PER_CLKOUT1* as its source, MSS_RCM.MMCx_CLK_SRC_SEL.CLKSRCSEL = 0x222
4. Poll for the CLKINUSE field of corresponding status register to reflect its new frequency change, MSS_RCM.MMCx_CLK_STATUS.CLKINUSE = 0x04

(FREQ = 48 MHz)

1. Program MMCSD *GCD* register with the value of 0x111 in-order to switch to a new desired frequency, MSS_RCM.MMCx_CLK_DIV_VAL.CLKDIV = 0x333
2. Poll for the CURRDIVR field of corresponding status register to reflect its new frequency change, MSS_RCM.MMCx_CLK_STATUS.CURRDIVIDER = 0x03
3. Update the MMCx *GCM* register with the value of 0x333 to select *PLL_PER_CLKOUT1* as its source, MSS_RCM.MMCx_CLK_SRC_SEL.CLKSRCSEL = 0x333
4. Poll for the CLKINUSE field of corresponding status register to reflect its new frequency change, MSS_RCM.MMCx_CLK_STATUS.CLKINUSE = 0x08

6.4.3.5.10 CPTS CLOCK

(FREQ = 250 MHz)

1. Program CPTS *GCD* register with the value of 0x111 in-order to switch to a new desired frequency, MSS_RCM.CPTS_CLK_DIV_VAL.CLKDIV = 0x111
2. Poll for the CURRDIVR field of corresponding status register to reflect its new frequency change, MSS_RCM.CPTS_CLK_STATUS.CURRDIVIDER = 0x01

3. Update the CPTS *GCM* register with the value of 0x333 to select *PLL_PER_CLKOUT1* as its source, MSS_RCM.CPTS_CLK_SRC_SEL.CLKSRCSEL = 0x333
4. Poll for the CLKINUSE field of corresponding status register to reflect its new frequency change, MSS_RCM.CPTS_CLK_STATUS.CLKINUSE = 0x08

6.4.3.5.11 HSM RTI CLOCK

(FREQ = 200 MHz)

1. Program HSM RTI *GCD* register with the value of 0x000 in-order to switch to a new desired frequency, MSS_RCM.HSM_RTI_CLK_DIV_VAL.CLKDIV = 0x000
2. Poll for the CURRDIVR field of corresponding status register to reflect its new frequency change, MSS_RCM.HSM_RTI_CLK_STATUS.CURRDIVIDER = 0x00
3. Update the HSM RTI *GCM* register with the value of 0x222 to select *SYS_CLK* as its source, MSS_RCM.HSM_RTI_CLK_SRC_SEL.CLKSRCSEL = 0x222
4. Poll for the CLKINUSE field of corresponding status register to reflect its new frequency change, MSS_RCM.HSM_RTI_CLK_STATUS.CLKINUSE = 0x04

6.4.3.5.12 HSM WDT CLOCK

(FREQ = 200 MHz)

1. Program HSM WDT *GCD* register with the value of 0x000 in-order to switch to a new desired frequency, MSS_RCM.HSM_WDT_CLK_DIV_VAL.CLKDIV = 0x000
2. Poll for the CURRDIVR field of corresponding status register to reflect its new frequency change, MSS_RCM.HSM_WDT_CLK_STATUS.CURRDIVIDER = 0x00
3. Update the HSM WDT *GCM* register with the value of 0x222 to select *SYS_CLK* as its source, MSS_RCM.HSM_WDT_CLK_SRC_SEL.CLKSRCSEL = 0x222
4. Poll for the CLKINUSE field of corresponding status register to reflect its new frequency change, MSS_RCM.HSM_WDT_CLK_STATUS.CLKINUSE = 0x04

6.4.3.5.13 HSM RTC CLOCK

(FREQ = 200 MHz)

1. Program HSM RTC *GCD* register with the value of 0x000 in-order to switch to a new desired frequency, MSS_RCM.HSM_RTC_CLK_DIV_VAL.CLKDIV = 0x000
2. Poll for the CURRDIVR field of corresponding status register to reflect its new frequency change, MSS_RCM.HSM_RTC_CLK_STATUS.CURRDIVIDER = 0x00
3. Update the HSM RTC *GCM*register with the value of 0x222 to select SYS_CLKas its source, MSS_RCM.HSM_RTC_CLK_SRC_SEL.CLKSRCSEL = 0x222
4. Poll for the CLKINUSE field of corresponding status register to reflect its new frequency change, It should read HSM_RTC_CLK_STATUS.CLKINUSE = 0x04

6.4.3.5.14 HSM DMTA CLOCK

(FREQ = 200 MHz)

1. Program HSM DMTA *GCD* register with the value of 0x000 in-order to switch to a new desired frequency, MSS_RCM.HSM_DMTA_CLK_DIV_VAL.CLKDIV = 0x000
2. Poll for the CURRDIVR field of corresponding status register to reflect its new frequency change, MSS_RCM.HSM_DMTA_CLK_STATUS.CURRDIVIDER = 0x00
3. Update the HSM DMTA *GCM* register with the value of 0x222 to select *SYS_CLK* as its source, MSS_RCM.HSM_DMTA_CLK_SRC_SEL.CLKSRCSEL = 0x222
4. Poll for the CLKINUSE field of corresponding status register to reflect its new frequency change, MSS_RCM.HSM_DMTA_CLK_STATUS.CLKINUSE = 0x04

6.4.3.5.15 HSM DMTB CLOCK

(FREQ = 200 MHz)

1. Program HSM DMTB *GCD* register with the value of 0x000 in-order to switch to a new desired frequency, MSS_RCM.HSM_DMTB_CLK_DIV_VAL.CLKDIV = 0x000

2. Poll for the CURRDIVR field of corresponding status register to reflect its new frequency change, MSS_RCM.HSM_DMTB_CLK_STATUS.CURRDIVIDER = 0x00
3. Update the HSM DMTB *GCM* register with the value of 0x222 to select *SYS_CLK* as its source, MSS_RCM.HSM_DMTB_CLK_SRC_SEL.CLKSRCSEL = 0x222
4. Poll for the CLKINUSE field of corresponding status register to reflect its new frequency change, MSS_RCM.HSM_DMTB_CLK_STATUS.CLKINUSE = 0x04

6.4.3.5.16 CONTROLSS PLL CLOCK

(FREQ = 400 MHz)

1. Program CONTROLSS PLL CLOCK *GCD*register with the value of 0x000 in-order to switch to a new desired frequency, MSS_RCM.CONTROLLSS_PLL_CLK_DIV_VAL.CLKDIVR = 0x000
2. Poll for the CURRDIVR field of corresponding status register to reflect its new frequency change, It should read CONTROLLSS_PLL_CLK_STATUS.CURRDIVIDER = 0x00
3. Update the CONTROLSS *GCM*register with the value of 0x222 to select *PLL_CORE_CLKOUT2*as its source, MSS_RCM.CONTROLSS_PLL_CLK_SRC_SEL.CLKSRCSEL = 0x222
4. Poll for the CLKINUSE field of corresponding status register to reflect its new frequency change, It should read MSS_RCM.CONTROLLSS_PLL_CLK_STATUS.CLKINUSE = 0x04

6.4.3.5.17 RGMII5 CLK

(FREQ = 5 MHz, Default configuration)

1. Program RGMII5 *GCD* register with the value of 0x636363 to obtain a new desired frequency divided from *PLL_CORE_CLKOUT1*, MSS_RCM.RGMII5_CLK_DIV_VAL.CLKDIV = 0x636363
2. Poll for the CURRDIVR field of corresponding status register to reflect its new frequency change, MSS_RCM.RGMII5_CLK_STATUS.CURRDIVIDER = 0x63

6.4.3.5.18 RGMII50 CLK

(FREQ = 50 MHz, Default configuration)

1. Program RGMII50 *GCD* register with the value of 0x999 to obtain a new desired frequency divided from *PLL_CORE_CLKOUT1*, MSS_RCM.RGMII50_CLK_DIV_VAL.CLKDIV = 0x999
2. Poll for the CURRDIVR field of corresponding status register to reflect its new frequency change, MSS_RCM.RGMII50_CLK_STATUS.CURRDIVIDER = 0x09

6.4.3.5.19 RGMII250 CLK

(FREQ = 250 MHz, Default configuration)

1. Program RGMII250 *GCD* register with the value of 0x111 to obtain a new desired frequency divided from *PLL_CORE_CLKOUT1*, MSS_RCM.RGMII_CLK_DIV_VAL.CLKDIV = 0x111
2. Poll for the CURRDIVR field of corresponding status register to reflect its new frequency change, MSS_RCM.RGMII_CLK_STATUS.CURRDIVIDER = 0x01

6.4.3.5.20 XTAL MMC 32K CLOCK

(FREQ = 32 KHz, Default configuration)

1. Program XTAL MMC 32K *GCD* register with the value of 0x30CC330C to obtain a new desired frequency divided from XTAL_CLK, MSS_RCM.XTAL_MMC_32K_CLK_DIV_VAL.CLKDIV = 0x30CC330C
2. Poll for the CURRDIVR field of corresponding status register to reflect its new frequency change, MSS_RCM.XTAL_MMC_32K _CLK_STATUS. CURRDIVIDER = 0x30C

Note

Please refer to the register description in the AM263Px Sitara Processors Technical Reference Manual Register Addendum for a more detailed explanation on how to configure the XTAL MMC 32K Clock

6.4.3.5.21 XTAL TEMPSENSE 32K CLOCK

(FREQ = 32 KHz, Default configuration)

1. Program XTAL TEMPSENSE 32K *GCD* register with the value of 0x30CC330C to obtain a new desired frequency divided from XTAL_CLK, MSS_RCM.XTAL_TEMPSENSE_32K_CLK_DIV_VAL.CLKDIV = 0x30CC330C
2. Poll for the CURRDIVR field of corresponding status register to reflect its new frequency change, MSS_RCM.XTAL_TEMPSENSE_32K_CLK_STATUS.CURRDIVIDER = 0x30C

Note

Please refer to the register description in the AM263Px Sitara Processors Technical Reference Manual Register Addendumfor a more detailed explanation on how to configure the XTAL TEMPSENSE 32K Clock

6.4.3.5.22 MSS_ELM CLOCK

(FREQ = 50 MHz, Default configuration)

1. Program MSS ELM *GCD* register with the value of to obtain a new desired frequency divided from *SYS_CLK*, MSS_RCM.MSS_LM_CLK_DIV_VAL.CLKDIV = 0x03
2. Poll for the CURRDIVR field of corresponding status register to reflect its new frequency change, MSS_RCM.MSS_ELM_CLK_STATUS.CURRDIVIDER = 0x03

6.4.4 Limp Mode

A dedicated coarse clock loss logic checks the XTAL clock against the RC CLK continuously to detect if the XTAL clock is toggling.

When this module detects a clock error on XTAL Clock, the error signal is routed to the GCM's to activate the Limp mode. When the limp mode is activated, all the GCM switch to RCCLK (clk source #5). This ensures the CPU continues to operate even if the XTAL Clock fails and can take the system to a safe state.

Switch to Limp mode feature is not enabled by default. The feature needs to be explicitly enabled by S/W by TOP_RCM.LIMP_MODE_EN.XTALCLK_LOSS_EN

In addition, error on DCC0 and CORE_PLL Phase lock loss can also trigger Limp-mode for added safety. This feature can be enabled by the bits TOP_RCM.LIMP_MODE_EN.COREPLL_LOSS_EN and TOP_RCM.LIMP_MODE_EN.DCC0_ERROR_EN

Note

RC_CLK is not an accurate clock and hence the performance of the system is not guaranteed in Limp mode.

6.4.5 Clocking Registers

For additional details related to device clocking registers, please refer to the *Control Module - MSS_RCM Registers* section of the Register Addendum.

7 Processors and Accelerators

This chapter describes the Processor and Accelerator modules in the device.

7.1 Arm Cortex R5F Subsystem (R5FSS)

This chapter describes the Arm Cortex R5F real-time microcontroller unit subsystem (R5FSS) in the device. There are two subsystems in the SoC named R5FSS0 and RF5SS1. The only difference between the two subsystem is that RF5SS0 has a ROM image of 128kB and R5FSS1 has no ROM. The SoC memory map for R5FSS0 with and without ROM is provided in Section 2.2. The ROM image handles initial configuration for the R5FSS0 CORE0 and initiates the secondary boot loader (SBL) for application download.

7.1.1 R5FSS Overview

The R5FSS is a dual-core implementation of the Arm® Cortex®-R5F processor configured for Dual-Core or Lockstep 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.

Note

The Cortex-R5F processor is a Cortex-R5 processor that includes the optional floating point unit (FPU) extension. In this TRM, all references to the Cortex-R5 processor apply to the Cortex-R5F processor by default.

7.1.1.1 R5FSS Features

Each R5FSS supports the following features:

- Dual-core Arm Cortex-R5F
  - Core revision: r1p3
  - Armv7-R profile
  - Dual-core and Lockstep mode support
  - • Dual-core mode: Two independently operating cores (asymmetric multi processing, no coherence)
    • Lockstep mode: One operating core (CORE0) and One lockstep core (CORE1)
      - CORE0 uses TCM resources of both cores
      - CORE1 caches and interrupts are unused in this mode
    • Support for switching to Dual-core mode from Lockstep mode by application (Efuse-enabled feature) - by triggering a CPU reset. (See device specific Datasheet for additional details.)
  - L1 memory system
    • 16KB instruction cache
      - 4x4KB ways
      - SECDED ECC protected per 64 bits
    • 16KB data cache
      - 4x4KB ways
      - SECDED ECC protected per 32 bits
    • 256KB tightly-coupled memory (TCM) per core
      - 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)
      - In Dual-Core mode, TCM is 128KB per core:
        • 32KB TCMA
        • 48KB TCMB0 + 48KB TCMB1
      - In Lockstep mode, TCM is 256KB in total:
        • 64KB TCMA
        • 96KB TCMB0 + 96KB TCMB1
  - L2 Memory System
    • AXI L2 port for each core with following features:
      - 4 Region Address Translator (RAT)
      - 4 Region Fast Local Copy (FLC) engine to accelerate boot process
      - Remote L2 cache controller with integrated tag RAM
  - Accelerator support
    • Each core has a Trignometric Math Unit (TMU) accelerator on the TCMA interface
  - 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 initiator pair (1 read, 1 write) for L2 memory accesses (per core)
  - 64-bit VBUSM target for TCM access (per core)
    - Also allows access to cache for debug purposes and for the TMU
  - 32-bit VBUSP initiator for peripheral access (per core)
  - 4x 32-bit VBUSP target configuration port (2x ECC Aggregator + 1x CCMR + 1x STC)
  - 32-bit VBUSP target debug port
    - Allows access to all R5FSS internal debug logic
- Synchronous clock domain crossing on all interfaces
  - CPU and interface clocks run at a 2:1 frequency ratio
- Integrated vectored interrupt manager (VIM)
  - 256 interrupts per core
    - Only interrupts connected to R5F CORE0 are available in Lockstep 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
    - Dual-Core or Lockstep capable
    - Software interrupt generation
- Integrated ECC aggregators
  - Support for error injection to all supported ECC memory blocks to test ECC functionality in safety critical applications
  - 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 CORE) and cross trigger matrix (CTM) components
  - Processor trace: Supported by embedded trace macrocell (ETM) (per CORE) and advanced trace bus (ATB) funnel components

See Section 7.1.3 for a functional block diagram and additional details related to the R5FSS.

7.1.1.2 R5FSS Not Supported Features

The R5FSS does *not* support the following native R5F features in this device:

- ACP port (no coherence)
- Multiple power domains

7.1.2 R5FSS Integration

This section describes the R5FSS integration in the device, including information about clocks, resets, and hardware requests.

7.1.2.1 R5FSS Integration

There are 2x R5FSS modules integrated in the device. The diagram below provides a visual representation of the device integration details.

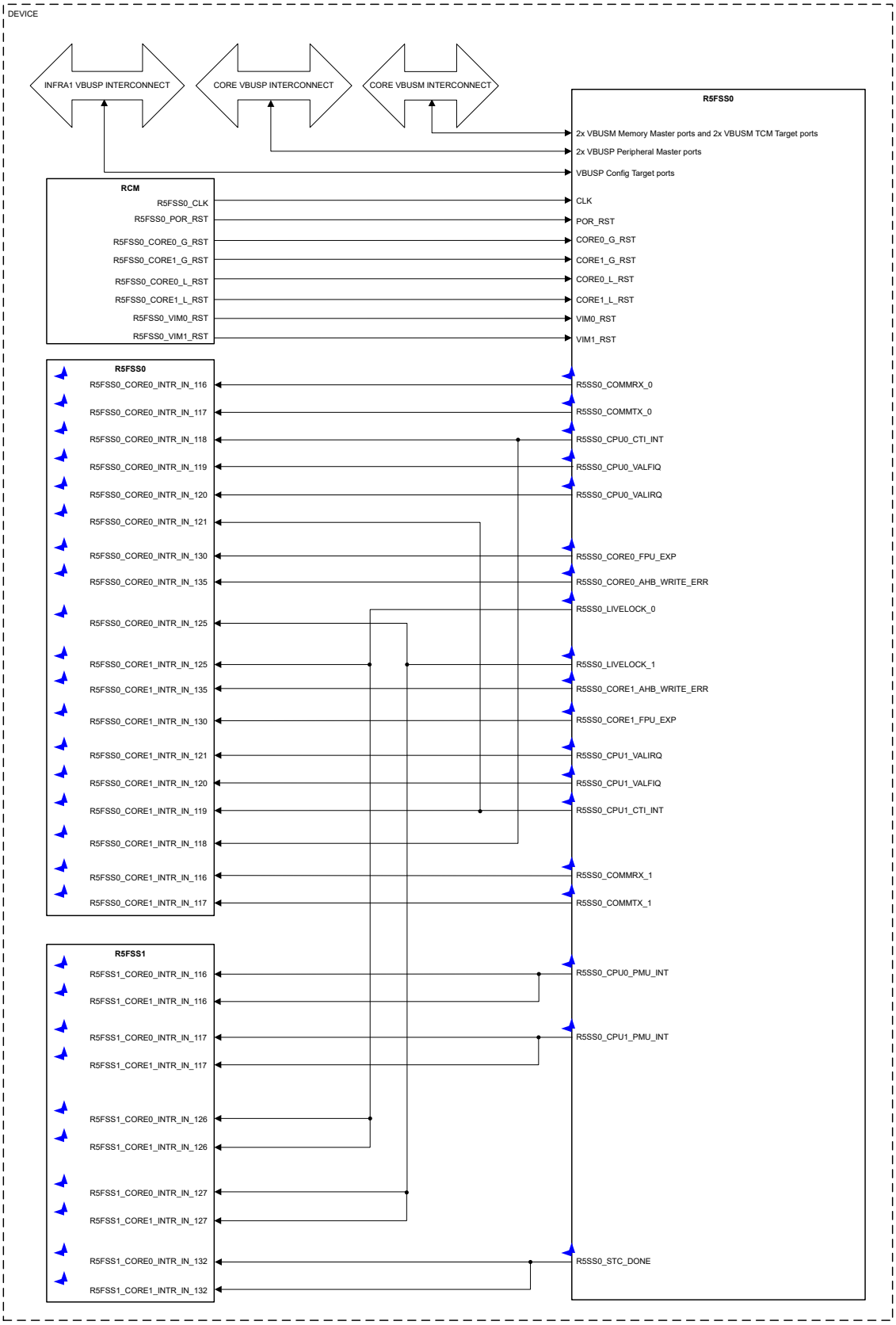


Figure 7-1. R5FSS0 Integration Diagram 1

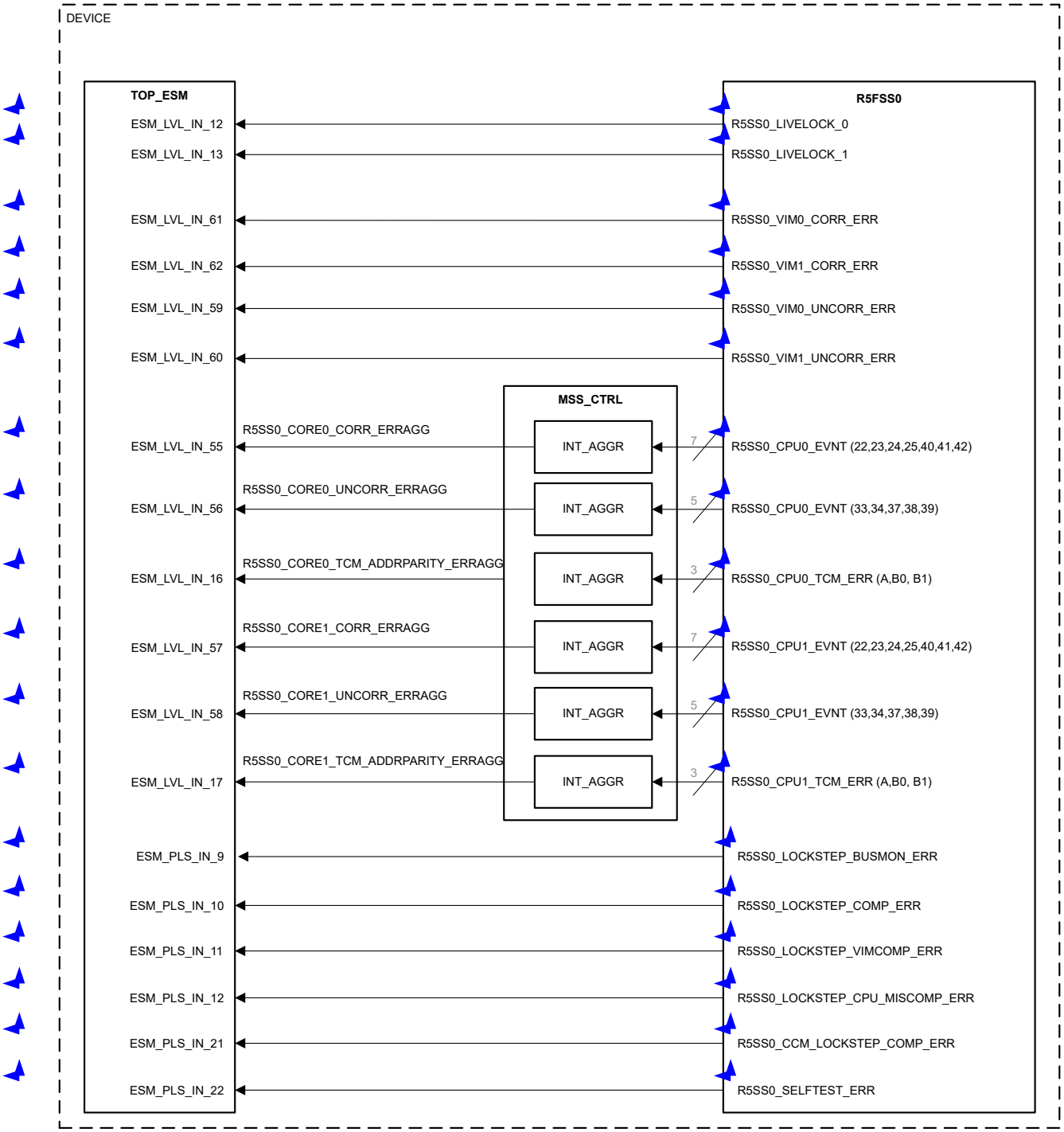


Figure 7-2. R5FSS0 Integration Diagram 2

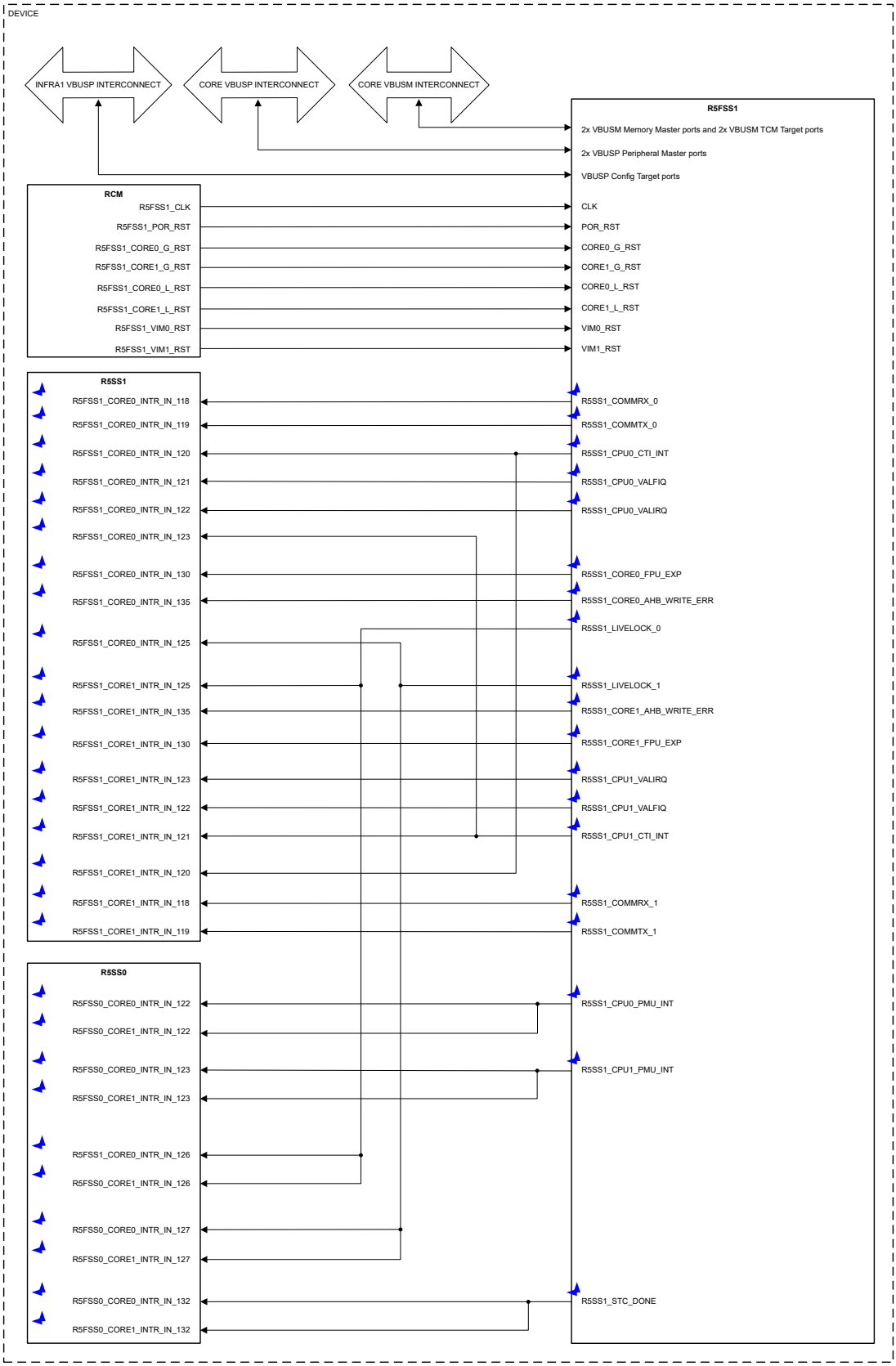


Figure 7-3. R5FSS1 Integration Diagram 1

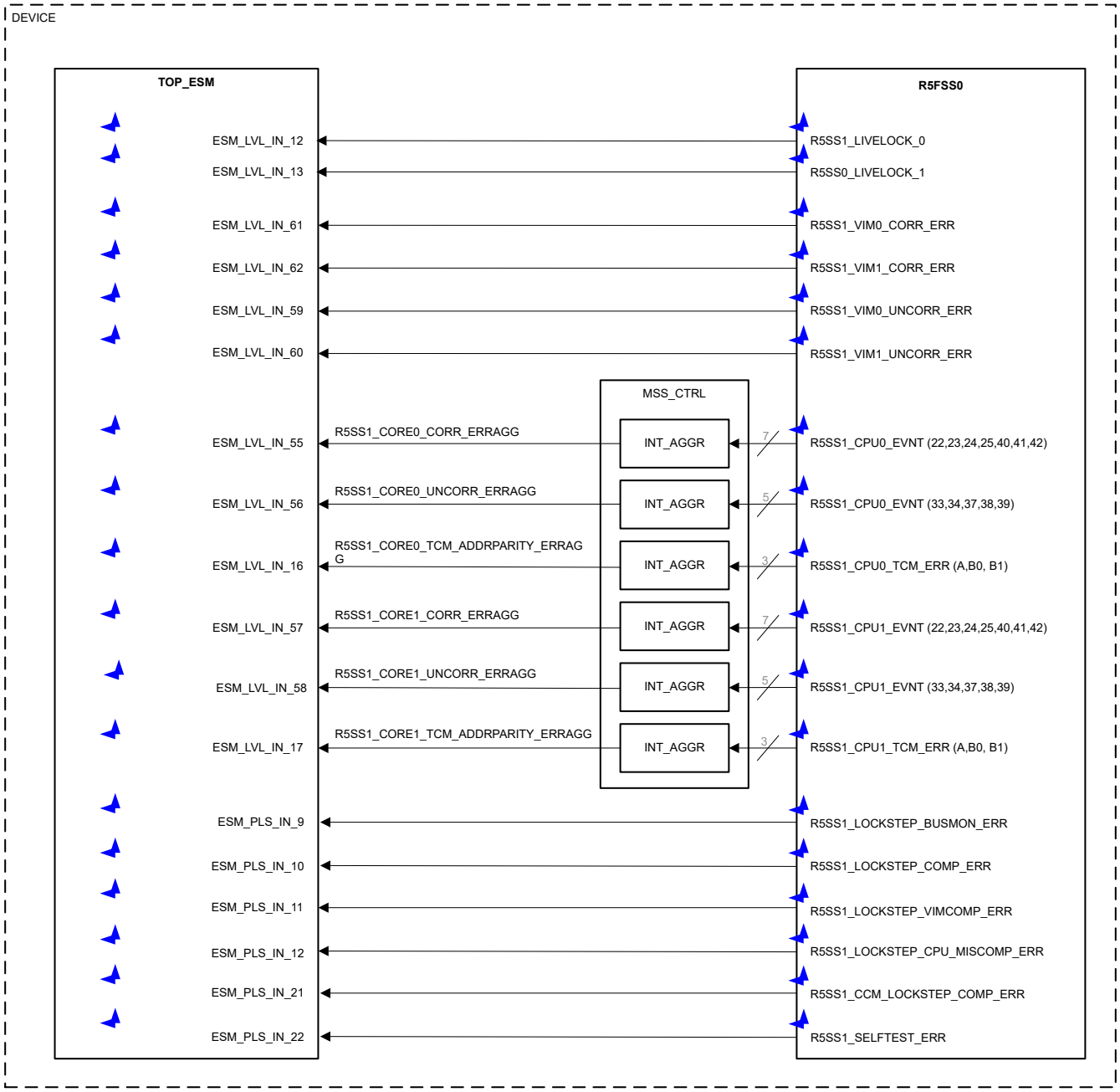


Figure 7-4. R5FSS1 Integration Diagram 2

The tables below summarize the device integration details of R5FSS0/1.

Table 7-1. R5FSS[0:1]Device Integration

This table describes the module device integration details.

Module Instance	SoC Interconnect
R5FSS[0:1]_CORE[0:1]	CORE VBUSM Interconnect
	CORE VBUSP Interconnect
	INFRA1 VBUSP Interconnect

Table 7-2. R5FSS[0:1] Clocks

This table describes the module clocking signals.

Module Instance	Module Clock Input	Source Clock Signal	Source		Description
R5FSS0	CLK	R5FSS0_CLK	MSS_RCM		Functional Clock. Interface clock is derived from functional clock
R5FSS1	CLK	R5FSS1_CLK	MSS_RCM		Functional Clock. Interface clock is derived from functional clock

Table 7-3. R5FSS[0:1] Resets

Module Instance	Module Reset Input	Source Reset Signal	Source	Description
R5FSS0	POR_RST	R5FSS0_POR_RST	MSS_RCM	R5FSS0 Power on Reset
	CORE0_G_RST	R5FSS0_CORE0_G_RST	MSS_RCM	R5FSS0 Core0 Subsystem reset
	CORE1_G_RST	R5FSS0_CORE1_G_RST	MSS_RCM	R5FSS0 Core1 Subsystem reset
	CORE0_L_RST	R5FSS0_CORE0_L_RST	MSS_RCM	R5FSS0 Core0 Local Reset
	CORE1_L_RST	R5FSS0_CORE1_L_RST	MSS_RCM	R5FSS0 Core1 Local Reset
	VIM0_RST	R5FSS0_VIM0_RST	MSS_RCM	R5FSS0 VIM0 Reset
	VIM1_RST	R5FSS0_VIM1_RST	MSS_RCM	R5FSS0 VIM1 Reset
R5FSS1	POR_RST	R5FSS1_POR_RST	MSS_RCM	R5FSS1 Power on Reset
	CORE0_RST	R5FSS1_CORE0_G_RST	MSS_RCM	R5FSS1 Core0 Main reset
	CORE1_RST	R5FSS1_CORE1_G_RST	MSS_RCM	R5FSS1 Core1 Main reset
	CORE0_L_RST	R5FSS1_CORE0_L_RST	MSS_RCM	R5FSS0 Core0 Local Reset
	CORE1_L_RST	R5FSS1_CORE1_L_RST	MSS_RCM	R5FSS0 Core1 Local Reset
	VIM0_RST	R5FSS1_VIM0_RST	MSS_RCM	R5FSS1 VIM0 Reset
	VIM1_RST	R5FSS1_VIM1_RST	MSS_RCM	R5FSS1 VIM1 Reset

Note

For more information on the interconnects, see the *System Interconnect* chapter.

For more information on power, reset, and clock management, see the corresponding sections within the *Device Configuration* chapter.

For more information on the device interrupt controllers, see the *Interrupt Controllers* chapter.

7.1.3 R5FSS Functional Description

7.1.3.1 R5FSS Block Diagram

Figure 7-5 shows the R5FSS block diagram.

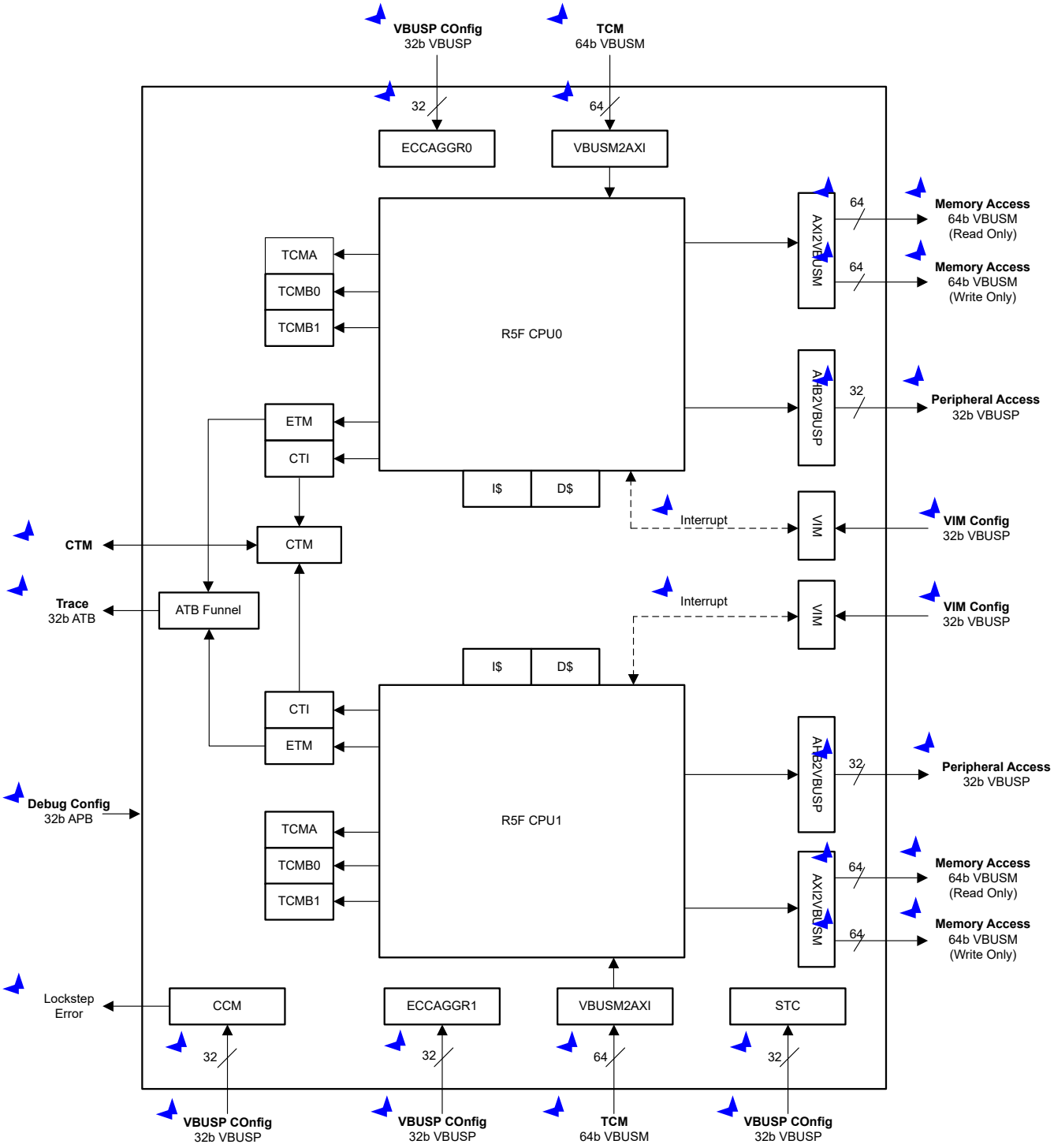


Figure 7-5. R5FSS Block Diagram

7.1.3.2 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: Dual Core or Lockstep.

In dual core mode, each R5F core works completely independent from the other (asymmetric multiprocessing, or AMP). Each core uses its own RAMs and interfaces, with no coherence between the two cores.

In Lockstep mode:

- CPU0 is the only operating core
- CPU1 operates as the lockstep core for CPU0
- CPU1 TCMs are stacked on CPU0 TCMs and are accessible only by CPU0 and CPU0 TCM interface
- The TCM size for CPU0 is essentially doubled in this mode (128KB)
- CPU1 caches and interrupts are not used

For a brief list of features supported by the R5F processor in this device, see Section 7.1.1.1. For more detailed description of this processor, see the *Arm Cortex-R5 Technical Reference Manual*.

7.1.3.2.1 L1 Caches

The R5F cores have a Harvard cache architrecture, which means each core has an independent L1 instruction cache (16KB) and L1 data cache (16KB). The instruction cache is protected by SECDED ECC per 64 bits. The data cache is protected by SECDED ECC per 32 bits.

7.1.3.2.2 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 bootstrap signals having a default value of 1 in this device. 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). Note also that the ATCM and BTCM sizes for CPU0 is 64KB each in Lockstep mode.

If a TCM is not enabled, then it does not appear in the R5F's 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. The base address of ATCM is 0x0000_0000 and the base address of BTCM is 0x0008_0000.

It is possible to preload a TCM with instructions and boot from it. See R5FSS Boot Options for details on TCM booting.

7.1.3.2.3 Trigonometric Math Unit (TMU)

Each R5F subsystem integrates two Trigonometric Math Unit (TMU) modules with one TMU associated to each R5F core. The TMU extends the capabilities of the R5F by speeding up the execution of common trigonometric and arithmetic operations. The instructions from Table 7-4 are supported by the TMU.

Table 7-4. Available Trigonometric Instructions

Instruction Name	Returned Output
SINPUF32	Returns the SINE of the input value
COSPUF32	Returns the COSINE of the input value
ATANPUF32	Returns the ATAN of the input value
QUADF32	Returns the quadrant value and the ratio of X and Y inputs which are provided as per unit values.
IEXP2F32	Returns inverse exponent of the input value
LOG2F32	Returns base-2 logarithm of the input value

The TMU supports the following features:

- The TMU operates at a maximum frequency of 200MHz
  - The TMU can run at either a 1:2 or 1:1 ratio from the core clock
- The TMU is connected to the TCM interface for low latency operation
- TMU has eight result registers to enable eight pipelined operations

Note

TMU operates in lock step or dual core mode in sync with the mode of operation for R5F. The lockstep diagnostics registers of TMU shall be integrated with the lockstep diagnostics registers of R5F to provide a single programing interface.

7.1.3.2.4 Region Address Translator (RAT)

The AXI L2 port of each R5F core integrates a 4 Region Address Translator (RAT). The RAT granularity is from 4kB up to theoretically 4GB and enables virtualized access to shared common library code.

7.1.3.2.5 Fast Local Copy (FLC)

The AXI L2 port of each R5F core integrates a Fast Local Copy (FLC) engine to accelerate boot up. Reducing boot time of R5F is essential to achieve 50ms CAN response.

The FLC engine is designed to leverage how application execution from internal SRAM is significantly faster than execution from external flash. A typical boot process involves critical application code getting copied over from flash to SRAM before executing from SRAM. The copy phase of this process results in increased boot time.

With the FLC engine, boot up is accelerated by redirecting the core access to flash when the copy is in progress and redirecting the access to SRAM when the content is in valid in SRAM. This enables core to execute immediately without waiting for the copy to complete. The operation is transparent to core.

The FLC supports the following features:

- Support up to four flash regions per FLC with 64kB granularity
- Internal DMA engine to copy code from flash to SRAM sequentially for all regions
- Redirect access to flash when the data is not yet available in SRAM
- Redirect access to SRAM when the data is available in SRAM
- Transparent to core after initial configuration of regions
- Pass-through for addresses not in the programmed regions
- The SRAM redirection does not wait till the entire region is mirrored, but is enabled as soon as the content at that address is valid in SRAM
- Interrupt generation when mirroring of region is complete
- Option to disable DMA copy and only enable SRAM redirection

7.1.3.2.6 Remote L2 Cache (RL2)

The AXI L2 port of each R5F core integrates a remote L2 cache controller with integrated tag RAM. AXI is a bus protocol defined by ARM and stands for Advanced eXtensible Interface. The L2 cache is remote which means the cache data memory is allocated from the available L2 SRAM in the device. Once allocated, this SRAM space is unavailable for application code / data.

RL2 supports the following features:

- Integrated tag RAM with 4096 cache lines support
- Eight-way set associative with LRU policy
- RL2 cache line size is aligned with R5F cache line size (32 bytes)
- ECC protection on tag RAM
- Software programmable cache lines to enable application configurable remote cache size of 8KB, 16KB, 32KB, 64KB or 128KB
  - The cache-able target space varies proportionately based on the cache size.
- Dual mode support to enable sharing two cache lines the same way which increases the remote cache size to 256kB while reducing the cache-able region of tag RAM
- Pass-through for addresses not within the cache-able region
- Support critical word first access from R5F towards L2 Interconnect
- Pass-through for accesses originating from FLC DMA

Note

RAT+ FLC+RL2 module operates in lock step or dual core mode in sync with the mode of operation for R5F. The lock step diagnostics registers of RAT+FLC+RL2 are integrated with the lockstep diagnostics registers of R5F to provide a single programing interface.

7.1.3.2.7 R5FSS Special Signals

Table 7-5 through Table 7-6 list some R5FSS features associated with special signals.

Table 7-5. 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 MSS_CTRL R5SS0_TEINIT register setting. Defaults to Arm mode
Dual core or Lockstep mode 0 = Dual Core mode 1 = Lockstep mode	Controlled via MSS_CTRL R5SS0_CONTROL register setting. Defaults to a value defined by eFuse/MMR control
CPUn execution halt when coming out of reset (CPUn_HALT)	Controlled via MSS_CTRL R5SS0_COREx_HALT register setting. Defaults to halted state. See R5 Core Halting and Unhalting for more detail.
CPUn exception vectors base address	Defaults to Bootvector RAM address 0x0000_0000
CPUn VIM base address	0x50F0 0000
CPUn non-maskable fast interrupts enable	Disabled
CPUn VBUSM peripheral port enabled at reset	Disabled, not used.
CPUn VBUSP peripheral port enable at reset	Defaults to Enabled state
CPUn VBUSP peripheral port base address	Defaults to 0x5000_0000
CPUn VBUSP peripheral port size	Defaults to 256MB 0x5000_0000 to 0x5FFFF_FFFF
CPUn VBUSM normal peripheral port base address	Not used
CPUn VBUSM normal peripheral port size	Not used
CPUn VBUSM virtual peripheral port base address	Not used
CPUn VBUSM virtual peripheral port size	Not used

Table 7-5. R5FSS0 Special Features (continued)

Feature	Comment
CPUn WFI state	Status logged into MSS_CTRL R5SS0_COREx_STAT register bit. See the R5 WFI section.
CPUn WFE state	Status logged into MSS_CTRL R5SS0_COREx_STAT register bit. See the R5 WFI section.
CPU Clockgate Control	Controlled via MSS_RCM R5SS0_COREx_GATE register setting. Individual Core clocks can be gated
CPUn TCM Bus Parity	Enabled

Table 7-6. R5FSS1 Special Features

Feature	Comment
Cluster affinity group ID	R5F Cluster 1 (ID = 0x1)
Exception handling state at reset 0 = Arm 1 = Thumb	Controlled via MSS_CTRL R5SS1_TEINIT register setting. Defaults to Arm mode
Dual core or Lockstep mode 0 = Dual Core mode 1 = Lockstep mode	Controlled via MSS_CTRL R5SS1_CONTROL register setting. Defaults to a value defined by eFuse/MMR control
CPUn execution halt when coming out of reset (CPUn_HALT)	Controlled via MSS_CTRL R5SS1_COREx_HALT register setting. Defaults to halted state. See R5 Core Halting and Unhalting for more detail.
CPUn exception vectors base address	Defaults to Bootvector RAM address 0x0000_0000
CPUn VIM base address	0x50F0 0000
CPUn non-maskable fast interrupts enable	Disabled
CPUn VBUSM peripheral port enabled at reset	Disabled, not used.
CPUn VBUSP peripheral port enable at reset	Defaults to Enabled state
CPUn VBUSP peripheral port base address	Defaults to 0x5000_0000
CPUn VBUSP peripheral port size	Defaults to 256MB 0x5000_0000 to 0x5FFFF_FFFF
CPUn VBUSM normal peripheral port base address	Not used
CPUn VBUSM normal peripheral port size	Not used
CPUn VBUSM virtual peripheral port base address	Not used
CPUn VBUSM virtual peripheral port size	Not used
CPUn WFI state	Status logged into MSS_CTRL R5SS1_COREx_STAT register bit. See the R5 WFI section.
CPUn WFE state	Status logged into MSS_CTRL R5SS1_COREx_STAT register bit. See the R5 WFI section.
CPU Clockgate Control	Controlled via MSS_RCM R5SS1_COREx_GATE register setting. Individual Core clocks can be gated
CPUn TCM Bus Parity	Enabled

Switching between Dual Core and Lockstep Mode

By default, the R5FSS[0-1] will be in lockstep mode. Switching to dual core mode or staying in lockstep mode is handled through setting a combination of eFuse and MMR bits. See Table 7-7 for a summary of possible combinations.

Table 7-7. Settings for Dual Core and Lockstep Modes

eFuse BitEFUSE1_ROW_12_R5SS[0-1]_FORCE_DUAL_CORE	eFuse BitEFUSE1_ROW_12_R5SS[0-1]_DUAL_CORE_DISABLE	MMR BitR5SS[0-1]_CONTROL_LOCK_STEP	R5FSS[0-1] Mode
0	0	0	Dual Core
0	0	1	Lockstep (default)

Table 7-7. Settings for Dual Core and Lockstep Modes (continued)

eFuse BitEFUSE1_ROW_12_R5SS[0-1]_FORCE_DUAL_CORE	eFuse BitEFUSE1_ROW_12_R5SS[0-1]_DUAL_CORE_DISABLE	MMR BitR5SS[0-1]_CONTROL_LOCK_STEP	R5FSS[0-1] Mode
1	X	X	Dual Core
0	1	X	Lockstep

Based on the part number, the eFuse bits will decide whether the MMR can be used for switching to dual core. Follow the below sequence in such cases.

- Set R5SSx_RST_ASSERDLY and MSS_RCM.R5SSx_RST2ASSERTDLY registers with required reset asserting and holding delay.
- Set:R5SS[0-1]_CONTROL_LOCK_STEP to 0.
- Set R5SS[0-1]_CONTROL_LOCK_STEP_SWITCH_WAIT to 0 or 7 based on application use case. Setting to 7 is recommended.
- By default, reset FSM (or any reset to R5FSS[0-1]_CORE[0-1]) will wait for CPU to go to WFI state for safe handling of system. Setting R5SS[0-1]_FORCE_WFI_ CR5_WFI_OVERIDE to 7 would override the WFI check but this is not recommended.
- Set R5SS[0-1]_CONTROL_RESET_FSM_TRIGGER to 7 will reset the R5FSS[0-1] and switch mode to dual core.
- Read R5SS[0-1]_STATUS_REG_ LOCK_STEP for status (0 is dual core and 1 is lockstep).

7.1.3.3 R5FSS Interfaces

7.1.3.3.1 Initiator Interfaces

The R5FSS has several controller interfaces per core:

- 64-bit VBUSM controller pair (1 read, 1 write) for L2 memory accesses; this is the main memory interface
- 32-bit VBUSP controller for peripheral access
  - Includes logic that provides the R5F CPU with a private access to VIM
  - Enabled at reset

7.1.3.3.2 Target Interfaces

The R5FSS has several target interfaces that define its internal memory space:

- 32-bit VBUSP configuration target (per core)
  - ECC aggregator block
- 64-bit VBUSM target (per core)
  - ATCM
  - BTCM
  - Instruction cache RAMs
  - Data cache RAMs
- 32-bit VBUSP Configuration target
  - Lockstep Compare block
  - Selftest Logic Block
- 32-bit debug target
  - Provides access to all R5FSS internal debug logic

The 64-bit VBUSM target interface provides 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.

The 64-bit VBUSM target target interface provides 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 target interfaces, there are peripherals (VIM) that are only accessible by the R5F. The R5F has an access to these modules via the VBUSP peripheral interface.

7.1.3.4 R5FSS Power, Clocking and Reset

7.1.3.4.1 R5FSS Power

R5FSS is powered by the SoC Core logic supply.

7.1.3.4.2 R5FSS Clocking

The R5FSS has a single clock input. Internally, CPU0 and CPU1 clocks are generated from this clock with individual clock gate control per Core. The interface clocks are derived from this clock internally through suitable division.

The Interface clock is an integer ratio of the CPU clock. The permitted ratio are 1:1 and 1:2 for CPU_CLK:INTERFACE_CLK. The Interface clock shall not exceed 200MHz.

Refer to R5SS and SYSCLK Clock Tree for more details regarding the sequence for choosing CPU and INTERFACE clocks.

The CPU core clock can be gated by writing 7 to R5SS[0-1]_CORE[0-1]_GATE_CLKGATE. However, the application code must ensure there are no pending transactions/instructions before executing the gating.

7.1.3.4.3 R5FSS Reset

The R5FSS has seven reset inputs:

- POR_RST : This is the reset for the full R5FSS including debug logic
- CORE0_G_RST : This resets the entire CPU0 logic including its associated VIM except the debug logic.
- CORE1_G_RST: This resets the entire CPU1 logic including its associated VIM except the debug logic
- CORE0_L_RST : THis resets CPU0 core
- CORE1_L_RST : THis resets CPU1 core
- VIM0_RST : THis resets VIM0
- VIM1_RST: This resets VIM1

The above resets can be controlled through RCM registers.

In addition to the reset signals, there are two halt signals:

- CORE0_HALT
- CORE1_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. See R5 Core Halting and Unhalting for more details.

7.1.3.5 R5FSS Vectored Interrupt Manager (VIM)

Note

For additional details related to the R5FSS VIM, refer to the Vectored Interrupt Manager (VIM) interrupt controller section of the *Interrupts* chapter.

7.1.3.6 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 for R5 related memories, it only supports a subset of the generic ECC aggregator functionality for R5 memories. However, the ECC aggregator supports full ECC aggregator functionality for VIM memories.

For a detailed description of the generic ECC aggregator functionality, see ECC Aggregator. For register descriptions of R5FSS CPU0 and CPU1 ECC aggregators, see *R5FSS_CPU0_ECC_AGGR_CFG_REGS Registers* and *R5FSS_CPU1_ECC_AGGR_CFG_REGS Registers*, respectively.

Table 7-8 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 (part of the ECC aggregator register space).

Table 7-8. 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
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

7.1.3.7 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.
- TCM locations: TCMs can be enabled or disabled and located at different places in the memory map, depending on bootstrap configuration. In addition, the TCM size varies depending on the mode of CPU being in Dual core or lockstep mode. For more details, see Tightly-Coupled Memories (TCM).
- Peripheral interface locations: Peripherals are accessed at address 0x5000_0000 over the VBUSP peripheral interface.

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.

See Memory Map for the complete R5F memory view for this device.

7.1.3.8 R5FSS Interrupts

All interrupts and events generated by the R5FSS are summarized in R5FSS Integration, along with their mapping. These processor events can be divided into the following groups:

- R5F CPU internal interrupts and events: these include the R5 EVENT signals and PMU interrupts. These are described in the *Arm Cortex-R5 Technical Reference Manual*.
- ECC aggregator interrupts: only VIM memory errors generate the interrupt. These are described in the *ECC Aggregator* chapter.
- TCM Address parity Error Interrupts: these are described in the *TCM Address Parity Error* section.
- Lockstep Compare Interrupts: these are described in the *Lockstep Compare* section.
- Selftest Logic Interrupt: interrupts and errors generated by selftest logic. These are described in the *Selftest Controller (STC)* chapter.

7.1.3.9 R5FSS Debug and Trace

The R5FSS supports standard Arm CoreSight debug and trace architecture. For more details, see the *On-chip Debug* chapter.

7.1.3.10 R5FSS Boot Options

R5FSS boots from 0x0000_0000 located in TCM. When the processor exits reset, it fetches the boot vector from this location.

The software must take the following steps:

1. Assert the correct bootstraps
   a. Enable the ATCM (set CPUn_INITRAMA) or BTCM (set CPUn_INITAMB). Default state is both are enabled and no additional configuration needed in this device
   b. Assert CPUn_LOCZRAMA properly for the desired TCMA. Default state is to boot from ATCM in this device.
2. Assert CPUn_HALT. Default is HALTED state.
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 TCM.
5. De-assert CPUn_HALT.

7.1.3.11 R5FSS Events

The R5F core generates several events as part of event bus. That can be monitored by the PMU for debugging.The R5 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 R5 core. For more details, refer to Arm R5 TRM.

7.1.3.11.1 R5FSS Core Memory ECC Events

The memory ECC-related events from the event bus are aggregated in MSS_CTRL and exported to ESM for monitoring as shown in *R5FSS Integration*.

There are four ECC interrupts to the ESM that aggregate different categories of ECC events:

- CPU0 correctable error or single bit error
- CPU1 Correctable error or single but error
- CPU0 Uncorrectable error or multi bit error
- CPU1 Uncorrectable error or multibit error

Table 7-9. R5 Event Bus Correctable Error or Single-bit Error

EVENT BUS Bit #	Description	Associated Status Register in MSS_CTRL
40	ATCM single-bit ECC error	R5SS*_CPU*_ECC_CORR_ERRAGG_STATUS[0] R5SS*_CPU*_ATCM_CORR_ERR
42	B1TCM single-bit ECC error	R5SS*_CPU*_ECC_CORR_ERRAGG_STATUS[1] R5SS*_CPU*_B1TCM_CORR_ERR

Table 7-9. R5 Event Bus Correctable Error or Single-bit Error (continued)

EVENT BUS Bit #	Description	Associated Status Register in MSS_CTRL
41	B0TCM single-bit ECC error	R5SS*_CPU*_ECC_CORR_ERRAGG_STATUS[2] R5SS*_CPU*_B0TCM_CORR_ERR
24	Data cache tag or dirty RAM parity error or correctable ECC error, from data-side or ACP	R5SS*_CPU*_ECC_CORR_ERRAGG_STATUS[3] R5SS*_CPU*_DTAG_CORR_ERR
25	Data cache data RAM parity error or correctable ECC error	R5SS*_CPU*_ECC_CORR_ERRAGG_STATUS[4] R5SS*_CPU*_DDATA_CORR_ERR
22	Instruction cache tag RAM parity or correctable ECC error	R5SS*_CPU*_ECC_CORR_ERRAGG_STATUS[5] R5SS*_CPU*_ITAG_CORR_ERR
23	Instruction cache data RAM parity or correctable ECC error	R5SS*_CPU*_ECC_CORR_ERRAGG_STATUS[6] R5SS*_CPU*_IDATA_CORR_ERR

Table 7-10. R5 Event Bus Uncorrectable Error or Multi-bit Error

EVENT BUS Bit #	Description	Associated Status Register in MSS_CTRL
37	ATCM multi-bit ECC error	R5SS*_CPU*_ECC_UNCORR_ERRAGG_STATUS[0] R5SS*_CPU*_ATCM_UNCORR_ERR
39	B1TCM multi-bit ECC error	R5SS*_CPU*_ECC_UNCORR_ERRAGG_STATUS[1] R5SS*_CPU*_B1TCM_UNCORR_ERR
38	B0TCM multi-bit ECC error	R5SS*_CPU*_ECC_UNCORR_ERRAGG_STATUS[2] R5SS*_CPU*_B0TCM_UNCORR_ERR
34	Data caches tag/dirty RAM fatal ECC error, from data-side or ACP.	R5SS*_CPU*_ECC_UNCORR_ERRAGG_STATUS[3] R5SS*_CPU*_DTAG_UNCORR_ERR
33	Data cache data RAM fatal ECC error	R5SS*_CPU*_ECC_UNCORR_ERRAGG_STATUS[4] R5SS*_CPU*_DDATA_UNCORR_ERR

7.1.3.12 R5FSS TCM Address Parity Error

R5FSS in this device is configured to generate TCM address and control bus parity. Parity error detection logic in the R5FSS detects if there is any parity error on TCM address bus. These errors are aggregated in the MSS_CTRL module and one interrupt per CPU is exported to ESM.

Table 7-11. R5 TCM Address Parity Error

Description	Associated Status Register in MSS_CTRL
ATCM bus Address parity Error	R5SS*_CPU*_TCM_ADDRPARITY_ERRAGG_STATUS [0] R5SS*_CPU*_ATCM*_PARITY_ERR
B0TCM bus Address parity Error	R5SS*_CPU*_TCM_ADDRPARITY_ERRAGG_STATUS [1] R5SS*_CPU*_B0TCM*_PARITY_ERR
B1TCM bus Address parity Error	R5SS*_CPU*_TCM_ADDRPARITY_ERRAGG_STATUS [2] R5SS*_CPU*_B1TCM*_PARITY_ERR

R5SS*_CPU*_TCM_ADDRPARITY_ERRAGG_STATUS_RAW: Provides raw status of TCM address Parity Error for each CPU Core

R5SS*_CPU*_TCM_ADDRPARITY_ERRAGG_STATUS: Provides masked status of TCM Address Parity Error for each CPU Core

R5SS*_CPU*_TCM_ADDRPARITY_ERRAGG_MASK: Mask register for TCM Address Parity Error

The register R5SS*_TCM_ADDRPARITY_CLR clears Parity Error interrupt.

The registers R5SS*_CORE*_ADDRPARITY_ERR_*TCM provides the Address location where the TCM address error occurred.

The R5SS*_TCM_ADDRPARITY_ERRFORCE register can be used to force error on TCM Address parity Error detection logic for diagnostic purpose.

7.1.3.13 R5FSS Lockstep Compare

This chapter describes the CPU compare module for the ARM® Cortex®-R5F (CCM-R5F). Each R5F subsystem (R5FSS) in the device implements two instances of the Cortex-R5F CPU that are running in lockstep to detect faults that may result in unsafe operating conditions. The CCM-R5F detects faults and signals them to the SOC error signaling module.

7.1.3.13.1 Overview

Safety-critical applications require run-time detection of faults in critical components in the device such as the Central Processing Unit (CPU) and the Vectored Interrupt Controller Module (VIM). For this purpose, the CPU Compare Module for Cortex-R5F (CCM-R5F) compares the core bus outputs of two Cortex-R5F CPUs running in a 1oo1D (one-out-of-one, with diagnostics) lockstep configuration. Each R5FSS also implements two VIM modules in 1oo1D (one-out-of-one, with diagnostic) lockstep configuration. Any difference in the core compare bus outputs of the CPUs or the VIMs is flagged as an error. For diagnostic purposes, the CCM-R5F also incorporates a self-test capability to allow for boot time checking of hardware faults within the CCM-R5F itself.

In addition to comparing the CPU's and VIM's outputs for fault detection during run-time, the CCM-R5F also incorporates one additional run-time diagnostic feature: the Checker-CPU Inactivity Monitor.

The Checker-CPU inactivity monitor monitors the checker CPU's key bus signals to the interconnect. When the two CPUs are in lockstep configuration, several key bus signals from the checker CPU which would have indicated a valid bus transaction to the interconnect on the microcontroller will be monitored. A list of the signals to be monitored is provided in theChecker CPU Signals to Monitor table. These signals from the checker CPU are expected to be inactive. All transactions between the lockstep CPUs and the rest of the system should only go through the main CPU. Any signals which indicate activity will be flagged as an error.

7.1.3.13.1.1 Main Features

The main features of the CCM-R5F are:

- Run-time detection of faults
  - Run-time compare of CPU's outputs
  - Run-time compare of VIM's outputs
  - Run-time inactivity monitor on the checker CPU's bus signals to the interconnect
- self-test capability
- error forcing capability

7.1.3.13.1.2 Block Diagram

Figure 7-6 shows the interconnect diagram of the CCM-R5F with the two Cortex-R5F CPUs and the two VIMs. The core bus outputs of the CPUs are compared in the CCM-R5F. To avoid common mode impacts, the signals of the CPUs to be compared are temporally diverse. The output signals of the primary CPU are delayed 2 cycles while the input signals of checker CPU are delayed 2 cycles. The two cycle delay strategy is also deployed between the two VIM modules. While in lockstep mode, the checker CPU's output signals to the system are clamped to inactive safe values. Key signals which would have indicated a valid bus transaction to the interconnect are monitored by the CCM-R5F. The same approach is used for the key power domains if inactive signals indicate that bus controllers inside these power domains are asserting valid bus transactions.

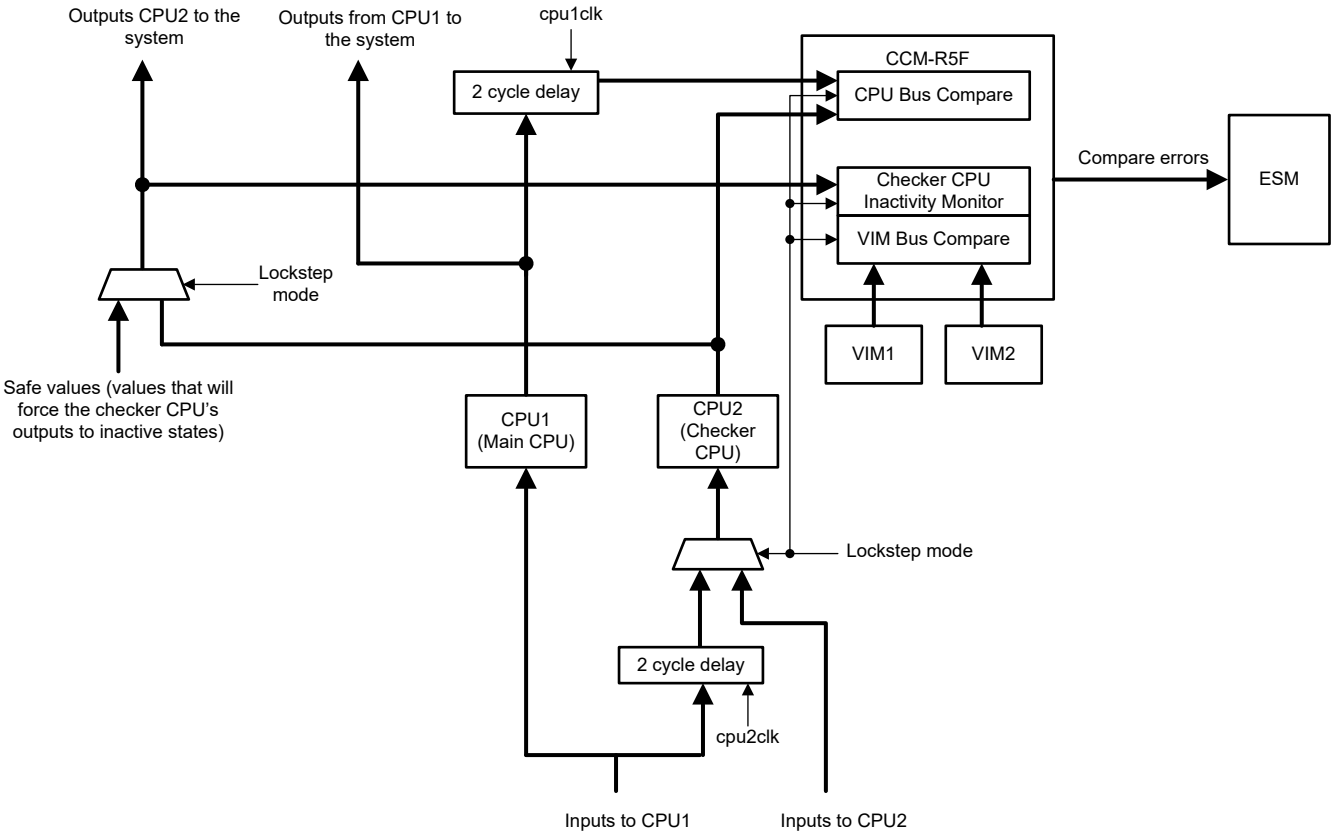


Figure 7-6. Block Diagram

7.1.3.13.2 Module Operation

As described in Overview, there are three different run-time diagnostics supported by the CCM-R5F. The CCM-R5F compares the core bus outputs of the primary and checker Cortex-R5F CPUs on the microcontroller and signals an error on any mismatch. This comparison is started 6 CPU clock cycles after the CPU comes out of reset to ensure that CPU output signals have propagated to a known value after reset. Once comparison is started, the CCM module continues to monitor the outputs of the two CPUs without any software intervention. If an error is detected by the CCM-R5F, a software handler is necessary to implement the appropriate response to the error dependent on application needs. The module principles of operation are applicable to both the CPU output compare as described above as well as to the VIM output compare.

7.1.3.13.2.1 CPU/VIM Output Compare Diagnostic

CPU / VIM Output Compare Diagnostic can run in one of the following four operating modes:

1. Active compare lockstep mode
2. Self-test
3. Error forcing
4. Self-test error forcing

The operating mode can be selected by writing a dedicated key to the key register (MKEYx) of the corresponding diagnostic.

Note

MKEY1 and MKEY2 are used to select the operating mode for the CPU Output Compare Diagnostic and VIM Output Compare Diagnostic, respectively.

7.1.3.13.2.1.1 Active Compare lockstep Mode

This is the default mode on start-up. In lockstep mode, the bus output signals of both CPUs and VIMs are compared. A difference in the CPU compare bus outputs is indicated by signaling an error to the ESM, which sets the error flag "CCM-R5F - CPU compare" and "CCM-R5F - VIM compare", respectively.

- CPU types of output signals to be compared:
  - Global signals
  - Interrupt signals
  - All L1 cache interface signals
  - All cache coherency signals
  - All L1 TCM interface signals
  - All L2 AXI interface signals
  - ETM interface signals
  - FPU signals
  - All AHB Peripheral port interface signals
  - All status and control signals
- VIM output signals to be compared:
  - nFIQ
  - nIRQ
  - IRQADDRV
  - IRQVECTADDR
- CPU types of output signals that are not compared:
  - All ACP interface signals
  - All AXI Peripheral port interface signals

Note

The CPU compare error asserts “CCM-R5F self-test error” flag as well. By doing this, the CPU compare error has two paths (“CCM-R5F - CPU compare” and “CCM-R5F self-test error” flag) to the ESM, so that even if one of the paths fails, the error is still propagated to the ESM. This is also true for "CCM-R5F - VIM compare" error flag.

Not all internal registers of the Cortex-R5F CPU have fixed values upon reset. To avoid an erroneous CCM-R5F 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, including function calls where the register values are pushed onto the stack.

7.1.3.13.2.1.2 Self-Test Mode

In self-test mode, the CCM-R5F checks itself for faults. During self-test, the compare error module output signal is deactivated. Any fault detected inside the CCM-R5F will be flagged by ESM error “CCM-R5F - self-test”.

In self-test mode, the CCM-R5F automatically generates test patterns to look for any hardware faults. If a fault is detected, then a self-test error flag is set, a self-test error signal is asserted and sent to the ESM, and the self-test is terminated immediately. If no fault is found during self-test, the self-test complete flag is set. In both cases, the CCM-R5F CPU / VIM Output Compare Diagnostic remains in self-test mode after the test has been terminated or completed, and the application needs to switch the CCM-R5F mode by writing another key to the mode key register (MKEY1 or MKEY2 depending which diagnostic is selected for self-test). During the self-test operation, the compare error signal output to the ESM is inactive irrespective of the compare result.

There are two types of patterns generated by CCM-R5F during self-test mode:

1. Compare Match Test
2. Compare Mismatch Test

CCM-R5F first generates Compare Match Test patterns, followed by Compare Mismatch Test patterns. Each test pattern is applied on both CPU signal inputs of the CCM-R5F's compare block and clocked for one cycle. The duration of self-test for CPU Output Compare Diagnostic is 4947 CPU clock cycles (GCLK1) and 151 system peripheral clock cycles (VCLK) for VIM Output Compare Diagnostic.

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 and not the memory-mapped register controls for the CCM-R5F. The self-test is not interruptible.

Self-test of all different diagnostics can be run at the same time.

7.1.3.13.2.1.2.1 Compare Match Test

During the Compare Match Test, there are four different test patterns generated to stimulate the CCM-R5F. An identical vector is applied to both input ports at the same time expecting a compare match. These patterns cause the self-test logic to exercise every CPU compare bus output signal in parallel. If the compare unit produces a compare mismatch then the self-test error flag is set, the self-test error signal is generated, and the Compare Match Test is terminated.

The four test patterns used for the Compare Match Test are:

- All 1s on both CPU / VIM signal ports
- All 0s on both CPU / VIM signal ports
- 0xAs on both CPU / VIM signal ports
- 0x5s on both CPU / VIM signal ports

These four test patterns will take four clock cycles to complete. Table 7-12 illustrates the sequence of Compare Match Test.

Table 7-12. Compare Match Test Sequence

CPU 1 (Main CPU) Signal Position									CPU 2 (Checker CPU) Signal Position									Cycle
n:8	7	6	5	4	3	2	1	0	n:8	7	6	5	4	3	2	1	0	
1s	1	1	1	1	1	1	1	1	1s	1	1	1	1	1	1	1	1	0
0s	0	0	0	0	0	0	0	0	0s	0	0	0	0	0	0	0	0	1
0xA	1	0	1	0	1	0	1	0	0xA	1	0	1	0	1	0	1	0	2
0x5	0	1	0	1	0	1	0	1	0x5	0	1	0	1	0	1	0	1	3

7.1.3.13.2.1.2.2 Compare Mismatch Test

During the Compare Mismatch Test, the number of test patterns is equal to twice the number of CPU output signals to compare in lockstep mode. An all 1s vector is applied to the CCM-R5F's CPU1 / VIM1 input port and the same pattern is also applied to the CCM-R5F's CPU2 /VIM2 input port but with one bit flipped starting from signal position 0. The un-equal vector will cause the CCM-R5F to expect a compare mismatch at signal position 0, if the CCM-R5F logic is working correctly. If, however, the CCM-R5F logic reports a compare match, the self-test error flag is set, the self-test error signal is asserted, and the Compare Mismatch Test is terminated.

This Compare Mismatch Test algorithm repeats in a domino fashion with the next signal position flipped while forcing all other signals to logic level 1. This sequence is repeated until every single signal position is verified on both CPU signal ports.

The Compare Mismatch Test is terminated if the CCM-R5F reports a compare match versus the expected compare mismatch. This test ensures that the compare unit is able to detect a mismatch on every CPU signal being compared. Table 7-13 illustrates the sequence of Compare Mismatch Test. There is no error signal sent to ESM if the expected errors are seen with each pattern.

Table 7-13. CPU / VIM Compare Mismatch Test Sequence

CPU 1 (Main CPU) Signal Position											CPU 2 (Checker CPU) Signal Position										Cycle
n	n–1:8		7	6	5	4	3	2	1	0	n	n–1:8		7	6	5	4	3	2	1	0	
1	1	1s	1	1	1	1	1	1	1	1	1	1	1s	1	1	1	1	1	1	1	0	0
1	1	1s	1	1	1	1	1	1	1	1	1	1	1s	1	1	1	1	1	1	0	1	1
1	1	1s	1	1	1	1	1	1	1	1	1	1	1s	1	1	1	1	1	0	1	1	2
1	1	1s	1	1	1	1	1	1	1	1	1	1	1s	1	1	1	1	0	1	1	1	3
::																						
1	1	1s	1	1	1	1	1	1	1	1	1	0	1s	1	1	1	1	1	1	1	1	–1
1	1	1s	1	1	1	1	1	1	1	1	0	1	1s	1	1	1	1	1	1	1	1	n
1	1	1s	1	1	1	1	1	1	1	0	1	1	1s	1	1	1	1	1	1	1	1	n+1
1	1	1s	1	1	1	1	1	1	0	1	1	1	1s	1	1	1	1	1	1	1	1	n+2
1	1	1s	1	1	1	1	1	0	1	1	1	1	1s	1	1	1	1	1	1	1	1	n+3
1	1	1s	1	1	1	1	0	1	1	1	1	1	1s	1	1	1	1	1	1	1	1	n+4
::																						
1	0	1s	1	1	1	1	1	1	1	1	1	1	1s	1	1	1	1	1	1	1	1	2n-1
0	1	1s	1	1	1	1	1	1	1	1	1	1	1s	1	1	1	1	1	1	1	1	2n

7.1.3.13.2.1.3 Error Forcing Mode

In error forcing mode, a test pattern is applied to the CPU / VIM related inputs of the CCM-R5F compare logic to force an error in the compare error output signal of the compare unit. Depending if error forcing mode is applied to the CPU Output Compare Diagnostic or VIM Output Compare Diagnostic, the ESM error flag “CCM-R5F - CPU compare” or “CCM-R5F - VIM compare” is expected after the error forcing mode completes. As a side effect, the “CCM-R5F self-test error” flag is also asserted whenever the CPU compare error is asserted.

Error forcing mode is similar to the Compare Mismatch Test operation of self-test mode in which an un-equal vector is applied to the CCM-R5F CPU signal ports. The error forcing mode forces the compare mismatch to actually assert the compare error output signal. This ensures that a fault in the path between CCM-R5F and ESM is detected.

Only one hardcoded test pattern is applied into CCM-R5F during error forcing mode. A repeated 0x5 pattern is applied to CPU1 / VIM1 signal port of CCM-R5F input while a repeated 0xA pattern is applied to the CPU2 / VIM2 signal port of CCM-R5F input. The error forcing mode takes one cycle to complete. Hence, the failing signature is presented for one clock cycle. After that, the mode is automatically switched to lockstep mode. The key register (MKEY1 for CPU output compare and MKEY2 for VIM output compare) will indicate the lockstep key mode once it is switched to lockstep mode. During the one cycle required by the error forcing test, the CPU / VIM output signals are not compared. The user should expect the ESM to trigger a response (report the CCM-R5F fail). If no error is detected by the ESM, then a hardware fault is present.

7.1.3.13.2.1.4 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 ESM error flag “CCM-R5F - self-test” is expected after the self-test error forcing mode completes. Once the expected errors are seen, the application can clean the error through the ESM module.

Table 7-14 shows what error signals and flags are asserted in different operating mode. The behavior of different modes in this table for CPU compare is also valid for other diagnostics such as VIM compare and Checker CPU Inactivity Monitor.

Table 7-14. Error Flags and Error Signals Generation in Each Mode

Mode	Key	Self Test Error Signal	Compare Error Signal	CMPE	STC	STET	STE
Active Compare Lockstep	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

7.1.3.13.2.2 CPU Input Inversion Diagnostic

There is another way to intentionally create a mismatch between the two CPUs' outputs as a diagnostic test to self-test the CCM-R5F's CPU Output Compare Diagnostic block. Before the CPU1's outputs are taken to the CCM-R5F, eight of the output signals are first exclusive-ORed bitwise with the 8-bit POLARITYINVERT register. After reset, the default value of the POLARITYINVERT register is all zeros. The resultant values of the 8 signals after the XOR logic with the POLARITYINVERT register will still be the same as the original 8 signal values. However, by programming the POLARITYINVERT to a non-zero values it will have the effect to invert the signal values. This intentional inversion on the inputs to the CCM-R5F will cause the CPU Output Compare Diagnostic to detect a compare error. See Figure 7-7 for illustration.

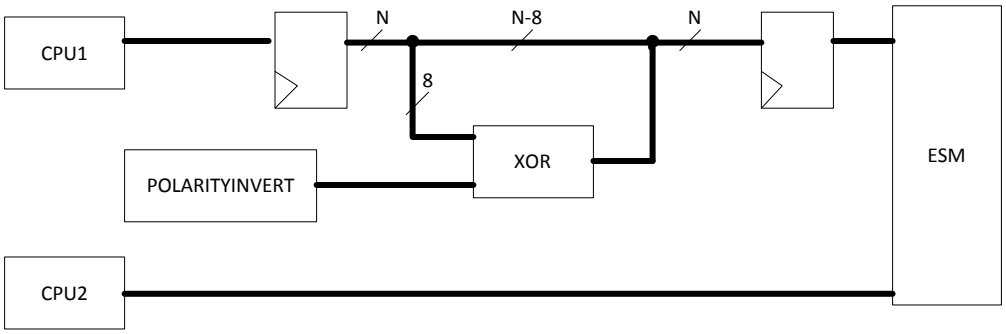


Figure 7-7. CPU Input Inversion Scheme

Table 7-15. CPU1 (Main CPU) Signals Being Inverted Before Being Compared

Signals	Remark
AWVALIDM	Indicates write address and control are valid
ARVALIDM	Indicates write address and control are valid
AWVALIDP	Indicates write address and control are valid
ARVALIDP	Indicates write address and control are valid
HTRANSP[1:0]	Indicates write address and control are valid

7.1.3.13.2.3 Checker CPU Inactivity Monitor

Similar to the CPU / VIM Output Compare Diagnostic, the Checker CPU Inactivity Monitor can also run in one of the following four operating modes:

1. Active compare
2. Self-test
3. Error forcing
4. Self-test error forcing

The operating mode can be selected by writing a dedicated key to the key register (MKEY3).

7.1.3.13.2.3.1 Active Compare Mode

This is the default mode on start-up. In this mode, several key bus signals such as the bus valid control signals from the checker CPU that would have indicated a valid bus transaction onto the interconnect are compared against their clamped safe values. While the two CPUs are in lockstep configuration, the outputs of the checker CPU are supposed to clamp to the inactive state that is all zeros. A difference between the checker CPU compare bus outputs and their respective inactive states is indicated by signaling an error to the ESM which sets the error flag "CCM-R5F - CPU1 AXIM Bus Monitor Failure".

Table 7-16. Checker CPU Signals to Monitor

Signals	Remark
AWVALIDM	When asserted, indicates address and control are valid on the Checker CPU's AXI controller port for write transaction.
ARVALIDM	When asserted, indicates address and control are valid on the Checker CPU's AXI controller port for read transaction.
AWVALIDP	When asserted, indicates address and control are valid on the Checker CPU's AXI peripheral port for write transaction.
ARVALIDP	When asserted, indicates address and control are valid on the Checker CPU's AXI peripheral port for read transaction.
BVALIDS	When asserted, indicates that a valid write response is available on the Checker CPU's AXI slave port for write transaction
RVALIDS	When asserted, indicates address and control are valid on the Checker CPU's AXI slave port for read transaction

7.1.3.13.2.3.2 Self-Test Mode

Similar to the other self-test described for CPU / VIM Output Compare Diagnostic, the Checker CPU Inactivity Monitor can be placed in self-test mode. In self-test mode, the CCM-R5F checks the Checker CPU Inactivity Monitor itself for faults. During self-test, the compare error module output signal is deactivated. Any fault detected inside the CCM-R5F will be flagged by ESM error ESM_PLS_EVENT_8 (R5FSS0_cpu_miscompare) or ESM_PLS_EVENT_12 (R5FSS1_cpu_miscompare). If a CPU Inactivity Monitor error is asserted while self-test mode is running, the self-test error for that CPU will also be asserted.

In self-test mode, the CCM-R5F automatically generates test patterns to look for any hardware faults. If a fault is detected, then a self-test error flag is set, a self-test error signal is asserted and sent to the ESM, and the self-test is terminated immediately. If no fault is found during self-test, the self-test complete flag is set. In both cases, the CCM-R5F Checker CPU Inactivity Monitor Diagnostic remains in self-test mode after the test has been terminated or completed, and the application needs to switch the CCM-R5F mode by writing another key to the mode key register (MKEY3). During the self-test operation, the compare error signal output to the ESM is inactive irrespective of the compare result.

There are also two types of patterns generated by CCM-R5F during self-test mode for Check CPU Inactivity Monitor. The difference here is the number of test patterns applied during self-test.

1. Compare Match Test
2. Compare Mismatch Test

CCM-R5F first generates Compare Match Test patterns, followed by Compare Mismatch Test patterns.

7.1.3.13.2.3.2.1 Compare Match Test

Since the comparison is done against the clamped values, and all compared signals are clamped to zero, only one test pattern is applied for the compare match test. A pattern of all-zeros are applied for the compare match test. The test will take one cycle. If the compare unit produces a compare mismatch then the self-test error flag is set, the self-test error signal is generated, and the Compare Match Test is terminated.

7.1.3.13.2.3.2.2 Compare Mismatch Test

During the Compare Mismatch Test, the number of test patterns is equal to the number of bus signals on the checker CPU to be monitored. There are a total of 6 signals being monitored on the checker CPU's level 2 interface and hence it takes 6 test patterns for the mismatch test. The mismatch test will take a total of 6 cycles to complete. An all 0's test vector is applied to the CCM-R5F's but with one bit flipped starting from signal position 0. The un-equal vector will cause the CCM-R5F to expect a compare mismatch at signal position 0, if the CCM-R5F logic is working correctly. If, however, the CCM-R5F logic reports a compare match, the self-test error flag is set, the self-test error signal is asserted, and the Compare Mismatch Test is terminated.

This Compare Mismatch Test algorithm repeats in a domino fashion with the next signal position flipped while forcing all other signals to logic level 0. This sequence is repeated until every inactivity monitor signal position is verified on the checker CPU .

Table 7-17 shows the sequence of Compare Mismatch Test. There is no error signal sent to ESM if the expected errors are seen with each pattern.

Table 7-17. Checker CPU Inactivity Monitor Compare Mismatch Test

Signal Position						
5	**4**	**3**	**2**	**1**	**0**	**Cycle**
0	0	0	0	0	1	0
0	0	0	0	1	0	1
0	0	0	1	0	0	2
0	0	1	0	0	0	3
0	1	0	0	0	0	4
1	0	0	0	0	0	5

7.1.3.13.2.3.3 Error Forcing Mode

In error forcing mode, a test pattern of all 1's is applied to the check CPU's compare logic to force an error in the compare error output signal of the compare unit. The ESM error flag "CCM-R5F - CPU1 AXIM Bus Inactivity failure" is expected after the error forcing mode completes. As a side effect, the "CCM-R5F self-test error" flag is also asserted whenever the CPU compare error is asserted.

The error forcing mode takes one cycle to complete. Hence, the failing signature is presented for one clock cycle. After that, the mode is automatically switched to active compare mode. The key register (MKEY3) will indicate the active compare mode once it is switched to active compare mode. During the one cycle required by the error forcing test, the checker CPU Inactivity Monitor is deactivated. User should expect the ESM to trigger a response (report the CCM-R5F fail). If no error is detected by ESM, then a hardware fault is present.

7.1.3.13.2.3.4 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 active compare mode and the key is switched to active compare after one clock cycle. The ESM error flag "CCM-R5F - self-test" is expected after the self-test error forcing mode completes. Once the expected errors are seen, the application can clean the error through the ESM module.

7.1.3.13.2.4 Operation During CPU Debug Mode

Certain debug operations place the CPU in a halting debug state where the code execution is halted. Because halting debug events are asynchronous, there is a possibility for the debug requests to cause loss of lockstep. CCM-R5F will disable all functional diagnostics upon detection of halting debug requests. Core compare error

will not be generated and flags will not update. A CPU reset is needed to ensure the CPUs are again in lockstep and will also re-enable the CCM-R5F.

7.1.3.13.3 Control Registers

Table 7-18 lists the CCM-R5F registers. Each register begins on a 32-bit word boundary. The registers support 32-bit, 16-bit, and 8-bit accesses. The base address for the control registers is FFFF F600h.

Table 7-18. Control Registers

Offset	Acronym	Register Description	Section
00h	CCMSR1	CCM-R5F Status Register 1	Section 7.1.3.13.3.1
04h	CCMKEYR1	CCM-R5F Key Register 1	Section 7.1.3.13.3.2
08h	CCMSR2	CCM-R5F Status Register 2	Section 7.1.3.13.3.3
0Ch	CCMKEYR2	CCM-R5F Key Register 2	Section 7.1.3.13.3.4
10h	CCMSR3	CCM-R5F Status Register 3	Section 7.1.3.13.3.5
14h	CCMKEYR3	CCM-R5F Key Register 3	Section 7.1.3.13.3.6
18h	CCMPOLCNTRL	Polarity Control Register	Section 7.1.3.13.3.7

7.1.3.13.3.1 CCM-R5F Status Register 1 (CCMSR1)

The contents of this register should be interpreted in context of what test was selected. That is, what mode is CCM operating.

Figure 7-8. CCM-R5F Status Register 1 (CCMSR1) (Offset = 00h)

31	17	16
Reserved		CPME1
R-0		R/W1CP-0

15	9	8	7	2	1	0
Reserved		STC1	Reserved		STET1	STE1
R-0		R-0	R-0		R-0	R-0

Table 7-19. CCM-R5F Status Register 1 (CCMSR1) Field Descriptions

Bit	Field	Value	Description
31-17	Reserved	0	Reads return 0. Writes have no effect.
16	CMPE1		Compare Error for CPU Output Compare Diagnostic. **Read in User and Privileged mode. Write in Privileged mode only.**
		0	Read: CPU signals are identical. Write: Leaves the bit unchanged.
		1	Read: CPU signal compare mismatch. Write: Clears the bit.
15-9	Reserved		Reads return 0. Writes have no effect.

Table 7-19. CCM-R5F Status Register 1 (CCMSR1) Field Descriptions (continued)

Bit	Field	Value	Description
8	STC1		Self-test Complete for CPU Output Compare Diagnostic. **Note:** This bit is always 0 when not in self-test mode. Once set, switching from self-test mode to other modes will clear this bit. **Read/Write in User and Privileged mode.**
		0	Read: Self-test on-going if self-test mode is entered. Write: Writes have no effect.
		1	Read: Self-test is complete. Write: Writes have no effect.
7-2	Reserved		Reads return 0. Writes have no effect.
1	STET1		Self-test Error Type for CPU Output Compare Diagnostic. **Read/Write in User and Privileged mode.**
		0	Read: Self-test failed during Compare Match Test if STE1 = 1. Write: Writes have no effect.
		1	Read: Self-test failed during Compare Mismatch Test if STE1 = 1. Write: Writes have no effect.
0	STE1		Self-test Error for CPU Output Compare Diagnostic. **Note:** This bit gets updated when the self-test is complete or an error is detected. **Read/Write in User and Privileged mode.**
		0	Read: Self-test passed. Write: Writes have no effect.
		1	Read: Self-test failed. Write: Writes have no effect.

7.1.3.13.3.2 CCM-R5F Key Register 1 (CCMKEYR1)

Figure 7-9. CCM-R5F Key Register 1 (CCMKEYR1) (Offset = 04h)

31	16
Reserved	
R-0	

15	4	3	0
Reserved		MKEY1	
R-0		R/WP-0	

Table 7-20. CCM-R5F Key Register 1 (CCMKEYR1) Field Descriptions

Bit	Field	Value	Description
31-4	Reserved	0	Reads return 0. Writes have no effect.
3-0	MKEY1	0	Mode Key to select operation for CPU Output Compare Diagnostic . **Read in User and Privileged mode. Write in Privileged mode only.** Read: Returns current value of the MKEY1. Write: Active Compare Lockstep mode.
		6h	Read: Returns current value of the MKEY1. Write: Self-test mode.
		9h	Read: Returns current value of the MKEY1. Write: Error Forcing mode.
		Fh	Read: Returns current value of the MKEY1. Write: Self-test Error Forcing mode.
		Other values	**Note:** It is recommended to not write any other key combinations. Invalid keys will result in switching operation to lockstep mode.

7.1.3.13.3.3 CCM-R5F Status Register 2 (CCMSR2)

Figure 7-10. CCM-R5F Status Register 2 (CCMSR2) (Offset = 08h)

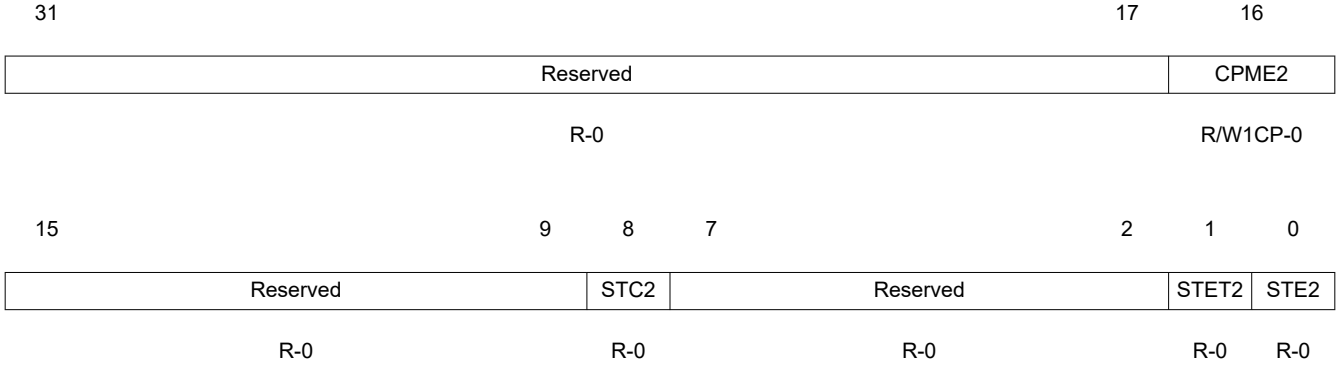


Table 7-21. CCM-R5F Status Register 2 (CCMSR2) Field Descriptions

Bit	Field	Value	Description
31-17	Reserved	0	Reads return 0. Writes have no effect.
16	CMPE2		Compare Error for VIM Output Compare Diagnostic. **Read in User and Privileged mode. Write in Privileged mode only.**
		0	Read: CPU signals are identical. Write: Leaves the bit unchanged.
		1	Read: CPU signal compare mismatch. Write: Clears the bit.
15-9	Reserved		Reads return 0. Writes have no effect.
8	STC2		Self-test Complete for VIM Output Compare Diagnostic. **Note:** This bit is always 0 when not in self-test mode. Once set, switching from self-test mode to other modes will clear this bit. **Read/Write in User and Privileged mode.**
		0	Read: Self-test on-going if self-test mode is entered. Write: Writes have no effect.
		1	Read: Self-test is complete. Write: Writes have no effect.
7-2	Reserved		Reads return 0. Writes have no effect.
1	STET2		Self-test Error Type for VIM Output Compare Diagnostic. **Read/Write in User and Privileged mode.**
		0	Read: Self-test failed during Compare Match Test if STE2 = 1. Write: Writes have no effect.
		1	Read: Self-test failed during Compare Mismatch Test if STE2 = 1. Write: Writes have no effect.
0	STE2		Self-test Error for VIM Output Compare Diagnostic. **Note:** This bit gets updated when the self-test is complete or an error is detected. **Read/Write in User and Privileged mode.**
		0	Read: Self-test passed. Write: Writes have no effect.
		1	Read: Self-test failed. Write: Writes have no effect.

7.1.3.13.3.4 CCM-R5F Key Register 2 (CCMKEYR2)

Figure 7-11. CCM-R5F Key Register 2 (CCMKEYR2) (Offset = 0Ch)

31	16
Reserved	
R-0	

15	4	3	0
Reserved		MKEY2	
R-0		R/WP-0	

Table 7-22. CCM-R5F Key Register 2 (CCMKEYR2) Field Descriptions

Bit	Field	Value	Description
31-4	Reserved	0	Reads return 0. Writes have no effect.
3-0	MKEY2	0	Mode Key to select operation for VIM Output Compare Diagnostic. **Read in User and Privileged mode. Write in Privileged mode only.** Read: Returns current value of the MKEY2. Write: Active Compare Lockstep mode.
		6h	Read: Returns current value of the MKEY2. Write: Self-test mode.
		9h	Read: Returns current value of the MKEY2. Write: Error Forcing mode.
		Fh	Read: Returns current value of the MKEY2. Write: Self-test Error Forcing mode.
		Other values	**Note:** It is recommended to not write any other key combinations. Invalid keys will result in switching operation to lockstep mode.

7.1.3.13.3.5 CCM-R5F Status Register 3 (CCMSR3)

Figure 7-12. CCM-R5F Status Register 3 (CCMSR3) (Offset = 10h)

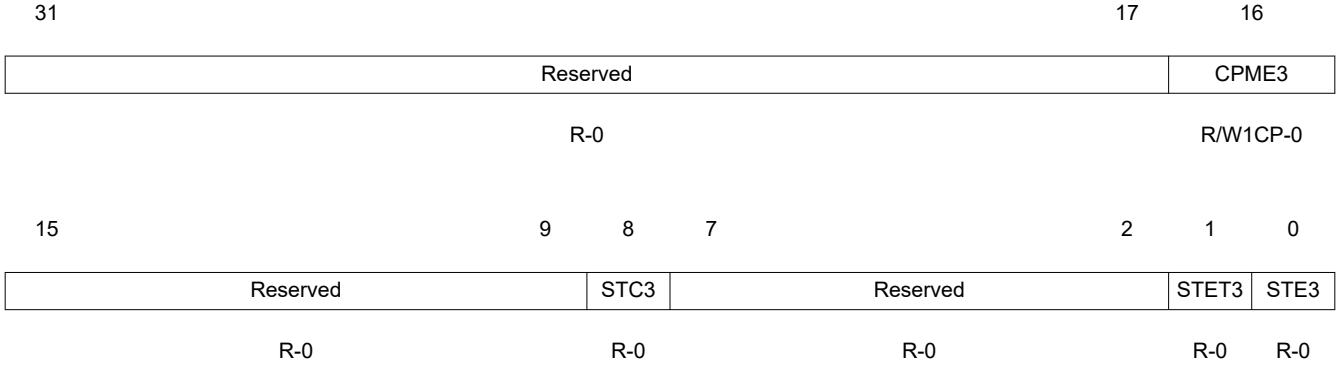


Table 7-23. CCM-R5F Status Register 3 (CCMSR3) Field Descriptions

Bit	Field	Value	Description
31-17	Reserved	0	Reads return 0. Writes have no effect.
16	CMPE3		Compare Error for Checker CPU Inactivity Monitor. **Read in User and Privileged mode. Write in Privileged mode only.**
		0	Read: CPU signals are identical. Write: Leaves the bit unchanged.
		1	Read: CPU signal compare mismatch. Write: Clears the bit.
15-9	Reserved		Reads return 0. Writes have no effect.
8	STC3		Self-test Complete for Checker CPU Inactivity Monitor. **Note:** This bit is always 0 when not in self-test mode. Once set, switching from self-test mode to other modes will clear this bit. **Read/Write in User and Privileged mode.**
		0	Read: Self-test on-going if self-test mode is entered. Write: Writes have no effect.
		1	Read: Self-test is complete. Write: Writes have no effect.
7-2	Reserved		Reads return 0. Writes have no effect.
1	STET3		Self-test Error Type for Checker CPU Inactivity Monitor. **Read/Write in User and Privileged mode.**
		0	Read: Self-test failed during Compare Match Test if STE3 = 1. Write: Writes have no effect.
		1	Read: Self-test failed during Compare Mismatch Test if STE3 = 1. Write: Writes have no effect.
0	STE3		Self-test Error for Checker CPU Inactivity Monitor. **Note:** This bit gets updated when the self-test is complete or an error is detected. **Read/Write in User and Privileged mode.**
		0	Read: Self-test passed. Write: Writes have no effect.
		1	Read: Self-test failed. Write: Writes have no effect.

7.1.3.13.3.6 CCM-R5F Key Register 3 (CCMKEYR3)

Figure 7-13. CCM-R5F Key Register 3 (CCMKEYR3) (Offset = 14h)

31	16
Reserved	
R-0	

15	4	3	0
Reserved		MKEY3	
R-0		R/WP-0	

Table 7-24. CCM-R5F Key Register 2 (CCMKEYR2) Field Descriptions

Bit	Field	Value	Description
31-4	Reserved	0	Reads return 0. Writes have no effect.
3-0	MKEY3	0	Mode Key to select operation for Checker CPU Inactivity Monitor. **Read in User and Privileged mode. Write in Privileged mode only.** Read: Returns current value of the MKEY3. Write: Active Compare Lockstep mode.
		6h	Read: Returns current value of the MKEY3. Write: Self-test mode.
		9h	Read: Returns current value of the MKEY3. Write: Error Forcing mode.
		Fh	Read: Returns current value of the MKEY3. Write: Self-test Error Forcing mode.
		Other values	**Note:** It is recommended to not write any other key combinations. Invalid keys will result in switching operation to lockstep mode.

7.1.3.13.3.7 CCM-R5F Polarity Control Register (CCMPOLCNTRL)

Figure 7-14. CCM-R5F Polarity Control Register (CCMPOLCNTRL) (Offset = 18h)

31	16
Reserved	
R-0	

15	8	7	0
Reserved		POLARITYINVERT	
R-0		R/WP-0	

Table 7-25. CCM-R5F Polarity Control Register (CCMPOLCNTRL) Field Descriptions

Bit	Field	Value	Description
31-8	Reserved	0	Reads return 0. Writes have no effect.

Table 7-25. CCM-R5F Polarity Control Register (CCMPOLCNTRL) Field Descriptions (continued)

Bit	Field	Value	Description
3-0	POLARITYINVERT		Polarity Inversion. This value is used to invert one of the 8 output compare signals from the CPU1 to the CCM-R5F. Inverting any one signal will lead to compare error by the CPU Output Compare Diagnostic. **Read in User and Privileged mode. Write in Privileged mode only.**

7.1.3.14 R5FSS Selftest Logic

Additional details regarding the R5FSS Selftest Logic are described in the Self-Test Controller (STC) chapter.

7.2 Trigonometric Math Unit (TMU)

7.2.1 TMU INTRODUCTION

This device integrates TMU(Trignometic Math Unit) module for accelerating many commonly used math functions in control applications such as SIN, COS, ATAN. The TMU module is connected to the TCMA interface of the R5 CPU as shown in Figure 1 . The TMU registers show up as TCM memory mapped registers for R5CPU at address 0x0006 0000.

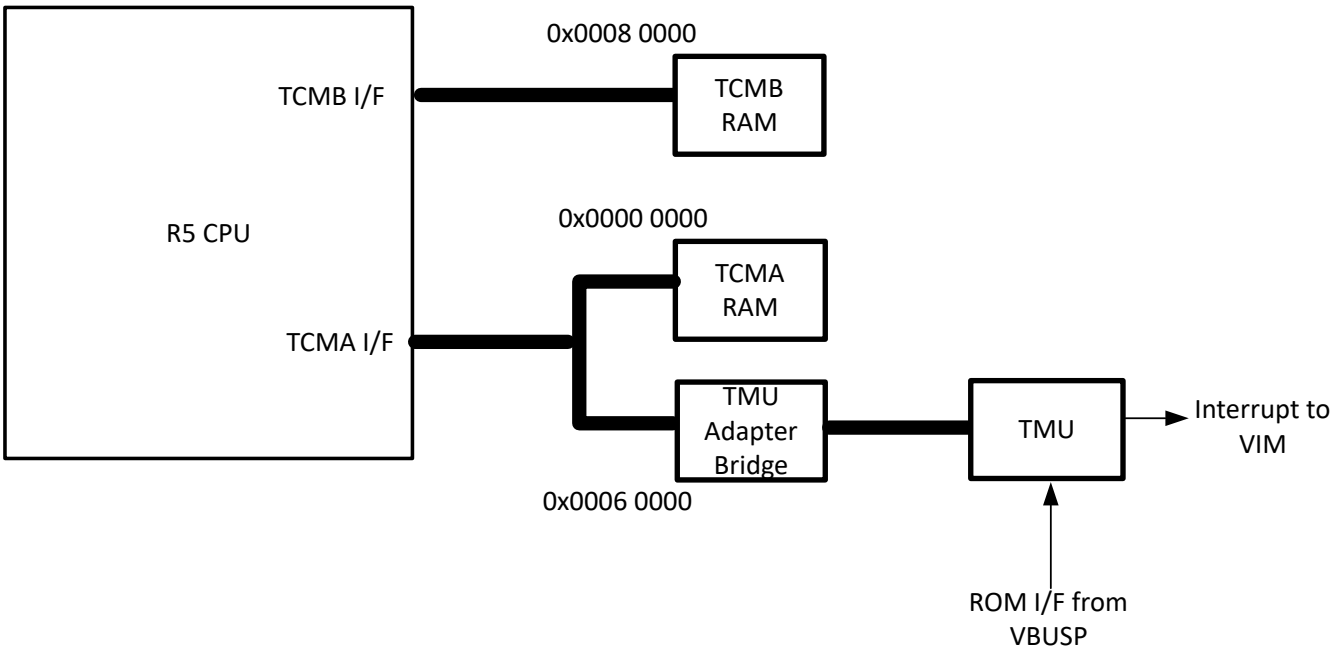


Figure 7-15. TMU Connection to R5 CPU

7.2.1.1 TMU Supported Features

TMU has the following features:

- Supports 8 critical trigonometric operations that are useful in a control loop algorithm
  - SIN
  - COS
  - ATAN
  - QUAD
  - IEXP (base of 2)
  - LOG (base of 2)
- Supports 8 result registers for taking advantage of the pipelined architecture of TMU
- TCM Adaptation logic for seamless integration of TMU to the R5 TCM port in a software transparent manner
- Single cycle Context save and restore operation
  - **Note**

    The CLA TMU integrated in AM26x platforms has limitations with single cycle context save. AM26x hardware does **not** support nesting interrupts, and therefore full TMU context save/restore is not feasible in the MCU PLUS SDK. **Only** single context save/restore is possible on AM26x devices. The C28x TMU hardware supports nested interrupts, and therefore supports full TMU context save/restore.
- Supports Underflow and Overflow interrupts to show mathematical operation errors

7.2.2 TMU Functional Operation

7.2.2.1 Supported Functions

Mathematical operations supported by the TMU accelerator is given in the below table.

Table 7-26. TMU Operations

Operation	Description	Number of Inputs	Number of Outputs	Equivalent Operation	Effective R5 Cycles for Operation(1)
SINPUF32	Returns the SINE of Input value	1 : OP1	1	Sin(OP1*2pi rad) OP1: -1.0 to 1.0	
COSPUF32	Returns the COSINE of Input value	1 : OP1	1	Cos(OP1*2pi rad) OP1: -1.0 to 1.0	
ATANPUF32	Returns the ATAN of Input value	1 : OP1	1	Atan(OP1) rad/2pi OP1: -1.0 to 1.0 Result: -0.125 to 0.125	
QUADF32	Returns the quadrant value and the ratio of X and Y inputs which are provided as per unit values.	2 OP1 : X OP2 : Y	2	Operation to assist in calculating Atan Result1 : Ratio of X & Y Result 2: Quadrant (X,Y) See Figure 2	
IEXP2F32	Returns inverse exponent of input value	1 : OP1	1	1/(2^OP1)	
LOG2F32	Returns base-2 logarithm of input value	1: OP1	1	Log2(OP1)	

(1) The cycles indicated in the table is theoretical best case for single operation with a barrier instruction inserted bw operand write and result readout. The actual cycles may vary based on the ARM CPU Load/Store state.

For pipelined operation, the effective cycle count for multiple operations will be significantly less than sum of cycle count of individual operation.

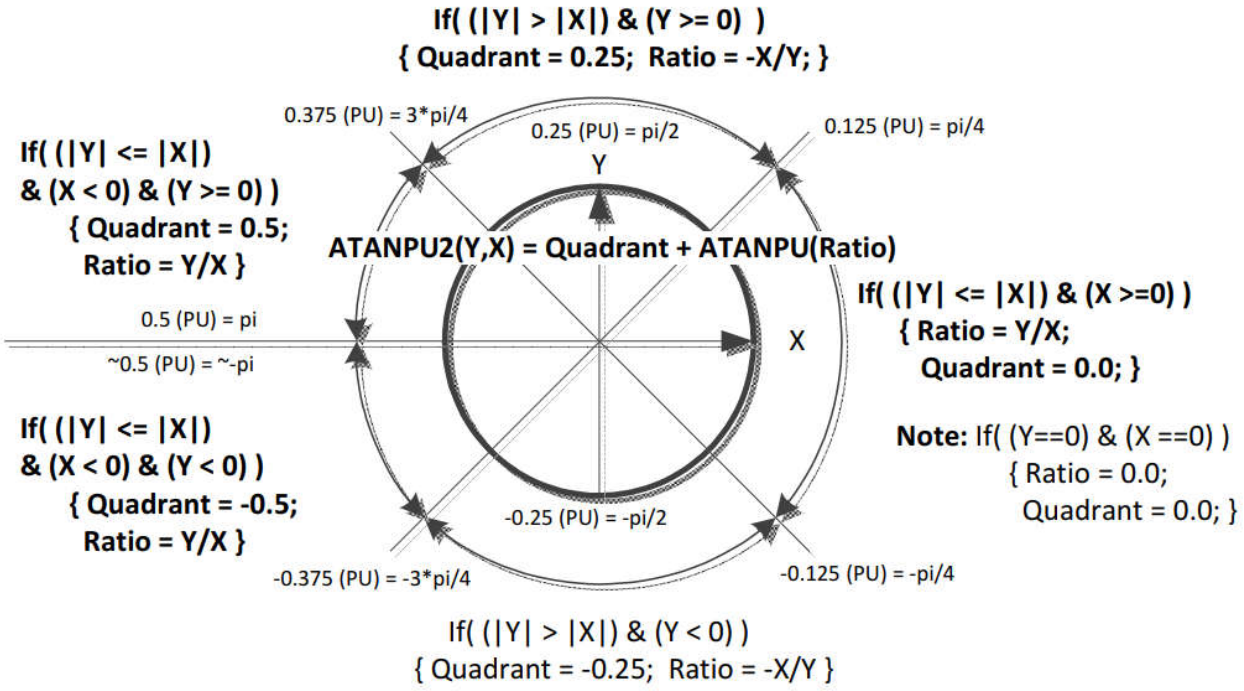


Figure 7-16. QUADF32 Operation

7.2.2.2 TMU Module Block diagram

Figure 2 Shows the TMU Block diagram.

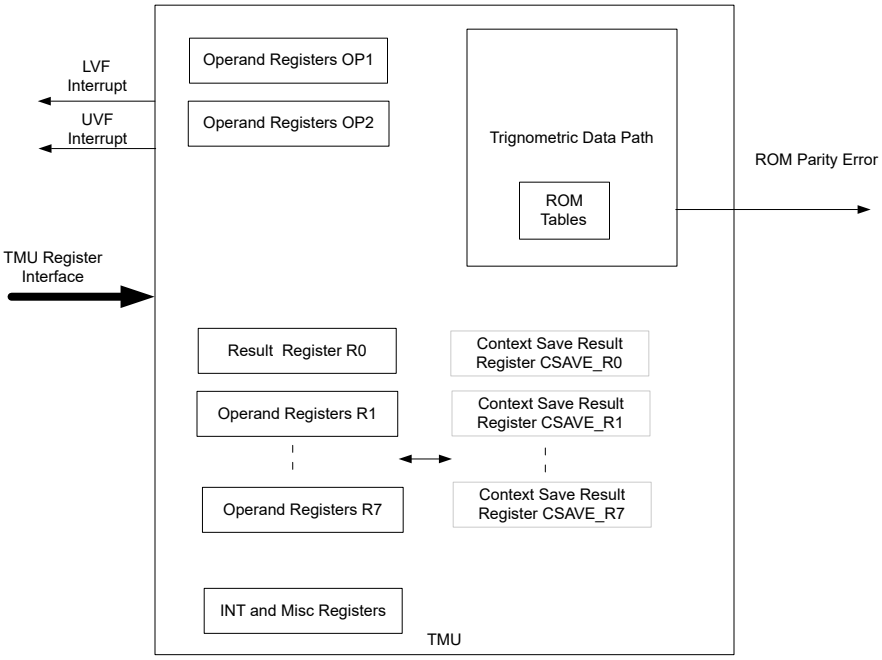


Figure 7-17. TMU Block Diagram

7.2.2.3 Operand Registers (OP1 and OP2)

There are two operand registers named OP1 and OP2. The input operand value to math operations will be written into these registers. One or two operand registers are used depending on type of operation. Trigonometric operation is triggered upon writing to OP1 register (See Table 1). In case an operation requires two operands then OP2 should be updated first then OP1 to trigger operation. OP1 register need not maintain its input value for more than a cycle due to pipeline implementation of TMU operations. It can also be overwritten in consecutive cycles with a new operand value for same or different operation

7.2.2.4 Result Registers

TMU operations can take 8 to 10 Effective R5 CPU cycles to return the result after OP1 is written There are 8 result registers in TMU. Result is updated into one of the result register when operation completes. These multiple result registers serve as temporary storage to enable back to back operations in consecutive cycles and keep the results separate. There is one operations which will update two result registers.

7.2.2.4.1 Operand and Result Register Structure

The Operand Register1 (OP1) is a single physical register aliased at multiple address locations. The choice of the alias address determines which operation needs to be triggered by the application and which result register is holding the final result

Table 7-27. Operand 1 Register

OP1 Register Alias	TMU Address offset (From 0x0006 0000)	Description
SINPUF32_R0	0x40	Operand corresponds to Sin Operation and Result should be available in R0
SINPUF32_R1	0x48	Operand corresponds to Sin Operation and Result should be available in R1
….		
SINPUF32_R7	0x78	Operand corresponds to Sin Operation and Result should be available in R7
COSPUF32_R0 – R7	0x80 – 0xB8	Operand corresponds to Cos Operation and Result should be available in Corresponding R register
ATANPUF32_R0-R7		
….		

Table 7-27. Operand 1 Register (continued)

OP1 Register Alias	TMU Address offset (From 0x0006 0000)	Description
….		

The Operand Register2 (OP2) is a single physical register. It is used only for operations needing two operands.

Table 7-28. Operand 2 Register

OP2 Register	TMU Address offset (From 0x0006 0000)	Description
QUADF32_OP2	0x240	Operand 2 Corresponding to QUADF32 operation. Y : QUADF32

There are 8 result registers

Table 7-29. Result Registers

Result Register	TMU Address offset (From 0x0006 0000)	Description
RESULT_R0-R7	0x280 – 0x2B8	Result Registers R0 to R7

7.2.2.5 Initiating TMU Operation

To initiate TMU operation, the Operand value needs to be written to a suitable alias OP1 register.

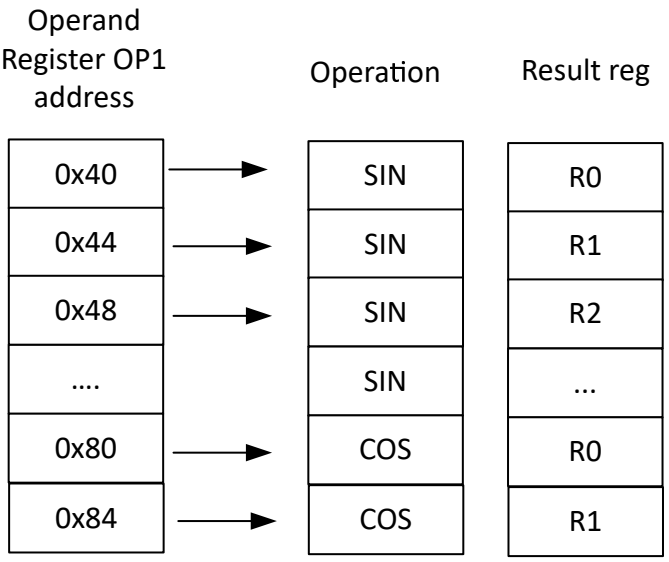


Figure 7-18. OP1 Alias Address Concept

For example, writing to offset 0x40 (SINPUF32_R0) triggers a SIN operation on value written and the result will be stored in R0 result register. It can be noted that no additional write is necessary to indicate type of operation and destination result register.

OP2 register does not require any such mapping as write to OP2 does not initiate TMU operation. In case of an operation which takes two operands, upon writing OP1, the value contained in OP2 will be taken as second operand. Note: OP2 needs to be written first followed by OP1 for the operation to return a valid value.

In case an operation requires two results registers, OP1 address determines the pair of result registers that will contain the result : Eg : If OP1 targets R0, then the result will be in R0-R1. If OP1 targets R6, then the result will be present in R6-R7 registers. The result should be read out in the same order as R0 followed by R1 , R1 followed by R2 etc.

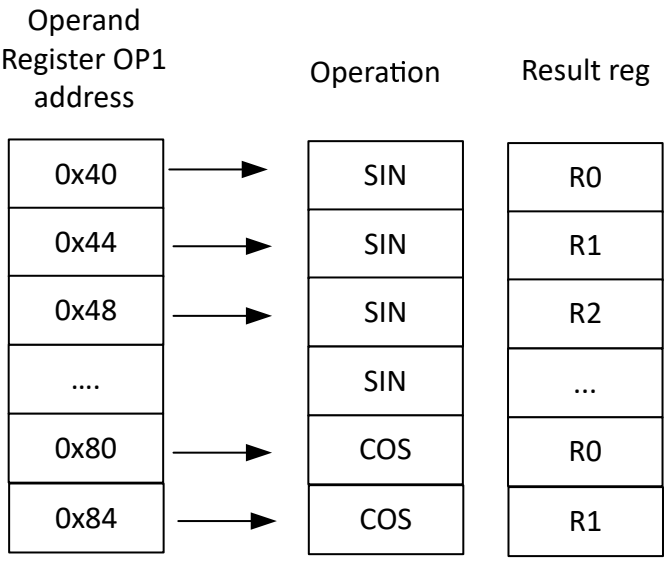


Figure 7-19. OP1 Alias Address Concept

7.2.2.5.1 Interrupt Context Save and Restore

TMU result registers supports context save and restore feature so that the TMU hardware can be used in an ISR context while simultaneously used in Main function.

Note

The CLA TMU integrated in AM26x platforms has limitations with single cycle context save. AM26x hardware does **not** support nesting interrupts, and therefore full TMU context save/restore is not feasible in the MCU PLUS SDK. **Only** single context save/restore is possible on AM26x devices. The C28x TMU hardware supports nested interrupts, and therefore supports full TMU context save/restore.

When the interrupt is taken and the ISR starts running, the context save can be initiated by writing ‘1’ to CONTEXT_SAVE.SAVE bit. TMU result registers are saved to CSAVE_<*> registers. Context save will happen only after all operations initiated before writing to CONTEXT_SAVE.SAVE are complete. This is to ensure that context save happens at correct point.

Even though TMU operations are multi-cycle, the TMU operation will have completed by the time context save operation is initiated in ISR. So no additional measure is needed in ISR. After saving context , the ISR can use the TMU without any restriction.

Restoring TMU registers from CSAVE_<*> registers can be initiated by writing ‘1’ to CONTEXT_RESTORE.RESTORE. This can be done as the last step of the ISR before returning from ISR.

Note

It is not necessary to use the context save/restore feature if different result registers are used by Main and ISR functions.

7.2.2.5.2 Pipelined Operation

As indicated in Table 7-26, each TMU operation will take some cycles to complete and update the Result register. However TMU supports triggering of new operation during the result waiting time provided that

1. The new operation result register is not same as previous result register.
2. The new operation operands do not depend on the previous operation result

Failing to adhere to the above conditions will lead to TMU result values being stale.

7.2.2.6 Result Reading Methods

TMU is interfaced to the TCMA bus of R5. TMU operation takes some cycles for the result to be available. Once the operation is triggered, if the result register is read prematurely, in order to avoid reading an incorrect result, the TCM bus is stalled until the valid result is updated in the TMU.

Note

Since TCM is treated as Normal memory by R5 CPU, in order to ensure the write is ordered before the read, a Barrier instruction(DNB) is necessary.

7.2.2.6.1 Single Operation

A single operation to TMU can be performed as follows

1. Write the Operand OP2 to TMU (*For two operand operation only*)
2. Write the Operand OP1 to TMU
3. DBN (*Barrier instruction to Order the Write first and then read*)
4. Read the Result1 from R0-R7 Register
5. Read the Result2 from R1-R7 Register (*For two result operation only*)

7.2.2.6.2 Pipelined Operation

A pipelined TMU operation can be perfomed as follows

1. Write the Operand OP1 of First Operation
2. Write the Operand OP1 for second Operation
3. …Up to total of 8 Operations
4. DBN (*Barrier instruction to Order the Write first and then read*)
5. NOP
6. NOP
7. NOP
8. NOP (*Four NOP's needed for avoiding hazard and ensure result is valid for pipeline mode*)
9. Read the Result for First Operation
10. Read the Result for Second Operation
11. …Up to total of 8 Results

7.2.2.7 TMU Data Format

The encoding of the floating-point formats is given in the below table

Table 7-30. IEEE 32Bit Single Precision Floating Point Format

S32	E32 (7:0)	M32 (22:0)	Value (V)
0	0	0	Zero (V = 0)
1	0	0	Negative Zero (V = -0)
0 +ive 1 -ive	0	Non-zero	De-normalized (V=(-1)^{S*} 2($^{-126)*}$ (0.M))
0 +ive 1 -ive	1 to 254	0 to 0x7FFFFF	Normal Range (V=(-1)$^{S\ *}$ 2$^{(E-127)\ *}$ (1.M))
0	254	0x7FFFFF	Positive Max (V = +Max)
1	254	0x7FFFFF	Negative Max (V = -Max)
0	Maximum = 255	0	Positive Infinity (V = +Infinity)
1	Maximum = 255	0	Negative Infinity (V = -Infinity)
X	Maximum = 255	Non-zero	Not A Number (V = NaN)

7.2.2.7.1 Negative Zero

All TMU operations generate a positive (S==0, E==0, M==0) zero, never a negative zero if the result of the operation is zero. All TMU operations treat negative zero operations as zero.

7.2.2.7.2 De-Normalized Numbers

A de-normalized operand (E==0, M!=0) input is treated as zero (E==0, M==0) by all TMU operations. TMU operations never generate a de-normalized value.

7.2.2.7.3 Underflow

Underflow occurs when an operation generates a value that is too small to represent in the given floating-point format. Under such cases, a zero value is returned. If a TMU operation generates an underflow condition, then the latched underflow flag (LUF) is set to 1. The LUF flag will remain latched until cleared by the user executing an instruction that clears the flag. It also generates an interrupt to the respective R5 Core **R5FSS*_CORE*_INTR_R5SS0_CPU0_TMU_LUF(#210)**

7.2.2.7.4 Overflow

Overflow occurs when an operation generates a value that is too large to represent in the given floating-point format. Under such cases, a ± Infinity value is returned. If a TMU operation generates an overflow condition, then the latched overflow flag (LVF) is set to 1. The LVF flag will remain latched until cleared by the user executing an instruction that clears the flag. It also generates an interrupt to the respective R5 Core **R5FSS*_CORE*_INTR_R5SS0_CPU0_TMU_LVF(#209)**

7.2.2.7.5 Rounding

There are various rounding formats supported by the IEEE standard. Rounding has no meaning for TMU operations (rounding is inherent in the implementation). Hence rounding mode is ignored by TMU operations.

7.2.2.7.6 Infinity and Not a Number (NaN)

An NaN operand (E==maximum, M!=0) input is treated as Infinity (E==maximum, M==0) for all operations. TMU operations will never generate a NaN value but Infinity instead.

7.2.2.7.7 Common Restrictions

For all the TMU instructions, the inputs are conditioned as follows (LVF and LUF are not affected):

- Negative zero is treated as positive zero
- Positive or negative denormalized numbers are treated as positive zero
- Positive and negative NaN are treated as positive and negative infinity, respectively

7.2.2.8 ROM Parity Error

TMU contains a ROM internal to the module for storing the TMU coefficients.

A parity check logic is implemented on the ROM access path internal to TMU. If a parity error is detected on the interface, an Error event is generated < >

7.2.3 Data Format

The treatment of the various IEEE floating-point numerical formats for this TMU is the same as the FPU implementation.

7.2.3.1 Floating Point Encoding

The encoding of the floating-point formats is given in Table 7-31.

Table 7-31. IEEE 32-Bit Single Precision Floating-Point Format

S32	E32 (7:0)	M32 (22:0)	Value (V)
0	0	0	Zero (V = 0)
1	0	0	Negative Zero (V = -0)
0 +ve 1 -ve	0	non zero	De-normalized (V=(-1)^{S}* 2$^{(-126)}$* (0.M))
0 +ve 1 -ve	1 to 254	0 to 0x7FFFFF	Normal Range (V=(-1)^{S} * 2$^{(E-127)}$ * (1.M))
0	254	0x7FFFFF	Positive Max (V = +Max)
1	254	0x7FFFFF	Negative Max (V = -Max)
0	Maximum = 255	0	Positive Infinity (V = +Infinity)
1	Maximum = 255	0	Negative Infinity (V = -Infinity)
x	Maximum = 255	non zero	Not A Number (V = NaN)

7.2.3.2 Negative Zero

All TMU operations generate a positive (S==0, E==0, M==0) zero, never a negative zero if the result of the operation is zero. All TMU operations treat negative zero operations as zero.

7.2.3.3 De-Normalized Numbers

A de-normalized operand (E==0, M!=0) input is treated as zero (E==0, M==0) by all TMU operations. TMU operations never generate a de-normalized value.

7.2.3.4 Underflow

Underflow occurs when an operation generates a value that is too small to represent in the given floating-point format. Under such cases, a zero value is returned. If a TMU operation generates an underflow condition, then the latched underflow flag (LUF) is set to 1. The LUF flag will remain latched until cleared by the user executing an instruction that clears the flag. It also generates an interrupt to the respective R5 Core **R5FSS*_CORE*_INTR_R5SS0_CPU0_TMU_LUF(#210)**

7.2.3.5 Overflow

Overflow occurs when an operation generates a value that is too large to represent in the given floating-point format. Under such cases, a ± Infinity value is returned. If a TMU operation generates an overflow condition, then the latched overflow flag (LVF) is set to 1. The LVF flag will remain latched until cleared by the user executing an instruction that clears the flag. It also generates an interrupt to the respective R5 Core **R5FSS*_CORE*_INTR_R5SS0_CPU0_TMU_LVF(#209)**

7.2.3.6 Rounding

There are various rounding formats supported by the IEEE standard. Rounding has no meaning for TMU operations (rounding is inherent in the implementation). Hence rounding mode is ignored by TMU operations.

7.2.3.7 Infinity and Not a Number (NaN)

An NaN operand (E==maximum, M!=0) input is treated as Infinity (E==maximum, M==0) for all operations. TMU operations will never generate a NaN value but Infinity instead.

7.2.3.8 Common Restrictions

For all the TMU instructions, the inputs are conditioned as follows (LVF and LUF are not affected):

- Negative zero is treated as positive zero
- Positive or negative denormalized numbers are treated as positive zero
- Positive and negative NaN are treated as positive and negative infinity, respectively

7.2.4 TMU Operation Psuedo Code

This chapter gives example for asm code for single operation and pipelined operation

7.2.4.1 Single Operation

The pseudo code for single operand (e.g. sinpuf32) is

1. Mov r0,#oprand1_data *// moving the operand1 to r0*
2. Mov r1, #oprand1_address *// moving the operand1 address to r1*
3. Mov r2, #resultR0_address *// moving the result R0 address to r2*
4. STR r0, [r1] *// write to do a single operand TMU operation*
5. DMB *// to ensure strongly ordered read write*
6. LDR r3, [r2] *// moving the TMU result R0 data into r3*

The pseudo code for two operands with two result (e.g. QUADF32) is

1. Mov r0,#oprand1_data // moving the operand1 to r0
2. Mov r1,#oprand2_data // moving the operand2 to r1
3. Mov r2,#oprand1_address // moving the operand1 address to r2
4. Mov r3,#oprand2_address // moving the operand2 address to r3
5. Mov r4, #resultR0_address // moving the result R0 address to r4
6. Mov r5, #resultR1_address // moving the result R1 address to r5
7. STR r1, [r3] // writing TMU operand2
8. STR r0, [r2] // writing TMU operand1
9. DMB // to ensure strongly ordered read write
10. LDR r6, [r4] // moving the TMU result R0 data into r6
11. LDR r7, [r5] // moving the TMU result R1 data into r7

7.2.4.2 Pipelined Operation

The pseudo code for single operand pipeline operation is

1. Mov r0,#oprand1_data // moving the operand1 to r0 for First Operation
2. Mov r1, #oprand1_address // moving the operand1 address to r1 for First Operation
3. Mov r2, #resultR0_address // moving the result R0 address to r2 for First Operation
4. Mov r3,#oprand1_data // moving the operand1 to r0 for Second Operation
5. Mov r4, #oprand1_address // moving the operand1 address to r1 for Second Operation
6. Mov r5, #resultR0_address // moving the result R0 address to r2 for Second Operation
7. STR r0, [r1] // write to do a single operand TMU operation for First Operation
8. STR r3, [r4] // write to do a single operand TMU operation for First Operation
9. DMB // to ensure strongly ordered read write
10. NOP
11. NOP
12. NOP
13. NOP
14. LDR r6, [r2] // moving the TMU result R0 data into r6
15. LDR r7, [r5] // moving the TMU result R1 data into r7

Note

TI recommends to read the results in the same order as the operation is performed.

7.3 Programmable Real-Time Unit Subsystem (PRU-ICSS)

This section describes the Programmable Real-Time Unit Subsystem in the device.

Note

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

Note

The PRU Subsystem (PRUSS) is a subset of the PRU Industrial Communication Subsystem (PRU-ICSS) found on other TI processors. The superset names "PRU-ICSS", and "ICSSM" are used in some parts of the TRM to refer to the PRU Subsystem. Reference the PRU-ICSS Module Integration section for information about PRU-ICSS features that are unsupported in PRU-ICSS.

7.3.1 PRU-ICSS Overview

The Programmable Real-Time Unit Subsystem (PRU-ICSS) consists of:

- Two 32-bit load/store RISC CPU cores - Programmable Real-Time Units (PRU0 and PRU1)
- Data RAMs per PRU core (DRAM[0:1])
- Instruction RAM per PRU core (IRAM[0])
- Shared RAM (SMEM/DRAM[2])
- Peripheral modules: UART, ECAP, IEP, MDIO
- Interrupt Controller (INTC) per core

The programmable nature of the PRU cores, along with their access to pins, events and all device resources, provides flexibility in implementing fast real-time responses, specialized data handling operations, custom peripheral interfaces, and in offloading tasks from the other processor cores of the device.

The PRU cores are programmed with a small, deterministic instruction set. Each PRU can operate independently or in coordination with each other and can also work in coordination with the device-level host CPU. This interaction between processors is determined by the nature of the firmware loaded into the PRU's instruction memory.

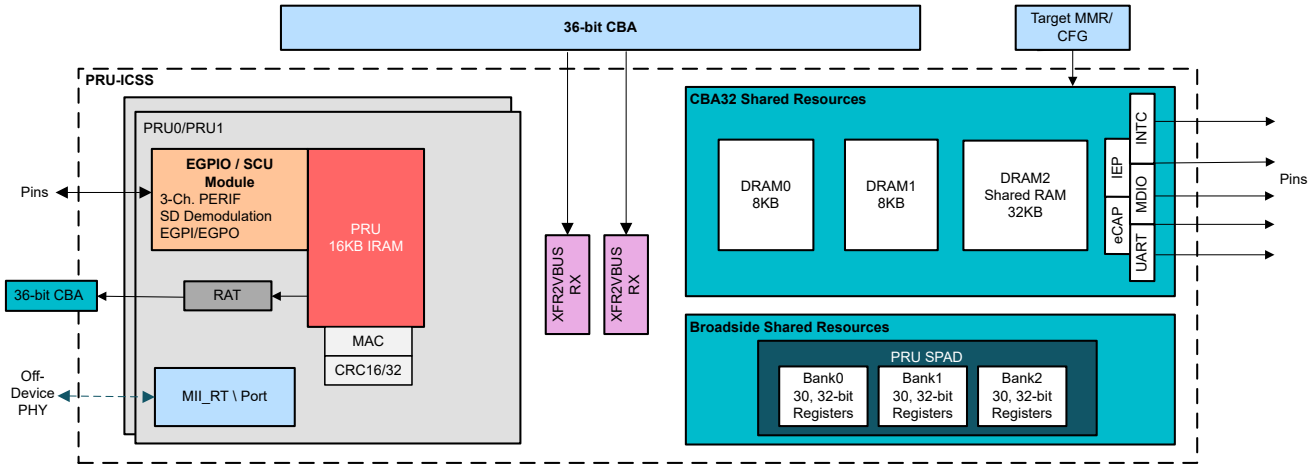


Figure 7-20. PRU-ICSS Overview

7.3.1.1 PRU-ICSS Key Features

The PRU-ICSS subsystem includes the following main features:

- Two 32-bit load/store RISC CPU cores — Programmable Real-Time Units (PRU0 and PRU1), each with:
  - 20 Enhanced General-Purpose Inputs (EGPI) and 20 Enhanced General-Purpose Outputs (EGPO)
  - Asynchronous capture [Serial Capture Unit (SCU)] with 3-channel peripheral interface and Sigma-Delta demodulation support
    - The 3-channel peripheral interface supports multiple different encoder protocols such as EnDAT 2.2, HDSL, and Tamagawa.
  - 16KB program memory per PRU (PRU0_IRAM and PRU1_IRAM) with ECC
  - MAC (Multiplier with optional Accumulation)
  - CRC16/CRC32 hardware accelerator
  - Broadside (32 Byte) connection to MII_RTn (where n = 1 or 2)
  - RX XFR2VBUS
- Scratchpad Memory (SPAD) with 3 banks of 30 × 32-bit registers:
  - 3 banks shared between the PRU0 and PRU1 cores
- 32 KB Shared general purpose memory RAM with ECC (SRAM/DRAM2), shared between PRU0 and PRU1
- Two 8 KB (shared) Data Memories with ECC (DRAM0 and DRAM1)
- 36-bit VBUSM Controller Port:
  - Optional address translation for all transactions to External Host

- 16 Software Events generated by 2 PRUs
- Two Real-Time Ethernet ports (MII_RT1 and MII_RT2) configurable to connect to each PRUn (where n = 0 or 1) to support multiple industrial communication protocols
- One Industrial Ethernet Peripheral (IEP0) to manage/generate Industrial Ethernet functions such as time stamping
  - Industrial Ethernet 64-bit timers support 10 capture and 16 compare events along with slow and fast compensation
- One MDIO port to control external Ethernet PHY
- One Enhanced Capture Module (ECAP0)
- Interrupt Controller (INTC)
  - Up to 32 internal events, generated by modules, internal to the PRU-ICSS
  - Up to 32 external events, generated by the system
  - Supports up to 10 interrupt channels
  - Generation of up to 10 Host interrupts:
    - Up to 2 Host interrupts, exported from the PRU-ICSS for signaling the Arm interrupt controllers (pulse and level provided)
  - Each system event can be enabled and disabled
  - Each host event can be enabled and disabled
  - Hardware prioritization of events
- One 32-bit VBUSP target port for memory mapped register and internal memories access
- Flexible power management support
- Integrated 32-bit Interconnect

7.3.1.2 Not Supported Features

The following PRU-ICSS features are not supported:

- Industrial Communications Subsystem features
  - Low power clock enable suport
  - The following GPIO and mux modes are not pinned out:
    - PR0_PRU0_GPIO7
      - PR0_PRU0_PERIF2_OUT
      - PR0_PRU0_SD3_D
    - PR0_PRU0_GPIO17
      - PR0_PRU0_SD8_D
    - PR0_PRU0_GPIO18
    - PR0_PRU0_GPIO19
  - SD mode on PRU1
  - Integrated PWM module

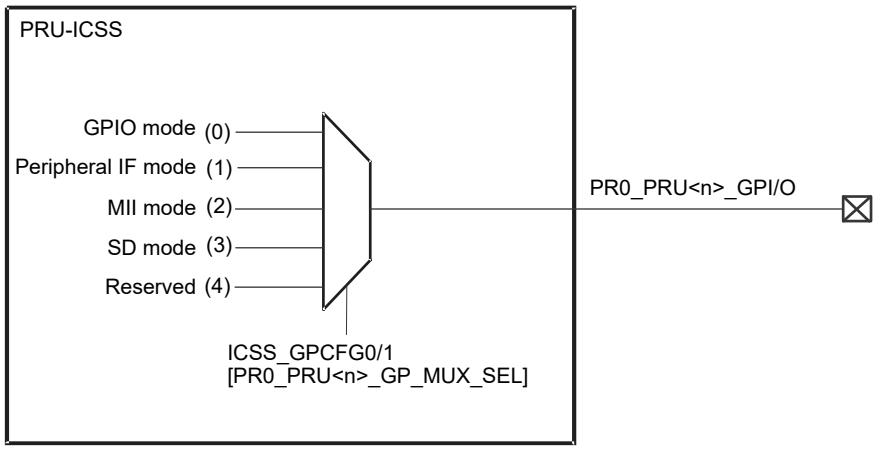
7.3.2 PRU-ICSS Environment

This section describes the PRU-ICSS external connections (environment).

7.3.2.1 PRU-ICSS Internal Pinmux

The PRU-ICSS external interface signals are described in Table 7-32. The PRU-ICSS has a large number of available I/O signals. Most of these are multiplexed with other functional signals at the device level.

The PRU-ICSS also support an internal wrapper multiplexing that expands the device top-level multiplexing. This wrapper multiplexing is controlled by the GPCFGx_REG register (where x = 0 or 1) in the PRU-ICSS CFG register space and allows MII_RT, 3 channel Peripheral Interface (with EnDAT capabilities), and Sigma Delta functionality to be muxed with the PRU GPI/O device signals, as shown in Figure 7-21. The PRU-ICSS wrapper multiplexing is described with the device-level signals in Table 7-32. Note that the device top-level muxing has higher priority over the internal PRU-ICSS muxing.



1. n represents a valid instance of PRU in a domain.

Figure 7-21. PRU-ICSS Internal Wrapper Multiplexing

Note

Additionally to PRU-ICSS wrapper multiplexing the device I/O logic maps the PRU-ICSS signals to the different device pins by programming the associated IOMUX CTRLMMR register.

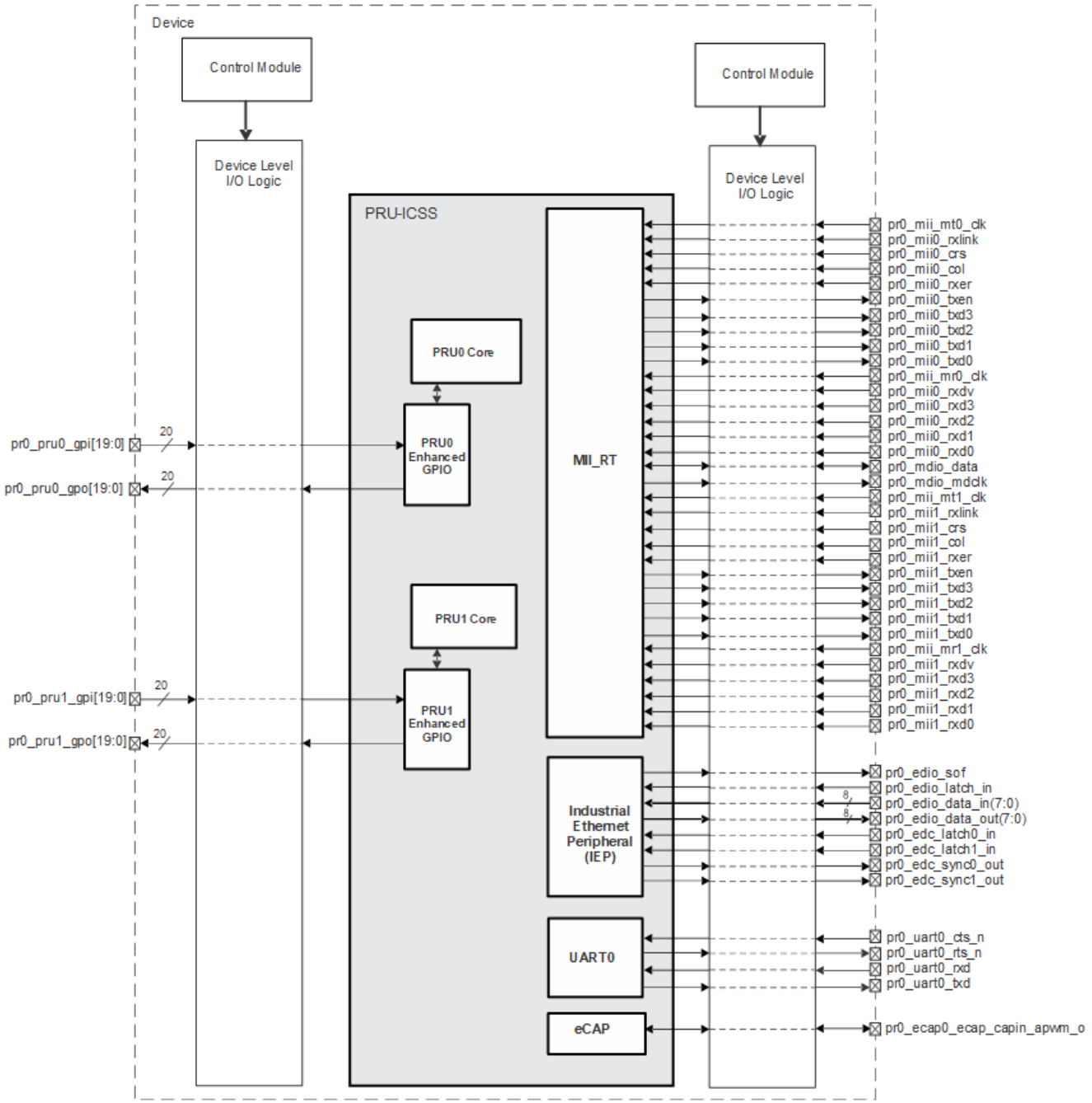


Figure 7-22. PRU-ICSS External Interface I/Os

PRU-ICSS I/O Signals

Table 7-32 describes the PRU-ICSS<k> I/O signals.

Note

<k> is the number of PRU-ICSS in the device. See the Data sheet for additional details.

Table 7-32. PRU-ICSS I/O Signals

Device Level Signal	Alternate Function via Internal Multiplexing				I/O[1]	Description	Pin Reset [2]
	ICSS_GPCFG0_REG[29-26] PR<k_PRU0_GP_MUX_SEL=						
PRU0 GP Signals	0h - GPIO mode (default)	1h - PERIF mode	2h - MII mode	3h - SD mode			
PR0_PRU0_GPO0	pr<k>_pru0_pru_r30_out[0]	pr<k>_pru0_perif0_clk			O	PRU0 R30 Outputs	0
PR0_PRU0_GPO1	pr<k>_pru0_pru_r30_out[1]	pr<k>_pru0_perif0_out		pr<k>_pru0_pru_r30_out[1]	O	PRU0 R30 Outputs	0
PR0_PRU0_GPO2	pr<k>_pru0_pru_r30_out[2]	pr<k>_pru0_perif0_out_en			O	PRU0 R30 Outputs	0
PR0_PRU0_GPO3	pr<k>_pru0_pru_r30_out[3]	pr<k>_pru0_perif1_clk			O	PRU0 R30 Outputs	0
PR0_PRU0_GPO4	pr<k>_pru0_pru_r30_out[4]	pr<k>_pru0_perif1_out			O	PRU0 R30 Outputs	0
PR0_PRU0_GPO5	pr<k>_pru0_pru_r30_out[5]	pr<k>_pru0_perif1_out_en			O	PRU0 R30 Outputs	0
PR0_PRU0_GPO6	pr<k>_pru0_pru_r30_out[6]	pr<k>_pru0_perif2_clk			O	PRU0 R30 Outputs	0
PR0_PRU0_GPO7[5]	pr<k>_pru0_pru_r30_out[7][5]	pr<k>_pru0_perif2_out[5]			O	PRU0 R30 Outputs	0
PR0_PRU0_GPO8	pr<k>_pru0_pru_r30_out[8]	pr<k>_pru0_perif2_out_en			O	PRU0 R30 Outputs	0
PR0_PRU0_GPO9	pr<k>_pru0_pru_r30_out[9]				O	PRU0 R30 Outputs	0
PR0_PRU0_GPO10	pr<k>_pru0_pru_r30_out[10]				O	PRU0 R30 Outputs	0
PR0_PRU0_GPO11	pr<k>_pru0_pru_r30_out[11]		pr<k>_mii1_txd[0][3]		O	PRU0 R30 Outputs	0
PR0_PRU0_GPO12	pr<k>_pru0_pru_r30_out[12]		pr<k>_mii1_txd[1][3]		O	PRU0 R30 Outputs	0
PR0_PRU0_GPO13	pr<k>_pru0_pru_r30_out[13]		pr<k>_mii1_txd[2][3]		O	PRU0 R30 Outputs	0
PR0_PRU0_GPO14	pr<k>_pru0_pru_r30_out[14]		pr<k>_mii1_txd[3][3]		O	PRU0 R30 Outputs	0
PR0_PRU0_GPO15	pr<k>_pru0_pru_r30_out[15]		pr<k>_mii1_txen[3]		O	PRU0 R30 Outputs	0
PR0_PRU0_GPO16	pr<k>_pru0_pru_r30_out[16]				O	PRU0 R30 Outputs	0
PR0_PRU0_GPO17[5]	pr<k>_pru0_pru_r30_out[17][5]				O	PRU0 R30 Outputs	0
PR0_PRU0_GPO18[5]	pr<k>_pru0_pru_r30_out[18][5]				O	PRU0 R30 Outputs	0
PR0_PRU0_GPO19[5]	pr<k>_pru0_pru_r30_out[19][5]				O	PRU0 R30 Outputs	0
PR0_PRU0_GPI0	pr<k>_pru0_pru_r31_in[0]		pr<k>_mii0_rxd[0]	pr<k>_pru0_sd0_clk	I	PRU0 R31 Inputs	HiZ
PR0_PRU0_GPI1	pr<k>_pru0_pru_r31_in[1]		pr<k>_mii0_rxd[1]	pr<k>_pru0_sd0_d	I	PRU0 R31 Inputs	HiZ
PR0_PRU0_GPI2	pr<k>_pru0_pru_r31_in[2]		pr<k>_mii0_rxd[2]	pr<k>_pru0_sd1_clk	I	PRU0 R31 Inputs	HiZ
PR0_PRU0_GPI3	pr<k>_pru0_pru_r31_in[3]		pr<k>_mii0_rxd[3]	pr<k>_pru0_sd1_d	I	PRU0 R31 Inputs	HiZ
PR0_PRU0_GPI4	pr<k>_pru0_pru_r31_in[4]		pr<k>_mii0_rxdv	pr<k>_pru0_sd2_clk	I	PRU0 R31 Inputs	HiZ
PR0_PRU0_GPI5	pr<k>_pru0_pru_r31_in[5]		pr<k>_mii0_rxer	pr<k>_pru0_sd2_d	I	PRU0 R31 Inputs	HiZ
PR0_PRU0_GPI6	pr<k>_pru0_pru_r31_in[6]		pr<k>_mii_mr0_clk	pr<k>_pru0_sd3_clk	I	PRU0 R31 Inputs	HiZ
PR0_PRU0_GPI7[5]	pr<k>_pru0_pru_r31_in[7][5]			pr<k>_pru0_sd3_d[5]	I	PRU0 R31 Inputs	HiZ

Table 7-32. PRU-ICSS I/O Signals (continued)

Device Level Signal	Alternate Function via Internal Multiplexing				I/O[1]	Description	Pin Reset [2]
PR0_PRU0_GPI8	pr<k>_pru0_pru_r31_in[8]		pr<k>_mii0_rxlink	pr<k>_pru0_sd4_clk	I	PRU0 R31 Inputs	HiZ
PR0_PRU0_GPI9	pr<k>_pru0_pru_r31_in[9]	pr<k>_pru0_perif0_in	pr<k>_mii0_col	pr<k>_pru0_sd4_d	I	PRU0 R31 Inputs	HiZ
PR0_PRU0_GPI10	pr<k>_pru0_pru_r31_in[10]	pr<k>_pru0_perif1_in	pr<k>_mii0_crs	pr<k>_pru0_sd5_clk	I	PRU0 R31 Inputs	HiZ
PR0_PRU0_GPI11	pr<k>_pru0_pru_r31_in[11]	pr<k>_pru0_perif2_in		pr<k>_pru0_sd5_d	I	PRU0 R31 Inputs	HiZ
PR0_PRU0_GPI12	pr<k>_pru0_pru_r31_in[12]			pr<k>_pru0_sd6_clk	I	PRU0 R31 Inputs	HiZ
PR0_PRU0_GPI13	pr<k>_pru0_pru_r31_in[13]			pr<k>_pru0_sd6_d	I	PRU0 R31 Inputs	HiZ
PR0_PRU0_GPI14	pr<k>_pru0_pru_r31_in[14]			pr<k>_pru0_sd7_clk	I	PRU0 R31 Inputs	HiZ
PR0_PRU0_GPI15	pr<k>_pru0_pru_r31_in[15]			pr<k>_pru0_sd7_d	I	PRU0 R31 Inputs	HiZ
PR0_PRU0_GPI16	pr<k>_pru0_pru_r31_in[16]	pr<k>_pru0_pru_r31_in[16]	pr<k>_mii_mt1_clk, pr<k>_pru0_pru_r31_in[16]	pr<k>_pru0_sd8_clk, pr<k>_pru0_pru_r31_in[16]	I	PRU0 R31 Inputs	HiZ
PR0_PRU0_GPI17[5]	pr<k>_pru0_pru_r31_in[17][5]			pr<k>_pru0_sd8_d[5]	I	PRU0 R31 Inputs	HiZ
PR0_PRU0_GPI18[5]	pr<k>_pru0_pru_r31_in[18][5]				I	PRU0 R31 Inputs	HiZ
PR0_PRU0_GPI19[5]	pr<k>_pru0_pru_r31_in[19][5]				I	PRU0 R31 Inputs	HiZ
PRU1 GP Signals	ICSS_GPCFG1_REG[29-26] PR0_PRU1_GP_MUX_SEL=						
	0h - GPIO mode (default)	1h - PERIF mode	2h - MII mode	3h - SD mode[4]			
PR0_PRU1_GPO0	pr<k>_pru1_pru_r30_out[0]	pr<k>_pru1_perif0_clk			O	PRU1 R30 Outputs	0
PR0_PRU1_GPO1	pr<k>_pru1_pru_r30_out[1]	pr<k>_pru1_perif0_out		pr<k>_pru1_pru_r30_out[1]	O	PRU1 R30 Outputs	0
PR0_PRU1_GPO2	pr<k>_pru1_pru_r30_out[2]	pr<k>_pru1_perif0_out_en			O	PRU1 R30 Outputs	0
PR0_PRU1_GPO3	pr<k>_pru1_pru_r30_out[3]	pr<k>_pru1_perif1_clk			O	PRU1 R30 Outputs	0
PR0_PRU1_GPO4	pr<k>_pru1_pru_r30_out[4]	pr<k>_pru1_perif1_out			O	PRU1 R30 Outputs	0
PR0_PRU1_GPO5	pr<k>_pru1_pru_r30_out[5]	pr<k>_pru1_perif1_out_en			O	PRU1 R30 Outputs	0
PR0_PRU1_GPO6	pr<k>_pru1_pru_r30_out[6]	pr<k>_pru1_perif2_clk			O	PRU1 R30 Outputs	0
PR0_PRU1_GPO7	pr<k>_pru1_pru_r30_out[7]	pr<k>_pru1_perif2_out			O	PRU1 R30 Outputs	0
PR0_PRU1_GPO8	pr<k>_pru1_pru_r30_out[8]	pr<k>_pru1_perif2_out_en			O	PRU1 R30 Outputs	0
PR0_PRU1_GPO9	pr<k>_pru1_pru_r30_out[9]				O	PRU1 R30 Outputs	0
PR0_PRU1_GPO10	pr<k>_pru1_pru_r30_out[10]				O	PRU1 R30 Outputs	0
PR0_PRU1_GPO11	pr<k>_pru1_pru_r30_out[11]		pr<k>_mii0_txd[0][3]		O	PRU1 R30 Outputs	0
PR0_PRU1_GPO12	pr<k>_pru1_pru_r30_out[12]		pr<k>_mii0_txd[1][3]		O	PRU1 R30 Outputs	0
PR0_PRU1_GPO13	pr<k>_pru1_pru_r30_out[13]		pr<k>_mii0_txd[2][3]		O	PRU1 R30 Outputs	0
PR0_PRU1_GPO14	pr<k>_pru1_pru_r30_out[14]		pr<k>_mii0_txd[3][3]		O	PRU1 R30 Outputs	0
PR0_PRU1_GPO15	pr<k>_pru1_pru_r30_out[15]		pr<k>_mii0_txen[3]		O	PRU1 R30 Outputs	0
PR0_PRU1_GPO16	pr<k>_pru1_pru_r30_out[16]				O	PRU1 R30 Outputs	0
PR0_PRU1_GPO17	pr<k>_pru1_pru_r30_out[17]				O	PRU1 R30 Outputs	0
PR0_PRU1_GPO18	pr<k>_pru1_pru_r30_out[18]				O	PRU1 R30 Outputs	0
PR0_PRU1_GPO19	pr<k>_pru1_pru_r30_out[19]				O	PRU1 R30 Outputs	0
PR0_PRU1_GPI0	pr<k>_pru1_pru_r31_in[0]		pr<k>_mii1_rxd[0]	pr<k>_pru1_sd0_clk[4]	I	PRU1 R31 Inputs	HiZ
PR0_PRU1_GPI1	pr<k>_pru1_pru_r31_in[1]		pr<k>_mii1_rxd[1]	pr<k>_pru1_sd0_d[4]	I	PRU1 R31 Inputs	HiZ

Table 7-32. PRU-ICSS I/O Signals (continued)

Device Level Signal	Alternate Function via Internal Multiplexing				I/O[1]	Description	Pin Reset [2]
PR0_PRU1_GPI2	pr<k>_pru1_pru_r31_in[2]		pr<k>_mii1_rxd[2]	pr<k>_pru1_sd1_clk[4]	I	PRU1 R31 Inputs	HiZ
PR0_PRU1_GPI3	pr<k>_pru1_pru_r31_in[3]		pr<k>_mii1_rxd[3]	pr<k>_pru1_sd1_d[4]	I	PRU1 R31 Inputs	HiZ
PR0_PRU1_GPI4	pr<k>_pru1_pru_r31_in[4]		pr<k>_mii1_rxdv	pr<k>_pru1_sd2_clk[4]	I	PRU1 R31 Inputs	HiZ
PR0_PRU1_GPI5	pr<k>_pru1_pru_r31_in[5]		pr<k>_mii1_rxer	pr<k>_pru1_sd2_d[4]	I	PRU1 R31 Inputs	HiZ
PR0_PRU1_GPI6	pr<k>_pru1_pru_r31_in[6]		pr<k>_mii_mr1_clk	pr<k>_pru1_sd3_clk[4]	I	PRU1 R31 Inputs	HiZ
PR0_PRU1_GPI7	pr<k>_pru1_pru_r31_in[7]			pr<k>_pru1_sd3_d[4]	I	PRU1 R31 Inputs	HiZ
PR0_PRU1_GPI8	pr<k>_pru1_pru_r31_in[8]		pr<k>_mii1_rxlink	pr<k>_pru1_sd4_clk[4]	I	PRU1 R31 Inputs	HiZ
PR0_PRU1_GPI9	pr<k>_pru1_pru_r31_in[9]	pr<k>_pru1_perif0_in	pr<k>_mii1_col	pr<k>_pru1_sd4_d[4]	I	PRU1 R31 Inputs	HiZ
PR0_PRU1_GPI10	pr<k>_pru1_pru_r31_in[10]	pr<k>_pru1_perif1_in	pr<k>_mii1_crs	pr<k>_pru1_sd5_clk[4]	I	PRU1 R31 Inputs	HiZ
PR0_PRU1_GPI11	pr<k>_pru1_pru_r31_in[11]	pr<k>_pru1_perif2_in		pr<k>_pru1_sd5_d[4]	I	PRU1 R31 Inputs	HiZ
PR0_PRU1_GPI12	pr<k>_pru1_pru_r31_in[12]			pr<k>_pru1_sd6_clk[4]	I	PRU1 R31 Inputs	HiZ
PR0_PRU1_GPI13	pr<k>_pru1_pru_r31_in[13]			pr<k>_pru1_sd6_d[4]	I	PRU1 R31 Inputs	HiZ
PR0_PRU1_GPI14	pr<k>_pru1_pru_r31_in[14]			pr<k>_pru1_sd7_clk[4]	I	PRU1 R31 Inputs	HiZ
PR0_PRU1_GPI15	pr<k>_pru1_pru_r31_in[15]			pr<k>_pru1_sd7_d[4]	I	PRU1 R31 Inputs	HiZ
PR0_PRU1_GPI16	pr<k>_pru1_pru_r31_in[16]	pr<k>_pru1_pru_r31_in[16]	pr<k>_mii_mt0_clk, pr<k>_pru1_pru_r31_in[16]	pr<k>_pru1_sd8_clk, pr<k>_pru1_pru_r31_in[16][4]	I	PRU1 R31 Inputs	HiZ
PR0_PRU1_GPI17	pr<k>_pru1_pru_r31_in[17]			pr<k>_pru1_sd8_d[4]	I	PRU1 R31 Inputs	HiZ
PR0_PRU1_GPI18	pr<k>_pru1_pru_r31_in[18]				I	PRU1 R31 Inputs	HiZ
PR0_PRU1_GPI19	pr<k>_pru1_pru_r31_in[19]				I	PRU1 R31 Inputs	HiZ
MDIO	MDIO						
PR0_MDIO0_MDC	pr<k>_mdio_mdclk				O	MDIO Clock	0
PR0_MDIO0_MDIO	pr<k>_mdio_data				I/O	MDIO Data	HiZ
Industrial Ethernet (IEP0)	Industrial Ethernet						
PR0_IEP0_EDIO_OUTVALID	pr<k>_iep0_edio_outvalid				O	IEP0 Digital I/O Output Valid	0
PR0_IEP0_EDIO_DATA_IN_OUT[31:30]	pr<k>_iep0_edio_data_in_out[31:30]				I/O	IEP0 Digital I/Os Data In/Out	HiZ
PR0_IEP0_EDC_SYNC_OUT0	pr<k>_iep0_edc_sync_out0				O	IEP0 Distributed Clock Sync Out	0
PR0_IEP0_EDC_SYNC_OUT1	pr<k>_iep0_edc_sync_out1				O	IEP0 Distributed Clock Sync Out	0
PR0_IEP0_EDC_LATCH_IN0	pr<k>_iep0_edc_latch_in0				I	IEP0 Distributed Clock Latch In	HiZ
PR0_IEP0_EDC_LATCH_IN1	pr<k>_iep0_edc_latch_in1				I	IEP0 Distributed Clock Latch In	HiZ
UART0	UART0						
PR0_UART0_CTSn	pr<k>_uart0_cts_n				I	UART0 Clear to Send	HiZ
PR0_UART0_RTSn	pr<k>_uart0_rts_n				O	UART0 Request to Send	1
PR0_UART0_RXD	pr<k>_uart0_rxd				I	UART0 Receive Data	HiZ

Table 7-32. PRU-ICSS I/O Signals (continued)

Device Level Signal	Alternate Function via Internal Multiplexing	I/O[1]	Description	Pin Reset [2]
PR0_UART0_TXD	pr<k>_uart0_txd	O	UART0 Transmit Data	1
ECAP0	ECAP0			
PR0_ECAP0_IN_APWM_OUT	pr<k>_ecap0_ecap_capin_apwm_o	I/O	Enhanced capture (ECAP0) input or Auxiliary PWM out	HiZ
PR0_ECAP0_SYNC_IN	pr<k>_ecap0_ecap_syncin	I	Enhanced capture (ECAP0) Sync In	0
PR0_ECAP0_SYNC_OUT	pr<k>_ecap0_ecap_syncout	O	Enhanced capture (ECAP0) Sync Out	0

(1) I = Input; O = Output
(2) HiZ = High Impedance
(3) The PRU internal pinmux mapping provided in the TRM is part of the original hardware definition of the PRU. However, due to the flexibility provided by the IP and associated firmware configurations, this is not necessarily a hard requirement. The first PRU implementation had the MII TX pins swapped during initial SoC integration and this convention was maintained for subsequent PRU revisions to enable firmware reuse. To make use of the SDK firmware, use the SYSCONFIG generated PRU pin mapping.
(4) SD Mode is not supported on PRU1
(5) The following IO are not pinned out at the device level and therefore not supported.

7.3.3 PRU-ICSS Integration

This section describes modules integration in the device, including information about clocks, resets, and hardware requests.

7.3.3.1 PRU-ICSS Integration

There is a singular Industrial Connectivity Subsystem (ICSS) that consists of two Programmable Real-Time Units (PRU) integrated in the device. The diagram below provides a visual representation of the device integration details.

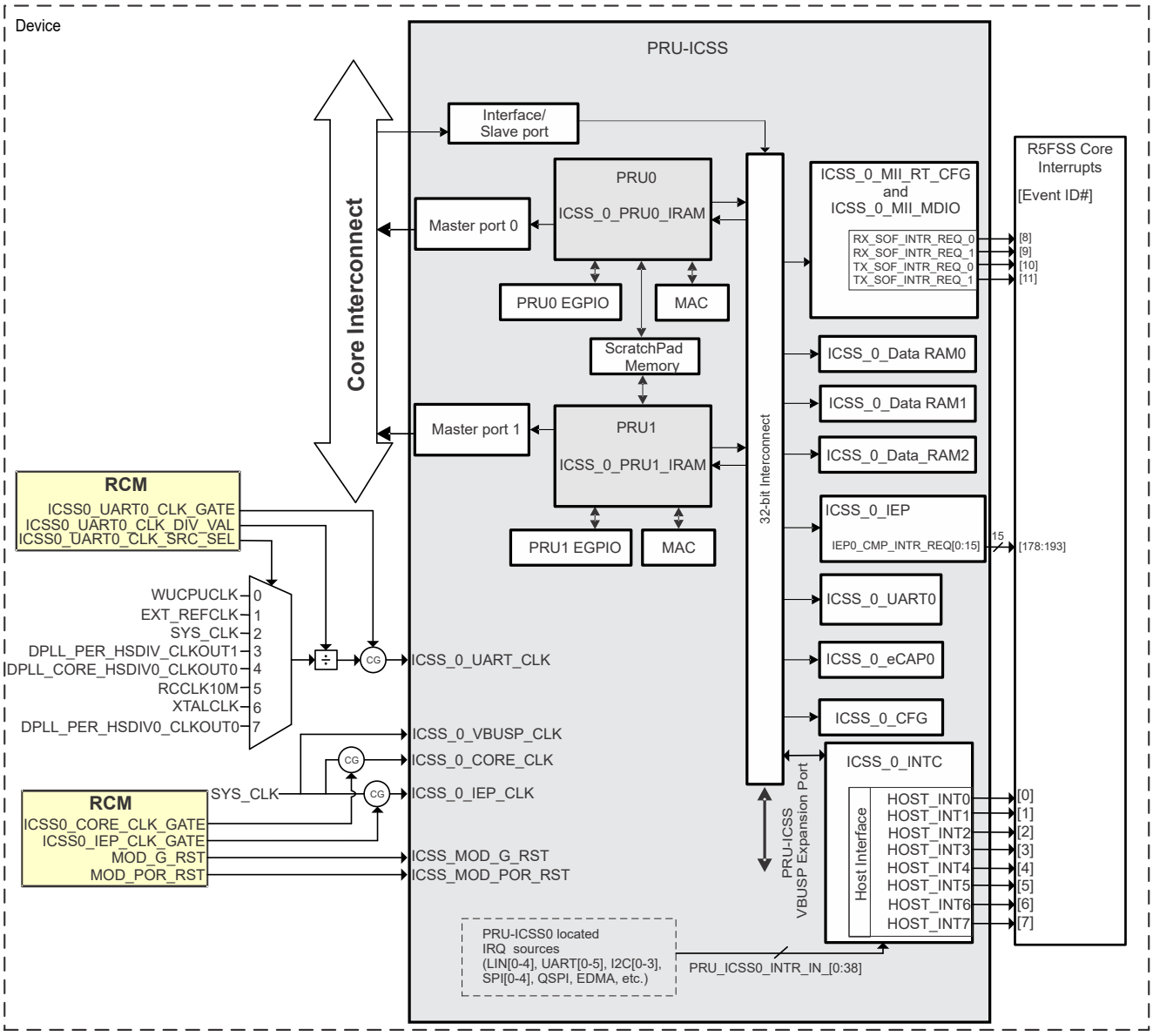


Figure 7-23. PRU-ICSS Integration Diagram

The tables below summarize the device integration details of PRU-ICSS.

Table 7-33. *PRU-ICSS* Device Integration

This table describes the module device integration details.

Module Instance	Device Allocation	SoC Interconnect
PRU0	✓	Core Interconnect

Table 7-33. *PRU-ICSS* Device Integration (continued)

This table describes the module device integration details.

Module Instance	Device Allocation	SoC Interconnect
PRU1	✓	Core Interconnect

Table 7-34. *PRU-ICSS* Clocks

This table describes the module clocking signals.

Module Instance	Module Clock Input	Source Clock Signal	Source	Default Freq	Description
ICSS	ICSS_ICLK (VBUSP_CLK)	ICSS_SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	ICSS Interface Clock
	ICSS_FCLK (CORE_CLK)	ICSS_CORE_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	ICSS Functional Core Clock
	ICSS_UART_CLK (UART_CLK)	WUCPUCLK	External XTAL or RC Oscillator	25 MHz	ICSS Selectable UART Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK: HSDIV0_CLKOUT0	400 MHz	
		RCCLK10M	Internal 10MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
		DPLL_PER_HSDIV0_CLKOUT0	PLL_PER_CLK:HSDIV0_CLKOUT0	160 MHz	
	ICSS_IEP_CLK (IEP_CLK)	ICSS_IEP_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	ICSS Industrial Ethernet Clock

Table 7-35. *PRU-ICSS* Resets

This table describes the module reset signals.

Module Instance	Module Reset Input	Source Reset Signal	Source	Description
ICSS	ICSS_RST (ICSS_WARM_RESET)	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	ICSS Warm Reset
	ICSS_RST (ICSS_POR_RESET)	POR (MOD_POR_RST)	RCM + POR Sources	ICSS Power on Reset

Note

For the device specific Interrupt Mapping, refer to Table 7-70.

Table 7-36. *PRU-ICSS* Interrupt Requests

This table describes the module interrupt requests.

Module Instance	Module Interrupt Signal	Destination Interrupt Input	Destination	Type	Description
ICSS_INTC	PR1_HOST_INTR_PEND_0	R5FSS Interrupt ID [0]	ALL R5FSS Cores	Level	PRU-ICSS Host interrupt 0
	PR1_HOST_INTR_PEND_1	R5FSS Interrupt ID [1]	ALL R5FSS Cores	Level	PRU-ICSS Host interrupt 1
	PR1_HOST_INTR_PEND_2	R5FSS Interrupt ID [2]	ALL R5FSS Cores	Level	PRU-ICSS Host interrupt 2
	PR1_HOST_INTR_PEND_3	R5FSS Interrupt ID [3]	ALL R5FSS Cores	Level	PRU-ICSS Host interrupt 3
	PR1_HOST_INTR_PEND_4	R5FSS Interrupt ID [4]	ALL R5FSS Cores	Level	PRU-ICSS Host interrupt 4
	PR1_HOST_INTR_PEND_5	R5FSS Interrupt ID [5]	ALL R5FSS Cores	Level	PRU-ICSS Host interrupt 5
	PR1_HOST_INTR_PEND_6	R5FSS Interrupt ID [6]	ALL R5FSS Cores	Level	PRU-ICSS Host interrupt 6
	PR1_HOST_INTR_PEND_7	R5FSS Interrupt ID [7]	ALL R5FSS Cores	Level	PRU-ICSS Host interrupt 7
ICSS_MII	PR1_RX_SOF_INTR_REQ_0	R5FSS Interrupt ID [8]	ALL R5FSS Cores	Level	PRU-ICSS RX SOF Interrupt Request 0
	PR1_RX_SOF_INTR_REQ_1	R5FSS Interrupt ID [9]	ALL R5FSS Cores	Level	PRU-ICSS RX SOF Interrupt Request 1
	PR1_TX_SOF_INTR_REQ_0	R5FSS Interrupt ID [10]	ALL R5FSS Cores	Level	PRU-ICSS TX SOF Interrupt Request 0
	PR1_TX_SOF_INTR_REQ_1	R5FSS Interrupt ID [11]	ALL R5FSS Cores	Level	PRU-ICSS TX SOF Interrupt Request 1

Table 7-36. *PRU-ICSS* Interrupt Requests (continued)

This table describes the module interrupt requests.

Module Instance	Module Interrupt Signal	Destination Interrupt Input	Destination	Type	Description
ICSS_IEP	PR1_IEP0_CMP_INTR_REQ_0	R5FSS Interrupt ID [178]	ALL R5FSS Cores	Level	PRU-ICSS IEP Compare Event Interrupt
	PR1_IEP0_CMP_INTR_REQ_1	R5FSS Interrupt ID [179]	ALL R5FSS Cores	Level	PRU-ICSS IEP Compare Event Interrupt
	PR1_IEP0_CMP_INTR_REQ_2	R5FSS Interrupt ID [180]	ALL R5FSS Cores	Level	PRU-ICSS IEP Compare Event Interrupt
	PR1_IEP0_CMP_INTR_REQ_3	R5FSS Interrupt ID [181]	ALL R5FSS Cores	Level	PRU-ICSS IEP Compare Event Interrupt
	PR1_IEP0_CMP_INTR_REQ_4	R5FSS Interrupt ID [182]	ALL R5FSS Cores	Level	PRU-ICSS IEP Compare Event Interrupt
	PR1_IEP0_CMP_INTR_REQ_5	R5FSS Interrupt ID [183]	ALL R5FSS Cores	Level	PRU-ICSS IEP Compare Event Interrupt
	PR1_IEP0_CMP_INTR_REQ_6	R5FSS Interrupt ID [184]	ALL R5FSS Cores	Level	PRU-ICSS IEP Compare Event Interrupt
	PR1_IEP0_CMP_INTR_REQ_7	R5FSS Interrupt ID [185]	ALL R5FSS Cores	Level	PRU-ICSS IEP Compare Event Interrupt
	PR1_IEP0_CMP_INTR_REQ_8	R5FSS Interrupt ID [186]	ALL R5FSS Cores	Level	PRU-ICSS IEP Compare Event Interrupt
	PR1_IEP0_CMP_INTR_REQ_9	R5FSS Interrupt ID [187]	ALL R5FSS Cores	Level	PRU-ICSS IEP Compare Event Interrupt
	PR1_IEP0_CMP_INTR_REQ_10	R5FSS Interrupt ID [188]	ALL R5FSS Cores	Level	PRU-ICSS IEP Compare Event Interrupt
	PR1_IEP0_CMP_INTR_REQ_11	R5FSS Interrupt ID [189]	ALL R5FSS Cores	Level	PRU-ICSS IEP Compare Event Interrupt
	PR1_IEP0_CMP_INTR_REQ_12	R5FSS Interrupt ID [190]	ALL R5FSS Cores	Level	PRU-ICSS IEP Compare Event Interrupt
	PR1_IEP0_CMP_INTR_REQ_13	R5FSS Interrupt ID [191]	ALL R5FSS Cores	Level	PRU-ICSS IEP Compare Event Interrupt
	PR1_IEP0_CMP_INTR_REQ_14	R5FSS Interrupt ID [192]	ALL R5FSS Cores	Level	PRU-ICSS IEP Compare Event Interrupt
	PR1_IEP0_CMP_INTR_REQ_15	R5FSS Interrupt ID [193]	ALL R5FSS Cores	Level	PRU-ICSS IEP Compare Event Interrupt

Note

For more information on the interconnects, see the *System Interconnect* chapter.

For more information on power, reset, and clock management, see the corresponding sections within the *Device Configuration* chapter.

For more information on the device interrupt controllers, see the *Interrupt Controllers* chapter.

7.3.4 PRU-ICSS Top Level Resources Functional Description

This section provides functional description of the device integrated PRU Subsystems modules.

The PRUn (where n = 0 or 1) cores within each PRU-ICSS have access to all resources on the SoC through the VBUSM Interface Controller port, and the external host processors can access the PRU-ICSS resources through the VBUSP Interface Target port. The use of XFR2VBUS allows BroadSide 32Bytes of data transfer to/ from SoC CBASS0 Interconnect at 256-bit bursts using the VBUSM Controller port. The 32-bit Internal CBASS Interconnect bus will be the primary interconnect between all components internal to the PRU-ICSS. There are two equally symmetrical halves in each PRU-ICSS known as SLICE0 and SLICE1. Each slice will share several resources while capable of working independently of each other. There are two sets of XFR2VBUS for each Slice. PRUs also has the ability to submit 32-bit bursts transitions, but this will require RAT configuration.

Each of the Slices contains one RAT (Region based Address Translation) module. The RAT module is used to translate 32-bit address of the PRU core to 48-bit physical address.

The PRU cores within the subsystems also have access to all resources on the SoC through the External CBASS0 Interconnect. A subsystem local Interrupt Controller — INTC handles system input events and posts events back to the device-level host CPUs.

Figure 7-24 shows an overview of the PRU-ICSS Functional Block Diagram.

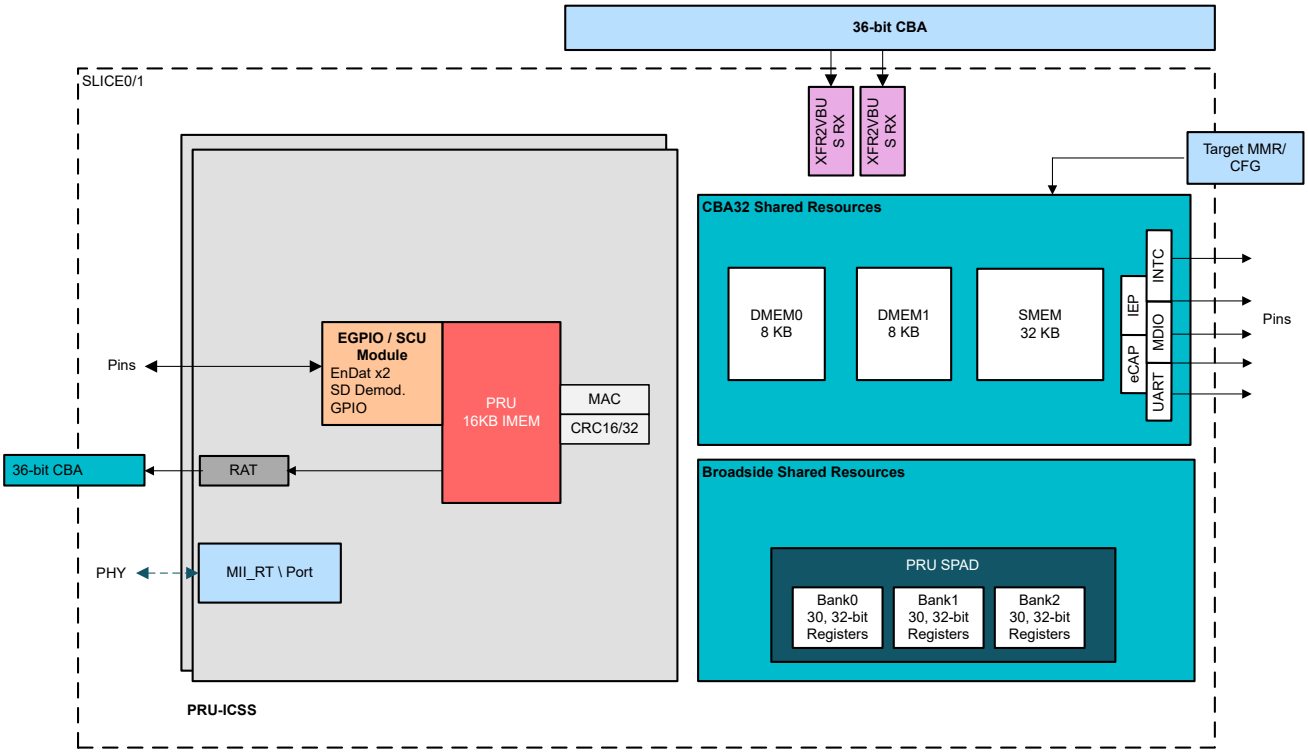


Figure 7-24. PRU-ICSS Functional Block Diagram

Table 7-37 summarizes the mapping between hardware modules and PRU0/1 cores.

Table 7-37. Hardware Module Broadside ID Mapping

Hardware Module	Broadside ID	
MPY/MAC	00	2 copies: PRU1/0
CRC16/32	01	2 copies: PRU1/0
SPAD Bank0	10	shared between PRU1/0
SPAD Bank1	11	shared between PRU1/0
SPAD Bank2	12	shared between PRU1/0

Table 7-37. Hardware Module Broadside ID Mapping (continued)

Hardware Module	Broadside ID	
RX L2	20/21	2 copies: PRU1/0
TX L2	40	2 copies: PRU1/0
XFR2VBUSP	0x60 for RD_ID0 0x61 for RD_ID1 0x62 for WD_ID0 0x63 for WD_ID1	2 copies shared of RX per SLICE 2 copies shared of TX per SLICE

7.3.4.1 PRU-ICSS Reset Management

The device supports warm reset isolation (Hard/Soft Reset, Watchdog Reset) on PRU-ICSS.

7.3.4.2 PRU-ICSS Power and Clock Management

The PRU-ICSS supports two levels of clock gating. First level gates all clocks inside the PRU-ICSS when requested by the RCM (Reset Control Manger. The second level allows user software to enable/disable clocks in the clock gating register ICSS_CGR_REG to some internal modules, as follows:

- IEP
- ECAP0
- UART0
- INTC

The appropriate configuration registers block controls its local module set inside PRU-ICSS.

7.3.4.2.1 PRU-ICSS CORE Clock Generation

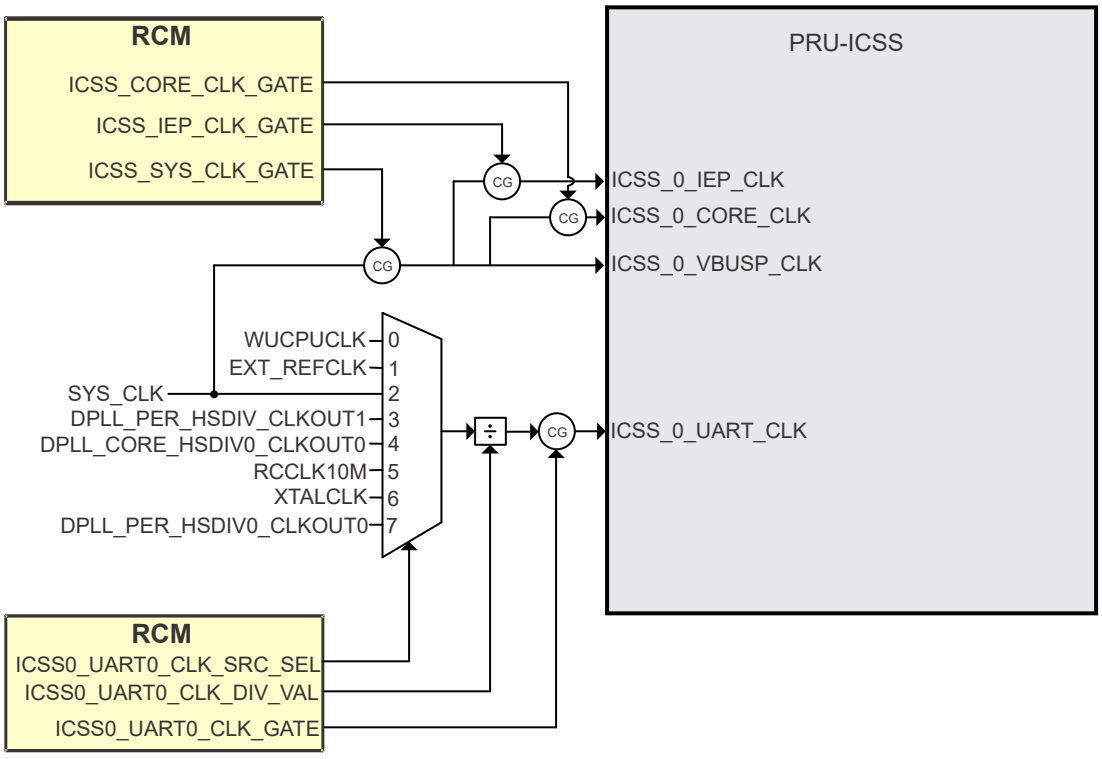


Figure 7-25. PRU-ICSS Clock Diagram

The CORE, BUS, and IEP Clock all use the 200 MHz SYS_CLK as a source clock. The UART clock is configurable by configuring the UART clock source select register as well as the UART clock divider value register. Each of these clock sources has a configurable clock gate that can be configured with the appropriate clock gate register

7.3.4.2.2 PRU-ICSS Protect

Write protect block allows software to enable safety protection to prevent corruption of key configuration and debug registers and the Instruction memories (IRAM's) of all PRU cores (PRU0/PRU1). Write protection is also supported for Data RAM0 and Data RAM1.

This is achived by blocking the byte enables during a write transaction if enabled. When enabled, it will prevent any unwanted write transaction to these elements. To Enable/Disable this feature, software will first need to unlock the write access to this block through the PROT_UNLOCK_KEY register then configure the protection through PROT_CFG register and relock.

7.3.4.2.3 Module Clock Configurations at PRU-ICSS Top Level

IEP functional clock source selection: The clock source selection to the IEP module is done in register CTRLMMR_PRU_ICSS_CLKSEL[19-16] IEP_CLKSEL (where n = 0 or 1) in the CTRL_MMR0 location. For more information on these PRU-ICSS level input clocks, refer to the *PRU-ICSS Integration*.

Enhanced GPIO clock divider settings: In certain sample/shift clock settings of the PRU0 and PRU1 EGPIOs (when enabled in serial mode) two cascaded fractional dividers are done in the PRU_ICSS_CFG top level configuration registers PRU_ICSS_GPCFG0 and PRU_ICSS_GPCFG1. In addition, EGPIO clock active edge selection control can be exerted via the bit PRU0_GPI_CLK_MODE for PRU0 EGPIO and PRU1_GPI_CLK_MODE for the PRU1 EGPIO.

- For the serial PRU0's EGPOs:
  - PRU_ICSSM_GPCFG0_REG[24-20] PRU0_GPO_DIV1
  - PRU_ICSSM_GPCFG0_REG[19-15] PRU0_GPO_DIV0
- For the serial PRU0's EGPIs:
  - PRU_ICSSM_GPCFG0_REG[12-8] PRU0_GPI_DIV1
  - PRU_ICSSM_GPCFG0_REG[7-3] PRU0_GPI_DIV0
- For the serial PRU1's EGPOs:
  - PRU_ICSSM_GPCFG1_REG[24-20] PRU1_GPO_DIV1
  - PRU_ICSSM_GPCFG1_REG[19-15] PRU1_GPO_DIV0
- For the serial PRU1's EGPIs:
  - PRU_ICSSM_GPCFG1_REG[12-8] PRU1_GPI_DIV1
  - PRU_ICSSM_GPCFG1_REG[7-3] PRU1_GPI_DIV0

7.3.4.3 Other PRU-ICSS Module Functional Registers at Subsystem Level

Enhanced GPIO. The other functional mode setting for PRUs EGPIOs at PRU-ICSS top registers level are:

- PRU_ICSSM_GPCFG0 / PRU_ICSSM_GPCFG1[14] PRU0_GPO_MODE (PRU0 or PRU1) — to select between direct or serial EGPO output mode of operation.
- PRU_ICSSM_GPCFG0 / PRU_ICSSM_GPCFG1[25] PRU0_GPO_SH_SEL (PRU0 or PRU1) — to select between the EGPO shadow registers 0 and 1 used for output shifting. For more details, refer to the Section 7.3.5.2.2.3.4, *Enhanced General-Purpose Module Outputs (R30)*.
- PRU_ICSSM_GPCFG0 / PRU_ICSSM_GPCFG1[1-0] PRU0_GPI_MODE (PRU0 or PRU1) — selects the EGPI input mode of operation (selects between "direct input", "parallel capture", "28-bit shift" or "MII_RT" modes).
- PRU_ICSSM_GPCFG0 / PRU_ICSSM_GPCFG1[13] PRU0_GPI_SB (PRU0 or PRU1) — start bit event status for 28-bit EGPI input shift mode. For more details, refer to the Section 7.3.5.2.2.3, *Enhanced General-Purpose Module Inputs (R31)*.

PRUs scratchpad (SPAD) memory priority and configuration related bits are located in the PRU_ICSSM_SPP register.

7.3.4.4 PRU-ICSS Memory Maps

The PRU-ICSS comprises various distinct addressable regions that are mapped to both a local and global memory map. The local memory maps are maps with respect to the PRU point of view. The global memory maps are maps with respect to the Host point of view, but can also be accessed by the PRU-ICSS. Each PRU-ICSS can also access the memories within the other PRU-ICSS subsystem without going through an external port, thanks to PRU-ICSS VBUSP expansion port.

7.3.4.4.1 PRU-ICSS Local Memory Map

The PRU-ICSS memory map is documented in Table 7-38 (Instruction Space) and in Table 7-39 (Data Space). Note that these two memory maps are implemented inside the PRU-ICSS and are local to the components of the PRU-ICSS.

7.3.4.4.1.1 PRU-ICSS Local Instruction Memory Map

Each PRU (PRU0 and PRU1) has a dedicated 16KB of Instruction Memory which needs to be initialized by an external to PRU-ICSS Host processor before a PRU core executes any instructions.

CAUTION

The PRU-ICSS PRU0/1_IRAM regions are ONLY accessible from controllers, external to the PRU-ICSS (like Arm) when the PRU0/PRU1 is NOT running. The access is via PRU-ICSStarget port on the device CBASS0 interconnect.

Table 7-38. PRU-ICSS Local Instruction Memory Map

Start Address	PRU0	PRU1
0000 0000h	16KB IRAM	16KB IRAM

7.3.4.4.1.2 PRU-ICSS Local Data Memory Map

The local data memory map in Table 7-39 allows each PRU core to access the PRU-ICSS addressable regions (both its own subsystem and the other subsystem) and the external host's memory map.

Table 7-39. PRU-ICSS Local Data Memory Map

Start Address	PRU0	PRU1
0000 0000h	Data 8KB RAM0	Data 8KB RAM1
0000 2000h	Data 8KB RAM1	Data 8KB RAM0
0000 8000h	RAT_SLICE0	RAT_SLICE0
0000 9000h	RAT_SLICE1	RAT_SLICE1
0001 0000h	Data 32 KB RAM2 (Shared RAM)	Data 32 KB RAM2 (Shared RAM)
0002 0000h	INTC	INTC
0002 2000h	PRU0 Control	PRU0 Control
0002 4000h	PRU1 Control	PRU1 Control
0002 4C00h	PROTECT	PROTECT
0002 6000h	CFG	CFG
0002 8000h	UART0	UART0
0002 E000h	IEP0	IEP0
0003 0000h	ECAP0	ECAP0
0003 2000h	MII_RT_CFG	MII_RT_CFG
0003 2400h	MII_MDIO	MII_MDIO
0003 3000h	MII_G_RT_CFG	MII_G_RT_CFG

7.3.4.4.2 PRU-ICSS Global Memory Map

The global view of the PRU-ICSS internal memories and control ports is shown in Table 7-40. The offset addresses of each region are implemented inside the PRU-ICSS but the global device memory mapping places the PRU-ICSS target port in the address range shown in the external PRU-ICSS Host top-level memory map.

The global memory map is with respect to the Host point of view (that is, device Arm), but the memory space can also be accessed by the PRU-ICSS itself. Note that PRU0 and PRU1 can use either the local or global addresses to access their internal memories, but using the local addresses provides access time several cycles faster than using the global addresses. This is because when accessing via the global address the access has to be routed through the CBASS0 switch fabric outside PRU-ICSS and back in through the PRU-ICSS target port.

Each of the PRU cores can access the rest of the device memory (including memory mapped peripheral and configuration registers) using the global memory space addresses.

Table 7-40. PRU-ICSS Global Memory Map

Offset Address	PRU-ICSS Target
0000 0000h	8KB Data RAM0
0000 2000h	8KB Data RAM1
0000 8000h	RAT_SLICE0
0000 9000h	RAT_SLICE1
0001 0000h	32 KB Data RAM2 (Shared Memory)
0002 0000h	PRU-ICSS INTC
0002 2000h	PRU0 Control
0002 2400h	PRU0 Debug
0002 4000h	PRU1 Control
0002 4400h	PRU1 Debug
0002 4C00h	PROTECT
0002 6000h	PRU-ICSS CFG
0002 8000h	PRU-ICSS UART0
0002 E000h	IEP0
0002 F000h	Reserved
0003 0000h	ECAP0
0003 2000h	MII_RT_CFG
0003 2400h	MII_MDIO
0003 4000h	PRU0 16 KB IRAM
0003 8000h	PRU1 16 KB IRAM

7.3.5 PRU-ICSS PRU Cores

This section describes the functionality of the two Programmable Real-time Unit (PRU) processors (PRU0 and PRU1), integrated in the device PRU-ICSS.

7.3.5.1 PRU Cores Overview

The PRU is a processor optimized for performing embedded tasks that require manipulation of packed memory mapped data structures, handling of system events that have tight real-time constraints and interfacing with systems external to the SoC. The PRU is both very small and very efficient at handling such tasks.

The major attributes of the PRU are shown in Table 7-41.

Table 7-41. PRU Features

Attribute	Value
IO Architecture	Load/Store
Data Flow Architecture	Register to Register
Core Level Bus Architecture	
Type	4-Bus Harvard (1 Instruction, 3 Data)
Instruction I/F	32-Bit Modified VBUSP Controller
Memory I/F 0	32-Bit VBUSP Controller
Memory I/F 1	32-Bit VBUSP Controller
Execution Model	
Issue Type	Scalar
Pipelining	None (Purposefully)
Ordering	In Order
ALU Type	Unsigned Integer
Registers	
General Purpose (GP)	30 (R1 – R30)
External Status	0 (R31)
GP/Indexing	0 (R0)
Addressability in Instruction	Bit, Byte (8-bit), Half-word (16-bit), Word (32-bit), Pointer
Addressing Modes	
Load Immediate	16-bit Immediate
Load/Store – Memory	Register Base + Register Offset Register Base + 8-bit Immediate Offset Register Base with auto increment/decrement Constant Table Base + Register Offset Constant Table Base + 8-bit Immediate Offset Constant Table Base with auto increment/ decrement
Data Path Width	32-bit
Instruction Width	32-bit
Accessibility to Internal PRU Structures	Provides 32-bit VBUSP target with three regions: • Instruction RAM • Control/Status registers • Debug access to internal registers (R0-R31) and constant table

The processor is based on a four-bus architecture which allows instructions to be fetched and executed concurrently with data transfers. In addition, an input is provided in order to allow external status information to be reflected in the internal processor status register. Figure 7-26 shows a block diagram of the processing element and the associated instruction RAM/ROM that contains the code that is to be executed.

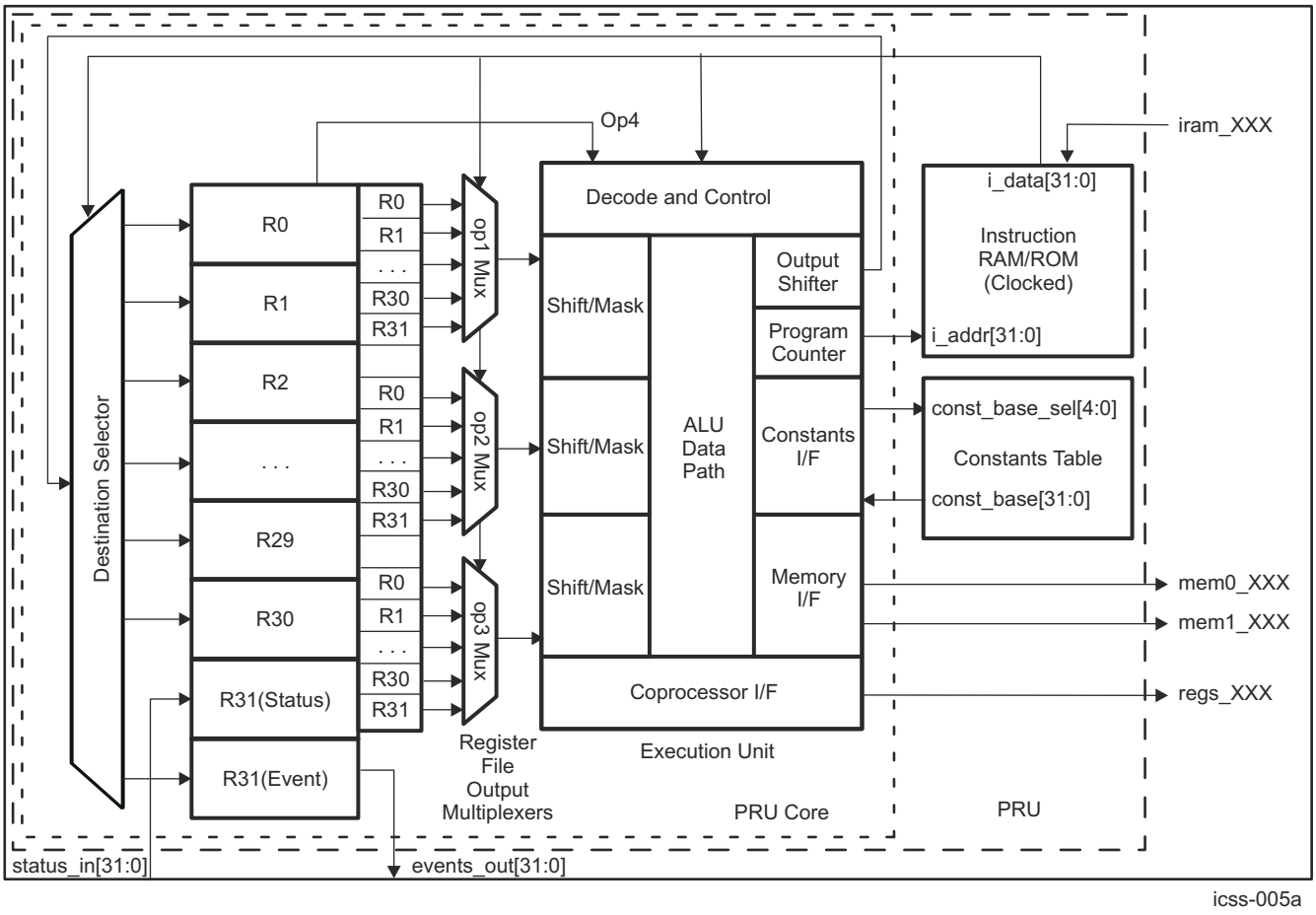


Figure 7-26. PRU Block Diagram

7.3.5.2 PRU Cores Functional Description

This section describes the PRU cores supported functionality by describing the constant table, module interface and enhanced GPIOs.

7.3.5.2.1 PRU Constant Table

The PRU Constants Table is a structure of hard-coded memory addresses for commonly used peripherals and memories. The constants table is used for more efficiently load/store data to these commonly accessed addresses by:

- Eliminating the PRU instruction that pre-loads a hard-coded address into the internal register file.
- Maximizing the usage of the PRU register file for embedded processing applications by moving many of the commonly used constant or deterministically calculated base addresses from the internal register file to an external table.

Table 7-42. PRU0/1 Constant Table

Entry No.	Region Pointed To	Value [31:0]
0	PRU-ICSS INTC (local)	0002_0000h
1	PRU-ICSS IEP (local)	0002_F000h
2	PRU-ICSS IEP_0x100 (local)	0002_F100h
3	PRU-ICSS ECAP0 (local)	0003_0000h
4	PRU-ICSS CFG (local)	0002_6000h
5	PRU-ICSS CFG_0x100 (local)	0002_6100h
6	PRU-ICSS INTC_0x200 (local)	0002_0200h
7	PRU-ICSS UART0 (local)	0002_8000h
8	PRU-ICSS IEP0_0x100 (local)	0002_E100h

Table 7-42. PRU0/1 Constant Table (continued)

Entry No.	Region Pointed To	Value [31:0]
9	PRU-ICSS CFG (local)	0003_3000h
10	RESERVED	RSERVED
11	PRU-ICSS PRU0 Control (local)	0002_2000h PRU0
	RESERVED	RESERVED
	RESERVED	RESERVED
	PRU-ICSS PRU1 Control (local)	0002_4000h PRU1
12	RESERVED	RESERVED
13	RESERVED	RESERVED
14	RESERVED	RESERVED
15	RESERVED	6000_0000h
16	RESERVED	7000_0000h
17	RESERVED	8000_0000h
18	RESERVED	9000_0000h
19	RESERVED	A000_0000h
20	RESERVED	B000_0000h
21	MDIO (local)	0003_2400h
22	RAT SLICE0 (local)	0000_8000h PRU0
	RAT SLICE1 (local)	0000_9000h PRU1
23	Reserved	C000_0000h
24	PRU-ICSS PRU0/PRU1 Data RAM (local)	0000_0n00h, n = c24_blk_index[3:0]
25	PRU-ICSS PRU1/PRU0 Data RAM (local)	0000_2n00h, n = c25_blk_index[3:0]
26	PRU-ICSS IEP (local)	0002_En00h, n = c26_blk_index[3:0]
27	PRU-ICSS MII_RT/SGMII0_CFG/SGMII1_CFG (local)	0003_2n00h, n = c27_blk_index[3:0]
28	PRU-ICSS Shared RAM (local)	00nn_nn00h, nnnn = c28_pointer[15:0]
29	RESERVED	0Dnn_nn00h, nnnn = c29_pointer[15:0]
30	RESERVED	0Enn_nn00h, nnnn = c30_pointer[15:0]
31	RESERVED	0Fnn_nn00h, nnnn = c31_pointer[15:0]

Note

The addresses in constants entries 24–31 are partially programmable. Their programmable bit field (for example, c24_blk_index[3:0]) is programmable through the PRU CTRL register space. As a general rule, the PRU should configure this field before using the partially programmable constant entries.

7.3.5.2.2 PRU Module Interface

The PRU module interface consists of the PRU internal registers 30 and 31 (R30 and R31). Figure 7-27 shows the PRU module interface and the functionality of R30 and R31. The register R31 serves as an interface with the dedicated PRU general purpose input (GPI) pins and PRU-ICSS INTC. Reading R31 returns status information from the GPI pins and PRU-ICSS INTC via the PRU Real Time Status Interface. Writing to R31 generates PRU system events via the PRU Event Interface. The register R30 serves as an interface with the dedicated PRU general purpose output (GPO) pins.

Note

The below sections cover different functional modes of the PRUn cores, (where n=0,1), enhanced GPIO (EGPIO) interface. The register bits which control EGPIO functionalities are part of the (PRU-ICSS CFG) space. For descriptions of these EGPIO register bitfield controls, refer to the Section 7.3.4.3.

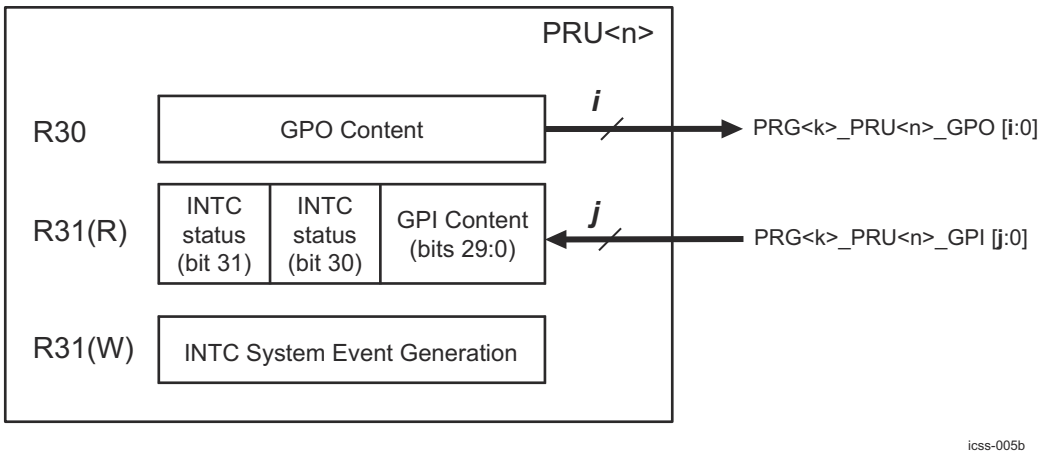


Figure 7-27. PRU Module Interface

7.3.5.2.2.1 Real-Time Status Interface Mapping (R31): Interrupt Events Input

The PRU Real Time Status Interface directly feeds information into register 31 (R31) of the PRU's internal register file. The firmware on the PRU uses the status information to make decisions during execution. The status interface is comprised of signals from different modules inside of the PRU-ICSS which require some level of interaction with the PRU. More details on the Host interrupts imported into bit 30 and 31 of register R31 of both the PRUs is provided in the , *PRU-ICSS Local Interrupt Controller*.

Table 7-43. Real-Time Status Interface Mapping (R31) Field Descriptions

Bit	Field	Description
31	pru_intr_in[1]	PRU Host Interrupt 1 from local PRU-ICSS INTC
30	pru_intr_in[0]	PRU Host Interrupt 0 from local PRU-ICSS INTC
29-0	pru<n>_r31_status[29:0]	Status inputs from primary input via Enhanced GPI port

7.3.5.2.2.2 Event Interface Mapping (R31): PRU System Events

This PRU Event Interface directly feeds pulsed event information out of the PRU's internal ALU. These events are exported out of the PRU-ICSS and need to be connected to the system interrupt controller at the SoC level. The event interface can be used by the firmware to create software interrupts from the PRU to the Host processor.

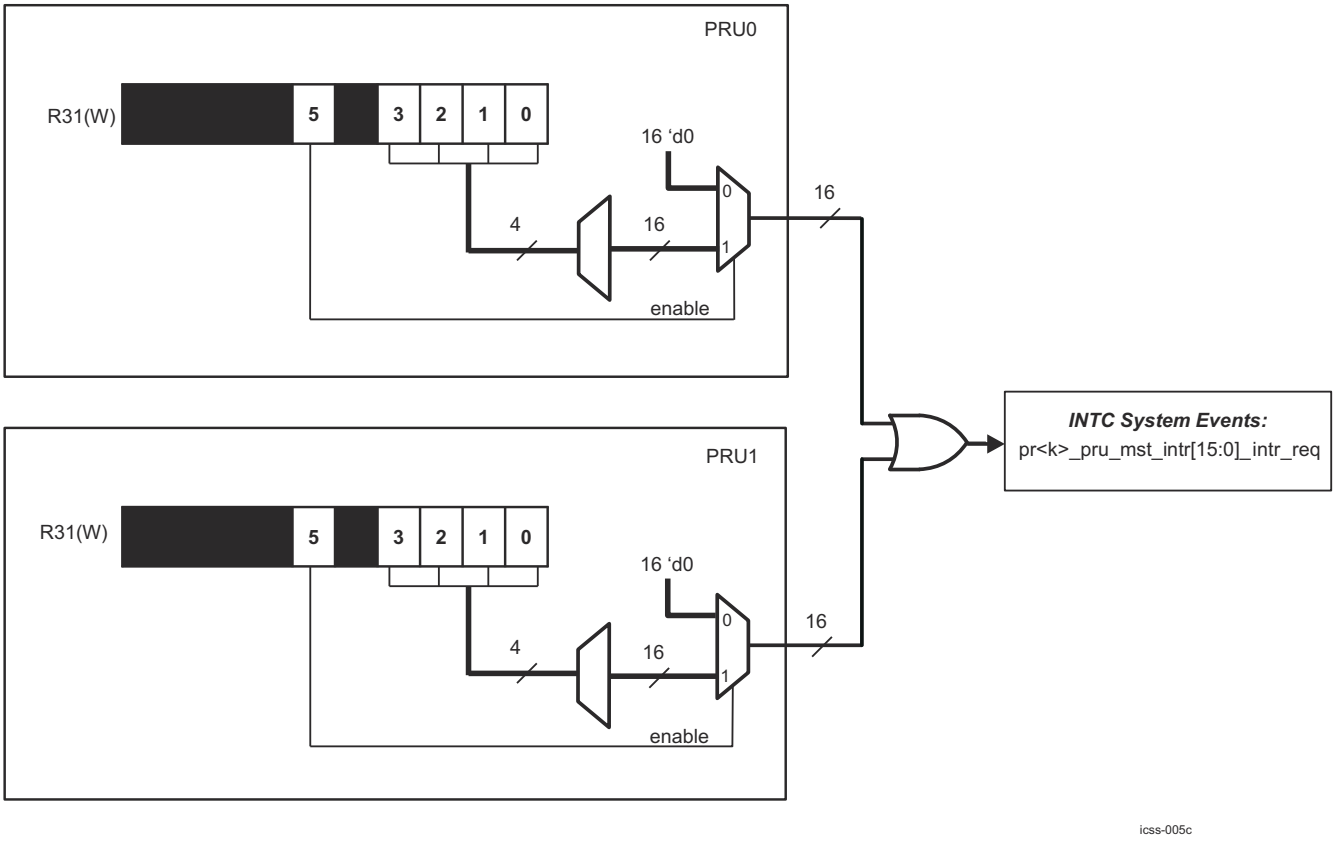


Figure 7-28. Event Interface Mapping (R31)

Table 7-44. Event Interface Mapping (R31) Field Descriptions

Bit	Field	Description
31-6	Reserved	
5	pru<n>_r31_vec_valid	Valid strobe for vector output
4	Reserved	
3-0	pru<n>_r31_vec[3:0]	Vector output

Simultaneously writing a '1' to pru<n>_r31_vec_valid (R31 bit 5) and a channel number from 0 to 15 to pru<n>_r31_vec[3:0] (R31 bits 3-0) creates a pulse on the output of the corresponding pr<k>_pru_mst_intr[x]_intr_req INTC system event. For example, writing '100000' will generate a pulse on prk_pru_mst_intr[0]_intr_req, writing '100001' will generate a pulse on prk_pru_mst_intr[1]_intr_req, and so on to where writing '101111' will generate a pulse on prk_pru_mst_intr[15]_intr_req and writing '0xxxxx' will not generate any system event pulses. The output values from both PRU cores in a subsystem are ORed together.

The output channels 0-15 are connected to the INTC system events 16-31, respectively. This allows the PRU to assert one of the system events 16-31 by writing to its own R31 register. The system event is used to either post a completion event to one of the host CPUs (Arm) or to signal the other PRU. The host to be signaled is determined by the system interrupt to interrupt channel mapping (programmable). The 16 events are named as prk_pru_mst_intr<15:0>_intr_req. See the *PRU-ICSS Interrupt Requests Mapping,* in the section*PRU-ICSS Local Interrupt Controller*, for more details.

7.3.5.2.2.3 General-Purpose Inputs (R31): Enhanced PRU GP Module

The PRU-ICSS implements an enhanced General Purpose Input/Output (GPIO) module with SCU that supports the following general-purpose input modes: direct input, 16-bit parallel capture, 28-bit serial shift in, and MII_RT. Register R31 serves as an interface with the general-purpose inputs. Table 7-45 describes the input modes in detail.

Note

Each PRU core can only be configured for one GPI mode at a time. Each mode uses the same R31 signals and internal register bits for different purposes. A summary is found in Table 7-46.

Note

The PRU_ICSSM_GPCFG0 register, bitfield [29-26] PR1_PRU0_GP_MUX_SEL (PRU0 or PRU1) in the PRU-ICSS CFG register space needs to be set to 0h for GP mode. For a given PRU core, the following IO modes are mutually exclusive: GP mode, Sigma Delta mode, and 3 channel Peripheral I/F mode.

Table 7-45. PRU R31 (GPI) Modes

Mode	Function	Configuration
Direct input	GPI[19:0] feeds directly into the PRU R31	Default state
16-bit parallel capture	DATAIN[0:15] is captured by the posedge or negedge of CLOCKIN	• Enabled by PRU_ICSSM_GPCFG0 register (PRU0 or PRU1) • CLOCKIN edge selected by PRU_ICSSM_GPCFG0 register
28-bit shift in	DATAIN is sampled and shifted into a 28-bit shift register. • Shift Counter (Cnt_16) feature is mapped to pru<n>_r31_status[28]. • SB (Start Bit detection) feature is mapped to pru<n>_r31_status[29]	• Enabled and disabled by PRU_ICSSM_GPECFG0 register (PRU0 or PRU1) • Cnt_16 is self clearing and is connected to the PRU INTC • Start Bit (SB) is cleared by PRU_ICSSM_GPECFG0 register • Start Bit value (0h or 1h) selected by PRU_ICSSM_GPECFG0 register
PERIF	The 3 channel Peripheral Interface supports functionality for operations utilized the EnDat 2.2 and BiSS protocols.	Enabled by PRU_ICSSM_GPCFG0[1-0] PRU0_GPI_MODE register (value: 1h), where n = 0 or 1
MII_RT	mii_rt_r31_status [29:0] internally driven by the MII_RT module	Enabled by PRU_ICSSM_GPCFG0[1-0] PRUn_GPI_MODE register (value: 2h), where n = 0 or 1
Sigma Delta	Up to nine channels of concurrent counting with clock source configuration for each channel.	Enabled by PRU_ICSSM_GPCFG0[1-0] PRUn_GPI_MODE register (value: 3h), where n = 0 or 1

Table 7-46. PRU GPI Signals and Configurations

Pad Names at Device Level[1]	GPI Modes					
	Direct input	Parallel Capture	28-Bit Shift in	PERIF	MII	Sigma Delta
PR<k>_PRU<n>_GPI0	GPI0	DATAIN0	DATAIN		RXD[0]	SD0_CLK
PR<k>_PRU<n>_GPI1	GPI1	DATAIN1			RXD[1]	SD0_DATA
PR<k>_PRU<n>_GPI2	GPI2	DATAIN2			RXD[3]	SD1_CLK
PR<k>_PRU<n>_GPI3	GPI3	DATAIN3			RXDV	SD1_DATA
PR<k>_PRU<n>_GPI4	GPI4	DATAIN4			RXER	SD2_CLK
PR<k>_PRU<n>_GPI5	GPI5	DATAIN5			RX_CLK	SD2_DATA
PR<k>_PRU<n>_GPI6	GPI6	DATAIN6				SD3_CLK
PR<k>_PRU<n>_GPI7	GPI7	DATAIN7			RXLINK	SD3_DATA
PR<k>_PRU<n>_GPI8	GPI8	DATAIN8			COL	SD4_CLK
PR<k>_PRU<n>_GPI9	GPI9	DATAIN9		PERIF0_IN	CRS	SD4_DATA
PR<k>_PRU<n>_GPI10	GPI10	DATAIN10		PERIF1_IN		SD5_CLK
PR<k>_PRU<n>_GPI11	GPI11	DATAIN11		PERIF2_IN		SD5_DATA

Table 7-46. PRU GPI Signals and Configurations (continued)

Pad Names at Device Level[1]	GPI Modes					
	Direct input	Parallel Capture	28-Bit Shift in	PERIF	MII	Sigma Delta
PR<k>_PRU<n>_GPI12	GPI12	DATAIN12				SD6_CLK
PR<k>_PRU<n>_GPI13	GPI13	DATAIN13				SD6_DATA
PR<k>_PRU<n>_GPI14	GPI14	DATAIN14				SD7_CLK
PR<k>_PRU<n>_GPI15	GPI15	DATAIN15				SD7_DATA
PR<k>_PRU<n>_GPI16	GPI16	CLOCKIN		R31_IN[16]	TX_CLK,R31_IN[16]	SD8_CLK, R31_IN[16]
PR<k>_PRU<n>_GPI17	GPI17					SD8_DATA
PR<k>_PRU<n>_GPI18	GPI18					
PR<k>_PRU<n>_GPI19	GPI19					

(1) These pins are also used for Sigma Delta or Peripheral I/F mode.

7.3.5.2.2.3.1 PRU EGPIs Direct Input

The pru<n>_r31_status[0:19] bits of the internal PRU register file are mapped to device-level, general purpose input pins (PRU0_GPI[0:19]). In GPI Direct Input mode, PRU0_GPI[0:19] feeds directly to pru<n>_r31_status[0:19].

Each PRU of the PRU-ICSS has a separate mapping to device input signals - PRn_PRU0_GPI[19:0] for the PRU0 core and PRn_PRU1_GPI[19:0] for the PRU1 core. There are 40general purpose inputs in total. For more details, refer also to the *PRUSS-M Environment*. See the device's system reference guide or datasheet for device specific pin mapping.

Note

The following PRU IO are not pinned out at the device level: PR0_PRU0_GPIO[7,17,18,19]

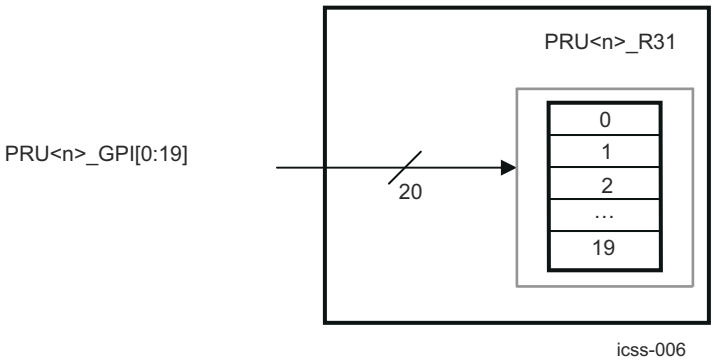


Figure 7-29. PRU R31 (EGPI) Direct Input Mode Block Diagram

7.3.5.2.2.3.2 PRU EGPIs 16-Bit Parallel Capture

The pru<n>_r31_status[0:15] and pru<n>_r31_status[16] bits of the internal PRU register file mapped to device-level, general purpose input pins (PRU0_DATAIN [0:15] and PRU0_CLOCKIN, respectively). PRU0_CLOCKIN is designated for an external strobe clock, and is used to capture PRU0_DATAIN [0:15].

The PRU<n>_DATAIN can be captured either by the positive or the negative edge of PRU<n>_CLOCK, programmable through the PRU-ICSS CFG register space. If the clocking is configured through the PRU-ICSS

CFG register to be positive, then it will equal PRU<n>_CLOCK; however, if the clocking is configured to be negative, then it will equal PRU<n>_CLOCK inverted.

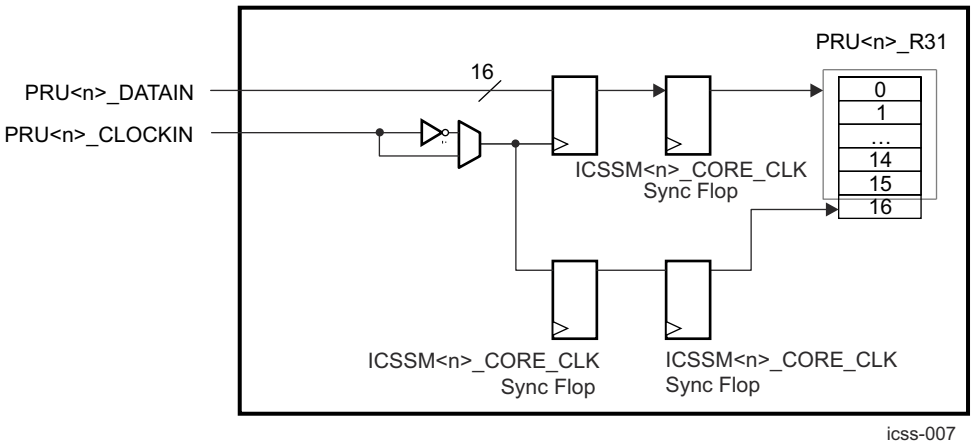


Figure 7-30. PRU R31 (EGPI) 16-Bit Parallel Capture Mode Block Diagram

7.3.5.2.2.3.3 PRU EGPIs 28-Bit Shift In

In 28-bit shift in mode, the device-level, general-purpose input pin PRU<n>_DATAIN is sampled and shifted into a 28-bit shift register on an internal clock pulse. The register fills in LSB order (from bit 0 to 27) and then overflows into a bit bucket. The 28-bit register is mapped to pru<n>_r31_status[0:27] and can be cleared in software through the PRU_ICSS_GPCFG0[13] PRU0_GPI_SB register (PRU0 or PRU1).

Note that by default, the PRU will continually capture and shift the DATAIN input when the GPI mode has been set. However, clearing the PRU_ICSS_GPCFG0[1] PRU0_GPI_SHIFT_EN bit will freeze the shift operation.

The shift rate is controlled by the effective divisor of two cascaded dividers applied to the PRU-ICSS<n>_CORE_CLK clock (200MHz). These cascaded dividers can each be configured through the PRU-ICSS CFG register space to a value of {1, 1.5, …, 16}. Table 7-47 shows sample effective clock values and the divisor values that can be used to generate these clocks.

Table 7-47. PRU EGPIs Effective Clock Values

Generated clock	PRU0_GPI_DIV0	PRU0_GPI_DIV1
8-MHz	12.5 (17h)	2 (02h)
10-MHz	10 (12h)	2 (02h)
16-MHz	16 (1Eh)	1 (00h)
20-MHz	10 (12h)	1 (00h)

The 28-bit shift mode also supports the following features:

- SB (Start Bit detection) is mapped to pru<n>_r31_status[29] and is set when the first 1 (default) or 0 is captured on PRU<n>_DATAIN. The Start Bit value (1 or 0) is configured through the PRU_ICSS_GPECFG0[0] PRU0_GPI_SB_P bit (PRU0 or PRU1). The SB flag in pru<n>_r31_status[29] is cleared in software through the PRU-ICSS CFG register space.
- Cnt_16 (Shift Counter) is mapped to pru<n>_r31_status[28] and is set on every 16 shift clock samples after the Start Bit has been received. CNT_16 is self clearing and is connected to the local PRU-ICSS INTC. See the *PRU-ICSS Local Interrupt Controller* for more details.
- The PRU_ICSS_GPECFG0[1] PRU0_GPI_SHIFT_EN bit can stop or freeze the current shift operation and disable the search for a new Start Bit, if an SB event has not occurred.

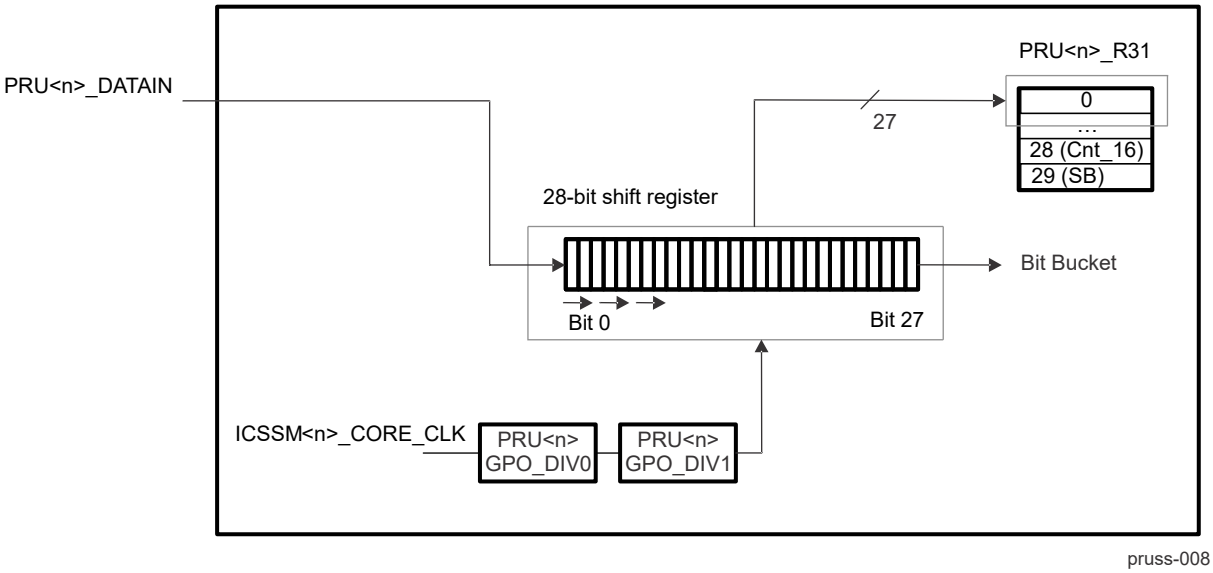


Figure 7-31. PRU R31 (EGPI) 28-Bit Shift Mode

7.3.5.2.2.3.3.1 PRU EGPI Programming Model

Follow this steps to configure the PRU EGPI in 28-bit shift input mode:

1. Clear PRU_ICSS_GPECFG0[1] PRU0_GPI_SHIFT_EN bit (PRU0 or PRU1)
2. Clear/Set PRU_ICSS_GPECFG0[0] PRU0_GPI_SB_P bit (PRU0 or PRU1)
3. Clear Start Bit by writing 1h to PRU_ICSS_GPCFG0[13] PRU0_GPI_SB bit
4. Program the dividers through:
   PRU_ICSS_GPCFG0[24-20] PRU0_GPO_DIV1 bit (PRU0 or PRU1)
   PRU_ICSS_GPCFG0[19-15] PRU0_GPO_DIV0 bit (PRU0 or PRU1)
5. Enable Shift Input Mode by writing to PRU_ICSS_GPECFG0[1] PRU0_GPI_SHIFT_EN bit

7.3.5.2.2.3.4 General-Purpose Outputs (R30): Enhanced PRU GP Module

The PRU-ICSS implements an enhanced General Purpose Input/Output (GPIO) module that supports two general-purpose output modes: direct output and shift out.

Table 7-48 describes these modes in detail.

Note

Each PRU core can only be configured for one GPO mode at a time. Each mode uses the same R30 signals and internal register bits for different purposes. A summary is found in Table 7-48.

Note

The PRU_ICSS_GPCFG0 register, bitfield [29-26] PR1_PRU0_GP_MUX_SEL (PRU0 or PRU1) in the PRU-ICSS CFG register space needs to be set to 0h for GP mode. For a given PRU core, the following IO modes are mutually exclusive: GP mode, Sigma Delta mode, and 3 channel Peripheral I/F mode.

Table 7-48. PRU R30 (EGPO) Output Mode

Mode	Function	Configuration
Direct output	pru<n>_r30[19:0] feeds directly to GPO[19:0]	Default state

Table 7-48. PRU R30 (EGPO) Output Mode (continued)

Mode	Function	Configuration
Shift out	• pru<n>_r30[0] is shifted out on DATAOUT on every rising edge of pru<n>_r30[1] (CLOCKOUT). • LOAD_GPO_SH0 (Load Shadow Register 0) is mapped to pru<n>_r30[29]. • LOAD_GPO_SH1 (Load Shadow Register 1) is mapped to pru<n>_r30[30]. • ENABLE_SHIFT is mapped to pru<n>_r30[31].	Enabled by PRU_ICSS_GPCFG0 register (PRU0 or PRU1) Free Running Clock or Fixed Clock Count Mode selected by PRU_ICSS_GPECFG0 register.

Table 7-49. GPO Mode Descriptions

Pad Names at Device Level (1)	GPO Modes	
	Direct output	Shift out
PR<k>_PRU<n>_GPO0	GPO0	DATAOUT
PR<k>_PRU<n>_GPO1	GPO1	CLOCKOUT
PR<k>_PRU<n>_GPO2	GPO2	
PR<k>_PRU<n>_GPO3	GPO3	
PR<k>_PRU<n>_GPO4	GPO4	
PR<k>_PRU<n>_GPO5	GPO5	
PR<k>_PRU<n>_GPO6	GPO6	
PR<k>_PRU<n>_GPO7	GPO7	
PR<k>_PRU<n>_GPO8	GPO8	
PR<k>_PRU<n>_GPO9	GPO9	
PR<k>_PRU<n>_GPO10	GPO10	
PR<k>_PRU<n>_GPO11	GPO11	
PR<k>_PRU<n>_GPO12	GPO12	
PR<k>_PRU<n>_GPO13	GPO13	
PR<k>_PRU<n>_GPO14	GPO14	
PR<k>_PRU<n>_GPO15	GPO15	
PR<k>_PRU<n>_GPO16	GPO16	
PR<k>_PRU<n>_GPO17	GPO17	
PR<k>_PRU<n>_GPO18	GPO18	
PR<k>_PRU<n>_GPO19	GPO19	

(1) These pins are also used for Sigma Delta or Peripheral I/F mode.

7.3.5.2.2.3.4.1 PRU EGPOs Direct Output

The PRU0_r30 [19:0] bits of the internal PRU register files are mapped to device-level, general-purpose output pins (PRU0_GPO[0:19]). In GPO Direct Output mode, PRU0_r30[0:19] feed directly to PRU0_GPO[0:19]. Each PRU of the PRU-ICSS has a separate mapping to pins, so that there are 40 total general-purpose outputs from the PRU-ICSS. See the device's system reference guide or datasheet for device-specific pin mapping.

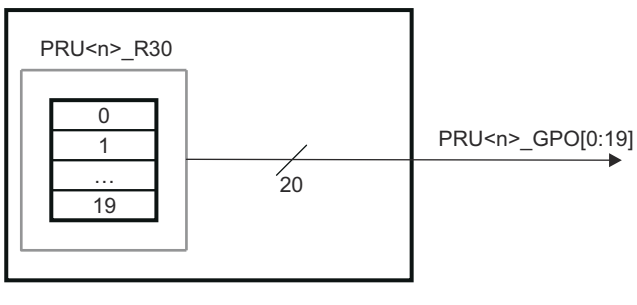


Figure 7-32. PRU R30 (EGPO) Direct Output Mode Block Diagram

7.3.5.2.2.3.4.2 PRU EGPO Shift Out

In shift out mode, data is shifted out of PRU0_r30[0] (PRU0_DATAOUT) on every rising edge of PRU0_r30[1] (PRU0_CLOCK). The shift rate is controlled by the effective divisor of two cascaded dividers applied to the PRU-ICSS<n>_CORE_CLK clock (200MHz). These cascaded dividers can each be configured through the PRU-ICSS CFG register space to a value of {1, 1.5, …, 16}. Table 7-50 shows sample effective clock values and the divisor values that can be used to generate these clocks. Note that shift out mode supports two clocking submodes - Free Running Clock Mode (default) and Fixed Clock Count Mode. The clocking submode is selected through PRU_ICSS_GPECFG0[5] PRU0_GPO_SHIFT_CLK_FREE. In Free Running Clock Mode, PRU0_CLOCKOUT is a free running clock that starts when the PRU GPO mode is set to shift out mode.

Table 7-50. Effective Clock Values

Generated Clock	PRU0_GPO_DIV0	PRU0_GPO_DIV1
8 MHz	12.5 (17h)	2 (02h)
10 MHz	10 (12h)	2 (02h)
16 MHz	16 (1Eh)	1 (00h)
20 MHz	10 (12h)	1 (00h)

Shift out mode uses two 16-bit shadow registers (GPO_SH0 and GPO_SH1) to support ping-pong buffers. Each shadow register has independent load controls programmable through PRU0_r30[29:30] (PRU0_LOAD_GPO_SH[0:1]). While PRU0_LOAD_GPO_SH[0:1] is set, the contents of PRU<n>_R30[0:15] are loaded into GPO_SH0 and GPO_SH1 shadow registers.

The data shift will start from the LSB or MSB of GPO_SH0 when PRU<n>_R30[31] (PRU0_ENABLE_SHIFT) is set. The LSB or MSB setting is configurable through PRU_ICSS_GPECFG0[4] PRU0_GPO_SHIFT_SWAP. Note that if no new data is loaded into GPO_SH0/GPO_SH1 after shift operation, the shift operation will continue looping and shifting out the pre-loaded data.

For Free Running Clock Mode, the shift operation will continue until PRU0_ENABLE_SHIFT is cleared. When PRU0_ENABLE_SHIFT is cleared, the shift operation will finish shifting out the current shadow register, stop, and then reset.

For Fixed Clock Count Mode, the number of data bits to be shifted out is defined by PRU_ICSS_GPECFG0[15-8] PRU0_GPO_SHIFT_CNT. PRU<n>_CLOCKOUT will stop either high or low with the last data bit. The last data bit will remain persistent. However, the clock stop state is configurable through PRU_ICSS_GPECFG0[16] PRU0_GPO_SHIFT_CLK_HIGH.

The source of PR<k>_PRU<n>_GPO[2:15] is configurable by PRU_ICSS_GPECFG0[6] PRU0_GPO_SHIFT_GP_EN. By default, if any device-level pins mapped to PRU<n>_R30[2-15] are configured for the PR<k>_PRU<n>_GPO[2:15] pinmux mode, then these pins will reflect the shadow register value written to PRU<n>_R30. Any pin configured for a different pinmux setting will not reflect the shadow register value written to PRU<n>_R30. However, setting PRU_ICSS_GPECFG0[6] PRU0_GPO_SHIFT_GP_EN = 1h allows PRU<n>_R30[2:15] to be controlled by PRU<n>_R30_SHADOW[2-15], which is updated by PRU<n>_R30[2:15] when PRU<n>_R30[28] = 1h.

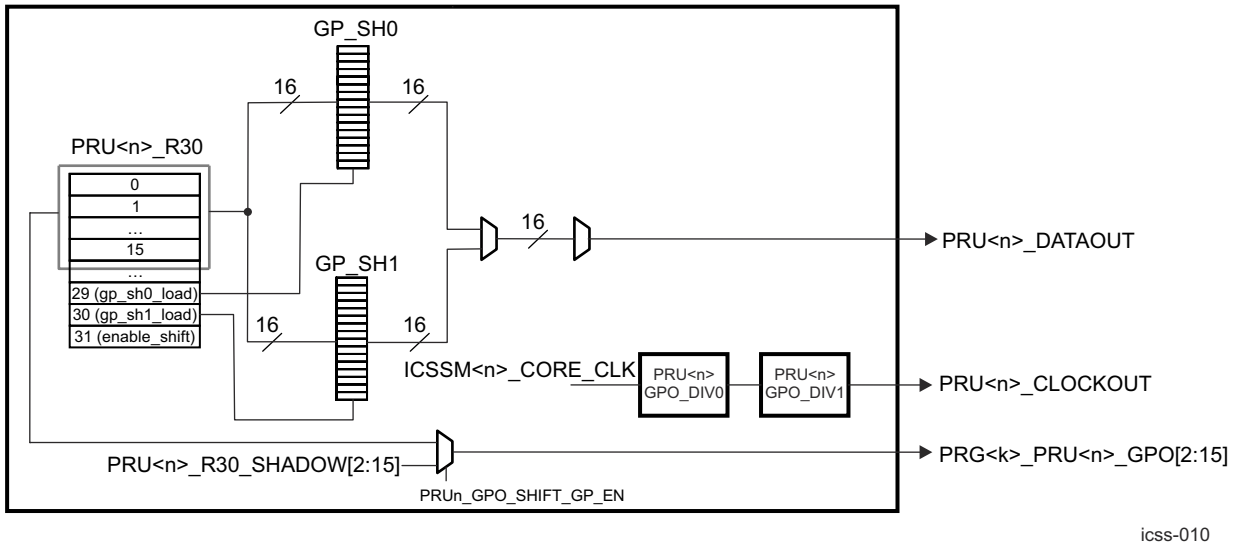


Figure 7-33. PRU R30 (GPO) Shift Out Mode Block Diagram

7.3.5.2.2.3.4.2.1 PRU EGPO Programming Model

After the PRU is intilized, the software should only enable Shift Out Mode configuration per intilization.

7.3.5.2.2.3.5 Sigma Delta (SD) Decimation Filtering

Sigma-delta Sinc filtering is achieved by the combination of PRU hardware and firmware. PRU hardware provides hardware integrators that do the accumulation part of Sinc filtering, while the differentiation part is done in firmware.

The integrator serves to count the number of 1's per clock event. Each channel has three cascaded counters, which are the accumulators for the Sinc3 filter. Each counter is 28 bits, giving a maximum count of 268,435,456. Each channel has a free running rollover clock counter. This sample counter updates the count value on the effective clock event for that channel. Each channel also contains a programmable counter compare block, and the compare register has a size of 8 bits. However, the minimum value is 4 and maximum value is 256 due to the 28-bit accumulator. Once sample counter compare value is reached, the shadow register copy is updated and the shadow register copy flag is set.

Features of the integrators in PRUs SD Demodulator:

- Up to 9 channels concurrent counting
- Software can read all 3 stages accumulators
- Flexible clock source configuration for each channel; option of independent clock source for each channel or one clock source for three channels
- Programmable, 8-bit sample counter compare register; used to set the OSR of Sinc filter
- Three 28-bit cascaded counters per channel for accumulation, only Sinc3 and Sinc2 modes supported
- Common channel enable (all channels are active or none are active)
- Fast 1 and 0 min/max count sliding programmable window for each of the 9 channels

7.3.5.2.2.3.5.1 Sigma Delta Block Diagram and Signals

The Sigma Delta's I/Os are multiplexed with the PRU GPI/GPO signals, as shown in Table 7-51.

Note: The PR<k>_PRU<n>_GP_MUX_SEL bitfield in the PRU_ICSS_GPCFG0 register (where k = 0 or 1 and n = 0 or 1) must be set to 3h for configure the GPI/GPO signals for SD mode.

Table 7-51. PRU GPI Signals and Configurations for Sigma Delta

Signal Names at Device Level (1)	Sigma Delta (SD) Mode	Function
PR<k>_PRU<n>_GPI0	SD0_CLK	SD demodulator clock channel 0
PR<k>_PRU<n>_GPI1	SD0_D	SD demodulator data channel 0
PR<k>_PRU<n>_GPI2	SD1_CLK	SD demodulator clock channel 1
PR<k>_PRU<n>_GPI3	SD1_D	SD demodulator data channel 1

Table 7-51. PRU GPI Signals and Configurations for Sigma Delta (continued)

Signal Names at Device Level (1)	Sigma Delta (SD) Mode	Function
PR<k>_PRU<n>_GPI4	SD2_CLK	SD demodulator clock channel 2
PR<k>_PRU<n>_GPI5	SD2_D	SD demodulator data channel 2
PR<k>_PRU<n>_GPI6	SD3_CLK	SD demodulator clock channel 3
PR<k>_PRU<n>_GPI7	SD3_D	SD demodulator data channel 3
PR<k>_PRU<n>_GPI8	SD4_CLK	SD demodulator clock channel 4
PR<k>_PRU<n>_GPI9	SD4_D	SD demodulator data channel 4
PR<k>_PRU<n>_GPI10	SD5_CLK	SD demodulator clock channel 5
PR<k>_PRU<n>_GPI11	SD5_D	SD demodulator data channel 5
PR<k>_PRU<n>_GPI12	SD6_CLK	SD demodulator clock channel 6
PR<k>_PRU<n>_GPI13	SD6_D	SD demodulator data channel 6
PR<k>_PRU<n>_GPI14	SD7_CLK	SD demodulator clock channel 7
PR<k>_PRU<n>_GPI15	SD7_D	SD demodulator data channel 7
PR<k>_PRU<n>_GPI16	SD8_CLK	SD demodulator clock channel 8
PR<k>_PRU<n>_GPI17	SD8_D	SD demodulator data channel 8
PR<k>_PRU<n>_GPI18	-	
PR<k>_PRU<n>_GPI19	-	

(1) Note: These signals are shared with the GP and Peripheral I/Fs. To configure for Sigma Delta, PRU_ICSS_GPCFG0[29-26] PR1_PRU0_GP_MUX_SEL (where k = 0 or 1 and n = 0 or 1) needs to be set to 3h for SD mode.

The PR<k>_PRU0_GPI1 signal (muxed with SD0_D) can be used as SD_CLKOUT when PRU-ICSS generates clock. This is a trade-off as PRU application will lose one SD channel. SD_CLKOUT needs to go through a clock generator chip if driving multiple sigma delta modulators and also be looped back into PRU-ICSS as SD_CLKIN, typically pru_gpi16.

Note: To output the SD clock on PR<k>_PRU0_GPO1, this device requires that the PRU core be configured for both SD and shift out mode (PRU_ICSS_GPCFG0[29-26] PR1_PRU0_GP_MUX_SEL = 3h and PRU_ICSS_GPCFG0[14] PRU<n>_GPO_MODE = 1h). Be sure to configure the shift out mode's clock divisors before enabling shift out mode (PRU_ICSS_GPCFG0[14] PRU<n>_GPO_MODE = 1h). Additionally, the PRU-ICSS, PRU0 SD clock is routed to both PR0_PRU0_GPO1 and PR0_PRU1_GPO1. Figure 7-34 shows a block diagram of the Sigma Delta implementation. Full description of the PRU R30 and R31 registers are shown in Table 7-53 and Table 7-54.

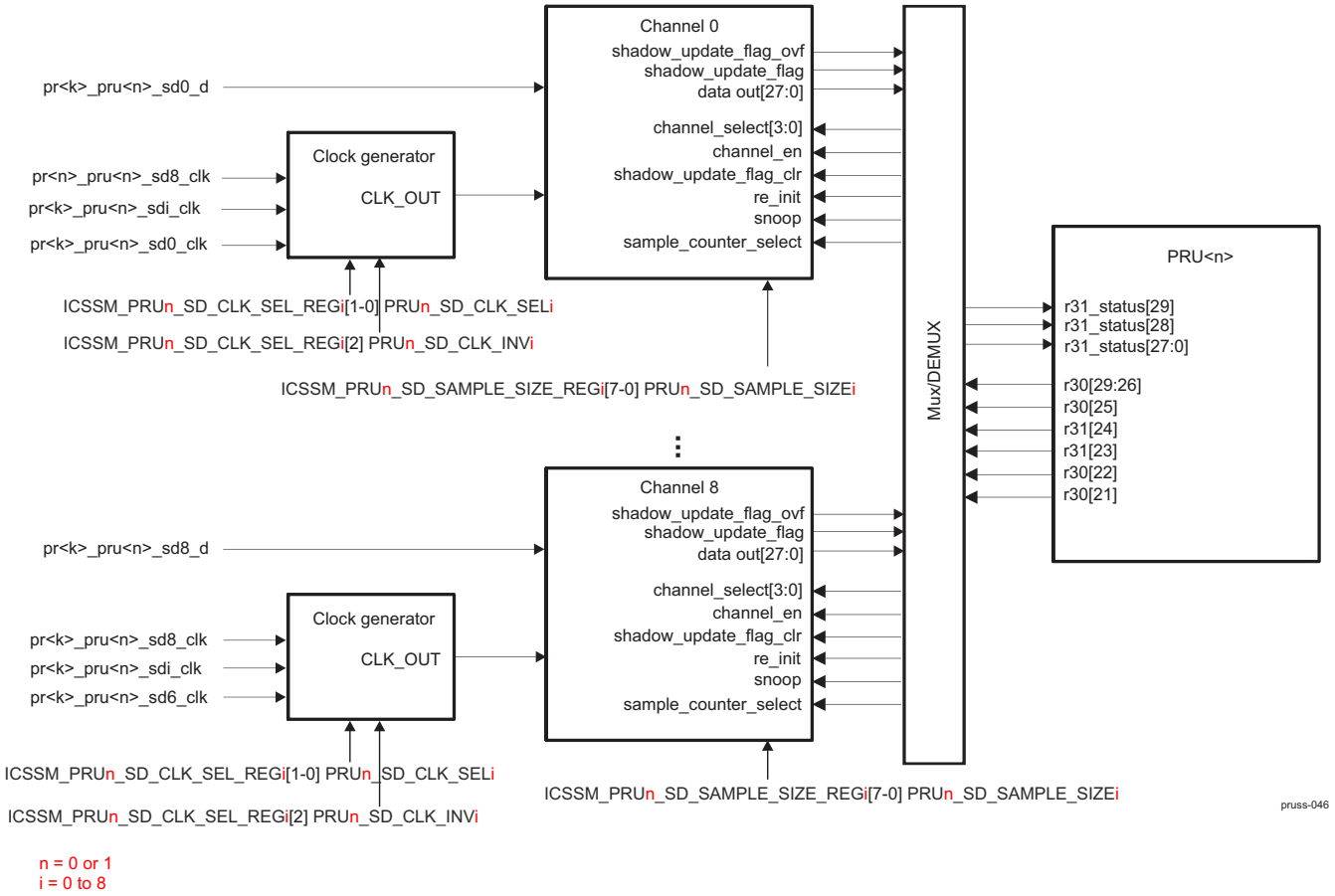


Figure 7-34. Sigma Delta Block Diagram

Note: Each channel can independently be configured to use one of three external clock sources. Table 7-52 shows the clock source options, selectable through PRU_ICSS_PRU0_SD_CLK_SELi[1-0] PRU0_SD_CLK_SELi (where n = 0 or 1 and i = 0 to 8).

Table 7-52. Sigma Delta External Clock Sources

PRU0_SD_CLK_SELi value	Clock Source
0	pr<k>_pru<n>_sd8_clk
1	pr<k>_pru<n>_sd<i>_clk
2	pr<k>_pru<n>_sd0_clk for sd0, sd1, and sd2; pr<k>_pru<n>_sd3_clk for sd3, sd4, and sd5; pr<k>_pru<n>_sd6_clk for sd6, sd7, and sd8

7.3.5.2.2.3.5.2 PRU R30 / R31 Interface

The PRU uses the R30 and R31 registers to interface with the Sigma Delta interface. Table 7-53 and Table 7-54 shows the R31 and R30 interface for the Sigma Delta mode. Note that only the parameters and data for one channel can be viewed at a time. The channel to be viewed is determined by the r30[29-26] (channel_select).

Table 7-53. Sigma Delta PRU Registers: R31

Bits	Field Name	Description
31-30	Reserved	
29	shadow_update_flag_ovf	Shadow update flag overflow, set when over sample count equals over sample size and shadow_update_flag is still set. Set bit R31[24] to clear the flag.
28	shadow_update_flag	Shadow update flag, set when over sample count equals over sample size and shadow_update_flag is still set. Set bit R31[24] to clear the flag.

Table 7-53. Sigma Delta PRU Registers: R31 (continued)

Bits	Field Name	Description
27-0	data_out[27-0]/ shadow_update_flag_clr (R31[24]) / re_init (R31[23])	data_out[27] (read): most-significant bit of sample data shadow_update_flag_clr (write): re_init (write): Set to reset all counters, flags, and shadow copy. Updates over_sample_size based on the current PRU_ICSS_PRU0_SD_SAMPLE_SIZEi[7-0] register (where n = 0 or 1 and i = 0 to 8) on the selected channel. shadow_update_flag_clr (write): Set to clear shadow_update_flag and shadow_update_flag_ovf (if set).

Table 7-54. Sigma Delta PRU Registers: R30

Bits	Field Name	Description
31-30	Reserved	
29-26	channel_select[3-0]	Channel select 0h: Channel 0 ... 8h: Channel 8 9h: Reserved ... Fh: Reserved
25	channel_en	Global Channel enable (effects all 9 channels). 0h: All channels disabled. Counters/flags are cleared. 1h: All channels enabled.
24-23	Reserved	
22	snoop	Enable snoop (i.e. fetch data) on the selected channel. 0h: acc1/acc2/acc3 shadow copy 1h: current acc1/acc2/acc3
21	sample_counter_select	Read sample counter. 0h: Not selected 1h: Sample count selected
20-0	Reserved	

The PRU-ICSS CFG register space has additional registers for controlling the SD demodulator module:

- PRU_ICSS_PRU0_SD_CLK_SELi[5-4] PRU0_SD_ACC_SELi (where n = 0 or 1, i = 0 to 8) - Selects accumulator 1, 2 or 3 as source (acc1/acc2/acc3).
- PRU_ICSS_PRU0_SD_CLK_SELi[2] PRU0_SD_CLK_INVi (where n = 0 or 1, i = 0 to 8) - Inverts clock.
- PRU_ICSS_PRU0_SD_CLK_SELi[1-0] PRU0_SD_CLK_SELi (where n = 0 or 1, i = 0 to 8) - Selects the clock source.
- PRU_ICSS_PRU0_SD_SAMPLE_SIZEi[7-0] PRU0_SD_SAMPLE_SIZEi (where n = 0 or 1, i = 0 to 8) - Selects number of samples to read before giving output.

7.3.5.2.2.3.5.3 Sigma Delta Description

Figure 7-35 shows a block diagram of the Sigma Delta hardware integrators and integration with the PRU R30 / R31 interface for a single channel.

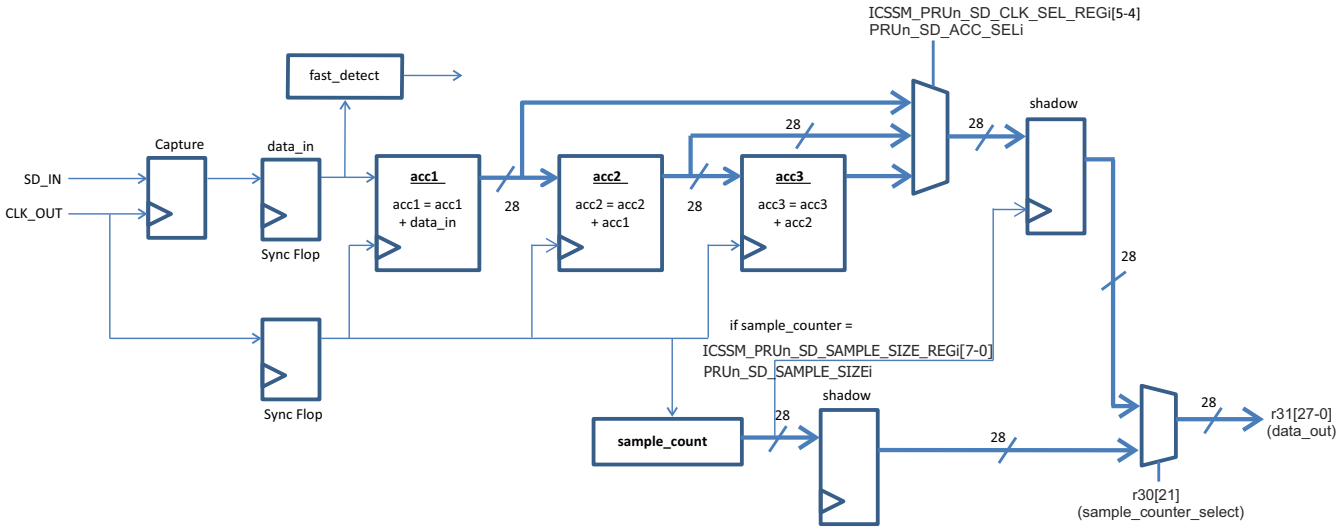


Figure 7-35. Sigma Delta Hardware Integrators Block Diagram (snoop = 0)

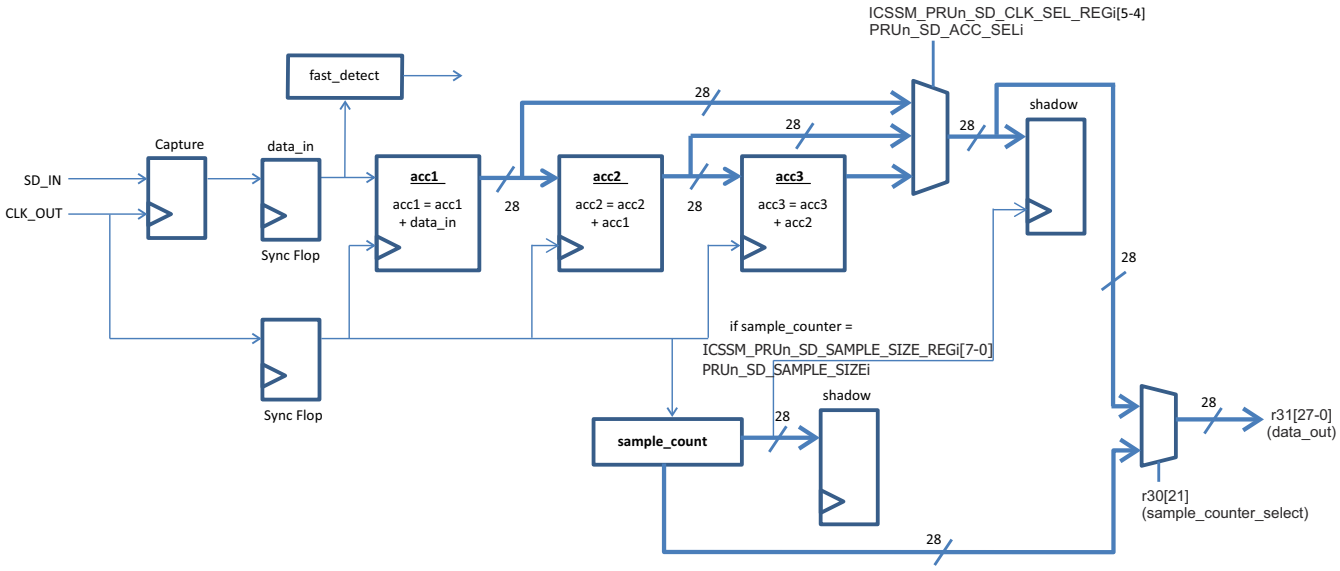


Figure 7-36. Sigma Delta Hardware Integrators Block Diagram (snoop = 1)

The three accumulators (acc1-acc3) for each channel are simple 28 bit adders. The input for acc1 is 1-bit, while the inputs for acc2 and acc3 are 28-bits. On each positive edge of the CLK_OUT, all three 28-bit counters (acc1-acc3) and the sample counter for each channel will get updated as follows:

```
acc1 = acc1 + data_in
acc2 = acc2 + acc1
acc3 = acc3 + acc2
sample_count = sample_count + 1
```

Each accumulator will rollover at 0xFF_FFFF. For example if acc2 = 0x10 and acc3 = 0xFF_FFFF, then acc3 will update to 0x00_0000F on the next clock event. Sample counter will rollover when it equals the defined sample size (PRU_ICSS_PRU0_SD_SAMPLE_SIZEi[7-0] PRU0_SD_SAMPLE_SIZEi).

Note that while the channels are not enabled, no operations are performed and all flags and counters are cleared. If a new sample size is to be loaded, the PRU firmware should assert re_init (r31[23]), and all stored count values are cleared to 0.

Fast detect block is used to detect fast changes in the amount of ones, presented in a programmable sliding window of 4 to 32 bits. The sliding window is controlled by PRU_ICSS_PRU0_SD_SAMPLE_SIZEi[10-8] PRU0_FD_WINDOW_SIZE_i bit field.

Fast detect must be enabled through the PRU_ICSS_PRU0_SD_SAMPLE_SIZEi[23] PRU0_FD_EN_i register before SD is enabled. It will start the compare after the first 32 sample clocks. Fast detect block will remain active until a re_init (r31[23]) is asserted.

The Sigma Delta interface has two status flags:

- Shadow update flag (r31[28])
- Shadow update flag overflow (r31[29])

When sample_counter equals the defined sample size (PRU_ICSS_PRU0_SD_SAMPLE_SIZEi[7-0] PRU0_SD_SAMPLE_SIZEi), then the acc1/acc2/acc3 shadow register copy will be updated, the shadow_update_flag (r31[28]) will be set, and sample_counter will rollover to 0. The PRU firmware can clear this flag by writing '1' to shadow_update_flag_clr (r31[28]). If sample_count equals the defined sample size and the shadow_update_flag is still set, then shadow_update_flag_ovf (r31[29]) will be set. Similarly, the PRU firmware can clear this flag by writing '1' to shadow_update_flag_ovf_clr (r31[29]). Note that the clear operation for both flags has a higher priority than the set event.

The PRU firmware can monitor the acc2/acc3 and sample_counter values through data_out[27-0] (r31[27-0]). Table 7-55 shows the configuration options for data_out[27-0].

Table 7-55. Data_out[27-0] Configuration Options

snoop (r30[22])	sample_counter_select (r30[21])	data_out (r31[27-0])
0	0	Reads acc1/acc2/acc3 shadow register copy. See Figure 7-35 *Sigma Delta Hardware Integrators Block Diagram (snoop = 0).*
1	0	Reads acc1/acc2/acc3 directly. See Figure 7-36 *Sigma Delta Hardware Integrators Block Diagram (snoop = 1).*
0	1	Reads sample_counter shadow register copy. See Figure 7-35 *Sigma Delta Hardware Integrators Block Diagram (snoop = 0).*
1	1	Reads sample_counter directly. See Figure 7-36 *Sigma Delta Hardware Integrators Block Diagram (snoop = 1).*

7.3.5.2.2.3.5.4 Sigma Delta Basic Programming Example

The following programming example assumes that the PRU is configured for Sigma Delta Mode (**PRU_ICSS_GPCFG0**[29-26] PR1_PRU<n>_GP_MUX_SEL = 3h).

1. Configure clock sources, accumulator source, and sample size:
   a. PRU_ICSS_PRU0_SD_CLK_SELi[1-0] PRU0_SD_CLK_SELi (where n = 0 or 1, i = 0 to 8) for clock source
   b. PRU_ICSS_PRU0_SD_CLK_SELi[2] PRU0_SD_CLK_INVi (where n = 0 or 1, i = 0 to 8) for clock polarity
   c. PRU_ICSS_PRU0_SD_CLK_SELi[5-4] PRU0_SD_ACC_SELi (where n = 0 or 1, i = 0 to 8) - for accumulator source (acc1/acc2/acc3)
   d. PRU_ICSS_PRUSS_SD_PRU0_SAMPLE_SIZEi[7-0] PRU0_SD_SAMPLE_SIZE for sample size
2. Reinitialize all channels whose sample size was configured
   a. Select channel by writing to channel_select (r30[29-26])
   b. Delay at least 1 PRU cycle before executing re_int in step 2c.
   c. Reinitialize selected channel by writing to re_init (r31[23])
   d. Repeat steps 2a & 2b for all configured channels
3. Enable all channels by writing '1' to channel_en (r30[25])
4. Select channel by writing to channel_select (r30[29-26])
   a. Poll shadow_update_flag (r31[28]) to detect when acc1/acc2/acc3 shadow register copy data is ready to be read
   b. Delay at least 1 PRU cycle before polling shadow_update_flag in Step 4c.

c. Read data_out[27-0] (r31[27-0])
d. Clear shadow_update_flag by writing ‘1’ to r31[24]

5. Repeat step 4 for new channel

7.3.5.2.2.3.6 Three Channel Peripheral Interface

The 3 channel Peripheral Interface supports functionality for operations utilizing the EnDat 2.2 and BiSS protocols. The 3 channel Peripheral Interface supports both 2 wire and 4 wire serial RS-485 communication. The following table shows the supported encoder protocols for the PRU-ICSS.

Table 7-56. Three Channel Peripheral Interface Supported Encoder Protocols

Encoder Protocol	Number of wire in RS-485 Communication
EnDat 2.2	4 wire
BiSS	4 wire
HDSL	2 wire
Tamagawa	2 wire

This module supports the following features:

- 3 channels with baud range from 100 kHz to 16 MHz
- PRU_ICSS_UART_CLK (default) or PRU_ICSS_ICLK controller clock is an input to independent div16fr clock dividers to produce a 1X clock (PERIF<m>_CLK) and oversampling clock
- Half-duplex (TX and RX are not supported concurrently)
- TX FIFO size of 32 bits
- RX FIFO size of 4 bits
- Configurable shift size/oversampling on RX
- Optional RX frame size auto shut off
- Programmable HW delay 1 (wire delay, controlling when the clock signal is first driven low) and delay 2 (tst delay, controlling when the clock signal is first driven high) on TX operation
- Optional programmable TX termination
- Individual TX channel start trigger (tx_channel_go) or simultaneous TX start trigger for all channels (tx_global_go)
- Flexible HW assisted clock output generation to allow free running, stop high and stop low (after last RX data), or stop high (after last TX data) operation with optional software clock override feature
- Optional SW direct snoop of data input
- RX Start Bit of '1' or '0'

7.3.5.2.2.3.6.1 Peripheral Interface Block Diagram and Signal Configuration

The Peripheral Interface’s I/Os are multiplexed with the PRU GPI/GPO signals, as shown in Table 7-57. The PR1_PRU<n>_GP_MUX_SEL bitfield in the PRU_ICSS_GPCFG0 register (PRU0 or PRU1) must be set to 1h for configure the GPI/GPO signals for Peripheral I/F mode.

Table 7-57. PRU GPI/GPO Signals and Configurations for Peripheral I/F(1)

Pad Names at Device Level(2) (3)	Peripheral I/F Mode (PRU_ICSS_GPCFG0[29-26] PR1_PRU0_GP_MUX_SEL = 1h)
PR<k>_PRU<n>_GPI0	
PR<k>_PRU<n>_GPI1	
PR<k>_PRU<n>_GPI2	
PR<k>_PRU<n>_GPI3	
PR<k>_PRU<n>_GPI4	
PR<k>_PRU<n>_GPI5	
PR<k>_PRU<n>_GPI6	
PR<k>_PRU<n>_GPI7	
PR<k>_PRU<n>_GPI8	
PR<k>_PRU<n>_GPI9	PERIF0_IN
PR<k>_PRU<n>_GPI10	PERIF1_IN

Table 7-57. PRU GPI/GPO Signals and Configurations for Peripheral I/F[1] (continued)

Pad Names at Device Level[2] [3]	Peripheral I/F Mode (PRU_ICSS_GPCFG0[29-26] PR1_PRU0_GP_MUX_SEL = 1h)
PR<k>_PRU<n>_GPI11	PERIF2_IN
PR<k>_PRU<n>_GPI12	
PR<k>_PRU<n>_GPI13	
PR<k>_PRU<n>_GPI14	
PR<k>_PRU<n>_GPI15	
PR<k>_PRU<n>_GPI16	
PR<k>_PRU<n>_GPI17	
PR<k>_PRU<n>_GPI18	
PR<k>_PRU<n>_GPI19	
PR<k>_PRU<n>_GPO0	PERIF0_CLK
PR<k>_PRU<n>_GPO1	PERIF0_OUT
PR<k>_PRU<n>_GPO2	PERIF0_OUT_EN
PR<k>_PRU<n>_GPO3	PERIF1_CLK
PR<k>_PRU<n>_GPO4	PERIF1_OUT
PR<k>_PRU<n>_GPO5	PERIF1_OUT_EN
PR<k>_PRU<n>_GPO6	PERIF2_CLK
PR<k>_PRU<n>_GPO7	PERIF2_OUT
PR<k>_PRU<n>_GPO8	PERIF2_OUT_EN
PR<k>_PRU<n>_GPO9	
PR<k>_PRU<n>_GPO10	
PR<k>_PRU<n>_GPO11	
PR<k>_PRU<n>_GPO12	
PR<k>_PRU<n>_GPO13	
PR<k>_PRU<n>_GPO14	
PR<k>_PRU<n>_GPO15	
PR<k>_PRU<n>_GPO16	
PR<k>_PRU<n>_GPO17	
PR<k>_PRU<n>_GPO18	
PR<k>_PRU<n>_GPO19	

(1) Usage of the Peripheral Interface signals are not restricted to only ENDAT interfaces.
(2) Note: These signals are shared with the GP, MII and Sigma Delta modes. To configure for Peripheral I/F, PRU_ICSS_GPCFG0[29-26] PR1_PRU0_GP_MUX_SEL needs to be set to 1h.
(3) Some devices may not pin out all 29 bits of R31 and all 32 bits of R30. For which pins are available on this device, see *PRUSS Environment*. See the device-specific Datasheet for device pin mapping.

A block diagram for the Peripheral I/F is included in Figure 7-37. As shown, each channel is composed of four I/Os:

- PERIF<m>_IN - RX input data
- PERIF<m>_CLK - Clock (CLK_OUT) generated by the 1x (or TX) clock. The default value is 1.
- PERIF<m>_OUT - TX output data. The default value is 0.
- PERIF<m>_OUT_EN - TX enable output (1 = TX mode, 0 = RX mode). The default value is 0. Note: This signal is auto controlled by hardware.

Figure 7-37. Peripheral I/F Block Diagram

7.3.5.2.2.3.6.2 PRU R30 and R31 Interface

The PRU uses the R30 and R31 registers to interface with the Peripheral I/F. Table 7-58 shows the R31 and R30 interface for the Peripheral I/F RX mode, and Table 7-59 shows the comparable interface for the TX mode.

Table 7-58. Peripheral I/F RX

Register	Bits	Field name	Description
R31	31-30	Reserved	PRU Host Interrupts 1/0 from local INTC
	29	ovf2	Overflow Flag for Channel 2. Write 1 to clear.
	28	ovf1	Overflow Flag for Channel 1. Write 1 to clear.
	27	ovf0	Overflow Flag for Channel 0. Write 1 to clear.
	26	val2	Valid Flag for Channel 2. Write 1 to clear.
	25	val1	Valid Flag for Channel 1. Write 1 to clear.
	24	val0	Valid Flag for Channel 0. Write 1 to clear.
	23-16	rx_data_out2	Oversampled Data Output for Channel 2. Note: These bits are shared with the TX Interface. When TX_FIFO has stopped transmission, RX data will be selected.
	15-8	rx_data_out1	Oversampled Data Output for Channel 1. Note: These bits are shared with the TX Interface. When TX_FIFO has stopped transmission, RX data will be selected.
	7:0	rx_data_out0	Oversampled Data Output for Channel 0. Note: These bits are shared with the TX Interface. When TX_FIFO has stopped transmission, RX data will be selected.
R30	31-27	Reserved	
	26	rx_en2	RX Enable for Channel 2. 0h: Channel not enabled, all counters/flags will get reset 1h: Channel is enabled
	25	rx_en1	RX Enable for Channel 1. 0h: Channel not enabled, all counters/flags will get reset 1h: Channel is enabled
	24	rx_en0	RX Enable for Channel 0. 0h: Channel not enabled, all counters/flags will get reset 1h: Channel is enabled
	23-0	Reserved	

Table 7-59. Peripheral I/F TX

Register	Bits	Field name	Description
R31	31-30	Reserved	
	29-22	Reserved	
	21	tx_global_reinit_active/ busy2	Tx_global_reinit action has some latency do to clocking. This status shows if action is completed. 1h: Active 0h: Done For non reinit case, this bit states that last bit is on tx wire. It does not mean the clock is off. 1h: Last bit is not done 0h: Last bit on tx wire Note that by using rx auto arm feature, the observation is lost at rx enable. This can be used to determine when to enable rx during non-auto arm case.
	20	tx_global_go	TX global start of all channels. Note: FIFO must not be empty. If empty, transmit will not start.
	19	tx_global_reinit	Reinit all channels into default mode. This clears all flags and state machines for all channels. Note: Sequence should be assert tx_global_reinit then de-assert rx_en. This will ensure TX and RX are in reset/default state. User must assert this after the frame has been sent and TX is not busy.
	18	tx_channel_go	TX start the channel transmit (selected by tx_ch_sel). Note: FIFO must not be empty.
	17	unr2	Under Run Flag for Channel 2. This flag is only set when the tx_frame_count is nonzero and FIFO is empty at time to send data.
	16	ovr2	Over Run Flag for Channel 2
	15-14	Reserved	

Table 7-59. Peripheral I/F TX (continued)

Register	Bits	Field name	Description
	13	tx_global_reinit_active/ busy1	Tx_global_reinit action has some latency do to clocking. This status shows if action is completed. 1h: Active 0h: Done For non reinit case, this bit states that last bit is on tx wire. It does not mean the clock is off. 1h: Last bit is not done 0h: Last bit on tx wire Note that by using rx auto arm feature, the observation is lost at rx enable. This can be used to determine when to enable rx during non-auto arm case.
	12-10	tx_fifo_sts1	TX FIFO occupancy status for Channel 1. 0 :0 Empty 1h: 1 word 2h: 2 words 3h: 3 words 4h: Full 5h-7h: Reserved
	9	unr1	Under Run Flag for Channel 1. This flag is only set when the tx_frame_count is nonzero and FIFO is empty at time to send data.
	8	ovr1	Over Run Flag for Channel 1
	7-6	Reserved	
	5	tx_global_reinit_active/ busy0	Tx_global_reinit action has some latency do to clocking. This status shows if action is completed. 1h: Active 0h: Done For non reinit case, this bit states that last bit is on tx wire. It does not mean the clock is off. 1h: Last bit is not done 0h: Last bit on tx wire Note that by using rx auto arm feature, the observation is lost at rx enable. This can be used to determine when to enable rx during non-auto arm case.
	4-2	tx_fifo_sts0	TX FIFO occupancy status for Channel 0. 0h: Empty 1h: 1 word 2h: 2 words 3h: 3 words 4h: Full 5h-7h: Reserved
	1	unr0	Under Run Flag for Channel 0. This flag is only set when the tx_frame_count is nonzero and FIFO is empty at time to send data.
	0	ovr0	Over Run Flag for Channel 0
R30	31-21	Reserved	
	20-19	clk_mode	CLK_OUT mode. 0h: Free-running/stop-low. Clock will remain free-running until the receive module has received the number of bits indicated in rx_frame_counter and then the clock will stop low. 1h: Free-running/stop-high (default). Clock will remain free-running until the receive module has received the number of bits indicated in rx_frame_counter and then the clock will stop high. Note: This is the default/reset state, and a hardware reset or reinit will return clk_mode to this state. Note: The initial state of the clock will be high, but the clock will not start until TX GO event. 2h: Free-run. NOTE: You must do a reinit to get out of this clock mode then you can update clk_mode to a different mode. Also if you do multiple TX GO, the 2nd go should have tst_delay and wire_delay zero since the clock is free running after the first go. 3h: Stop high after transmit. Clock will run until the last TX bit is sent and stops high.
	18	Reserved	
	17-16	tx_ch_sel	TX channel select. 0h: Channel 0 1h: Channel 1 2h: Channel 2 3h: Reserved
	15-9	Reserved	

Table 7-59. Peripheral I/F TX (continued)

Register	Bits	Field name	Description
	7-0	tx_data	TX data for FIFO. Notes: FIFO transmits MSB first and is 32-bits deep. TX_FIFO_SWAP_BITS bit in the PRU-ICSS CFG register space can be used to flip the load order of bits. The FIFO has 2 modes of operation: 1. Preload and Go. This should be done for EnDAT and frames less than 32-bits. 2. Continuous mode. This should be done for frames bigger than 32-bits. In continuous mode, software needs to keep up with the line rate and ensure that the FIFO is never empty. When the FIFO is at 2 byte level, software needs to load the next 2 bytes. If software waits till the end of the empty state, it is possible to get the TX into a bad state. The FIFO state can be recovered via re-init.

Note

The PRU-ICSS CFG register space has additional registers for controlling the Peripheral I/F module.

7.3.5.2.2.3.6.3 Clock Generation

7.3.5.2.2.3.6.3.1 Configuration

The Peripheral I/F module has two source clock options, PRU_ICSS_UART_CLK (default) and PRU_ICSS_ICLK. There are two independent clock dividers (div16) for the 1x and oversampling (OS) clocks, and each clock divider is configurable by two cascading dividers:

- [31-16] PRU0_ED_TX_DIV_FACTOR and [15] PRU0_ED_TX_DIV_FACTOR_FRAC for the 1x clock
- [31-16] PRU0_ED_RX_DIV_FACTOR and [15] PRU0_ED_RX_DIV_FACTOR_FRAC for the OS clock

The 1x clock is output on the PERIF<m>_CLK signal. In TX mode, the output data is read from the TX FIFO at this 1x clock rate. The default value of this clock is high and the start and stop conditions for this clock are described in Section 3.5.2.2.3.6.3.2 *Clock Output Start Conditions* and Section 3.5.2.2.3.6.3.3 *Stop Conditions*.

In RX mode, the input data is sampled at the OS clock rate. Note: The OS clock rate divided by the 1x clock rate must equal [2-0] PRU0_ED_RX_SAMPLE_SIZE.

Example clock rates and divisor values relative to the 192-MHz PRU_ICSS_UART_CLK source are shown in Table 7-60.

Table 7-60. Clock Rate Examples for 192-MHz PRUSSn_UART_GFCLK Clock Source

TX_DIV_FACTOR	1x Clock	RX_DIV_FACTOR	RX_DIV_FACTOR_FRAC	OS Clock	Oversample Factor
12	16 MHz	1	1.5	128 MHz	8x
16	12 MHz	2	1	96 MHz	8x
24	8 MHz	3	1	64 MHz	8x
32	6 MHz	4	1	48 MHz	8x
48	4 MHz	6	1	32 MHz	8x
96	2 MHz	12	1	16 MHz	8x
192	1 MHz	24	1	8 MHz	8x

7.3.5.2.2.3.6.3.2 Clock Output Start Conditions

This section describes the configurable start conditions for the PERIF<m>_CLK. The software can completely control via PRU_ICSS_PRU0_ED_CHm_CFG0 when bit [29] PRU0_ED_CLK_OUT_OVR_ENm = 1h (where n = 0 or 1 and m = 0 to 2). By default however, the PRU hardware will control the clocks as described in the following sections.

7.3.5.2.2.3.6.3.2.1 TX Mode (RX_EN = 0)

In TX mode, the PERIF<m>_CLK begins after the firmware loads the TX FIFO and sets either r31[20] (tx_global_go) or r30[17-16] (tx_channel_go) to 1h. After the “go” bit is set, the delay1 (wire delay) compensation counter for each channel begins. After delay1 is complete, PERIF<m>_CLK is driven low and then the delay2 (tst) counter begins. After the delay2 counter expires, the PERIF<m>_CLK starts running (first low and then

high). Therefore, first rising edge of PERIF<m>_CLK (measured from the go bit) = delay1 (tx wire delay) + delay2 (tst_counter delay) + half of the 1x clock frequency (since the clock starts low).

Figure 7-38 shows the start condition for TX mode. As shown in the figure, the default value of clock is high. The PRU-ICSS CFG register space has additional registers for controlling the TX start timing delay values:

- Delay 1: PRU_ICSS_PRU0_ED_CHm_CFG0_REG[10-0] PRU0_ED_TX_WDLYm (where n = 0 or 1 and m = 0 to 2)
- Delay 2: PRU_ICSS_PRU0_ED_CHm_CFG1_REG[15-0] PRU0_ED_TST_DELAY_COUNTERm (where n = 0 or 1 and m = 0 to 2)

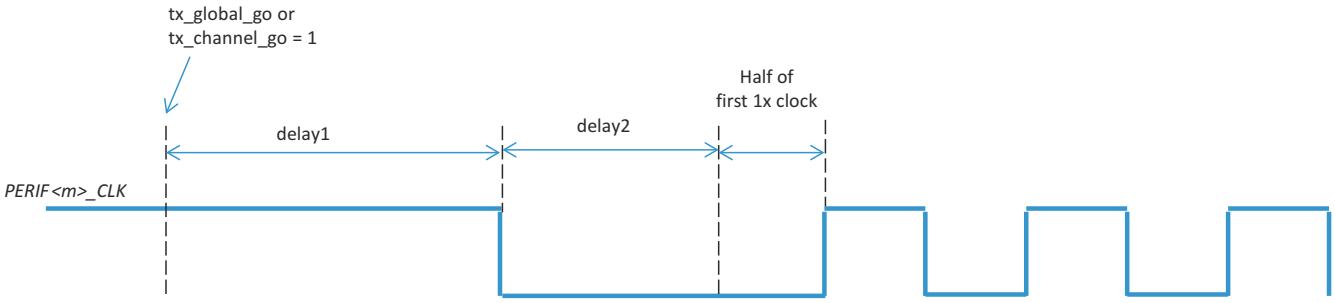


Figure 7-38. TX Mode Start Condition

7.3.5.2.2.3.6.3.2.2 RX Mode (RX_EN = 1)

In RX mode, the PERIF<m>_CLK will start running whenever the RX_EN is set. Note that the PRU firmware in this mode is responsible for any delay conditions.

The hardware can also auto-enable RX mode at the end of a TX transaction. The PRU_ICSS_PRU0_ED_CHm_CFG1_REG[31-16] PRU0_ED_RX_EN_COUNTERm (where n = 0 or 1 and m = 0 to 2) is used to program a delay between the last TX bit sent and when the RX_EN is set.

7.3.5.2.2.3.6.3.3 Stop Conditions

The r30[20-19] (clk_mode[1:0]) value determines the stop condition for PERIF<m>_CLK. There are 4 options available:

clk_mode_value	Description
0	Stop low on last RX frame
1	Stop high on last RX frame
2	Run continuously
3	Stop high on last TX bit

The last RX frame is configured by PRU_ICSS_PRU0_ED_CHm_CFG0_REG[27-16] PRU0_ED_RX_FRAME_SIZEm, and the last TX bit is configured by PRU_ICSS_PRU0_ED_CHm_CFG0_REG[15-11] PRU0_ED_TX_FRAME_SIZEm (where n = 0 or 1 and m = 0 to 2). Each stop condition is shown in Figure 7-39 through Figure 7-42.

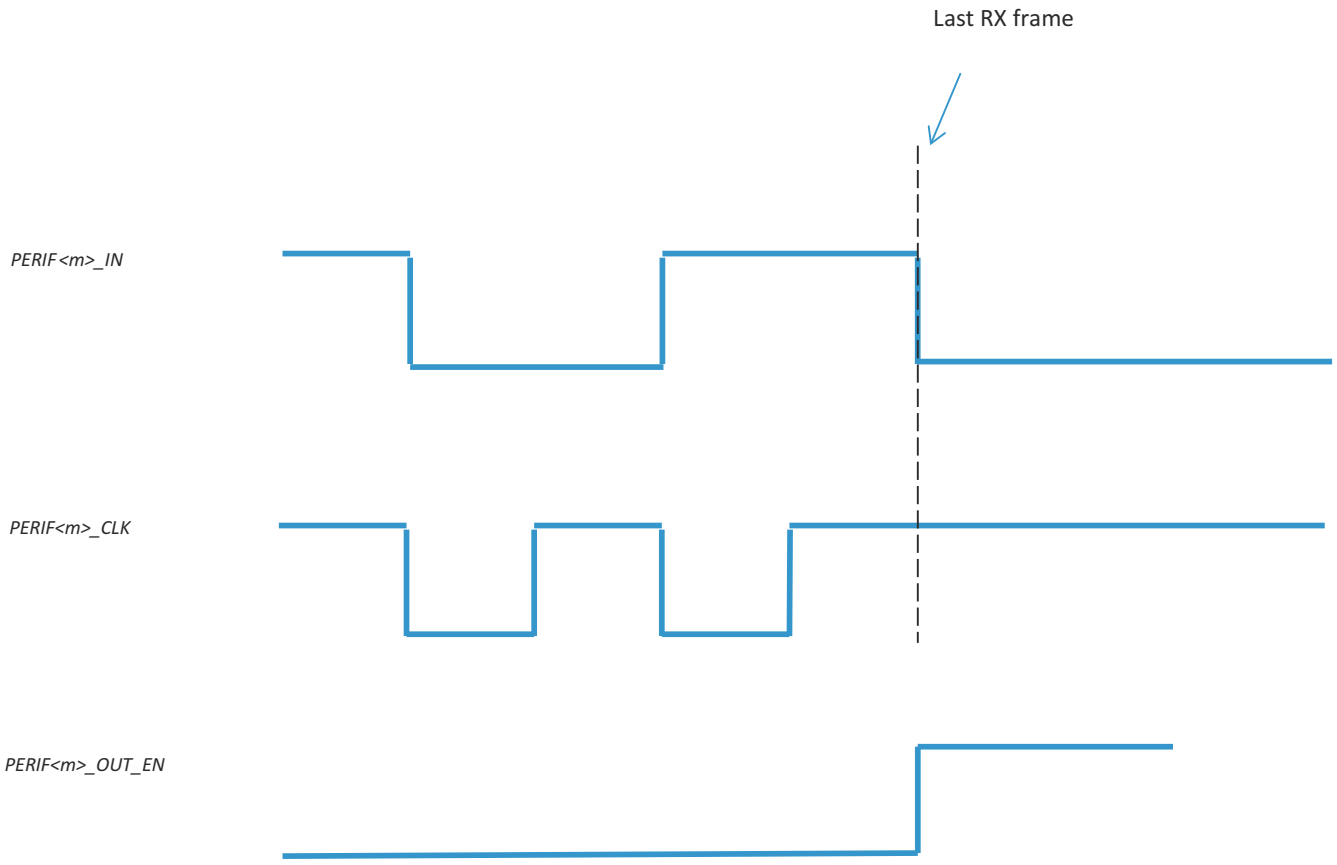


Figure 7-39. PERIF<m>_CLK Stop High on Last RX Frame

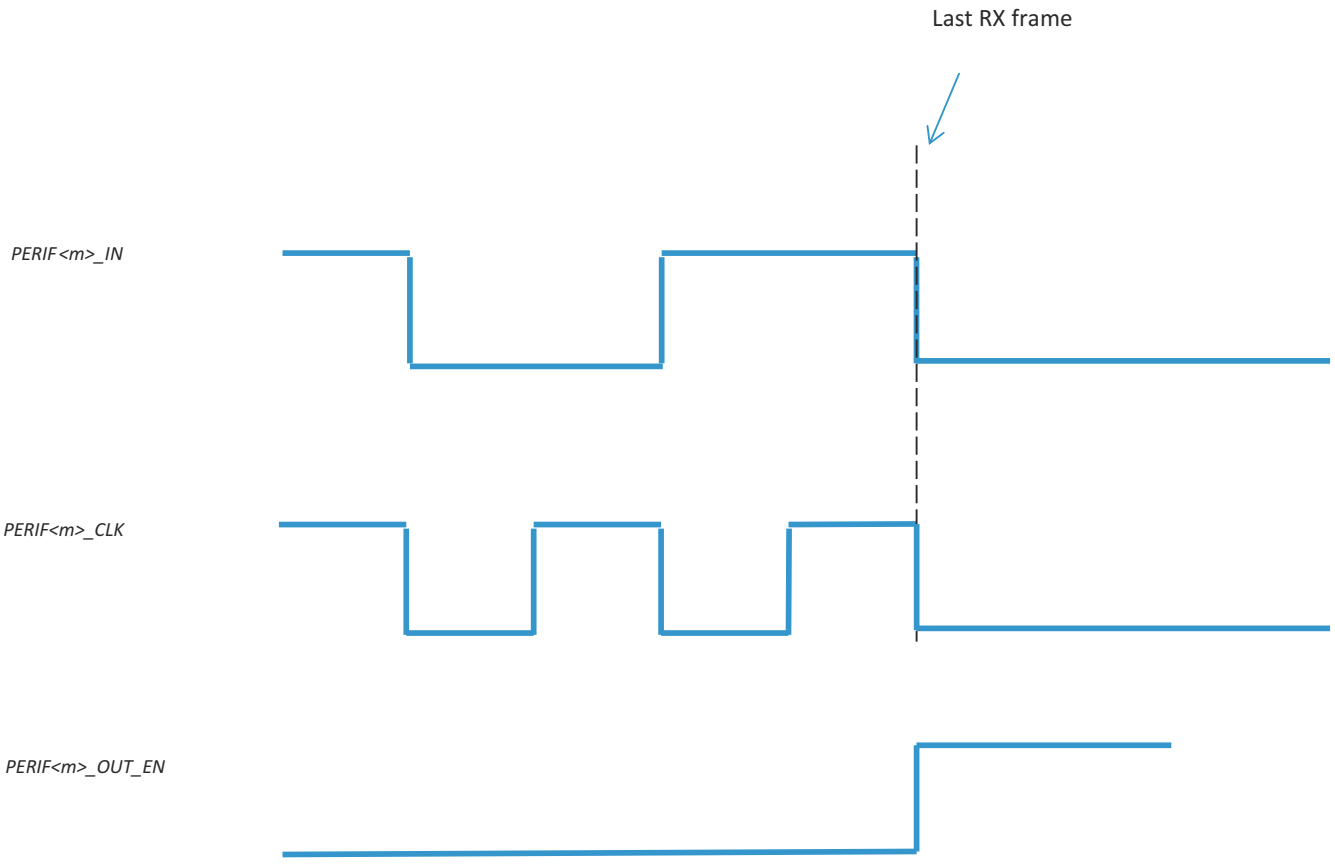


Figure 7-40. PERIF<m>_CLK Stop Low on Last RX Frame

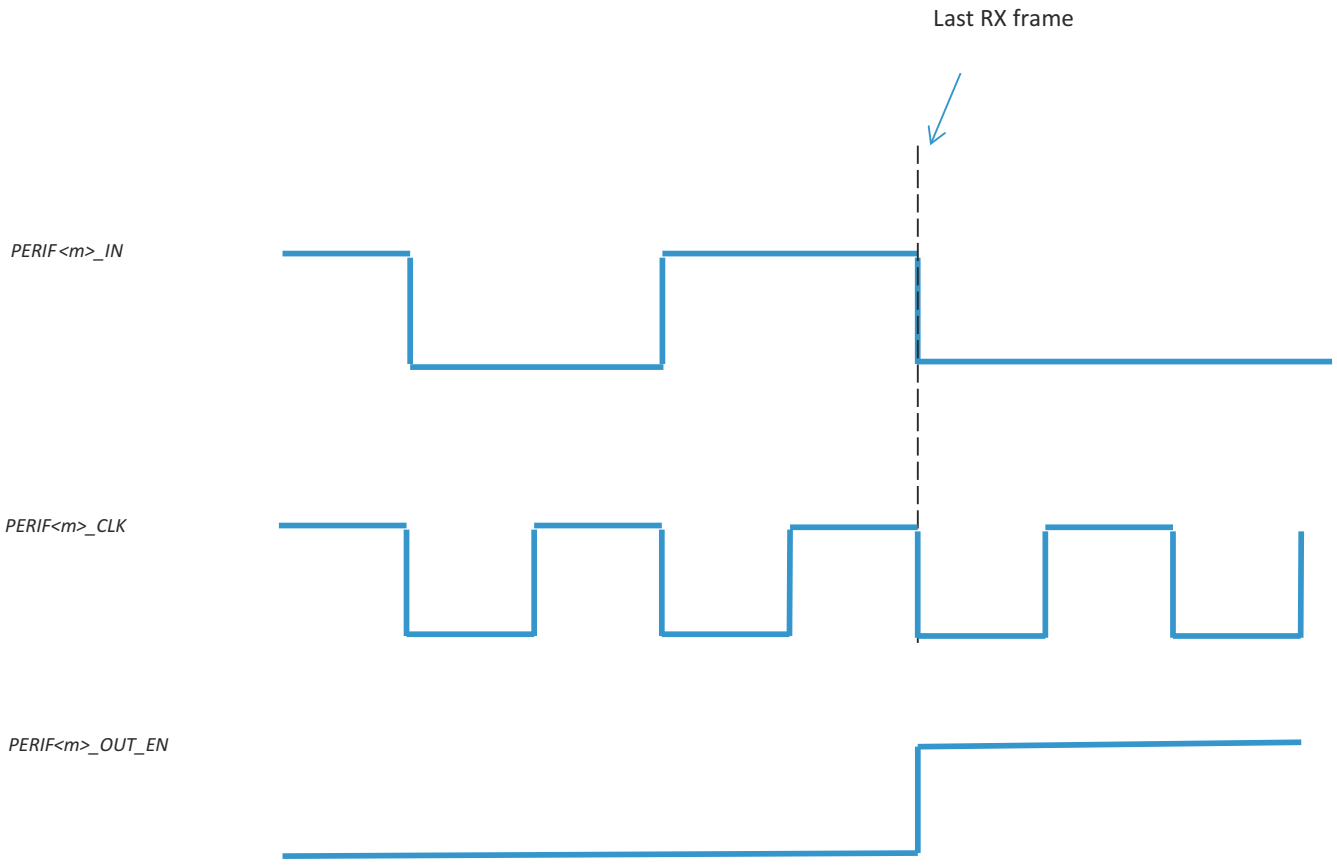


Figure 7-41. PERIF<m>_CLK Run Continuously

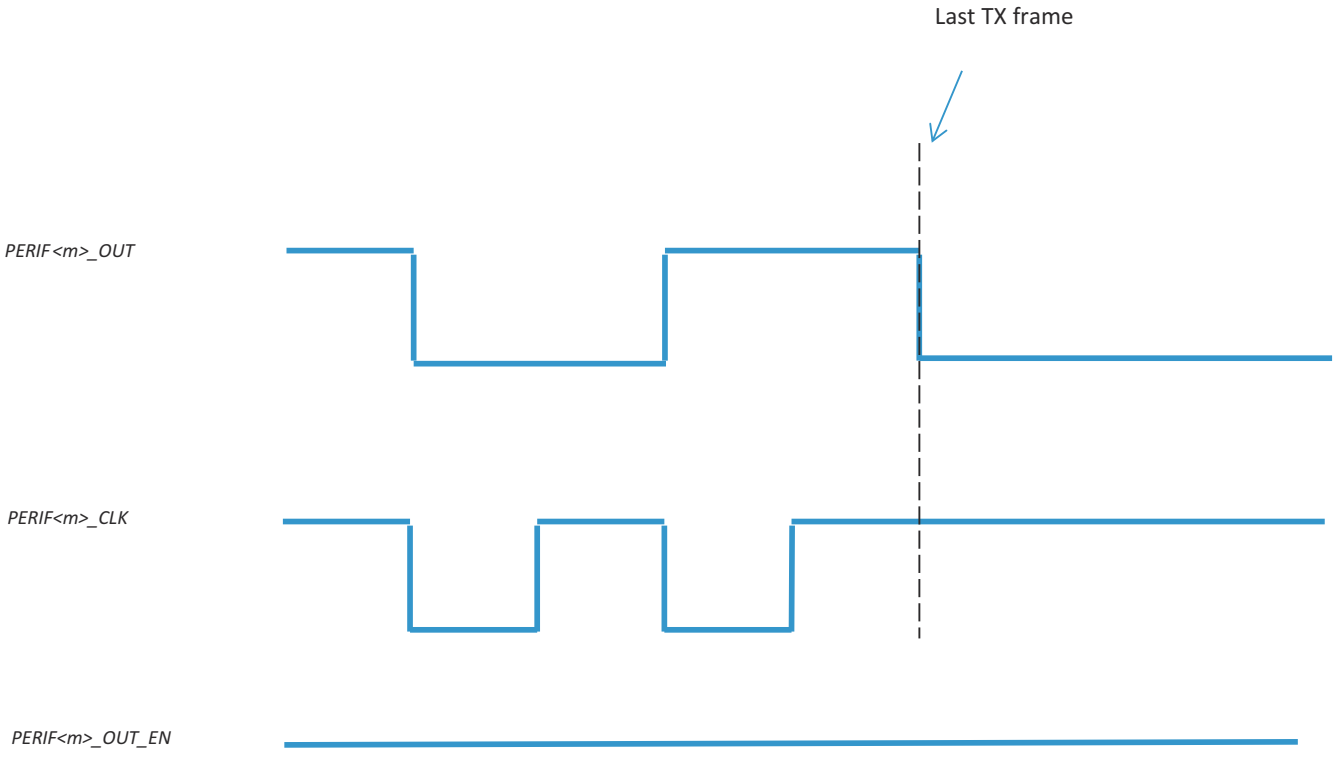


Figure 7-42. PERIF<m>_CLK Stop High on Last TX Bit

7.3.5.2.2.3.6.4 Three Peripheral Mode Basic Programming Model

The following programming models assume that the PRU is configured for 3 Peripheral Mode (PRU_ICSS_GPCFG0[29-26] PR1_PRU0_GP_MUX_SEL = 1h).

7.3.5.2.2.3.6.4.1 Clock Generation

Follow these steps to configure Peripheral I/F clocks using the HW control of the clock:

1. Select TX and RX clock sources:
   a. PRU_ICSS_PRU0_ED_TX_CFG_REG[4] PRU0_ED_TX_CLK_SEL for the TX clock source
   b. PRU_ICSS_PRU0_ED_RX_CFG_REG[4] PRU0_ED_RX_CLK_SEL for the RX clock source
2. Configure the 1x (TX) clock frequency:
   a. Write Division Factor to PRU_ICSS_PRU0_ED_TX_CFG_REG[31-16] PRU0_ED_TX_DIV_FACTOR
   b. Write Fraction division factor to PRU_ICSS_PRU0_ED_TX_CFG_REGISTER[15] PRU0_ED_TX_DIV_FACTOR_FRAC
3. Configure the oversampling (RX) frequency and oversample size:
   a. Write Division Factor to PRU_ICSS_PRU0_ED_RX_CFG_REG[31-16] PRU0_ED_RX_DIV_FACTOR
   b. Write Fraction division factor to PRU_ICSS_PRU0_ED_RX_CFG_REG[15] PRU0_ED_RX_DIV_FACTOR_FRAC
   c. Write RX oversample size to PRU_ICSS_PRU0_ED_RX_CFG_REG[2-0] PRU0_ED_RX_SAMPLE_SIZE
4. Select the clk_mode to configure how the PERIF<m>_CLK signal ends after TX/RX:
   a. Write to r30[20-19] (clk_mode). Note: The clk_mode setting can also be changed per transaction.
5. Configure the wire, tst, and rx_en_counter delay values:
   a. PRU_ICSS_PRU0_ED_CHm_CFG0_REG[10-0] PRU0_ED_TX_WDLYm for wire delay (where n = 0 or 1 and m = 0 to 2)
   b. PRU_ICSS_PRU0_ED_CHm_CFG1_REG[15-0] PRU0_ED_TST_DELAY_COUNTERm for tst delay (where n = 0 or 1 and m = 0 to 2)
   c. PRU_ICSS_PRU0_ED_CHm_CFG1_REG[31-16] PRU0_ED_RX_EN_COUNTER for auto-delay between TX and RX (where n = 0 or 1 and m = 0 to 2)

7.3.5.2.2.3.6.4.2 TX - Single Shot

Follow these steps to configure the Peripheral I/F channel(s) for a single shot transmission:

1. (Optional) Configure TX FIFO for MSB (default) or LSB:
   a. PRU_ICSS_PRU0_ED_CHm_CFG0_REG[31] PRU0_ED_TX_FIFO_SWAP_BITSm (where n = 0 or 1 and m = 0 to 2)
2. Pre-load TX FIFO:
   a. Select TX channel by writing the desired channel number to R30[17-16] (tx_ch_sel)
   b. Write 1-4 bytes of data to r30[7-0] (tx_data). At each r30[7-0] write, data will be pushed into the FIFO.
   c. Repeat Steps 2a and 2b for all desired channels.
3. Configure TX frame size if less than 4 full bytes loaded into FIFO:
   a. PRU_ICSS_PRU0_ED_CHm_CFG0_REG[15-11] PRU0_ED_TX_FRAME_SIZEm (where n = 0 or 1 and m = 0 to 2)
4. Push TX FIFO data to PERIF<m>_OUT (see Section 3.5.2.2.3.6.3.2 for the PERIF<m>_CLK and PERIF<m>_OUT start time relationship);
   a. To start TX on all channels, set r31[20] = 1 (tx_global_go).
   b. To start TX on individual channel:
      i. Select TX channel by writing the desired channel number to R30[17-16] (tx_ch_sel)
      ii. Set R31[18] = 1 (tx_channel_go)
5. If PRU_ICSS_PRU0_ED_CHm_CFG1_REG[31-16] PRU0_ED_RX_EN_COUNTERm > 0 (where n = 0 or 1 and m = 0 to 2), then the channel will automatically switch into RX mode. See Section 3.5.2.2.3.6.4.4 for an example of how to program and configure RX content.
6. If PRU_ICSS_PRU0_ED_CHm_CFG1_REG[31-16] PRU0_ED_RX_EN_COUNTERm = 0, poll either r31[21, 13, or 5] (tx_global_reinit_active/busy[2,1,0]) or PRU_ICSS_PRU0_ED_TX_CFG_REG[7, 6, or 5] PRU0_ED_BUSY_m (where m = 0 to 2, indicates channel number) for when TX is complete

Note

The PERIF<m>_CLK Peripheral I/F requires that PERIF<m>_CLK be in a high state at the beginning of a new transaction. If the clock ended the single shot transmission in low state, then the clock needs to be reset before sending more data. The steps to reset PERIF<m>_CLK are:

1. Set R31[19] = 1 (tx_global_reinit) to reset clock high
2. Wait until PRU0_ED_BUSY_m bit is cleared
3. Re-configure R30[20-19] (clk_mode), since reinit will reset the clk_mode to "Free-running/stop-high" mode

7.3.5.2.2.3.6.4.3 TX - Continuous FIFO Loading

Follow these steps to configure the Peripheral I/F channel(s) for a continuous loading transmission:

1. (Optional) Configure TX FIFO for MSB (default) or LSB:
   a. PRU_ICSS_PRU0_ED_CHm_CFG0_REG[31] PRU0_ED_TX_FIFO_SWAP_BITSm
2. Pre-load TX FIFO:
   a. Select TX channel by writing the desired channel number to r30[17-16] (tx_ch_sel)
   b. Write 1-4 bytes of data to r30[7-0] (tx_data). At each r30[7-0] write, data will be pushed into the FIFO.
   c. Repeat Steps 2a and 2b for all desired channels.
3. Configure TX frame size to continuously transmit the TX FIFO until empty:
   a. Set PRU_ICSS_PRU0_ED_CHm_CFG0_REG[15-11] PRU0_ED_TX_FRAME_SIZEm = 0h
4. Push TX FIFO data to PERIF<m>_OUT (see Section 3.5.2.2.3.6.3.2 for the PERIF<m>_CLK and PERIF<m>_OUT start time relationship):
   a. To start TX on all channels, set r31[20] = 1 (tx_global_go).
   b. To start TX on individual channel:
      i. Select TX channel by writing the desired channel number to r30[17-16] (tx_ch_sel)
      ii. Set r31[18] = 1 (tx_channel_go)
5. Monitor line rate and reload FIFO:
   a. Polling r31[xx, 12-10, 4-2] (tx_fifo_sts<m>)
   b. When FIFO level is at 2 bytes, load next 2 bytes of data (see Step 2). Do not let the FIFO get close to 0. Once the FIFO runs empty, the hardware will assume the PRU has reached end of the last transmit. Any new writes to the FIFO will NOT be sent until the software sends another tx_channel_go bit. Note: There are also underrun and overrun error flags that can be monitored.
6. To end TX operation, do not send any new data to FIFO.
   a. If PRU_ICSS_PRU0_ED_CHm_CFG1_REG[31-16] PRU0_ED_RX_EN_COUNTERm > 0 (where n = 0 or 1 and m = 0 to 2), then the channel will automatically switch into RX mode. See Section 3.5.2.2.3.6.4.4 for an example of how to program and configure RX content.
   b. If PRU_ICSS_PRU0_ED_CHm_CFG1_REG[31-16] PRU0_ED_RX_EN_COUNTERm = 0, poll either r31[21, 13, or 5] (tx_global_reinit_active/busy[2,1,0]) or PRU_ICSS_PRU0_ED_TX_CFG_REG[7, 6, or 5] PRU0_ED_BUSY_m (where m = 0 to 2, indicates channel number) for when TX is complete

Note

The PERIF<m>_CLK Peripheral I/F requires that PERIF<m>_CLK be in a high state at the beginning of a new transaction. If the clock ended the continuous loading transmission in low state, then the clock needs to be reset before sending more data. The steps to reset PERIF<m>_CLK are:

1. Set R31[19] = 1 (tx_global_reinit) to reset clock high
2. Wait until PRU0_ED_BUSY_m is cleared
3. Re-configure R30[20-19] (clk_mode), since reinit will reset the clk_mode to "Free-running/stop-high" mode

7.3.5.2.2.3.6.4.4 RX - Auto Arm or Non-Auto Arm

Follow these steps to configure the Peripheral I/F channel(s) to receive data:

1. Configure RX and frame size:

   a. PRU_ICSS_PRU0_ED_CHm_CFG0_REG[27-16] PRU0_ED_RX_FRAME_SIZEm (where n = 0 or 1 and m = 0 to 2)
2. Configure start bit polarity:
   a. PRU_ICSS_PRU0_ED_RX_CFG_REG[3] RX_SB_POL (PRU0 or PRU1)
   b. For the non-auto arm use case, set r30[26, 25, 24] = 1 (rx_en<m>)
   c. For the auto arm use case, rx_en<m> will be automatically enabled at the end of a TX operation when PRU_ICSS_PRU0_ED_CHm_CFG1_REG[31-16] PRU0_ED_RX_EN_COUNTERm > 0 (where n = 0 or 1 and m = 0 to 2)
3. RX FIFO will start filling on the first start bit (PERIF<m>_IN = 1). The data will be captured on the positive edge of the PERIF<m>_CLK and shifted into the LSB position of the 8-bit shadow register.
4. Poll for r31[26, 25, 24] (val<m>) assertion. The valid flag will be asserted when n bits of data (determined by PRU_ICSS_PRU0_ED_RX_CFG_REG[2-0] PRU0_ED_RX_SAMPLE_SIZE) have been collected.
5. Fetch data by reading r31[23-16, 15-8, 7-0] (rx_data_out<m>). The data will remain constant for one data frame, and PRU must read data and clear valid flag within this time. Otherwise, an overflow will occur – r31[29, 28, 27] (ovf<m>) = 1 - indicating that val<m> has been continuously asserted for longer than one data frame.
6. The clock will be stopped based on the r30[20-19] (clk_mode) configured before the start of the RX operation.
7. Clear r30[26, 25, 24] (rx_en<m>) to disable RX mode. All counters and flags will be reset.

7.3.5.3 PRU-ICSS RAM Index Allocation

The PRU-ICSS module includes integrated ECC Aggregator module to test ECC functionality.

Table 7-61 shows the mapping of the RAM IDs to the ECC RAMs serviced by the ECC Aggregator.

Table 7-61. Mapping of the RAM IDs to the ECC RAMs

RAM Index	ECC RAM Prefix	Description
0	pr<k>_dram0	Data RAM0 (8KB)
1	pr<k>_dram1	Data RAM1 (8KB)
2	pr<k>_pru0_iram	PRU0 Instruction Memory (16KB)
3	pr<k>_pru1_iram	PRU1 Instruction Memory (16KB)
4	pr<k>_ram	Shared Data RAM2 (32KB)

7.3.6 PRU-ICSS Broadside Accelerators

7.3.6.1 PRU-ICSS Broadside Accelerators Overview

The PRU-ICSS supports a broadside interface, which uses the XFR (XIN, XOUT, or XCHG) instruction to transfer the contents of PRUn (where n = 0 or 1) registers to or from accelerators. This interface enables up to 31 registers (R0-R30, or 124 bytes) to be transferred in a single instruction. This section details the various accelerators that are available to the PRUn through the broadside interface.

Each of those functions have a unique XIN ID to determine which operation will occur. For more information see Table 7-37.

7.3.6.2 PRU-ICSS Data Processing Accelerators Functional

7.3.6.2.1 PRU Multiplier with Accumulation (MPY/MAC)

This section describes the MAC (multiplier with accumulation) module integrated to PRU0/PRU1 cores.

Each of the two PRU cores (PRU0/PRU1) has a designated unsigned multiplier with accumulation (MPY/MAC). The MAC supports two modes of operation: Multiply Only and Multiply and Accumulate.

The MAC is directly connected with the PRU internal registers R25-R29 and uses the broadside load/store PRU interface and XFR instructions to both control and mode of the MAC and import the multiplication results into the PRU.

The PRU MPY/MAC features are:

- Configurable Multiply Only and Multiply and Accumulate functionality via PRU register R25
- 32-bit operands with direct connection to PRU registers R28 and R29
- 64-bit result (with carry flag) with direct connection to PRU registers R26 and R27
- One clock cycle per operation
- PRU broadside interface and XFR instructions (XIN, XOUT) allow for importing multiplication results and initiating accumulate function

7.3.6.2.1.1 PRU MAC Operations

7.3.6.2.1.1.1 PRU versus MAC Interface

The MAC directly connects with the PRU internal registers R25-R29 through use of the PRU broadside interface and XFR instructions. Figure 7-43 shows the functionality of each register.

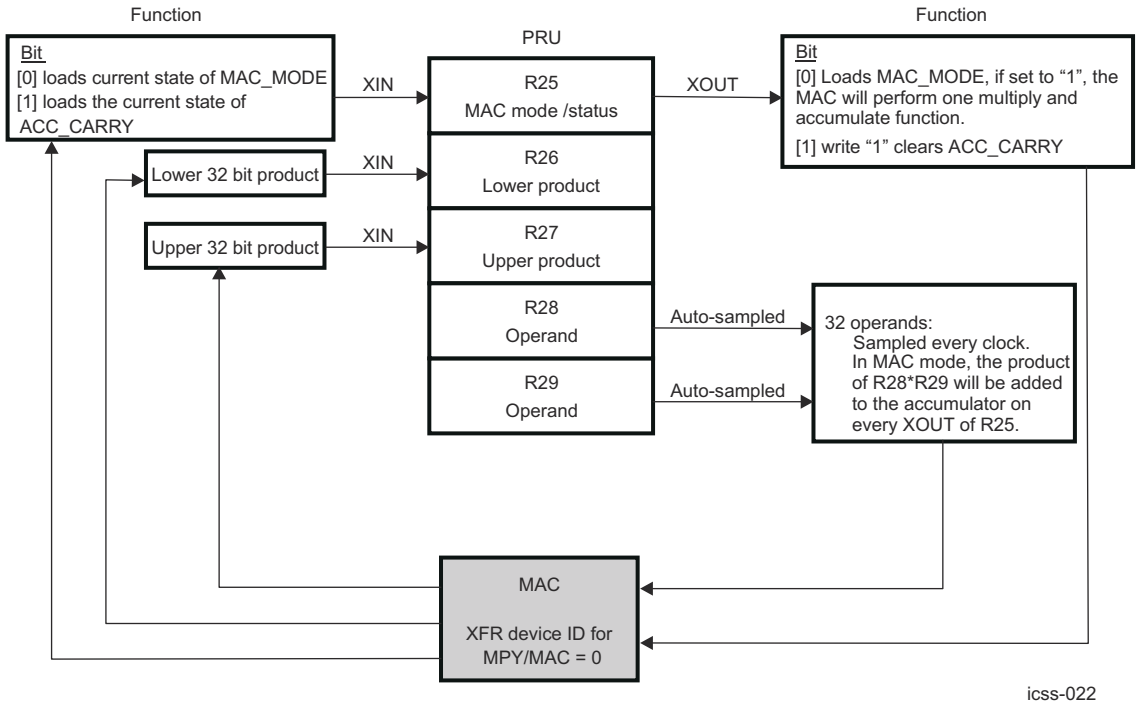


Figure 7-43. Integration of the PRU and MPY/MAC

The XFR instructions (XIN and XOUT) are used to load/store register contents between the PRU core and the MAC. These instructions define the start, size, direction of the operation, and device ID. The device ID number corresponding to the MPY/MAC is shown in Table 7-62.

Table 7-62. MPY/MAC XFR ID

Device ID	Function
0	Selects MPY/MAC

The PRU register R25 is mapped to the MAC_CTRL_STATUS register (Table 7-63). The MAC's current status (MAC_MODE and ACC_CARRY states) is loaded into R25 using the XIN command on R25. The PRU sets the MAC's mode and clears the ACC_CARRY using the XOUT command on R25.

Table 7-63. MAC_CTRL_STATUS Register (R25) Field Descriptions

Bit	Field	Description
7-2	RESERVED	Reserved
1	ACC_CARRY	Write 1 to clear. It is sticky. It is set 0 cycles after the event. 0h: 64-bit accumulator carry has not occurred 1h: 64-bit accumulator carry occurred
0	MAC_MODE	0h: Accumulation mode disabled and accumulator is cleared 1h: Accumulation mode enabled

The two 32-bit operands for the multiplication are loaded into R28 and R29. These registers have a direction connection with the MAC. Therefore, XOUT is not required to load the MAC. In multiply mode, the MAC samples these registers every clock cycle. In multiply and accumulate mode, the MAC samples these registers every XOUT R25[7-0] transaction when MAC_MODE = 1.

The product from the MAC is linked to R26 (lower 32 bits) and R27 (upper 32 bits). The product is loaded into register R26 and R27 using XIN.

7.3.6.2.1.1.2 Multiply only mode(default state), MAC_MODE = 0

The Figure 7-44 summarizes the MAC operation in "Multiply-only"mode, in which the MAC multiplies the contents of R28 and R29 on every clock cycle.

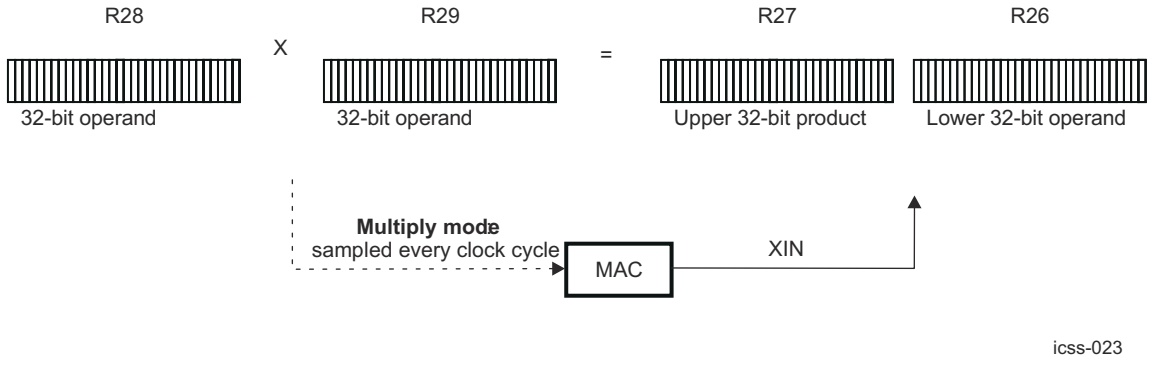


Figure 7-44. MAC Multiply-only Mode- Functional Diagram

7.3.6.2.1.1.2.1 Programming PRU MAC in "Multiply-ONLY" mode

The following steps are performed by the PRU firmware for multiply-only mode:

1. 1. Enable multiply only MAC_MODE.
    a. (a) Clear R25[0] for multiply only mode.
    b. (b) Store MAC_MODE to MAC using XOUT instruction with the following parameters:

- Device ID = 0
- Base register = R25
- Size = 1

2. 2. Load operands into R28 and R29.
3. 3. Delay at least 1 PRU cycle before executing XIN in step 4.
4. 4. Load product into PRU using XIN instruction on R26, R27.

Repeat steps 2 and 4 for each new operand.

7.3.6.2.1.1.3 Multiply and Accumulate Mode, MAC_MODE = 1

The Figure 7-45 summarizes the MAC operation in "Multiply and Accumulate" mode. On every XOUT R25_REG[7-0] transaction, the MAC multiplies the contents of R28 and R29, adds the product to its accumulated result, and sets ACC_CARRY if an accumulation overflow occurs.

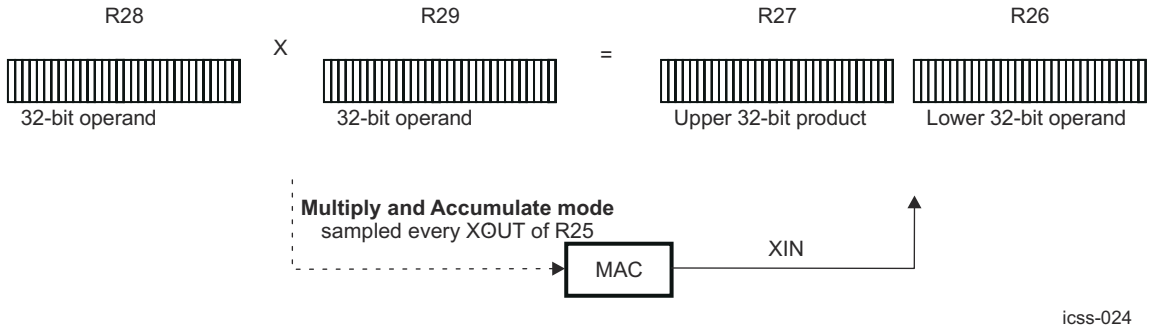


Figure 7-45. MAC Multiply and Accumulate Mode Functional Diagram

7.3.6.2.1.1.3.1 Programming PRU MAC in Multiply and Accumulate Mode

The following steps are performed by the PRU firmware for multiply and accumulate mode:

1. Enable multiply and accumulate MAC_MODE.

(a) Set R25[1-0] = 1 for accumulate mode.

(b) Store MAC_MODE to MAC using XOUT instruction with the following parameters:

- Device ID = 0

- Base register = R25

- Size = 1

2. Clear accumulator and carry flag.

(a) Set R25[1-0] = 3 to clear accumulator (R25[1]=1) and preserve accumulate mode (R25[0]=1).

(b) Store accumulator to MAC using XOUT instruction on R25.

3. Load operands into R28 and R29.

4. Multiply and accumulate, XOUT R25[1-0] = 1

Repeat step 4 for each multiply and accumulate using same operands.

Repeat step 3 and 4 for each multiply and accumulate for new operands.

5. Load the accumulated product into R26, R27, and the ACC_CARRY status into R25 using the XIN instruction.

Note

Steps one and two are required to set the accumulator mode and clear the accumulator and carry flag.

7.3.6.2.2 PRU CRC16/32 Module

Each of the PRU0/PRU1 cores have a designated CRC16/32 module.

In general, CRC adds error detection capability to communication systems. The CRC encoder appends redundant bits (or CRC bits) to the systematic data message. During reception of the data message, the received data is also encoded with the same CRC encoder. The 2 sets of CRC bits are compared together. If they match, there were no transmission errors; and if they don't match, a transmission error has been detected.

CRC16/32 supports the following features:

- Supports CRC32:
  - $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^{8}+x^{7}+x^{5}+x^{4}+x^{2}+x+1$
- Supports CRC16:
  - $x^{16}+x^{15}+x^{2}+1$
- Supports CRC16 - CCITT:
  - $x^{16}+x^{12}+x^{5}+1$
- PRU broadside interface and XFR instructions (XIN, XOUT) allow for importing CRC results and executing accumulate function

7.3.6.2.2.1 PRU and CRC16/32 Interface

The CRC16/32 module directly connects with the PRU internal registers R25-R29 through use of the PRU broadside interface and XFR instructions. Table 7-64 shows the functionality of each register.

The XFR instructions (XIN/XOUT/XCHG) are used to load/store register contents between the PRU core and the CRC16/32 module. These instructions define the start, size, direction of the operation, and device ID. The XFR device ID number corresponding to the CRC16/32 module is 1.

Table 7-64. CRC Register to PRU Port Mapping

CRC Register	R/W	Description	PRU Mapping
CRC_CFG	W	Always write all 4 bytes. bit [0] CRC32_ENABLE: 0: CRC16 mode is selected. Hardware will auto-set init state of CRC_SEED to 0000_0000h. However, for CRC16-CCITT software will need to write the init state of FFFF_FFFFh to CRC_SEED. Note: The CRC16 result value is only 16-bits. 1: CRC32 mode is selected. Hardware will auto-set init state of CRC_SEED will be FFFF_FFFFh. bit [1] CRC_32B_NOT_EMPTY: 0: CRC 32Byte buffer is empty 1: CRC 32Byte buffer is not empty bit [2] CRC16_MOD_ENABLE: 0: CRC16 ($x^{16}+x^{15}+x^{2}+1$) 1: CRC16-CCITT ($x^{16}+x^{12}+x^{5}+1$) - Note: CRC32_ENABLE field must = 0.	R25
CRC_DATA_8_BFLIP	R	8-bit flip of CRC_DATA. CRC_DATA_8_BFLIP has the same byte order as CRC_DATA[31-0], but each byte has all bits flipped. CRC_DATA_32_FLIP[7-0] = CRC_DATA[0-7] CRC_DATA_32_FLIP[15-8] = CRC_DATA[8-15] CRC_DATA_32_FLIP[23-16] = CRC_DATA[16-23] CRC_DATA_32_FLIP[31-24] = CRC_DATA[24-31] For CRC16, only CRC_DATA_8_BFLIP[15-0] are valid. No auto reset on CRC_DATA_8_BFLIP read.	R27
CRC_SEED	W	CRC SEED value. Hardware will auto-initialize the CRC_SEED value to 0000_0000h for CRC16 and FFFF_FFFFh for CRC32. Software only needs to initialize CRC_SEED if a different default value is required. For CRC16-CCITT, software needs to update initial CRC_SEED value to FFFF_FFFFh. Always write 4 bytes. Note: Reading the CRC_DATA register will reset the CRC value to the CRC_SEED state.	R28

Table 7-64. CRC Register to PRU Port Mapping (continued)

CRC Register	R/W	Description	PRU Mapping
CRC_DATA_32_BFLIP	R	Full 32-bit flip of CRC_DATA CRC_DATA_32_BFLIP[0] = CRC_DATA[31] … CRC_DATA_32_BFLIP[31] = CRC_DATA[0] For CRC16, only CRC_DATA_32_BFLIP[31-16] are valid. No auto reset on CRC_DATA_32_BFLIP read.	R28
CRC_DATA	RW	For Write, must use a fixed width throughout the session. The CRC module supports lower 8-bit, or lower 16-bit, or full 32-bit data widths. For Read, LSB or CRC_DATA[0] is first bit on the wire. For Read, reset the CRC_DATA back to CRC_SEED state. Note: Firmware must add 1 to 2 NOPs after the last XOUT to the XIN. For CRC16, only CRC_DATA[15-0] is valid.	R29

7.3.6.2.2.2 CRC Programming Model

The following steps are performed by the PRU firmware to use the CRC module:

Step1: Configuration (optional)

1. Configure CRC type:
   For CRC32 operation, set CRC32_ENABLE using XOUT instruction with the following parameters:
   - Device ID = 1
   - Base register = R25
   - Size = 1
2. Update CRC_SEED, if required using XOUT with the following parameters:
   - Device ID = 1
   - Base register = R28
   - Size = 1 to 4

Step 2:

1. Load new CRC data into R29
2. Push CRC data to the CRC16/32 module using XOUT with the following parameters:
   - Device ID = 1
   - Base register = R29
   - Size = 1 to 4
3. 1 or 2 NOPS
4. Load the accumulated CRC result into the PRU using the XIN instruction with the following parameters:
   - Device ID = 1
   - Base register = R29
   - Size = 4

Repeat Step 2, numbers 1 and 2 for each new CRC data.

Note

When a session starts, the PRU firmware must use the same write data width throughout the session.

7.3.6.2.2.3 PRU and CRC16/32 Interface (R9:R2)

The PRU-ICSS system implements a new wide 32-Bytes data path. The firmware can perform one XOUT of 32-Bytes, the hardware will feed the CRC16/32 4-Bytes at a time. This will take 20 clock cycles for CRC16 and 12 clock cycles for CRC32 for a 32-Bytes XOUT.

Table 7-65. PRU Register to XFR Mapping

PRU Register	XFR ID	Domain/Function	Description
R9:R2 Data	1	Data	XOUT Only 1-Byte to 32-Bytes in size LSB packed and no gaps, for example: 32-Bytes push R9:R2

Table 7-65. PRU Register to XFR Mapping (continued)

PRU Register	XFR ID	Domain/Function	Description
			16-Bytes push R5:R2
			4-Bytes push R2
			7-Bytes push R3(b2.b0):R2
			1-Bytes push R2(b0)

7.3.6.2.3 PRU-ICSS Scratch Pad Memory

The PRU-ICSS supports a scratch pad with up to four independent banks accessible by the PRU cores. The PRU cores interact with the scratch pad through broadside load/store PRU interface and XFR instructions. The scratch pad can be used as a temporary place holder for the register contents of the PRU.

7.3.6.2.3.1 PRU0/1 Scratch Pad Overview

The PRU-ICSS scratch pad supports the following features:

- PRU0 and PRU1 cores have three Scratch Pad banks of 30, 32-bit registers (R29 to R0)
- Flexible load/store options:
  - Load/store one byte of R<n> or load/store (R29 to R0) to Bank0, Bank1, Bank2 or Bank3
  - User-defined start byte and length of the transfer
  - Length of transfer ranges from one byte of a register to the entire register content (R29 to R0)
  - Simultaneous transaction supported between PRU0 <-> Bank<n> and PRU1 <-> Bank<m>
- XFR (XIN/XOUT/XCHG) instructions operate in one clock cycle
- Optional XIN/XOUT shift functionality allows remapping of registers (R<n> -> R<m>) during load store operation

Figure 7-46 shows a simplified model of the Scratch Pad and PRU cores integration.

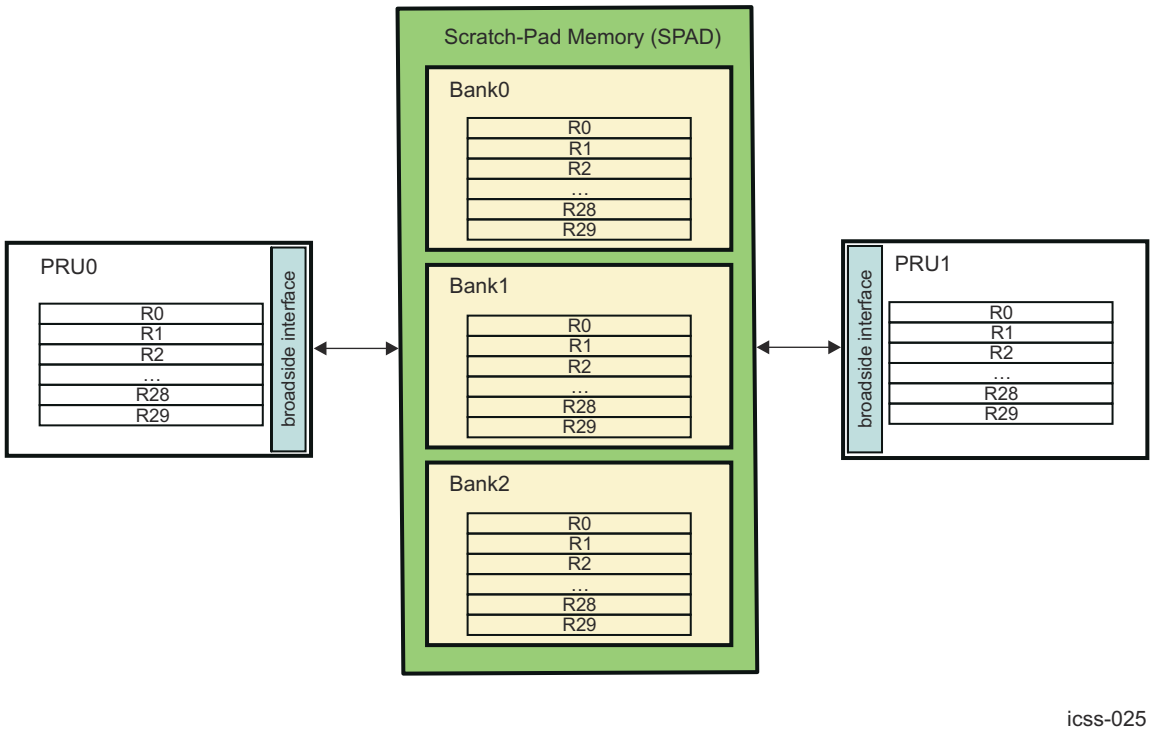


Figure 7-46. Scratch Pad and PRU Integration

7.3.6.2.3.2 PRU0 /1 Scratch Pad Operations

XFR instructions are used to load/store register contents between the PRU cores and the scratch pad banks. These instructions define the start, size, direction of the operation, and device ID. The device ID corresponds to the external source or destination (either a scratch pad bank or the other PRU core). The device ID numbers are shown in Table 7-66.

Table 7-66. Scratch Pad XFR ID

Device ID	Function/Operation
10	Selects Bank0
11	Selects Bank1
12	Selects Bank2

A collision occurs when two XOUT commands simultaneously access the same asset or device ID. Table 7-67 shows the priority assigned to each operation when a collision occurs.

Table 7-67. Scratch Pad XFR Collision and Stall Conditions

Operation	Collision and Stall Handling
PRU<n> XOUT (->) bank[j]	If both PRU cores access the same bank simultaneously, PRU0 is given priority. PRU1 will temporarily stall until the PRU0 operation completes.

7.3.6.2.3.2.1 Optional XIN/XOUT Shift

The optional XIN/XOUT shift functionality allows register contents to be remapped or shifted within the destination's register space. For example, the contents of PRU0 R6-R8 could be remapped to Bank1 R10-12.

The shift feature is enabled or disabled through the PRU subsystem level register PRU_ICSS_SPP_REG[1] XFR_SHIFT_EN bit. When enabled, R0[4-0] (internal to the PRU) defines the number of 32-bit registers in which content is shifted in the scratch pad bank. Note that scratch pad banks do not have registers R30 or R31.

7.3.6.2.3.2.2 Scratch Pad Operations Examples

The following PRU firmware examples demonstrate the shift functionality. Note: These assume the XFR_SHIFT_EN bit of the PRU_ICSS_SPP_REG register of the PRU-ICSS CFG register space has been set.

XOUT Shift By 4 Registers

Store R4:R7 to R8:R11 in Bank0:

- Load 4 into R0.b0
- XOUT using the following parameters:
  - Device ID = 10
  - Base register = R4
  - Size = 16

XOUT Shift By 9 Registers, With Wrap Around

Store R25:R29 to R4:R9 in Bank1:

- Load 9 into R0.b0
- XOUT using the following parameters:
  - Device ID = 11
  - Base register = R25
  - Size = 20

XIN Shift By 10 Registers

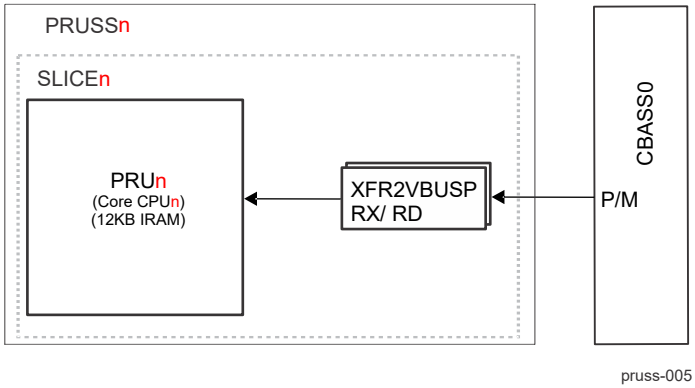
Load R14:R16 from Bank2 to R4:R6:

- Load 10 into R0.b0
- XIN using the following parameters:
  - Device ID = 12
  - Base register = R4
  - Size = 12

7.3.6.3 PRU-ICSS Data Movement Accelerators Functional

7.3.6.3.1 PRU-ICSS XFR2VBUS Hardware Accelerator

The PRU core can write and read data packets to and from port queues, located in the MSMC SRAM into PRU core registers via XFR2VBUS hardware accelerator. Each of the PRU-ICSS Slices has implemented two RX XFR2VBUS hardware accelerators.



1. n reprsents a valid instance of PRU in a domain.

Figure 7-47. XFR2VBUS Hardware Accelerator

Supported features:

- 2 x XFR2VBUS RX threads

XFR2VBUS RX buffer features:

- 1 x 64 Byte deep RX/Read buffer
  - 4 Byte, 32 Byte, or 64 Byte read size per RD (read) command
  - Optional automatic read command with incrementing address on pop of read data
  - 32 Byte optimization mode available

The ownership of commands and data is flexible. The XFR2VBUS accelerator is shared between PRU cores. Status is available to both cores.

Note: The ownership should be preplanned and static per use model.

The XFR2VBUS is a simple hardware accelerator wich is used to get the lowest read round trip latency from MSMC and to decouple the latency seen by the PRU. Each XFR2VBUS instance is connected to the CBASS0.

The PRU-ICSS system has a total of 2 XFR2VBUS RX hardware accelerators.

7.3.6.3.1.1 Blocking Conditions

The only blocking condition is caused when the VBUSM command/data FIFO is full. It is required that the external bandwidth is very high. All egress commands and data should get sent without head of line blocking. Based on arbitration some delay is possible.

7.3.6.3.1.2 Read Operation with Auto Disabled

The XFR2VBUS supports 1 command in its command FIFO, 1 XOUT to define the address and size (4 Byte, 32 Byte, 64 Byte, aligned). This will cause the VBUSP read command to be issued. Only 1 read command can be in flight. The read address defines the offset of the 64 Byte of read data. The read size defines the size of the transfer, 4 Byte, 32 Byte, 64 Byte, aligned Offset + size must be aligned to 32 Byte width of the bus. 1 XIN, the software can see the status of command FIFO and read data FIFO.

Note: XIN of the read data will fully pop the data, independent of XIN size.

7.3.6.3.1.3 Read Operation with Auto Enabled

The same features as Auto Disabled are valid with the following exceptions:

64 Byte mode:

- Address needs to be MOD 0x40/64 Byte aligned
- Size needs to be 64 Byte
- 1 XOUT to define the start address, which needs to be MOD 0x40/64 Byte aligned

- The XFR2VBUS will issue 1 new command every time this is 64 Bytes available in the read data FIFO
- 1 XIN of read data will cause a new command to be issued since, 64 Bytes are available
- To stop the issuing new read commands, disable auto mode

32 Byte mode:

- Size needs to be 32 Byte
- 1 XOUT to define the start address, which needs to be MOD 0x20/32 Byte aligned
- The XFR2VBUS will issue 1 new command every time this is 32 Bytes available in the read data FIFO
- 1 XIN of read data will cause a new command to be issued since, 32 Bytes are available
- To stop the issuing of new read command, disable auto mode

Note: XIN of the read data will fully pop the data, independent of XIN size.

7.3.6.3.1.4 PRU to XFR2VBUS Interface

RD_ID0 = 0x60

RD_ID1 = 0x61

Table 7-68. Read Commands

PRU Register	BS ID	Access Type	Register	Notes
R17-R2	RD_ID1/0	XIN	RD_DATA	Read Data
R18[0]	RD_ID1/0	XOUT	RD_AUTO	Read Auto Mode If 0 -> 1, must write RD_ADDR If 1 -> 0, must not write RD_ADDR, must drain RD_DATA/RD_CMD If 0 -> 0, must write RD_ADDR When set, every RD_DATA pop will cause a new read command and read address to increment by 0x20 for the next read command if size is set to 32 Bytes 0x40 for the next read command if size is set to 64 Bytes In this case, user must set the address to be ether mod 0x20 or 0x40. 4 Byte mode is not supported.
R18[2-1]	RD_ID1/0	XOUT	RD_SIZE	Read Size 0h: 4 Bytes 1h: Reserved 2h: 32 Bytes 3h: 64 Bytes
R18[0]	RD_ID1/0	XIN	RD_BUSY	Read Busy Status 0h: Idle 1h: Active (RD CMD FIFO LEVEL !=0) or (RD DATA FIFO LEVEL !=0)
R18[1]	RD_ID1/0	XIN	RD_CMD_FL	Read command FIFO Level 0h: Empty 1h: Occupied Note: It only pop the read command FIFO after the read data has arrived
R18[2]	RD_ID1/0	XIN	RD_DATA_FL	Read data FIFO Level 0h: Empty 1h: Occupied 32 byte or 64 byte Note: In 64 byte mode, the user must wait for RD_MST_REQ = 0h before reading the FIFO.

Table 7-68. Read Commands (continued)

PRU Register	BS ID	Access Type	Register	Notes
R18[3]	RD_ID1/0	XIN	RD_MST_REQ	RD MST RED 0h = Last data has been latched 1h = Last data is still in flight Note: In Auto mode, the user must insure that this bit is 0h and wait an additional NOP before user disables Auto mode to prevent a race condition.
R20:R19	RD_ID1/0	XOUT	RD_ADDR	Read address 48-bits 0x20 for the next read command if size is set to 32 Bytes 0x40 for the next read command if size is set to 64 Bytes The address can be the full 48-bits or just the lower 32-bits of the address, it will use the current state of the upper 16-bits

7.3.6.3.1.5 XFR2VBUS Programming Model

Read:

- Wait RD_BUSY = 0h
- XOUT R18 (configure RD_AUTO/ RD_SIZE); R19 (RD_ADDR)
- Wait WR_BUSY = 0h OR RD_DATA_FL = 1h
- XIN RD_DATA (Repeat if RD_AUTO is enabled and need new RD_DATA, must always check RD_DATA_FL before XIN RD_DATA)

7.3.7 PRU-ICSS Local INTC

The PRU-ICSS interrupt controller (INTC) maps interrupts coming from different parts of the device (mapped to PRU-ICSS) to a reduced set of PRU-ICSS interrupt channels.

The interrupt controller has the following features:

- Capturing up to 64 System Events (inputs):
- Supports up to 10 output interrupt channels.
- Generation of 10 Host Interrupts
  - 2 Host Interrupts shared between the PRUs (PRU0 and PRU1).
  - 8 Host Interrupts exported from the PRU-ICSS internal INTC for signaling the device level interrupt controllers (pulse and level provided).
- Each event can be enabled and disabled.
- Each host event can be enabled and disabled.
- Hardware prioritization of events.

7.3.7.1 PRU-ICSS Interrupt Controller Functional Description

The PRU-ICSS INTC supports up to 64 interrupts from different peripherals and PRUs. The INTC maps these events to 10 channels inside the INTC (see Figure 7-48). Interrupts from these 10 channels are further mapped to 10 Host Interrupts.

- Any of the 64 internal interrupts can be mapped to any of the 10 channels.
- Multiple interrupts can be mapped to a single channel.
- An interrupt should not be mapped to more than one channel.
- Any of the 10 channels can be mapped to any of the 10 host interrupts. It is recommended to map channel “x” to host interrupt “x”, where x is from 0 to 9.
- A channel should not be mapped to more than one host interrupt
- For channels mapping to the same host interrupt, lower number channels have higher priority.
- For interrupts on same channel, priority is determined by the hardware interrupt number. The lower the interrupt number, the higher the priority.
- Host Interrupt 0 is connected to bit 30 in register 31 (R31) of PRU0 and PRU1 in parallel.
- Host Interrupt 1 is connected to bit 31 in register 31 (R31) for PRU0 and PRU1 in parallel.
- Host Interrupts 2 through 9 exported from PRU-ICSS and mapped to device level interrupt controllers.

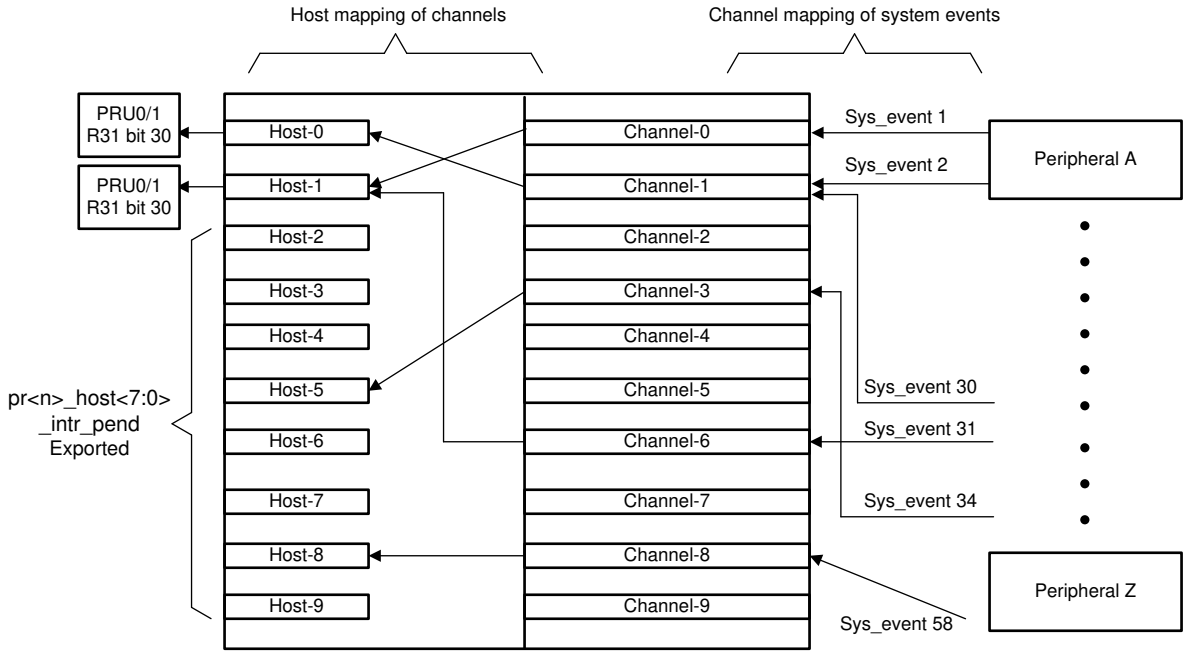


Figure 7-48. PRU-ICSS Interrupt Controller Block Diagram

7.3.7.1.1 PRU-ICSS Interrupt Controller Events

All PRU-ICSS system events are interrupt inputs. System events 32 through 63 are external and generated from different peripherals. System events 0 through 31 are internal for PRU-ICSS sources.

7.3.7.1.2 PRU-ICSS Interrupt Controller System Events Flow

The ICSS_INTC module controls the event mapping to the host interrupt interface. Events are generated by the device peripherals or PRUs. The INTC receives the internal interrupts and maps them to internal channels. The channels are used to group interrupts together and to prioritize them. These channels are then mapped onto the host interrupts. Interrupts from the system side are active high in polarity.

The INTC encompasses many functions to process the system interrupts and prepare them for the host interface. These functions are: processing, enabling, status, channel mapping, host interrupt mapping, prioritization, and host interfacing. Figure 7-49 illustrates the flow of interrupts through the functions to the host. The following subsections describe each part of the flow.

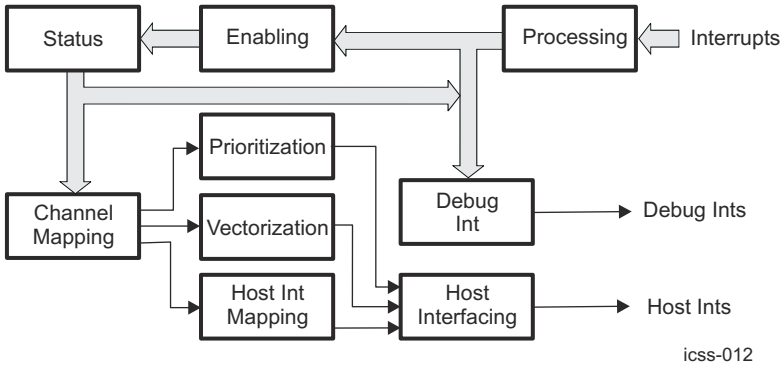


Figure 7-49. Flow of System Interrupts to Host

7.3.7.1.2.1 PRU-ICSS Interrupt Processing

This block does following tasks:

- Synchronization of slower and asynchronous interrupts
- Conversion of polarity to active high
- Conversion of interrupt type to pulse interrupts

After the processing block, all interrupts will be active high pulses.

7.3.7.1.2.1.1 PRU-ICSS Interrupt Enabling

The next stage of INTC is to enable interrupts based on programmed settings. The following sequence has to be followed to enable interrupts:

- Enable required interrupts: System interrupts that are required to get propagated to host are to be enabled individually by writing to [9-0] ENABLE_SET_INDEX bit field in the interrupt enable indexed set register (ICSS_INTC_ENABLE_SET_INDEX_REG). The interrupt to enable is the index value written. This sets the Enable Register bit of the given index.
- Enable required host interrupts: By writing 1h to the appropriate bit of the [9-0] HINT_ENABLE_SET_INDEX bit field in the host interrupt enable indexed set register (ICSS_INTC_HINT_ENABLE_SET_INDEX_REG), enable the required host interrupts. The host interrupt to enable is the index value written. This enables the host interrupt output or triggers the output again if that host interrupt is already enabled.
- Enable all host interrupts: By setting the [0] ENABLE_HINT_ANY bit in the global enable register (ICSS_INTC_GLOBAL_ENABLE_HINT_REG) to 1h, all host interrupts will be enabled. Individual host interrupts are still enabled or disabled from their individual enables and are not overridden by the global enable.

7.3.7.1.2.2 PRU-ICSS Interrupt Status Checking

The next stage is to capture which interrupts are pending. There are two kinds of pending status: raw status and enabled status. Raw status is the pending status of the interrupt without regards to the enable bit for the

interrupt. Enabled status is the pending status of the interrupts with the enable bits active. When the enable bit is inactive, the enabled status will always be inactive. The enabled status of interrupts is captured in interrupt status enabled/clear registers (ICSS_INTC_ENA_STATUS_REG0 to ICSS_INTC_ENA_STATUS_REG4).

Status of interrupt 'N' is indicated by the N-th bit of ICSS_INTC_ENA_STATUS_REG0 to ICSS_INTC_ENA_STATUS_REG4. Since there are 160 interrupts, five 32-bit registers are used to capture the enabled status of interrupts. The pending status reflects whether the interrupt occurred since the last time the status register bit was cleared. Each bit in the status register can be individually cleared.

7.3.7.1.2.3 PRU-ICSS Interrupt Channel Mapping

The INTC has 10 internal channels to which enabled interrupts can be mapped. Channel 0 has highest priority and channel 9 has the lowest priority. Channels are used to group the interrupts into a smaller number of priorities that can be given to a host interface with a very small number of interrupt inputs.

When multiple interrupts are mapped to the same channel their interrupts are ORed together so that when either is active the output is active. The channel map registers (ICSS_INTC_CH_MAP_REGi) define the channel for each interrupt. There is one register per 4 interrupts; therefore, there are 16 channel map registers for a of 64 interrupts. The channel for each interrupt can be set using these registers.

7.3.7.1.2.3.1 PRU-ICSS Host Interrupt Mapping

The hosts can be the local PRU processors (PRU0 and PRU1) as well as device processors located outside PRU-ICSS such as ARM, etc. The 10 channels from the INTC can be mapped to any of the 10 Host interrupts. The Host map registers (ICSS_INTC_HINT_MAP_REG0 to ICSS_INTC_HINT_MAP_REG4) define the channel for each interrupt. There is one register per 4 channels; therefore, there are 3 host map registers for 10 channels. When multiple channels are mapped to the same host interrupt, then prioritization is done to select which interrupt is in the highest-priority channel and which should be sent first to the host.

7.3.7.1.2.3.2 PRU-ICSS Interrupt Prioritization

The next stage of the INTC is prioritization. Since multiple interrupts can feed into a single channel and multiple channels can feed into a single host interrupt, it is necessary to read the status of all interrupts to determine the highest priority interrupt that is pending. The INTC provides hardware to perform this prioritization with a given scheme so that software does not have to do this. There are two levels of prioritizations:

- The first level of prioritization is between the active channels for a host interrupt. Channel 0 has the highest priority and channel 9 has the lowest. So the first level of prioritization picks the lowest numbered active channel.
- The second level of prioritization is between the active interrupts for the prioritized channel. The interrupt in position 0 has the highest priority and interrupt 159 has the lowest priority. So the second level of prioritization picks the lowest position active interrupt.

This is the final prioritized interrupt for the host interrupt and is stored in the global prioritized innterrupt register (ICSS_INTC_GLB_PRI_INTR_REG). The highest priority pending interrupt with respect to each host interrupts can be obtained using the host interrupt prioritized interrupt registers (ICSS_INTC_PRI_HINT_REGj where j = 0 to 19).

7.3.7.1.2.4 PRU-ICSS Interrupt Nesting

The INTC can also perform a nesting function in its prioritization. Nesting is a method of disabling certain interrupts (usually lower-priority interrupts) when an interrupt is taken so that only those desired interrupts can trigger to the host while it is servicing the current interrupt. The typical usage is to nest on the current interrupt and disable all interrupts of the same or lower priority (or channel). Then the host will only be interrupted from a higher priority interrupt.

The nesting is done in one of three methods:

1. Nesting for all host interrupts, based on channel priority: When an interrupt is taken, the nesting level is set to its channel priority. From then, that channel priority and all lower priority channels will be disabled from generating host interrupts and only higher priority channels are allowed. When the interrupt is completely serviced, the nesting level is returned to its original value. When there is no interrupt being serviced, there are no channels disabled due to nesting. The global nesting level register

(ICSS_INTC_GLB_NEST_LEVEL_REG) allows the checking and setting of the global nesting level across all host interrupts. The nesting level is the channel (and all of lower priority channels) that are nested out because of a current interrupt.

2. Nesting for individual host interrupts, based on channel priority: Always nest based on channel priority for each host interrupt individually. When an interrupt is taken on a host interrupt, then, the nesting level is set to its channel priority for just that host interrupt, and other host interrupts do not have their nesting affected. Then for that host interrupt, equal or lower priority channels will not interrupt the host but may on other host interrupts if programmed. When the interrupt is completely serviced the nesting level for the host interrupt is returned to its original value. The host interrupt nesting level registers (ICSS_INTC_NEST_LEVEL_REGj where j = 0 to 19) display and control the nesting level for each host interrupt. The nesting level controls which channel and lower priority channels are nested. There is one register per host interrupt.
3. Software manually performs the nesting of interrupts. When an interrupt is taken, the software will disable all the host interrupts, manually update the enables for any or all the interrupts, and then re-enables all the host interrupts. This now allows only the interrupts that are still enabled to trigger to the host. When the interrupt is completely serviced the software must reverse the changes to re-enable the nested out interrupts. This method requires the most software interaction but gives the most flexibility if simple channel based nesting mechanisms are not adequate.

7.3.7.1.2.5 PRU-ICSS Interrupt Status Clearing

After servicing the interrupt (after execution of the ISR), interrupt status is to be cleared. If a interrupt status is not cleared, then another host interrupt may not be triggered or another host interrupt may be triggered incorrectly. It is also essential to clear all interrupts before the PRU is halted as the PRU does not power down unless all the interrupt status are cleared. For clearing the status of an interrupt, whose interrupt number is N, write a 1h to the Nth bit position in the interrupt status enabled/clear registers (ICSS_INTC_ENA_STATUS_REG0 to ICSS_INTC_ENA_STATUS_REG4). Interrupt N can also be cleared by writing the value N into the interrupt status indexed clear register (ICSS_INTC_STATUS_CLR_INDEX_REG).

7.3.7.1.3 PRU-ICSS Interrupt Disabling

At any time, if any interrupt is not to be propagated to the host, then that interrupt should be disabled. For disabling an interrupt whose interrupt number is N, write a 1h to the Nth bit in the interrupt enable clear registers (ICSS_INTC_ENABLE_CLR_REG0 to ICSS_INTC_ENABLE_CLR_REG4). Interrupt N can also be disabled by writing the value N in the interrupt enable clear index register (ICSS_INTC_ENABLE_CLR_INDEX_REG).

7.3.7.2 PRU-ICSS Interrupt Controller Basic Programming Model

Follow these steps to configure the interrupt controller.

1. Set polarity and type of event through the Interrupt Polarity Registers (ICSS_INTC_POLARITY_REG0 to ICSS_INTC_POLARITY_REG4) and the Interrupt Type Registers (ICSS_INTC_POLARITY_REG0 to ICSS_INTC_POLARITY_REG4). Polarity of all interrupts is always high. Type of all interrupts is always pulse (after the processing block).
2. Map event to INTC channel through ICSS_INTC_CH_MAP_REGi (where i=0 to 39) channel mapping registers.
3. Map channel to host interrupt through ICSS_INTC_HINT_MAP_REG0 to ICSS_INTC_HINT_MAP_REG4 registers. Recommended channel “x” to be mapped to host interrupt “x”.
4. Clear interrupt by writing 1h to ICSS_INTC_ENA_STATUS_REG0 to ICSS_INTC_ENA_STATUS_REG4 registers.
5. Enable host interrupt by writing index value to ICSS_INTC_HINT_ENABLE_SET_INDEX_REG register.
6. Enable interrupt nesting if desired.
7. Globally enable all interrupts through register ICSS_INTC_GLOBAL_ENABLE_HINT_REG[0] ENABLE_HINT_ANY bit.

7.3.7.3 PRU-ICSS Interrupt Requests Mapping

The PRU-ICSS Interrupt Controller lines 0 through 31 are mapped to internal events which are generated by PRU-ICSS integrated modules. Lines 32 to 63 are external and generated from different peripherals. Table 7-69 shows mapping of the different PRU-ICSS internally sourced IRQ events to PRU-ICSS INTC interrupt lines 0 through 63.

Table 7-69. PRU-ICSS IP Internal Interrupts

Event Number	Source	
	MII_RT_REG.MII_RT_EVENT_EN =1 mode (default)	MII_RT_REG .MII_RT_EVENT_EN =0 mode
PRU-ICSS INTC		
63	pr0_slv_intr[31]_intr_pend(external)	
62	pr0_slv_intr[30]_intr_pend(external)	
61	pr0_slv_intr[29]_intr_pend(external)	
60	pr0_slv_intr[28]_intr_pend(external)	
59	pr0_slv_intr[27]_intr_pend(external)	
58	pr0_slv_intr[26]_intr_pend(external)	
57	pr0_slv_intr[25]_intr_pend(external)	
56	pr0_slv_intr[24]_intr_pend(external)	
55	pr0_mii1_col and pr0_mii1_txen (external)	pr0_slv_intr<23>_intr_pend(external)
54	PRU1_RX_EOF	pr0_slv_intr<22>_intr_pend(external)
53	MDIO_MII_LINK[1]	pr0_slv_intr<21>_intr_pend(external)
52	PORT1_TX_OVERFLOW	pr0_slv_intr<20>_intr_pend(external)
51	PORT1_TX_UNDERFLOW	pr0_slv_intr<19>_intr_pend(external)
50	PRU1_RX_OVERFLOW	pr0_slv_intr<18>_intr_pend(external)
49	PRU1_RX_NIBBLE_ODD	pr0_slv_intr<17>_intr_pend(external)
48	PRU1_RX_CRC	pr0_slv_intr<16>_intr_pend(external)
47	PRU1_RX_SOF	pr0_slv_intr<15>_intr_pend(external)
46	PRU1_RX_SFD	pr0_slv_intr<46>_intr_pend(external)
45	PRU1_RX_ERR32	pr0_slv_intr<45>_intr_pend(external)
44	PRU1_RX_ERR	pr0_slv_intr<44>_intr_pend(external)
43	pr0_mii0_col and pr0_mii0_txen (external)	pr0_slv_intr<43>_intr_pend(external)
42	PRU0_RX_EOF	pr0_slv_intr<42>_intr_pend(external)
41	MDIO_MII_LINK[0]	pr0_slv_intr<41>_intr_pend(external)
40	PORT0_TX_OVERFLOW	pr0_slv_intr<40>_intr_pend(external)
39	PORT0_TX_UNDERFLOW	pr0_slv_intr<39>_intr_pend(external)
38	PRU0_RX_OVERFLOW	pr0_slv_intr<38>_intr_pend(external)
37	PRU0_RX_NIBBLE_ODD	pr0_slv_intr<37>_intr_pend(external)
36	PRU0_RX_CRC	pr0_slv_intr<36>_intr_pend(external)
35	PRU0_RX_SOF	pr0_slv_intr<35>_intr_pend(external)
34	PRU0_RX_SFD	pr0_slv_intr<34>_intr_pend(external)
33	PRU0_RX_ERR32	pr0_slv_intr<33>_intr_pend(external)
32	PRU0_RX_ERR	pr0_slv_intr<32>_intr_pend(external)
31	pr0_pru_mst_intr[15]_intr_req	
30	pr0_pru_mst_intr[14]_intr_req	
29	pr0_pru_mst_intr[13]_intr_req	
28	pr0_pru_mst_intr[12]_intr_req	
27	pr0_pru_mst_intr[11]_intr_req	
26	pr0_pru_mst_intr[10]_intr_req	

Table 7-69. PRU-ICSS IP Internal Interrupts (continued)

Event Number	Source	
	MII_RT_REG.MII_RT_EVENT_EN =1 mode (default)	MII_RT_REG .MII_RT_EVENT_EN =0 mode
25	pr0_pru_mst_intr[9]_intr_req	
24	pr0_pru_mst_intr[8]_intr_req	
23	pr0_pru_mst_intr[7]_intr_req	
22	pr0_pru_mst_intr[6]_intr_req	
21	pr0_pru_mst_intr[5]_intr_req	
20	pr0_pru_mst_intr[4]_intr_req	
19	pr0_pru_mst_intr[3]_intr_req	
18	pr0_pru_mst_intr[2]_intr_req	
17	pr0_pru_mst_intr[1]_intr_req	
16	pr0_pru_mst_intr[0]_intr_req	
15	pr0_ecap_intr_req	
14	pr0_sync0_out_pend	
13	pr0_sync1_out_pend	
12	pr0_latch0_in (input to PRU-ICSS)	
11	pr0_latch1_in (input to PRU-ICSS)	
10	pr0_pdi_wd_exp_pend	
9	pr0_pd_wd_exp_pend	
8	pr0_digio_event_req	
7	pr0_iep_tim_cap_cmp_pend	
6	pr0_uart0_uint_intr_req	
5	pr0_uart0_utxevt_intr_req	
4	pr0_uart0_urxevt_intr_req	
3	reset_iso_req	
2	pr0_pru1_r31_status_cnt16	
1	pr0_pru0_r31_status_cnt16	
0	pr0_ecc_err_intr	

Table 7-70. AM263Px-Specific PRU-ICSS Interrupt Mapping

Event Number	Source	
	MII_RT_REG.MII_RT_EVENT_EN =1 mode (default)	MII_RT_REG .MII_RT_EVENT_EN =0 mode
	PRU-ICSS INTC	
63	CONTROLSS Output XBAR[15]	
62	CONTROLSS Output XBAR[14]	
61	CONTROLSS Output XBAR[13]	
60	CONTROLSS Output XBAR[12]	
59	CONTROLSS Output XBAR[11]	
58	CONTROLSS Output XBAR[10]	
57	CONTROLSS Output XBAR[9]	
56	CONTROLSS Output XBAR[8]	
55	pr0_mii1_col and pr0_mii1_txen (external)	CONTROLSS Output XBAR[7]
54	PRU0_RX_EOF	CONTROLSS Output XBAR[6]
53	MDIO_MII_LINK[1]	CONTROLSS Output XBAR[5]
52	PORT0_TX_OVERFLOW	CONTROLSS Output XBAR[4]
51	PORT0_TX_UNDERFLOW	CONTROLSS Output XBAR[3]
50	PRU0_RX_OVERFLOW	CONTROLSS Output XBAR[2]

Table 7-70. AM263Px-Specific PRU-ICSS Interrupt Mapping (continued)

Event Number	Source	
	MII_RT_REG.MII_RT_EVENT_EN =1 mode (default)	MII_RT_REG .MII_RT_EVENT_EN =0 mode
49	PRU0_RX_NIBBLE_ODD	CONTROLSS Output XBAR[1]
48	PRU0_RX_CRC	CONTROLSS Output XBAR[0]
47	PRU0_RX_SOF	PRU-ICSS XBAR INTR[15]
46	PRU0_RX_SFD	PRU-ICSS XBAR INTR[14]
45	PRU0_RX_ERR32	PRU-ICSS XBAR INTR[13]
44	PRU0_RX_ERR	PRU-ICSS XBAR INTR[12]
43	pr0_mii0_col and pr0_mii0_txen (external)	PRU-ICSS XBAR INTR[11]
42	PRU0_RX_EOF	PRU-ICSS XBAR INTR[10]
41	MDIO_MII_LINK[0]	PRU-ICSS XBAR INTR[9]
40	PORT0_TX_OVERFLOW	PRU-ICSS XBAR INTR[8]
39	PORT0_TX_UNDERFLOW	PRU-ICSS XBAR INTR[7]
38	PRU0_RX_OVERFLOW	PRU-ICSS XBAR INTR[6]
37	PRU0_RX_NIBBLE_ODD	PRU-ICSS XBAR INTR[5]
36	PRU0_RX_CRC	PRU-ICSS XBAR INTR[4]
35	PRU0_RX_SOF	PRU-ICSS XBAR INTR[3]
34	PRU0_RX_SFD	PRU-ICSS XBAR INTR[2]
33	PRU0_RX_ERR32	PRU-ICSS XBAR INTR[1]
32	PRU0_RX_ERR	PRU-ICSS XBAR INTR[0]
31	pr0_pru_mst_intr[15]_intr_req	
30	pr0_pru_mst_intr[14]_intr_req	
29	pr0_pru_mst_intr[13]_intr_req	
28	pr0_pru_mst_intr[12]_intr_req	
27	pr0_pru_mst_intr[11]_intr_req	
26	pr0_pru_mst_intr[10]_intr_req	
25	pr0_pru_mst_intr[9]_intr_req	
24	pr0_pru_mst_intr[8]_intr_req	
23	pr0_pru_mst_intr[7]_intr_req	
22	pr0_pru_mst_intr[6]_intr_req	
21	pr0_pru_mst_intr[5]_intr_req	
20	pr0_pru_mst_intr[4]_intr_req	
19	pr0_pru_mst_intr[3]_intr_req	
18	pr0_pru_mst_intr[2]_intr_req	
17	pr0_pru_mst_intr[1]_intr_req	
16	pr0_pru_mst_intr[0]_intr_req	
15	pr0_ecap_intr_req	
14	pr0_sync0_out_pend	
13	pr0_sync1_out_pend	
12	pr0_latch0_in (input to PRU-ICSS)	
11	pr0_latch1_in (input to PRU-ICSS)	
10	pr0_pdi_wd_exp_pend	
9	pr0_pd_wd_exp_pend	
8	pr0_digio_event_req	
7	pr0_iep_tim_cap_cmp_pend	
6	pr0_uart0_uint_intr_req	

Table 7-70. AM263Px-Specific PRU-ICSS Interrupt Mapping (continued)

Event Number	Source	
	MII_RT_REG.MII_RT_EVENT_EN =1 mode (default)	MII_RT_REG .MII_RT_EVENT_EN =0 mode
5	pr0_uart0_utxevt_intr_req	
4	pr0_uart0_urxevt_intr_req	
3	reset_iso_req	
2	pr0_pru1_r31_status_cnt16	
1	pr0_pru0_r31_status_cnt16	
0	pr0_ecc_err_intr	

7.3.8 PRU-ICSS UART Module

This section describes an Universal Asynchronous Receive and Transmit (UART) module integrated into the PRU-ICSS subsystem. Hereinafter the module will be referred as PRU-ICSS UART0.

7.3.8.1 PRU-ICSS UART Overview

The PRU-ICSS UART0 peripheral is based on the industry standard TL16C550 asynchronous communications element, which in turn is a functional upgrade of the TL16C450. The information in this chapter assumes that user is familiar with these standards.

Functionally similar to the TL16C450 on power up (single character or TL16C450 mode), the PRU-ICSS UART0 can be placed in an alternate FIFO (TL16C550) mode. This relieves the CPU of excessive software overhead by buffering received and transmitted characters. The receiver and transmitter FIFOs store up to 16 bytes including three additional bits of error status per byte for the receiver FIFO.

The PRU-ICSS UART0 performs serial-to-parallel conversions on data received from a peripheral device and parallel-to-serial conversion on data received from the CPU. The CPU can read the PRU-ICSS UART0 status at any time. The PRU-ICSS UART0 includes control capability and a processor interrupt system that can be tailored to minimize software management of the communications link.

The PRU-ICSS UART0 includes a programmable baud generator capable of dividing the PRU-ICSS UART0 input clock by divisors from 1 to 65535 and producing a 16× reference clock or a 13× reference clock for the internal transmitter and receiver logic.

7.3.8.2 PRU-ICSS UART Environment

This section describes the PRU-ICSS UART0 module interface to the device environment.

7.3.8.2.1 PRU-ICSS UART Pin Multiplexing

Pin multiplexing is controlled using a combination of hardware configuration at device reset and software programmable register settings. For more information on the PRU-ICSS UART0 pin multiplexing, refer to the IO_MUX Registers chapter of the Register Addendum

7.3.8.2.2 PRU-ICSS UART Signal Descriptions

The PRU-ICSS UART0 utilize a minimal number of signal connections to interface with external devices. The PRU-ICSS UART0 signal descriptions are described in Table 7-71.

Table 7-71. PRUSS_UART0 Signal Descriptions

Signal Name	Signal Type	Function
UART0_TXD	Output	Serial data transmit
UART0_RXD	Input	Serial data receive
$\overline{\text{UART0\_CTS}}$	Input	Clear-to-Send handshaking signal
$\overline{\text{UART0\_RTS}}$	Output	Request-to-Send handshaking signal

7.3.8.2.3 PRU-ICSS UART Protocol Description and Data Format

7.3.8.2.3.1 PRU-ICSS UART Transmission Protocol

The PRU-ICSS UART0 transmitter section includes a transmitter hold register (THR), memory mapped in the register UART_RBR_TBR[17-8] TBR_DATA bitfield and a transmitter shift register (TSR), which is NOT memory mapped. When the PRU-ICSS UART0 is in the FIFO mode, THR is a 16-byte FIFO. Transmitter section control is a function of the PRU-ICSS UART0 line control register UART_LCTR. Based on the settings chosen in this register, the PRU-ICSS UART0 transmitter sends the following to the receiving device:

- 1 START bit
- 5, 6, 7, or 8 data bits
- 1 PARITY bit (optional)
- 1, 1.5, or 2 STOP bits

THR receives data from the internal data bus, and when TSR (transmitter shift register) is ready, the PRU-ICSS UART0 moves the data from THR to TSR. The PRU-ICSS UART0 serializes the data in TSR and transmits the data on the UART0_TXD pin.

In the non-FIFO mode, if THR is empty and the Transmitter Holding Register Empty interrupt (THRE) is enabled in the interrupt enable register (UART_INT_EN[1] ETBEI), an interrupt is generated. This interrupt is cleared when a character is loaded into THR or the interrupt identification register UART_INT_FIFO bitfield [3-1] IIR_INTID is read. In the FIFO mode, the interrupt is generated when the transmitter FIFO is empty, and it is cleared when at least one byte is loaded into the FIFO or UART_INT_FIFO[3-1] IIR_INTID bitfield is read.

7.3.8.2.3.2 PRU-ICSS UART Reception Protocol

The PRU-ICSS UART0 receiver section includes a receiver shift register (RSR), that is not memory mapped, and a receiver buffer register (RBR), memory mapped as the register UART_RBR_TBR[7-0] RBR_DATA bitfield. When the PRU-ICSS UART0 is in the FIFO mode, RBR is a 16-byte FIFO. Receiver section control is a function of the PRU-ICSS UART0 line control register - UART_LCTR. Based on the settings chosen in this register, the PRU-ICSS UART0 receiver accepts the following from the transmitting device:

- 1 START bit
- 5, 6, 7, or 8 data bits
- 1 PARITY bit (optional)
- 1 STOP bit (any other STOP bits transferred with the above data are not detected)

RSR receives the data bits from the UART0_RXD pin. Then RSR concatenates the data bits and moves the resulting value into RBR (or the receiver FIFO), accessible in the RBR_TBR[7-0] RBR_DATA register bitfield. The PRU-ICSS UART0 also stores three bits of error status information next to each received character, to record a parity error, framing error, or break.

In the non-FIFO mode, when a character is placed in RBR and the receiver data available interrupt is enabled in the interrupt enable register - UART_INT_EN[0] ERBI, an interrupt is generated. This interrupt is cleared when the character is read from RBR. In the FIFO mode, the interrupt is generated when the FIFO is filled to the trigger level selected in the FIFO control MSB part of the register UART_INT_FIFO, and it is cleared when the FIFO contents drop below the trigger level.

7.3.8.2.3.3 PRU-ICSS UART Data Format

The PRU-ICSS UART0 transmits in the following format:

1 START bit + data bits (5, 6, 7, 8) + 1 PARITY bit (optional) + STOP bit (1, 1.5, 2)

It transmits 1 START bit; 5, 6, 7, or 8 data bits, depending on the data width selection; 1 PARITY bit, if parity is selected; and 1, 1.5, or 2 STOP bits, depending on the STOP bit selection.

The PRU-ICSS UART0 receives in the following format:

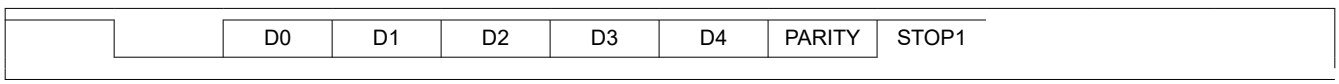
1 START bit + data bits (5, 6, 7, 8) + 1 PARITY bit (optional) + 1 STOP bit

It receives 1 START bit; 5, 6, 7, or 8 data bits, depending on the data width selection; 1 PARITY bit, if parity is selected; and 1 STOP bit.

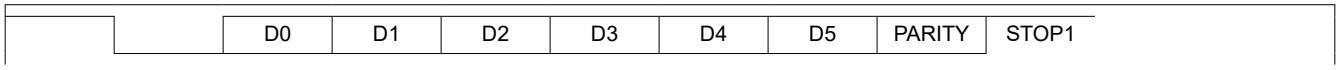
The protocol formats are shown in Figure 7-50.

Figure 7-50. PRU-ICSS UART Protocol Formats

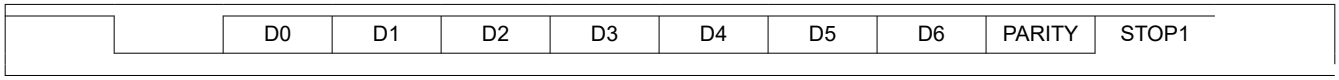
Transmit/Receive for 5-bit data, parity Enable, 1 STOP bit



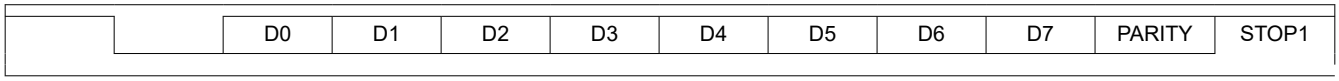
Transmit/Receive for 6-bit data, parity Enable, 1 STOP bit



Transmit/Receive for 7-bit data, parity Enable, 1 STOP bit



Transmit/Receive for 8-bit data, parity Enable, 1 STOP bit



7.3.8.2.3.3.1 Frame Formatting

Character length is specified using the UART_LCTR[0] WLS0 and UART_LCTR[1] WLS1 bits (see Table 7-73).

The number of stop-bits is specified using the UART_LCTR[2] STB bit (see Table 7-73).

The parity bit is programmed using the UART_LCTR[5] SP, UART_LCTR[4] EPS, and UART_LCTR[3] PEN bits (see Table 7-72).

Table 7-72. Relationship Between SP, EPS, and PEN Bits in LCTR

SP Bit	EPS Bit	PEN Bit	Parity Option
x	x	0	Parity disabled: No PARITY bit is transmitted or checked.
0	0	1	Odd parity selected: Odd number of logic 1s.
0	1	1	Even parity selected: Even number of logic 1s.
1	0	1	Stick parity selected with PARITY bit transmitted and checked as set.
1	1	1	Stick parity selected with PARITY bit transmitted and checked as cleared.

Table 7-73. Number of STOP Bits Generated

STB Bit	WLS Bit	Word Length Selected with WLS Bits	Number of STOP Bits Generated	Baud Clock (BCLK) Cycles
0	x	Any word length	1	16
1	0h	5 bits	1.5	24
1	1h	6 bits	2	32
1	2h	7 bits	2	32
1	3h	8 bits	2	32

7.3.8.2.4 PRU-ICSS UART Clock Generation and Control

The PRU-ICSS UART0 bit clock is derived from an input clock to the PRU-ICSS UART0. See the device-specific Datasheet to check the maximum data rate supported by the PRU-ICSS UART0.

Figure 7-51 is a conceptual clock generation diagram for the PRU-ICSS UART0. The processor clock generator receives a signal from an external clock source and produces a PRU-ICSS UART0 input clock with a programmed frequency. The PRU-ICSS UART0 contains a programmable baud generator that takes an input clock and divides it by a divisor in the range between 1 and (2^{16} - 1) to produce a baud clock (BCLK).
The frequency of BCLK is sixteen times (16×) the baud rate (each received or transmitted bit lasts 16 BCLK cycles) or thirteen times (13×) the baud rate (each received or transmitted bit lasts 13 BCLK cycles). When the PRU-ICSS UART0 is receiving, the bit is sampled in the 8th BCLK cycle for 16× over sampling mode and on the 6th BCLK cycle for 13× over-sampling mode. The 16× or 13× reference clock is selected by configuring the mode definition register: UART_MODE[0] OSM_SEL bit. The formula to calculate the divisor is:

$$\text{Divisor} = \frac{\text{UART input clock frequency}}{\text{Desired baud rate} \times 16} \quad \text{[MODE.OSM\_SEL = 0h]}$$

icss-13

$$\text{Divisor} = \frac{\text{UART input clock frequency}}{\text{Desired baud rate} \times 13} \quad \text{[MODE.OSM\_SEL = 1h]}$$

icss-14

Two 8-bit register fields:

- UART_DIVMSB[7-0] DLH
- UART_DIVLSB[7-0] DLL,

called divisor latches, hold this 16-bit divisor. DLH holds the most significant bits of the divisor, and DLL holds the least significant bits of the divisor. For information about these register fields, see the PRU-ICSS UART0 register descriptions in the *PRU_UART_UART0 Registers*. These divisor latches must be loaded during initialization of the PRU-ICSS UART0 in order to ensure desired operation of the baud generator. Writing to the divisor latches results in two wait states being inserted during the write access while the baud generator is loaded with the new value.

Figure 7-52 summarizes the relationship between the transferred data bit, BCLK, and the PRU-ICSS UART0 input clock. Note that the timing relationship depicted in Figure 7-52 shows that each bit lasts for 16 BCLK cycles. This is in case of 16x over-sampling mode. For 13× over-sampling mode each bit lasts for 13 BCLK cycles.

Example baud rates and divisor values relative to a 150 MHz PRU-ICSS UART0 input clock and 16× over-sampling mode are shown in Table 7-74.

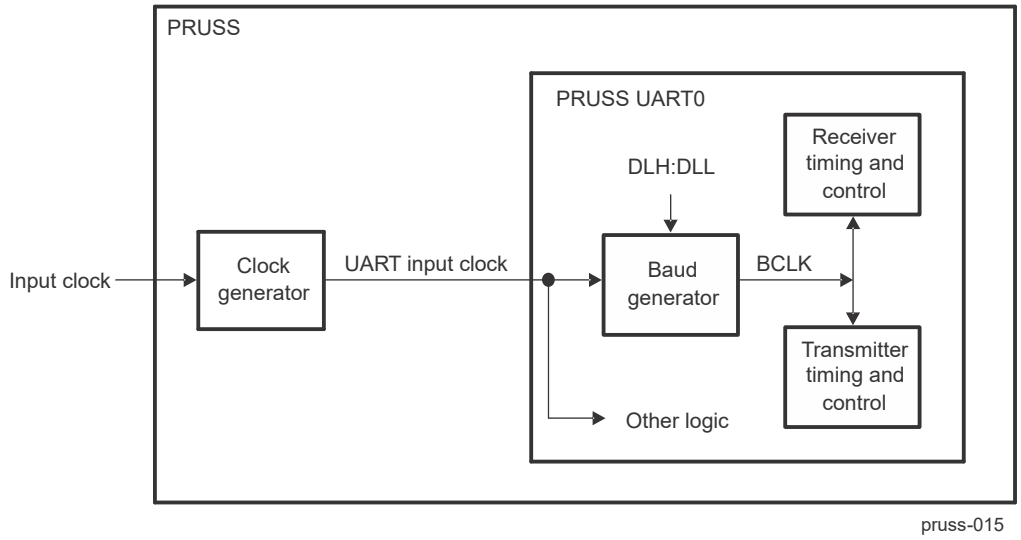


Figure 7-51. PRU-ICSS UART Clock Generation Diagram

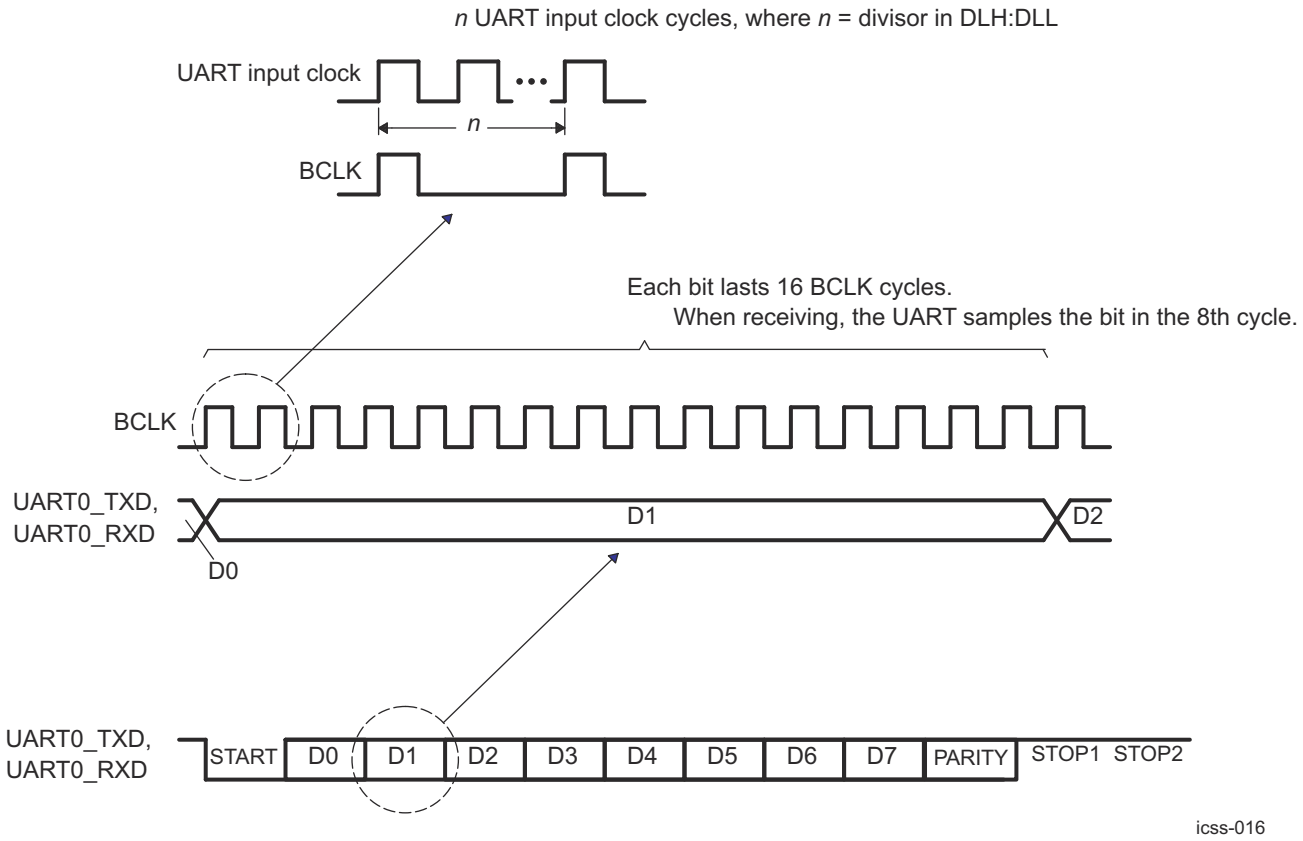


Figure 7-52. Relationships Between PRU-ICSS UART Data Bit, BCLK, and Input Clock

Table 7-74. Baud Rate Examples for 192-MHZ PRU-ICSS UART Input Clock and 16× Over-sampling Mode

Baud Rate	Divisor Value	Actual Baud Rate	Error (%)
2400	5000	2400	0.00
4800	2500	4800	0.00
9600	1250	9600	0.00
19200	625	19200	0.00
38400	313	38338.658	–0.16
56000	214	56074.766	0.13
115200	104	115384.6	0.16
128000	94	127659.574	–0.27
3000000	4	3000000	0.00
6000000	2	3000000	0.00
12000000	1	12000000	12000000

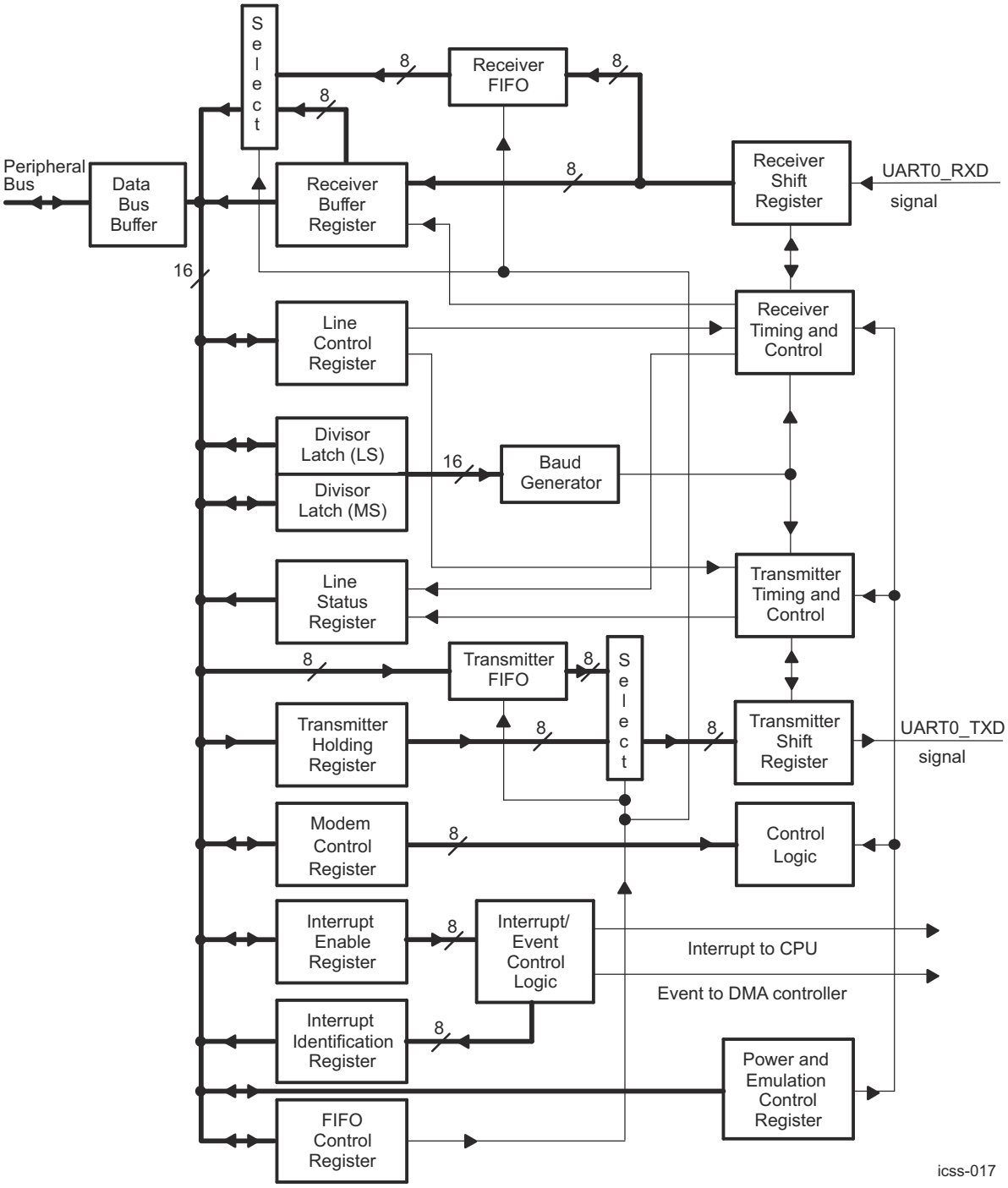
Table 7-75. Baud Rate Examples for 192-MHZ PRU-ICSS UART Input Clock and 13× Over-sampling Mode

Baud Rate	Divisor Value	Actual Baud Rate	Error (%)
2400	6154	2399.940	–0.0025
4800	3077	4799.880	–0.0025
9600	1538	9602.881	0.03
19200	769	19205.762	0.03
38400	385	38361.638	–0.10
56000	264	55944.056	–0.10
115200	128	115384.6	0.16
128000	115	128428.094	0.33

7.3.8.3 PRU-ICSS UART Functional Description

7.3.8.3.1 PRU-ICSS UART Functional Block Diagram

A functional block diagram of the PRU-ICSS UART0 is shown in Figure 7-53.



NOTE: The value *n* indicates the applicable UART where there are multiple instances. For the PRU-ICSS, there is only one instance and all UART signals should reflect this (e.g., UART0_TXD instead of UARTn_TXD).

Figure 7-53. PRU-ICSS UART Block Diagram

7.3.8.3.2 PRU-ICSS UART Reset Considerations

7.3.8.3.2.1 PRU-ICSS UART Software Reset Considerations

Two bits in the power and emulation management register - UART_PWR, control resetting the parts of the PRU-ICSS UART0:

- The bit [14]UTRST controls resetting the transmitter only. If bit [14]UTRST = 1h, the transmitter is enabled and active;
  if bit [14]UTRST = 0h, the transmitter is disabled and in reset state.
- The bit [13]URRST controls resetting the receiver only. If [13]URRST = 1h, the receiver is enabled and active;
  if bit [13]URRST = 0h, the receiver is disabled and in reset state.

In each case, putting the receiver and/or transmitter in reset will reset the state machine of the affected portion but will not affect the PRU-ICSS UART0 registers.

7.3.8.3.2.2 PRU-ICSS UART Hardware Reset Considerations

When the processor RESET pin is asserted, the entire processor is reset and is held in the reset state until the RESET pin is released. As part of a device reset, the PRU-ICSS UART0 state machine is reset and the PRU-ICSS UART0 registers are forced to their default states. The default states of the registers are shown in .

7.3.8.3.3 PRU-ICSS UART Power Management

The PRU-ICSS UART0 peripheral can be placed in reduced-power modes to conserve power during periods of low activity. The power management of the PRU-ICSS UART0 peripheral and other PRU-ICSS peripherals is controlled by the device Reset Control Manager (RCM). The RCM acts as a power management controller for all of the peripherals on the device. For more details on the power management procedures using the PSC, refer to the *Power Management* section.

7.3.8.3.4 PRU-ICSS UART Interrupt Support

7.3.8.3.4.1 PRU-ICSS UART Interrupt Events and Requests

The PRU-ICSS UART0 generates the interrupt requests described in Table 7-76. All requests are multiplexed through an arbiter to a single PRU-ICSS UART0 interrupt request to the CPU, as shown in Figure 7-54. Each of the interrupt requests has an enable bit in the interrupt enable register (IER) - UART_INT_EN and is recorded in [3-1]IIR_INTID bitfield of UART_INT_FIFO register.

If an interrupt occurs and the corresponding enable bit is set to 1h, the interrupt request is recorded in corresponding UART_INT_FIFO[3-1] IIR_INTID bitfield and is forwarded to the CPU. If an interrupt occurs and the corresponding enable bit is cleared to 0h, the interrupt request is blocked. The interrupt request is neither recorded in UART_INT_FIFO[3-1] IIR_INTID, nor forwarded to the CPU.

7.3.8.3.4.2 PRU-ICSS UART Interrupt Multiplexing

The PRU-ICSS UART0 have dedicated interrupt signals to the CPU and the interrupts are not multiplexed with any other interrupt source.

Table 7-76. PRU-ICSS UART Interrupt Requests Descriptions

PRU-ICSS UART0 Interrupt Request	Interrupt Source	Comment
THREINT	THR-empty condition: The transmitter holding register (THR) or the transmitter FIFO is empty. All of the data has been copied from THR, (i.e. UART_RBR_TBR[7-0] RBR_DATA) to the transmitter shift register (TSR).	If THREINT is enabled in UART_INT_EN register by setting the [1]ETBEI bit, it is recorded in [3-1]IIR_INTID bitfield. As an alternative to using THREINT, the CPU can poll the THRE bit in the line status register UART_LSR1.
RDAINT	Receive data available in non-FIFO mode or trigger level reached in the FIFO mode.	If RDAINT is enabled in UART_INT_EN register, by setting the [0]ERBI bit, it is recorded in INTID bitfield. As an alternative to using RDAINT, the CPU can poll the [0]DR bit in the line status register UART_LSR1. In the FIFO mode, this is not a functionally equivalent alternative because the [0]DR bit does not respond to the FIFO trigger level. The [0]DR bit only indicates the presence or absence of unread characters.

Table 7-76. PRU-ICSS UART Interrupt Requests Descriptions (continued)

PRU-ICSS UART0 Interrupt Request	Interrupt Source	Comment
RTOINT	Receiver time-out condition (in the FIFO mode only): No characters have been removed from or input to the receiver FIFO during the last four character times (see Table 7-78), and there is at least one character in the receiver FIFO during this time.	The receiver time-out interrupt prevents the PRU-ICSS UART0 from waiting indefinitely, in the case when the receiver FIFO level is below the trigger level and thus does not generate a receiver data-ready interrupt. If RTOINT is enabled in UART_INT_EN register, by setting the [0]ERBI bit, it is recorded in UART_INT_FIFO[3-1] IIR_INTID bitfield. There is no status bit to reflect the occurrence of a time-out condition.
RLSINT	Receiver line status condition: An overrun error, parity error, framing error, or break has occurred.	If RLSINT is enabled in INT_EN register, by setting the [2]ELSI bit, it is recorded in UART_INT_FIFO[3-1] IIR_INTID bitfield. As an alternative to using RLSINT, the CPU can poll the following bits in the line status register UART_LSR1: overrun error indicator (bit [1]OE), parity error indicator (bit [2]PE), framing error indicator ([3]FE), and break indicator ([4]BI).

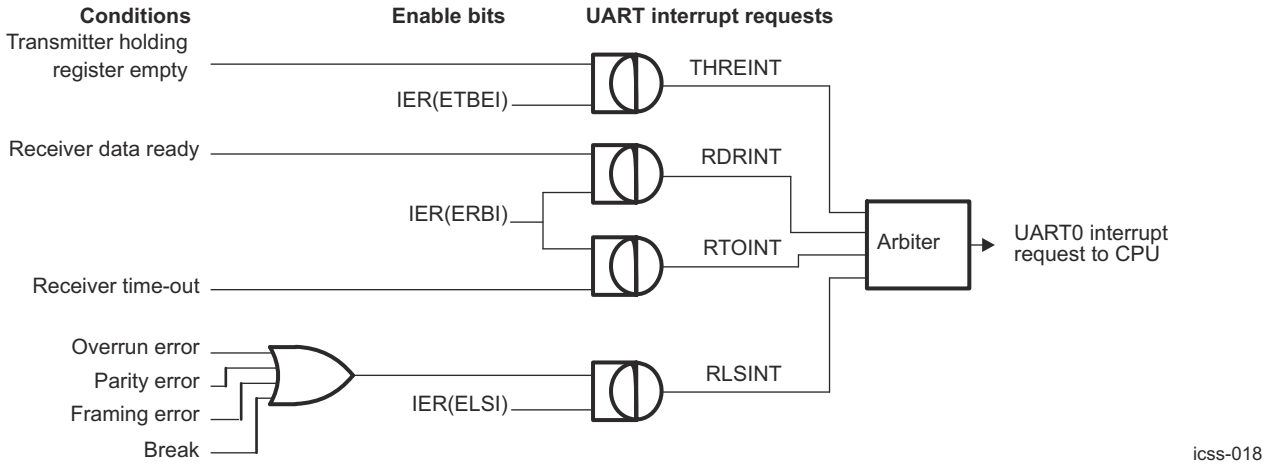


Figure 7-54. PRU-ICSS UART Interrupt Request Enable Paths

Table 7-77. Interrupt Identification and Interrupt Clearing Information

Priority Level	IIR Bits				Interrupt Type	Interrupt Source	Event That Clears Interrupt
	3	2	1	0			
None	0	0	0	1	None	None	None
1	0	1	1	0	Receiver line status	Overrun error, parity error, framing error, or break is detected.	For an overrun error, reading the line status register UART_LSR1, clears the interrupt. For a parity error, framing error, or break, the interrupt is cleared only after all the erroneous data have been read.
2	0	1	0	0	Receiver data-ready	Non-FIFO mode: Receiver data is ready.	Non-FIFO mode: The receiver buffer register (RBR) is read.
						FIFO mode: Trigger level reached. If four character times pass with no access of the FIFO, the interrupt is asserted again.	FIFO mode: The FIFO drops below the trigger level. (1)
2	1	1	0	0	Receiver time-out	FIFO mode only: No characters have been removed from or input to the receiver FIFO during the last four character times and there is at least one character in the receiver FIFO during this time.	One of the following events: • A character is read from the receiver FIFO (1) • A new character arrives in the receiver FIFO • The [13]URRST bit in the power and emulation management register (UART_PWR) is loaded with 0h.
3	0	0	1	0	Transmitter holding register empty	Non-FIFO mode: Transmitter holding register (THR) is empty.	A character is written to the transmitter holding register (UART_RBR_TBR) or the interrupt identification register (UART_INT_FIFO) is read.
						FIFO mode: Transmitter FIFO is empty.	

(1) In the FIFO mode, the receiver data-ready interrupt or receiver time-out interrupt is cleared by the CPU or by the DMA controller, whichever reads from the receiver FIFO first.

7.3.8.3.5 PRU-ICSS UART DMA Event Support

In the FIFO mode, the PRU-ICSS UART0 generates the following two DMA events:

- **Receive event (URXEVT):** The trigger level for the receiver FIFO (1, 4, 8, or 14 characters) is set with the FIFO control UART_INT_FIFO[7-6] IIR_FIFOEN bitfield. Every time the trigger level is reached or a receiver time-out occurs, the PRU-ICSS UART0 sends a receive event to the UDMA controller. In response, the UDMA controller reads the data from the receiver FIFO by way of the receiver buffer register UART_RBR_TBR[7-0] RBR_DATA. Note that the receive event is not asserted if the data at the top of the receiver FIFO is erroneous even if the trigger level has been reached.
- **Transmit event (UTXEVT):** When the transmitter FIFO is empty (when the last byte in the transmitter FIFO has been copied to the transmitter shift register), the PRU-ICSS UART0 sends an UTXEVT signal to the UDMA controller. In response, the UDMA controller refills the transmitter FIFO by way of the transmitter holding register (THR) - UART_RBR_TBR[7-0] RBR_DATA. The UTXEVT signal is also sent to the UDMA controller when the PRU-ICSS UART0 is taken out of reset using the [14]UTRST bit in the power and emulation management register (UART_PWR).

Activity in DMA channels can be synchronized to these events. In the non-FIFO mode, the PRU-ICSS UART0 generates no DMA events. Any DMA channel synchronized to either of these events must be enabled at the time the PRU-ICSS UART0 event is generated. Otherwise, the DMA channel will miss the event and, unless the PRU-ICSS UART0 generates a new event, no data transfer will occur.

7.3.8.3.6 PRU-ICSS UART Operations

7.3.8.3.6.1 PRU-ICSS UART FIFO Modes

The following two modes can be used for servicing the receiver and transmitter FIFOs:

- FIFO interrupt mode. The FIFO is enabled and the associated interrupts are enabled. Interrupts are sent to the CPU to indicate when specific events occur.
- FIFO poll mode. The FIFO is enabled but the associated interrupts are disabled. The CPU polls status bits to detect specific events.

Because the receiver FIFO and the transmitter FIFO are controlled separately, either one or both can be placed into the interrupt mode or the poll mode.

7.3.8.3.6.1.1 PRU-ICSS UART FIFO Interrupt Mode

When the receiver FIFO is enabled in the FIFO control register (FCR), mapped in the MSB part of the register UART_INT_FIFO, and the receiver interrupts are enabled in the interrupt enable register UART_INT_EN, the interrupt mode is selected for the receiver FIFO. The following are important points about the receiver interrupts:

- The receiver data-ready interrupt is issued to the CPU when the FIFO has reached the trigger level that is programmed in FCR. It is cleared when the CPU or the DMA controller reads enough characters from the FIFO such that the FIFO drops below its programmed trigger level.
- The receiver line status interrupt is generated in response to an overrun error, a parity error, a framing error, or a break. This interrupt has higher priority than the receiver data-ready interrupt. For details, see Section 7.3.8.3.4.
- The data-ready ([0]DR) bit in the line status register - UART_LSR1, indicates the presence or absence of characters in the receiver FIFO. The [0]DR bit is set when a character is transferred from the receiver shift register (RSR) to the empty receiver FIFO. The [0]DR bit remains set until the FIFO is empty again.
- A receiver time-out interrupt occurs if all of the following conditions exist:
  - At least one character is in the FIFO,
  - The most recent character was received more than four continuous character times ago. A character time is the time allotted for 1 START bit, *n* data bits, 1 PARITY bit, and 1 STOP bit, where *n* depends on the word length selected with the WLS0 and WLS1 bits of the line control register UART_LCTR. See Table 7-78.
  - The most recent read of the FIFO has occurred more than four continuous character times before.
- Character times are calculated by using the baud rate.
- When a receiver time-out interrupt has occurred, it is cleared and the time-out timer is cleared when the CPU or the EDMA controller reads one character from the receiver FIFO. The interrupt is also cleared if a new character is received in the FIFO or if the URRST bit is cleared in the power and emulation management register - PWM.
- If a receiver time-out interrupt has not occurred, the time-out timer is cleared after a new character is received or after the CPU or EDMA reads the receiver FIFO.

When the transmitter FIFO is enabled in UART_INT_FIFO[0] IIR_IPEND bit and the transmitter holding register empty (THRE) interrupt is enabled in UART_INT_EN[1] ETBEI bit, the interrupt mode is selected for the transmitter FIFO. The THRE interrupt occurs when the transmitter FIFO is empty. It is cleared when the transmitter hold register (THR) UART_RBR_TBR[7-0] RBR_DATA bitfield is loaded (1 to 16 characters may be written to the transmitter FIFO while servicing this interrupt) or the [3-1]IIR_INTID bitfield is read in the interrupt identification register UART_INT_FIFO.

Table 7-78. Character Time for Word Lengths

Word Length (*n*)	Character Time	Four Character Times
5	Time for 8 bits	Time for 32 bits
6	Time for 9 bits	Time for 36 bits
7	Time for 10 bits	Time for 40 bits
8	Time for 11 bits	Time for 44 bits

7.3.8.3.6.1.2 PRU-ICSS UART FIFO Poll Mode

When the receiver FIFO is enabled in the FIFO control register (via setting the UART_INT_FIFO[0] IIR_IPEND to 1h) and the receiver interrupts are disabled in the interrupt enable register (UART_INT_EN), the poll mode is selected for the receiver FIFO. Similarly, when the transmitter FIFO is enabled via setting the same bit (UART_INT_FIFO[0] IIR_IPEND to 1h) and the transmitter interrupts are disabled, the transmitted FIFO is in the poll mode. In the poll mode, the CPU detects events by checking bits in the line status register - UART_LSR1:

- The UART_LSR1[7] RXFIFOE bit indicates whether there are any errors in the receiver FIFO.
- The UART_LSR1[6] TEMT bit indicates that both the transmitter holding register (THR) and the transmitter shift register (TSR) are empty.
- The UART_LSR1[5] THRE bit indicates when THR (mapped in the UART_RBR_TBR[7-0] RBR_DATA bitfield) is empty.
- The following line status register - UART_LSR1 bits specify which error or errors have occurred:

  - UART_LSR1[4] BI - Break Interrupt
  - UART_LSR1[3] FE – Framing Error
  - UART_LSR1[2] PE – Parity Error
  - UART_LSR1[1] OE – Overrun Error
- The UART_LSR1[0] DR (data-ready) bit is set as long as there is at least one byte in the receiver FIFO.

Also, in the FIFO poll mode:

- The interrupt identification ([3-1] IIR_INTID) bit field in register UART_INT_FIFO are not affected by any events because the interrupts are disabled.
- The PRU-ICSS UART0 does not indicate when the receiver FIFO trigger level is reached or when a receiver time-out occurs.

7.3.8.3.6.2 PRU-ICSS UART Autoflow Control

The PRU-ICSS UART0 can employ autoflow control by connecting the $\overline{\text{UART0\_CTS}}$ and $\overline{\text{UART0\_RTS}}$ signals. The $\overline{\text{UART0\_CTS}}$ input must be active before the transmitter FIFO can transmit data. The $\overline{\text{UART0\_RTS}}$ becomes active when the receiver needs more data and notifies the sending device. When $\overline{\text{UART0\_RTS}}$ is connected to $\overline{\text{UART0\_CTS}}$, data transmission does not occur unless the receiver FIFO has space for the data. Therefore, when two UARTs are connected as shown in Figure 7-55 with autoflow enabled (UART_MCTR[5] AFE = 1h), overrun errors are eliminated.

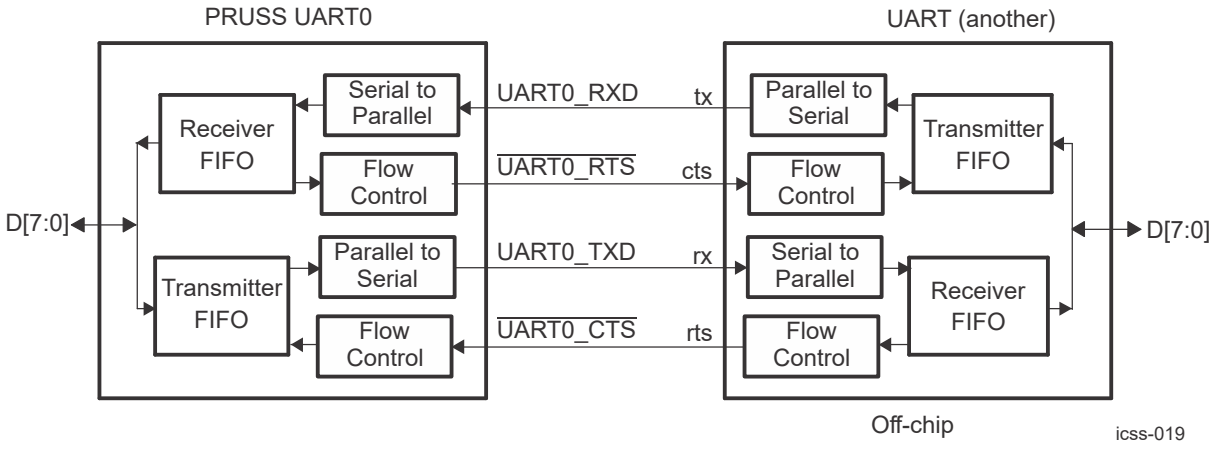
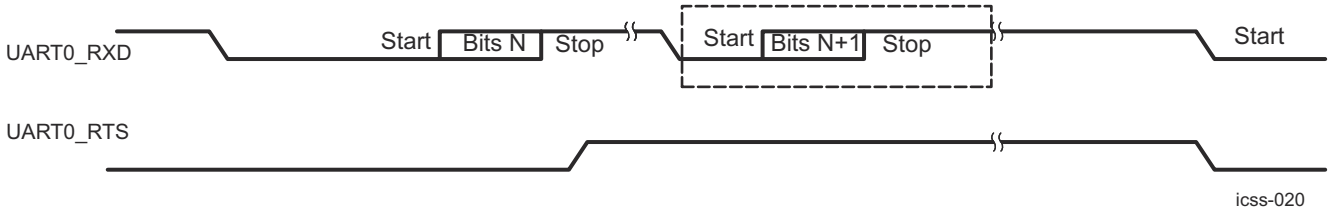


Figure 7-55. UART Interface Using Autoflow Diagram

7.3.8.3.6.2.1 PRU-ICSS UART Signal $\overline{\text{UART0\_RTS}}$ Behavior

$\overline{\text{UART0\_RTS}}$ data flow control originates in the receiver block (see Figure 7-53). When the receiver FIFO level reaches a trigger level of 1, 4, 8, or 14 (see Figure 7-56), $\overline{\text{UART0\_RTS}}$ is deasserted. The sending UART may send an additional byte after the trigger level is reached (assuming the sending UART has another byte to send), because it may not recognize the deassertion of $\overline{\text{UART0\_RTS}}$ until after it has begun sending the additional byte. For trigger level 1, 4, and 8, $\overline{\text{UART0\_RTS}}$ is automatically reasserted once the receiver FIFO is emptied. For trigger level 14, $\overline{\text{UART0\_RTS}}$ is automatically reasserted once the receiver FIFO drops below the trigger level.

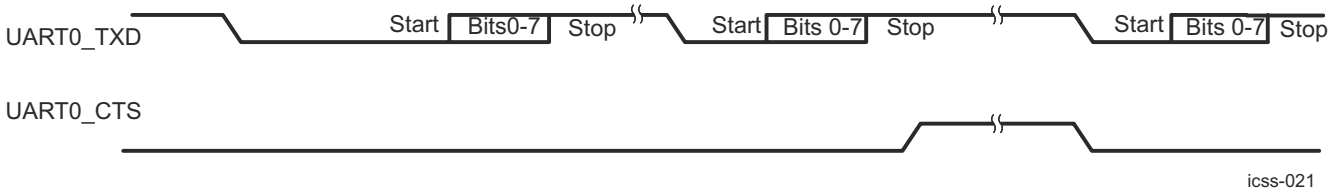


A. N = Receiver FIFO trigger level.
B. The two blocks in dashed lines cover the case where an additional byte is sent.

Figure 7-56. Autoflow Functional Timing Waveforms for $\overline{\text{UART0\_RTS}}$

7.3.8.3.6.2.2 PRU-ICSS UART Signal UART0_CTS Behavior

The transmitter checks UART0_CTS before sending the next data byte. If UART0_CTS is active, the transmitter sends the next byte. To stop the transmitter from sending the following byte, UART0_CTS must be released before the middle of the last STOP bit that is currently being sent (see Figure 7-57). When flow control is enabled, UART0_CTS level changes do not trigger interrupts because the device automatically controls its own transmitter. Without autoflow control, the transmitter sends any data present in the transmitter FIFO and a receiver overrun error may result.



A. When UART0_CTS is active (low), the transmitter keeps sending serial data out.
B. When UART0_CTS goes high before the middle of the last STOP bit of the current byte, the transmitter finishes sending the current byte but it does not send the next byte.
C. When UART0_CTS goes from high to low, the transmitter begins sending data again.

Figure 7-57. Autoflow Functional Timing Waveforms for UART0_CTS

7.3.8.3.6.3 PRU-ICSS UART Loopback Control

The PRU-ICSS UART0 can be placed in the diagnostic mode using the [4]LOOP bit in the modem control register - UART_MCTR, which internally connects the PRU-ICSS UART0 output back to the PRU-ICSS UART0's input. In this mode, the transmit and receive data paths, the transmitter and receiver interrupts, and the modem control interrupts can be verified without connecting to another UART.

7.3.8.3.7 PRU-ICSS UART Emulation Considerations

The [0]FREE bit in the power and emulation management register (UART_PWR) determines how the PRU-ICSS UART0 responds to an emulation suspend event such as an emulator halt or breakpoint. If bit UART_PWR[0] FREE = 0h and a transmission is in progress, the PRU-ICSS UART0 halts after completing the one-word transmission; if bit UART_PWR[0] FREE = 0h and a transmission is not in progress, the PRU-ICSS UART0 halts immediately. If UART_PWR[0] FREE = 1h, the PRU-ICSS UART0 does not halt and continues operating normally.

Note also that most emulator accesses are transparent to PRU-ICSS UART0 operation. Emulator read operations do not affect any register contents, status bits, or operating states, with the exception of the interrupt identification register (UART_INT_FIFO). Emulator writes, however, may affect register contents and may affect PRU-ICSS UART0 operation, depending on what register is accessed and what value is written.

The PRU-ICSS UART0 registers can be read from or written to during emulation suspend events, even if the PRU-ICSS activity has stopped.

7.3.8.3.8 PRU-ICSS UART Exception Processing

7.3.8.3.8.1 PRU-ICSS UART Divisor Latch Not Programmed

Since the processor reset signal has no effect on the divisor latch, the divisor latch will have an unknown value after power up. If the divisor latch is not programmed after power up, the baud clock (BCLK) will not operate and will instead be set to a constant logic 1 state.

The divisor latch values should always be reinitialized following a processor reset.

7.3.8.3.8.2 Changing Operating Mode During Busy Serial Communication of PRU-ICSS UART

Since the serial link characteristics are based on how the control registers are programmed, the PRU-ICSS UART0 module will expect the control registers to be static while it is busy engaging in a serial communication. Therefore, changing the control registers while the module is still busy communicating with another serial device will most likely cause an error condition and should be avoided.

7.3.9 PRU-ICSS ECAP Module

7.3.9.1 PRU-ICSS eCAP Overview

7.3.9.1.1 Purpose of the PRU-ICSS eCAP Peripheral

The device PRU-ICSS integrated **enhanced capture (eCAP)** module targets:

- 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

7.3.9.1.2 PRU-ICSS eCAP Features

The device PRU-ICSS integrated eCAP module (signified as PRUSS1_eCAP_0 throughout the *PRU-ICSS eCAP Module* section) includes the following features:

- 32-bit time base counter
- 4-event time-stamp registers (each 32 bits)
- Edge polarity selection for up to four sequenced time-stamp capture events
- Interrupt on either of the four events
- Single shot capture of up to four event time-stamps
- Continuous mode capture of time-stamps in a four-deep circular buffer
- 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

7.3.9.2 PRU-ICSS ECAP Functional Description

For full description of the PRU-ICSS ECAP0 module and functionality, refer to the *Enhanced Capture (eCAP) Module.*

7.3.9.2.1 PRU-ICSS Capture and APWM Operating Mode

The PRU-ICSS_eCAP_0 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 PRU-ICSS_ECAP_CAP1 and PRU-ICSS_ECAP_CAP2 registers become the active period and compare registers, respectively, while PRU-ICSS_ECAP_CAP3 and PRUSS_ECAP_CAP4 registers become the period and capture shadow registers, respectively. Figure 7-58 is a high-level view of both the capture and auxiliary pulse-width modulator (APWM) modes of operation.

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 PRU-ICSS_ECAP_CAP1/PRU-ICSS_ECAP_CAP2 active registers also writes the same value to the corresponding shadow registers PRU-ICSS_ECAP_CAP3/PRU-ICSS_ECAP_CAP4. This emulates immediate mode. Writing to the shadow registers PRU-ICSS_ECAP_CAP3/PRU-ICSS_ECAP_CAP4 invokes the shadow mode.

Figure 7-58. PRU-ICSS Capture and APWM Modes of Operation

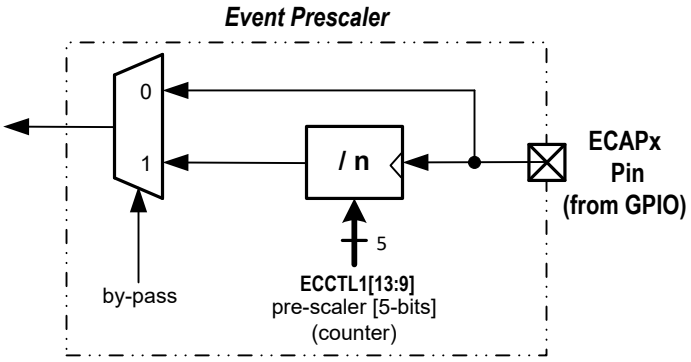
7.3.9.2.2 PRU-ICSS eCAP Capture Mode Description

Figure 7-59 shows the various components that implement the capture function.

Figure 7-59. Capture Function Diagram

7.3.9.2.2.1 PRU-ICSS 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 7-60 shows a functional diagram and Figure 7-61 shows the operation of the prescale function.



A. When a prescale value of 1 is chosen (PRU-ICSS_ECAP_ECCTL1[13:9] = 0b0000) the input capture signal by-passes the prescale logic completely.

Figure 7-60. Event Prescale Control

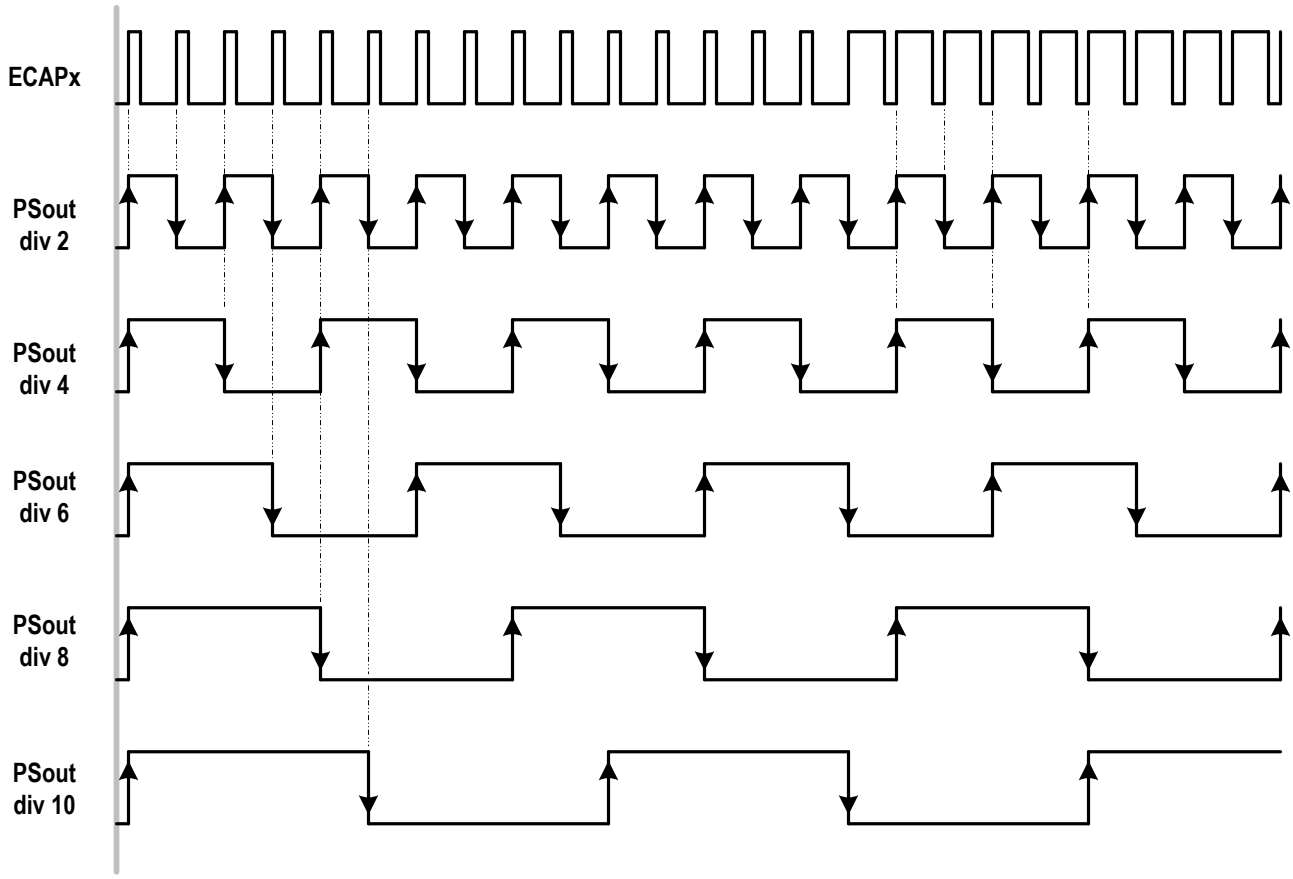


Figure 7-61. Prescale Function Waveforms

7.3.9.2.2.2 PRU-ICSS 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 Modulo4 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.

7.3.9.2.2.3 eCAP Continuous/One-Shot Control

- The Mod4 (2 bit) counter is incremented via edge qualified events (CEVT1 - CEVT4).
- 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, and when equal stops the Mod4 counter and inhibits further loads of the PRU-ICSS_ECAP_CAP1 - PRU-ICSS_ECAP_CAP4 registers. This occurs during one-shot operation.

The continuous/one-shot block 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 PRU-ICSS_ECAP_CAP1-4 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 PRU-ICSS_ECAP_CAP1-4 registers again, providing the 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 PRU-ICSS_ECAP_CAP1-4 in a circular buffer sequence.

7.3.9.2.2.4 PRU-ICSS eCAP 32-bit Counter and Phase Control

This counter provides the time-base for event captures, and is clocked via the system clock.

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-LD4 signals.

Note

The PRU-ICSS_eCAP_0 "SYNCIn" hardware event synchronization input and "SYNCOut" hardware synchronization output are NOT implemented in the PRU-ICSS. However, a software-forced synchronization via bit PRU-ICSS_ECAP_ECCTL2[8] SWSYNC, can be used as an alternative, provided that PRU-ICSS_ECAP_ECCTL2[5] SYNCI_EN bit is set to 0b1.

7.3.9.2.2.5 PRU-ICSS Enhanced Capture 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 control bit CAPLDEN. During one-shot operation, this bit is cleared (loading is inhibited) automatically when a stop condition occurs, StopValue = Mod4.

PRU-ICSS_ECAP_CAP1 and PRU-ICSS_ECAP_CAP2 registers become the active period and compare registers, respectively, in APWM mode.

PRU-ICSS_ECAP_CAP3 and PRUSS_ECAP_CAP4 registers become the respective shadow registers (APRD and ACMP) for PRU-ICSS_ECAP_CAP1 and PRU-ICSS_ECAP_CAP2 during APWM operation.

7.3.9.2.2.6 PRU-ICSS eCAP Interrupt Control

An Interrupt can be generated on capture events (CEVT1-CEVT4, CNTOVF) or APWM events (CTR = PRD, CTR = CMP). See *Interrupts in PRU-ICSS eCAP Module*.

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 of the PRU-ICSS eCAP module "pr1_ecap_intr_req" aggregated IRQ mapped on the PRU-ICSS1_IRQ_15 input line of the local PRU-ICSS1_INTC. See also Table 7-69.

Seven interrupt events (CEVT1, CEVT2, CEVT3, CEVT4, CNTOVF, CTR = PRD, CTR = CMP) can be generated. The interrupt enable register (PRU-ICSS_ECAP_ECEINT) is used to enable/disable individual interrupt event sources. The interrupt flag register (PRU-ICSS_ECAP_ECFLG) indicates if any interrupt event has been latched and contains the global interrupt flag bit (INT). An interrupt pulse is generated to the PRU-ICSS1_INTC local interrupt controller only if any of the interrupt events are enabled, the flag bit is 1, and the INT flag bit is 0. The interrupt service routine must clear the global interrupt flag bit and the serviced event via the interrupt clear register (PRU-ICSS_ECAP_ECCLR) before any other interrupt pulses are generated. You can force an interrupt event via the interrupt force register (PRU-ICSS_ECAP_ECFRC). This is useful for test purposes.

7.3.9.2.2.7 PRU-ICSS eCAP Shadow Load and Lockout Control

In capture mode, this logic inhibits (locks out) any shadow loading of PRU-ICSS_ECAP_CAP1 or PRU-ICSS_ECAP_CAP2 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 PRU-ICSS_ECAP_CAP1 or PRU-ICSS_ECAP_CAP2 immediately upon writing a new value.
- On period equal, CTR[31-0] = PRD[31-0]

7.3.9.2.2.8 CEVT Flag Registers

Note

The CEVT1, CEVT2, CEVT3, CEVT4 flags are only active in capture mode (PRx_ECAP_ECCTL2[9] CAPAPWM == 0b'0). The CTR = PRD, CTR = CMP flags are only valid in APWM mode (PRx_ECAP_ECCTL2[9] CAPAPWM == 0b'1). CNTOVF flag is valid in both modes.

7.3.9.2.3 PRU-ICSS eCAP Module 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 PRUSS_ECAP_CAP1 / PRUSS_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 shadow registers APRD and ACMP (PRUSS_ECAP_CAP3/ PRUSS_ECAP_CAP4). The shadow register contents are transferred over to PRUSS_ECAP_CAP1 / PRUSS_ECAP_CAP2 registers either immediately upon a write, or on a CTR = PRD trigger.
- In APWM mode, writing to PRUSS_ECAP_CAP1 / PRUSS_ECAP_CAP2 active registers will also write the same value to the corresponding shadow registers PRUSS_ECAP_CAP3/PRUSS_ECAP_CAP4. This emulates immediate mode. Writing to the shadow registers PRUSS_ECAP_CAP3/PRUSS_ECAP_CAP4 will invoke the shadow mode.
- During initialization, you must write to the active registers for both period and compare. This automatically copies the initial values into the shadow values. For subsequent compare updates, during run-time, you only need to use the shadow registers.

Figure 7-62. PWM Waveform Details Of eCAP APWM Mode Operation

The behavior of APWM active-high mode (APWMPOL == 0) is:

```
CMP = 00000000h, output low for duration of period (0% duty)
```

```
CMP = 00000001h, output high 1 cycle
```

```
CMP = 00000002h, 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
```

The behavior of APWM active-low mode (APWMPOL == 1) is:

```
CMP = 00000000h, output high for duration of period (0% duty)
```

```
CMP = 00000001h, output low 1 cycle
```

```
CMP = 00000002h, 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
```

7.3.10 PRU-ICSS MII_RT Module

7.3.10.1 PRU-ICSS MII_RT Introduction

The Real-time Media Independent Interface (MII_RT) provides a programmable I/O interface for the PRUs to access and control up to two MII ports. The MII_RT module can also be configured to push and pull data independent of the PRU cores.

Note

In order to guarantee the MII_RT I/O timing values published in the device data sheet, the TX_CLK_DELAYn (where n = 0 or 1) bit field in the MII_RT_TXCFG0/1 register must be set to 0h (default value).

7.3.10.1.1 PRU-ICSS MII_RT Features

The PRU-ICSS MII_RT module supports:

- Two MII ports
  - Each MII port has:
    - 32-Bytes RX L1 FIFO
    - 64-Bytes RX L2 FIFO (two memory banks: Bank0 = 32-Bytes and Bank1 = 32-Bytes)
    - 40-Bytes TX L1 FIFO one per port
    - 64-Bytes TX L2 FIFO one per port
  - Rate decoupling on TX L1 FIFO
  - Configurable pre-amble removal on RX L1 FIFO and insertion on TX L1 FIFO
  - Sync frame delimiter detection
  - Configurable TX L1 FIFO trigger (10 bits with 40 ns ticks)
- MII port multiplexer per direction to support line/ring structure
  - Link detection through RX_ERR
- Cyclic redundancy check (CRC)
  - CRC32 generation on TX path
  - CRC32 checker on RX path

7.3.10.1.2 Unsupported Features

The PRU-ICSS MII_RT module does not support:

- Auto padding in TX L1 FIFO
- Dynamic TX multiplexer switching during packet handling:
  - Can allow one PRU to handle both MII interfaces and a second PRU to manage the host and switch functions.

7.3.10.1.3 PRU-ICSS MII_RT Block Diagram

Figure 7-63 shows the MII_RT in context of the PRU-ICSS.

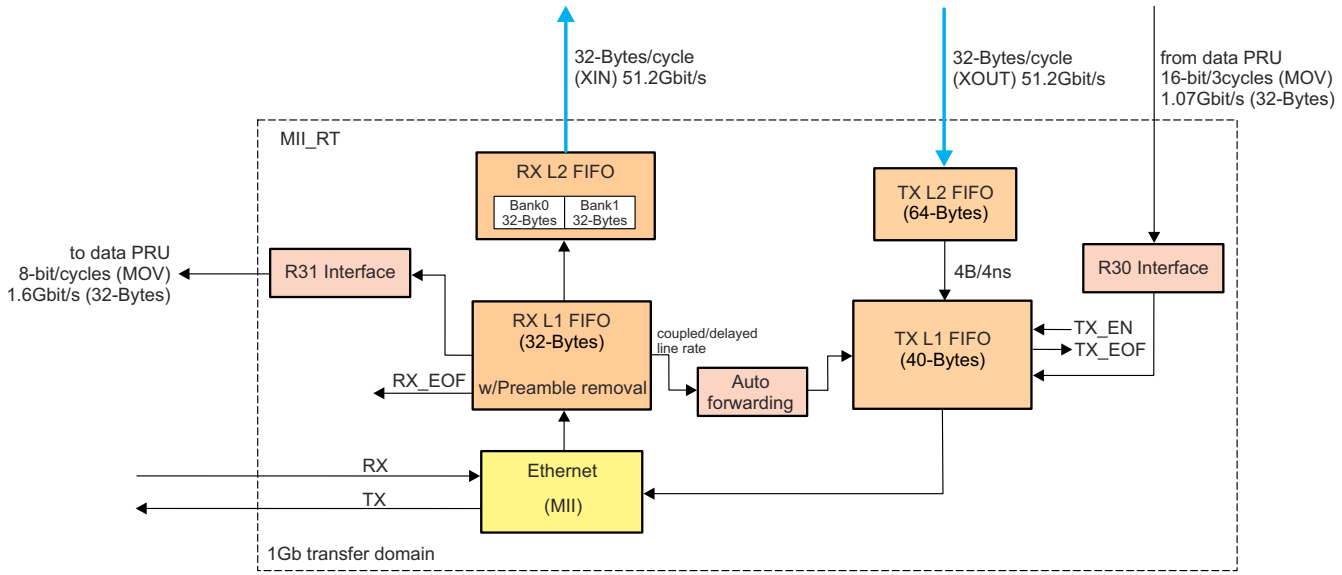


Figure 7-63. PRU-ICSS MII_RT Block Diagram

7.3.10.2 MII_RT Functional Description

7.3.10.2.1 MII_RT Data Path Configuration

The MII_RT module supports three basic data path configurations. These configurations are compared in Table 7-79 and described in the following sections.

Table 7-79. MII_RT Data Path Configuration Comparison

Configuration	PRU Dependency	Data Servicing	Port-to-Port Latency
Auto-forward	Snoop only	One word in flight	Low
8- or 16-bit processing with on-the-fly modifications (RX L1)	Yes	One word or byte in flight	Low
32-byte double buffer or ping-pong processing (RX L2)	Yes	Multi-words in flight	Medium (application-dependent)

7.3.10.2.1.1 Auto-forward with Optional PRU Snoop

Data is automatically forwarded from the MII RX port to the MII TX port without manipulations, as shown in Figure 7-64. This configuration does not depend on the PRU core. However, it does support an option for PRU to snoop or monitor the received data through the RX L2, shown in Figure 7-65. The PRU does not access data and status bits through R31, and it does not modify and push data.

Figure 7-64. Auto-forward

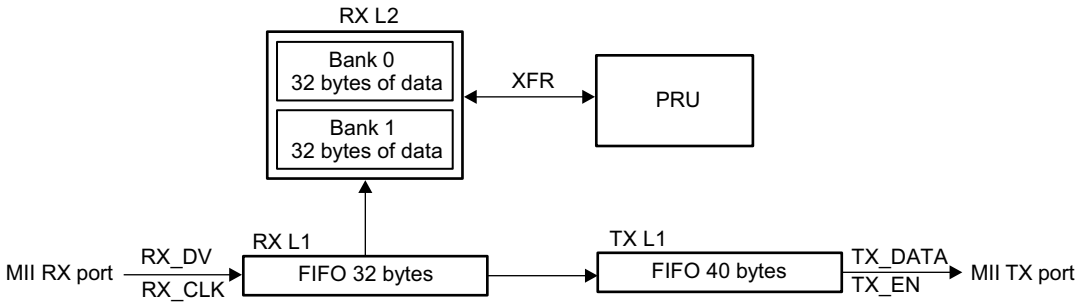


Figure 7-65. Auto-forward with PRU Snoop

7.3.10.2.1.2 8- or 16-bit Processing with On-the-Fly Modifications

This configuration services one byte or word in flight and has low latency. The PRU has the option to manipulate the received word and control popping data from the RX L1 FIFO and pushing it on the TX L1 FIFO.

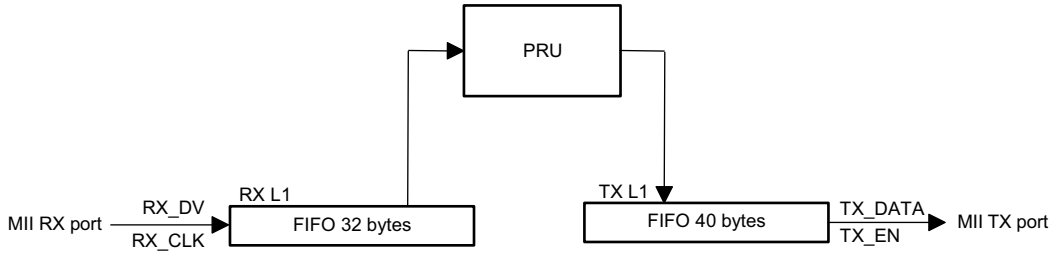


Figure 7-66. 8- or 16-bit Processing with On-the-Fly Modifications

7.3.10.2.1.3 32-byte Double Buffer or Ping-Pong Processing

This configuration supports high bandwidth, high efficiency transactions. Often implementations using this mode permit relaxed servicing requirements allowing the PRU to manipulate the received data before transmitting.

Data received in this configuration is passed into the RX L2 buffer. The PRU reads multiple bytes of data from one of the RX L2 banks through the high bandwidth broadside interface and XFR instructions. The PRU can then store or manipulate data before pushing it to the TX L1 FIFO for transmission on the MII TX port.

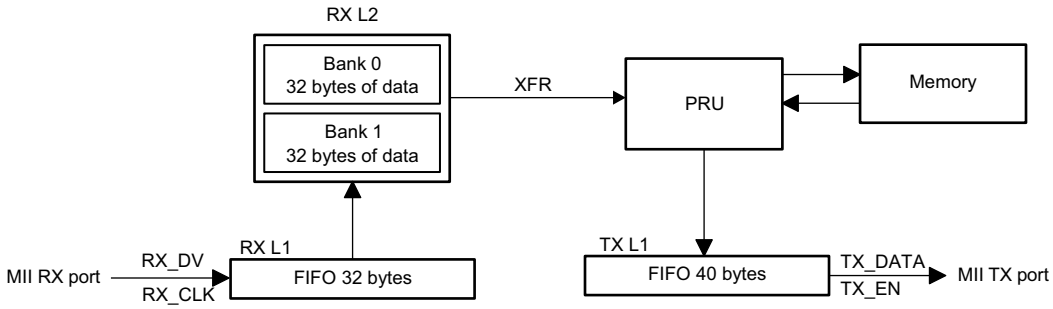


Figure 7-67. 32-byte Double Buffer or Ping-Pong Processing

7.3.10.2.2 MII_RT Definition and Terms

7.3.10.2.2.1 MII_RT Data Frame Structure

The data received and transmitted over MII conforms with the frame structure shown in Table 7-80.

Table 7-80. MII_RT Frame Structure

Inter-frame	Preamble	Start of Frame Delimiter (SFD)	Data	Cyclic Redundancy Check (CRC)

The data following the SFD is formatted in a 4-bit nibble structure. Figure 7-68 illustrates the nibble order. The MSB arriving first is on the LSB side of a nibble. When receiving data, the MII_RT receive logic will wait for the next nibble to arrive before constructing a byte and delivering to the PRU.

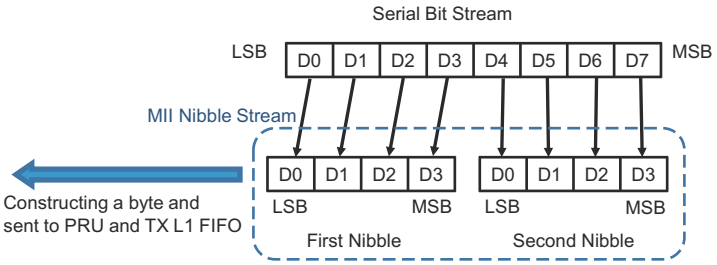


Figure 7-68. Data Nibble Structure

7.3.10.2.2.2 PRU R30 and R31

The PRU registers R30 and R31 are used to receive, transmit, and control the data for the PRU. As shown in Figure 7-69, the R31 is used to access data in the RX L1 FIFO, the R30 is used to transmit data from the PRU, and the R31 output is used the control the flow of receive and transmit. For more details about these registers, see the following sections.

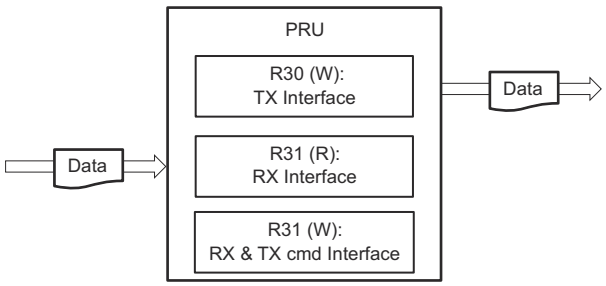


Figure 7-69. PRU R30, R31 Operations

7.3.10.2.2.3 RX and TX L1 FIFO Data Movement

To advance the next data byte seen by R31, the PRU must pop the data from the RX L1 FIFO. Likewise, the PRU can push the data from R30 to the TX L1 FIFO. These operations are illustrated in Figure 7-70.

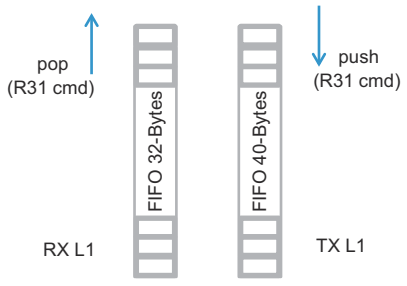


Figure 7-70. Reading and Writing FIFO Data

7.3.10.2.2.4 Receive CRC Computation

For the incoming data, the MII_RT calculates CRC32 and then compares against the value provided in the incoming frame. If there is a mismatch, the MII_RT signals ERROR_CRC to the PRU. If a previous node or Ethernet device appended an error nibble, the CRC calculation of received packet will be wrong because the longer frame and the frame length will end at a 4-bit boundary instead of the usual 8-bit boundary. When RX_DV goes inactive on the 4-bit boundary, the interface will assert DATA_RDY and BYTE_RDY flag with the ERROR NIBBLE. The error event is also mapped into the PRU-ICSS INTC.

7.3.10.2.2.5 Transmit CRC Computation

For the outgoing data, the MII_RT calculates the CRC32 value and inserts it into outgoing packets. The CRC value computed on each MII transmit path is also available in memory map registers that can be read by the PRU and used primarily for debug and diagnostic purposes. The CRC is inserted into the outgoing packet based on the commands received through the R31 register of the PRU. The CRC will be inserted into the TX L1 FIFO, and there must be enough room to store the CRC value in the FIFO or else the FIFO will overflow. As Table 7-81 shows, the CRC programming model supports three sequences that provide more flexibility. Note: “cmdR31” indicates write to the mentioned bits of the R31 command interface.

Table 7-81. TX CRC Programming Models

Option 1	Step 1: cmdR31 [TX_CRC_HIGH + TX_CRC_LOW + TX_EOF]
Option 2	**Note: Only valid when TX L2 is disabled.** Step 1: cmdR31 [TX_CRC_HIGH] Step 2: wait > 6 clocks (PRU cycles) Step 3: cmdR31 [TX_CRC_LOW + TX_EOF]
Option 3	**Note: Only valid when TX L2 is disabled.** Step 1: cmdR31 [TX_CRC_HIGH] Step 2: wait > 6 clocks (PRU cycles) Step 3: read TX_CRC0[31-0] TX_CRC0 and TX_CRC1[31-0] TX_CRC1 Step 4: modify CRC[15-0] Step 5: cmdR31 [TX_PUSH16 + TX_EOF + TX_ERROR_NIBBLE]

7.3.10.2.2.6 Transmit CRC Computation for fragmented frames

Fragmented frames have a special CRC32. Each fragment CRC32 is based on the previous fragements, so a running total. In addition TX_CRC_HIGH is inverted for all fragments expect the last fragment. The final fragement has a normal CRC which is based on the full frame.

7.3.10.2.3 RX MII Interface

The RX MII interface is composed of multiple components that perform various tasks - latch received data, start of frame detection, start frame delimiter detection, CRC calculation and error detection, enhanced link detection through RX error detection and interface to PRU register R31.

Table 7-82 includes more details about the internal signals and output of these components

7.3.10.2.3.1 RX MII Receive Data Latch

The receive data from the MII interface is stored in the receive data FIFO which is 32 bytes. The PRU can access this data through the register R31. Depending on the configuration settings, the data can be latched on reception of one or two bytes. In each scheme, the configured number of nibbles is assembled before being copied into the PRU registers. Figure 7-71 shows the inputs and outputs of the data latch logic block.

The receiver logic in MII_RT can be programmed through the MII_RT MII_RT_RXCFG0 and MII_RT MII_RT_RXCFG1 registers to remove or retain the preamble + SFD from incoming frames.

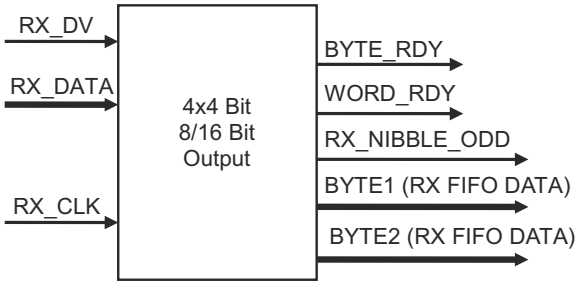


Figure 7-71. RX Data Latch

7.3.10.2.3.2 RX MII Start of Frame Detection

The start of frame detection logic tracks the frame boundaries and signals the beginning of a frame to other components of the PRU-ICSS. This logic detects two events:

- Start of Frame (SOF) event that occurs when Receive Data Valid MII signal is sampled high.
- Start of Frame Delimiter (SFD) event is seen on MII Receive Data bus.

These event triggers can be used to add timestamp to the frames. The notification for these events is available through R31 as well as through INTC which is integrated in the PRU-ICSS. Figure 7-72 shows the inputs and outputs of the start of frame detection logic block.

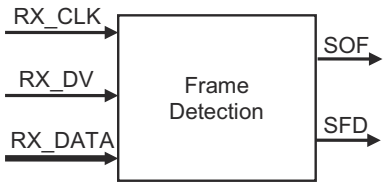


Figure 7-72. RX MII Start of Frame Detection

7.3.10.2.3.3 CRC Error Detection

For each incoming frame, the CRC is calculated by the MII_RT and compared against the CRC included in the frame. When the two values do not match, a CRC error is flagged. The ERROR_CRC indication is available in the register interface (PRU R31 Receive Interface) as well as in the FIFO interface (RX L2 Status Interface). It is also provided to the INTC which is integrated in the PRU-ICSS. Figure 7-73 shows the inputs and outputs of CRC error detection logic block.

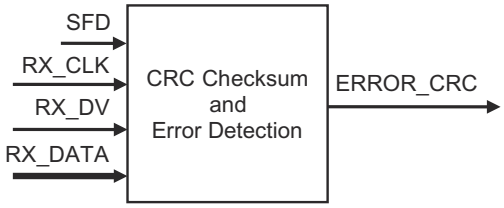


Figure 7-73. CRC Error Detection

7.3.10.2.3.4 RX Error Detection and Action

The RX error detection logic tracks the receive error signaled by the physical layer and informs the PRU-ICSS INTC whenever an error is detected. Figure 7-74 shows the inputs and outputs of the RX error detection logic block. Note the following dependencies:

- RX_ERR signal is only sampled when RX_DV is asserted.
- All nibbles are discarded post RX_ERR event, including the nibble which had RX_ERR asserted. This state will remain until EOF occurs.
- Due to this fact, RX L1 FIFO and RX L2 FIFO will never receive any data with RX_ERR or post RX_ERR during that frame.

Note

RX error detection logic is supported only for MII mode and this feature is not supported for RGMII and SGMII modes of operation.

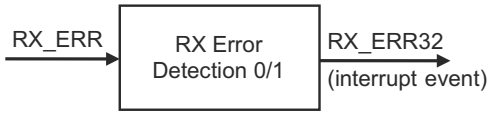
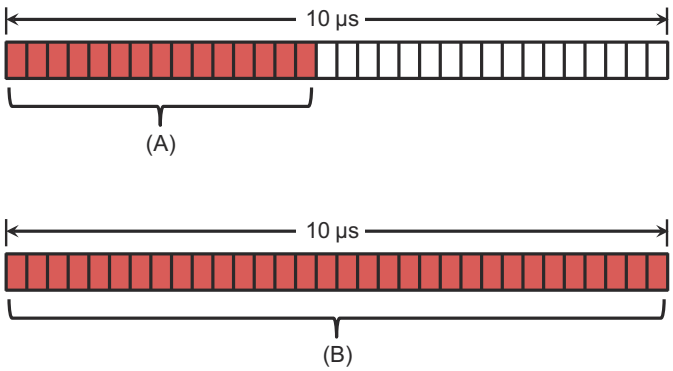


Figure 7-74. RX Error Detection

Note

Note for SGMII and RGMII modes, RX_ERR is sampled after SFD and during the payload if one occurs then it can be detected by R31 and/or INTC same as MII. The MII RX_ERR counter counts for every MII nibble.

This submodule also keeps track of a running count of receive error events within a 10 µs error detection window, as shown in Figure 7-75. The INTC is notified when 32 or more events have occurred in a 10 µs error detection window. The error detection window is not a sliding window but a non-overlapping window with no specific initialization time with respect to incoming traffic. The timer starts its 10 µs counts immediately after de-assertion of reset to the MII_RT module.



A. There are fewer than 32 consecutive error events in the 10 µs window. The detection module will not forward to the interrupt controller (INTC).
B. There are more or equal to 32 error events in the 10 µs window. The detection module will notify the interrupt controller (INTC).

Figure 7-75. Error Detection Window with Running Counter

7.3.10.2.3.5 RX Data Path Options to PRU

There are two data path options for delivering received data to the PRU, described further in the subsequent sections:

1. RX MII port → RX L1 FIFO → PRU (one word in flight)
2. RX MII port → RX L1 FIFO → RX L2 buffer → PRU (multi-word in flight)

Once the PRU has received RX data, the PRU can both manipulate received data or send data to the TX MII Interface.

7.3.10.2.3.6 RX MII Port → RX L1 FIFO → PRU

The RX L1 FIFO to PRU interface is depicted in Figure 7-76. In this mode, the data received from the MII interface is fed into the 32-byte RX L1 FIFO. The first data byte into the FIFO is automatically available in R31 of the PRU. Therefore, the PRU firmware can directly operate on this data without having to read it in a separate instruction. This allows the PRU to access receive data with low latency.

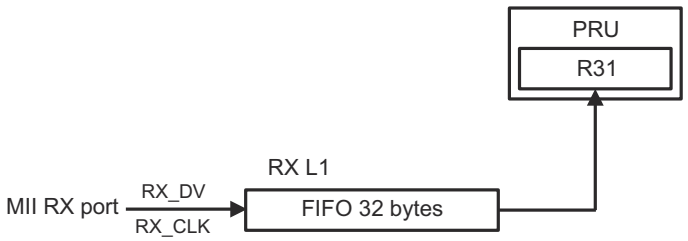


Figure 7-76. RX L1 to PRU Interface

When the new data is received, the PRU is provided with up to two bytes at a time in the R31 register, as shown in Figure 7-77. Once the PRU processes the incoming data, it instructs the MII_RT by writing to the R31 command interface bits to pop one or two bytes of data from the 32-byte RX FIFO. The pop operation causes current contents of R31 to be refreshed with new data from the incoming packet. Each time the data is popped, the status bits change to indicate so. If the pop is completed and there is no new data, the status bits immediately change to indicate no new data.

Note: The current R31 content, including data, will be lost after issuing the pop operation. If this information needs to be accessed later, the PRU should store the existing R31 content before popping new data.

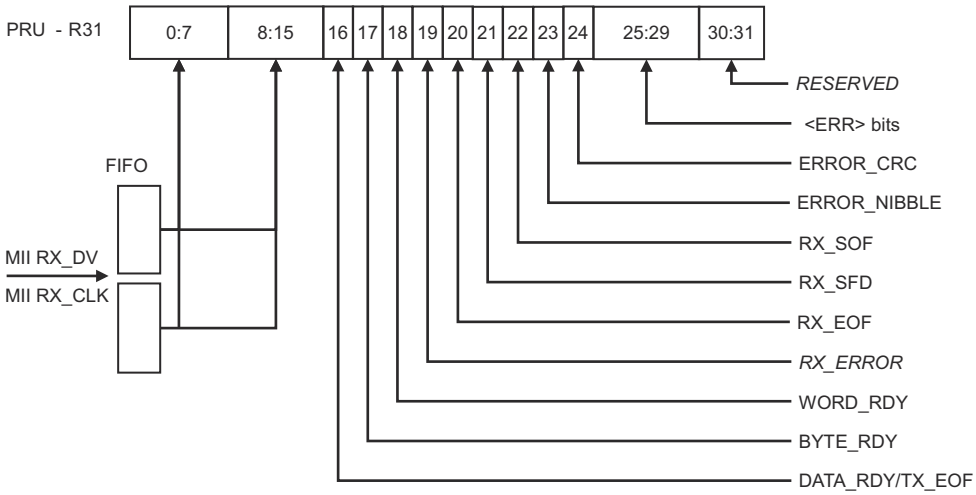


Figure 7-77. MII RX Data to PRU R31 (R) and RX FIFO

Table 7-82 describes the receive interface data and status contents provided by the R31 register. These contents are available when R31 is read. To configure this register, the PRU GPI mode should be set for MII_RT mode in the CFG register space. Note the following:

1. If the data from receive path is not read in time, it could cause an overflow event because the data is still continuously provided to the 32-byte receive FIFO. Due to the receive FIFO overflow, the data gets automatically discarded to avoid lack of space in the FIFO. At the same time, an interrupt is raised to the INTC through a system event (PRU<n>_RX_OVERFLOW). To detect an overflow condition, the PRU should poll for this system event condition and a RX RESET command through the R31 command interface is required to clear out from this condition. Note that the received Ethernet frame is corrupted and should not be used for further processing as bytes have been dropped due to the overflow condition. A FIFO reset is recommended.
2. The receive data in the R31 register is available following synchronization to the PRU clock domain. So, there is a finite delay (120 ns) when data is available from MII interface and it is accessible to the PRU.
3. The receive FIFO also has the capability to be reset through software. When reset, all contents of receive FIFO are purged and it may result in the current frame not being received as expected. When a frame is being received and the PRU resets the RX FIFO, the remaining frame is not placed into the RX FIFO. However, any new frame arriving on the receive MII port will be stored in the FIFO.

Table 7-82. PRU R31: Receive Interface Data and Status (Read Mode)

Bits	Field Name	Description
31-30	RESERVED	In case of register interface, these bits are provided to PRU by other modules in PRU-ICSS. From the MII_RT module point of view, these bits are always zero.
29	RX_MIN_FRM_CNT_ERR	RX_MIN_FRM_CNT_ERR is set to 1 when the count of total bytes of incoming frame is less than the value defined by RX_MIN_FRM_CNT. RX_MIN_FRM_CNT_ERR is cleared by RX_ERROR_CLR. Cleared by RX_ERROR_CLR or RX_L2_DONE. Note, during backpressure the status will not get updated by a new paket in L1 FIFO. The flag is valid for the current paket in L2 FIFO.
28	RX_MAX_FRM_CNT_ERR	RX_MAX_FRM_CNT_ERR is set to 1 when the count of total bytes of incoming frame is more than the value defined by RX_MAX_FRM_CNT_ERR. RX_MAX_FRM_CNT_ERR is cleared by RX_ERROR_CLR. Cleared by RX_ERROR_CLR or RX_L2_DONE. Note, during backpressure the status will not get updated by a new paket in L1 FIFO. The flag is valid for the current paket in L2 FIFO.
27	RX_EOF_ERROR	RX_EOF_ERROR is set to 1 when an RX_EOF event or RX_ERROR event occurs. RX_EOF_ERROR is cleared by RX_ EOF_CLR and/or RX_ ERROR_CLR.
26	RX_MAX_PRE_CNT_ERR	RX_MAX_PRE_CNT_ERR is set to 1 when the number of nibbles equaling 0x5 before SFD event (0xD5) is more than the value defined by PRUSS_MII_RT_RX_PCNT0/1 [RX_MAX_PCNT]. RX_MAX_PRE_CNT_ERR is cleared by RX_ERROR_CLR.
25	RX_ERR	RX_ERR is set to 1 when pr1_mii0/1_rxer is asserted while pr1_mii0/1_rxdv bit is set. RX_ERR is cleared by RX_ERROR_CLR.
24	ERROR_CRC	ERROR_CRC indicates that the frame has a CRC mismatch. This bit is valid when the RX_EOF bit is set. It should be noted that ERROR_CRC bit is ready in early status, which means it is calculated before data is available in RXL1 FIFO. ERROR_CRC is cleared by RX_ERROR_CLR. Cleared by RX_ERROR_CLR or RX_L2_DONE. Note, during backpressure the status will not get updated by a new paket in L1 FIFO. The flag is valid for the current paket in L2 FIFO.
23	ERROR_NIBBLE	ERROR_NIBBLE indicates that the frame ended in odd nibble. It should be considered valid only when the RX_EOF bit and pr1_mii0/1_rxdv are set. Nibble counter is enabled post SFD event. It should be noted that ERROR_NIBBLE bit is ready in early status, which means it is calculated before data is available in RXL1 FIFO. ERROR_NIBBLE is cleared by RX_ERROR_CLR.
22	RX_SOF	RX_SOF transitions from low to high when the frame data starts to arrive and pr1_mii0/1_rxdv is asserted. Note: There will be a small sync delay of 0ns – 5ns. The recommended time to clear this bit via RX_SOF_CLR is at the end of frame (EOF). It should be noted that RX_SOF bit is ready in early status, which means it is calculated before data is available in RXL1 FIFO.
21	RX_SFD	RX_SFD transitions from low to high when the SFD sequence (0xD5) post RX_SOF is observed on the receive MII data. The recommended time to clear this bit via RX_SFD_CLR is at the end of frame (EOF). It should be noted that RX_SFD bit is ready in early status, which means it is calculated before data is available in RXL1 FIFO.
20	RX_EOF	RX_EOF indicates that the frame has ended and pr1_mii0/1_rxdv is de-asserted. It also validates the CRC match bit. Note: There will be a small sync delay of 0ns – 5ns. It should be noted that RX_EOF bit is ready in early status, which means it is calculated before data is available in RXL1 FIFO. Note: Also if RX_L2_EOF_SCLR_DIS is set, then this flag will remain asserted when RX_L2 is enabled until RX_EOF_CLR. Cleared by RX_ERROR_CLR or RX_L2_DONE. Note, during backpressure the status will not get updated by a new paket in L1 FIFO. The flag is valid for the current paket in L2 FIFO.

Table 7-82. PRU R31: Receive Interface Data and Status (Read Mode) (continued)

Bits	Field Name	Description
19	RX_ERROR	RX_ERROR indicates one or more of the following errors occurred: • RX_MAX/MIN_FRM_CNT_ERR • RX_MAX/MIN_PRE_CNT_ERR • RX_ERR RX_ERROR is cleared by RX_ERROR_CLR.
18	WORD_RDY	WORD_RDY indicates that all four nibbles in R31 have valid data. There is a 2 clock cycle latency from the command RX_POP16 to WORD_RDY update. Therefore, firmware needs to insure it does not read WORD_RDY until 2 clock cycles after RX_POP16.
17	BYTE_RDY	BYTE_RDY indicates that the lower two nibbles in R31 have valid data. There is a 2 clock cycle latency from the command RX_POP8 to BYTE_RDY update. Therefore, PRU firmware needs to insure it does not read BYTE_RDY until 2 clock cycles after RX_POP8.
16	DATA_RDY/ TX_EOF	When RX_DATA_RDY_MODE_DIS = 0: DATA_RDY indicates there is valid data in R31 ready to be read. This bit goes to zero when the PRU does a POP8/16 and there is no new data left in the receive MII port. This bit is high if there is more receive data for PRU to read. There is a 2 clock cycle latency from the command RX_POP16/8 to WORD_RDY/BYTE_RDY update. Therefore, PRU firmware needs to insure it does not read BYTE_RDY/ WORD_RDY until 2 clock cycles after RX_POP16/8. When RX_DATA_RDY_MODE_DIS = 1: TX_EOF indicates an TX EOF event (i.e. a 1 --> 0 transition on TX_EN) has occurred. This bit will clear when TX_RESET is set or when new data is first loaded. PRU firmware can wait until TX_EOF = 1, then start a new TX Frame by immediately loading new data.
15-8	BYTE1	Data Byte 1. This data is available such that it is safe to read by the PRU when the DATA_RDY/BYTE_RDY/WORD_RDY bits are asserted.
7-0	BYTE0	Data Byte 0. This data is available such that it is safe to read by the PRU when the DATA_RDY/BYTE_RDY/WORD_RDY bits are asserted.

7.3.10.2.3.7 RX MII Port → RX L1 FIFO → RX L2 Buffer → PRU

The RX L2 is an optional high performance buffer between the RX L1 FIFO and the PRU. Figure 7-78 illustrates the receive data path using RX L2 buffer. This data path is characterized by multi-word in flight transactions.

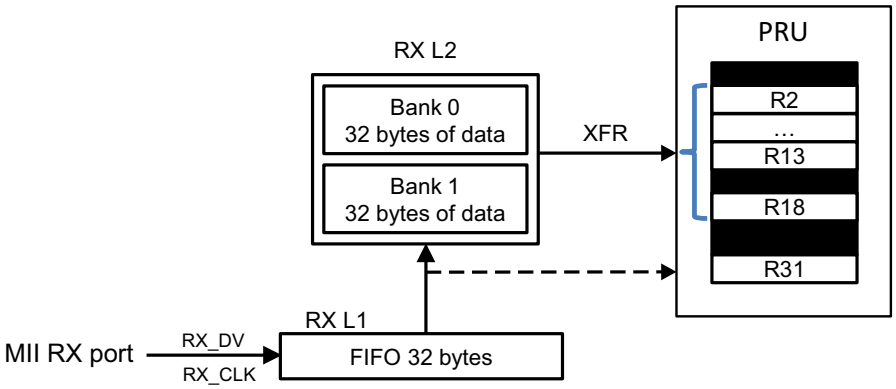


Figure 7-78. RX L2 to PRU Interface

The 64-byte RX L2 buffer is divided into two 32 byte banks, or ping/pong buffers. When the RX L2 is enabled, the incoming data from the MII RX port will transmit first to the 32 byte RX L1 FIFO. RX L1 pushes data into RX L2, starting when the first byte is ready until the final EOF marker. The RX L2 buffer will apply backpressure to the RX L1 FIFO after RX_L2_EOF event occur and until RX_L2_DONE event. Therefore, it is the PRU firmware's responsibility to fetch the data in RX L2 before it is overwritten by the cyclic buffer. The RX L1 will remain near empty, with only one byte (nibble) stored.

Each RX L2 bank holds up to 32 bytes of data, and every four nibbles (or 16 bits) of data has a corresponding 8-bit status. The data and status information are stored in packed arrays. In each bank, R2 to R9 contains the data packed array and R10 to R13 contains the status packed array. Figure 7-79 shows the relationship of the data registers and status registers. The RX L2 status registers record status information about the received data, such as ERROR_CRC, RX_ERROR, STATUS_RDY, etc. The RX L2 status register details are described in Table 7-83. Note: RX_RESET clears all Data and Status elements and resets R18.

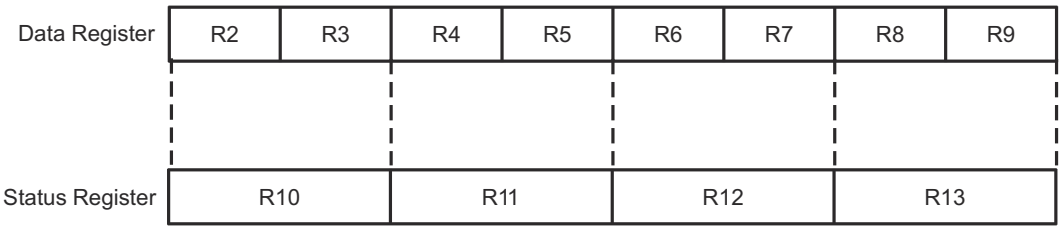


Figure 7-79. Data and Status Register Dependency

7.3.10.2.3.7.1 RX L2 Status in mode 0, none IET mode (when ICSS_M_CFG[2] RX_L2_G_EN= 0h)

Table 7-83. RX L2 Status in mode 0

Bit	Field Name	Description
7	ERROR_CRC	ERROR_CRC indicates that the frame has a CRC mismatch. This bit is valid when the RX_EOF bit is set. It should be noted that ERROR_CRC bit is ready in early status, which means it is calculated before data is available in RX L1 FIFO. ERROR_CRC will only be set for one entry, self clear on next entry.
6	ERROR_NIBBLE	ERROR_NIBBLE indicates that the frame ended in odd nibble. It should be considered valid only when the RX_EOF bit and pr1_mii0/1_rxdv are set. Nibble counter is enabled post SFD event. It should be noted that ERROR_NIBBLE bit is ready in early status, which means it is calculated before data is available in RX L1 FIFO. ERROR_NIBBLE will only be set for one entry, self clear on next entry.
5	RX_SOF	RX_SOF transitions from low to high when the frame data starts to arrive and pr1_mii0/1_rxdv is asserted. Note: There will be a small sync delay of 0ns – 5ns. It should be noted that RX_SOF bit is ready in early status, which means it is calculated before data is available in RX L1 FIFO. RX_SOF will only be set for one entry, self clear on next entry.
4	RX_SFD	RX_SFD transitions from low to high when the SFD sequence (0xD5) post RX_SOF is observed on the receive MII data. It should be noted that RX_SFD bit is ready in early status, which means it is calculated before data is available in RX + L1 FIFO. RX_SOF will only be set for one entry, self clear on next entry.
3	RX_EOF	RX_EOF indicates that the frame has ended and pr1_mii0/1_rxdv is de-asserted. It also validates the CRC match bit. Note: There will be a small sync delay of 0ns – 5ns. It should be noted that RX_EOF bit is ready in early status, which means it is calculated before data is available in RXL1 FIFO. If RX_L2_EOF_SCLR_DIS = 1, then RX_EOF will remain set until RX_EOF_CLR event. Otherwise, RX_ERROR is self-clearing on next entry.
2	RX_ERROR	RX_ERROR indicates one or more of the following errors occurred: • RX_MAX/MIN_FRM_CNT_ERR • RX_MAX/MIN_PRE_CNT_ERR • RX_ERR RX_ERROR is cleared by RX_ERROR_CLR.
1	STATUS_RDY	STATUS_RDY is set when RX_EOF or write pointer advanced by 2. This is a simple method for software to determine if RX_EOF event has occurred or new data is available. If RX_EOF is not set, all status bits are static.

Table 7-83. RX L2 Status in mode 0 (continued)

Bit	Field Name	Description
0	RX_ERR	RX_ERR is set to 1 when pr1_mii0/1_rxer is asserted while pr1_mii0/1_rxdv bit is set. It will get set for first pr1_mii0/1_rxer event and self clear on SOF for the next FRAME.

7.3.10.2.3.7.2 RX L2 XFR Identification

Bank 0 and Bank 1 are used as ping/pong buffers. RX L2 supports the reading of a write pointer in R18 that allows software to determine which bank has active write transactions, as well as the specific write address within packed data arrays.

The PRU interacts with the RX L2 buffer using the high performance XFR read instructions and broadside interface. Table 7-84 shows the device XFR ID numbers for each bank.

Table 7-84. RX L2 XFR ID

Device ID	Function	Description
20	Selects RX L2 Bank0	R2:R9 Data packed array R10:R13 Status packed array mode 0
21	Selects RX L2 Bank1	R2:R9 Data packed array R10:R13 Status packed array mode 0
20/21	Byte pointer of current write	R18[5-0] Pointer indicating location of current write in data packed array. 0 = Bank0.R2.Byte0 (default and reset value) 1 = Bank0.R2.Byte1 2 = Bank0.R2.Byte2 3 = Bank0.R2.Byte3 4 = Bank0.R3.Byte0 … 63=Bank1.R9.Byte3

7.3.10.2.3.7.3 RX L2 XFR Status

XFR read transactions are passive and have no effect on any status or other states in RX L2. The firmware can also read R18 to determine which Bank has active write transactions and the location of the transaction. With this information, the firmware can read multiple times the stable preserved data. Note: When RX L1 data is written to RX L2, the next status byte gets cleared at the same time the current status byte gets updated. The rest of the status buffer is persistent. When software is accessing any register of the ping/ pong buffer, software needs to issue an XFER read transaction to fetch the latest/current state of the ping/pong buffer. The PRU registers will not reflect the current snapshot of L2 unless an XFER is issued by software.

7.3.10.2.3.7.4 Broadside Stitch FIFO

A simple 2 deep by 32 Byte Wide broadside FIFO is attached to ID = 08 to enable the firmware to efficiently pack fragments into aligned data words.

Table 7-85. RX L2 XFR ID

Device ID	Function	Description
08	64 Bytes XOUT 1 Byte to 32 Bytes, LSB justify XIN 32 Bytes. Note: FIFO has less than 32 Bytes, it will only return the valid data LSB justified	R2:R9 Data packed array
08	It will subtract 4 bytes from the current wr_ptr	R10[0] Control
08	It will reset the rd and wr ptrs	R10[1] Reset

7.3.10.2.4 PRU-ICSS TX MII Interface

The PRU core directly drives the MII transmit interface via its R30 internal register. The contents of R30 register and RX Data from receive interface are taken and fed into a transmit FIFO (TX L2 FIFO - 64 Bytes).

Data to be transmitted is loaded into the TX L1 FIFO. The transmit FIFO (TX L1) stores up to 40 Bytes of transmit data. Note that this includes the preamble bytes. From the transmit FIFO (TX L1), the data is sent to the MII TX port of the PHY by the MII_RT transmit logic.

The transmit FIFO also has the capability to be reset through software (TX_RESET). When reset, all contents of transmit FIFO are purged and this may result in a frame not getting transmitted as expected, if the transmission is already ongoing. Any new data written in the transmit FIFO results in a new frame being composed and transmitted. An overflow event will require a TX_RESET to recover from this condition.

There are four dependencies that must be true for TX_EN to assert:

1. TX L1 FIFO not empty
2. Interpacket gap (IPG) timer expiration
3. RX_DV to TX_EN timer expiration
4. TX_EN compare timer expiration

The transmit interface also provides an underflow error signal in case there was no data loaded when TX_EN triggered. The transmit underflow signal is mapped to the INTC in PRU-ICSS. The current FIFO fill level cannot be accessed by PRU firmware. The firmware can issue an R31 command via R31 bit 29 (TX_EOF) to indicate that the last byte has been written into the TX FIFO.

7.3.10.2.4.1 TX Data Path Options to TX L1 FIFO

There are two data path options for delivering data to the TX L1 FIFO and transmit port, described further in the subsequent sections:

1. PRU → TX L1 FIFO → TX MII port
2. RX L1 FIFO → TX L1 FIFO → TX MII port

7.3.10.2.4.1.1 PRU → TX L1 FIFO → TX MII Port

The PRU can be used to feed data into the TX L1 FIFO using the R30 and R31 registers, shown in Figure 7-80. The PRU has the option to write up two or four bytes of R30 and then pushes the data into the TX L1 FIFO by writing to the R31 command interface.

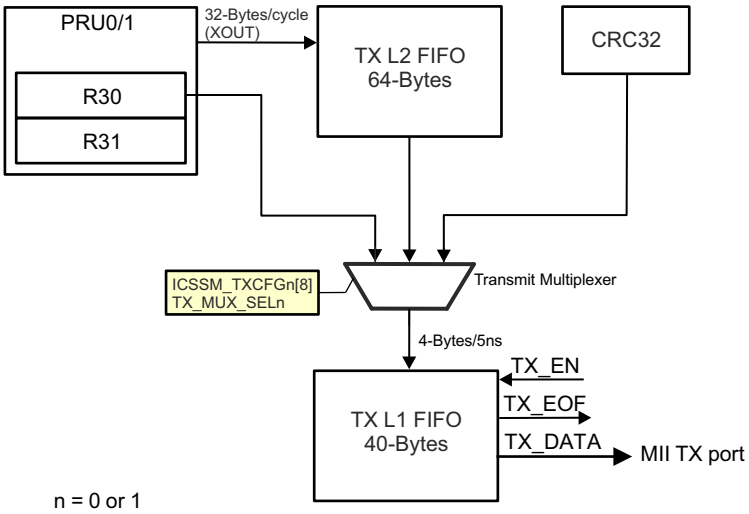


Figure 7-80. PRU to TX L1 FIFO Interface

7.3.10.2.4.1.1.1 TX L2 FIFO Features

- 64-Bytes deep TX L2 FIFO which feeds data into a 40-Bytes deep TX L1 FIFO
- Maximum of 64-Bytes Broad Side load, minimum of 1-Byte Broad Side load, R2.b0(start)

- **Note: MII_RT_TXCFG0/1[3] TX_BYTE_SWAPn bit (where n = 0 or 1) is not supported for TX L2 FIFO.**
- FIFO level status available through MII_RT_TX_FIFO_LEVEL0[7-0] TX_FIFO_LEVEL0 and MII_RT_TX_FIFO_LEVEL1[7-0] TX_FIFO_LEVEL1 registers
- 2 FIFO threshold events: 32-Bytes (available and empty) or 64-Bytes (available)
- Total bytes sent for a current/last frame is available for software
- New frame can start after TX L2 FIFO is empty, but before TX L1 FIFO is empty from an old frame
  - **Note: Only supported for RGMII and SGMII modes of operation**
  - New frame can start after 5 or more core clock cycles after it is drained, this is required to finish the CRC for the first frame

Table 7-86. TX L2 XFR Mapping

Device ID	Function	Description
40	Data	R17:R2 Data, XOUT Only 1-Byte to 64-Bytes in size LSB packed and no gaps, for example 64-Bytes push R17:R2 32-Bytes push R9:R2 16-Bytes push R5:R2 4-Bytes push R2 7-Bytes push R3(b2.b0):R2 1-Bytes push R2(b0) Can do back to back
40	Control	Control of TX L2 FIFO
40	Status	Status of TX L2 FIFO

Table 7-87. TX L2 Control

BS ID	BS R	Bit	Name	Type	Reset	Description
40	R18	1-0	TAG insertion mode	WO	0h	Sets the TAG mode for next frame or current frame. It will have a one time action per frame. After action, software must rearm for a new action 1 cmd per packet. MII_RT_TXCFG0/1[1] TX_AUTO_PREAMBLEn bit (where n = 0 or 1) must be set to 1h. Note: This bit is self cleared.
40	R18	2	VLAN removal	WO	0h	If set, it will remove 4-Bytes of VLAN. This is only valid when TX L2 FIFO is enabled through ICSS_M_CFG[1] TX_L2_ENABLE bit (value 1h) and MII_RT_TXCFG0/1[1] TX_AUTO_PREAMBLEn = 1h (where n = 0 or 1) then Byte13, Byte14, Byte15, Byte16 will get removed. Note: Byte1 is the first byte which is pushed by PRU core. Note that the first 2 bytes of the VLAN must match the value in MII_RT_TX_VLAN_TYPE_TAG_PORT0/1[15-0] TX_VLAN_TYPE_TAG bit field (reset state is 81h). If not, the 4-Bytes will NOT get removed. A RXVLANRemoval flag will get set if the action occurred Allow all combos of TAG + VLAN IN. Note: This will be defined in a matrix. Note: This bit is self cleared.
40	R18	4	EXP_FRAME	WO	0h	Must be set for all EXP_FRAME. We have 3 types of frames: • Implicit EXP_FRAME not set PRE_FRAME • Not set SMD == 0xd5 EXP_FRAME (use a SMD_EXP) • EXP_FRAME set PRE_FRAME not set. PRE_FRAME (use MII_RT_SMDT1S_CFG .SMDT1S_n for intial and MII_RT_SMDT1C_CFG .SMDT1C_n + MII_RT_FRAG_CNT_CFG .FRAG_CNT_n for non intial) EXP_FRAME set PRE_FRAME not set. Note: This bit is self cleared on EOF.

Table 7-87. TX L2 Control (continued)

BS ID	BS R	Bit	Name	Type	Reset	Description
40	R18	6	RESERVED	R	0h	Reserved
40	R18	7	EOF_MCRC_REQ	WO		Set this bit before TX L1 FIFO is empty to generate a MCRC vs CRC. Note: This bit is self cleared on EOF.
40	R18	11-8	RESERVED	R	0h	Reserved

Table 7-88. TX L2 Status

BS ID	BS R	Bit	Name	Type	Reset	Description
40	R19	11-0	TXL2ByteSentCount	R	0h	This bit field defines the number of bytes transmitted to TX L1 FIFO. The count will remain persistent until the next frame starts. This will get used to determine the number of bytes which got transmitted after a Preemption event. This includes all data pushed by the PRU core, which did get transmitted. It does not include the CRC.
40	R19	12	RESERVED	R	0h	Reserved
40	R19	13	RXVLAN Removal	R/W1C	0h	This bit will be set when VLAN removal occured. VLAN removal will only occur if TPID value is equal to the value in MII_RT_TX_VLAN_TYPE_TAG_PORT0/1[15-0] TX_VLAN_TYPE_TAG bit field (reset state is 81h). Software can clear sticky used for debug.
40	R19	14	RESERVED	R	0h	Reserved
40	R19	15	RESERVED	R	0h	Reserved
40	R19	22-16	TXL2Occ	R	0h	This bit field defines the current number of bytes in TX L2 FIFO. 0h: 0 bytes, or empty FIFO buffer 1h: 1 byte ... 64h = 64 bytes, or full FIFO buffer

Table 7-89. TX L2 TAG Modes

TAG Mode	At	What to Push/Add to the Frame
0	None	Nothing
1	Push TAG1	Byte 13 = VLAN_PORT<1/0>[7-0] Byte 14 = VLAN_PORT<1/0>[15-8] Byte 15 = VLAN_PORT<1/0>[23-16] Byte 16 = VLAN_PORT<1/0>[31-24] Used for Host. Host -> PRU-ICSS -> add VLAN TAG -> Port SFD offset issues
2	Push TAG2	Byte 13 = HTAG_PORT<1/0>[7-0] Byte 14 = HTAG_PORT<1/0>[15-8] Byte 15 = HTAG_PORT<1/0>[23-16] Byte 16 = HTAG_PORT<1/0>[31-24] Byte 17 = SEQ_PORT<1/0>[7-0] Byte 18 = SEQ_PORT<1/0>[15-8]

Table 7-89. TX L2 TAG Modes (continued)

TAG Mode	At	What to Push/Add to the Frame
3	Push TAG3	Byte 13 = VLAN_PORT<1/0>[7-0] Byte 14 = VLAN_PORT<1/0>[15-8] Byte 15 = VLAN_PORT<1/0>[23-16] Byte 16 = VLAN_PORT<1/0>[31-24] Byte 17 = HTAG_PORT<1/0>[7-0] Byte 18 = HTAG_PORT<1/0>[15-8] Byte 19 = HTAG_PORT<1/0>[23-16] Byte 20 = HTAG_PORT<1/0>[31-24] Byte 21 = SEQ_PORT<1/0>[7-0] Byte 22 = SEQ_PORT<1/0>[15-8]

7.3.10.2.4.1.1.2 TX Insertion

There are 3 TAG Insertion modes that software can select. Note that the mode must be selected before the first byte is pushed into the TX FIFO. The values, pushed into the TX FIFO are defined in the MII_RT_TX_FIFO_LEVEL0/1[7-0] TX_FIFO_LEVELn registers (where n = 0 or 1).

Table 7-90. TX VLAN_TAG Cases

Case	RM VLAN	ADD VLAN	ADD HSR	In packet	Out packet
1	1	0	0	If pkt[B13:B14] == vlan_type_id	B16, B15, B14, B13 will be removed. If packet is less than 64-Bytes, 0s will get added before the CRC.
2	1	0	0	If pkt[B13:B14] != vlan_type_id	No effect.
3	0	1	0	X	VLAN_TAG added 4-Bytes. Start B13.
4	0	0	1	X	HSR_TAG added 6-Bytes. Start B13.
5	0	1	1	X	VLAN+TAG and HSR_TAG added 10-Bytes. Start B13.
6	1	1	0	If pkt[B13:B14] == vlan_type_id	B16, B15, B14, B13 will be replaced with VLAN_TAG.
7	1	1	0	If pkt[B13:B14] != vlan_type_id	No effect.
8	1	1	1	If pkt[B13:B14] == vlan_type_id	B16, B15, B14, B13 will be replaced with VLAN_TAG. Then HSR_TAG wil be added.
9	1	1	1	If pkt[B13:B14] != vlan_type_id	No effect.
10	1	0	1	If pkt[B13:B14] = vlan_type_id	B16, B15, B14, B13 will be removed and HSR_TAG added 6-Bytes. Start B13.
11	1	0	1	If pkt[B13:B14] != vlan_type_id	HSR_TAG added 6-Bytes. Start B13

7.3.10.2.4.1.1.3 TX Preemption

Case 1) Preemptible frame which got fragmented.

1. Idle -> preemptible frame when:
   - PRE_FRAME is set before first data push
2. preemptible -> frag when
   - EOF_MCRC_REQ is set after the last data is pushed into TX L2 FIFO and before TX L1 FIFO is empty
3. frag -> frag when
   - PRE_FRAME is set before first data push and EOF_MCRC_REQ is set after the last data is pushed into TX L2 FIFO and before TX L1 FIFO is empty
4. frag -> lfrag when:
   - PRE_FRAME is set before first data push and TX_EOF_REQ set after the last data is pushed into TX L2 FIFO and before TX L1 FIFO is empty
5. lfrag -> Idle when CRC is pushed into TX L1 FIFO

Case 2) Preemptible frame which did not get fragmented.

1. Idle -> preemptible frame when:
   - PRE_FRAME is set before first data push
2. preemptible -> Idle when
   - TX_EOF_REQ is set before TX L1 FIFO is empty

Note: Express frames can and will occur between the fragments.

Rules:

- Preemptible must get set before the first data is pushed into TX L2 FIFO
- EOF_MCRC_REQ is set after the last data is pushed into TX L2 FIFO and before TX L1 FIFO is empty
- TX_EOF_REQ can only get asserted on the last frag or preemptible frame which did not get fragmented. It can not get asserted in none last fragments.

Note: TX_EOF_REQ can not get asserted when EOF_MCRC_REQ is asserted.

7.3.10.2.4.1.1.3.1 TX Preemption Programming Model

Start a new frame.

1. Wait until R31.TX_EOF event
2. Load data into TX L2 FIFO until the full frame is completed
3. Issue a R30.TX_EOF + TX_CRC_HIGH + TX_CRC_LOW

Figure 7-81 shows the R30 transmit interface. The lower 16 bits of the R30 (or FIFO transmit word) contain transmit data nibbles. When MII_RT_TXCFG0/1[11] TX_32_MODE_ENn = 0h - default value (where n = 0 or 1), then the upper 16 bits contain mask information. Alternatively, when MII_RT_TXCFG0/1[11] TX_32_MODE_ENn = 1h (where n = 0 or 1), then the upper 16 bits contain transmit data nibbles. The operation to be performed on the transmit interface is controlled by PRU writes to the R31 command interface. Table 7-91 describes the supported configurations for 8, 16, and 32 bit TX push operations.

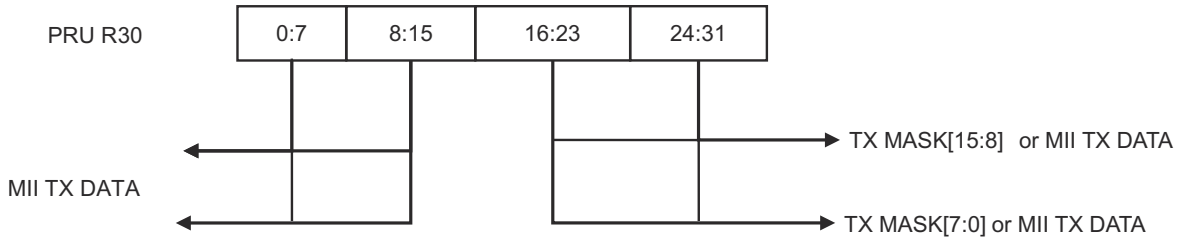


Figure 7-81. PRU to TX MII Interface

Table 7-91. TX Push

R31[25] TX_PUSH16/32	R31[24] TX_PUSH8/32	Supported R30 bits	TX_32_MODE_EN	TX_BYTE_SWAP	TX Push Action
0	1	X	0	X	8 bits of TXDATA (R30[7-0]) pushed post TX mask
1	0	X	0	X	16 bits of TXDATA (R30[15-0]) pushed post TX mask
1	1	X	0	X	Illegal
X	X	0x000000FF	1	0	8 bits of TXDATA (R30[7-0]) pushed
X	X	0x0000FFFF	1	0	16 bits of TXDATA (R30[15-0]) pushed
X	X	0xFFFFFFFF	1	X	32 bits of TXDATA (R30[31-0]) pushed
X	X	All other - reserved	1	X	Reserved

Using MII_RT_TXCFG0/1[11] TX_32_MODE_ENn = 0h and the TX mask, the PRU can send a mix of R30 and RX L1 FIFO data to the TX L1 FIFO. Note the TX mask is only available when the PRU is fed one word or byte at a time by the RX L1 FIFO. It is not applicable when the RX L2 buffer is enabled. To disable TX mask, set TXMASK to 0xFFFF.

As shown in Figure 7-82, the PRU drives the MII transmit interface through its R30 register. The contents of R30 and RX data from the receive interface (RX L1 FIFO) are taken and fed into a 40-Bytes transmit FIFO (TX L1 FIFO).

If MII_RT_TXCFG0/1[11] TX_32_MODE_ENn = 0h (where n = 0 or 1), then before transmission, a mask is applied to the data portion of the R30 register. By using the mask, the PRU firmware can control whether received data from the RX L1 FIFO is sent to transmit, R30 data is sent to transmit, or a mix of the two is sent. The Boolean equation that is used by MII_RT to compose TX data is:

TXDATA[7/15-0] = (R30[7/15-0] & MASK[7/15-0]) | (RXDATA[7/15-0] & ~MASK [7/15-0])

As shown in the equation, a mask of FFh will lead to the R30[7:0] being transmitted in an 8-bit transmit operation. A mask of 0h will lead to receive data being sent out in a 16-bit transmit operation.

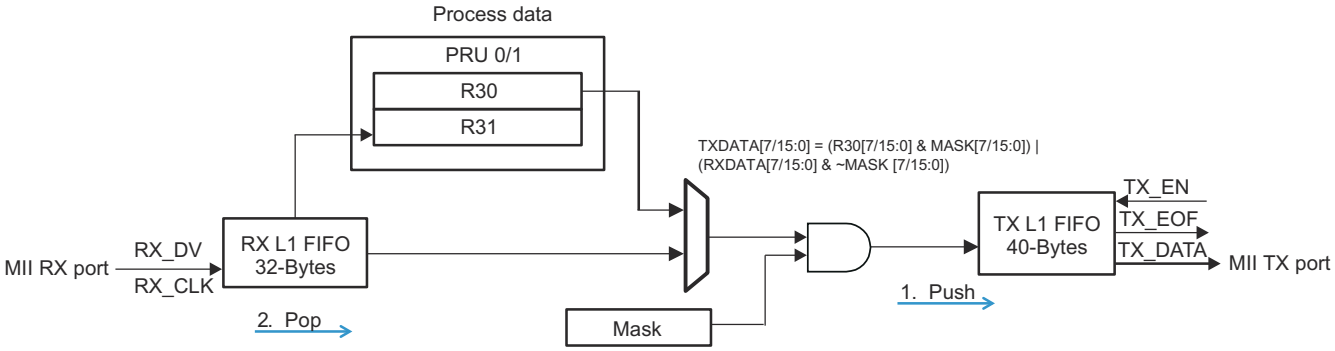


Figure 7-82. TX Mask Mode (MII_RT_TXCFG0/1[TX_32_MODE_ENn] = 0h)

7.3.10.2.4.1.2 RX L1 FIFO → TX L1 FIFO (Direct Connection) → TX MII Port

When MII_RT_TXCFG0/1[9] PRE_TX_AUTO_SEQUENCEn is set to 1h (where n = 0 or 1), the data frame is passed from the RX L1 FIFO to TX L1 FIFO without any interaction of the PRU. This mode of operations is shown in Figure 7-83. The RX L1 FIFO will push data into TX L1 FIFO as long as it is enabled and not full.

There is no PRU dependency in this mode and no option for the PRU to perform any operation to the TX L1 FIFO. RX_RESET clears all data and status elements.

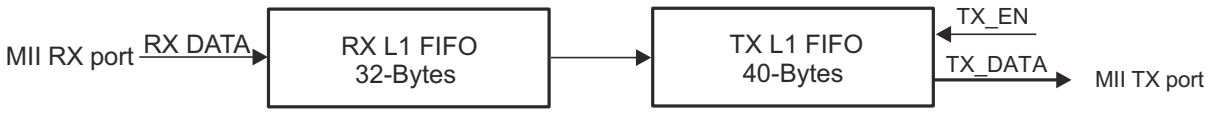


Figure 7-83. RX L1 to TX L1 Interface

For ESC protocols, software should enable [6]RX_AUTO_FWD_PRE0/1 and [4]RX_L2_EN0/1 bits in MII_RT_RXCFG0/1 registers.

For non ESC protocols, software can enable MII_RT_TXCFG0/1[1] TX_AUTO_PREAMBLEn and MII_RT_RXCFG0/1[2] RX_CUT_PREAMBLEn bit (where n = 0 or 1) to insure full preamble is generated for each TX frame.

The PRU core can read the passing through frame by polling the standard R31 register. In Direct mode, the PRU R31 Command is ignored and disabled, except for TX_RESET and RX_RESET.

The following are the legal configurations supported for Direct Connection:

- Configuration 1:
  - PORT1.RX -> PRU1 (snoop only)

  - PORT1.RX -> PORT0.TX
- Configuration 2:
  - PORT0.RX -> PRU0 (snoop only)
  - PORT0.RX -> PORT1.TX
- Configuration 3:
  - PORT1.RX -> PORT1.TX
- Configuration 4:
  - PORT0.RX -> PORT0.TX

7.3.10.2.5 PRU R31 Command Interface

The PRU uses writes to R31[31-16] to control the reception and transmission of packets in direct and register mode. Table 7-92 lists the available commands. Each bit in the table is a single clock pulse output from the PRU. When more than one action is to be performed in the same instant, the PRU firmware must set those command bits in one instruction.

Table 7-92. PRU R31: Command Interface (Write Mode)

Bit	Command	Description
31	TX_CRC_ERR	TX_CRC_ERR command when set will add 0xA5 byte to the TX L1 FIFO if the current FCS is valid. This bit can only be set with the TX_EOF command and optionally with the TX_ERROR_NIBBLE command. It cannot get set with any other commands, and the PRU firmware must wait > 2 clocks from the last command. Note: For proper operations auto-forward preamble must be enabled.
30	TX_RESET	TX_RESET command is used to reset the transmit FIFO and clear all its contents. This is required to recover from a TX FIFO overrun.
29	TX_EOF	TX_EOF command is used to indicate that the data loaded is considered last for the current frame
28	TX_ERROR_NIBBLE	TX_ERROR_NIBBLE command is used to insert an error nibble. This makes the frame invalid. Also, it will add 0x0 after the 32-bit CRC.
27	TX_CRC_HIGH	TX_CRC_HIGH command ends the CRC calculations and pushes CRC[31-16] to append to the outgoing frame in the TX L1 FIFO. Note: TX_CRC0/1 will become valid after 6 clock cycles.
26	TX_CRC_LOW	TX_CRC_LOW command pushes CRC[15-0] to append to the outgoing frame in the TX L1 FIFO.
25	TX_PUSH16	TX_PUSH16 command pushes R30[15-0] when MII_RT_TXCFG0/1[11] TX_32_MODE_ENn = 0h (where n = 0 or 1). See Table 7-91,*TX Push* for more details. Note: There are no restrictions on concurrent PUSH/POP nor R30 requirements to maintain data. Back to back PUSH is supported.
24	TX_PUSH8	TX_PUSH8 command pushes R30[7-0] when MII_RT_TXCFG0/1[11] TX_32_MODE_ENn = 0h (where n = 0 or 1). See Table 7-91,*TX Push* for more details. Note: There are no restrictions on concurrent PUSH/POP nor R30 requirements to maintain data. Back to back PUSH is supported.
23	RX_ ERROR_CLR	RX_ERROR_CLR command is used to clear RX_ ERROR indicator bit by writing 1h.
22	RX_EOF_CLR	RX_EOF_CLR command is used to clear RX_EOF status indicator bit by writing 1h.
21	RX_SFD_CLR	RX_SFD_CLR command is used to clear RX_SFD indicator bit by writing 1h.
20	RX_SOF_CLR	RX_SOF_CLR command is used to clear RX_SOF indicator bit by writing 1h.
19	Reserved	Reserved

Table 7-92. PRU R31: Command Interface (Write Mode) (continued)

Bit	Command	Description
18	RX_RESET	RX_RESET is used to reset the receive FIFO and clear all contents. This is required to recover from a RX FIFO overrun, if software does not want to undrain. The typical use case is assertion after RX_EOF. If asserted during an active frame, the following actions will occur: 1. Terminate the current frame 2. Block/terminate all new data 3. Flush/clear all FIFO elements 4. Cause RX state machine into an idle state 5. Cause EOF event 6. Cause minimum frame error, if you abort before minimum size reached
17	RX_POP16	RX_POP16 command advances the receive traffic by two bytes. This is only required when you are using R31 to read the data. After R31[15-0] is ready to read by PRU, it will set 1h to WORD_RDY, and the next new data will be allowed to advance. RX_POP16 to WORD_RDY update has 2 clock cycles latency. Firmware needs to insure it does not read WORD_RDY/BYTE_RDY until 2 clock cycles after RX_POP16.
16	RX_POP8	RX_POP8 command advances the receive traffic by one bytes. This is only required when you are using R31 to read the data. After R31[7-0] is ready to read by PRU, it will set 1h to BYTE_RDY, and the next new data will be allowed to advance. RX_POP8 to BYTE_RDY update has 2 clock cycles latency. Firmware needs to insure it does not read WORD_RDY/BYTE_RDY until 2 clock cycles after RX_POP8.

7.3.10.2.6 Other Configuration Options

7.3.10.2.6.1 Nibble and Byte Order

The PRU core is little endian. To support big endian, the MII_RT supports optional nibble swapping on both the RX and TX side.

On the receive side, the order of the two data bytes in RX R31 and the RX L2 buffer are configurable through the RX_BYTE_SWAP0/1 bit in the MII_RT_RXCFG0/1 registers, as shown in Table 7-93. Note: The Nibble0 is the first nibble received.

Table 7-93. RX Nibble and Byte Order

Configuration	Order
MII_RT_RXCFG0/1[5] RX_BYTE_SWAPn = 0h (default), where n = 0 or 1	R31[15-8] / RXL2[15-8] = Byte1{Nibble3,Nibble2} R31[7-0] / RXL2[7-0] = Byte0{Nibble1,Nibble0}
MII_RT_RXCFG0/1[5] RX_BYTE_SWAPn = 1h, where n = 0 or 1	R31[15-8] / RXL2[15-8] = Byte0{Nibble1,Nibble0} R31[7-0] / RXL2[7-0] = Byte1{Nibble3,Nibble2}

On the transmit side, the order of the two data bytes and mask bytes in TX R30 are configurable through the TX_BYTE_SWAP0/1 bit in the MII_RT_TXCFG0/1 registers, as shown in Table 7-94. Note the Nibble0 is the first nibble transmitted.

Table 7-94. TX Nibble and Byte Order

Configuration	Order
MII_RT_TXCFG0/1[3] TX_BYTE_SWAPn = 0h (default), where n = 0 or 1	If MII_RT_TXCFG0/1[11] TX_32_MODE_ENn = 0h, where n = 0 or 1 R30[15-8] = Byte1{Nibble3,Nibble2} R30[7-0] = Byte0{Nibble1,Nibble0} R30[31-24] = TX_MASK[15-8] R30[23-16] = TX_MASK[7-0] If MII_RT_TXCFG0/1[11] TX_32_MODE_ENn = 1h, R30[31-24] = Byte3{Nibble7,Nibble6} R30[23-16] = Byte2{Nibble5,Nibble4} R30[15-8] = Byte1{Nibble3,Nibble2} R30[7-0] = Byte0{Nibble1,Nibble0}

Table 7-94. TX Nibble and Byte Order (continued)

Configuration	Order
MII_RT_TXCFG0/1[3] TX_BYTE_SWAPn = 1h, where n = 0 or 1	If MII_RT_TXCFG0/1[11] TX_32_MODE_EN = 0h, R30[15-8] = Byte0{Nibble1,Nibble0} R30[7-0] = Byte1{Nibble3,Nibble2} R30[31-24] = TX_MASK[7-0] R30[23-16] = TX_MASK[15-8] If MII_RT_TXCFG0/1[11] TX_32_MODE_EN = 1h, Only 32-bit push is supported. R30[31-24] = Byte0{Nibble1,Nibble0} R30[23-16] = Byte1{Nibble3,Nibble2} R30[15-8] = Byte2{Nibble5,Nibble4} R30[7-0] = Byte3{Nibble7,Nibble6}

7.3.10.2.6.2 MII_RT Preamble Source

The MII_RT module has the option to preserve and forward a received preamble in the TX data stream, use a preamble provided by the PRU, or auto-generate a preamble. These configurations are highlighted in Table 7-95.

Table 7-95. Preamble Configuration Options

RX_CUT_PREAMBLE	Determines whether RX preamble is passed to the RX L1/L2 FIFO
RX_AUTO_FWD_PRE	Determines whether RX preamble is automatically passed to TX L1 FIFO
TX_AUTO_PREAMBLE	TX interface logic auto-generates and appends preamble to TX data stream with the first push of data into the TX L1 FIFO. Note that enabling this option does fill the TX FIFO with the preamble length, hence software has to consider this to not overrun the TX FIFO.

7.3.10.2.6.3 PRU and MII Port Multiplexer

The MII_RT module supports configurable PRU core to MII TXn / RXn port mapping. By default, PRU0 is mapped to TX1 and RX0 and PRU1 is mapped to TX0 and RX1. However, the system supports the flexibility to map any PRU core to any TX and RX port. For example, the input to PRU0 can be either RX_MII0 or RX_MII1. Similarly, the input to TX_MII0 can be either PRU0 or PRU1.

7.3.10.2.6.3.1 Receive Multiplexer

A multiplexer is provided to allow selecting either of the two MII interfaces (RX_MII0 or RX_MII1) for the receive data that is sent to PRU. Figure 7-84 shows a simple diagram of PRU receive multiplexer.

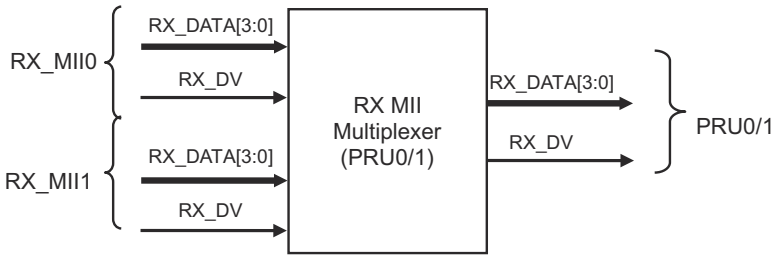


Figure 7-84. MII Receive Multiplexer

There are two receive multiplexer instances to enable selection of RX MII path for each PRU. The select lines of the RX multiplexers are driven from the PRU-ICSS programmable registers (MII_RT_RXCFG0/1[3] RX_MUX_SELn, where n = 0 or 1).

7.3.10.2.6.3.2 Transmit Multiplexer

On the MII transmit ports, there is a multiplexer for each MII transmit port that enables selection of either the transmit data from the PRUs or from the RX MII interface of the other MII interface. Figure 7-85 shows a simple diagram of PRU transmit multiplexer.

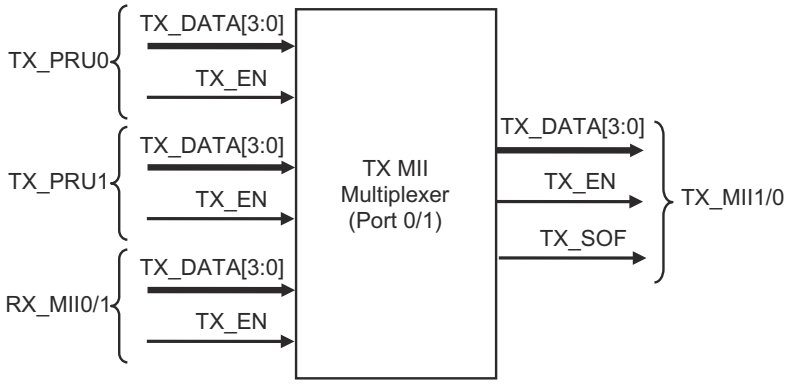


Figure 7-85. MII Transmit Multiplexer

The transmit multiplexers enable the PRU-ICSS to either operate in a bypass mode where the PRU is not involved in processing MII traffic or use of one of the PRU cores for transmitting data into the MII interface. There are two instances of the TX MII multiplexer and the select lines for each TX multiplexer are provided by the PRU-ICSS programmable registers (MII_RT_TXCFG0/1[8] TX_MUX_SELn, where n = 0 or 1). The select lines are common between register and FIFO interface. It is expected that the select lines will not change during the course of a frame so that can avoid data exchange error.

7.3.10.2.6.4 RX L2 Scratch Pad

When the RX L2 is disabled (MII_RT_RXCFG0/1[4] RX_L2_ENn = 0h, where n = 0 or 1), the RX L2 banks can be used as a generic scratch pad. In scratch pad mode, RX L2 Bank0 and RX L2 Bank1 operate like simple read/write memory mapped registers. All XFR size and start operations are supported. RX_RESET has no effect in this mode. This mode is shown in Figure 7-86.

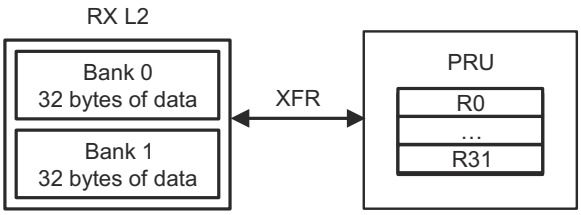


Figure 7-86. Scratch Pad Mode

7.3.11 PRU-ICSS MII MDIO Module

This section describes the PRU-ICSS (where n = 0 or 1) integrated MII management interface (MII_MDIO module).

7.3.11.1 PRU-ICSS MII MDIO Overview

The following features are supported:

- Clause 22 and Clause 45.
- Up to 32 PHY addresses.
- Two user access registers to control and monitor up to two PHYs simultaneously.
- Slave interface for configuration and control (MII RT MDIO CFG)
- Each PHY can be individually enabled to be polled.
- The inter-poll gap between PHY polls can be changed.
- State Change Mode of operation to monitor up to 32 PHYs simultaneously.
- Manual control by software for GPIO operations.

The PRU-ICSS MII MDIO management I/F module implements the *802.3 serial management interface* to interrogate and control two Ethernet PHYs simultaneously using a shared two-wire bus. Figure 7-87 shows a device with two MACs, each connected to an Ethernet PHY, being managed by the MII interface module using a shared bus.

The Figure 7-87 gives an overview of the MII MDIO management interface.

Note

This MDIO Interface is dedicated for the PRU-ICSS MII Ports. This device also makes use of another MDIO interface that is dedticated for the CPSW.

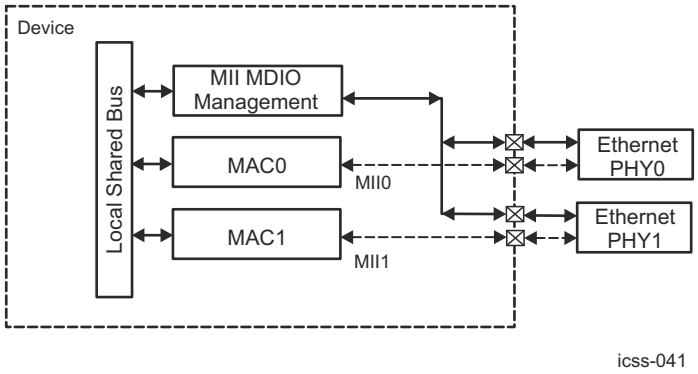


Figure 7-87. Device PRU-ICSS MII MDIO Management Interface Overview

7.3.11.2 PRU-ICSS MII MDIO Functional Description

The MII Management interface incorporates:

- *MDIO Registers* - The MDIO register block provides a VBUSP 3.0 compliant slave interface to the MDIO module. Host interaction with this module is facilitated through the registers in this block.
- *Control and Schedule* - The control and register logic in the MDIO module contain the state machine and scheduling logic which control the wire side operation.
- *MDIO Interface* - The MDIO interface block provides the serial interface to the MDIO interface.

The MDIO logic is fully synchronous to the PRU-ICSS local shared bus clock.

7.3.11.2.1 MDIO Clause 22 Frame Formats

The below Table 7-96 shows the read and write format of the Clause 22 Management interface frames.

Table 7-96. MDIO Clause 22 Frame Formats

Preamble	Start Delimiter	Operation Code	PHY Address	Register Address	Turnaround	Data
MDIO Clause 22 Read Frame Format						

Table 7-96. MDIO Clause 22 Frame Formats (continued)

Preamble	Start Delimiter	Operation Code	PHY Address	Register Address	Turnaround	Data
FFFFFFFFh	01	10	AAAAA	RRRRR	Z0	DDDD.DDDD.DDDD.DDDD
MDIO Clause 22 Write Frame Format						
FFFFFFFFh	01	01	AAAAA	RRRRR	10	DDDD.DDDD.DDDD.DDDD

The default or idle state of the two wire serial interface is a logic one. All tri-state drivers should be disabled and the PHY's pull-up resistor will pull the MDIO line to a logic one. Prior to initiating any other transaction, the station management entity shall send a preamble sequence of 32 contiguous logic one bits on the MDIO line with 32 corresponding cycles on MDCLK to provide the PHY with a pattern that it can use to establish synchronization. A PHY shall observe a sequence of 32 contiguous logic one bits on MDIO with 32 corresponding MDCLK cycles before it responds to any other transaction.

Preamble
The start of a frame is indicated by a preamble, which consists of a sequence of 32 contiguous bits all of which are a "1". This sequence provides the Ethernet PHY a pattern to use to establish synchronization.

Start Delimiter
The preamble is followed by the start delimiter which is indicated by a "01" pattern. The pattern assures transitions from the default logic one state to zero and back to one.

Operation Code
The operation code for a read is "10", while the operation code for a write is a "01".

Ethernet PHY Address
The PHY address is 5 bits allowing 32 unique values. The first bit transmitted is the MSB of the PHY address.

Register Address
The Register address is 5 bits allowing 32 registers to be addressed within each PHY.

Turnaround
An idle bit time during which no device actively drives the MDIO signal shall be inserted between the *Register Address* field and the *Data* field of a read frame in order to avoid contention. During a read frame, the PHY shall drive a zero bit onto MDIO for the first bit time following the idle bit and preceding the Data field. During a write frame, this field shall consist of a one bit followed by a zero bit.

Data
The Data field is 16 bits. The first bit transmitted and received is the MSB of the data word.

7.3.11.2.1.1 PRU-ICSS MDIO Control and Interface Signals

The Table 7-97 shows the PRU_ICSS (where n = 0 to 2) MII MDIO signals and their availability at the device boundary.

Table 7-97. PRU-ICSS MII MDIO Control and Interface Signals

MDIO Control Signals			
Pin Name	**Type**	**Available as device I/O**	**Function**
MDIO_LINKINT[1:0]	O	N.A.	Serial interface link change interrupt. Indicates a change in the state of the PHY link.
MDIO_USERINT[1:0]	O	N.A.	Serial interface user command event complete interrupt.
MDIO Interface Signals			
Pin Name	**Type**	**Available as device I/O**	**Function**
MDIO_I	I	device bidirectioanal **pr0_mdio_data**, and pr1_mdio_mdclk pin in input mode	Serial data input

Table 7-97. PRU-ICSS MII MDIO Control and Interface Signals (continued)

MDIO Control Signals			
MDIO_O	O	device bidirectional **pr0_mdio_data** and pr1_mdio_mdclk pin in output mode	Serial data output
MDIO_OE_N	O	N.A.	Serial data output enable. Asserted "0" when data output is valid
MDCLK_O	O	device output - **pr0_mdio_mdclk** pr1_mdio_mdclk	Serial clock output
MLINK_I[1:0]	I	N.A.	Optional link status inputs from PHY. Each input is connected to a single PHY. Unused inputs are tied '0'.

7.3.11.2.2 MDIO Clause 45 Frame Formats

The below Table 7-98 shows the address frame format. Table 7-99 shows read, and write format of the supported Clause 45 frames. Post-increment accesses are not supported.

Table 7-98. MDIO Clause 45 Address Frame Formats

Pre-amble	Start Delimiter	Operation Code	PHY Address	MMD Number	Turnaround	Address
MDIO Clause 45 Address Frame Format						
FFFFFFFFh	00	00	AAAAA	RRRRR	10	AAAA.AAAA.AAAA.AAAA

Table 7-99. MDIO Clause 45 Frame Formats

Pre-amble	Start Delimiter	Operation Code	PHY Address	MMD Number	Turnaround	Data
MDIO Clause 45 Read Frame Format						
FFFFFFFFh	00	11	AAAAA	RRRRR	Z0	DDDD.DDDD.DDDD.DDDD
MDIO Clause 45 Write Frame Format						
FFFFFFFFh	00	01	AAAAA	RRRRR	10	DDDD.DDDD.DDDD.DDDD

The default or idle state of the two wire serial interface is a logic one. All tri-state drivers should be disabled and the PHY's pull-up resistor should pull the MDIO line to a logic one. Prior to initiating any other transaction, the station management entity shall send a preamble sequence of 32 contiguous logic one bits on the MDIO line with 32 corresponding cycles on MDCLK to provide the PHY with a pattern that it can use to establish synchronization. A PHY shall observe a sequence of 32 contiguous logic one bits on MDIO with 32 corresponding MDCLK cycles before it responds to any other transaction. The MDIO_USER_ADDR0_REG/ MDIO_USER_ADDR1_REG registers must be written before a read or write operation is performed to set the address used in the operation. Each read or write operation has a preceeding address frame.

Preamble
The start of a frame is indicated by a preamble, which consists of a sequence of 32 contiguous bits all of which are a "1". This sequence provides the PHY a pattern to use to establish synchronization. The preamble is required in clause 45 operation.

Start Delimiter
The preamble is followed by the start delimiter which is indicated by a "00" pattern.

Operation Code
The operation code for address is "00". The operation code for a read is "11", while the operation code for a write is a "01".

Ethernet PHY Address
The PHY address is 5 bits allowing 32 unique values. The first bit transmitted is the MSB of the PHY address.

MMD Number

The MMD number is 5 bits allowing 32 unique values. The first bit transmitted is the MSB.

Turnaround
An idle bit time during which no device actively drives the MDIO signal shall be inserted between the **MMD Number** field and the **Data** field of a read frame in order to avoid contention. During a read frame, the PHY shall drive a zero bit onto MDIO for the first bit time following the idle bit and preceding the Data field. During a write frame, this field shall consist of a one bit followed by a zero bit.

Address
The address field is 16-bits on address operations. The first bit transmitted is the MSB of the address word. Each read/write operation initiated has an automatic address operation initiated first that uses the MDIO_USER_ADDR0_REG[15-0] USER_ADDR0 or MDIO_USER_ADDR1_REG[15-0] USER_ADDR1 register values as the 16-bit address.

Data
The Data field is 16 bits on read and write operations. The first bit transmitted and received is the MSB of the data word.

7.3.11.2.3 PRU-ICSS MII MDIO Interractions

The MDIO module will remain idle until enabled by setting the [30] ENABLE bit in the MDIO MDIO_CONTROL_REG register. The MDIO will then continuously poll the link status from within the Generic Status Register of all possible 32 PHY addresses in turn recording the results in the MDIO MDIO_LINK_REG register. Individual PHY's can be enabled or disabled for polling through the associated bit in the MDIO MDIO_POLL_EN_REG register. The MDIO MDIO_LINK_REG and MDIO_ALIVE_REG register bit values are updated on the poll of each PHY. In *Normal Mode*, the link status of two of the 32 possible PHY addresses can also be determined using the MLINK pin inputs. The bit [7] LINKSEL in the MDIO MDIO_USER_PHY_SEL_REG_0/1 register determines the status input that is used. A change in the link status of the two PHYs being monitored will set the appropriate bit ([1-0] LINKINTRAW) in the MDIO MDIO_LINK_INT_RAW_REG register and the MDIO_LINK_INT_MASKED_REG[1-0] LINKINTMASKED register, if enabled by the [6] LINKINT_ENABLE bit in the MDIO MDIO_USER_PHY_SEL_REG_0/1 register. In *State Change Mode*, a change in any PHY status will be indicated on the MDIO_LINK_INT_RAW_REG[0] LINKINTRAW interrupt if enabled.

The MDIO MDIO_ALIVE_REG register is updated by the MDIO module if the PHY acknowledged the read of the generic status register. In addition, any PHY register read transactions initiated by the host also cause the MDIO MDIO_ALIVE_REG register to be updated.

At any time, the host can define a transaction for the MDIO module to undertake using the [15-0] DATA, [20-16] PHYADR, [25-21] REGADR, and [30] WRITE fields in a MDIO_USER_ACCESS_REG_0/1 register. When the host sets the [31] GO bit in this register, the MDIO interface module will begin the transaction without any further intervention from the host. Upon completion, the MDIO will clear the [31] GO bit and set the [1-0] USERINTRAW bit field in the MDIO_USER_INT_RAW_REG register corresponding to the MDIO_USER_ACCESS_REG_0/1 register being used. The corresponding bit in the MDIO_USER_INT_MASKED_REG register may also be set depending on the mask setting in the MDIO_USER_INT_MASK_SET_REG and MDIO_USER_INT_MASK_CLEAR_REG registers. A round-robin arbitration scheme is used to schedule transactions which may queued by the host in different MDIO_USER_ACCESS_REG_0/1 registers. The host should check the status of the [31] GO bit in the MDIO_USER_ACCESS_REG_0/1 register before initiating a new transaction to ensure that the previous transaction has completed. The host can use the [29] ACK bit in the MDIO_USER_ACCESS_REG_0/1 register to determine the status of a read transaction.

Software may use the MDIO module to set up the auto-negotiation parameters of each PHY attached to a MAC port, retrieve the negotiation results, and set up the MAC Control register in the corresponding MAC.

7.3.11.2.4 PRU-ICSS MII MDIO Interrupts

7.3.11.2.4.1 Normal Mode ([30]STATECHANGEMODE = 0h)

The MDIO will assert the MDIO_LINKINT signals if there is a change in the link state of the Ethernet PHY corresponding to the address in the [4-0] PHYADR_MON field of the MDIO MDIO_USER_PHY_SEL_REG_j

(where j = 0 or 1) registers and the corresponding [6] LINKINT_ENABLE bit is set. The MDIO_LINKINT event is also captured in the MDIO MDIO_LINK_INT_MASKED_REG register. MDIO_LINKINT[0] and MDIO_LINKINT[1] correspond to the MDIO MDIO_USER_PHY_SEL_REG_0 and MDIO_USER_PHY_SEL_REG_1 registers, respectively.

When the [31] GO bit in the MDIO_USER_ACCESS_REG_j (where j = 0 or 1) registers transitions from ‘1’ to ‘0’, indicating the completion of a user access, and the corresponding [1-0] USERINTMASKSET field in the

MDIO_USER_INT_MASK_SET_REG register is set, the MDIO_USERINT signal is asserted ‘1’.
The MDIO_USERINT event is also captured in the MDIO_USER_INT_MASKED_REG register.
MDIO_USERINT[0] and MDIO_USERINT[1] correspond to the MDIO_USER_ACCESS_REG_0 and MDIO_USER_ACCESS_REG_1 registers, respectively.

7.3.11.2.4.2 State Change Mode ([30]STATECHANGEMODE = 1h)

In *State Change Mode*, the MDIO will assert MDIO_LINKINT[0] when any bit in the MDIO MDIO_ALIVE_REG or MDIO MDIO_LINK_REG registers changes due to MDIO operations. The MDIO_LINKINT event is also captured in the MDIO MDIO_LINK_INT_MASKED_REG register. MDIO_LINKINT[1] output and the MDIO MDIO_USER_PHY_SEL_REG_j (where j = 0 or 1) registers are unused in *State Change Mode*.

7.3.11.2.5 Manual Mode

Manual Mode allows software to directly control the serial clock output (MDCLK_O), the serial data output enable (MDIO_OE_N), and the serial data output (MDIO_O). The serial data input can also be read (MDIO_I). This mode is enabled when the [31] MANUALMODE bit is set in the MDIO MDIO_POLL_REG register. *Manual Mode* is intended to be used by software for slow speed general purpose IO operations and not for MDIO PHY operations.

7.3.11.3 PRU-ICSS MII MDIO Receive/Transmit Frame Host Software Interface

To facilitate transmission and reception of serial management frames, the host has to perform the following operations:

- Configure the [20] PREAMBLE and [15-0] CLKDIV fields in the MDIO MDIO_CONTROL_REG register.
- Enable the MDIO module by setting the [30] ENABLE bit in the MDIO MDIO_CONTROL_REG register. If Byte access is being used, the [30] ENABLE bit should be written last.
- The MDIO MDIO_ALIVE_REG register can be read after a delay to determine which Ethernet PHYs responded.
- Set up the appropriate PHY addresses in the MDIO MDIO_USER_PHY_SEL_REG_j[4-0] PHYADR_MON bits.
- Set up the appropriate [6] LINKINT_ENABLE bit in the MDIO MDIO_USER_PHY_SEL_REG_j registers.
- Set up the appropriate [7] LINKSEL bits in the MDIO MDIO_USER_PHY_SEL_REG_j registers.
- Set up the appropriate [1-0] USERINTMASKEDSET field in the MDIO MDIO_USER_INT_MASK_SET_REG register.
- To write to an Ethernet PHY register the host should first check to ensure that the [31] GO bit in a MDIO MDIO_USER_ACCESS_REG_j registers is cleared. The [31] GO, [30] WRITE, [25-21] REGADR, [20-16] PHYADR and data fields in that MDIO MDIO_USER_ACCESS_REG_j registers can then be updated to the appropriate value. If byte access is being used, the [31] GO bit should be written last. The write operation to the PHY will be scheduled and completed by the module. Completion of the write operation can be determined by examining the [31] GO bit in the MDIO MDIO_USER_ACCESS_REG_j registers. It also results in a transition on the appropriate MDIO_INT signal and the corresponding bit in the MDIO MDIO_USER_INT_MASKED_REG register based on the setting of the MDIO MDIO_USER_INT_MASK_SET_REG register.
- To read from an Ethernet PHY register the host should first check to ensure that the [31] GO bit in a MDIO MDIO_USER_ACCESS_REG_j registers is cleared. The [31] GO, [25-21] REGADR, and [20-16] PHYADR fields in that MDIO MDIO_USER_ACCESS_REG_j registers can then be updated to the appropriate value. The read data value will be available in the [15-0] DATA field of the MDIO MDIO_USER_ACCESS_REG_j registers after the module completes the read operation on the serial bus. The completion of the read operation can be determined by examining the [31] GO and [29] ACK bits in the MDIO MDIO_USER_ACCESS_REG_j registers. It also results in a transition on the appropriate MDIO_INT

signal and the corresponding bit in the MDIO MDIO_USER_INT_MASKED_REG register based on the setting of the MDIO MDIO_USER_INT_MASK_SET_REG register.

- The module de-asserts the MDIO_USERINT signal when the host writes to the appropriate [1-0] USERINTMASKED bit field in the MDIO MDIO_USER_INT_MASKED_REG register or the [1-0] USERINTRAW bit field in the MDIO MDIO_USER_INT_RAW_REG register.
- The host can poll the MDIO MDIO_LINK_REG register periodically or use the MDIO_LINKINT signals to determine the state of the serial interface to a particular Ethernet PHY.
- The module de-asserts the MDIO_LINKINT when the host writes to the appropriate [1-0] LINKINTRAW bit field in the MDIO_MDIO LINK_INT_RAW_REG register or the [1-0] LINKINTMASKED bit field in the MDIO_MDIO LINK_INT_MASKED_REG register.

Table 7-100. Summary of the PRU-ICSS MII MDIO Functional Registers

Address Offset	Register Mnemonic	Register Name	Register Purpose
00h	**MDIOVer**	MDIO_MDIO_VERSION_REG	Module version register
04h	**MDIOControl**	MDIO_CONTROL_REG	Module control register
08h	**MDIOAlive**	MDIO_ALIVE_REG	Ethernet PHY acknowledge status register
0Ch	**MDIOLink**	MDIO_LINK_REG	Ethernet PHY link status register
10h	**MDIOLinkIntRaw**	MDIO_LINK_INT_RAW_REG	Link status change interrupt register (raw value)
14h	**MDIOLinkIntMasked**	MDIO_LINK_INT_MASKED_REG	Link status change interrupt register (masked value)
18h	**MDIOLinkIntMaskSet**	MDIO_LINK_INT_MASK_SET_REG	Link status change interrupt mask set register
1Ch	**MDIOLinkIntMaskClr**	MDIO_LINK_INT_MASK_CLEAR_REG	Link status change interrupt mask clear register
20h	**MDIOUserIntRaw**	MDIO_USER_INT_RAW_REG	User command complete interrupt register (raw value)
24h	**MDIOUserIntMasked**	MDIO_USER_INT_MASKED_REG	User command complete interrupt register (masked value)
28h	**MDIOUserIntMaskSet**	MDIO_USER_INT_MASK_SET_REG	User interrupt mask set register
2Ch	**MDIOUserIntMaskClr**	MDIO_USER_INT_MASK_CLEAR_REG	User interrupt mask clear register
30h	**MDIOManual_IF**	MDIO_MANUAL_IF_REG	Manual interface register
34h	**MDIOPoll_IPG**	MDIO_POLL_REG	Poll and IPG register
38h	**MDIOPoll_En**	MDIO_POLL_EN_REG	Poll Enable register
3Ch	**MDIOClause**	MDIO_CLAUS45_REG	Clause 22 or 45 enable register
40h	**MDIOUser_Addr0**	MDIO_USER_ADDR0_REG	User Address register 0
44h	**MDIOUser_Addr1**	MDIO_USER_ADDR1_REG	User Address register 1
48h – 7Ch	**Reserved**	-	Reserved
80h	**MDIOUserAccess0**	MDIO_USER_ACCESS_REG_0	User access register 0
84h	**MDIOUserPhySel0**	MDIO_USER_PHY_SEL_REG_0	User PHY select register 0
88h	**MDIOUserAccess1**	MDIO_USER_ACCESS_REG_1	User access register 1

Table 7-100. Summary of the PRU-ICSS MII MDIO Functional Registers (continued)

Address Offset	Register Mnemonic	Register Name	Register Purpose
8Ch	**MDIOUserPhySel1**	MDIO_USER_PHY_SEL_REG_1	User PHY select register 1
90h – FFh	**Reserved**	-	Reserved

7.3.12 PRU-ICSS IEP

This section describes the Industrial Ethernet Peripheral (IEP) Module which is part of the PRU-ICSS.

7.3.12.1 PRU-ICSS IEP Overview

The Industrial Ethernet Peripheral (IEP) performs hardware work required for Industrial Ethernet functions. The IEP module features an industrial ethernet timer with 16 compare events, industrial ethernet sync generator and latch capture, industrial ethernet watchdog timer, and a digital I/O port (DIGIO).

7.3.12.2 PRU-ICSS IEP Functional Description

This section provides the functional description of the IEP component. The PRU-ICSS module implements one Industrial Ethernet Peripheral (IEP0). The IEP functional block diagram is shown in Figure 7-88.

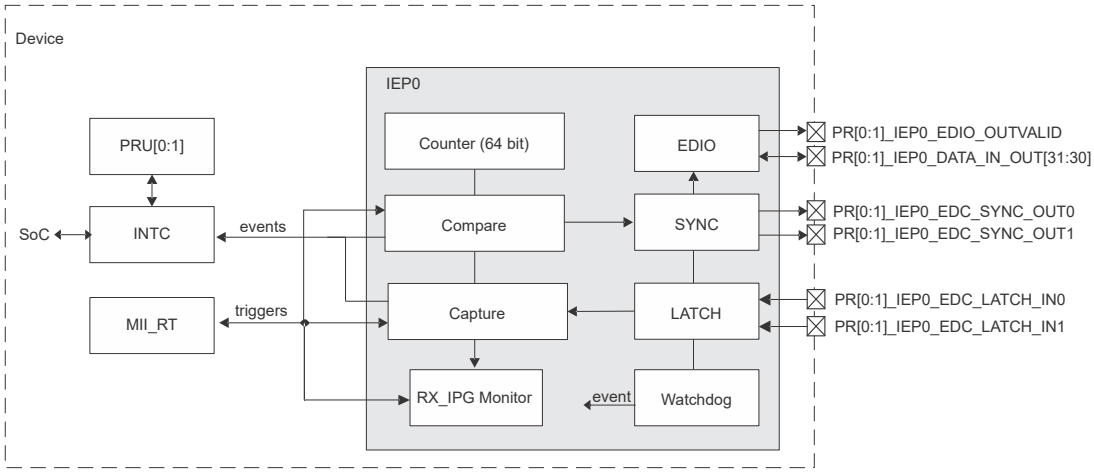


Figure 7-88. IEP Functional Block Diagram

7.3.12.2.1 PRU-ICSS IEP Clock Generation

The IEP has a selectable module input clock (PRU_ICSS_IEP_CLK, see also *PRU-ICSS in Module Integration*). The clock source is selected by the state of the CTRLMMR_PRU_ICSS_CLKSEL[19-16] IEP_CLKSEL bit within the CTRL_MMR0 register space. Two clock sources are supported for the IEP input clock:

- PRU_ICSS_IEP_CLK (where n = 0 or 1): The source clock for IEP module can be selected through IEP Clock Multiplexer (see also *PRU-ICSS in Module Integration*). The default functional source clock for IEP is MAIN_PLL3_HSDIV1_CLKOUT, derived from PLL3 HSDIV1. The IEP functional clock (PRU_ICSS_IEP_CLK) runs at 200 or 250 MHz.
- PRU_ICSS_ICLK (where n = 0 or 1): The PRU-ICSS interface clock is derived as divided version of the device PLLCTRL output clock (SYSCLK0/2).

Switching from PRU_ICSS_IEP_CLK to PRU_ICSS_ICLK is done by writing 1h to the PRU_ICSS_IEPCLK_REG/PRU_ICSS0_IEPCLK_REG[0] IEP_OCP_CLK_EN bit. This is a one time configuration step before enabling the IEP function. Switching back from PRU_ICSS_ICLK to PRU_ICSS_IEP_CLK is only supported through a hardware reset of the PRU-ICSS.

CAUTION

When software enables the clock (at PRU-ICSS level) to the IEP module clock input via setting bit PRU_ICSS_IEPCLK_REG/PRU_ICSS0_IEPCLK_REG[0] IEP_OCP_CLK_EN to 1h in the PRUSS_CFG space, there must be NO in-flight transactions to the IEP block.

CAUTION

Switching from PRU_ICSS_IEP_CLK (the IEP specific functional clock source) to the PRU_ICSS_CORE_CLK source is supported ONLY in software. Switching back from PRU_ICSS_CORE_CLK to PRU_ICSS_IEP_CLK is ONLY supported via assertion of a hardware reset to the PRU-ICSS.

7.3.12.2.2 PRU-ICSS IEP Timer

The IEP timer is a simple 64-bit timer. This timer is intended for use by industrial ethernet functions but can also be leveraged as a generic timer in other applications.

7.3.12.2.2.1 PRU-ICSS IEP Timer Features

The IEP timer supports the following features:

- One controller 64-bit count-up counter with an overflow status bit.
  - Runs on ICSS_IEP_CLK or ICSS_ICLK clock.
  - Write 1h to clear status.
  - Supports a programmable increment value from 1 to 16 (default 5).
  - An optional compensation method allows the increment value to apply compensation increment value from 1 to 16 count up to 2^{24} ICSS_IEP_CLK events with additional slow compensation mode.
- 10× 64-bit capture registers:
  - 8 capture inputs, with optional synchronous or asynchronous mode:
    - 6× rise capture registers: ICSS_IEP_CAPRi_REG0/ IEP_CAPRi_REG1 (where i=0 to 5)
    - 2× rise and fall capture registers: IEP_CAPR6_REG0/ IEP_CAPR6_REG1 and IEP_CAPR7_REG0/ IEP_CAPR7_REG1, each combined with a fall capture - IEP_CAPF6_REG0/ IEP_CAPF6_REG1 and IEP_CAPF7_REG0/ IEP_CAPF7_REG1, respectively
    - One global event (any capture event) output for interrupt
- 16× 64-bit compare registers: IEP_CMPj_REG0/ IEP_CMPj_REG1 (where j = 0 to 15) and IEP_CMP_STATUS_REG[15-0] CMP_STATUS
  - 16 status bits, write 1h to clear
  - 16 individual event outputs
  - One global event output for interrupt generation triggered by any compare event
- 32 outputs, one high-level and one high-pulse for each compare hit event
- IEP_CMP_CFG_REG[0] CMP0_RST_CNT_EN, if enabled, resets the controller counter on the next ICSS_IEP_CLK/ ICSS_ICLK cycle
- Controller counter reset-state is programmable
- Optional 32-bit shadow mode of operation, which can be configured through IEP_CMP_CFG_REG[17] SHADOW_EN bit

7.3.12.2.3 32-Bit Shadow Mode

The IEP module can be configured in 32-bit shadow mode when IEP_CMP_CFG_REG[17] SHADOW_EN bit is set to 1h (default value is 0h, e.g. 64-bit mode of operation is enabled). In this mode, the controller counter will be in 32-bit mode of operation. This enables the shadow copy functionality of the compare registers.

Rules of operation:

1. Switching the state of the controller counter from 32-bit Shadow mode to 64-bit mode of operation, the counter should be disabled and bit IEP_CMP_CFG_REG[17] SHADOW_EN should be cleared to 0h (default value).
2. A new compare update (IEP_CMP_CFG_REG[16-1] CMP_EN = 1h - enables CMP[0:15] event, where [0]CMP_EN maps to CMP0) should be set 4 cycle counts before the next rollover or reset of the counter and to insure the correct shadow copy to active update is correct.

Sequence of operation:

1. Disable counter trough IEP_GLOBAL_CFG_REG[0] CNT_ENABLE = 0h (default value).
2. Clear controller counter through IEP_CMP_CFG_REG[17] SHADOW_EN = 0h (64-bit mode of operation)
3. Enable 32-bit Shadow mode through IEP_CMP_CFG_REG[17] SHADOW_EN bit (value: 1h)

4. Program IEP_CMPm_REGn (where m = 0 to 15 and n = 0 to 1); use the upper 32-bits (IEP_CMP0_REG1[31-0] CMP0_1) of the 64-bit CMP
   a. The lower 32-bits are the active compare value (IEP_CMPm_REG0[31-0] CMPm_0, where m = 0 to 15), software can only read this bits
   b. The upper 32-bits are the shadow copy (IEP_CMPm_REG1[31-0] CMPm_1, where m = 0 to 15), software can write and read this bits
5. Enable counter trough IEP_GLOBAL_CFG_REG[0] CNT_ENABLE = 1h

After the counter is enabled, then software can load a new set of CMP[0:15] without affecting the current active values of (IEP_CMPm_REG0[31-0] CMPm_0, where m = 0 to 15). Only when the counter is reset to 32-bit Shadow mode (IEP_CMP_CFG_REG[17] SHADOW_EN = 1h), it will load the shadow copy of IEP_CMPm_REG1[31-0] CMPm_1 into local copy.

Shadow compare value (IEP_CMPm_REG1[31-0] CMPm_1) is loaded into active register IEP_CMPm_REG0[31-0] CMPm_0 when the controller counter is configured in 32-bit Shadow mode. If IEP_CMP_CFG_REG[0] CMP0_RST_CNT_EN bit is enabled (value 1h) to reset the counter, the next reset event will be defined by the last CMP0 update.

7.3.12.2.4 PRU-ICSS IEP Timer Basic Programming Sequence

Follow these basic steps to configure the IEP Timer.

Counter maintains/function:

1. Once enabled, the counter will count every PRU_ICSS_IEP_CLK cycle, default rate of 200 MHz.
2. It is a free running counter with a sticky over flag status bit.
3. The counter over flow flag (IEP_GLOBAL_STATUS_REG[0] CNT_OVF) will get set when the counter switches/rollsover from 0xFFFF_FFFF to 0x0000_0000.
4. The counter will continue to count up. The software will need to read/clear the counter over flow flag and increment the MSB in software variable.

Compare function:

1. Initialize timer to known state (default values)
   - Disable counter (IEP_GLOBAL_CFG_REG[0] CNT_ENABLE = 0)
   - Reset Count Register (IEP_COUNT_REG0, IEP_COUNT_REG1) by writing FFFFFFFFh to clear
   - Clear overflow status register (IEP_GLOBAL_STATUS_REG[0] CNT_OVF = 1)
   - Clear compare status (IEP_CMP_STATUS_REG[15-0] CMP_STATUS) by writing FFFFFFFFh to clear
2. Set compare values IEP_CMPj_REG0, IEP_CMPj_REG1 (where j = 0 to 15)
3. Enable compare events (IEP_CMP_CFG_REG[16-1] CMP_EN = 1)
4. Set increment value (IEP_GLOBAL_CFG_REG[7-4] DEFAULT_INC)
5. Set compensation value (IEP_COMPEN_REG[22-0] COMPEN_CNT)
6. Enable counter (IEP_GLOBAL_CFG_REG[0] CNT_ENABLE = 1)

Capture function:

1. Update/Enable the counter if required
2. Program the enable the desired capture event
3. Wait for global capture event
4. Read/Clear the capture status to determine which capture event occurred
5. Read the capture count

7.3.12.2.5 Industrial Ethernet Mapping

Some of the capture inputs and compare registers are mapped to specific industrial Ethernet functions in hardware, shown in Table 7-101. All capture inputs are mapped to industrial Ethernet functions, and these inputs are not available for any other application. The CMP1 and CMP2 compare registers also function as the start time triggers for SYNC0 and SYNC1, respectively.

Table 7-101. IEP Timer Mode Mapping

Capture Input	IEP Line/Function
IEP_CAPR0_REG0/ IEP_CAPR0_REG1, rise only	If EXT_CAP_EN[0] = 0h (Internal source is selected)/ PRU0_RX_SOF If EXT_CAP_EN[0] = 1h (External source is selected)/ ICSS_IEP_CAP_INTR_REQ0
IEP_CAPR1_REG0/ IEP_CAPR1_REG1, rise only	If EXT_CAP_EN[1] = 0h (Internal source is selected)/ PRU0_RX_SFD If EXT_CAP_EN[1] = 1h (External source is selected)/ ICSS_IEP0_CAP_INTR_REQ1
IEP_CAPR2_REG0/ IEP_CAPR2_REG1, rise only	If EXT_CAP_EN[2] = 0h (Internal source is selected)/ PRU1_RX_SOF If EXT_CAP_EN[2] = 1h (External source is selected)/ ICSS_IEP_CAP_INTR_REQ2
IEP_CAPR3_REG0/ IEP_CAPR3_REG1, rise only	If EXT_CAP_EN[3] = 0h (Internal source is selected)/ PRU1_RX_SFD If EXT_CAP_EN[3] = 1h (External source is selected)/ ICSS_IEP_CAP_INTR_REQ3
IEP_CAPR4_REG0/ IEP_CAPR4_REG1, rise only	If EXT_CAP_EN[4] = 0h (Internal source is selected)/ PORT0_TX_SOF; For MII mode uses loopback for lower jitter 40ns versus 4ns. If EXT_CAP_EN[4] = 1h (External source is selected)/ ICSS_IEP_CAP_INTR_REQ4
IEP_CAPR5_REG0/ IEP_CAPR5_REG1, rise only	If EXT_CAP_EN[5] = 0h (Internal source is selected)/ PORT1_TX_SOF For MII mode uses loopback for lower jitter 40ns versus 4ns. If EXT_CAP_EN[5] = 1h (External source is selected)/ ICSS_IEP_CAP_INTR_REQ5
IEP_CAPR6_REG0/ IEP_CAPR6_REG1 - rise and IEP_CAPF6_REG0/ IEP_CAPF6_REG1 - fall	PR_IEP_EDC_LATCH_IN0 (IO inputs at SoC level)
IEP_CAPR7_REG0/ IEP_CAPR7_REG1 - rise and IEP_CAPF7_REG0/ IEP_CAPF7_REG1 - fall	PR_IEP_EDC_LATCH_IN1 (IO inputs at SoC level)
IEP_CMP1_REG0/ IEP_CMP1_REG1	For SYNC0 trigger of start time
IEP_CMP2_REG0/ IEP_CMP2_REG1	For SYNC1 trigger of start time; only valid in the SYNC2 independent mode
IEP_CMP3_REG0/ IEP_CMP3_REG1	For MII TX0 start trigger, if MII register MII_RT_TXCFG0/1[2] TX_EN_MODEn is enabled (where n = 0 or 1).
IEP_CMP4_REG0/ IEP_CMP4_REG1	For MII TX1 start trigger, if MII register MII_RT_TXCFG0/1[2] TX_EN_MODEn is enabled (where n = 0 or 1).

7.3.12.2.6 PRU-ICSS IEP Sync0/Sync1 Module

The industrial ethernet sync block supports the generation of two synchronization signals: SYNC0 and SYNC1. SYNC0 and SYNC1 can be directly mapped to output signals (pr<k>_iep<n>_edc_sync_out0 and pr<k>_iep<n>_edc_sync_out1) for external devices to use. They can also be used for internal synchronization within the PRU-ICSS. These signals are also mapped as system events and can therefore be mapped to the Arm core's Host interrupts.

7.3.12.2.6.1 PRU-ICSS IEP Sync0/Sync1 Features

The industrial ethernet sync block supports the following features:

- Two synchronize generation signals (SYNC0, SYNC1)
  - Activation time synchronized with IEP Timer
- IEP_CMP1_REG0/ IEP_CMP1_REG1 triggers SYNC0 activation time
- IEP_CMP2_REG0/ IEP_CMP2_REG1 triggers SYNC1 activation time (only valid in the SYNC2 independent mode)
  - Pulse width defined by registers or acknowledge mode (remain asserted until software acknowledged)
  - Cyclic or single-shot operation
  - Option to enable or disable sync generation
- Programmable number of clock cycles between the start of SYNC0 to the start of SYNC1

7.3.12.2.6.2 PRU-ICSS IEP Sync0/Sync1 Generation Modes

There are four modes of operation for the sync signals: cyclic mode, single shot mode, cyclic with acknowledge mode, and single shot with acknowledge mode. Figure 7-89 shows examples of these modes. The start time is set by the IEP_SYNC_START_REG[31-0] SYNC_START bit field. The cycle time is configured by the IEP_SYNC0_PERIOD_REG[31-0] SYNC0_PERIOD bit field. The pulse length is defined by IEP_SYNC_PWIDTH_REG[31-0] SYNC_HPW bit field.

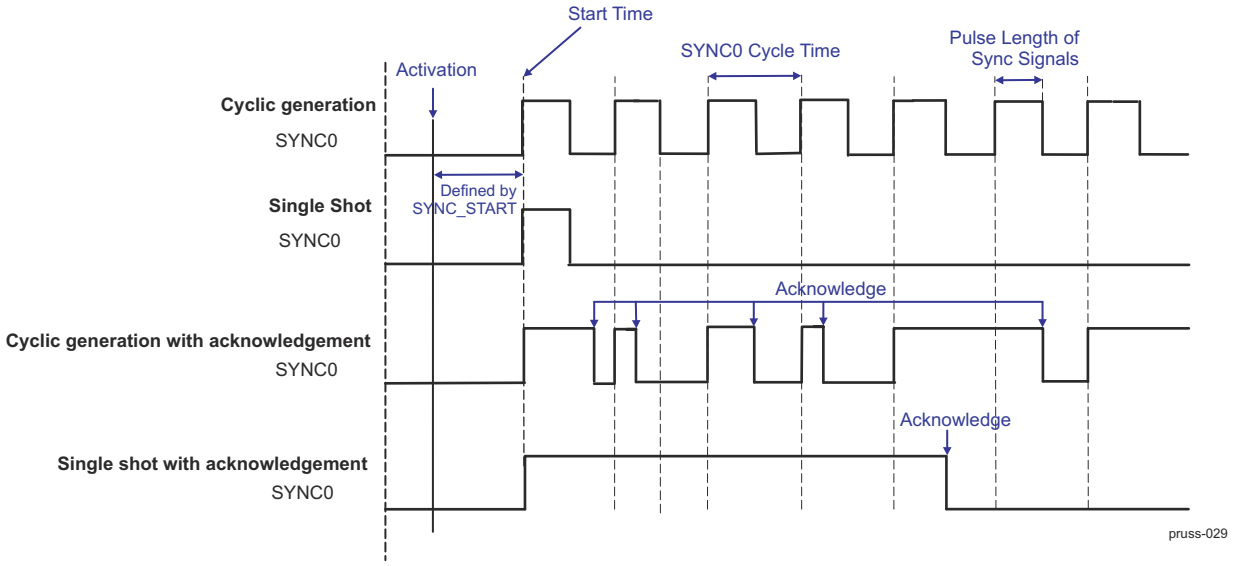


Figure 7-89. PRU-ICSS IEP SYNC0 Signal Generation Modes

In SYNC1 dependent mode (IEP_SYNC_CTRL_REG[8] SYNC1_IND_EN = 0h), SYNC1 depends on SYNC0 and the start time of the SYNC1 can be defined by the IEP_SYNC1_DELAY_REG register. Figure 7-90 shows different examples when changing the value in the IEP_SYNC1_DELAY_REG register. Note: If the SYNC1 delay time is 0, SYNC1 reflects SYNC0. Cyclic generation cannot be used for network time synchronized applications because only the CMP1/CMP2 hit occurs in the compensated time domain.

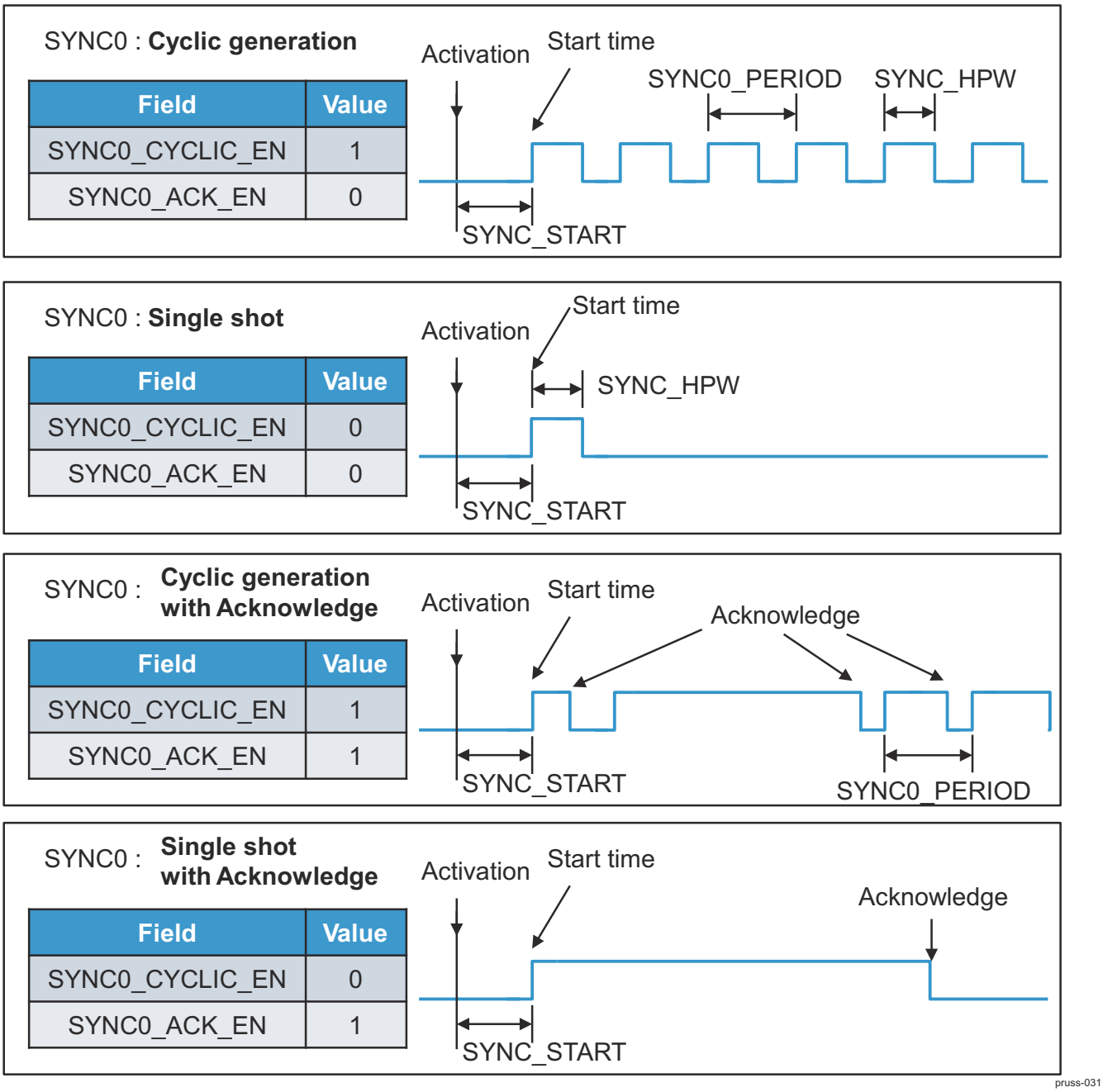


Figure 7-90. Examples of the Dependent Mode of SYNC1

7.3.12.2.7 PRU-ICSS IEP WatchDog

In industrial ethernet applications, the watchdog timer (WD) is used to monitor process data communication and to turn off the outputs of the digital input/output (DIGIO) functional block after a set time. The WD will thereby protect the system from errors or faults by timeout or expiration. The expiration is used to initiate corrective action in order to keep the system in a safe state and restore normal operation based on configuration. Therefore, if the system is stable, the watchdog timer should be regularly reset or cleared to avoid timeout or expiration.

The IEP watchdog timer supports the following features:

- One 16-bit pre-divider for generating a WD clock (default 100µs) based on PRU_ICSS_IEP_CLK input
- Two 16-bit Watchdog Timers:
  - PDI_WD for Sync Managers WD, used in conjunction with digital input/output (DIGIO)

– PD_WD for data link layer user WD, used in conjunction with data link layer or application layer interface actions

Note

For more details on the PRU-ICSS Industrial Ethernet Watchdog timer, refer also to the Watchdog timer register descriptions covered in the PRU_ICSS_IEP chapter of the Register Addendum.

7.3.12.2.8 PRU-ICSS IEP DIGIO

The IEP digital I/O (DIGIO) block provides dedicated I/Os for industrial ethernet protocols. The digital inputs can be sampled when specific events occur or continuously as a raw input. Likewise, driving the digital outputs can be triggered by specific events or controlled by software. The timing, delay cycle clocks, data sources, and data valid of the digital input and outputs are controlled by the IEP_DIGIO_CTRL_REG and IEP_DIGIO_EXP_REG registers. Additionally, the IEP DIGIO block can be used as generic I/Os in other applications.

7.3.12.2.8.1 PRU-ICSS IEP DIGIO Features

The IEP digital I/O supports the following features:

- Digital data output:
  - 4 channels (PR<k>_IEP0_EDIO_DATA_IN_OUT[31:28])
  - Five event options for driving output data output:
    - End of frame event (PRU0/1_RX_EOF)
    - SYNC0 events
    - SYNC1 events
    - Watchdog trigger
    - Software enable
- Digital data out enable (optional tri-state control)
- Digital data input:
  - 4 channels (PR<k>_IEP0_EDIO_DATA_IN_OUT[31:28])
  - IEP_DIGIO_DATA_IN_RAW_REG supports direct sampling of PR<k>_IEP0_EDIO_DATA_IN_OUT[31:28]
  - IEP_DIGIO_DATA_IN_REG supports four event options to trigger sampling of PR<k>_IEP0_EDIO_DATA_IN_OUT[31:28]:
    - Start of frame event in start of frame (SOF) mode
    - pr<k>_iep<n>_edc_latch_in<0/1> event
    - SYNC0 events: pr<k>_iep<n>_edc_sync_out0
    - SYNC1 events: pr<k>_iep<n>_edc_sync_out1

The industrial digital I/Os supported by the PRU-ICSS IEP peripheral are described in Table 7-102.

Table 7-102. PRU-ICSS IEP Digital IOs

Direction	Port	Mapped to Device I/Os	Notes
output	PR<k>_IEP0_EDIO_DATA_IN_OUT[0:31]	PR0_IEP0_EDIO_DATA_IN_OUT[31:28]	Only PR<k>_IEP0_EDIO_DATA_IN_OUT[31:28] are exported to device pins as a bidirectional.
output	PR<k>_EDIO_DATA_OUT_EN[0:31]	No	Optional tri-state control for DATA_OUT
output	PR<k>_IEP0_EDIO_OUTVALID	PR0_IEP0_EDIO_OUTVALID	Will pulse even same data
output	PR<k>_EDIO_SOF	No	PRU<0/1>_RX_SOF defined by IEP_DIGIO_EXP_REG[12] SOF_SEL
input	PR<k>_EDIO_OE_EXT	No	
output	PR<k>_EDIO_WD_TRIG	No	Just export of IEP_WD_STATUS_REG[0] PD_WD_STAT
output	PR<k>_EDIO_DATA_ENA	No	Reserved. Driven low.
input	PR<k>_IEP<n>_EDC_LATCH_IN0	PR0_IEP<n>_EDC_LATCH_IN0	
input	PR<k>_IEP<n>_EDC_LATCH_IN1	PR0_IEP<n>_EDC_LATCH_IN1	
output	PR<k>_IEP<n>_EDC_SYNC_OUT0	PR0_IEP<n>_EDC_SYNC_OUT0	

Table 7-102. PRU-ICSS IEP Digital IOs (continued)

Direction	Port	Mapped to Device I/Os	Notes
output	PR<k>_IEP<n>_EDC_SYNC_OUT1	PR0_IEP<n>_EDC_SYNC_OUT1	

7.3.12.2.8.2 PRU-ICSS IEP DIGIO Block Diagrams

Figure 7-91 shows the signals and registers for capturing the DIGIO data in. Note that bit field [5-4]IN_MODE in the IEP_DIGIO_CTRL_REG register must be set to 1h for data to be latched on the external PR<k>_IEP<n>_EDC_LATCH_IN0 signal. In PRU0/1_RX_SOF mode, the delay time of capturing PR<k>_IEP0_EDIO_DATA_IN_OUT[31:28] is programmable through the [11-8]SOF_DLY bit of the IEP_DIGIO_EXP_REG register.

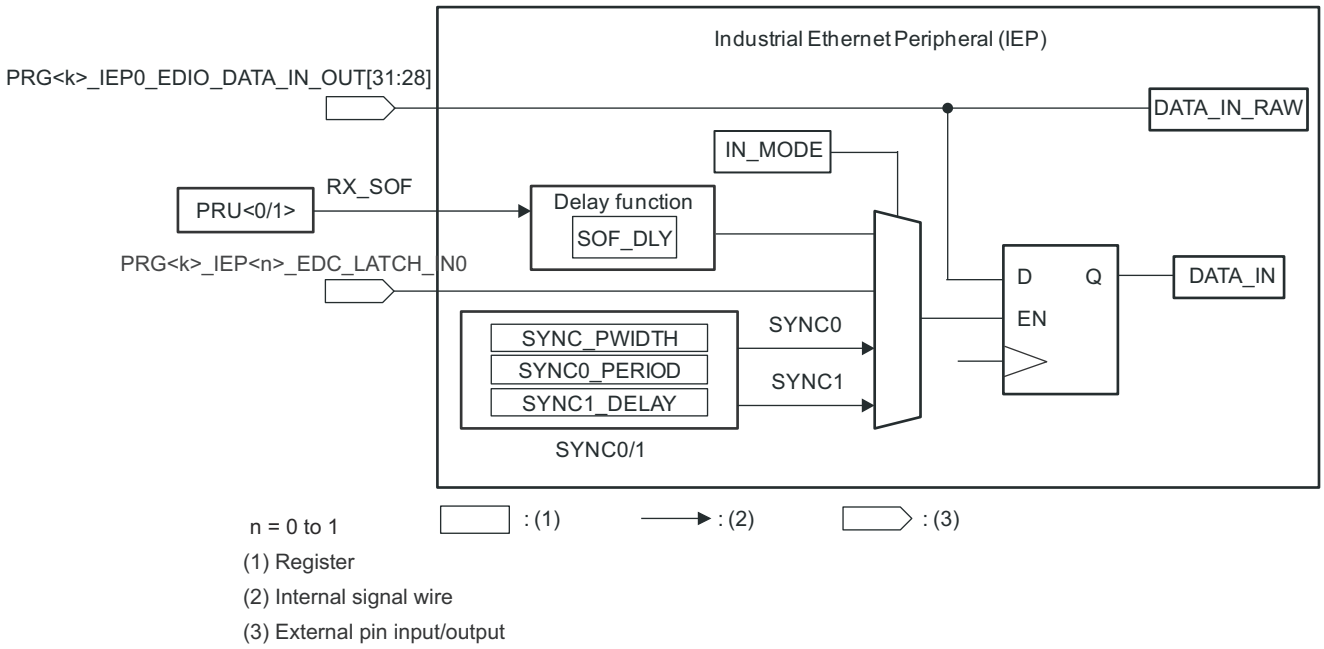
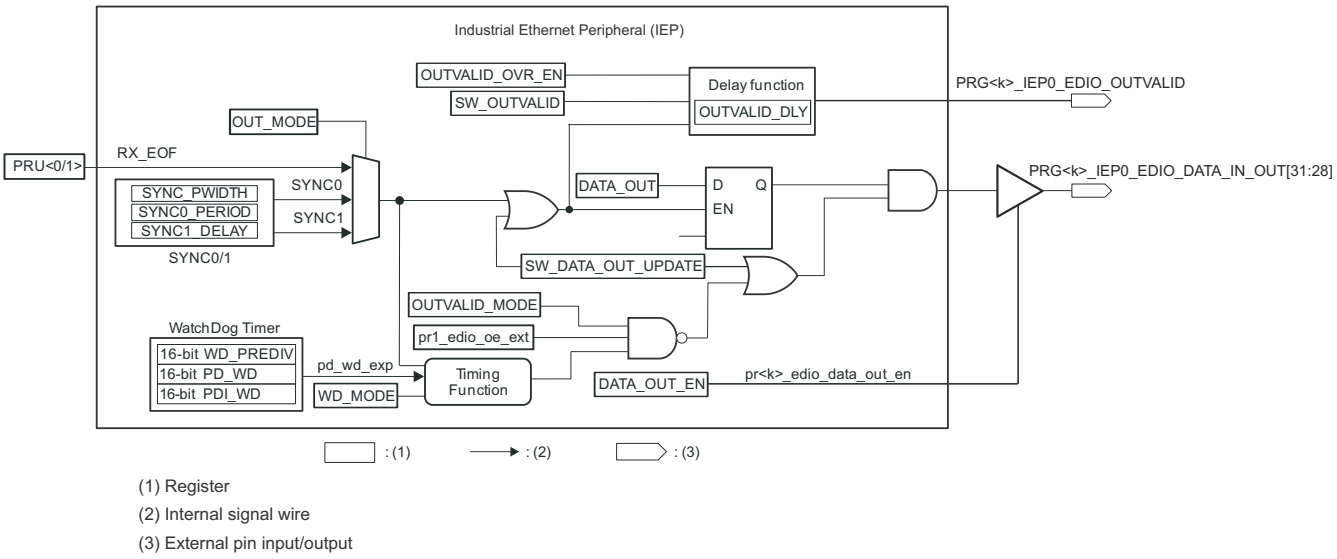


Figure 7-91. IEP DIGIO Data In

Figure 7-92 shows the signals and registers for driving the DIGIO data out.

The PR<k>_IEP0_EDIO_DATA_IN_OUT[31:28] is immediately forced to zero when IEP_DIGIO_CTRL_REG[1] OUTVALID_MODE = 1h, pr1_edio_oe_ext = 1h, and pd_wd_exp = 1h, or the next update hardware post pd_wd_exp. Delay assertion of PR<k>_IEP0_EDIO_OUTVALID from PR<k>_IEP0_EDIO_DATA_IN_OUT[31:28] update events are controlled by software through IEP_DIGIO_EXP_REG[2] SW_OUTVALID.



(1) Register
(2) Internal signal wire
(3) External pin input/output

Figure 7-92. IEP DIGIO Data Out

7.3.12.2.8.3 PRU-ICSS IEP Basic Programming Model

Follow these steps to configure and read the DIGIO Data Input:

1. Read IEP_DIGIO_DATA_IN_RAW_REG for raw input data

or

1. Enable sampling of PR<k>_IEP0_EDIO_DATA_IN_OUT[31:28] by setting IEP_DIGIO_CTRL_REG[5-4] IN_MODE = 1h.
2. Read IEP_DIGIO_DATA_IN_REG for data sampled upon PR<k>_IEP<n>_EDC_LATCH_IN0 posedge

Follow these steps to configure and write to the DIGIO Data Output:

1. Pre-configure DIGIO by setting IEP_DIGIO_EXP_REG[1] OUTVALID_OVR_EN and IEP_DIGIO_EXP_REG[0] SW_DATA_OUT_UP
2. Write to IEP_DIGIO_DATA_OUT_REG to configure output data.
3. To HiZ output, set corresponding IEP_DIGIO_DATA_OUT_EN_REG[31-0] DATA_OUT_EN bits to 1h (clear to 0h to drive value stored in IEP_DIGIO_DATA_OUT_REG).

7.4 Hardware Security Module (HSM)

This chapter describes the Hardware Security Module (HSM) in the device.

The HSM module is responsible for booting up of the device, enabling the main R5FSS core, defining/controlling overall security of the device based on boot options provided. It also has a DTHE (Data Transform and Hashing Engine) which is a wrapper around crypto IP with some additional capability including CRC and Checksum.

Note

The TRM provides a high-level overview of the HSM. For details on specific modules and the HSM Register Map, please request access to the HSM Addendum.

Table 7-103 provides a list of abbreviations related to hardware security.

Table 7-103. Abbreviations

Abbreviation	Description
AES	Advanced Encryption Standard
DRBG	Deterministic random bit generator
ECC	Elliptic curve cryptography
HMAC	Keyed-hashing for message authentication
ISC	Initiator-side security control
PKA	Public key cryptography
RSA	Rivest–Shamir–Adleman cryptosystem
SHA	Secure hash algorithm
TRNG	True random number generator

7.5 Real-time Control Subsystem (CONTROLSS)

The integrated real-time Control Subsystem (CONTROLSS) enables closed loop control systems with flexible interconnection between data acquisition, actuator modules, and other control signal resources. The CONTROLSS module consists of the following control peripherals:

Analog Control Peripherals

- 5x Analog to Digital Converter (ADC) modules
  - 12-bit resolution with 4MSPS sample rate
  - Programmable 6x single-ended or 3x differential channels
  - 3.2V full scale voltage range with 1.8V reference (32/18 internal input scaling)
  - Support for internal or external 1.8V ADC VREF reference voltage (2% internal reference accuracy error)
  - Two common external calibration pins
  - 4x Post-processing blocks per ADC
  - Multiple ADC trigger sources including CPU timers, GPIO/Input XBAR, and EPWM SOCa/SOCb signals.
- 1x Resolver with 2x dedicated SAR ADCs configurable in the following modes:
  - 2x motor position sensing units
  - 2x General Purpose ADCs with 4x channels, 12-bit resolution with 3MSPS sample rate
- 1x Buffered Digital to Analog (DAC) module
  - 12-bit resolution
- 10x Comparator Subsystem A (CMPSSA)
  - 2 comparators + 2 DACs
  - Window comparison on one input OR
  - Compare two inputs OR
  - Single threshold compare of single input
- 10x Comparator Subsystem B (CMPSSB)
  - 2 comparators + 2 DACs
  - Window comparison on one input OR
  - Single threshold compare of single input

Digital Control Peripherals

- 32x Enhanced Pulse-width Modulation (EPWM) modules
- 16x Enhanced Capture (ECAP) modules
- 2x Sigma-Delta Filter (SDFM) modules
- 3x Enhanced Quadrature Encoder Pulse (EQEP) modules
- 4x Fast Serial Interface Transmitter (FSITX) modules
- 4x Fast Serial Interface Receiver (FSIRX) modules

7.5.1 Real-time Control Subsystem (CONTROLSS) Overview

The AM263Px Real-time Control subsystem or CONTROLSS enables closed loop control systems with flexible interconnection between data acquisition, actuator modules, and other control signal resources.

A real-time control system is typically composed of four main elements:

- *Sensing*: or feedback acquisition. The application needs to measure several key parameters (voltage, current, motor speed, temperature) in an accurate manner and at a very precise moment in time.
- *Processing*: Use the sensing information to apply control algorithms to the incoming data and calculate the next output command.
- *Control*: The command is applied to the system, typically via a PWM unit driving the power electronics system, for example, the motor turns faster, the current to the solar installed system is reduced, the car is accelerating.
- *Interface*: The ability of the device to communicate to other external components. While not necessarily involved in the control of the system, communications to other system components also has to co-exist with the main control loop.

The CONTROLSS consists of various control peripherals to enable full integration the **Sensing** and **Control** functionality of the device within real-time applications. The components of the subsystem are described in the following sections.

Note

In regards to various tables, diagrams, and descriptions throughout this chapter in the AM263Px device TRM.

References to the *C28x* component/functional block (also referred to as *CPU* or *processor core*) is synonymous with the Arm Cortex®-R5F MCU subsystem (*R5FSS*) cores.

References to SYSCLK are synonymous with the CONTROLSS_PLL/2 (200 MHz) source clock.

References to the Peripheral Interrupt Expansion (*PIE*) unit component/functional block is synonymous with the Vectored Interrupt Manager (*VIM*).

References to the Control Logic Accelerator (*CLA*) and Configurable Logic Block (*CLB*) component/functional block are not applicable to this device and can be ignored.

7.5.2 Analog-to-Digital Converter (ADC)

The analog-to-digital converter (ADC) module described in this chapter is a Type 4 ADC.

7.5.2.1 Introduction

The ADC module is a 12-bit successive approximation (SAR) style ADC . The ADC is composed of a core and a wrapper. The core is composed of the analog circuits which include the channel select MUX, the sample-and-hold (S/H) circuit, the successive approximation circuits, voltage reference circuits, and other analog support circuits. The wrapper is composed of the digital circuits that configure and control the ADC. These circuits include the logic for programmable conversions, result registers, interfaces to analog circuits, interfaces to the peripheral buses, post-processing circuits, and interfaces to other on-chip modules.

Each ADC module consists of a single sample-and-hold (S/H) circuit. The ADC module is designed to be duplicated multiple times on the same chip, allowing simultaneous sampling or independent operation of multiple ADCs. The ADC wrapper is start-of-conversion (SOC) based (see Section 7.5.2.4).

7.5.2.1.1 Features

Each ADC has the following features:

- 3.2V default voltage with support to 3.3V depending on ADC modes. (See *Signal Mode* and *ADC Modes of Operation*).
- External reference set by VREFHI and VREFLO pins
- Single-ended signal conversions
- Input multiplexer with up to 6 channels
- External channel mux option to expand available ADC channels
- 16 configurable SOCs
- Type 4 digital wrappers that enhances the ADC capabilities to include:
  - Over and Under Sampling
  - External Channel support with at least 2 bit external select per ADC
- 16 individually addressable result registers
- Two trigger repeater modules, enabling customizable hardware oversampling and undersampling modes with little or no CPU overhead
- Multiple trigger sources
  - S/W (with available global synchronization for multiple ADCs) - software immediate start
  - All ePWMs - ADCSOC A or B
  - GPIO XINT5
  - RTI Timers 0/1/2/3/4/5/6/7
  - ADCINT1/2
  - Input XBAR
  - ECAP events in capture mode (CEVT1, CEVT2, CEVT3, and CEVT4) and APWM mode (period match, compare match, or both)
- Four flexible VIM interrupts
- Burst mode
- Four post-processing blocks, each with:
  - Saturating offset calibration
  - Error from set-point calculation
  - High, low, and zero-crossing compare, with interrupt and ePWM trip capability
  - Trigger-to-sample delay capture
  - Connections to 12 Simultaneous Compare Blocks (also known as ADC Safety Tiles)
  - Aggregation functions: max, min, sum, and average (binary shift)
  - Absolute value function
- Result safety checkers to compare SOC results on same ADC or multiple ADC instances

Additionally, there are 2 Resolvers ADCs (SAR ADCs also known as ADC_R0 and ADC_R1) that can be used as general-purpose ADC. The SAR ADC_Rs have the additional features:

- Maximum sampling rate of 3.125 Msps
- 50-MHz clock to Resolver
- Additional SAR ADC details can be found in the *Resolver Features* and *Resolver Integration* sections.

7.5.2.2 ADC Integration

There are 5x Analog-to-Digital Converter (ADC) modules integrated in the device.

Note

For each ADC[0:4]:

- Analog input channels ADCIN[0:5] have dedicated pins.
- Analog input channels ADCIN[6:7] are tied to shared ADC_CAL[0:1] pins, respectively.
- Refer to *ADC Triggers* for the full list of ADC sample trigger options.

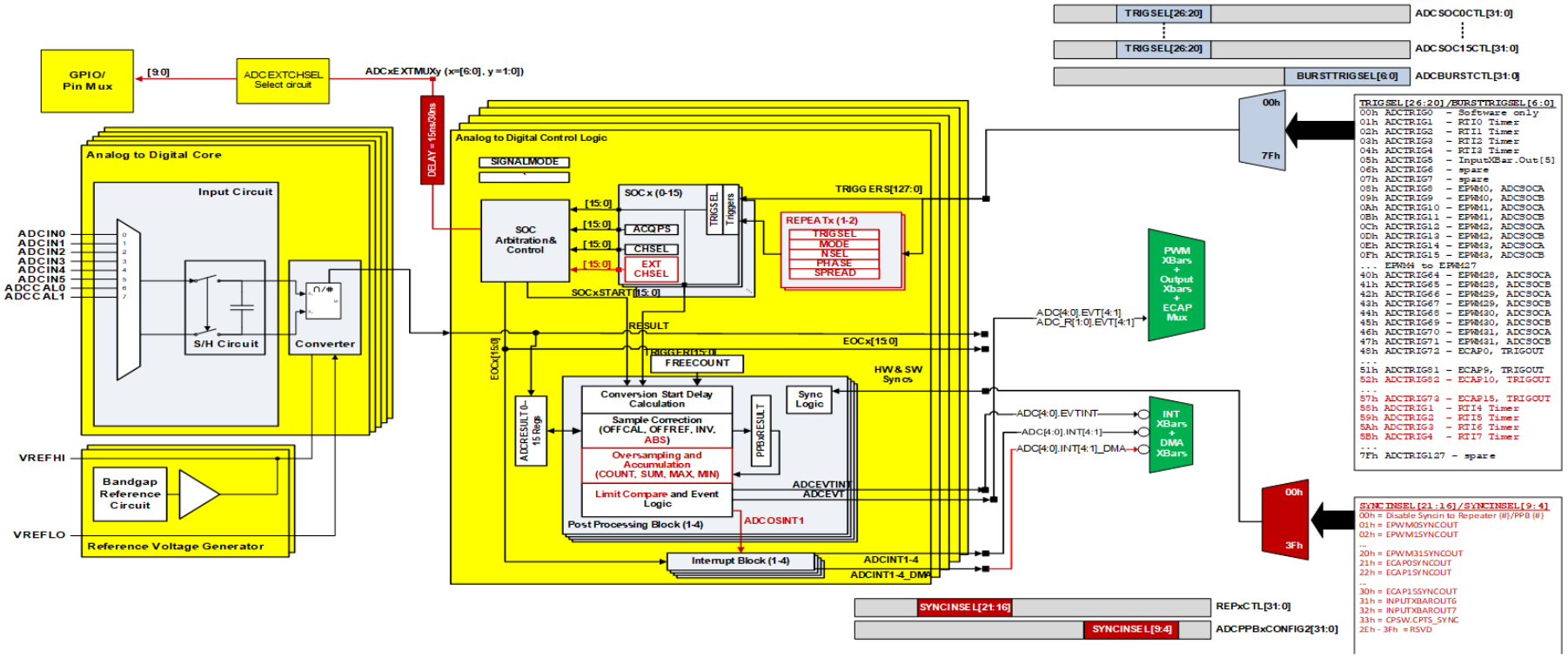


Figure 7-93. ADC Integration Diagram

7.5.2.3 ADC Configurability

Some ADC configurations are individually controlled by the SOCs, while others are globally controlled per ADC module. Table 7-104 summarizes the basic ADC options and the level of configurability. The subsequent sections discuss these configurations.

Table 7-104. ADC Options and Configuration Levels

Options	Configurability
Clock	Per module(1)
Resolution	Not configurable (12-bit only)
Signal mode	Per module
Reference voltage source	Not configurable (external or internal reference only)
Trigger source	Per SOC(1)
Converted channel	Per SOC
Acquisition window duration	Per SOC(1)
EOC location	Per module
Burst Mode	Per module(1)

(1) Writing these values differently to different ADC modules can cause the ADCs to operate asynchronously. See Section 7.5.2.15.1 for guidance on when the ADCs are operating synchronously or asynchronously.

7.5.2.3.1 Clock Configuration

The base ADC clock is provided directly by the system clock (SYSCLK). SYSCLK is used to generate the ADC acquisition window. The register ADCCTL2 has a PRESCALE field that determines the ADCCLK. ADCCLK is used to clock the converter, and is only active during the conversion phase. At all other times, including during the sample-and-hold window, the ADCCLK signal is gated off.

In 16-bit mode, the core requires approximately 29.5 ADCCLK cycles to process a voltage into a conversion result, while in 12-bit mode, this process requires approximately 11.5 ADCCLK cycles (and 13.5 for ADC_R0/1). The choice of resolution also determines the necessary duration of the acquisition window.

Section 7.5.2.15.2.

Note

To determine an appropriate value for ADCCTL2.PRESCALE, see the device data sheet to determine the maximum SYSCLK and ADCCLK frequency.

7.5.2.3.2 Resolution

The resolution of the ADC determines how finely the analog range is quantized into digital values. Each ADC module supports a fixed resolution of 12 bits.

7.5.2.3.3 Voltage Reference

7.5.2.3.3.1 ADC Reference - Internal Buffer Control Registers

There are two internal ADC reference buffers in the device, REFBUF0 and REFBUF1, to provide precise reference of 1.8 V to ADC. REFBUF0 is associated with ADC0, ADC1, and ADC2. REFBUF1 is associated with ADC3 and ADC4.

The ADC can operate with the internal reference or an external reference. Both internal and external reference are connected to the same package balls. Only one reference can be active at any given time

ADC Reference connection is shown in Figure 7-94.

The Voltage rails ADC_VREF*_G0 connects to REFBUF0, ADC0.

The voltage rails ADC_VREF*_G1 connects to ADC1 and ADC2.

The voltage rails ADC_VREF*_G2 connects to REFBUF1, ADC3, and ADC4.

The voltage rails ADC_VREF*_G3 connects to the ADC_R0 and ADC_R1

If internal reference is required to be used for ADC1 and ADC2, a board connection is provided to connect ADC_VREF*_G0 to ADC_VREF*_G1.

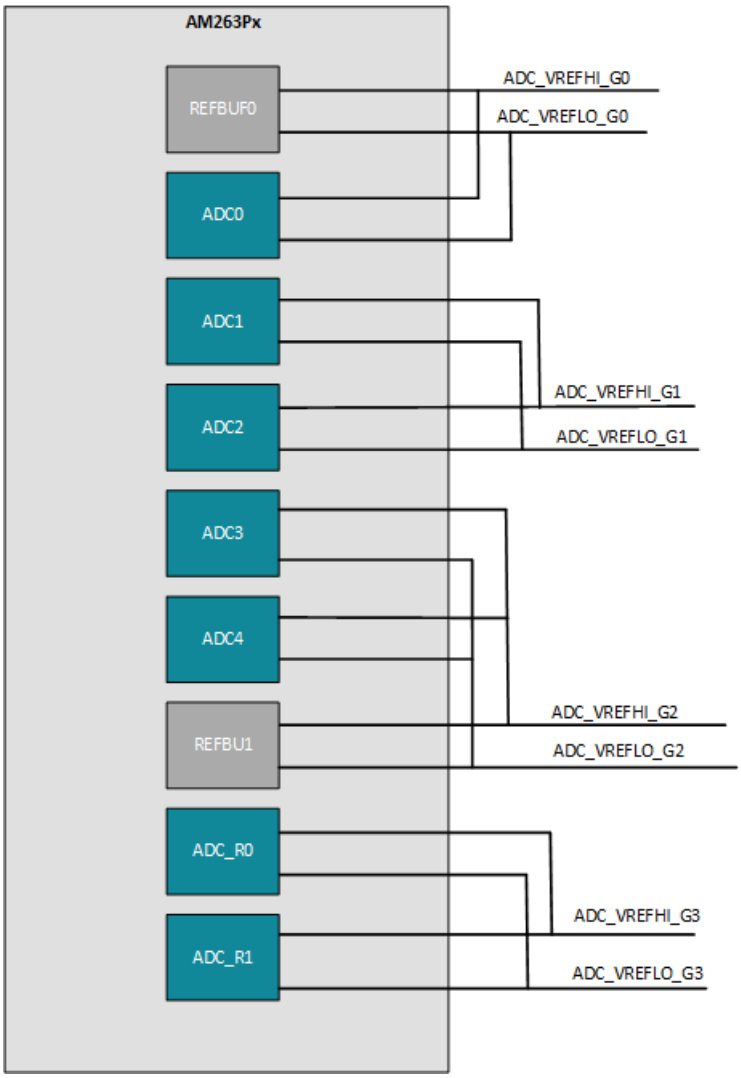


Figure 7-94. ADC Reference Connectivity Diagram

Internal reference are disabled by default. If external reference is not used, internal reference buffers can be enabled by the application for driving the ADC reference.

The ADC_REFBUF0_CTRL register is used to enable ADC Reference Buffer 0. Similarly, the ADC_REFBUF1_CTRL register is used to enable ADC Reference Buffer 1.

The MASK_ANA_ISO register must be set to 0x7 before ADC reference buffers are enabled. This prevents any undesirable behavior from voltage monitors triggering SOC reset.

Note

After the reference buffer is enabled, program the MASK_ANA_ISO back to 0x0 to re-enable the voltage monitors.

There are voltage monitors on all the three reference voltage rails for safe the reliable operation of ADC.

The ADC_REF_COMP_CTRL register is used to enable the reference monitor comparators. ADC_REF_COMP_CTRL. ADC0_REFOK_EN enables the voltage monitor on ADC_VREF*_G0.

ADC_REF_COMP_CTRL. ADC12_REFOK_EN enables the voltage monitor on ADC_VREF*_G1.

ADC_REF_COMP_CTRL. ADC34_REFOK_EN enables the voltage monitor on ADC_VREF*_G2.

The status of the ADC reference rails is indicated in the ADC_REF_GOOD_STATUS register.

Note

ADC cannot be enabled without proper ADC voltage reference.

7.5.2.3.4 ADC Modes of Operation

This section describes the ADC Modes of Operation.

Single-ended Mode:

The ADC output code is a 12-bit value, but internally it can generates slightly more than 12 effective data bits. There are operating modes that can be selected to clip and/or shift the ADC output to utilize different regions in SE (single-ended) mode. These modes can be selected by utilizing bits adcX_cfg_1p1v[79:80]. The default mode (00) simply clips the output within 0 and 4095 to maintain a 12 bit value; this effectively limits the maximum input of the ADC to 3.2V.

Table 7-105. ADC Output Code Clip and Shift Options

Input		Output	
Equation	**Input Voltage**	**Raw O/P**	**Default(00)**
3.3/4224 * 0	0	0	0
3.3/4224 * 1	0.00078125	1	1
3.3/4224 * 2	0.0015625	2	2
...	...	...	...
3.3/4224 * 127	0.09921875	127	127
3.3/4224 * 128	0.1	128	128
3.3/4224 * 129	0.10078125	129	129
...	...	...	...
3.3/4224 * 4094	3.1984375	4094	4094
3.3/4224 * 4095	3.19921875	4095	4095
3.3/4224 * 4096	3.2	4096	4095
...	...	...	...
3.3/4224 * 4221	3.29765625	4221	4095
3.3/4224 * 4222	3.2984375	4222	4095
3.3/4224 * 4223	3.29921875	4223	4095

Differential Mode:

In differential mode ADC output code can be estimated using the following equation

ADC Output Code = floor((VinPX-VinMX) / step_size + 2112

Where step_size=(VrefP-VrefM)*33/18*2/4096

Please note the addition factor is 2112 as the ADC generates a full-scale code in raw o/p mode as 4223. This 64 LSB shift can be compensated if adcX_cfg_1p1v[79:80] is set to 01 in differential mode

Input Configuration:

The ADC can be operated in both single-ended and differential mode.

Table 7-106. ADC Input Selection Logic

chsel_1p1v<2:0>	diff_mode_1p1v=1	diff_mode_1p1v=0
000	inP0-inM0	inP0
001	inM0-inP0	inM0
010	inP1-inM1	inP1
011	inM1-inP1	inM1
100	inP2-inM2	inP2
101	inM2-inP2	inM2
110	inP3-inM3	inP3
111	inM3-inP3	inM3

7.5.2.3.5 ADC Usage and Configuration Note

The bits adcX_cfg_1p1v[79:80] are common to all 5 ADC modules. Depending on the individual ADC (ADC0-ADC4) configuration (Single-ended or Differential) the module will behave according to following information.

ADC configured in Single-ended Mode:

- Selection=00: Normal ADC o/p [+0.0V = code 0 and +3.2V = code 4095]
- Selection=10: Reserved
- Selection=11: Reserved

ADC configured in Differential Mode:

- Selection=00: Normal ADC Output with a code differential offset of 64. The output code rolls over above input 3.2V.

  (A 3.2V to 3.3V input voltage will result in the same output code as 0-100mV input voltage)
- Selection=01: Normal Differential ADC Output

  [-3.2V Differential = code 0 and +3.2V Differential = code 4095]
- Selection=10: Reserved
- Selection=11: Reserved

7.5.2.3.6 Interpreting Conversion Results

Based on a given ADC conversion result, the corresponding analog input is given in Table 7-107 and Table 7-108. This corresponds to the center of the possible range of analog voltages that can produce this conversion result.

Table 7-107. 12-Bit Digital-to-Analog Formulas

Digital Value	Analog Equivalent
when ADCRESULTy = 0	$\text{ADCINx} \leq \text{VREFLO}$ (2)
when 0 < ADCRESULTy < 4095	$\text{ADCINx} = (\text{VREFHI} - \text{VREFLO})\left(\frac{\text{ADCRESULTy}}{4096}\right) + \text{VREFLO}$ (3)

Table 7-107. 12-Bit Digital-to-Analog Formulas (continued)

Digital Value	Analog Equivalent
when ADCRESULTy = 4095	$ADCINx \geq VREFHI$ (4)

The ADC can be operated in both single-ended and differential mode; the inputs can be configured according to Table 7-108.

Table 7-108. ADC Input Selection Logic

chsel_1p1v<2:0>	diff_mode_1p1v=1	diff_mode_1p1v=0
000	inp0-inm0	inp0
001	inm0-inp0	inm0
010	inp1-inm1	inp1
011	inm1-inp1	inm1
100	inp2-inm2	inp2
101	inm2-inp2	inm2
110	inp3-inm3	inp3
111	inm3-inp3	inm3

Single-ended Mode:

The ADC is designed for 12-bit output, but internally the ADC generates slightly more than 12 bits. There are certain modes that can be selected utilizing bits adcX_cfg_1p1v<80:79> to clip and shift the ADC output to utilize different regions in SE (single-ended) mode. The default mode (00) clips the output within 0 and 4095 to maintain a 12-bit value; this effectively limits the maximum input of the ADC to 3.2V. In the shifted mode (01), the ADC effectively shifts the input so that the ADC creates a 12-bit output between 0.1V and 3.3V.

Input		Output (adcX_cfg_1p1v<80:79>)	
Equation	Input Voltage	Raw O/P	Default(00)
3.3/4224 * 0	0	0	0
3.3/4224 * 1	0.00078125	1	1
3.3/4224 * 2	0.0015625	2	2
...	...	...	...
3.3/4224 * 127	0.09921875	127	127
3.3/4224 * 128	0.1	128	128
3.3/4224 * 129	0.10078125	129	129
...	...	...	...
3.3/4224 * 4094	3.1984375	4094	4094
3.3/4224 * 4095	3.19921875	4095	4095
3.3/4224 * 4096	3.2	4096	4095
...	...	...	...
3.3/4224 * 4221	3.29765625	4221	4095
3.3/4224 * 4222	3.2984375	4222	4095
3.3/4224 * 4223	3.29921875	4223	4095

Differential Mode:

In differential mode ADC output code can be estimated using the following equation:

ADC Output Code = **floor((VinpX-VinmX)/step_size + 2112)**

Where step_size = (VrefP-VrefM) * 33/18/4224

Note the addition factor is 2112 as the ADC generates a full-scale code in raw o/p mode as 4223. This 64-LSB shift can be compensated if adcX_cfg_1p1v<80:79> is set to (01) in differential mode

The adcX_cfg_1p1v<80:79> is not normally accessible, but can be overwritten by following the below steps. The bits in this and surrounding registers are critical for ADC functionality. Care must be taken so that only the bits <80:79> are changed as desired, while remaining bits remain in the default programmed state.

- Write to EFUSE_OVERRIDE_ADC_CFG2 (24-bit value corresponding to adc0_cfg_1p1v<87:64>)
- Write to EFUSE_OVERRIDE_ADC_CFG_CTRL for the override to take effect.

7.5.2.3.7 ADC-CMPSS Signal Connections

In each ADC, two sets of differential pins shall be shared with pins of two CMPSSA and remaining one pair of differential pins shall be connected to two independent pins of CMPSSB. These pins are demonstrated in Figure 7-95 and Table 7-109 where the CHSEL values determine how the inputs are fed into ADC.

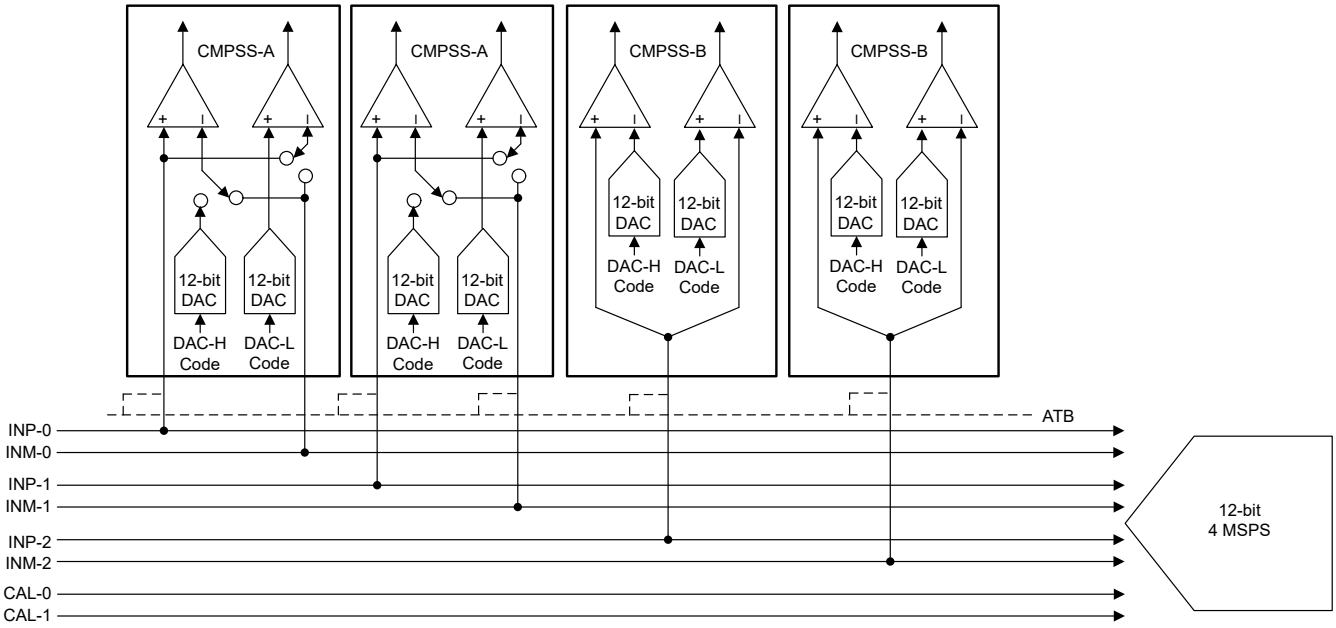


Figure 7-95. CMPSS and ADC Connections

Table 7-109. Connectivity between ADC Inputs to CMPSS Signals

Signal/Pin Name	ADC Input	CMPSS Input
ADC0 Channels		
ADC0_AIN0	ADC0:inp0 (+IN0)	CMPSSA0:inH (+IN)
ADC0_AIN1	ADC0:inm0 (-IN0)	CMPSSA0:inL (-IN)
ADC0_AIN2	ADC0:inp1 (+IN1)	CMPSSA1:inH (+IN)
ADC0_AIN3	ADC0:inm1 (-IN1)	CMPSSA1:inL (-IN)
ADC0_AIN4	ADC0:inp2 (+IN2)	CMPSSB0:inH/inL (+IN/-IN)
ADC0_AIN5	ADC0:inm2 (-IN2)	CMPSSB1:inH/inL (+IN/-IN)
ADC_CAL1	ADC0:inm3 (-IN3)	X
ADC_CAL0	ADC0:inp3 (+IN3)	X
ADC1 Channels		
ADC1_AIN0	ADC1:inp0 (+IN0)	CMPSSA2:inH (+IN)
ADC1_AIN1	ADC1:inm0 (-IN0)	CMPSSA2:inL (-IN)
ADC1_AIN2	ADC1:inp1 (+IN1)	CMPSSA3:inH (+IN)
ADC1_AIN3	ADC1:inm1 (-IN1)	CMPSSA3:inL (-IN)
ADC1_AIN4	ADC1:inp2 (+IN2)	CMPSSB2:inH/inL (+IN/-IN)
ADC1_AIN5	ADC1:inm2 (-IN2)	CMPSSB3:inH/inL (+IN/-IN)

Table 7-109. Connectivity between ADC Inputs to CMPSS Signals (continued)

Signal/Pin Name	ADC Input	CMPSS Input
ADC_CAL1	ADC1:inm3 (-IN3)	X
ADC_CAL0	ADC1:inp3 (+IN3)	X
ADC2 Channels		
ADC2_AIN0	ADC2:inp0 (+IN0)	CMPSSA4:inH (+IN)
ADC2_AIN1	ADC2:inm0 (-IN0)	CMPSSA4:inL (-IN)
ADC2_AIN2	ADC2:inp1 (+IN1)	CMPSSA5:inH (+IN)
ADC2_AIN3	ADC2:inm1 (-IN1)	CMPSSA5:inL (-IN)
ADC2_AIN4	ADC2:inp2 (+IN2)	CMPSSB4:inH/inL (+IN/-IN)
ADC2_AIN5	ADC2:inm2 (-IN2)	CMPSSB5:inH/inL (+IN/-IN)
ADC_CAL1	ADC2:inm3 (-IN3)	X
ADC_CAL0	ADC2:inp3 (+IN3)	X
ADC3 Channels		
ADC3_AIN0	ADC3:inp0 (+IN0)	CMPSSA6:inH (+IN)
ADC3_AIN1	ADC3:inm0 (-IN0)	CMPSSA6:inL (-IN)
ADC3_AIN2	ADC3:inp1 (+IN1)	CMPSSA7:inH (+IN)
ADC3_AIN3	ADC3:inm1 (-IN1)	CMPSSA7:inL (-IN)
ADC3_AIN4	ADC3:inp2 (+IN2)	CMPSSB6:inH/inL (+IN/-IN)
ADC3_AIN5	ADC3:inm2 (-IN2)	CMPSSB7:inH/inL (+IN/-IN)
ADC_CAL1	ADC3:inm3 (-IN3)	X
ADC_CAL0	ADC3:inp3 (+IN3)	X
ADC4 Channels		
ADC4_AIN0	ADC4:inp0 (+IN0)	CMPSSA8:inH (+IN)
ADC4_AIN1	ADC4:inm0 (-IN0)	CMPSSA8:inL (-IN)
ADC4_AIN2	ADC4:inp1 (+IN1)	CMPSSA9:inH (+IN)
ADC4_AIN3	ADC4:inm1 (-IN1)	CMPSSA9:inL (-IN)
ADC4_AIN4	ADC4:inp2 (+IN2)	CMPSSB8:inH/inL (+IN/-IN)
ADC4_AIN5	ADC4:inm2 (-IN2)	CMPSSB9:inH/inL (+IN/-IN)
ADC_CAL0	ADC4:inp3 (+IN3)	X
ADC_CAL1	ADC4:inm3 (-IN3)	X

Note

In the **ADC Input** column in ADC-CMPSS Signal Connectivity Table above, "inp" stands for positive inputs and "inm" stands for negative inputs.

7.5.2.4 SOC Principle of Operation

The ADC triggering and conversion sequencing is accomplished through configurable start-of-conversions (SOCs). Each SOC is a configuration set defining the single conversion of a single channel. In that set, there are three configurations: the trigger source that starts the conversion, the channel to convert, and the acquisition (sample) window duration. Upon receiving the trigger configured for a SOC, the wrapper makes sure that the specified channel is captured using the specified acquisition window duration.

Multiple SOCs can be configured for the same trigger, channel, and acquisition window as desired. Configuring multiple SOCs to use the same trigger allows the trigger to generate a sequence of conversions. Configuring multiple SOCs to use the same trigger and channel allows for oversampling.

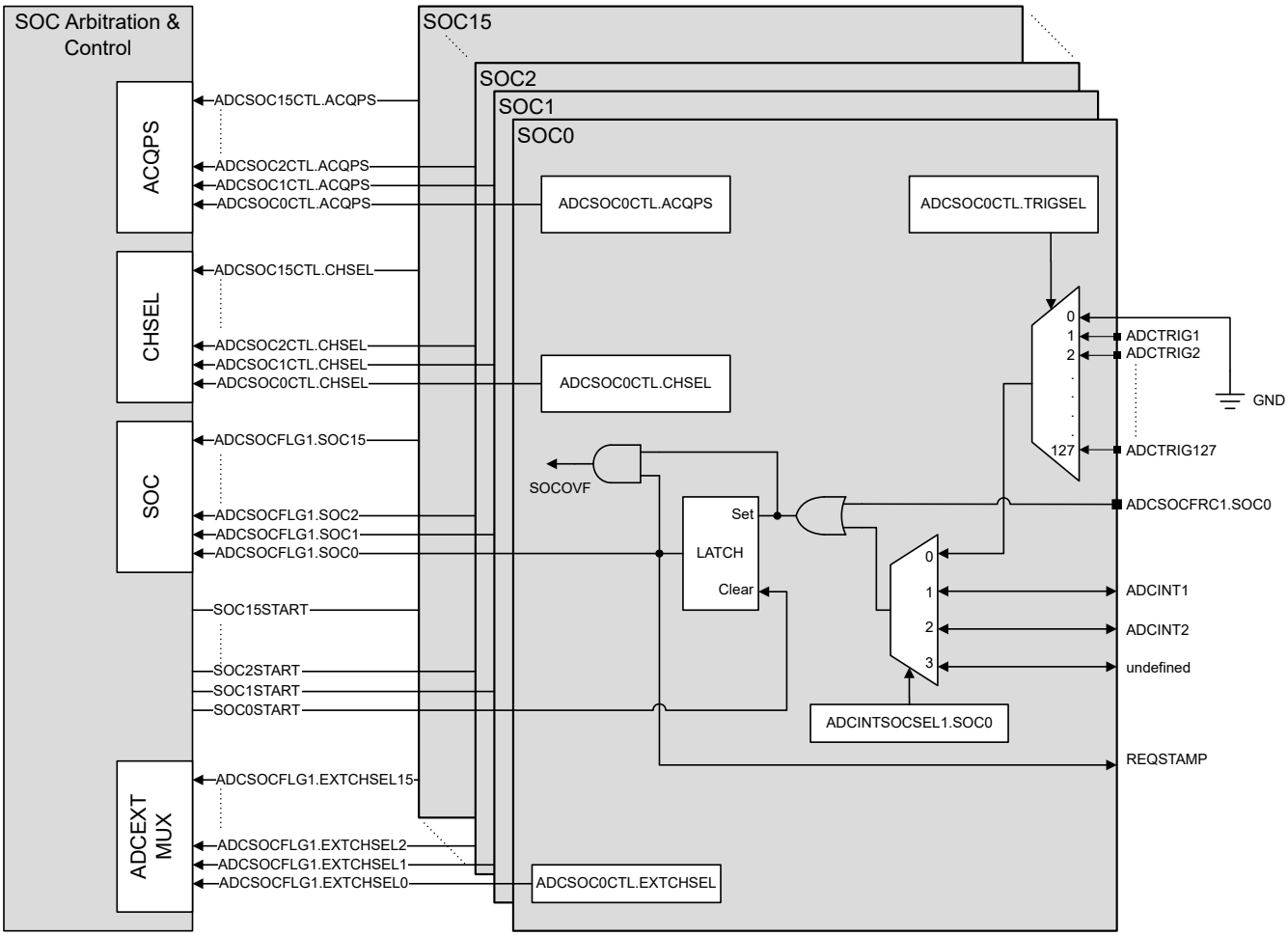


Figure 7-96. SOC Block Diagram

7.5.2.5 SOC Configuration Examples

The following sections provide some specific examples of how to configure the SOCs to produce some conversions.

7.5.2.5.1 Single Conversion from ePWM Trigger

SOC5 is chosen arbitrarily. Any of the SOCs can be used.

Assuming a 100ns sample window is desired with a SYSCLK frequency of MHz, then the acquisition window duration must be cycles. The ACQPS field must be set to .

As configured, when ePWM3 matches the period and generates the SOCB signal, the ADC begins sampling channel ADCINA1 (SOC5) immediately if the ADC is idle. If the ADC is busy, ADCINA1 begins sampling when SOC5 gains priority (see Section 7.5.2.6). The ADC control logic samples ADCINA1 with the specified acquisition window width of 100ns. Immediately after the acquisition is complete, the ADC begins converting the sampled voltage to a digital value. When the ADC conversion is complete, the results are available in the ADCRESULT5 register (see Section 7.5.2.14 for exact sample, conversion, and result latch timings).

7.5.2.5.2 Oversampled Conversion from ePWM Trigger

To configure the ADC to oversample ADCINA1 4 times, we use the same configurations as the previous example, but apply them to SOC5, SOC6, SOC7, and SOC8.

As configured, when ePWM3 matches the period and generates the SOCB signal, the ADC begins sampling channel ADCINA1 (SOC5) immediately if the ADC is idle. If the ADC is busy, ADCINA1 begins sampling when SOC5 gains priority (see Section 7.5.2.6). Once the conversion is complete for SOC5, SOC6 begins converting ADCINA1 and the results for SOC5 are placed in the ADCRESULT5 register. All four conversions eventually are completed sequentially, with the results in ADCRESULT5, ADCRESULT6, ADCRESULT7, and ADCRESULT8 for SOC5, SOC6, SOC7, and SOC8, respectively.

Note

It is possible, but unlikely, that the ADC can begin converting SOC6, SOC7, or SOC8 before SOC5 depending on the position of the round-robin pointer when the ePWM trigger is received. See Section 7.5.2.6 to understand how the next SOC to be converted is chosen.

7.5.2.5.3 Multiple Conversions from CPU Timer Trigger

This example shows how to sample multiple signals with different acquisition window requirements. CPU1 Timer 2 is used to generate the trigger. To see how to configure the CPU timer, see the *System Control and Interrupts* chapter.

A good first step when designing a sampling scheme with many signals is to list out the signals and the required acquisition window. From this, calculate the necessary number of SYSCLK cycles for each signal, then the ACQPS register setting. This is shown in , where a SYCLK of is assumed (cycle time).

Next decide which ADC pins to connect to each signal. This is highly dependent on the application board layout. Once the pins are selected, determining the value of CHSEL is straightforward (see Table 7-110).

Table 7-110. Example Connections for Multiple Signal Sampling

Signal Name	ADC Pin	CHSEL Register Value
Signal 1	ADCINA5	5
Signal 2	ADCINA0	0
Signal 3	ADCINA3	3
Signal 4	ADCINA2	2

With the information tabulated, generate the SOC configurations:

```
AdcaRegs.ADCSOC0CTL.bit.CHSEL = 5;        //SOC0 converts ADCINA5
AdcaRegs.ADCSOC0CTL.bit.ACQPS = 23;       //SOC0 uses a sample duration of 24 SYSCLK cycles
AdcaRegs.ADCSOC0CTL.bit.TRIGSEL = 3;      //SOC0 begins conversion on CPU1 Timer 2
AdcaRegs.ADCSOC1CTL.bit.CHSEL = 0;        //SOC1 converts ADCINA0
AdcaRegs.ADCSOC1CTL.bit.ACQPS = 88;       //SOC1 uses a sample duration of 89 SYSCLK cycles
AdcaRegs.ADCSOC1CTL.bit.TRIGSEL = 3;      //SOC1 begins conversion on CPU1 Timer 2
AdcaRegs.ADCSOC2CTL.bit.CHSEL = 3;        //SOC2 converts ADCINA3
AdcaRegs.ADCSOC2CTL.bit.ACQPS = 21;       //SOC2 uses a sample duration of 22 SYSCLK cycles
AdcaRegs.ADCSOC2CTL.bit.TRIGSEL = 3;      //SOC2 begins conversion on CPU1 Timer 2
AdcaRegs.ADCSOC3CTL.bit.CHSEL = 2;        //SOC3 converts ADCINA2
AdcaRegs.ADCSOC3CTL.bit.ACQPS = 58;       //SOC3 uses a sample duration of 59 SYSCLK cycles
AdcaRegs.ADCSOC3CTL.bit.TRIGSEL = 3;      //SOC3 begins conversion on CPU1 Timer 2
```

As configured, when CPU1 Timer 2 generates an event, SOC0, SOC1, SOC2, and SOC3 eventually is sampled and converted, in that order. The conversion results for ACINA5 (Signal 1) are in ADCRESULT0. Similarly, The results for ADCINA0 (Signal 2), ADCINA3 (Signal 3), and ADCINA2 (Signal 4) are in ADCRESULT1, ADCRESULT2, and ADCRESULT3, respectively.

Note

There is a possibility, but unlikely, that the ADC can begin converting SOC1, SOC2, or SOC3 before SOC0 depending on the position of the round-robin pointer when the CPU Timer trigger is received. See Section 7.5.2.6 to understand how the next SOC to be converted is chosen.

7.5.2.5.4 Software Triggering of SOCs

At any point, whether or not the SOCs have been configured to accept a specific trigger, a software trigger can set the SOCs to be converted. This is accomplished by writing bits in the ADCSOCFRC1 register.

Software triggering of the previous example without waiting for the CPU1 Timer 2 to generate the trigger can be accomplished by the statement:

```
AdcaRegs.ADCSOCFRC1.all = 0x000F;        //set SOC flags for SOC0 to SOC3
```

7.5.2.6 ADC Conversion Priority

When multiple SOC flags are set at the same time, one of two forms of priority determines the converted order. The default priority method is round-robin. In this scheme, no SOC has an inherent higher priority than another. Priority depends on the round-robin pointer (RRPOINTER). The RRPOINTER reflected in the

ADCSOCPRIORITYCTL register points to the last SOC converted. The highest priority SOC is given to the next value greater than the RRPOINTER value, wrapping around back to SOC0 after SOC15. At reset the value is 16 since 0 indicates a conversion has already occurred. When RRPOINTER equals 16 the highest priority is given to SOC0. The RRPOINTER is reset when the ADC module is reset or when the reset value is written to the SOCPRICTL register. The ADC module is reset by writing and clearing the SOFTPRES bit corresponding to the ADC instance.

An example of the round-robin priority method is given in Figure 7-97.

The SOCPRIORITY field in the ADCSOCPRIORITYCTL register can be used to assign high priority from a single to all of the SOCs. When configured as high priority, an SOC interrupts the round-robin wheel after any current conversion completes and inserts in as the next conversion. After the conversion completes, the round-robin wheel continues where the conversion was interrupted. If two high priority SOCs are triggered at the same time, the SOC with the lower number takes precedence.

High priority mode is assigned first to SOC0, then in increasing numerical order. The value written in the SOCPRIORITY field defines the first SOC that is not high priority. In other words, if a value of 4 is written into SOCPRIORITY, then SOC0, SOC1, SOC2, and SOC3 are defined as high priority, with SOC0 the highest.

An example using high priority SOC's is given in Figure 7-98.

A After reset, SOC0 is highest priority SOC; SOC7 receives trigger; SOC7 configured channel is converted immediately.

B RRPOINTER changes to point to SOC 7; SOC8 is now highest priority SOC.

C SOC2 & SOC12 triggers rcvd. simultaneously; SOC12 is first on round robin wheel; SOC12 configured channel is converted while SOC2 stays pending.

D RRPOINTER changes to point to SOC 12; SOC2 configured channel is now converted.

E RRPOINTER changes to point to SOC 2; SOC3 is now highest priority SOC.

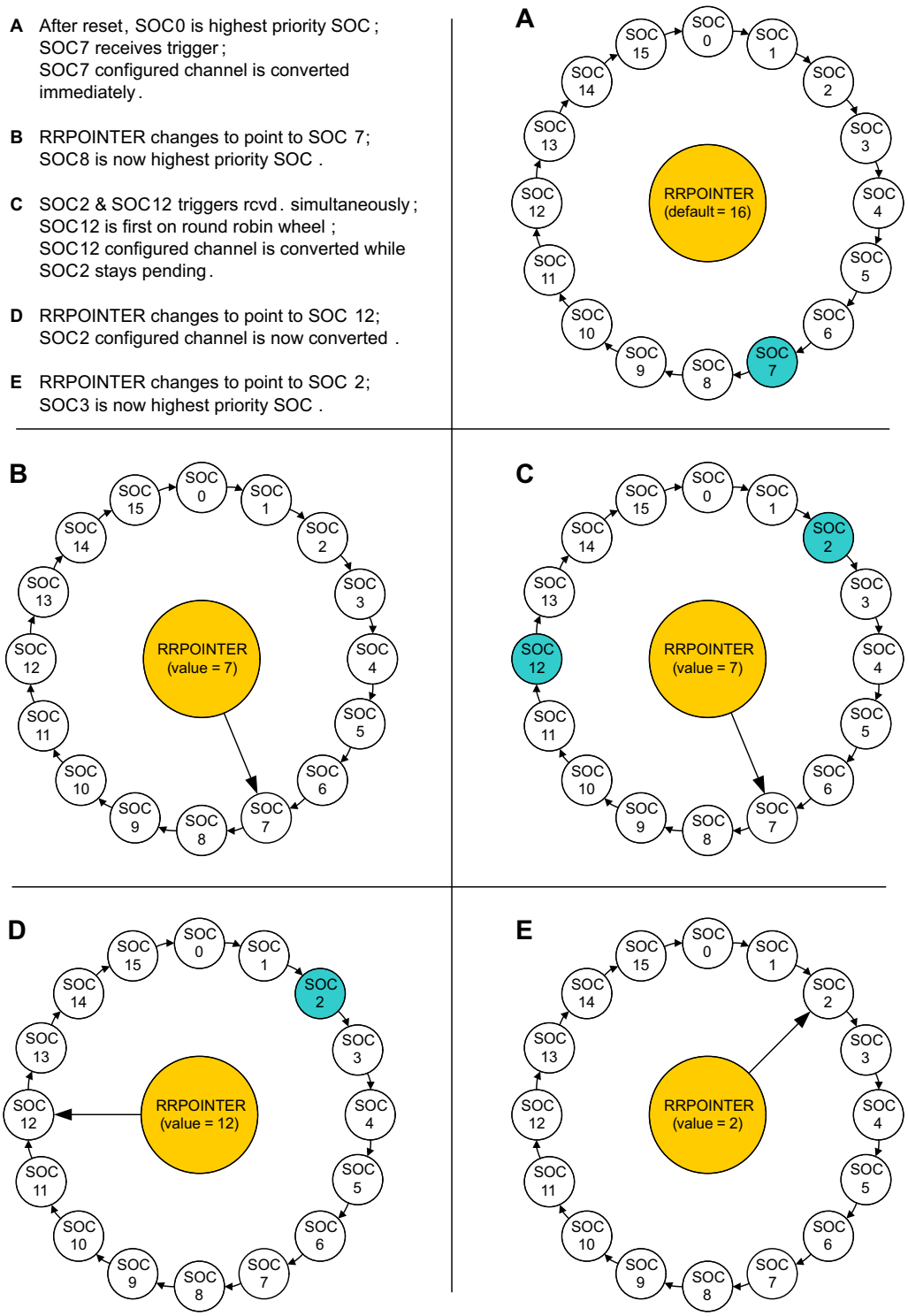


Figure 7-97. Round Robin Priority Example

Example when SOCPRIORITY = 4

A After reset, SOC4 is 1st on round robin wheel ;
SOC7 receives trigger ;
SOC7 configured channel is converted immediately .

B RRPOINTER changes to point to SOC 7;
SOC8 is now 1st on round robin wheel .

C SOC2 & SOC12 triggers rcvd . simultaneously ;
SOC2 interrupts round robin wheel and SOC 2 configured channel is converted while SOC 12 stays pending .

D RRPOINTER stays pointing to 7;
SOC12 configured channel is now converted .

E RRPOINTER changes to point to SOC 12;
SOC13 is now 1st on round robin wheel .

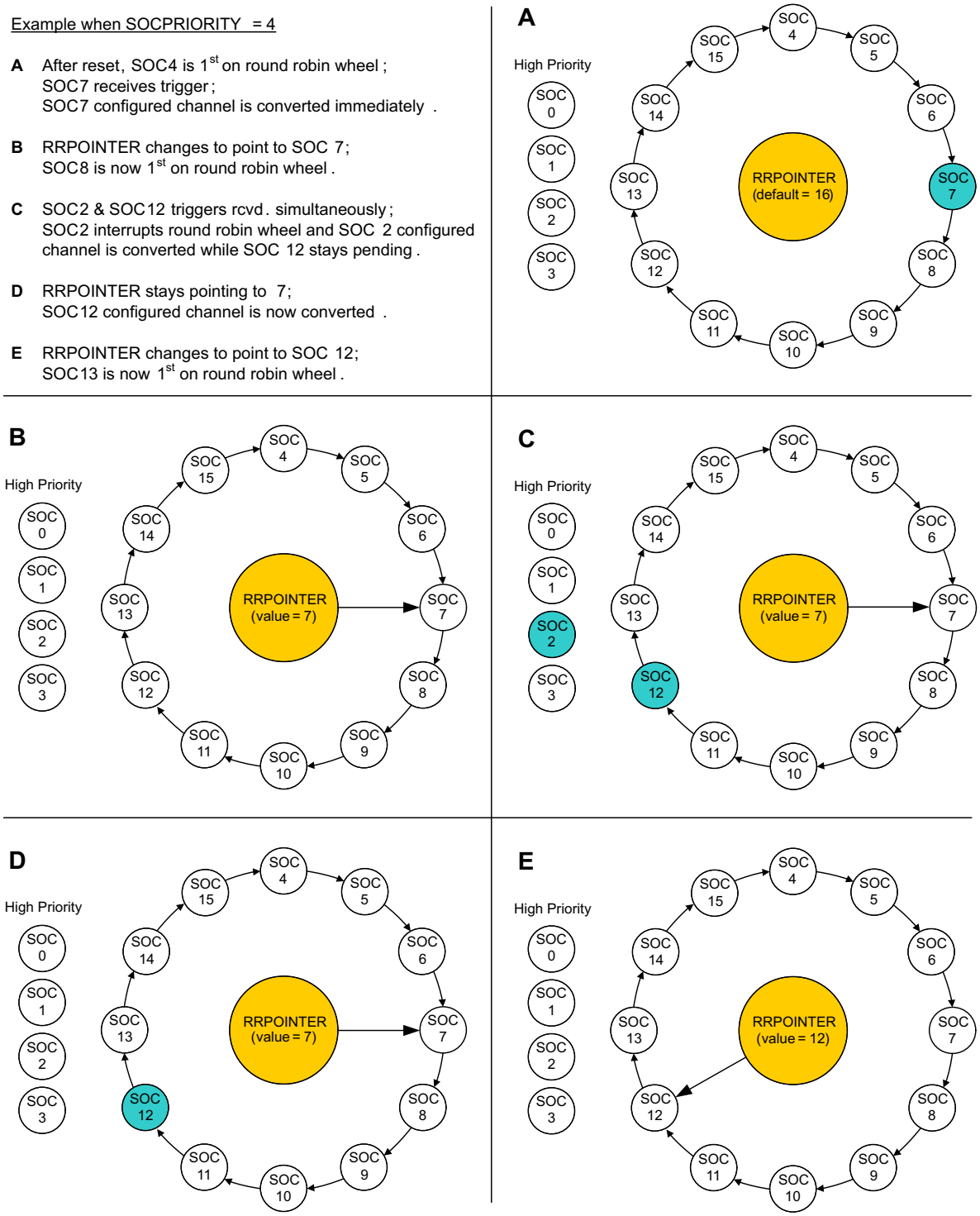


Figure 7-98. High Priority Example

7.5.2.7 Burst Mode

Burst mode allows a single trigger to walk through the round-robin SOCs one or more at a time. Setting the bit BURSTEN in the ADCBURSTCTL register configures the ADC wrapper for burst mode. This causes the TRIGSEL field to be ignored, but only for SOCs that are configured for round-robin operation (not high priority). Instead of the TRIGSEL field, all round-robin SOCs are triggered based on the BURSTTRIG field in the ADCBURSTCTL register. Upon reception of the burst trigger, the ADC wrapper does not set all round-robin SOCs to be converted, but only (ADCBURSTCTL.BURSTSIZE + 1) SOCs. The first SOC to be set is the SOC with the highest priority based on the round-robin pointer, and subsequent SOCs are set until BURSTSIZE SOCs have been set.

Note

When configuring the ADC for burst mode, the user is responsible for ensuring that each burst of conversions is allowed to complete before the next burst trigger is received. The value of (ADCBURSTCTL.BURSTSIZE + 1) must be less than or equal to the number of SOCs configured for round-robin priority. If the previous burst is not complete at the time when a new burst trigger arrives, for each SOC that was already pending and receives a new trigger, the corresponding overflow flag in ADCSOCOVF1 is set.

For example, if SOCPRIORITY = 12, that is, SOC12, SOC13, SOC14, and SOC15 are in round-robin, ADCBURSTCTL.BURSTSIZE setting must be ≤3 for burst mode to operate correctly.

7.5.2.7.1 Burst Mode Example

Burst mode can be used to sample a different set of signals on every other trigger. In the following example, ADCIN7 and ADCIN5 are converted on the first trigger from CPU1 Timer 2 and every other trigger thereafter. ADCIN2 and ACIN3 are converted on the second trigger from CPU1 Timer 2 and every other trigger thereafter. All signals are converted with 20 SYSCLK cycle wide acquisition windows, but different durations can be configured for each SOC as desired.

```
AdcaRegs.BURSTCTL.BURSTEN = 1;              //Enable ADC burst mode
AdcaRegs.BURSTCTL.BURSTTRIG = 3;            //CPU1 Timer 2 triggers burst of conversions
AdcaRegs.BURSTCTL.BURSTSIZE = 1;            //conversion bursts are 1 + 1 = 2 conversions long
AdcaRegs.SOCPRICTL.bit.SOCPRIORITY = 12;    //SOC0 to SOC11 are high priority
AdcaRegs.ADCSOC12CTL.bit.CHSEL = 7;         //SOC12 converts ADCINA7
AdcaRegs.ADCSOC12CTL.bit.ACQPS = 19;        //SOC12 uses sample duration of 20 SYSCLK cycles
AdcaRegs.ADCSOC13CTL.bit.CHSEL = 5;         //SOC13 converts ADCINA5
AdcaRegs.ADCSOC13CTL.bit.ACQPS = 19;        //SOC13 uses sample duration of 20 SYSCLK cycles
AdcaRegs.ADCSOC14CTL.bit.CHSEL = 2;         //SOC14 converts ADCINA2
AdcaRegs.ADCSOC14CTL.bit.ACQPS = 19;        //SOC14 uses sample duration of 20 SYSCLK cycles
AdcaRegs.ADCSOC15CTL.bit.CHSEL = 3;         //SOC15 converts ADCINA3
AdcaRegs.ADCSOC15CTL.bit.ACQPS = 19;        //SOC15 uses sample duration of 20 SYSCLK cycles
```

When the first CPU1 Timer 2 trigger is received, SOC12 and SOC13 are converted immediately if the ADC is idle. If the ADC is busy, SOC12 and SOC13 are converted once the SOCs gain priority. The results for SOC12 and SOC13 are in ADCRESULT12 and ADCRESULT13, respectively. After SOC13 completes, the round-robin pointer gives the highest priority to SOC14. Because of this, when the next CPU1 Timer 2 trigger is received, SOC14 and SOC15 is set as pending and eventually converted. The results for SOC14 and SOC15 are in ADCRESULT14 and ADCRESULT15, respectively. Subsequent triggers continue to toggle between converting SOC12 and SOC13, and converting SOC14 and SOC15.

While the above example toggles between two sets of conversions, three or more different sets of conversions can be achieved using a similar approach.

7.5.2.7.2 Burst Mode Priority Example

An example of priority resolution using burst mode and high-priority SOCs is presented in Figure 7-99.

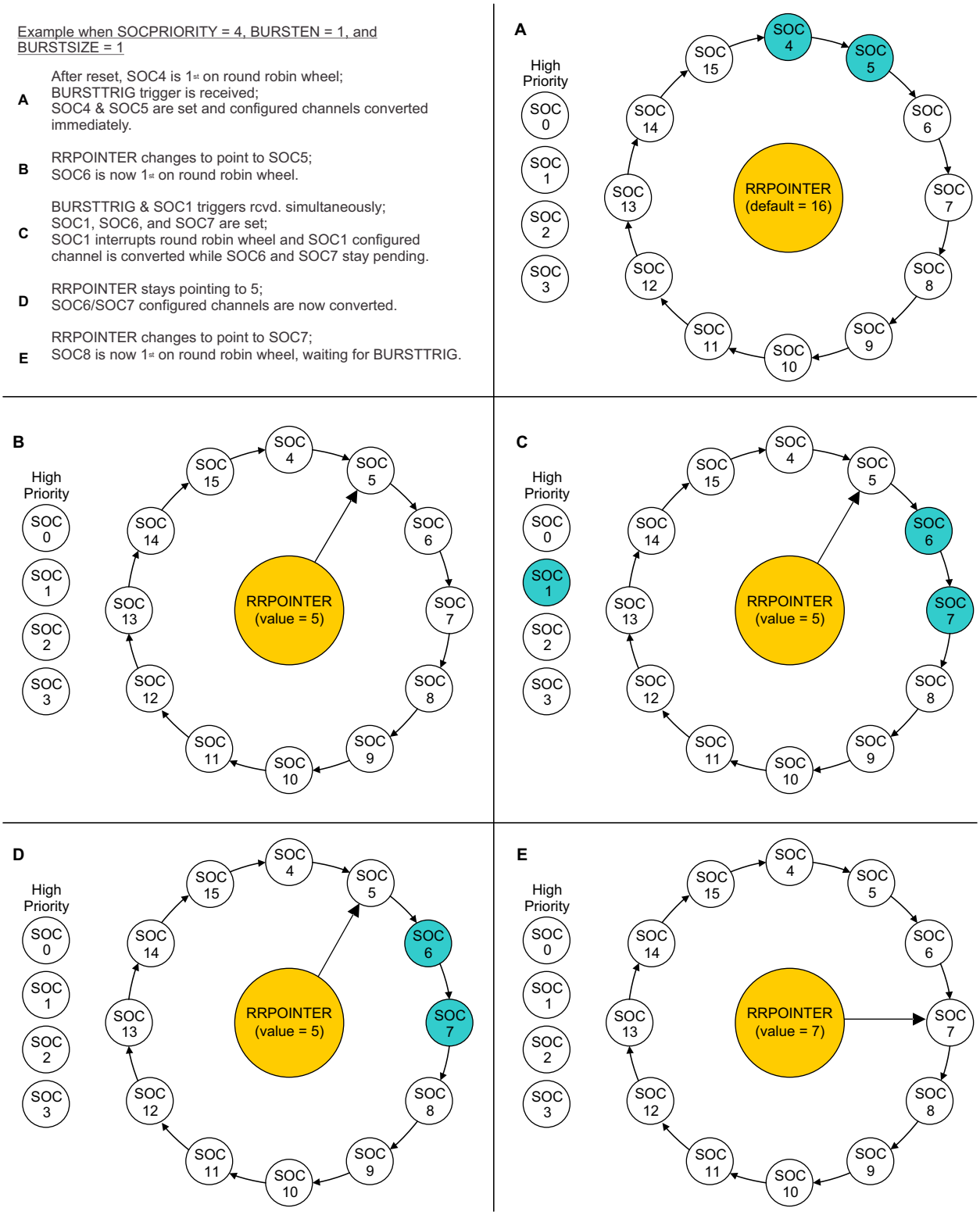


Figure 7-99. Burst Priority Example

7.5.2.8 EOC and Interrupt Operation

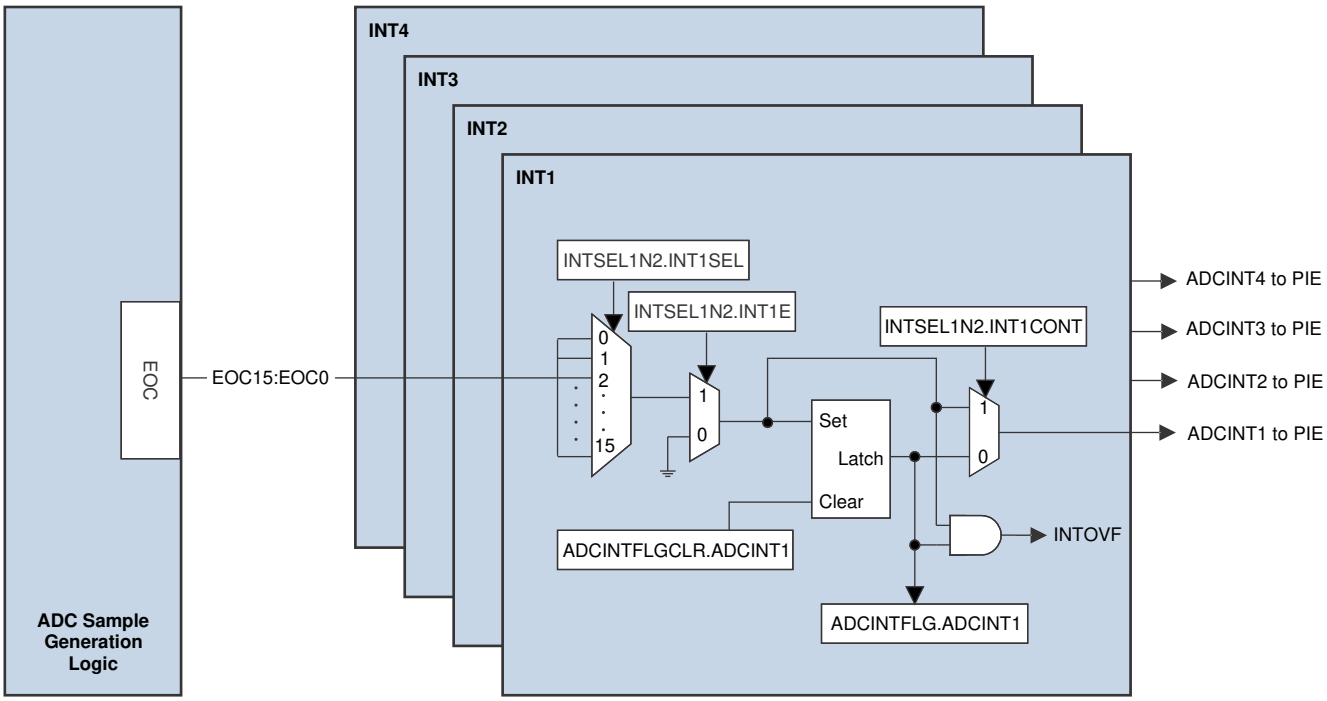
Each SOC has a corresponding end-of-conversion (EOC) signal. This EOC signal can be used to trigger an ADC interrupt. The ADC can be configured to generate the EOC pulse at either the end of the acquisition window or at the end of the voltage conversion. This is configured using the bit INTPULSEPOS in the ADCCTL1 register. See Section 7.5.2.14 for exact EOC pulse location.

It is also possible to generate an ADC interrupt based on a PPB oversampling logic event, such as when the sample count matches the configured limit. There are four oversampling interrupt (OSINT) flags available in each module for this purpose. Any of the ADCINT flags can be configured for an OSINT by configuring the INTxSEL field the corresponding ADCINTSELxNy register.

Note

The ADCCTL1.ADCBSY bit being clear does not indicate that all conversions in a set of SOCs have completed, only that the ADC is ready to process the next conversion. To determine if a sequence of SOCs is complete, link an ADCINT flag to the last SOC in the sequence and monitor that ADCINT flag.

ADC EoC Interrupts shows a block diagram of the ADC interrupt structure. The ADCINT1 and ADCINT2 Signals can be configured to generate an SoCx trigger to help create a continuous stream of conversions.



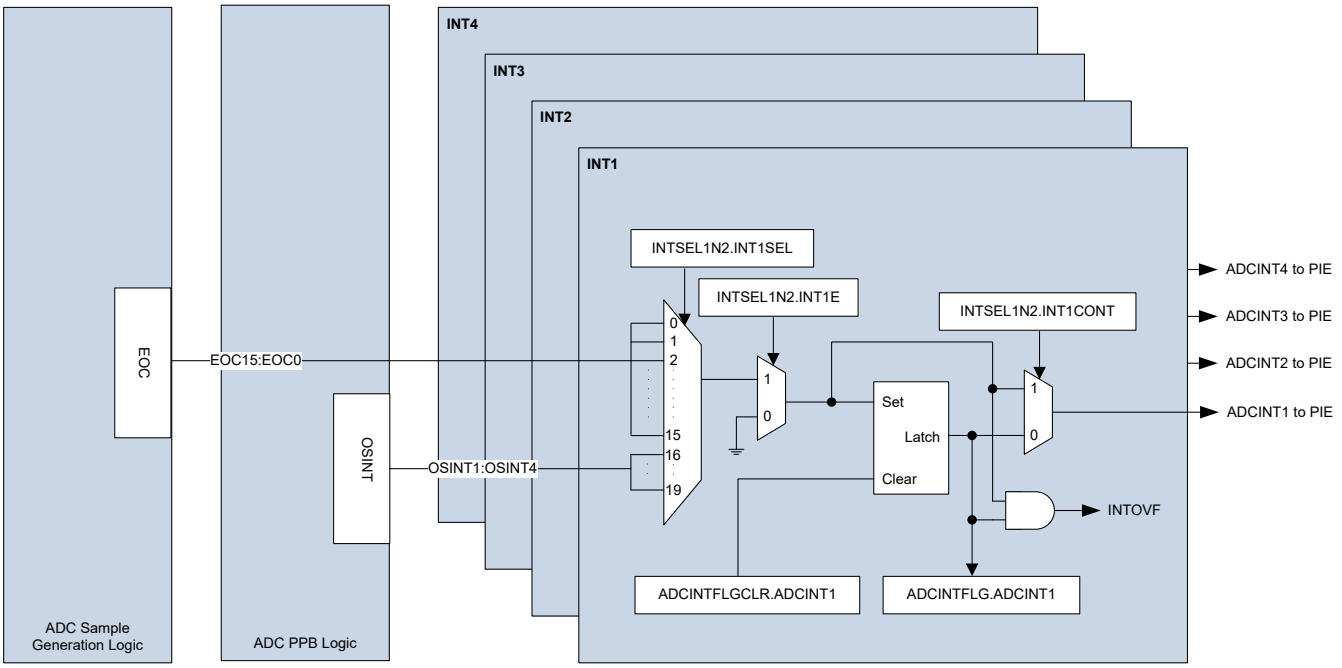


Figure 7-100. ADC EOC Interrupts

7.5.2.8.1 Interrupt Overflow

If the EOC signal sets a flag in the ADCINTFLG register, but that flag is already set, an interrupt overflow occurs. By default, overflow interrupts are not passed on to the PIE module. When an overflow occurs on a given flag in the ADCINTFLG register, the corresponding flag in the ADCINOVF register is set. This overflow flag is only used to detect that an overflow has occurred; the flag does not block further interrupts from propagating to the PIE module.

When an ADC interrupt overflow occurs, the application must check the appropriate ADCINTOVF flag inside the ISR or in the background loop and take appropriate action when an overflow is detected. The following code snippets demonstrate how to check the ADCINTOVF flag inside the ISR after attempting to clear the ADCINT flag.

```
// Clear the interrupt flag
AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1;      //clear INT1 flag for ADC-A

// Check if an overflow has occurred
if(1 == AdcaRegs.ADCINTOVF.bit.ADCINT1)     //ADCINT overflow occurred
{
    AdcaRegs.ADCINTOVFCLR.bit.ADCINT1 = 1   //Clear overflow flag
    AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1   //Re-clear ADCINT flag
}
```

```
//
// Clear the interrupt flag
//
ADC_clearInterruptStatus(ADCA_BASE, ADC_INT_NUMBER1);

//
// Check if an overflow has occurred
//
if(true == ADC_getInterruptOverflowStatus(ADCA_BASE, ADC_INT_NUMBER1))
{
    ADC_clearInterruptOverflowStatus(ADCA_BASE, ADC_INT_NUMBER1);
    ADC_clearInterruptStatus(ADCA_BASE, ADC_INT_NUMBER1);
}
```

7.5.2.8.2 Continue to Interrupt Mode

The INTxCONT bits in the ADCINTSEL1N2 and ADCINTSEL3N4 registers configure how interrupts are handled when an ADCINTFLG has not yet been cleared from a prior interrupt. This mode is disabled by default and additional overlapping interrupts are not issued to the PIE. By activating this mode, ADC interrupts always reach the PIE. If interrupts occur while ADCINTFLG is set, the ADCINTOVF register remains set regardless of the configuration of the INTxCONT bits.

7.5.2.9 Post-Processing Blocks

Each ADC module contains four post-processing blocks (PPB). These blocks can be associated with any of the RESULT registers using the ADCPPBxCONFIG.CONFIG bit field. The post-processing blocks have the ability to:

- Remove an offset associated with the ADCIN channel
- Subtract out a reference value
- Aggregate successive samples using sum, max, and min calculations
- Automatically calculate average of oversampled conversions without CPU overhead, when sample count is a power of 2
- Transform the conversion result into an absolute value
- Flag a zero-crossing point, with the option to trip a PWM and generate an interrupt
- Flag a high or low compare limit, with the option to trip a PWM and generate an interrupt
- Record the delay between the associated SOC trigger and when sampling actually begins

Figure 7-101 presents the structure of each PPB. Subsequent sections explain the use of each submodule.

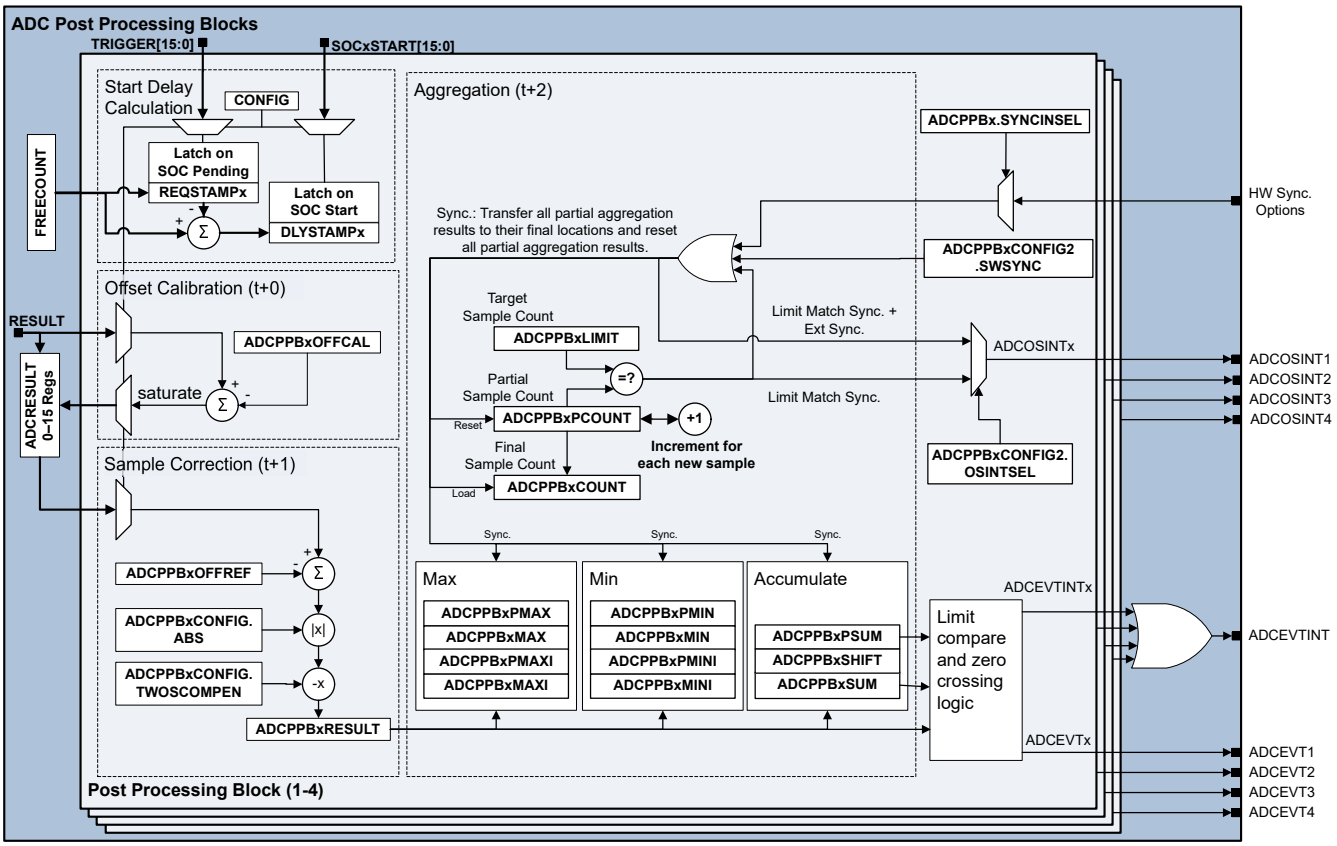


Figure 7-101. ADC PPB Block Diagram

7.5.2.9.1 PPB Offset Correction

In many applications, external sensors and signal sources produce an offset. A global trimming of the ADC offset is not enough to compensate for these offsets, which vary from channel to channel. The post-processing block can remove these offsets with zero overhead, saving numerous cycles in tight control loops.

Offset correction is accomplished by first pointing the ADCPPBxCONFIG.CONFIG to the desired SOC, then writing an offset correction value to the ADCPPBxOFFCAL.OFFCAL register. The post-processing block automatically adds or subtracts the value in the OFFCAL register from the raw conversion result and stores the value in the ADCRESULT register. This addition/subtraction saturates at 0 on the low end and 4095 on the high end.

Note

- Writing a 0 to the OFFCAL register effectively disables the offset correction feature, passing the raw result unchanged to the ADCRESULT register.
- To point multiple PPBs to the same SOC is possible. In this case, the OFFCAL value that is actually applied comes from the PPB with the lowest number.

7.5.2.9.2 PPB Error Calculation

In many applications, an error from a set point or expected value must be computed from the digital output of an ADC conversion. In other cases, a bipolar signal is necessary or convenient for control calculations. The PPB can perform these functions automatically, reducing the sample to output latency and reducing software overhead.

Error calculation is accomplished by first pointing the ADCPPBxCONFIG.CONFIG to the desired SOC, then writing a value to the ADCPPBxOFFCAL.OFFREF register. The post-processing block automatically subtracts the value in the OFFREF register from the ADCRESULT value and stores the value in the ADCPPBxRESULT register. This subtraction produces a sign-extended 32-bit result. It is also possible to selectively invert the calculated value before storing in the ADCPPBxRESULT register by setting the TWOSCOMPEN bit in the ADCPPBxCONFIG register.

If desired, the absolute value of the ADC result after the offset reference calculation can be obtained by setting the ADDCPPBxCONFIG.ABSEN bit. The absolute value is computed before evaluating the TWOSCOMPEN logic, so setting both TWOSCOMPEN and ABSEN always results in a negative value stored in ADCPPBxRESULT.

Note

- Do not write a value larger than 12 bits to the ADCPPBxOFFREF register.
- Since the ADCPPBxRESULT register is unique for each PPB, to point multiple PPBs to the same SOC and get different results for each PPB is possible.
- Writing a 0 to the ADCPPBxOFFREF register effectively disables the error calculation feature, passing the ADCRESULT value unchanged to the ADCPPBxRESULT register.

7.5.2.9.3 PPB Limit Detection and Zero-Crossing Detection

Many applications perform a limit check against the ADC conversion results. The PPB can automatically perform a check against high and low limits, or whenever ADCPPBxRESULT changes sign. Based on these comparisons, the PPB can generate a trip to the PWM and an interrupt automatically, lowering the sample to ePWM latency and reducing software overhead. This functionality also enables safety-conscious applications to trip the ePWM based on an out-of-range ADC conversion without any CPU intervention.

To enable this functionality, first point the ADCPPBxCONFIG.CONFIG to the desired SOC, then write a value to one or both of the registers ADCPPBxTRIPHI.LIMITHI and ADCPPBxTRIPLO.LIMITLO (zero-crossing detection does not require further configuration). Whenever these limits are exceeded, the PPBxTRIPHI bit or PPBxTRIPLO bit is set in the ADCEVTSTAT register. Note that the PPBxZERO bit in the ADCEVTSTAT register is gated by end-of-conversion (EOC), not by the sign change in the ADCPPBxRESULT register. The ADCEVTCLR register has corresponding bits to clear these event flags. The ADCEVTSEL register has

corresponding bits which allow the events to propagate through to the PWM. The ADCEVTINTSEL register has corresponding bits that allow the events to propagate through to the PIE.

One PIE interrupt is shared between all the PPBs for a given ADC module as shown in Figure 7-102.

Figure 7-103 illustrates the ADC limit compare and zero-crossing logic.

Note

- If different actions need to be taken for different PPB events from the same ADC module, then the ADCEVTINT ISR has to read the PPB event flags in the ADCEVTSTAT register to determine which event caused the interrupt.
- If different ePWM trips need to be generated separately for high compare, low compare, and zero-crossing, this can be achieved by pointing multiple PPBs to the same SOC.
- The zero-crossing detect circuit considers a result of zero to be positive.

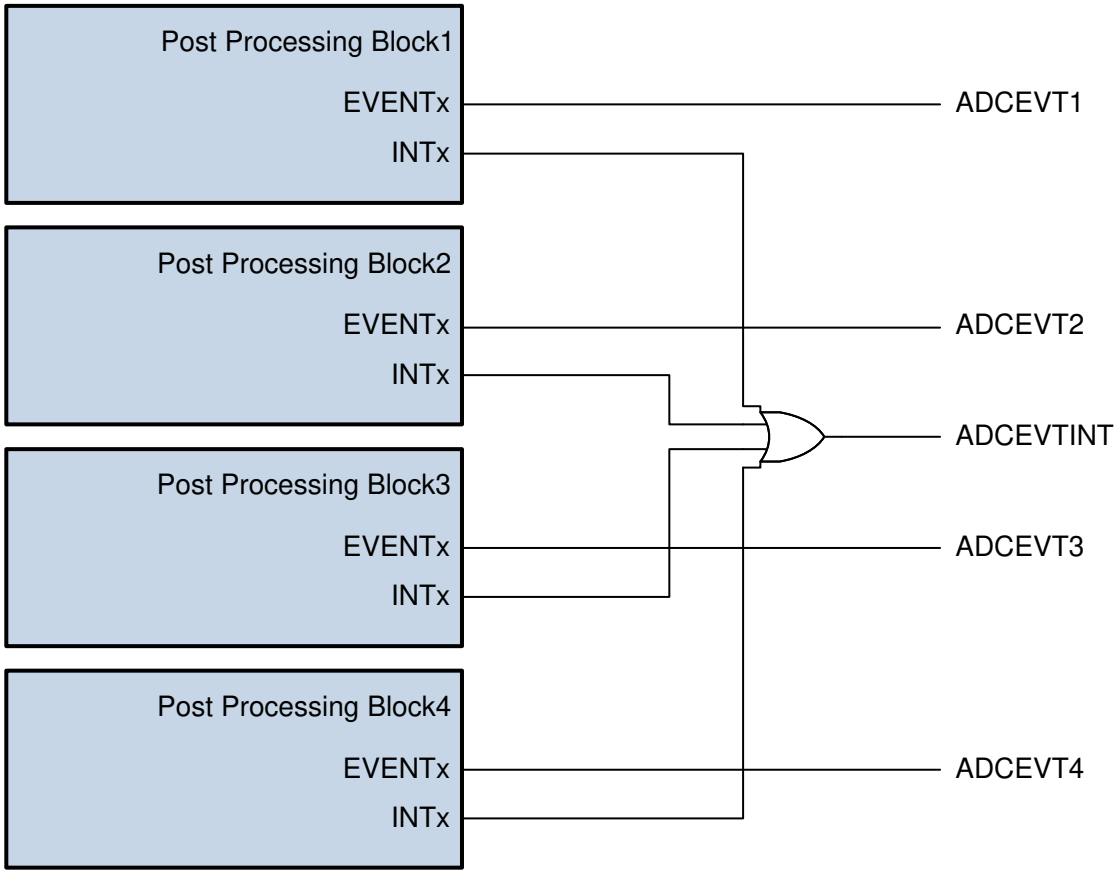


Figure 7-102. ADC PPB Interrupt Event

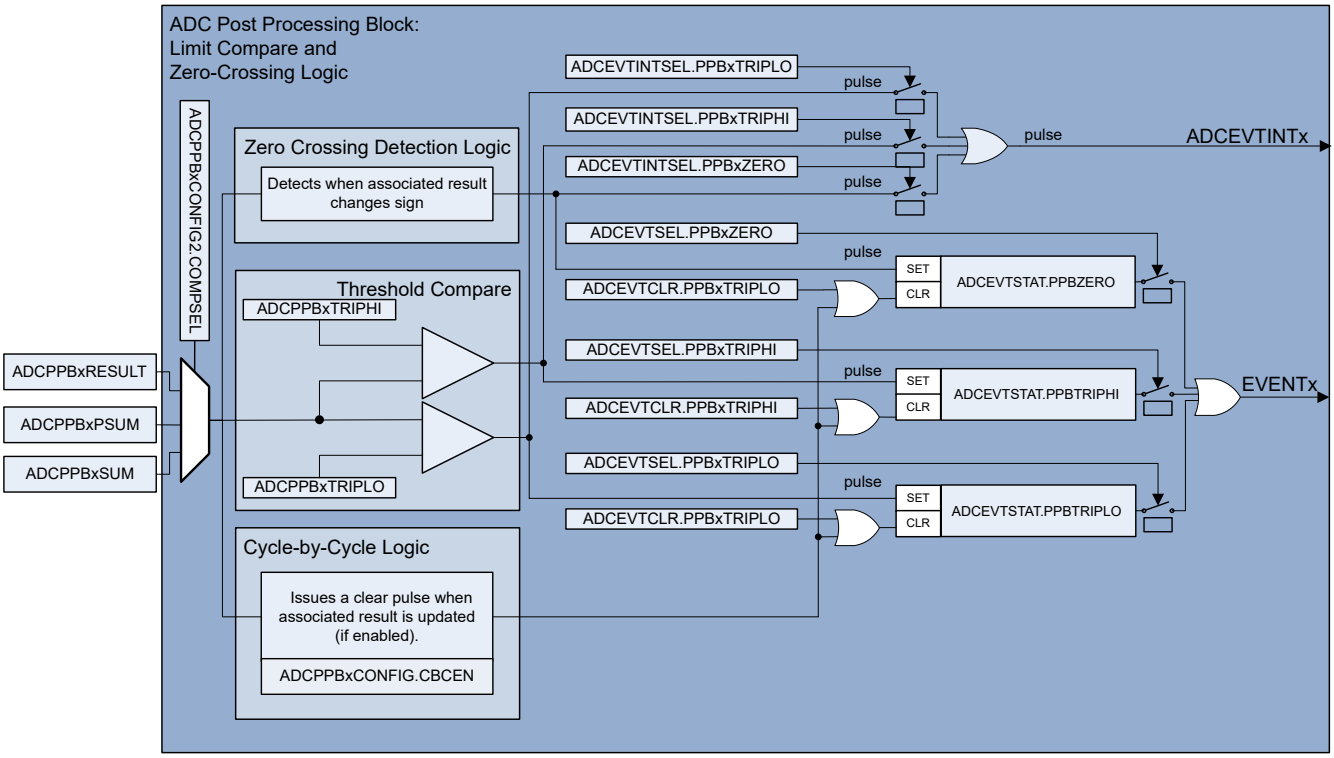


Figure 7-103. ADC PPB Limit Compare and Zero-Crossing Logic

7.5.2.9.4 PPB Sample Delay Capture

When multiple control loops are running asynchronously on the same ADC, there is a chance that an ADC request from two or more loops collide, causing one of the samples to be delayed. This shows up as a measurement error in the system. By knowing when this delay occurs and the amount of delay that has occurred, software can employ extrapolation techniques to reduce the error.

To this effect, each PPB has the field DLYSTAMP in the ADCPPBxSTAMP register. This field contains the number of SYSCLK cycles between when the associate SOC was triggered and when the SOC began converting.

This is achieved by having a global 12-bit free running counter based off of SYSCLK, which is in the field FREECOUNT in the ADCCOUNTER register. When the trigger for the associated SOC arrives, the value of this counter is loaded into the bit field ADCPPBxTRIPLO.REQSTAMP. When the actual sample window for that SOC begins, the value in REQSTAMP is subtracted from the current FREECOUNT value and stored in DLYSTAMP.

Note

If more than 4096 SYSCLK cycles elapse between the SOC trigger and the actual start of the SOC acquisition, the FREECOUNT register can overflow more than once, leading to incorrect DLYSTAMP value. Be cautious when using very slow conversions to prevent this from happening.

The sample delay capture does not function, if the associated SOC is triggered using software. The sample delay capture, however, correctly records the delay, if the software triggering of a different SOC causes the SOC associated with the PPB to be delayed

7.5.2.10 Result Safety Checker

For safety-critical applications, this device provides the ability to automatically compare ADC conversion results from multiple ADC modules against each other for consistency. The number of available safety checker tiles is specified in the device data sheet. Each ADC checker tile captures conversion results from the associated ADCs as soon as the conversions are complete, and compares the absolute value of the difference to the configured tolerance. If the computed delta is out of range, the checker can generate a trip event signal that is sent to an ePWM or output crossbar, and can also trigger an CPU interrupt. Figure 7-104 illustrates the structure and operation of an ADC result safety checker tile.

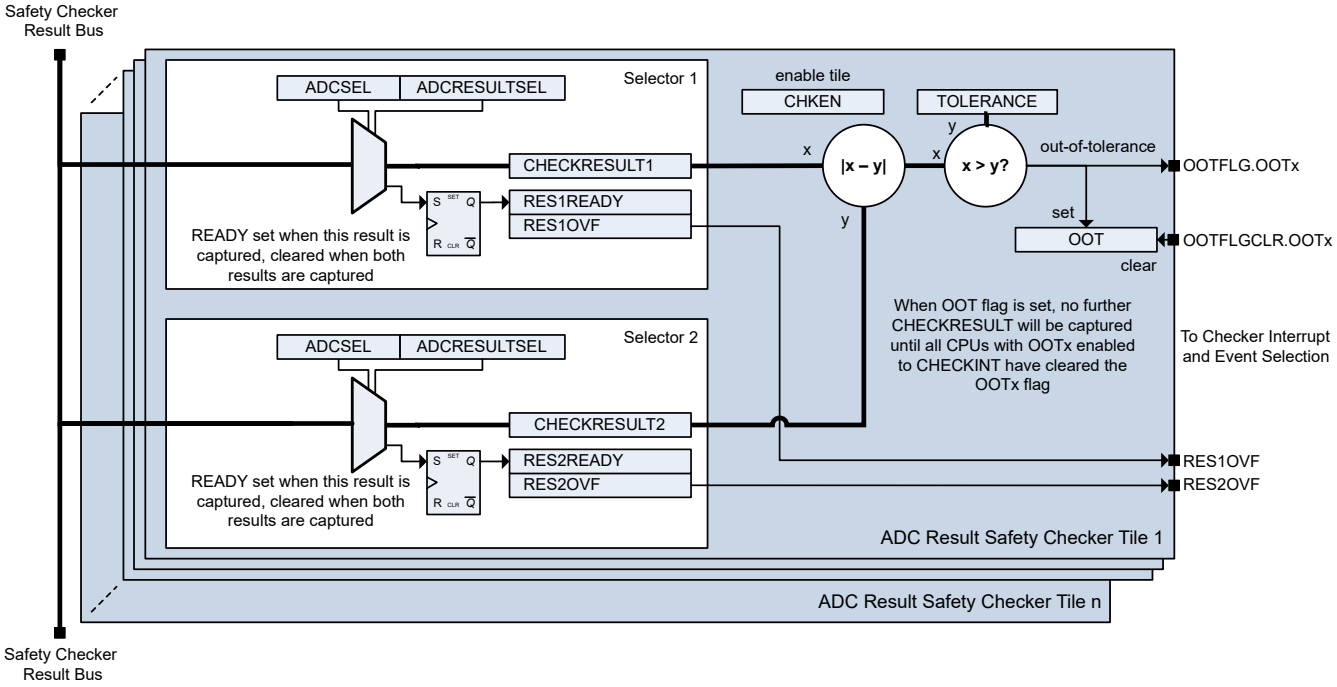


Figure 7-104. ADC Safety Checker Tile Diagram

7.5.2.11 Opens/Shorts Detection Circuit (OSDETECT)

The opens/shorts detection circuit (OSDETECT) can be used to detect pin faults in the system. The circuit connects to the ADC input after the channel select multiplexer but before the S+H circuit as shown in Figure 7-105.

Note

- The divider resistance tolerances can vary widely; hence, this feature must not be used to check for conversion accuracy.
- See the data sheet for implementation and availability of analog input channels.
- Due to high drive impedance, a S+H duration much longer than the ADC minimum is needed.

Figure 7-105. Opens/Shorts Detection Circuit

The circuit can be operated by writing a value to the DETECTCFG field in the ADCOSDETECT register. This causes the circuit to source a voltage onto the input during the S+H phase of any conversion. The voltage and drive strength of the OSDETECT circuit for different DETECTCFG settings is given in DETECTCFG Settings.

Table 7-111. TOP_CTRL ADC OSD Control Settings

TOP_CTRL. ADCx_OSD_CHEN	TOP_CTRL. ADCx_OSD_CTRL	Function	Drive Impedance	5K Voltage	7K Voltage
0	0	Off	Open	Open	Open
1	0	Zero Scale	5K || 7K	GND	GND
1	1	Zero Scale	5K	GND	Open
1	2	Zero Scale	7K	Open	GND
1	3	Full Scale	5K || 7K	3.3V VDD	3.3V VDD
1	4	Full Scale	5K	3.3V VDD	Open
1	5	Full Scale	7K	Open	3.3V VDD
1	6	5/12 Scale	5K || 7K	GND	3.3V VDD
1	7	7/12 Scale	5K || 7K	3.3V VDD	GND

Table 7-112. TOP_CTRL ADC_R OSD Control Settings

TOP_CTRL. adc5adc6_OSD_CTRL	Function	Drive Impedance	5K Voltage	7K Voltage
0	Zero Scale	5K || 7K	GND	GND
1	Zero Scale	5K	GND	Open
2	Zero Scale	7K	Open	GND
3	Full Scale	5K || 7K	3.3V VDD	3.3V VDD
4	Full Scale	5K	3.3V VDD	Open
5	Full Scale	7K	Open	3.3V VDD
6	5/12 Scale	5K || 7K	GND	3.3V VDD
7	7/12 Scale	5K || 7K	3.3V VDD	GND

Table 7-113. Channel Enable Control for ADC_R[0:1]

adc5adc6_osd_ctrl_1p1v[7:0] (Channel Enable)							
Bit 0	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7
ADC_R0_CH3	ADC_R0_CH2	ADC_R0_CH1	ADC_R0_CH0	ADC_R1_CH0	ADC_R1_CH1	ADC_R1_CH2	ADC_R1_CH3
Enabled if 1	Enabled if 1	Enabled if 1	Enabled if 1	Enabled if 1	Enabled if 1	Enabled if 1	Enabled if 1

7.5.2.11.1 Implementation

A representative circuit with the OSDETECT implementation consists of the signal source with series resistance R_S, shunt capacitor C_P, the equivalent OSDETECT resistance $R_{OSDETECT}$ and voltage $V_{OSDETECT}$ is shown in Figure 7-106 and can be used as a basis to calculate the signal level going in to the sampling capacitor. $R_{OSDETECT}$ and $V_{OSDETECT}$ are the equivalent input resistance and voltage source contributed by the OSDETECT circuit with values shown in DETECTCFG Settings for the different configuration settings. Refer to Figure 7-106 when deriving the input signal to S/H if signal source V_S is driving while the OSDETECT feature is enabled.

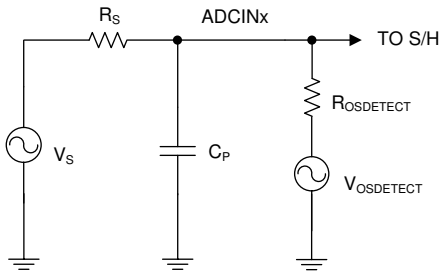


Figure 7-106. Input Circuit Equivalent with OSDETECT Enabled

The input impedance R_S and C_P are integral parts of the signal source or can have been implemented in the design to precondition the signal or to control signal settling time to meet S/H requirements. The input path has to be considered when using the OSDETECT feature, as this affects the conversion results. For instance, driving an input signal when this feature is enabled connects signal V_S to the OSDETECT circuit through R_S and affects the ADC results. Larger C_P values (in the order greater than hundreds of pF) require using higher ACQPS to make sure the signal at the input has settled prior to conversion.

To enable the circuit:

1. Configure the ADC for conversion (for example, channel, SOC, ACQPS, prescaler, trigger, and so on).
2. Set up the ADCOSDETECT register for the desired voltage divider connection as shown in DETECTCFG Settings.
3. Initiate a conversion and inspect the conversion result.

Interpret the results based on what is driving on the input side and what are the values of R_S and C_P. If the V_S signal can be disconnected from the input pin, the circuit can be used to detect open and shorted input pins as described in the following sections.

7.5.2.11.2 Detecting an Open Input Pin

By cycling through the various OSDETECT settings, the input signal is pulled towards the sourced voltages. An input with good drive strength (pin not open) is minimally affected. However, if the pin is open, the sampled voltages is close to the source voltages specified in DETECTCFG Settings.

7.5.2.11.3 Detecting a Shorted Input Pin

By cycling through the various OSDETECT settings, the input signal is pulled towards the sourced voltages. An input with finite drive strength (pin not shorted) is pulled toward each sourced voltage. However, if the pin is shorted, the signal remains at the same voltage.

7.5.2.12 Power-Up Sequence

Upon device power-up or system level reset, the ADC is powered down and disabled. When powering up the ADC, use the following sequence:

1. SYSCLK is used to generate the ADC acquisition window.
2. Set the desired ADC clock divider in the PRESCALE field of ADCCTL2.
3. Power up the ADC by setting the ADCPWDNZ bit in ADCCTL1.
4. Allow a delay before sampling. See the data sheet for the necessary time.

If multiple ADCs are powered up simultaneously, steps 1 and step 3 can each be done for all ADCs in one write instruction. Also, only one delay is necessary as long as the delay occurs after all the ADCs have begun powering up.

7.5.2.13 ADC Calibration

During the fabrication and test process, Texas Instruments calibrates the gain, offset, and linearity of the ADCs. These trim settings are stored in TI reserved OTP memory, and can be loaded using C-callable functions.

- The Device_cal() function copies the trim values for ADC from OTP memory to the respective trim registers.
- Calibration information is stored in TOP_CTRL registers and copied to ADC registers ADCINLTRIM1-6 (offsets 0xE0-0xF4) during power up

Until the appropriate factory trim is loaded, the ADC and other analog modules are not specified to operate within the data sheet specifications. Similarly, if trim values other than the factory settings are placed into the trim registers, the ADC (and other modules) is not specified to operate within the data sheet specifications.

The boot ROM calls the calibration functions, so trim values are initially populated without user intervention. However, if the trims are cleared due to a module reset or modified for some other reason, then the user must call the calibration functions (defined in the C2000Ware header files).

7.5.2.13.1 ADC Zero Offset Calibration

ADC offset error is determined and calibrated during factory testing. However, the user still has the option to perform offset calibration if the end application specifically requires this. This section describes how to perform offset calibration using internal VREFLO connection for single-ended operation.

Zero offset error is defined as the difference from 0 that occurs when converting a voltage at VREFLO. The zero offset error can be positive or negative. To correct this error, an adjustment of equal magnitude and opposite polarity is written into the ADCOFFTRIMx register. The value contained in this register is applied before the results are available in the ADC result registers. This operation is fully contained within the ADC core, so the timing of the results is not affected, and the full dynamic range of the ADC is maintained for any trim value.

Note

Regardless of the converter resolution, the size of each ADCOFFTRIMx step is (VREFHI-VREFLO)/ 65536.

Use the following procedure to re-calibrate the ADC offset in 12-bit single-ended mode:

1. Set ADCOFFTRIMx to +112 steps (0x70). This adds an artificial offset to account for negative offset that can reside in the ADC core.
2. Perform some multiple of 16 conversions on VREFLO (internal connection), accumulating the results (for example, 32*16 conversions = 512 conversions). Use the maximum value of ACQPS to make sure longer settling time to account for parasitic impedance of internal VREFLO connections.
3. Divide the accumulated result by the multiple of 16 (for example, for 512 conversions, divide by 32).
4. Set ADCOFFTRIMx to 112 – result from step 3.

7.5.2.14 ADC Timings

The process of converting an analog voltage to a digital value is broken down into an S+H phase and a conversion phase. The ADC sample and hold circuits (S+H) are clocked by SYSCLK while the ADC conversion process is clocked by ADCCLK. ADCCLK is generated by dividing down SYSCLK based on the PRESCALE field in the ADCCTL2 register.

The S+H duration is the value of the ACQPS field of the SOC being converted, plus one, times the SYSCLK period. The user must make sure that this duration exceeds both 1 ADCCLK period and the minimum S+H duration specified in the data sheet. The conversion time is approximately 10.5 ADCCLK cycles 11.5 ADCCLK cycles. See the timing diagrams and tables in Section 7.5.2.14.1 for exact timings.

7.5.2.14.1 ADC Timing Diagrams

The following diagrams show the ADC conversion timings for two SOCs given the following assumptions:

- SOC0 and SOC1 are configured to use the same trigger.
- No other SOCs are converting or pending when the trigger occurs.
- The round robin pointer is in a state that causes SOC0 to convert first.
- ADCINTSEL is configured to set an ADCINT flag upon end of conversion for SOC0 (whether this flag propagates through to the CPU to cause an interrupt is determined by the configurations in the VIM module).

Table 7-114. ADC Timing Parameter Descriptions

Parameter	Description
t_{SH}	The duration of the S+H window. At the end of this window, the value on the S+H capacitor becomes the voltage to be converted into a digital value. The duration is given by (ACQPS + 1) SYSCLK cycles. ACQPS can be configured individually for each SOC, so t_{SH} is not necessarily the same for different SOCs. **Note:** The value on the S+H capacitor is captured approximately 5ns before the end of the S+H window regardless of device clock settings.
t_{LAT}	The time from the end of the S+H window until the ADC results latch in the ADCRESULTx register. If the ADCRESULTx register is read before this time, the previous conversion results are returned.
t_{EOC}	The time from the end of the S+H window until the S+H window for the next ADC conversion can begin. The subsequent sample can start before the conversion results are latched.
t_{INT}	The time from the end of the S+H window until an ADCINT flag is set (if configured). If the INTPULSEPOS bit in the ADCCTL1 register is set, t_{INT} coincides with the end of conversion (EOC) signal. If the INTPULSEPOS bit is 0, and the OFFSET field in the ADCINTCYCLE register is not 0, then there is a delay of OFFSET SYSCLK cycles before the ADCINT flag is set. This delay can be used to enter the ISR or trigger the DMA exactly when the sample is ready. If the INTPULSEPOS bit is 0, t_{INT} coincides with the end of the S+H window. If t_{INT} triggers a read of the ADC result register (directly through DMA or indirectly by triggering an ISR that reads the result), care must be taken to make sure the read occurs after the results latch (otherwise, the previous results are read).
t_{DMA}	The time from the end of the S+H window until a DMA read of the ADC conversion result is triggered, when ADCCTL1.TDMAEN = 1. If TDMAEN is set to 0, then the DMA trigger occurs at T_{INT}. In certain conditions, the ADCINT flag can be set before the ADCRESULT value is latched. To make sure that the DMA read occurs after the ADCRESULT value has been latched, write 1 to ADCCTL1.TDMAEN to enable DMA timings.

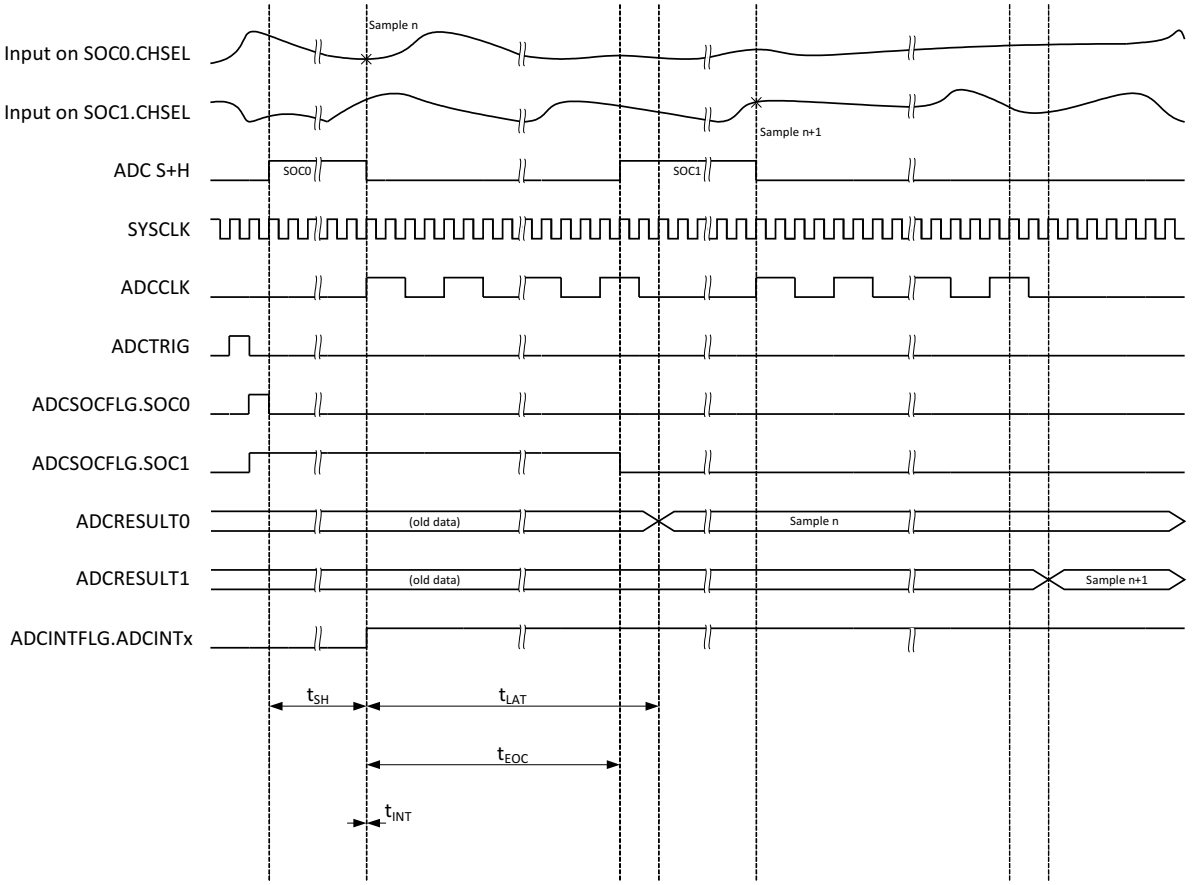


Figure 7-107. ADC Timings for 12-bit Mode in Early Interrupt Mode

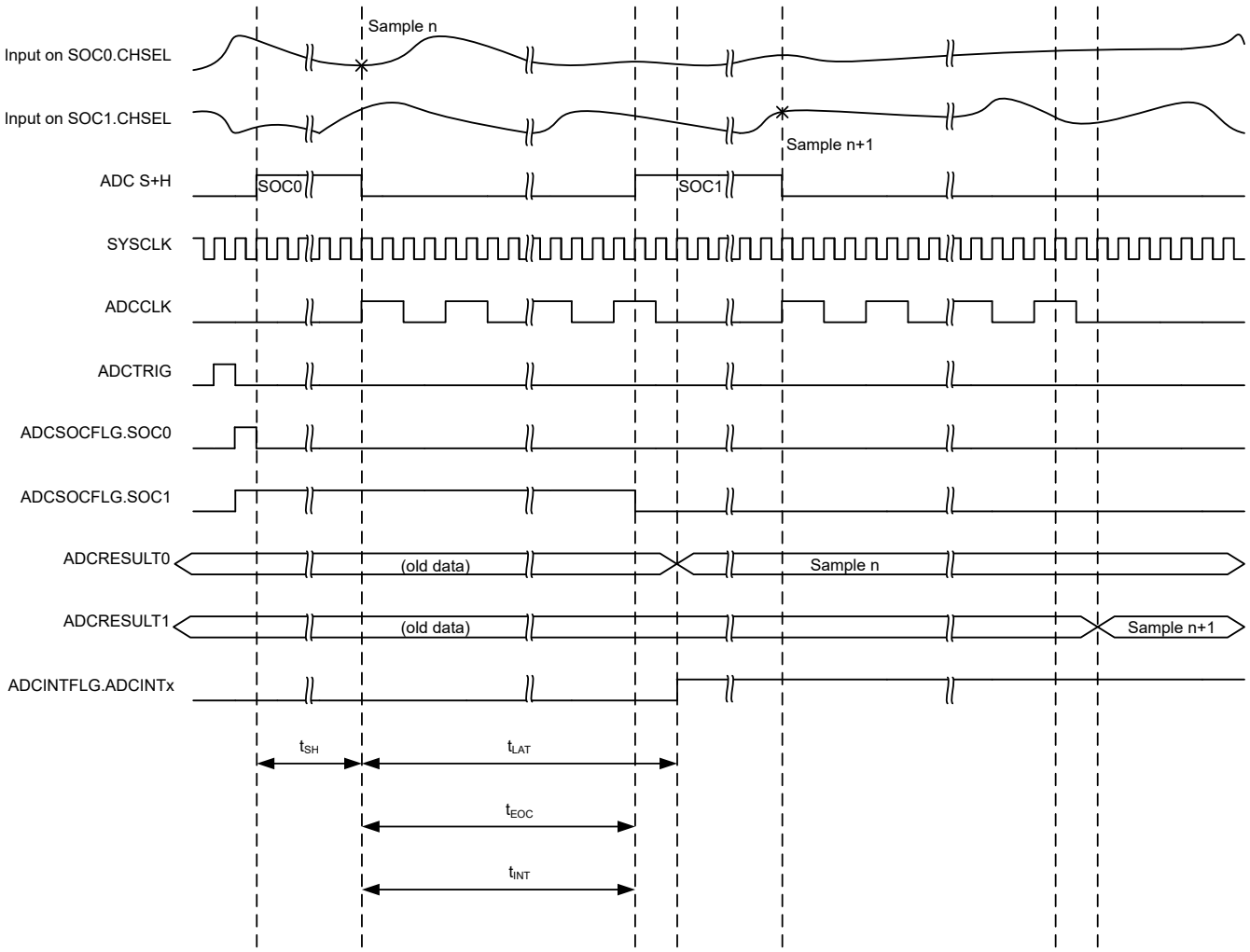


Figure 7-108. ADC Timings for 12-bit Mode in Late Interrupt Mode

Table 7-115. ADC Timings in 12-bit Mode with SAMPCAPRESETSEL = 1

ADCCLK Prescale		SYSCLK Cycles			
ADCCTL2. PRESCALE	Prescale Ratio	t_{EOC}	t_{LAT}	t_{EINT}	t_{LINT}
0	1	11	13	1	11
1	1.5	16	18	1	16
2	2	21	23	1	21
3	2.5	26	38	1	26
4	3	31	34	1	31
5	3.5	36	39	1	36
6	4	41	44	1	41
7	4.5	46	49	1	46
8	5	51	55	1	51
9	5.5	56	60	1	56
10	6	61	65	1	61
11	6.5	66	70	1	66
12	7	71	76	1	71
13	7.5	76	81	1	76

Table 7-115. ADC Timings in 12-bit Mode with SAMPCAPRESETSEL = 1 (continued)

ADCCLK Prescale		SYSCLK Cycles			
ADCCTL2. PRESCALE	Prescale Ratio	t_{EOC}	t_{LAT}	t_{EINT}	t_{LINT}
14	8	81	86	1	81
15	8.5	86	91	1	86

Table 7-116. ADC Timings in 16-bit Mode with SAMPCAPRESETSEL = 1

ADCCLK Prescale		SYSCLK Cycles			
ADCCTL2. PRESCALE	Prescale Ratio	t_{EOC}	t_{LAT}	t_{EINT}	t_{LINT}
0	1	31	32	1	31
1	1.5	45	47	1	45
2	2	60	61	1	60
3	2.5	75	75	1	75
4	3	90	91	1	90
5	3.5	104	106	1	104
6	4	119	120	1	119
7	4.5	134	134	1	134
8	5	149	150	1	149
9	5.5	163	165	1	163
10	6	178	179	1	178
11	6.5	193	193	1	193
12	7	208	209	1	208
13	7.5	222	224	1	222
14	8	237	238	1	237
15	8.5	252	252	1	252

7.5.2.15 Additional Information

The following sections contain additional practical information.

7.5.2.15.1 Ensuring Synchronous Operation

For best performance, all ADCs on the device must be operated synchronously. The device data sheet specifies the performance in both synchronous and asynchronous mode for those parameters which differ between the modes of operation.

To make sure synchronous operation, all ADCs on the device must operate in lockstep. This is accomplished by writing configurations to all ADCs that cause the sampling and conversion phases of all ADCs to be exactly aligned. The easiest way to accomplish this is to write identical values to the SOC configurations for each ADC for trigger select and ACQPS (S+H duration). In addition, synchronous ADCs must also configure identical values for the SOC priority control, burst mode, burst trigger, and burst size.

7.5.2.15.1.1 Basic Synchronous Operation

The following example configures two SOCs each on ADCA and ADCB with identical trigger select and ACQPS values. This results in synchronous operation between ADCA and ADCB. For devices with more than two ADCs, the same principles can be used to synchronize all the ADCs.

```
Example: Basic Synchronous Operation
AdcaRegs.ADCSOC0CTL.bit.CHSEL = 4;      //SOC0 converts ADCINA4
AdcaRegs.ADCSOC0CTL.bit.ACQPS = 19;     //SOC0 uses sample duration of 20 SYSCLK cycles
AdcaRegs.ADCSOC0CTL.bit.TRIGSEL = 10;   //SOC0 begins conversion on ePWM3 SOCB
AdcbRegs.ADCSOC0CTL.bit.CHSEL = 0;      //SOC0 converts ADCINB0
AdcbRegs.ADCSOC0CTL.bit.ACQPS = 19;     //SOC0 uses sample duration of 20 SYSCLK cycles
AdcbRegs.ADCSOC0CTL.bit.TRIGSEL = 10;   //SOC0 begins conversion on ePWM3 SOCB

AdcaRegs.ADCSOC1CTL.bit.CHSEL = 4;      //SOC1 converts ADCINA4
AdcaRegs.ADCSOC1CTL.bit.ACQPS = 30;     //SOC1 uses sample duration of 31 SYSCLK cycles
AdcaRegs.ADCSOC1CTL.bit.TRIGSEL = 10;   //SOC1 begins conversion on ePWM3 SOCB
AdcbRegs.ADCSOC1CTL.bit.CHSEL = 1;      //SOC1 converts ADCINB1
AdcbRegs.ADCSOC1CTL.bit.ACQPS = 30;     //SOC1 uses sample duration of 31 SYSCLK cycles
AdcbRegs.ADCSOC1CTL.bit.TRIGSEL = 10;   //SOC1 begins conversion on ePWM3 SOCB
```

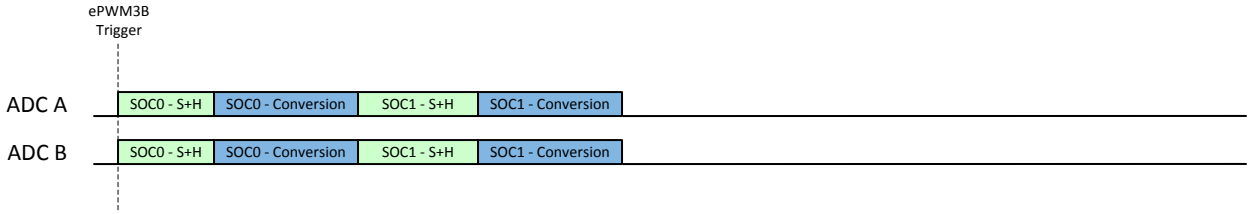


Figure 7-109. Example: Basic Synchronous Operation

Several things can be noted from Figure 7-109. First, while the ACQPS values must be the same for SOCs with the same number, different ACQPS values can be used for SOCs with different numbers. Because of this, synchronous operation does not require a single global S+H time, but instead only channels sampled simultaneously require identical S+H durations. Another important point from this example is that any channel select value can be used for any SOC. Finally, this example assumes round-robin operation. If high-priority SOCs are to be used, the priority must be configured the same on all ADCs.

7.5.2.15.1.2 Synchronous Operation with Multiple Trigger Sources

As long as each set of SOCs has identical trigger select and ACQPS settings, multiple trigger sources can be used while still achieving synchronous operation.

The following example demonstrates synchronous operation between ADCA and ADCB while using three SOCs and two trigger sources. Figure 7-110 demonstrates that any combination of relative trigger timings still results in synchronous operation.

```
Example: Synchronous Operation With Multiple Trigger Sources
AdcaRegs.ADCSOC0CTL.bit.CHSEL = 4;      //SOC0 converts ADCINA4
AdcaRegs.ADCSOC0CTL.bit.ACQPS = 19;     //SOC0 uses sample duration of 20 SYSCLK cycles
AdcaRegs.ADCSOC0CTL.bit.TRIGSEL = 10;   //SOC0 begins conversion on ePWM3 SOCB
AdcbRegs.ADCSOC0CTL.bit.CHSEL = 0;      //SOC0 converts ADCINB0
AdcbRegs.ADCSOC0CTL.bit.ACQPS = 19;     //SOC0 uses sample duration of 20 SYSCLK cycles
AdcbRegs.ADCSOC0CTL.bit.TRIGSEL = 10;   //SOC0 begins conversion on ePWM3 SOCB

AdcaRegs.ADCSOC1CTL.bit.CHSEL = 4;      //SOC1 converts ADCINA4
AdcaRegs.ADCSOC1CTL.bit.ACQPS = 30;     //SOC1 uses sample duration of 31 SYSCLK cycles
AdcaRegs.ADCSOC1CTL.bit.TRIGSEL = 10;   //SOC1 begins conversion on ePWM3 SOCB
AdcbRegs.ADCSOC1CTL.bit.CHSEL = 1;      //SOC1 converts ADCINB1
AdcbRegs.ADCSOC1CTL.bit.ACQPS = 30;     //SOC1 uses sample duration of 31 SYSCLK cycles
AdcbRegs.ADCSOC1CTL.bit.TRIGSEL = 10;   //SOC1 begins conversion on ePWM3 SOCB

AdcaRegs.ADCSOC2CTL.bit.CHSEL = 0;      //SOC2 converts ADCINA0
AdcaRegs.ADCSOC2CTL.bit.ACQPS = 19;     //SOC2 uses sample duration of 20 SYSCLK cycles
AdcaRegs.ADCSOC2CTL.bit.TRIGSEL = 2;    //SOC2 begins conversion on CPU Timer1
AdcbRegs.ADCSOC2CTL.bit.CHSEL = 2;      //SOC2 converts ADCINB2
AdcbRegs.ADCSOC2CTL.bit.ACQPS = 19;     //SOC2 uses sample duration of 20 SYSCLK cycles
AdcbRegs.ADCSOC2CTL.bit.TRIGSEL = 2;    //SOC2 begins conversion on CPU Timer1
```

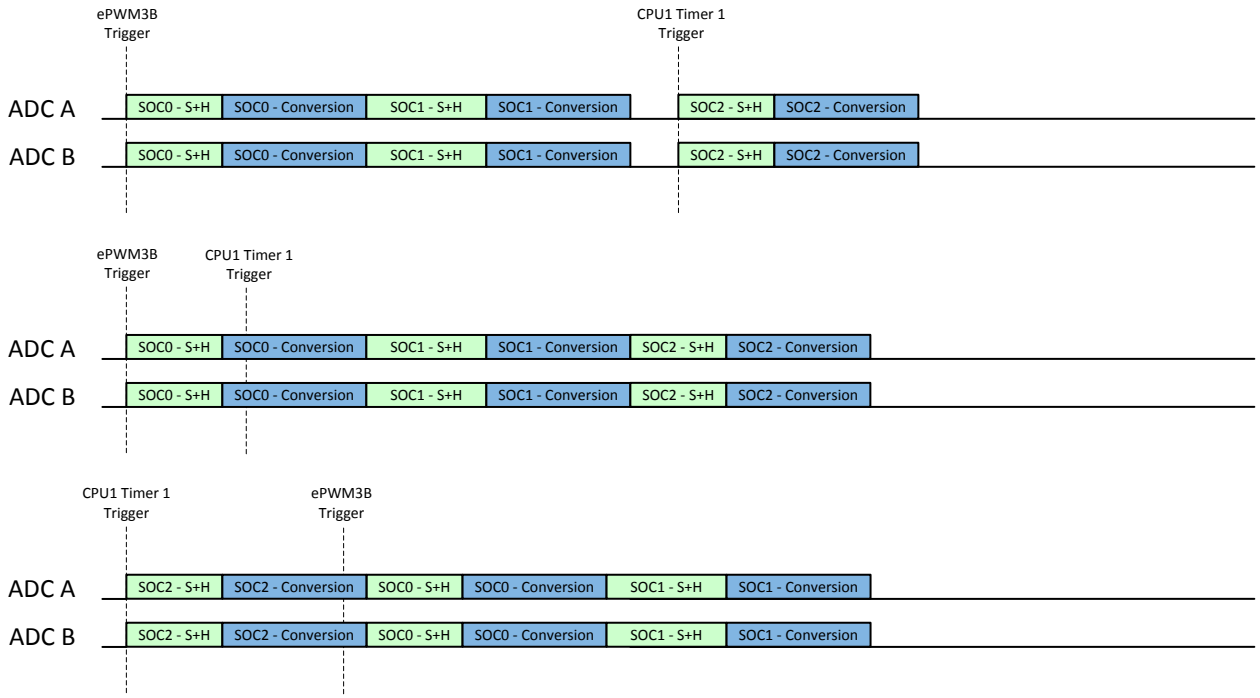


Figure 7-110. Example: Synchronous Operation with Multiple Trigger Sources

Note that any trigger source that can be selected in the TRIGSEL field can be used except for software triggering. There is no way to issue the software triggers for all ADCs simultaneously, so likely results in asynchronous operation. ADCINT1 or ADCINT2 can also be used as a trigger as long as the ADCINTSOCSEL1 and ADCINTSOCSEL2 registers are configured identically for all ADCs and software triggering is not used to start the chain of conversions.

7.5.2.15.1.3 Synchronous Operation with Uneven SOC Numbers

If only one trigger source is used, one ADC can use more SOCs than the other ADCs while still operating synchronously.

```
Example: Synchronous Operation With Uneven SOC Numbers
AdcaRegs.ADCSOC0CTL.bit.CHSEL = 4;      //SOC0 converts ADCINA4
AdcaRegs.ADCSOC0CTL.bit.ACQPS = 19;     //SOC0 uses sample duration of 20 SYSCLK cycles
AdcaRegs.ADCSOC0CTL.bit.TRIGSEL = 10;   //SOC0 begins conversion on ePWM3 SOCB
AdcbRegs.ADCSOC0CTL.bit.CHSEL = 0;      //SOC0 converts ADCINB0
AdcbRegs.ADCSOC0CTL.bit.ACQPS = 19;     //SOC0 uses sample duration of 20 SYSCLK cycles
AdcbRegs.ADCSOC0CTL.bit.TRIGSEL = 10;   //SOC0 begins conversion on ePWM3 SOCB

AdcaRegs.ADCSOC1CTL.bit.CHSEL = 4;      //SOC1 converts ADCINA4
AdcaRegs.ADCSOC1CTL.bit.ACQPS = 30;     //SOC1 uses sample duration of 31 SYSCLK cycles
AdcaRegs.ADCSOC1CTL.bit.TRIGSEL = 10;   //SOC1 begins conversion on ePWM3 SOCB
AdcbRegs.ADCSOC1CTL.bit.CHSEL = 1;      //SOC1 converts ADCINB1
AdcbRegs.ADCSOC1CTL.bit.ACQPS = 30;     //SOC1 uses sample duration of 31 SYSCLK cycles
AdcbRegs.ADCSOC1CTL.bit.TRIGSEL = 10;   //SOC1 begins conversion on ePWM3 SOCB

AdcaRegs.ADCSOC2CTL.bit.CHSEL = 0;      //SOC2 converts ADCINA0
AdcaRegs.ADCSOC2CTL.bit.ACQPS = 19;     //SOC2 uses sample duration of 20 SYSCLK cycles
AdcaRegs.ADCSOC2CTL.bit.TRIGSEL = 10;   //SOC2 begins conversion on ePWM3 SOCB
```

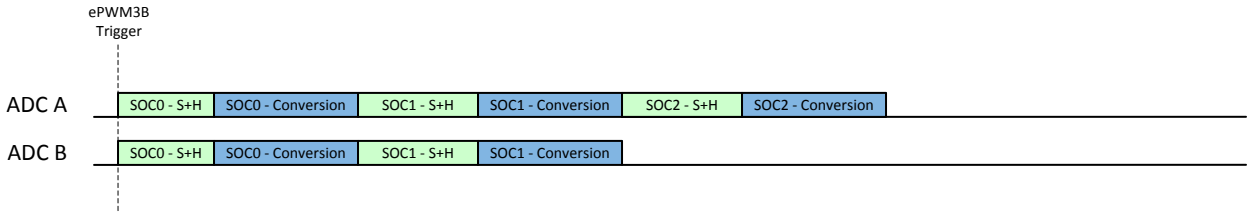


Figure 7-111. Example: Synchronous Operation with Uneven SOC Numbers

Note that if the trigger comes again before all SOCs have completed the conversions, ADCB begins converting immediately on SOC0 while ADCA does not start converting SOC0 again until SOC2 is complete. This results in asynchronous operation, so care must be taken to not overflow the trigger.

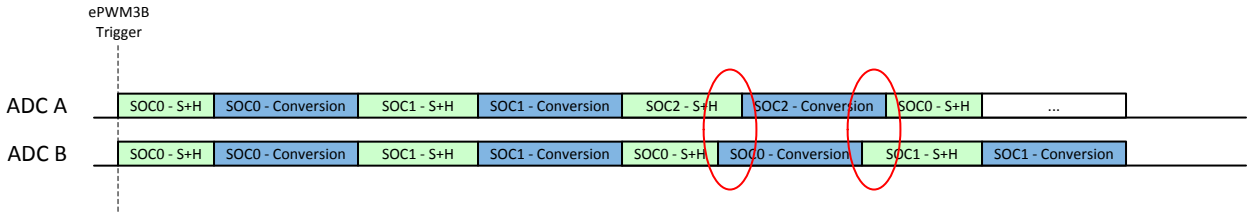


Figure 7-112. Example: Asynchronous Operation with Uneven SOC Numbers – Trigger Overflow

7.5.2.15.1.4 Non-overlapping Conversions

If conversion timings can be made sure to not overlap by the user, then it is not necessary to configure all SOCs identically on all ADCs to achieve performance equivalent to synchronous operation. For example, if the two ADC triggers in a system come from two ePWM sources that are always 180-degrees out-of-phase, then SOC0 can be used for both ADCA and ADCB with different trigger sources and different ACQPS values.

```
Example: Operation with Non-overlapping Conversions
//ePWM3 SOCA and SOCB are 180 degrees out of phase
AdcaRegs.ADCSOC0CTL.bit.CHSEL = 4;      //SOC0 converts ADCINA4
AdcaRegs.ADCSOC0CTL.bit.ACQPS = 19;     //SOC0 uses sample duration of 20 SYSCLK cycles
AdcaRegs.ADCSOC0CTL.bit.TRIGSEL = 10;   //SOC0 begins conversion on ePWM3 SOCB
AdcbRegs.ADCSOC0CTL.bit.CHSEL = 0;      //SOC0 converts ADCINB0
AdcbRegs.ADCSOC0CTL.bit.ACQPS = 19;     //SOC0 uses sample duration of 20 SYSCLK cycles
AdcbRegs.ADCSOC0CTL.bit.TRIGSEL = 9;    //SOC0 begins conversion on ePWM3 SOCA
```

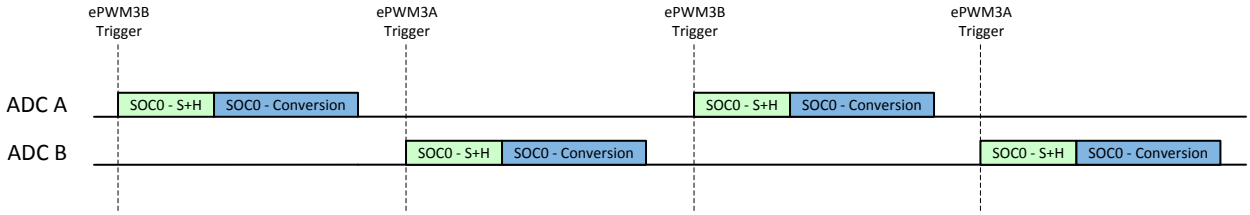


Figure 7-113. Example: Synchronous Equivalent Operation with Non-Overlapping Conversions

7.5.2.15.2 Choosing an Acquisition Window Duration

For correct operation, the input signal to the ADC must be allowed adequate time to charge the sample and hold capacitor, Ch. Typically, the S+H duration is chosen such that the sampling capacitor is charged to within ½ LSB or ¼ LSB of the final value, depending on the tolerable settling error.

The best methodology to determine the required settling time is to simulate the ADC and ADC driving circuits to make sure adequate settling performance. See *ADC Input Circuit Evaluation for C2000 MCUs* and *Charge-Sharing Driving Circuits for C2000 ADCs* for additional guidance on ADC signal conditioning circuit design and evaluation.

An approximation of the required settling time can also be determined using an RC settling model. The time constant for the model is given by the equation:

$$\tau = (R_S + R_{on}) \times C_h + R_S \times \left(C_S + C_p\right) \tag{5}$$

And the number of time constants needed is given by the equation:

$$k = ln\left(\frac{2^n}{settling\ error}\right) - ln\left(\frac{C_S + C_P}{CH}\right) \tag{6}$$

So the total S+H time must be set to at least:

$$t = k \cdot \tau \tag{7}$$

Where the following parameters are provided by the ADC input model in the device data sheet:

- n = ADC resolution (in bits)
- R_{ON} = ADC sampling switch resistance (provided in Ω)
- C_H = ADC sampling capacitor (provided in pF)
- C_p = ADC channel parasitic input capacitance (provided in pF)

And the following parameters are dependent on the application design:

- settling error = tolerable settling error (in LSBs)
- R_s = ADC driving circuit source impedance (typically in Ω or kΩ)
- C_S = capacitance on ADC input pin (typically in pF or nF)

For example, assuming the following parameters:

- n = 12-bits
- R_{ON} = 500Ω
- C_H = 12.5pF
- C_p = 12.7pF
- settling error = ¼ LSB
- R_s = 180Ω
- C_s = 150pF

The time constant is calculated as:

$$\tau = (180\Omega + 500\Omega) \times 12.5pF + 180\Omega \times (150pF + 12.7pF) = 37.8ns \quad (8)$$

And the number of required time constants is:

$$k = ln\left(\frac{2^{12}}{0.25}\right) - ln\left(\frac{150pF + 12.7pF}{12.5pF}\right) = 9.70 - 2.57 = 7.13 \quad (9)$$

So the S+H time must be set to at least: 37.8ns ˣ 7.13 = 270ns

If SYSCLK = , then each SYSCLK cycle is . S+H duration is 270ns/ = SYSCLK cycles, so ACQPS for this input is set to at least CEILING() – 1 = .

While this gives a rough estimate of the required acquisition window, a better method is to setup a circuit with the ADC input model, a model of the source impedance/capacitance, and any board parasitics in SPICE (or similar software) and simulate to verify that the sampling capacitor settles to the desired accuracy.

Note

The device data sheet specifies a minimum ADC S+H window duration. Do not use an ACQPS value that gives a duration less than this specification.

7.5.2.15.3 Achieving Simultaneous Sampling

While each ADC does not have dual S+H circuits, achieving simultaneous sampling is accomplished by setting the SOC triggers on two or more ADC modules to use the same trigger source. The following example demonstrates simultaneous sampling on ADCs based on an ePWM3 event. are sampled. An acquisition window of 20 SYSCLK cycles is used, but different durations are possible.

When the ePWM3 trigger is received, all ADCs begin converting in parallel immediately. All results are stored in the ADCRESULT0 register for each ADC. Note that this assumes that all ADCs are idle when the trigger is received. If one or more ADCs is busy, the samples do not happen at exactly the same time.

7.5.2.15.4 Result Register Mapping

The ADC results and the ADC PPB results are duplicated for each memory bus controller in the system. Bus controllers include all R5FSS core present on the specific part family and part number. For each bus controller, no access configuration is needed to allow read access to the result registers and no contention occurs in cases where multiple bus controllers try to read the ADC results simultaneously.

7.5.2.15.5 Internal Temperature Sensor

The internal temperature sensor measures the junction temperature of the device. The output of the sensor can be sampled with the ADC through an internal connection. This can be enabled on channel by setting the ENABLE bit in the TSNSCTL register.

7.5.2.15.6 Designing an External Reference Circuit

For best performance, the externally generated reference voltage must be buffered by a precision op-amp with good bandwidth and low output impedance before being driven into the ADC reference pin. A capacitor between the high and low reference pins must be placed on the PCB as close to the pins as practical to help absorb high-frequency currents. A series resistor (typically <1Ω) in series with this capacitor is sometimes necessary to maintain op-amp stability.

7.5.2.15.7 ADC-DAC Loopback Testing

For system diagnostic or functional safety purposes, the user application can perform a loopback test of the ADC module to verify that the ADC is converting correctly. Using the output of the DAC in the first CMPSS module, the device can be configured to supply a series of known voltages to the input of the converter, and the conversion result verified against expected results. Loopback test mode is enabled by setting the bit corresponding to the ADC module under test in the ADCLOOPBACK register in the analog subsystem module to 1. Figure 7-114 shows the connection between the CMPSS DAC output and the ADC.

Figure 7-114. CMPSS to ADC Loopback Connection

In ADC loopback test mode, the following special considerations apply:

- The ADC module always samples the CMPSS1 DACL output, regardless of what channel is selected in the ADCSOCxCTL.CHSEL field.
- The minimum sampling window size (ACQPS) when converting the DAC output is .
- The output resolution of the CMPSS DAC is 6 bits. The lower 6 bits of the input DACVAL are discarded.
- ADC loopback test mode affects CMPSS trip voltages. Avoid enabling ADC loopback mode during regular CMPSS operation.

For more information on the CMPSS module and how to configure the CMPSS DAC, see the *Comparator Subsystem (CMPSS)* chapter.

7.5.3 Resolver to Digital Converter

7.5.3.1 Overview

7.5.3.1.1 Principle of Operation

A **Resolver** is a type of rotary electrical transformer used for measuring degrees of rotation. The resolver is typically directly mounted to the shaft of an electric motor. A typical resolver consists of a rotary transformer (exciter winding) and two windings separated by 90 degrees on the stator.

Resolver-front-end systems take a PWM Signal from the Resolver to Digital converter subsystem and generate an excitation sinusoidal signal (typically 7 V_{rms}) which is applied on the excitation coil. Rotation of the motor modulates the excitation signal and causes a modulated sine and cosine output directly related to the mechanical angle of the rotor with respect to stator. This sine and cosine outputs are monitored on the coupled coils or outputs of the Resolver. Electrical Zero (EZ) of a Resolver is defined as the position of the rotor with respect to the stator at which there is minimum voltage amplitude across the Sine winding and the maximum voltage amplitude across the Cosine winding when the input winding is excited with the rated voltage.

The Resolver to Digital Converter(RDC) subsystem generates an excitation signal as a Sine modulated PWM signal and that signal interfaces to an external amplifier which extracts the Sine wave and excites the excitation coil of the resolver. The sine and cosine outputs from the coils of the resolver after proper voltage scaling using external amplifiers ,are input to the internal SAR ADCs of the Resolver to Digital converter subsystem. The Resolver to Digital converter subsystem reads the output of the ADCs, processes the incoming digital data, and estimates the angle of the rotor and the angular velocity. It also performs diagnostic checks.

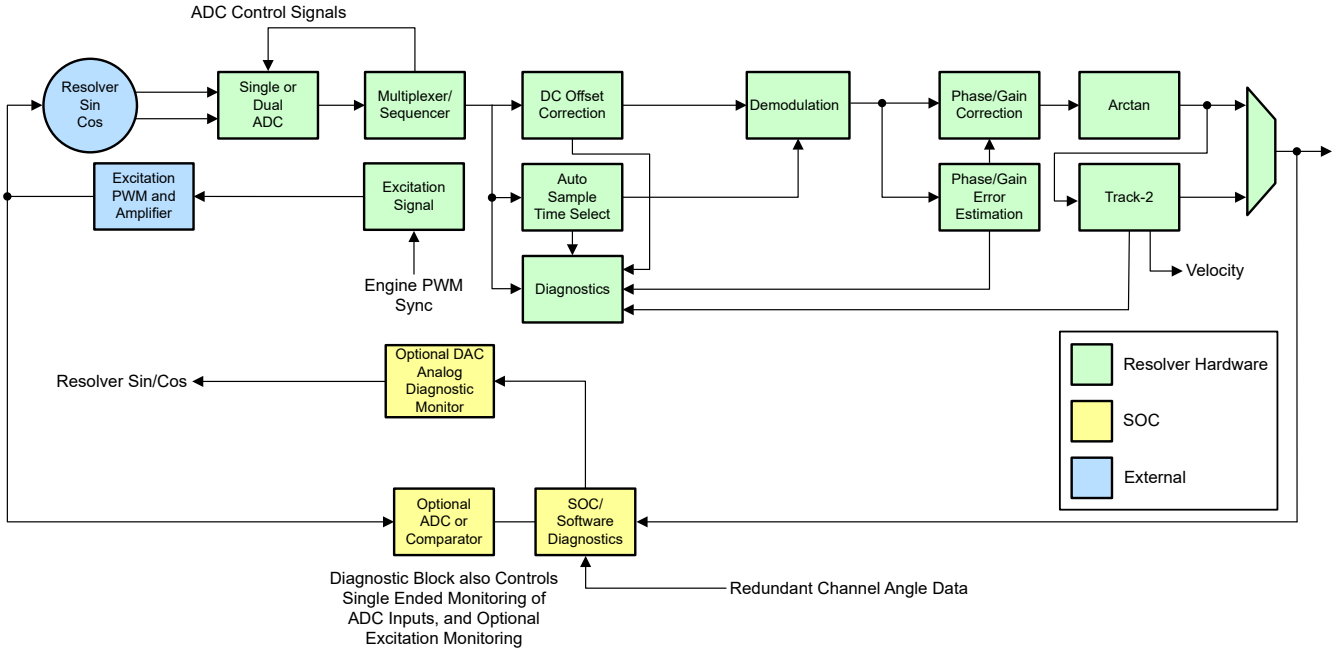


Figure 7-115. Block Diagram of the Resolver

7.5.3.1.2 Supported Features

- 2x Resolver modules with 12 bit SAR ADCs
  - 3.125 Msps max sampling rate
  - 50MHz clock to ADC
- 2x independent resolvers can be interfaced with one AM263P device, each with track2 and arctan function
- Software can read either arctan or track2 results for position/speed of each resolver
- Single ended and Differential modes support
- Flexible SAR ADC usage modes for sin/cos
  - 2x SAR ADCs sample one resolver channel: 1x ADC samples sine and 1x ADC samples cosine simultaneously
  - 1x SAR ADC samples one resolver channel: sine and cosine are sampled sequentially, then post processed to align the sampling phase. Other ADC can be used for general purpose SOC use

  - 2x SAR ADCs sample two resolver channels: One ADC samples the sine of one resolver, the other ADC samples the cosine of the same resolver simultenously. Next clock cycle both ADCs sample the sine and cosine signal of the other resolver channel enabling simultaneous sampling of each channel
- Fault detection and Glitch-filtering
- Auto offset, gain and phase correction
- Auto sampling phase optimization and phase delay compensation with track-2
- Programmable sample point count (to be averaged) at peak
- Auto PWM phase adjust
- FIR bandpass filter for improved resolution and offset elimination
- Bypass oversampling/FIR feature for fast but less accurate tracking with arctan
- Produce a RES0_PWMOUTx for the generation of one common excitation frequency signal with a pwm_sync_in to enable better EMC immunity
- Diagnostics signals for failures (if enabled)

7.5.3.1.3 Safety Features

- Safety SIL-3 and ASIL-D rating targeted
- Frequency imbalance: Compare zero crossing of sine with zero crossing of adjacent cosine
- Phase imbalance: Time delay between zero crossing of sine wrt cosine
- Gain and Offset mismatch
- Over and Under voltage sampling of Sine and Cosine Signals
- Individually programmable mask for all fault signals
- Programmable tracking error threshold and glitch timer
- Programmable phase, offset and gain error fault thresholds
- Redundant channel compare fault detection, (arctan to track-2 comparison or track2 to track2) with programmable threshold and glitch timer
- Window comparators to monitor analog input faults and excitation signal faults with programmable threshold and glitch settings and report type of input fault: Open, Short, Short to Supply, Short to Gnd

Note

Any error condition generates an event if enabled.

7.5.3.1.4 Performance Specification

- Sin/cos signal 3.3V, SAR: 12 bits with maximum sampling rate of 2.941176Msps
- Excitation Frequency selectable at 1KHz, 5KHz, 10KHz, 20KHz. 10 bits / 20Msps for motor max RPM
- Angle accuracy at ±0.1 degree
- Resolution linearity (±2 LSB) @13 bits
- Velocity accuracy (±5 LSB) @16 bits
- Rate(speed) accuracy (±0.03rps)
- Maximum 8000 rps with 20KHz Excitation frequency supported, in Mode 0 with one Resolver
- Conversion time < 5µS (Assuming constant speed)
- Settling time 1ms
- Static fault modes to be detected: Open, Short, Cross coupling
- Dynamic fault modes: Programmable Over Voltage and Under Voltage faults, Phase and Amplitude mismatch faults modes

7.5.3.1.5 Fault Detection Support

- Freq imbalance: Compare zero crossing of sine with zero crossing of adjacent cosine
- Phase imbalance: Time delay between zero crossing of sine with respect to cosine (calibrated)
- Gain mismatch (calibrated)
- Offset (calibrated)
- Over Voltage Fault Detection
- Under Voltage Fault Detection
- Redundancy
- Individual programmable mask for all fault signals
- Programmable tracking error threshold and glitch timer

- Programmable phase, offset and gain error fault thresholds
- Redundant channel compares fault detection, (arctan to track2 comparison) with programmable threshold and glitch timer
- Analog input faults with programmable threshold and glitch settings and detection of type of input faults: Open, Short, Short to supply, Short to GND
- Faults of excitation signal with programmable threshold and glitch settings and detection of type of input faults: Open, Short, Short to supply, Short to GND
- Excitation Frequency fault detection with respect to frequency for Sine and Cosine signals from Resolver

7.5.3.1.6 Programmer's Guide

Calibration must be done in a controlled setting with a simple constant rotation of resolver for the logic to tune the errors in the signal. After tuning, software can read the tuned values which were calculated, and use them in the future.

Before operating the Resolver-to-Digital-Converter (RDC), all the registers need to be set correctly. master_en must get cleared before software adjusts any MMRs. After the MMRs are set correctly, then software can set back master_en.

A resolver sensor needs an excitation signal to convert rotational information into a Sine Cosine modulated signal pair, which are then converted to digital angular information through the RDC. This excitation signal is generated by the internal RDC PWM generator which is then filtered and amplified by an external amplifier(outside AM263P). The output of the external amplifier drives the resolver coil. The sine and cosine outputs of the resolver are sampled, converted to digital by ADCs, and processed further by RDC.

Step-1 Power-On Diagnostics: Before initiating the PWM signal and activating the RDC, software is recommended to run diagnostic checks on the resolver ADCs:

There are two types of ADC dedicated diagnostic tests:

1. Open-Short tests: Resolver Cos and Sin coils can be connected to RDC ADC inputs through only passive components including pull/up and pull/down bias resistors or they can be connected through an amplifier. During power-on these connections can be checked through ADC internal Open Short Detection(OSD). Please refer to Open-Short section of ADC Chapter for more details.
2. Runtime diagnostic tests with 2 channels dedicated for diagnostics. While ADCs are not sampling the resolver inputs, they will sample two inputs to check full functionality of the ADCs for safety purpose.

Step-2 Sequencer: After diagnostic tests, the next step is to select the desired operating mode through sequencer settings in resolver_REGS_global_cfg[11:8]. Refer to Multiplexer/Sequencer Front End section for details. RDC subsystem can support one or two hardware Resolvers as described.

Step-3 Excitation signal: Next step is to initialize the excitation signal. The resolver sensor will need a sinusoidal excitation signal with a programmable frequency(using resolver_REGS_excit_sample_cfg1[7:0]) that will be in sync with the ADC sampling. This signal will be a PWM signal, filtered and amplified externally. Before RDC processes incoming data, enough time needs to be allocated for the external PWM to analog signal conversion to settle to its operating frequency. A typical excitation signal PWM filter amplifier(ALM2403-Q1) is shown in Filter and Amplifier Circuit figure and its startup time can be referred for typical settling time of a filter amplifier in datasheet of ALM2403-Q1()

Step-4 Oversample Ratio, Bandpass filter and/or Offset Correction: After excitation frequency settles, resolver inputs will be valid, and calibration and decimation process can be started by software. Each period of excitation frequency is over-sampled by a programmable ratio(using resolver_REGS_excit_sample_cfg1[15:8]). Default over-sample ratio is 20. This oversamplling enables to use a bandpass filter centered around excitation frequency by providing enough bandwidth. Oversampling also enables offset correction to settle faster and more accurately and also to detect the ideal decimation point.

Software needs to decide bandpass filter to be enabled or disabled(using resolver_REGS_dc_off_cfg1_0[8]). Enabling bandpass filter will introduce phase delay, but it will significantly improve noise rejection. This bandpass filter will also reject DC offset. If bandpass filter is enabled, DC offset correction can be disabled. Enabling DC offset correction when bpass filter is enabled will not degrade or improve signal. Regardless of DC correction being enabled, DC offset estimation always runs and that way DC offset monitoring can monitor faults. **Note: Bandpass filter is only designed for oversample ratio-20.**

If bandpass filter is disabled, it is recommended to enable DC offset correction. Note that if there is no valid excitation signal, DC offset correction will saturate the input signal to the RDC. This condition is monitored in fault detection modes. At this point DC offset fault detection and excitation signal monitor fault detection modes need to be enabled. Refer to above sections to program bandpass filter and offset correction.

Note

If bandpass filter is disabled, and offset correction needs to be enabled, in this case, offset correction needs to be enabled after ideal sample time selection converges. As explained in next step, ideal sample time checks the peaks to decide optimum sampling point. Although any noise will be averaged, offset correction may introduce false peaks, initially reducing the accuracy.

Step-5 Ideal Sample time selection, and decimation: In order to demodulate the rotation signal, resolver needs to sample the input signal during the peak point of the excitation signal. There are multiple considerations for this:

1. Ideal sample time selection block oversamples the input signal by 20 and decides the ideal sampling point.
2. Motor pwm currents are sampled by SOC ADCs. It would be a good practice to align sampling of the resolver signal, with the sampling of the corresponding motor PWM current. This improves the motor control loop by eliminating the latency. A synchronization pulse coming from motor PWM block which should to be used to synchronize resolver ADC sampling time.
3. To demodulate the signal, the peak of excitation signal needs to be sampled. Additionally, software can also enable sampling the negative peak of excitation frequency to improve settling. RDC finds the negative peak, and takes care of the sign automatically as explained in Section .
4. If DC offset correction is needed, it needs to be enabled after auto-ideal time selection.

Once the ideal sample time selection is done and configured as manual value, the DC offset correction can be enabled. This will prevent the DC offset correction from interfering with the ideal sample selection algorithm.

Step-6 Differential Phase and Gain Mismatch Correction: Ideally sin and cos signals should have a perfect phase delay of 90 degrees, and their amplitudes should match. If there's a common phase delay, this can be handled by the factory calibration by resetting the 0 deg position, and any minor common gain error will cancel out during arctan calculation. Gross common gain errors will be detected by fault detection mechanisms as explained in fault detection section. In real applications, there will be both differential phase and gain mismatch. Phase and Gain Calibration needs to be done after ideal sample time selection has settled. This is done by enabling the estimation, reading the estimated values and replacing them with manual values. The estimated values can be helpful in the diagnostics.

Step-7 Arctan and Track2 outputs: Output of phase and gain correction goes to arctan block. The arctan data feeds to Track-2 loop. Outputs from the Arctan, or the Track-2 can be read directly from the registers.

CAUTION

Please note that there is a hardware limitation on Arctan offset and hardware track2 velocity resolver_REGS_velocity_track2_0[31:0] sampling. Thus a Software track2 can be used in place of hardware track2 . The resolver_angle_speed from the SDK implements the Software track2 for velocity sampling.

After enough time is allowed for excitation signal to stabilize, ideal sampling time can be calculated, and offset correction can be enabled. They work independently, so they can be enabled at the same time, or sequentially.

RDC is designed to meet stringent ASIL-D requirements, and following diagnostic features are supported, they are classified under three main groups: Degradation of Signal (DOS), Loss of Signal (LOS), Loss of Track (LOT):

1. Monitor Sin or Cos offset drift (DOS)
2. Monitor Sin or Cos gain drift (DOS)
3. Monitor phase drift (phase drift between sin and cos channels) (DOS)
4. Monitor excitation frequency degradation or loss (DOS)

5. Monitor signal integrity by observing Sin and Cos zero crossings (DOS)
6. Monitor signal integrity by checking Sin^2+Cos^2=Constant (DOS)
7. Monitor Sin or Cos saturation or very high amplitude (DOS)
8. Monitor weak Sin or Cos signal below a threshold (LOS)
9. Monitor track2 loop locking to incoming angle (LOT)
10. Monitor ADC health through calibration channels (DOS)

Software SDK offers the building blocks to read, check, enable interrupts from the diagnostics data. Steps for further diagnostics programming will be included in a later TRM release.

7.5.3.2 Integration

7.5.3.2.1 Resolver to Digital Convertor Integration

There are 2x Resolver-to-Digital Converter (RDC) modules integrated in the device as shown in Figure 7-116.

Note

For each RDC Module:

- Resolver Module input channels have ADC_R0_AIN[3:0] and ADC_R1_AIN[3:0] dedicated pins.
- ADC_VREFLO_G3 and ADC_VREFHI_G3 are dedicated voltage reference pins for Resolver modules.
- Refer to Figure 7-117 for full list of all RDC pins.

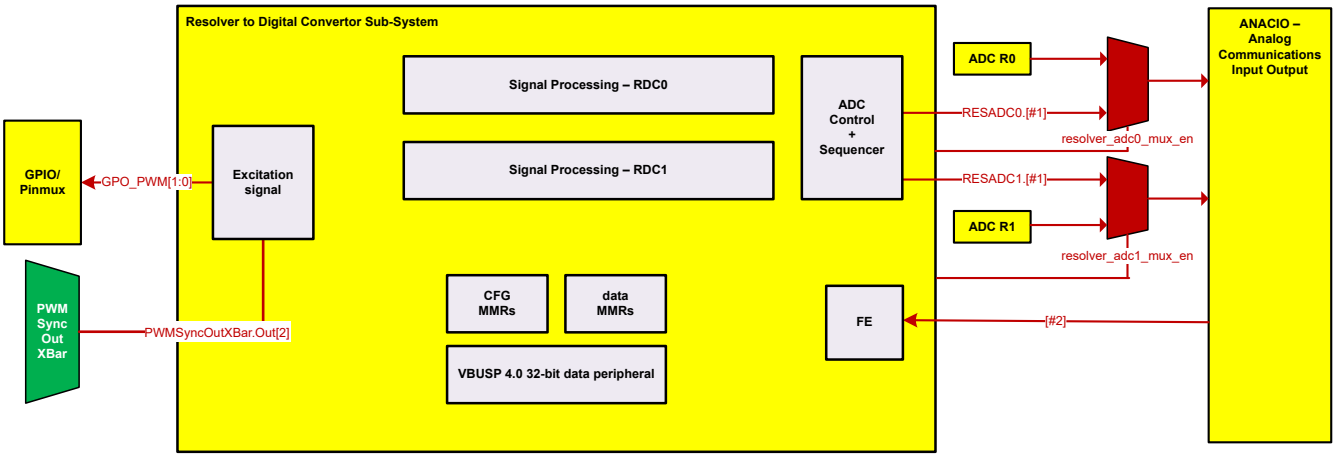


Figure 7-116. RDC Integration Diagram

7.5.3.2.2 Functional Description

The RDC signal processing uses an analog front end (SAR ADC) and Digital Processing unit called RDC Core. Each RDC Core is identical as shown in Figure 7-117.

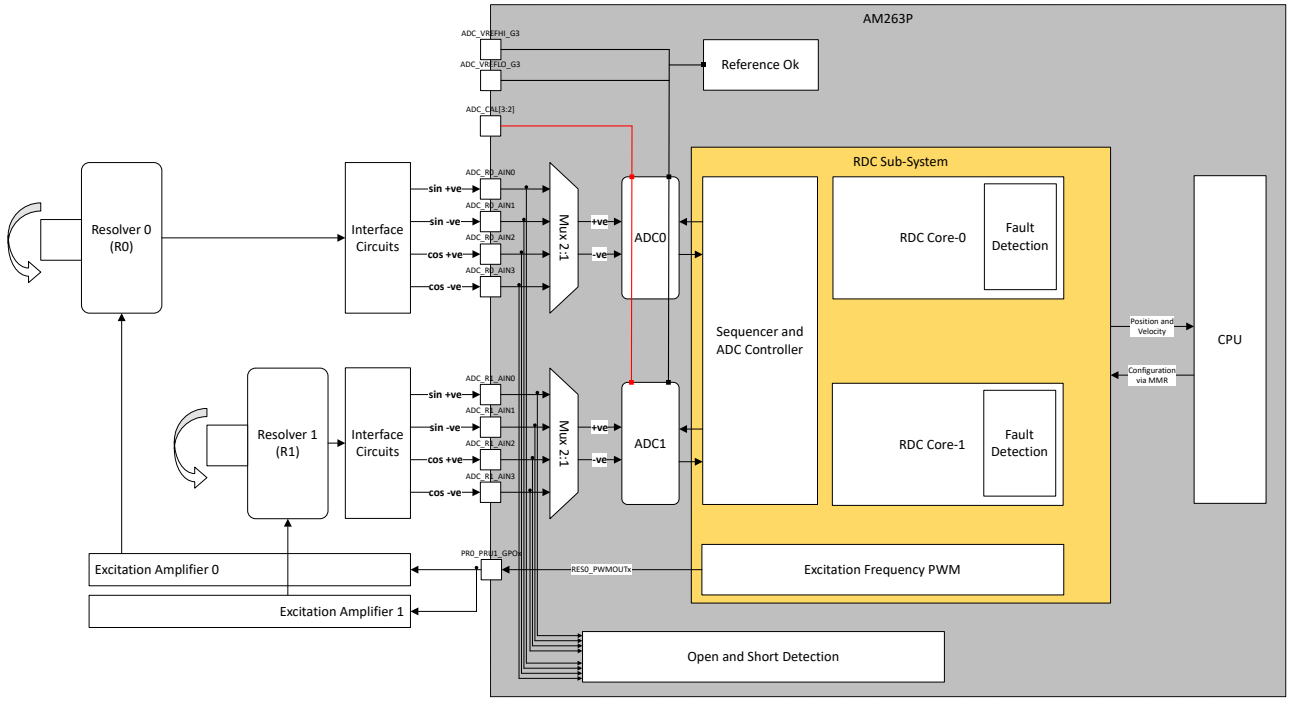


Figure 7-117. Resolver-RDC System Block Diagram

7.5.3.2.2.1 Resolver to Digital Converter(RDC) Sub System

Resolver to Digital Converter(RDC) contains the signal processing blocks to extract the position and velocity from the sin cos inputs from the external hardware Resolver connected to the motor mechanically. As shown on Figure 7-115and Figure 7-117, there are two identical blocks, termed as RDC Core-0 and RDC-Core1, on the SOC.

7.5.3.2.2.1.1 Sequencer and RDC Modes of Operation

The front end is responsible for latching, demuxing, and optional averaging the sin/cos inputs before it is presented to the RDC modules.

Table 7-117. Resolver Sampling Modes

Mode	External Resolvers connected	ADCs used	RDC Cores used	Sample Sequence _sX indicates Xth sample	Note
0	Resolver0(R0)	ADCR0 and ADCR1	RDC Core-0	R0sin_ADCR0_s0 + R0cos_ADCR1_s0	Sampling of R0's sin and cos signals concurrently across the ADCR0 and ADCR1. This is the Default mode of operation.
1	Resolver0(R0)	ADCR0	RDC Core-0	R0sin_ADCR0_s0 -> R0cos_ADCR0_s0	Sampling of R0's sin and cos signals in a staggered manner by ADCR0.
2	Resolver0(R0)	ADCR0	RDC Core-0	R0sin_ADCR0_s0 -> R0cos_ADCR0_s0 -> R0cos_ADCR0_s1 -> R0sin_ADCR0_s1 ->	Sampling of R0's sin and cos signals in a staggered manner by ADCR0. The 2 samples of sin are averaged and 2 samples of cos are averaged.Averaging enabled
3	Resolver0(R0) and Resolver1(R1)	ADCR0 and ADCR1	RDC Core-0 and RDC Core-1	R0sin_ADCR0_s0 + R0cos_ADCR1_s0 -> R1sin_ADCR0_s0 + R1cos_ADCR1_s0 ->	Sampling of R0's sin and cos signals concurrently across ADCR0 and ADCR1. Followed by the sampling of R1's sin and cos signals concurrently across ADCR0 and ADCR1. So, the delay is staggered between the 2 resolvers.
4	Resolver0(R0) and Resolver1(R1)	ADCR0 and ADCR1	RDC Core-0 and RDC Core-1	R0sin_ADCR0_s0 + R1sin_ADCR1_s0 -> R0cos_ADCR0_s0 + R1cos_ADCR1_s0 ->	Sampling of sin signals of both R0 and R1 concurrently across ADCR0 and ADCR1. Followed by the sampling of cos signals of both R0 and R1 concurrently across ADCR0 and ADCR1. So, the delay is staggered between the sin and cos signals.
5	Resolver0(R0) and Resolver1(R1)	ADCR0 and ADCR1	RDC Core-0 and RDC Core-1	R0sin_ADCR0_s0 + R1sin_ADCR1_s0 -> R0cos_ADCR0_s0 + R1cos_ADCR1_s0 -> R0cos_ADCR0_s1 + R1cos_ADCR1_s1 -> R0sin_ADCR0_s1 + R1sin_ADCR1_s1 ->	Sampling of sin signals of both R0 and R1 concurrently across ADCR0 and ADCR1. Followed by the sampling of cos signals of both R0 and R1 concurrently across ADCR0 and ADCR1. Followed by the sampling of cos signals of both R0 and R1 concurrently across ADCR0 and ADCR1. Followed by the sampling of sin signals of both R0 and R1 concurrently across ADCR0 and ADCR1. The 2 samples of sin are averaged and 2 samples of cos are averaged for the respective Resolvers.Averaging enabled So, the delay is staggered between the sin and cos signals.

The Resolver modes from above are explained using the below diagrams:

7.5.3.2.2.1.1.1 Mode 0

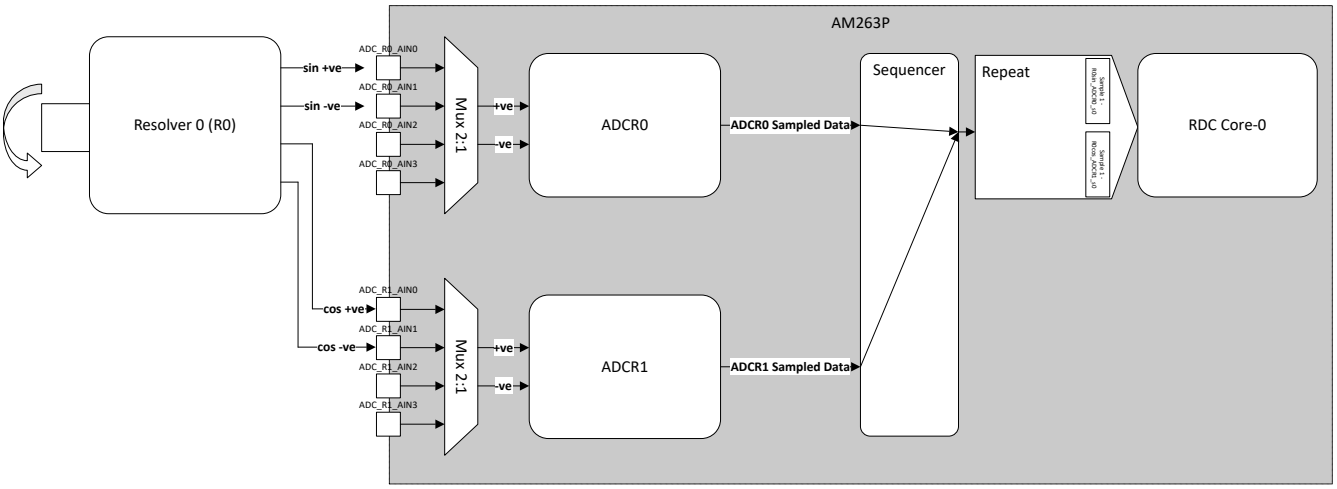


Figure 7-118. Mode -0

Resolver to Digital Converter(RDCs) used : RDC Core-0

Number of External Resolvers connected : 1

In Mode 0 as shown in Figure 7-118, the Resolver 0's sin and cos signals are concurrently sampled using ADCR0 and ADCR1.

This is the Default mode of operation.

7.5.3.2.2.1.1.2 Mode 1

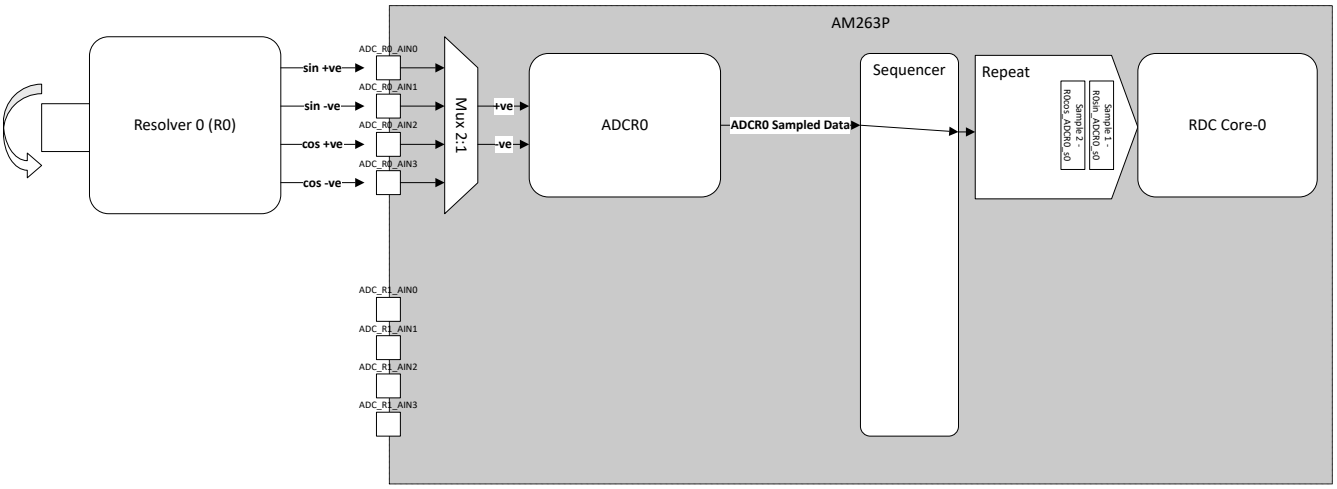


Figure 7-119. Mode -1

Resolver to Digital Converter(RDCs) used : RDC Core-0

Number of External Resolvers connected : 1

In Mode 1 as shown in Figure 7-119, the Resolver 0's sin and cos signals are sampled in a staggered manner using ADCR0.

7.5.3.2.2.1.1.3 Mode 2

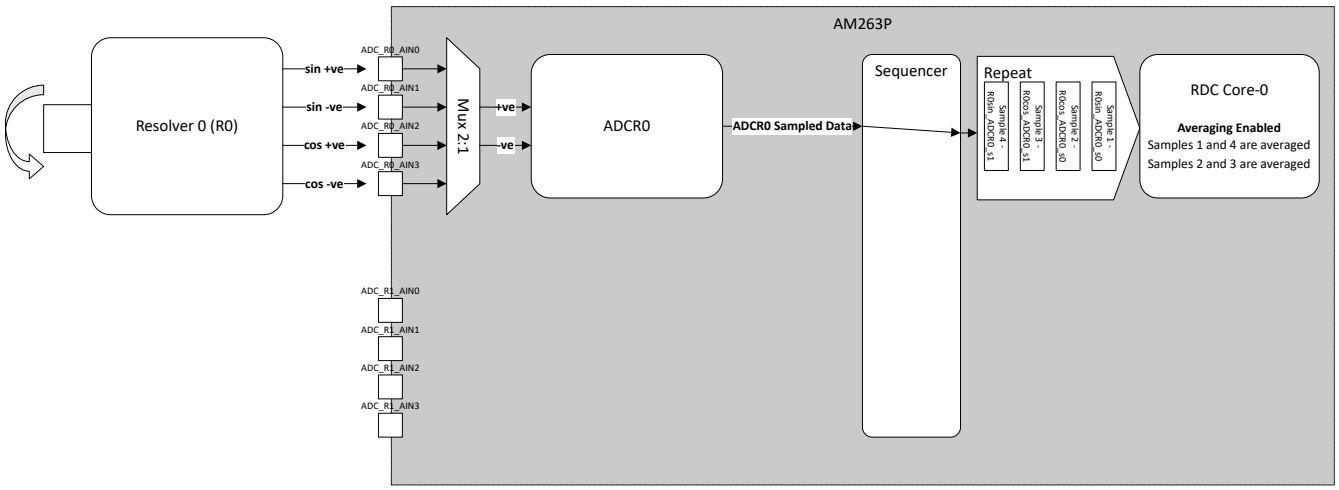


Figure 7-120. Mode -2

Resolver to Digital Converter(RDCs) used : RDC Core-0

Number of External Resolvers connected : 1

In Mode 2 as shown in Figure 7-120, the Resolver 0's sin and cos signals are sampled in a staggered manner using ADCR0 similar to Mode-1, but here 2 samples of sin and 2 samples of cos are averaged.

Averaging is enabled.

7.5.3.2.2.1.1.4 Mode 3

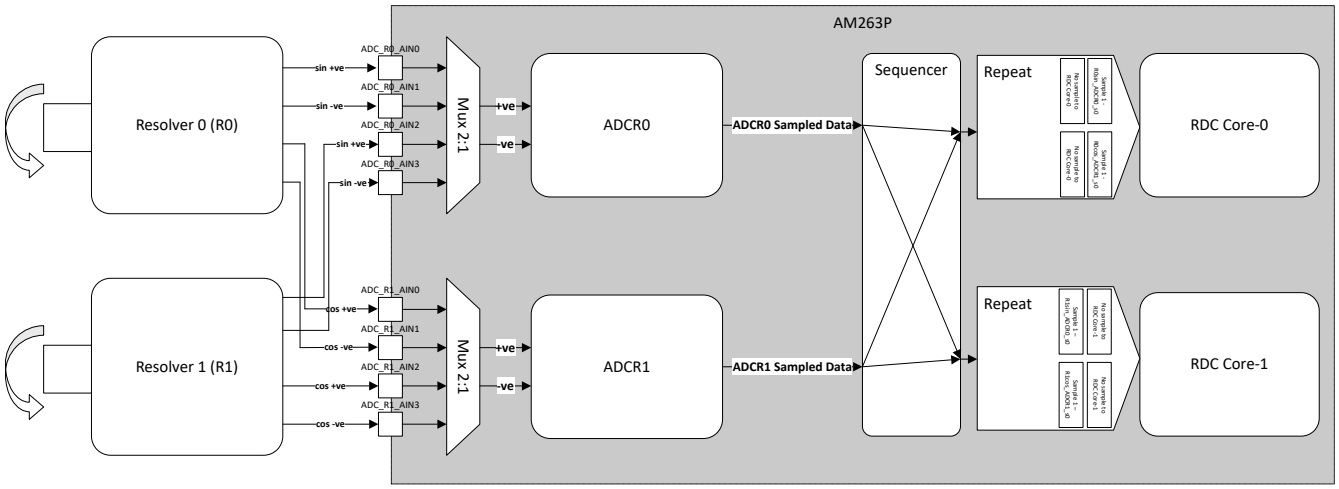


Figure 7-121. Mode -3

Resolver to Digital Converter(RDCs) used : RDC Core-0 and RDC Core-1

Number of External Resolvers connected : 2

In Mode 3 as shown in Figure 7-121,

1. First the Resolver 0's sin and cos signals are concurrently sampled using ADCR0 and ADCR1.
2. Next, the Resolver 1's sin and cos signals are concurrently sampled using ADCR0 and ADCR1

7.5.3.2.2.1.1.5 Mode 4

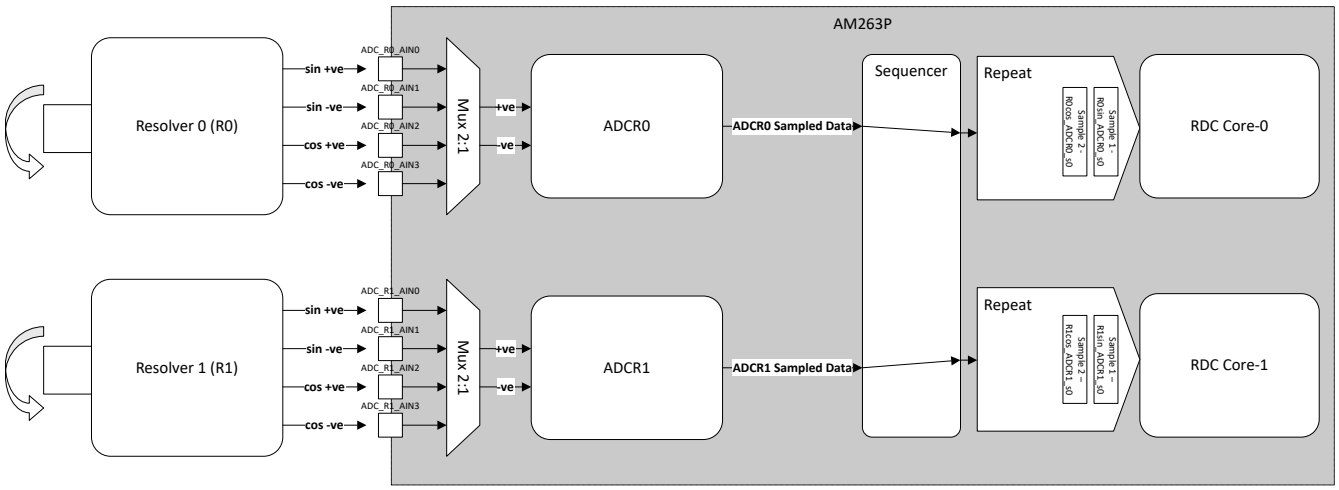


Figure 7-122. Mode -4

Resolver to Digital Converter(RDCs) used : RDC Core-0 and RDC Core-1

Number of External Resolvers connected :2

In Mode 4 as shown in Figure 7-122,

1. First sin signals of both Resolver 0 and Resolver 1 are concurrently sampled across ADCR0 and ADCR1.
2. Next, cos signals of both Resolver 0 and Resolver 1 are concurrently sampled across ADCR0 and ADCR1

7.5.3.2.2.1.1.6 Mode 5

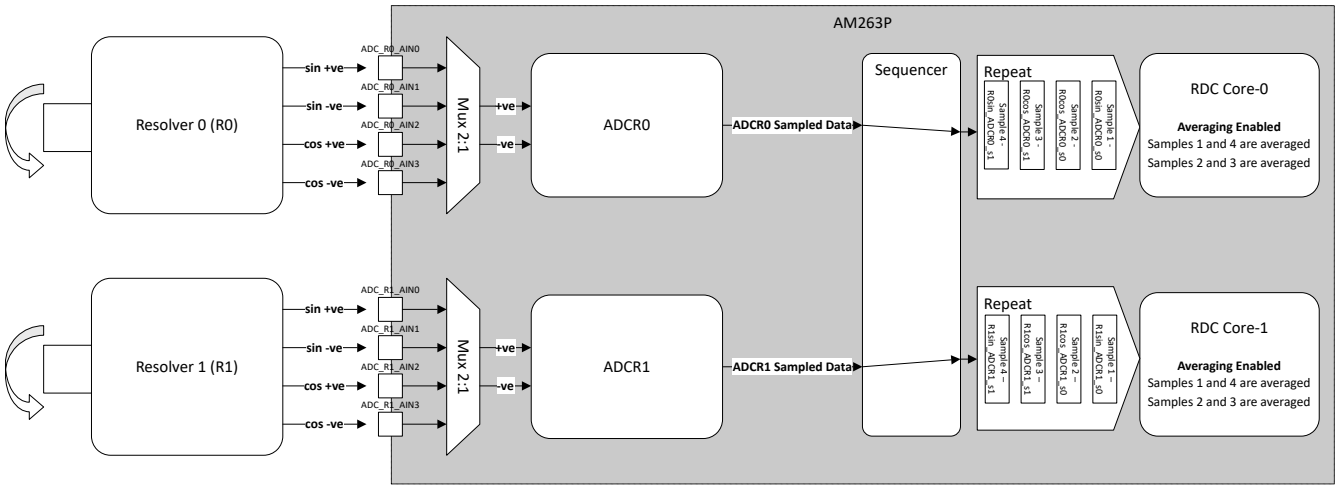


Figure 7-123. Mode -5

Resolver to Digital Converter(RDCs) used : RDC Core-0 and RDC Core-1

Number of External Resolvers connected : 2

In Mode 5 as shown in Figure 7-123,

1. First sin signals of both Resolver 0 and Resolver 1 are concurrently sampled across ADCR0 and ADCR1.
2. Next, cos signals of both Resolver 0 and Resolver 1 are concurrently sampled across ADCR0 and ADCR1.
3. Next, cos signals of both Resolver 0 and Resolver 1 are concurrently sampled across ADCR0 and ADCR1.
4. Next, sin signals of both Resolver 0 and Resolver 1 are concurrently sampled across ADCR0 and ADCR1.
5. The 2 samples of sin are averaged and 2 samples of cos are averaged, for the respective Resolvers.

Averaging is enabled, thus the delay is staggered between the sin and cos signals.

7.5.3.2.2.1.2 Excitation Signal and PWM

The coil of a Resolver needs an excitation signal which gets modulated as per the shaft rotation. This excitation signal, modulated as per the rotation of the shaft is then output as the sine and cosine outputs of the resolver.

To generate the excitation signal, the Resolver sub-system in AM263P generates a Sinusoidal PWM signal(RESx_PWMOUTx) where the PWM's duty cycle is modulated as per the Sine excitation signal. This PWM signal needs to be passed through an external low pass filter to obtain the sine excitation signal. Refer Figure 7-125 below, showing ALM2403 application for example.

The excitation signal is a sine wave with a programmable frequencies of 1KHz, 5KHz, 10KHz, and 20KHz with programmable phase of 0 to 360°. The Resolver Sub-System outputs this Sinusoidal PWM as RESx_PWMOUTx on Mux Mode 8 of AM263P PR0_PRU1_GPIO8, PR0_PRU1_GPIO10, PR0_PRU1_GPIO13, PR0_PRU1_GPIO14 ball pins.

Note

One single excitation signal is generated even if two resolvers are working. It is recommended to use two separate amplifiers for two external Resolvers.

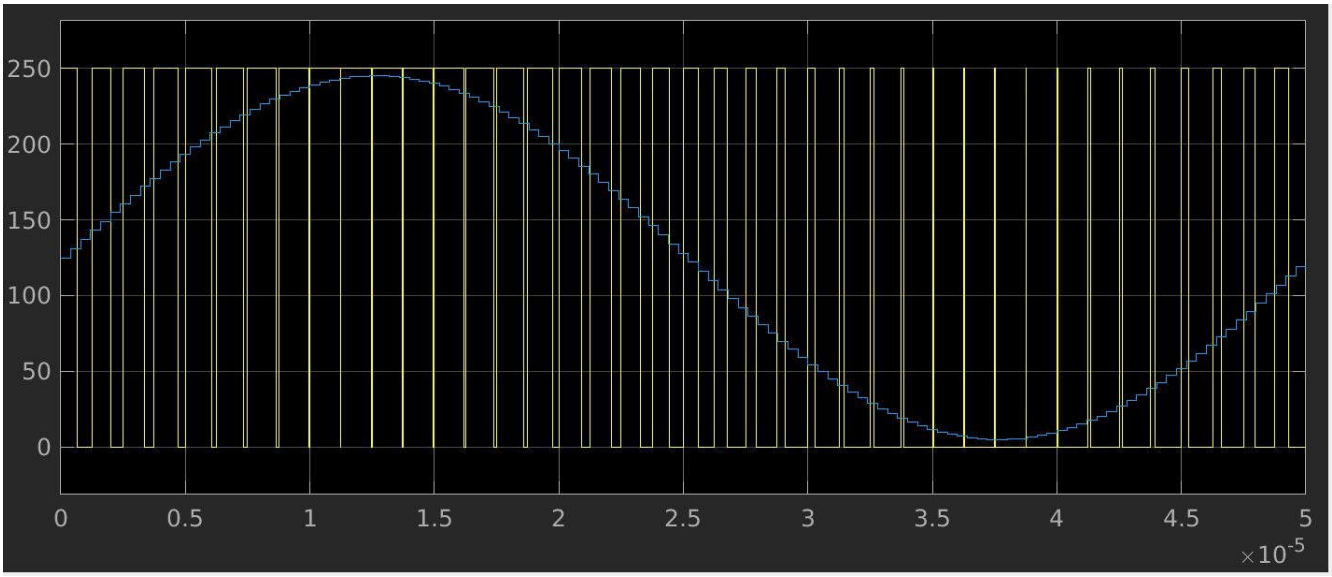


Figure 7-124. Excitation Signal Generation through PWM (Example for 20KHz)

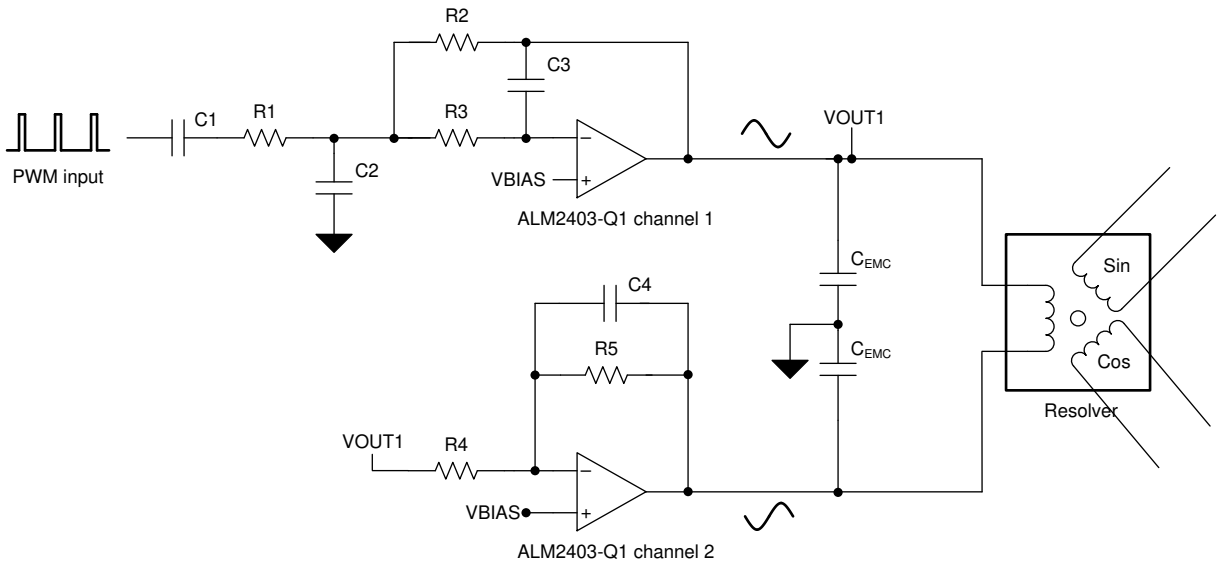


Figure 7-125. Filter and Amplifier Circuit Converting the RESx_PWMOUTx Signal to an Analog sine Wave Driving the Resolver Excitation Coil

The PWM base frequency must meet the requirements of external excitation amplifier's second order low pass filter, and must be at least 32 times the maximum supported frequency. Using a 800KHz PWM frequency meets that requirement with it being 40 times the highest excitation frequency of 20KHz. RDC can tolerate a second or third harmonic for the excitation signal up to 10% at the final resolver output.

The PWM generation circuit is shown below in Figure 7-126.

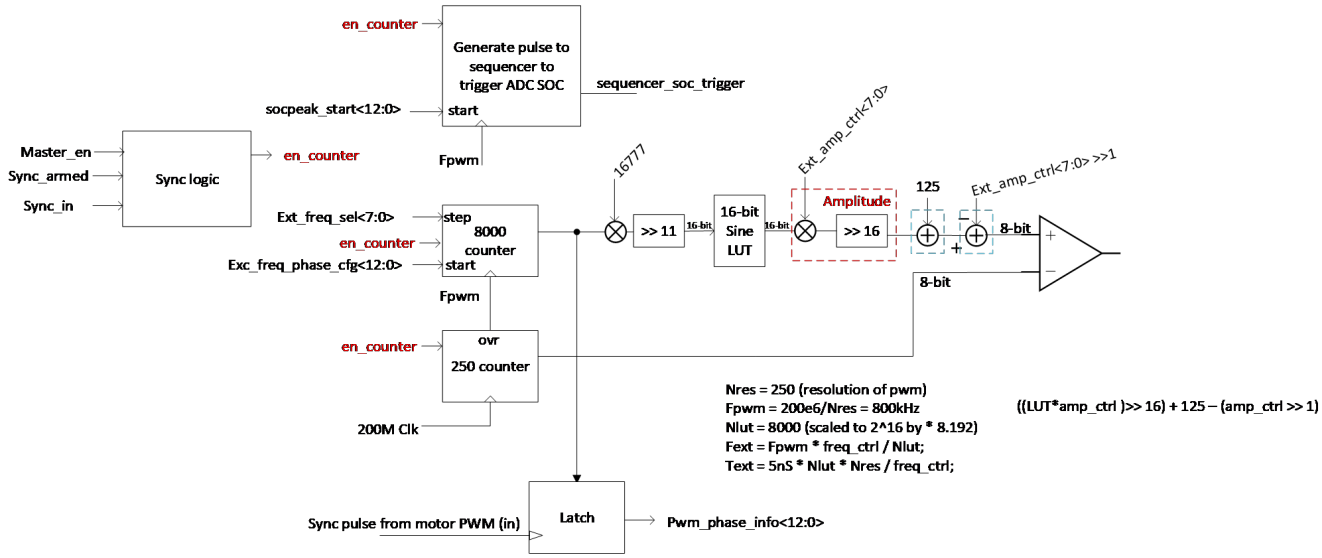


Figure 7-126. PWM Generation Circuit

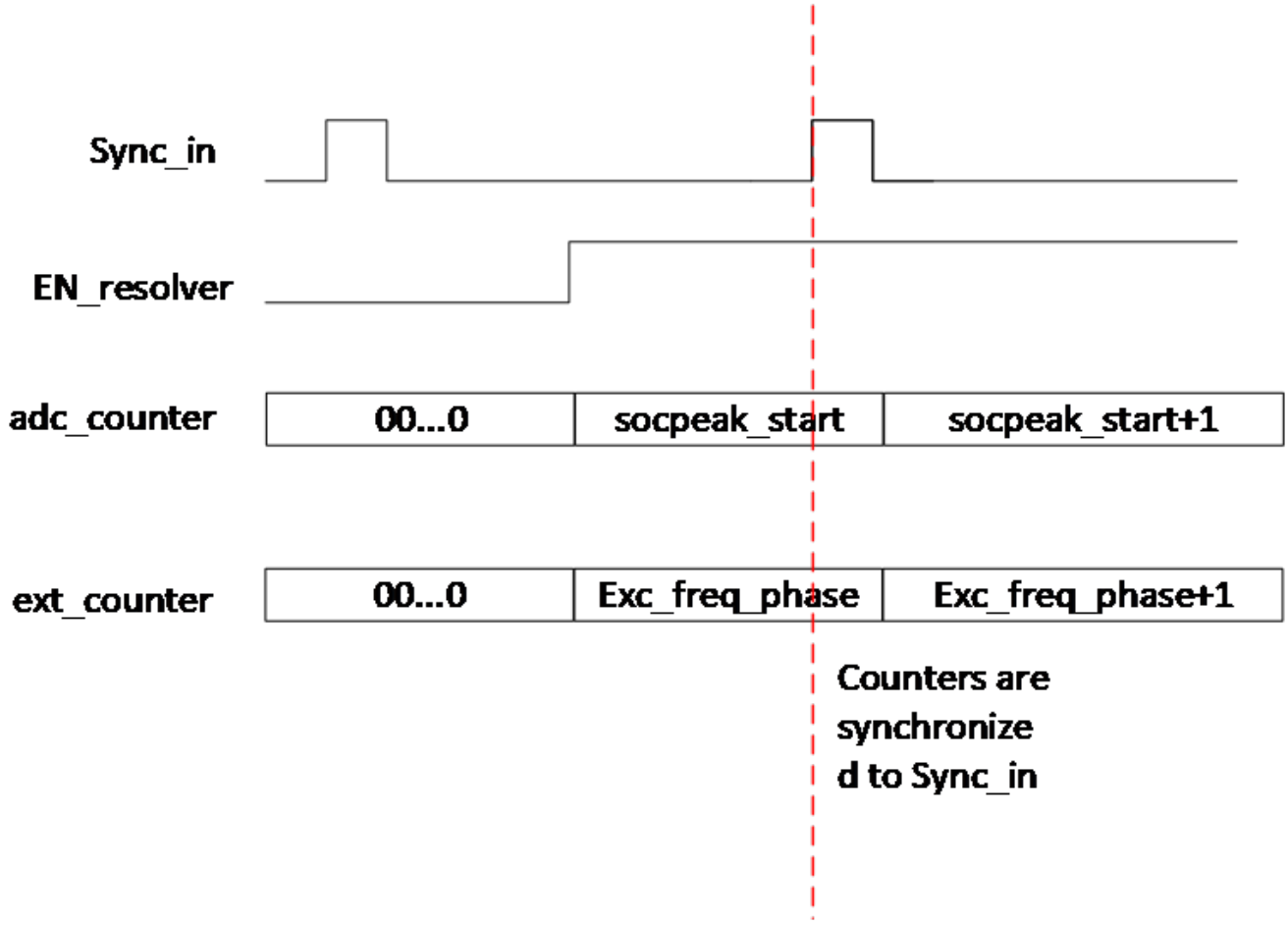


Figure 7-127. PWM Generator Block

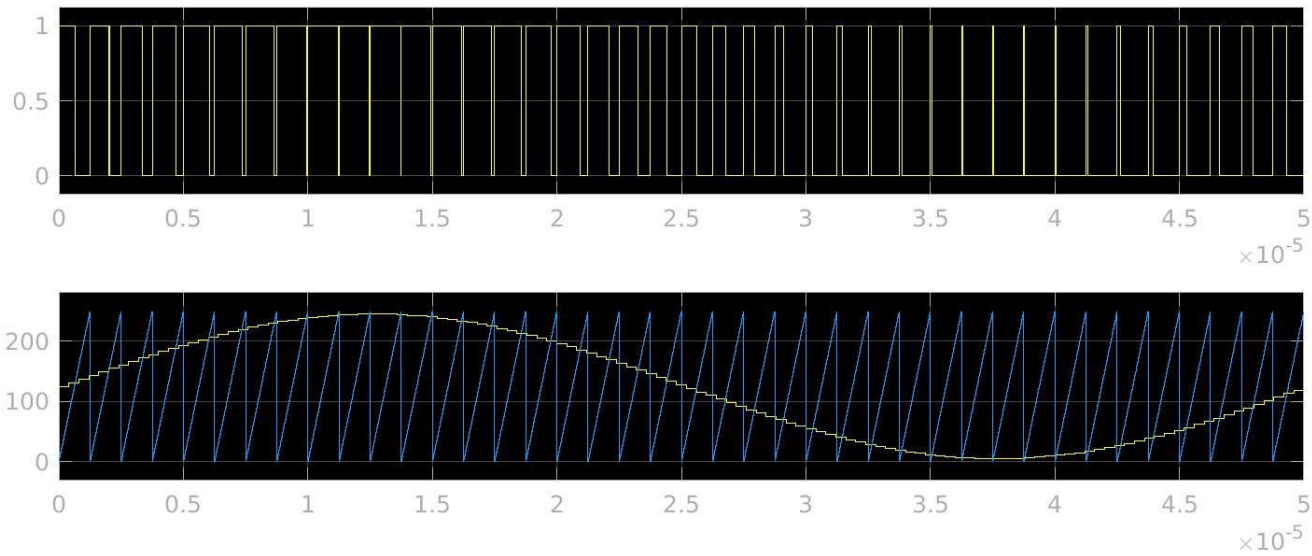


Figure 7-128. PWM Comparator Output (top), and Output of the Counter and Sine Wave (bottom)

The top plot of Figure 7-128 shows the PWM comparator output, and the bottom plot shows the output of the counter and sine wave which are inputs to the pwm comparator. Note the sine wave signal attenuation to avoid clipping. This example shows a 20KHz sine wave generation with 40 pulses per sine wave period.

For PWM generation, a 250-counter counts from 0 to 249 with 200MHz clock. That generates a periodic sawtooth waveform at 800KHz as shown in Figure 7-128. The overflow counter triggers 8000-counter that counts from 0 to 7999 with steps of exc_freq_sel. That counter also generates a sawtooth waveform at 800KHz (in total sync with 250-counter) counting from 0 to 7999. After scaling with * 8.192, this value controls a lookup table of 16 bit sine wave entries. The 16 bit values are multiplied by the gain control (ext_amp_cntrl), then left shifted (divided by 2^{16}), resulting in a 8bit digital sine waveform. If ext_amp_cntrl = 250, then the gain is 1. If it is 225, then gain is = 225/250 = 0.9. Further 125 is added to this signed sine wave, and it goes to a comparator comparing it to the incoming sawtooth waveform generating PWM signals.

The RDC over-samples the excitation frequency with a programmable integer number, and based on that, a table is provided for supported excitation frequency and oversample ratio combinations. Note that oversample ratio OSR = adc_sample_rate × 2, and effective excitation frequency Fexc = 100 × exc_freq_sel in Hz.

This PWM excitation block also supports synchronizing ADCs that sample the motor current with the resolver ADC SOC(start of conversion) signals as shown in Figure 7-129.

The sync pulse(PWMSYNCOUT_XBAR[2]) coming from the motor-PWM-ADC latches the 8000-counter, which indicates the precise phase information. Note that motor PWM might be an integer multiple of excitation frequency, in this case it may trigger the latch at multiple equal intervals of the counter. The user needs to decide which one to use as the resolver ADC sampling time. By reading this info (pwm_phase_info) and programming the value (socpeak_start) user can control the phase of the resolver ADC SOC (start of conversion) signal. Also if a phase difference is desired between those signals, it can be easily implemented by offsetting this value. Socpeak_start and pwm_phase_info registers map 0 to 7999 to 0 to 360° × (7999/8000). Ideally the resolver ADC should also sample the sine and cosine coils at the peak of the excitation signal. After the motor PWM ADC and resolver ADC sampling times are synchronized, through the phase control of resolver PWM (exc_freq_phase_cfg), the excitation signal peak needs to be aligned. This can be achieved by software monitoring the ($\sin^2$ + $\cos^2$) signal and shifting the phase until maximum value is achieved, thus compensating for any phase delay on board.

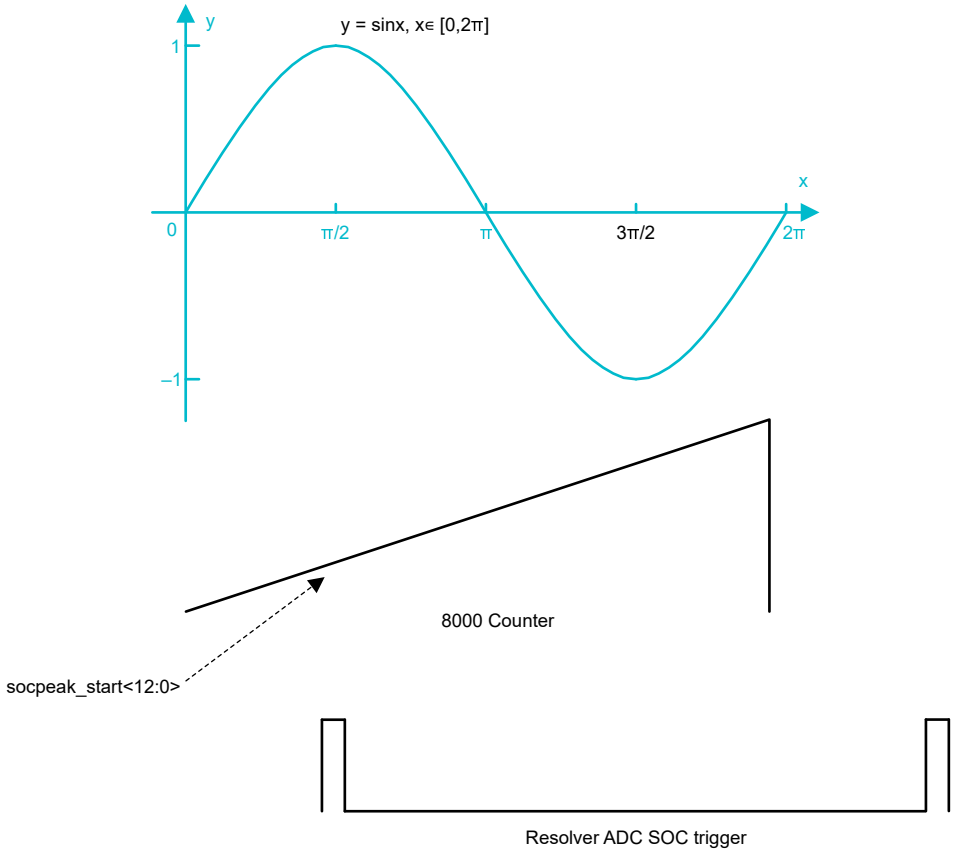


Figure 7-129. Resolver ADC Start of Conversion Trigger Phase Adjustment with respect to Excitation Frequency

7.5.3.2.2.1.3 Offset Correction

This Offset Correction block show in Figure 7-130 calculates and eliminates the DC offset. It runs independently on Sin and Cos channels. The offset correction can be enabled anytime and the resolver shaft need not be spinning. DC average of incoming sine (or cosine) signals are averaged independently (through programmable time constants) and corrected. There is a programmable hysteresis to avoid noise affecting the data.

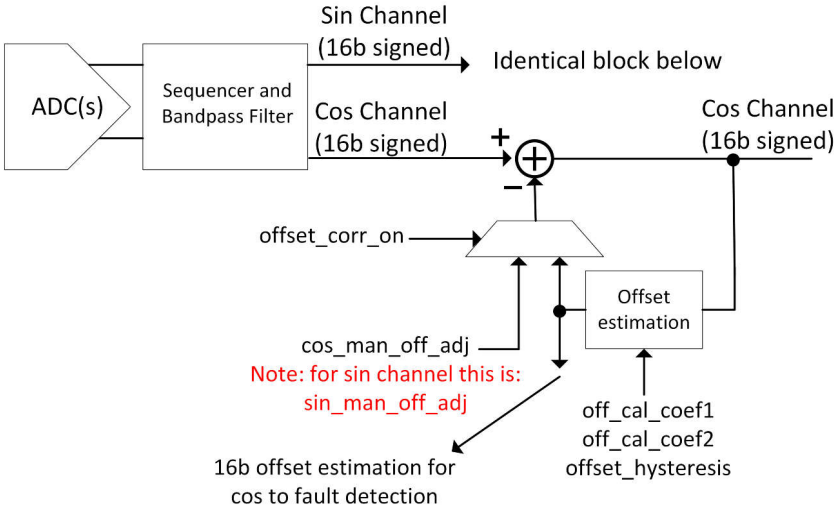


Figure 7-130. DC Offset Correction Block Diagram

7.5.3.2.2.1.4 Auto Sample Time Select

The resolver excitation signal gets modulated by the rotation of the motor creating sine and cosine envelopes(sine and cosine modulated Excitation signal). The demodulation happens by sampling the incoming sine and cosine signals at the peaks(as shown in Figure 7-131) of the excitation signal. Since there will be a phase delay along the signal chain, in a real system the peaks can be at any point. So to sample at the peaks, the RDC oversamples the excitation signal and runs an auto-detection loop to decide the ideal sampling time.

The RDC block has an auto-detect feature to compensate for the delay in signal chain and find the ideal sample point. Before the software enables *ideal_sample_time* selection logic, it needs to make sure the excitation frequency from the external amplifier has settled. The threshold value *sample_det_threshold* is also used to ignore unsettled, low amplitude data from external amplifier before peak detection starts.

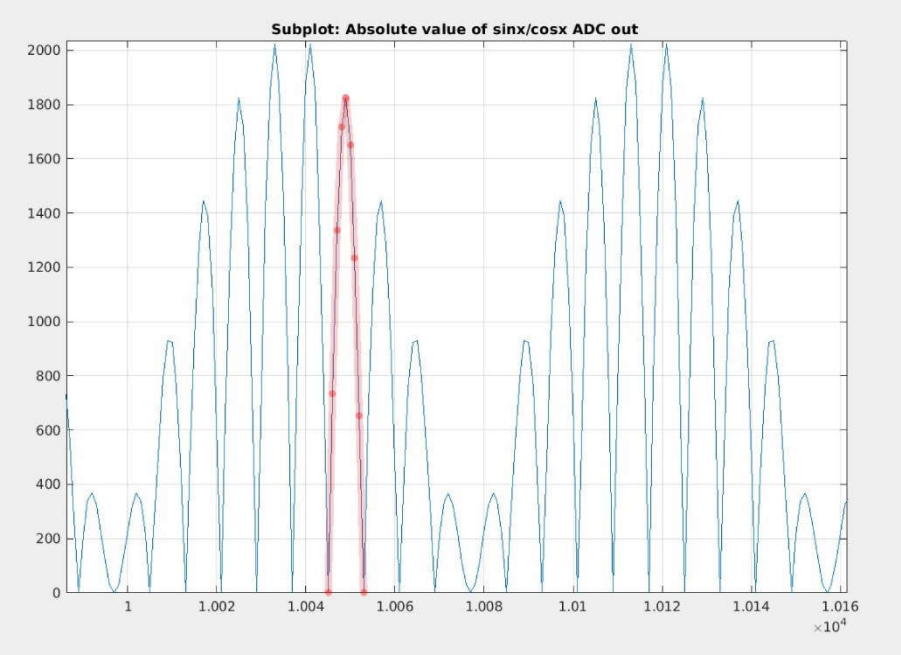


Figure 7-131. Oversampling the Excitation Signal and Deciding Ideal Sampling Time

Auto ideal sample detection is performed through positive peak detection of modulation signal peaks on either Sin or Cos channel. Note that depending where the motor and hence Resolver shaft position is, the ideal sample peaks may be negative or positive.

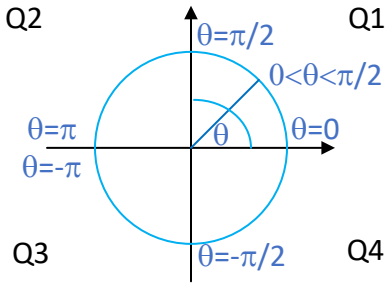


Figure 7-132. Rotor Position on Ideal Sin-Cos Circle

For example ,consider the Sin channel being used for peak detection. With respect to the Figure 7-132, if the rotor zero position is in Q1 or Q2 region, note that the positive peaks of the excitation signal will give the correct sampling point. However if the peak detection is run when the zero position of the rotor is in Q3 or Q4 region, then the positive peak detection of excitation signal will cause the angle position to be shifted by 180 degrees which has to be accounted for accordingly. Hence the RDC sub-system's algorithm has different modes, for find the ideal sampling position as described below in Section 7.5.3.2.2.1.4.1 to Section 7.5.3.2.2.1.4.4 sections.

Once the ideal sample time is determined it can be stored in a non-volatile memory and programmed in ideal_sample_mode.

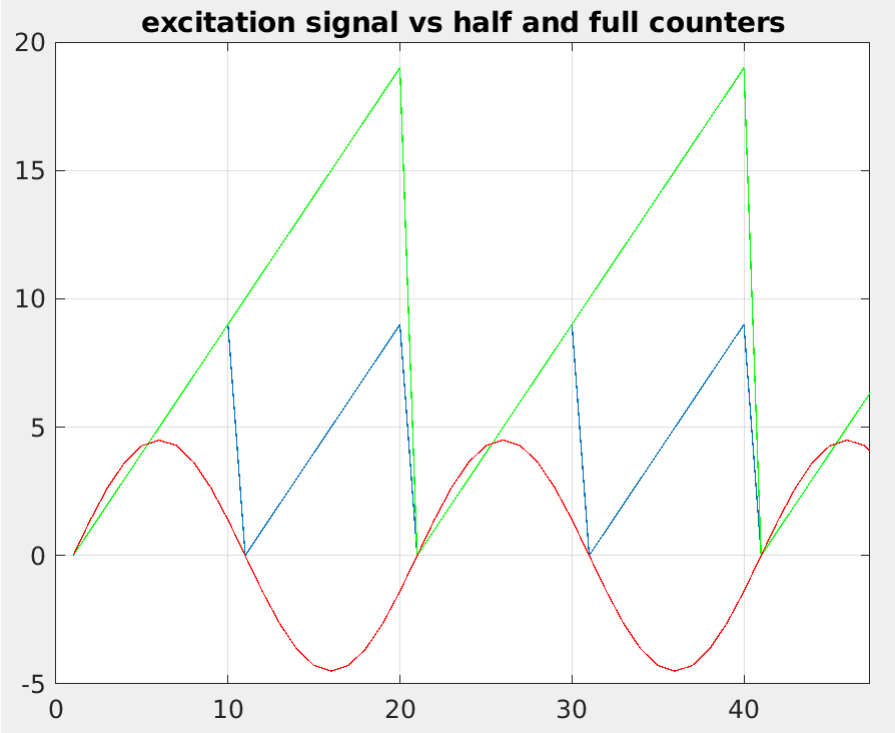


Figure 7-133. Excitation Signal vs Internal Counters

If enable_bottom control bit is set to 1, sin and cos signals are sampled at both positive and negative peak of the excitation signal. RDC auto-corrects the sign of the sample. This improves the response time of the loop, since without increasing the sampling rate, data rate of angle output is doubled.

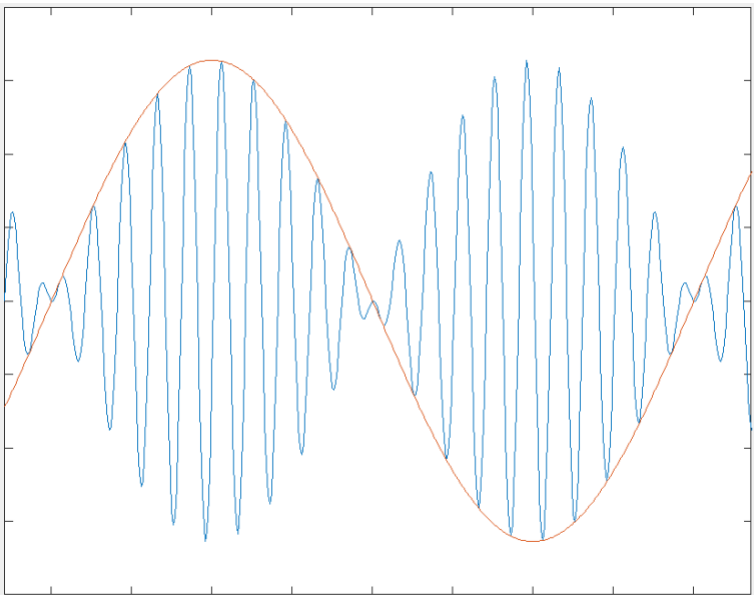


Figure 7-134. Sine Modulated Excitation Frequency

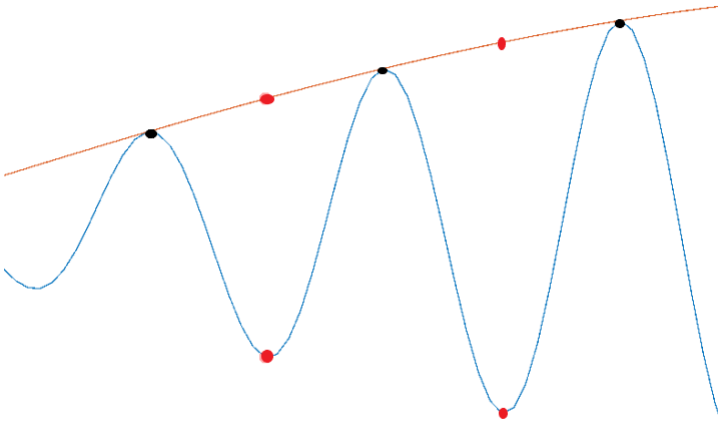


Figure 7-135. Ideal Sampling Points

The above two figures show a sine signal(sine modulated excitation frequency signal) and Ideal sample points shown on the sine signal.

7.5.3.2.2.1.4.1 Ideal Sample Mode 0

This is the recommended mode of operation if the zero-position of the rotor is in Q1 during detection. Both Sin and Cos channels will be positive, the algorithm will select the strongest signal.

Ideal sample time selection is done in two phases.

Phase-1: In the first phase logic checks the sine channel and cos channel (clock: oversample coefficient × excitation frequency). It has one counter for sin, and one counter for cosine. These counters get incremented each time a sample is above sample_det_threshold. When total number of samples of both counters reach

$2^{peak\_avg\_limit}$, two counters are compared. The counter with a larger count is selected as the valid channel to detect demodulation peak. If they are equal, sin channel is selected.

Phase-2: In the second phase, a moving average of 3 samples is continuously compared to each other. If the average starts falling down while previously was moving up, or equal, this is recorded as the local peak. In order to avoid false triggers when the signal is low, samples below sample_det_threshold are ignored. After $2^{peak\_avg\_limit}$, ideal local peaks are collected and are averaged to find the ideal peak sampling point.

7.5.3.2.2.1.4.2 Ideal Sample Mode 1

This is the recommended mode of operation if the zero-position of the rotor is in Q1 or Q2 during detection (ideally between 45 to 135 degrees). Sin channel will be positive.

Similar to previous Ideal Sample Mode 0, but Phase-1 is bypassed and sin channel is used for Phase-2 to detect ideal peak sampling point.

7.5.3.2.2.1.4.3 Ideal Sample Mode 2

This is the recommended mode of operation if the zero-position of the rotor is in Q1 or Q4 during detection (ideally between 45 to -45 degrees). Sin channel will be positive.

Similar to Ideal Sample Mode 1, but Phase-1 is bypassed and cos channel used for Phase-2 to detect ideal peak sampling point.

7.5.3.2.2.1.4.4 Ideal Sample Mode 3

Auto ideal sample time selection is bypassed, ideal_sample_time_ovr register is used to decide sample time

Note that in Modes 0 to 2, when the hardware loop detects the ideal_sample_time, the loop will also fill out 20 histogram registers. If the ideal sample location is close to 0, with a noisy signal sometimes the loop may converge to 19, or to 0. The average will give an incorrect value of 10 for the ideal sample time location. To avoid this, it's recommended to check the histogram registers (OBS_PEAKHISTOGRAM_XX). Then software can decide which point to take as ideal sample time location, and program that in ideal_sample_mode =3.

7.5.3.2.2.1.5 Automatic Gain and Phase Correction

Ideally the Resolver's sine and cosine coils need to have a 90° phase shift. They are also expected to have identical gains. Due to misalignment of the coils during resolver manufacturing, and impedance mismatch on the signal paths to the RDC, gains of Sine and Cosine may not match, yielding to an error in angle detection. The RDC can correct those gains after observing multiple rotations of the Resolver shaft. The timing of Auto Gain and Phase correction are depicted in Figure 7-136:

- **Step 1**: Excitation frequency is generated, incoming DC offset is cancelled by averaging the sin and cos signals over multiple excitation signal periods and correcting for the deviation from ideal mid-point (or bandpass filter needs to be enabled). Ideal sample time is selected, and resolver starts generating angle data. For this loop to work correctly, resolver shaft needs to be rotating.
- **Step 2**: Over each rotation period, if gain and phase correction is enabled, the gain and phase deviation of sin and cos signals with respect to each other is calculated. Effect of noise and acceleration can be minimized by averaging over multiple rotations (motor needs to be rotating).

The automatic gain correction can be enabled alone, or automatic gain and phase correction can be enabled together. This auto gain and phase correction can also be bypassed, and manually adjusted.

Automatic Gain and Phase correction consists of two blocks: Estimation logic and Correction logic. Estimation logic can be enabled while the correction logic is either enabled or disabled. For Correction logic to be enabled the Estimation logic also needs to be enabled.

1. **bypassphasegaincorr**: 0 enables estimation logic, 1 disables estimation logic
2. **autogaincontrol**: 0 disables correction logic, 1 enables correction logic
3. **autophasecontrol**: 0 disables correction logic, 1 enables correction logic
4. **gainsinbyp0**: 16 bit unsigned gain control for sin channel if autogaincontrol is disabled.(2^{14} corresponds to gain of 1)
5. **gaincosbyp0**: 16 bit unsigned gain control for cos channel if autogaincontrol is disabled. (2^{14} corresponds to gain of 1)

6. **phasecosbyp**: 16 bit signed phase control for cos channel if autophase is bypassed. (2^{15} corresponds to phase adjust of 90° and -2^{15} corresponds to phase adjust of -90°)

Gain and Phase Correction and Timing

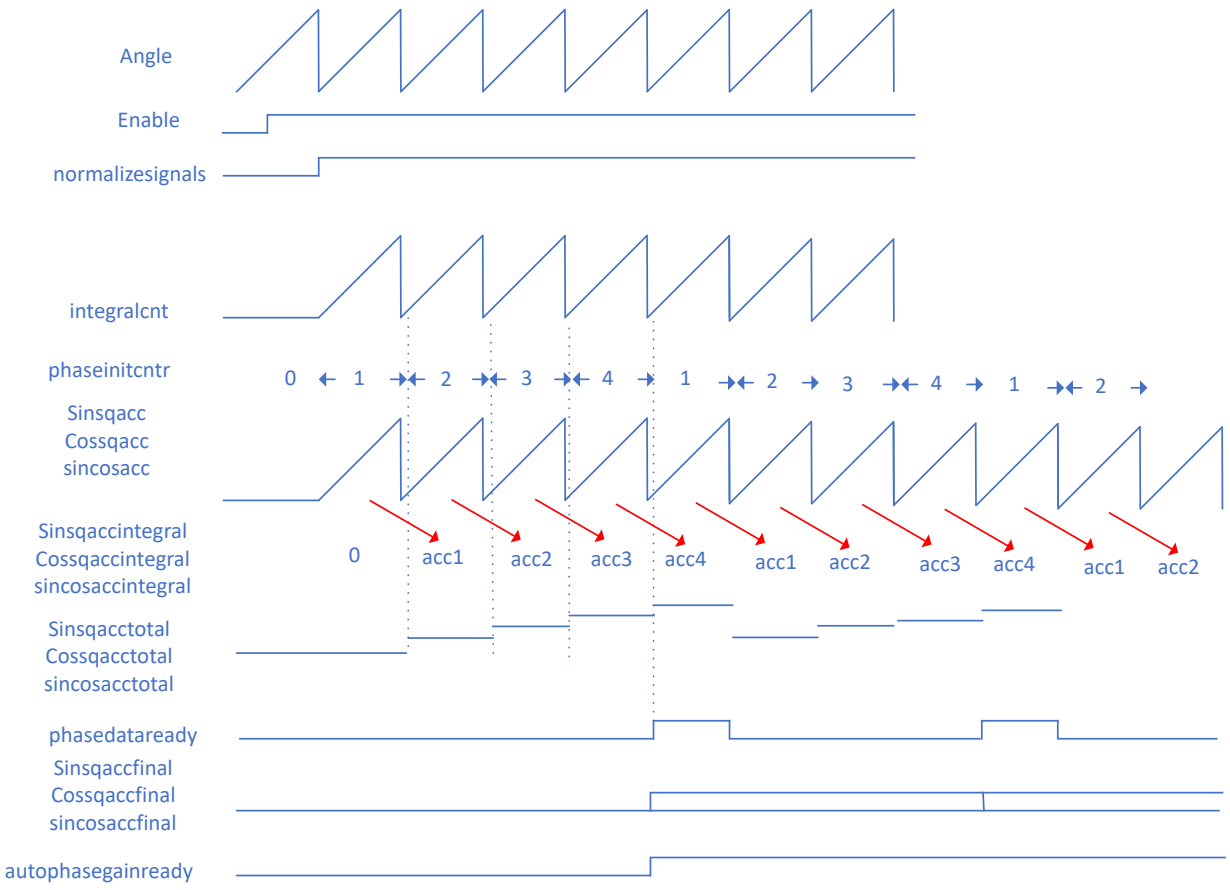


Figure 7-136. Gain and Phase Correction and Timing

The phase estimation and gain estimation registers can be read to monitor the estimated phase and gain values of sin and cos signals. Writing to the estimation registers to correct Phase and Gain values is also possible.

To read the estimated gain and phase values and values and calculate the analog value refer to the below formulae:

- Estimated analog gain of sin : $\sqrt{}$ (2^{29} / sinsqaccfinal)
- Estimated analog gain of cos: $\sqrt{}$ (2^{29} / cossqaccfinal)
- Estimated differential phase error between sin and cos (deg) = phaseestimatefinal × 90 / (2^{15})

Note

Resolver shaft needs to be rotating for Automatic Gain and Phase Correction to work.

The estimated gain correction values for sin and cos channels will map the sin and cos data path to perfectly full 16 bits scale. In order to avoid clipping (due to noise and glitches), it is recommended to scale the gain values also.

7.5.3.2.2.1.6 Glitch Filter and Decimation

This block demodulates the sin and cos signals from the modulated signals received from Resolver.

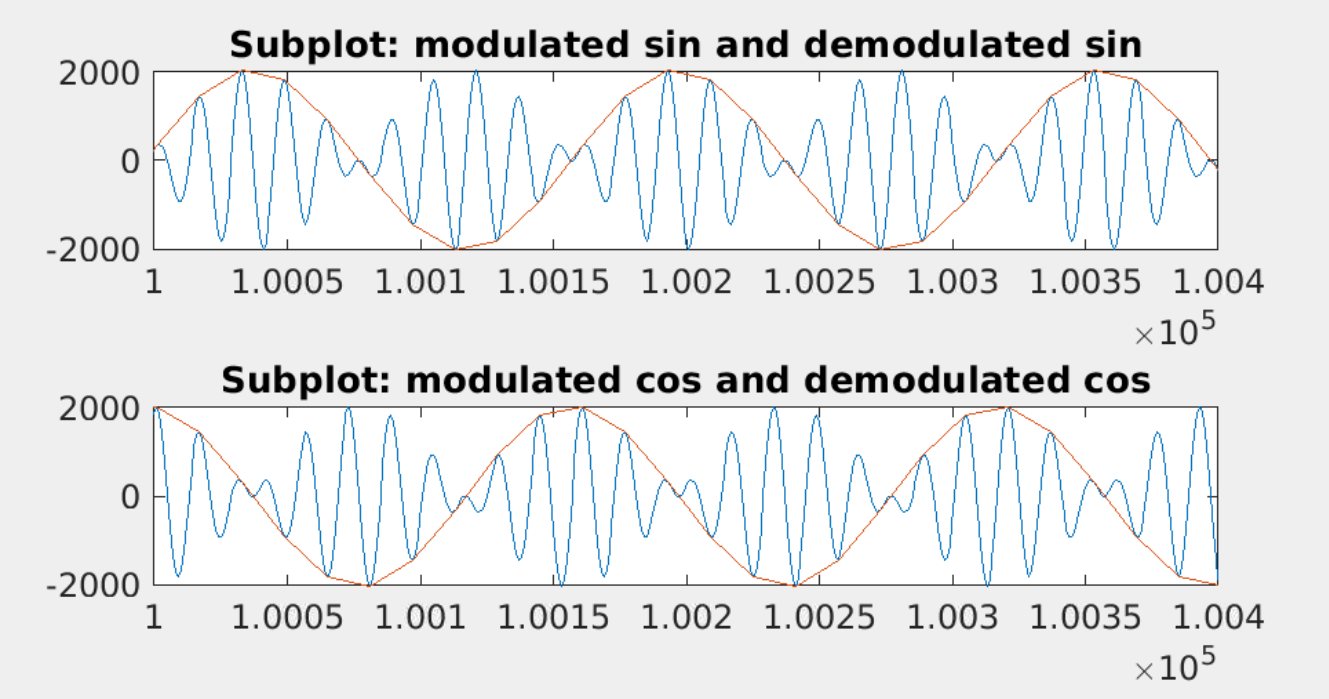


Figure 7-137. Selecting Ideal Sampling Point for Demodulation

Resolver oversamples the excitation frequency (programmable samples per T_{exc}), and decides on the ideal sampling time compensating for phase shift that occurs on excitation signal path. The envelope or the sin and cos signals are re-constructed by the values obtained at the sampling that was done at the peaks of the sine and cosine input signals. This results in the sin and cos signals which the RDC can further pass on to the Arctan or Track-2 blocks for calculating angles and velocity.

7.5.3.2.2.1.7 Arctan

This is the second option to recover the angle. It feeds the sin and cos to a simple arctan function block, which produces the angle. This block has no noise rejection feedback loop, hence lower latency but higher noise pass through. It has a precision of 16 bits. Refer to Figure 7-115.

7.5.3.2.2.1.8 Track2

Resolver calculates the arctan of sin and cos inputs. It then feeds this arctan signal to a second order phase locked loop (Track2).

Simplified Diagram of Track 2 Loop

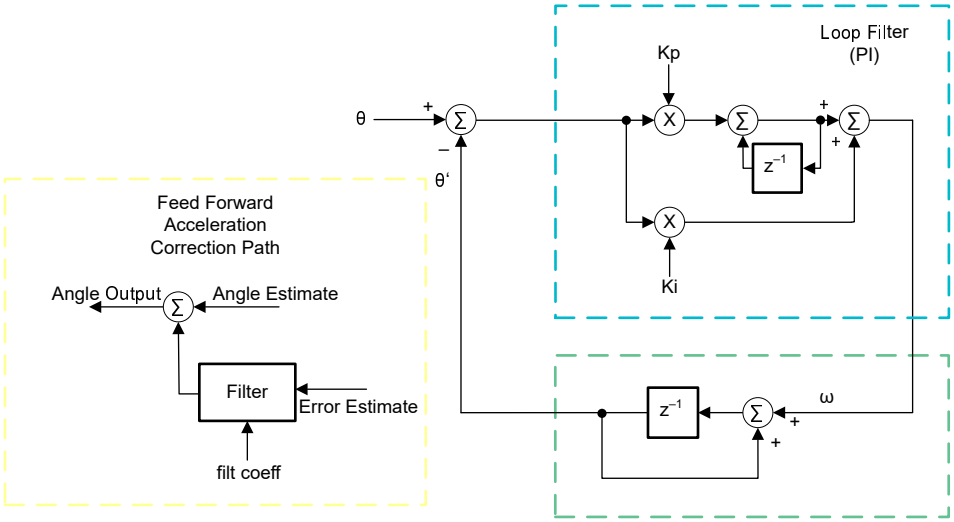


Figure 7-138. Simplified Diagram of Track 2 Loop

The rotational spped(in frequency units) can be calculated based on following formula:

Speed = velocityout × $(1/T_s)$ × $(1/2^{32})$: This is rotational speed in Hz.

where

- exc_freq_sel : Excitation frequency setting MMR
- enable_bottom(0 or 1) : bottom sampling control MMR: 1 doubles the data rate
- F_s = 100 × exc_freq_sel × (1 + enable_bottom) : This is the effective sampling rate of track2 loop
- $T_s = 1/F_s$: This is the effective sampling time of track2 loop
- velocityout: 32b signed register output of track2

7.5.3.2.2.1.9 ADC IF

All outputs driven to SAR ADC are from a flop with no feedback. All of these flops reside in resolver_adc_if_flops module.

Definition of terms for ADC-RDC timing interface:

- **System clock freq f_{sys}**= 50MHz
- **System clock period T_{sys}**= 5nS
- **ADC sampling frequency**: $f_{ADC}= 1/T_{ADC}$
- **ADC sampling period**: T_{ADC} = (17 + soc_width) × T_{sys}
- **Resolver excitation frequency** (1KHz to 20KHz): f_{EXC} = exc_freq_sel × 100
- **Resolver excitation period**: $T_{EXC} = 1/ f_{EXC}$ (for 10KHz = 0.1mS = 20K × Tsys)
- **Resolver oversampling ratio (number of ADC samples over one excitation period): OVR** : 2 × adc_sample_rate
- **Resolver sampling period**: $T_{EXCOVR} = T_{EXC}$ / OVR

  (if T_{EXC} = 0.1mS, and OVR=20, T_{EXC} = 1K × T_{sys})
- **Resolver demodulation frequency**:

  = f_{EXC} , if doublesample = 0

  = 2 × f_{EXC} , if doublesample = 1
- **Sample_select_counter**: The internal resolver counter that runs from 0 to (OVR - 1). This decides ideal sample location

- Positive peak of excitation signal if doublesample = 0
- Both positive and negative peak if doublesample = 1
- Runs from 0 to (2 × adc_sample_rate – 1) for every time period of T_{EXC}.

7.5.3.2.2.1.10 Interrupts

The RESOLVER produces one HW event output active high level and active high pulse.

All of the error tracking/diagnostic events can get mapped to generate an event to notify the host an error occurred. The software can emulate a hardware event to verify the full path does not have a fault; this is important for safety.

There are 4 MMRs for SW to interact with:

- IRQSTATUS_RAW_SYS(resolver_REGS_irqstatus_raw_sys[31:1])
- IRQSTATUS_SYS(resolver_REGS_irqstatus_sys[31:1])
- IRQENABLE_SET_SYS(resolver_REGS_irqenable_set_sys[31:1])
- IRQENABLE_CLR_SYS(resolver_REGS_irqenable_clr_sys[31:1])

The reading of IRQSTATUS_RAW_SYS shows the status of the RAW event, if set it is active, if cleared it is inactive. Even if set, the external event can be deasserted because it is not enabled along with corresponding event in the IRQSTATUS_SYS status(post mask/enable). The writing of IRQSTATUS_RAW_SYS allows software emulate emulate a hardware event, but the software must first enable that event by writing to IRQENABLE_SET_SYS.

The writing of IRQENABLE_SET_SYS is required to enable the hardware event to be asserted.

The reading of IRQSTATUS_SYS shows which hardware events are active. It requires the event to get enabled via IRQENABLE_SET_SYS and that the hardware event did occur. To clear the hardware event, software must write 1 to clear that bit in the IRQSTATUS_SYS.

The writing of IRQENABLE_CLR_SYS allows the software to disable hardware events.

The typical software programming model:

- Enable the necessary hardware events via IRQENABLE_SET_SYS.
- For Safety check out, software must set software event by writing to IRQSTATUS_RAW_SYS.
- To clear the event, the software must write 1 to clear that bit in IRQSTATUS_SYS.

The hardware level event output remains asserted as long as one or more events are active.

7.5.3.3 Programmer's Guide

Calibration must be done in a controlled setting with a simple constant rotation of resolver for the logic to tune the errors in the signal. After tuning, software can read the tuned values which were calculated, and use them in the future.

Before operating the Resolver-to-Digital-Converter (RDC), all the registers need to be set correctly. master_en must get cleared before software adjusts any MMRs. After the MMRs are set correctly, then software can set back master_en.

A resolver sensor needs an excitation signal to convert rotational information into a Sine Cosine modulated signal pair, which are then converted to digital angular information through the RDC. This excitation signal is generated by the internal RDC PWM generator which is then filtered and amplified by an external amplifier(outside AM263P). The output of the external amplifier drives the resolver coil. The sine and cosine outputs of the resolver are sampled, converted to digital by ADCs, and processed further by RDC.

Step-1 Power-On Diagnostics: Before initiating the PWM signal and activating the RDC, software is recommended to run diagnostic checks on the resolver ADCs:

There are two types of ADC dedicated diagnostic tests:

1. Open-Short tests: Resolver Cos and Sin coils can be connected to RDC ADC inputs through only passive components including pull/up and pull/down bias resistors or they can be connected through an amplifier. During power-on these connections can be checked through ADC internal Open Short Detection(OSD). Please refer to Open-Short section of ADC Chapter for more details.

2. Runtime diagnostic tests with 2 channels dedicated for diagnostics. While ADCs are not sampling the resolver inputs, they will sample two inputs to check full functionality of the ADCs for safety purpose.

Step-2 Sequencer: After diagnostic tests, the next step is to select the desired operating mode through sequencer settings in resolver_REGS_global_cfg[11:8]. Refer to Multiplexer/Sequencer Front End section for details. RDC subsystem can support one or two hardware Resolvers as described.

Step-3 Excitation signal: Next step is to initialize the excitation signal. The resolver sensor will need a sinusoidal excitation signal with a programmable frequency(using resolver_REGS_excit_sample_cfg1[7:0]) that will be in sync with the ADC sampling. This signal will be a PWM signal, filtered and amplified externally. Before RDC processes incoming data, enough time needs to be allocated for the external PWM to analog signal conversion to settle to its operating frequency. A typical excitation signal PWM filter amplifier(ALM2403-Q1) is shown in Filter and Amplifier Circuit figure and its startup time can be referred for typical settling time of a filter amplifier in datasheet of ALM2403-Q1()

Step-4 Oversample Ratio, Bandpass filter and/or Offset Correction: After excitation frequency settles, resolver inputs will be valid, and calibration and decimation process can be started by software. Each period of excitation frequency is over-sampled by a programmable ratio(using resolver_REGS_excit_sample_cfg1[15:8]). Default over-sample ratio is 20. This oversamplling enables to use a bandpass filter centered around excitation frequency by providing enough bandwidth. Oversampling also enables offset correction to settle faster and more accurately and also to detect the ideal decimation point.

Software needs to decide bandpass filter to be enabled or disabled(using resolver_REGS_dc_off_cfg1_0[8]). Enabling bandpass filter will introduce phase delay, but it will significantly improve noise rejection. This bandpass filter will also reject DC offset. If bandpass filter is enabled, DC offset correction can be disabled. Enabling DC offset correction when bpass filter is enabled will not degrade or improve signal. Regardless of DC correction being enabled, DC offset estimation always runs and that way DC offset monitoring can monitor faults. **Note: Bandpass filter is only designed for oversample ratio-20.**

If bandpass filter is disabled, it is recommended to enable DC offset correction. Note that if there is no valid excitation signal, DC offset correction will saturate the input signal to the RDC. This condition is monitored in fault detection modes. At this point DC offset fault detection and excitation signal monitor fault detection modes need to be enabled. Refer to above sections to program bandpass filter and offset correction.

Note

If bandpass filter is disabled, and offset correction needs to be enabled, in this case, offset correction needs to be enabled after ideal sample time selection converges. As explained in next step, ideal sample time checks the peaks to decide optimum sampling point. Although any noise will be averaged, offset correction may introduce false peaks, initially reducing the accuracy.

Step-5 Ideal Sample time selection, and decimation: In order to demodulate the rotation signal, resolver needs to sample the input signal during the peak point of the excitation signal. There are multiple considerations for this:

1. Ideal sample time selection block oversamples the input signal by 20 and decides the ideal sampling point.
2. Motor pwm currents are sampled by SOC ADCs. It would be a good practice to align sampling of the resolver signal, with the sampling of the corresponding motor PWM current. This improves the motor control loop by eliminating the latency. A synchronization pulse coming from motor PWM block which should to be used to synchronize resolver ADC sampling time.
3. To demodulate the signal, the peak of excitation signal needs to be sampled. Additionally, software can also enable sampling the negative peak of excitation frequency to improve settling. RDC finds the negative peak, and takes care of the sign automatically as explained in Section .
4. If DC offset correction is needed, it needs to be enabled after auto-ideal time selection.

Once the ideal sample time selection is done and configured as manual value, the DC offset correction can be enabled. This will prevent the DC offset correction from interfering with the ideal sample selection algorithm.

Step-6 Differential Phase and Gain Mismatch Correction: Ideally sin and cos signals should have a perfect phase delay of 90 degrees, and their amplitudes should match. If there's a common phase delay, this can be

handled by the factory calibration by resetting the 0 deg position, and any minor common gain error will cancel out during arctan calculation. Gross common gain errors will be detected by fault detection mechanisms as explained in fault detection section. In real applications, there will be both differential phase and gain mismatch. Phase and Gain Calibration needs to be done after ideal sample time selection has settled. This is done by enabling the estimation, reading the estimated values and replacing them with manual values. The estimated values can be helpful in the diagnostics.

Step-7 Arctan and Track2 outputs: Output of phase and gain correction goes to arctan block. The arctan data feeds to Track-2 loop. Outputs from the Arctan, or the Track-2 can be read directly from the registers.

CAUTION

Please note that there is a hardware limitation on Arctan offset and hardware track2 velocity resolver_REGS_velocity_track2_0[31:0] sampling. Thus a Software track2 can be used in place of hardware track2 . The resolver_angle_speed from the SDK implements the Software track2 for velocity sampling.

After enough time is allowed for excitation signal to stabilize, ideal sampling time can be calculated, and offset correction can be enabled. They work independently, so they can be enabled at the same time, or sequentially.

RDC is designed to meet stringent ASIL-D requirements, and following diagnostic features are supported, they are classified under three main groups: Degradation of Signal (DOS), Loss of Signal (LOS), Loss of Track (LOT):

1. Monitor Sin or Cos offset drift (DOS)
2. Monitor Sin or Cos gain drift (DOS)
3. Monitor phase drift (phase drift between sin and cos channels) (DOS)
4. Monitor excitation frequency degradation or loss (DOS)
5. Monitor signal integrity by observing Sin and Cos zero crossings (DOS)
6. Monitor signal integrity by checking $Sin^2+Cos^2=Constant$ (DOS)
7. Monitor Sin or Cos saturation or very high amplitude (DOS)
8. Monitor weak Sin or Cos signal below a threshold (LOS)
9. Monitor track2 loop locking to incoming angle (LOT)
10. Monitor ADC health through calibration channels (DOS)

Software SDK offers the building blocks to read, check, enable interrupts from the diagnostics data. Steps for further diagnostics programming will be included in a later TRM release.

7.5.4 Comparator Subsystem (CMPSS)

The Comparator Subsystem (CMPSS) consists of analog comparators and supporting circuits that are useful for power applications such as peak current mode control, switched-mode power, power factor correction, voltage trip monitoring, and so forth.

7.5.4.1 Introduction

The comparator subsystem is built around a number of modules. Each subsystem contains two comparators, two reference 12-bit DACs, and two digital filters. Comparators are denoted "H" or "L" within each module where “H” and “L” represent high and low, respectively. Each comparator generates a digital output which indicates whether the voltage on the positive input is greater than the voltage on the negative input. The positive input of the comparator is driven from external pins.

Each comparator output passes through a programmable digital filter that can remove spurious trip signals. An unfiltered output is also available if filtering is not required. The negative input for only COMPH(CMPSSA) can be driven by an external pin or by programmable 12-bit DAC. The negative input for COMPL (CMPSSA) can only be driven by 12-bit DAC. The negative input for CMPSSB (COMPH and COMPL) can only be driven by the programmable 12-bit DAC.

7.5.4.1.1 Features

Each CMPSS includes:

- Two analog comparators
- Two independently programmable reference 12-bit DACs
- One decrementing ramp generator
- Two digital filters, max filter clock prescale = 2^{16}
- Ability to synchronize submodules with EPWMSYNCPER
- Ability to extend clear signal with EPWMBLANK
- Ability to synchronize output with SYSCLK
- Ability to latch output
- Ability to invert output
- Option to use hysteresis on the input
- Option for negative input of comparator to be driven by an external signal or by the reference DAC for COMPH
- VDACREF is the DAC reference voltage
- Diode emulation support
  - The system works with EPWM to support the Diode emulation feature
  - Details about Diode emulation can be found in *ePWM Modules Overview*
- Ramp generator prescaler

CMPSSA has the above features, and the additional support of INH and INL as a muxable input for the COMPL positive signal. Figure 7-139and Figure 7-140 shows the differences.

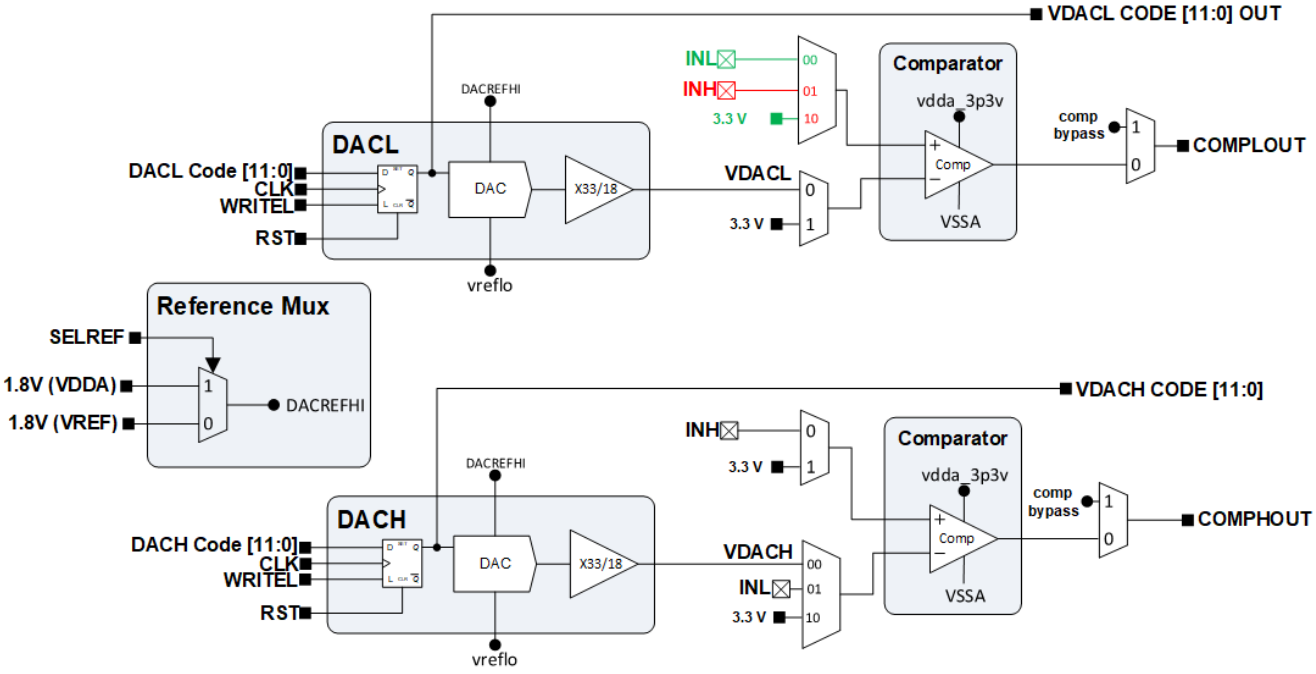


Figure 7-139. CMPSSA Block Diagram

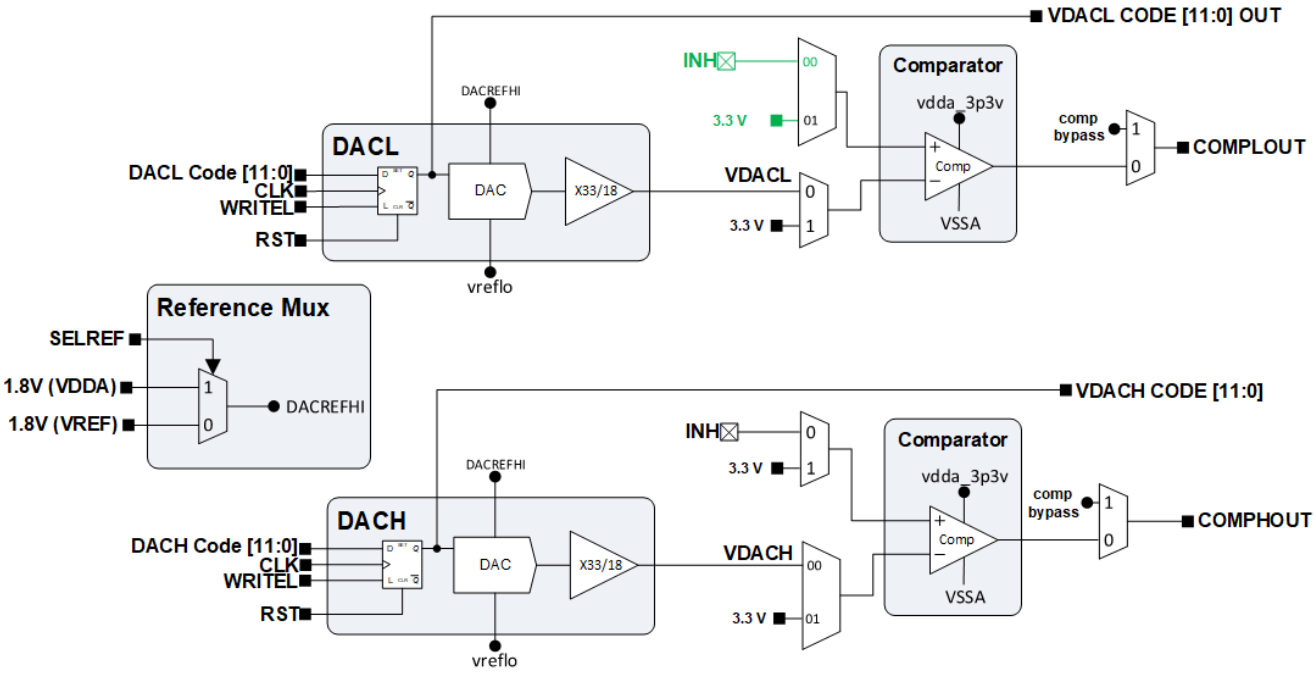


Figure 7-140. CMPSSB Block Diagram

7.5.4.1.2 Comparator

Section 7.5.4.1.3 shows several comparators. The comparator generates a high digital output when the voltage on the positive input is greater than the voltage on the negative input, and a low digital output when the voltage on the positive input is less than the voltage on the negative input. The comparator is illustrated in Figure 7-141.

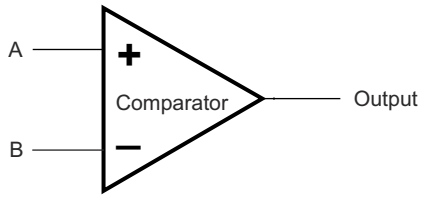


Figure 7-141. Comparator Block Diagram

Voltages	Output
Voltage A > Voltage B	1
Voltage A < Voltage B	0

7.5.4.1.3 Block Diagram

The block diagram for the CMPSS is shown in the images below.

- CTRIPx(x= "H" or "L") signals are connected to the ePWM X-BAR for ePWM trip response. See the *Enhanced Pulse Width Modulator (ePWM)* chapter for more details on the ePWM X-BAR mux configuration.
- CTRIPxOUTx(x= "H" or "L") signals are connected to the Output X-BAR for external signaling. See the *General-Purpose Input/Output (GPIO)* chapter for more details on the Output X-BAR mux configuration.

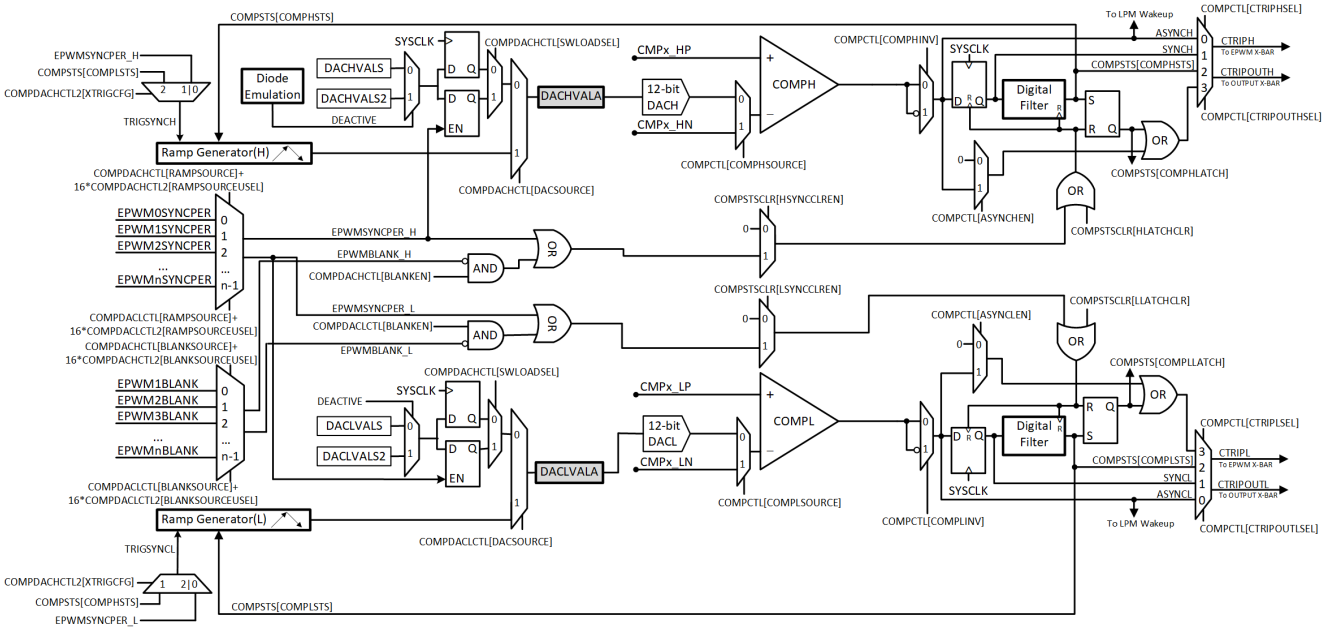


Figure 7-142. CMPSS Module Block Diagram - Detailed

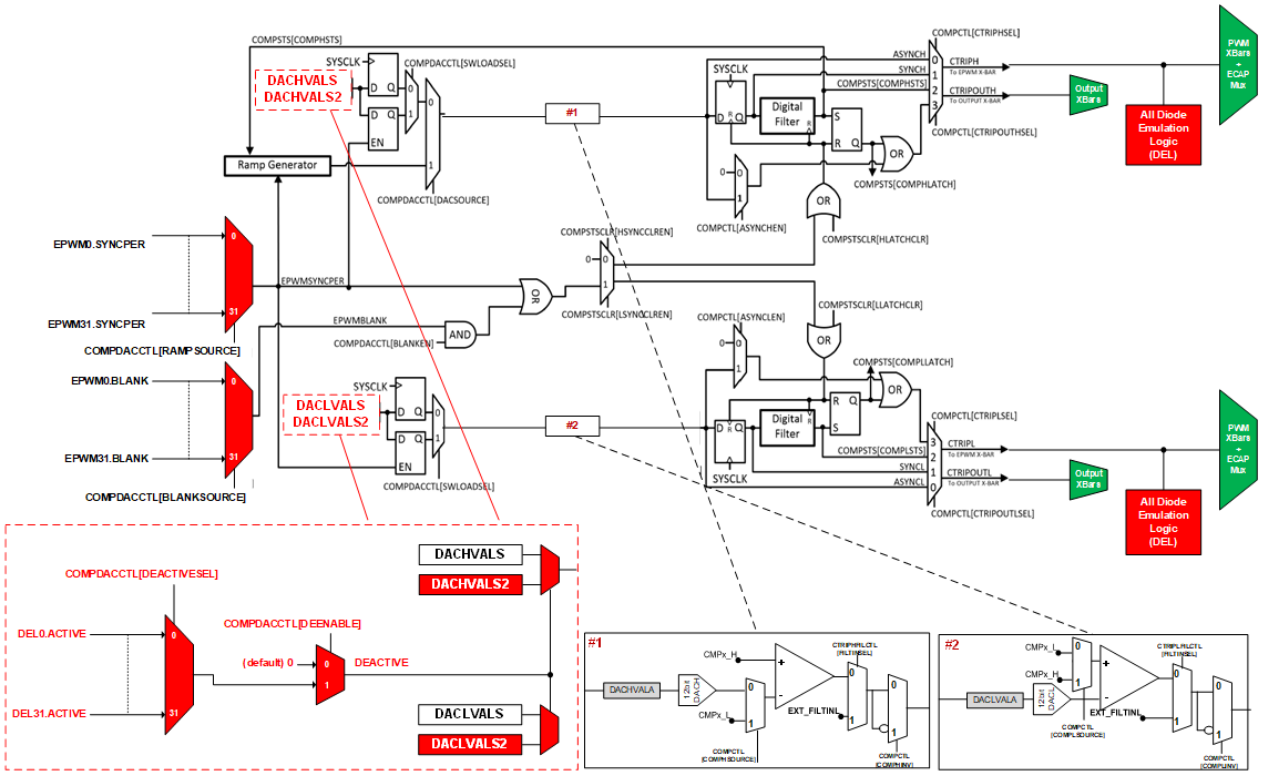


Figure 7-143. CMPSSA Module Block Diagram - Integration

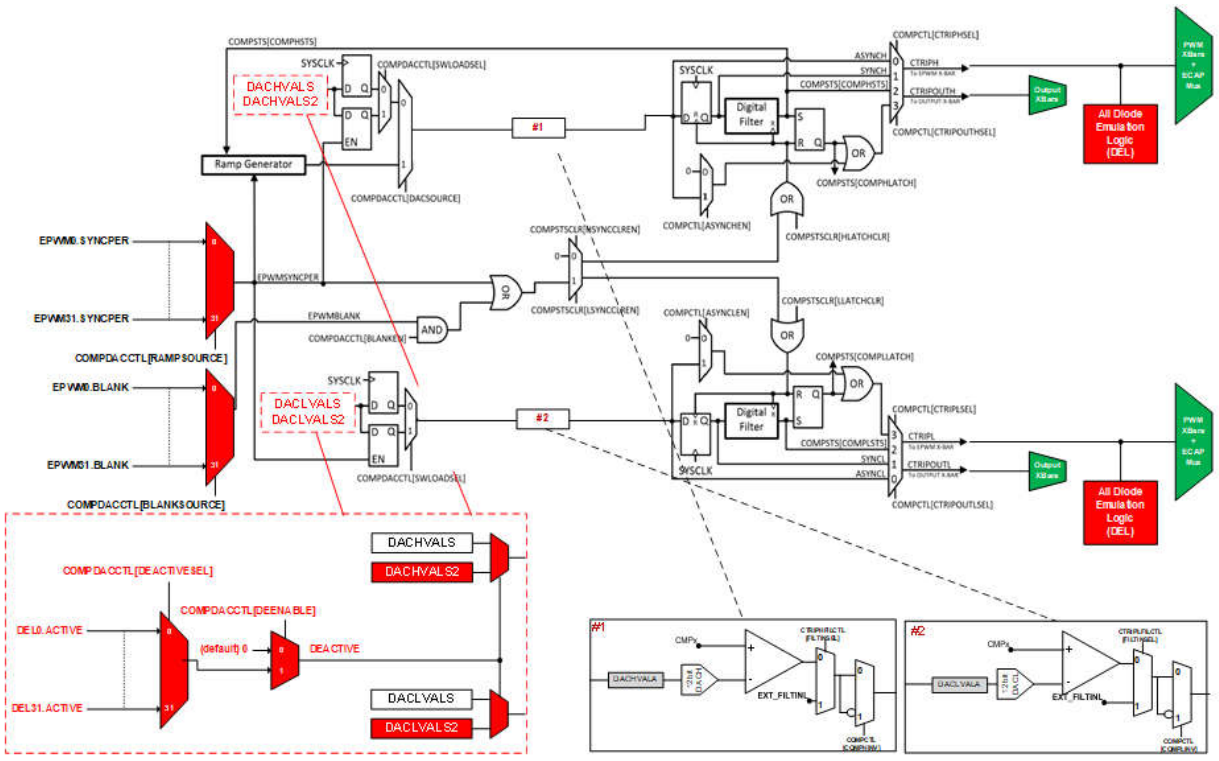


Figure 7-144. CMPSSB Module Block Diagram - Integration

Note

Colors help highlight key parts of the diagram, but do not contain meaning otherwise.

This concludes the CMPSS introduction. Additional foundational material can be found at:

- Comparator Subsystem Training
- Real-Time Cntrol Reference Guide (Refer to the Comparator section)

7.5.4.2 ADC-CMPSS Signal Connections

In each ADC, two sets of differential pins shall be shared with pins of two CMPSSA and remaining one pair of differential pins shall be connected to two independent pins of CMPSSB. These pins are demonstrated in Figure 7-145 and Table 7-118 where the CHSEL values determine how the inputs are fed into ADC.

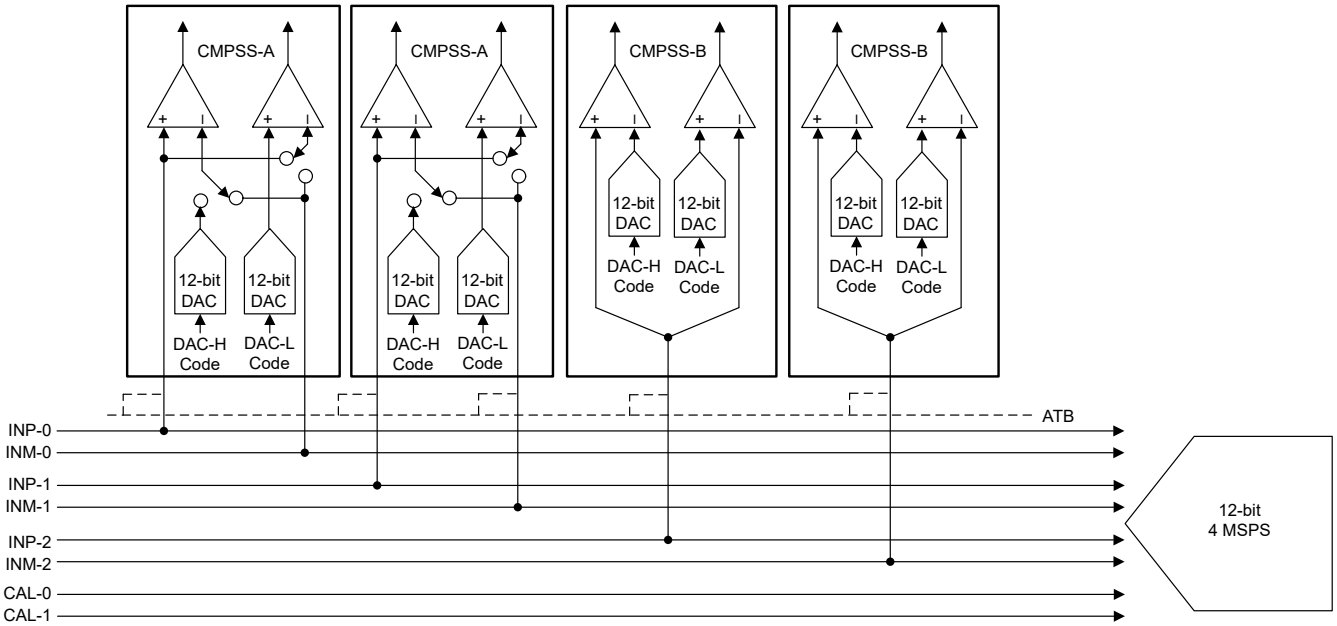


Figure 7-145. CMPSS and ADC Connections

Table 7-118. Connectivity between ADC Inputs to CMPSS Signals

Signal/Pin Name	ADC Input	CMPSS Input
ADC0 Channels		
ADC0_AIN0	ADC0:inp0 (+IN0)	CMPSSA0:inH (+IN)
ADC0_AIN1	ADC0:inm0 (-IN0)	CMPSSA0:inL (-IN)
ADC0_AIN2	ADC0:inp1 (+IN1)	CMPSSA1:inH (+IN)
ADC0_AIN3	ADC0:inm1 (-IN1)	CMPSSA1:inL (-IN)
ADC0_AIN4	ADC0:inp2 (+IN2)	CMPSSB0:inH/inL (+IN/-IN)
ADC0_AIN5	ADC0:inm2 (-IN2)	CMPSSB1:inH/inL (+IN/-IN)
ADC_CAL1	ADC0:inm3 (-IN3)	X
ADC_CAL0	ADC0:inp3 (+IN3)	X
ADC1 Channels		
ADC1_AIN0	ADC1:inp0 (+IN0)	CMPSSA2:inH (+IN)
ADC1_AIN1	ADC1:inm0 (-IN0)	CMPSSA2:inL (-IN)
ADC1_AIN2	ADC1:inp1 (+IN1)	CMPSSA3:inH (+IN)
ADC1_AIN3	ADC1:inm1 (-IN1)	CMPSSA3:inL (-IN)
ADC1_AIN4	ADC1:inp2 (+IN2)	CMPSSB2:inH/inL (+IN/-IN)
ADC1_AIN5	ADC1:inm2 (-IN2)	CMPSSB3:inH/inL (+IN/-IN)
ADC_CAL1	ADC1:inm3 (-IN3)	X
ADC_CAL0	ADC1:inp3 (+IN3)	X
ADC2 Channels		
ADC2_AIN0	ADC2:inp0 (+IN0)	CMPSSA4:inH (+IN)
ADC2_AIN1	ADC2:inm0 (-IN0)	CMPSSA4:inL (-IN)
ADC2_AIN2	ADC2:inp1 (+IN1)	CMPSSA5:inH (+IN)
ADC2_AIN3	ADC2:inm1 (-IN1)	CMPSSA5:inL (-IN)
ADC2_AIN4	ADC2:inp2 (+IN2)	CMPSSB4:inH/inL (+IN/-IN)
ADC2_AIN5	ADC2:inm2 (-IN2)	CMPSSB5:inH/inL (+IN/-IN)
ADC_CAL1	ADC2:inm3 (-IN3)	X
ADC_CAL0	ADC2:inp3 (+IN3)	X
ADC3 Channels		

Table 7-118. Connectivity between ADC Inputs to CMPSS Signals (continued)

Signal/Pin Name	ADC Input	CMPSS Input
ADC3_AIN0	ADC3:inp0 (+IN0)	CMPSSA6:inH (+IN)
ADC3_AIN1	ADC3:inm0 (-IN0)	CMPSSA6:inL (-IN)
ADC3_AIN2	ADC3:inp1 (+IN1)	CMPSSA7:inH (+IN)
ADC3_AIN3	ADC3:inm1 (-IN1)	CMPSSA7:inL (-IN)
ADC3_AIN4	ADC3:inp2 (+IN2)	CMPSSB6:inH/inL (+IN/-IN)
ADC3_AIN5	ADC3:inm2 (-IN2)	CMPSSB7:inH/inL (+IN/-IN)
ADC_CAL1	ADC3:inm3 (-IN3)	X
ADC_CAL0	ADC3:inp3 (+IN3)	X
ADC4 Channels		
ADC4_AIN0	ADC4:inp0 (+IN0)	CMPSSA8:inH (+IN)
ADC4_AIN1	ADC4:inm0 (-IN0)	CMPSSA8:inL (-IN)
ADC4_AIN2	ADC4:inp1 (+IN1)	CMPSSA9:inH (+IN)
ADC4_AIN3	ADC4:inm1 (-IN1)	CMPSSA9:inL (-IN)
ADC4_AIN4	ADC4:inp2 (+IN2)	CMPSSB8:inH/inL (+IN/-IN)
ADC4_AIN5	ADC4:inm2 (-IN2)	CMPSSB9:inH/inL (+IN/-IN)
ADC_CAL0	ADC4:inp3 (+IN3)	X
ADC_CAL1	ADC4:inm3 (-IN3)	X

Note

In the **ADC Input** column in ADC-CMPSS Signal Connectivity Table above, "inp" stands for positive inputs and "inm" stands for negative inputs.

7.5.4.3 Reference DAC

Each reference 12-bit DAC can be configured to drive a reference voltage into the negative input of the respective comparator. The reference 12-bit DAC output is internal only and cannot be observed externally.

Two sets of DACxVAL registers, DACxVALA and DACxVALS, are present for each reference 12-bit DAC. DACxVALA is a read-only register that actively controls the reference 12-bit DAC value. DACxVALS is a writable shadow register that loads into DACxVALA either immediately or synchronized with the next EPWMSYNCPER event. The high and low reference 12-bit DAC (DACx) can optionally source the register DACxVALA value from the ramp generator instead of the register DACxVALS.

The operating range of the reference 12-bit DAC is bounded by DACREF and VSSA. The high-voltage reference is VDDA by default, but the high voltage reference can be configured to be VDAC using the COMPDACCTL register. The reference 12-bit DAC is illustrated in Figure 7-146.

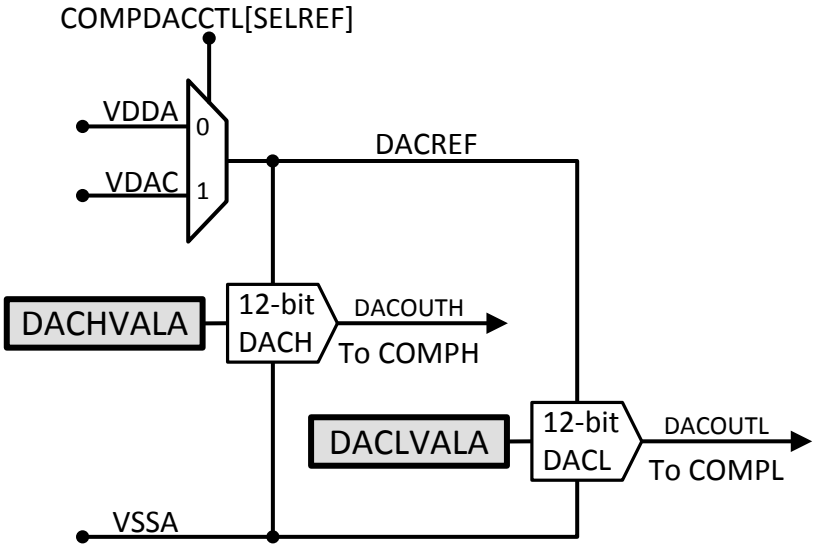


Figure 7-146. Reference DAC Block Diagram

The output of the reference 12-bit DAC can be calculated as:

$$DACOUT = \frac{DACVALA*DACREF*33}{4096} \times \frac{33}{18} \quad (10)$$

Note

- In the situations where both the DACH and DACL are driving the high and low comparators, a trip on one comparator can temporarily disturb the DAC output of the other comparator. The amount and length of time of this disturbance is specified in the device data sheet as “CMPSS DAC output disturbance” and “CMPSS DAC disturbance time”, respectively.
- Users must design their system carefully so that if the input signal crosses either DACH or DACL and trips the associated comparator, the input signal stays more than a “CMPSS DAC output disturbance” away from the other comparator trip point for “CMPSS DAC disturbance time”.
- The DACH setting must always be higher than the DACL setting. If the user is not using DACL, then DACLVALS register should be programmed to maximum, so that COMPL does not trip and affect DACH. In this case, there is no limitation on the DACHVALS setting. Accordingly, when not using the DACH, the user must set the DACHVALS register to the maximum.
- The CMPSS instance can be enabled before programming the reference DAC values.

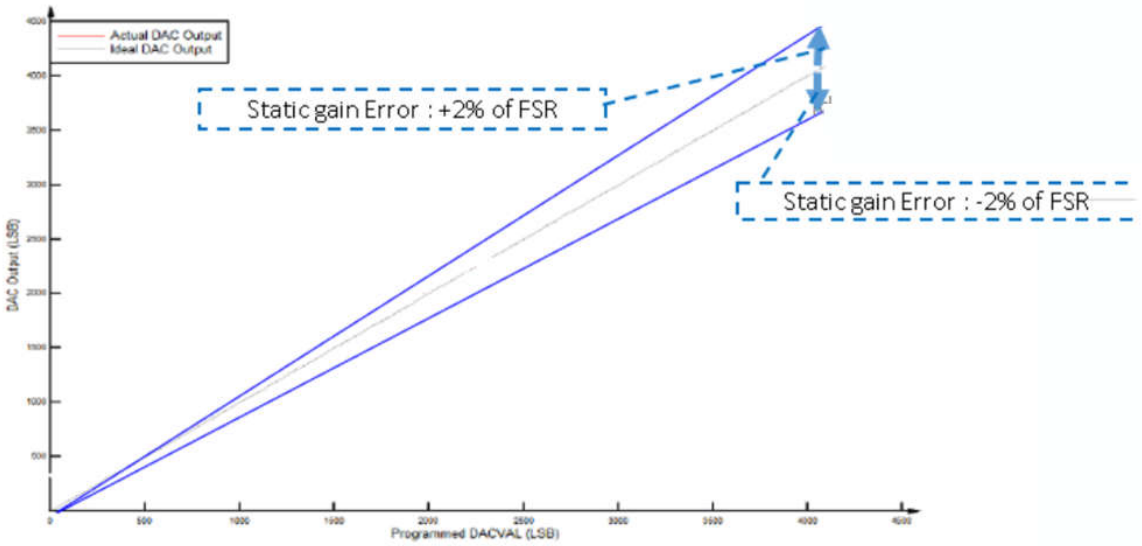


Figure 7-147. CMPSS DAC Static Offset

Note

CMPSS DAC threshold value drifts with change in temperature, so one needs to take care of the below characteristics for DAC calibration

- CMPSS DAC generates 0-3.3V for 12 bit code
- CMPSS comparator input (aka offset error) is -20 mV to + 20 mV
- CMPSS DAC Static Offset error is -45 mV to 45 mV and is shown in *CMPSS DAC Static Offset*
- CMPSS DAC Static Gain Error is -2 % to 2 % of FSR

7.5.4.4 Ramp Generator

This section discusses the characteristics and behavior of the ramp generator.

7.5.4.4.1 Ramp Generator Overview

The ramp generator produces a falling ramp input for the high-reference 12-bit DAC when selected. In this mode, the reference 12-bit DAC uses the most-significant 12 bits of the RAMPSTS countdown register as the input. The low 4 bits of the RAMPSTS countdown register effectively act as a prescale for the falling ramp rate configurable with RAMPxSTEPVALA

The ramp generator is enabled by setting DACSOURCE = 1. When DACSOURCE = 1 is selected, the value of RAMPSTS is loaded from RAMPxREFS and the register remains static until the selected EPWMSYNCPER signal is received. After receiving the selected EPWMSYNCPER signal, the value of RAMPDECVALA is subtracted from RAMPSTS on every subsequent SYSCLK cycle.

To prevent the subtraction from commencing a SYSCLK cycle after a EPWMSYNCPER event, the RAMPDLYA register that serves as a delay counter can be used to hold off the RAMPSTS subtraction. On receiving a EPWMSYNCPER event, the value of RAMPDLYA is decremented by one on every SYSCLK cycle until the register reaches zero. So, the RAMPSTS subtraction only begins when RAMPDLYA is zero.

7.5.4.4.2 Ramp Generator Behavior

The ramp generator makes state changes on every rising edge of DACSOURCE, EPWMSYNCPER, EPWMSYNCPER_H, and COMPHSTS.

On the rising edge of DACSOURCE: RAMPHREFA, RAMPHSTEPVALA, and RAMPDLYA are loaded with their shadow registers. RAMPSTS is loaded with RAMPHREFS.

On the rising edge of the selected EPWMSYNCPER_H: RAMPHREFA, RAMPHSTEPVALA, and RAMPDLYA are loaded with their shadow registers. RAMPSTS is loaded with RAMPHREFS and starts decrementing when RAMPDLYA counter reaches zero.

On the rising edge of COMPHSTS with RAMPLOADSEL = 1: RAMPHREFA, RAMPxREFA, RAMPxSTEPVALA, and RAMPDLYA are loaded with their shadow registers. RAMPSTS is loaded with RAMPxREFS and stops decrementing.

On the rising edge of COMPHSTS with RAMPLOADSEL = 0: RAMPSTS is loaded with RAMPHREFA and stops decrementing.

Additionally, if the value of RAMPSTS reaches zero, the RAMPSTS register remains static at zero until the next EPWMSYNCPER_H is received. These state changes are illustrated in the ramp generator block diagram in Figure 7-148.

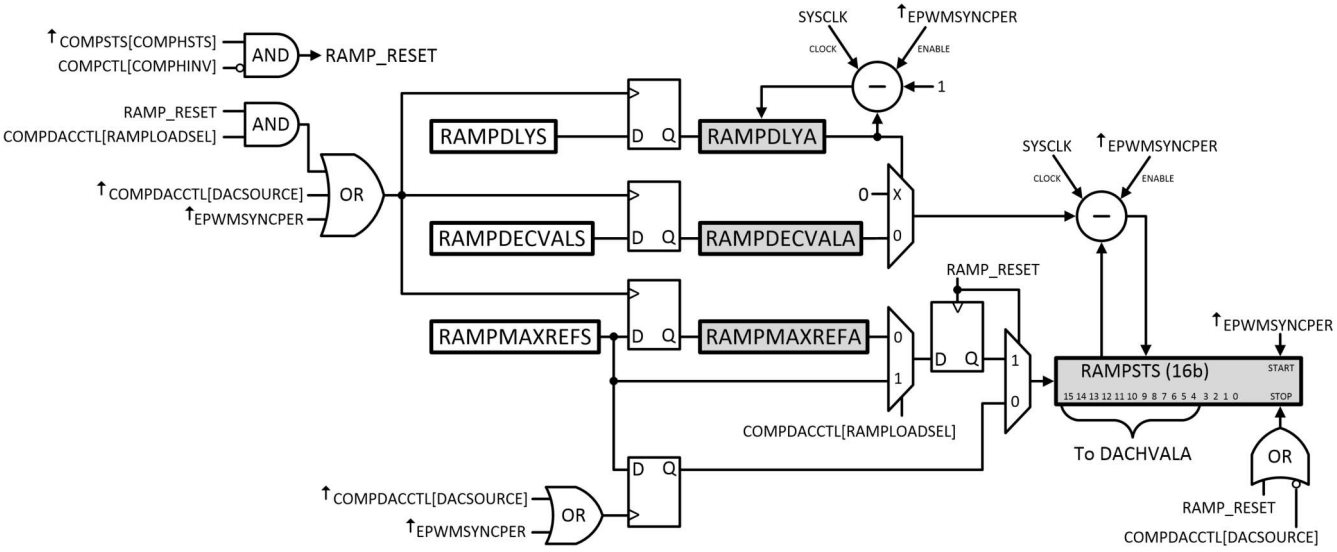


Figure 7-148. Ramp Generator Block Diagram

7.5.4.4.3 Ramp Generator Behavior at Corner Cases

Since the ramp generator makes state changes on every rising edge of EPWMSYNCPER_H and COMPHSTS, the following behavior can be expected on instances when these two events occur simultaneously or very close together.

Case 1: COMPHSTS rising edge occurs one or more cycles before EPWMSYNCPER_H rising edge. RAMPSTS stops decrementing on COMPHSTS rising edge event. RAMPSTS starts decrementing on EPWMSYNCPER_H rising edge event when RAMPDLYA reaches 0.

Case 2: COMPHSTS rising edge occurs simultaneously as EPWMSYNCPER_H rising edge. EPWMSYNCPER rising edge event takes precedence and RAMPSTS starts decrementing when RAMPDLYA reaches 0. COMPHSTS rising edge event is ignored and does not halt RAMPSTS.

Case 3: COMPHSTS rising edge occurs one or more cycles after EPWMSYNCPER_H rising edge but before RAMPDLYA reaches 0. RAMPSTS does not decrement when RAMPDLYA reaches 0.

Case 4: COMPHSTS rising edge occurs simultaneously as RAMPDLYA reaches 0 from EPWMSYNCPER_H rising edge. RAMPSTS does not decrement.

This behavior is also illustrated in the below image.

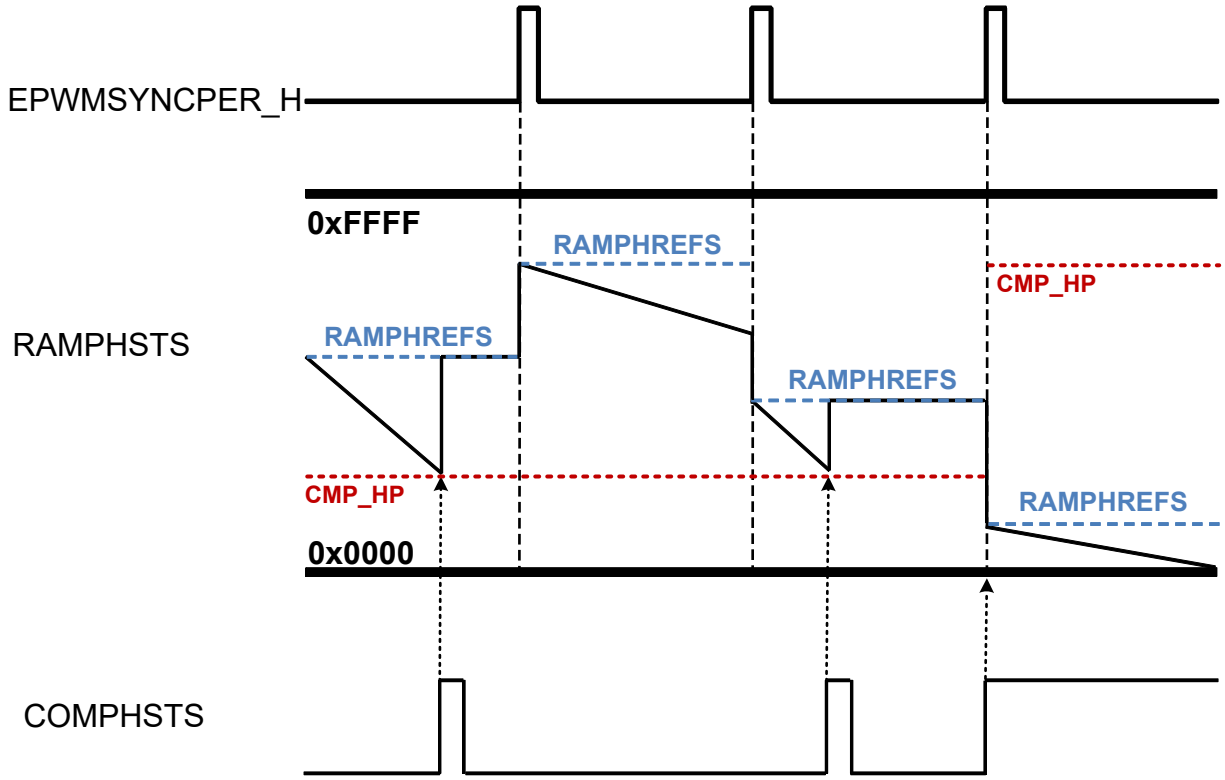


Figure 7-149. Ramp Generator Behavior

7.5.4.5 Digital Filter

The digital filter works on a window of FIFO samples (SAMPWIN) taken from the input. The filter output resolves to the majority value of the sample window, where majority is defined by the threshold (THRESH) value. If the majority threshold is not satisfied, the filter output remains unchanged.

For proper operation, the value of THRESH must be greater than SAMPWIN / 2 and less than or equal to SAMPWIN.

A prescale function (CLKPRESCALE) determines the filter sampling rate, where the filter FIFO captures one sample every prescale system clocks. Old data from the FIFO is discarded.

Note that for SAMPWIN, THRESH and CLKPRESCALE, the internal number used by the digital filter is + 1 in all cases. In essence, samples = SAMPWIN + 1, threshold = THRESH + 1 and prescale = CLKPRESCALE + 1.

A conceptual model of the digital filter is shown in Figure 7-150.

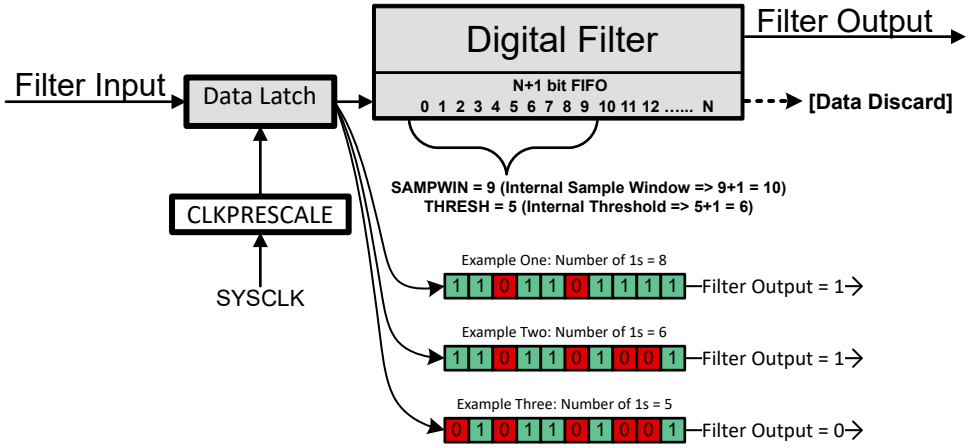


Figure 7-150. Digital Filter Behavior

Equivalent C code of the filter implementation is:

```
if (FILTER_OUTPUT == 0) {
    if (Num_1s_in_SAMPWIN >= THRESH) {
        FILTER_OUTPUT = 1;
    }
}
else {
    if (Num_0s_in_SAMPWIN >= THRESH) {
        FILTER_OUTPUT = 0;
    }
}
```

7.5.4.5.1 Filter Initialization Sequence

For proper operation of the digital filter, the following initialization sequence is recommended:

1. Configure and enable the comparator for operation.
2. Configure the digital filter parameters for operation:
   - Set SAMPWIN for the number of samples to monitor in the FIFO window.
   - Set THRESH for the threshold required for majority qualification.
   - Set CLKPRESCALE for the digital filter clock prescale value.
3. Initialize the sample values in the digital FIFO window by setting FILINIT.
4. Clear COMPSTS latch using COMPSTSCLR, if the latched path is desired.
5. Configure the CTRIP and CTRIPOUT signal paths.
6. If desired, configure the destination module, for example, ePWM, GPIO, and so on to accept the filtered signals.

7.5.4.6 Using the CMPSS

7.5.4.6.1 LATCHCLR, EPWMSYNCPER and EPWMBLANK Signals

The LATCHCLR signal holds the digital filter, synchronization block, and the latch output in reset (0) after the required delays. The LATCHCLR signal is activated in software using xLATCHCLR (x = H or L). The LATCHCLR signal can also be activated by EPWMSYNCPER when xSYNCCLREN (x = H or L) is set. If a longer LATCHCLR signal is required, the EPWMBLANK signal can be used to extend the LATCHCLR signal by setting BLANKEN.

EPWMSYNCPER and EPWMBLANK (BLANKWDW) come from the Time-Base and Digital Compare submodules of the EPWM, respectively. For a detailed description of how these two signals are generated, refer to the respective submodule section in the *Enhanced Pulse Width Modulator (ePWM)* chapter

The EPWMSYNCPER signal that loads DACxVALA when COMPDACCTL [SWLOADSEL] = 1 is a level trigger load. If TBCTR and TBPRD of the EPWM are both 0, EPWMSYNCPER is held at level high and DACxVALA is loaded immediately from DACxVALS irrespective of the value of COMPDACCTL [SWLOADSEL]. Due to this, configure the EPWM first before setting COMPDACCTL [SWLOADSEL] to 1.

Note

The name of the sync signal that the CMPSS receives from the EPWM has been updated from PWMSYNC to EPWMSYNCPER (SYNCPER/PWMSYNCPER/EPWMxSYNCPER) to avoid confusion with the other EPWM sync signals EPWMSYNCINEN and EPWMSYNCOUTEN. For a description of what are these signals, see the *Enhanced Pulse Width Modulator (ePWM)* chapter.

7.5.4.6.2 Synchronizer, Digital Filter, and Latch Delays

The synchronization block adds a delay of 1-2 sysclks. If the digital filter is bypassed (all filter settings are 0), the digital filter adds a delay of 2 sysclks. The latch adds 1 sysclk delay.

7.5.4.6.3 Calibrating the CMPSS Trip Levels

The CMPSS has two sources of offset errors: comparator offset error and compdac offset error. In the data sheet the comparator offset error is referred to as ***Input referred offset error*** and compdac offset error is referred to as ***Static offset error***. See the device data sheet for their values.

If both inputs of the comparator are driven from a pin, only the comparator offset error applies. However if the inverting input of the comparator is driven from the compdac, then only the compdac offset error applies. This is because the compdac offset error includes comparator offset error.

Due to the offset errors, the CMPSS must be calibrated to make sure trips happen at the expected levels. The following flow outlines how the calibration can be performed if the inverting input of the comparator is driven from the compdac.

Notes before calibration:

1. A static DC signal is required on the non-inverting input of the comparator.
2. Hysteresis can be disabled for calibration and can be re-enabled after calibration is complete.
3. A noisy input can make calibration difficult, so use the latch with non-zero filter settings depending on how noisy is the signal on the non-inverting input.

This approach sweeps down the compdac:

1. Set the starting compdac value to max, 0xFFF.
   - Optional: Instead of setting the starting compdac value to maximum, set to ***Vtarget*** + ***Static offset error*** + ***Margin***. Where ***Vtarget*** is the approximate DC voltage on the non-inverting input, ***Static offset error*** is the compdac offset error specification and ***Margin*** is some amount of guard band. This can lead to a faster calibration but only works if ***Vtarget*** is known. Alternatively, if ***Vtarget*** is unknown, the ADC can be used to convert ***Vtarget***.
2. Decrement compdac value by 1.
3. Wait for compdac to settle.
4. Clear latch.
5. Wait for possible latch set.
6. If latch is set, trip code is found exit.
   - Optional: The trip code can be double checked by:
     a. Increasing compdac value by 1.
     b. Clear latch.
     c. Wait for possible latch set.
     d. Latch can be unset.
7. If latch is unset, go back to step 2 and repeat.

It is also possible to calibrate the CMPSS, if both inputs of the comparator are driven from a pin. For this case, the flow stays the same but the voltage on the inverting pin of the comparator is swept externally.

7.5.4.6.3.1 CMPSS Hysteresis

The CMPSS DAC is used as the reference to determine how much hysteresis to apply. Therefore, hysteresis scales with the CMPASS DAC reference voltage.

Hysteresis can be disabled for calibration, and Hysteresis can be re-enabled after calibration is complete through COMPLHYS and COMPHHYS. Each of these is a 4 bit field of CMPSS CONFIG1 registers, and bit 2 of the field follow the behavior shown in Figure 7-151

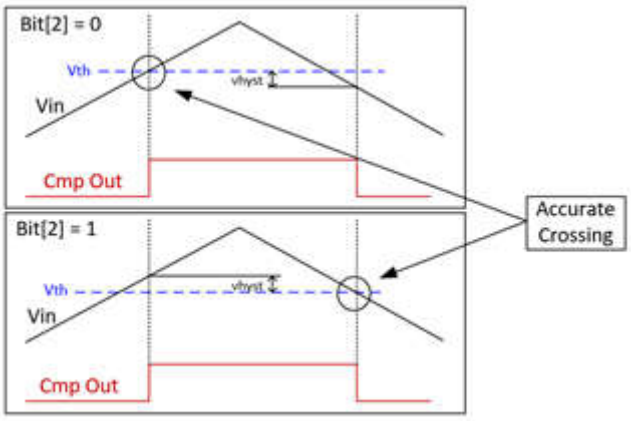


Figure 7-151. Bit 2 Crossing

7.5.4.7 Enabling and Disabling the CMPSS Clock

If the clock to the CMPSS module is disabled while the comparator is active, the following behavior can be expected:

- The comparator remains unaffected and continues to trip from voltages on the inputs.
- If the reference 12-bit DAC is driving the negative input of the comparator, the voltage on the negative input remains static and unaffected but DACVALA can no longer be updated from the ramp generator or DACVALS.
- The ramp generator, synchronize block and digital filter freeze on their current states.

Enabling the clock to the CMPSS restores the clock to the state before the clock was disabled.

7.5.4.8 CMPSS Programming Guide

Driver Information

Driver features are available at the CMPSS driver page.

Software API Information

The SDFM driver provides an API to configure the CMPSS module. Full documentation is located on APIs for CMPSS

Example Usage

The below links shows an example on how to use SDFM

- CMPSS Asynchronous trip

7.5.5 Buffered Digital-to-Analog Converter (DAC)

The buffered digital-to-analog converter (DAC) is an analog module that can output a programmable, arbitrary reference voltage.

7.5.5.1 Introduction

The buffered DAC module consists of an internal 12-bit DAC and an analog output buffer that is capable of driving an external load. The buffered DAC is a general-purpose DAC that can be used to generate a DC voltage in addition to AC waveforms such as sine waves, square waves, triangle waves and so forth. Software writes to the DAC value register can take effect immediately or can be synchronized with EPWMSYNCPER events.

Note

For the load conditions of the buffered DAC, see the device-specific data sheet.

7.5.5.1.1 Features

Each buffered DAC has the following features:

- 12-bit programmable internal DAC
- Selectable reference voltage source
- Pull-down resistor on output
- Ability to synchronize with EPWMSYNCPER

7.5.5.1.2 Block Diagram

The block diagram for the buffered DAC is shown in Figure 7-152.

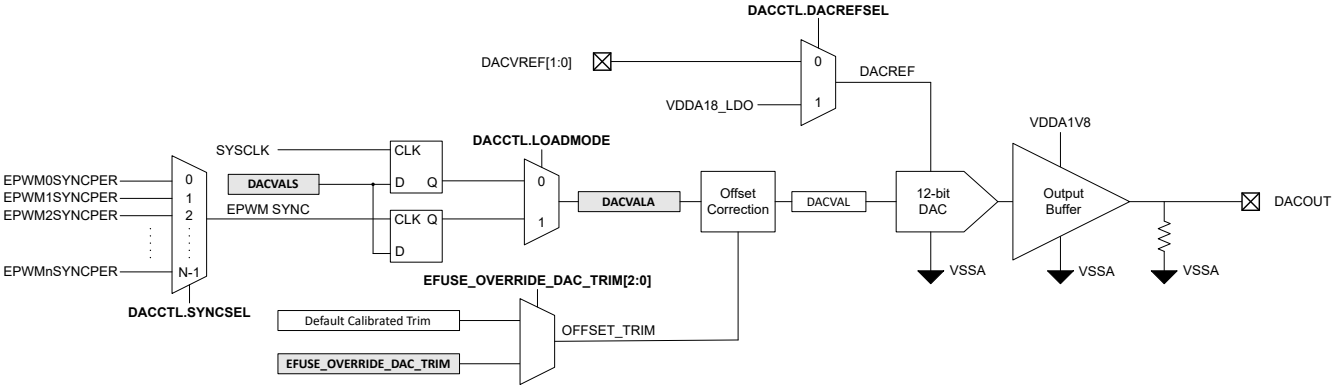


Figure 7-152. DAC Module Block Diagram

7.5.5.2 Using the DAC

As seen in Figure 7-152 two sets of DACVAL registers, DACVALA and DACVALS, are present in the buffered DAC module. DACVALA is a read-only register that actively controls the buffered DAC value. DACVALS is a writable shadow register that loads into DACVALA either immediately or synchronized with the next EPWMSYNCPER event. DACVALA update source is selected by the CONTROLSS_DAC0_DACCTL register LOADMODE bit. The power-on default of LOADMODE = 0, which selects the immediate update mode.

Note

If the clock to the buffered DAC is disabled while the buffered DAC is outputting a voltage, the output voltage remains unaffected, but DACVALA and DACVALS is no longer updated with register writes. Enabling the clock to the buffered DAC restores the DAC to the state before the clock was disabled.

The internal DAC reference voltage source, DACREF, is selectable between DACVREF and VDDA18_LDO. The CONTROLSS_DAC0 register DACREFSEL bit selects between these two VREF sources. The DACVREF source routes to the DACVREF pins of the device. The VDDA18_LDO source selects an internal route to the on-die 1.8V analog LDO output. In all normal applications the DACVREF pins should be used as the VREF source. The VDDA18_LDO VREF source option is present only for diagnostic or debug purposes. The power-on default of DACREFSEL = 0, which selects the DACVREF source pins.

Before the selected DACVALA register value is applied to the DAC, an additional calibration offset value is applied. During production DACOUT is calibrated to a nominal 1.8V VREF source offset with the calibration

offset stored in e-fuse for use by the buffered DAC. The power-on default options select this pre-calibrated offset value.

Assuming this pre-calibrated, default offset value is used, the output voltage DACOUT (in volts) is calculated with the following equation:

$$DACOUT = (DACVALA/4096) \cdot DACVREF \cdot (33 \ / \ 18) \quad (11)$$

See the below Section 7.5.5.2.2 for more information on the DAC offset adjustment.

Note

The output buffer of the buffered DAC can exhibit non-linear behavior near the supply rails (VDDA18/ VSSA). To determine the linear range of the buffered DAC, see the device-specific data sheet.

7.5.5.2.1 Initialization Sequence

The following sequence of steps will setup the buffered DAC for basic operation.

1. Enable the buffered DAC clock. See the Clock Selection for CONTROLSS_PLL clock controls.
2. Select the DACREF source with DACREFSEL bit in the DACCTL register.
3. Power up the buffered DAC with DACOUTEN in the CONTROLSS_DAC0_DACOUTEN register.
4. Wait for the power-up time to elapse before writing a new voltage value into the CONTROLSS_DAC0_DAC_DACVALS register. See the device-specific datasheet to determine the power-up time of the buffered DAC.

Note

For predictable behavior of the buffered DAC, two consecutive writes to DACVALS should not be spaced 1024 codes or more apart. Consecutive DACVALS values that are 1024 codes allow for the required settling time of the buffered DAC, resulting in

7.5.5.2.2 DAC Offset Adjustment

Zero offset error is defined as the difference between the voltage at midcode (2048) and 0.9V (for 1.8V reference voltage). DAC offset error is calibrated using an external 1.8V reference voltage and loaded into the DAC offset trim register by default. If the DAC is used at any reference voltage other than 1.8V, the offset trim must be adjusted to ensure that offset error performance stays within the device-specific data manual limits. The default DAC trim can be overridden from the EFUSE_OVERRIDE_DAC_TRIM register of TOP_CTRL.DAC0_TRIM register. The DAC register space (CONTROLSS_DAC) is not used for any DAC trim function. Changing the default DAC offset trim using EFUSE_OVERRIDE_DAC_TRIM register is not recommended and only meant for debug or diagnostic purpose.

7.5.5.2.3 EPWMSYNCPER Signal

The EPWMSYNCPER signal comes from the Time-Base submodule of the EPWM. For a detailed description of how this signal is generated, refer to Enhanced Pulse Width Modulator (ePWM).

When DACCTL.LOADMODE = 1, the selected EPWMSYNCPER signal will load new DACVALA values. The EPWMSYNCPER signal operates as a high logic-level trigger load. If TBCTR and TBPRD of the EPWM are both 0, EPWMSYNCPER is held at a high level and DACVALA is immediately loaded from DACVALS irrespective of the value of DACCTL.LOADMODE. To control the timing of this initial EPWMSYNCPER triggered load the EPWM and EPWMSYNCPER output should be configured prior to setting DACCTL.LOADMODE = 1.

Note

When using EPWMSYNCPER to load in new DACVALA values, unexpected value changes may be observed in the DAC active value register and resulting DAC output. In this case the DACVAL register is likely receiving an intermediate value. The DACVALA register will take on the full programmed value after 1 or more EPWM periods. Dividing the EPWM clock by 2, 4, or 8, to slow down the EPWM period and can fix this issue.

7.5.5.3 Lock Registers

The CONTROLSS_DAC0_DACLOCK register is provided to prevent spurious writes from modifying the CONTROLSS_DAC0_DACCTL, CONTROLSS_DAC0_DACVALS, and CONTROLSS_DAC0_DACOUTEN registers. Once a register is protected through CONTROLSS_DAC0_DACLOCK, write access are locked out until the device is reset.

Note

Once a CONTROLSS_DAC0_DACLOCK field is set only a full power on reset of the device will clear the lock and enable modification of the locked register.

7.5.5.4 DAC Programming Guide

Drive Information

Driver features are available at the DAC driver page.

Software API Information

The DAC driver provides an API to configure the DAC module. Full documentation is located on APIs for DAC.

Example Usage

The below links show examples on how to use DAC

- DAC Constant Voltage
- DAC Ramp Wave
- DAC Random Voltage
- DAC Sine DMA
- DAC Sine Wave
- DAC Square Wave

7.5.6 Enhanced Pulse Width Modulator (ePWM)

The enhanced pulse width modulator (ePWM) peripheral is a key element in controlling many of the power electronic systems found in both commercial and industrial equipment. These systems include digital motor control, switch mode power supply control, uninterruptible power supplies (UPS), and other forms of power conversion. The ePWM peripheral can also perform a digital-to-analog (DAC) function, where the duty cycle is equivalent to a DAC analog value; it is sometimes referred to as a power DAC.

This chapter is applicable for ePWM type 5. Type 5 EPWM is fully compatible with type 4 EPWM.

7.5.6.1 Introduction

This chapter includes an overview and information about each submodule:

- Time Base (TB) Submodule
- Counter Compare (CC) Submodule
- Action Qualifier (AQ) Submodule
- Dead-Band Generator (DB) Submodule
- PWM Chopper (PC) Submodule
- Trip Zone (TZ) Submodule
- Diode Emulation (DE) Submodule
- Minimum Dead-Band (MINDB) and Illegal Combo Logic (ICL) Submodule
- Event Trigger (ET) Submodule
- Digital Compare (DC) Submodule

An effective PWM peripheral must be able to generate complex pulse width waveforms with minimal CPU overhead or intervention and must 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 submodules with separate resources 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 the operation quickly.

In this document, the letter x within a signal or submodule 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 instance. Thus, EPWM1A and EPWM1B belong to ePWM1 and likewise EPWM4A and EPWM4B belong to ePWM4.

The ePWM Type 5 is functionally compatible to Type 4. Type 5 has the following enhancements in addition to the Type 4 features:

- **PWM SYNC Related Enhancements:** Additional external sync option is added in to the EPWMSYNCSEL register. This allows for the configuration of up to 3 independent sync chains with external sync options.
- **Linking and Global Load Enhancements:** DBRED:DBREDHR and DBREDHR and DBFED:DBFEDHR have the ability to be linked across ePWM modules.

  Global load pulse selection for shadow to active load can now occur when the time-base counter equals CMPCU, CMPCD, CMPDU, or CMPDD.
- **XCMP Complex Waveform Generator:** XCMP mode has been added to allow for generation of multiple ePWM pulses, with high resolution, in a given ePWM cycle. Up to 8 new compare registers are added to achieve this functionality.
- **Digital Compare Submodule Enhancements:** Event detection within the digital compare capture module is able to detect an occurrence of a trip event in a configured time window.

  Pulse selection for blanking and capture alignment now includes a blanking window mix selection (BLANKPULSEMIX). This is added for LLC topologies where blanking window settings need to be changed on the fly - providing greater configurability to do this.
- **Trip-Zone Submodule Enhancements:** A CAPEVENT signal can generate a CBC or One-shot trip event.
- **Diode Emulation Submodule:** The diode emulation mode was added to provide hardware features and the necessary hooks into other IPs to implement a robust diode mode sense and control in a noisy environment.
- **Minimum Dead-Band and Illegal Combo Logic Submodule:** The minimum dead-band logic was added to provide the ability to configure the minimum dead-band duration between a complimentary set of ePWMs.

  To detect and make sure that under no circumstances, the ePWM states result in potentially hazardous combinations, a Look Up Table (LUT) has been added that can be used to re-configure the ePWM outputs.
- **Event Trigger Submodule Enhancements:** To enable unevenly spaced over-sampling of the ePWM period, the event trigger module trigger select is modified such that multiple events can trigger SOCA, SOCB, and INT events (ETINTMIX).
- **OTTO-HRPWM Enhancement:** OTTO-HRPWM module now has 3 additional delay lines for CMPBHR, DBREDHR, DBFEDHR

The ePWM Type 4 is functionally compatible to Type 2 (a Type 3 does not exist). Type 4 has the following enhancements in addition to the Type 2 features:

- **Register Address Map:** Additional registers are required for new features on ePWM Type 4. The ePWM register address space has been remapped for better alignment and easy usage.
- **Delayed Trip Functionality:** Changes have been added to achieve deadband insertion capabilities to support, for example, delayed trip functionality needed for peak current mode control type application scenarios. This has been accomplished by allowing comparator events to go into the Action Qualifier as a trigger event (Events T1 and T2). If comparator T1 / T2 events are used to edit the PWM, changes to the PWM waveform do not take place immediately. Instead, the waveform synchronizes to the next TBCLK.
- **Dead-Band Generator Submodule Enhancements:** Shadowing of the DBCTL register to allow dynamic configuration changes.
- **One Shot and Global Load of Registers:** The ePWM Type 4 allows one shot and global load capability from shadow to active registers to avoid partial loads in, for example, multiphase applications. ePWM Type 4 also allows a programmable prescale of shadow to active load events. ePWM Type 4 Global Load can simplify ePWM software by removing interrupts and ensuring that all registers are loaded at the same time.
- **Trip-Zone Submodule Enhancements:** Independent flags have been added to reflect the trip status for each of the TZ sources. Changes have been made to the trip-zone submodule to support certain power converter switching techniques like valley switching.
- **Digital Compare Submodule Enhancements:** Blanking window filter register width has been increased from 8 to 16 bits. DCCAP functionality has been enhanced to provide more programmability.
- **PWM SYNC Related Enhancements:** The ePWM Type 4 allows PWM SYNCOUT generation based on CMPC and CMPD events. These events can also be used for PWMSYNC pulse selection.

The ePWM Type 2 is fully compatible to Type 1. Type 2 has the following enhancements in addition to the Type 1 features:

- **High-Resolution Dead-Band Capability:** High-resolution capability is added to dead-band RED and FED in half-cycle clocking mode.
- **Dead-Band Generator Submodule Enhancements:** The ePWM Type 2 has features to enable both RED and FED on either PWM outputs. Provides increased dead band with 14-bit counters and dead-band / dead-band high-resolution registers are shadowed
- **High-Resolution Extension available on ePWMxB outputs:** Provides the ability to enable high-resolution period and duty cycle control on ePWMxB outputs. This is discussed in more detail in High-Resolution Pulse Width Modulator (HRPWM).
- **Counter Compare Submodule Enhancements:** The ePWM Type 2 allows interrupts and SOC events to be generated by additional counter compares CMPC and CMPD.
- **Event Trigger Submodule Enhancements:** Prescaling logic to issue interrupt requests and ADC start of conversion expanded up to every 15 events. It allows software initialization of event counters on SYNC event.
- **Digital Compare Submodule Enhancements:** Digital Compare Trip Select logic [DCTRIPSEL] has up to 12 external trip sources selected by the Input X-BAR logic in addition to an ability to OR all of them (up to 14 [external and internal sources]) to create the respective DCxEVTs.
- **Simultaneous Writes to TBPRD and CMPx Registers:** This feature allows writes to TBPRD, CMPA:CMPAHR, CMPB:CMPBHR, CMPC and CMPD of any ePWM module to be tied to any other ePWM module, and also allows all ePWM modules to be tied to a particular ePWM module if desired.
- **Shadow to Active Load on SYNC of TBPRD and CMP Registers:** This feature supports simultaneous writes of TBPRD and CMPA/B/C/D registers.

The ePWM Type 1 is fully compatible to Type 0. Type 1 has the following enhancements in addition to the Type 0 features:

- **Increased Dead-Band Resolution:** Dead-band clocking has been enhanced to allow half-cycle clocking to double resolution.
- **Enhanced Interrupt and SOC Generation:** Interrupts and ADC start-of-conversion can now be generated on both the TBCTR == zero and TBCTR == period events. This feature enables dual edge PWM control. Additionally, the ADC start-of-conversion can be generated from an event defined in the digital compare submodule.
- **High-Resolution Period Capability:** Provides the ability to enable high-resolution period. This is discussed in more detail in High-Resolution Pulse Width Modulator (HRPWM).

- **Digital Compare Submodule:** The digital compare submodule enhances the event triggering and trip zone submodules by providing filtering, blanking and improved trip functionality to digital compare signals. Such features are essential for peak current mode control and for support of analog comparators.

Note

The name of the sync signal that goes to the CMPSS has been updated from PWMSYNC to EPWMSYNCPER (SYNCPER/PWMSYNCPER/EPWMxSYNCPER) to avoid confusion with the other EPWM sync signals EPWMSYNCI and EPWMSYNCO. For a description of these signals, see Table 7-120.

7.5.6.1.1 Submodule Overview

The ePWM module represents one complete PWM channel composed of two PWM outputs: EPWMxA and EPWMxB. Multiple ePWM modules are instanced within a device as shown in *Multiple ePWM Modules* Each ePWM instance is identical with one exception. Some instances include a hardware extension that allows more precise control of the PWM outputs. This extension is the high-resolution pulse width modulator (HRPWM) and is described in High-Resolution Pulse Width Modulator (HRPWM). See the device data sheet to determine which ePWM instances include this feature. Each ePWM module is indicated by a numerical value starting with 0. For example, ePWM0 is the first instance and ePWM2 is the third instance in the system and ePWMx indicates any instance.

The ePWM modules are chained together by way of a clock synchronization scheme that allows them to operate as a single system when required. Additionally, this synchronization scheme can be extended to the capture peripheral submodules (eCAP). The number of submodules is device-dependent and based on target application needs. Submodules can also operate standalone.

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.
- All events can trigger both CPU interrupts and ADC start of conversion (SOC)
- Programmable event prescaling minimizes CPU overhead on interrupts.
- PWM chopping by high-frequency carrier signal, useful for pulse transformer gate drives.

Each ePWM module is connected to the input/output signals shown in Figure 7-154. The signals are described in detail in subsequent sections.

Each ePWM module consists of eight submodules and is connected within a system by way of the signals shown in Figure 7-154. The order in which the ePWM modules are connected can differ from what is shown in the figure. See Time-Base Counter Synchronization for the synchronization scheme for a particular device.

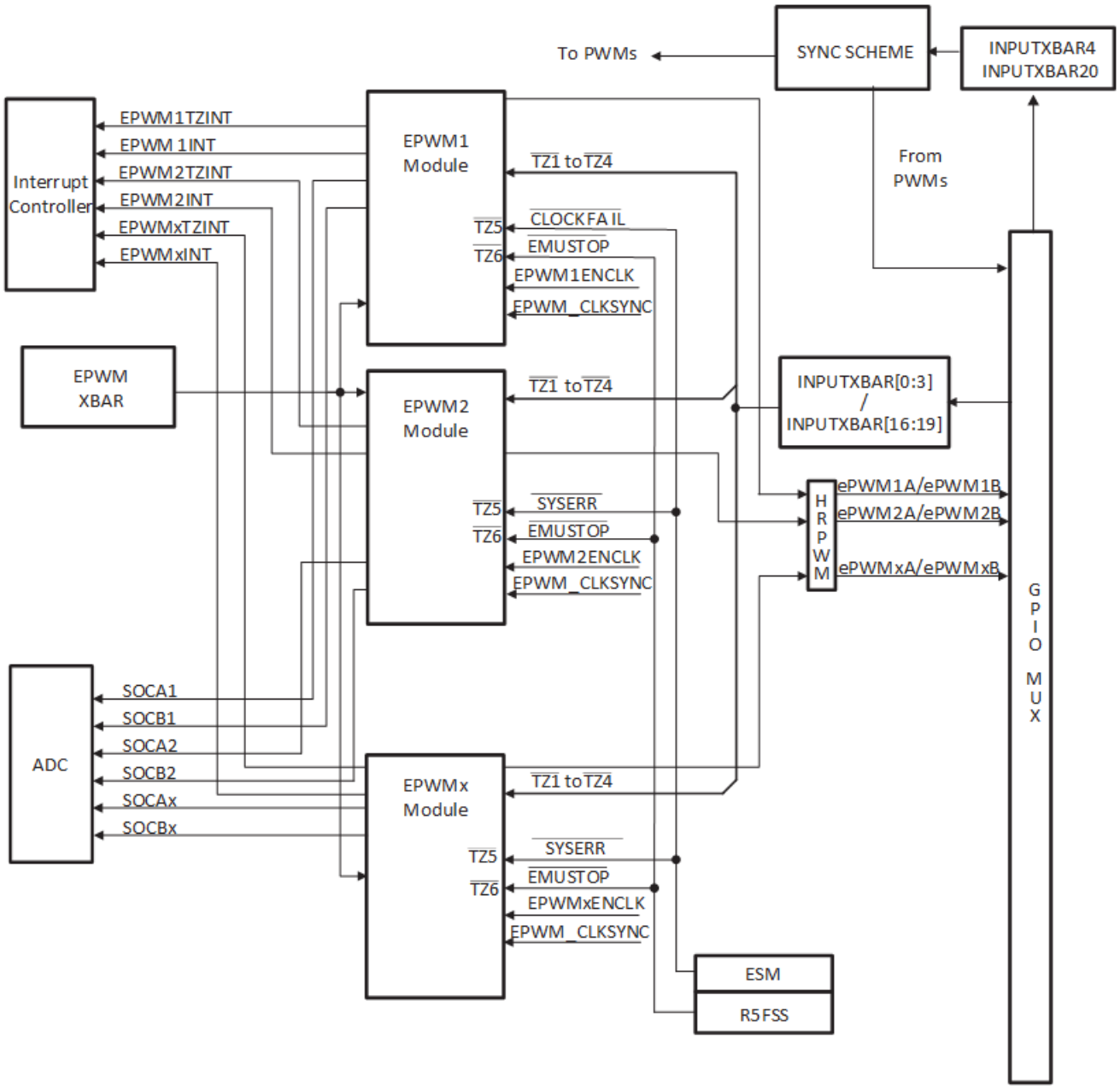


Figure 7-153. Multiple ePWM Modules

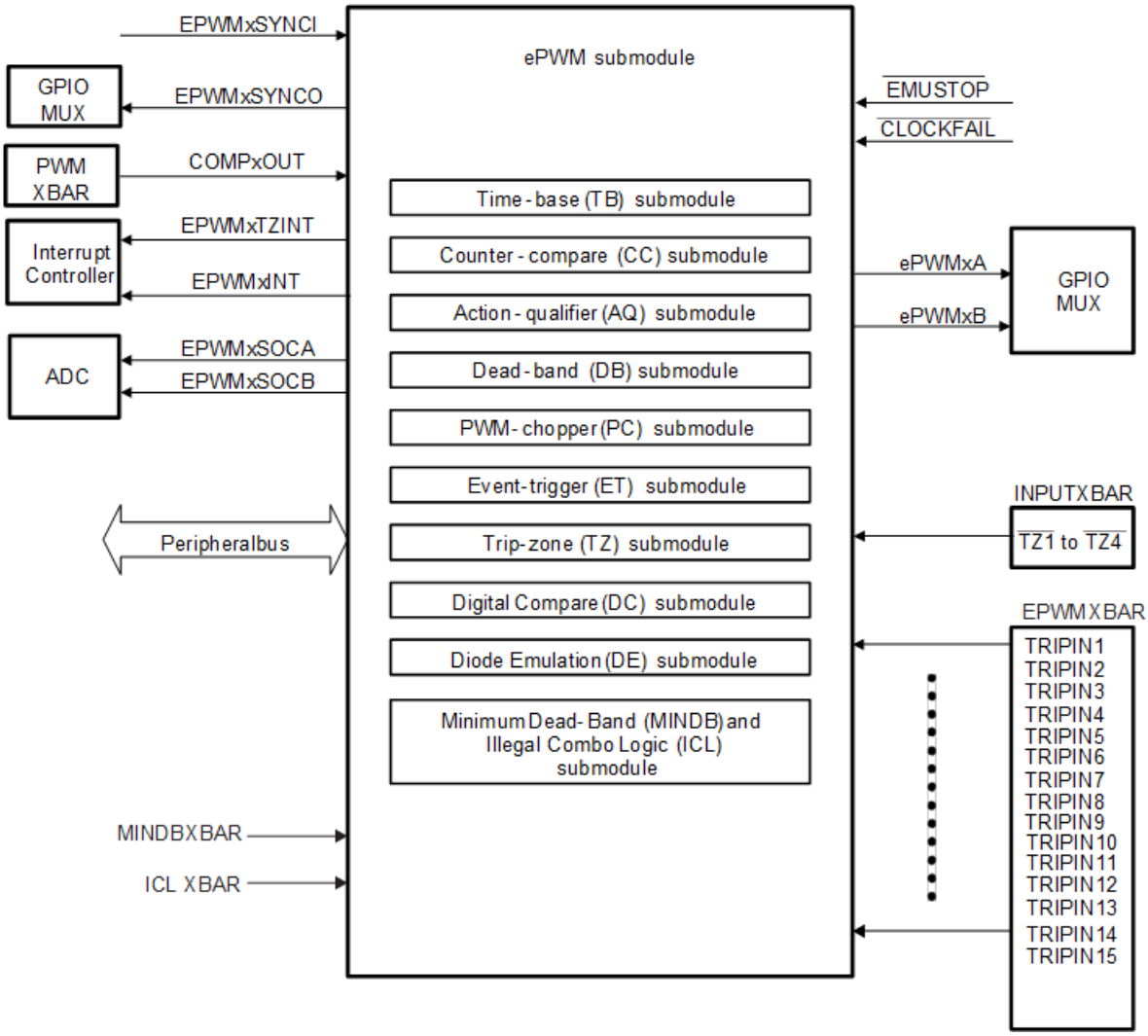


Figure 7-154. Submodules and Signal Connections for an ePWM Module

Figure 7-155 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
- **Trip-zone signals ($\overline{TZ1}$ to $\overline{TZ6}$)**

  These input signals alert the ePWM module of fault conditions external to the ePWM module. Each submodule on a device can be configured to either use or ignore any of the trip-zone signals
- **Time-base synchronization input (EPWMxSYNCI), output (EPWMxSYNCO), and peripheral (EPWMxSYNCPER) signals**

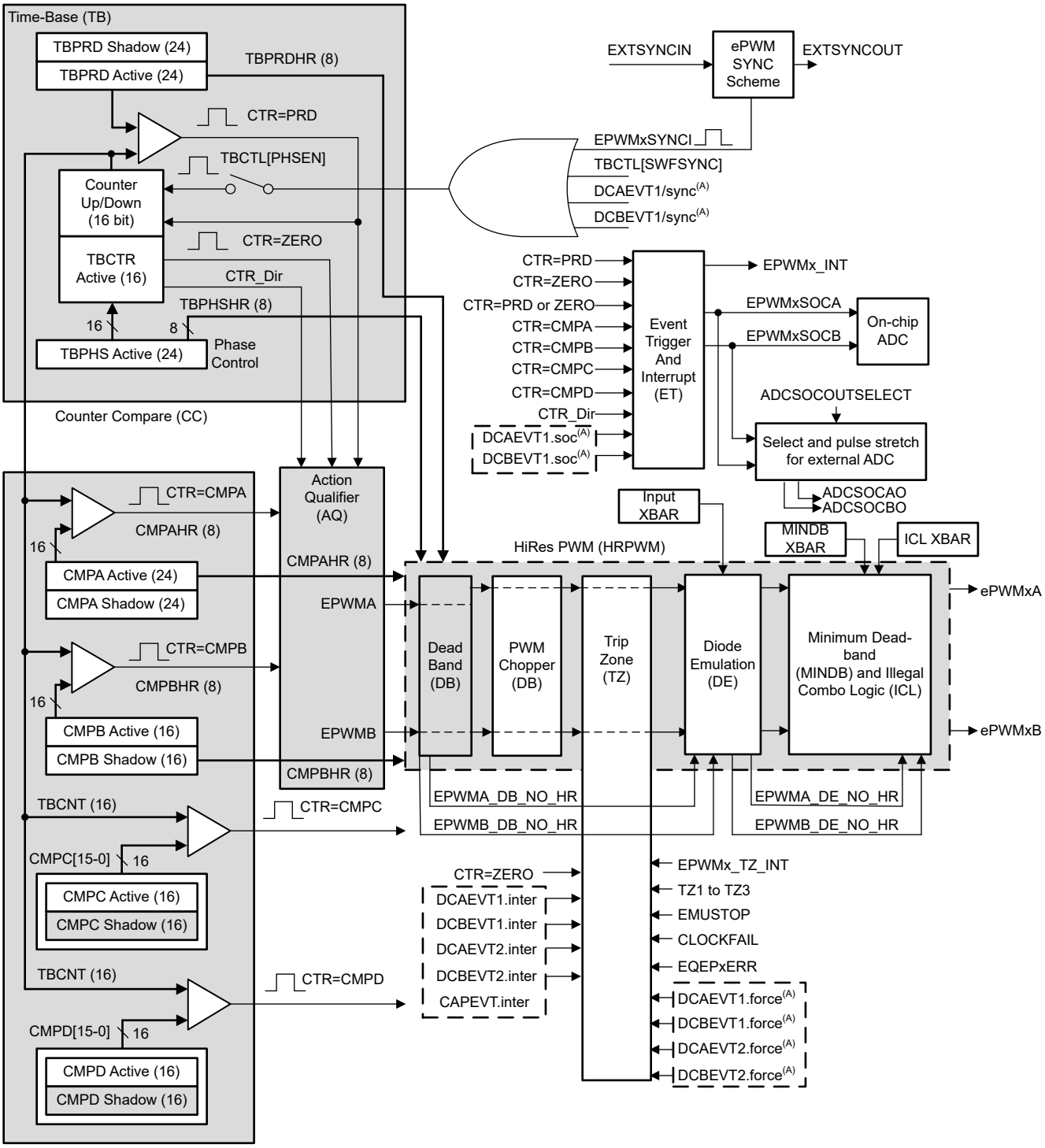
  For more information, see *Time-Base Counter Synchronization*

  Each ePWM module also generates another PWMSYNC signal called EPWMxSYNCPER. EPWMxSYNCPER goes to the CMPSS for synchronization purposes. Functionality is configured using the HRPCTL register, but has no relation with the HRPWM. For more information on how EPWMxSYNCPER is used by the CMPSS, see their respective chapters.
- **ADC start-of-conversion signals (EPWMxSOCA and EPWMxSOCB)**

  Each ePWM module has two ADC start of conversion signals. Any ePWM module can trigger a start of conversion. Whichever event triggers the start of conversion is configured in the event-trigger submodule of the ePWM.
- **Comparator output signals (COMPxOUT)**

  Output signals from the comparator module can be fed through EPWM X-BAR to one or all of the and in conjunction with the trip zone signals can generate digital compare events.
- **Peripheral bus**

  The peripheral bus is 32-bits wide and allows both 16-bit and 32-bit writes to the ePWM register file.



A. These events are generated by the ePWM Digital Compare (DC) submodule based on the levels of the TRIPIN inputs.

Figure 7-155. ePWM Modules and Critical Internal Signal Interconnects

7.5.6.1.2 EPWM Related Collateral

Foundational Materials

- C2000 Academy - EPWM
- *Real-Time Control Reference Guide*
  - Refer to the EPWM section

Getting Started Materials

- *C2000 ePWM Developer's Guide Application Report*
- Enhanced Pulse Width Modulator (ePWM) Training for C2000 MCUs (Video)
- *Flexible PWMs Enable Multi-Axis Drives, Multi-Level Inverters Application Report*
- Getting Started with the C2000 ePWM Module (Video)
- *Using PWM Output as a Digital-to-Analog Converter on a TMS320F280x Digital Signal Control Application Report*
  - Chapters 1 to 6 are Fundamental material, derivations, and explanations that are useful for learning about how PWM can be used to implement a DAC. Subsequent chapters are Getting Started and Expert material for implementing in a system.
- *Using the Enhanced Pulse Width Modulator (ePWM) Module Application Report*

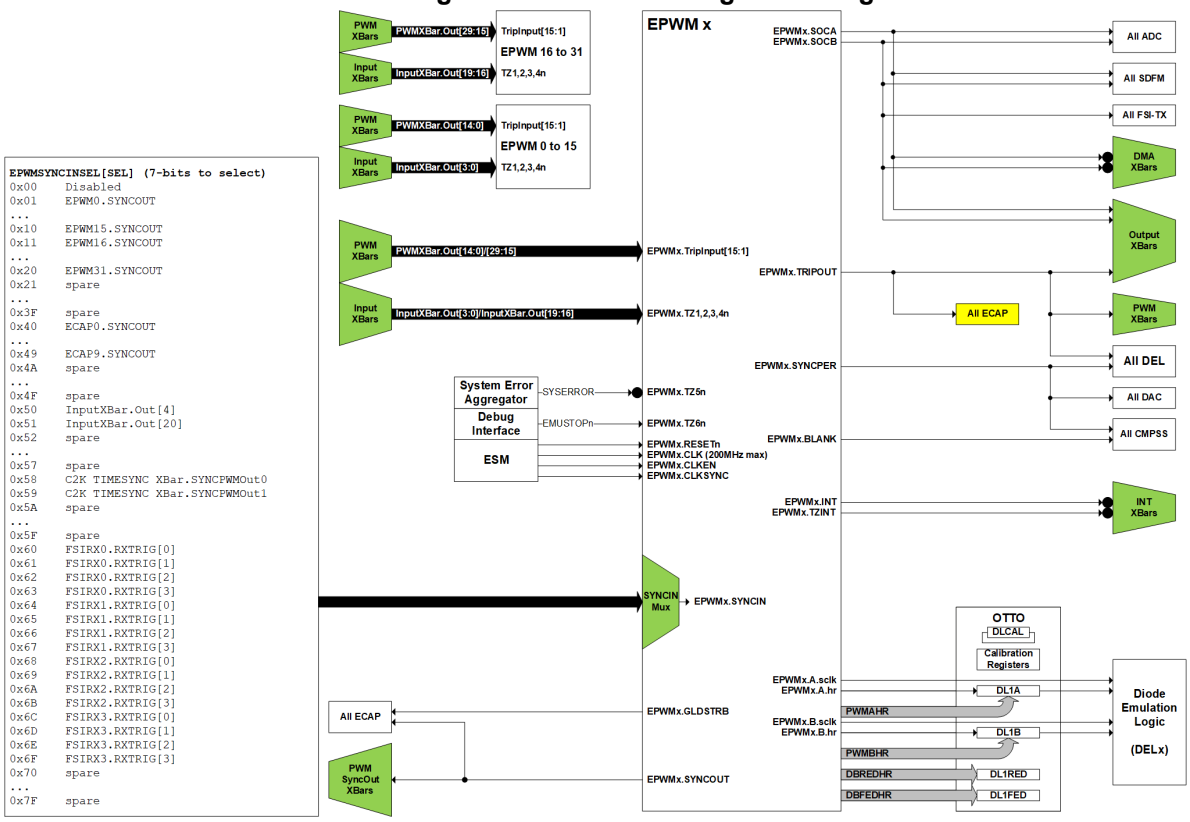
Expert Materials

- C2000 real-time microcontrollers - Reference designs
  - See TI designs related to specific end applications used.
- *Leverage New Type ePWM Features for Multiple Phase Control Application Report*

7.5.6.2 EPWM Integration

There are 32x EPWM modules integrated in the device. The diagram below provides a visual representation of the device integration details.

Figure 7-156. ePWM Integration Diagram



7.5.6.3 ePWM Modules Overview

32 submodules are included in every ePWM peripheral. Each of these submodules performs specific tasks that can be configured by software.

Table 7-119 lists the key submodules together with a list of their main configuration parameters. For example, if you need to adjust or control the duty cycle of a PWM waveform, see the counter-compare submodule in Section 7.5.6.5 for relevant details.

Table 7-119. Submodule Configuration Parameters

Submodule	Configuration Parameter or Option
Time Base (TB)	• Scale the time-base clock (TBCLK) relative to the ePWM clock (EPWMCLK). • Configure the PWM time-base counter (TBCTR) frequency or period. • Set the mode for the time-base counter: – 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. • Simultaneous writes to the TBPRD registers on all PWM's corresponding to the configuration on EPWMXLINK. • Configure how the time-base counter behaves when the device is halted by an emulator. • Specify the source for the synchronization output of the ePWM module • Configure one shot and global load of registers in this module.

Table 7-119. Submodule Configuration Parameters (continued)

Submodule	Configuration Parameter or Option
Counter Compare (CC)	• Specify the PWM duty cycle for output EPWMxA and output EPWMxB • Specify the time at which switching events occur on the EPWMxA or EPWMxB output • Specify the programmable delay for interrupt and SOC generation with additional comparators • Simultaneous writes to the CMPA, CMPB, CMPC, CMPD registers on all PWM's corresponding to the configuration on EPWMXLINK. • Configure one shot and global load of registers in this module. • Generate up to four pulses in one ePWM period through the complex waveform (XCMP) mode feature
Action Qualifier (AQ)	• Specify the type of action taken when a time-base counter-compare, trip-zone submodule, or comparator event occurs: – No action taken – Output EPWMxA and EPWMxB switched high – Output EPWMxA and EPWMxB switched low – Output EPWMxA and EPWMxB toggled • Force the PWM output state through software control • Configure and control the PWM dead band through software • Configure one shot and global load of registers in this module.
Dead-Band Generator (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. • Option to enable half-cycle clocking for double resolution. • Allow EPWMxB phase shifting with respect to the EPWMxA output. • Configure one shot and global load of registers in this module. • Simultaneous writes to the DBRED, DBREDHR, DBFED, DBFEDHR registers on all PWM's corresponding to the configuration on EPWMXLINK2.
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.
Trip Zone (TZ)	• Configure the ePWM module to react to one, all, or none of the trip-zone signals or digital compare events. • Specify the trip action taken when a fault occurs: – Force EPWMxA and EPWMxB high – Force EPWMxA and EPWMxB low – Force EPWMxA and EPWMxB to a high-impedance state – Configure EPWMxA and EPWMxB to ignore any trip condition. • Configure how often the ePWM reacts to each trip-zone signal: – One-shot – Cycle-by-cycle • Enable the trip-zone to initiate an interrupt. • Bypass the trip-zone module entirely. • Programmable option for cycle-by-cycle trip clear • If desired, independently configure trip actions taken when time-base counter is counting down.
Diode Emulation	• Choose any of the comparator outputs as trips to detect entry into DE mode. • Monitor the DE mode duration and generate a trip event to PWMs. • Ability to switch the comparator thresholds, dynamically in hardware upon DE mode entry. • Cycle-by-cycle and one-shot modes of clearing/de-evaluating the DE condition.

Table 7-119. Submodule Configuration Parameters (continued)

Submodule	Configuration Parameter or Option
Minimum Dead-Band(MINDB) and Illegal Combo Logic (ICL)	• Add a minimum amount of delay between ePWM channels • Define non-supported output combinations and drive output high or low if combination occurs
Event Trigger (ET)	• Enable the ePWM events that trigger an interrupt. • Enable ePWM events that trigger an ADC start-of-conversion event. • Specify the rate at which events cause triggers (every occurrence or every 2nd or up to 15th occurrence) • Poll, set, or clear event flags
Digital Compare (DC)	• Enables comparator (COMP) module outputs and trip zone signals which are configured using the Input X-BAR to create events and filtered events • Specify event-filtering options to capture TBCTR counter, generate blanking window, or insert delay in PWM output or time-base counter based on captured value.

7.5.6.4 Time-Base (TB) Submodule

Each ePWM module has their own time-base submodule that determines all of the event timing for the ePWM module. Built-in synchronization logic allows the time-base of multiple ePWM modules to work together as a single system.

Figure 7-157 illustrates the time-base submodule within the ePWM.

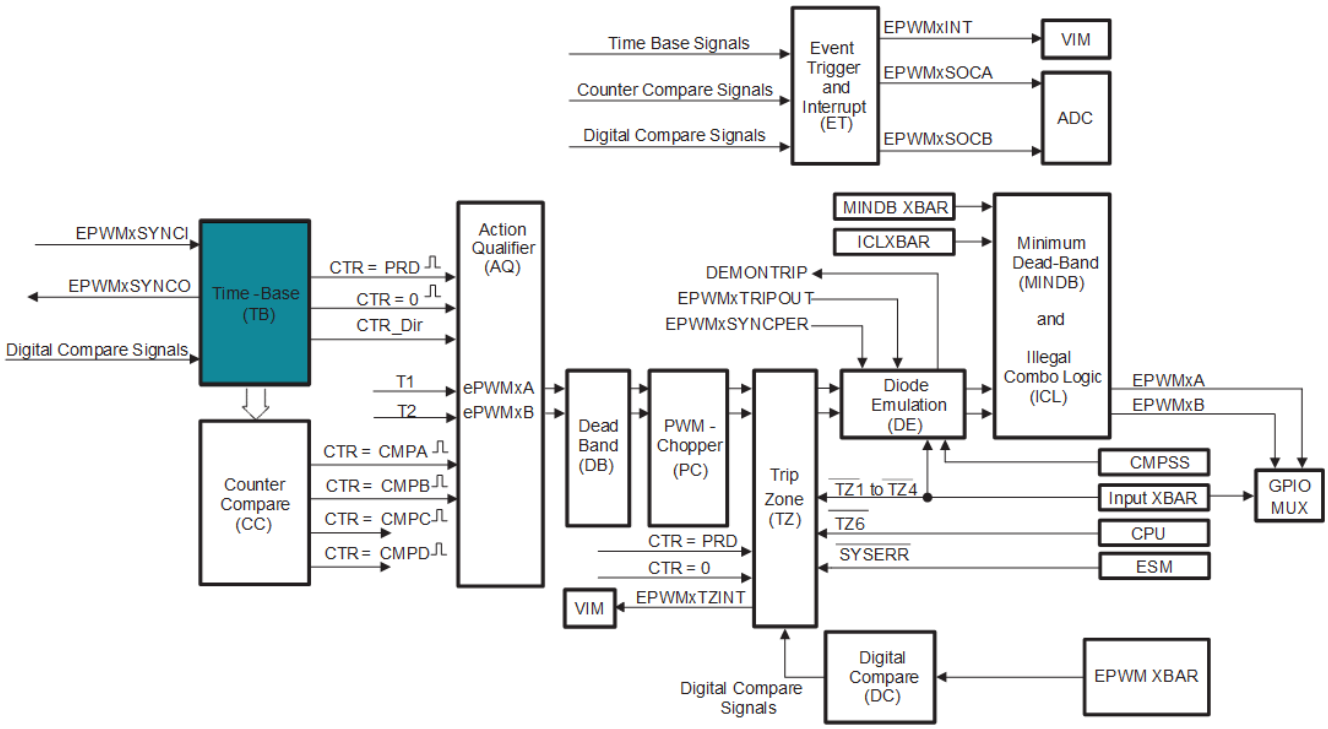


Figure 7-157. Time-Base Submodule

7.5.6.4.1 Purpose of the Time-Base Submodule

The time-base submodule can be configured for the following:

- Specify the ePWM time-base counter (TBCTR) frequency or period to control how often events occur.
- Manage time-base synchronization with other ePWM modules.
- Maintain a phase relationship with other ePWM modules.
- Set the time-base counter to count-up, count-down, or count-up-and-down mode.
- Generate the following events:
  - CTR = PRD: Time-base counter equal to the specified period (TBCTR = TBPRD).
  - CTR = Zero: Time-base counter equal to zero (TBCTR = 0x00).

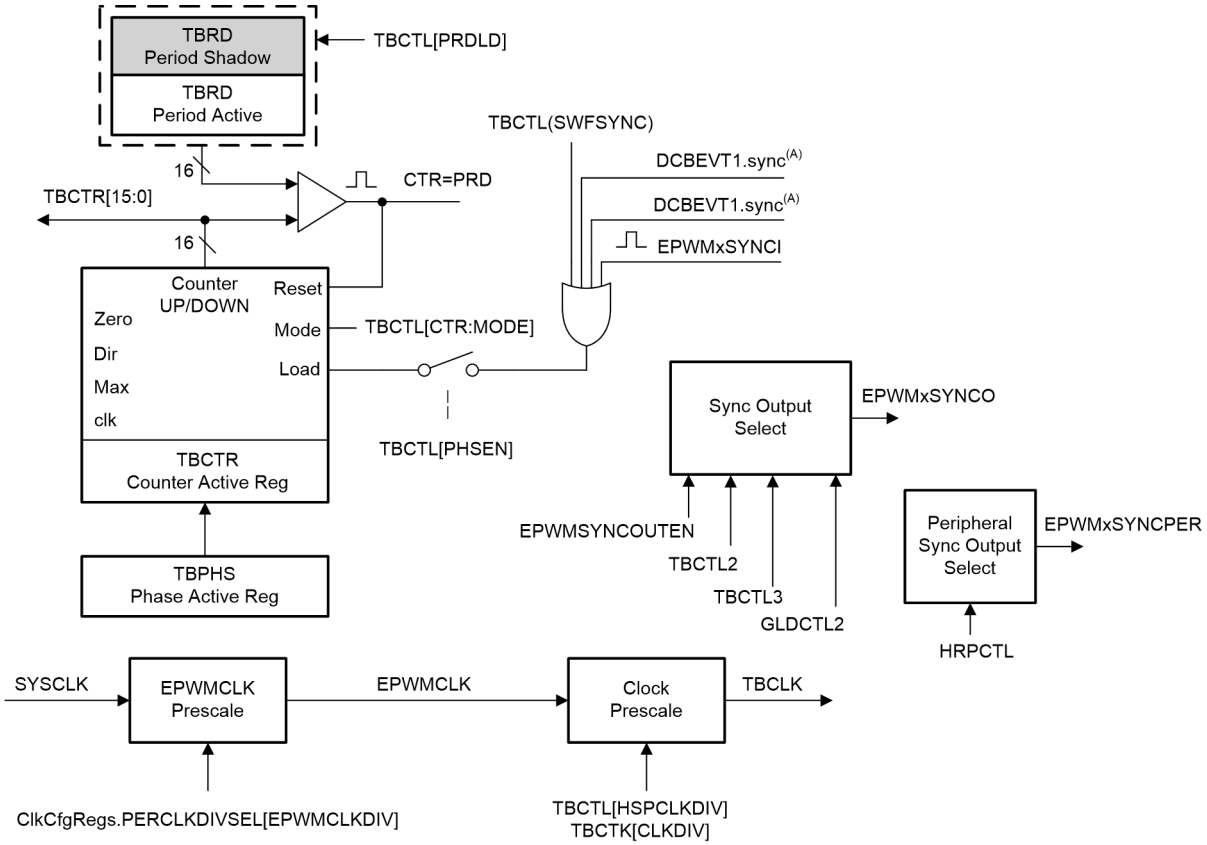
- Configure the rate of the time-base clock; a prescaled version of the ePWM clock (EPWMCLK). This allows the time-base counter to increment/decrement at a slower rate.

Note

If required by the application code to update the TBCTR value through software while the TBCTR is counting, note that the time-base module needs at least 1 TBCLK cycle for the time-base related events to be realized. Hence, the TBCTR can be written with TBCTR = PRD-1 instead of TBCTR = PRD (in case the counter is counting up) and can be written as TBCTR = 1 instead of TBCTR = 0 (in case the counter is counting down) for the events to be realized.

7.5.6.4.2 Controlling and Monitoring the Time-Base Submodule

The block diagram in Figure 7-158 shows the critical signals and registers of the time-base submodule. Table 7-120 provides descriptions of the key signals associated with the time-base submodule.



A. These signals are generated by the digital compare (DC) submodule.

Figure 7-158. Time-Base Submodule Signals and Registers

Table 7-120. Key Time-Base Signals

Signal	Description
EPWMxSYNCI	Time-base synchronization input. Input pulse used to synchronize the time-base counter with the counter of other ePWM modules. For more information on all of the signals available for synchronization, see EPWMSYNCINSEL. For information on the synchronization order of a particular device, see Time-Base Counter Synchronization
EPWMxSYNCO	Time-base synchronization output. This output pulse is used to synchronize the counter of other ePWM modules. Using EPWMSYNCOUTEN, TBCTL2, TBCTL3 and GLDCTL2, the source of the output pulse is selected.
EPWMxSYNCPER	Time-base peripheral synchronization output. This output signal is used to synchronize the CMPSS to the EPWM. The output signal can be configured using the HRPCTL register. Note that this signal has no relation with the HRPWM.
CTR = PRD	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 TBCTR = TBPRD.
CTR = Zero	Time-base counter equal to zero This signal is generated whenever the counter value is zero. That is when TBCTR equals 0x00.
CTR = CMPB	Time-base counter equal to active counter-compare B register (TBCTR = CMPB). This event is generated by the counter-compare submodule and used by the synchronization out logic
CTR_dir	Time-base counter direction. Indicates the current direction of the ePWM's time-base counter. The signal is high when the counter is increasing and the signal is low when the counter is decreasing.
CTR_max	Time-base counter equal max value. (TBCTR = 0xFFFF) Generated event when the TBCTR value reaches the maximum value. This signal is only used only as a status bit
TBCLK	Time-base clock. This is a prescaled version of the ePWM clock (EPWMCLK) and is used by all submodules within the ePWM. This clock determines the rate at which time-base counter increments or decrements.

7.5.6.4.3 Calculating PWM Period and Frequency

The frequency of PWM events is controlled by the time-base period (TBPRD) register and the mode of the time-base counter. Figure 7-159 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 (TBPRD = 4). The time increment for each step is defined by the time-base clock (TBCLK) which is a prescaled version of the ePWM clock (EPWMCLK).

The time-base counter has three modes of operation selected by the time-base control register (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 the counter reaches zero. At this point, the counter repeats the pattern and begins to increment.
- **Up-Count Mode:** In up-count mode, the time-base counter starts from zero and increments until the counter 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 the counter reaches zero. When the counter reaches zero, the time-base counter is reset to the period value and begins to decrement once again.

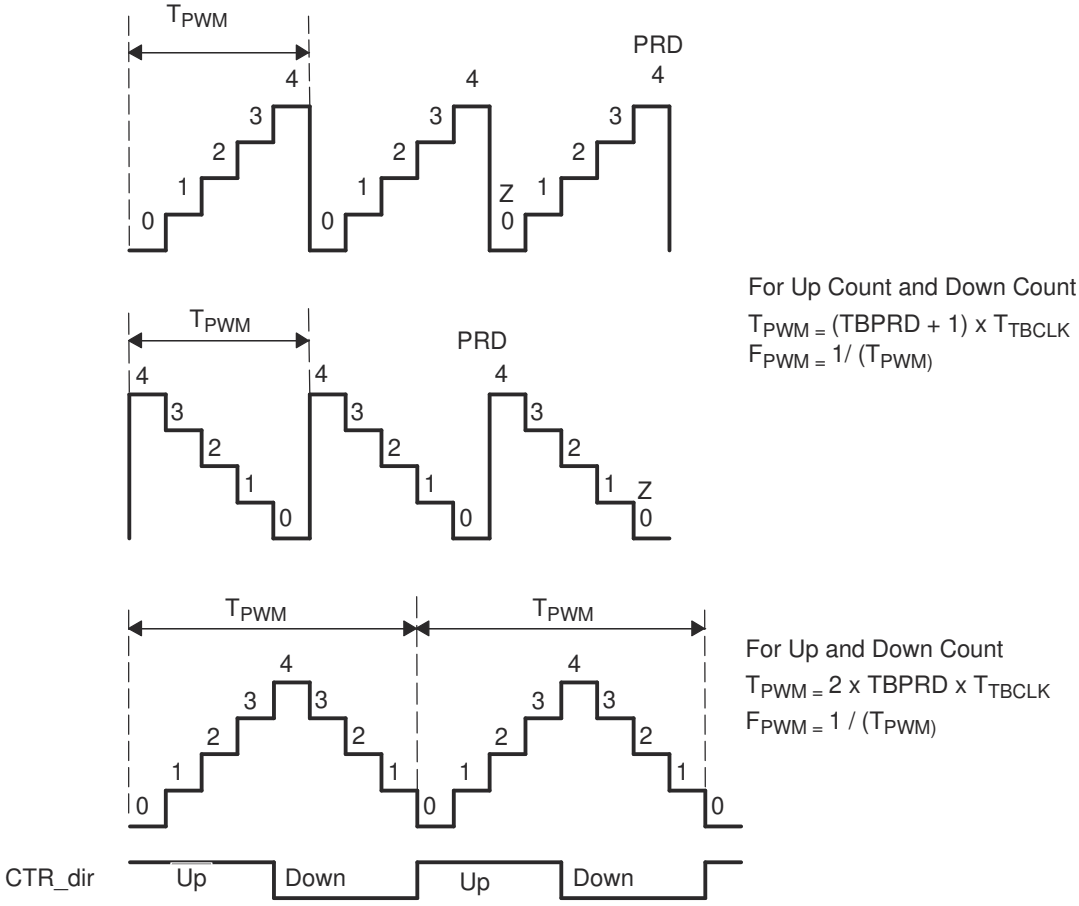


Figure 7-159. Time-Base Frequency and Period

7.5.6.4.3.1 Time-Base Period Shadow Register

The time-base period register (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 provide a temporary holding location for the active register and have no direct effect on any control hardware. At a strategic point in time, the shadow register 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 TBCTL[PRDLD] bit. This bit enables and disables the TBPRD shadow register as follows:

- **Time-Base Period Shadow Mode:** The TBPRD shadow register is enabled when TBCTL[PRDLD] = 0. Reads from and writes to the TBPRD memory address go to the shadow register. The shadow register contents are transferred to the active register (TBPRD (Active) ← TBPRD (shadow)) when the time-base counter equals zero (TBCTR = 0x00) and/or a sync event as determined by the TBCTL2[PRDLDSYNC] bit. The PRDLDSYNC bit is valid only if TBCTL[PRDLD] = 0. By default the TBPRD shadow register is enabled. The sources for the SYNC input is explained in Section 7.5.6.4.3.3.

  The global load control mechanism can also be used with the time-base period register by configuring the appropriate bits in the global load configuration register (GLDCFG). When global load mode is selected the transfer of contents from shadow register to active register, for all registers that have this mode enabled, occurs at the same event as defined by the configuration bits in Global Shadow to Active Load Control Register (GLDCTL). Global load control mechanism is explained in Section 7.5.6.4.8
- **Time-Base Period Immediate Load Mode:** If immediate load mode is selected (TBCTL[PRDLD] = 1), then a read from or a write to the TBPRD memory address goes directly to the active register.

7.5.6.4.3.2 Time-Base Clock Synchronization

The EPWM_CLKSYNC bit in the CONTROLSS_CTRL register allows all users to globally synchronize all enabled ePWM modules to the time-base clock (TBCLK). When set, all enabled ePWM module clocks are started with the first rising edge of TBCLK aligned. For perfectly synchronized TBCLKs, the prescalers for each ePWM module must be set identically.

The proper procedure for enabling ePWM clocks is as follows:

1. Set EPWM_CLKSYNC bit for correspont ePWM instance = 0
2. Configure ePWM modules
3. Set EPWM_CLKSYNC bit for corresponding ePWM instance = 1

7.5.6.4.3.3 Time-Base Counter Synchronization

The ePWM synchronization scheme allows for increased flexibility of synchronization of the ePWM modules. Each ePWM module has a synchronization input (SYNCI), a synchronization output (SYNCO) and a peripheral synchronization output (SYNCPER). In Figure 7-160, EXTSYNCIN1 is sourced from INPUTXBAR5 and EXTSYNCIN2 is sourced from INPUTXBAR6, which can be configured to select any GPIO as the synchronization input. Refer to for a list of all sync inputs including INPUTXBAR5 and INPUTXBAR6. Figure 7-161 shows the sources that can be used for EXTSYNCOUT.

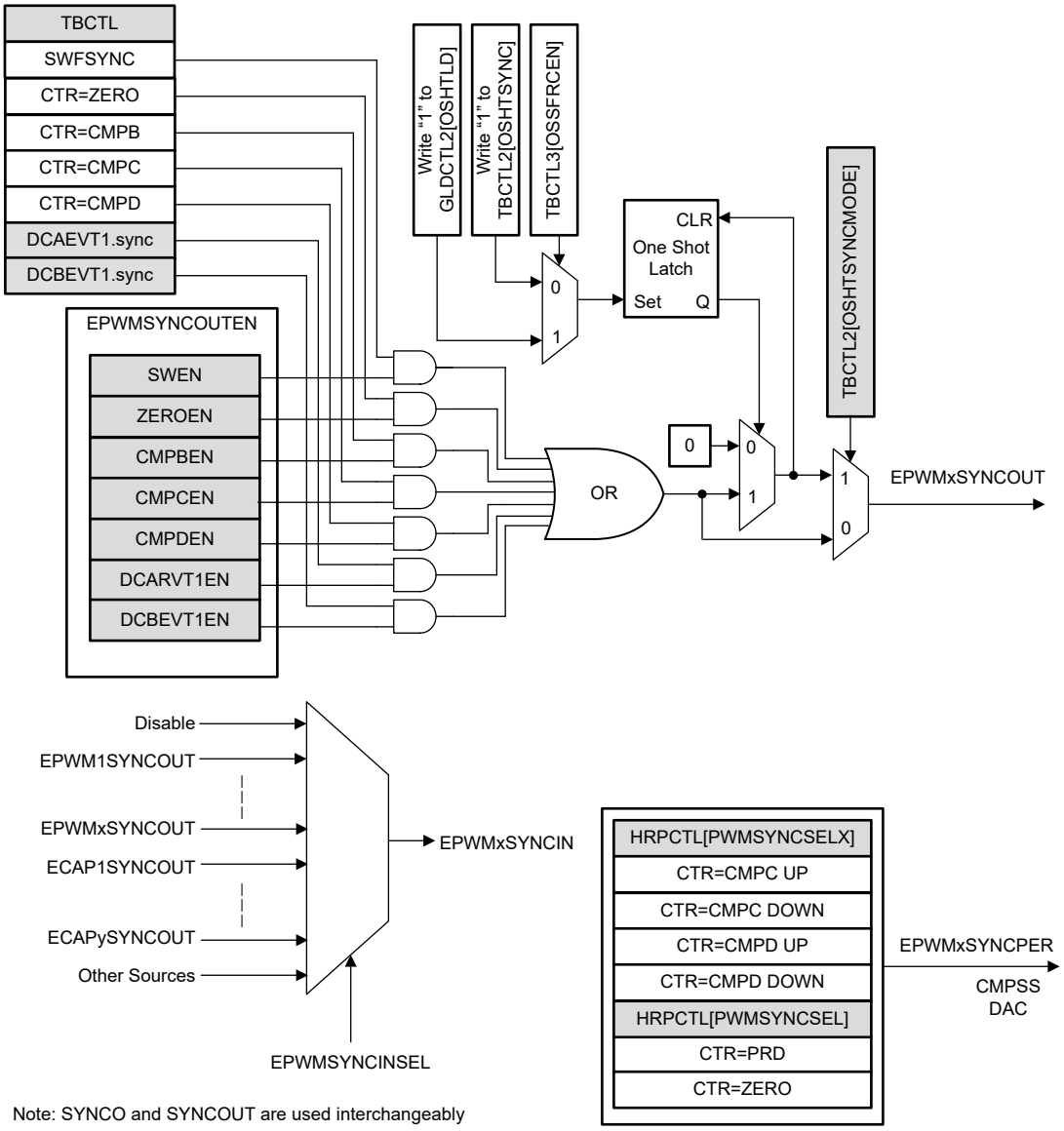


Figure 7-160. Time-Base Counter Synchronization Scheme

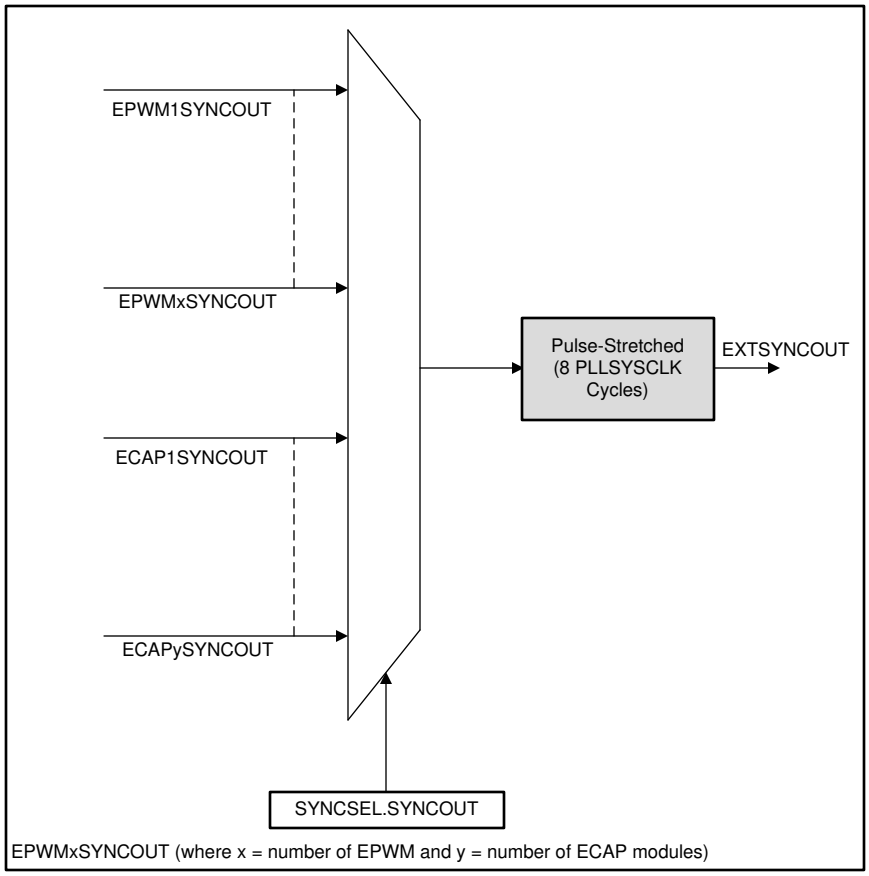


Figure 7-161. ePWM External SYNC Output

Note

See the data sheet for the number of ePWM and eCAP modules available on your specific device.

Each ePWM module can be configured to use or ignore the synchronization input. If the TBCTL[PHSEN] bit is set, then the time-base counter (TBCTR) of the ePWM module is automatically loaded with the phase register (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 (TBPHS → TBCTR). This operation occurs on the next valid time-base clock (TBCLK) edge.
- **Software Forced Synchronization Pulse:** Writing a 1 to the TBCTL[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.
- **Digital Compare Event Synchronization Pulse:** DCAEVT1 and DCBEVT1 digital compare events can be configured to generate synchronization pulses which have the same affect as EPWMxSYNCI.

Note

If the EPWMxSYNCI signal is held high, the sync does not continuously occur. The EPWMxSYNCI is rising edge activated. Don't use multiple edges in a PWM cycle if sync functionality is used.

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 TBCTL[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 PHSDIR bit is ignored in count-up or count-down modes. See Figure 7-162 through Figure 7-165 for examples.

Clearing the TBCTL[PHSEN] bit configures the ePWM to ignore the synchronization input pulse.

7.5.6.4.3.4 ePWM SYNC Selection

Table 7-121 specifies the sources for the ePWM SYNC input and output

Table 7-121. ePWM SYNC Selection

EPWMSYNCINSEL.SEL	SYNC Source
0x0	Reserved
0x1	EPWM0 SYNCOUT
0x2	EPWM1 SYNCOUT
0x3	EPWM2 SYNCOUT
0x4	EPWM3 SYNCOUT
0x5	EPWM4 SYNCOUT
0x6	EPWM5 SYNCOUT
0x7	EPWM6 SYNCOUT
0x8	EPWM7 SYNCOUT
0x9	EPWM8 SYNCOUT
0xA	EPWM9 SYNCOUT
0xB	EPWM10 SYNCOUT
0xC	EPWM11 SYNCOUT
0xD	EPWM12 SYNCOUT
0xE	EPWM13 SYNCOUT
0xF	EPWM14 SYNCOUT
0x10	EPWM15 SYNCOUT
0x11	EPWM16 SYNCOUT
0x12	EPWM17 SYNCOUT
0x13	EPWM18 SYNCOUT
0x14	EPWM19 SYNCOUT
0x15	EPWM20 SYNCOUT
0x16	EPWM21 SYNCOUT
0x17	EPWM22 SYNCOUT
0x18	EPWM23 SYNCOUT
0x19-0x3F	Reserved
0x40	ECAP0 SYNCOUT
0x41	ECAP1 SYNCOUT
0x42	ECAP2 SYNCOUT
0x43	ECAP3 SYNCOUT
0x44	ECAP4 SYNCOUT
0x45	ECAP5 SYNCOUT
0x46	ECAP6 SYNCOUT
0x47	ECAP7 SYNCOUT
0x48	ECAP8 SYNCOUT
0x49	ECAP9 SYNCOUT
0x4A-0x4F	Reserved
0x50	INPUTXBAR OUT.4
0x51	INPUTXBAR OUT.20
0x52-0x57	Reserved
0x58	TIMESYNCXBAR SYNCPWMOUT0
0x59	TIMESYNCXBAR SYNCPWMOUT1

Table 7-121. ePWM SYNC Selection (continued)

EPWMSYNCINSEL.SEL	SYNC Source
0x5A-0x5F	Reserved
0x60	FSI RX0 RXTRIG0
0x61	FSI RX0 RXTRIG1
0x62	FSI RX0 RXTRIG2
0x63	FSI RX0 RXTRIG3
0x64	FSI RX1 RXTRIG0
0x65	FSI RX1 RXTRIG1
0x66	FSI RX1 RXTRIG2
0x67	FSI RX1 RXTRIG3
0x68	FSI RX2 RXTRIG0
0x69	FSI RX2 RXTRIG1
0x6A	FSI RX2 RXTRIG2
0x6B	FSI RX2 RXTRIG3
0x6C	FSI RX3 RXTRIG0
0x6D	FSI RX3 RXTRIG1
0x6E	FSI RX3 RXTRIG2
0x6F	FSI RX3 RXTRIG3
0x70-0x7F	Reserved

7.5.6.4.4 Phase Locking the Time-Base Clocks of Multiple ePWM Modules

The CONTROLSS_CTRL.EPWM_CLKSYNC register has bits corresponding to each instance of ePWM. When EPWM_CLKSYNC = 0, the time-base clock of all corredsponding ePWM modules are stopped (default). When EPWM_CLKSYNC = 1, all corresponding ePWMs time-base clocks are started with the rising edge of TBCLK aligned. For perfectly synchronized TBCLKs, the prescaler bits in the TBCTL register of each ePWM module must be set identically.

The EPWM_CLKSYNC bit can be used to globally synchronize the time-base clocks of all enabled ePWM modules on a device. These bits are part of the CONTROLSS_CTRL register. When EPWM_CLKSYNC = 0, the time-base clock of all corresponding ePWM modules are stopped (default). When EPWM_CLKSYNC = 1, all corresponding ePWM modules' time-base clocks are started with the rising edge of TBCLK aligned. For perfectly synchronized TBCLKs, the prescaler bits in the TBCTL register of each ePWM module must be set identically. The proper procedure for enabling the ePWM clocks is:

1. Set EPWM_CLKSYNC = 0. This stops the time-base clock within any enabled ePWM module.
2. Configure the prescaler values and desired ePWM modes.
3. Set EPWM_CLKSYNC = 1.

7.5.6.4.5 Simultaneous Writes to TBPRD and CMPx Registers Between ePWM Modules

For variable frequency applications, there is a need for simultaneous writes of TBPRD and CMPx registers between ePWM modules. This prevents situations where a CTR = 0 or CTR = PRD pulse forces a shadow to active load of these registers before all registers are updated between ePWM modules (resulting in some registers being loaded from new shadow values while others are loaded from old shadow values). To support this, an ePWM register linking scheme for TBPRD:TBPRDHR, CMPA:CMPAHR, CMPB:CMPBHR, CMPC, and CMPD registers between PWM modules has been added.

Refer to the register description for EPWMXLINK to see the linked register bit-field values for corresponding ePWM. An example of using the EPWMXLINK is linking ePWM2 CMPA with CMPA of ePWM1 through ePWM2's EPWMXLINK[CMPALINK] register bit-field. In this case, a write to CMPA of ePWM1 also changes the CMPA value for ePWM2.

7.5.6.4.6 Time-Base Counter Modes and Timing Waveforms

The time-base counter operates in one of four modes:

- Up-count mode that is asymmetrical
- Down-count mode that is asymmetrical
- Up-down-count that is symmetrical
- Frozen where the time-base counter is held constant at the current value

To illustrate the operation of the first three modes, the following timing diagrams show when events are generated and how the time-base responds to an EPWMxSYNCI signal.

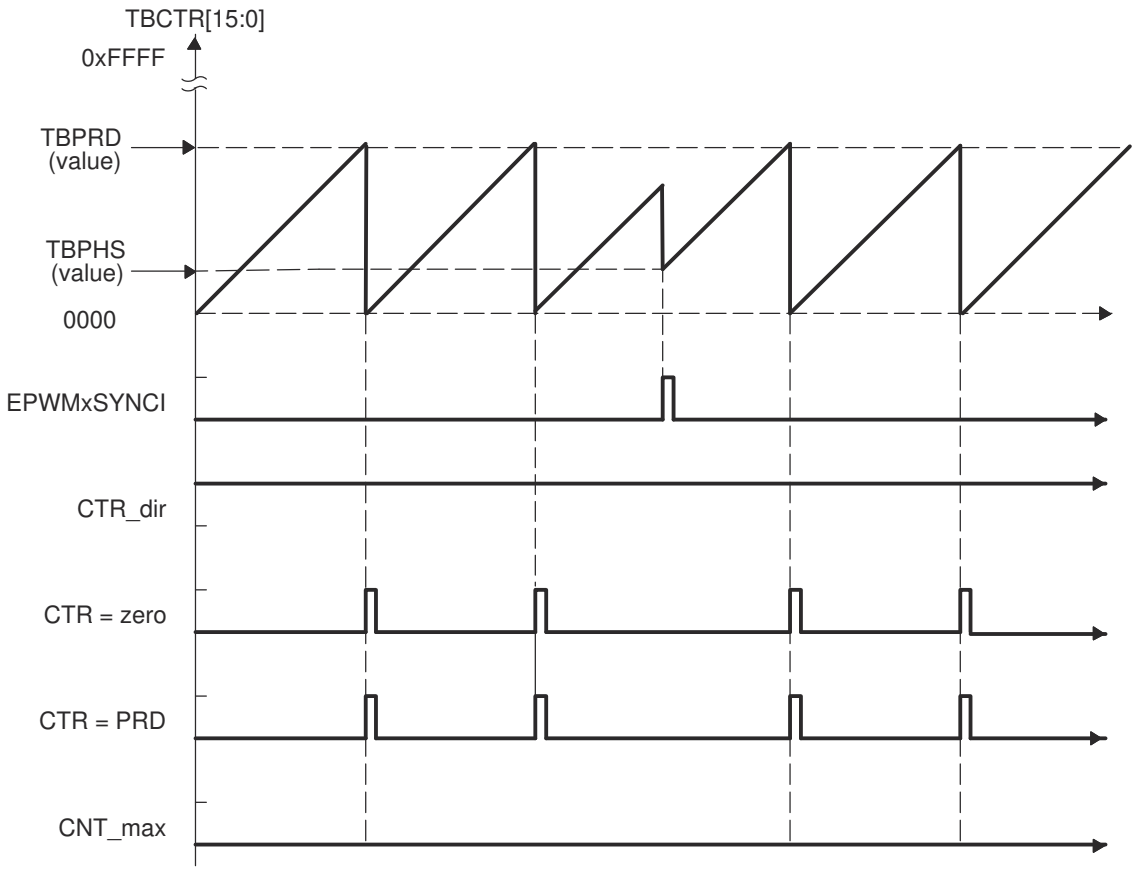


Figure 7-162. Time-Base Up-Count Mode Waveforms

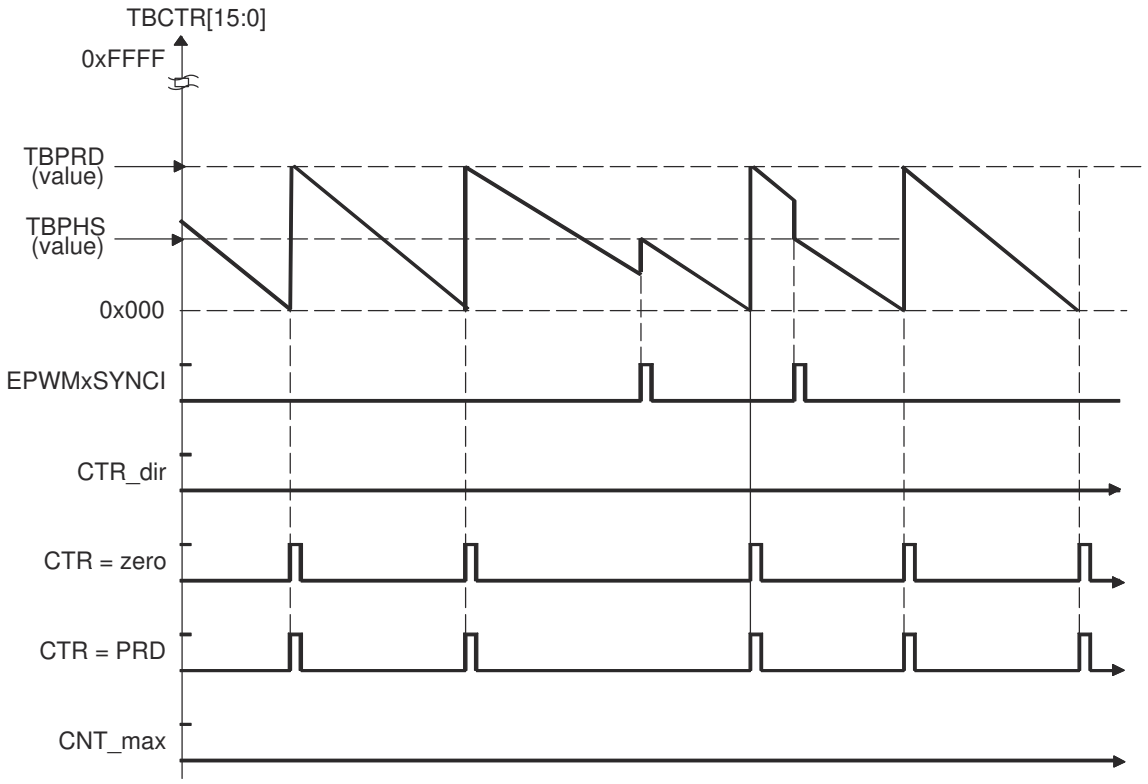


Figure 7-163. Time-Base Down-Count Mode Waveforms

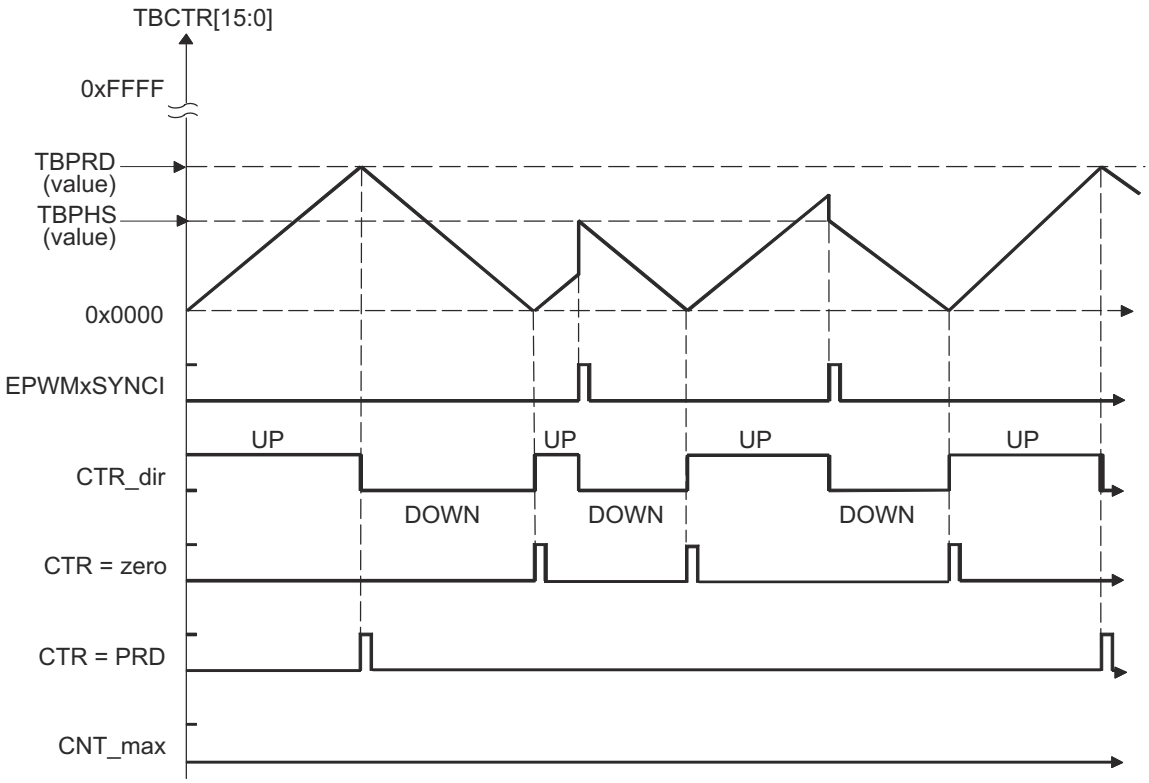


Figure 7-164. Time-Base Up-Down-Count Waveforms, TBCTL[PHSDIR = 0] Count Down On Synchronization Event

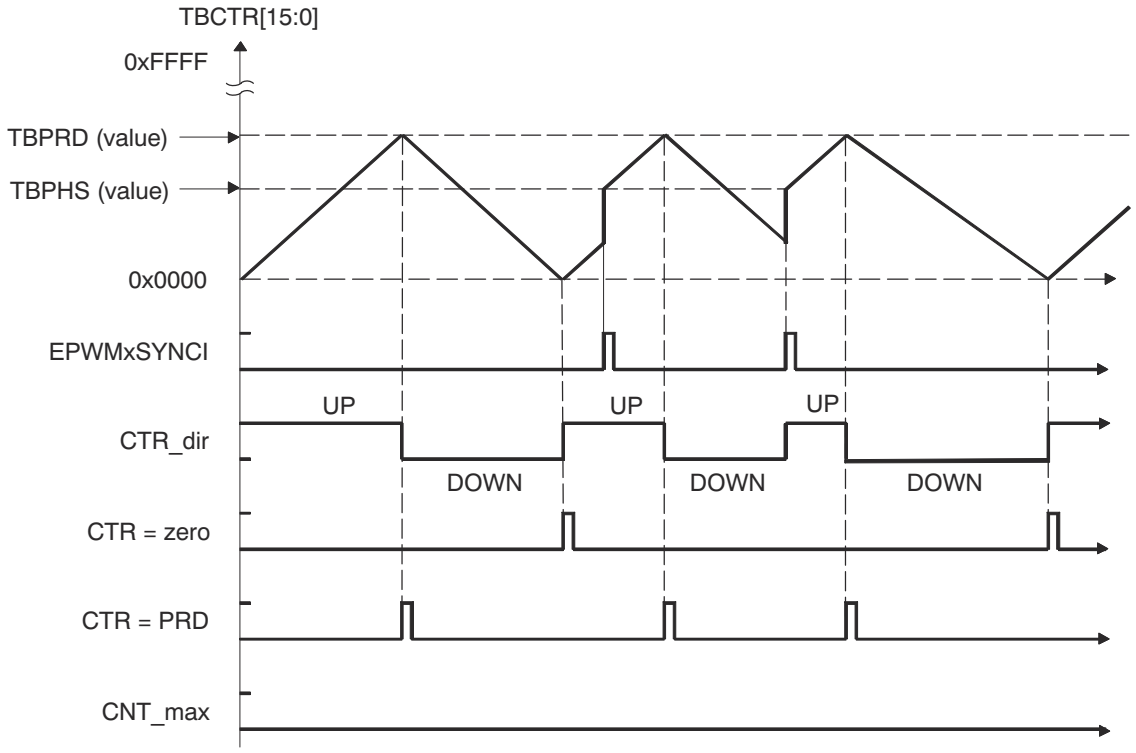


Figure 7-165. Time-Base Up-Down Count Waveforms, TBCTL[PHSDIR = 1] Count Up On Synchronization Event

7.5.6.4.7 Edge Detection Within a Programmable TBCTR Range

An edge detection within a programmable TBCTR range is added in type 5 ePWM.

This logic is primarily intended to detect an occurrence of a trip event in a configured time window. The window is configured by MIN and MAX values configured in the XMINMAX register sets. Refer to Event Detection for more details.

Using the CAPIN signal and the CAPGATE signal, the Capture Control Logic can generate a CAPEVT signal if an edge is **NOT** detected within a specified range of TBCTR values. More information about CAPIN and CAPENT is located in Input Signal Detection.

7.5.6.4.8 Global Load

Figure 7-166 shows the signals and registers associated with the global load feature.

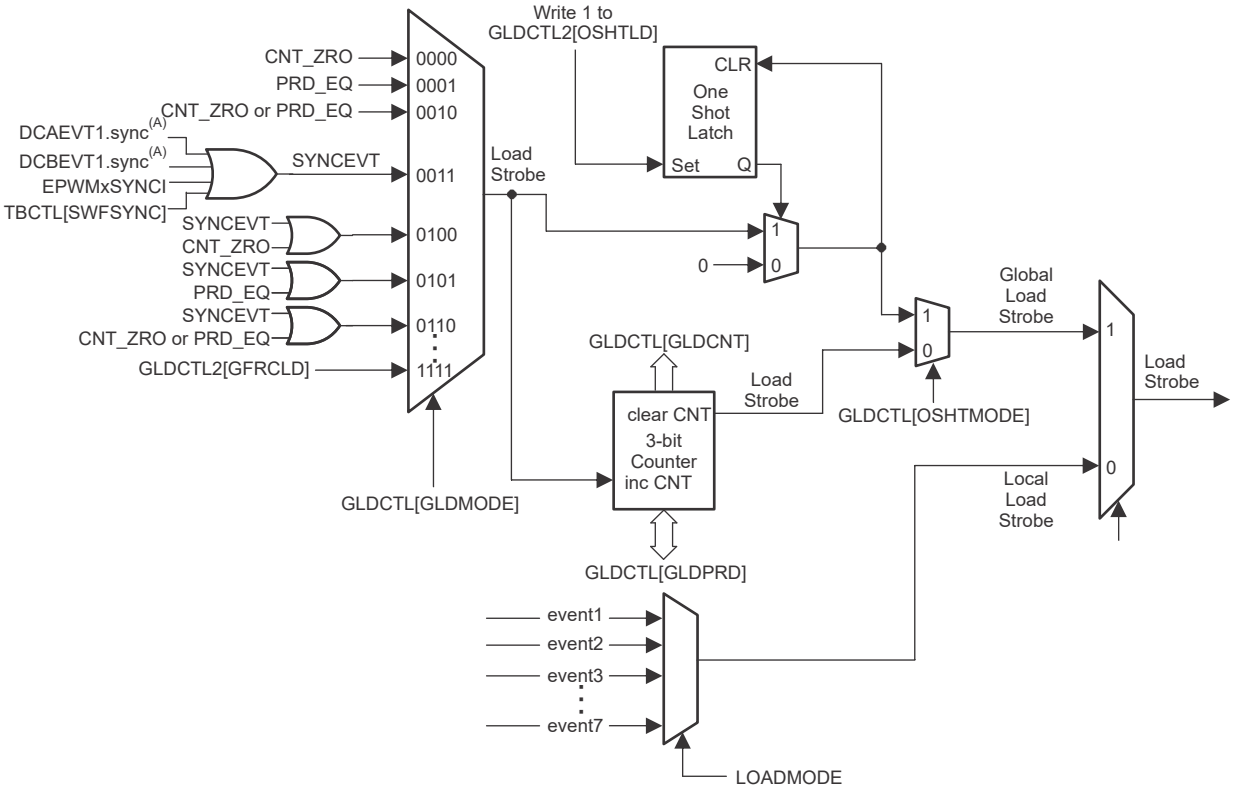


Figure 7-166. Global Load: Signals and Registers

Note

The SYNCEVT signal is only propagated through when PHSEN is SET.

When this feature is enabled, the transfer of contents from the shadow register to the active register, for all registers that have this mode enabled, occurs at the same event as defined by the configuration bits in Global Shadow to Active Load Control Register (GLDCTL[GLDMODE]). When GLDCTL[GLD] = 1, shadow to active load event selection bits for individual shadowed registers are ignored and global load mode takes effect for the corresponding registers enabled by GLDCFG[REGx]., where REGx is the register for which global load mode needs to be set.

When GLDCTL[GLD] = 1 and GLDCFG[REGx] = 0, global load mode does not affect the corresponding register (REGx). Shadow to active load event selection bits for individual shadowed registers decide how the transfer of contents from shadow register to active register takes place.

7.5.6.4.8.1 Global Load Pulse Pre-Scalar

This feature provides the capability to choose shadow to active transfers to happen once in 'N' occurrences of selected global load pulse (GLDCTL[GLDMODE]). This pre-scale functionality is not available for registers that cannot or are not configured to use the global load mechanism (that is, GLDCTL[GLD] = 0 or GLDCFG[REGx] = 0).

7.5.6.4.8.2 One-Shot Load Mode

This feature allows users to cause the shadow register to active register transfers to occur once. When GLDCTL2[OSHTLD] = 1 the shadow to active register transfer, for registers that are configured to use the global load mechanism, takes place on the event selected by GLDCTL[GLDMODE].

Software force loading of contents from shadow register to active register is possible by using GLDCTL2[GFRCLD]. The GLDCTL2 register can also be linked across multiple PWM modules by using EPWMXLINK[GLDCTL2LINK]. This, along with the one-shot load mode feature discussed above, provides a method to correctly update multiple PWM registers in one or more PWM modules at certain PWM events or, if desired, in the same clock cycle. This is very useful in variable frequency applications and/or multi-phase interleaved applications.

Note

One-shot load mode must not be used when high-resolution mode is enabled.

7.5.6.4.8.3 One-Shot Sync Mode

To enable the one-shot sync mode to generate a SYNCOUT pulse, configure the TBCTL2[OSHTSYNCMODE] bit and set the TBCTL2[OSHTSYNC] bit as shown in Figure 7-167.

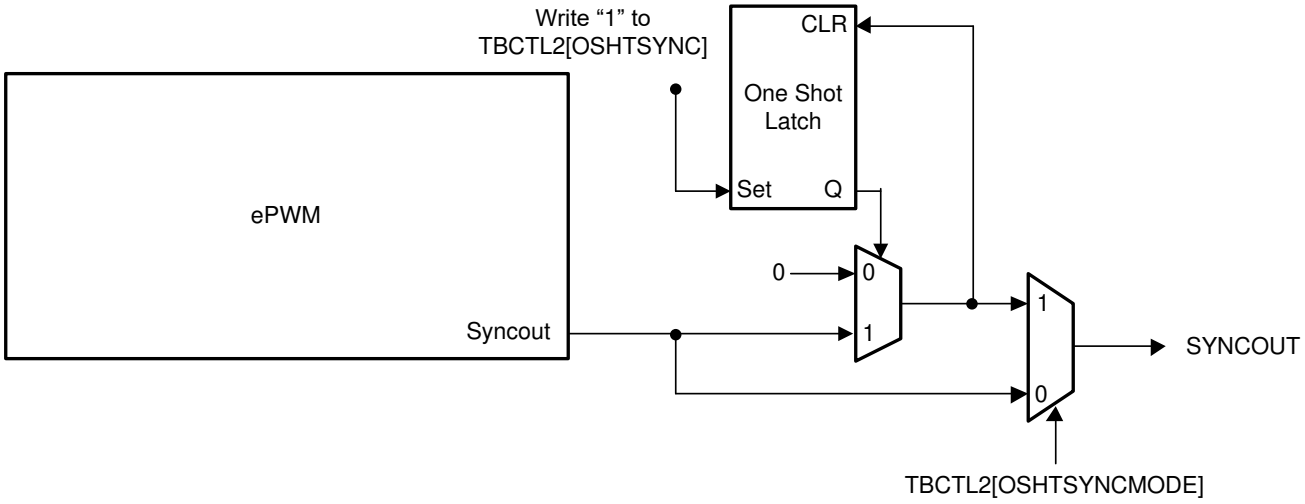


Figure 7-167. One-Shot Sync Mode

7.5.6.5 Counter-Compare (CC) Submodule

Figure 7-168 illustrates the counter-compare submodule within the ePWM.

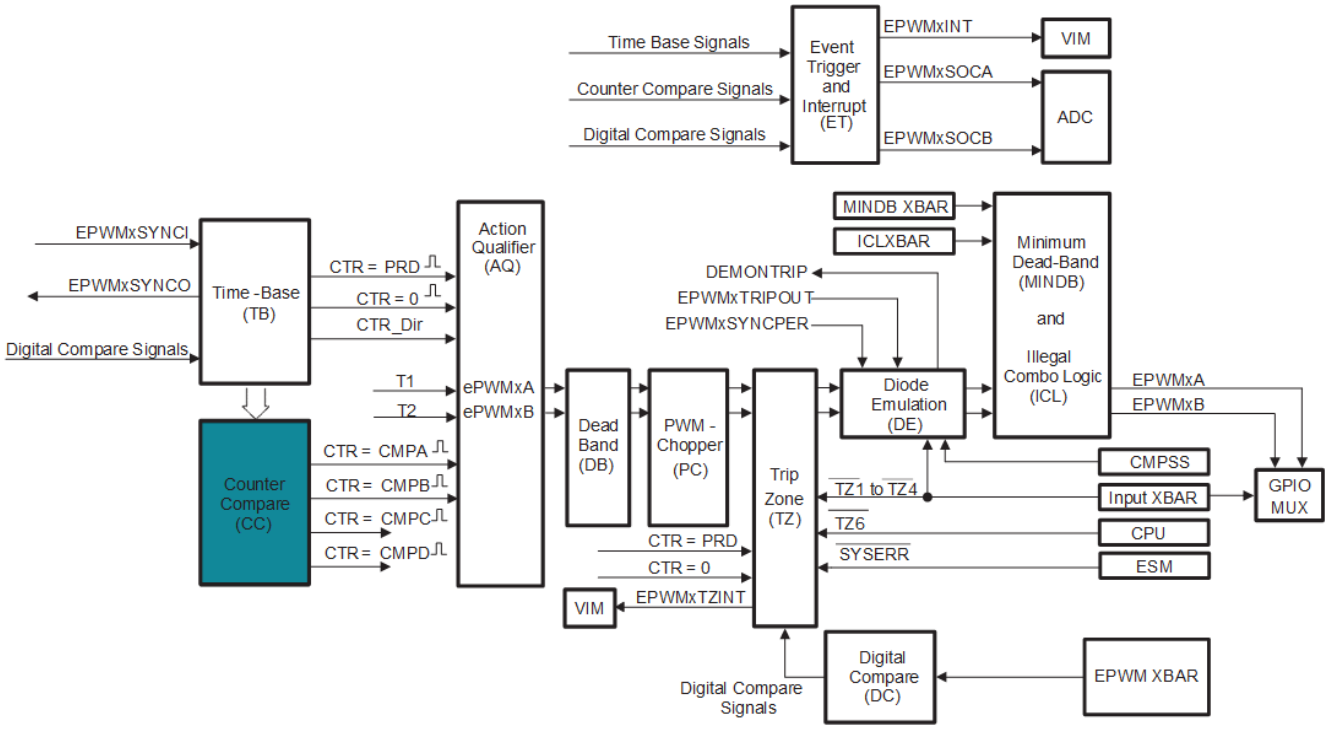


Figure 7-168. Counter-Compare Submodule

7.5.6.5.1 Purpose of the Counter-Compare Submodule

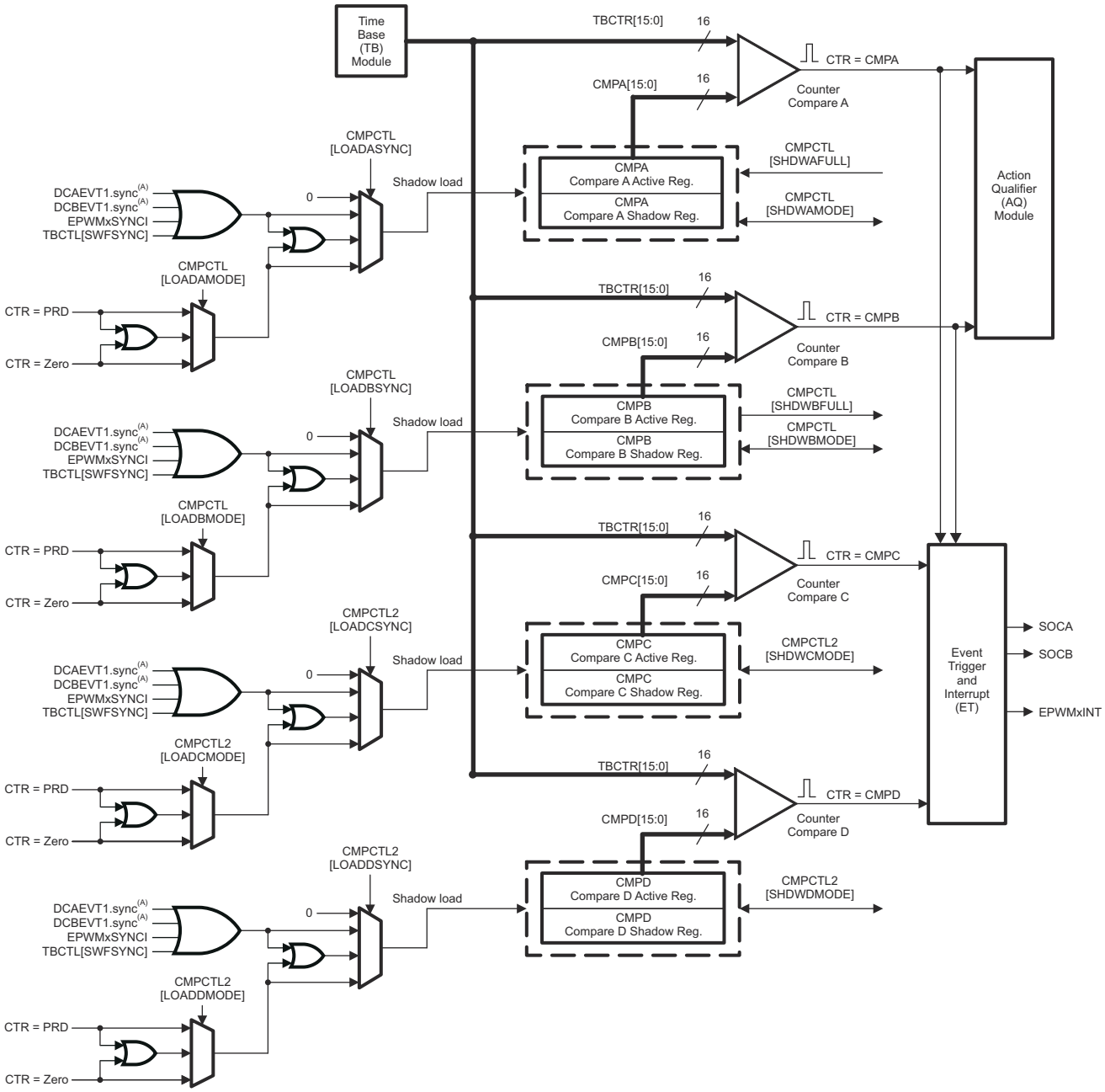
The counter-compare submodule takes as input the time-base counter value. This value is continuously compared to the counter-compare A (CMPA), counter-compare B (CMPB), counter-compare C (CMPC), and counter-compare D (CMPD) registers. When the time-base counter is equal to one of the compare registers, the counter-compare unit generates an appropriate event.

The counter-compare:

- Generates events based on programmable time stamps using the CMPA, CMPB, CMPC, and CMPD registers:
  - CTR = CMPA: Time-base counter equals counter-compare A register (TBCTR = CMPA)
  - CTR = CMPB: Time-base counter equals counter-compare B register (TBCTR = CMPB)
  - CTR = CMPC: Time-base counter equals counter-compare C register (TBCTR = CMPC)
  - CTR = CMPD: Time-base counter equals counter-compare D register (TBCTR = CMPD)
- Controls the PWM duty cycle, if the action-qualifier submodule is configured appropriately using counter-compare A (CMPA) and counter-compare B (CMPB)
- Shadows new compare values to prevent corruption or glitches during the active PWM cycle

7.5.6.5.2 Controlling and Monitoring the Counter-Compare Submodule

The counter-compare submodule operation is shown in Figure 7-169.



A. These events are generated by the ePWM digital compare (DC) submodule based on the levels of the TRIPIN inputs (for example, CMPSSx and $\overline{TZ}$ signals).

Figure 7-169. Detailed View of the Counter-Compare Submodule

7.5.6.5.3 Operational Highlights for the Counter-Compare Submodule

The counter-compare submodule is responsible for generating events that can be used in the action-qualifier and event-trigger submodules. There are four independent compare events:

1. CTR = CMPA: Time-base counter equal to counter-compare A register (TBCTR = CMPA).
2. CTR = CMPB: Time-base counter equal to counter-compare B register (TBCTR = CMPB).
3. CTR = CMPC: Time-base counter equal to counter-compare C register (TBCTR = CMPC). This event can be used to generate an event in the event trigger submodule only.
4. CTR = CMPD: Time-base counter equal to counter-compare D register (TBCTR = CMPD). This event can be used to generate an event in the event trigger submodule only

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 0x00-TBPRD; and once per cycle if the compare value is equal to 0x00 or equal to TBPRD. These events are applied to the action-qualifier submodule where the events are qualified by the counter direction and converted into actions if enabled. Refer to Section 7.5.6.6.1 for more details.

The counter-compare registers CMPA and CMPB 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 occur 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. The register that is written to or read from is determined by the CMPCTL[SHDWAMODE] and CMPCTL[SHDWBMODE] bits. These bits enable and disable the CMPC shadow register and CMPD shadow register, respectively. The behavior of the two load modes is:

Shadow Mode:

The shadow mode for the CMPA is enabled by clearing the CMPCTL[SHDWAMODE] bit and the shadow register for CMPB is enabled by clearing the CMPCTL[SHDWBMODE] bit. Shadow mode is enabled by default for both CMPA and CMPB.

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 as specified by the CMPCTL[LOADAMODE], CMPCTL[LOADBMODE], CMPCTL[LOADASYNC], and CMPCTL[LOADBSYNC] register bits:

- CTR = PRD: Time-base counter equal to the period (TBCTR = TBPRD).
- CTR = Zero: Time-base counter equal to zero (TBCTR = 0x00)
- Both CTR = PRD and CTR = Zero
- SYNC event caused by DCAEVT1 or DCBEVT1 or EPWMxSYNCI or TBCTL[SWFSYNC]
- Both SYNC event or a selection made by LOADAMODE/LOADBMODE

Only the active register contents are used by the counter-compare submodule to generate events to be sent to the action-qualifier.

Note

Refer to Section 7.5.6.6.5 for valid configurations of CMPA/CMPB and LOADAMODE/LOADBMODE.

Immediate Load Mode:

If the immediate load mode is selected (that is, CMPCTL[SHDWAMODE] = 1 or CMPCTL[SHDWBMODE] = 1), then a read from or a write to the register goes directly to the active register.

Additional Comparators

The counter-compare submodule on ePWMs type 2 and later are responsible for generating two additional independent compare events based on two compare registers, which is fed to Event Trigger submodule:

1. CTR = CMPC: Time-base counter equal to counter-compare C register (TBCTR = CMPC).
2. CTR = CMPD: Time-base counter equal to counter-compare D register (TBCTR = CMPD).

The counter-compare registers CMPC and CMPD each have an associated shadow register. By default this register is shadowed. The memory address of the active register and the shadow register is identical. The value in the active CMPC and CMPD register is compared to the time-base counter (TBCTR). When the values are equal, the counter compare module generates a “time-base counter equal to counter compare C or counter compare D ” event respectively. Shadowing of this register is enabled and disabled by the CMPCTL2[SHDWCMODE] and CMPCTL2[SHDWDMODE] bit. These bits enable and disable the CMPC shadow register and CMPD shadow register respectively. The behavior of the two load modes is described below:

Shadow Mode:

The shadow mode for the CMPC is enabled by clearing the CMPCTL2[SHDWCMODE] bit and the shadow register for CMPD is enabled by clearing the CMPCTL2[SHDWDMODE] bit. Shadow mode is enabled by default for both CMPC and CMPD.

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 as specified by the CMPCTL2[LOADCMODE], CMPCTL2[LOADDMODE], CMPCTL2[LOADCSYNC], and CMPCTL2[LOADDSYNC] register bits:

- CTR = PRD: Time-base counter equal to the period (TBCTR = TBPRD).
- CTR = Zero: Time-base counter equal to zero (TBCTR = 0x00)
- Both CTR = PRD and CTR = Zero
- SYNC event caused by DCAEVT1 or DCBEVT1 or EPWMxSYNCI or TBCTL[SWFSYNC]
- Both SYNC event or a selection made by LOADCMODE/LOADDMODE

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 the immediate load mode is selected (that is, CMPCTL2[SHDWCMODE] = 1 or CMPCTL2[SHDWDMODE] = 1), then a read from or a write to the register goes directly to the active register.

Global Load Support

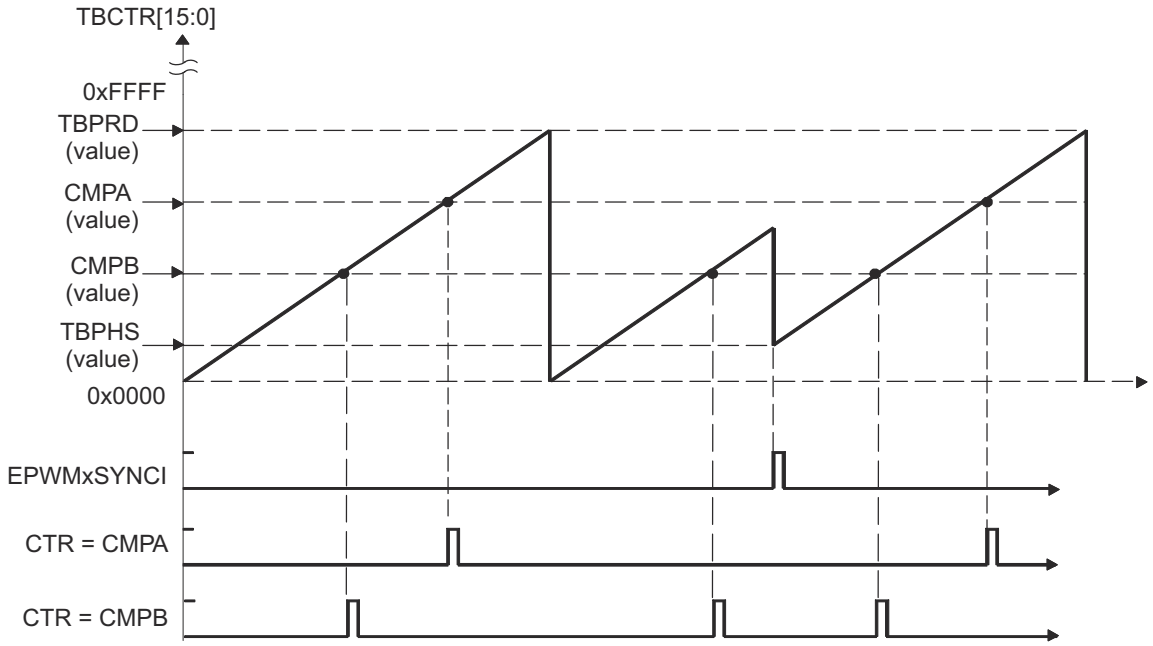
The global load control mechanism can also be used for all counter-compare registers by configuring the appropriate bits in the global load configuration register (GLDCFG). When the global load mode is selected the transfer of contents from shadow register to active register, for all registers that have this mode enabled, occurs at the same event as defined by the configuration bits in the Global Shadow to Active Load Control Register (GLDCTL). The global load control mechanism is explained in Section 7.5.6.4.8.

7.5.6.5.4 Count Mode Timing Waveforms

The counter-compare module can generate compare events in all three 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.

To best illustrate the operation of the first three modes, the timing diagrams in Figure 7-170 through Figure 7-172 show when events are generated and how the EPWMxSYNCI signal interacts.



An EPWMxSYNCI external synchronization event can cause a discontinuity in the TBCTR count sequence. This can lead to a compare event being skipped. This skipping is considered normal operation and must be taken into account.

Figure 7-170. Counter-Compare Event Waveforms in Up-Count Mode

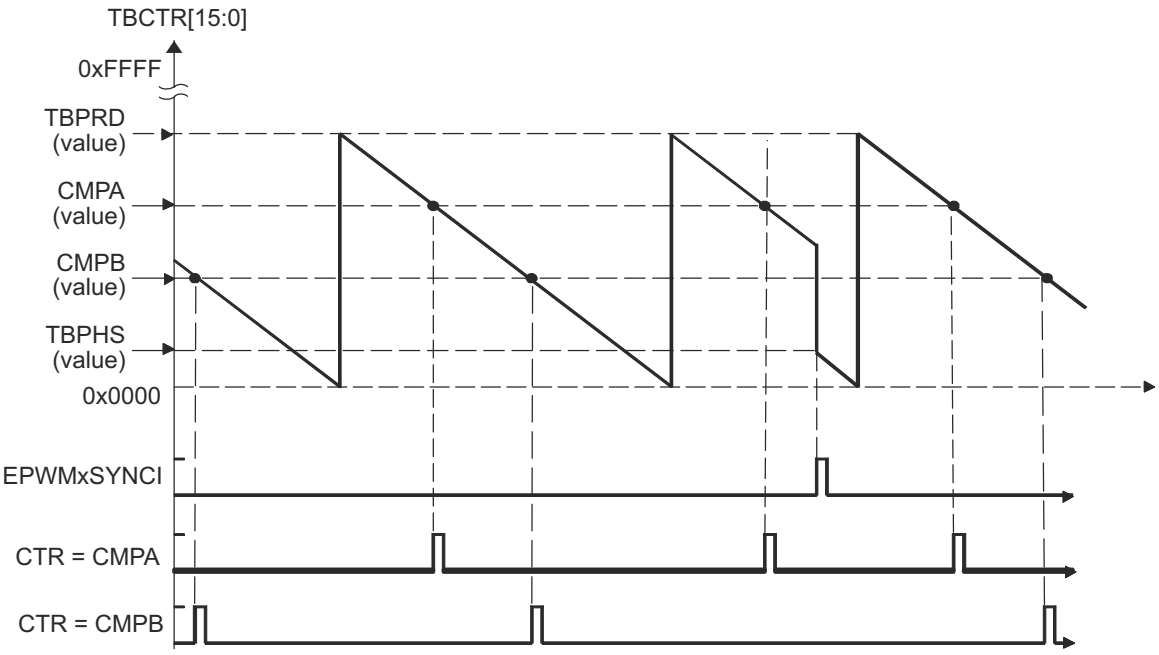


Figure 7-171. Counter-Compare Events in Down-Count Mode

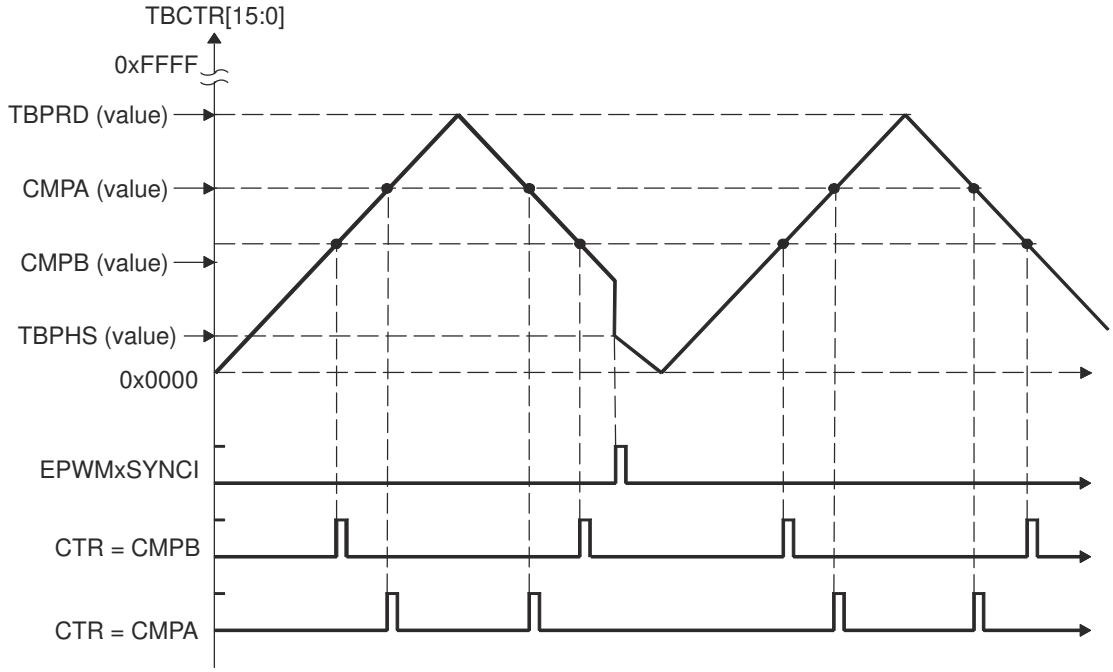


Figure 7-172. Counter-Compare Events In Up-Down-Count Mode, TBCTL[PHSDIR = 0] Count Down On Synchronization Event

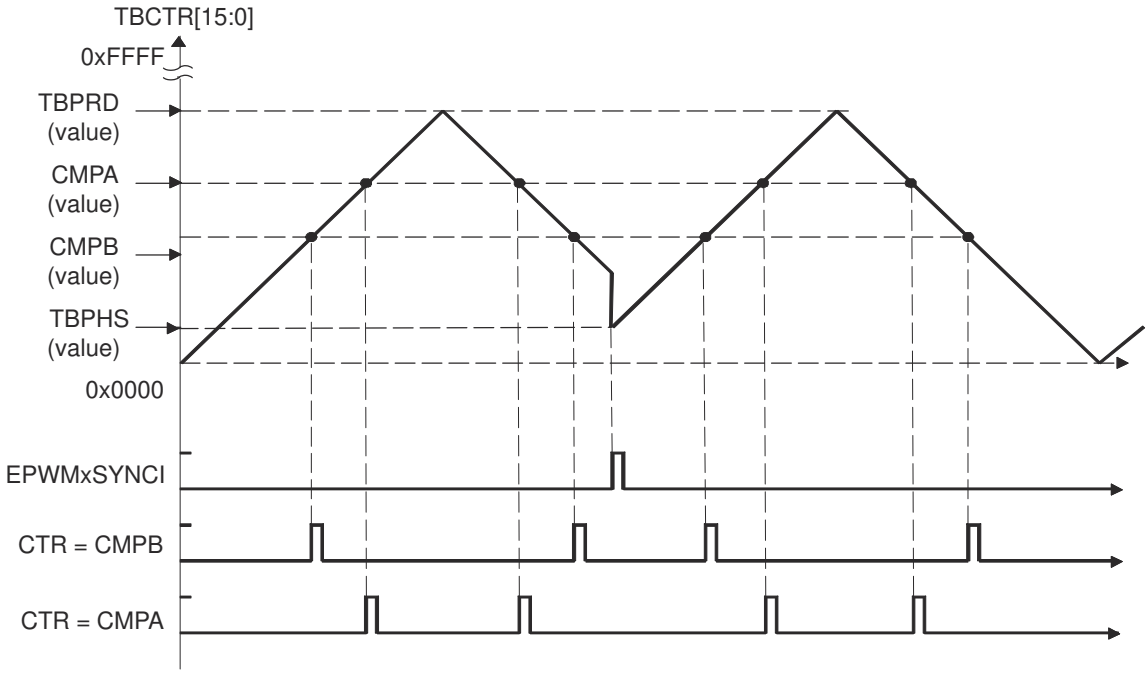


Figure 7-173. Counter-Compare Events In Up-Down-Count Mode, TBCTL[PHSDIR = 1] Count Up On Synchronization Event

7.5.6.6 Action-Qualifier (AQ) Submodule

The action-qualifier submodule has the most important role in waveform construction and PWM generation. The action-qualifier submodule decides which events are converted into various action types, thereby, producing the required switched waveforms at the EPWMxA and EPWMxB outputs.

Figure 7-174 illustrates the action-qualifer submodule within the ePWM

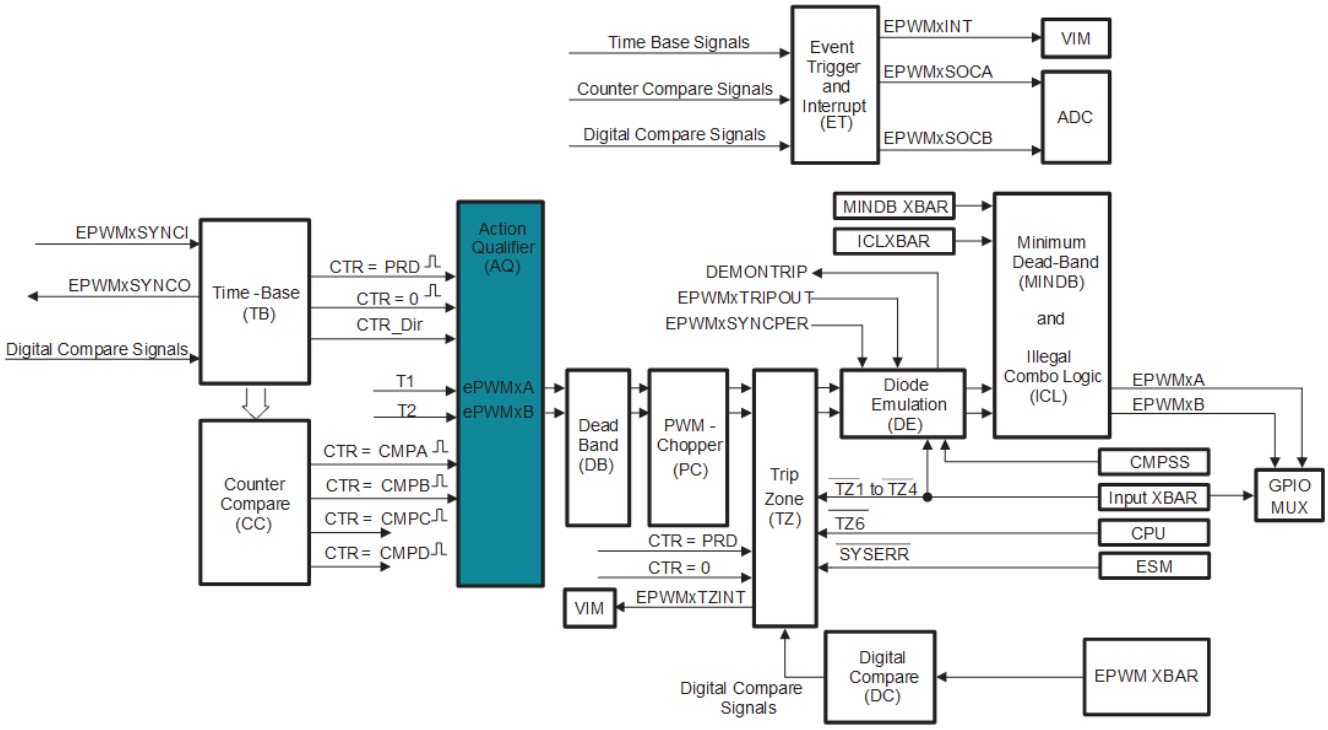


Figure 7-174. Action-Qualifier Submodule

7.5.6.6.1 Purpose of the Action-Qualifier Submodule

The action-qualifier submodule is responsible for the following:

- Qualifying and generating actions (set, clear, toggle) based on the following events:
  - CTR = PRD: Time-base counter equal to the period (TBCTR = TBPRD).
  - CTR = Zero: Time-base counter equal to zero (TBCTR = 0x00)
  - CTR = CMPA: Time-base counter equal to the counter-compare A register (TBCTR = CMPA)
  - CTR = CMPB: Time-base counter equal to the counter-compare B register (TBCTR = CMPB)
- T1, T2 events: Trigger events based on comparator, trip or syncin events
- Managing priority when these events occur concurrently
- Providing independent control of events when the time-base counter is increasing and when it is decreasing

7.5.6.6.2 Action-Qualifier Submodule Control and Status Register Definitions

The action-qualifier submodule operation is shown in Figure 7-175.

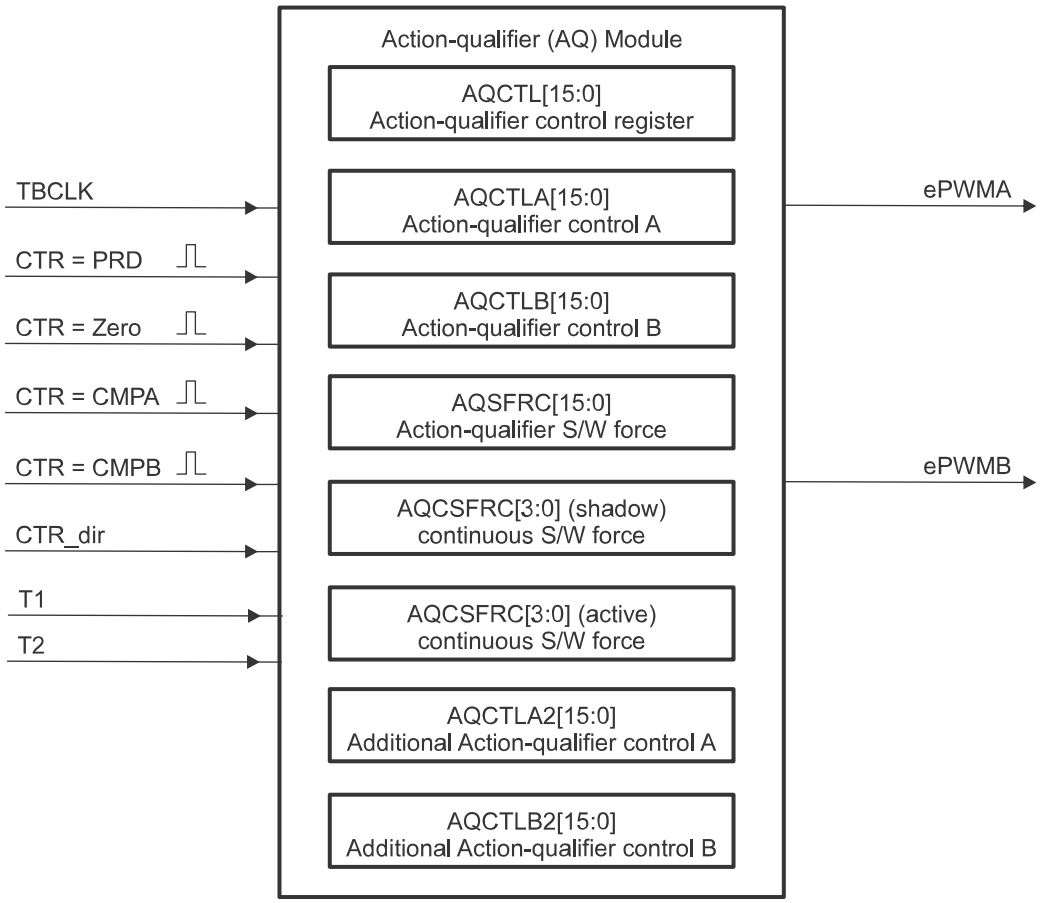


Figure 7-175. Action-Qualifier Submodule Inputs and Outputs

For convenience, the possible input events are summarized again in Table 7-122

Table 7-122. Action-Qualifier Submodule Possible Input Events

Signal	Description	Registers Compared
CTR = PRD	Time-base counter equal to the period value	TBCTR = TBPRD
CTR = Zero	Time-base counter equal to 0	TBCTR = 0x00
CTR = CMPA	Time-base counter equal to the counter-compare A	TBCTR = CMPA
CTR = CMPB	Time-base counter equal to the counter-compare B	TBCTR = CMPB
T1 event	Based on comparator, trip, or syncin events	None
T2 event	Based on comparator, trip, or syncin events	None
Software forced event	Asynchronous event initiated by software	

The software forced action is a useful asynchronous event. This control is handled by the AQSFRC and AQCSFRC registers.

Note

If the CSFA is not used in shadow mode, the RLDCSF bit must be configured to disable shadow mode.

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 and ADC start of conversion. See the description in Section 7.5.6.12 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. For example, both CTR = CMPA and CTR = CMPB can operate on output EPWMxA.

For clarity, the illustrations in this chapter use a set of symbolic actions. These symbols are summarized in Figure 7-176. 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 by way of the counter-compare A and B registers, respectively. To turn off or disable an action, use the "Do Nothing option"(the default at reset).

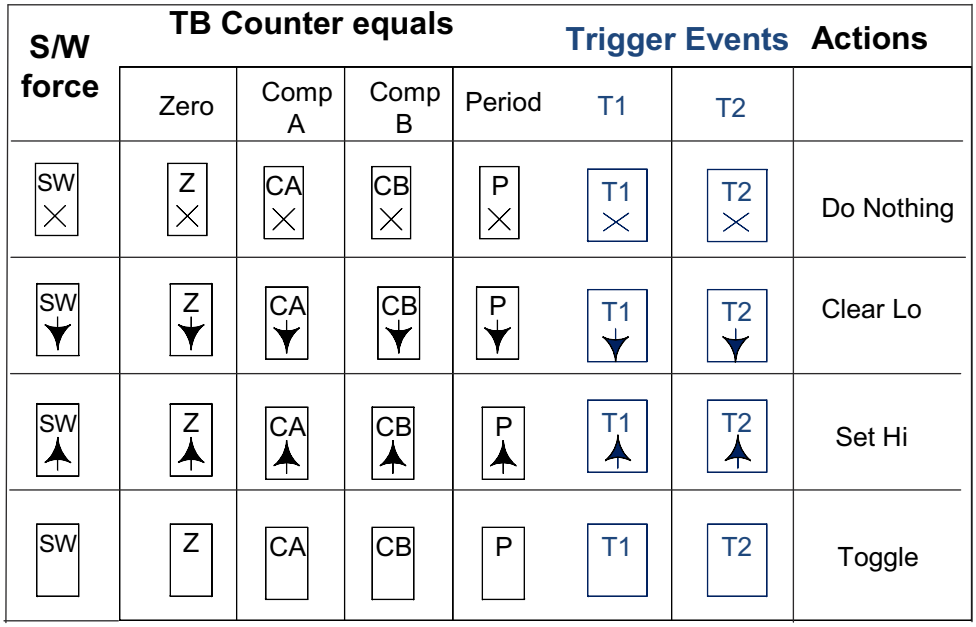


Figure 7-176. Possible Action-Qualifier Actions for EPWMxA and EPWMxB Outputs

The Action Qualifier Trigger Event Source Selection register (AQTSRCSEL) is used to select the source for T1 and T2 events. T1/T2 selection and configuration of a trip/digital-compare event in Action Qualifier submodule is independent of the configuration of that event in the Trip-Zone submodule. A particular trip event can or cannot be configured to cause trip action in the Trip Zone submodule, but the same event can be used by the Action Qualifier to generate T1/T2 for controlling PWM generation.

7.5.6.6.3 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 7-123. A priority level of 1 is the highest priority and level 10 is the lowest. The priority changes slightly depending on the direction of TBCTR.

Table 7-123. Action-Qualifier Event Priority for Up-Down-Count Mode

Priority Level	Event If TBCTR is Incrementing TBCTR = Zero up to TBCTR = TBPRD	Event If TBCTR is Decrementing TBCTR = TBPRD down to TBCTR = 1
1 (Highest)	Software forced event	Software forced event
2	T1 on up-count (T1U)	T1 on down-count (T1D)
3	T2 on up-count (T2U)	T2 on down-count (T2D)
4	Counter equals CMPB on up-count (CBU)	Counter equals CMPB on down-count (CBD)
5	Counter equals CMPA on up-count (CAU)	Counter equals CMPA on down-count (CAD)
6	Counter equals zero	Counter equals period (TBPRD)
7	T1 on down-count (T1D)	T1 on up-count (T1U)
8(Lowest)	T2 on down-count (T2D)	T2 on up-count (T2U)

Table 7-124 shows the action-qualifier priority for up-count mode. In this case, the counter direction is always defined as up; therefore, down-count events never are taken.

Table 7-124. Action-Qualifier Event Priority for Up-Count Mode

Priority Level	Event
1 (Highest)	Software forced event
2	Counter equal to period (TBPRD)
3	T1 on up-count (T1U)
4	T2 on up-count (T2U)
5	Counter equal to CMPB on up-count (CBU)
6	Counter equal to CMPA on up-count (CAU)
7 (Lowest)	Counter equal to Zero

Table 7-125 shows the action-qualifier priority for down-count mode. In this case, the counter direction is always defined as down; therefore, up-count events never are taken.

Table 7-125. Action-Qualifier Event Priority for Down-Count Mode

Priority Level	Event
1 (Highest)	Software forced event
2	Counter equal to Zero
3	T1 on down-count (T1D)
4	T2 on down-count (T2D)
5	Counter equal to CMPB on down-count (CBD)
6	Counter equal to CMPA on down-count (CAD)
7 (Lowest)	Counter equal to period (TBPRD)

It is possible to set the compare value greater than the period. In this case, the action takes place as shown in Table 7-126.

Table 7-126. Behavior if CMPA/CMPB is Greater than the Period

Counter Mode	Compare on Up-Count Event CAD/CBD	Compare on Down-Count Event CAD/CBD
Up-Count Mode	If CMPA/CMPB ≤ TBPRD period, then the event occurs on a compare match (TBCTR=CMPA or CMPB). If CMPA/CMPB > TBPRD, then the event does not occur.	Never occurs.
Down-Count Mode	Never occurs.	If CMPA/CMPB < TBPRD, the event occurs on a compare match (TBCTR=CMPA or CMPB). If CMPA/CMPB ≥ TBPRD, the event occurs on a period match (TBCTR=TBPRD).
Up-Down Count Mode	If CMPA/CMPB < TBPRD and the counter is incrementing, the event occurs on a compare match (TBCTR=CMPA or CMPB). If CMPA/CMPB is ≥ TBPRD, the event occurs on a period match (TBCTR = TBPRD).	If CMPA/CMPB < TBPRD and the counter is decrementing, the event occurs on a compare match (TBCTR=CMPA or CMPB). If CMPA/CMPB ≥ TBPRD, the event occurs on a period match (TBCTR=TBPRD).

7.5.6.6.4 AQCTLA and AQCTLB Shadow Mode Operations

To enable Action Qualifier mode changes which must occur at the end of a period even when the phase changes, shadowing of the AQCTLA and AQCTLB registers has been added on ePWMs type 2 and later. Additionally, shadow to active load on SYNC of these registers is supported as well. Shadowing of this register is enabled and disabled by the AQCTL[SHDWAQAMODE] and AQCTL[SHDWAQBMODE] bits. These bits enable and disable the AQCTLA shadow register and AQCTLB shadow register, respectively. The behavior of the two load modes is:

Shadow Mode:

The shadow mode for the AQCTLA is enabled by setting the AQCTL[SHDWAQAMODE] bit, and the shadow register for AQCTLB is enabled by setting the AQCTL[SHDWAQBMODE] bit. Shadow mode is disabled by default for both AQCTLA and AQCTLB. The memory address of the active register and the shadow register is identical.

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 as specified by the AQCTL[LDAQAMODE], AQCTL[LDAQBMODE], AQCTL[LDAQASYNC], and AQCTL[LDAQBSYNC] register bits:

- CTR = PRD: Time-base counter equal to the period (TBCTR = TBPRD).
- CTR = Zero: Time-base counter equal to zero (TBCTR = 0x00)
- Both CTR = PRD and CTR = Zero
- SYNC event caused by DCAEVT1 or DCBEVT1 or EPWMxSYNCI or TBCTL[SWFSYNC]
- Both SYNC event or a selection made by LDAQAMODE/LDAQBMODE

Global Load Support

Global load control mechanism can also be used for AQCTLA:AQCTLA2, AQCTLB:AQCTLB2, and AQCSFRC registers by configuring the appropriate bits in the global load configuration register (GLDCFG). When global load mode is selected, the transfer of contents from shadow register to active register for all registers that have this mode enabled, occurs at the same event as defined by the configuration bits in the Global Shadow to Active Load Control Register (GLDCTL). The global load control mechanism is explained in Section 7.5.6.4.8.

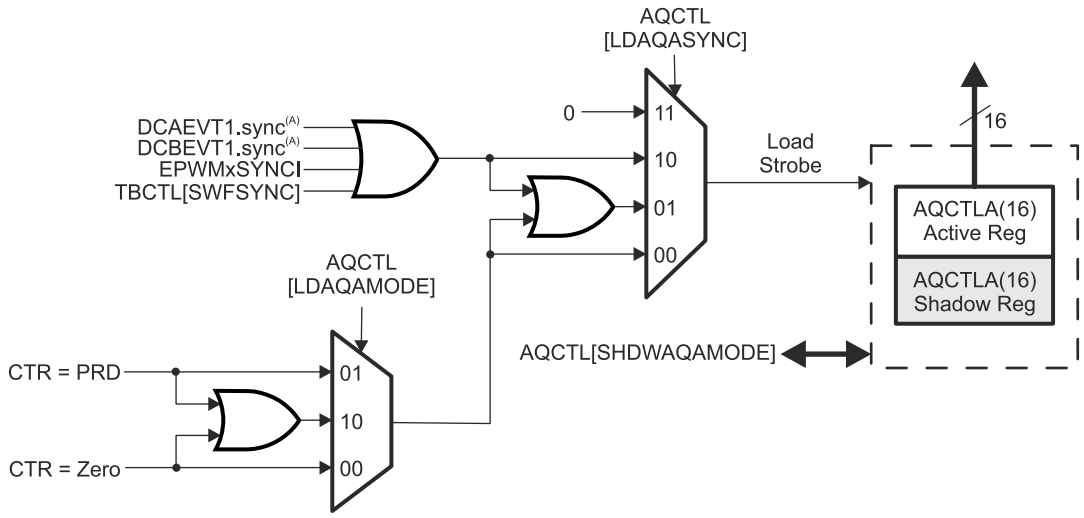
Immediate Load Mode:

If the immediate load mode is selected (that is, AQCTL[SHDWAQAMODE] = 0 or AQCTL[SHDWAQBMODE] = 0), then a read from or a write to the register goes directly to the active register. See Figure 7-177 and Figure 7-178

Note

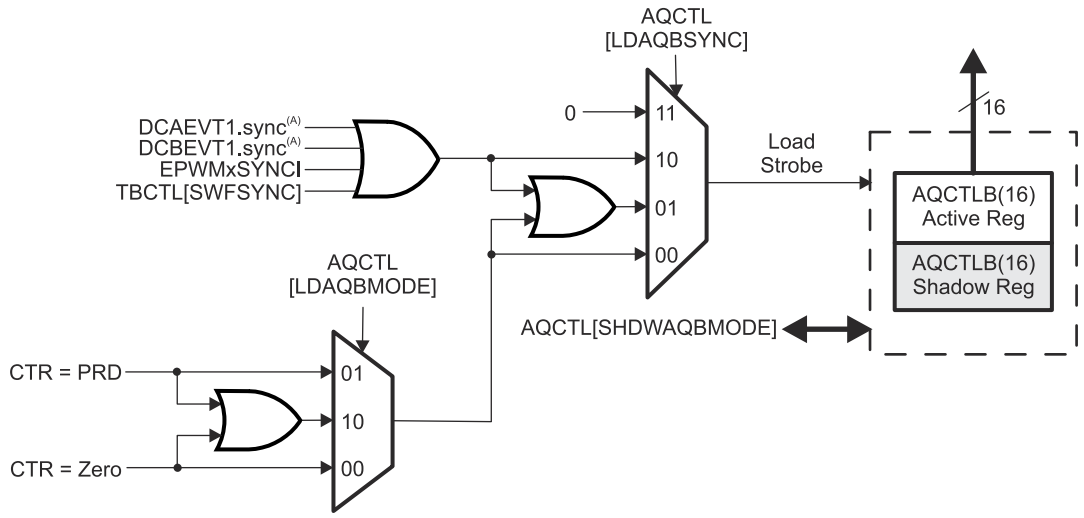
Shadow to Active Load of Action Qualifier Output A/B Control Register [AQCTLA and AQCTLB] on CMPA = 0 or CMPB = 0 boundary

If the Counter-Compare A Register (CMPA) or Counter-Compare B Register (CMPB) is set to a value of 0 and the action qualifier action on AQCTLA and AQCTLB is configured to occur in the same instant as a shadow to active load (that is, CMPA=0 and AQCTLA shadow to active load on TBCTR=0 using AQCTL register LDAQAMODE and LDAQAMODE bits), then both events enter contention. It is recommended to use a Non-Zero Counter-Compare when using Shadow to Active Load of Action Qualifier Output A/B Control Register on TBCTR = 0 boundary.



A. These events are generated by the ePWM digital compare (DC) submodule based on the levels of the TRIPIN inputs (for example, CMPSSx and $\overline{TZ}$ signals).

Figure 7-177. AQCTL[SHDWAQAMODE]



A. These events are generated by the ePWM digital compare (DC) submodule based on the levels of the TRIPIN inputs (for example, CMPSSx and $\overline{TZ}$ signals).

Figure 7-178. AQCTL[SHDWAQBMODE]

7.5.6.6.5 Configuration Requirements for Common Waveforms

Note

The waveforms in this chapter show the behavior of the ePWMs for a static compare register value. In a running system, the active compare registers (CMPA and CMPB) are typically updated from their respective shadow registers once every period. Specify when the update takes place: either when the time-base counter reaches zero or when the time-base counter reaches the 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 loading CMPA/CMPB on zero, then use CMPA/CMPB values greater than or equal to 1.
- If loading CMPA/CMPB on period, then use CMPA/CMPB values less than or equal to TBPRD-1.

  This means there is always 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 CMPA/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, you **must** load CMPA/CMPB on TBPRD. When CMPA/CMPB is not loaded on TBCTR=PRD, boundary conditions can occur depending on the timing of the write and the value written to CMPA/CMPB. 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.

When using up-count mode to generate an asymmetric PWM with deadband enabled:

- To achieve 0%-100% PWM use the following configuration: When the CMPA value is too close to 0 or PRD such that the following conditions are met (CMPX < Deadband) or (CMPX > PRD – Deadband), the actions specified by the AQCTL register for CMPX do not take effect. To avoid this, the AQCTL settings must be altered under these conditions only to generate either high or low pulses for CAU event (both set or both clear). Make sure that this software update is occurring synchronous to the PWM carrier cycle, and shadow mode is enabled.

When using up-down count mode to generate an asymmetric PWM with deadband enabled:

- To achieve 0%-100% PWM use the following configuration: When the CMPA value is too close to 0 or PRD such that the following conditions are met (CMPX < Deadband/2) or (CMPX > PRD – (Deadband)/2), the actions specified by the AQCTL register for CMPX do not take effect. To avoid this, the AQCTL settings must be altered under these conditions only to generate either high or low pulses for both CAU or CAD events (both set or both clear). Make sure that this software update is occurring synchronous to the PWM carrier cycle, and shadow mode is enabled.

See Using Enhanced Pulse Width Modulator (ePWM) Module for 0-100% Duty Cycle Control.

Figure 7-179 shows how a symmetric PWM waveform can be generated using the up-down-count mode of the TBCTR. 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 pulls the PWM output high. Likewise when the counter is decrementing, the compare match pulls the PWM signal low. When CMPA = 0, the PWM signal is high for the entire period giving a 100% duty waveform. When CMPA = TBPRD, the PWM signal is low achieving 0% duty.

When using this configuration in practice, if loading CMPA/CMPB on zero, then use CMPA/CMPB values greater than or equal to 1. If loading CMPA/CMPB on period, then use CMPA/CMPB values less than or equal to TBPRD-1. This means there is always a pulse of at least one TBCLK cycle in a PWM period which, when very short, tend to be ignored by the system.

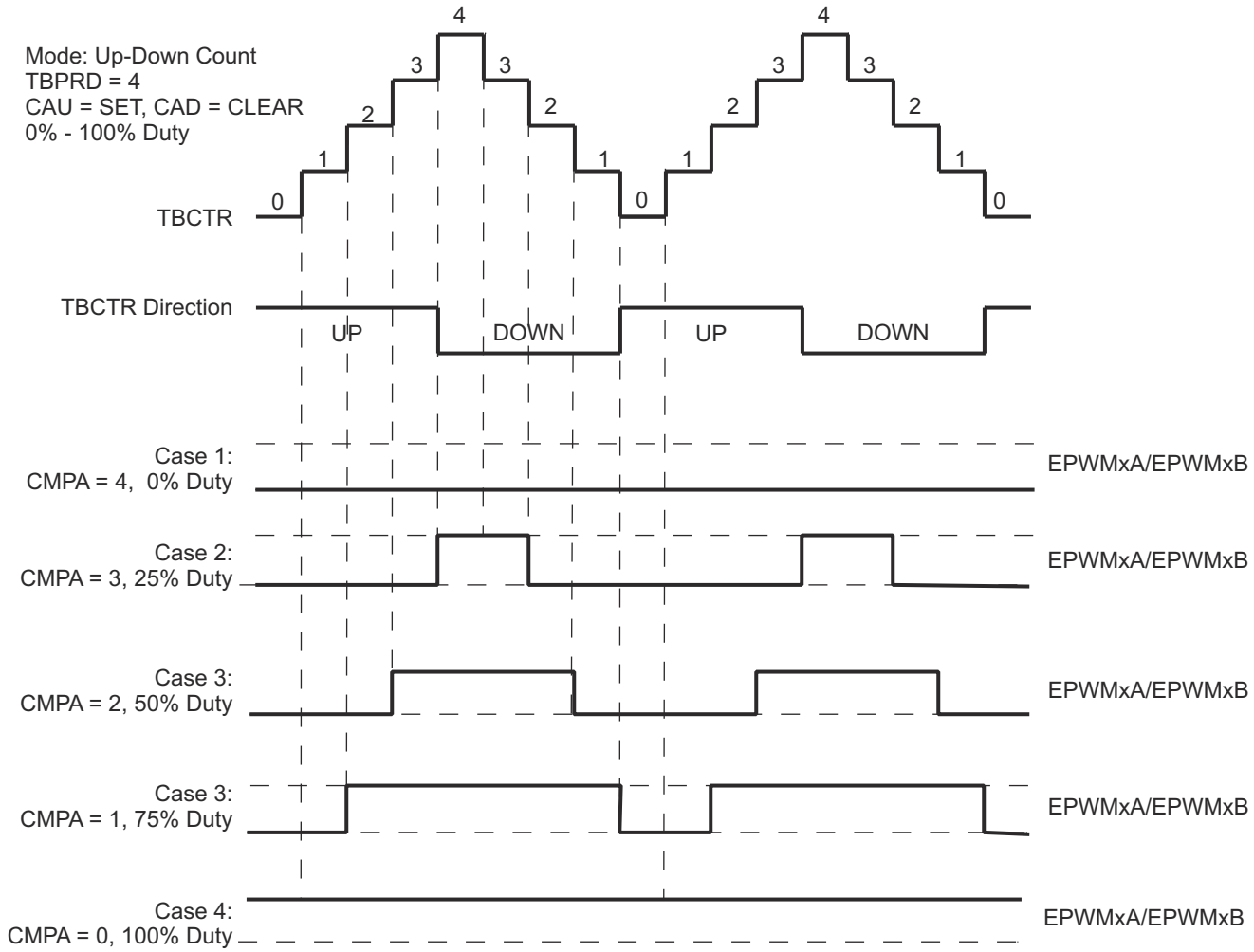
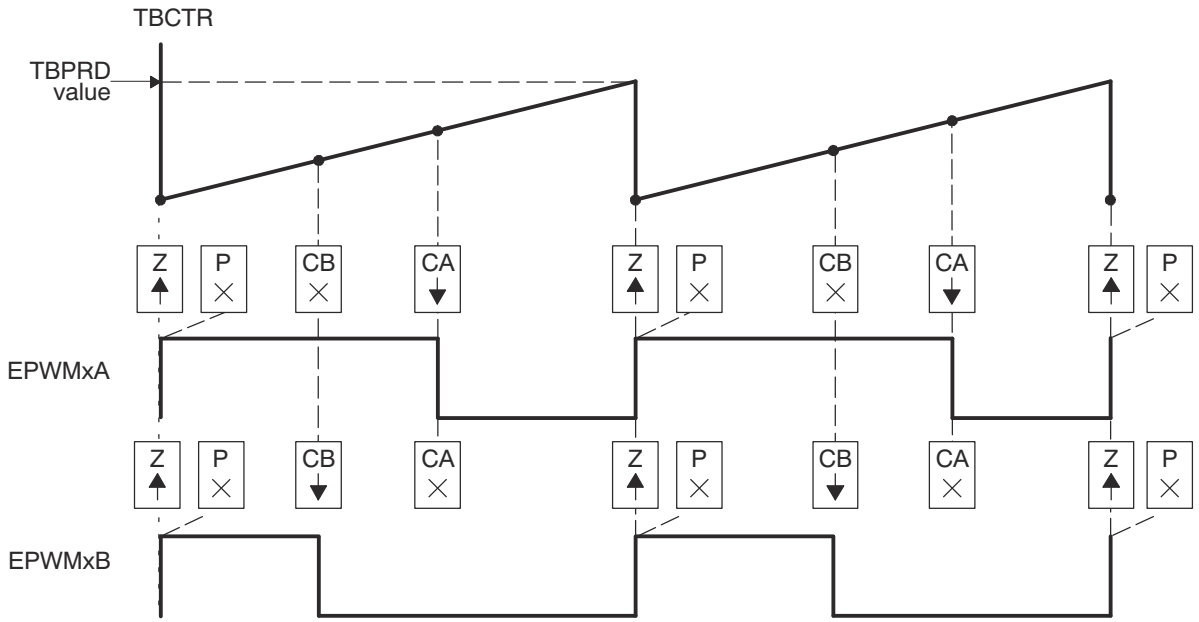


Figure 7-179. Up-Down Count Mode Symmetrical Waveform

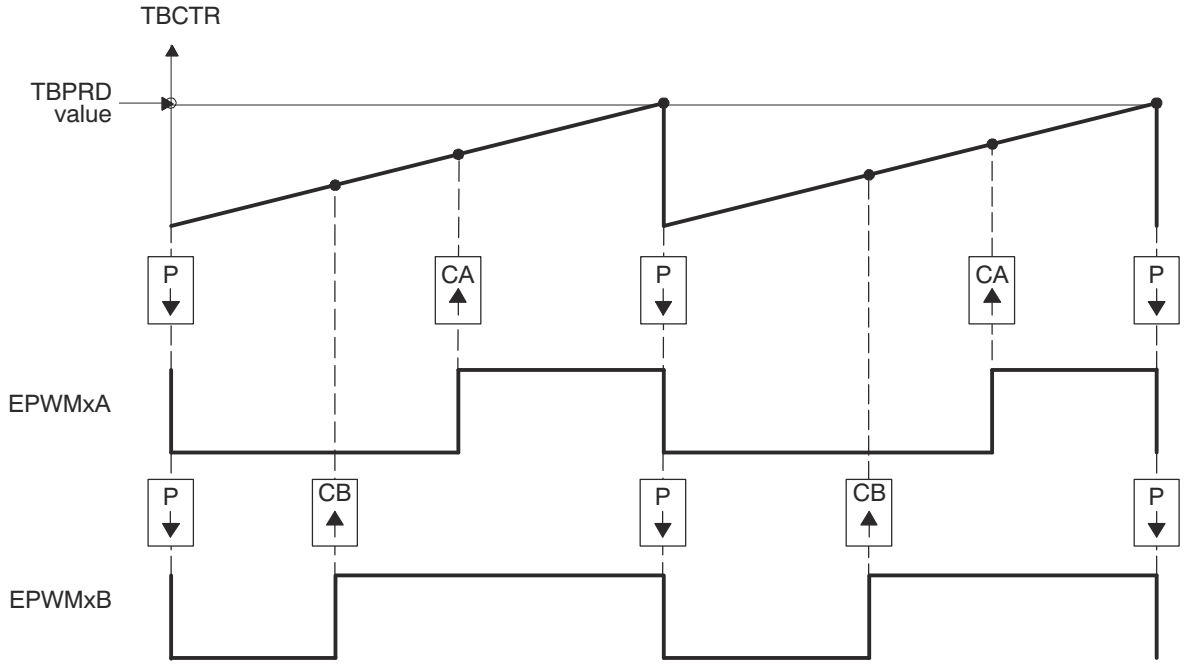
The PWM waveforms in Figure 7-180 through Figure 7-186 show some common action-qualifier configurations. Some conventions used in the figures and examples are as follows:

- TBPRD, CMPA, and CMPB refer to the value written in their respective registers. 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 count-down mode, Up means up-count mode and Down means down-count mode
- Sym = Symmetric, Asym = Asymmetric



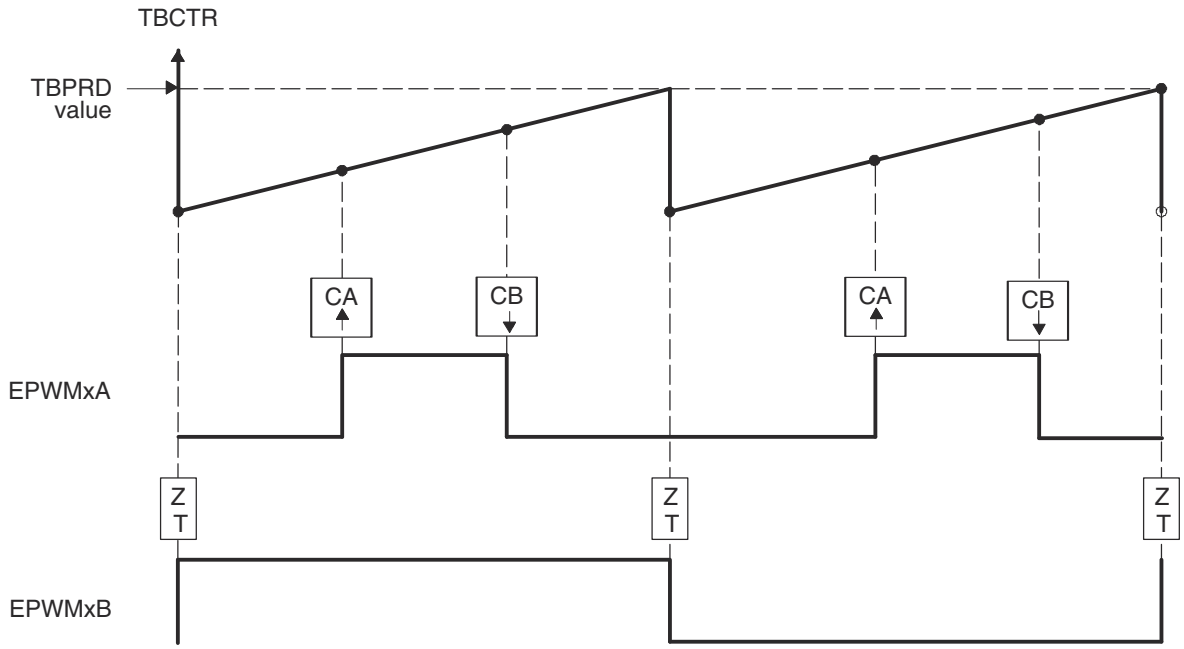
A. PWM period = (TBPRD + 1) × T_{TBCLK}
B. Duty modulation for EPWMxA is set by CMPA, and is active high (that is, high time duty proportional to CMPA).
C. Duty modulation for EPWMxB is set by CMPB and is active high (that is, high time duty proportional to CMPB).
D. The "Do Nothing" actions (X) are shown for completeness, but are not shown on subsequent diagrams.
E. Actions at zero and period, although appearing to occur concurrently, are actually separated by one TBCLK period. TBCTR wraps from period to 0000.

Figure 7-180. Up, Single Edge Asymmetric Waveform, with Independent Modulation on EPWMxA and EPWMxB—Active High



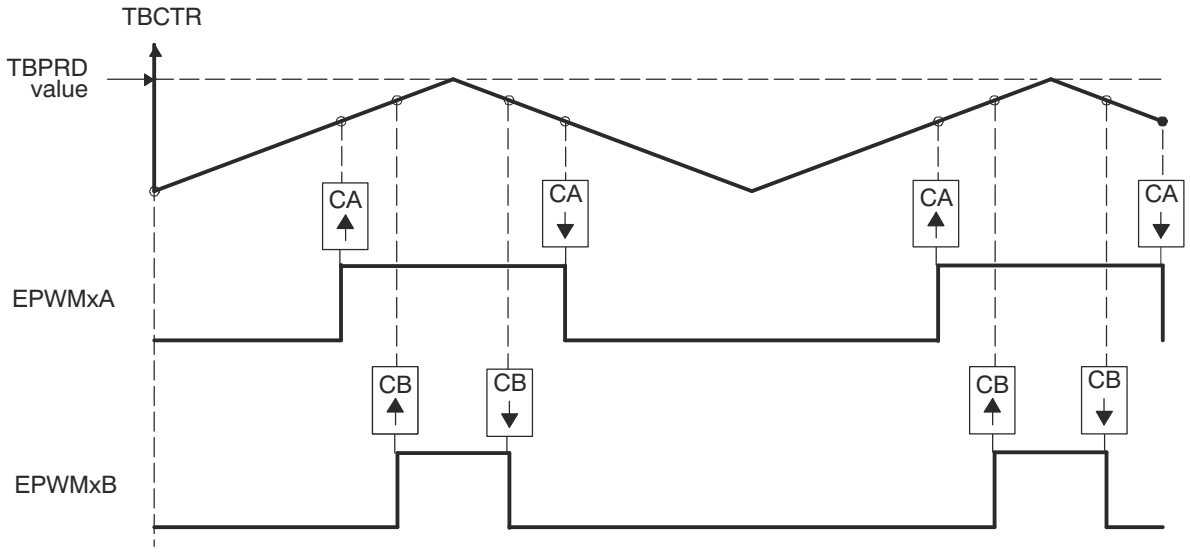
A. PWM period = (TBPRD + 1) × T_{TBCLK}
B. Duty modulation for EPWMxA is set by CMPA, and is active low (that is, the low time duty is proportional to CMPA).
C. Duty modulation for EPWMxB is set by CMPB and is active low (that is, the low time duty is proportional to CMPB).
D. Actions at zero and period, although appearing to occur concurrently, are actually separated by one TBCLK period. TBCTR wraps from period to 0000.

Figure 7-181. Up, Single Edge Asymmetric Waveform with Independent Modulation on EPWMxA and EPWMxB—Active Low



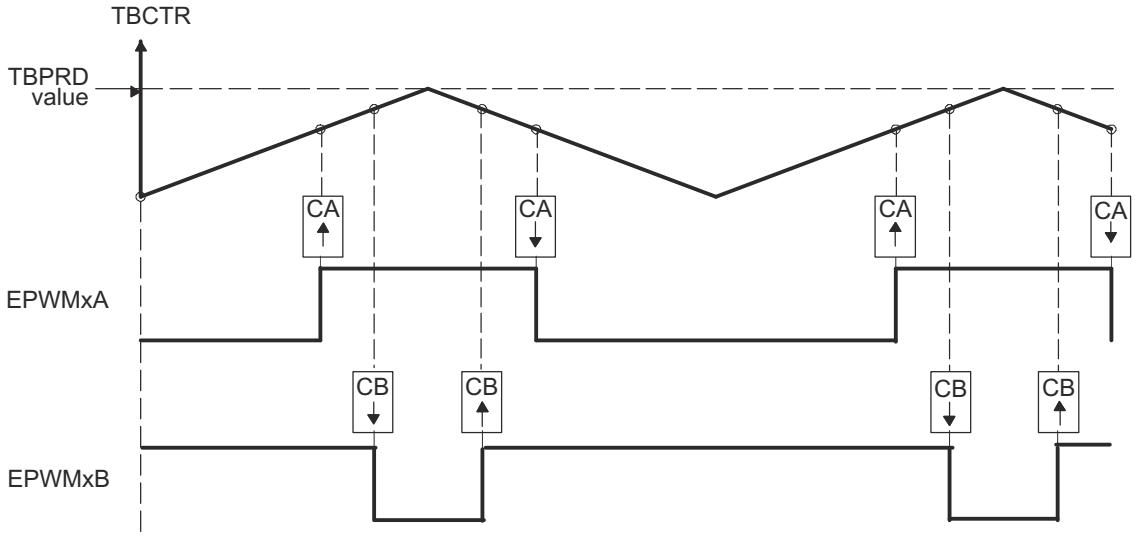
A. PWM frequency = 1/((TBPRD + 1) × T_{TBCLK})
B. Pulse can be placed anywhere within the PWM cycle (0000 - TBPRD)
C. High time duty proportional to (CMPB - CMPA)

Figure 7-182. Up-Count, Pulse Placement Asymmetric Waveform With Independent Modulation on EPWMxA



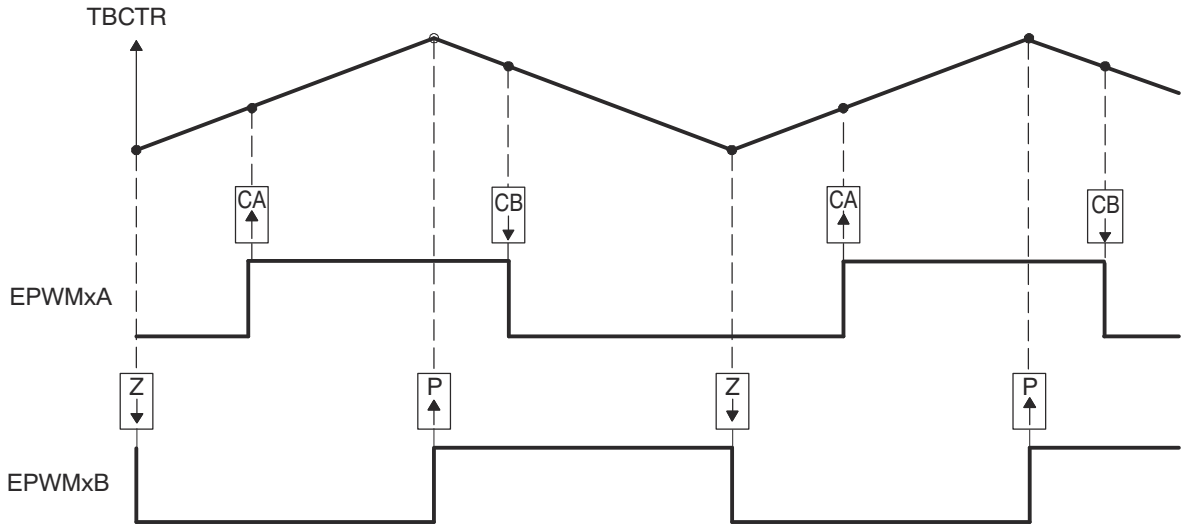
A. PWM period = 2 x TBPRD × T_{TBCLK}
B. Duty modulation for EPWMxA is set by CMPA, and is active low (that is, the low time duty is proportional to CMPA).
C. Duty modulation for EPWMxB is set by CMPB and is active low (that is, the low time duty is proportional to CMPB).
D. Outputs EPWMxA and EPWMxB can drive independent power switches.

Figure 7-183. Up-Down Count, Dual-Edge Symmetric Waveform, with Independent Modulation on EPWMxA and EPWMxB — Active Low



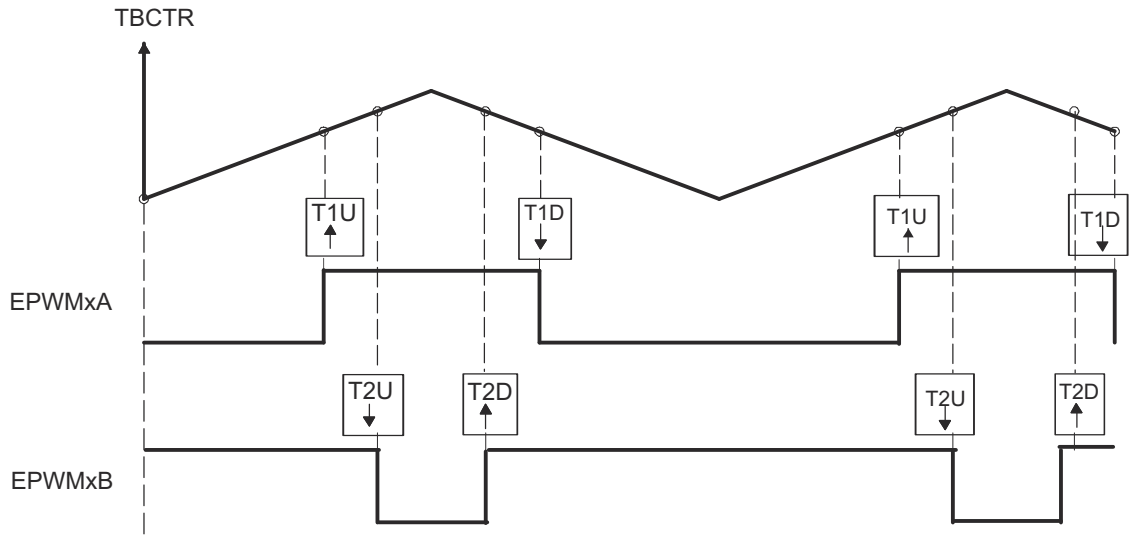
A. PWM period = 2 × TBPRD × T_{TBCLK}
B. Duty modulation for EPWMxA is set by CMPA, and is active low, that is, low time duty proportional to CMPA.
C. Duty modulation for EPWMxB is set by CMPB and is active high, that is, high time duty proportional to CMPB.
D. Outputs EPWMx can drive upper/lower (complementary) 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 7-184. Up-Down Count, Dual-Edge Symmetric Waveform, with Independent Modulation on EPWMxA and EPWMxB — Complementary



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, Clear on CMPA, Set on CMPB).
F. Duty modulation for EPWMxB is fixed at 50% (utilizes spare action resources for EPWMxB).

Figure 7-185. Up-Down Count, Dual-Edge Asymmetric Waveform, with Independent Modulation on EPWMxA—Active Low



A. PWM period = 2 × TBPRD × TTBCLK
B. Independent T1 event actions when counter is counting up and when the counter is counting down are used to generate EPWMxA output.
C. Independent T2 event actions when counter is counting up and when the counter is counting down are used to generate EPWMxB output.
D. TZ1 is selected as the source for T1.
E. TZ2 is selected as the source for T2.

Figure 7-186. Up-Down Count, PWM Waveform Generation Utilizing T1 and T2 Events

7.5.6.7 Dead-Band Generator (DB) Submodule

Figure 7-187 illustrates the dead-band submodule within the ePWM.

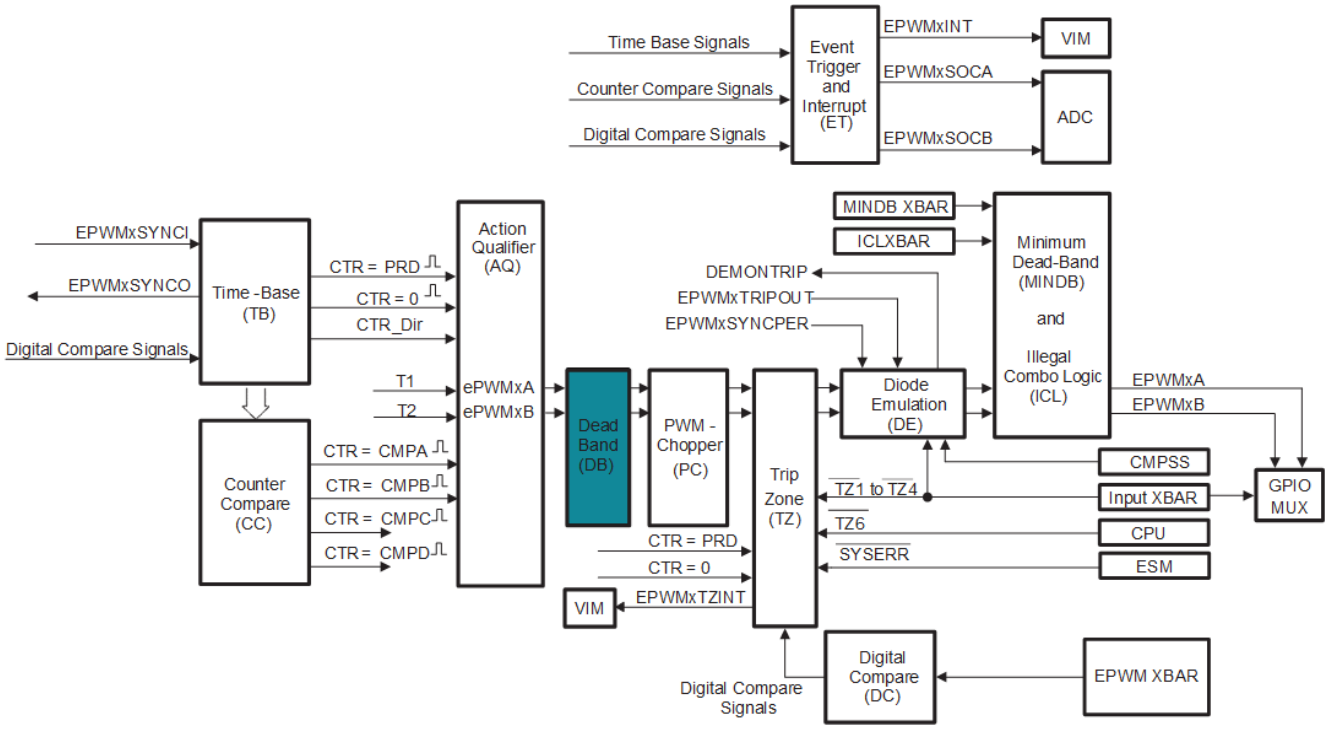


Figure 7-187. Dead_Band Submodule

7.5.6.7.1 Purpose of the Dead-Band Submodule

The action-qualifier (AQ) module section discussed how the AQ module 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 submodule described here must be used.

The key functions of the dead-band module 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

7.5.6.7.2 Dead-band Submodule Additional Operating Modes

On type 1 ePWM RED can appear on one channel output and FED can appear on the other channel output.

The following list shows the distinct difference between type 1 and type 4 modules with respect to dead-band operating modes:

- By adding S6, S7, and S8 in Figure 7-188, RED and FED can appear on both the A-channel and B-channel outputs. Additionally, both RED and FED together can be applied to either the A-channel or B-channel outputs to allow B-channel phase shifting with respect to the A-channel.

Note

Phase shifting B-channel with respect to the A-channel using the dead-band submodule additional operating modes has limitations with respect to the choice of RED and FED delay with respect to the operating duty cycle of the ePWMxA and ePWMxB outputs.

- The dead-band counters have also been increased to 14 bits
- Deadband and deadband high-resolution registers are now shadowed
- High-resolution deadband RED and FED have been enabled using the DBREDHR and DBFEDHR registers

Note

The PWM chopper is not enabled when high-resolution deadband is enabled.

High-resolution deadband RED and FED requires half-cycle clocking mode (DBCTL[HALFCYCLE] = 1).

Cannot have both RED and FED together applied to both ePWMxA and ePWMxB. RED and FED together can be applied only to either OutA OR OutB.

Phase shifting B-channel with respect to the A-channel: When PWMxB is derived from PWMxA using the DEDB_MODE bit and by delaying rising edge and falling edge by the phase shift amount. When the duty cycle value on PWMxA is less than this phase shift amount, PWMxA's falling edge has precedence over the delayed rising edge for PWMxB. Make sure the duty cycle value of the current waveform applied to the dead-band module is greater than the required phase shift amount.

The Type 4 action qualifier and dead-band outputs of the ePWM module are delayed by one TBCLK cycle in comparison to the Type 2 ePWM module, although the Type 4 behavior is the same as the Type 3 PWM. Both PWMA and PWMB signals are delayed under all circumstances.

Shadow Mode:

The shadow mode for the DBRED is enabled by setting the DBCTL[SHDWDBREDMODE] bit and the shadow register for DBFED is enabled by setting the DBCTL [SHDWDBFEDMODE] bit. Shadow mode is disabled by default for both DBRED and DBFED. The memory address of the active register and the shadow register is identical.

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 as specified by the DBCTL [LOADREDMODE] and DBCTL [LOADFEDMODE] register bits:

- CTR = PRD: Time-base counter equal to the period (TBCTR = TBPRD).
- CTR = Zero: Time-base counter equal to zero (TBCTR = 0x00)
- Both CTR = PRD and CTR = Zero

The DBCTL register can be shadowed. The shadow mode for DBCTL is enabled by setting the DBCTL2[SHDWDBCTLMODE] bit. 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 as specified by the DBCTL2[LOADDBCTLMODE] register bit:

- CTR = PRD: Time-base counter equal to the period (TBCTR = TBPRD)
- CTR = Zero: Time-base counter equal to zero (TBCTR = 0x00)
- Both CTR = PRD and CTR = Zero

Note

The application software must enable shadow load mode in the DBCTL[SHDWDBREDMODE] and DBCTL[SHDWDBFEDMODE] **before** programming values for the DBRED and DBFED registers. If the shadow register is enabled **after** programming the DBRED and DBFED registers, the DBRED and DBFED registers are loaded with a value of 0.

Global Load Support

Global load control mechanism can also be used for DBRED:DBREDHR, DBFED:DBFEDHR, and DBCTL registers by configuring the appropriate bits in the global load configuration register (GLDCFG). When global load mode is selected the transfer of contents from shadow register to active register, for all registers that have this mode enabled, occurs at the same event as defined by the configuration bits in the Global Shadow to Active Load Control Register (GLDCTL). The Global load control mechanism is explained in Section 7.5.6.4.8.

Note

When DBRED/DBFED active is loaded with a new shadow value while DB counters are counting, the new DBRED/DBFED value only affects the NEXT PWMx edge and not the current edge.

A Deadband value of zero cannot be used when the Global Shadow to Active Load is set to occur at CTR=ZERO. Similarly, a Deadband value of PRD cannot be used when the Global Shadow to Active Load is set to occur at CTR=PRD.

TBPRDHR cannot be used with Global load. If high-resolution period must be changed in the application, users must write to the individual period registers from an ePWM ISR (The ISR must be synchronous with the PWM switching period), where the Global Load One-Shot bit is also written to.

7.5.6.7.3 Simultaneous Writes to DBRED and DBFED Registers Between ePWM Modules (Type 5 EPWM)

LinkingDBRED and DBFED Starting with type 5 EPWM, the DBRED and DBFED values can be linked from one ePWM to another. This allowsfor simultaneous writes to all linked ePWM registers. For more information, review the EPWMXLINK2 register.

Similar to the EPWMXLINK register, the register description for EPWMXLINK2 clearly explains the linked registerbit-field values for corresponding ePWM. An example of using the EPWMXLINK2 is linking ePWM2

7.5.6.7.4 Operational Highlights for the Dead-Band Submodule

The configuration options for the dead-band submodule are shown in Figure 7-188.

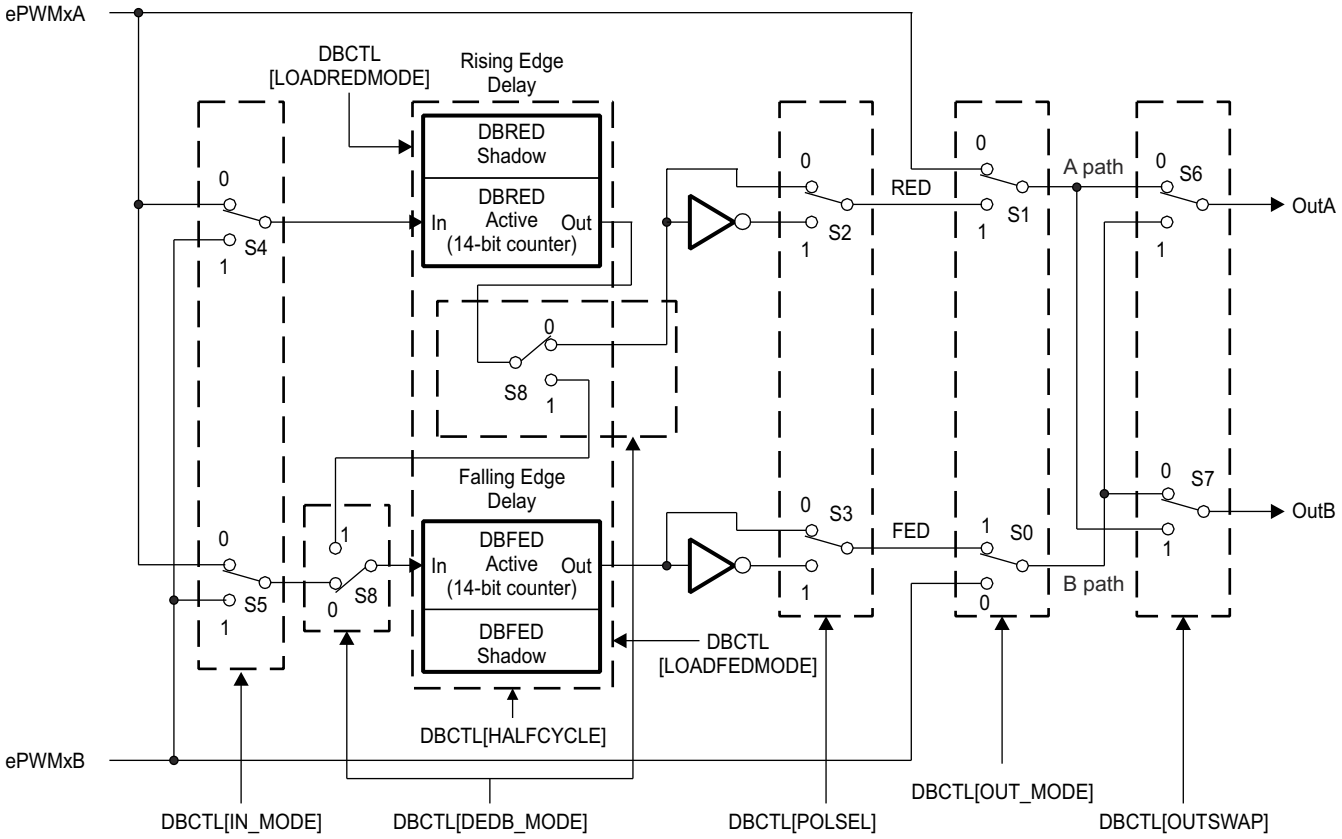


Figure 7-188. Configuration Options for the Dead-Band Submodule

Although all combinations are supported, not all are typical usage modes. Table 7-127 documents some classical dead-band configurations. These modes assume that the DBCTL[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 7-127 fall into the following categories:

- **Mode 1: Bypass both falling-edge delay (FED) and rising-edge delay (RED):** Allows the user to fully disable the dead-band submodule from the PWM signal path.
- **Mode 2-5: Classical Dead-Band Polarity Settings:** These represent typical polarity configurations that can address all the active-high and active-low modes required by available industry power switch gate drivers. The waveforms for these typical cases are shown in Figure 7-189 Note that to generate equivalent waveforms to Figure 7-189, 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 7-127 show combinations where either the falling-edge-delay (FED) or rising-edge-delay (RED) blocks are bypassed.

Figure 7-189 shows waveforms for typical cases where 0% < duty < 100%.

Table 7-127. Classical Dead-Band Operating Modes

Mode	Mode Description	DBCTL[POLSEL]		DBCTL[OUT_MODE]	
		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

Table 7-128. Additional Dead-Band Operating Modes

Mode Description	DBCTL[DEDB-MODE]	DBCTL[OUTSWAP]	
	S8	S6	S7
EPWMxA and EPWMxB signals are as defined by OUT-MODE bits.	0	0	0
EPWMxA = A-path as defined by OUT-MODE bits. EPWMxB = A-path as defined by OUT-MODE bits (rising edge delay or delay-bypassed A-signal path)	0	0	1
EPWMxA = B-path as defined by OUT-MODE bits (falling edge delay or delay-bypassed B-signal path) EPWMxB = B-path as defined by OUT-MODE bits	0	1	0
EPWMxA = B-path as defined by OUT-MODE bits (falling edge delay or delay-bypassed B-signal path) EPWMxB = A-path as defined by OUT-MODE bits (rising edge delay or delay-bypassed A-signal path)	0	1	1
Rising edge delay applied to EPWMxA / EPWMxB as selected by S4 switch (IN-MODE bits) on A signal path only. Falling edge delay applied to EPWMxA / EPWMxB as selected by S5 switch (IN-MODE bits) on B signal path only.	0	X	X
Rising edge delay and falling edge delay applied to source selected by S4 switch (IN-MODE bits) and output to B signal path only.(1)	1	X	X

(1) When this bit is set to 1, the user can always either set OUT_MODE bits such that Apath = InA or set OUTSWAP bits such that EPWMxA=Bpath. Otherwise, EPWMxA is invalid.

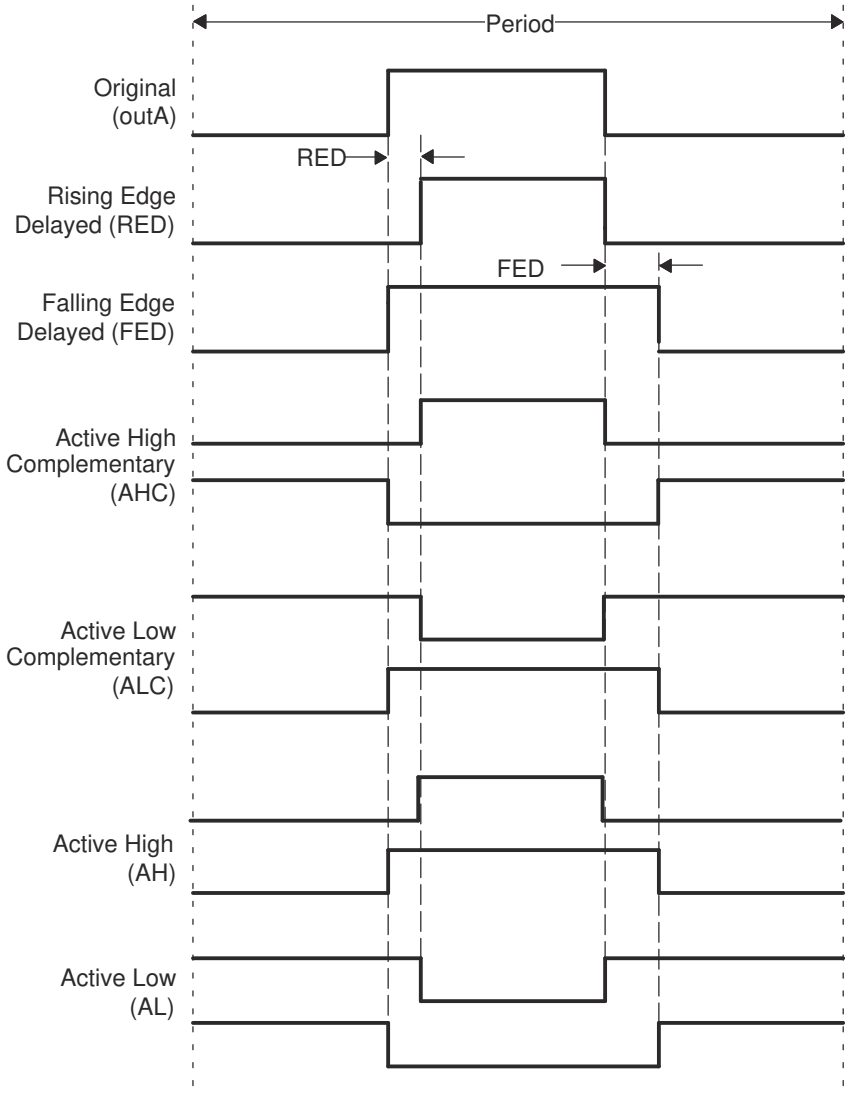


Figure 7-189. Dead-Band Waveforms for Typical Cases (0% < Duty < 100%)

The dead-band submodule supports independent values for rising-edge (RED) and falling-edge (FED) delays. The amount of delay is programmed using the DBRED and DBFED registers. These are 10-bit registers and their value represents the number of time-base clock, TBCLK, periods by which a signal edge is delayed. For example, the formula to calculate falling-edge-delay and rising-edge-delay is:

$FED = DBFED \times T_{TBCLK}$

$RED = DBRED \times T_{TBCLK}$

Where T_{TBCLK} is the period of TBCLK, the prescaled version of EPWMCLK.

For convenience, delay values for various TBCLK options are shown in Table 7-129. The ePWM input clock frequency that these delay values been computed by is 100 MHz.

Table 7-129. Dead-Band Delay Values in µS as a Function of DBFED and DBRED

Dead-Band Value	Dead-Band Delay in µS		
DBFED, DBRED	TBCLK = EPWMCLK/1	TBCLK = EPWMCLK /2	TBCLK = EPWMCLK/4
1	0.01 µS	0.02 µS	0.04 µS
5	0.05 µS	0.10 µS	0.20 µS
10	0.10 µS	0.20 µS	0.40 µS
100	1.00 µS	2.00 µS	4.00 µS
200	2.00 µS	4.00 µS	8.00 µS
400	4.00 µS	8.00 µS	16.00 µS
500	5.00 µS	10.00 µS	20.00 µS
600	6.00 µS	12.00 µS	24.00 µS
700	7.00 µS	14.00 µS	28.00 µS
800	8.00 µS	16.00 µS	32.00 µS
900	9.00 µS	18.00 µS	36.00 µS
1000	10.00 µS	20.00 µS	40.00 µS

When half-cycle clocking is enabled, the formula to calculate the falling-edge-delay and rising-edge-delay becomes:

$FED = DBFED \times T_{TBCLK}/2$

$RED = DBRED \times T_{TBCLK}/2$

7.5.6.8 Minimum Dead-Band (MINDB) + Illegal Combination Logic (ICL) Submodules

Figure 7-190 illustrates the minimum dead-band and illegal combo submodule within the ePWM.

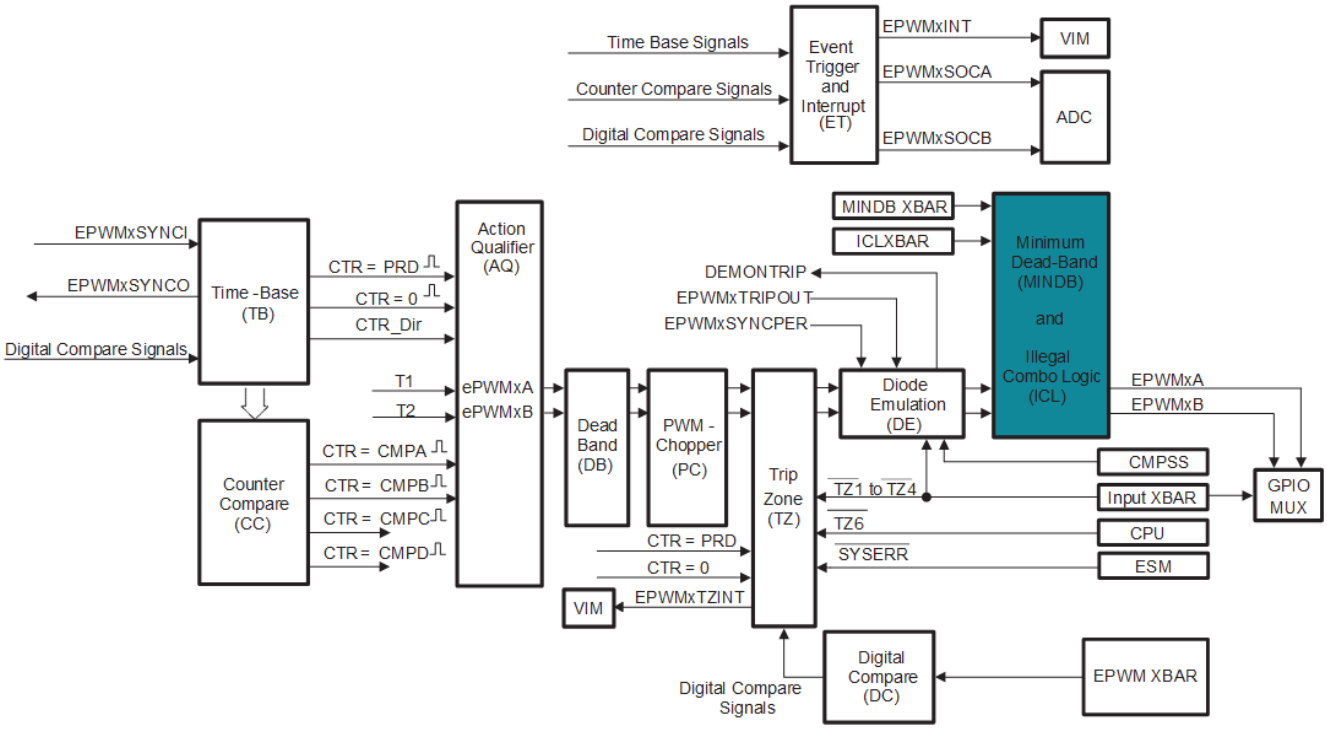


Figure 7-190. Minimum Dead-Band & Illegal Combo Logic Submodule

7.5.6.8.1 Minimum Dead-Band (MINDB)

To make sure that the minimum dead band property is not violated, as the application switches between normal mode and DE mode and due to the PWMs potentially switching based on trip inputs, a minimum dead band circuitry show in Figure 7-191 is required.

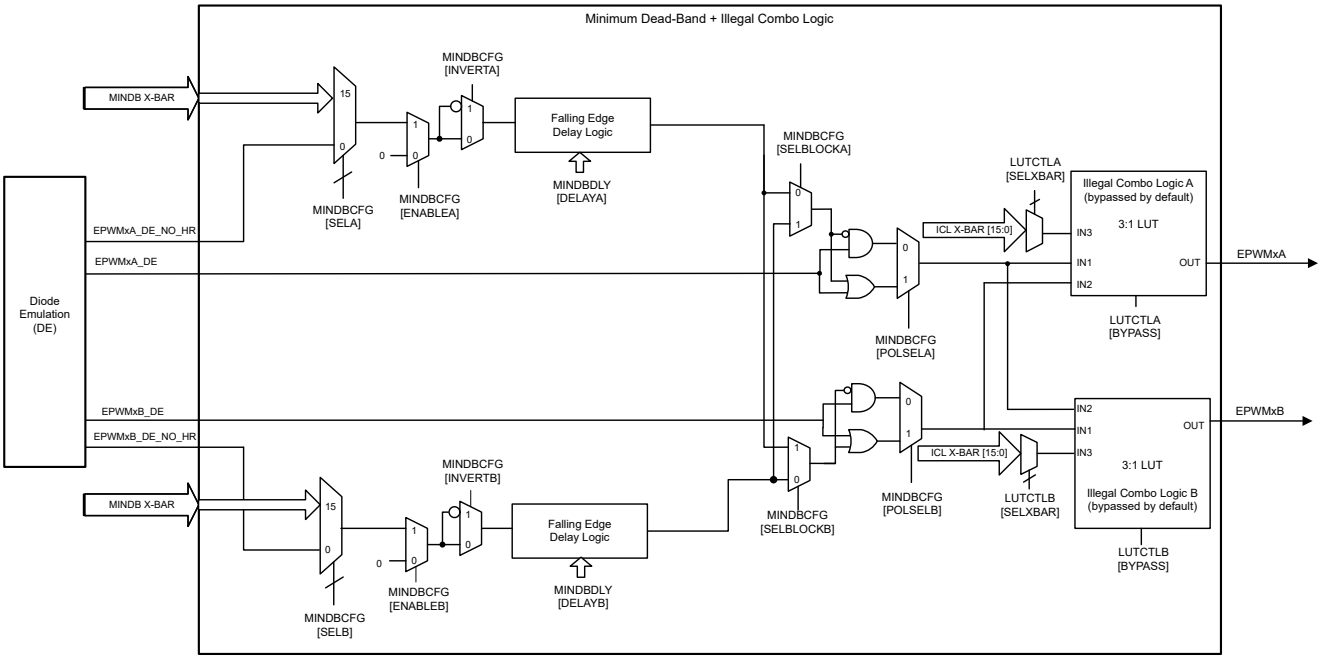


Figure 7-191. Minimum Dead-Band and Illegal Combo Logic Block Diagram

The minimum dead band block provides the ability to configure the minimum dead band duration between a complimentary set of PWMs.

Minimum dead-band logic involves generating a blocking signal (BLOCKA, BLOCKB) after the falling edge of the EPWMA/B_DE. These block signals are used to block transition on the other signal. The input to BLOCKA(B) signal generators is configurable. Normally the sources are EPWMA/B_DB_NO_HR. However, there is a provision provided to select any of the MINDB X-BAR outputs. This provides flexibility to support some of the other application scenarios.

The selected source is fed to the BLOCK signal generation logic. Block signal generation involves, detecting the falling edge based on which BLOCK signal goes high and stretching the BLOCK signal for DELAYA/B cycles, which are software configurable.

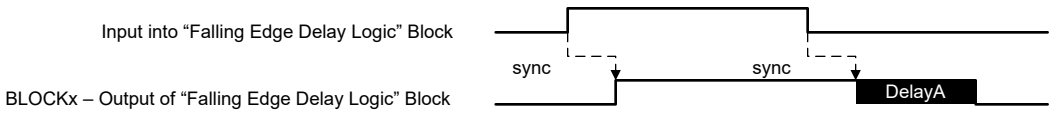


Figure 7-192. Minimum Dead-Band Block Signal Generation

In Figure 7-193 and Figure 7-194, EPWMxA_DE and BLOCKB are getting ANDed and EPWMxB_DE and BLOCKA are getting ANDed.

Note

Red shade is indicative of incorrect scenario.

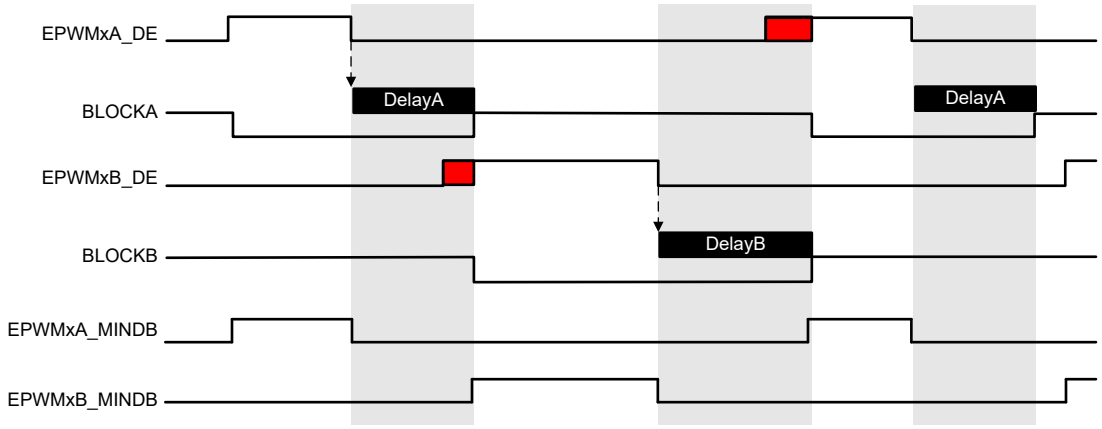


Figure 7-193. Example: Rising Edge on EPWMxA_DE and EPWMxB_DE While Delay is Being Applied

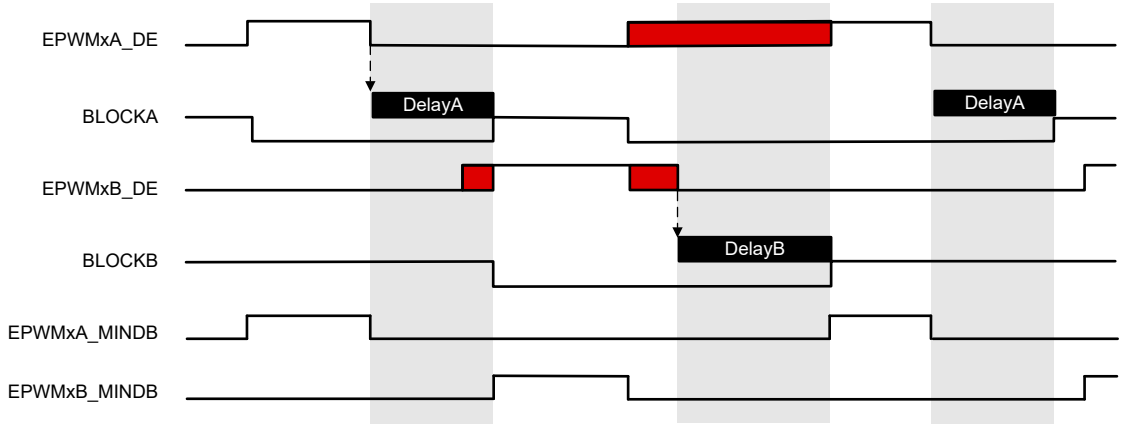


Figure 7-194. Example: Rising Edge on EPWMxA_DE while EPWMxB_DE is Still High

Figure 7-195 illustrates that a rising edge during the delay application does not affect the BLOCKA generation, same behavior is applied to BLOCKB.

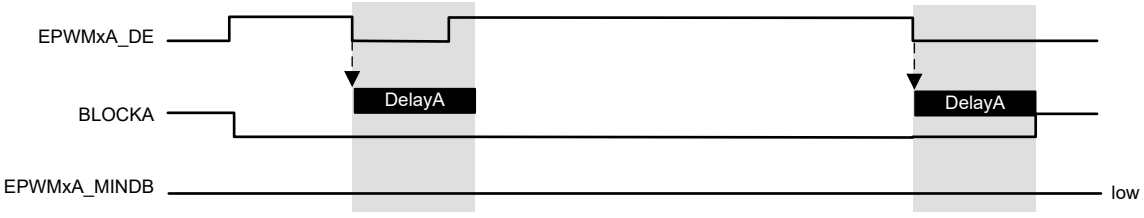


Figure 7-195. Rising Edge During Delay

Figure 7-196 showcases what happens when another falling edge occurs during the delay application. In this scenario, BLOCKA stays low until both DELAYA values are complete, same behavior is applied to BLOCKB.

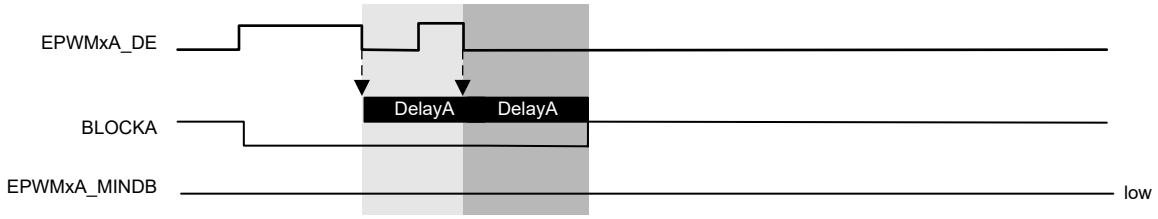


Figure 7-196. Rising Edge and Falling Edge During Delay

7.5.6.8.2 Illegal Combo Logic (ICL)

As PWM generation logic gets more configurable and the interaction between multiple PWM instances increases, there is potential for corner cases during applications resulting in unintended PWM states. To detect and make sure that under no circumstance, the PWM states result in potentially hazardous combinations, a Look Up Table (LUT) has been added. By default, the LUT logic is bypassed, LUTXTLx[BYPASS]. When not bypassed, based on the combination of values on Input 3 (IN3), Input 2 (IN2), and Input 1 (IN1), the value driven on OUT is determined by the bits in the LUTCTLx [23:16] register. Input 3 into the LUT comes from one of the ICL X-BAR inputs.

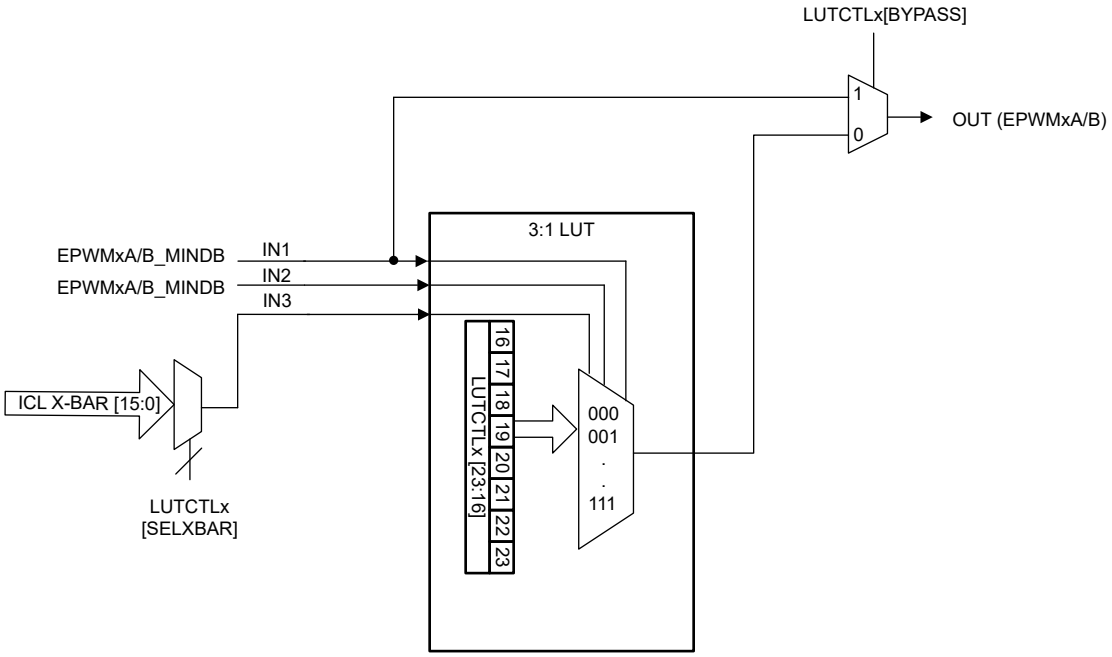


Figure 7-197. Illegal Combo Logic Block Diagram

7.5.6.9 PWM Chopper (PC) Submodule

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 pulse transformer-based gate drivers to control the power switching elements are needed.

Figure 7-198 illustrates the PWM chopper submodule within the ePWM.

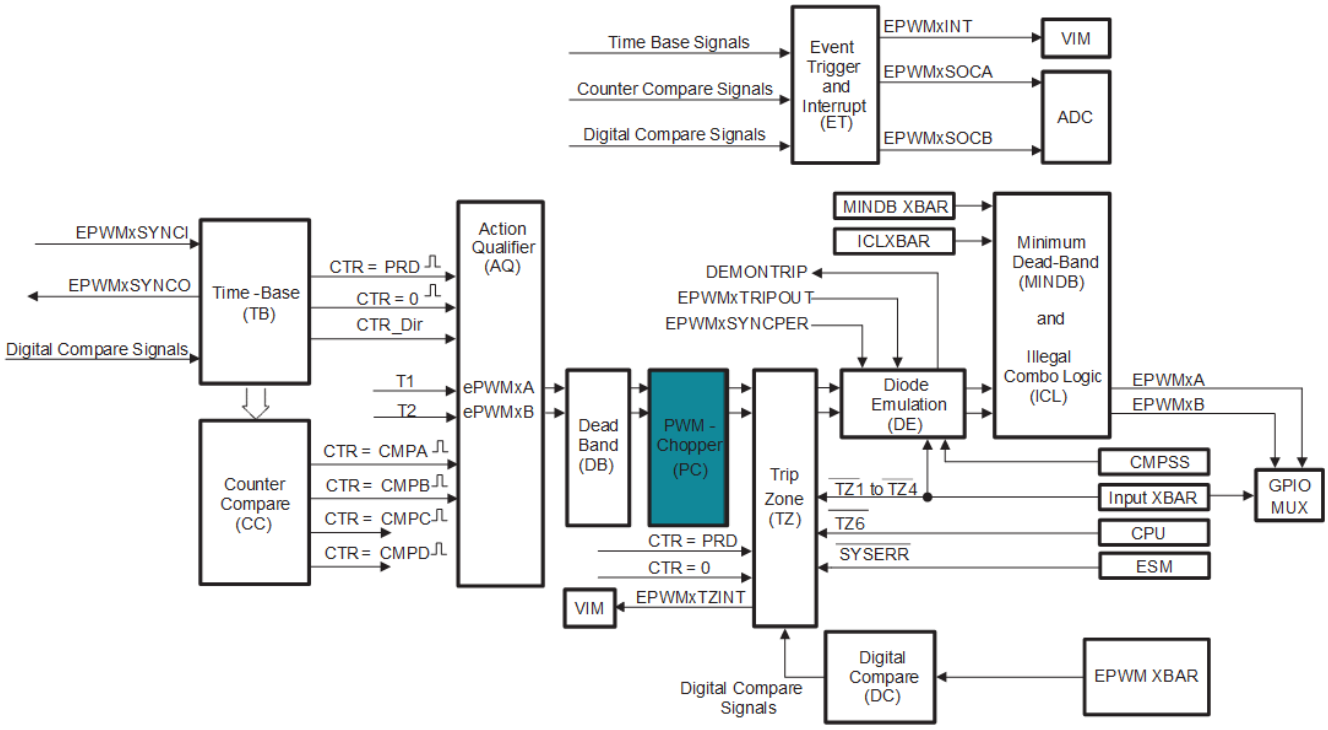


Figure 7-198. PWM Chopper Submodule

7.5.6.9.1 Purpose of the PWM Chopper Submodule

The key functions of the PWM chopper submodule are:

- 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

7.5.6.9.2 Operational Highlights for the PWM Chopper Submodule

Figure 7-199 shows the operational details of the PWM chopper submodule. The carrier clock is derived from EPWMCLK. The clock frequency and duty cycle are controlled using the CHPFREQ and CHPDUTY bits in the PCCTL register. The one-shot block is a feature that provides a high energy first pulse to make sure hard and fast power switch turn on, while the subsequent pulses sustain pulses, making sure the power switch remains on. The one-shot width is programmed using the OSHTWTH bits. The PWM chopper submodule can be fully disabled (bypassed) using the CHPEN bit.

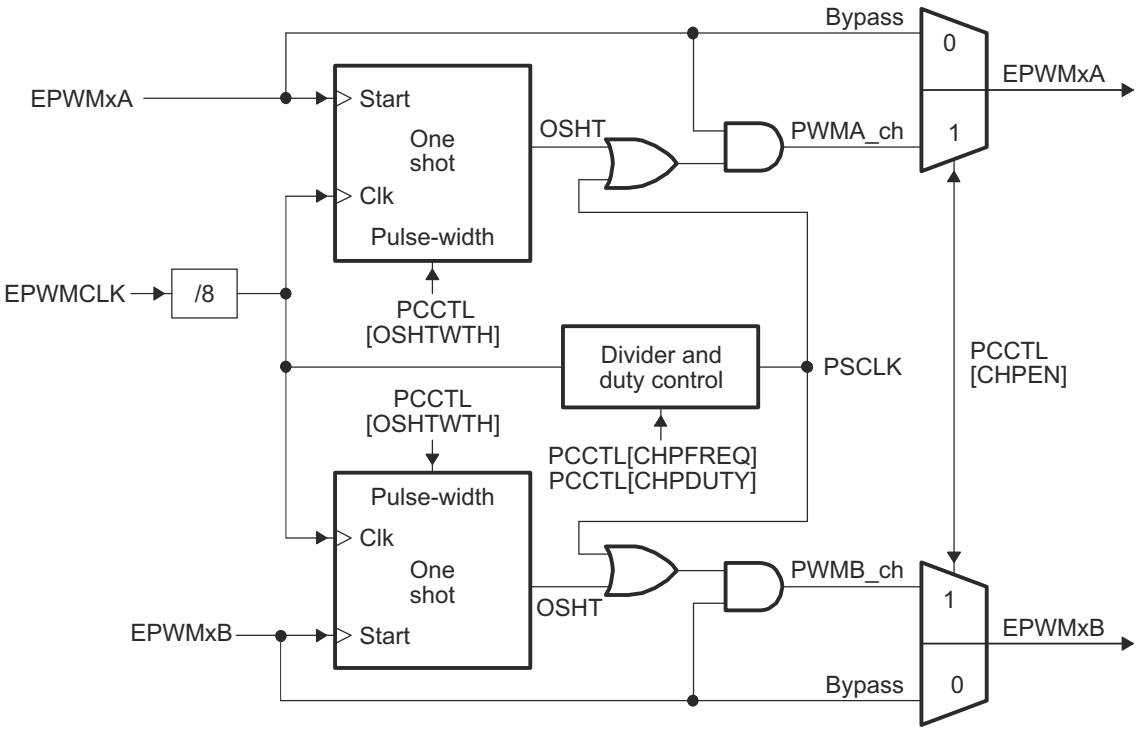


Figure 7-199. PWM Chopper Submodule Operational Details

7.5.6.9.3 Waveforms

Figure 7-200 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.

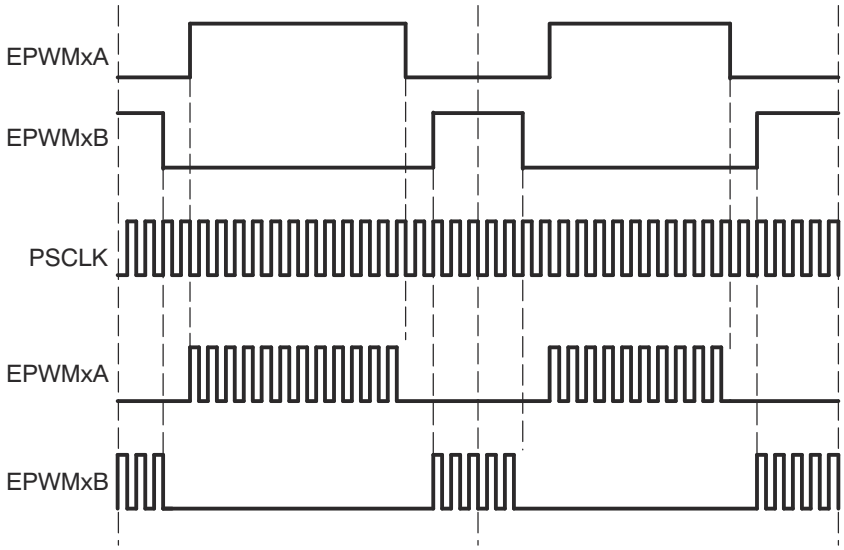


Figure 7-200. Simple PWM Chopper Submodule Waveforms Showing Chopping Action Only

7.5.6.9.3.1 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:

$$T_{1stpulse} = T_{EPWMCLK} \times 8 \times OSHTWTH$$

Where $T_{EPWMCLK}$ is the period of the system clock (EPWMCLK) and OSHTWTH is the four control bits (value from 1 to 16)

Figure 7-201 shows the first and subsequent sustaining pulses and Table 7-130 gives the possible pulse width values for a EPWMCLK = 80 MHz.

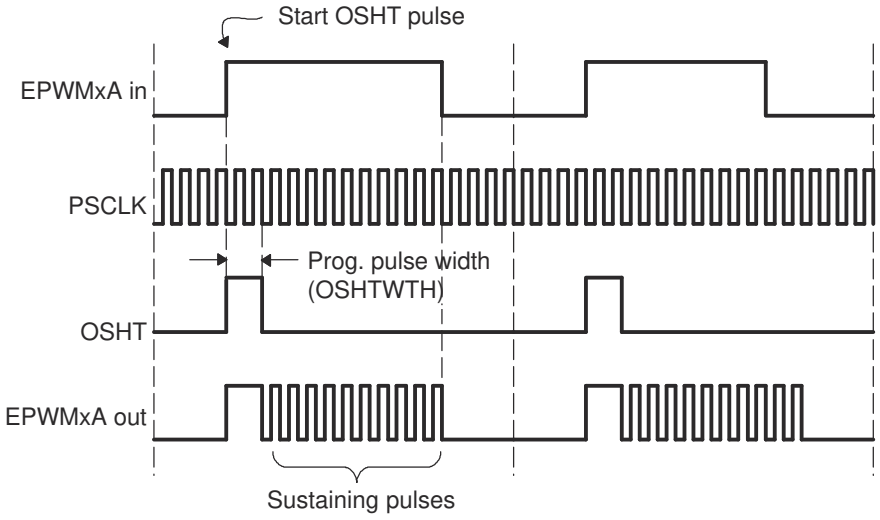


Figure 7-201. PWM Chopper Submodule Waveforms Showing the First Pulse and Subsequent Sustaining Pulses

Table 7-130. Possible Pulse Width Values for EPWMCLK = 80 MHz

OSHTWTH (hex)	Pulse Width (nS)
0	100
1	200
2	300
3	400
4	500
5	600
6	700
7	800
8	900
9	1000
A	1100
B	1200
C	1300
D	1400
E	1500
F	1600

7.5.6.9.3.2 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 make sure 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 using software control.

Figure 7-202 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%.

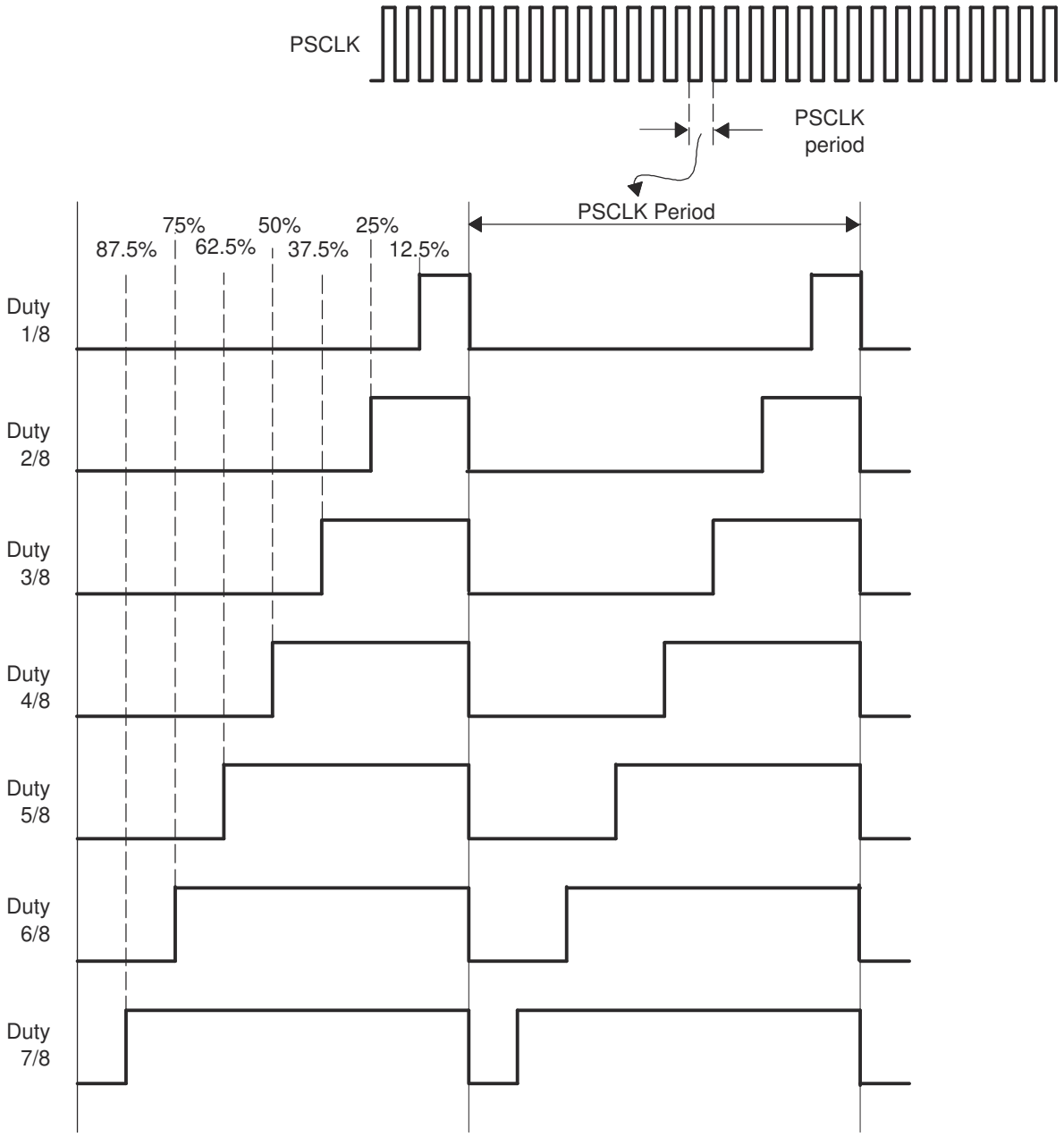


Figure 7-202. PWM Chopper Submodule Waveforms Showing the Pulse Width (Duty Cycle) Control of Sustaining Pulses

7.5.6.10 Trip-Zone (TZ) Submodule

Each ePWM module is connected to six $\overline{TZn}$ signals ($\overline{TZ1}$ to $\overline{TZ6}$). $\overline{TZ1}$ to $\overline{TZ4}$ are sourced from the GPIO mux. $\overline{TZ5}$ is connected to the system error fail logic, and $\overline{TZ6}$ is sourced from the EMUSTOP output from the CPU. These signals indicate external fault or trip conditions, and the ePWM outputs can be programmed to respond accordingly when faults occur.

Figure 7-203 illustrates the trip-zone submodule within the ePWM.

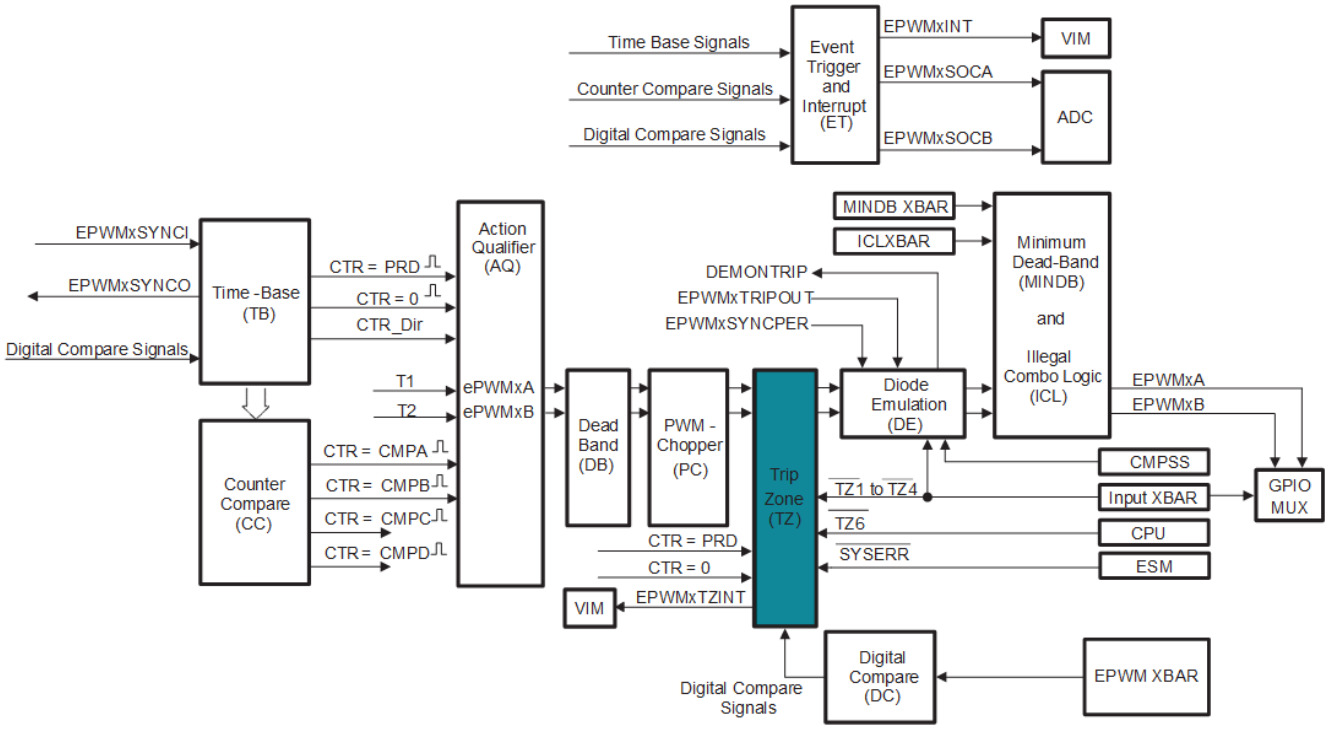


Figure 7-203. Trip-Zone Submodule

7.5.6.10.1 Purpose of the Trip-Zone Submodule

The key functions of the trip-zone submodule are:

- Trip inputs $\overline{TZ1}$ to $\overline{TZ6}$ 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.
- Support for digital compare tripping (DC) based on state of on-chip analog comparator module outputs and $\overline{TZ1}$ to $\overline{TZ3}$ signals.
- Each trip-zone input and digital compare (DC) submodule DCAEVT1/2 or DCBEVT1/2 force event can be allocated to either one-shot or cycle-by-cycle operation.
- Interrupt generation is possible on any trip-zone input.
- Software-forced tripping is also supported.
- The trip-zone submodule can be fully bypassed if the submodule is not required.

7.5.6.10.2 Operational Highlights for the Trip-Zone Submodule

The following sections describe the operational highlights and configuration options for the trip-zone submodule.

The trip-zone signals $\overline{TZ1}$ to $\overline{TZ6}$ (also collectively referred to as $\overline{TZn}$) are active-low input signals. When one of these signals goes low, or when a DCAEVT1/2 or DCBEVT1/2 force happens based on the TZDCSEL register event selection, the indication is that a trip event has occurred. Each ePWM module can be individually configured to ignore or use each of the trip-zone signals or DC events. Which trip-zone signals or DC events are used by a particular ePWM module is determined by the TZSEL register for that specific ePWM module. The trip-zone signals can or cannot be synchronized to the ePWMclock (EPWMCLK) and digitally filtered within the GPIO MUX block. A minimum of 3*TBCLK low pulse width on $\overline{TZn}$ inputs is sufficient to trigger a fault condition on the ePWM module. If the pulse width is less than this, the trip condition cannot be latched by CBC or OST latches. 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 $\overline{TZn}$ inputs. The GPIOs or peripherals must be appropriately configured.

Each $\overline{TZn}$ input can be individually configured to provide either a cycle-by-cycle or one-shot trip event for an ePWM module. DCAEVT1 and DCBEVT1 events can be configured to directly trip an ePWM module or provide a one-shot trip event to the module. Likewise, DCAEVT2 and DCBEVT2 events can also be configured to directly trip an ePWM module or provide a cycle-by-cycle trip event to the module. This configuration is determined by the TZSEL[DCAEVT1/2], TZSEL[DCBEVT1/2], TZSEL[CBCn], and TZSEL[OSHTn] control bits (where n corresponds to the trip input), respectively.

- **Cycle-by-Cycle (CBC):**

  When a cycle-by-cycle trip event occurs, the action specified in the TZCTL[TZA] and TZCTL[TZB] bits is carried out immediately on the EPWMxA and EPWMxB outputs. Table 7-131 lists some of the possible actions. Independent actions can be specified based on the occurrence of the event while the counter is counting up or while the counter is counting down by appropriately configuring bits in the TZCTL2 register. Actions specified in the TZCTL2 register take effect only when the ETZE bit in TZCTL2 is set.

  Additionally, when a cycle-by-cycle trip event occurs, the cycle-by-cycle trip event flag (TZFLG[CBC]) is set and a EPWMx_TZINT interrupt is generated when enabled in the TZEINT register and interrupt controller. A corresponding flag for the event that caused the CBC event is also set in register TZCBCFLG.

  If the CBC interrupt is enabled using the TZEINT register and DCAEVT2 or DCBEVT2 are selected as CBC trip sources using the TZSEL register, it is not necessary to also enable the DCAEVT2 or DCBEVT2 interrupts in the TZEINT register, as the DC events trigger interrupts through the CBC mechanism.

  The specified condition on the inputs is automatically cleared based on the selection made with TZCLR[CBCPULSE] if the trip event is no longer present. Therefore, in this mode, the trip event is cleared or reset every PWM cycle. The TZFLG[CBC] and TZCBCFLG flag bits remain set until the flag bits are manually cleared by writing to the TZCLR[CBC] and TZCBCCLR flag bits. If the cycle-by-cycle trip event is still present when the TZFLG[CBC] and TZCBCFLG register bits are cleared, then these bits are again immediately set.

- **One-Shot (OSHT):**

  When a one-shot trip event occurs, the action specified in the TZCTL[TZA] and TZCTL[TZB] bits is carried out immediately on the EPWMxA and EPWMxB output. Table 7-131 lists some of the possible actions. Independent actions can be specified based on the occurrence of the event while the counter is counting up and while the counter is counting down by appropriately configuring bits in TZCTL2 register. Actions specified in TZCTL2 register take effect only when ETZE bit in TZCTL2 is set.

  Additionally, when a one-shot trip event occurs, the one-shot trip event flag (TZFLG[OST]) is set and a EPWMx_TZINT interrupt is generated when enabled in the TZEINT register and interrupt controller. A corresponding flag for the event that caused the OST event is also set in register TZOSTFLG. The one-shot trip condition must be cleared manually by writing to the TZCLR[OST] bit. If desired, the TZOSTFLG register bit can be cleared by manually writing to the corresponding bit in the TZOSTCLR register.

If the one-shot interrupt is enabled using the TZEINT register and DCAEVT1 or DCBEVT1 are selected as OSHT trip sources using the TZSEL register, it is not necessary to also enable the DCAEVT1 or DCBEVT1 interrupts in the TZEINT register, as the DC events trigger interrupts through the OSHT mechanism.

Note

Clear the TZFLG and TZOSTFLG flags after making sure that the TRIPIN source of the OST has become inactive. Otherwise, if interrupts are enabled, depending on when the flags are cleared, an OST interrupt can occur and the OST flags are zero.

- **Digital Compare Events (DCAEVT1/2 and DCBEVT1/2):**

  A digital compare DCAEVT1/2 or DCBEVT1/2 event is generated based on a combination of the DCAH/DCAL and DCBH/DCBL signals as selected by the TZDCSEL register. The signals which source the DCAH/DCAL and DCBH/DCBL signals are selected using the DCTRIPSEL register and can be either trip zone input pins or analog comparator CMPSSx signals. For more information on the digital compare submodule signals, see Section 7.5.6.13.

  When a digital compare event occurs, the action specified in the TZCTL[DCAEVT1/2] and TZCTL[DCBEVT1/2] bits is carried out immediately on the EPWMxA and EPWMxB output. Table 7-131 lists the possible actions. Independent actions can be specified based on the occurrence of the event while the counter is counting up and while the counter is counting down by appropriately configuring bits in TZCTLDCA and TZCTLDCB register. Actions specified in TZCTLDCA and TZCTLDCB registers take effect only when ETZE bit in TZCTL2 is set.

  In addition, the relevant DC trip event flag (TZFLG[DCAEVT1/2] / TZFLG[DCBEVT1/2]) is set and a EPWMx_TZINT interrupt is generated when enabled in the TZEINT register and interrupt controller.

  The specified condition on the pins is automatically cleared when the DC trip event is no longer present. The TZFLG[DCAEVT1/2] or TZFLG[DCBEVT1/2] flag bit remains set until the flag is manually cleared by writing to the TZCLR[DCAEVT1/2] or TZCLR[DCBEVT1/2] bit. If the DC trip event is still present when the TZFLG[DCAEVT1/2] or TZFLG[DCBEVT1/2] flag is cleared, then the flag is again immediately set.

- **Edge detection within a programmable TBCTR range (CAPEVT):**

  An edge detection within a programmable TBCTR range is added in type 5 ePWM. When a CAPIN edge does not occur within a specified range of TBCTR values, the CAPEVT signal is generated. The TBCTR range during which a CAPIN edge must occur is determined by XMINMAX_ACTIVE register. A gating signal CAPGATE can also be used to gate the CAPIN edge. For more information on the CAPEVT signal, see Section 7.5.6.13.4.4.

  In addition, the EPWMx_TZINT interrupt is generated when enabled in the TZEINT register and interrupt controller.

  The TZFLG[CAPEVT] flag bit remains set until the flag is manually cleared by writing to the TZCLR[CAPEVT] bit. If the CAPEVT event is still present when the TZFLG[CAPEVT] flag is cleared, then the flag is again immediately set.

The action taken when a trip event occurs can be configured individually for each of the ePWM output pins by way of the TZCTL, TZCTL2, TZCTLDCA, and TZCTLDCB register bit fields. Some of the possible actions, shown in Table 7-131, can be taken on a trip event.

The trip signal generated by the ePWM module can be selected through the TZTRIPOUTSEL register. This register has an ORed version of all the enabled trip signals. The TRIPOUT signal is routed to eCAP Trip Mux,PWM-XBAR and Output-XBAR.

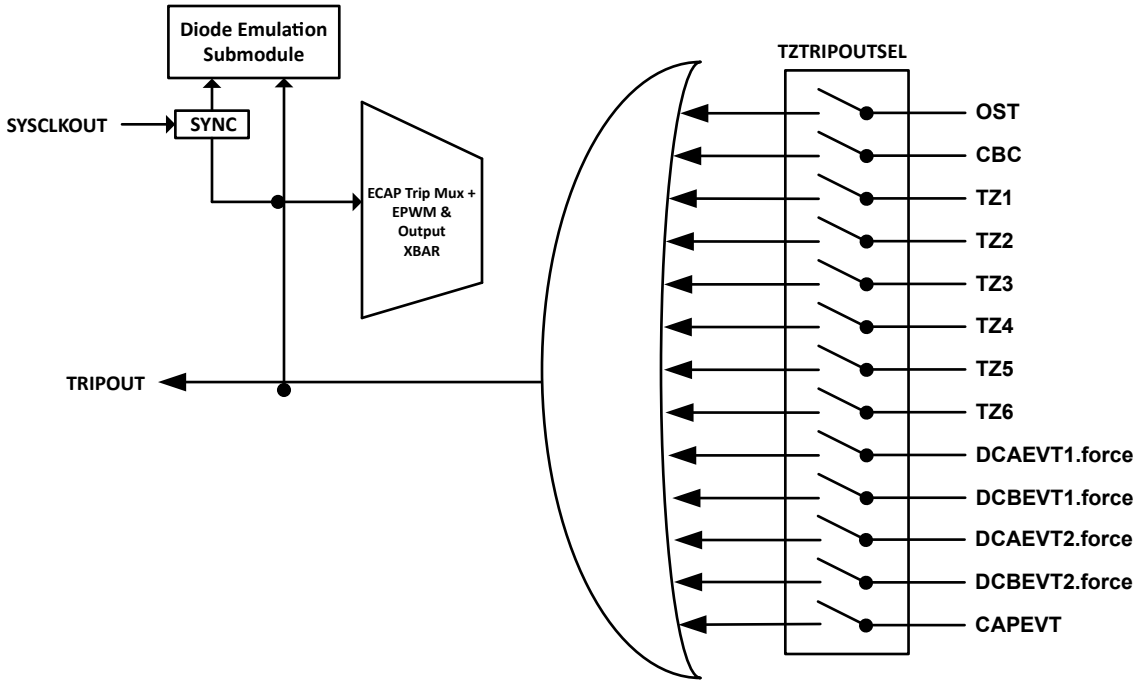


Figure 7-204. Trip-Zone TRIPOUT Selection

Table 7-131. Possible Actions On a Trip Event

TZCTL Register Bitfield Settings	EPWMxA and EPWMxB	Comment
0,0	High-Impedance	Tripped
0,1	Force to High State	Tripped
1,0	Force to Low State	Tripped
1,1	No Change	Do Nothing. No change is made to the output.

Example 7-1. Trip-Zone Configurations

Scenario A:
A one-shot trip event on $\overline{TZ1}$ pulls both EPWM1A, EPWM1B low and also forces EPWM2A and EPWM2B high.

- Configure the ePWM1 registers as follows:
  - TZSEL[OSHT1] = 1: enables $\overline{TZ1}$ as a one-shot event source for ePWM1
  - TZCTL[TZA] = 2: EPWM1A is forced low on a trip event.
  - TZCTL[TZB] = 2: EPWM1B is forced low on a trip event.
- Configure the ePWM2 registers as follows:
  - TZSEL[OSHT1] = 1: enables $\overline{TZ1}$ as a one-shot event source for ePWM2
  - TZCTL[TZA] = 1: EPWM2A is forced high on a trip event.
  - TZCTL[TZB] = 1: EPWM2B is forced high on a trip event.

Scenario B:
A cycle-by-cycle event on $\overline{TZ5}$ pulls both EPWM1A, EPWM1B low.
A one-shot event on $\overline{TZ1}$ or $\overline{TZ6}$ puts EPWM2A into a high impedance state.

- Configure the ePWM1 registers as follows:
  - TZSEL[CBC5] = 1: enables $\overline{TZ5}$ as a cycle-by-cycle event source for ePWM1
  - TZCTL[TZA] = 2: EPWM1A is forced low on a trip event.
  - TZCTL[TZB] = 2: EPWM1B is forced low on a trip event.
- Configure the ePWM2 registers as follows:
  - TZSEL[OSHT1] = 1: enables $\overline{TZ1}$ as a one-shot event source for ePWM2
  - TZSEL[OSHT6] = 1: enables $\overline{TZ6}$ as a one-shot event source for ePWM2
  - TZCTL[TZA] = 0: EPWM2A is put into a high-impedance state on a trip event.
  - TZCTL[TZB] = 3: EPWM2B ignores the trip event.

Note

When configuring the GPIOs and INPUT X-BAR/EPWM X-BAR options, be aware that a change in the X-BAR input selections can cause an unwanted event. Therefore, set up the GPIO and X-BAR input configurations before enabling the ePWM Trip-Zone. If a requirement is to change the GPIO/X-BAR configurations while the ePWM Trip-Zone is enabled, the user can turn off the TRIPs by clearing the TZSEL register and reconfiguring the TRIP selection (TZSEL) after the INPUT XBAR selection is changed.

7.5.6.10.3 Generating Trip Event Interrupts

Figure 7-205 and Figure 7-206 illustrate the trip-zone submodule control and interrupt logic, respectively. DCAEVT1/2 and DCBEVT1/2 signals are described in further detail in Digital Compare (DC) Submodule.

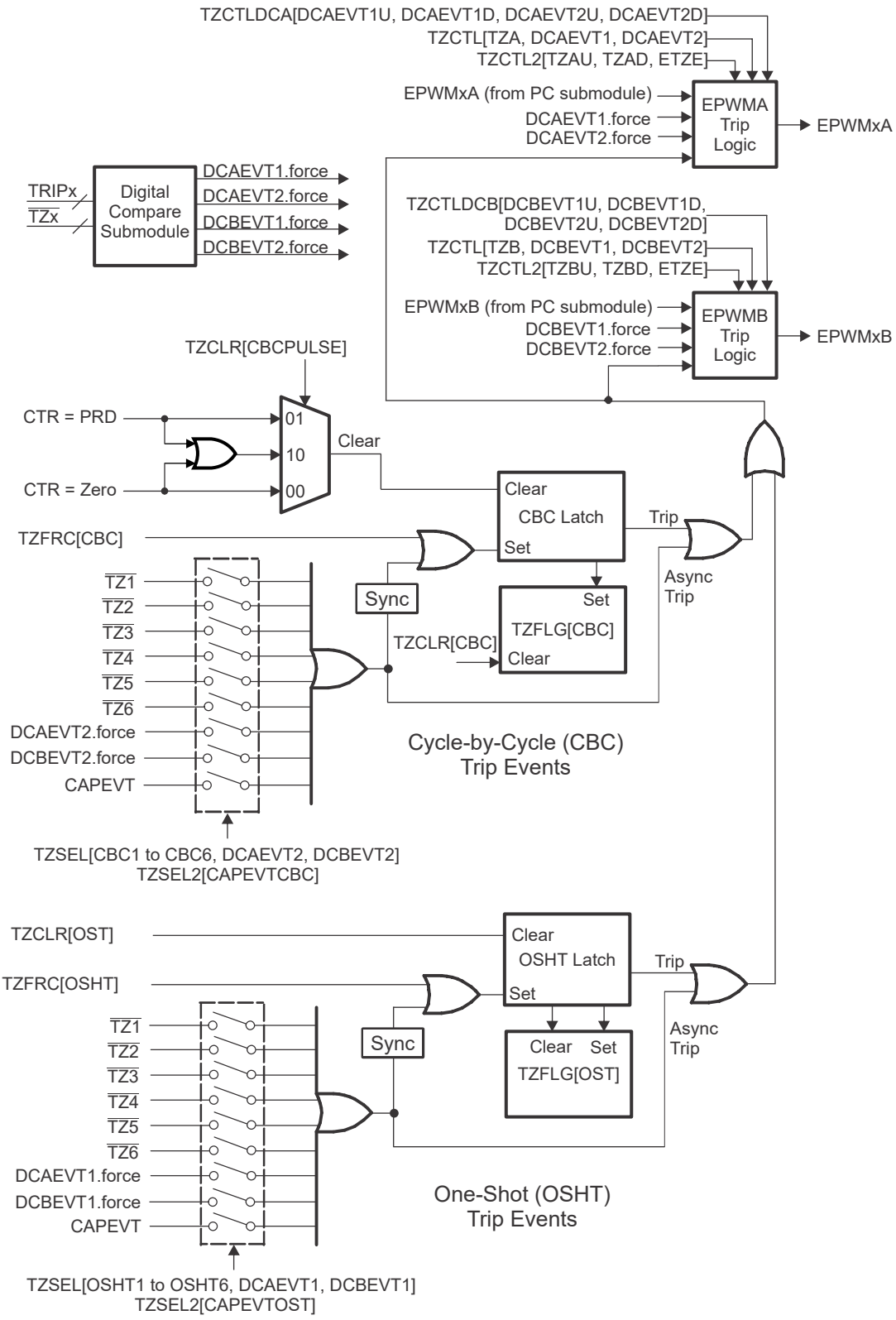


Figure 7-205. Trip-Zone Submodule Mode Control Logic

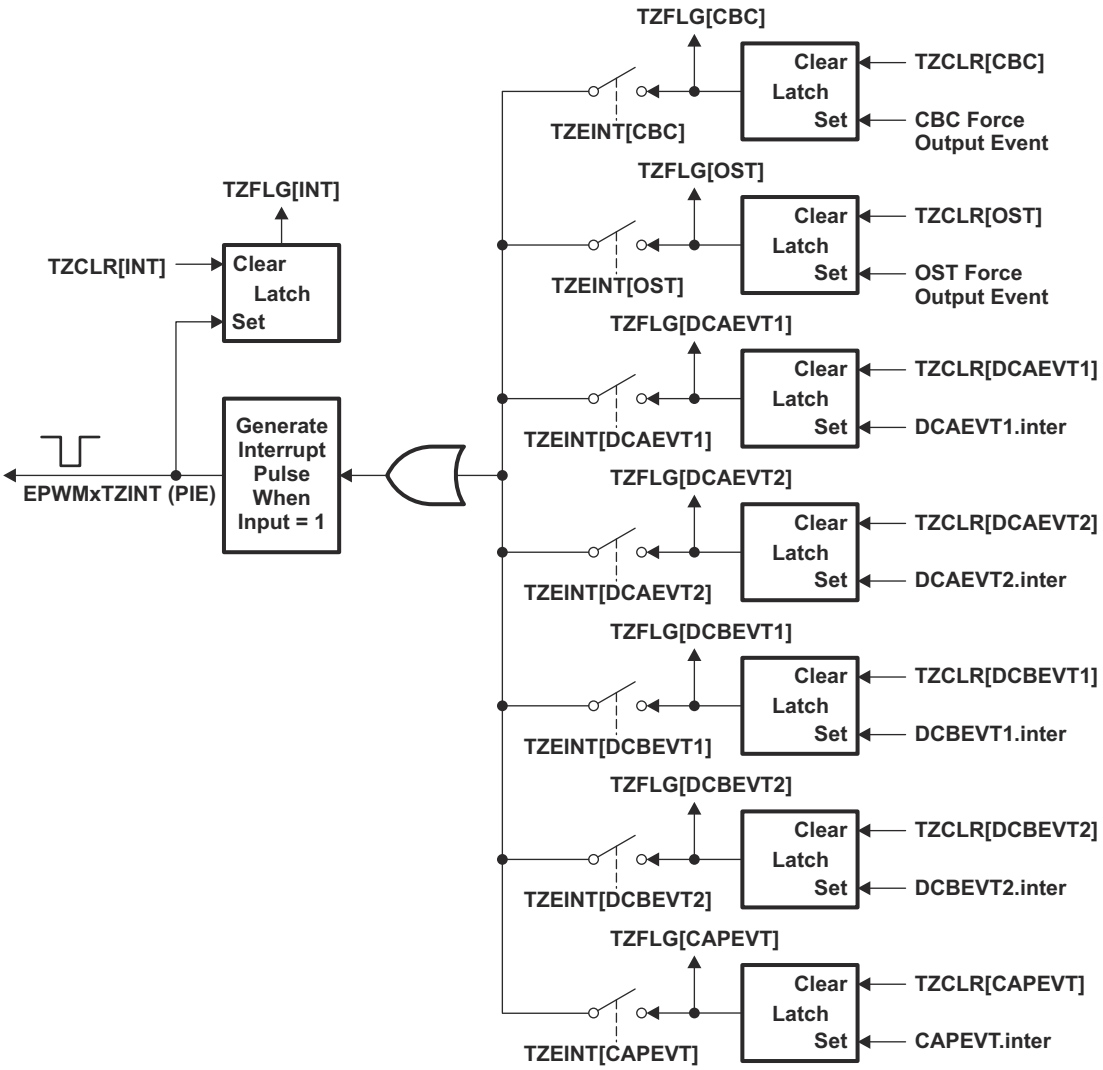


Figure 7-206. Trip-Zone Submodule Interrupt Logic

The signal CAPEVT is generated from the Capture Control Logic and is available in type 5 EPWM.

These individual flags for the CBC, OST and DCxEVTy can be used to detect the source of the EPWMxTZINT Interrupt. When multiple sources are used to generate the EPWMxTZINT interrupt, reading and clearing the flags takes different actions based on the specific event.

7.5.6.11 Diode Emulation (DE) Submodule

The purpose of the Diode Emulation logic is to provide hardware features and the necessary hooks into other IPs to implement robust diode mode sense and control in a noisy environment.

Diode Emulation features include:

- Ability to choose any of the comparator outputs as trips to detect entry into DE mode.
- Ability to switch the comparator thresholds, dynamically in hardware upon DE mode entry.
- Two modes of clearing/de-evaluating the DE condition:
  - Software clear
  - Cycle-by-cycle clear on PWMSYNC event
- Configurable source selects of ePWM in DE mode.
- Ability to monitor the DE mode duration and generate a trip event to ePWMs.

Figure 7-207 illustrates the diode emulation submodule within the ePWM.

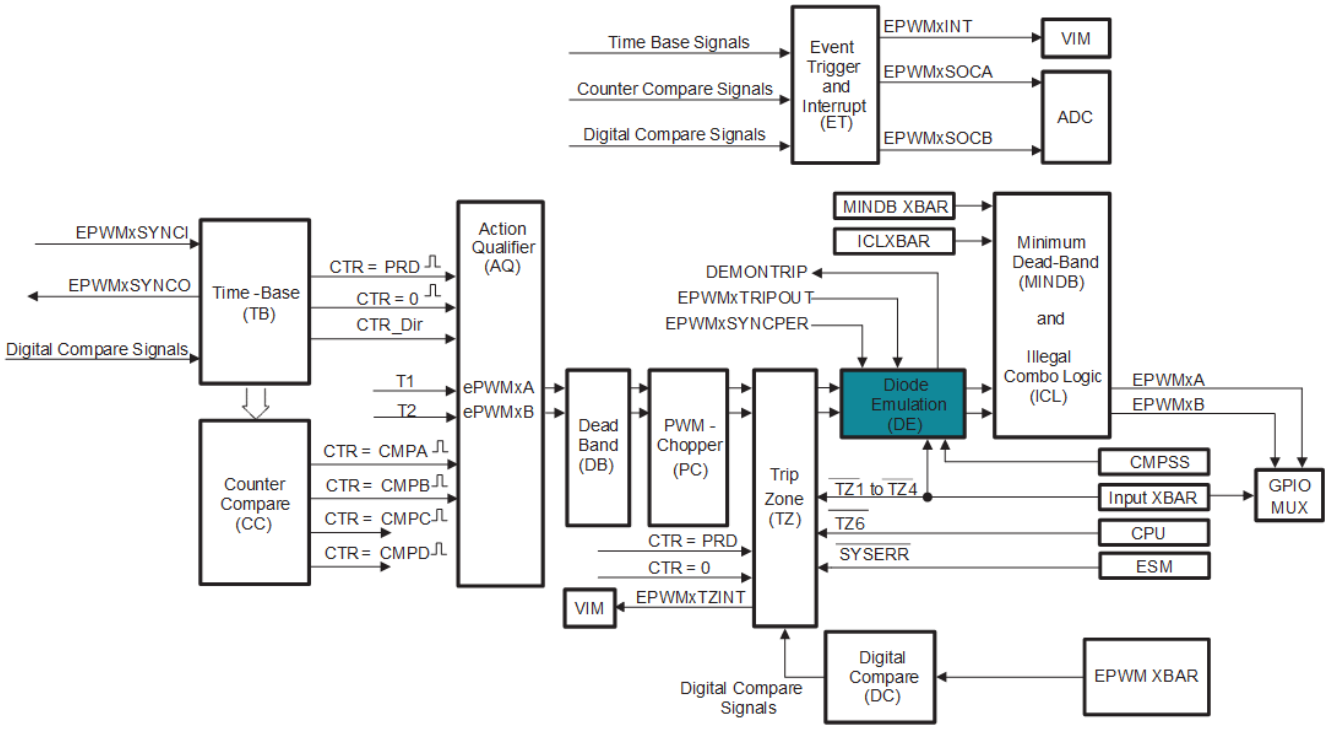


Figure 7-207. Diode Emulation Submodule

Figure 7-208 shows the interfaces to the DE block. As can be seen from the diagram, DE function is associated with an instance of ePWMx. The EPWMxA and EPWMxB signals from a given instance of ePWM module pass through the associated DE block and minimum dead band logic. In addition to EPWMxA and EPWMxB, two signals, EPWMxA_DB_NO_HR and EPWMxB_DB_NO_HR are tapped from the ePWM modules. These two signals are PWM signals that are tapped before the signals pass through the high-resolution delay lines and come from the dead-band submodule outputs. If high-resolution is not used, then EPWMxA_DB_NO_HR and EPWMxB_DB_NO_HR are just the outputs of the dead-band submodule.

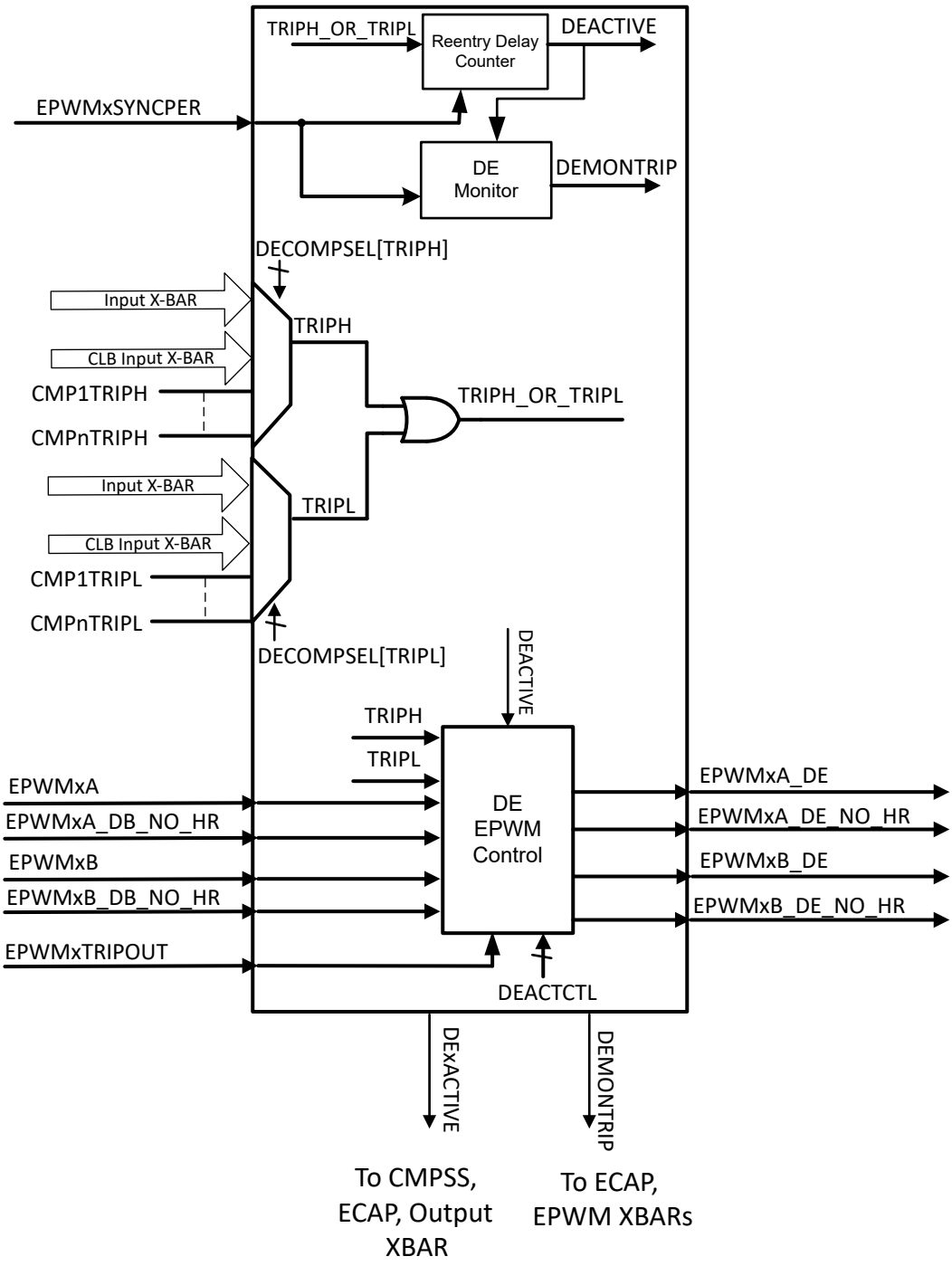


Figure 7-208. Diode Emulation Block Diagram

The DE block can be configured to select one of the comparators or one of the outputs of the Input XBAR, as source of trip signals (TRIPH, TRIPL). The selected comparator is responsible for monitoring the current in the external power converter. Comparator thresholds (High and Low threshold) can be set such that, any breach of these thresholds indicates a need to switch to the DE mode.

Once DE mode is entered, indicated by setting of DEACTIVE flag, the ePWMs sent out of DE block are controlled by configuration registers in the DE block, and are not be the same as ePWMA/B from the associated ePWM instance. Once the DEACTIVE flag is set, the threshold settings of the selected CMPSS are switched to a new set of values (a narrower region). DEACTIVE flag from all of the ePWM instances are hooked up to all the comparator sub-systems (CMPSSy) to enable threshold value switch on DE mode entry. Refer to the CMPSS chapter for more details on how the new threshold values are set.

7.5.6.11.1 DEACTIVE Mode

DE mode is entered when TRIPH_OR_TRIPL signal from the selected comparator (CMPSS) goes high. Once the diode emulation mode is entered, typically TRIPH or TRIPL are set and cleared in a sequence (at a given instance of time, TRIPH is high or TRIPL is high – never both) until the current settles within the threshold band.

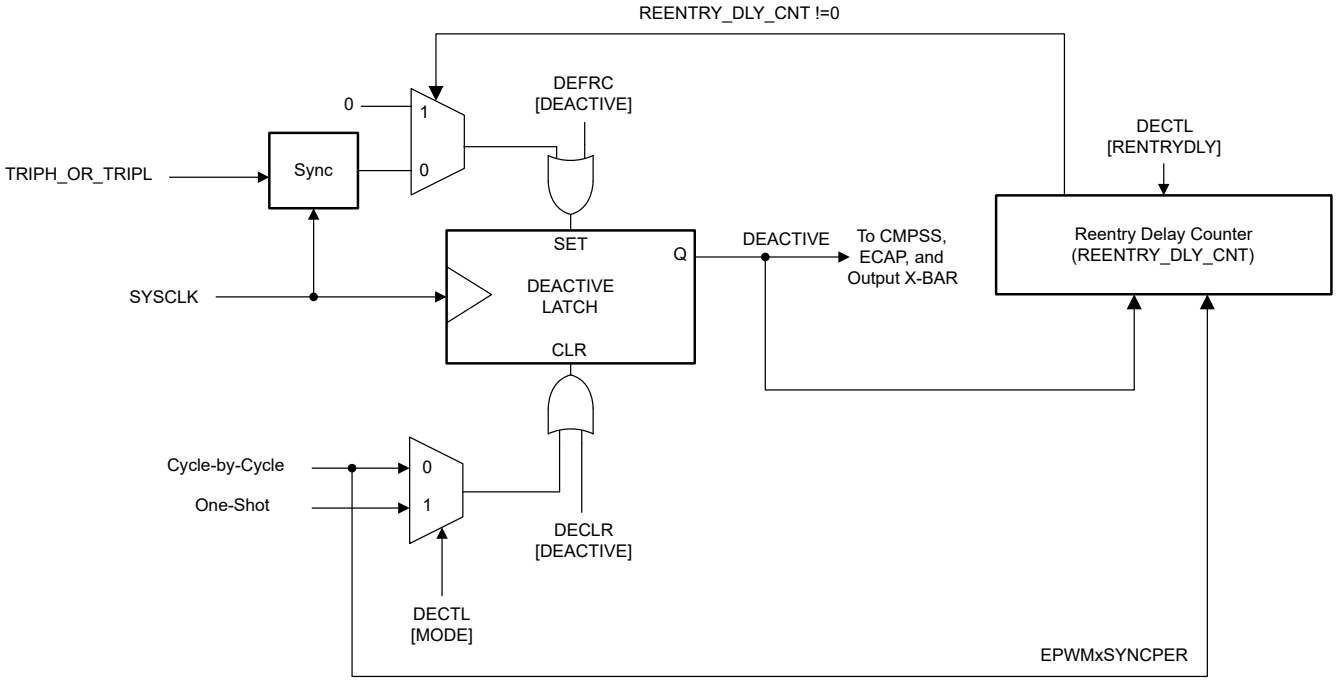


Figure 7-209. DEACTIVE Flag Functionality

Once the DEACTIVE flag is set, the thresholds on CMPSS are changed to an alternate set of thresholds, and also ePWMA/B out of DE function are being controlled by the DEACTCTL register settings. Figure 7-210 demonstrates an example timing diagram illustrating entry into DE mode.

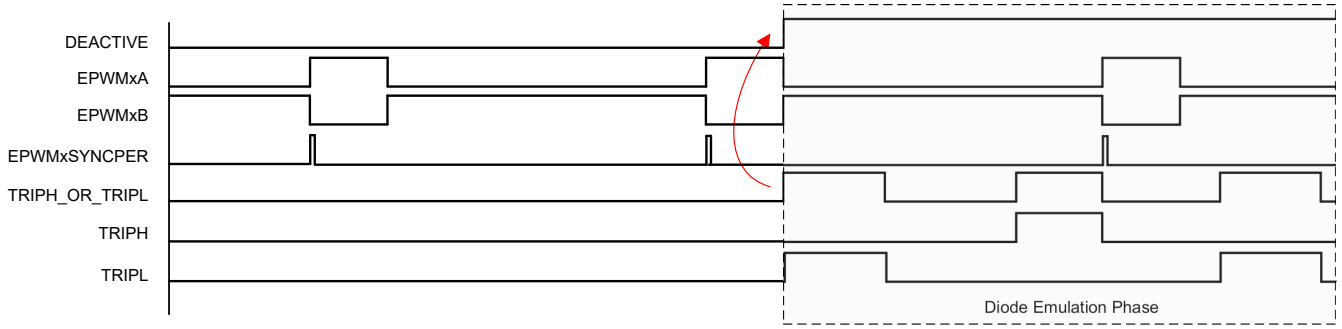


Figure 7-210. Example Timing Sequence Illustrating DE Mode Entry

7.5.6.11.2 Exiting DE Mode

DE mode can be exited in two ways, based on the DECTL[MODE] setting:

- Software clear of DEACTIVE flag, DECLR[CLR]
- Cycle-by-cycle clear mode, in which TRIPH_OR_TRIPL is evaluated on every EPWMxSYNCPER and if the trip condition is not present, then DEACTIVE flag is cleared. Figure 7-211 illustrates the clearing of the DEACTIVE flag based on EPWMxSYNCPER.

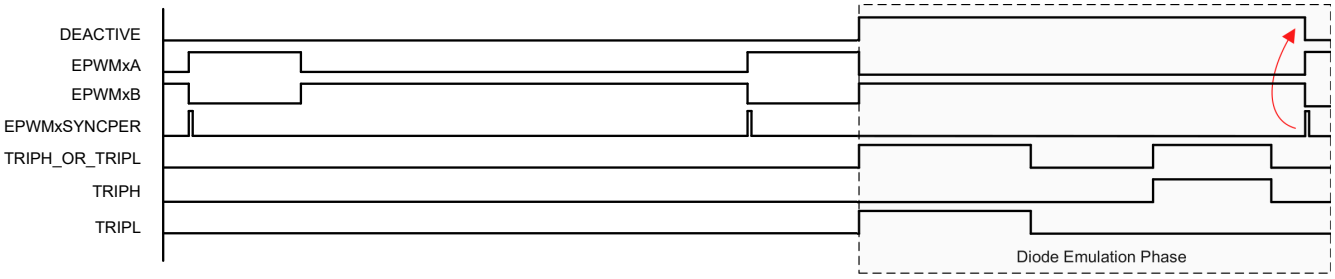


Figure 7-211. Cycle-by-Cycle Mode

7.5.6.11.3 Re-Entering DE Mode

Once DE mode is exited, DE mode can be delayed for a certain duration until reentry. This is accomplished by configuring the DECTL[REENTRYDLY] field. REENTRYDLY determines the window in which TRIP signals are prevented from setting the DEACTIVE flag. On a falling edge of DEACTIVE, an internal counter is loaded with the DECTL[REENTRYDLY] value. The counter is decremented on every EPWMxSYNCPER as long as the count value is greater than 0. While the count value is greater than 0, TRIP signals are blocked and DEACTIVE flag is not set even if TRIP events are active.

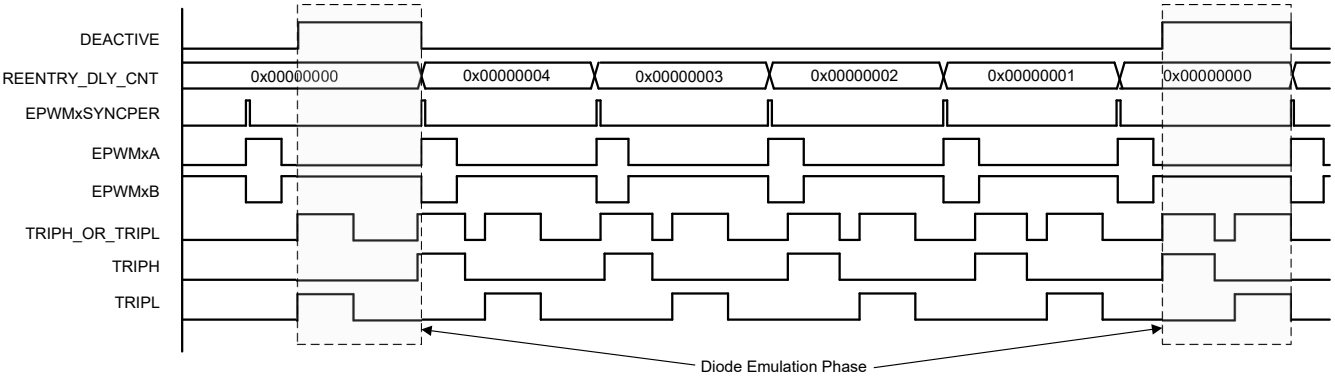


Figure 7-212. DE Mode Reentry Sequence

Figure 7-213 illustrates the circuit driving the EPWMxA/B signals from the DE block. As can be observed, when DEACTIVE flag is not set, EPWMxA_DE, EPWMxB_DE, EPWMxA_DE_NO_HR, and EPWMxB_DE_NO_HR are driven by EPWMA/B and EPWMA/B_DB_NO_HR respectively. When DEACTIVE flag is set, EPWMA/B_DE are be driven by TRIPH, TRIPL, constant 0, or a constant 1 signal based on the configuration of the DEATCTL[PWMA], DEATCTL[PWMB], DEATCTL[TRIPSELA], DEATCTL[TRIPSELB] fields. When a PWMTRIP signal from the associated ePWM trips, EPWMA/B_DE are be driven by the input PWM signals configured through the DECTL[TRIPENABLE] field.

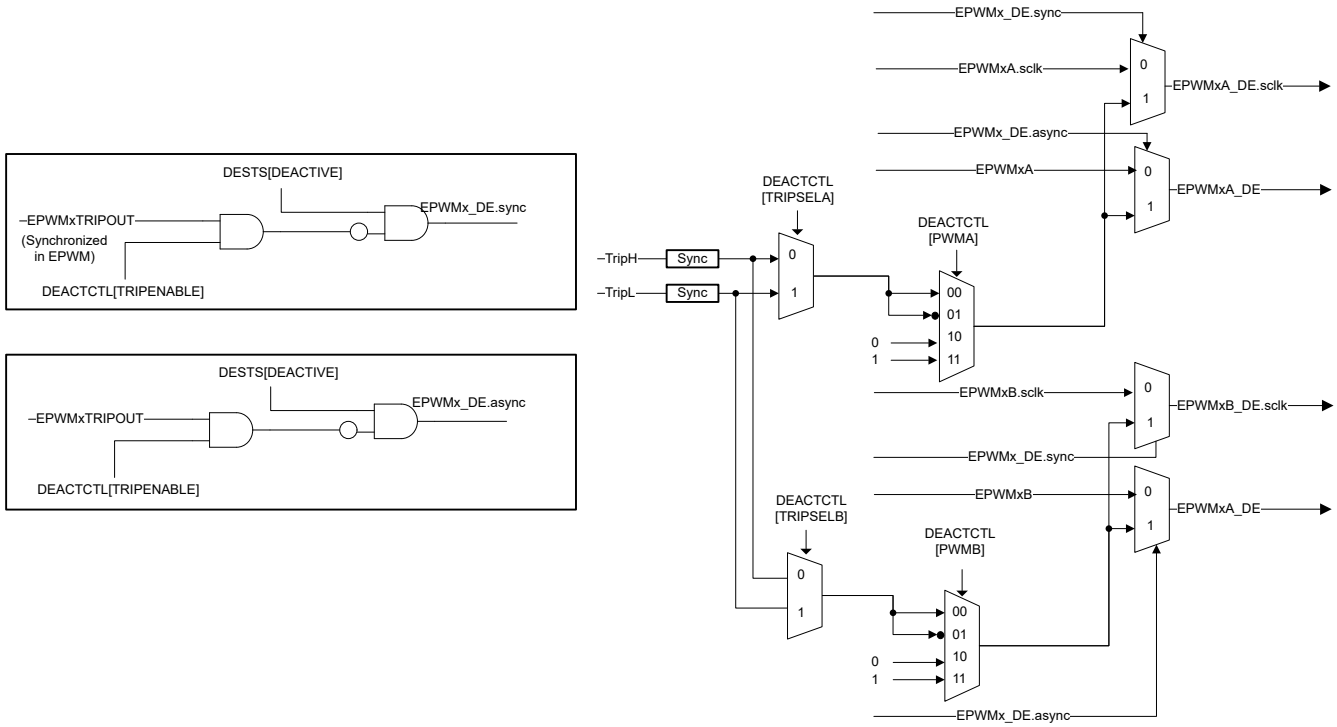


Figure 7-213. Diode Emulation Circuit

Figure 7-214 shows an example waveform, in which DEACTCTL[PWMA] is configured to select TripL as the source, DEACTCTL[PWMB] is configured to select TripH as the source and DEACTCTL[PWMAPOL] and DEACTCTL[PWMBPOL] are both 0.

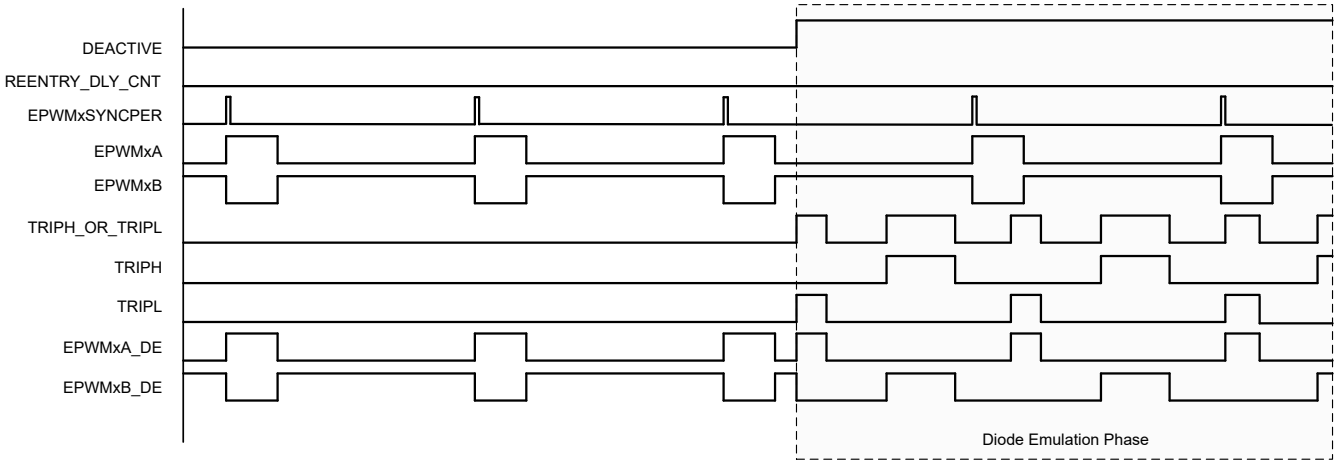


Figure 7-214. Diode Emulation Mode Timing Diagram

7.5.6.11.4 DE Monitor

To detect extended DE phase, which is beyond the expected duration, a DE mode monitor counter, DEMONCNT, is provided. This 16-bit counter monitors the frequency of diode mode trip events. The counter if enabled, DEMONCTL[ENABLE], increments on a PWMSYNC event, in steps of DEMONSTEP[INCSTEP] when TRIPH_OR_TRIPL is high, and decrements on a PWMSYNC event, in steps of DEMONSTEP[DECSTEP] when TRIPH_OR_TRIPL is low. If counter exceeds DEMONTHRES[THRESHOLD], then a DEMONTRIP pulse is generated and the counter is cleared. The counter value is saturated to 0 during an underflow and 0xffff on an overflow. The counter is cleared when DECTL[ENABLE] is cleared.

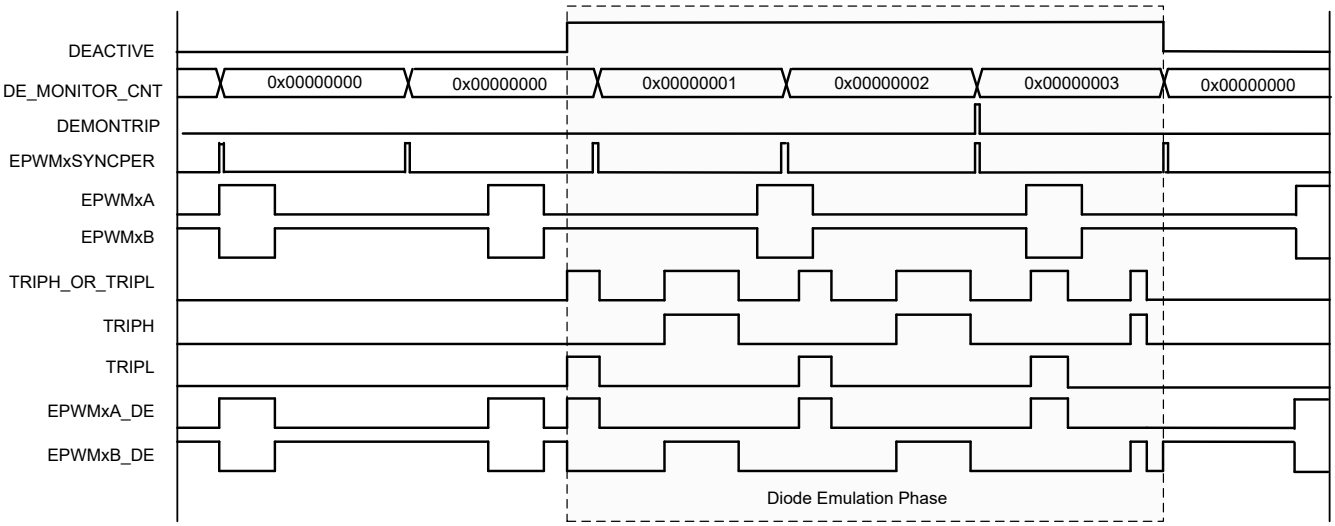


Figure 7-215. DE Mode Monitor Sequence

7.5.6.12 Event-Trigger (ET) Submodule

The key functions of the event-trigger submodule are:

- Receives event inputs generated by the time-base, counter-compare, and digital-compare submodules
- Uses the time-base direction information for up/down event qualification
- Uses prescaling logic to issue interrupt requests and ADC start of conversion at:
  - Every event
  - Every second event
  - Up to every fifteenth event
- Provides full visibility of event generation using event counters and flags
- Allows software forcing of Interrupts and ADC start of conversion

The event-trigger submodule manages the events generated by the time-base submodule, the counter-compare submodule, and the digital-compare submodule to generate an interrupt to the CPU and a start of conversion pulse to the ADC when a selected event occurs.

Figure 7-216 illustrates the event-trigger submodule within the ePWM.

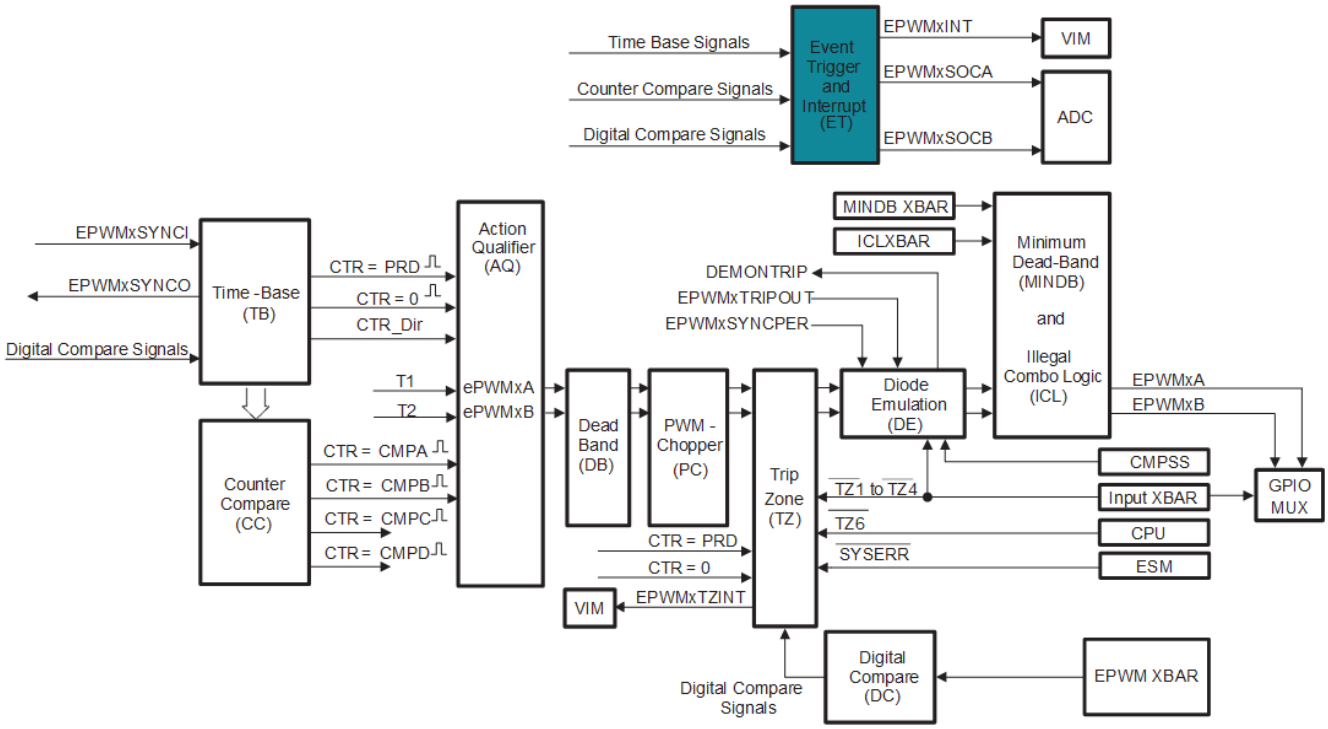


Figure 7-216. Event-Trigger Submodule

7.5.6.12.1 Operational Overview of the ePWM Event-Trigger Submodule

The event-trigger submodule monitors various event conditions (shown as inputs on the left side of Figure 7-217 and can be configured to prescale these events before issuing an Interrupt request or an ADC start of conversion. The event-trigger prescaling logic can issue Interrupt requests and ADC start of conversion at:

- Every event
- Every second event
- Up to every fifteenth event

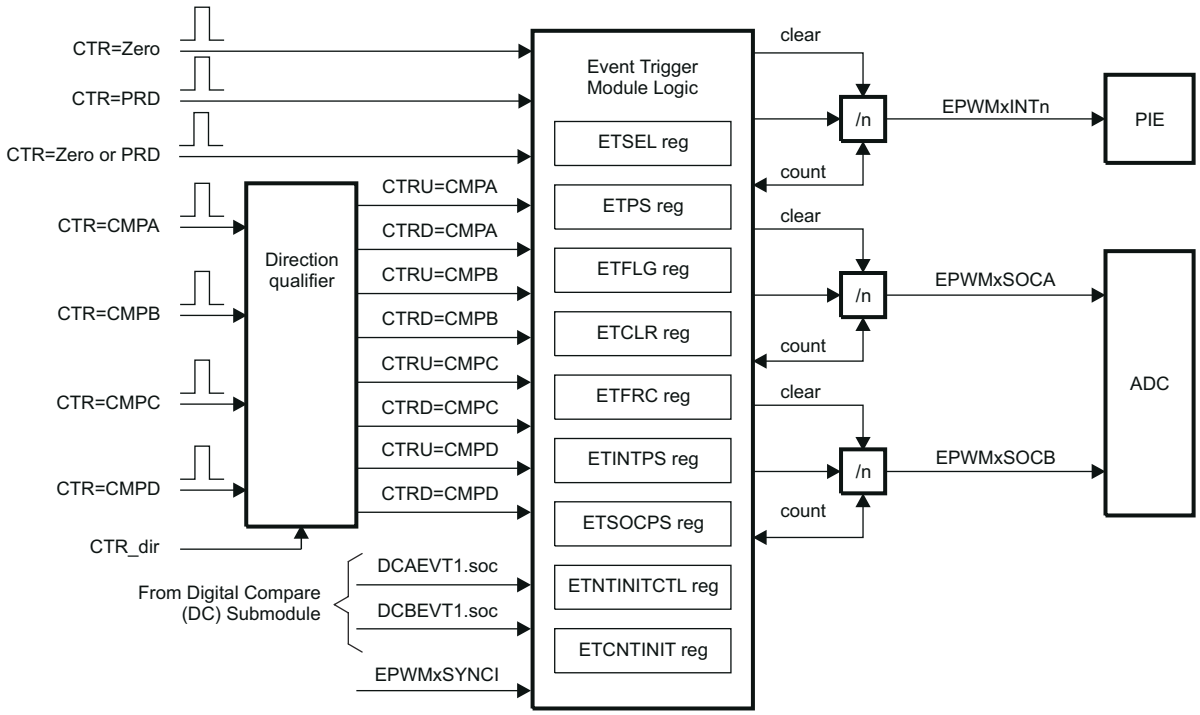


Figure 7-217. Event-Trigger Submodule Showing Event Inputs and Prescaled Outputs

- ETSEL - This selects which of the possible events trigger an interrupt or start an ADC conversion.
- ETPS - This programs the event prescaling options mentioned above.
- ETFLG - These are flag bits indicating status of the selected and prescaled events.
- ETCLR - These bits allow clearing the flag bits in the ETFLG register using software.
- ETFRC - These bits allow software forcing of an event. Useful for debugging or software intervention.
- ETINTPS - This programs the interrupt event prescaling options, supporting count and period up to 15 events.
- ETSOCPS - This programs the SOC event prescaling options, supporting count and period up to 15 events.
- ETCNTINITCTL - These bits enable ETCNTINIT initialization using SYNC event or using software force.
- ETCNTINIT - These bits allow initializing INT/SOCA/SOCB counters on SYNC events (or software force) with user programmed value.

A more detailed look at how the various register bits interact with the Interrupt and ADC start of conversion logic are shown in Figure 7-218, Figure 7-219, and Figure 7-220.

Figure 7-218 shows the event-trigger's interrupt generation logic. The interrupt-period (ETPS[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.

The selection made on ETPS[INTPSSEL] bit determines whether ETPS [INTCNT, and INTPRD] registers or ETINTPS [INTCNT2, and INTPRD2] registers bit fields determine frequency of events (interrupt once every 0-15 events).

The event that can cause an interrupt is configured by the interrupt selection (ETSEL[INTSEL]) and (ETSEL[INTSELCMP]) bits. The event can be one of the following:

- Time-base counter equal to zero (TBCTR = 0x00).
- Time-base counter equal to period (TBCTR = TBPRD).
- Time-base counter equal to zero or period (TBCTR = 0x00 || TBCTR = TBPRD).
- Time-base counter equal to the compare A register (CMPA) when the timer is incrementing.
- Time-base counter equal to the compare A register (CMPA) when the timer is decrementing.
- Time-base counter equal to the compare B register (CMPB) when the timer is incrementing.
- Time-base counter equal to the compare B register (CMPB) when the timer is decrementing.
- Time-base counter equal to the compare C register (CMPC) when the timer is incrementing.
- Time-base counter equal to the compare C register (CMPC) when the timer is decrementing.
- Time-base counter equal to the compare D register (CMPD) when the timer is incrementing.
- Time-base counter equal to the compare D register (CMPD) when the timer is decrementing.

The number of events that have occurred can be read from the interrupt event counter ETPS[INTCNT] or ETINTPS[INTCNT2] register bits based off of the selection made using ETPS[INTPSSEL]. That is, when the specified event occurs the ETPS[INTCNT] or ETINTPS[INTCNT2] bits are incremented until the bits reach the value specified by ETPS[INTPRD] or ETINTPS[INTPRD2] determined again by the selection made in ETPS[INTPSSEL]. When ETPS[INTCNT] = ETPS[INTPRD], the counter stops counting and the counter output is set. The counter is only cleared when an interrupt is sent to the interrupt controller.

When ETPS[INTCNT] reaches ETPS[INTPRD], the following behavior occurs. The following behavior is also applicable to ETINTPS[INTCNT2] and ETINTPS[INTPRD2]:

- If interrupts are enabled, ETSEL[INTEN] = 1 and the interrupt flag is clear, ETFLG[INT] = 0, then an interrupt pulse is generated and the interrupt flag is set, ETFLG[INT] = 1, and the event counter is cleared ETPS[INTCNT] = 0. The counter begins counting events again.
- If interrupts are disabled, ETSEL[INTEN] = 0, or the interrupt flag is set, ETFLG[INT] = 1, the counter stops counting events when the counter reaches the period value ETPS[INTCNT] = ETPS[INTPRD].
- If interrupts are enabled, but the interrupt flag is already set, then the counter holds the output high until the ENTFLG[INT] flag is cleared. This allows for one interrupt to be pending while one is serviced.

Writing a 0 to the INTPRD bits automatically clears the counter (INTCNT = 0) and the counter output resets (so no interrupts are generated). For all other writes to INTPRD, INTCNT retains the previous value. INTCNT resets when INTCNT overflows. Writing a 1 to the ETFRC[INT] bit increments the event counter INTCNT. The counter behaves as previously described when INTCNT = INTPRD. When INTPRD = 0, the counter is disabled and hence no events are detected and the ETFRC[INT] bit is also ignored. The same applies to ETINTPS[INTCNT2] and ETINTPS[INTPRD2].

The previous definition means that an interrupt on every event, on every second event, or on every third event if using the INTCNT and INTPRD can be generated. An interrupt on every event up to 15 events if using the INTCNT2 and INTPRD2 can be generated.

The INTCNT2 value can be initialized with the value from ETCNTINIT[INTINIT] based on the selection made in ETCNTINITCTL[INTINITEN]. When ETCNTINITCTL[INTINITEN] is set, then initialization of INTCNT2 counter with contents of ETCNTINIT[INTINIT] on a SYNC event or software force is determined by ETCNTINITCTL[INTINITFRC].

ETINTMIX, ETSOCAMIX and ETSOCBMIX Signals

In type 5 ePWM, the Event-Trigger submodule can generate and use ETINTMIX, ETSOCAMIX and ETSOCBMIX signals.

- **ETINTMIX**: This signal is a generated from the ORed combination of the sources enabled in the ETINTMIXEN register. The ETINTMIX signal can be used as a source for the EPWMxINT interrupt.

- **ETSOCAMIX**: This signal is a generated from the ORed combination of the sources enabled in the ETSOCAMIXEN register. The ETSOCAMIX signal can be used as a source for the EPWMxSOCA trigger signal.
- **ETSOCBMIX**: This signal is a generated from the ORed combination of the sources enabled in the ETSOCBMIXEN register. The ETSOCBMIX signal can be used as a source for the EPWMxSOCB trigger signal

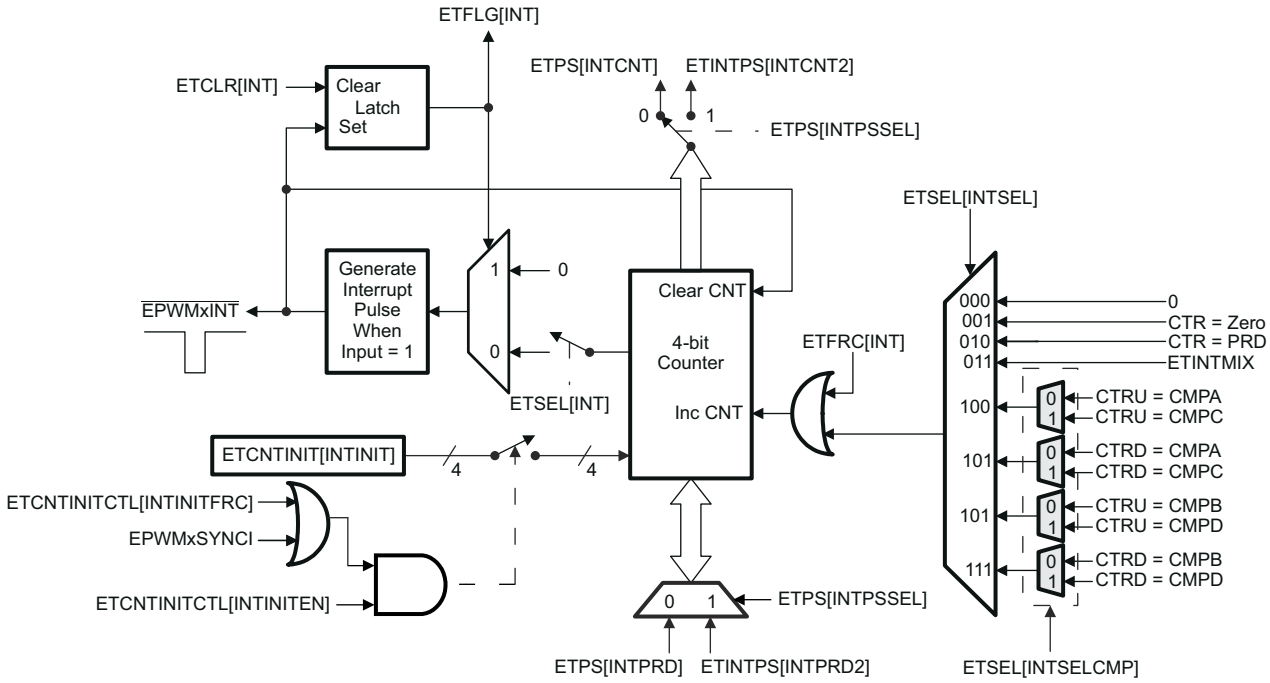
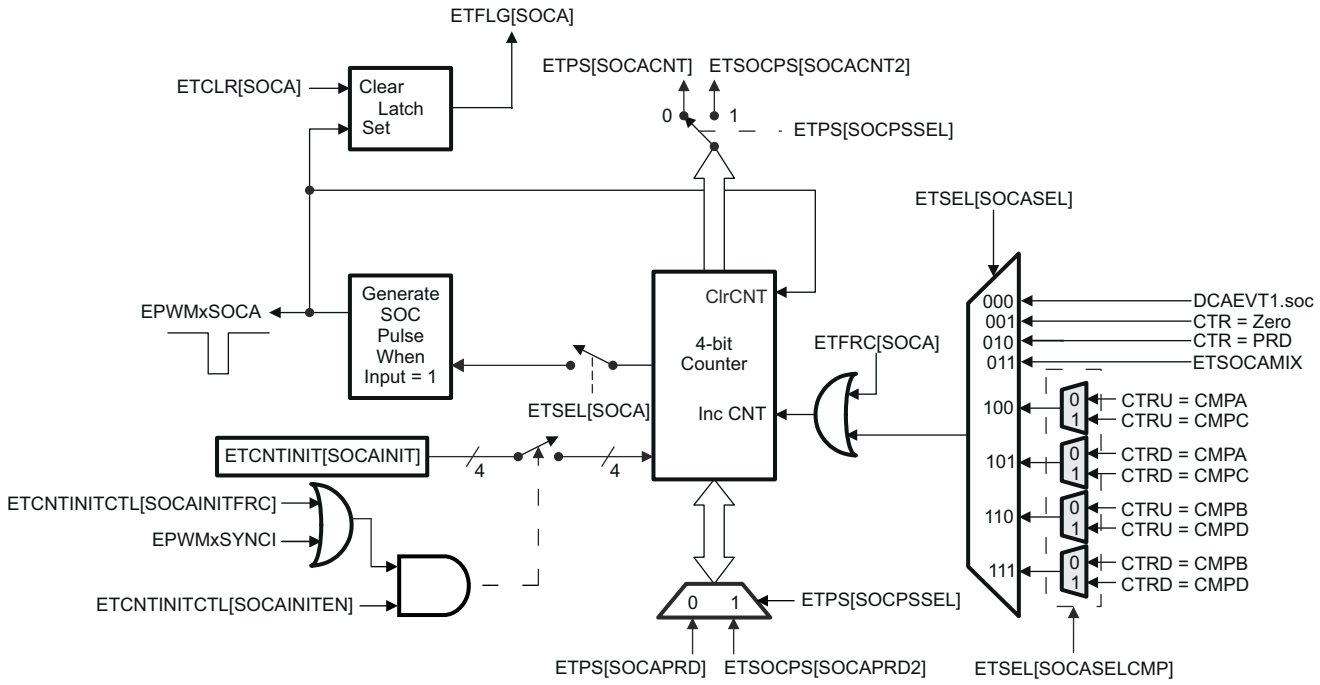


Figure 7-218. Event-Trigger Interrupt Generator

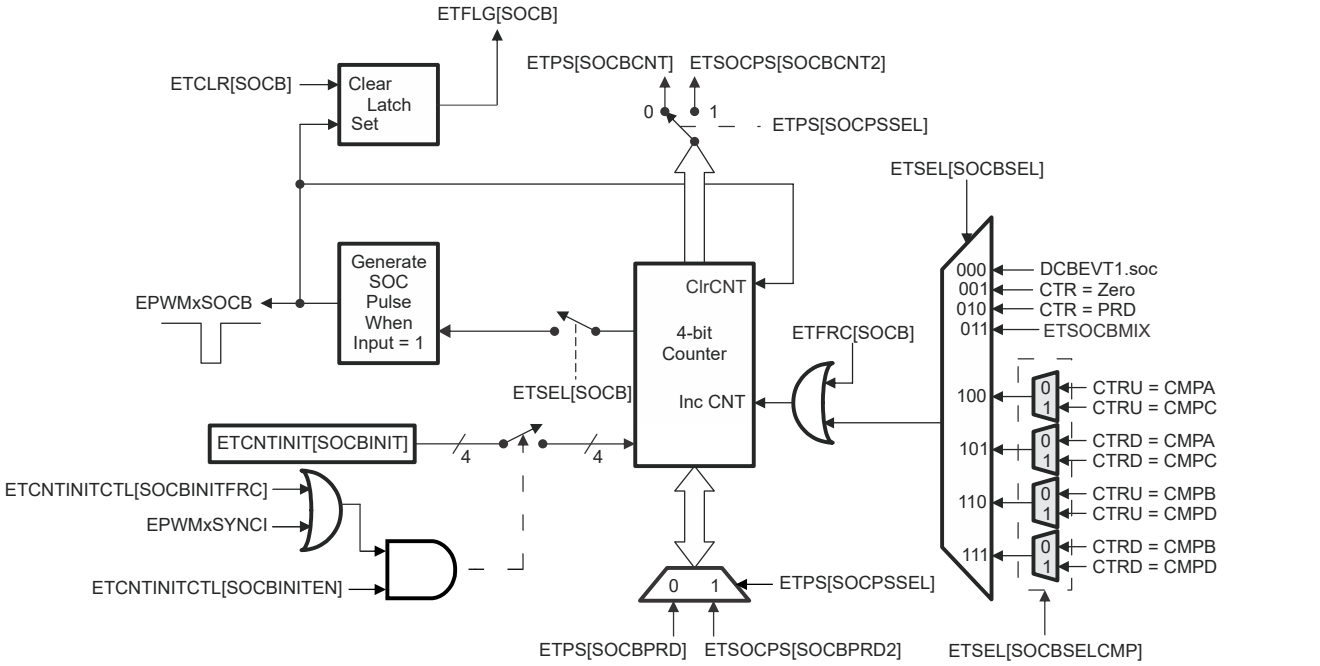
Figure 7-219 shows the operation of the event-trigger's start-of-conversion-A (SOCA) pulse generator. The enhancements include SOCASELCMP and SOCBSELCMP bit fields defined in the ETSEL register enable CMPC and CMPD events respectively to cause a start of conversion. The ETPS[SOCPSSEL] bit field determines whether SOCACNT2 and SOCAPRD2 take control or not. The ETPS[SOCACNT] counter and ETPS[SOCAPRD] period values behave similarly to the interrupt generator except that the pulses are continuously generated. That is, the pulse flag ETFLG[SOCA] is latched when a pulse is generated, but the interrupt generator does not stop further pulse generation. The enable and disable bit ETSEL[SOCAEN] 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 triggers an SOCA and SOCB pulse can be configured separately in the ETSEL[SOCASEL] and ETSEL[SOCBSEL] bits. The possible events are the same events that can be specified for the interrupt generation logic with the addition of the DCAEVT1.soc and DCBEVT1.soc event signals from the digital compare (DC) submodule. The SOCACNT2 initialization scheme is very similar to the interrupt generator with respective enable, value initialize and SYNC or software force options.



NOTE: The DCAEVT1.soc signals are generated by the Digital Compare (DC) submodule

Figure 7-219. Event-Trigger SOCA Pulse Generator

Figure 7-220 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.



NOTE: The DCBEVT1.soc signals are generated by the Digital Compare (DC) submodule

Figure 7-220. Event-Trigger SOCB Pulse Generator

7.5.6.13 Digital Compare (DC) Submodule

Figure 7-221 illustrates where the digital compare (DC) submodule signals interface to other submodules in the ePWM system.

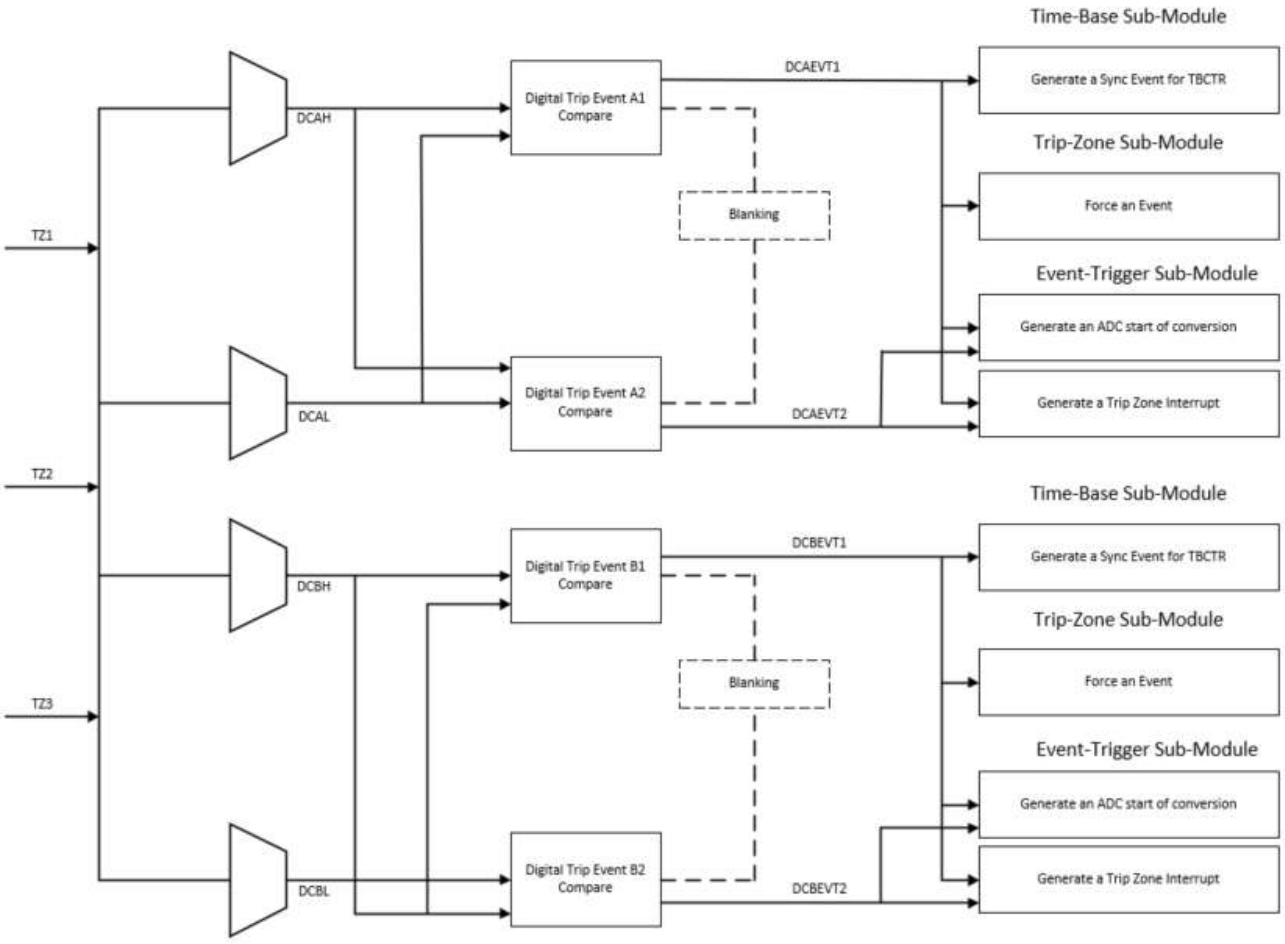


Figure 7-221. The Digital Compare Architecture

On this device, any of the GPIO pins can be flexibly mapped to be the trip-zone input and trip inputs to the trip-zone submodule and digital compare submodule. The Input X-BAR Input Select (INPUTxSELECT) register defines which GPIO pins gets assigned to be the trip-zone inputs / trip inputs.

The digital compare (DC) submodule compares signals external to the ePWM module (for instance, CMPSSx signals from the analog comparators) to directly generate PWM events/actions which then feed to the event-trigger, trip-zone, and time-base submodules. Additionally, blanking window functionality is supported to filter noise or unwanted pulses from the DC event signals.

Note

The user is responsible for driving the correct state on the selected pin before enabling the clock and configuring the trip input for the respective ePWM peripheral to avoid spurious latch of the TRIP signal.

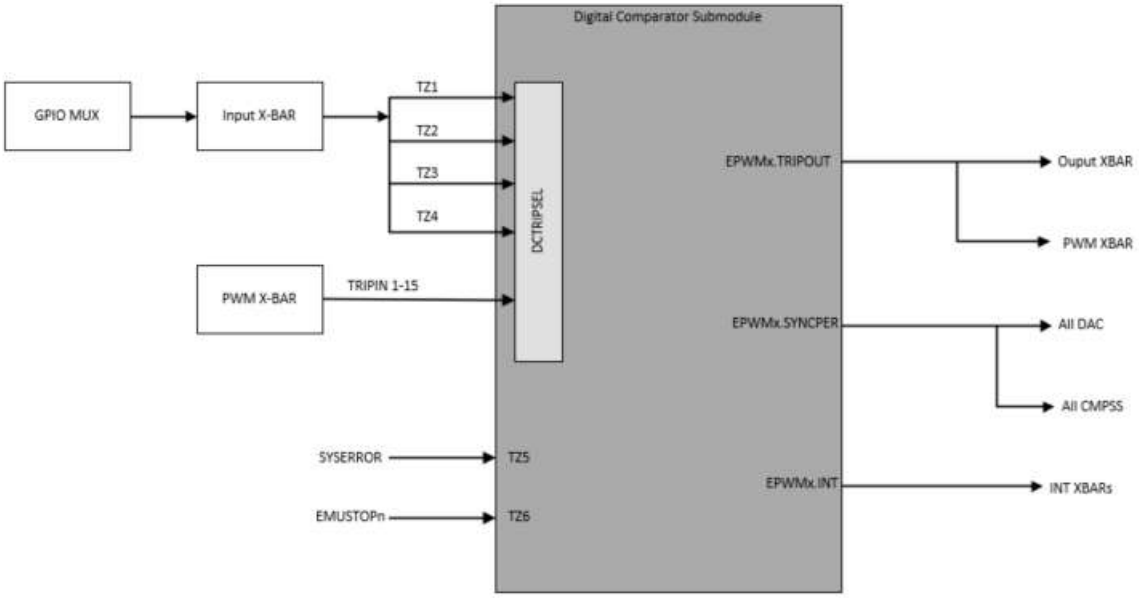


Figure 7-222. GPIO MUX-to-Trip Input Connectivity

7.5.6.13.1 Purpose of the Digital Compare Submodule

The key functions of the digital compare submodule are:

- Analog comparator (COMP) module outputs fed though the Input X-BAR, EPWM X-BAR, externally using the GPIO peripheral, interrupt controller signals, ECC error signals, TZ1, $\overline{TZ2}$, and $\overline{TZ3}$ inputs generate Digital Compare A High/Low (DCAH, DCAL) and Digital Compare B High/Low (DCBH, DCBL) signals.
- DCAH/L and DCBH/L signals trigger events that can then either be filtered or applied directly to the trip-zone, event-trigger, and time-base submodules to:
  - generate a trip zone interrupt
  - generate an ADC start of conversion
  - force an event
  - generate a synchronization event for synchronizing the ePWM module TBCTR.
- Event filtering (blanking window logic) can optionally blank the input signal to remove noise.

7.5.6.13.2 Enhanced Trip Action Using CMPSS

To allow multiple CMPSS at a time to affect DCA/BEVTx events and trip actions, there is a OR logic to bring together ALL trip inputs (up to 15) from sources external to the ePWM module and feed into DCAH, DCAL, DCBH, and DCBL as a “combinational input” using the DCTRIPSEL register. This is configured by selecting “Trip combination input” (value of 0xF) in the DCTRIPSEL register.

There is a discrete choice of which trip inputs to put through the combinational logic for generating the DCAH, DCAL, DCBH, and DCBL signals. This is achieved using the DCAHTRIPSEL, DCALTRIPSEL, DCBHTRIPSEL, and DCBLTRIPSEL register selections. Inputs selected for combinational input are passed through to the DCTRIPSEL register.

7.5.6.13.3 Using CMPSS to Trip the ePWM on a Cycle-by-Cycle Basis

When using the CMPSS to trip the ePWM on a cycle-by-cycle basis, steps can be taken to prevent an asserted comparator trip state in one PWM cycle from extending into the following cycle. The CMPSS can be used to signal a trip condition to the downstream ePWM modules. For applications like peak current mode control, only one trip event per PWM cycle is expected. Under certain conditions, it is possible for a sustained or late trip event (arriving near the end of a PWM cycle) to carry over into the next PWM cycle if precautions are not taken. If either the CMPSS Digital Filter or the ePWM Digital Compare (DC) submodule is configured to qualify the comparator trip signal, “N” number of clock cycles of qualification are introduced before the ePWM trip logic can respond to logic changes of the trip signal. Once an ePWM trip condition is qualified, the trip condition remains active for N clock cycles after the comparator trip signal has de-asserted. If a qualified comparator trip signal remains asserted within N clock cycles prior to the end of a PWM cycle, the trip condition is not cleared until after the following PWM cycle has started. Thus, the new PWM cycle detects a trip condition as soon as the cycle begins.

To avoid this undesired trip condition, the application can take steps to make sure that the qualified trip signal seen by the ePWM trip logic is deasserted prior to the end of each PWM cycle. This can be accomplished through various methods:

- Design the system such that a comparator trip is not asserted within N clock cycles prior to the end of the PWM cycle.
- Activate blanking of the comparator trip signal using the ePWM event filter at least two clock cycles prior to the PWMSYNCPER signal and continue blanking for at least N clock cycles into the next PWM cycle.
- If the CMPSS COMPxLATCH path is used, clear the COMPxLATCH at least N clock cycles prior to the end of the PWM cycle. The latch can be cleared by software (using COMPSTSCLR) or by generating an early PWMSYNCPER signal. The ePWM modules on this device include the ability to generate PWMSYNCPER upon a CMPC or CMPD match (using HRPCTL) for arbitrary PWMSYNCPER placement within the PWM cycle.

7.5.6.13.4 Operation Highlights of the Digital Compare Submodule

The following sections describe the operational highlights and configuration options for the digital compare submodule.

7.5.6.13.4.1 Digital Compare Events

As described in Section 7.5.6.13.1, trip zone inputs ($\overline{TZ1}$, $\overline{TZ2}$, and $\overline{TZ3}$) and CMPSSx signals from the analog comparator (COMP) module can be selected using the DCTRIPSEL bits to generate the Digital Compare A High and Low (DCAH/L) and Digital Compare B High and Low (DCBH/L) signals. Then, the configuration of the TZDCSEL register qualifies the actions on the selected DCAH/L and DCBH/L signals, which generate the DCAEVT1/2 and DCBEVT1/2 events (Event Qualification A and B).

Note

The $\overline{TZn}$ signals, when used as a DCEVT tripping functions, are treated as a normal input signal and can be defined to be active-high or active-low inputs. ePWM outputs are asynchronously tripped when either the $\overline{TZn}$, DCAEVTx.force, or DCBEVTx.force signals are active. For the condition to remain latched, a minimum of 3 * TBCLK sync pulse width is required. If pulse width is < 3*TBCLK sync pulse width, the trip condition can or can not get latched by CBC or OST latches.

The DCAEVT1/2 and DCBEVT1/2 events can then be filtered to provide a filtered version of the event signals (DCEVTFILT) or the filtering can be bypassed. Filtering is discussed further in Event Filtering. Either the DCAEVT1/2 and DCBEVT1/2 event signals or the filtered DCEVTFILT event signals can generate a force to the trip zone module, a TZ interrupt, an ADC SOC, or a PWM sync signal.

- **force signal:** DCAEVT1/2.force signals force trip zone conditions which either directly influence the output on the EPWMxA pin (using TZCTL, TZCTLDCA, TZCTLDCB register configurations) or, if the DCAEVT1/2 signals are selected as one-shot or cycle-by-cycle trip sources (using the TZSEL register), the DCAEVT1/2.force signals can effect the trip action using the TZCTL or TZCTL2 register configurations. The DCBEVT1/2.force signals behaves similarly, but affect the EPWMxB output pin instead of the EPWMxA output pin.

  The priority of conflicting actions on the TZCTL, TZCTL2, TZCTLDCA and TZCTLDCB registers is as follows (highest priority overrides lower priority):

  Output EPWMxA:
  - TZA (highest) -> DCAEVT1 -> DCAEVT2 (lowest)
  - TZAU (highest) -> DCAEVT1U -> DCAEVT2U (lowest)
  - TZAD (highest) -> DCAEVT1D -> DCAEVT2D (lowest)

  Output EPWMxB:
  - TZB (highest) -> DCBEVT1 -> DCBEVT2 (lowest)
  - TZBU (highest) -> DCBEVT1U -> DCBEVT2U (lowest)
  - TZBD (highest) -> DCBEVT1D -> DCBEVT2D (lowest)
- **interrupt signal:** DCAEVT1/2.interrupt signals generate trip zone interrupts to the interrupt controller. To enable the interrupt, set the DCAEVT1, DCAEVT2, DCBEVT1, or DCBEVT2 bits in the TZEINT register. Once one of these events occurs, an EPWMxTZINT interrupt is triggered, and the corresponding bit in the TZCLR register must be set to clear the interrupt.
- **soc signal:** The DCAEVT1.soc signal interfaces with the event-trigger submodule and can be selected as an event which generates an ADC start-of-conversion-A (SOCA) pulse using the ETSEL[SOCASEL] bit. Likewise, the DCBEVT1.soc signal can be selected as an event which generates an ADC start-of-conversion-B (SOCB) pulse using the ETSEL[SOCBSEL] bit.
- **sync signal:** The DCAEVT1.sync and DCBEVT1.sync events are ORed with the EPWMxSYNCI input signal and the TBCTL[SWFSYNC] signal to generate a synchronization pulse to the time-base counter.

Figure 7-223 and Figure 7-224 show how the DCxEVT1, DCxEVT2, or DCEVTFLT signals are processed to generate the digital compare A and B event force, interrupt, soc and sync signals.

In some of the applications like Phase Shifted Full Bridge (PSFB) Converters, it is required that different actions are taken on a CBC trip event and an OST trip event. This can be achieved using the DCxEVT1LAT.

- This latch can be cleared on CNT=0, CTR=PRD, and CNT=0 OR CTR=PRD events based on the setting of DCxCTL.EVTy.LATCLRSEL setting. This is similar to CBC latch clear mechanism.
- DCxEVTy.force signal can be chosen to be either the latched version or the unlatched version based on DCxCTL.EVTyLATSEL value.
- The status of DCxEVTyLAT signal can be accessed by reading DCxCTL.EVTyLAT field.

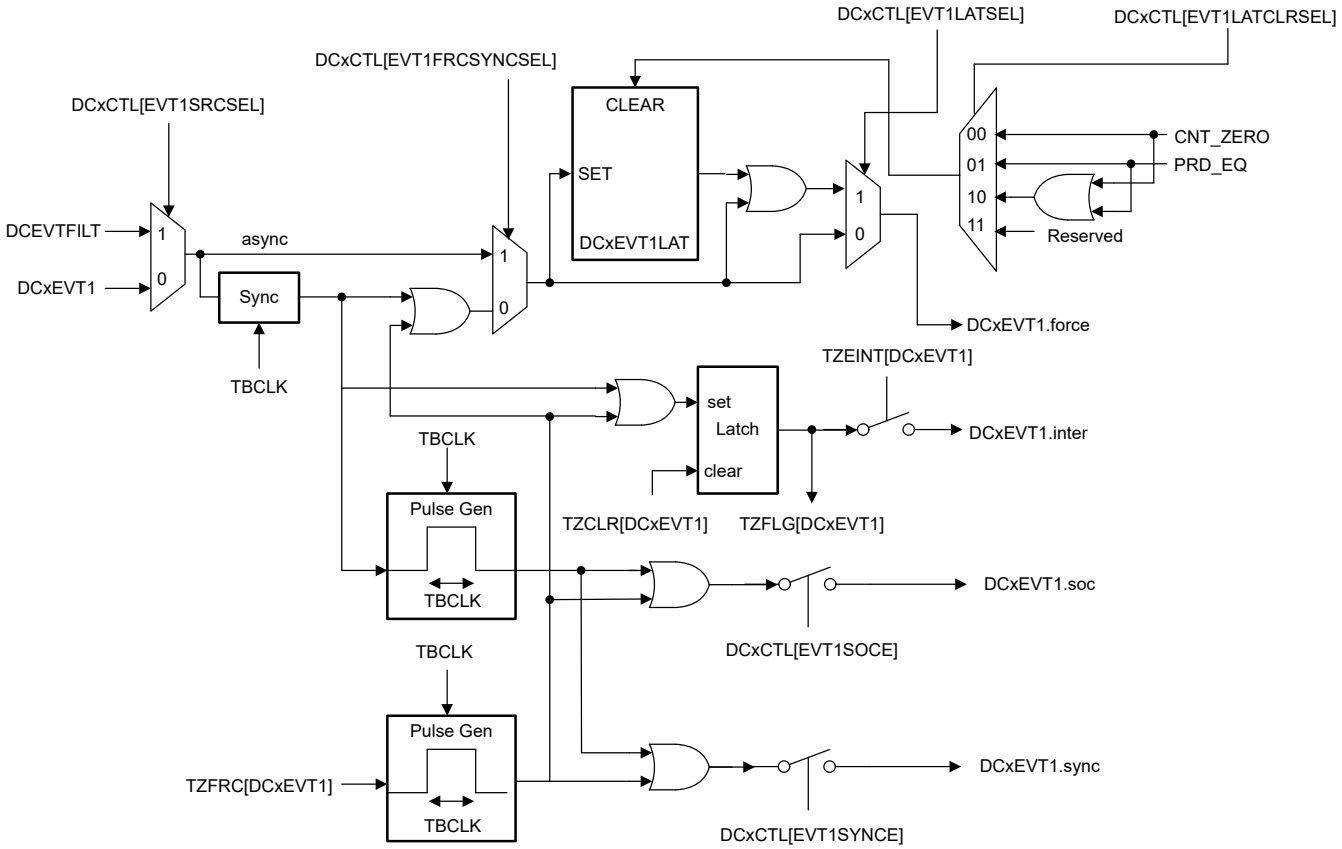


Figure 7-223. DCxEVT1 Event Triggering

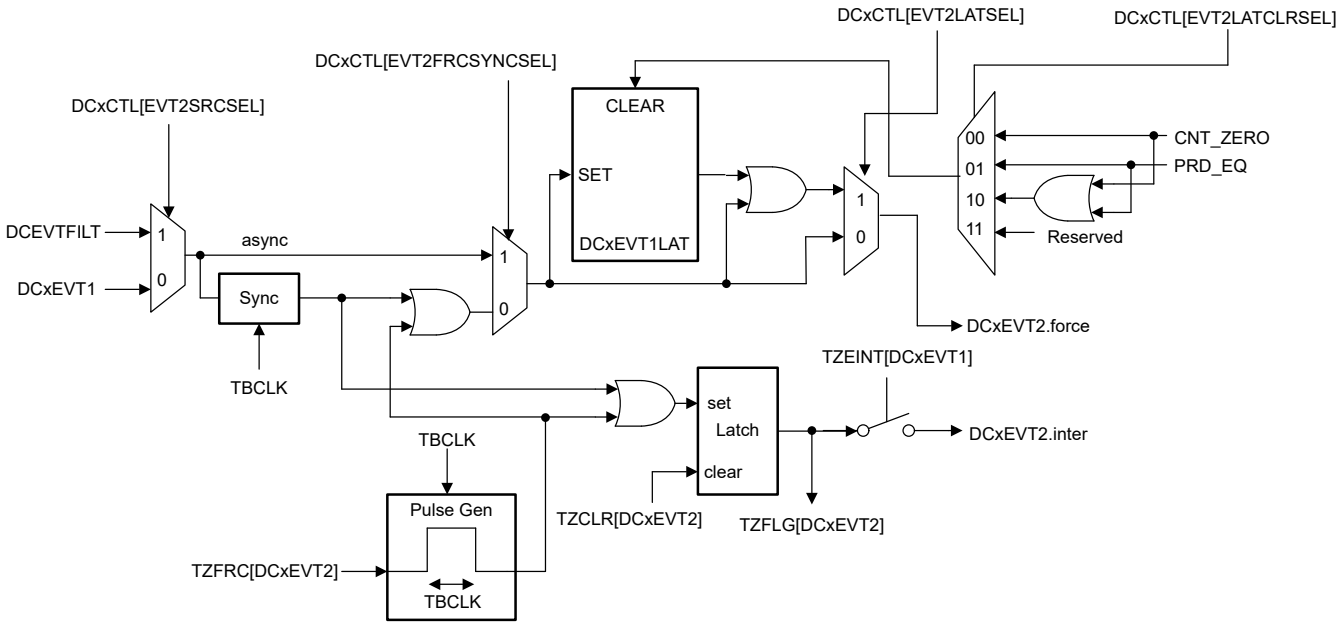


Figure 7-224. DCxEVT2 Event Triggering

Note

In some of the applications like Phase Shifted Full Bridge (PSFB) Converters, DCxEVT1LAT can be used on a CBC trip event and an OST trip event.

7.5.6.13.4.2 Valley Switching

Event filtering depicts the valley switching function along with the event filtering logic described in Section 7.5.6.13.4.3. This function can be used to achieve programmable valley switching without any additional external circuitry. This module provides an on-chip hardware mechanism that can:

- Capture the oscillation period
- Accurately delay the PWM switching instant
- Allow a programmable number of edges before the delay takes effect
- Provide multiple choices of triggers and events
- Allow easy adaptability for optimum performance under changing system/operating conditions

The DCxEVTy signal needs further processing to support valley switching. Here is a brief description of how valley switching function is enabled:

1. Select one of the DCxEVTy events as input to the valley switching block (DCFCTL[SRCSEL]) with an option to add the blanking window (Blank Control Logic). This is where the comparator output (or external input) above is selected as an input to the valley switching block.
2. Configure the edge filter to capture ‘n’ rising, falling or both edges through the edge selection logic (DCFCTL[EDGEMODE, EDGECOUNT]).
3. Select the correct event to reset and restart the edge filter (VCAPCTL[TRIGSEL]). Edge capturing event is triggered or armed by this selected edge.
4. Enable valley capture logic (VCAPCTL[VCAPE]).
5. Select the start edge that indicates the start of capture for oscillation period measurement (VCNTCFG[STARTEDGE]). This is where the 16-bit counter starts counting.
6. Select the stop edge (VCNTCFG[STOPEDGE]) that indicates the edge at which the 16-bit counter stops counting. The captured counter value (CNTVAL) provides oscillation period information.
   - The STOPEDGE value must always be greater than STARTEDGE value.
7. Configure and apply the captured delay (CNTVAL) to the edge filtered DCxEVTy signal. The CNTVAL value can be applied as is or applied in conjunction with a software programmed value (useful for offset adjustment) (SWVDELVAL) or only a fraction of the delay can be applied with or without SWVDELVAL. This is useful to correctly apply a delay corresponding to the valley point. (VCAPCTL[VDELAYDIV])
8. Configure VCAPCTL[EDGEFILTDLYSEL] to apply hardware delay based on the captured value above.

Once the counter is stopped, counter value is copied into CNTVAL register and counter is reset to zero. No further captures are done until the logic is triggered again by occurrence of event selected by VCAPCTL[TRIGSEL]. In this implementation, the software trigger is used as the source for VCAPCTL[TRIGSEL]. Upon occurrence of the trigger event, irrespective of the current status of the counter, the counter is reset and starts counting from zero upon occurrence of the STARTEDGE. Similarly, upon occurrence of the trigger event, the edge filter is reset and starts counting from zero upon occurrence of the STARTEDGE.

Output from the valley switching block (DCEVTFILT) is then used to synchronize the PWM time-base. The process is shown in Figure 7-225.

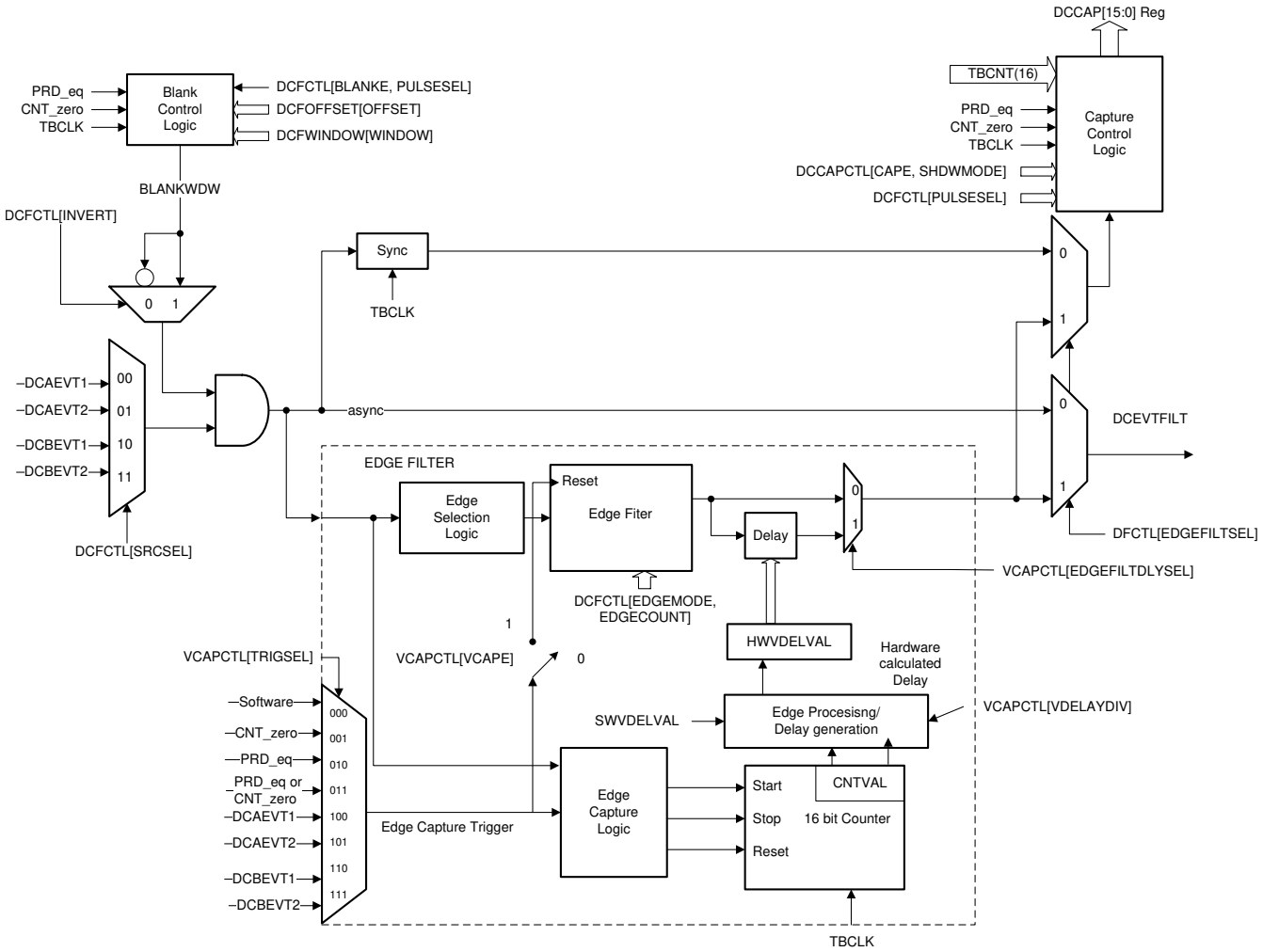


Figure 7-225. Valley Switching

7.5.6.13.4.3 Event Filtering

Blank Control Logic: The DCAEVT1/2 and DCBEVT1/2 events can be filtered using event filtering logic to remove noise by optionally blanking events for a certain period of time. This is useful for cases where the analog comparator outputs can be selected to trigger DCAEVT1/2 and DCBEVT1/2 events, and the blank control logic is used to filter out potential noise on the signal prior to tripping the PWM outputs or generating an interrupt or ADC start-of-conversion. Blank control logic is used to define a blanking window, which ignores all event occurrences on the signal while the window is active. The blanking window is configured in the DCFCTL, DCFOFFSET, and DCFWINDOW registers. The DCFCTL register enables the blanking window and aligns the blanking window to either a CTR = PRD pulse or a CTR = 0 pulse or both CTR = PRD and CTR = 0 as specified by DCFCTL[PULSESEL]. DCFCTL[SRCSEL] selects the DCxEVTy event source for the DCEVTFILT signal. An offset value in TBCLK counts is programmed into the DCFOFFSET register, which determines at what point after the CTR = PRD or CTR = 0 pulse the blanking window starts. The duration of the blanking window, in number of TBCLK counts after the offset counter expires, is written to the DCFWINDOW register by the application. Before and after the blanking window ends, events can generate soc, sync, interrupt, and force signals as before. Figure 7-226 shows the details of the event filtering logic.

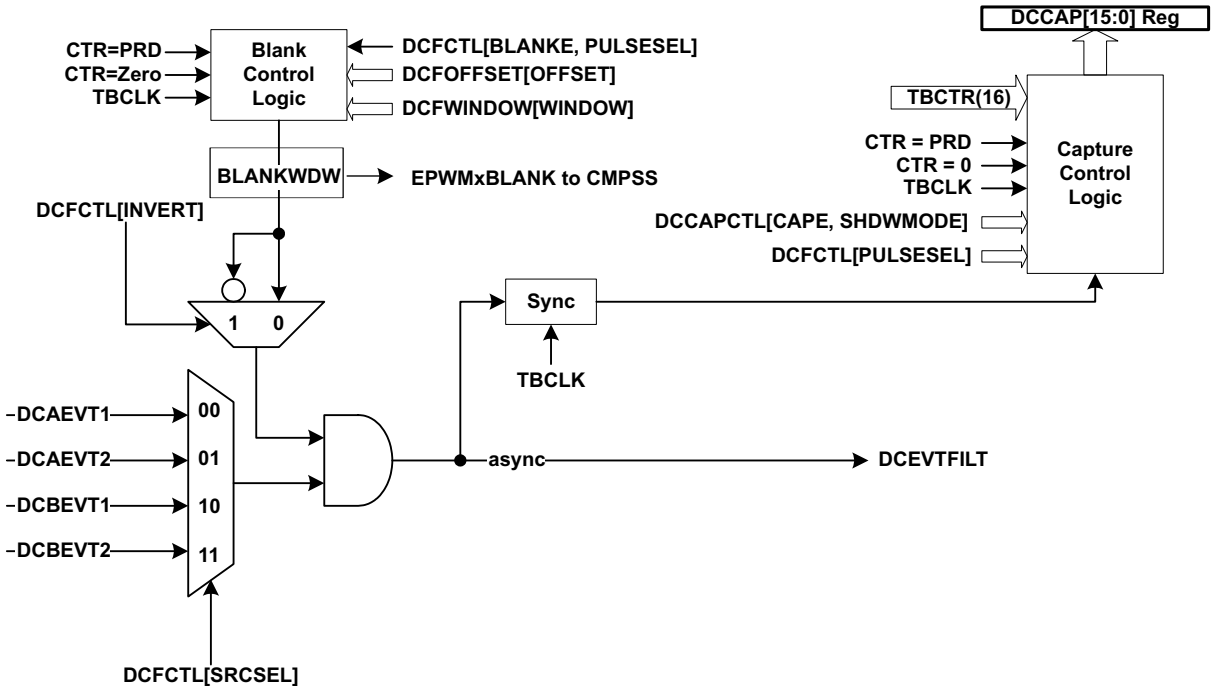


Figure 7-226. Event Filtering

Capture Control Logic: The event filtering can also capture the TBCTR value of the selected DCxEVTy event as configured in the DCCAPCTL register. When capture control logic is enabled, the selected DCxEVTy event triggers capture of the TBCTR to the active register. The CPU reads directly from the active register unless shadow mode is enabled by DCCAPCTL[SHDWMODE]. When shadow mode is enabled, the active register information is copied to shadow register on the event specified by DCFCCTL[PULSESEL], and the CPU reads from the shadow register. After the selected DCxEVTy event, no further capture events occur until the event specified by DCCAPCTL[CAPMODE]. The CAPMODE can be configured two ways: (1) no further capture events occur until the event defined by DCFCTL[PULSESEL] or (2) no further capture events occur until the compare-event flag at DCCAPCTL[CAPSTS] is cleared by DCCAPCTL[CAPCLR].

Figure 7-227 illustrates several timing conditions for the offset and blanking window within an ePWM period. Notice that if the blanking window crosses the CTR = 0 or CTR = PRD boundary, the next window still starts at the same offset value after the CTR = 0 or CTR = PRD pulse.

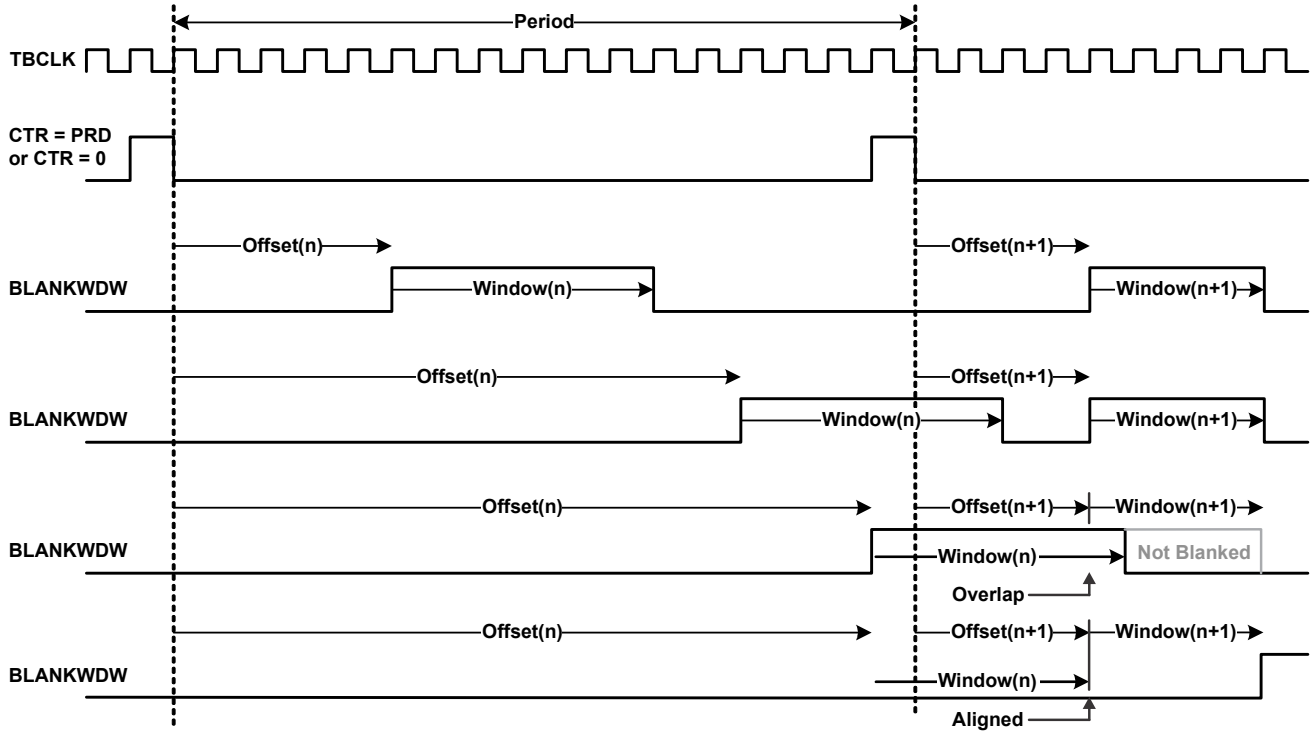


Figure 7-227. Blanking Window Timing Diagram

BLANKPULSEMIX Signals

The DCFCTL MUX (available for Blank Control Logic and Capture Control Logic) has new options that allows the mux to select the BLANKPULSEMIX signal. The BLANKPULSEMIX signal is used, if the signal is selected by DCFCTL[PULSESEL]

BLANKPULSEMIX and DCCAPMIX Signals

The CAPCTL MUX (available in the Capture Control Logic) and DCFCTL MUX (available for Blank Control Logic and Capture Control Logic) have new options in type 5 ePWM which allows them to select the DCCAPMIX or BLANKPULSEMIX signal respectively.

In type 5 ePWM, the shadow load signal for the Capture Control Logic can be different from the blanking window alignment signal (which is selected by DCFCTL[PULSESEL]). The CAPCTL mux can be configured to use the DCCAPMIX signal

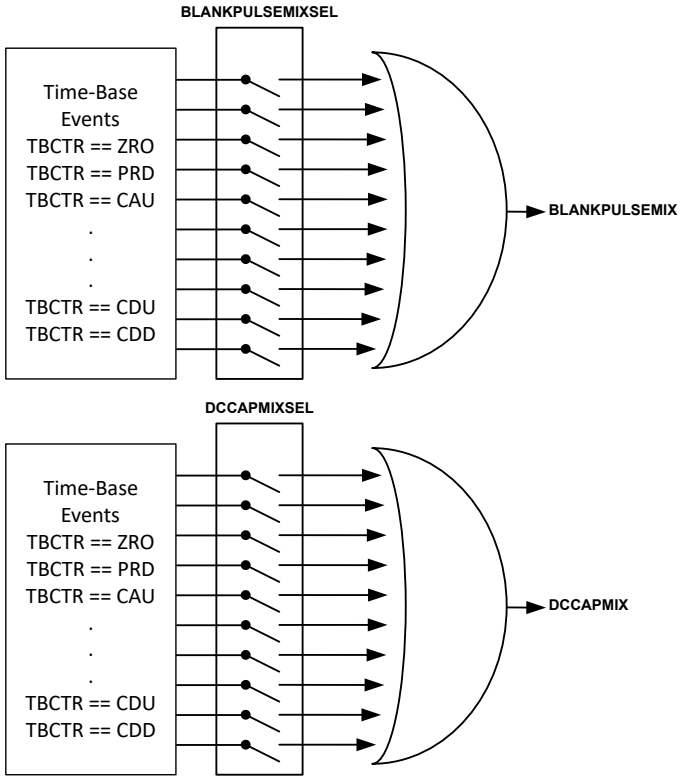


Figure 7-228. BLANKPULSEMIX and DCCAPMIX Signal Source

7.5.6.13.4.4 Event Detection

This logic is primarily intended to detect an occurrence of a trip event in a configured time window. The window is configured by MIN and MAX values configured in the XMINMAX register sets.

Figure 7-229 indicates the window spread across MIN and MAX bounds and the edge of the chosen signal occurring in that window. The purpose of this block is to detect the occurrence of such edge. If no such edge occurs, this module generates a trip event as well as an interrupt, if configured.

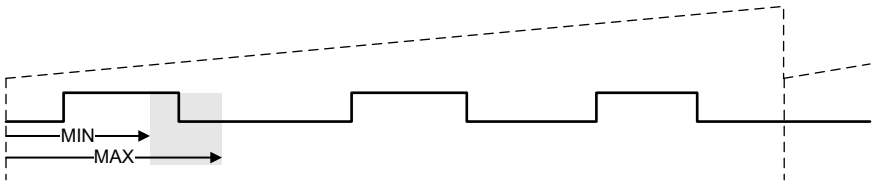


Figure 7-229. MIN, MAX Settings and Window for Capture Event Detection

7.5.6.13.4.4.1 Input Signal Detection

The CAPTRIPSEL, CAPINTRIPSEL and CAPGATETRIPSEL muxes are used for signal selection. Figure 7-230 shows how the CAPIN and CAPGATE signal source is selected.

CAPIN Input: This signal (any input coming from EPWM X-BAR) can be configured as the signal input on which the edge detection logic is performed.

CAPGATE Input: This signal (any input coming from EPWM X-BAR) can be configured as the gating signal to Min/Max logic. This signal gates the CAPIN input signal.

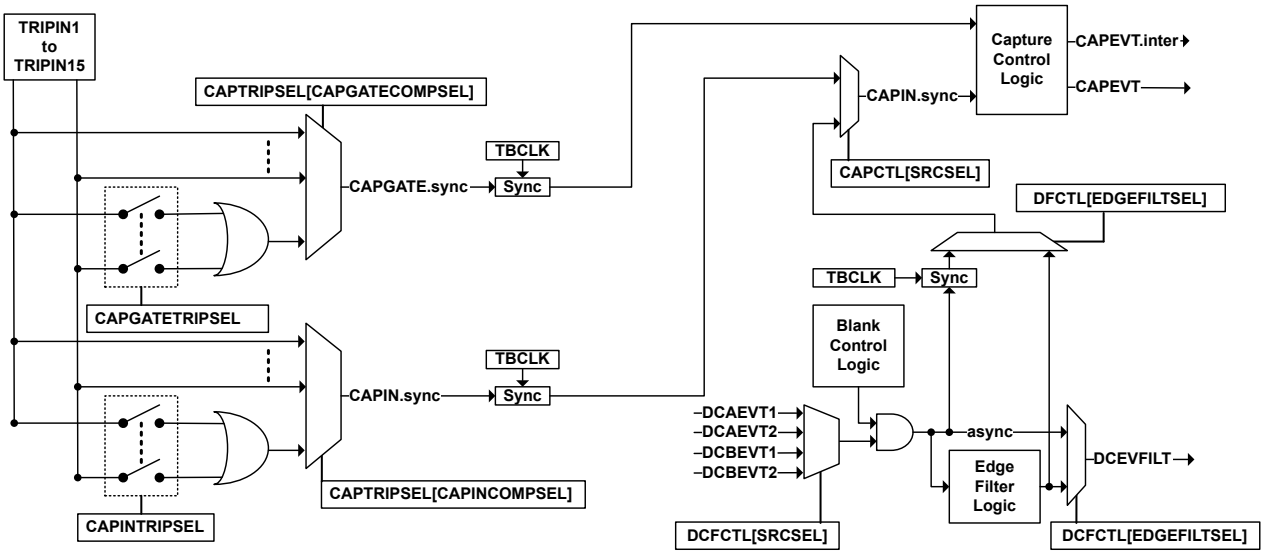


Figure 7-230. CAPIN and CAPGATE Source Selection

Once selected, Figure 7-231 demonstrates how the CAPGATE and CAPIN signals propagate into the counter capture logic. The logic works in the following way:

- CAPCTL[GATEPOL] is used for the polarity selection of the gating input to be optionally inverted or tied to a 0 or 1.
- CAPCTL[CAPINPOL] can be used to select the edge polarity of CAPIN.sync signal. CAPIN.sync signal is selected from the DCEVTFILT options and the CAPIN signal using CAPCTL[SRCSEL] bits.

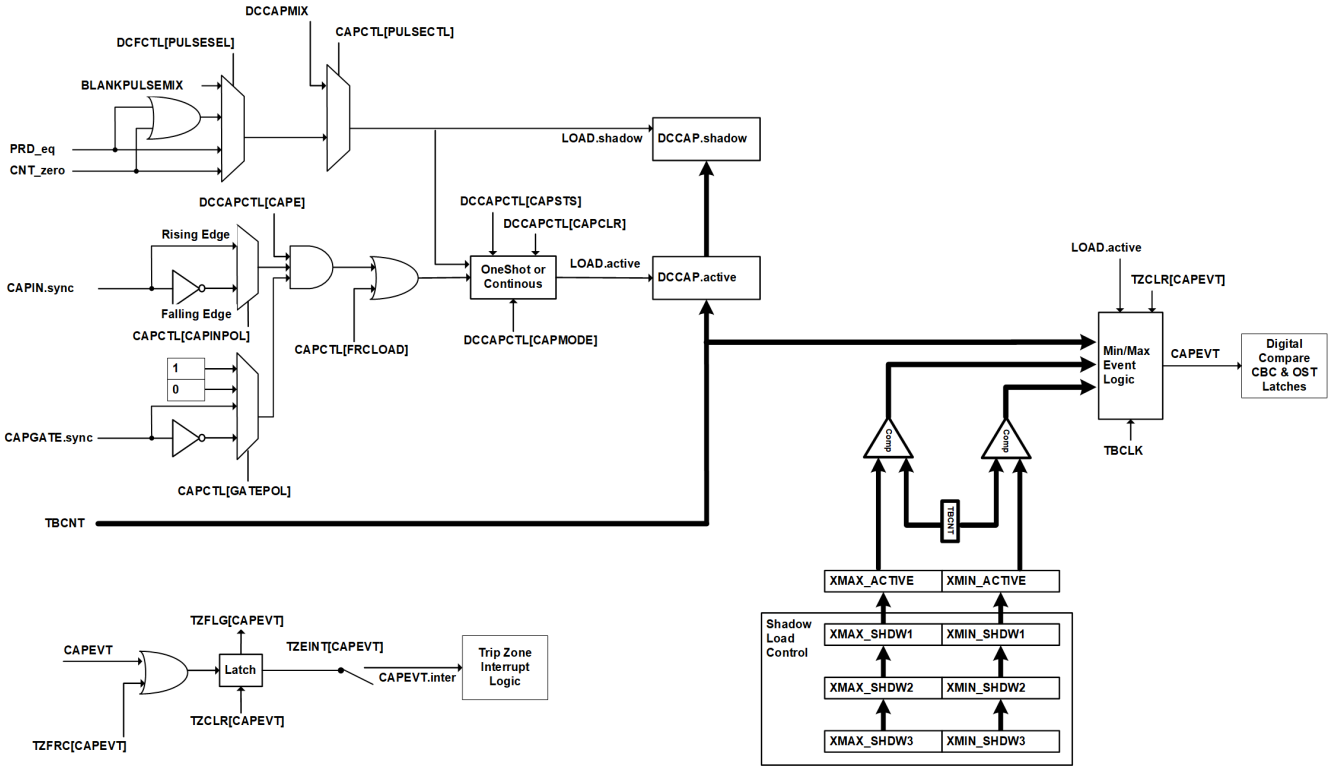


Figure 7-231. Counter Capture Logic

7.5.6.13.4.4.2 MIN and MAX Detection Circuit

The XMINMAX register has XMIN and XMAX fields that can be programmed to set the MIN and MAX limits of the programmable edge detection window. These registers have 3-level buffering similar to the XCMPn registers. The shadow to active loading of these registers is always in sync with the buffer pointers. Any shadow to active loads occur as per the XLOAD register configuration defined for the XCMPn registers such that the MIN and MAX values used are always in line with the corresponding XPRD/XCMPn values used for a given PWM cycle.

The logic works in the following way:

- The TBCNT value is continually monitored and compared against the active MIN value. Match of TBCNT to the active MIN value triggers the edge monitoring occurrence.
- When the TBCNT value reached the MIN value, the active LOAD signal is monitored waiting for an edge event to occur.
- If an edge vent occurs before TBCNT reaches the active MAX value, then no further action is taken. The logic resets and TBCNT is compared to the active MIN value again.
- If no edge occurs and TBCNT reaches the active MAX value, then the CAPEVT signal is set high and a CAP interrupt signal can also be generated. The CAPEVT signal needs to be cleared through software for TBCNT to be monitored against the MIN value again.

The Min and Max monitoring is enabled and disabled in three ways:

- By enabling/disabling the circuit via the DCCAPCTL[CAPE] bit
- By the CAPGATE signal which can be sourced from an TRIPINPUT signal to the module.
- By writing the same value into the XMIN and XMAX bits.

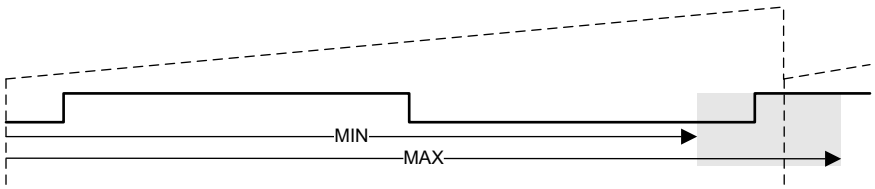


Figure 7-232. Capture Logic Boundary Condition

Note

Possible boundary condition of MIN/MAX window exceeding the period value: In this case, the XMAX bit can have a value lower than the XMIN bit such that the window can go over the period boundary.

7.5.6.14 XCMP Submodule

7.5.6.14.1 XCMP Complex Waveform Generator Mode

The XCMP complex waveform generator mode is available in the type 5 ePWM and is enabled when XCMPEN is set. The main feature of the XCMP mode is to generate multiple ePWM pulses, with high resolution edge placement if needed, within one ePWM period.

XCMP features include:

- Up to eight counter compare registers XCMP1-XCMP8
- High resolution (HRPWM) edge placement support
- Up-Count counter mode support

Note

Down-Count and Up-Down-Count counter modes are not supported

- Pulse generation is only supported on XCMP1-8 matches (no support for counter events such as PRD and ZRO, or T1/T2 events)
- ePWM module synchronization is not allowed in XCMP mode

Note

The application software must disable the ePWM synchronization when XCMP mode is enabled.

- XCMP1-8 are loaded through CMPA and CMPB

Note

A minimum of 4 cycles difference (including the HR component) between adjacent XCMP values must be maintained to make sure of minimum pulse width.

- The eight XCMPn registers, can be allocated to either CMPA or CMPB through the application software configuration
- XAQCTLA and XAQCTLB registers determine the actions taken on the ePWM output for each XCMP1-8 counter matches
- Up to three ePWM period cycles can be configured at once through three shadow buffers
  - Each shadow buffer contains shadow registers for XCMP1-8, XTBPRD, XAQCTLA, XAQCTLB, CMPC, CMPD, and XMINMAX (which is used for CAPEVT signal generation)
  - Shadow buffer SHDW2 and SHDW3 can be repeated up to eight times
- All ePWM modules can be linked to trigger the start of their shadow loading at the same time through EPWMXLINKXLOAD

7.5.6.14.2 MIN-MAX Event Logic

The XMINMAX register has XMIN and XMAX fields that can be programmed to set the MIN and MAX limits of the programmable edge detection window. CAPIN signal (any input coming from EPWM X-BAR) can be configured as the signal input on which the edge detection logic is performed.TheDCCAP captures the rising or falling edge of the gated CAPIN signal.

The XMIN and XMAX shadowing occurs following the same logic as the XCMPn registers. The XLOADCTL register configures the shadowing for the XMAX and XMIN values.

Note

XMIN and XMAX values can be written in way that the TBCTR=PRD event falls within their range. For example, TBPRD = 100, XMIN = 95, XMAC = 5 is a valid configuration.

7.5.6.14.3 XCMP Shadow Buffers

Three SHDW buffers are available for XCMP configurations. Each SHDW buffer contains the XCMP1-8 values (CMPA and CMPB values), XTBPRD (TBPRD value), XCMPC (CMPC value), XCMPD (CMPD value), XAQCTLA and XAQCTLB. Each SHDW buffer also contains the XMINMAX values which are used for CAPEVT signal generation.

With the three SHDW buffer (SHDW1, SHDW2 and SHDW3) the values used for the upcoming ePWM period cycles can be buffered.

With XCMPEN set, the load of the active registers are controlled by the XLOADCTL and XLOAD registers. The shadow to active loading of the registers (other than XMINMAX, XCMPC, XCMPD) are always done three cycles prior to TBCTR==ZERO event. XMINMAX, XCMPC and XCMPD shadow loading is done at TBCTR==PRD.

There are two load modes configured by XLOADCTL[LOADMODE]:

- **LOADONCE** Mode (XLOADCTL[LOADMODE] = 0)
  - In LOADONCE mode, XLOADCTL[SHDWBUFPTR_ LOADONCE] is used to set the pointer location of the shadow buffer.
  - XLOADCTL[SHDWBUFPTR_ LOADONCE] is set by the user and is **NOT** automatically decremented. Upon the occurrence of the first load strobe (write of ‘1’ to XLOAD[STARTLD] bit), active register set is loaded from the XLOADCTL[SHDWBUFPTR_ LOADONCE] SHDW selected by the user. Further load strobes are ignored, and ePWM waveform generation continues with the active register set until next XLOAD[STARTLD] is initiated.
  - When the software sets the XLOAD[STARTLD] bit again, the active register set is loaded from the XLOADCTL[SHDWBUFPTR_ LOADONCE] SHDW selected by the user. If the user wants to initiate a SHDW load from a different shadow register set, then the software can update the XLOADCTL[SHDWBUFPTR_ LOADONCE] register accordingly before setting the XLOADCTL[STARTLD].

- **LOADMULTIPLE** Mode (XLOADCTL[LOADMODE] = 1)
  - XLOADCTL[SHDWBUFPTR_ LOADMULTIPLE] always points to the current shadow register set that is loaded into the active registers set.
  - Setting the XLOAD[STARTLD] bit initiates a load strobe. The SHDW buffer pointer resets to XLOADCTL[SHDWLEVEL] and the corresponding buffer contents are loaded to the active register set. When the next valid load strobe arrives, XLOADCTL[SHDWBUFPTR_ LOADMULTIPLE] is decremented by 1 and the corresponding buffer contents are loaded to the active register set. This continues until the XLOADCTL[SHDWBUFPTR_ LOADMULTIPLE] value reaches 1. At this time SHDW1 values get copied to the active register set. Further load strobes are ignored and the ePWM waveform generation continues with the active register set until next XLOAD[STARTLD] is initiated.
  - Once the XLOADCTL[SHDWBUFPTR_ LOADMULTIPLE] value reaches 1, no further decrements to the this pointer are done until the next STARTLD initiation. This means the XLOADCTL[SHDWBUFPTR_ LOADMULTIPLE] remains at value of '1', indicating that the SHDW1 register set is in use till the next load initiation by user.
  - For a SHDWLEVEL of 3 buffers SHDW3 is loaded first followed by SHDW2 and SHDW1. Then until the next STARTLD write by the software, the SHDW1 values are in use.

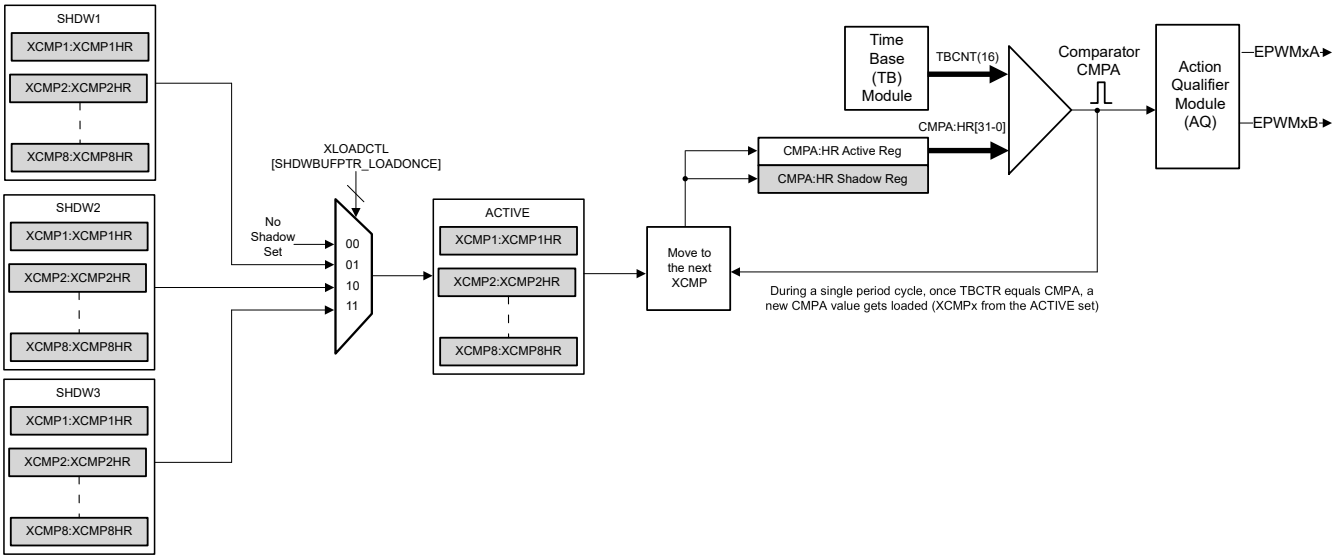


Figure 7-233. XCMP- Load Once Functionality

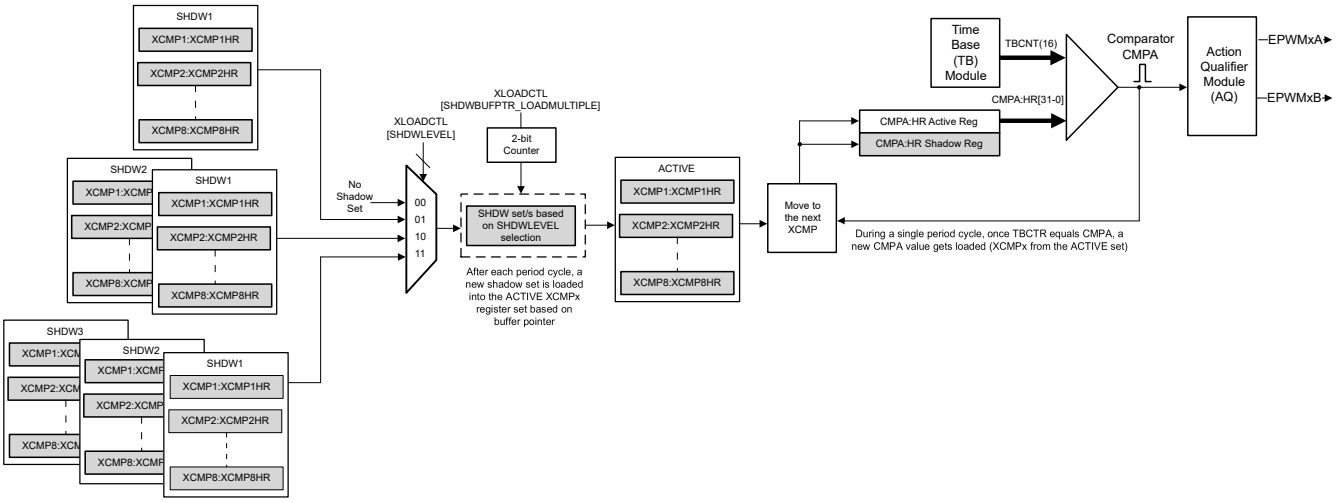


Figure 7-234. XCMP- Load Multiple Functionality

With this new loading scheme, the global load functionality also changes when using XCMP mode. In this new configuration, once a write to STARLD occurs, the next time the time base counter equals zero or a force load software write occurs, the shadow buffer pointers get reset, based on the load mode (load once or load multiple).

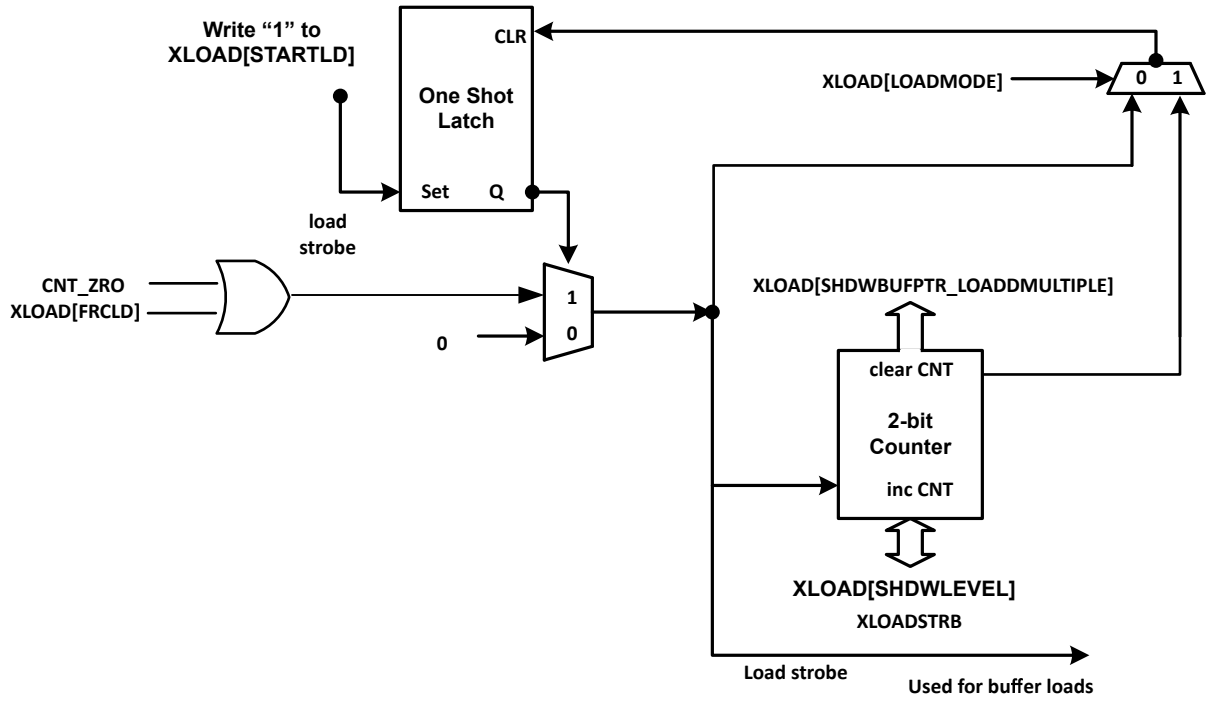


Figure 7-235. Global Load: Signals and Registers

Shadow buffers can also be repeated more than once. Shadow buffer repeat counters are:

- Users can optionally repeat each shadow buffer multiple times. This option sets the repeat count for SHDW2 and SHDW3 buffers before the pointer moves to the SHDW1 buffer. SHDW1 buffer by default repeats until the next load is initiated by the software and hence there is no configurable repeat option for SHDW1 buffer.
- Repeat counter option of the shadow buffers is applicable in LOADMULTIPLE mode. In the LOADONCE mode, user can manually keep track of the repeat counts and move to the SHDW pointer buffer.
- Each shadow buffer has a 3-bit counter. Each buffer can be set to repeat up to 8 times before moving the pointer to the next buffer.
- XLOADCTL[RPTBUF2PRD] and XLOADCTL[RPTBUF3PRD] are used to control the repeat period for each SHDW buffer.

No shadowing can be set by setting the XLOADCTL[SHDWLEVEL] to '0'. In this case, the ACTIVE registers are available for use (XCMP1_ACTIVE, XCMP2_ACTIVE, and so on).

7.5.6.14.4 XCMP Allocation to CMPA and CMPB

The first criteria that must be selected is whether both EPWM channel A and channel B outputs are required. If both channel A and channel B are required, XCMP registers must be assigned to both CMPA and CMPB. The XCMPn registers loaded to CMPA are used for configuring the A channel through XAQCTLA actions. The XCMPn registers loaded to CMPB are used for configuring the B channel through XAQCTLB actions.

XCMP allocation to CMPA and CMPB is done through XCMPCTL1.XCMPSPLIT. If both channel A and channel B are required in the system, then the XCMPCTL1.XCMPSPLIT must be set. This allows CMPA to use XCMP1-n (where n has a maximum value of 4) while CMPB uses XCMP5-m (where m has a maximum value of 8). If only channel A is needed, then XCMPCTL1.XCMPSPLIT must be cleared, allowing CMPA to use XCMP1-n (where n has a maximum value of 8), which means up to eight edges can be generated on channel A.

Note

The maximum number of edges that channel B can have is four, when XCMP5-8 are allocated to CMPB and all four XCMP5-8 are used by setting the XCMPB_ALLOC to use all available XCMPs.

XCMPA_ALLOC and XCMPB_ALLOC determines how many of the available XCMPs for each CMPA and CMPB must be used in the ePWM configuration.

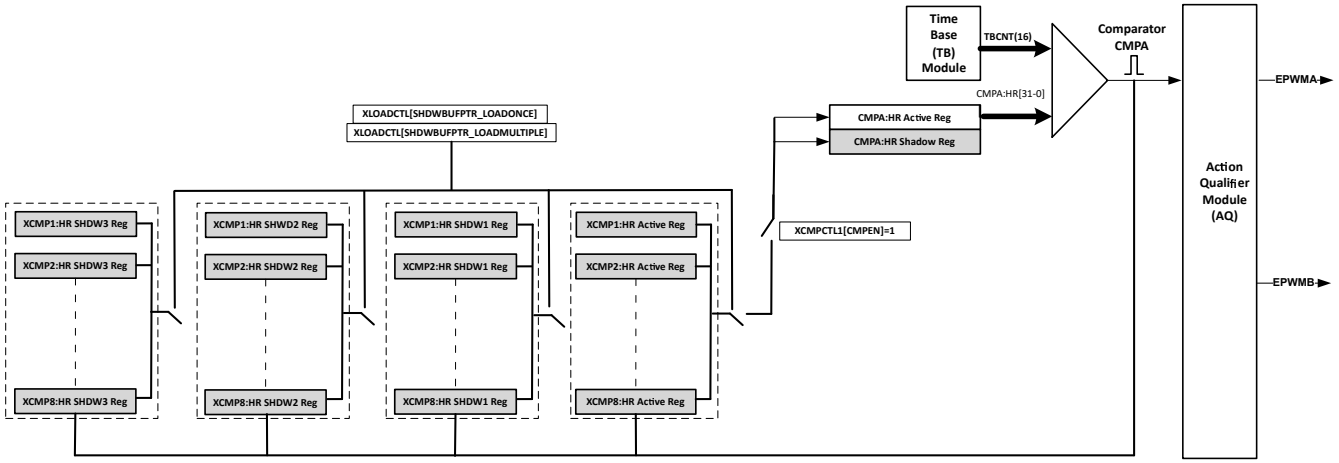


Figure 7-236. Allocate All XCMP1-8 to CMPA

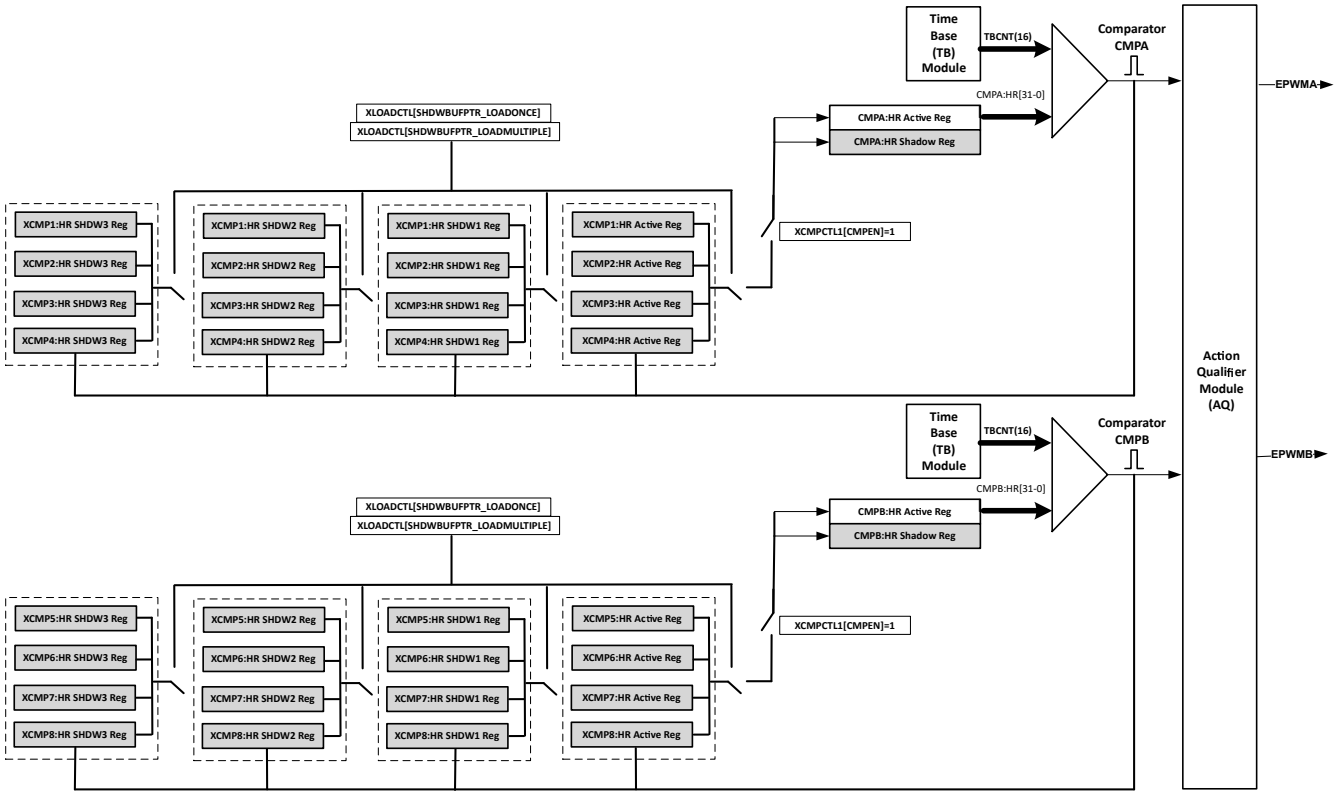


Figure 7-237. XCMP1-4 Allocated to CMPA and XCMP5-8 Allocated to CMPB

7.5.6.14.5 XCMP Operation

The XCMP complex waveform generation mode is described in this section.

The XCMP mode can be used to generate multiple edges within one ePWM period. The application software must write the location of the ePWM waveform edges to the XCMP registers. Each XCMPn register assigned

and used for an ePWM CMPx (CMPA or CMPB) must be spaced out according to the following guidelines to make sure of correct waveform generation.

Figure 7-238 shows an example of four XCMP values being loaded into CMPA during one period cycle and the remaining four XCMP values being used for CMPB. When the action for the last XCMP value loaded into CMPA/CMPB in a period is met, the last value for CMPA/CMPB remains until the next time TBCTR=0 due to a new shadow set load.

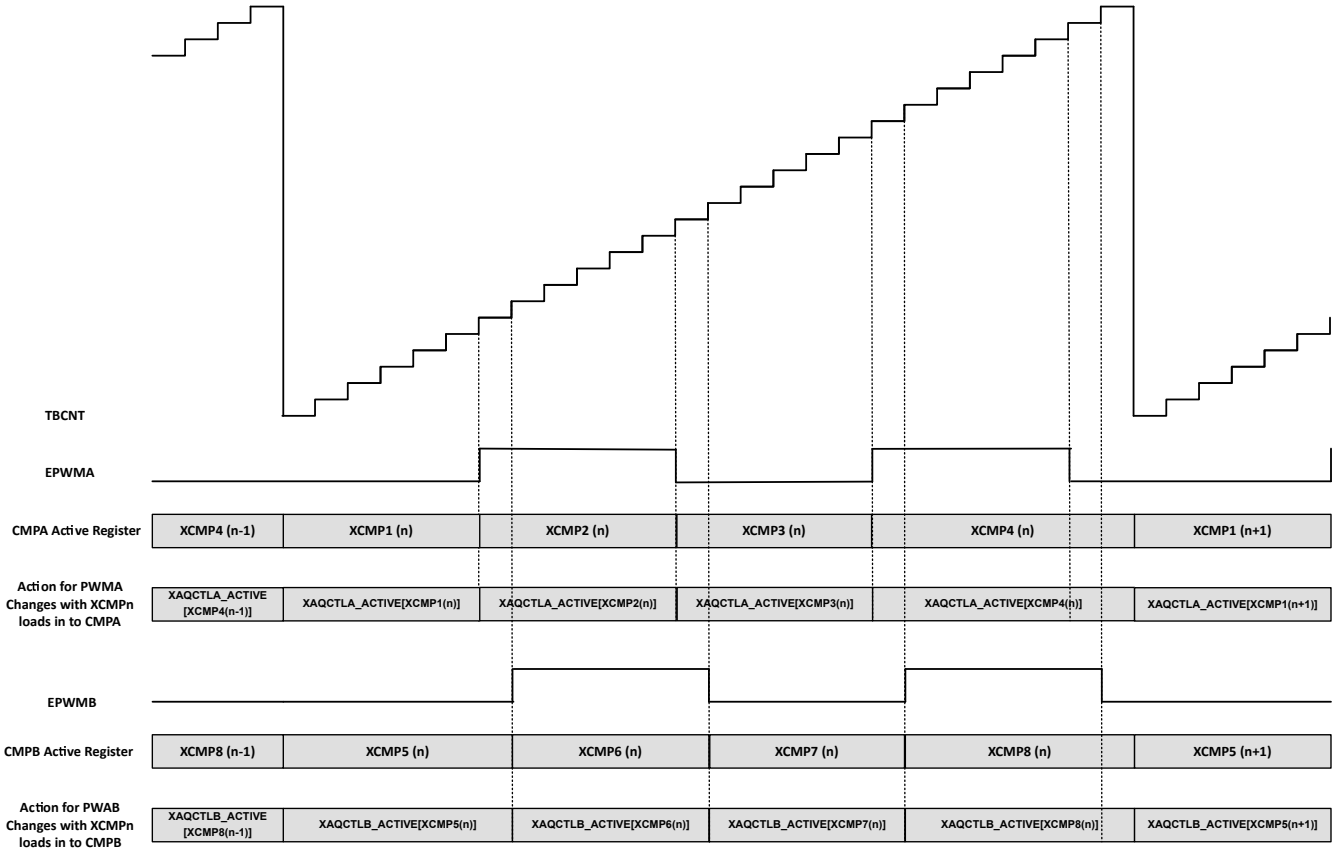


Figure 7-238. CMPA and CMPB values being loaded from XCMP registers

Assume XCMP1-3 are assigned and used by CMPA (XCMP4 is not used), and XCMP5-6 are assigned and used by CMPB (XCMP7 and XCMP8 are not used):

- For XCMP1-8 to be split between CMPA and CMPB, software must write XCMPCTL1[XCMPSPLIT] = 1
- For CMPA to only use XCMP1-3, software must write XCMPCTL1[CMPA_ALLOC] = 3
- For CMPB to only use XCMP5-6, software must write XCMPCTL1[CMPB_ALLOC] = 6

For XCMP1-3 in this scenario, since all are used by CMPA, the values written to XCMP1, XCMP2, and XCMP3 must:

- Without high-resolution edge placement requirement: XCMP(n+1) > (XCMPn) + 1
- With high-resolution edge placement requirement: XCMP(n+1) > (XCMPn) + 3

The requirements above for the minimum difference between XCMP(n+1) and XCMPn must be met in the application software.

The actions taken for each XCMP1-8 must be configured in XAQCTLA and XAQCTLB.

If shadowing is required then the XCMP1-8, XAQCTLA and XAQCTLB values must be written to the corresponding shadow buffer. As an example, Table 7-132 shows how the shadow buffers are used in LOADMULTIPLE mode.

The SHDW buffers 2 and 3 can also be repeated more than once by using the RPTBUF2PRD and RPTBUF3PRD.

Table 7-132. SHDW Buffer Loading Example

	XCMPn, XTBPRD			XTBPRD, TBPRD	XCMPn: XCMPnHR	CMPA: CMPAHR	What happens next?
	SHDW3FULL	SHDW2FULL	SHDW1FULL	Active	Active	Active	
CPU Initialization	Set	Set	Set				Registers initialized by CPU. Load event occurs.
ePWM Cycle 1	Clear	Set	Set	XTBPRD_ SHDW3	XCMPn_ SHDW3	XCMPn_ SHDW3	SHDWBUFPTR set to 3
ePWM Cycle 2	Clear	Clear	Set	XTBPRD_ SHDW2	XCMPn_ SHDW2	XCMPn_ SHDW2	SHDWBUFPTR set to 2
ePWM Cycle 3	Clear	Clear	Clear	XTBPRD_ SHDW1	XCMPn_ SHDW1	XCMPn_ SHDW1	SHDWBUFPTR set to 1
ePWM Cycle 4	Clear	Clear	Clear	XTBPRD_ SHDW1	XCMPn_ SHDW1	XCMPn_ SHDW1	SHDWBUFPTR set to 1 No shadow to active loading from buffer. Operation continues with values in XCMPn_ACTIVE registers.
ePWM Cycle 5	Clear	Clear	Clear	XTBPRD_ SHDW1	XCMPn_ SHDW1	XCMPn_ SHDW1	SHDWBUFPTR set to 1 Continues operation with same values in XCMPn_ACTIVE until the next buffer load event
CPU Load (During ePWM Cycle 5)	Set	Set	Set	XTBPRD_ SHDW1	XCMPn_ SHDW1	XCMPn_ SHDW1	CPU loads new shadow value set. Load event occurs. SHDWBUFPTR set to 3
ePWM Cycle 6	Clear	Set	Set	XTBPRD_ SHDW3	XCMPn_ SHDW3	XCMPn_ SHDW3	SHDWBUFPTR set to 3
ePWM Cycle 7	Clear	Clear	Set	XTBPRD_ SHDW2	XCMPn_ SHDW2	XCMPn_ SHDW2	SHDWBUFPTR set to 2
ePWM Cycle 8	Clear	Clear	Clear	XTBPRD_ SHDW1	XCMPn_ SHDW1	XCMPn_ SHDW1	SHDWBUFPTR set to 1
ePWM Cycle 9	Clear	Clear	Clear	XTBPRD_ SHDW1	XCMPn_ SHDW1	XCMPn_ SHDW1	Continues operation with the same values in XCMPn_ACTIVE until the next buffer load event. SHDWBUFPTR set to 1
ePWM Cycle 10	Set	Set	Set	XTBPRD_ SHDW1	XCMPn_ SHDW1	XCMPn_ SHDW1	CPU loads new shadow register set. Load event occurs. SHDWBUFPTR set to 3

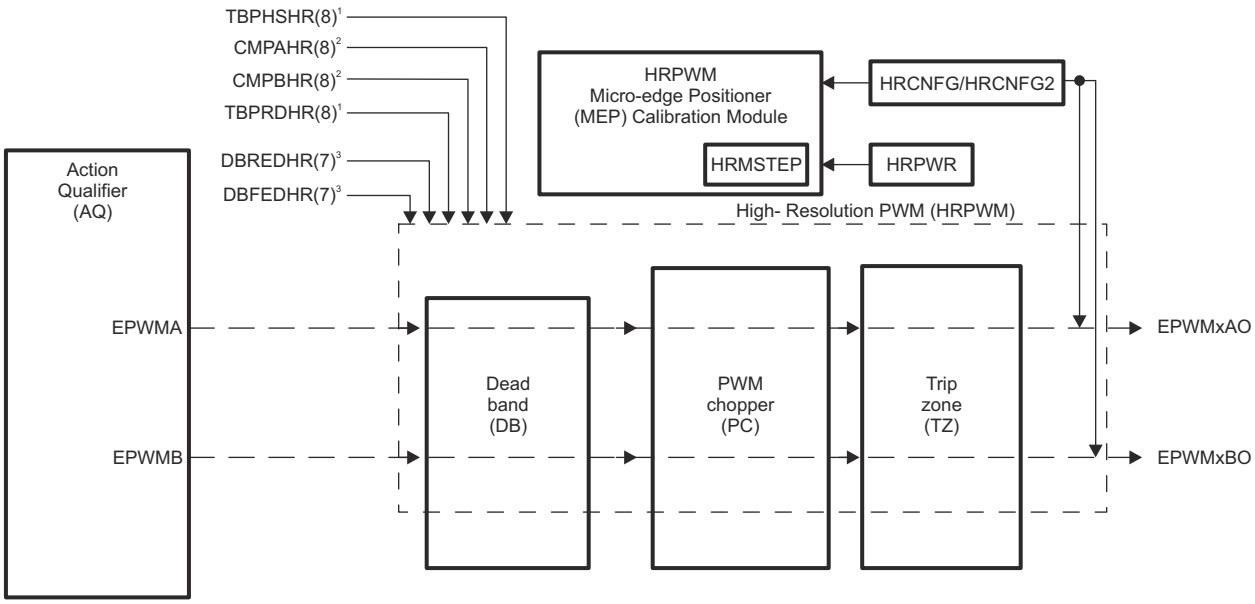
7.5.6.15 High-Resolution Pulse Width Modulator (HRPWM)

Figure 7-239 shows a block diagram of the HRPWM. This module extends the time resolution capabilities of the conventionally derived digital pulse width modulator (PWM). HRPWM is typically used when PWM resolution falls below approximately 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, Compare B and Phase registers
- Implemented using the A and B signal path of PWM, that is, on the EPWMxA and EPWMxB output
- Dead band high-resolution control for falling and rising edge delay in half cycle clocking operation
- Enables high-resolution output swapping on the EPWMxA and EPWMxB output
- Enables high-resolution output on EPWMxB signal output using inversion of EPWMxA signal output
- Enables high-resolution period, duty and phase control on the EPWMxA and EPWMxB output on devices with an ePWM module

Note

See the device data sheet to determine if your device has an ePWM module with high-resolution period support.



A. From ePWM Time-base (TB) submodule
B. From ePWM counter-compare (CC) submodule
C. From ePWM Deadband (DB) submodule

Figure 7-239. HRPWM Block Diagram

The ePWM peripheral is used to perform a function mathematically equivalent to a digital-to-analog converter (DAC). As shown in Figure 7-240, the effective resolution for conventionally generated PWM is a function of PWM frequency (or period) and system clock frequency.

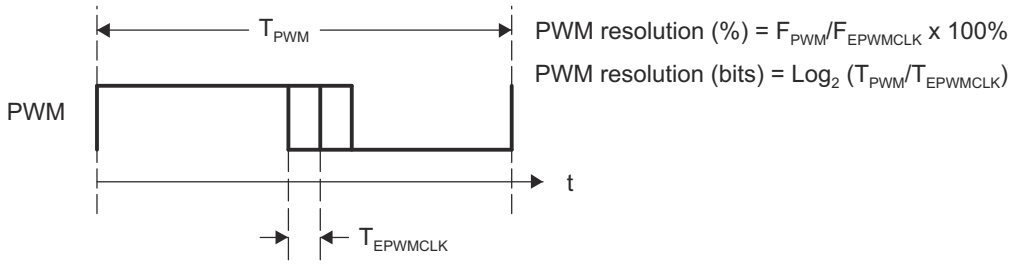


Figure 7-240. Resolution Calculations for Conventionally Generated PWM

If the required PWM operating frequency does not offer sufficient resolution in PWM mode, consider using HRPWM. As an example of improved performance offered by HRPWM, Table 7-133 shows resolution in bits for various PWM frequencies. These values assume a MEP step size of 180 ps. See the device data sheet for typical and maximum performance specifications for the MEP.

Table 7-133. Resolution for PWM and HRPWM

PWM Frequency (kHz)	Regular Resolution (PWM) 100 MHz EPWMCLK		High Resolution (HRPWM)	
	Bits	%	Bits	%
20	12.3	0.02	18.1	0.000
50	11	0.05	16.8	0.001
100	10	0.1	15.8	0.002
150	9.4	0.15	15.2	0.003
200	9	0.2	14.8	0.004
250	8.6	0.25	14.4	0.005
500	7.6	0.5	13.4	0.009
1000	6.6	1	12.4	0.018
1500	6.1	1.5	11.9	0.027
2000	5.6	2	11.4	0.036

Although each application can differ, typical low-frequency PWM operation (below 250 kHz) does 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
- Multiphase buck, boost, and flyback
- Phase-shifted full bridge
- Direct modulation of D-Class power amplifiers

7.5.6.15.1 Operational Description of HRPWM

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. See the device data sheet for the typical MEP step size on a particular device. The HRPWM also has a self-check software diagnostics mode to check if the MEP logic is running as designed under all operating conditions. Details on software diagnostics and functions are in Scale Factor Optimizing Software (SFO).

Figure 7-241 shows the relationship between one coarse system clock and edge position in terms of MEP steps, which are controlled using an 8-bit field in the Compare A extension register (CMPAHR). The same operating logic applies to CMPBHR as well.

To generate an HRPWM waveform, configure the ePWM registers to generate a conventional PWM of a given frequency and polarity. The HRPWM works together with the ePWM registers to extend edge resolution. Although many programming combinations are possible, only a few are needed and practical.

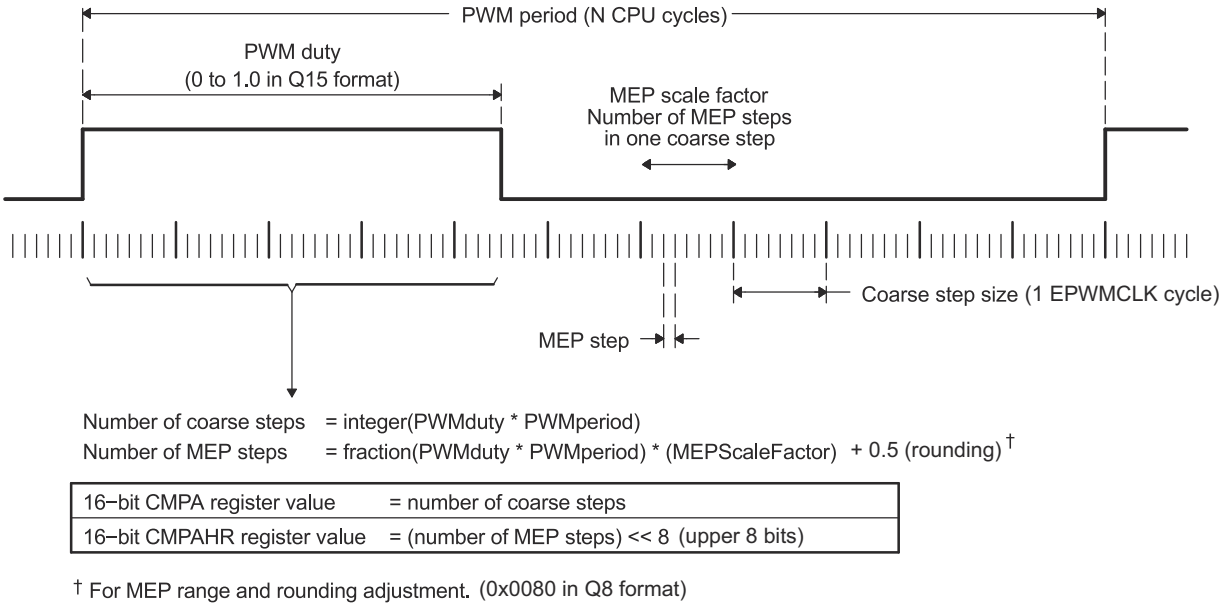
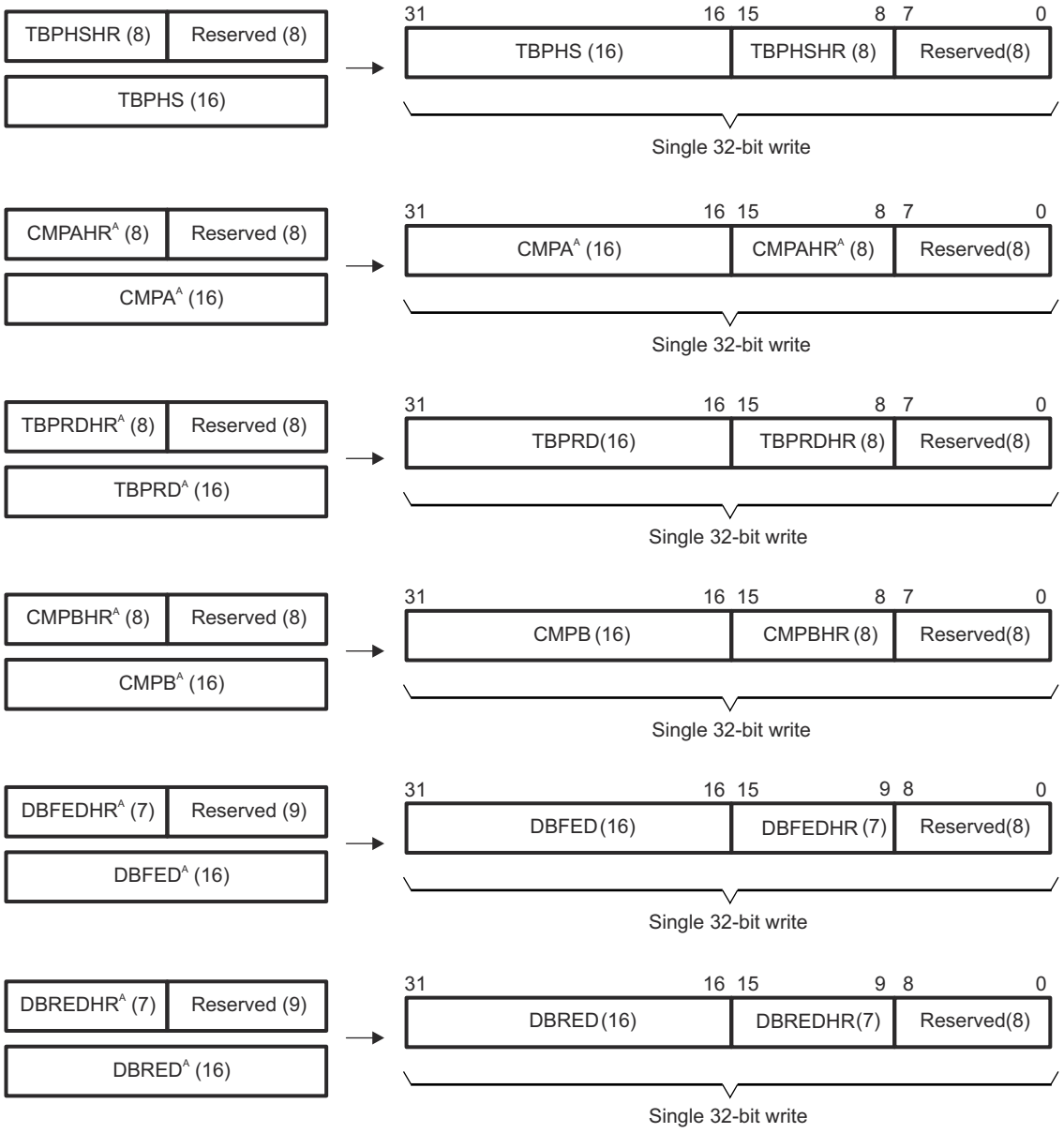


Figure 7-241. Operating Logic Using MEP

7.5.6.15.1.1 Controlling the HRPWM Capabilities

The MEP of the HRPWM is controlled by six extension registers. These HRPWM registers are concatenated with the 16-bit TBPHS, TBPRD, CMPA, CMPBM, DBREDM, and DBFEDM registers used to control PWM operation.

- TBPHSHR - Time Base Phase High Resolution Register
- CMPAHR - Counter Compare A High Resolution Register; CMPAHR is for use with the AQ output of Channel A, and is not related to CMPA
- TBPRDHR - Time Base Period High Resolution Register. (available on some devices)
- CMPBHR - Counter Compare B High Resolution Register; CMPBHR is for use with the AQ output of Channel B, and is not related to CMPB
- DBREDHR - Dead-band Generator Rising Edge Delay High Resolution Register
- DBFEDHR - Dead-band Generator Falling Edge Delay High Resolution Register



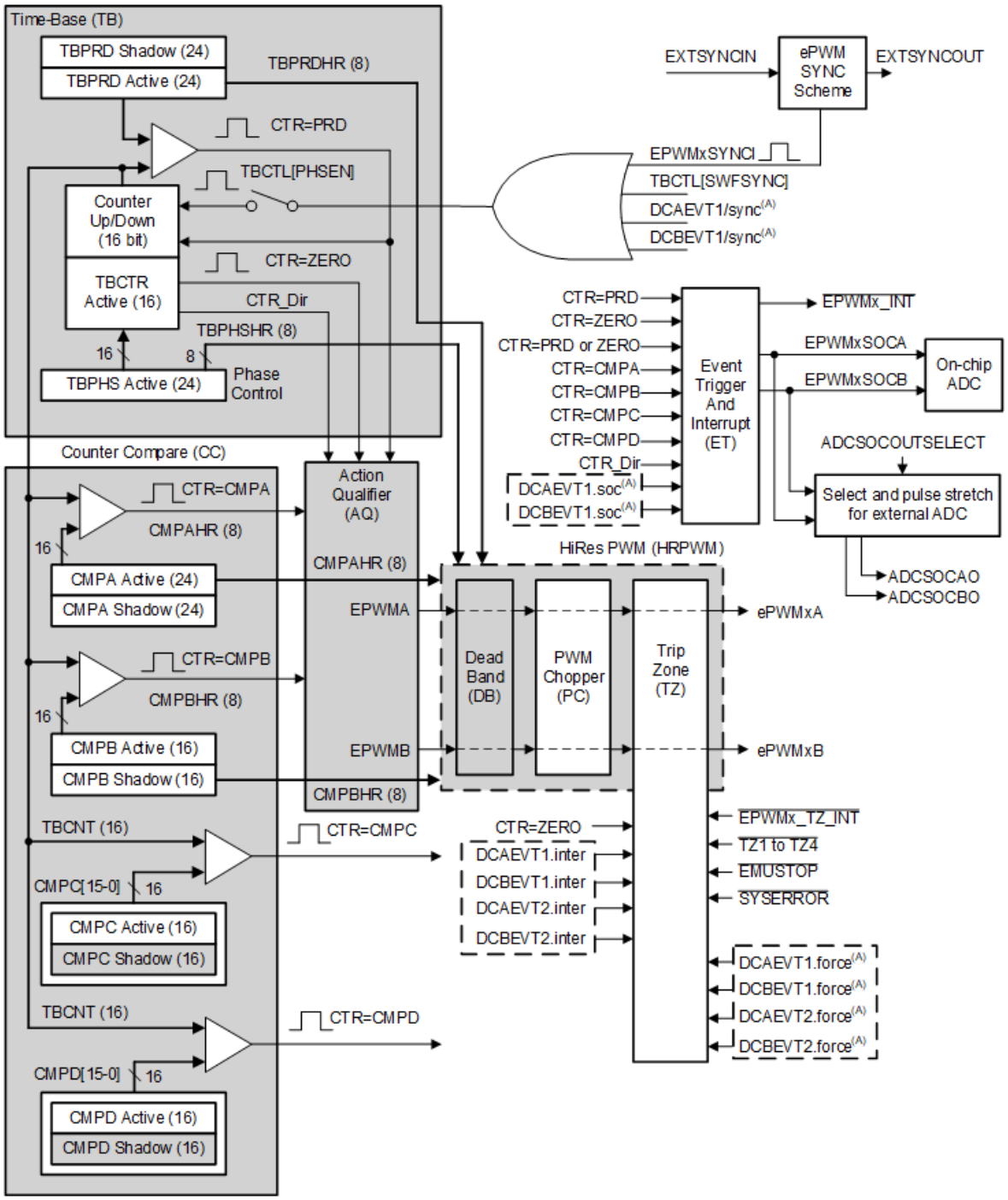
A. Dependent upon your device, these registers can be mirrored and can be written to at two different memory locations.

Figure 7-242. HRPWM Extension Registers and Memory Configuration

Note

HRPWM capabilities on Deadband Rising Edge Delay and Falling Edge Delay is applicable only during dead band half cycle clocking Operation. The number of MEP steps is half in size [bits 15:9] than duty and phase high-resolution registers for the same reason.

HRPWM capabilities are controlled using the Channel A and B PWM signal path. HRPWM support on the Dead band signal path is available by properly configuring the HRCNFG2 register. Figure 7-243 shows how the HRPWM interfaces with the 8-bit extension registers.



A. These events are generated by the ePWM Digital Compare (DC) submodule based on the levels of the TRIPIN inputs.

Figure 7-243. HRPWM System Interface

7.5.6.15.1.2 HRPWM Source Clock

Each HRPWM module is clocked from the respective EPWMxCLK. HRCAL has a separate clock. For example, HRPWM1 is sourced from EPWM1CLK while HRPWM2 is clocked from the EPWM2CLK. Figure 7-244 shows the HRCAL and HRPWM modules are sourced from their respective HRCAL and ePWM clock source.

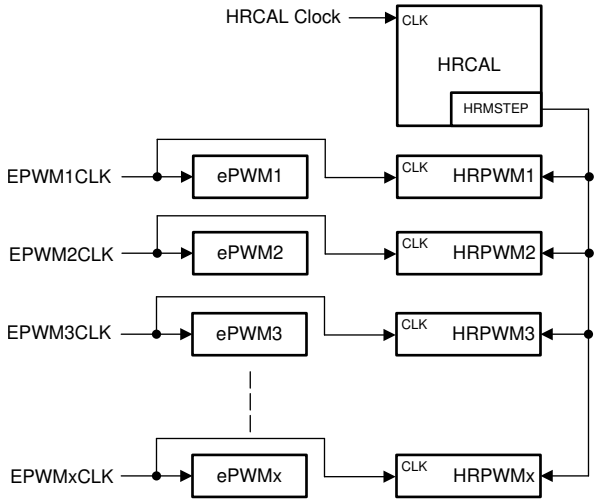


Figure 7-244. HRPWM and HRCAL Source Clock

7.5.6.15.1.3 Configuring the HRPWM

Once the ePWM has been configured to provide conventional PWM of a given frequency and polarity, the HRPWM is configured by programming the HRCNFG register in that particular ePWM module's register space. This register provides the following configuration options:

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 (CMPA or CMPB high-resolution control), while BE is used for topologies requiring phase shifting, for example, phase shifted full bridge (TBPHS or TBPRD high-resolution control).
Control Mode	The MEP is programmed to be controlled either from the CMPAHR/CMPBHR register in case of duty cycle control or the TBPHSHR register (phase control). RE or FE control mode can be used with the CMPAHR or CMPBHR register. BE control mode can be used with the TBPHSHR register. When the MEP is controlled from the TBPRDHR register (period control), the duty cycle and phase can also be controlled using their respective high-resolution registers.
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 CMPAHR, CMPBHR, and TBPRDHR registers and can be chosen to be the same as the regular load option for the CMPA/CMPB register. If TBPHSHR is used, then this option has no effect.
High-Resolution B Signal Control	The B signal path of an ePWM channel can generate a high-resolution output by outputting an inverted version of the high-resolution ePWMxA signal on the ePWMxB pin. HRPWM module can also enable high-resolution features on the B signal path independently of the A signal path as well.
Auto-conversion Mode	This mode is used in conjunction with the scale factor optimization (SFO) software only. For a type 4 HRPWM module, below is a description of the Auto-conversion Mode taking CMPAHR as an example. If auto-conversion is enabled, CMPAHR = fraction(PWMduty*PWMperiod)<<8. The scale factor optimization software calculates the MEP scale factor in the background code and automatically updates the HRMSTEP register with the calculated number of MEP steps per coarse step. The MEP Calibration Module then uses the values in the HRMSTEP and CMPAHR registers to automatically calculate the appropriate number of MEP steps represented

by the fractional duty cycle and moves the high-resolution ePWM signal edge accordingly. If auto-conversion is disabled, the CMPAHR register behaves like a type 0 HRPWM module and CMPAHR = (fraction(PWMduty * PWMperiod) * MEP Scale Factor + 0.5)<<8). All calculations need to be performed by your code in this mode, and the HRMSTEP register is ignored. Auto-conversion for high-resolution period has the same behavior as auto-conversion for high-resolution duty cycle. Auto-conversion must always be enabled for high-resolution period mode.

Note

If the HRPWM module is configured in UP-DOWN counter mode, the shadow mode for the HRPWM registers must be set to load on both ZERO AND PERIOD. New values from the user are loaded to the shadow registers only at CTR=ZERO, but the shadow mode of for the registers must be set to both ZERO AND PERIOD. The CTR=PRD event is used for specific internal logic inside the HRPWM module.

Auto-conversion Mode performs the calculation for CMPBHR , DBREDHR, and DBFEDHR. The scale factor optimization software calculates the MEP scale factor in the background code and automatically updates the HRMSTEP register with the calculated number of MEP steps per coarse step. The MEP Calibration Module then uses the values in the HRMSTEP and CMPBHR or DBREDHR/DBFEDHR register to automatically calculate the appropriate number of MEP steps represented by the fractional components and moves the high-resolution ePWM signal edge accordingly. If auto-conversion is disabled, CMPBHR behaves the same as CMPAHR. CMPBHR = (fraction(PWMduty * PWMperiod) * MEP Scale Factor + 0.5)<<8).

Linking CMPBHR to CMPAHR

Starting with EPWM Type 5, the user has the option to link the CMPBHR value to the CMPAHR value. This allows for EPWM channel A and EPWM channel B outputs to both be controlled by CMPAHR. This feature is enabled through CMPCTL.LINKDUTYHR register. This feature is commonly used when the HRPWM is configured for complimentary output mode.

7.5.6.15.1.4 Configuring High-Resolution in Deadband Rising-Edge and Falling-Edge Delay

Once the ePWM has been configured to provide conventional PWM of a given frequency, polarity, and dead band enabled in half-cycle clocking mode, the high-resolution operation on dead band RED and FED lines are enabled by programming the HRCNFG2 register in that particular ePWM module register space. This register provides the following configuration options:

Edge Mode The MEP can be programmed to provide precise position control on the dead band rising edge (RED), dead band falling edge (FED), or both edges (rising edge of DBRED signal and falling edge of DBFED signal) at the same time.

Control Mode Selects the time event that loads the shadow value in the active register for DBRED and DBFED in high-resolution mode. Select the pulse to match the selection in the ePWM DBCTL[LOADREDMODE] and DBCTL[LOADFEDMODE] bits.

7.5.6.15.1.5 Principle of Operation

The MEP logic is capable of placing an edge in one of 255 (8 bits) discrete time steps (see the device data sheet for typical MEP step size). The MEP works with the TBM and CCM registers to be certain that time steps are applied and that edge placement accuracy is maintained over a wide range of PWM frequencies, system clock frequencies, and other operating conditions. Table 7-134 shows the typical range of operating frequencies supported by the HRPWM.

Table 7-134. Relationship Between MEP Steps, PWM Frequency, and Resolution

System (MHz)	MEP Steps Per EPWMCLK (1) (2) (3)	PWM Minimum (Hz) (4)	PWM Maximum (MHz)	Resolution at Maximum (Bits) (5)
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) TBCLK = EPWMCLK.
(2) Table data based on a MEP time resolution of 180 ps (this is an example value. See the device data sheet for MEP limits)
(3) MEP steps applied = $T_{EPWMCLK}$/180 ps in this example.
(4) PWM minimum frequency is based on a maximum period value,(TBPRD = 65535). PWM mode is asymmetrical up-count.
(5) Resolution in bits is given for the maximum PWM frequency stated.

7.5.6.15.1.5.1 Edge Positioning

Note

The following example is presented using the [CMPA:CMPAHR] register combination. The theory of operation and equations are the same, if intending to use the [CMPBM:CMPBHRM] for duty cycle control.

In a typical power control loop, a digital controller issues a duty command, usually expressed in a per unit or percentage terms. 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%. As shown in Figure 7-245, a compare value of 32 counts (duty = 40%) is the closest to 40.5% that can be attained. This is equivalent to an edge position of 320 ns instead of the desired 324 ns. This data is shown in Table 7-135.

By utilizing the MEP, an edge position much closer to the desired point of 324 ns can be achieved. Table 7-135 shows that in addition to the CMPA value, 22 steps of the MEP (CMPAHR register) positions 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 ps.

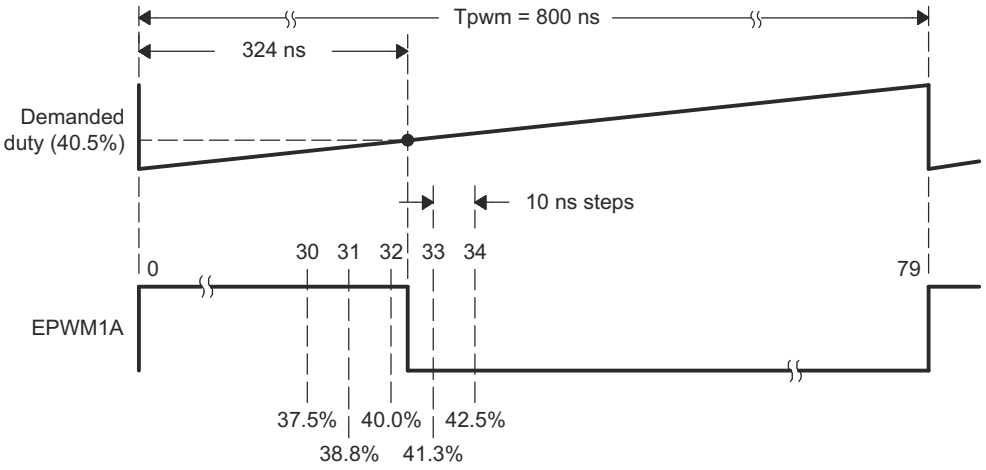


Figure 7-245. Required PWM Waveform for a Requested Duty = 40.5%

Table 7-135. CMPA versus Duty (left), and [CMPA:CMPAHR] versus 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) Assumed MEP step size for the above example = 180 ps. See the device-specific data sheet for typical and maximum MEP values.
(2) TBCLK = 100 MHz, 10 ns
(3) For a PWM Period register value of 80 counts, PWM Period = 80 × 10 ns = 800 ns, PWM frequency = 1/800 ns = 1.25 MHz

7.5.6.15.1.5.2 Scaling Considerations

The mechanics of how to position an edge precisely in time has been demonstrated using the resources of the standard CMPA and MEP (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 nanoseconds (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:

TBCLK	=	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 n/180 ps), **MEP_ScaleFactor**	=	55
Value to keep CMPAHR within the range of 1-255 and fractional rounding constant (default value)	=	0.5 (0080h in Q8 format)

Step 1: Percentage Integer Duty value conversion for CMPA register

CMPA register value	=	int(**PWMDuty*****PWMPeriod**); int means integer part
	=	int(0.405 * 80)
	=	int(32.4)
CMPA register value	=	32 (20h)

Step 2: Fractional value conversion for CMPAHR register

CMPAHR	=	(frac(**PWMDuty*****PWMPeriod**)***MEP_ScaleFactor** + 0.5) <<8); frac means fractional part
	=	(frac(32.4) * 55 + 0.5) <<8; Shifting is to move the value to the high byte of CMPAHR.
	=	(0.4 * 55 + 0.5) <<8
	=	(22 + 0.5) <<8
	=	22.5 * 256; Shifting left by 8 is the same as multiplying by 256.
	=	5760 (1680h)
CMPAHR	=	1680h CMPAHR value = 1600h (lower 8 bits are ignored by hardware).

Note

If the AUTOCONV bit (HRCNFG.6) is set and the MEP_ScaleFactor is in the HRMSTEP register, then CMPAHR / CMPBHR register value = frac (PWMDuty*PWMperiod<<8). The rest of the conversion calculations are performed automatically in hardware, and the correct MEP-scaled signal edge appears on the ePWM channel output. If AUTOCONV is not set, the above calculations must be performed by software.

The MEP scale factor (MEP_ScaleFactor) varies with the system clock and DSP operating conditions. TI provides an MEP scale factor optimizing (SFO) software C function, which uses the built in diagnostics in each HRPWM and returns the best scale factor for a given operating point.

The scale factor varies slowly over a limited range so the optimizing C function can be run very slowly in a background loop.

The CMPA, CMPB, CMPAHR and CMPBHR registers are configured in memory so that the 32-bit data capability of the CPU can write this as a single concatenated value, that is, [CMPA:CMPAHR], [CMPB:CMPBHR], and so on.

The mapping scheme has been implemented in C, and the actual implementation takes advantage of the 32-bit CPU architecture and examples are provided in the Section 7.5.6.19.

7.5.6.15.1.5.3 Duty Cycle Range Limitation

In high-resolution mode, the MEP is not active for 100% of the PWM period and becomes operational:

- Three EPWMCLK cycles after the period starts when high-resolution period (TBPRDHR) control is not enabled.
- When high-resolution period (TBPRDHR) control is enabled using the HRPCTL register:
  - In up-count mode: three EPWMCLK cycles after the period starts until three EPWMCLK cycles before the period ends.
  - In up-down count mode: when counting up, three cycles after CTR = 0 until three cycles before CTR = PRD, and when counting down, three cycles after CTR = PRD until three cycles before CTR = 0.
- When using DBREDHR or DBFEDHR, DBRED or DBFED (the register corresponding to the edge with high-resolution displacement) must be greater than or equal to 7.

Duty cycle range limitations are illustrated in Figure 7-246 to Figure 7-249. This limitation imposes a duty cycle limit on the MEP. For example, precision edge control is not available all the way down to 0% duty cycle. When high-resolution period control is disabled, regular PWM duty control is fully operational down to 0% duty cycle despite the unavailability of HRPWM features in the first three cycles. In most applications, this cannot be an issue as the controller regulation point is usually not designed to be close to 0% duty cycle. To better understand the useable duty cycle range, see Table 7-136. When high-resolution period control is enabled (HRPCTL[HRPE]=1), the duty cycle must not fall within the restricted range; otherwise, there can be undefined behavior on the ePWMxA output.

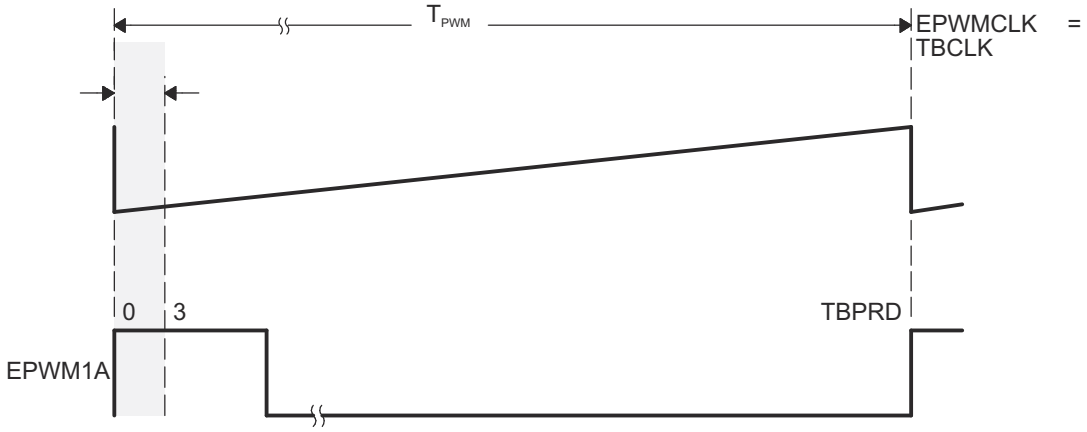


Figure 7-246. Low % Duty Cycle Range Limitation Example (HRPCTL[HRPE] = 0)

Table 7-136. Duty Cycle Range Limitation for Three EPWMCLK/TBCLK Cycles

PWM Frequency(1) (kHz)	3 Cycles Minimum Duty	3 Cycles Maximum Duty(2)
200	0.6%	99.4%
400	1.2%	98.8%
600	1.8%	98.2%
800	2.4%	97.6%
1000	3%	97%
1200	3.6%	96.4%
1400	4.2%	95.8%
1600	4.8%	95.2%
1800	5.4%	94.6%
2000	6%	94%

(1) EPWMCLK = TBCLK = 100 MHz
(2) This limitation applies only if high-resolution period (TBPRDHR) control is enabled.

If the application demands HRPWM operation below the minimum duty cycle limitation, then the HRPWM can be configured to operate in count-down mode with the rising edge position (REP) controlled by the MEP when high-resolution period is disabled (HRPCTL[HRPE] = 0). This is illustrated in Figure 7-247. In this configuration, the minimum duty cycle limitation is no longer an issue. However, there is a maximum duty limitation with same percent numbers as given in Table 7-136.

CAUTION

If the application has enabled high-resolution period control (HRPCTL[HRPE]=1), the duty cycle must not fall within the restricted range; otherwise, there can be undefined behavior on the ePWM output.

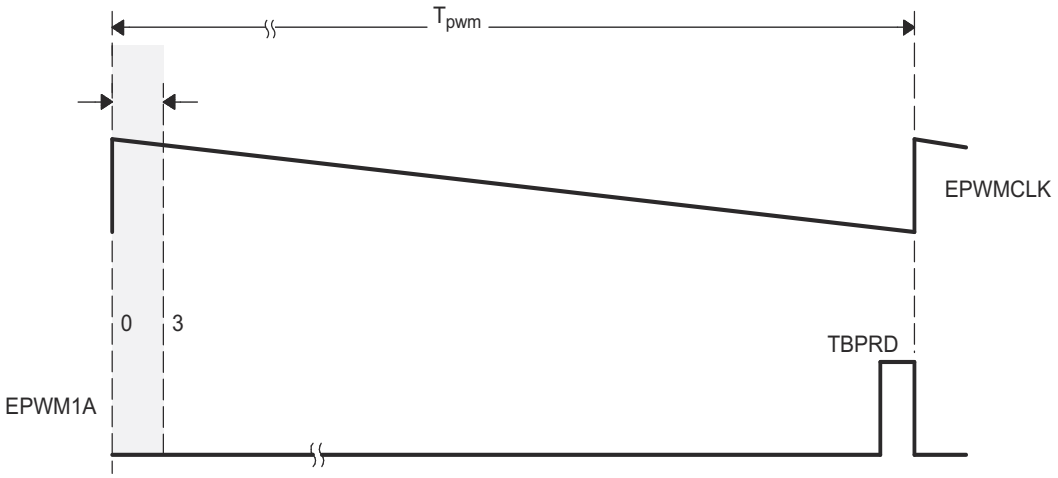


Figure 7-247. High % Duty Cycle Range Limitation Example (HRPCTL[HRPE] = 0)

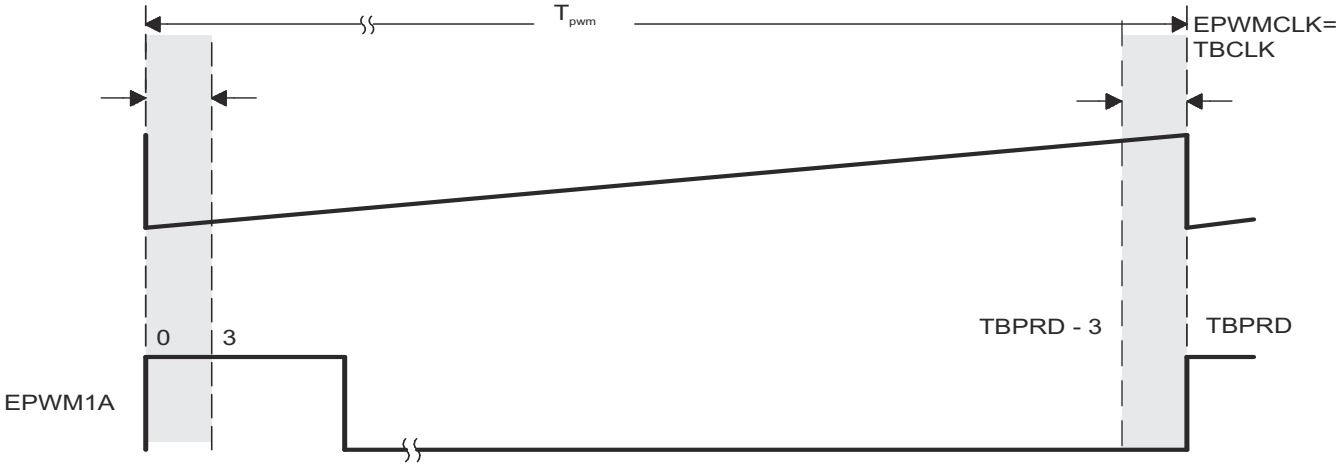


Figure 7-248. Up-Count Duty Cycle Range Limitation Example (HRPCTL[HRPE]=1)

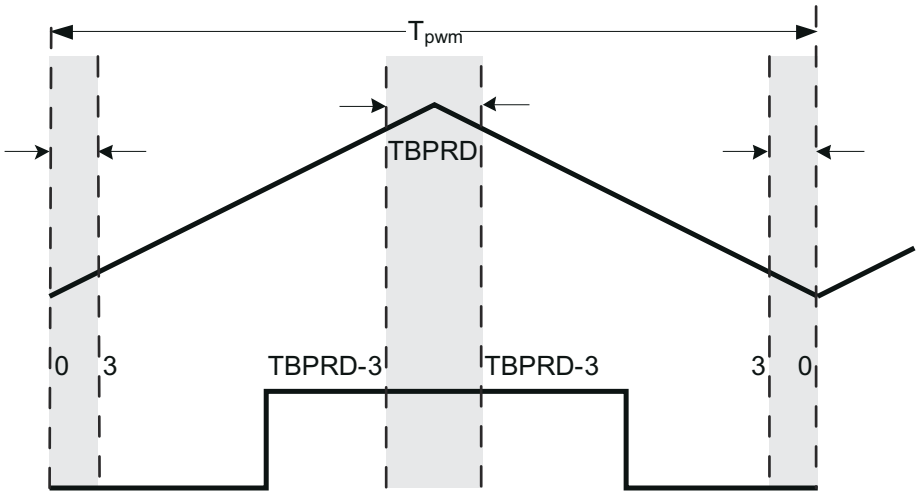


Figure 7-249. Up-Down Count Duty Cycle Range Limitation Example (HRPCTL[HRPE]=1)

7.5.6.15.1.5.4 High-Resolution Period

High-resolution period control using the MEP logic is supported on devices with a Type 1 ePWM module or greater.

Note

When high-resolution period control is enabled, on ePWMxA only, and not ePWMxB output and conversely, the non high-resolution output has ±1 TBCLK cycle jitter in up-count mode and ±2 TBCLK cycle jitter in up-down count mode.

The scaling procedure described for duty cycle in Section 7.5.6.15.1.5.2 applies for high-resolution period as well:

Assumptions for this example:

TBCLK	= 10 ns (100 MHz)
Required PWM frequency	= 175 kHz (period of 571.428)
Number of MEP steps per coarse step at 180 ps (MEP_ScaleFactor)	= 55 (10 ns / 180 ps)
Value to keep TBPRDHR within range of 1-255 and fractional rounding constant (default value)	= 0.5 (0080h in Q8 format)

Problem:

In up-count mode:

- If TBPRD = 571, then PWM frequency = 174.82 kHz (period = (571+1)* T_{TBCLK}).
- If TBPRD = 570, then PWM frequency = 175.13 kHz (period = (570+1)* T_{TBCLK}).

In up-down count mode:

- If TBPRD = 286, then PWM frequency = 174.82 kHz (period = (286*2)* T_{TBCLK}).
- If TBPRD = 285, then PWM frequency = 175.44 kHz (period = (285*2)* T_{TBCLK}).

Solution:

With 55 MEP steps per coarse step at 180 ps each:

Step 1: Percentage Integer Period value conversion for TBPRD register

Integer period value	= 571 * T_{TBCLK}
	= int (571.428) * T_{TBCLK}
	= int (PWMperiod) * T_{TBCLK}
In up-count mode:	
TBPRD	= 570 (TBPRD = period value - 1)
	= 023Ah
In up-down count mode:	= 285 (TBPRD = period value / 2)
TBPRD	= 011Dh

Step 2: Fractional value conversion for TBPRDHR register

In up-count mode:

TBPRDHR register value	= (frac(PWMperiod) * MEP_ScaleFactor + 0.5)
If auto-conversion enabled and HRMSTEP =	
MEP_ScaleFactor value (55):	=frac (PWMperiod) << 8 (Shifting is to move the value to the high byte of TBPRDHR)
TBPRDHR register value	=frac (571.428) << 8
	=0.428 × 256
	=6D00h
The auto-conversion then automatically performs the calculation, such that TBPRDHR MEP delay is scaled by hardware to:	=((TBPRDHR(15:0) >> 8) × HRMSTEP + 80h) << 8
	= (006Dh × 55 + 80h) >> 8
	=(17EBh) >> 8
Period MEP delay	=0017h MEP Steps

In up-down count mode:

TBPRDHR register value	= (frac(PWMperiod) * MEP_ScaleFactor + 0.5)
If auto-conversion enabled and HRMSTEP =	
MEP_ScaleFactor value (55):	=frac (PWMperiod / 2) << 8 (Shifting is to move the value to the high byte of TBPRDHR)
TBPRDHR register value	=frac (285.714) << 8
	=0.714 × 256
	=B600h
The auto-conversion then automatically performs the calculation, such that TBPRDHR MEP delay is scaled by hardware to:	=((TBPRDHR(15:0) >> 8) × HRMSTEP + 80h) << 8
	= (00B6h × 55 + 80h) >> 8
	=(279Ah) >> 8
Period MEP delay	=0027h MEP Steps

7.5.6.15.1.5.4.1 High-Resolution Period Configuration

To use high-resolution period, the ePWMx module must be initialized in the exact order presented.

The following steps use CMPA with shadow registers and the corresponding HRCNFG bits for high-resolution operation on EPWMxA. For high-resolution operation on EPWMxB, make the appropriate substitutions with the B channel fields.

1. Enable ePWMx clock
2. Enable HRPWM clock
3. Disable EPWM_SYNC
4. Configure ePWMx registers - AQ, TBPRD, CC, and so on.
   - ePWMx can only be configured for up-count or up-down count modes. High-resolution period is not compatible with down-count mode.
   - TBPRD and CC registers must be configured for shadow loads.
   - CMPCTL[LOADAMODE]
     - In up-count mode: CMPCTL[LOADAMODE] = 1 (load on CTR = PRD)
     - In up-down count mode: CMPCTL[LOADAMODE] = 2 (load on CTR=0 or CTR=PRD)
5. Configure the HRCNFG register such that:
   - HRCNFG[HRLOAD] = 2 (load on either CTR = 0 or CTR = PRD)
   - HRCNFG[AUTOCONV] = 1 (Enable auto-conversion)
   - HRCNFG[EDGMODE] = 3 (MEP control on both edges)
6. For TBPHS:TBPHSHR synchronization with high-resolution period, set both HRPCTL[TBPSHRLOADE] = 1 and TBCTL[PHSEN] = 1. In up-down count mode these bits must be set to 1 regardless of the contents of TBPHSHR.
7. Enable high-resolution period control (HRPCTL[HRPE] = 1)
8. Enable EPWM_CLKSYNC
9. TBCTL[SWFSYNC] = 1
10. HRMSTEP must contain an accurate MEP scale factor (# of MEP steps per EPWMCLK coarse step) because auto-conversion is enabled. The MEP scale factor can be acquired using the SFO() function.
11. To control high-resolution period, write to the TBPRDHR(M) registers.

Note

When high-resolution period mode is enabled, an EPWMxSYNC pulse introduces ±1-2 cycle jitter to the PWM (±1 cycle in up-count mode and ±2 cycle in up-down count mode). For this reason, EPWMxSYNCO source cannot be set to CTR = 0 or CTR = CMPB. Otherwise, the jitter occurs on every PWM cycle with the synchronization pulse.

When EPWMxSYNCI is EPWMxSYNCO source, a software synchronization pulse can be issued only once during high-resolution period initialization. If a software sync pulse is applied while the PWM is running, the jitter appears on the PWM output at the time of the sync pulse.

7.5.6.15.1.6 Deadband High-Resolution Operation

Note

In up-count mode, the dead-band module is not available when any high-resolution mode is enabled.

Assumptions for this example:

System clock	= 10 ns (100 MHz)
Deadband enabled in half-cycle mode, TBCLK = EPWMCLK	
Required PWM frequency	1.33 MHz (1 / 750 ns)
Required PWM duty cycle	0.5 (50%)
Required Deadband Rising Edge Delay	5% over duty
Required Deadband Rising Edge Delay in ns	(0.05 * 375 ns) = 18.75 ns

Note

Similar to the duty cycle restrictions when using HRPWM, the DBRED and DBFED values must be greater than 3 to use high-resolution deadband.

Deadband delay values as a function of DBFED and DBRED:

When half-cycle clocking is enabled, the formula to calculate the falling-edge-delay and rising-edge-delay becomes:

FED = DBFED * TBCLK / 2

RED = DBRED * TBCLK / 2

DBRED and DBFED calculated values:

Required Dead band Rising Edge Delay in ns = 18.75 ns

DBRED = RED / (TBCLK / 2)

DBRED = 18.75 ns / 5 ns

DBRED Required = 3.75 ns

With 55 MEP steps per coarse step at 180 ps each:

Step 1: Integer Deadband value conversion for DBREDM register

Integer DBRED value	= int (RED / (TBCLK / 2))
	= int (3.75)
DBRED	= 3

Step 2: Fractional value conversion for Deadband high-resolution register DBREDHR

DBREDHR register value	= (frac(DBRED Required) * MEP_ScaleFactor + 0.5) << 8 (Shifting is to move the value to the high byte of DBREDHR)
	= (frac (3.75) * 55 + 0.5) << 8
	= (0.75 * 55 + 0.5) << 8
	= (41.75) * 256 Shifting left by 8 is the same as multiplying by 256.
DBREDHR value	= 29C0h MEP Steps
	Hardware ignores lower 9 bits in the above calculated DBREDHR value

Note

If the AUTOCONV bit (HRCNFG.6) is set and the MEP_ScaleFactor is in the HRMSTEP register, then DBREDHR:DBRED = frac((required DB value) < <8). The rest of the conversion calculations are performed automatically in hardware, and the correct MEP-scaled signal edge appears on the ePWM channel output. If AUTOCONV is not set, the above calculations must be performed by software.

7.5.6.15.1.7 Scale Factor Optimizing Software (SFO)

The micro edge positioner (MEP) logic is capable of placing an edge in one of 255 discrete time steps. As previously mentioned, the size of these steps is on the order of 150 ps (see the device data sheet for typical MEP step size on your device). The MEP step size varies based on worst-case process parameters, operating temperature, and voltage. MEP step size increases with decreasing voltage and increasing temperature and decreases with increasing voltage and decreasing temperature. Applications that use the HRPWM feature can use the TI-supplied MEP scale factor optimization (SFO) software function. The SFO function helps to dynamically determine the number of MEP steps per EPWMCLK period while the HRPWM is in operation.

To utilize the MEP capabilities effectively, the correct value for the MEP scaling factor needs to be known by the software. To accomplish this, the HRPWM module has built in self-check and diagnostic capabilities that can be used to determine the optimum MEP scale factor value for any operating condition. TI provides a C-callable library containing one SFO function that utilizes this hardware and determines the optimum MEP scale factor. As such, MEP control and diagnostics registers are reserved for TI use.

A detailed description of the SFO library and examples are listed in Section 7.5.6.19.

7.5.6.15.1.8 HRPWM Examples Using Optimized Assembly Code

The best way to understand how to use the HRPWM capabilities is through two real examples:

1. Simple buck converter using asymmetrical PWM (count-up) with active high polarity.
2. DAC function using simple R+C reconstruction filter.

The following examples all have initialization and configuration code written in C. To make these easier to understand, the #defines shown below are used.

Example 7-2. #Defines for HRPWM Header Files

```
// HRPWM (High Resolution PWM) //
================================
// HRCNFG
#define HR_Disable 0x0
#define HR_REP 0x1          // Rising Edge position
#define HR_FEP 0x2          // Falling Edge position
#define HR_BEP 0x3          // Both Edge position #define HR_CMP 0x0 // CMPAHR controlled
#define HR_PHS 0x1          // TBPHSHR controlled #define HR_CTR_ZERO 0x0 // CTR = Zero event
#define HR_CTR_PRD 0x1      // CTR = Period event
#define HR_CTR_ZERO_PRD 0x2 // CTR = ZERO or Period event
#define HR_NORM_B  0x0      // Normal ePWMxB output
#define HR_INVERT_B 0x1     // ePWMxB is inverted ePWMxA output
```

7.5.6.15.1.8.1 Implementing a Simple Buck Converter

In this example, the PWM requirements are:

- PWM frequency = 1 MHz (that is, TBPRD = 100)
- PWM mode = asymmetrical, up-count
- Resolution = 12.7 bits (with a MEP step size of 150 ps)

Figure 7-250 and Figure 7-251 show the required PWM waveform. As explained previously, configuration for the ePWM1 module is almost identical to the normal case except that the appropriate MEP options need to be enabled/selected.

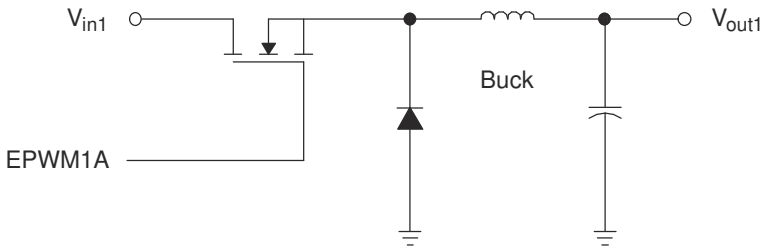


Figure 7-250. Simple Buck Controlled Converter Using a Single PWM

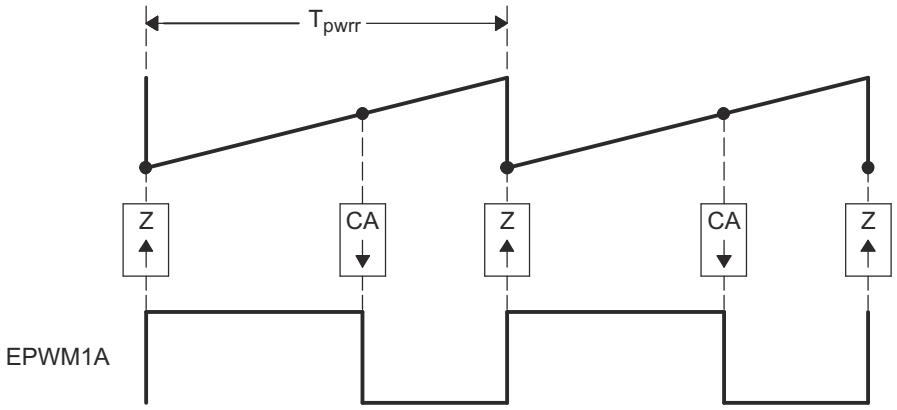


Figure 7-251. PWM Waveform Generated for Simple Buck Controlled Converter

The example code shown consists of two main parts:

- Initialization code (executed once)
- Run time code (typically executed within an ISR)

Example 7-3 shows the Initialization code. The first part is configured for conventional PWM. The second part sets up the HRPWM resources.

This example assumes MEP step size of 150 ps and does not use the SFO library.

Example 7-4 shows an assembly example of run-time code for the HRPWM buck converter.

Example 7-3. HRPWM Buck Converter Initialization Code

```
void HrBuckDrvCnf(void)
 {
// Config for conventional PWM first
EPwm1Regs.TBCTL.bit.PRDLD = TB_IMMEDIATE;         // set Immediate load
EPwm1Regs.TBPRD = 100;                            // Period set for 1000 kHz PWM
hrbuck_period = 200;                              // Used for Q15 to Q0 scaling
EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP;
EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE;           // EPWM1 is the Sync Source

EPwm1Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;
EPwm1Regs.TBCTL.bit.CLKDIV = TB_DIV1;
// Note: ChB is initialized here only for comparison purposes, it is not required

EPwm1Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;
EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
EPwm1Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;     // optional
EPwm1Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;       // optional
EPwm1Regs.AQCTLA.bit.ZRO = AQ_SET;
EPwm1Regs.AQCTLA.bit.CAU = AQ_CLEAR;
EPwm1Regs.AQCTLB.bit.ZRO = AQ_SET;                // optional
EPwm1Regs.AQCTLB.bit.CBU = AQ_CLEAR;              // optional
// Now configure the HRPWM resources
EALLOW;                                           // Note these registers are protected
                                                  // and act only on ChA
EPwm1Regs.HRCNFG.all = 0x0;                       // clear all bits first
EPwm1Regs.HRCNFG.bit.EDGMODE = HR_FEP;            // Control Falling Edge Position
EPwm1Regs.HRCNFG.bit.CTLMODE = HR_CMP;            // CMPAHR controls the MEP
EPwm1Regs.HRCNFG.bit.HRLOAD = HR_CTR_ZERO;        // Shadow load on CTR=Zero
EDIS;
MEP_ScaleFactor = 66*256;                         // Start with typical Scale Factor
                                                  // value for 100 MHz
                                                  // Note: Use SFO functions to update
                                                     MEP_ScaleFactor dynamically
}
```

Example 7-4. HRPWM Buck Converter Run-Time Code

```
EPWM1_BASE .set 0x6800
CMPAHR1 .set EPWM1_BASE+0x8
;=============================================
HRBUCK_DRV; (can execute within an ISR or loop)
;=============================================
     MOVW DP, #_HRBUCK_In
     MOVL XAR2,@_HRBUCK_In       ; Pointer to Input Q15 Duty (XAR2)
     MOVL XAR3,#CMPAHR1          ; Pointer to HRPWM CMPA reg (XAR3)

; Output for EPWM1A (HRPWM)
     MOV T,*XAR2 ; T <= Duty
     MPYU ACC,T,@_hrbuck_period  ; Q15 to Q0 scaling based on Period
     MOV T,@_MEP_ScaleFactor     ; MEP scale factor (from optimizer s/w)
     MPYU P,T,@AL                ; P <= T * AL, Optimizer scaling
     MOVH @AL,P                  ; AL <= P, move result back to ACC
     ADD ACC, #0x080             ; MEP range and rounding adjustment
     MOVL *XAR3,ACC              ; CMPA:CMPAHR(31:8) <= ACC

; Output for EPWM1B (Regular Res) Optional - for comparison purpose only
     MOV *+XAR3[2],AH            ; Store ACCH to regular CMPB
```

7.5.6.15.1.8.2 Implementing a DAC Function Using an R+C Reconstruction Filter

In this example, the PWM requirements are:

- PWM frequency = 400 kHz (that is, TBPRD = 250)
- PWM mode = Asymmetrical, Up-count
- Resolution = 14 bits (MEP step size = 150 ps)

Figure 7-252 and Figure 7-253 show the DAC function and the required PWM waveform. As explained previously, configuration for the ePWM1 module is almost identical to the normal case except that the appropriate MEP options need to be enabled/selected.

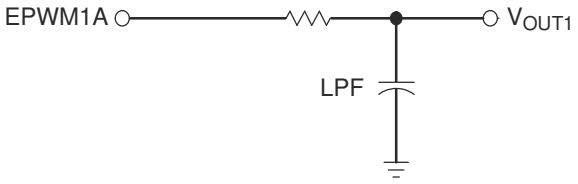


Figure 7-252. Simple Reconstruction Filter for a PWM-based DAC

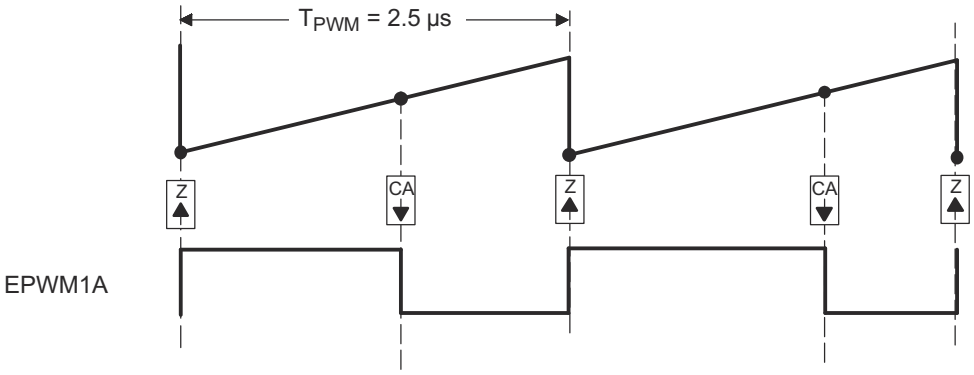


Figure 7-253. PWM Waveform Generated for the PWM DAC Function

The example code shown consists of two main parts:

- Initialization code (executed once)
- Run time code (typically executed within an ISR)

This example assumes a typical MEP_SP and does not use the SFO library.

Example 7-5 shows the Initialization code. The first part is configured for conventional PWM. The second part sets up the HRPWM resources.

Example 7-6 shows an assembly example of run-time code that can execute in a high-speed ISR loop.

Example 7-5. PWM DAC Function Initialization Code

```
void HrPwmDacDrvCnf(void)
{
// Config for conventional PWM first
EPwm1Regs.TBCTL.bit.PRDLD = TB_IMMEDIATE;      // Set Immediate load
EPwm1Regs.TBPRD = 250;                         // Period set for 400 kHz PWM
hrDAC_period = 250;                            // Used for Q15 to Q0 scaling
EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP;
EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE;        // EPWM1 is the Sync Source

EPwm1Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;
EPwm1Regs.TBCTL.bit.CLKDIV = TB_DIV1;
// Note: ChB is initialized here only for comparison purposes, it is not required

EPwm1Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;
EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
EPwm1Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;  // optional
EPwm1Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;    // optional

EPwm1Regs.AQCTLA.bit.ZRO = AQ_SET;
EPwm1Regs.AQCTLA.bit.CAU = AQ_CLEAR;
EPwm1Regs.AQCTLB.bit.ZRO = AQ_SET;             // optional
EPwm1Regs.AQCTLB.bit.CBU = AQ_CLEAR;           // optional
// Now configure the HRPWM resources
EALLOW;                                        // Note these registers are protected
                                               // and act only on ChA.
EPwm1Regs.HRCNFG.all = 0x0;                    // Clear all bits first
EPwm1Regs.HRCNFG.bit.EDGMODE = HR_FEP;         // Control falling edge position
EPwm1Regs.HRCNFG.bit.CTLMODE = HR_CMP;         // CMPAHR controls the MEP.
EPwm1Regs.HRCNFG.bit.HRLOAD = HR_CTR_ZERO;     // Shadow load on CTR=Zero.
EDIS;
MEP_ScaleFactor = 66*256;                      // Start with typical Scale Factor
                                               // value for 100 MHz.
                                               // Use SFO functions to update MEP_ScaleFactor
                                               // dynamically.
}
```

Example 7-6. PWM DAC Function Run-Time Code

```
EPWM1_BASE .set 0x6800
CMPAHR1 .set EPWM1_BASE+0x8
;================================================
HRPWM_DAC_DRV; (can execute within an ISR or loop)
;================================================
      MOVW DP, #_HRDAC_In
      MOVL XAR2,@_HRDAC_In             ; Pointer to input Q15 duty (XAR2)
      MOVL XAR3,#CMPAHR1               ; Pointer to HRPWM CMPA reg (XAR3)

; Output for EPWM1A (HRPWM
      MOV T,*XAR2                      ; T <= duty
      MPY ACC,T,@_hrDAC_period         ; Q15 to Q0 scaling based on period
      ADD ACC,@_HrDAC_period<<15       ; Offset for bipolar operation
      MOV T,@_MEP_ScaleFactor          ; MEP scale factor (from optimizer s/w)
      MPYU P,T,@AL                     ; P <= T * AL, optimizer scaling
      MOVH @AL,P                       ; AL <= P, move result back to ACC
      ADD ACC, #0x080                  ; MEP range and rounding adjustment
      MOVL *XAR3,ACC                   ; CMPA:CMPAHR(31:8) <= ACC

; Output for EPWM1B (Regular Res) Optional - for comparison purpose only
      MOV *+XAR3[2],AH                 ; Store ACCH to regular CMPB
```

7.5.6.16 ePWM Crossbar (XBAR)

Figure 7-254 shows the architecture of the ePWM Crossbar (XBAR). This module enables selection of various trigger sources into any of the dedicated EPWM trips inputs.

Note

Refer to the Crossbar (XBAR) chapter for more information on the XBAR modules, including XBAR flags.

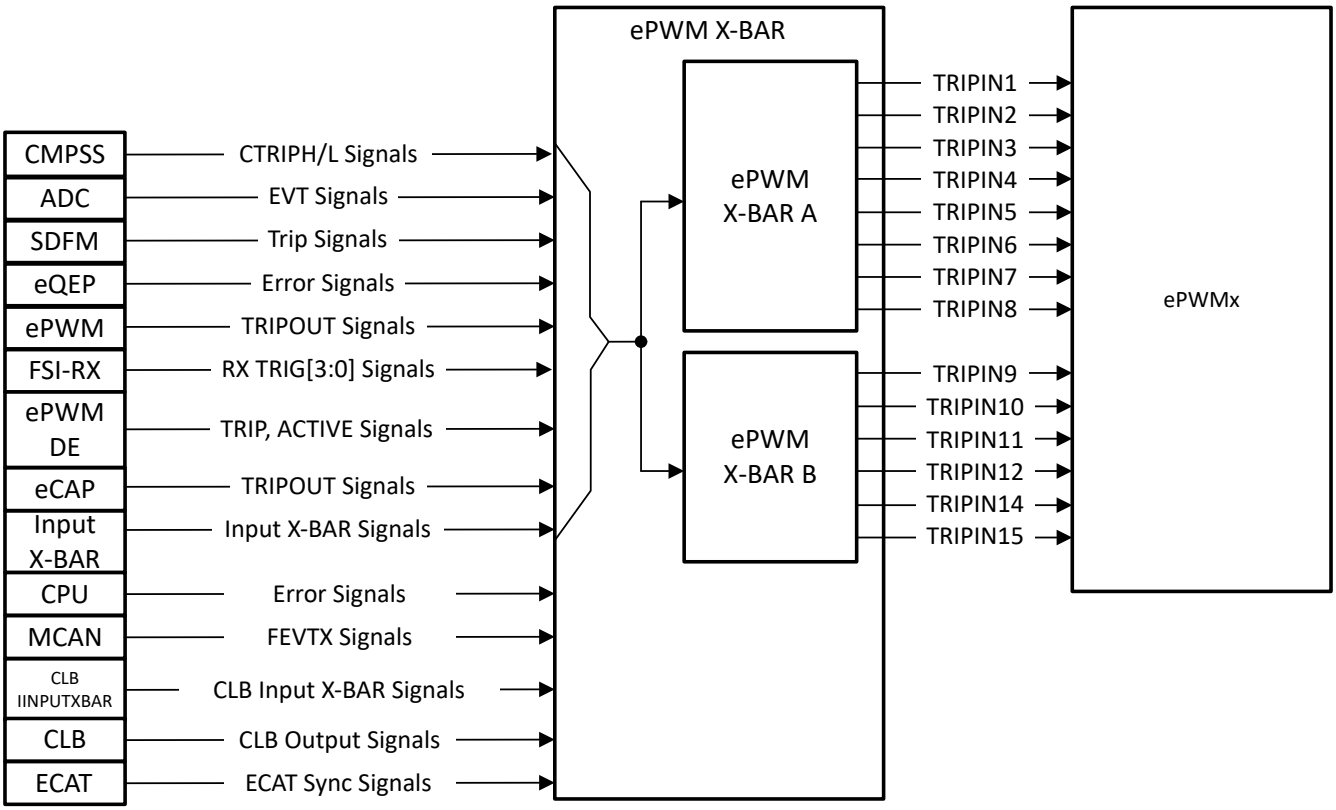


Figure 7-254. ePWM XBAR

7.5.6.17 Register Lock Protection

The register lock protection mechanism is added to protect the critical ePWM registers from being corrupted by accidental writes in case of runaway code. The register EPWMLOCK contains the definition of Lock bits (Table 7-137 shows the lock bits and the corresponding registers). This register also has a KEY field; writes to this register succeed only if the KEY field is written with a value of 0xa5a5. Refer to the register descriptions for more details.

Table 7-137. Lock Bits and Corresponding Registers

Bit Field	Definition	Registers Locked
HRLOCK	HRPWM Register Set Lock	HRCNFG, HRPWR, HRMSTEP, HRPCTL
GLLOCK	Global Load Register Set Lock	GLDCTL, GLDCFG
TZCFGLOCK	TripZone Register Set Lock	TZSEL, TZDCSEL, TZCTL, TZCTL2, TZCTLDCA, TZCTLDCB, TZEINT
TZCLRLOCK	TripZone Clear Register Set Lock	TZCLR, TZCBCCLR, TZOSTCLR, TZFRC
DCLOCK	Digital Compare Register Set Lock	DCTRIPSEL, DCACTL, DCBCTL, DCFCTL, DCCAPCTL, DCAHTRIPSEL, DCALTRIPSEL, DCBHTRIPSEL, DCBLTRIPSEL

Note

Due to the presence of the KEY field in the same register, only 32-bit writes succeed if the KEY matches. The 16-bit writes to the upper or lower half of this register are ignored.

7.5.6.18 Applications to Power Topologies

An ePWM module has all the local resources necessary to operate completely as a standalone module or to operate in synchronization with other identical ePWM modules.

7.5.6.18.1 Overview of Multiple Modules

Previously in this chapter, all discussions have described the operation of a single module. To facilitate the understanding of multiple modules working together in a system, the ePWM module described in reference is represented by the more simplified block diagram shown in Figure 7-255. This simplified ePWM block shows only the key resources needed to explain how a multiswitch power topology is controlled with multiple ePWM modules working together.

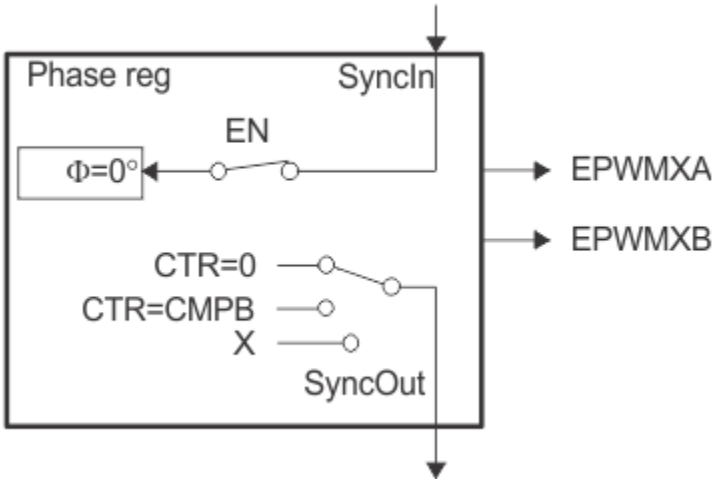


Figure 7-255. Simplified ePWM Module

7.5.6.18.2 Key Configuration Capabilities

The key configuration choices available to each module are as follows:

- Options for SyncIn
  - Load own counter with phase register on an incoming sync strobe—enable (EN) switch closed
  - Do nothing or ignore incoming sync strobe—enable switch open
  - Sync Source mode, provides a sync at PWM boundaries—SyncOut connected to CTR = PRD
  - Sync Source mode, provides a sync at any programmable point in time—SyncOut connected to CTR = CMPB
  - Module is in standalone mode and provides no sync to other modules—SyncOut connected to X (disabled)
- Options for SyncOut
  - Sync Source mode, provides a sync at PWM boundaries—SyncOut connected to CTR = PRD
  - Sync Source mode, provides a sync at any programmable point in time—SyncOut connected to CTR = CMPB
  - Module is in standalone mode and provides no sync to other modules—SyncOut connected to X (disabled)

For each choice of SyncOut, a module can also choose to load its own counter with a new phase value on a SyncIn strobe input or choose to ignore the value (that is, by the enable switch). Although various combinations are possible, the two most common—Sync Source module and Sync Receiver module modes —are shown in Figure 7-256.

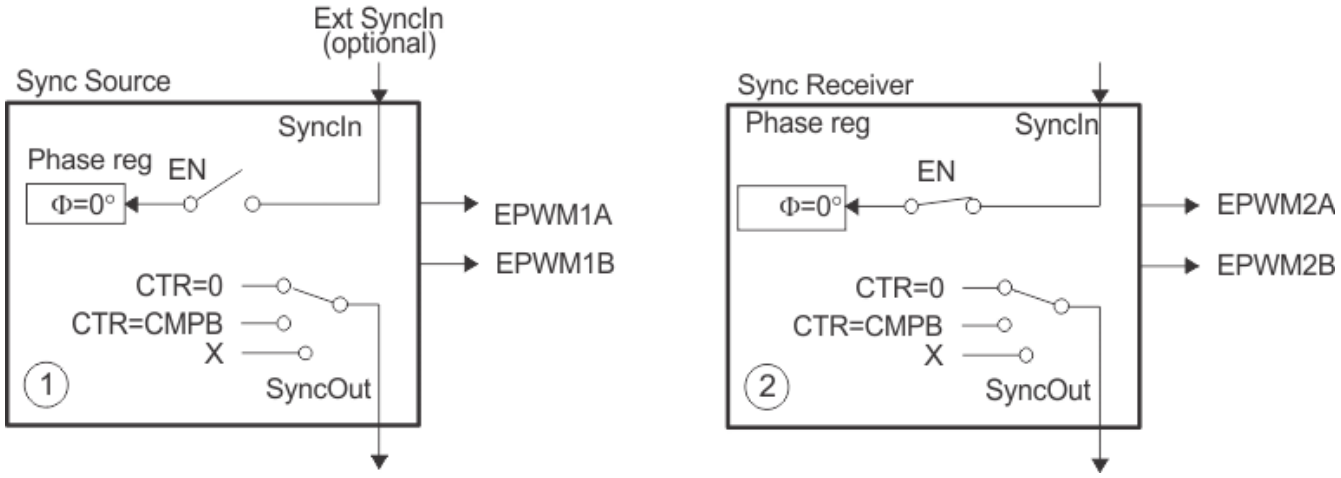


Figure 7-256. EPWM1 Configured as a Typical Sync Source, EPWM2 Configured as a Sync Receiver

7.5.6.18.3 Controlling Multiple Buck Converters With Independent Frequencies

One of the simplest power converter topologies is the buck. A single ePWM module configured as a sync source can control two buck stages with the same PWM frequency. If independent frequency control is required for each buck converter, then one ePWM module must be allocated for each converter stage. Figure 7-257 shows four buck stages, each running at independent frequencies. In this case, all four ePWM modules are configured as Sync Sources and no synchronization is used. Figure 7-258 shows the waveforms generated by the setup shown in Figure 7-257; note that only three waveforms are shown, although there are four stages.

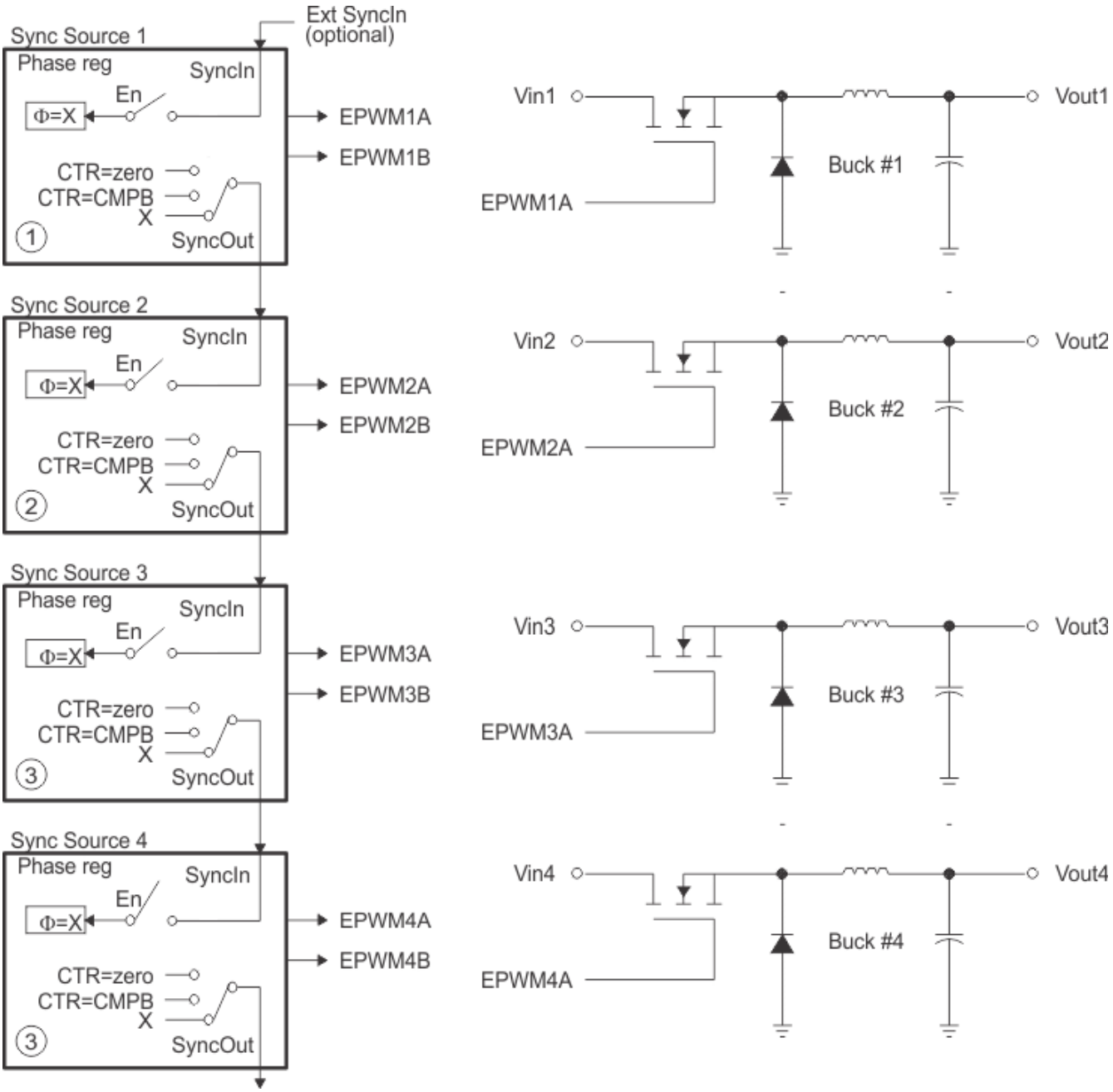


Figure 7-257. Control of Four Buck Stages. Here $F_{PWM1} \neq F_{PWM2} \neq F_{PWM3} \neq F_{PWM4}$

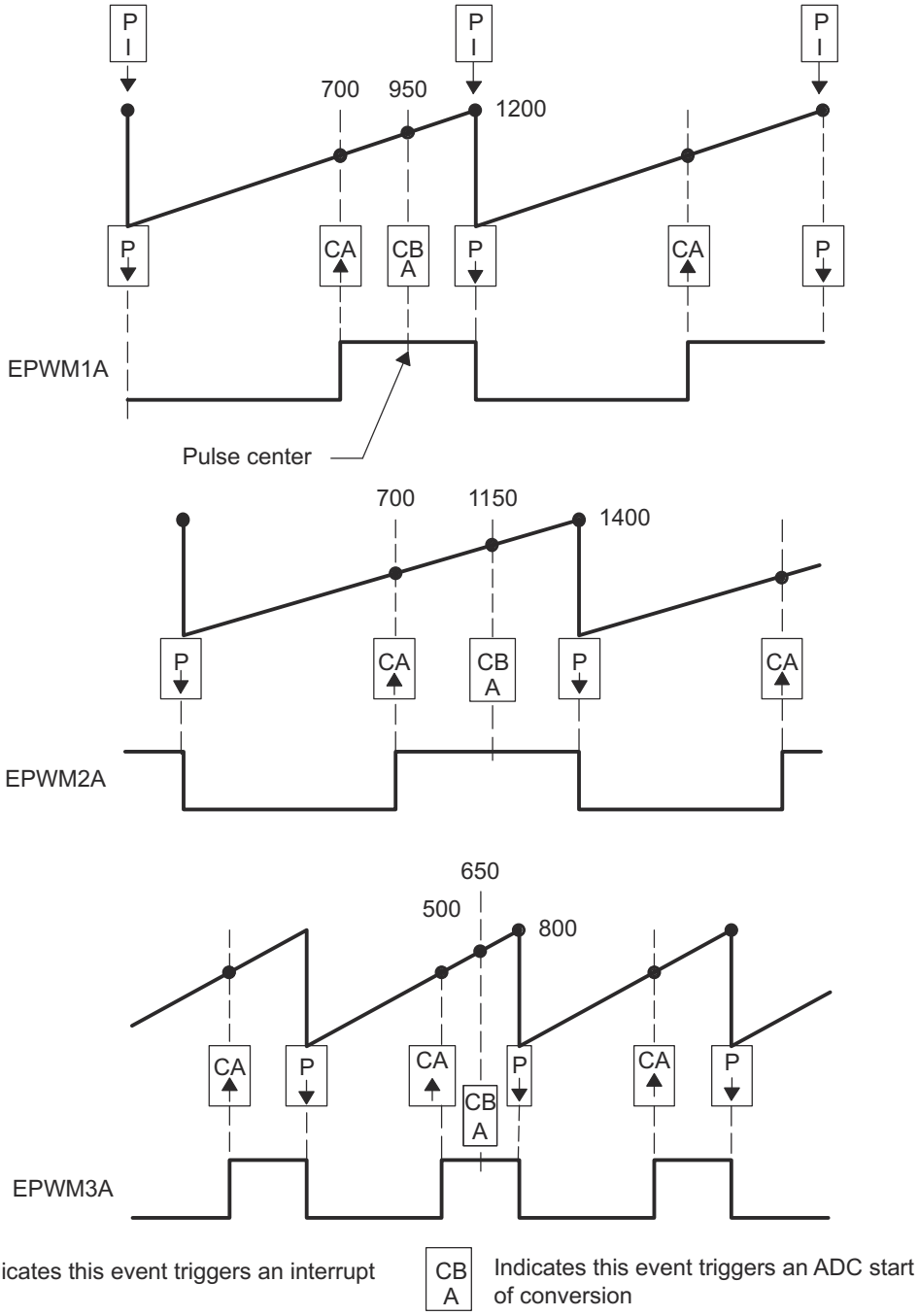
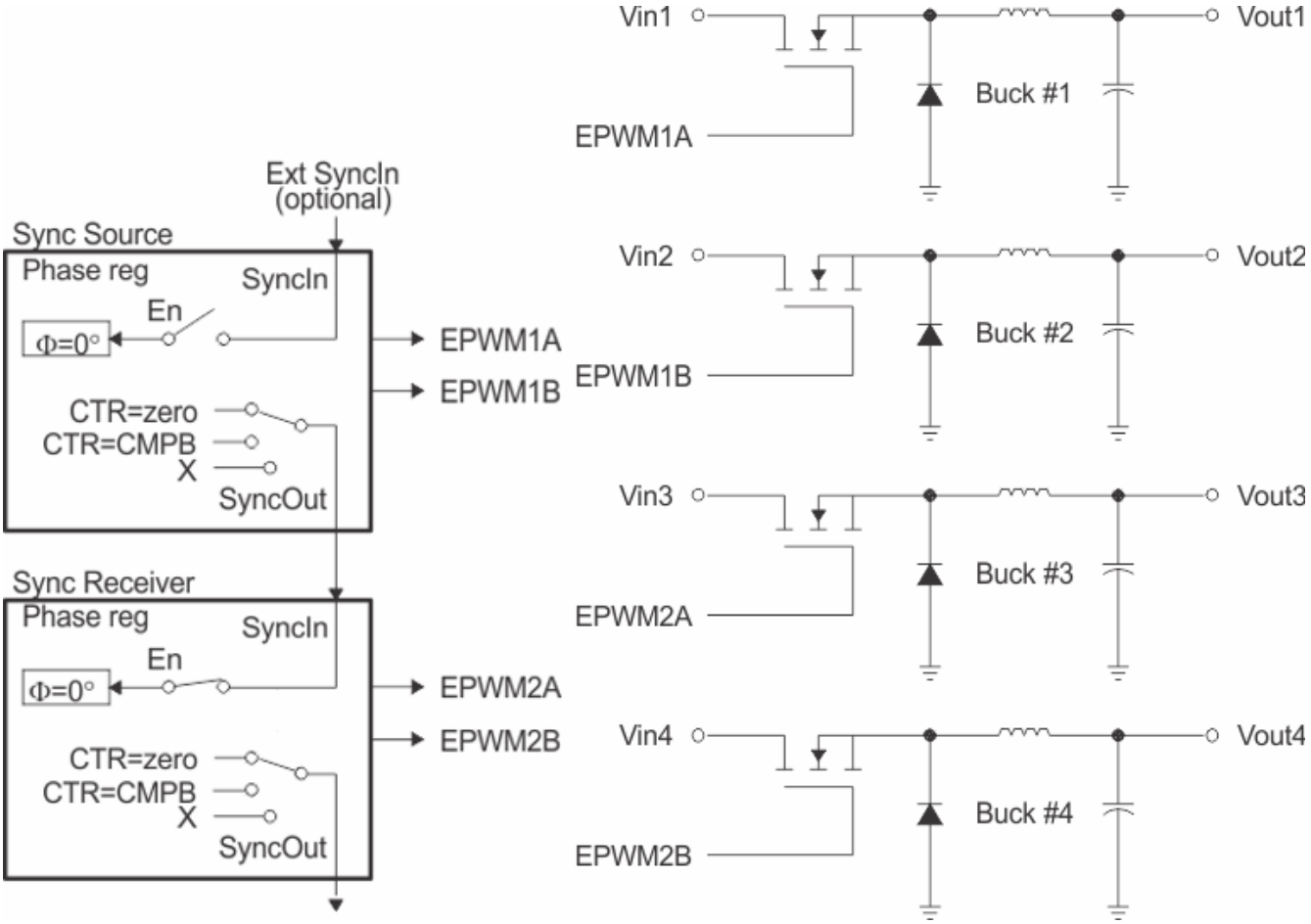


Figure 7-258. Buck Waveforms for Control of Four Buck Stages (Note: Only three bucks shown here)

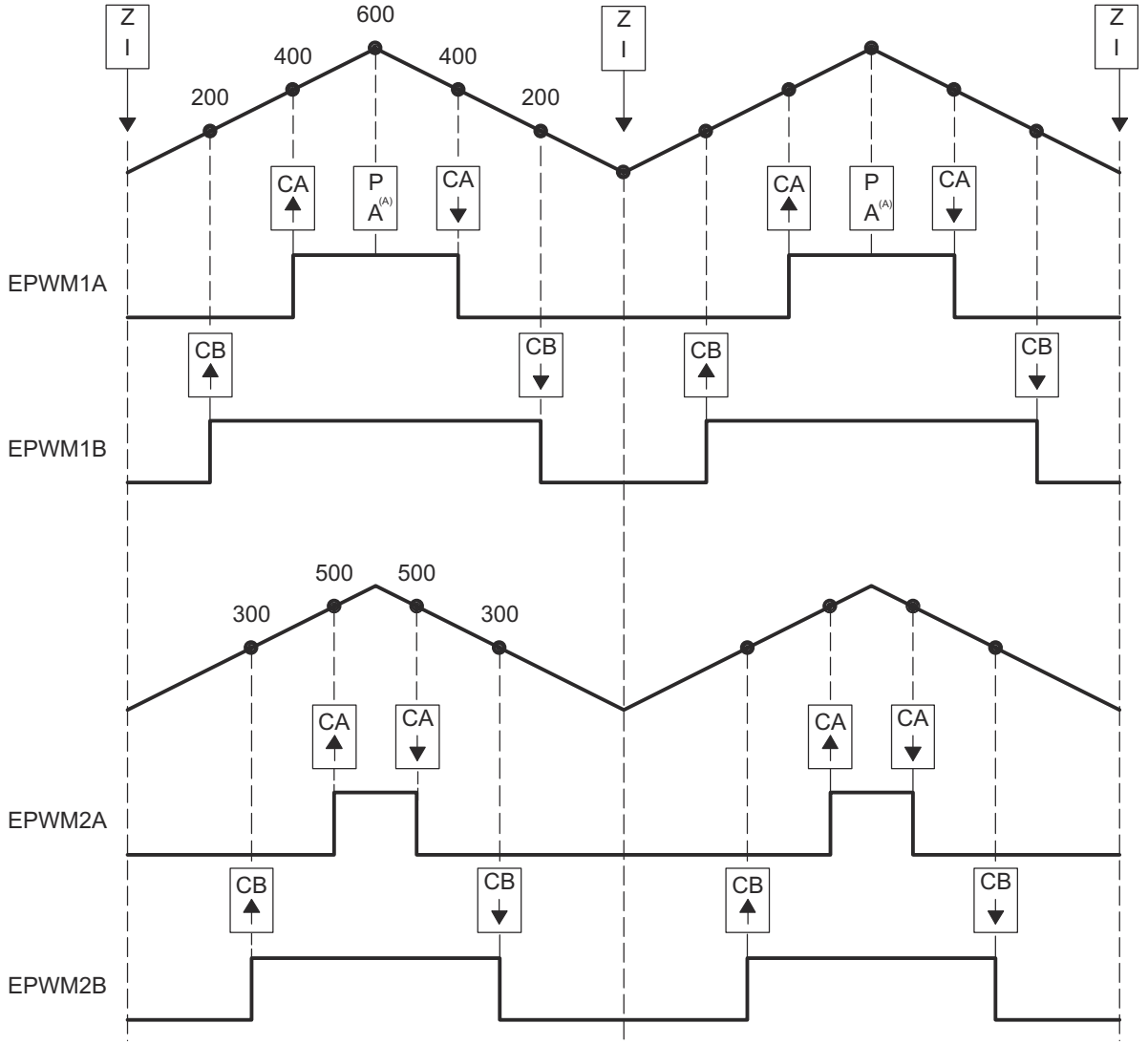
7.5.6.18.4 Controlling Multiple Buck Converters With Same Frequencies

If synchronization is a requirement, ePWM module 2 is configured as a sync receiver and operates at integer multiple (N) frequencies of module 1. The sync signal from sync source to sync receiver makes sure these modules remain locked. Figure 7-259 shows such a configuration; Figure 7-260 shows the waveforms generated by the configuration.



A. φ = X indicates value in phase register is a "don't care"

Figure 7-259. Control of Four Buck Stages. (Note: F_{PWM2} = N x F_{PWM1})



A. Starts ADC conversion.

Figure 7-260. Buck Waveforms for Control of Four Buck Stages (Note: $F_{PWM2} = F_{PWM1}$)

7.5.6.18.5 Controlling Multiple Half H-Bridge (HHB) Converters

Topologies that require control of multiple switching elements can also be addressed with these same ePWM modules. It is possible to control a Half-H bridge stage with a single ePWM module. This control can be extended to multiple stages. Figure 7-261 shows control of two synchronized Half-H bridge stages where stage 2 can operate at integer multiple (N) frequencies of stage 1. Figure 7-262 shows the waveforms generated by the configuration shown in Figure 7-261.

ePWM module 2 (sync receiver) is configured for Sync flow-through; if required, this configuration allows for a third Half-H bridge to be controlled by ePWM module 3 and also, most importantly, to remain in synchronization with sync source ePWM module 1.

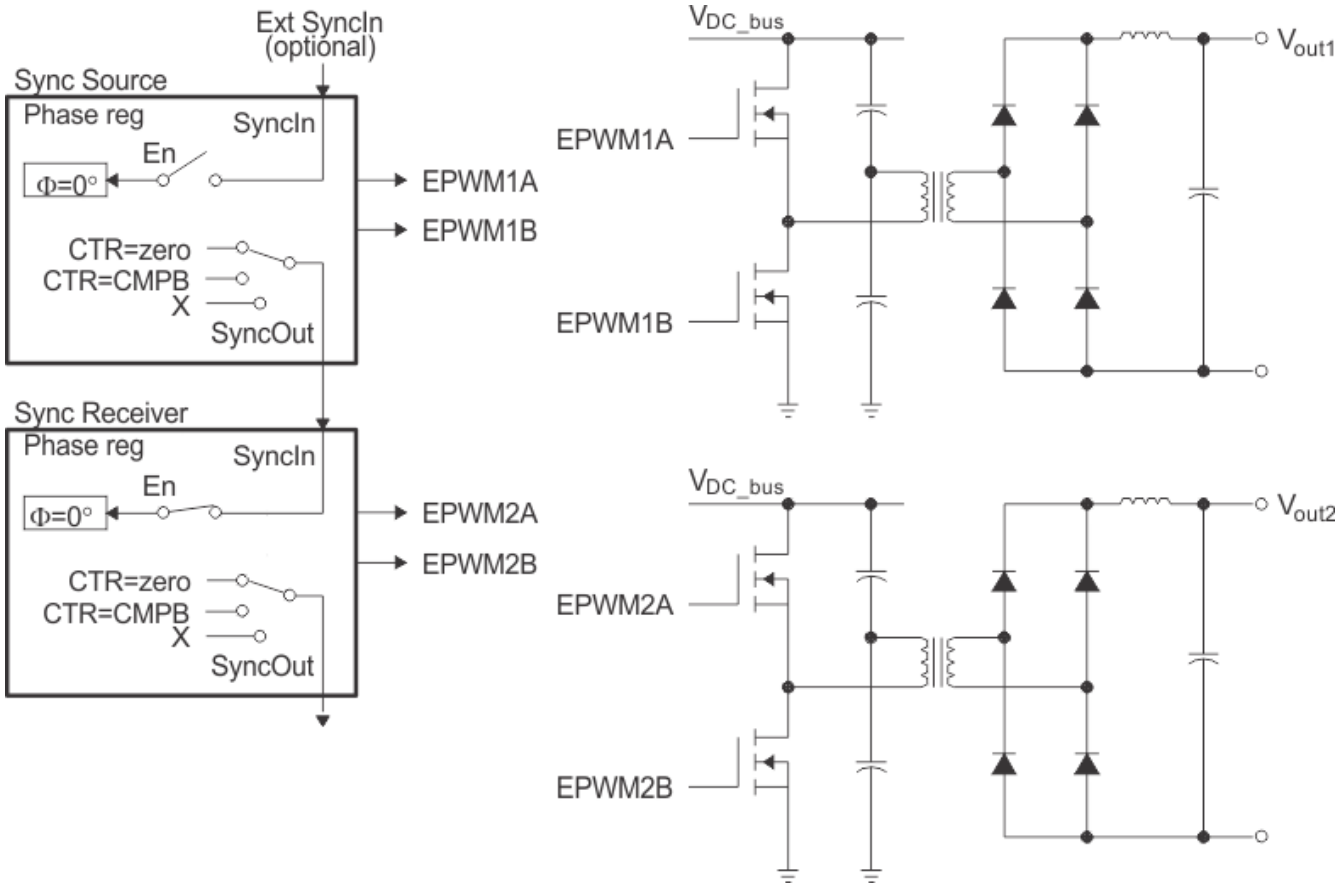


Figure 7-261. Control of Two Half-H Bridge Stages (F_{PWM2} = N x F_{PWM1})

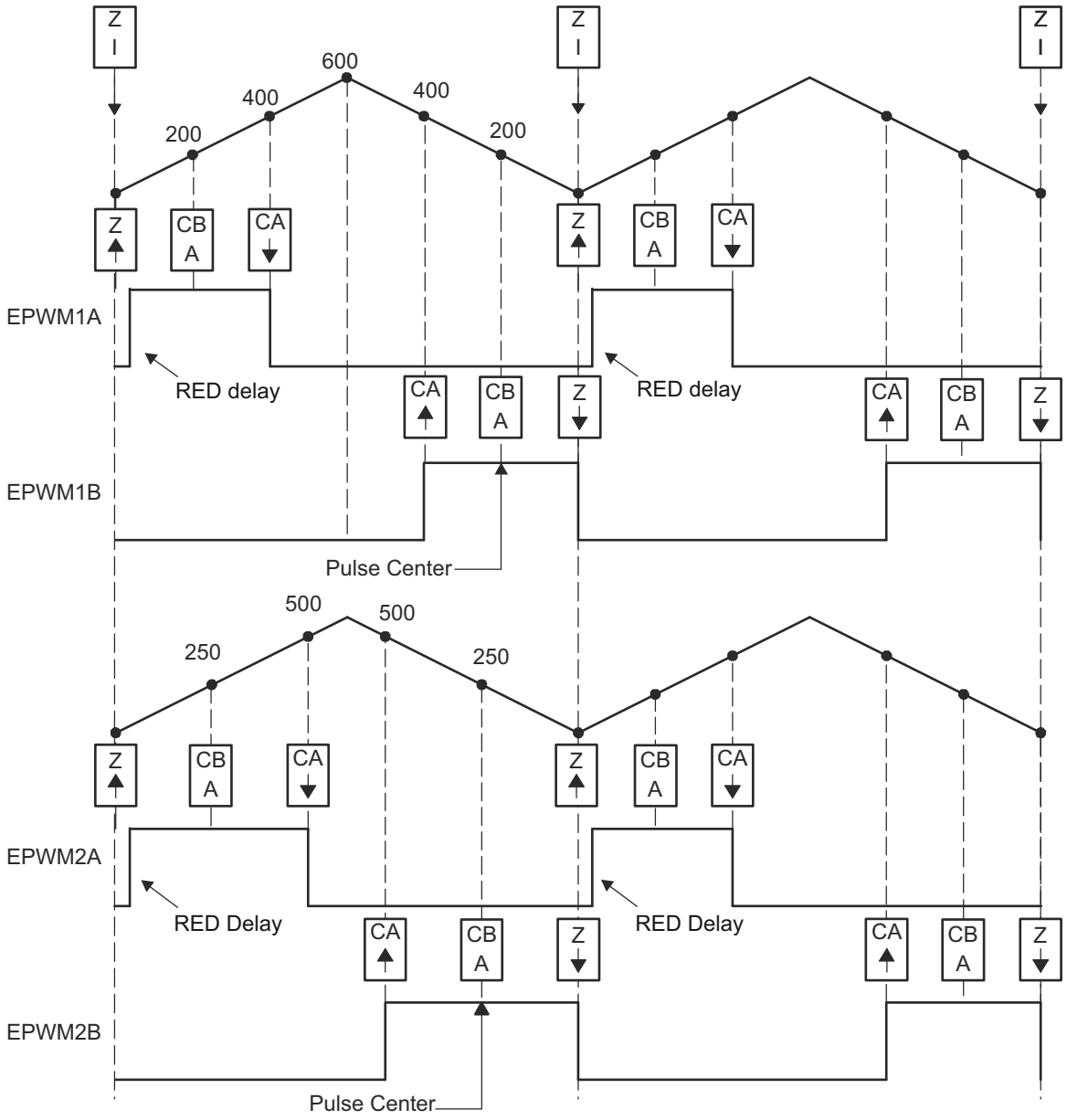


Figure 7-262. Half-H Bridge Waveforms for Control of Two Half-H Bridge Stages (Note: Here $F_{PWM2} = F_{PWM1}$)

7.5.6.18.6 Controlling Dual 3-Phase Inverters for Motors (ACI and PMSM)

The idea of multiple modules controlling a single power stage can be extended to the 3-phase inverter case. In such a case, six switching elements are controlled using three PWM modules, one for each leg of the inverter. Each leg must switch at the same frequency and all legs must be synchronized. A sync receivers configuration easily addresses this requirement. Figure 7-263 shows how six PWM modules control two independent 3-phase inverters; each running a motor.

As in the cases shown in the previous sections, we have a choice of running each inverter at a different frequency (module 1 and module 4 are>sync sources as in Figure 7-263), or both inverters can be synchronized by using one sync source (module 1) and five sync receivers. In this case, the frequency of modules 4, 5, and 6 (all equal) can be integer multiples of the frequency for modules 1, 2, and 3 (also all equal).

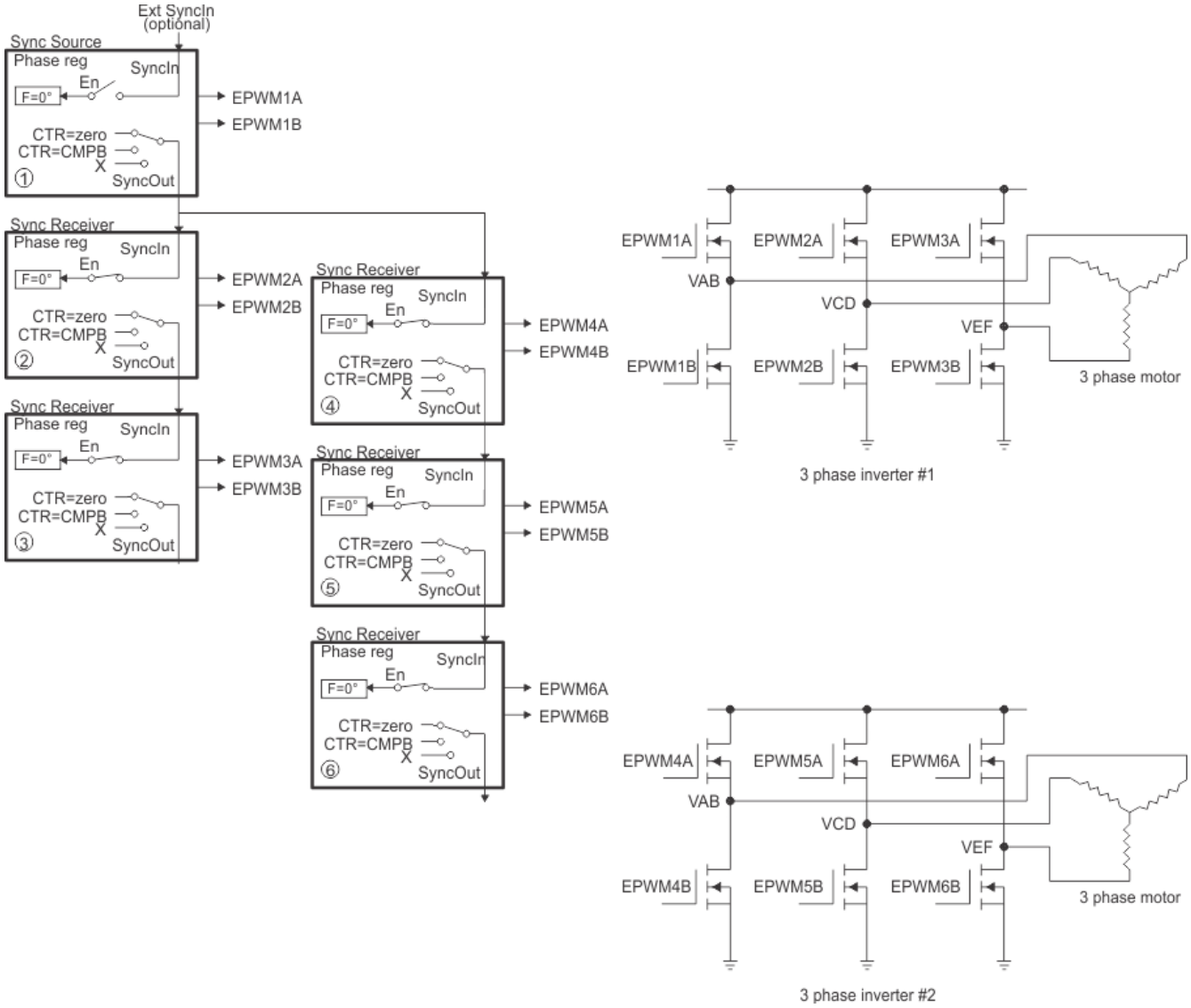


Figure 7-263. Control of Dual 3-Phase Inverter Stages as Is Commonly Used in Motor Control

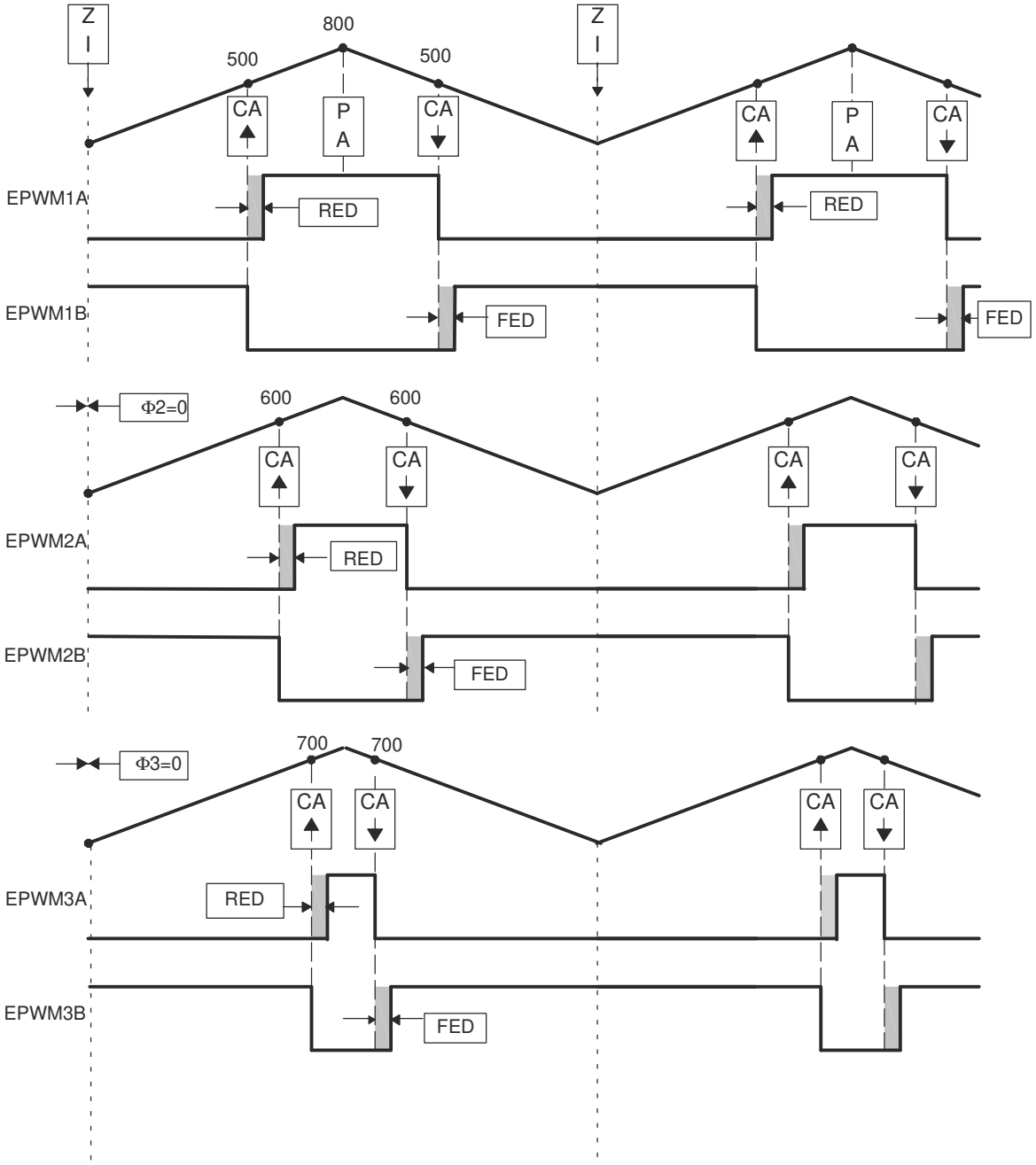


Figure 7-264. 3-Phase Inverter Waveforms for Control of Dual 3-Phase Inverter Stages (Only One Inverter Shown)

7.5.6.18.7 Practical Applications Using Phase Control Between PWM Modules

So far, none of the examples have made use of the phase register (TBPHS). It has either been set to zero or the value has been a don't care. However, by programming appropriate values into TBPHS, multiple PWM modules can address another class of power topologies that rely on phase relationship between legs (or stages) for correct operation. As described in the time-base submodule section, a PWM module can be configured to allow a SyncIn pulse to cause the TBPHS register to be loaded into the TBCTR register. To illustrate this concept, Figure 7-265 shows a sync source and sync receiver module with a phase relationship of 120° (that is, the sync receiver leads the sync source).

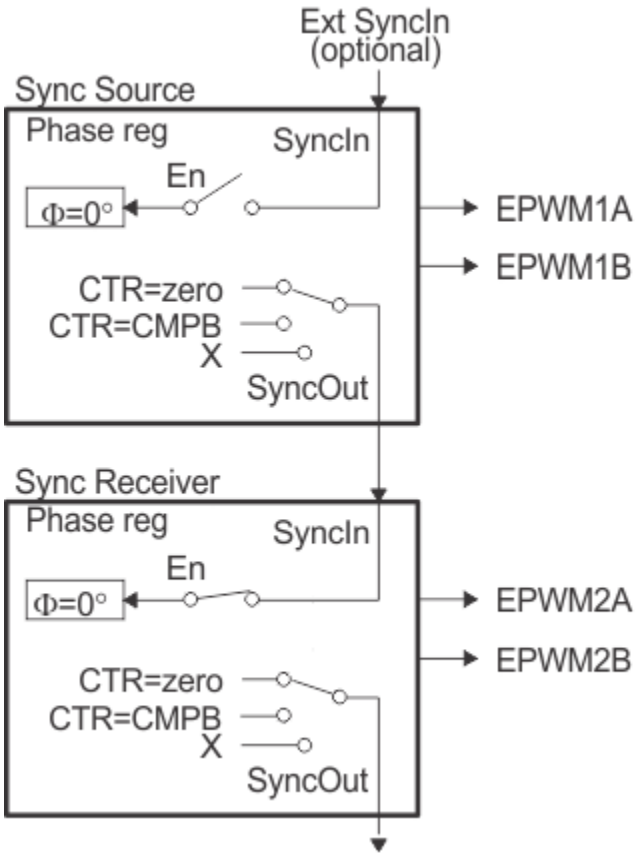


Figure 7-265. Configuring Two PWM Modules for Phase Control

Figure 7-266 shows the associated timing waveforms for this configuration. Here, TBPRD = 600 for both sync source and sync receiver. For the sync receiver, TBPHS = 200 (that is, 200/600 x 360° = 120°). Whenever the sync source generates a SyncIn pulse (CTR = PRD), the value of TBPHS = 200 is loaded into the sync receiver TBCTR register so the sync receiver time-base is always leading the sync source time-base by 120°.

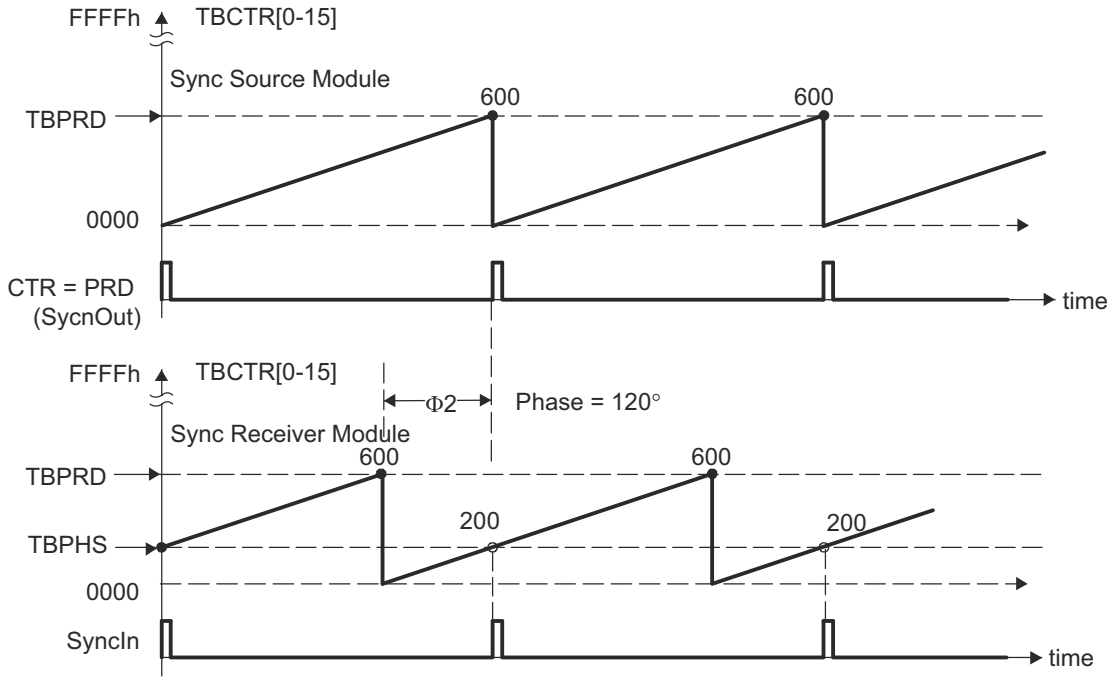


Figure 7-266. Timing Waveforms Associated with Phase Control Between Two Modules

7.5.6.18.8 Controlling a 3-Phase Interleaved DC/DC Converter

A popular power topology that makes use of phase-offset between modules is shown in Figure 7-267. This system uses three PWM modules, with module 1 configured as the sync source. To work, the phase relationship between adjacent modules must be F = 120°. This is achieved by setting thesync receiver TBPHS registers 2 and 3 with values of 1/3 and 2/3 of the period value, respectively. For example, if the period register is loaded with a value of 600 counts, then TBPHS (sync receiver 2) = 200 and TBPHS (sync receiver 3) = 400. Both sync receiver modules are synchronized to the sync source module 1.

This concept can be extended to four or more phases, by setting the TBPHS values appropriately. The following formula gives the TBPHS values for N phases:

TBPHS(N,M) = (TBPRD/N) x (M-1)

Where:

N = number of phases

M = PWM module number

For example, for the 3-phase case (N=3), TBPRD = 600,

TBPHS(3,2) = (600/3) x (2-1) = 200 (that is, Phase value for Sync Receiver module 2)

TBPHS(3,3) = 400 (that is, Phase value for Sync Receiver module 3)

Figure 7-268 shows the waveforms for the configuration in Figure 7-267.

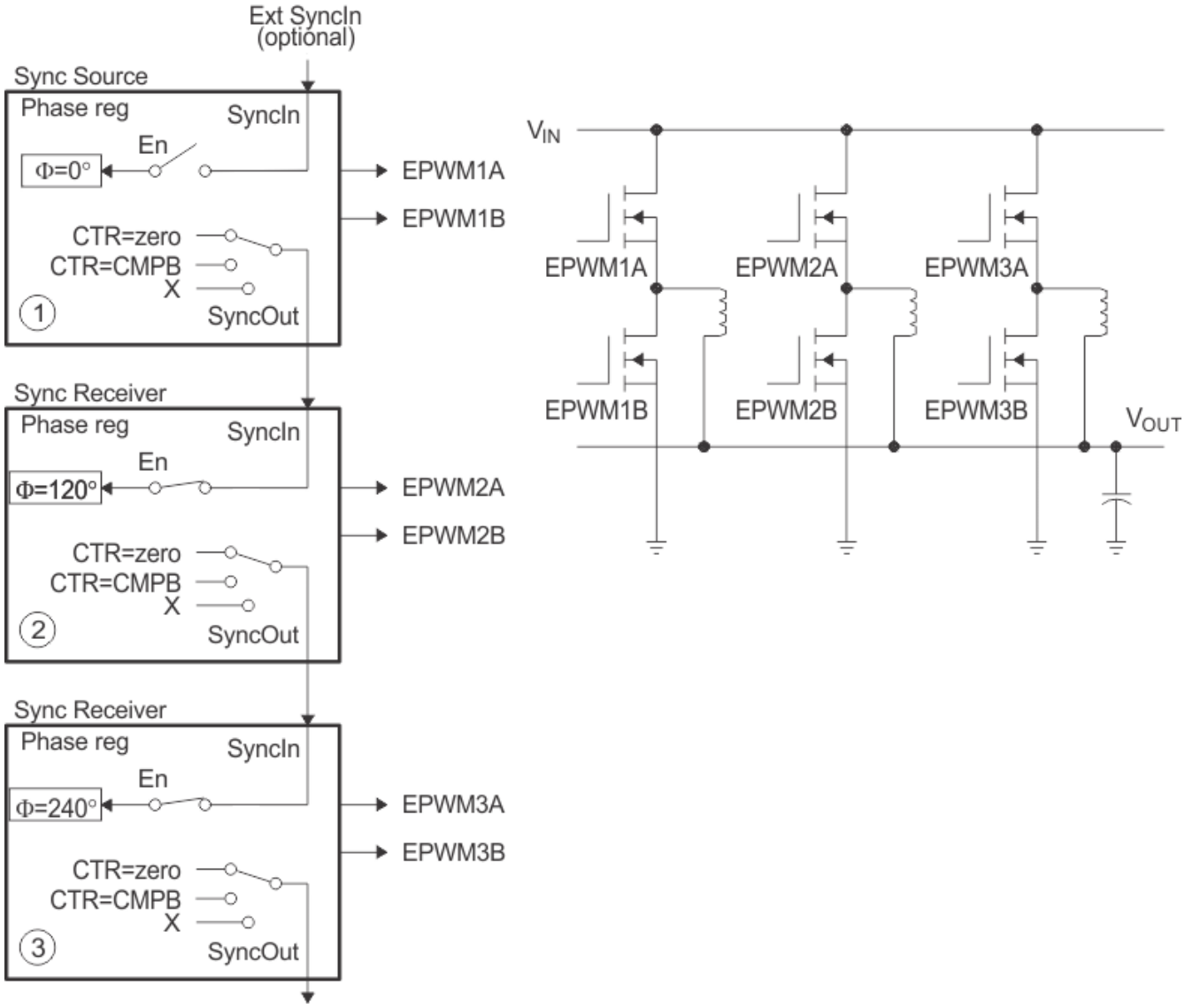


Figure 7-267. Control of 3-Phase Interleaved DC/DC Converter

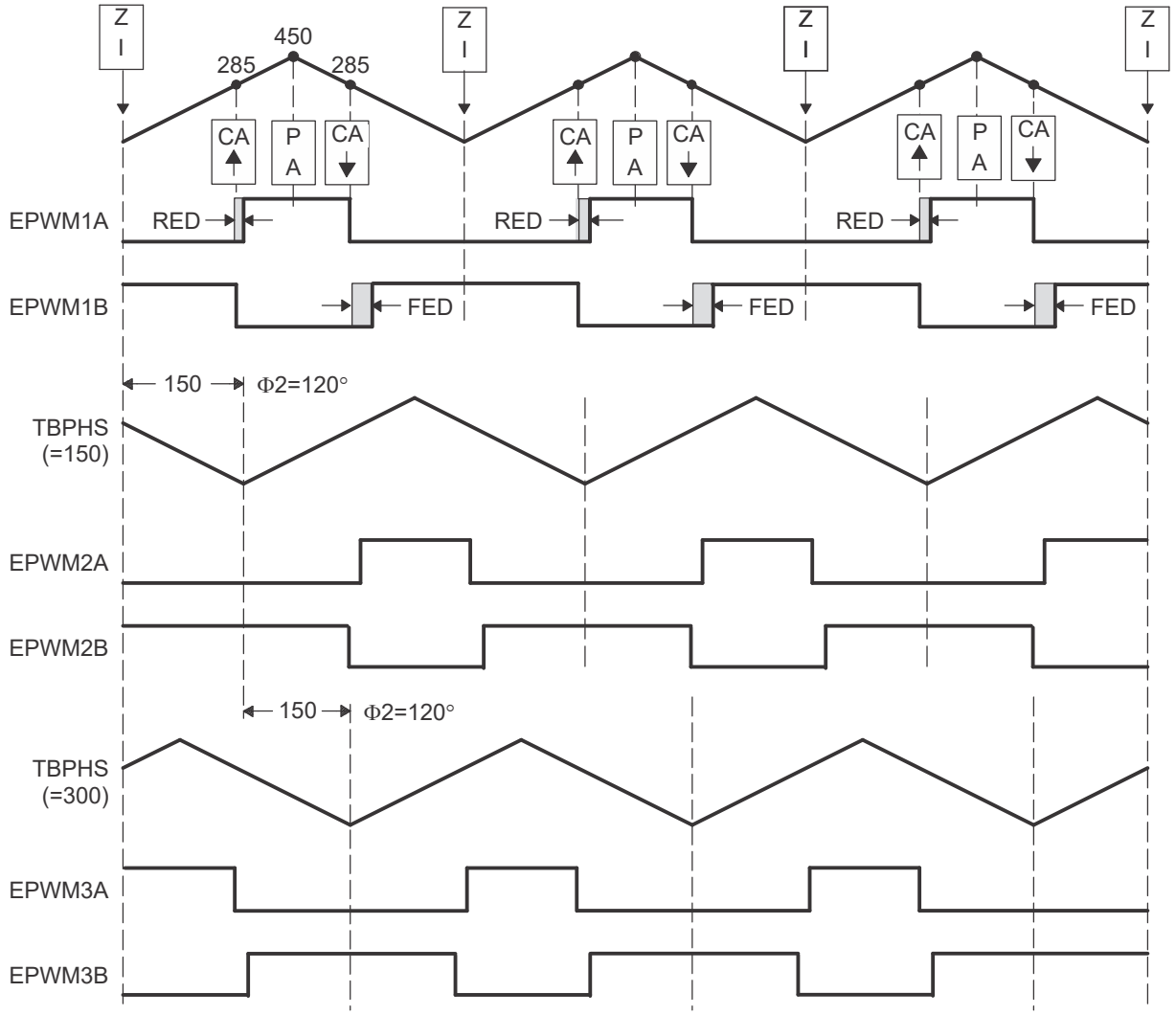


Figure 7-268. 3-Phase Interleaved DC/DC Converter Waveforms for Control of 3-Phase Interleaved DC/DC Converter

7.5.6.18.9 Controlling Zero Voltage Switched Full Bridge (ZVSFB) Converter

The example given in Figure 7-269 assumes a static or constant phase relationship between legs (modules). In such a case, control is achieved by modulating the duty cycle. It is also possible to dynamically change the phase value on a cycle-by-cycle basis. This feature lends itself to controlling a class of power topologies known as *phase-shifted full bridge*, or *zero voltage switched full bridge.* Here the controlled parameter is not duty cycle (this is kept constant at approximately 50 percent); instead it is the phase relationship between legs. Such a system can be implemented by allocating the resources of two PWM modules to control a single power stage, which in turn requires control of four switching elements. Figure 7-270 shows a sync source and sync receiver module combination synchronized together to control a full H-bridge. In this case, both sync source and sync receiver modules are required to switch at the same PWM frequency. The phase is controlled by using the sync receiver phase register (TBPHS). The sync source phase register is not used and therefore can be initialized to zero.

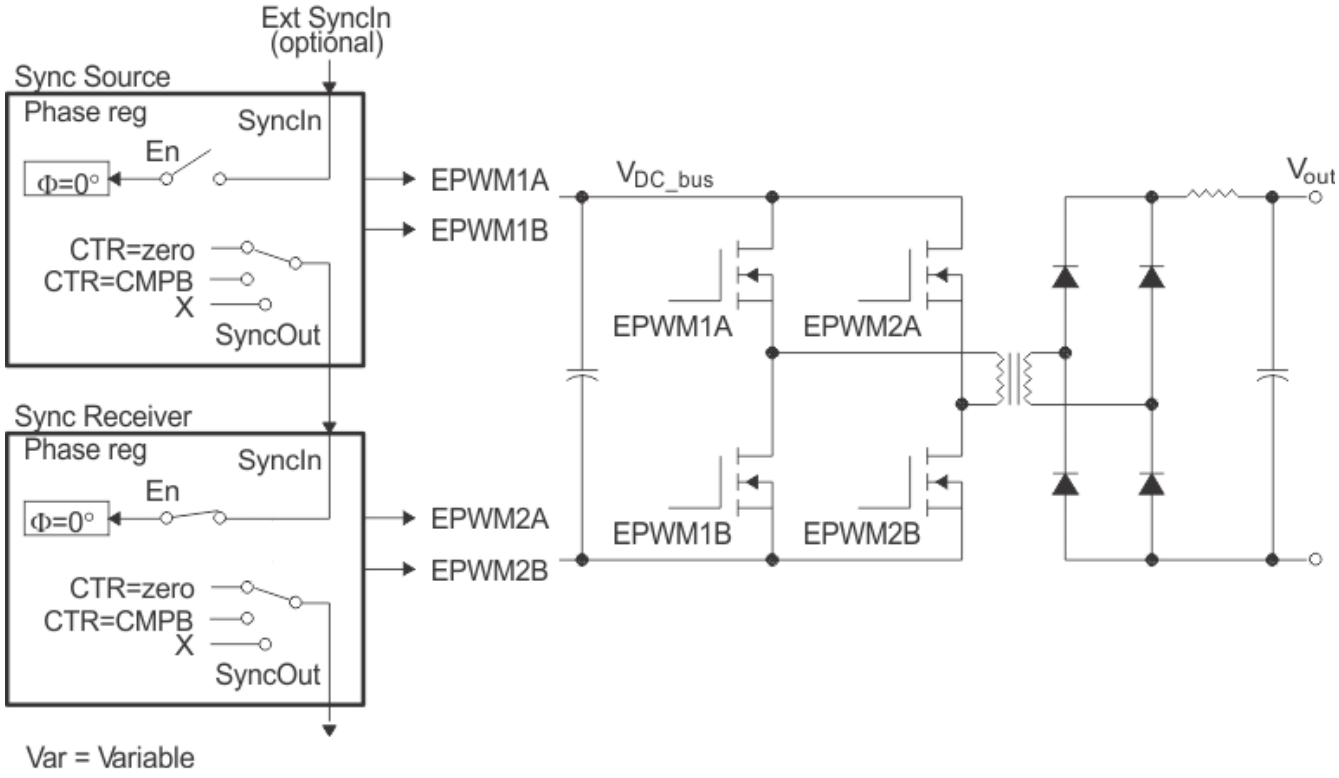


Figure 7-269. Control of Full-H Bridge Stage (F_{PWM2} = F_{PWM1})

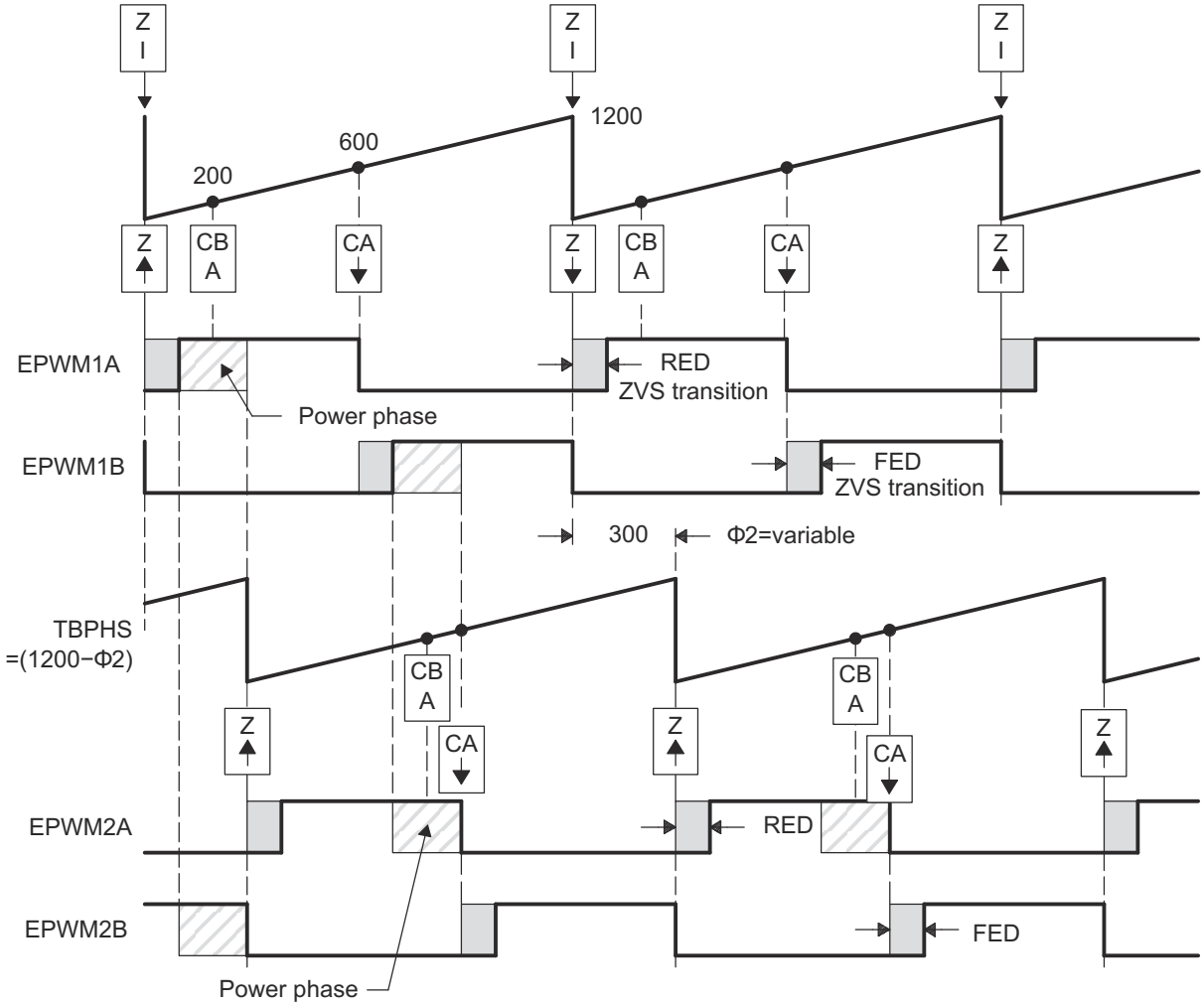


Figure 7-270. ZVS Full-H Bridge Waveforms

7.5.6.18.10 Controlling a Peak Current Mode Controlled Buck Module

Peak current control techniques offer a number of benefits like automatic over current limiting, fast correction for input voltage variations and reducing magnetic saturation. Figure 7-271 shows the use of ePWM1A along with the on-chip analog comparator for buck converter topology. The output current is sensed through a current sense resistor and fed to the positive terminal of the on-chip comparator. The internal programmable 12-bit DAC can be used to provide a reference peak current at the negative terminal of the comparator. Alternatively, an external reference can be connected at this input. The comparator output is an input to the Digital compare sub-module. The ePWM module is configured in such a way so as to trip the ePWM1A output as soon as the sensed current reaches the peak reference value. A cycle-by-cycle trip mechanism is used. Figure 7-272 shows the waveforms generated by the configuration.

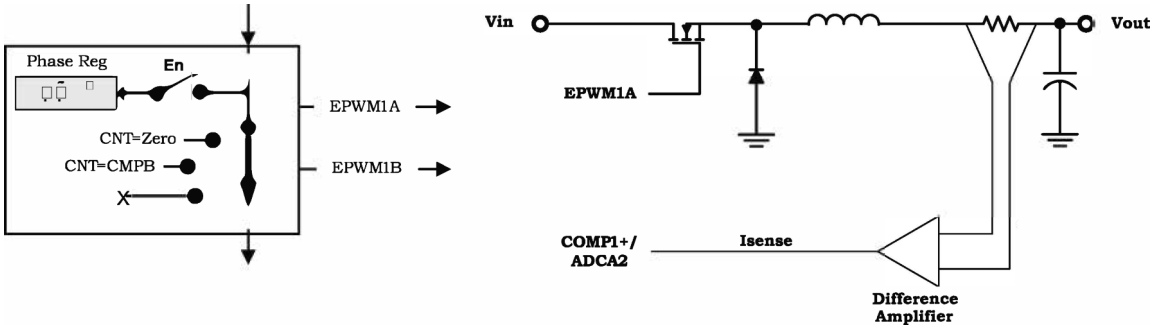


Figure 7-271. Peak Current Mode Control of Buck Converter

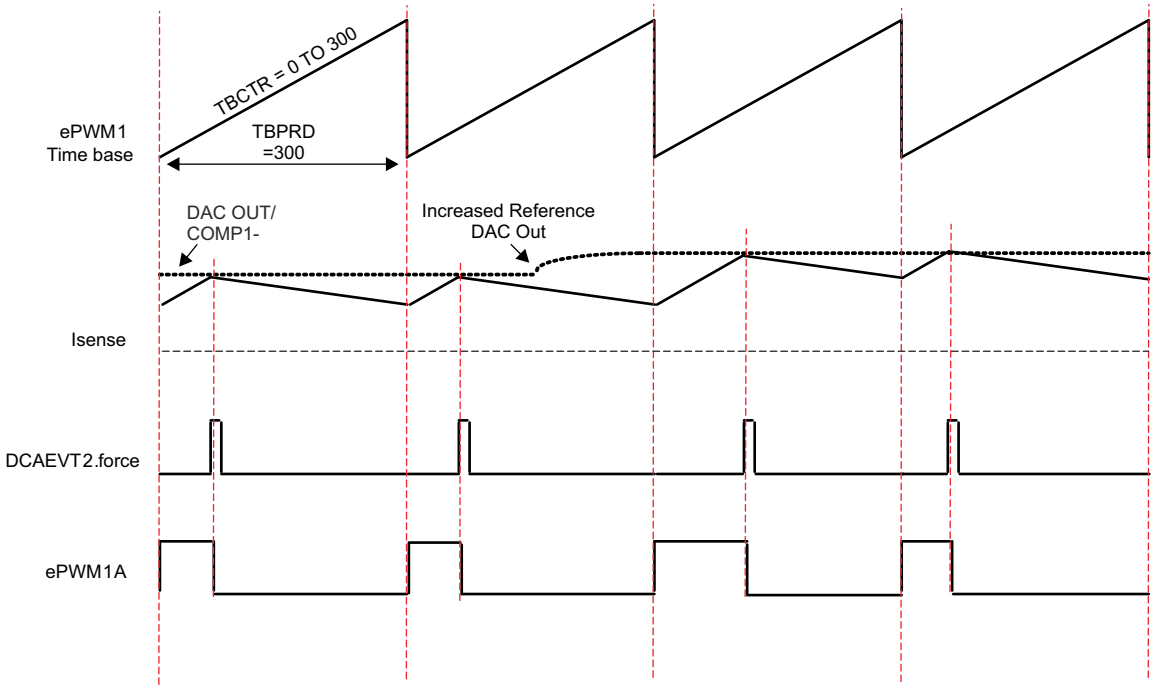


Figure 7-272. Peak Current Mode Control Waveforms for Control of Buck Converter

7.5.6.18.11 Controlling H-Bridge LLC Resonant Converter

Various topologies of resonant converters are well-known in the field of power electronics for many years. In addition to these, H-bridge LLC resonant converter topology has recently gained popularity in many consumer electronics applications where high efficiency and power density are required. In this example, single channel configuration of ePWM1 is detailed, yet the configuration can easily be extended to multichannel. Here the controlled parameter is not duty cycle (this is kept constant at approximately 50 percent); instead the parameter is frequency. Although the deadband is not controlled and kept constant as 300 ns (that is, 30 at 100 MHz TBCLK), the user can update the deadband in real time to enhance the efficiency by adjusting enough time delay for soft switching.

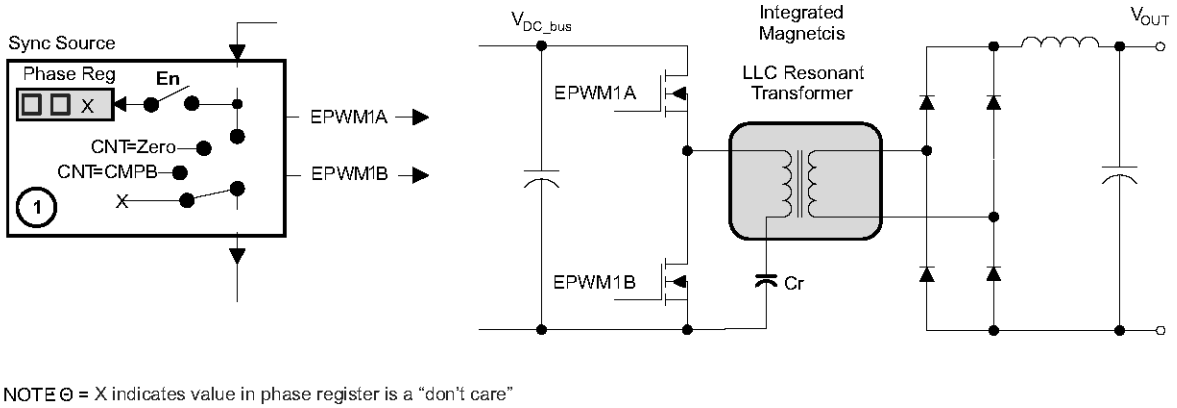


Figure 7-273. Control of Two Resonant Converter Stages

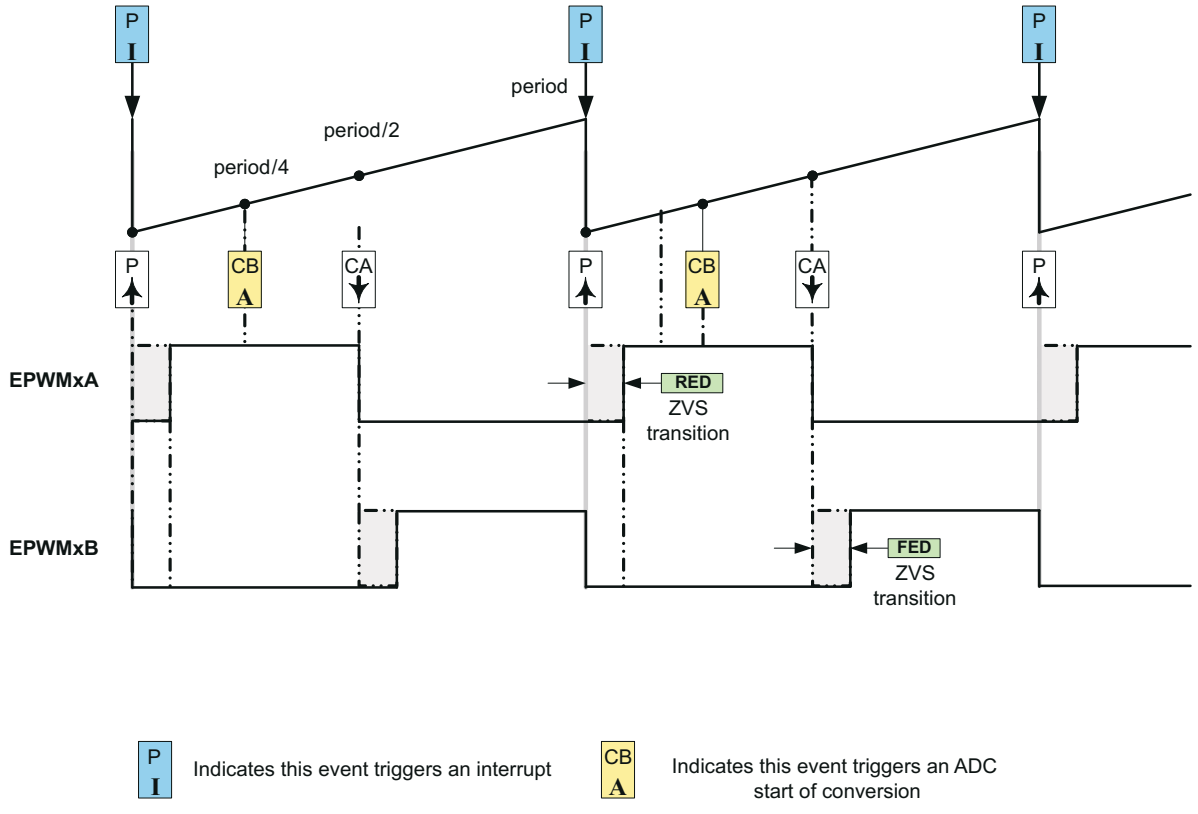


Figure 7-274. H-Bridge LLC Resonant Converter PWM Waveforms

7.5.6.19 EPWM Programming Guide

Driver Information

Driver features are available at the EPWM driver page.

Software API Information

The EPWM driver provides an API to configure the EPWM module. Full documentation is located on APIs for EPWM.

Example Usage

The below links show examples on how to use the EPWM:

- EPWM HR Duty Cycle
- EPWM Trip Zone
- EPWM DMA
- EPWM Valley Switching
- EPWM HR UpDown
- EPWM Protection Solution using PRU
- EPWM Minimum Deadband
- EPWM Deadband
- EPWM Illegal Combo Logic
- EPWM HR Deadband SFO
- EPWM HR Phase Shift SFO
- EPWM HR Duty Cycle SFO

7.5.7 Enhanced Capture (eCAP)

This chapter describes the enhanced capture (eCAP) module, which is used in systems where accurate timing of external events is important.

The enhanced capture (eCAP) module is a Type 3.

7.5.7.1 Introduction

7.5.7.1.1 Features

The features of the eCAP module include:

- Speed measurements of rotating machinery (for example, toothed sprockets sensed by way of 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 eCAP module features described in this chapter include:

- 32 bit time base with 5nS time resolution when sourced with a 200MHz system clock
- 4-event time-stamp registers (each 32 bits)
- Edge polarity selection for up to four sequenced time-stamp capture events
- Interrupt on either of the four events
- Single-shot capture of up to four event time-stamps
- Continuous mode capture of time stamps in a four-deep circular buffer
- Absolute time-stamp capture
- Difference (Delta) mode time-stamp capture
- When not used in capture mode, the eCAP module can be configured as a single-channel PWM output

The capture functionality of the Type 1 eCAP is enhanced from the Type 0 eCAP with the following added features:

- Event filter reset bit
  - Writing a 1 to ECCTL2[CTRFILTRESET] clears the event filter, the modulo counter, and any pending interrupts flags. Resetting the bit is useful for initialization and debug.
- Modulo counter status bits
  - The modulo counter (ECCTL2 [MODCNTRSTS]) indicates which capture register is loaded next. In the Type 0 eCAP, to know the current state of the modulo counter was not possible
- DMA trigger source
  - eCAPxDMA was added as a DMA trigger. CEVT[1-4] can be configured as the source for eCAPxDMA.
- Input multiplexer
  - ECCTL0 [INPUTSEL] selects one of 128 input signals, which are detailed in Table 7-138.

The capture functionality of the Type 2 eCAP is enhanced from the Type 1 eCAP with the following added features:

- Added ECAPxSYNCINSEL register
  - ECAPxSYNCINSEL register is added for each eCAP to select an external SYNCIN. Every eCAP can have a separate SYNCIN signal.

The capture functionality of the Type 3 eCAP is enhanced from the Type 2 eCAP with the following added features:

- Two signal monitoring units to monitor edge, pulse width, and period
  - Signal monitoring can optionally be tightly coupled with ePWM global load strobes and trip events
- Increased the number of multiplexed capture inputs from 128 to 256
- DMA event generation capability in PWM mode of operation
- ADC SOC generation capability, to trigger ADC conversion

7.5.7.2 eCAP Integration

There are 16x eCAP modules integrated in the device. Figure 7-275 provides a visual representation of the device integration details.

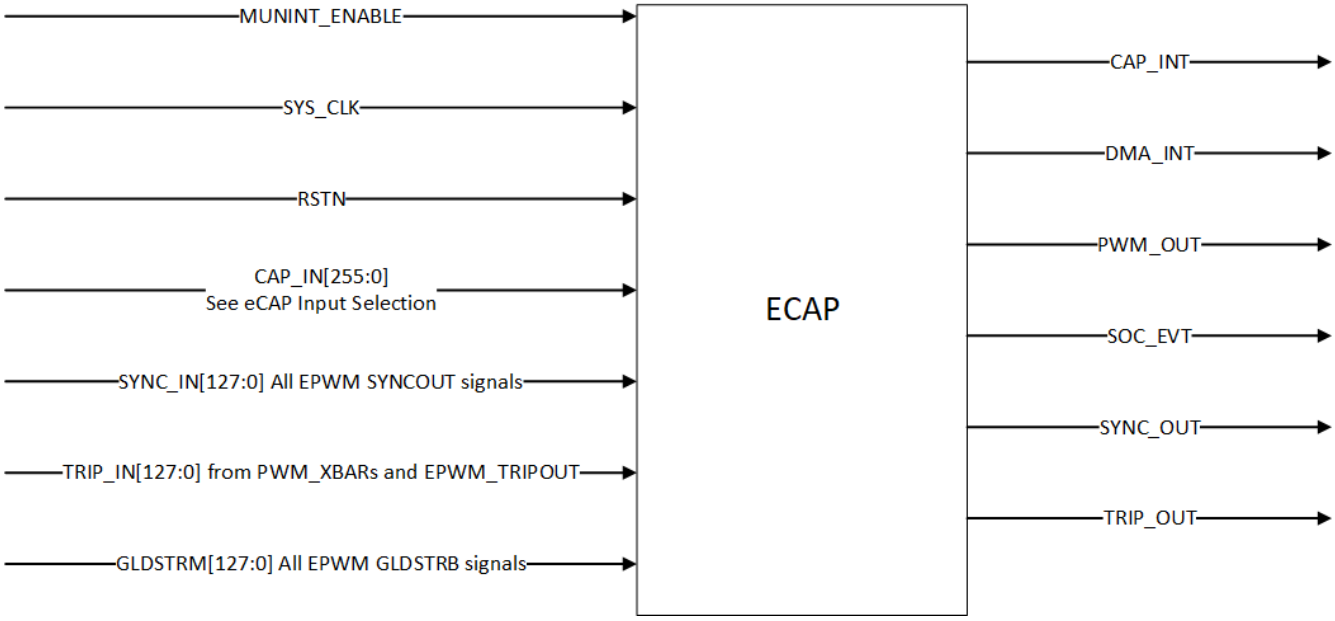


Figure 7-275. eCAP Integration Diagram

- MUNIT_Enable: This bit is used to enable/disable the signal monitoring block.
- RSTN: This bit is used to reset the eCAP module.
- SYS_CLK: Its 200MHz system clock which is functional clock for ECAP.
- CAP_IN: Capture inputs can be connected using the INPUTXBAR, PWMXBAR, adc_evt, etc. (Table 7-138).
  - 256:1 input multiplexer is used to select the capture input.
- SYNC_IN: eCAP modules can be synchronized with each other by selecting a common SYNCIN source. SYNCIN source for eCAP can be either software sync-in or external sync-in.
- TRIP_IN: The signal monitoring block can be disabled from monitoring the signal by external trip signals. It is re-enabled by removing the trip-in signal.
- GLDSTRB: This signal is used to load shadow values to MIN/MAX reg while signal monitoring.
- CAP_INT: Interrupt signal generated as a part of capture/PWM event.
- DMA_INT: DMA request signal.
- PWM_OUT: PWM output in APWM mode.
- SOC_EVT: Used to generate SOC signal for ADC during any capture/PWM event.
- SYNC_OUT: This can be used to synchronize the eCAP with other eCAPs or with other modules like PWM.
- TRIP_OUT: Trip signal is generated upon signal monitoring error. All the signal monitoring error events are OR-ed and provided as trip out.

7.5.7.2.1 eCAP Input Selection

The Input X-BAR connects the device pins to the module as input. Any GPIO on the device can be configured as an input. The GPIO input qualification can be set to synchronous or asynchronous mode. Using synchronized inputs can help with noise immunity but affects the eCAP accuracy by ±2 cycles.

When using the eCAP module, a 256:1 input multiplexer must also be configured. This multiplexer can select a variety of inputs detailed in Table 7-138 by configuring ECCTL0.INPUTSEL.

Table 7-138. eCAP Input Selection

Selection of eCAP Input	Select Index	Selection of eCAP Input	Select Index	Selection of eCAP Input	Select Index
FSI_RX0.TRIG0	0	EPWM25.SOCA	79	CMPSSA8.CTRIPL	158
FSI_RX0.TRIG1	1	EPWM26.SOCA	80	CMPSSA8.CTRIPH	159
FSI_RX0.TRIG2	2	EPWM27.SOCA	81	CMPSSA9.CTRIPL	160

Table 7-138. eCAP Input Selection (continued)

Selection of eCAP Input	Select Index	Selection of eCAP Input	Select Index	Selection of eCAP Input	Select Index
FSI_RX0.TRIG3	3	EPWM28.SOCA	82	CMPSSA9.CTRIPH	161
FSI_RX1.TRIG0	4	EPWM29.SOCA	83	CMPSSB0.CTRIPL	162
FSI_RX1.TRIG1	5	EPWM30.SOCA	84	CMPSSB0.CTRIPH	163
FSI_RX1.TRIG2	6	EPWM31.SOCA	85	CMPSSB1.CTRIPL	164
FSI_RX1.TRIG3	7	EPWM0.SOCB	86	CMPSSB1.CTRIPH	165
FSI_RX2.TRIG0	8	EPWM1.SOCB	87	CMPSSB2.CTRIPL	166
FSI_RX2.TRIG1	9	EPWM2.SOCB	88	CMPSSB2.CTRIPH	167
FSI_RX2.TRIG2	10	EPWM3.SOCB	89	CMPSSB3.CTRIPL	168
FSI_RX2.TRIG3	11	EPWM4.SOCB	90	CMPSSB3.CTRIPH	169
FSI_RX3.TRIG0	12	EPWM5.SOCB	91	CMPSSB4.CTRIPL	170
FSI_RX3.TRIG1	13	EPWM6.SOCB	92	CMPSSB4.CTRIPH	171
FSI_RX3.TRIG2	14	EPWM7.SOCB	93	CMPSSB5.CTRIPL	172
FSI_RX3.TRIG3	15	EPWM8.SOCB	94	CMPSSB5.CTRIPH	173
EQEP0.SYNCQI	16	EPWM9.SOCB	95	CMPSSB6.CTRIPL	174
EQEP0.SYNCQS	17	EPWM10.SOCB	96	CMPSSB6.CTRIPH	175
EQEP1.SYNCQI	18	EPWM11.SOCB	97	CMPSSB7.CTRIPL	176
EQEP1.SYNCQS	19	EPWM12.SOCB	98	CMPSSB7.CTRIPH	177
EQEP2.SYNCQI	20	EPWM13.SOCB	99	CMPSSB8.CTRIPL	178
EQEP2.SYNCQS	21	EPWM14.SOCB	100	CMPSSB8.CTRIPH	179
EPWM0.DELACTIVE	22	EPWM15.SOCB	101	CMPSSB9.CTRIPL	180
EPWM1.DELACTIVE	23	EPWM16.SOCB	102	CMPSSB9.CTRIPH	181
EPWM2.DELACTIVE	24	EPWM17.SOCB	103	ADC0.ADCTRIPEVT0	182
EPWM3.DELACTIVE	25	EPWM18.SOCB	104	ADC0.ADCTRIPEVT1	183
EPWM4.DELACTIVE	26	EPWM19.SOCB	105	ADC0.ADCTRIPEVT2	184
EPWM5.DELACTIVE	27	EPWM20.SOCB	106	ADC0.ADCTRIPEVT3	185
EPWM6.DELACTIVE	28	EPWM21.SOCB	107	ADC1.ADCTRIPEVT0	186
EPWM7.DELACTIVE	29	EPWM22.SOCB	108	ADC1.ADCTRIPEVT1	187
EPWM8.DELACTIVE	30	EPWM23.SOCB	109	ADC1.ADCTRIPEVT2	188
EPWM9.DELACTIVE	31	EPWM24.SOCB	110	ADC1.ADCTRIPEVT3	189
EPWM10.DELACTIVE	32	EPWM25.SOCB	111	ADC2.ADCTRIPEVT0	190
EPWM11.DELACTIVE	33	EPWM26.SOCB	112	ADC2.ADCTRIPEVT1	191
EPWM12.DELACTIVE	34	EPWM27.SOCB	113	ADC2.ADCTRIPEVT2	192
EPWM13.DELACTIVE	35	EPWM28.SOCB	114	ADC2.ADCTRIPEVT3	193
EPWM14.DELACTIVE	36	EPWM29.SOCB	115	ADC3.ADCTRIPEVT0	194
EPWM15.DELACTIVE	37	EPWM30.SOCB	116	ADC3.ADCTRIPEVT1	195
EPWM16.DELACTIVE	38	EPWM31.SOCB	117	ADC3.ADCTRIPEVT2	196
EPWM17.DELACTIVE	39	SDFM0.COMP1H	118	ADC3.ADCTRIPEVT3	197
EPWM18.DELACTIVE	40	SDFM0.COMP1L	119	ADC4.ADCTRIPEVT0	198
EPWM19.DELACTIVE	41	SDFM0.COMPZ1	120	ADC4.ADCTRIPEVT1	199
EPWM20.DELACTIVE	42	SDFM0.COMP2H	121	ADC4.ADCTRIPEVT2	200
EPWM21.DELACTIVE	43	SDFM0.COMP2L	122	ADC4.ADCTRIPEVT3	201
EPWM22.DELACTIVE	44	SDFM0.COMPZ2	123	INPUTXBAR.OUT0	202
EPWM23.DELACTIVE	45	SDFM0.COMP3H	124	INPUTXBAR.OUT1	203
EPWM24.DELACTIVE	46	SDFM0.COMP3L	125	INPUTXBAR.OUT2	204
EPWM25.DELACTIVE	47	SDFM0.COMPZ3	126	INPUTXBAR.OUT3	205
EPWM26.DELACTIVE	48	SDFM0.COMP4H	127	INPUTXBAR.OUT4	206
EPWM27.DELACTIVE	49	SDFM0.COMP4L	128	INPUTXBAR.OUT5	207

Table 7-138. eCAP Input Selection (continued)

Selection of eCAP Input	Select Index	Selection of eCAP Input	Select Index	Selection of eCAP Input	Select Index
EPWM28.DELACTIVE	50	SDFM0.COMPZ4	129	INPUTXBAR.OUT6	208
EPWM29.DELACTIVE	51	SDFM1.COMP1H	130	INPUTXBAR.OUT7	209
EPWM30.DELACTIVE	52	SDFM1.COMP1L	131	INPUTXBAR.OUT8	210
EPWM31.DELACTIVE	53	SDFM1.COMPZ1	132	INPUTXBAR.OUT9	211
EPWM0.SOCA	54	SDFM1.COMP2H	133	INPUTXBAR.OUT10	212
EPWM1.SOCA	55	SDFM1.COMP2L	134	INPUTXBAR.OUT11	213
EPWM2.SOCA	56	SDFM1.COMPZ2	135	INPUTXBAR.OUT12	214
EPWM3.SOCA	57	SDFM1.COMP3H	136	INPUTXBAR.OUT13	215
EPWM4.SOCA	58	SDFM1.COMP3L	137	INPUTXBAR.OUT14	216
EPWM5.SOCA	59	SDFM1.COMPZ3	138	INPUTXBAR.OUT15	217
EPWM6.SOCA	60	SDFM1.COMP4H	139	INPUTXBAR.OUT16	218
EPWM7.SOCA	61	SDFM1.COMP4L	140	INPUTXBAR.OUT17	219
EPWM8.SOCA	62	SDFM1.COMPZ4	141	INPUTXBAR.OUT18	220
EPWM9.SOCA	63	CMPSSA0.CTRIPL	142	INPUTXBAR.OUT19	221
EPWM10.SOCA	64	CMPSSA0.CTRIPH	143	INPUTXBAR.OUT20	222
EPWM11.SOCA	65	CMPSSA1.CTRIPL	144	INPUTXBAR.OUT21	223
EPWM12.SOCA	66	CMPSSA1.CTRIPH	145	INPUTXBAR.OUT22	224
EPWM13.SOCA	67	CMPSSA2.CTRIPL	146	INPUTXBAR.OUT23	225
EPWM14.SOCA	68	CMPSSA2.CTRIPH	147	INPUTXBAR.OUT24	226
EPWM15.SOCA	69	CMPSSA3.CTRIPL	148	INPUTXBAR.OUT25	227
EPWM16.SOCA	70	CMPSSA3.CTRIPH	149	INPUTXBAR.OUT26	228
EPWM17.SOCA	71	CMPSSA4.CTRIPL	150	INPUTXBAR.OUT27	229
EPWM18.SOCA	72	CMPSSA4.CTRIPH	151	INPUTXBAR.OUT28	230
EPWM19.SOCA	73	CMPSSA5.CTRIPL	152	INPUTXBAR.OUT29	231
EPWM20.SOCA	74	CMPSSA5.CTRIPH	153	INPUTXBAR.OUT30	232
EPWM21.SOCA	75	CMPSSA6.CTRIPL	154	INPUTXBAR.OUT31	233
EPWM22.SOCA	76	CMPSSA6.CTRIPH	155	Reserved	234
EPWM23.SOCA	77	CMPSSA7.CTRIPL	156	Reserved	...
EPWM24.SOCA	78	CMPSSA7.CTRIPH	157	Reserved	256

Note

ECAPxIN has to be at least 2 * SYSCLK-cycles wide to be properly captured by the eCAP module; otherwise, the input pulse can get missed from sampling by the SYSCLK.

7.5.7.3 Description

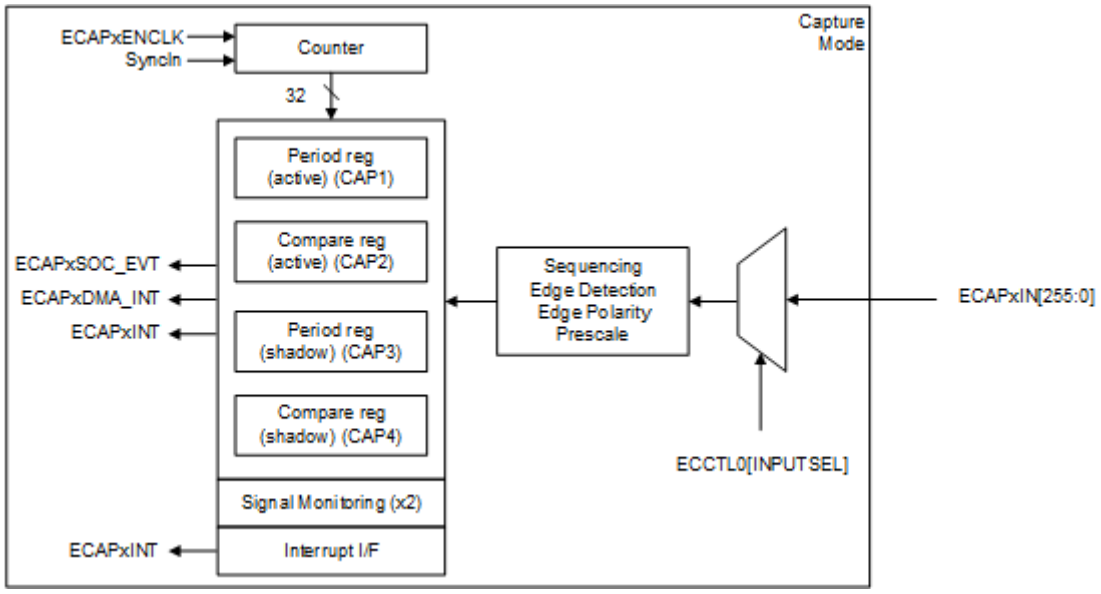
The eCAP module represents one complete capture channel that can be instantiated multiple times, depending on the target device. In the context of this guide, one eCAP channel has the following independent key resources:

- Capture inputs can be connected using the Input XBAR
- 256:1 input multiplexer
- Output XBAR is used to configure output in APWM mode
- 32-bit time base (counter)
- 4 x 32-bit time-stamp capture registers (CAP1-CAP4)
- Four-stage sequencer (modulo4 counter) that is synchronized to external events, eCAP pin rising/falling edges.
- Modulo counter status register (MODCNTRSTS) to indicate sequencer state
- Independent edge polarity (rising/falling edge) selection for all four events

- Input capture signal prescaling (from 2-62 or bypass)
- One-shot compare register (two bits) to freeze captures after 1-4 time-stamp events
- Control for continuous time-stamp captures using a four-deep circular buffer (CAP1-CAP4) scheme
- Interrupt capabilities on any of the four capture events
- Separate DMA trigger
- Signal monitoring capability for edge, pulse width, and period
- DMA event generation capability in APWM mode
- ADC SOC event generation capability, to trigger ADC conversion

7.5.7.4 Capture Mode Operation

Figure 7-276 shows the block diagram that implements the capture function.

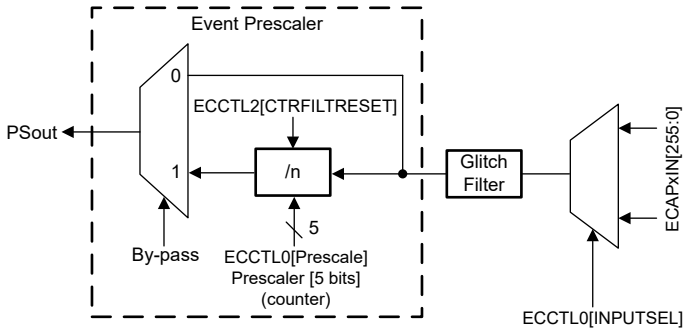


A. A single pin is shared between CAP and APWM functions. In capture mode, the pin is an input; in APWM mode, the pin is an output.

Figure 7-276. eCAP Capture Mode Block Diagram

7.5.7.4.1 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 7-277 shows a functional diagram and Figure 7-278 shows the operation of the prescale function. The event prescaler can be reset by setting the ECCTL2.CTRFILTRESET register bit.



A. When a prescale value of 1 is chosen (ECCTL1[13:9] = 0,0,0,0,0), the input capture signal bypasses the prescale logic completely.
B. The first rising edge after prescale configuration change is not passed to Capture logic, prescaler value takes into effect on the second rising edge after the configuration.

Figure 7-277. Event Prescale Control

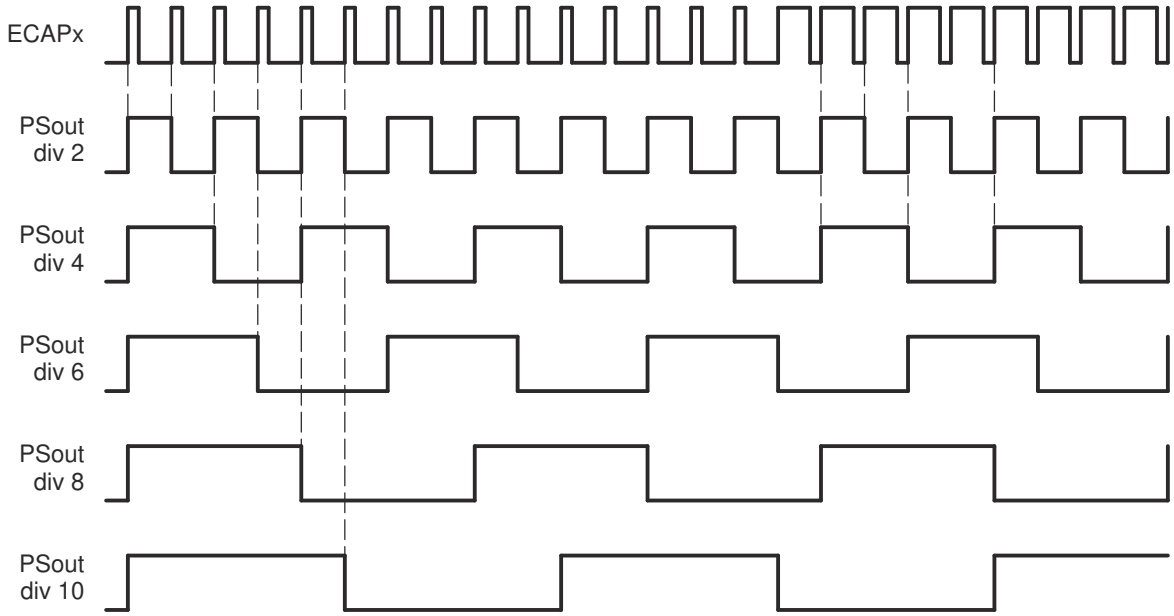


Figure 7-278. Prescale Function Waveforms

7.5.7.4.2 Glitch Filter

A glitch filter is included to reduce internal and external noise glitches on the source signal being measured by the eCAP.

The glitch filter can be used to filter out glitches of a specified time period in terms of SYSCLK cycles. The supported range is from 1 to 15 cycles. By default, the glitch filter is disabled (ECCCTL0[QUALPRD]=0) to maintain compatibility.

7.5.7.4.3 Input Capture Signal Selection

Functionality and features include:

- eCAP has up to 256 input capture sources. These enable pulse width measurements of internal design signals if necessary.
- ECCCTL0[INPUTSEL] can be used select one of the 256 inputs.

7.5.7.4.4 Modulo 4 Counter

Functionality and features include:

- The Mod4 (2 bit) counter is incremented via edge qualified events (CEVT1-CEVT4)

  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, and when equal, stops the Mod4 counter and inhibits further loads of the CAP1-CAP4 registers. This occurs during *one-shot* operation.

7.5.7.4.5 Edge Polarity Select and Qualifier

Functionality and features include:

- Four independent edge polarity (rising edge/falling edge) selection muxes are used, one for each capture event.
- Each edge (up to 4) is event qualified by the Modulo4 sequencer.
- The edge event is gated to the respective CAPx register by the Mod4 counter. The CAPx register is loaded on the falling edge.

7.5.7.4.6 Continuous/One-Shot Control

Operation of eCAP in Continuous/One-Shot mode:

- The Mod4 (2-bit) counter is incremented using edge qualified events (CEVT1-CEVT4).
- The Mod4 counter continues counting (0->1->2->3->0) and wraps around unless stopped.
- During one-shot operation, a 2-bit stop register (STOP_WRAP) is used to compare the Mod4 counter output, and when equal, stops the Mod4 counter and inhibits further loads of the CAP1-CAP4 registers. In this mode, if TSCCTR counter is configured to reset on capture event (CEVTx) by configuring ECCTL1.CTRRSTx bit, the operation still keeps resetting the TSCCTR counter on capture event (CEVTx) after the STOP_WRAP value is reached and re-arm (REARM) has not occurred.

The continuous/one-shot block controls the start, stop and reset (zero) functions of the Mod4 counter, using a mono-shot type of action that can be triggered by the stop-value comparator and re-armed using 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 CAP1-4 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 CAP1-4 registers again, providing the 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 CAP1-4 in a circular buffer sequence.

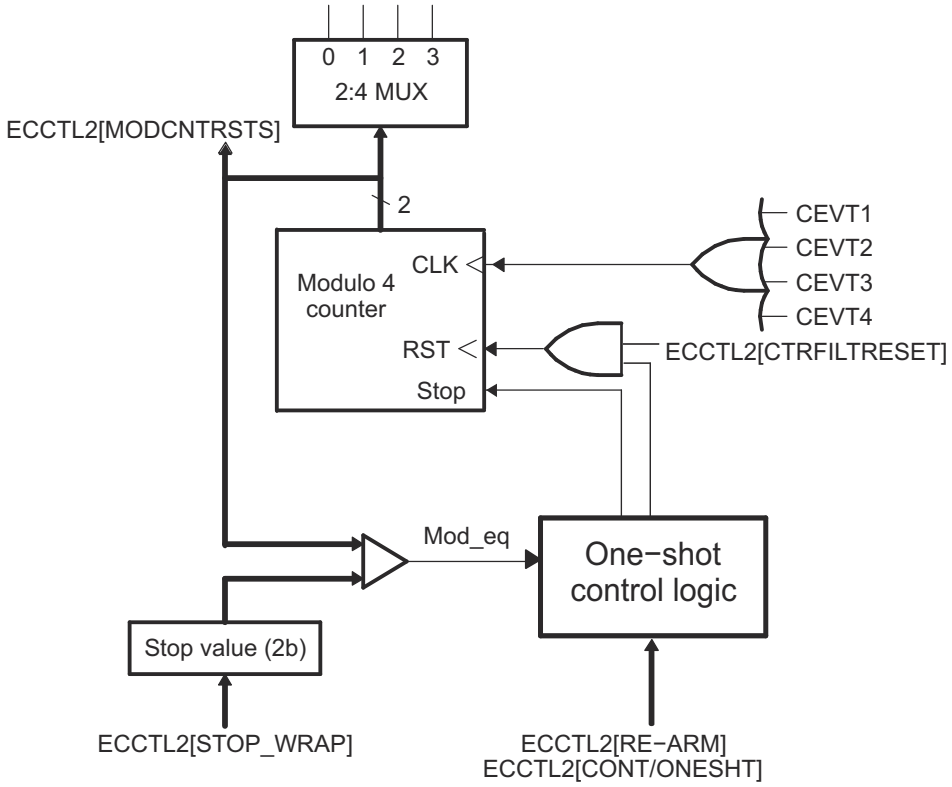


Figure 7-279. Details of the Continuous/One-shot Block

7.5.7.4.7 32-Bit Counter and Phase Control

This counter provides the time-base for event captures, and is clocked using the system clock.

A phase register is provided to achieve synchronization with other counters using 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 the counter value is reset to 0 by any of the LD1-LD4 signals.

7.5.7.4.8 CAP1-CAP4 Registers

These 32-bit registers are supplied by the 32-bit counter timer bus, CTR[0-31], and are loaded (capture a time-stamp) when their respective LD inputs are strobed.

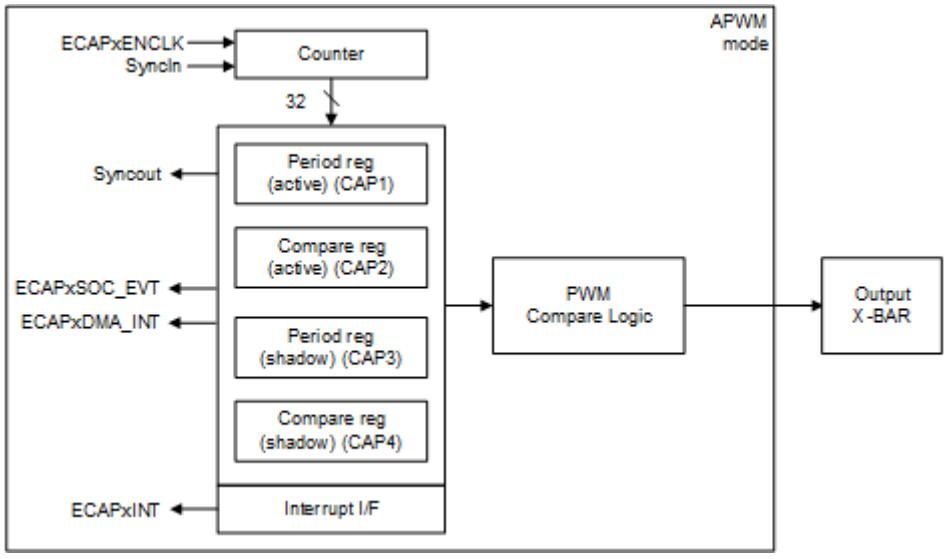
Control bit CAPLDEN can inhibit loading of the capture registers. During one-shot operation, this bit is cleared (loading is inhibited) automatically when a stop condition occurs, StopValue = Mod4.

CAP1 and CAP2 registers become the active period and compare registers, respectively, in APWM mode.

CAP3 and CAP4 registers become the respective shadow registers (APRD and ACMP) for CAP1 and CAP2 during APWM operation.

7.5.7.5 APWM Mode Operation

eCAP module is used to implement a single-channel PWM generator (with 32-bit capabilities) when the eCAP module 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 CAP1 and CAP2 registers become the active period and compare registers, respectively, while CAP3 and CAP4 registers become the period and compare shadow registers, respectively. Figure 7-280 is a high-level view of both the capture and auxiliary pulse-width modulator (APWM) modes of operation.



A. A single pin is shared between CAP and APWM functions. In capture mode, the pin is an input; in APWM mode, the pin is an output.
B. In APWM mode, writing any value to CAP1/CAP2 active registers also writes the same value to the corresponding shadow registers CAP3/CAP4. This emulates immediate mode. Writing to the shadow registers CAP3/CAP4 invokes the shadow mode.

Figure 7-280. eCAP APWM Mode Block Diagram

Main operating highlights of the APWM section:

- The time-stamp counter bus is made available for comparison by way of 2 digital (32-bit) comparators.
- When CAP1/2 registers are not used in capture mode, their contents can be used as Period and Compare values in APWM mode.
- Double buffering is achieved using shadow registers APRD and ACMP (CAP3/4). The shadow register contents are transferred over to CAP1/2 registers, either immediately upon a write, or on a CTR = PRD trigger.
- In APWM mode, writing to CAP1/CAP2 active registers also writes the same value to the corresponding shadow registers CAP3/CAP4. This emulates immediate mode. Writing to the shadow registers CAP3/CAP4 invokes the shadow mode.
- During initialization, write to the active registers for both period and compare. This automatically copies the initial values into the shadow values. For subsequent compare updates during run-time, use the shadow registers.

Figure 7-281 further descries the output of the eCAP in APWM mode based on the CMP and PRD values.

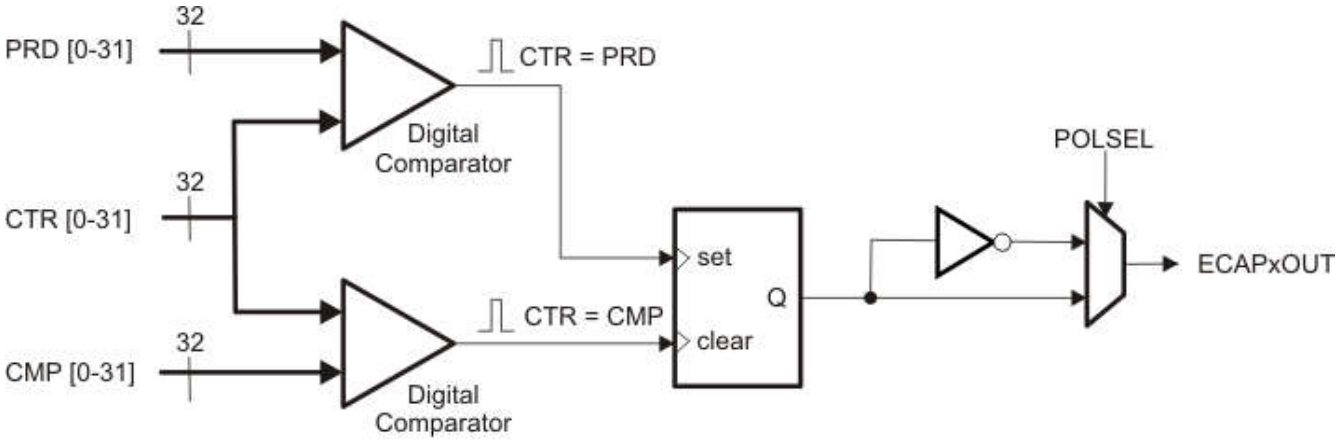


Figure 7-281. Counter Compare Operation

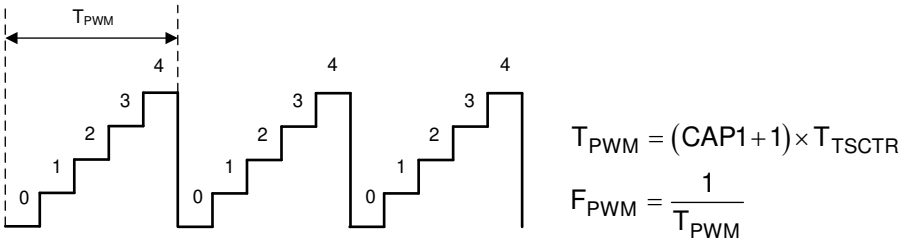


Figure 7-282. Time-Base Frequency and Period Calculation

APWM Mode Operation – Active High mode

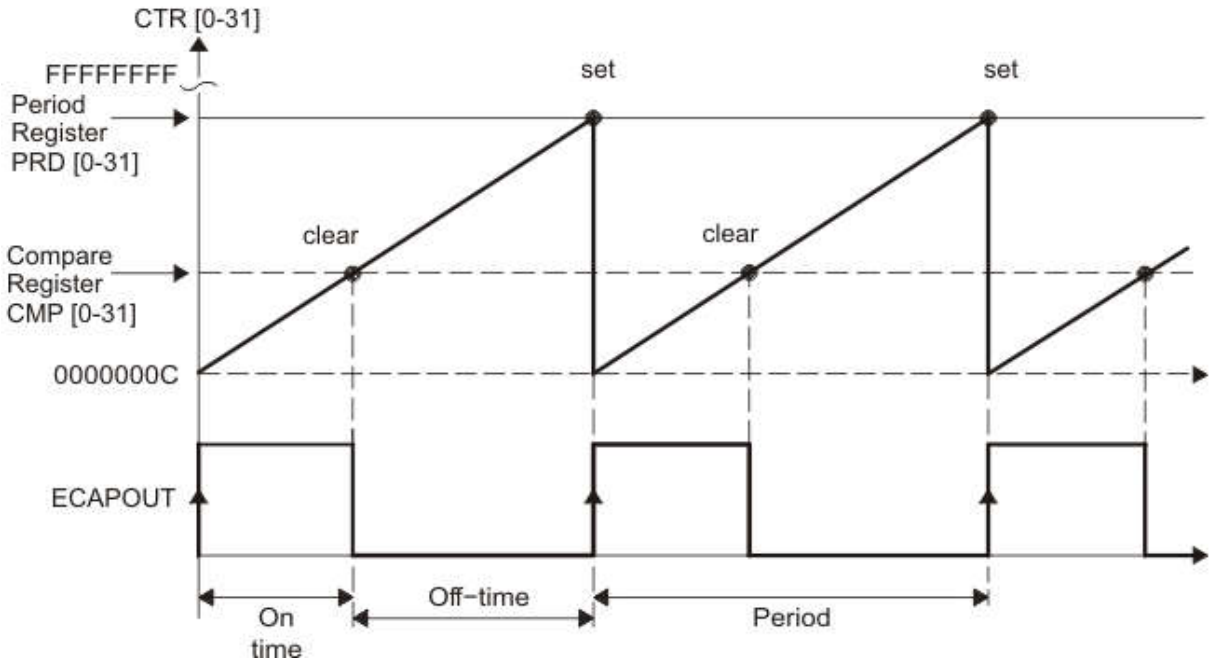


Figure 7-283. APWM Mode Operation (Active High Mode – APWMPOL == 0)

The behavior of APWM active high mode (APWMPOL == 0) is as follows:

```
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
```

APWM Mode Operation – Active Low mode

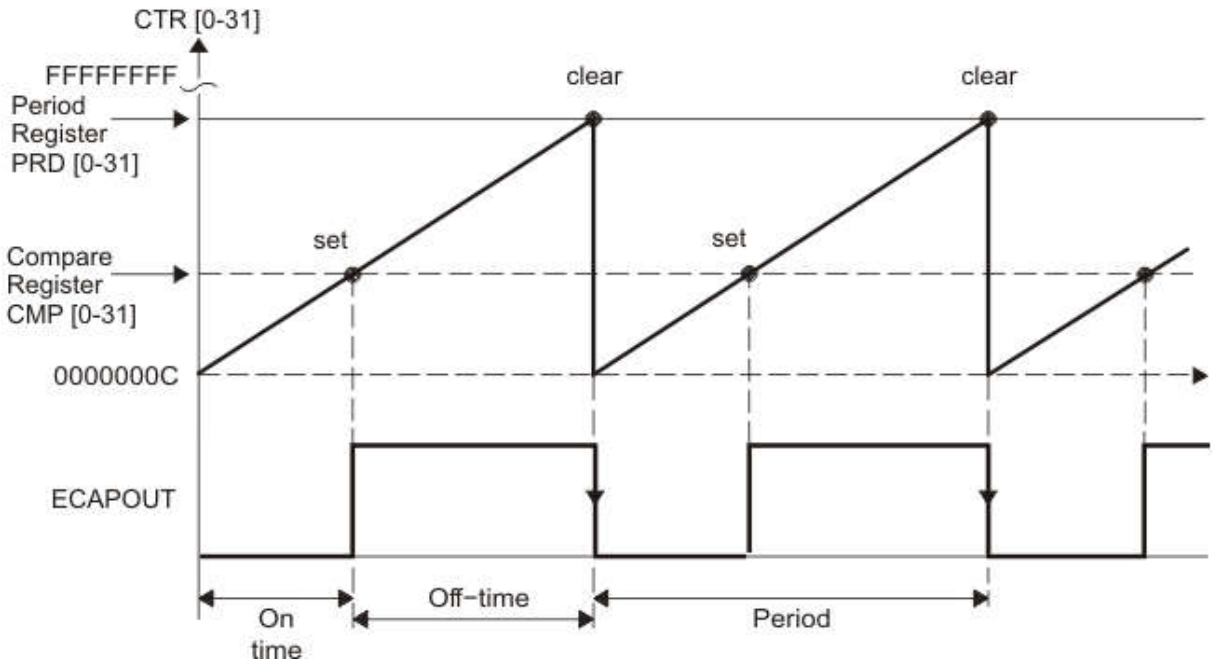


Figure 7-284. APWM Mode Operation (Active Low Mode – APWMPOL == 1) Details

The behavior of APWM active low mode (APWMPOL == 1) is as follows:

```
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
```

7.5.7.6 eCAP Synchronization and Events

External events can be used to synchronize the eCAP and send out sync, interrupt, DMA, and SOC events.

7.5.7.6.1 eCAP Synchronization

eCAP modules can be synchronized with each other by selecting a common SYNCIN source. SYNCIN source for eCAP can be either software sync-in or external sync-in. The external sync-in signal can come from the EPWM. The SWSYNC of the eCAP module is logical ORed with the SYNC signal as shown in Figure 7-285. The SYNC signal is defined by the selection of ECAPxSYNCINSEL[SEL] as shown in Figure 7-286.

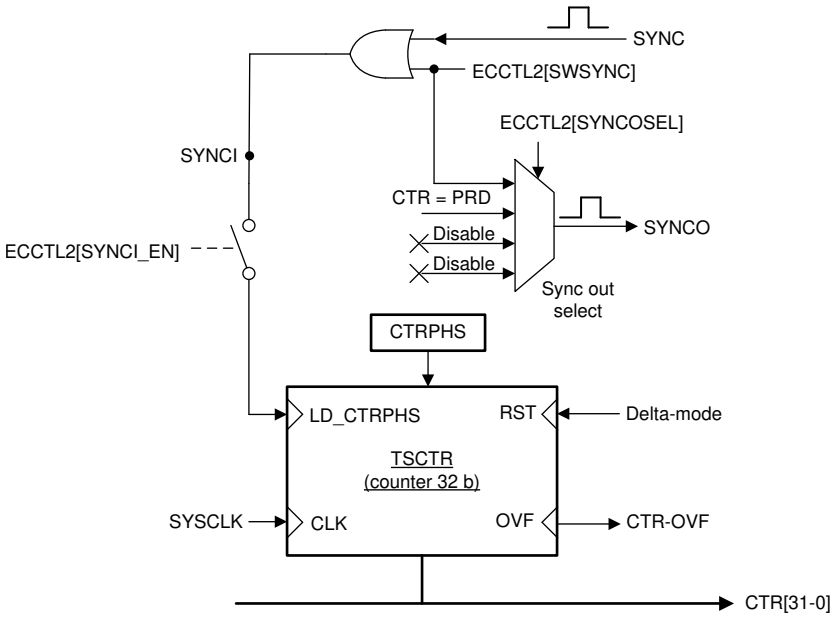


Figure 7-285. Details of the Counter and Synchronization Block

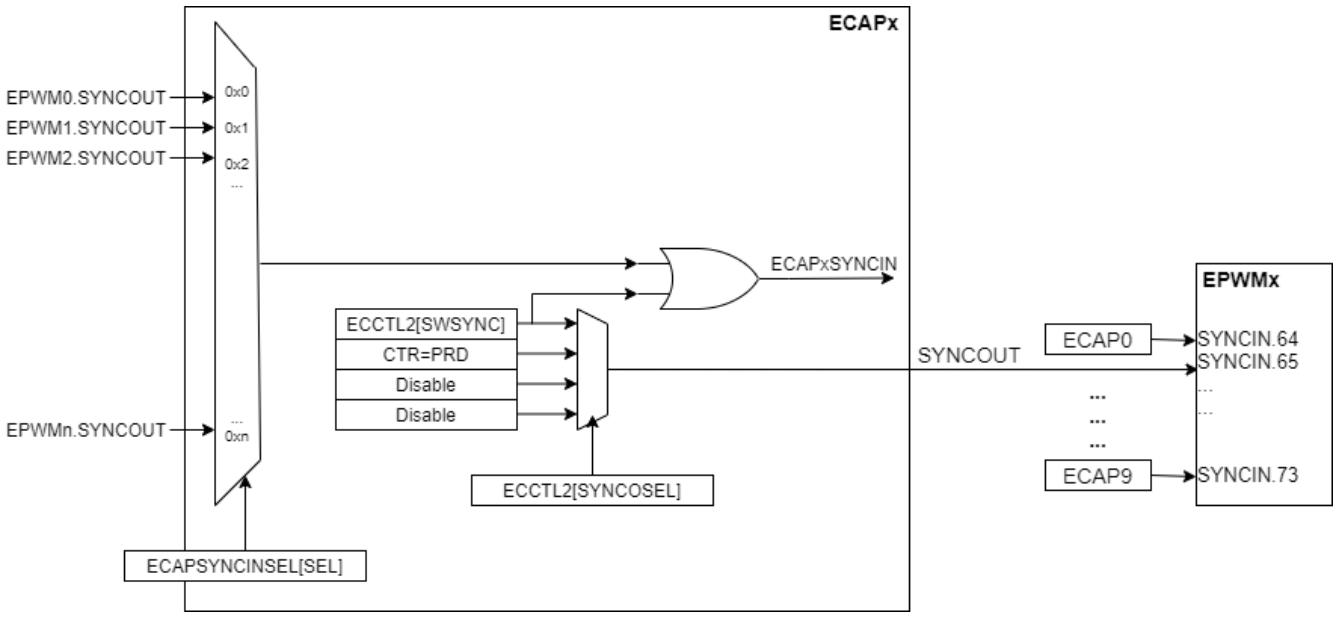


Figure 7-286. eCAP Synchronization Scheme

7.5.7.6.1.1 Example 1 - Using SWSYNC with ECAP Module

Implement the following steps to use SWSYNC with ECAP1 and ECAP2.

- Configure ECAP[1..2].ECAPSYNCINSEL.SEL = 0x0 to disable external SYNCIN coming to eCAP1.
- Configure ECAP[1..2].ECCTL2.SWSYNC = 0x1, to force Software Synchronization of the TSCTR counter.

To use SWSYNC with other eCAP modules, make sure that the previous eCAP chain is not generating a SYNCOUT signal that interferes with the software synchronization.

7.5.7.6.2 Interrupt Control

Operation and features of the eCAP interrupt control include (see Figure 7-287):

- An interrupt can be generated on capture events (CEVT1-CEVT4, CTROVF) or APWM events (CTR = PRD, CTR = CMP).
- An interrupt can be generated on signal monitoring errors (MUNIT_1_ERROR_EVT1, MUNIT_1_ERROR_EVT1, MUNIT_2_ERROR_EVT1, MUNIT_2_ERROR_EVT2)
- A counter overflow event (FFFFFFFF->00000000) is also provided as an interrupt source (CTROVF).
- The capture events are edge and sequencer-qualified (ordered in time) by the polarity select and Mod4 gating, respectively.
- One of these events can be selected as the interrupt source (from the eCAPx module) going to the PIE
- Seven interrupt events (CEVT1, CEVT2, CEVT3, CEVT4, CNTOVF, CTR=PRD, CTR=CMP) can be generated.
- An additional four interrupt events (MUNIT_1_ERROR_EVT1, MUNIT_1_ERROR_EVT1, MUNIT_2_ERROR_EVT1, MUNIT_2_ERROR_EVT2) can be generated from the signal monitoring unit.
- The interrupt enable register (ECEINT) is used to enable/disable individual interrupt event sources. The interrupt flag register (ECFLG) indicates if any interrupt event has been latched and contains the global interrupt flag bit (INT). An interrupt pulse is generated to the PIE only if any of the interrupt events are enabled, the flag bit is 1, and the INT flag bit is 0. The interrupt service routine must clear the global interrupt flag bit and the serviced event using the interrupt clear register (ECCLR) before any other interrupt pulses are generated. All interrupt flags are cleared upon an event filter reset by writing a 1 to ECCTL2[CLRFILTRESET]. To force an interrupt event, use the interrupt force register (ECFRC). This is useful for test purposes.

Note

The CEVT1, CEVT2, CEVT3, CEVT4 flags are only active in capture mode (ECCTL2[CAP/APWM == 0]). The CTR=PRD, CTR=CMP flags are only valid in APWM mode (ECCTL2[CAP/APWM == 1]). CNTOVF flag is valid in both modes.

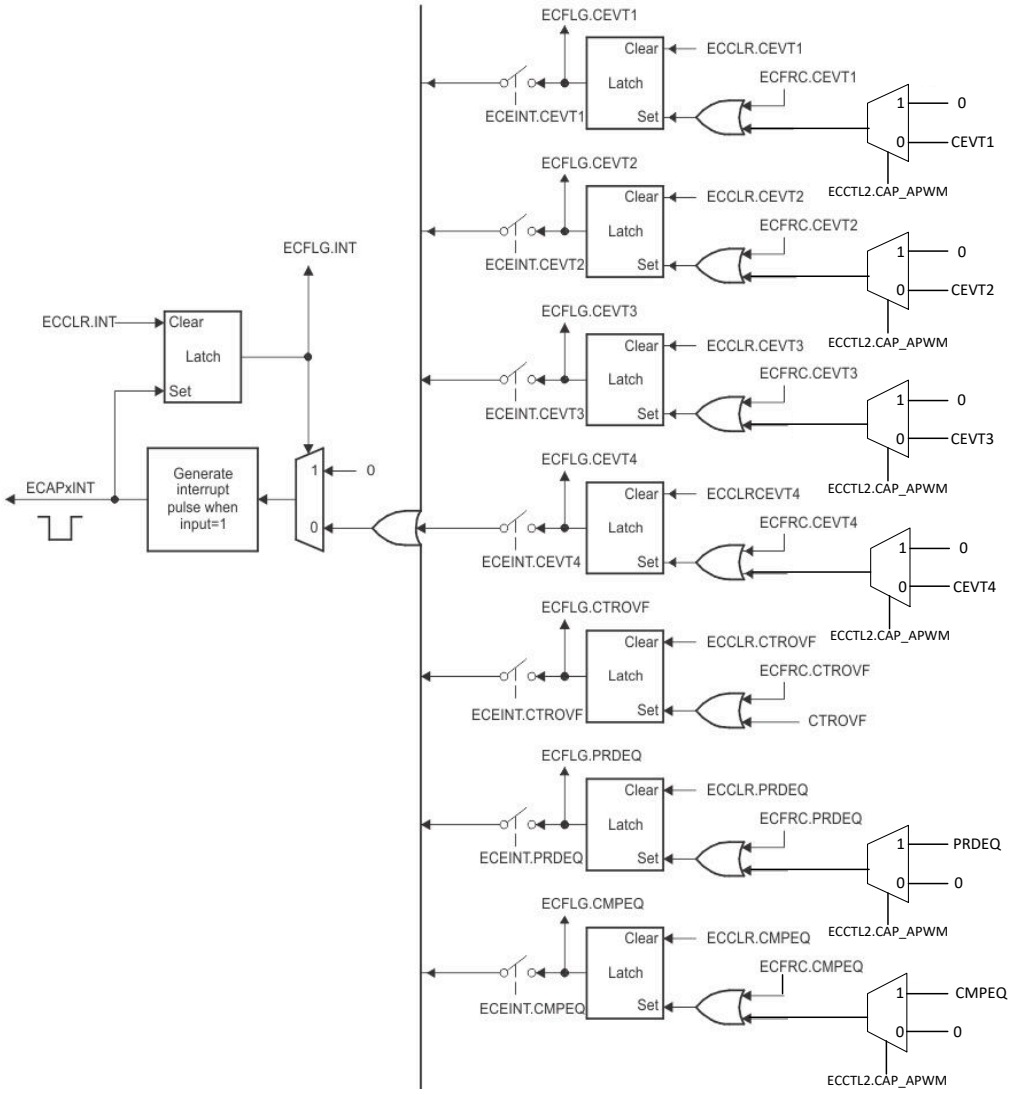


Figure 7-287. Interrupts in eCAP Module

7.5.7.6.3 DMA Interrupt

On Type 0 eCAP modules, the CPU was required to begin data transfers using DMA. New to the Type 1 eCAP, a separate DMA Trigger (ECAP_DMA_INT) enables continuous transfer of capture data from eCAP registers to on-chip memory using DMA. Any one of the four available interrupt events (CEVT1, CEVT2, CEVT3, and CEVT4) can be selected as the trigger source for ECAP_DMA_INT using ECCTL2 [DMAEVTSEL].

New to the Type 3 eCAP is the ability to trigger DMA events in APWM mode. Any one of three available events (period match, compare match, or both) can be selected as the trigger source for ECAP_DMA_INT using ECCTL2 [DMAEVTSEL].

Note

ECAPxINT interrupt cannot be used as DMA trigger because after first interrupt, no further ECAPxINT is generated until CPU clears ECFLG[INT] in interrupt service routine which is not possible on DMA without CPU intervention.

7.5.7.6.4 ADC SOC Event

Type 3 introduces the capability to generate ADC SOC events in capture mode and in APWM mode of operation. The ability to start ADC conversions allows for increased APWM functionality, as well as the ability to synchronize capture events with ADC samples.

In capture mode, one of the four available interrupt events (CEVT1, CEVT2, CEVT3, and CEVT4) can be selected as ECAP_SOC_EVT using ECCCTL0[SOCEVTSEL].

In APWM mode, any one of three available events (period match, compare match, or both) can be selected as ECAP_SOC_EVT using ECCCTL0[SOCEVTSEL].

7.5.7.6.5 Shadow Load and Lockout Control

In capture mode, this logic inhibits (locks out) any shadow loading of CAP1 or CAP2 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 CAP1 or CAP2 immediately upon writing a new value.
- On period equal, CTR[31:0] = PRD[31:0].

7.5.7.7 Signal Monitoring Unit

The signal monitoring unit can be used for edge, pulse width, and period monitoring of eCAP input signals. This allows for detection that is useful for many applications. For example, ePWM pulse width boundary monitoring can be accomplished for safety applications.

The high-level features of the signal monitoring unit include:
- Measure pulse width (high or low) and check if it is in expected range
- Measure period (rise-to-rise or fall-to-fall) and check if it is in expected range
- Monitor signal edge (rise or fall) and check if it occurs in a user-programmed time window

7.5.7.7.1 Pulse Width and Period Monitoring

The signal monitoring unit has the ability to measure pulse width (either low or high) or period (rise-to-rise edge or fall-to-fall edge) and automatically generate an error when the pulse width is outside of a programmable expected range.

The expected pulse width range is programmable using the following configuration registers (or their respective shadow registers):
- MUNIT_#_MIN programs the minimum pulse width capture value
- MUNIT_#_MAX programs the maximum pulse width capture value

Any pulse width outside of these programmed bounds triggers one of two error events:
- MUNIT_#_ERROR_EVT1 generated when measured pulse width is less than MUNIT_#_MIN
- MUNIT_#_ERROR_EVT2 generated when measured pulse width is greater than MUNIT_#_MAX

The following diagram provides an example in which the measured pulse width exceeds the MAX value, generating an ERROR_EVT2 event.

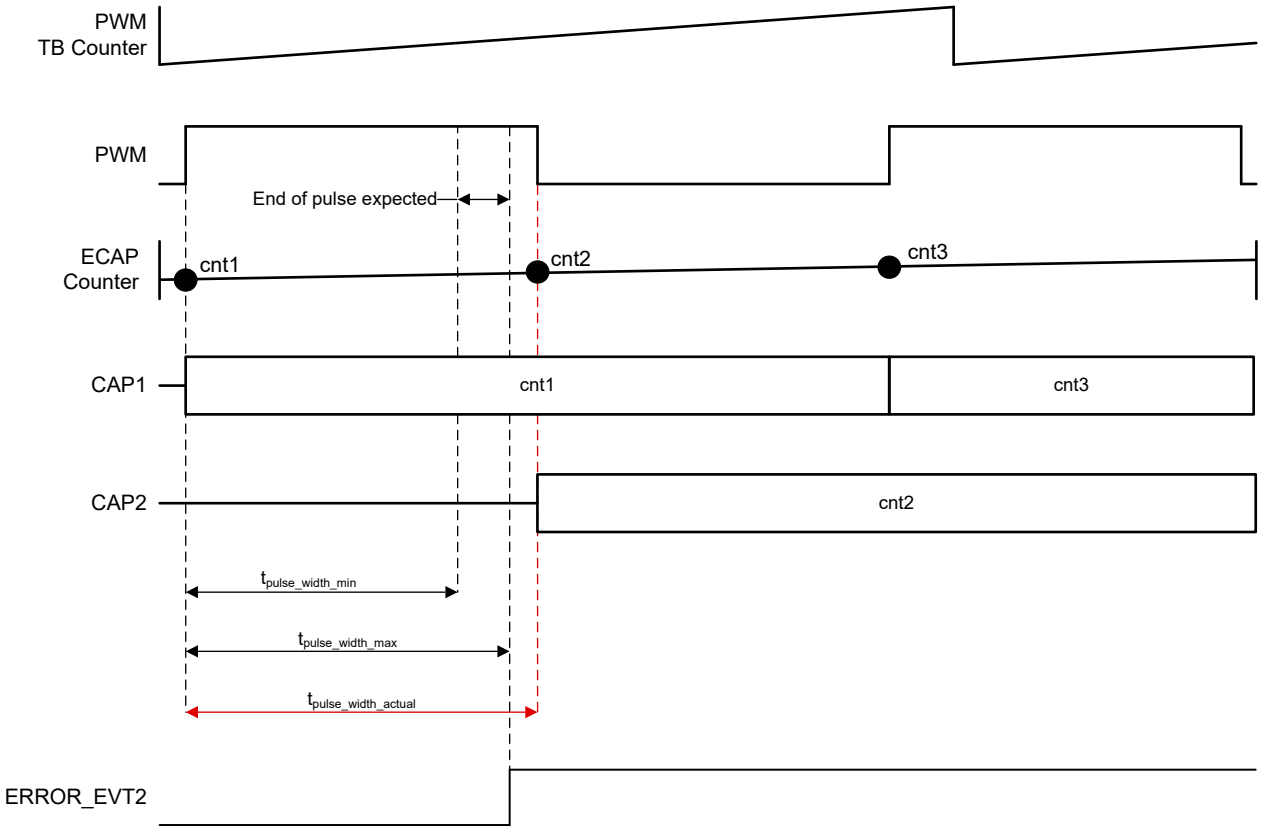


Figure 7-288. eCAP Signal Monitoring Unit Pulse Width Error Example

Configuration Requirements

To enable this mode, the following settings must be configured:

- Absolute mode must be set for the eCAP counter, so that the counter is free running and does not get reset on any capture events
- Continuous mode must be enabled (one-shot mode can be used, but is not recommended given the short duration)
- Sync feature for the counter must be disabled (ECCTL2.SYNCI_EN = 0)
- A minimum of two captures must be enabled (ECCTL2.STOP_WRAP >= 1, and at least CAP1 and CAP2 enabled)
- Capture Edge (ECCTL1.CAPxPOL) of used capture modules (any of CAP1 to CAP4) must be configured to capture two edges of interest
  - High pulse: one rising edge and one falling edge
  - Low pulse: one rising edge and one falling edge
  - Period rise-to-rise: two rising edges
  - Period fall-to-fall: two falling edges

Note

If a pulse width is greater than the MAX value, a second edge can arrive late or never even occur. Because of this, the DISABLE_EARLY_MAX_ERR field in the MUNIT_#_CTL register can be used to choose when a MAX error occurs. By setting the bit to 0, an error is generated as soon as the pulse width is greater than the specified maximum value. By setting the bit to 1, an error is generated when the second event has occurred.

7.5.7.7.2 Edge Monitoring

The signal monitoring unit has the ability to monitor and check if a rise or fall edge occurs within a specified time window and automatically generate an error when an edge occurs outside of this window.

The time window of an expected edge event can be programmed using the following configuration registers (or their respective shadow registers):

- MUNIT_#_MIN programs the minimum pulse width capture value
- MUNIT_#_MAX programs the maximum pulse with capture value

Any edge that occurs outside of these programmed bounds triggers the following error event:

- MUNIT_#_ERROR_EVT1 generated when edge occurs outside the bounds of MUNIT_#_MIN and MUNIT_#_MAX.

Additionally, ERROR_EVT2 is generated if either MIN or MAX did not occur between two sync events.

Configuration Requirements

To enable this mode, the following settings must be configured:

- The eCAP counter must be synced with an ePWM module
- Absolute mode must be set for the eCAP counter, so that the counter is free running and does not get reset on any capture events
- Continuous mode can be enabled (one-shot mode can be used, but is not recommended given the modes short duration)
- A minimum of one capture can be enabled (ECCTL2.STOP_WRAP >= 0, and at least CAP1 enabled)
- Capture Edge (ECCTL1.CAPxPOL) of used capture modules (any of CAP1 to CAP4) must be configured to capture an edge of interest

Note

The following are important considerations when configuring the edge monitoring feature:

- If the ePWM counter or eCAP counter are loaded with a non-zero phase value, the MIN and MAX values must be adjusted accordingly in SW. This also applies when the glitch filter is enabled, as the glitch filter delays the signal by QUALPRD+1
- The edge monitoring logic restarts on a sync event. This is to avoid any deadlock in case MIN, MAX, or both events do not occur between two sync events. ERROR_EVT2 is generated, if MIN or MAX match did not occur between two sync events
- The time window defined using MIN and MAX can not cross the sync boundary
- MIN and MAX are counter values (or number of clock cycles for a 200 MHz system clock). For width monitoring, the pulse/period width is between MIN and MAX no. of counter counts (or MIN*5 ns < pulse/period < MAX*5 ns). For edge monitoring, the edge is expected to occur between counter values of MIN and MAX after the sync (or MIN*5 ns < edge < MAX*5 ns, with reference to sync).

The following diagram provides an example in which a rising edge does not occur during the expected window, generating an ERROR_EVT1 event.

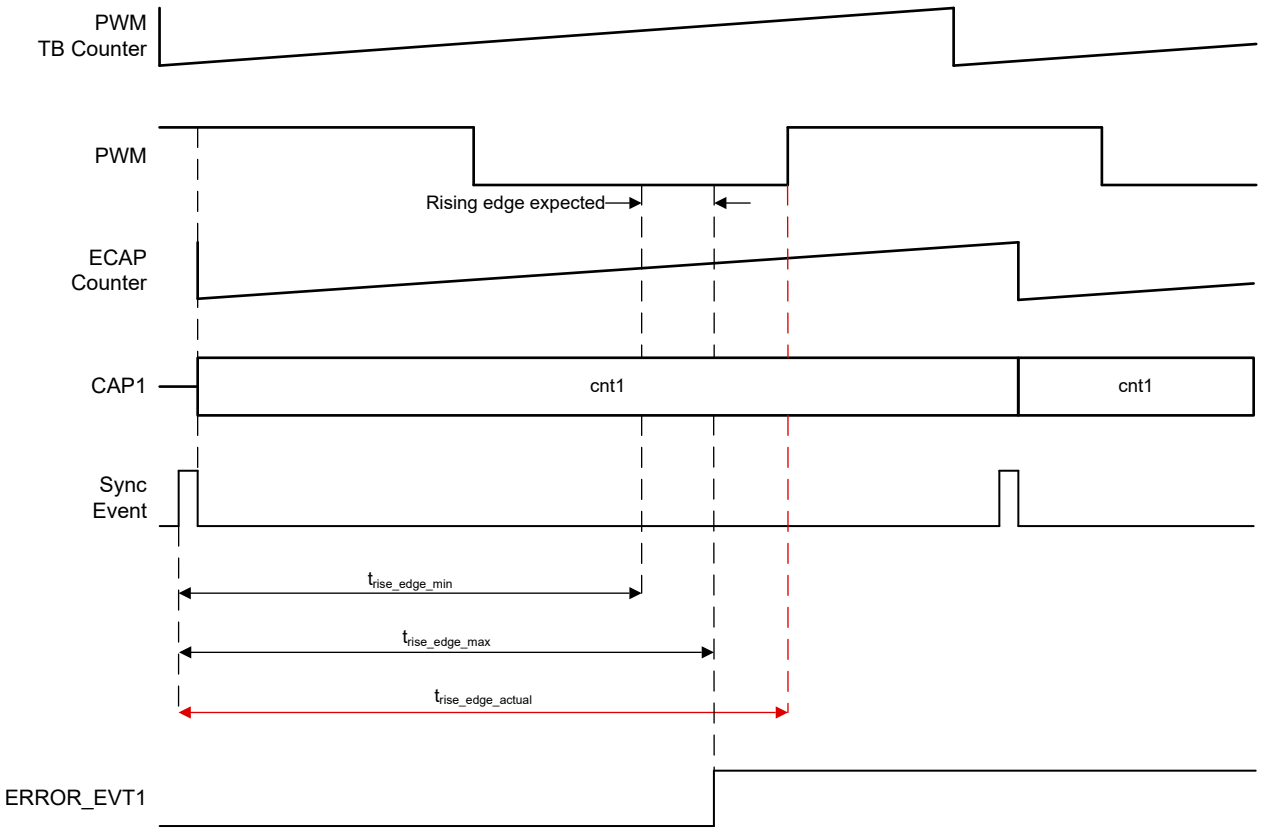


Figure 7-289. eCAP Signal Monitoring Unit Edge Error Example

7.5.7.7.3 Error Events

When a signal monitoring error occurs, the signal monitoring is disabled by clearing MUNIT_x_CTL.EN. In addition, further captures are disabled by clearing ECCTL1.CAPLDEN, but time stamp values in CAP1..4 are retained for debug purpose. To re-enable signal monitoring MUNIT_x_CTL.EN and ECCTL1.CAPLDEN need to be set again. CEVTx is generated even after an error is detected and further captures are disabled.

7.5.7.7.4 Disabling the Signal Monitoring Unit

When monitoring the PWMs, PWMs can be tripped by the external trip event that is forced to a known state. Under this condition, PWM signal monitoring is disabled temporarily until the trip condition is cleared. To support this feature, PWM trip signals are brought into eCAP module through external XBARs. In addition, an internal MUX is provided to select one of many XBAR signals. Signal monitoring is disabled as long on selected trip signal is active. Note that signal monitoring is automatically enabled when trip condition is cleared.

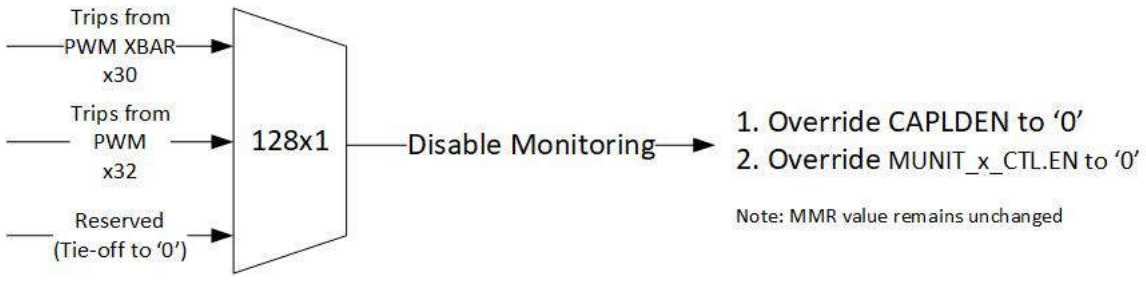


Figure 7-290. ECAP Signal Monitoring Unit Trip Signals

Note

EPWM trips are asynchronous in nature, PWM are tripped asynchronously and immediately. As a result of this, there can be a race condition that can lead to signal monitoring error. This can't be avoided and has to be handled in SW.

7.5.7.7.5 Shadow Control

Shadow registers for MIN and MAX values enable the application to change these values dynamically as the PWM configuration changes. However, shadow to active loading need to happen at certain point of time to keep these in sync with ePWM module. Global load strobe (EPWMx.GLDSTRB) from ePWM is used for this purpose. Since a given eCAP module can be associated with any ePWM module, a mux is provided select one of EPWMx.GLDSTRB in a system.

Shadow registers are copied to active registers on following events:

- SW event by writing '1' to MUNIT_{#}_SHADOW_CTL.SWSYNC
  - Usage: User programs shadow registers and writes '1' to MUNIT_{#}_SHADOW_CTL.SWSYNC to copy these values to active registers
- On eCAP sync event selected by ECAPSYNCINSEL.SEL
- On ePWM Global Load event selected by MUNIT_COMMON_CTL.GLDSTRBSEL

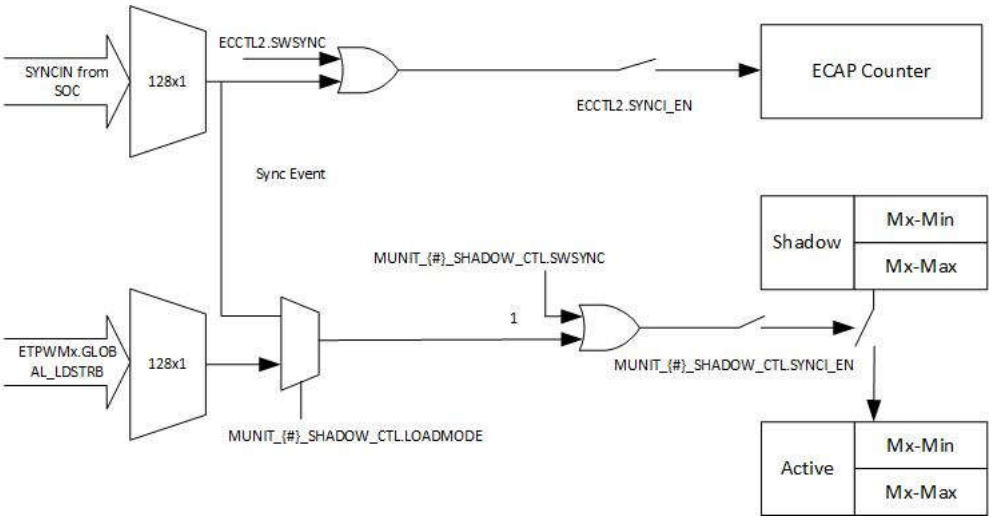


Figure 7-291. ECAP Signal Monitoring Unit Shadow Control

Note

If shadow to active event occurs while signal monitoring in the midst of pulse (after first edge has occurred) or edge (after time window started) monitoring, the current check gets aborted and new values then take effect from next pulse or sync cycle respectively. This is to make sure that false errors are not generated.

7.5.7.7.6 Trip Signal

Trip signal is generated upon signal monitoring errors. All the signal monitoring error events are OR-ed and provided as a trip output. The trip signal remains active until interrupt flags are cleared in ECFLG register. Trip cannot be disabled in eCAP, instead the trip has to be deselected in external XBAR if there is no intent to use the feature.

The ECAPx.TRIPOUT signal can be used to trip the EPWM modules. This can cause the EPWMx.TRIPOUT signal, which is fed back to eCAP module to also trip, which can disable the monitoring function and also cause a false trip if this feature is enabled. Therefore, if ECAPx.TRIPOUT is used, it is recommended that EPWMx.TRIPOUT be disabled.

7.5.7.8 Application of the eCAP Module

The following sections provide applications examples to show how to operate the eCAP module.

7.5.7.8.1 Example 1 - Absolute Time-Stamp Operation Rising-Edge Trigger

Figure 7-292 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 FFFFFFFF (maximum value), the Mod4 counter wraps around to 00000000 (not shown in Figure 7-292), if this occurs, the CTROVF (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 fourth event); hence, event CEVT4 can conveniently be used to trigger an interrupt and the CPU can read data from the CAPx registers.

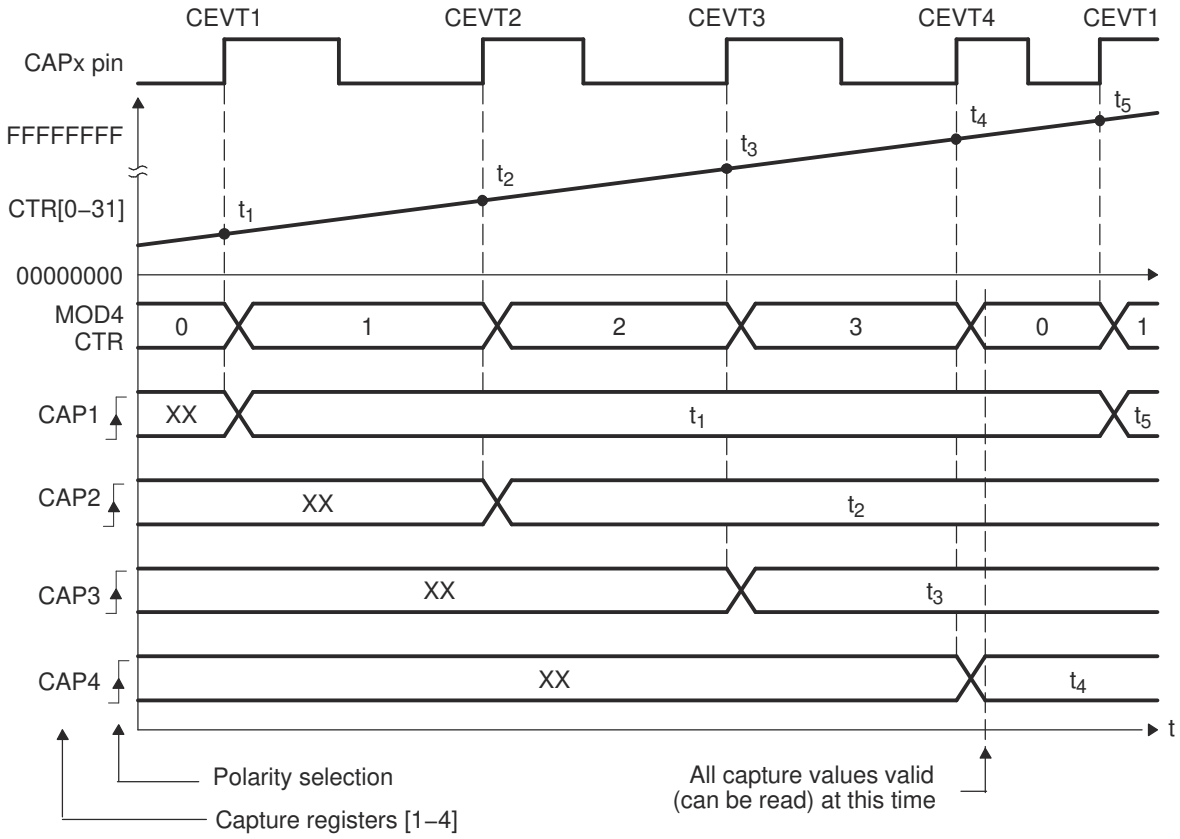


Figure 7-292. Capture Sequence for Absolute Time-stamp and Rising-Edge Detect

7.5.7.8.2 Example 2 - Absolute Time-Stamp Operation Rising- and Falling-Edge Trigger

In Figure 7-293, 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, that is: Period1 = $t_3 – t_1$, Period2 = $t_5 – t_3$, …and so on. Duty Cycle1 (on-time %) = $(t_2 – t_1)$ / Period1 x 100%, and so on. Duty Cycle1 (off-time %) = $(t_3 – t_2)$ / Period1 x 100%, and so on.

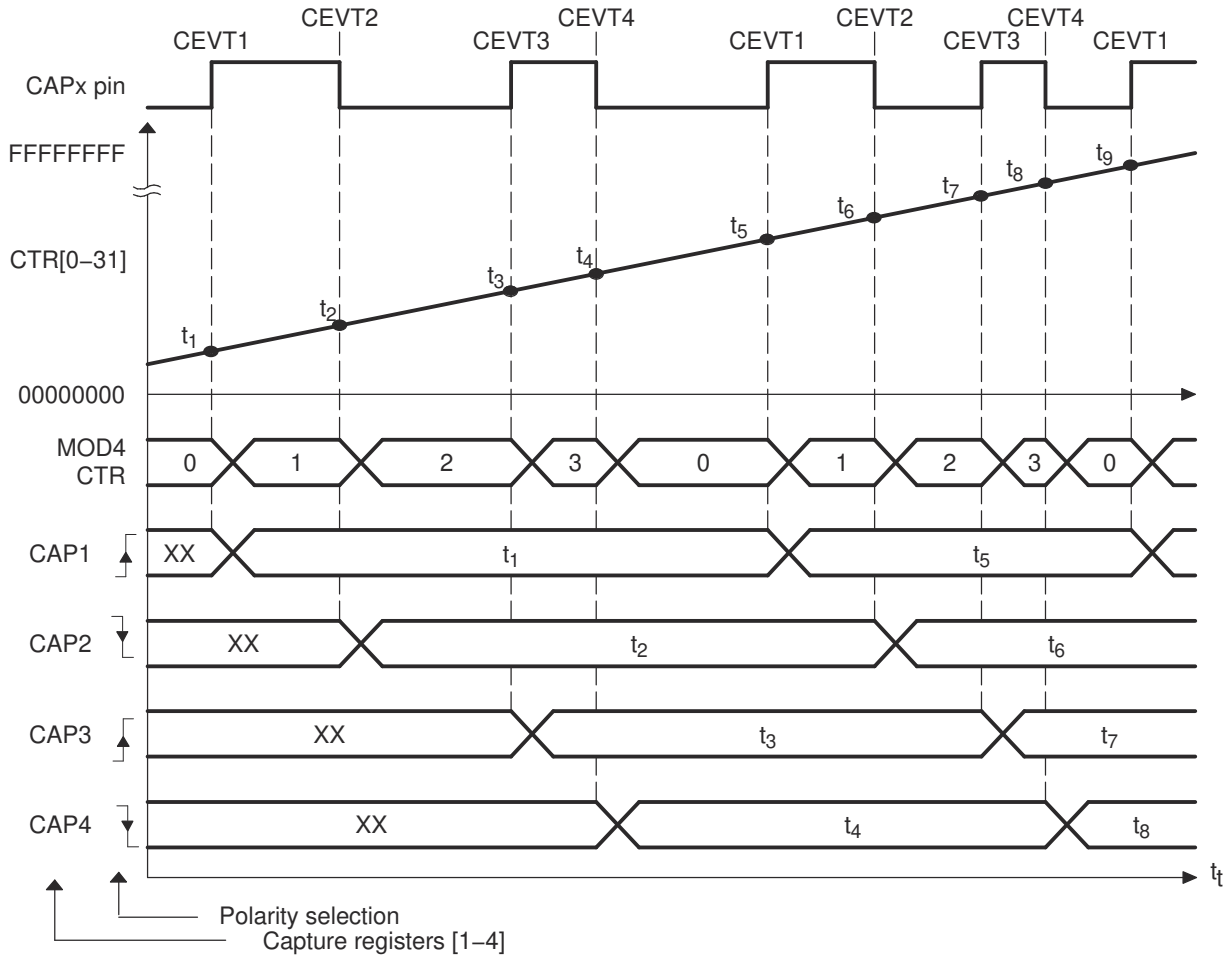


Figure 7-293. Capture Sequence for Absolute Time-stamp with Rising- and Falling-Edge Detect

7.5.7.8.3 Example 3 - Time Difference (Delta) Operation Rising-Edge Trigger

Figure 7-294 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 FFFFFFFF (maximum value), before the next event, the Mod4 counter wraps around to 00000000 and continues, a CNTOVF (counter overflow) flag is set, and an interrupt (if enabled) occurs. The advantage of Delta-time mode is that the CAPx contents directly give timing data without the need for CPU calculations, that is, Period1 = T_1, Period2 = T_2, and so on. As shown in Figure 7-294, 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.

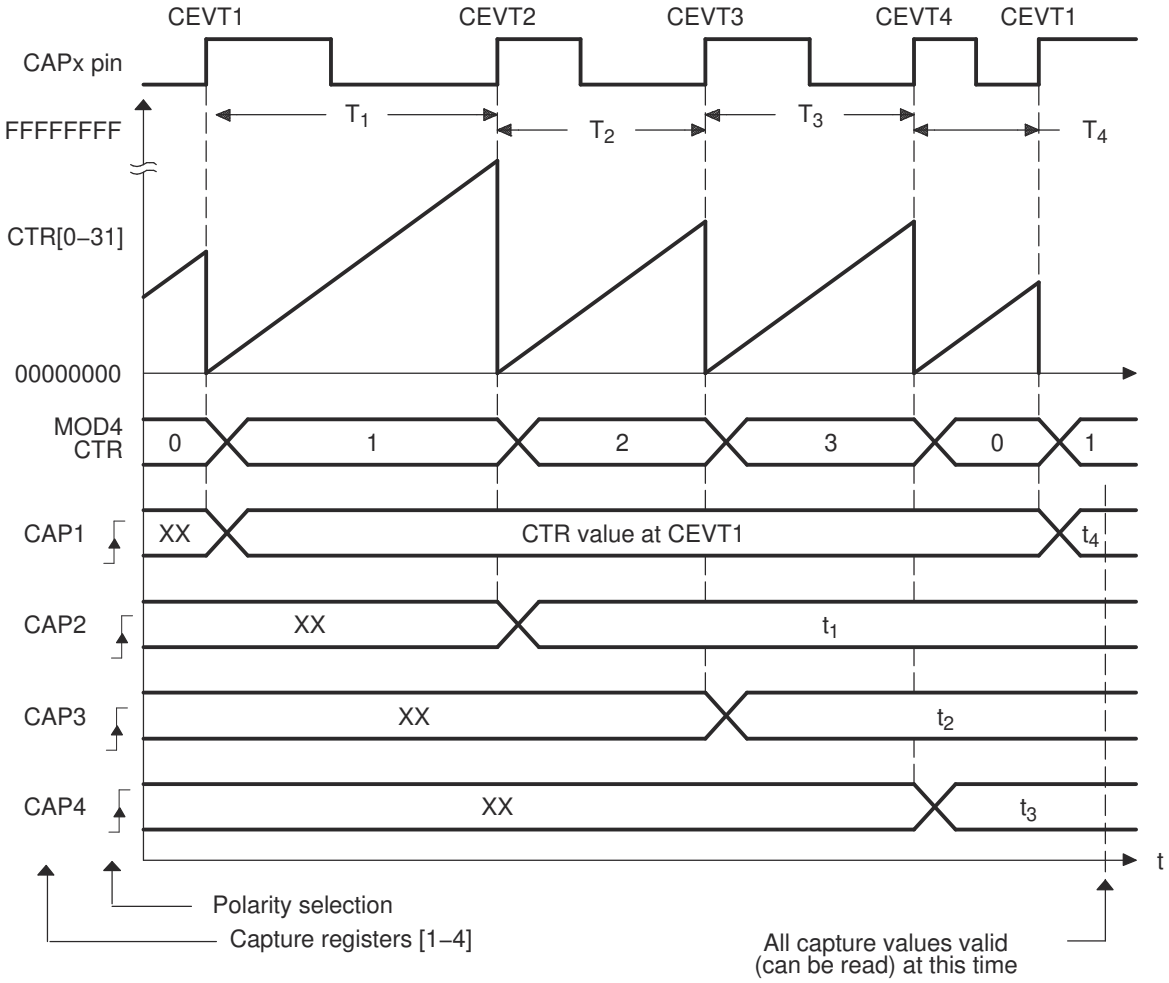


Figure 7-294. Capture Sequence for Delta Mode Time-stamp and Rising Edge Detect

7.5.7.8.4 Example 4 - Time Difference (Delta) Operation Rising- and Falling-Edge Trigger

In Figure 7-295, 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, that is: Period1 = T_1+T_2, Period2 = T_3+T_4, and so on. Duty Cycle1 (on-time %) = T_1 / Period1 x 100%, Duty Cycle1 (off-time %) = T_2 / Period1 x 100%, and so on.

During initialization, write to the active registers for both period and compare. This action automatically copies the init values into the shadow values. For subsequent compare updates during run-time, the shadow registers must be used.

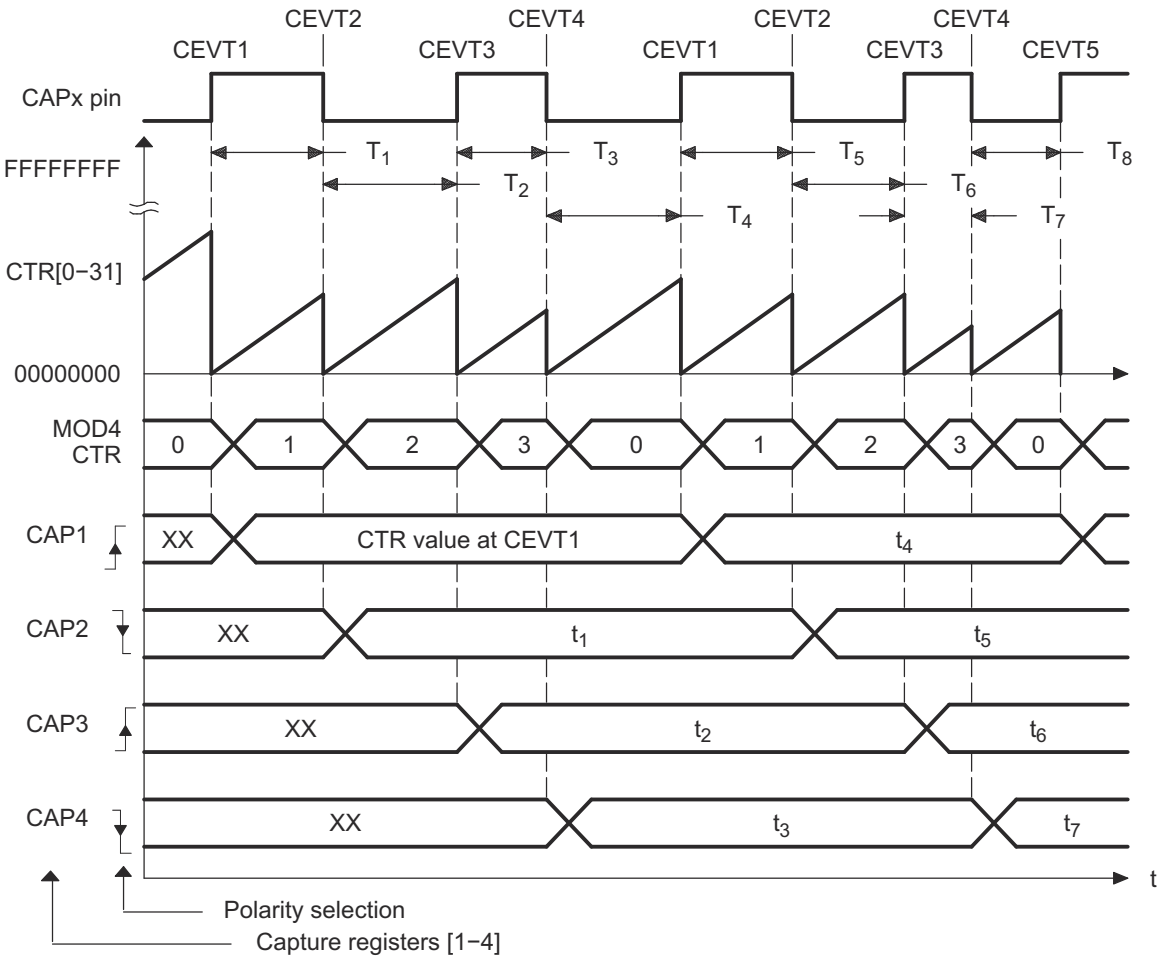


Figure 7-295. Capture Sequence for Delta Mode Time-stamp with Rising- and Falling-Edge Detect

7.5.7.9 Application of the APWM Mode

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 the APWMx output pin. 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.

7.5.7.9.1 Example 1 - Simple PWM Generation (Independent Channels)

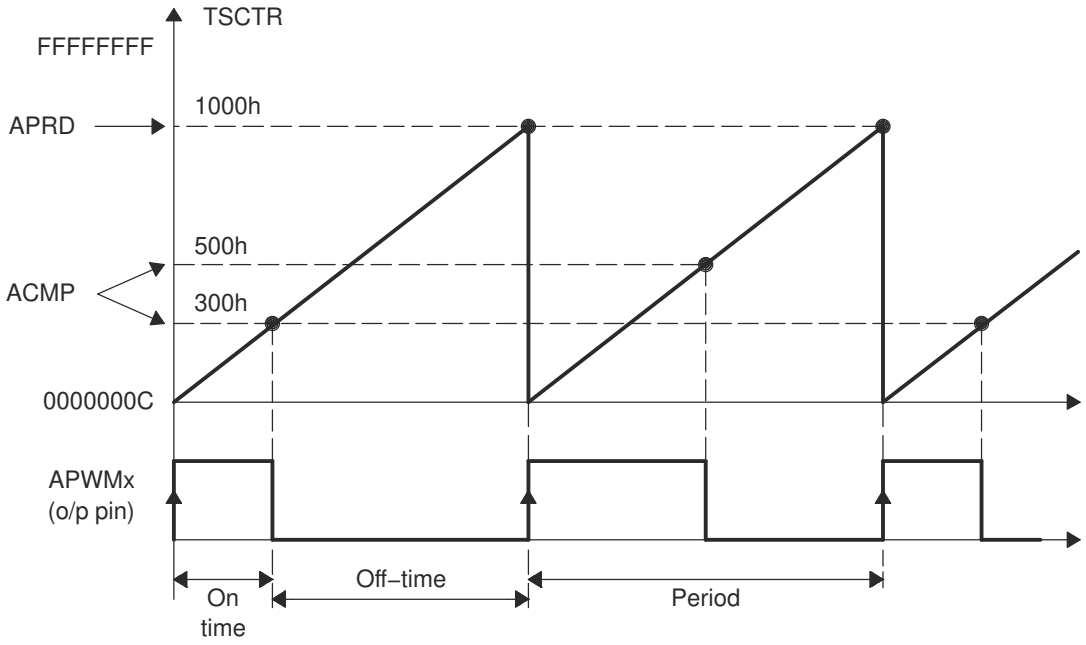


Figure 7-296. PWM Waveform Details of APWM Mode Operation

7.5.7.10 eCAP Programming Guide

Driver Information

Driver features are available at the ECAP driver page.

Software API Information

The eCAP driver provides an API to configure the eCAP module. Full documentation is located on APIs for ECAP.

Example Usage

The below links show examples on how to use the eCAP module:

- ECAP Capture PWM
- ECAP APWM Mode

7.5.8 Enhanced Quadrature Encoder Pulse (eQEP)

The enhanced quadrature encoder pulse (eQEP) module 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 a high-performance motion and position-control system.

7.5.8.1 Introduction

An incremental encoder disk is patterned with a track of slots along the periphery, as shown in Figure 7-297. These slots create an alternating pattern of dark and light lines. The disk count is defined as the number of dark and 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

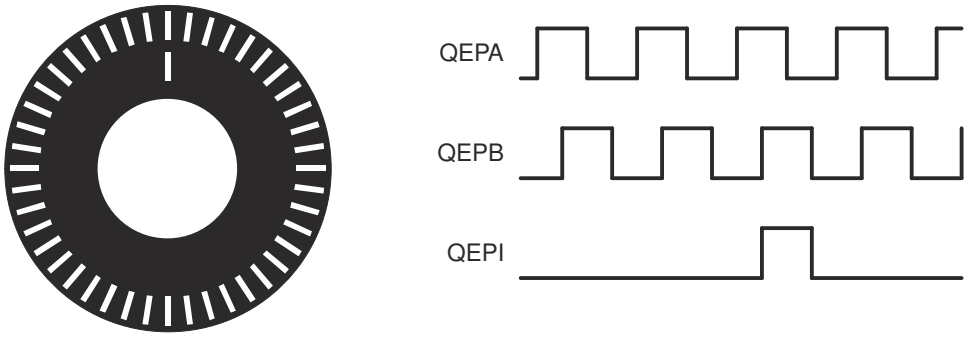
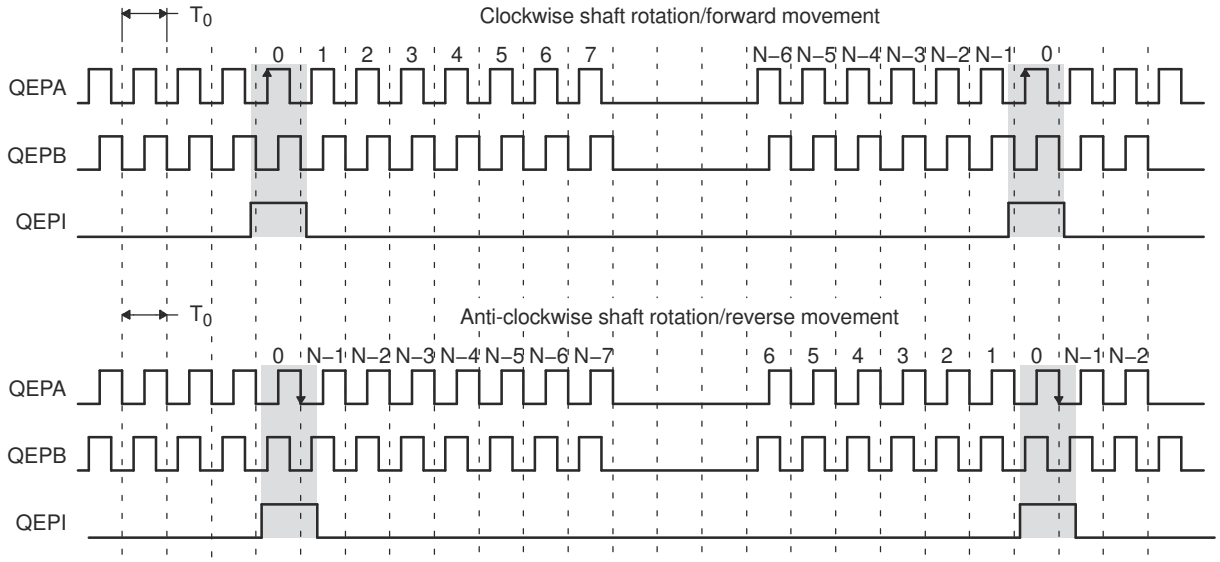


Figure 7-297. Optical Encoder Disk

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 detected 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° 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 conversely, as shown in Figure 7-298.



Legend: N = lines per revolution

Figure 7-298. QEP Encoder Output Signal for Forward/Reverse Movement

The encoder wheel typically makes one revolution for every revolution of the motor, or the wheel can 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 166.6kHz, so

by measuring the frequency of either the QEPA or QEPB output, the processor can determine the velocity of the motor.

Quadrature encoders from different manufacturers come with two forms of index pulse (gated index pulse or ungated index pulse) as shown in Figure 7-299. 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.

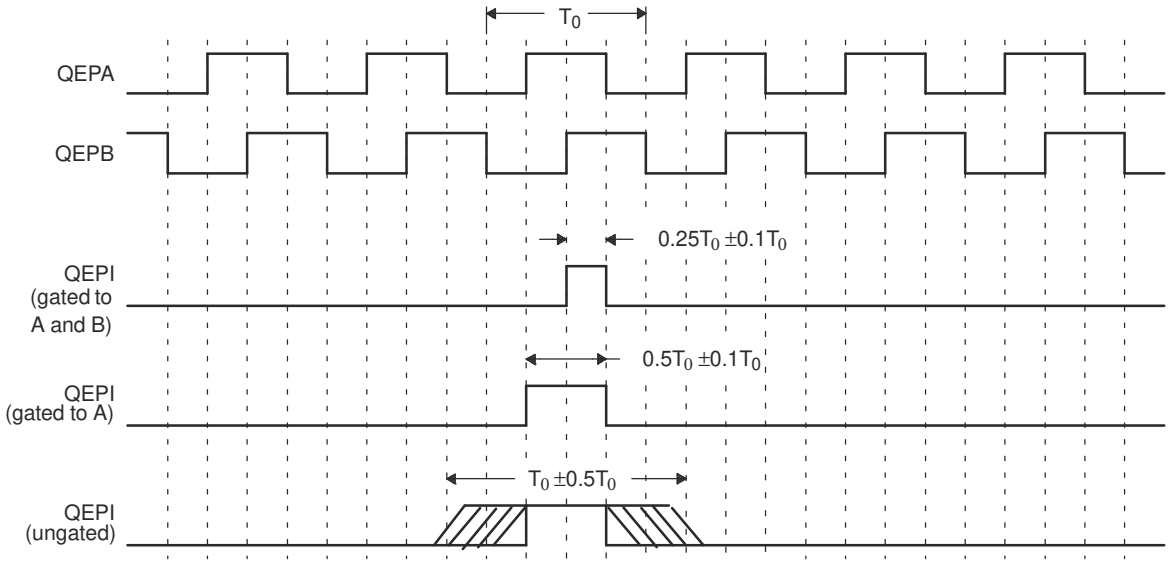


Figure 7-299. Index Pulse Example

Some typical applications of shaft encoders include robotics and computer input in the form of a mouse. Inside your mouse you can see where the mouse ball spins a pair of axles (a left/right, and an up/down axle). These axles are connected to optical shaft encoders that effectively tell the computer how fast and in what direction the mouse is moving.

General Issues: Estimating velocity from a digital position sensor is a cost-effective strategy in motor control. Two different first order approximations for velocity can be written as:

$$v(k) \approx \frac{x(k) - x(k-1)}{T} = \frac{\Delta X}{T} \quad (12)$$

$$v(k) \approx \frac{X}{t(k) - t(k-1)} = \frac{X}{\Delta T} \quad (13)$$

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 12 is the conventional approach to velocity estimation and requires a time base to provide a unit time event for velocity calculation. Unit time is basically 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 12 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 400Hz. 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 12rpm when sampled at 400Hz. While this resolution can be satisfactory at moderate or high speeds, for example 1% error at 1200rpm, this resolution clearly proves inadequate at low speeds. In fact, at speeds below 12rpm, the speed estimate is erroneously zero much of the time.

At low speed, Equation 13 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 13 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, as does Equation 12. 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 13 at low speed and have the DSP software switch over to Equation 12 when the motor speed rises above some specified threshold.

7.5.8.2 Configuring Device Pins

The GPIO mux registers must be configured to connect this peripheral to the device pins.

For proper operation of the eQEP module, input GPIO pins must be configured using the GPIO MUX registers.

See the GPIO chapter starting at Section 13.1.1 for more details on GPIO MUX settings.

7.5.8.3 EQEP Integration

There are 3x EQEP modules integrated in the device. The diagram below provides a visual representation of the device integration details.

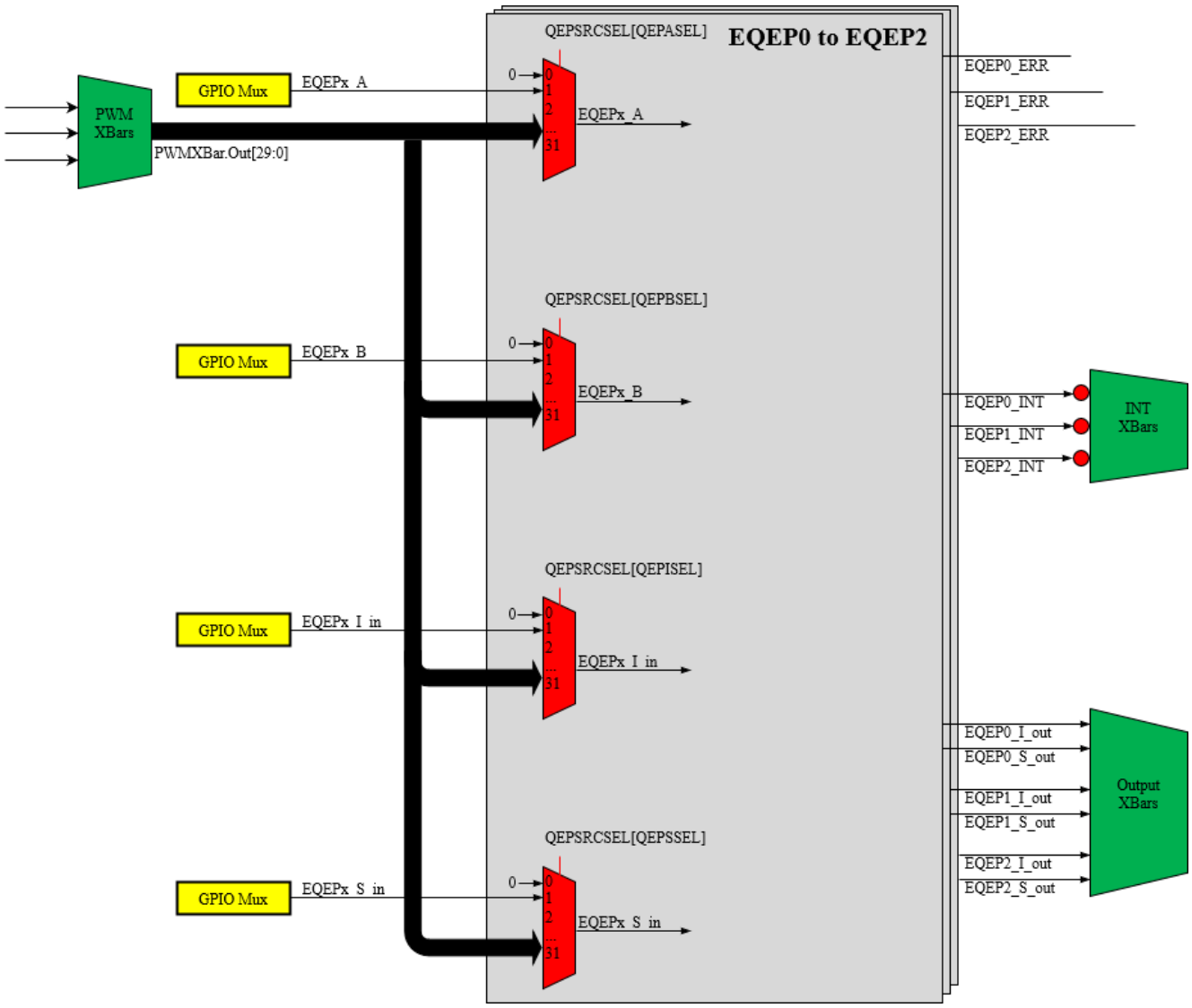


Figure 7-300. EQEP Integration Diagram

7.5.8.4 Description

This section provides the eQEP inputs, memory map, and functional description.

7.5.8.4.1 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. The eQEP module requires that the QEPA, QEPB, and QEPI inputs are synchronized to SYSCLK prior to entering the module. The application code can enable the synchronous GPIO input feature on any eQEP-enabled GPIO pins.

- **QEPA/XCLK and QEPB/XDIR**

  These two pins can be used in quadrature-clock mode or direction-count mode.
  - Quadrature-clock Mode

    The eQEP encoders provide two square wave signals (A and B) 90 electrical degrees out of phase. This 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 conversely. 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 QEPA pin provides the clock input and the QEPB pin provides the direction input.
- **QEPI: 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.
- **QEPS: 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.

Input signals to the eQEP (QEPA, QEPB, QEPI and QEPS) can come from multiple sources; that is, device pin, CMPSSx, or PWMXBARx. One typical use case is if SinCos transducers are used in the motor control system to estimate the position of motor shaft and Index signal is coming from traditional rotary encoder, source of the eQEP signals (QEPA, QEPB and QEPI) can be configured as output of CMPSSx which decodes the Sin, Cos and Index signals. Figure 7-301 illustrates the use case.

Selection of the source of Input signals (QEPA, QEPB, and QEPI) is user-configurable through the QEPSRCSEL register as shown in eQEP Input Source Select Table.

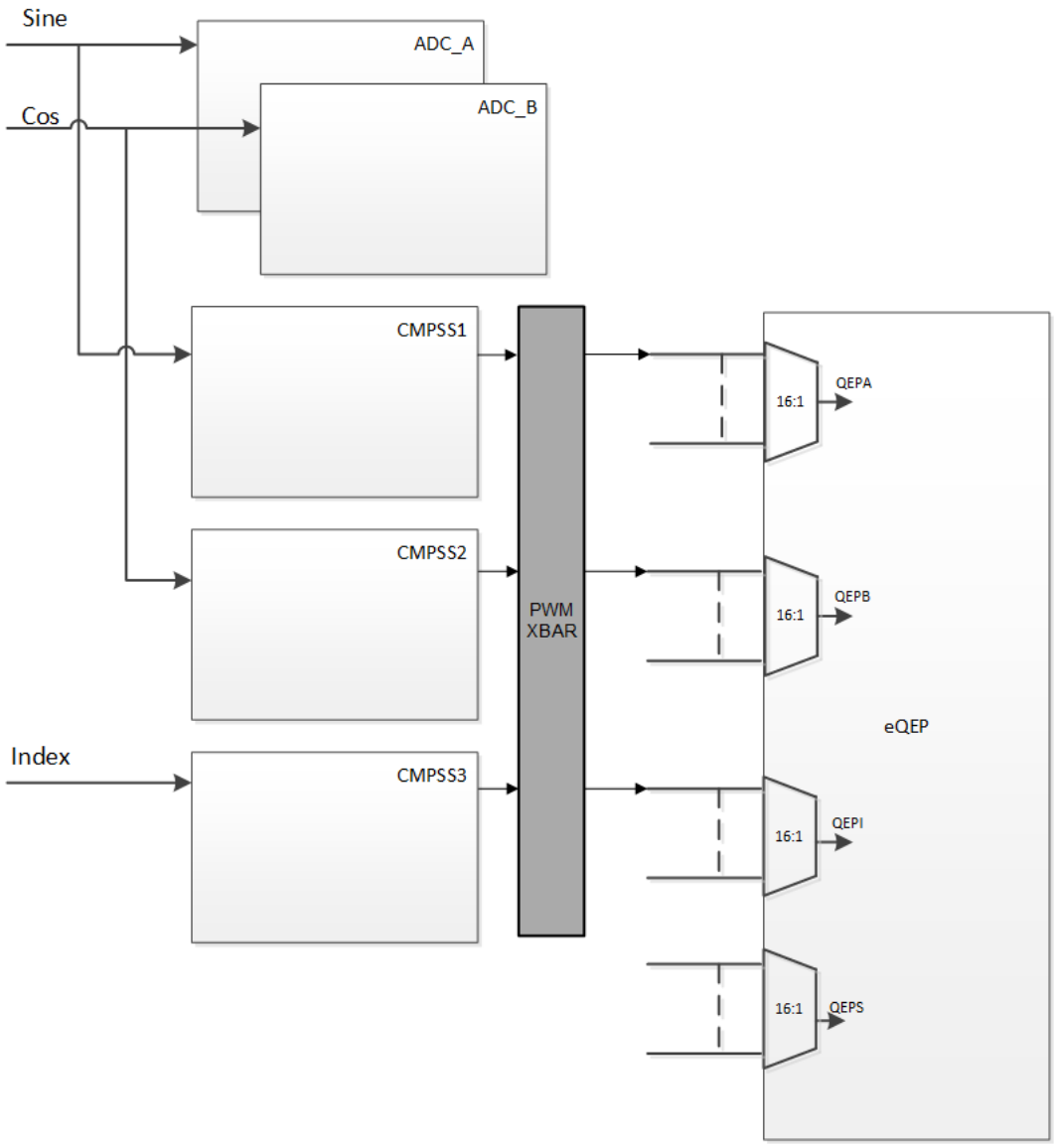


Figure 7-301. Using eQEP to Decode Signals from SinCos Transducer

Table 7-139. eQEP Input Source Select Table

QEPSRCSEL.QEPASEL	Source for QEPA	QEPSRCSEL.QEPBSEL	Source for QEPB
0	Tie-low	0	Tie-low
1	EQEP_A_PAD	1	EQEP_B_PAD
2	PWMXBAR.OUT0	2	PWMXBAR.OUT0
3	PWMXBAR.OUT1	3	PWMXBAR.OUT1
4	PWMXBAR.OUT2	4	PWMXBAR.OUT2
5	PWMXBAR.OUT3	5	PWMXBAR.OUT3
6	PWMXBAR.OUT4	6	PWMXBAR.OUT4
7	PWMXBAR.OUT5	7	PWMXBAR.OUT5
8	PWMXBAR.OUT6	8	PWMXBAR.OUT6
9	PWMXBAR.OUT7	9	PWMXBAR.OUT7
10	PWMXBAR.OUT8	10	PWMXBAR.OUT8
11	PWMXBAR.OUT9	11	PWMXBAR.OUT9
12	PWMXBAR.OUT10	12	PWMXBAR.OUT10
13	PWMXBAR.OUT11	13	PWMXBAR.OUT11
14	PWMXBAR.OUT12	14	PWMXBAR.OUT12
15	PWMXBAR.OUT13	15	PWMXBAR.OUT13
16	PWMXBAR.OUT14	16	PWMXBAR.OUT14
17	PWMXBAR.OUT15	17	PWMXBAR.OUT15
18	PWMXBAR.OUT16	18	PWMXBAR.OUT16
19	PWMXBAR.OUT7	19	PWMXBAR.OUT7
20	PWMXBAR.OUT18	20	PWMXBAR.OUT18
21	PWMXBAR.OUT19	21	PWMXBAR.OUT19
22	PWMXBAR.OUT20	22	PWMXBAR.OUT20
23	PWMXBAR.OUT21	23	PWMXBAR.OUT21
24	PWMXBAR.OUT22	24	PWMXBAR.OUT22
25	PWMXBAR.OUT23	25	PWMXBAR.OUT23
26	PWMXBAR.OUT24	26	PWMXBAR.OUT24
27	PWMXBAR.OUT25	27	PWMXBAR.OUT25
28	PWMXBAR.OUT26	28	PWMXBAR.OUT26
29	PWMXBAR.OUT27	29	PWMXBAR.OUT27
30	PWMXBAR.OUT28	30	PWMXBAR.OUT28
31	PWMXBAR.OUT29	31	PWMXBAR.OUT29

Note

Configuration of QEPSRCSEL register to select the source of QEPA, QEPB, and QEPI signals can lead to unexpected transition on these signals, which can cause an undesirable outcome if eQEP is already running. Please make sure eQEP is disabled before configuring the QEPSRCSEL register for input signals.

7.5.8.4.2 Functional Description

The eQEP peripheral contains the following major functional units (as shown in Figure 7-302):

- Programmable input qualification for each pin (part of the GPIO MUX)
- Quadrature Decoder Unit (QDU)
- Position Counter and Control Unit (PCCU) for position measurement
- Quadrature edge-capture (QCAP) unit for low-speed measurement
- Unit time(UTIME) base for speed/frequency measurement
- Watchdog timer for detecting stalls (QWDOG)

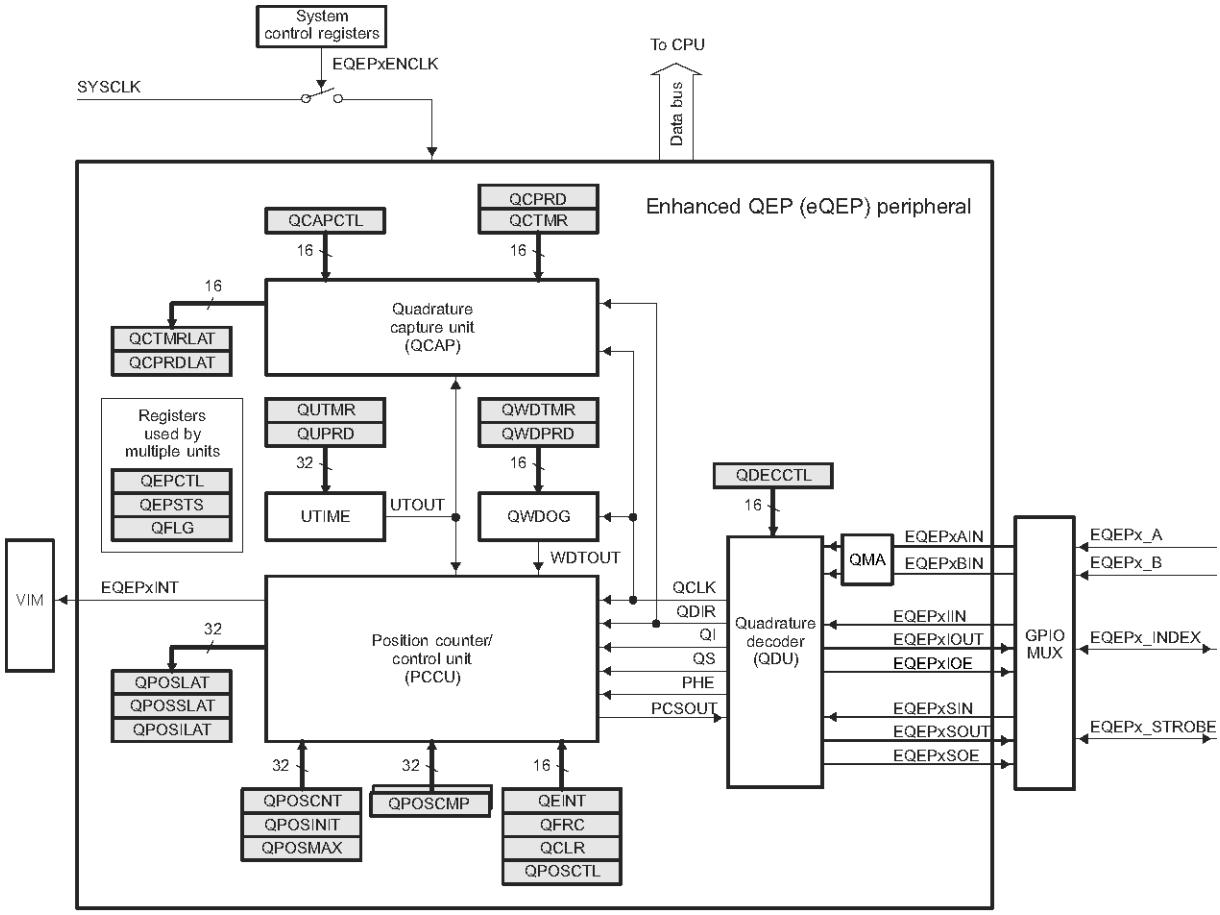


Figure 7-302. Functional Block Diagram of the eQEP Peripheral

7.5.8.4.3 eQEP Memory Map

lists the registers with the memory locations, sizes, and reset values.

7.5.8.5 Quadrature Decoder Unit (QDU)

Figure 7-303 shows a functional block diagram of the QDU.

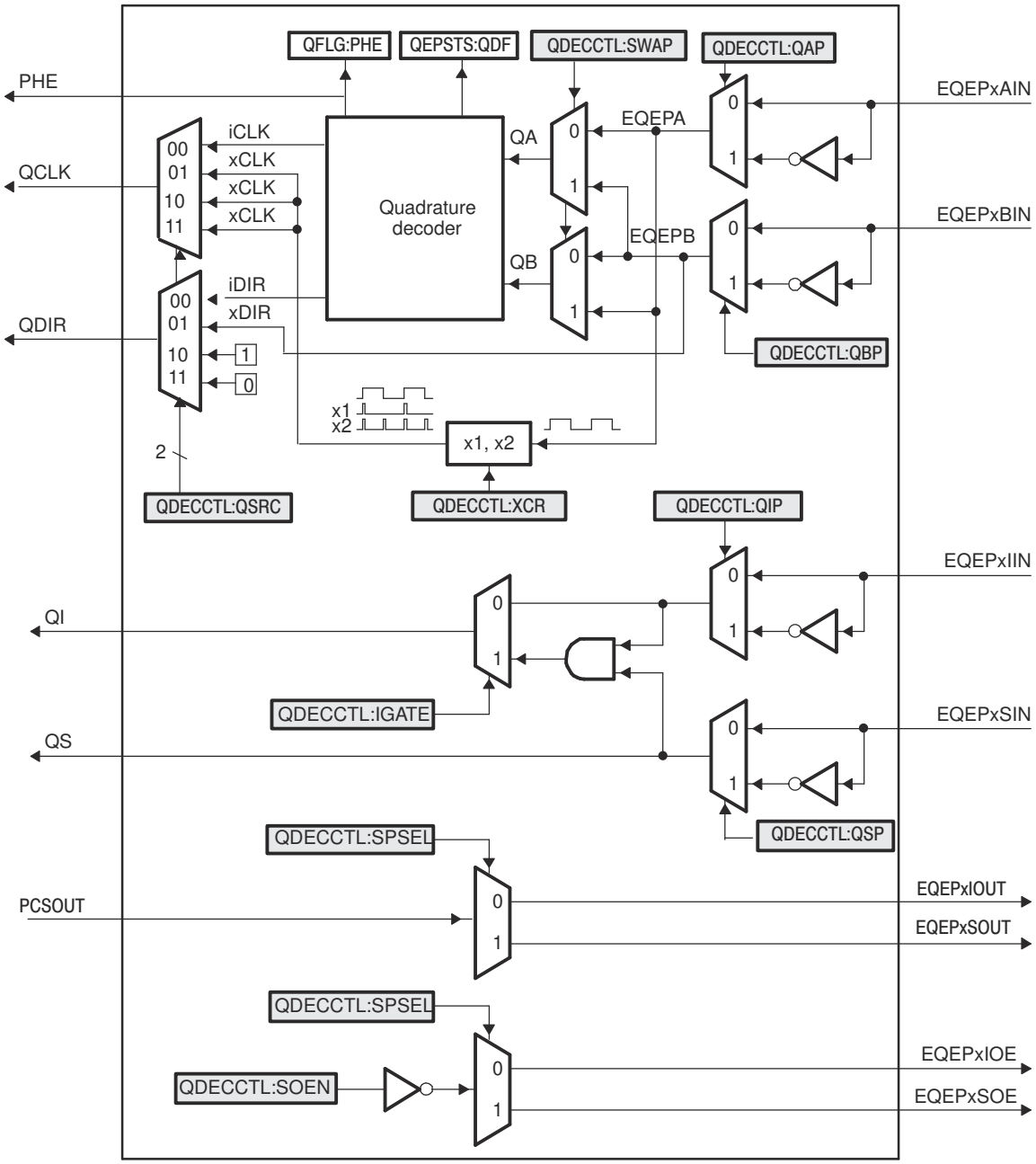


Figure 7-303. Functional Block Diagram of Decoder Unit

7.5.8.5.1 Position Counter Input Modes

Clock and direction input to the position counter is selected using QDECCTL[QSRC] bits, based on interface input requirement as follows:

- Quadrature-count mode
- Direction-count mode
- UP-count mode
- DOWN-count mode

7.5.8.5.1.1 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 (QEPA, QEPB) is the leading sequence and accordingly updates the direction information in the QEPSTS[QDF] bit. Table 7-140 and Figure 7-304 show the direction decoding logic in truth table and state machine form. Both edges of the QEPA and QEPB 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 7-305 shows the direction decoding and clock generation from the eQEP input signals.

Table 7-140. 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	Decrement
	QA↓	UP	Increment
	QB↓	TOGGLE	Increment or Decrement
QB↓	QA↓	DOWN	Decrement
	QA↑	UP	Increment
	QB↑	TOGGLE	Increment or Decrement

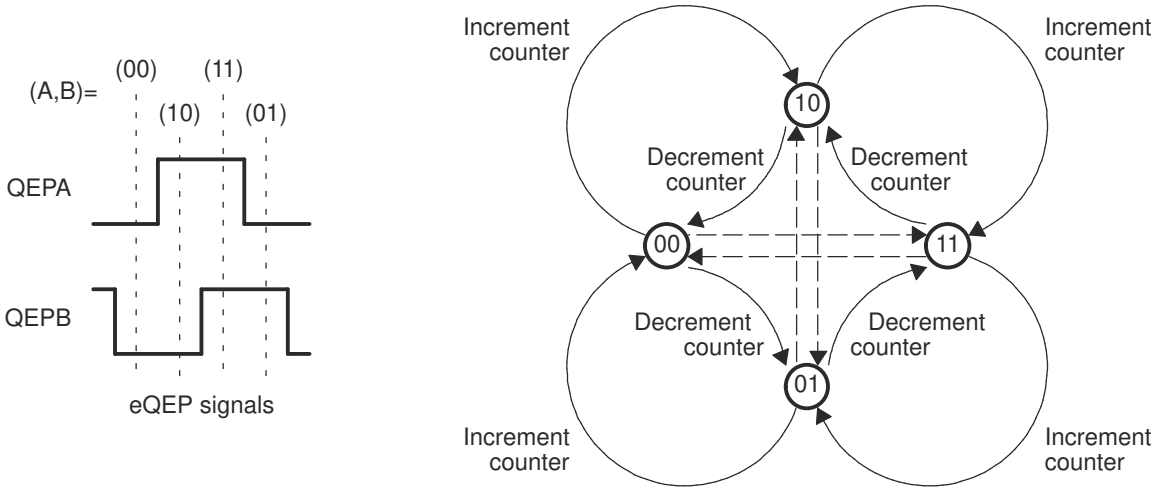


Figure 7-304. Quadrature Decoder State Machine

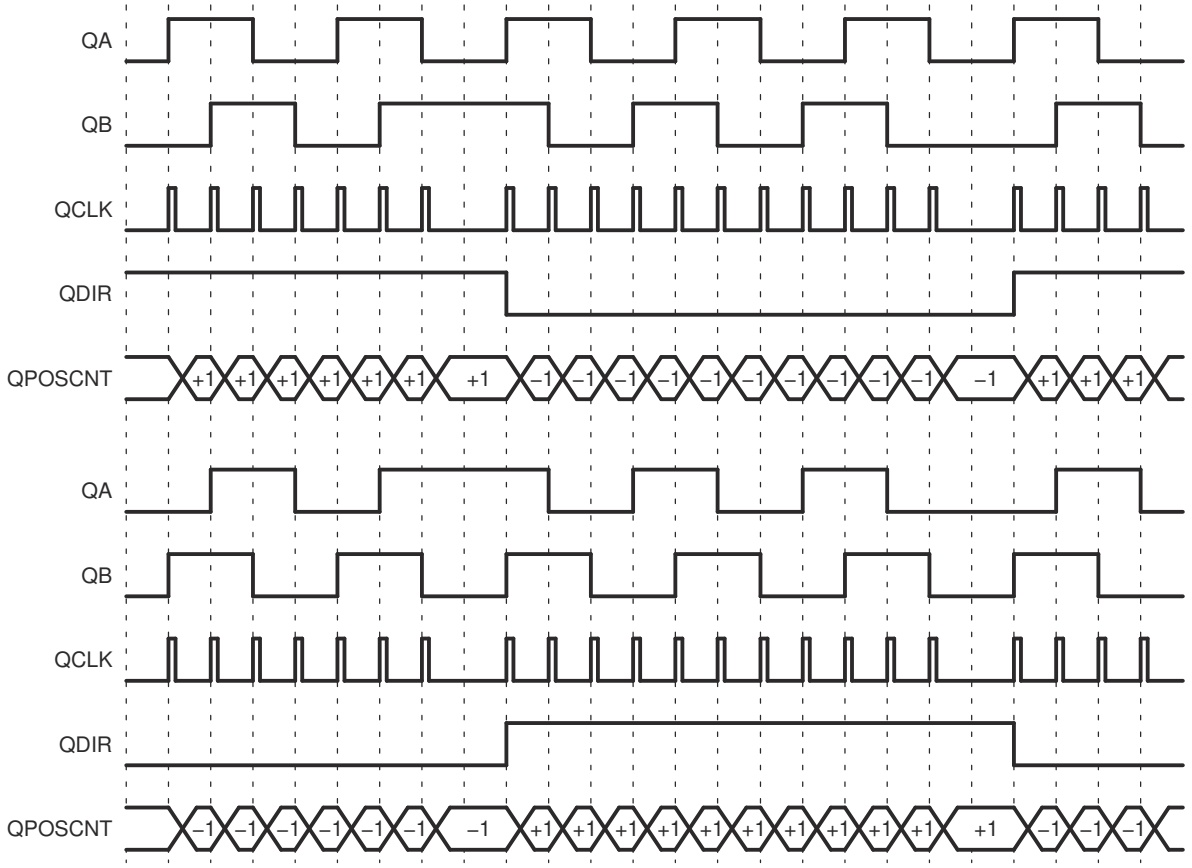


Figure 7-305. Quadrature-clock and Direction Decoding

Phase Error Flag In normal operating conditions, quadrature inputs QEPA and QEPB is 90 degrees out of phase. The phase error flag (PHE) is set in the QFLG register and the QPOSCNT value can be incorrect and offset by multiples of 1 or 3. That is, when edge transition is detected simultaneously on the QEPA and QEPB signals to optionally generate interrupts. State transitions marked by dashed lines in Figure 7-304 are invalid transitions that generate a phase error.

Count Multiplication The eQEP position counter provides 4x times the resolution of an input clock by generating a quadrature-clock (QCLK) on the rising/falling edges of both eQEP input clocks (QEPA and QEPB) as shown in Figure 7-305.

Reverse Count In normal quadrature count operation, QEPA input is applied to the QA input of the quadrature decoder and the QEPB input is applied to the QB input of the quadrature decoder. Reverse counting is enabled by setting the SWAP bit in the QDECCTL register. This swaps the input to the quadrature decoder; thereby, reversing the counting direction.

7.5.8.5.1.2 Direction-Count Mode

Some position encoders provide direction and clock outputs, instead of quadrature outputs. In such cases, direction-count mode can be used. The QEPA input provides the clock for the position counter and the QEPB input has the direction information. The position counter is incremented on every rising edge of a QEPA input when the direction input is high, and decremented when the direction input is low.

7.5.8.5.1.3 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 QEPA input. Clearing the QDECCTL[XCR] bit enables clock generation to the position counter on both edges of the QEPA input; thereby, increasing the measurement resolution by a factor of 2x. In up-count mode, we recommend that the application not configure QEPB as a GPIO mux option, or make sure that a signal edge is not generated on the QEPB input.

7.5.8.5.1.4 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 QEPA input. Clearing the QDECCTL[XCR] bit enables clock generation to the position counter on both edges of a QEPA input, thereby increasing the measurement resolution by a factor of 2x. In down-count mode, the application must not configure QEPB as a GPIO mux option or make sure that a signal edge is not generated on the QEPB input.

7.5.8.5.2 eQEP Input Polarity Selection

Each eQEP input can be inverted using QDECCTL[8:5] control bits. As an example, setting the QDECCTL[QIP] bit inverts the index input.

7.5.8.5.3 Position-Compare Sync Output

The enhanced 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 (QPOSCNT) and the position- compare register (QPOSCMP). This sync signal can be output using an index pin or strobe pin of the EQEP peripheral.

Setting the QDECCTL[SOEN] bit enables the position-compare sync output and the QDECCTL[SPSEL] bit selects either an eQEP index pin or an eQEP strobe pin.

7.5.8.6 Position Counter and Control Unit (PCCU)

The position-counter and control unit provides two configuration registers (QEPCTL and QPOSCTL) for setting up position-counter operational modes, position-counter initialization/latch modes and position-compare logic for sync signal generation.

7.5.8.6.1 Position Counter Operating Modes

Position-counter data can 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 the 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 the position counter provides a rotor angle with respect to the index pulse position.

The 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, the position counter is reset to 0 on overflow and to the QPOSMAX register value on underflow. Overflow occurs when the position counter counts up after the QPOSMAX value. Underflow occurs when the position counter counts down after 0. The Interrupt flag is set to indicate overflow/underflow in QFLG register.

7.5.8.6.1.1 Position Counter Reset on Index Event (QEPCTL[PCRM]=00)

If the index event occurs during the 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 QPOSMAX register on the next eQEP clock.

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 (QEPSTS[FIMF]) and direction on the first index event marker (QEPSTS[FIDF]) in QEPSTS registers, the eQEP peripheral 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 QEPB during the forward direction, then all the subsequent reset must be aligned with the falling edge of QEPB for the forward rotation and on the rising edge of QEPB for the reverse rotation as shown in Figure 7-306.

The position-counter value is latched to the QPOSILAT register and direction information is recorded in the QEPSTS[QDLF] bit on every index event marker. The position-counter error flag (QEPSTS[PCEF]) and error interrupt flag (QFLG[PCE]) are set if the latched value is not equal to 0 or QPOSMAX. The position-counter error flag (QEPSTS[PCEF]) is updated on every index event marker and an interrupt flag (QFLG[PCE]) is set on error that can be cleared only through software.

The index event latch configuration QEPCTL[IEL] must be configured to 00 or 11 when pcrm=0 and the position counter error flag/interrupt flag are generated only in index event reset mode. The position counter value is latched into the IPOSLAT register on every index marker.

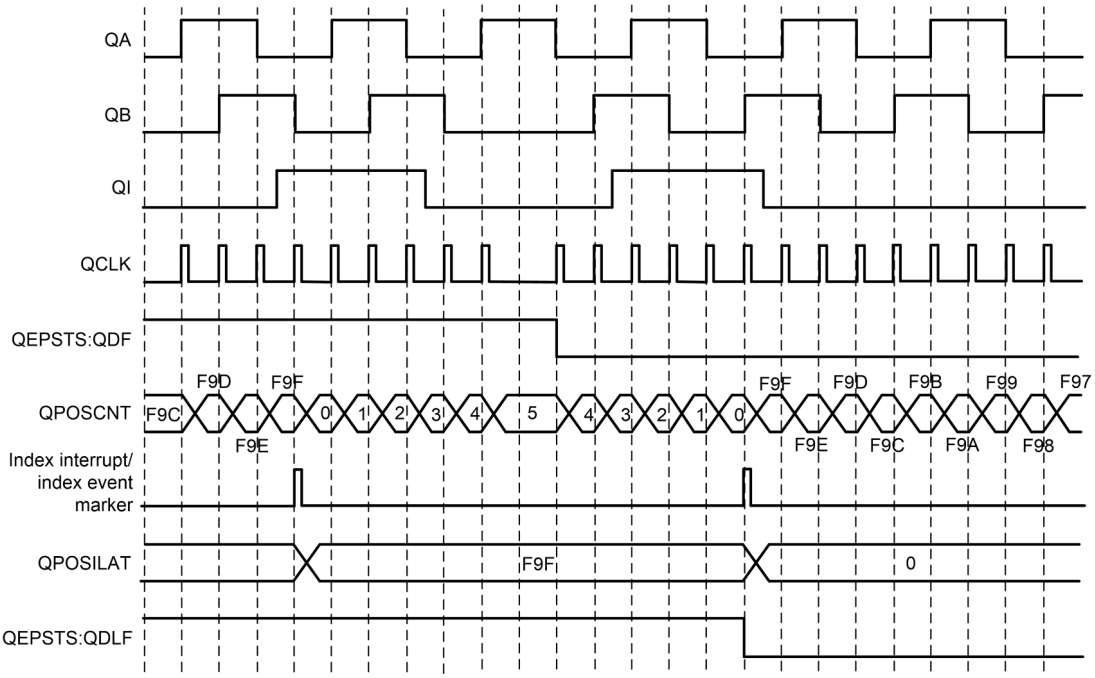


Figure 7-306. Position Counter Reset by Index Pulse for 1000-Line Encoder (QPOSMAX = 3999 or 0xF9F)

Note

In case of a boundary condition where the time period between the Index Event and the previous QCLK edge is less than SYSCLK period, then QPOSCNT gets reset to zero or QPOSMAX in the same SYSCLK cycle and does not wait for the next QCLK edge to occur.

7.5.8.6.1.2 Position Counter Reset on Maximum Position (QEPCTL[PCRM]=01)

If the position counter is equal to QPOSMAX, 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 ZERO, then the position counter is reset to QPOSMAX on the next QEP clock for reverse movement and position-counter underflow flag is set. Figure 7-307 shows the position-counter reset operation in this mode.

The first index marker fields (QEPSTS[FIDF] and QEPSTS[FIMF]) are not applicable in this mode.

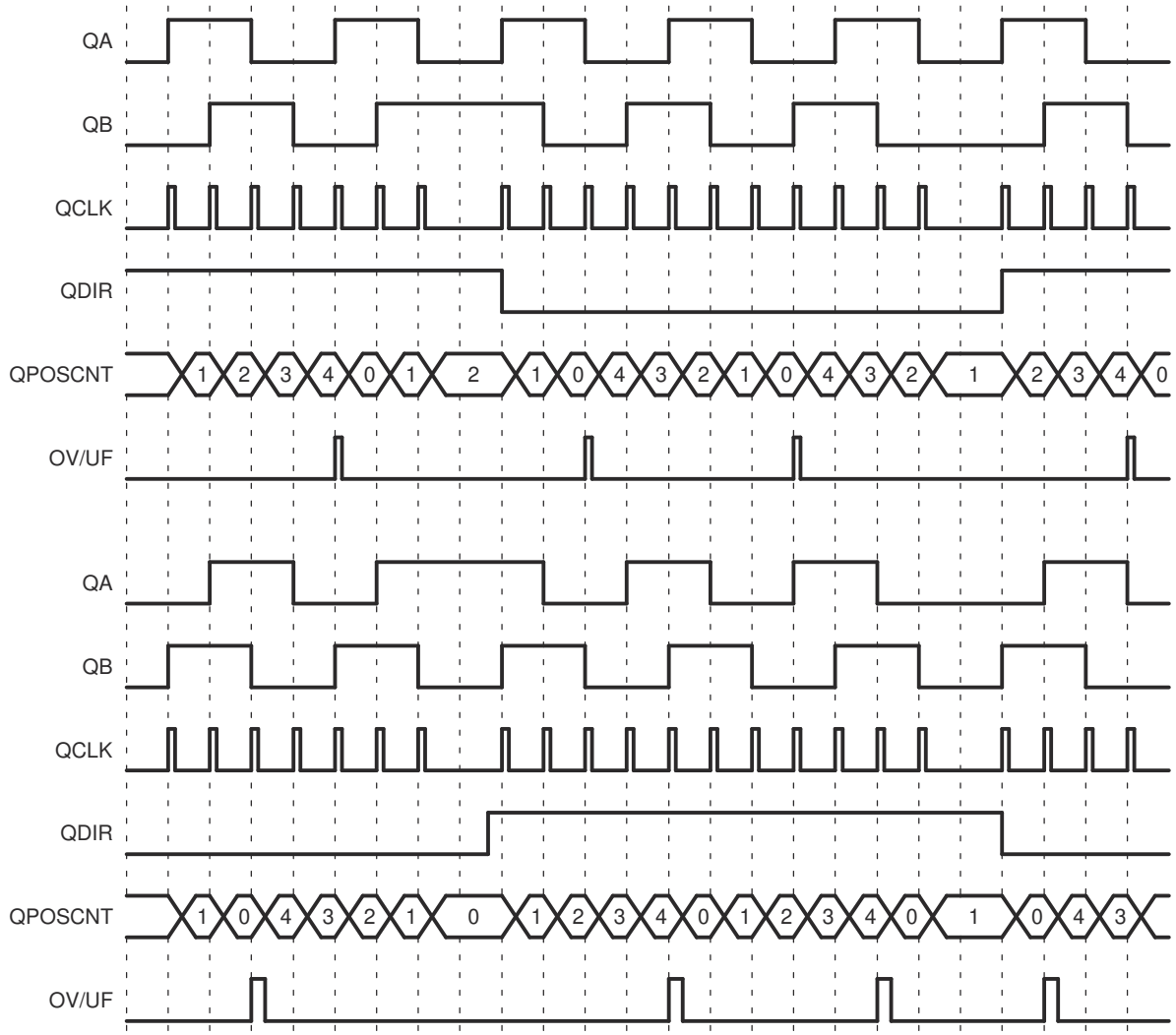


Figure 7-307. Position Counter Underflow/Overflow (QPOSMAX = 4)

7.5.8.6.1.3 Position Counter Reset on the First Index Event (QEPCTL[PCRM] = 10)

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 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, the position-counter value is reset based on the maximum position as described in Section 7.5.8.6.1.2.

The first index marker fields (QEPSTS[FIDF] and QEPSTS[FIMF]) are not applicable in this mode.

7.5.8.6.1.4 Position Counter Reset on Unit Time-out Event (QEPCTL[PCRM] = 11)

In this mode, QPOSCNT is set to 0 or QPOMAX, depending on the direction mode selected by QDECCTL[QSRC] bits on a unit time event. This is useful for frequency measurement.

7.5.8.6.2 Position Counter Latch

The eQEP index and strobe input can be configured to latch the position counter (QPOSCNT) into QPOSILAT and QPOSSLAT, respectively, on occurrence of a definite event on these pins.

7.5.8.6.2.1 Index Event Latch

In some applications, it is not desirable to reset the position counter on every index event and instead it can be required to operate the position counter in full 32-bit mode (QEPCTL[PCRM] = 01 and QEPCTL[PCRM] = 10 modes).

In such cases, the eQEP position counter can be configured to latch on the following events and direction information is recorded in the QEPSTS[QDLF] bit on every index event marker.

- Latch on Rising edge (QEPCTL[IEL]=01)
- Latch on Falling edge (QEPCTL[IEL]=10)
- Latch on Index Event Marker (QEPCTL[IEL]=11)

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 (QFLG[IEL]) is set when the position counter is latched to the QPOSILAT register. The index event latch configuration bits (QEPCTL[IEL]) are ignored when QEPCTL[PCRM] = 00.

Latch on Rising Edge (QEPCTL[IEL]=01)	The position-counter value (QPOSCNT) is latched to the QPOSILAT register on every rising edge of an index input.
Latch on Falling Edge (QEPCTL[IEL] = 10)	The position-counter value (QPOSCNT) is latched to the QPOSILAT register on every falling edge of index input.
Latch on Index Event Marker/Software Index Marker (QEPCTL[IEL] = 11	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 (QEPSTS[FIMF]) and the direction on the first index event marker (QEPSTS[FIDF]) in the QEPSTS registers. The eQEP peripheral also remembers the quadrature edge on the first index marker so that the same relative quadrature transition is used for latching the position counter (QEPCTL[IEL]=11).

Figure 7-308 shows the position counter latch using an index event marker.

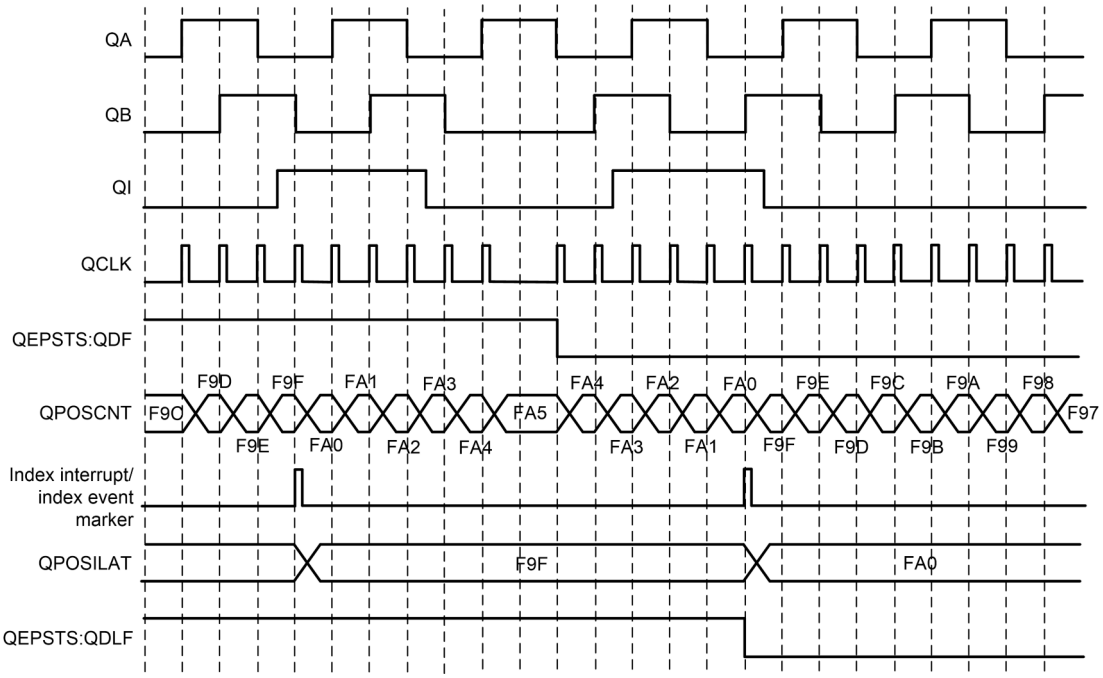


Figure 7-308. Software Index Marker for 1000-line Encoder (QEPCTL[IEL] = 1)

7.5.8.6.2.2 Strobe Event Latch

The position-counter value is latched to the QPOSSLAT register on the rising edge of the strobe input by clearing the QEPCTL[SEL] bit.

If the QEPCTL[SEL] bit is set, then the position-counter value is latched to the 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 7-309.

The strobe event latch interrupt flag (QFLG[SEL) is set when the position counter is latched to the QPOSSLAT register.

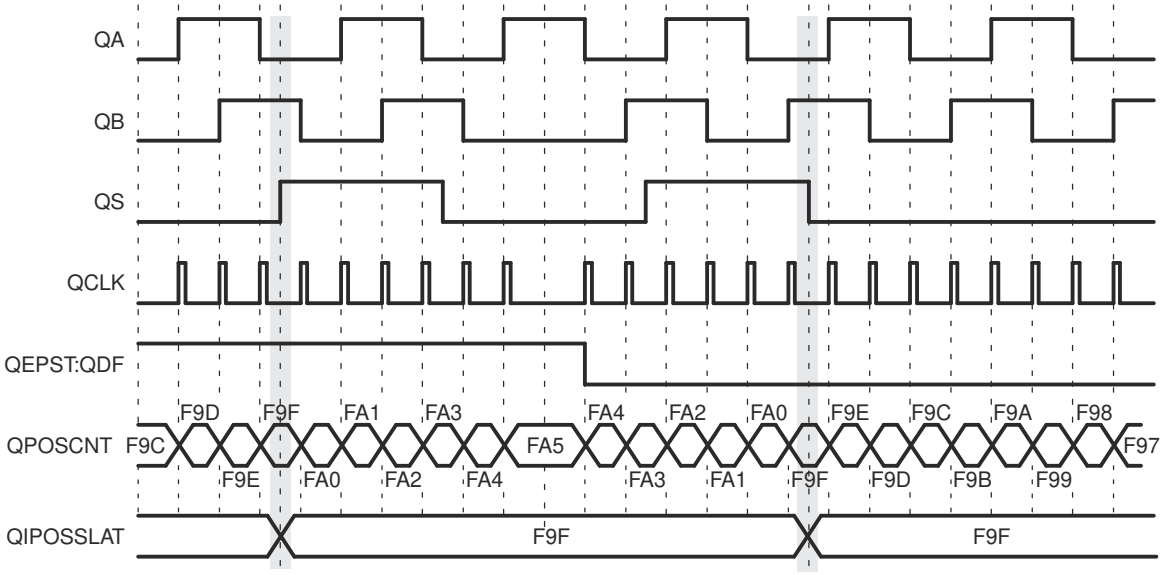


Figure 7-309. Strobe Event Latch (QEPCTL[SEL] = 1)

7.5.8.6.3 Position Counter Initialization

The position counter can be initialized using the following events:

- Index event
- Strobe event
- Software initialization

Index Event Initialization (IEI)	The QEPI 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 QEPCTL[IEI] bits are 10, then the position counter (QPOSCNT) is initialized with a value in the QPOSINIT register on the rising edge of index input. Conversely, if the QEPCTL[IEI] bits are 11, initialization is on the falling edge of the index input.
Strobe Event Initialization (SEI)	If the QEPCTL[SEI] bits are 10, then the position counter is initialized with a value in the QPOSINIT register on the rising edge of strobe input. If QEPCTL[SEL] bits are 11, then the position counter is initialized with a value in the QPOSINIT register on the rising edge of strobe input for forward direction and on the falling edge of strobe input for reverse direction.
Software Initialization (SWI)	The position counter can be initialized in software by writing a 1 to the QEPCTL[SWI] bit. This bit is not automatically cleared. While the bit is still set, if a 1 is written to the bit again, the position counter is re-initialized.

7.5.8.6.4 eQEP Position-compare Unit

The eQEP peripheral includes a position-compare unit that is used to generate a sync output and interrupt on a position-compare match. Figure 7-310 shows a diagram. The position-compare (QPOSCMP) register is shadowed and shadow mode can be enabled or disabled using the QPOSCTL[PSSHDW] bit. If the shadow mode is not enabled, the CPU writes directly to the active position compare register.

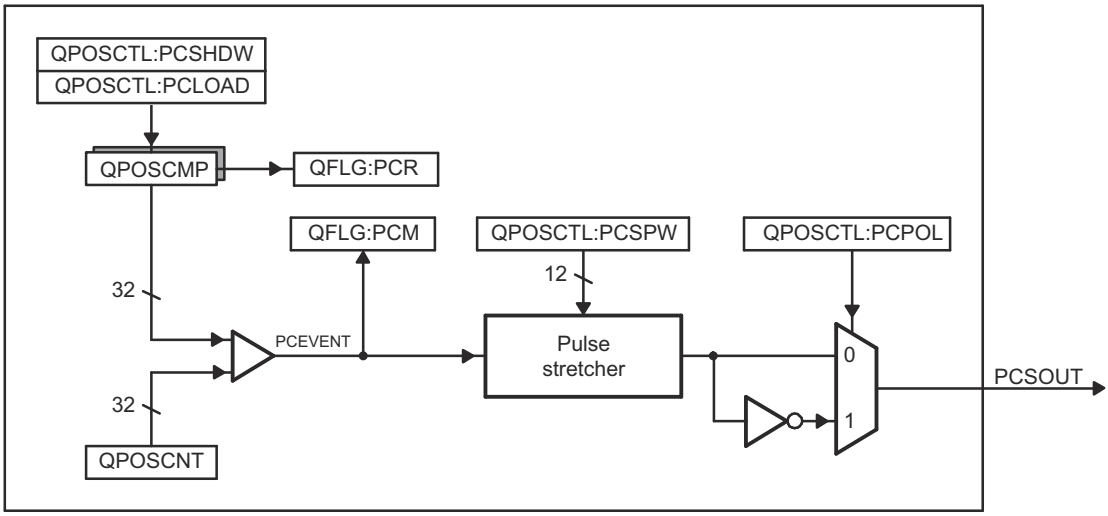


Figure 7-310. eQEP Position-compare Unit

In shadow mode, you can configure the position-compare unit (QPOSCTL[PCLOAD]) to load the shadow register value into the active register on the following events, and to generate the position-compare ready (QFLG[PCR]) interrupt after loading.

- Load on compare match
- Load on position-counter zero event

The position-compare match (QFLG[PCM]) is set when the position-counter value (QPOSCNT) matches with the active position-compare register (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 QPOSCMP = 2, the position-compare unit generates a position-compare event on 1 to 2 transitions of the eQEP position counter for forward counting direction and on 3 to 2 transitions of the eQEP position counter for reverse counting direction (see Figure 7-311).

See the register section for the layout of the eQEP Position-Compare Control Register (QPOSCTL) and description of the QPOSCTL bit fields.

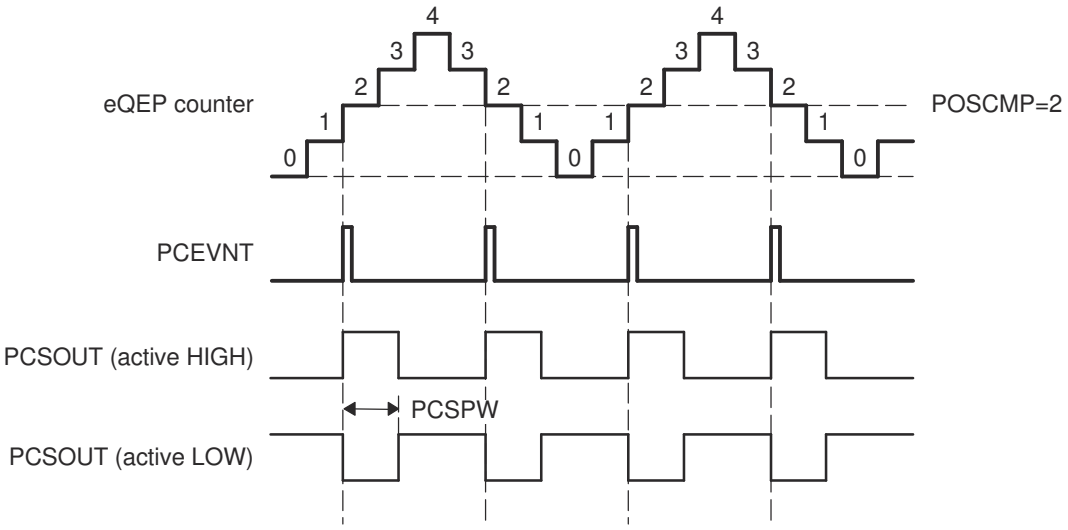


Figure 7-311. eQEP Position-compare Event Generation Points

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 7-312.

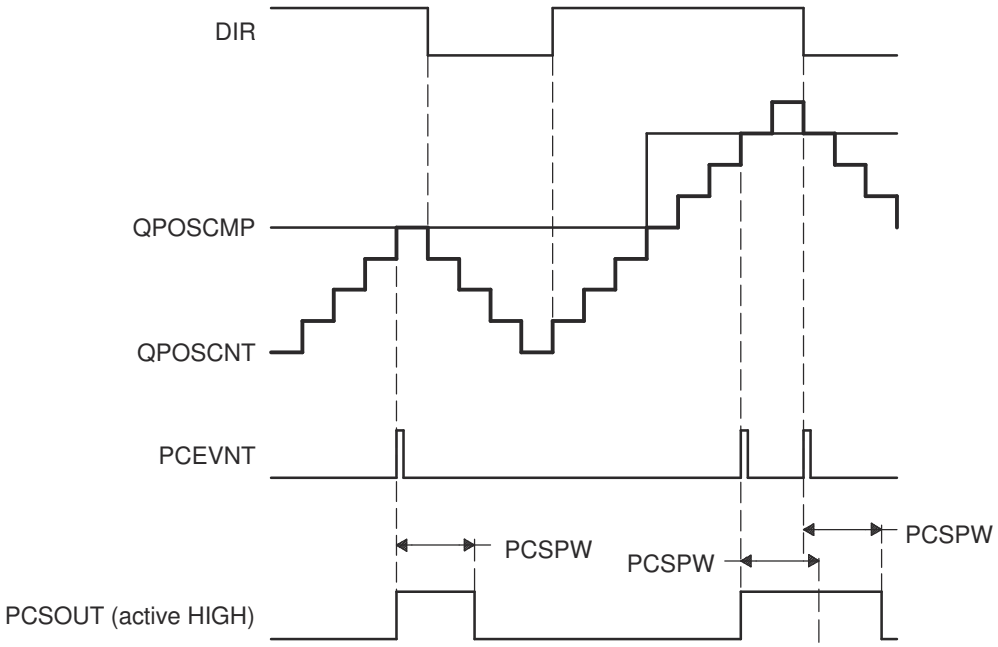


Figure 7-312. eQEP Position-compare Sync Output Pulse Stretcher

7.5.8.7 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 7-313. This feature is typically used for low-speed measurement using the following formula:

$$v(k) = \frac{X}{t(k) - t(k-1)} = \frac{X}{\Delta T} \quad (14)$$

where:

- X = Unit position is defined by integer multiple of quadrature edges (see Figure 7-314)
- ΔT = Elapsed time between unit position events
- v(k) = Velocity at time instant "k"

The eQEP capture timer (QCTMR) runs from prescaled SYSCLKOUT and the prescaler is programmed by the QCAPCTL[CCPS] bits. The capture timer (QCTMR) value is latched into the capture period register (QCPRD) on every unit position event and then the capture timer is reset, a flag is set in QEPSTS:UPEVNT to indicate that new value is latched into the QCPRD register. Software can check this status flag before reading the period register for low speed measurement, and clear the flag by writing 1.

Time measurement (ΔT) between unit position events is 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.

If the QEP capture timer overflows between unit position events, then the timer sets the QEP capture overflow flag (QEPSTS[COEF]) in the status register and the QCPRDLAT register is set to 0xFFFF. If direction change occurs between the unit position events, then the error flag is set in the status register (QEPSTS[CDEF]) and the QCPRDLAT register is set to 0xFFFF.

The Capture Timer (QCTMR) and Capture Period register (QCPRD) can be configured to latch on the following events:

- CPU read of QPOSCNT register
- Unit time-out event

If the QEPCTL[QCLM] bit is cleared, then the capture timer and capture period values are latched into the QCTMRLAT and QCPRDLAT registers, respectively, when the CPU reads the position counter (QPOSCNT).

If the QEPCTL[QCLM] bit is set, then the position counter, capture timer, and capture period values are latched into the QPOSLAT, QCTMRLAT and QCPRDLAT registers, respectively, on unit time out.

Figure 7-315 shows the capture unit operation along with the position counter.

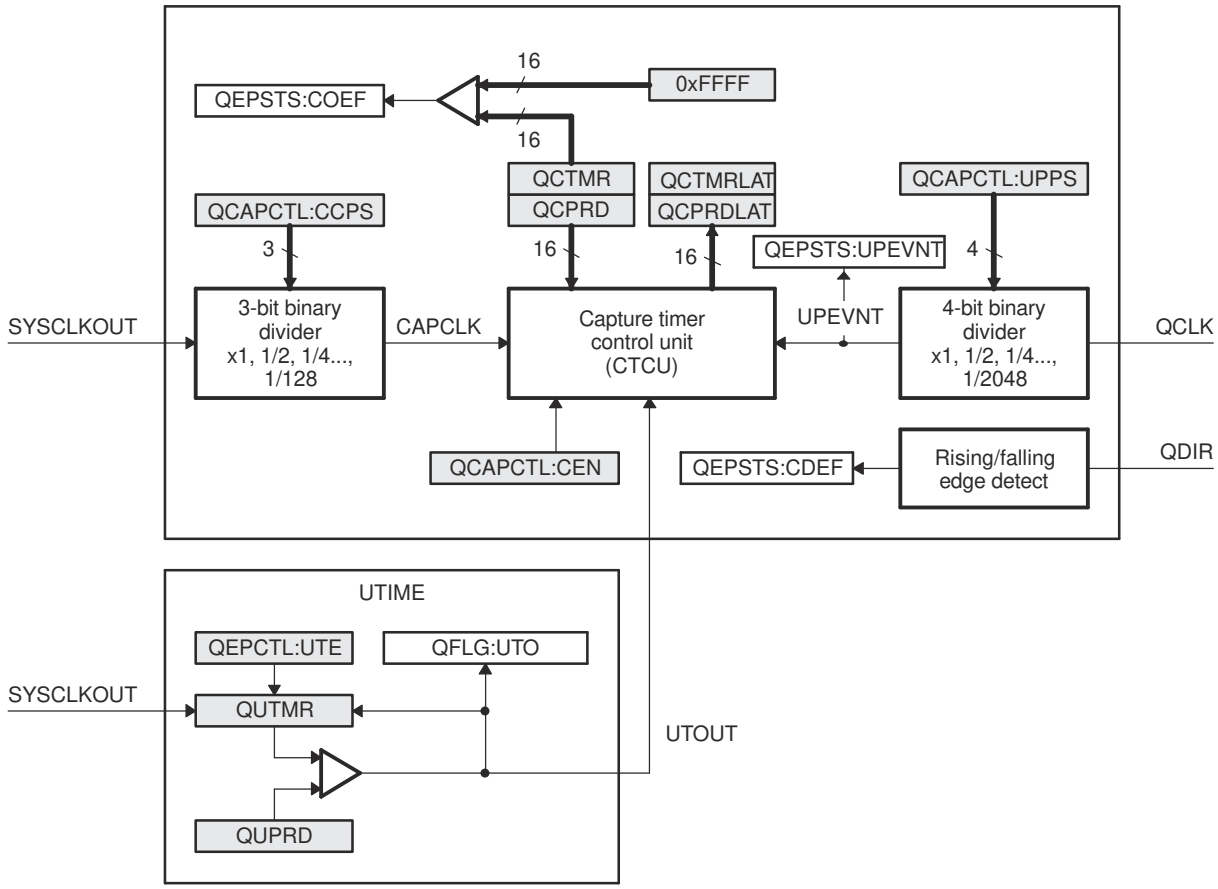
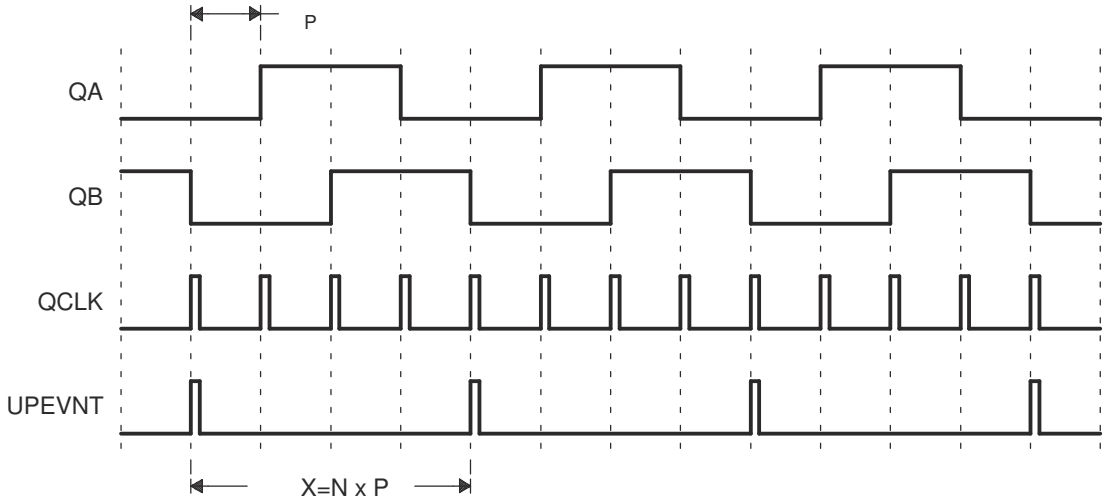


Figure 7-313. eQEP Edge Capture Unit

CAUTION

The QCAPCTL[UPPS] prescaler cannot be modified dynamically (such as switching the unit event prescaler from QCLK/4 to QCLK/8). Doing so can result in undefined behavior. The QCAPCTL[CPPS] prescaler can be modified dynamically (such as switching CAPCLK prescaling mode from SYSCLK/4 to SYSCLK/8) only after the capture unit is disabled.



N = Number of quadrature periods selected using QCAPCTL[UPPS] bits

Figure 7-314. Unit Position Event for Low Speed Measurement (QCAPCTL[UPPS] = 0010)

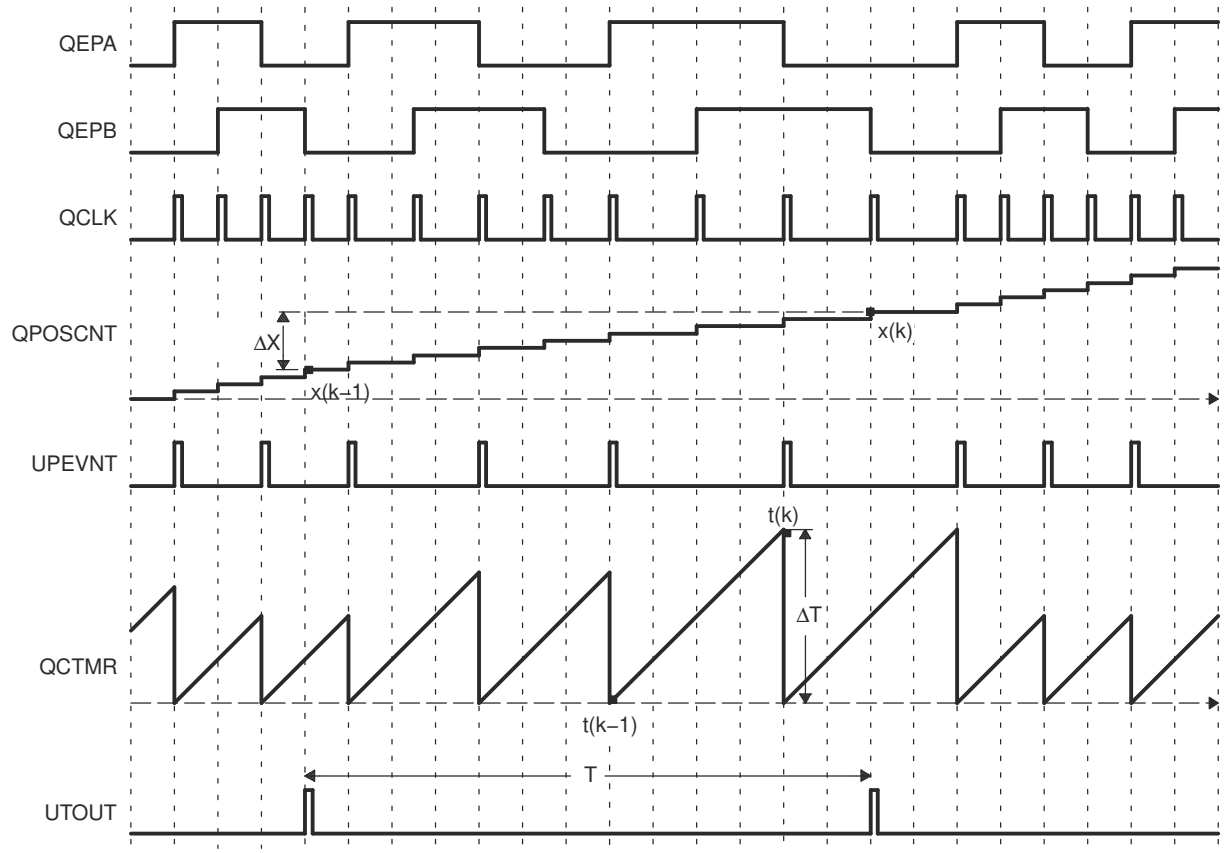


Figure 7-315. eQEP Edge Capture Unit - Timing Details

Velocity calculation equation:

$$v(k) = \frac{x(k) - x(k-1)}{T} = \frac{\Delta X}{T} \text{ o} \quad (15)$$

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 QUPRD and QCAPCTL[UPPS] registers. Incremental position output and incremental time output is available in the QPOSLAT and QCPRDLAT registers.

Parameter	Relevant Register to Configure or Read the Information
T	Unit Period Register (QUPRD)
ΔX	Incremental Position = QPOSLAT(k) - QPOSLAT(K-1)
X	Fixed-unit position defined by sensor resolution and QCAPCTL[UPPS] bits
ΔT	Capture Period Latch (QCPRDLAT)

7.5.8.8 eQEP Watchdog

The eQEP peripheral contains a 16-bit watchdog timer (Figure 7-316) 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 quadrature clock event (pulse) resets the watchdog timer. If no quadrature clock event is detected until a period match (QWDPRD = QWDTMR), then the watchdog timer times out and the watchdog interrupt flag is set (QFLG[WTO]). The time-out value is programmable through the watchdog period register (QWDPRD).

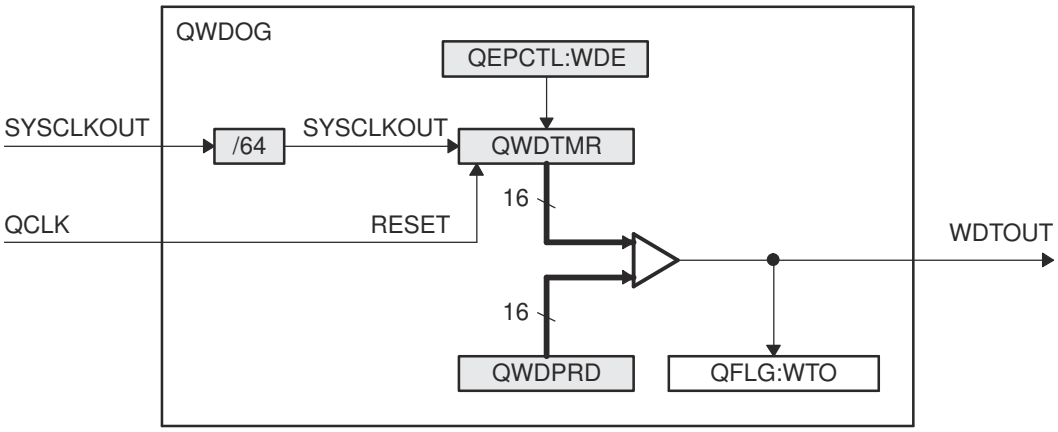


Figure 7-316. eQEP Watchdog Timer

7.5.8.9 eQEP Unit Timer Base

The eQEP peripheral includes a 32-bit timer (QUTMR) that is clocked by SYSCLKOUT to generate periodic interrupts for velocity calculations, see Figure 7-317. Whenever the unit timer (QUTMR) matches the unit period register (QUPRD), the eQEP peripheral resets the unit timer (QUTMR) and also generates the unit time out interrupt flag (QFLG[UTO]). The unit timer gets reset whenever timer value equals to configured period value.

The eQEP peripheral can be configured to latch the position counter, capture timer, and capture period values on a unit time out event so that latched values are used for velocity calculation as described in Section 7.5.8.7.

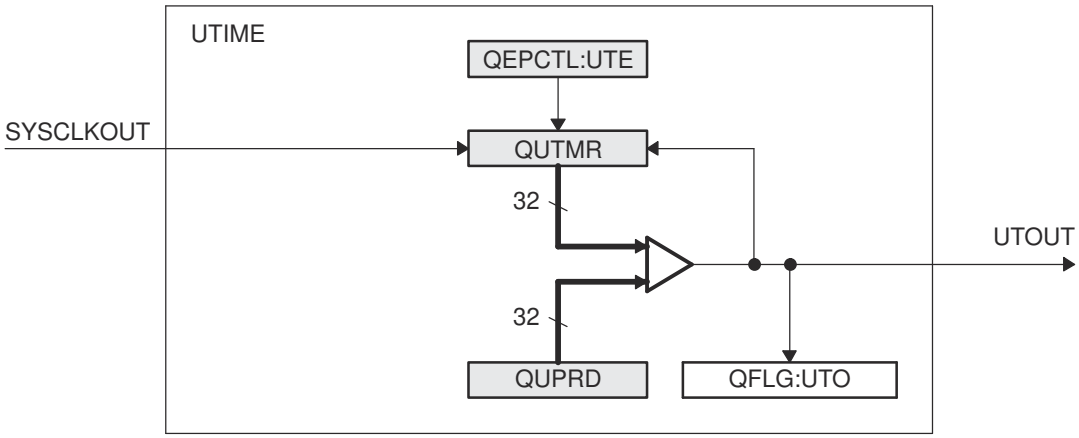


Figure 7-317. eQEP Unit Timer Base

7.5.8.10 eQEP Interrupt Structure

Figure 7-318 shows how the interrupt mechanism works in the eQEP module.

Figure 7-318. 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 (QEINT) is used to enable/disable individual interrupt event sources. The interrupt flag register (QFLG) indicates if any interrupt event has been latched and contains the global interrupt flag bit (INT).

An interrupt pulse is generated to when:

1. Interrupt is enabled for eQEP event inside QEINT register
2. Interrupt flag for eQEP event inside QFLG register is set, and
3. Global interrupt status flag bit QFLG[INT] had been cleared for previously generated interrupt event. The interrupt service routine needs to clear the global interrupt flag bit and the serviced event, by way of the interrupt clear register (QCLR), before any other interrupt pulses are generated. If either flags inside the QFLG register are not cleared, further interrupt events do not generate an interrupt to . You can force an interrupt event by way of the interrupt force register (QFRC), which is useful for test purposes.

7.5.8.11 EQEP Programming Guide

Driver Information

Driver features are available at the eQEP driver page

Software API Information

The eQEP driver provides an API to configure the eQEP module. Full documentation is located on APIs for eQEP

Example Usage

The below links show examples on how to use eQEP

- eQEP Frequency Measurement
- eQEP Position Speed

7.5.9 Fast Serial Interface (FSI)

This chapter contains a general description of the Fast Serial Interface (FSI) module. The FSI is a serial peripheral capable of reliable high-speed communication across isolation barriers.

7.5.9.1 Introduction

The Fast Serial Interface (FSI) module is a serial communication peripheral capable of reliable high-speed communication across isolation devices. Galvanic isolation devices are used in situations where two different electronic circuits, which do not have common power and ground connections, must exchange information. Though isolation devices facilitate these signal communications, isolation devices can also introduce a large delay on the signal lines and add skew between the signals. The FSI is designed specifically to make sure reliable high-speed communication for system scenarios that involve communication across isolation barriers without adding components.

The FSI consists of independent transmitter (FSITX) and receiver (FSIRX) cores. The FSITX and FSIRX cores are configured and operated independently.

For additional information on the FSI module, refer to *Fast Serial Interface (FSI) Skew Compensation*.

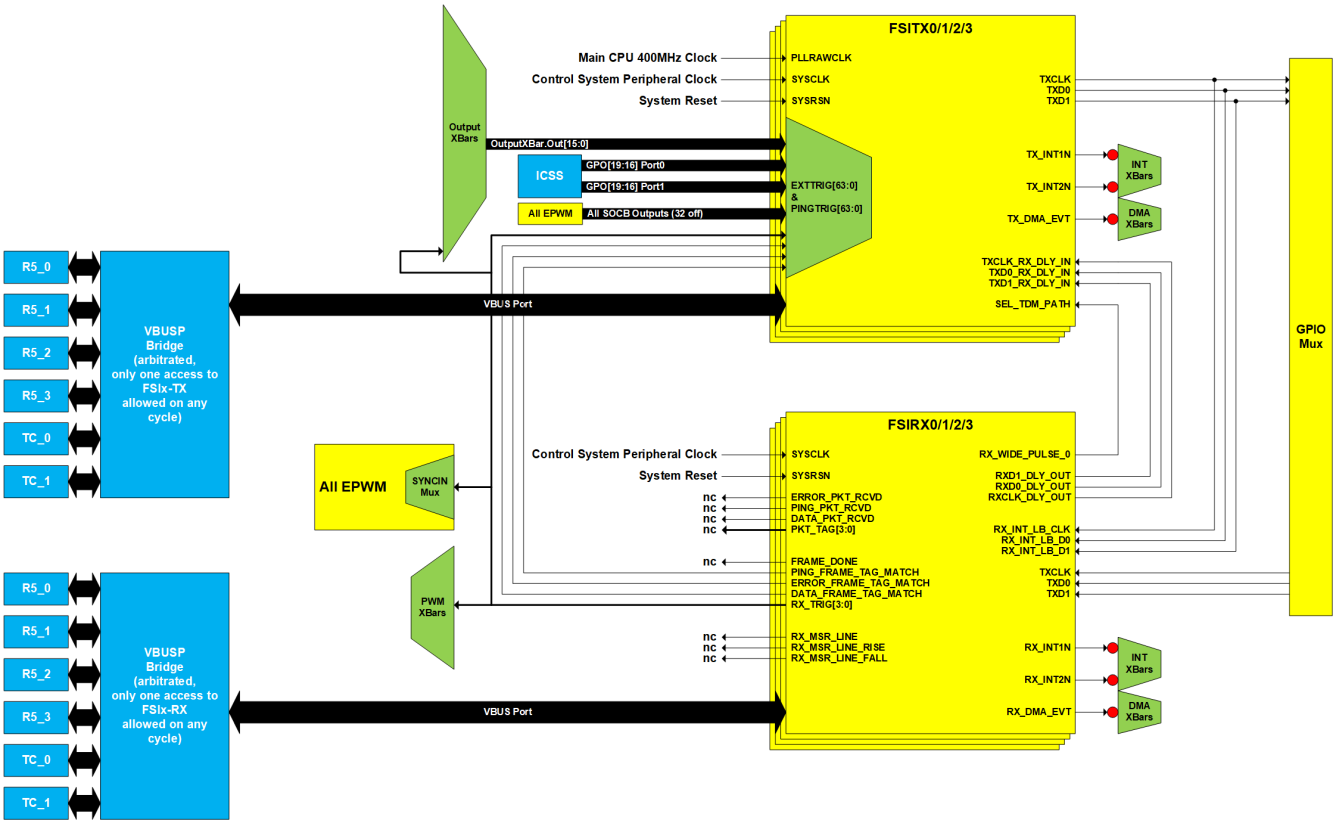
7.5.9.1.1 FSI Features

The FSI module includes the following features:

- Independent transmitter and receiver cores
- Source-synchronous transmission
- Double Data Rate (DDR)
- One or two data lines
- Programmable data length
- Skew adjustment block to compensate for board and system delay mismatches
- Frame error detection
- Programmable frame tagging for message filtering
- Hardware ping to detect line breaks during communication (ping watchdog)
- Two interrupts per FSI core
- Externally-triggered frame generation
- Hardware- or software-calculated CRC
- Embedded ECC computation module
- Register write protection
- FSI-SPI compatibility mode (limited features available)
- Tag match notifications

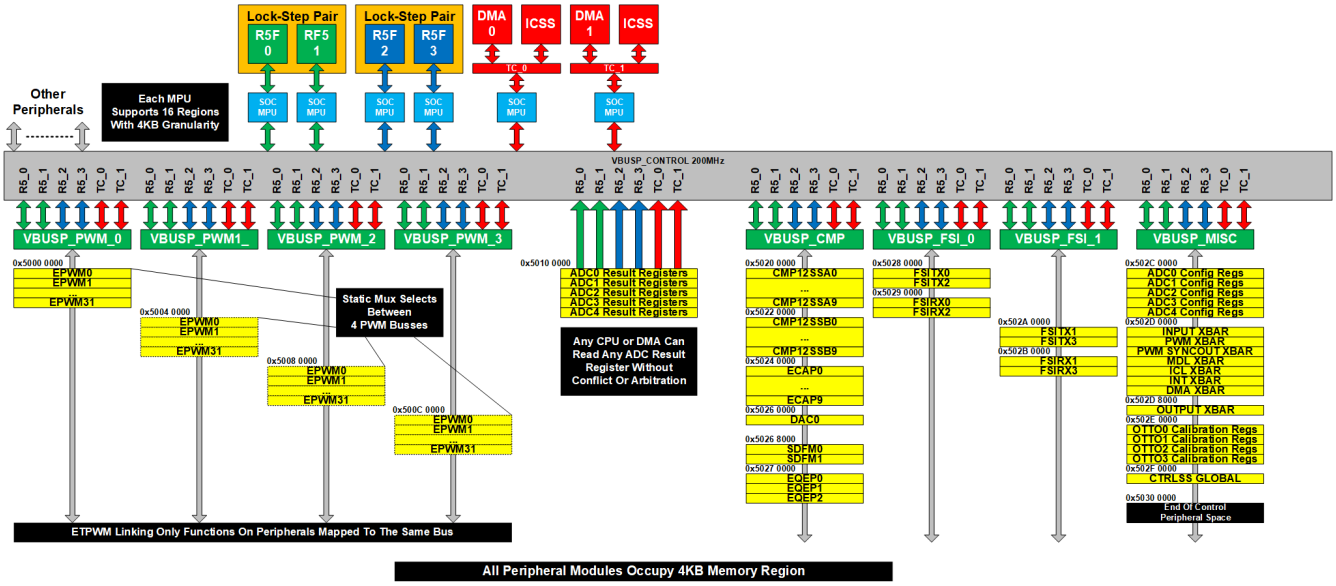
7.5.9.1.2 Block Diagram

This device contains 4 instance of FSI TX and FSI RX cores. The integration details are captured below:



7.5.9.2 System-level Integration

This section describes the device-level integration of the FSI module. Some of the features can require additional configuration of modules that are not within the scope of this chapter, the details can be found elsewhere in this TRM.



The FSI IP clock provided is 200MHz system clock with the option to gate using GLOBAL_CTRLSS_FSI_RX[x]_CLK_GATE:CLK_GATE and GLOBAL_CTRLSS_FSI_TX[x]_CLK_GATE:CLK_GATE.

Software generated reset is provided and can be controlled using GLOBAL_CTRL_FSI_RX[x]_RST:RST and GLOBAL_CTRL_FSI_TX[x]_RST:RST.

7.5.9.2.1 Signal Description

FSI is a point-to-point communication protocol. Hence, an FSI transmitter core communicates directly to a single FSI receiver core. Similarly, an FSI receiver core receives data from a single FSI transmitter core.

Each FSI core has three signals: one clock and two data signals. Data is always transmitted or received with the most-significant bit of each frame field being first. If multi-lane transmissions are not used, the TXD1 and RXD1 signals can be left unconnected and their GPIOs repurposed for other application needs. Table 7-141 and Table 7-142 describe the various signals that can be selected by the PADCONFIG register to be brought out to device pins.

CAUTION
The maximum RXCLK rate is SYSCLK/2 and must not exceed this limit.

Table 7-141. FSI Receiver Core Signals

Signal Name	Direction	Description	Inactive Level(1)
RXCLK	Input	This is the receive clock input signal for the FSI receive module. This must be connected to TXCLK of the transmitting FSI module.	Logic High
RXD0	Input	This is the primary data input line for reception. This must be connected to the TXD0 of the transmitting FSI module.	Logic High
RXD1	Input	This is an additional data input line for reception. This signal must be connected to the TXD1 of the transmitting FSI module, if multi-lane transmission is used.	Logic High

(1) Inactive level refers to the state of the pin while the module is not actively receiving data.

Table 7-142. FSI Transmitter Core Signals

Signal Name	Direction	Description	Inactive Level(1)
TXCLK	Output	This is the transmit clock and is driven by the FSI transmit module. During a transmission, four clock edges are transmitted before the start of frame phase (preamble) and four clock edges follow the last bit of the frame (postamble). Data is transmitted on both edges of the clock. In FSI-SPI compatibility mode, the preamble and the post frame clock edges are not transmitted. Data is transmitted only on one edge of the clock. Data transmits on rising edge and received on falling edge of the clock.	Logic High
TXD0	Output	This is the primary data output line for transmission and is driven by the FSI transmit module. When the FSI is configured for multi-lane transmission, TXD0 contains all the even numbered bits of the data and CRC bytes. Other frame fields such as frame type, start-of-frame, tag, and end-of-frame are transmitted in full.	Logic High
TXD1	Output	This is an additional data output line for transmission, if the FSI is configured for multi-lane transmission. This signal is driven by the FSI transmit module. During transmission, the data bits are split between TXD0 and TXD1. TXD1 contains all the odd numbered bits of the data and CRC bytes. This applies only to the data words and the CRC bytes. Other data frame related information like Frame Type, Start-of-Frame, Tag and End-of-frame, the state of this line are identical to TXD0.	Logic High

(1) Inactive level refers to the state of the pin while the module is not actively transmitting, or held in reset.

7.5.9.2.1.1 Configuring Device Pins

The GPIO mux registers must be configured to connect this peripheral to the device pins.

Some IO functionality is defined by GPIO register settings independent of this peripheral. For input signals, the GPIO input qualification must be set to asynchronous mode by setting the appropriate QUAL_SEL register bits to 0x3. The internal pullups can be configured with the PUPDSEL register bit. See the *General Purpose Input-Output (GPIO)* chapter for more details on the GPIO mux and settings.

7.5.9.2.2 FSI Interrupts

Each FSI module contains multiple interrupt sources that can be assigned to two different interrupt vectors: INT1 and INT2. Each interrupt source has an associated status flag, force, and clear bits in the EVT_STS, EVT_FRC, and the EVT_CLR registers, respectively.

Each interrupt can be assigned to either interrupt vector, INT1 and INT2, to allow for two priority levels. Alternately, the interrupt source can be prevented from generating any interrupt, though the status flag can still be set and monitored by software. The transmitter events are assigned to either interrupt vector in the TX_INT_CTRL register. The receiver events are assigned an interrupt vector using RX_INT1_CTRL and RX_INT2_CTRL registers. If an interrupt is not required, make sure the bit is not set in the respective INT_CTRL register.

7.5.9.2.2.1 Transmitter Interrupts

The transmitter can generate the following interrupts:

- **Frame Done (FRAME_DONE):** This event indicates that FSI has completed transmitting a frame.
- **Buffer Underrun (BUF_UNDERRUN):** This event indicates that the transmit buffer has experienced underrun. Buffer underrun occurs when the transmitter tries to read data from a location which has not yet be written to by the CPU, or DMA.
- **Buffer Overrun (BUF_OVERRUN):** The buffer overrun interrupt is generated when the buffer has experienced overrun. Buffer overrun can occur if a piece of data is overwritten before the data has been transmitted.
- **Ping Frame Triggered (PING_TRIGGERED):** The ping frame triggered interrupt is generated when the ping frame has been triggered. This bit is set when the ping counter has timed out or an external ping trigger event has occurred.

7.5.9.2.2.2 Receiver Interrupts

The receiver core is capable of generating interrupts from many different events:

- **Ping Watchdog Timeout (PING_WD_TO):** This event indicates that the ping watchdog timer has timed out. The receiver has not received a valid frame within the time period specified in the RX_PING_WD_REF register.
- **Frame Watchdog Timeout (FRAME_WD_TO):** This event indicates that the frame watchdog timer has timed out. The conditions of this timeout are set using the RX_FRAME_WD_CTRL register. As soon as the start of frame phase is detected, the frame watchdog counter starts counting from 0. The end of frame phase must complete by the time the watchdog counter reaches the reference value. If this does not happen, the watchdog times out and this event is generated. If this event occurs, the receiver must undergo a soft reset and subsequent resynchronization to resume proper operation.
- **CRC Error (CRC_ERR):** This error indicates that a CRC error has occurred. A CRC error is generated when the received CRC and the computed CRC do not match.
- **Frame Type Error (TYPE_ERR):** This error indicates that an invalid frame type has been received. If this error occurs, the receiver must undergo a soft reset and subsequent resynchronization to resume proper operation.
- **End-of-Frame Error (EOF_ERR):** This error indicates that an invalid end-of-frame bit pattern has been received. If this error occurs, the receiver must undergo a soft reset and subsequent resynchronization to resume proper operation.
- **Receive Buffer Overrun (BUF_OVERRUN):** This event indicates that an overrun condition has occurred in the receive buffer.
- **Receive Buffer Underrun (BUF_UNDERRUN):** This event indicates that an underrun condition has occurred in the receive buffer. This condition occurs when software reads an empty buffer.
- **Frame Done (FRAME_DONE):** This event indicates that a valid frame has been received without error.
- **Error Frame Received (ERR_FRAME):** This event indicates that an error frame has been received.
- **Ping Frame Received (PING_FRAME):** This event indicates that a ping frame has been received.
- **Frame Overrun (FRAME_OVERRUN):** This event indicates that a new frame has been received while the FRAME_DONE flag was still set.
- **Data Frame Received (DATA_FRAME):** This event indicates that a data frame has been received.

7.5.9.2.2.3 Configuring Interrupts

To configure interrupts on the FSI, the application must select the interrupt vector for each desired event using the TX_INT_CTRL register for the transmitter, and RX_INT1_CTRL_ALT1_ and RX_INT2_CTRL_ALT1_ registers for the receiver. There is no module-level interrupt enable bit to configure.

Note

If an event is registered for both interrupt vectors, both interrupts fire. There are no hardware checks for overlapping interrupt vector assignments.

7.5.9.2.2.4 Handling Interrupts

Inside the interrupt service routine (ISR), the user must clear the event flag using the EVT_CLR register and then acknowledge the CPU interrupt.

If the one event occurs multiple times before the corresponding bit is cleared by software, no new interrupt is generated.

If multiple events occur simultaneously, or very close in time, it is possible to handle multiple conditions within a single interrupt. Each flag is independently set by hardware and must be cleared by application software. If multiple different events occur, the ISR can handle each in whatever order is deemed necessary by the application. It is not advisable to clear the full interrupt status register in every ISR. This can cause the application to miss events that can be detrimental to the application. A sample sequence for handling interrupts on the receiver follows; the transmitter routine is similar.

- On receiving an interrupt, copy the current state of the receive event and error status flag register (RX_EVT_STS_ALT1_) into a local snapshot variable.
- Read all of the bits from the snapshot to determine the events that require action.
- Perform the necessary actions for each of the events seen in the snapshot.
- Write to the receive event and error clear register (RX_EVT_CLR_ALT1_) with the snapshot to clear only those interrupts that were set at the beginning of the ISR.
- Repeat this sequence for every generated ISR.

There is a chance that another event occurred during the just-handled ISR since only the snapshot of events was handled and then cleared; an event flag can still be set at the end of the ISR. As soon as the ISR completes, a new interrupt is generated and this flag is still set and can be handled accordingly.

Software accesses tied to multiple events and handled within the same ISR can cause race conditions that cause the software to not function as desired. For example, it is recommended to use different interrupt lines if the user wants to enable events for both ping and data frames. If both events are handled within the same interrupt line, the software can only respond to one of the events if both events occur close in time.

7.5.9.2.3 DMA Interface

Both the transmitter and receiver are capable of using the DMA for automatic data transfers. The DMA trigger is independent from the interrupt signals. DMA events are only triggered on the completion of a data frame.

The transmitter DMA trigger is enabled by setting TX_DMA_CTRL.DMA_EVT_EN to 1. The transmitter must also set TX_OPER_CTRL_LO_ALT2_.START_MODE to 0x2 to allow either a write to the TX_FRAME_CTRL.START bit or to the TX_FRAME_TAG_UDATA register to start the transmission.

The receiver DMA trigger is enabled by setting RX_DMA_CTRL.DMA_EVT_EN to 1.

Refer to Section 7.5.9.3.2 and Section 7.5.9.3.3 for more DMA information specific to each FSI Module.

7.5.9.2.4 External Frame Trigger Mux

The FSI has two muxes connected to the transmitter module. These muxes are used to select triggers to start ping frames, and generic frames. These muxes are independently configured for each type of frame. The application can select one trigger source per frame type. Use of these triggers are optional.

The external ping frame trigger is configured by setting TX_PING_CTRL_ALT1_.EXT_TRIG_SEL to the index of the desired trigger. TX_PING_CTRL_ALT1_.EXT_TRIG_EN must also be set to allow the trigger to generate a ping frame.

The generic frame trigger is configured by setting TX_OPER_CTRL_HI_ALT1_.EXT_TRIG_SEL to the index of the desired trigger. TX_OPER_CTRL_LO_ALT2_.START_MODE must be set to 0x1 for a frame to be transmitted by an external trigger.

Table 7-143. External Trigger Sources (including PING) and Their Index

Index	External Trigger Source
0	FSI_RX[x].PING_FRAME_TAG_MATCH

Table 7-143. External Trigger Sources (including PING) and Their Index (continued)

Index	External Trigger Source
1	FSI_RX[x].ERROR_FRAME_TAG_MATCH
2	FSI_RX[x].DATA_FRAME_TAG_MATCH
3	Tie low
4	FSI_RX[x].TRIG0
5	FSI_RX[x].TRIG1
6	FSI_RX[x].TRIG2
7	FSI_RX[x].TRIG3
8	EPWM0.SOCB
9	EPWM1.SOCB
10	EPWM2.SOCB
11	EPWM3.SOCB
12	EPWM4.SOCB
13	EPWM5.SOCB
14	EPWM6.SOCB
15	EPWM7.SOCB
16	EPWM8.SOCB
17	EPWM9.SOCB
18	EPWM10.SOCB
19	EPWM11.SOCB
20	EPWM12.SOCB
21	EPWM13.SOCB
22	EPWM14.SOCB
23	EPWM15.SOCB
24	EPWM16.SOCB
25	EPWM17.SOCB
26	EPWM18.SOCB
27	EPWM19.SOCB
28	EPWM20.SOCB
29	EPWM21.SOCB
30	EPWM22.SOCB
31	EPWM23.SOCB
32	EPWM24.SOCB
33	EPWM25.SOCB
34	EPWM26.SOCB
35	EPWM27.SOCB
36	EPWM28.SOCB
37	EPWM29.SOCB
38	EPWM30.SOCB
39	EPWM31.SOCB
40	ICSM_PORT0.16
41	ICSM_PORT0.17
42	ICSM_PORT0.18
43	ICSM_PORT0.19
44	ICSM_PORT1.16
45	ICSM_PORT1.17

Table 7-143. External Trigger Sources (including PING) and Their Index (continued)

Index	External Trigger Source
46	ICSM_PORT1.18
47	ICSM_PORT1.19
48	OUTPUTXBAR.OUT0
49	OUTPUTXBAR.OUT1
50	OUTPUTXBAR.OUT2
51	OUTPUTXBAR.OUT3
52	OUTPUTXBAR.OUT4
53	OUTPUTXBAR.OUT5
54	OUTPUTXBAR.OUT6
55	OUTPUTXBAR.OUT7
56	OUTPUTXBAR.OUT8
57	OUTPUTXBAR.OUT9
58	OUTPUTXBAR.OUT10
59	OUTPUTXBAR.OUT11
60	OUTPUTXBAR.OUT12
61	OUTPUTXBAR.OUT13
62	OUTPUTXBAR.OUT14
63	OUTPUTXBAR.OUT15

7.5.9.3 FSI Functional Description

7.5.9.3.1 FSI Functional Description

The Fast Serial Interface Transmitter and Receiver modules (FSI_TX/FSI_RX) are two completely independent modules on the device. Each module has an independent set of control registers, clocking, and interrupts. The following sections describe the frame format and the various initialization and configuration procedures for both the transmitter and receiver.

7.5.9.3.2 FSI Transmitter Module

The FSI transmitter module handles the framing of data, CRC generation, and signal generation of TXCLK, TXD0, and TXD1, as well as interrupt generation. The operation of the transmitter core is controlled and configured through programmable control registers. The transmitter control registers allow the CPU to program, control, and monitor the operation of the FSI receiver. The transmit data buffer is accessible by the CPU and the DMA.

The transmitter has the following features:

- Automated ping frame generation
- Externally triggered ping frames
- Externally triggered data frames
- Software-configurable frame lengths
- 16-word data buffer
- Data buffer underrun and overrun detection
- Hardware-generated CRC on data bits
- Software ECC calculation on select data
- DMA support

Figure 7-319 shows the high-level block diagram of the FSI transmitter. Figure 7-320 shows the block diagram of the transmitter core submodule.

The following sections describe the various aspects of the FSI transmitter in detail.

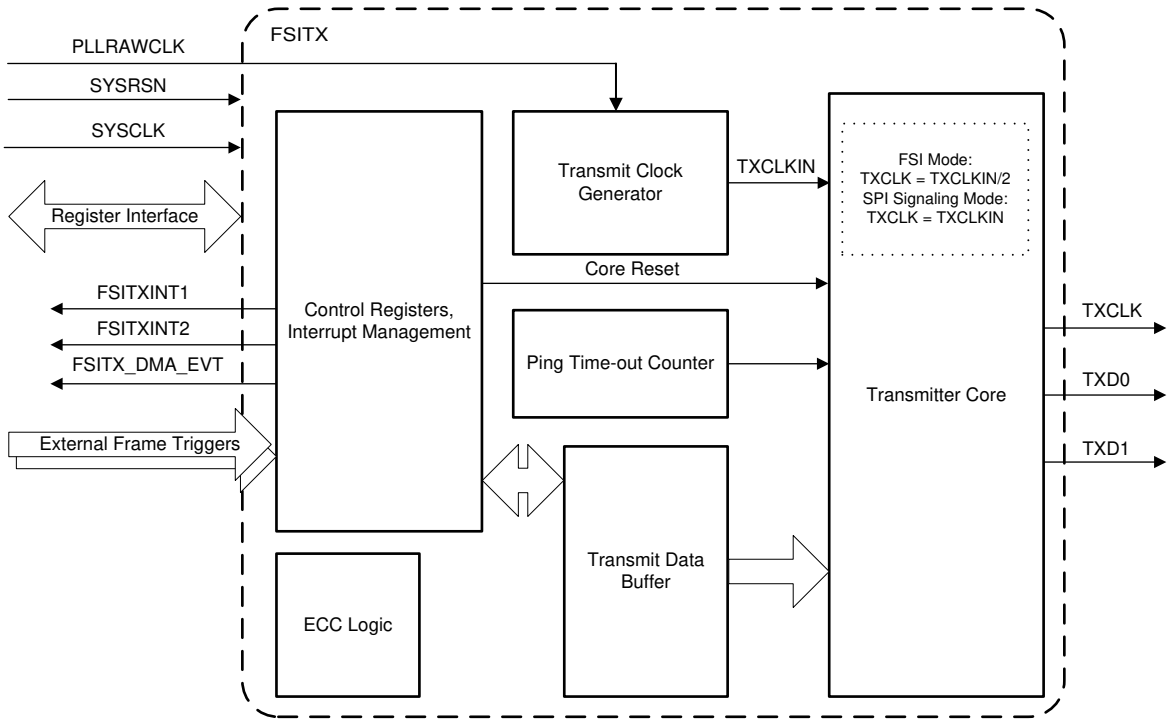


Figure 7-319. FSI Transmitter Block Diagram

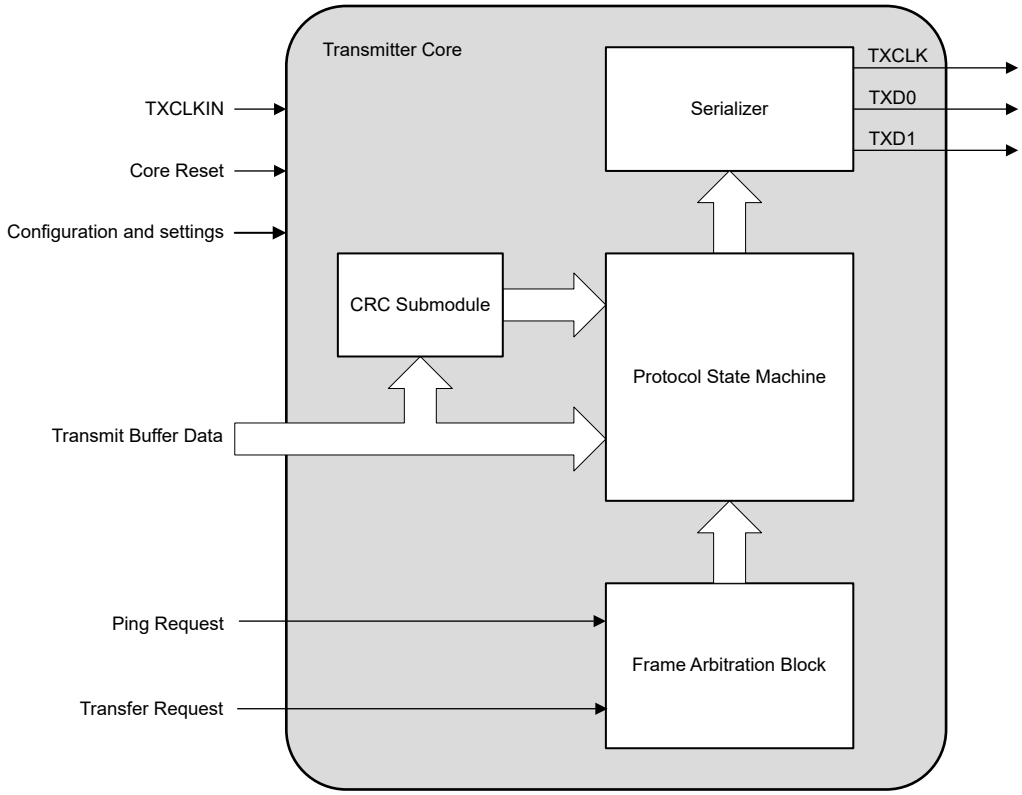


Figure 7-320. FSI Transmitter Core Block Diagram

7.5.9.3.2.1 Initialization

On the first initialization or after a module reset due to an underrun condition, the transmitter module executes the following initialization sequence to start or resume transmit operations.

1. Initialize the transmitter clock by setting TX_CLK_CTRL.CLK_RST to 1 and subsequently clearing the bit.
2. Set the clock to the transmitter core to PLLRAWCLK by setting TX_OPER_CTRL_LO_ALT2_.SEL_PLLCLK to 1.
3. Set the clock prescaler value to the desired rate by writing to TX_CLK_CTRL.PRESCALE_VAL.
4. Enable the transmitter clock divider by setting TX_CLK_CTRL.CLK_EN to 1.
5. Assert the transmitter module soft reset by writing 0xA501 to TX_MAIN_CTRL.
6. Wait four TXCLK cycles.
7. Release the transmitter core from reset by writing 0xA500 to TX_MAIN_CTRL.

After initialization and configuration, the transmitter module synchronizes with the receiver module before transmitting. The synchronization sequence is described in Section 7.5.9.4.1.

CAUTION

Do not change TX_CLK_CTRL.PRESCALE_VAL while the clock is enabled (TX_CLK_CTRL.CLK_EN = 1). Doing so can cause undefined behavior.

7.5.9.3.2.2 FSI_TX Clocking

The transmitter core registers and control logic run off of the device system clock (SYSCLK).

The FSI Transmit Clock (TXCLK) is derived from PLLRAWCLK. PLLRAWCLK is divided down by configuring the clock prescaler value (TX_CLK_CTRL.PRESCALE_VAL) then setting the clock divider enable bit (TX_CLK_CTRL.CLK_EN). The clock prescaler value can be set to divide PLLRAWCLK by 1 (TX_CLK_CTRL.PRESCALE_VAL = 0x0 or 0x1) through 255(TX_CLK_CTRL.PRESCAL_VAL = 0xFF). Though TXCLK and SYSCLK are both derived from PLLRAWCLK, TXCLK is asynchronous with respect to SYSCLK.

CAUTION

TXCLK must never be configured to be faster than SYSCLK/2.

7.5.9.3.2.3 Transmitting Frames

On the transmitter, the ping frame is the only frame that can be set up and transmitted without any further software or DMA intervention. Ping frames can be transmitted by any (or all) of the three sources: automatic ping timer, software, or external triggers.

Each available frame type can be sent multiple ways. Generically, the following steps must be executed before the frame is sent. These steps can be executed in any order before the start condition is set.

1. Configure the frame type
2. Set the frame tag
3. If the frame to be sent is a data frame:
   - Set the user data
   - Write to the data buffer
   - Set the word length if the frame is a software defined frame length
4. Set the start condition

Note

Transmit Frame Start Restriction:

A new frame transmission can be initiated by one of the methods selected in the TX_OPER_CTRL_LO_ALT2_.START_MODE bits. If there is already a PING frame transmission taking place, due to a hardware initiated PING timer, the new frame transmission begins as soon as the on-going PING transmission is completed.

Once a START of frame has been initiated, the next START of frame is recognized when the first frame has started transmitting the End-of-Frame (EOF) field. If a new START trigger arrives before the current transmission has reached the EOF field, the trigger is lost without a notification.

Note

There is no hardware check implemented to check whether the type field written by software is valid or not. If an invalid type is used and a frame transmission is initiated, the behavior is:

- The transmitted frame structure is exactly like an NWORD data frame. The size of the data frame is determined by the value in the TX_FRAME_CTRL.N_WORDS register.
- The frame type field of the transmitted data frame is transmitted as programmed. If this is received by an FSI receiver, a Type error is generated.

This mechanism can be used for force a Type error in a received frame for testing purposes.

The following sections describe the specific configuration for each frame type and start condition.

7.5.9.3.2.3.1 Software Triggered Frames

The most basic way to transmit a data frame is through software. Each step must be handled by the application. To send a data frame using software, the following steps must be executed. Steps 1-6 can be executed in any order before setting TX_FRAME_CTRL.START. Some fields do not need to be reconfigured for every transmission. The frame tag, user data, and frame type are sticky and are retransmitted in the subsequent frame unless modified by software.

1. Write the data to be transmitted to the next location of the transmit data buffer.
2. Set TX_FRAME_CTRL.FRAME_TYPE to the appropriate value for the type of frame to be transmitted.
3. Set TX_FRAME_CTRL.N_WORDS to 1 less than the number of words to be transmitted if TX_FRAME_CTRL.FRAME_TYPE is set to 0011, the frame type of the software-defined length data frame. That is, if 16 words are transmitted, N = 16, set TX_FRAME_CTRL.N_WORDS to 15.
4. When the frame is assembled before transmitting, the FSITX hardware calculates the CRC to be transmitted. If TX_OPER_CTRL_LO_ALT2_.SW_CRC is 1, the application can calculate a custom CRC value and then set TX_USER_CRC to the result.
5. Set TX_FRAME_TAG_UDATA.FRAME_TAG to the desired tag.
6. Set TX_FRAME_TAG_UDATA.USER_DATA to the desired user data.
7. Set TX_FRAME_CTRL.START to 1 to initiate the transmission of the data frame.

Once the frame transmission has started, the TX_FRAME_CTRL.START is cleared by hardware. To monitor if the frame has completed, the software can poll TX_EVT_STS.FRAME_DONE.

7.5.9.3.2.3.2 Externally Triggered Frames

The transmitter can transmit frames when triggered by an external source. See Section 7.5.9.2.4 for more information on the available external triggers.

To transmit frames using an external trigger, the application must follow the same procedure as described in Section 7.5.9.3.2.3.1. The only difference is that in Step 7, the start condition is automatically set when the external trigger condition is met rather than by software.

Note that by externally triggering frames, the frame information to be sent is pulled from the same registers described in the previous section. Because of this, it is possible to send any type of frame from an external trigger including ping, error, and data frames. Also, there is no hardware mechanism by which the FSI can determine if multiple triggers occur. The FSITX takes the data as is, and the application software makes sure that this data has been updated as necessary.

Using TX_EVT_STS fields either by polling or by interrupts, the application can populate or update the frame information to be sent in the next frame

7.5.9.3.2.3.3 Ping Frame Generation

Assuming the FSI transmitter has already been properly initialized, the following sequences can be used to configure and send ping frames.

7.5.9.3.2.3.3.1 Automatic Ping Frames

To generate periodic ping frames, the following steps must be followed:

1. Initialize the ping counter by writing 1 to TX_PING_CTRL_ALT1_.CNT_RST.
2. Set the desired ping tag to TX_PING_TAG.TAG.
3. Set the ping timer reference value to TX_PING_TO_REF.TO_REF.
4. Enable the ping timer by writing 1 to TX_PING_CTRL_ALT1_.TIMER_EN.

The ping timer is a free-running counter that counts up from 0. The current value of the ping timer counter is found in TX_PING_TO_CNT. When the current value of TX_PING_TO_CNT matches the reference value TX_PING_TO_REF.TO_REF, the TX_EVT_STS.PING_TRIGGERED is set. TX_PING_TO_CNT resets to 0 and resumes counting until the next match has occurred or the ping timer is halted by software (TX_PING_CTRL.TIMER_EN is set to 0).

7.5.9.3.2.3.3.2 Software Triggered Ping Frame

Software can also manually generate a ping frame. The process for sending a ping frame with software is very similar to sending the other types of frames. The following steps must be followed:

1. Set TX_FRAME_CTRL.FRAME_TYPE to 0000’b to denote that the frame being sent is a Ping Frame.
2. Set TX_FRAME_TAG_UDATA.FRAME_TAG to the desired value.
3. Write 1 to TX_FRAME_CTRL.START. This starts the transmission.

Once the frame transmission has started, the TX_FRAME_CTRL.START is cleared by hardware. To monitor if the frame has completed, the software can poll TX_EVT_STS.FRAME_DONE.

7.5.9.3.2.3.3.3 Externally Triggered Ping Frame

The last source for generating ping frames is an external trigger. One of up to 32 different triggers can be selected. See Section 7.5.9.2.4 for the list of input sources.

CAUTION

Ping frames can be triggered by both an external trigger source and the internal ping timer. If TX_PING_CTRL_ALT1_.EXT_TRIG_EN is set to 1, the external trigger source takes precedence and the ping timer is ignored.

7.5.9.3.2.3.4 Transmitting Frames with DMA

The FSI transmitter can send data that is continuously applied with the DMA. A DMA trigger is generated every time a data frame transmission is completed. This is concurrent with the FRAME_DONE signal that sets the TX_EVT_STS.FRAME_DONE flag.

To transmit continuous data with the DMA, some configurations need to be made on the transmitter:

First, set TX_DMA_CTRL.DMA_EVT_EN to 1. This allows the DMA trigger to propagate to the DMA module. Next, TX_OPER_CTRL_LO_ALT2_.START_MODE must be set to 0x2. The transmitter is now able to start a transmission using a software write to TX_FRAME_CTRL.START or TX_FRAME_TAG_UDATA..

The DMA must also be configured properly for the FSI to send the data. One way of using the DMA to continuously feed the transmit buffer is:

- Set up two DMA channels to be triggered by the same FSI transmitter and DMA trigger.
- Configure one channel to fill the transmit buffer.
- Configure the other channel to set the frame tag and user data fields
- Since the FSI transmit buffer is a 16-word circular buffer, make sure the DMA channel servicing the data buffer wraps the after 16 words are copied.

Note

Because the frame tag and user data must be written in to initiate the transmission of the frame, use two consecutive DMA channels. This makes sure that the DMA channels are always executed in sequence. The DMA channel servicing the data buffer must be the lower numbered channel and the tag/user data channel must be the next. For example, configure DMA channel 3 to service the data buffer, and configure DMA channel 4 to service the tag and user data.

7.5.9.3.2.4 Delay Line Control

The transmitter module has a programmable delay line on each of the external signal inputs: TXCLK, TXD0, and TXD1. The delay elements introduce delays on the respective lines and are placed before the FSITX signals are sent to the TDM signal selection mux (controlled by the SEL_TDM_PATH signal). This is to facilitate adjustment for signal delays introduced by system level components such as signal buffers, ferrite beads, isolators, and so on, or board delays such as uneven trace lengths, long cable length, and so on. The length of the delay is controlled by setting the TX_DLY_LINE_CTRL register values for each line. By default, no delay is introduced by the delay line elements. The delay values should only be adjusted while the FSITX is held in soft reset, ensuring that there are no active transmissions during this process. Figure 7-321 shows a representation of the delay line circuitry for the input signals. The implementation for TXCLK, TXD0, and TXD1 are replicas of this diagram. All circuits will behave similarly.

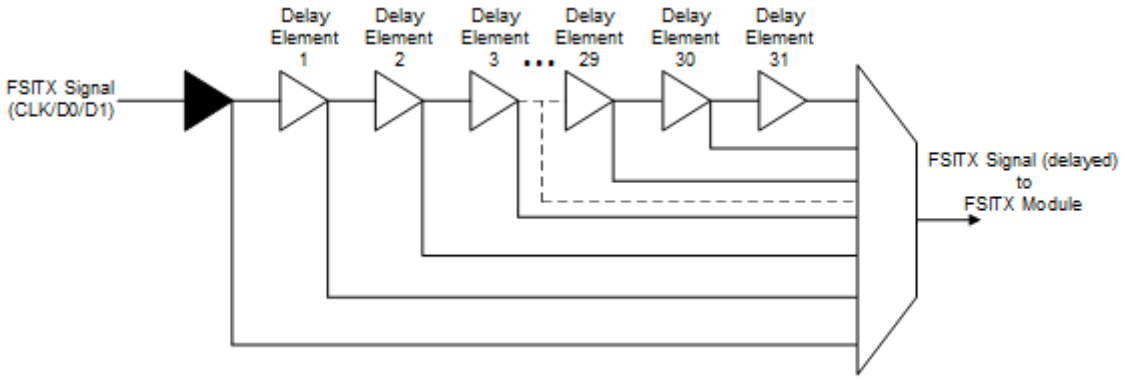


Figure 7-321. Delay Line Control Circuit

For more information on skew compensation, refer to the *Fast Serial Interface (FSI) Skew Compensation Application Report*.

7.5.9.3.2.5 Transmit Buffer Management

The FSI transmitter has a 16-word buffer that the FSI transmitter pulls data to transmit. This buffer is implemented as a circular buffer, not a FIFO, so some care must be taken to properly interpret buffer overrun and underrun, as well as the TX_BUF_PTR_STS register. These flags and pointers work under the assumption that the software or DMA is using the buffer as a circular buffer. This mode of operation is the only way that the overrun, underrun, and pointer status are meaningful. If data is being sourced by the DMA and there is some other periodic trigger mechanism trying to initiate transfers, underrun becomes a critical error. If an underrun happens, a buffer went out of sync. This not only affects the current transfer, but all future transfers also cannot be sure of due to the ring buffer. Under such conditions, the underrun needs a soft reset to cleanly recover. Alternately, the software can manually stop the transmitting, reset the buffer pointers, clear the remaining error conditions, and then restart transmission. The software method involves a few steps, while the soft reset is a single action and makes sure of a full reset of the control registers.

Due to the flexibility of the transmit buffer, software can implement a simple ping-pong buffer or randomly load and send from any location of the buffer. If the buffer is used in this manner, error flags and status fields can be ignored without adversely affecting the transmitter capability. Additionally, the CURR_WORD_CNT is also invalid if used in this way. The application can set the buffer pointer manually by writing the 4-bit index to TX_BUF_PTR_LOAD. This forces the transmitter to start picking the data from the indicated location in the buffer.

7.5.9.3.2.6 CRC Submodule

The FSI transmitter can supply the CRC to the frame being transmitted through the embedded hardware CRC submodule or by supplying a user-defined value. This is controlled by setting TX_OPER_CTRL_LO_ALT2_.SW_CRC appropriately.

If hardware CRC generation is selected (TX_OPER_CTRL_LO_ALT2_.SW_CRC = 0, the default), the CRC is computed by hardware on the data and user data fields using the CRC polynomial 0x7 ($x^8 + x^2 + x + 1$). The transmitter module automatically computes the CRC on the data fields without user intervention when the frame is transmitted. For more information on how the CRC is generated by the CRC submodule, refer to Section 7.5.9.3.7.

If software CRC generation is selected (TX_OPER_CTRL_LO_ALT2_.SW_CRC = 1), the CRC must be computed by software and placed in the TX_USER_CRC register. The next frame to be transmitted uses the value placed in the TX_USER_CRC register in place of the CRC value generated by the hardware.

As the TX_USER_CRC register is software-programmable, the application can use this field as an extra data field for application-specific purposes. If TX_USER_CRC is used in this manner, the CRC detection on the receiver is not valid and must be ignored.

7.5.9.3.2.7 Conditions in Which the Transmitter Must Undergo a Soft Reset

Unlike the receiver, there are no detectable errors that require a soft reset. A buffer overrun or underrun interrupt can or cannot require a soft reset to resume proper operation. This determination is up to the application software. Refer to Section 7.5.9.3.2.5 for more information on the transmit buffer.

7.5.9.3.2.8 Reset

The entire transmitter module and all transmitter registers are reset by SYSRSn. The transmitter core is reset by SYSRSn or by writing a 1 to TX_MAIN_CTRL.CORE_RST.

A module reset causes the registers to be reset to their default state.

7.5.9.3.3 FSI Receiver Module

The receiver module interfaces to the FSI clock (RXCLK), and data lines (RXD0 and RXD1) after they pass through an optional programmable delay line. The receiver core handles the data framing, CRC computation, and frame-related error checking. The receiver bit clock and state machine are run by the RXCLK input, which is asynchronous to the device system clock.

The receiver control registers allow the CPU to program, control, and monitor the operation of the FSI receiver. The receive data buffer is accessible by the CPU and the DMA.

The receiver core has the following features:

- 16-word data buffer
- Multiple supported frame types
- Ping frame watchdog
- Frame watchdog
- CRC calculation and comparison in hardware
- ECC detection
- Programmable delay line control on incoming signals
- DMA support
- FSI-SPI compatibility mode

Figure 7-322 provides a high-level overview of the internal modules present in the FSI receiver. Figure 7-323 shows a view of the FSI receiver core submodule. Not all data paths and internal connections are shown.

The following sections describe the various aspects of the FSI receiver module.

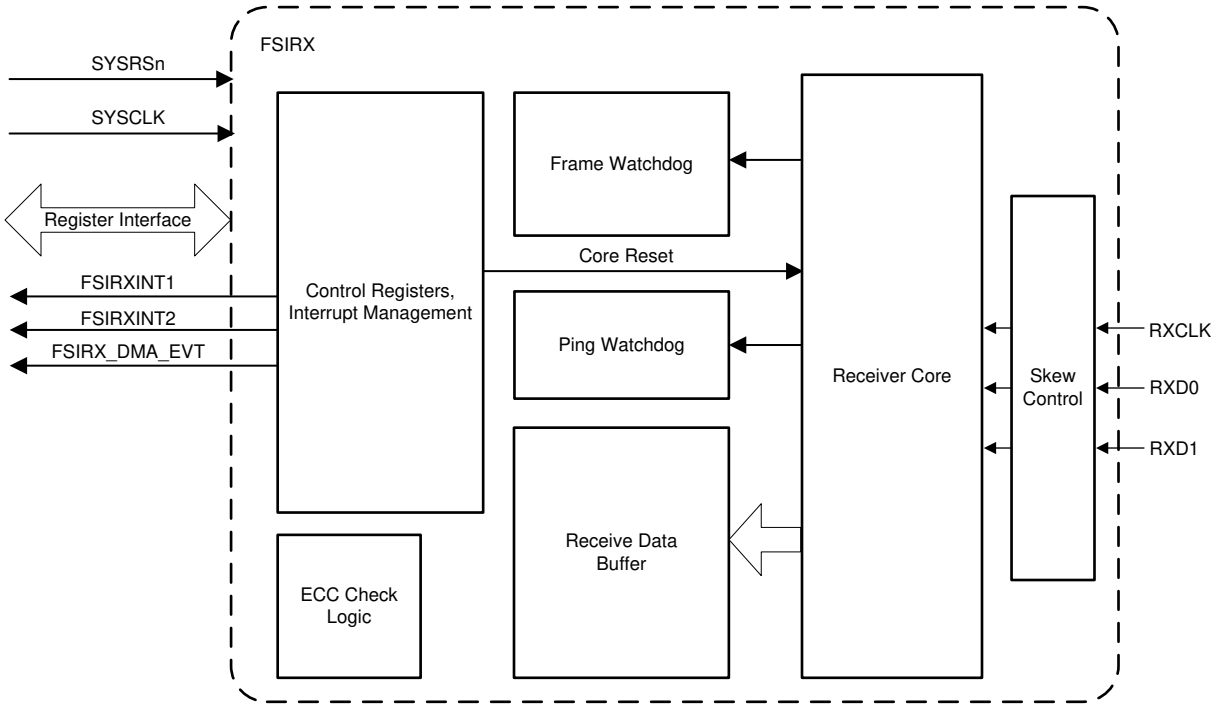


Figure 7-322. FSI Receiver Block Diagram

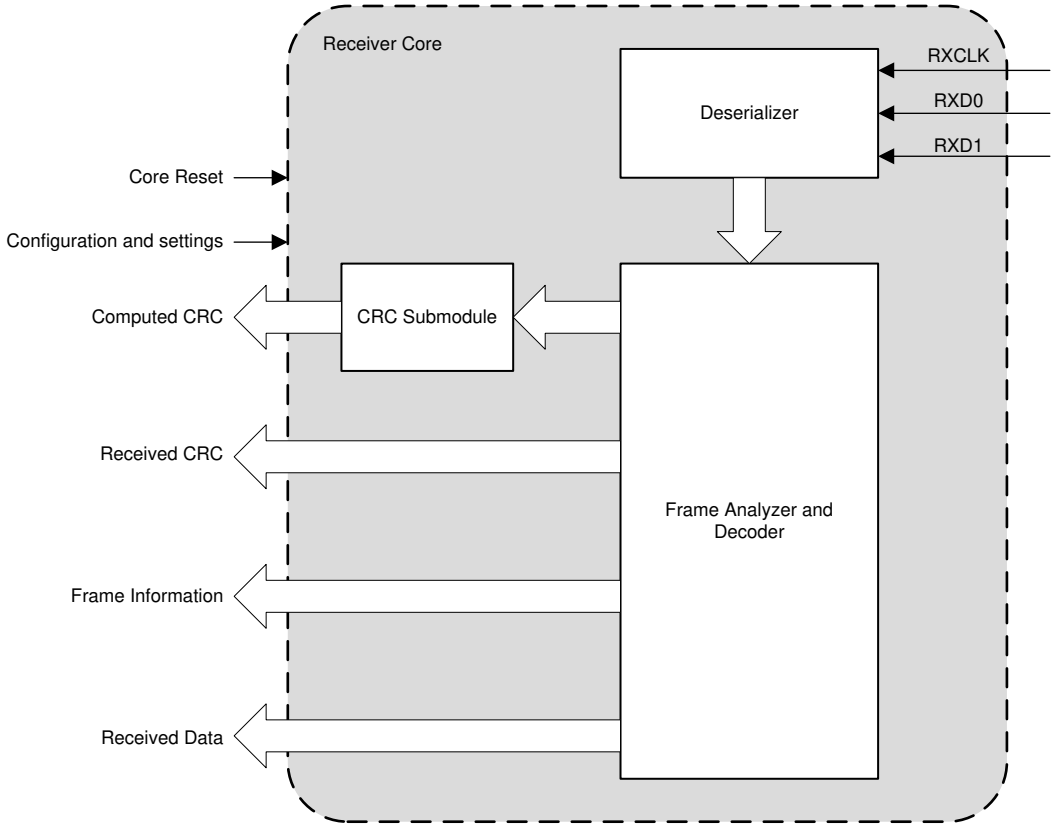


Figure 7-323. FSI Receiver Core Block Diagram

7.5.9.3.3.1 Initialization

On the first initialization or after a module reset following any frame error, the receiver module asserts and releases the receiver core reset bit (RX_MAIN_CTRL_ALTC_.CORE_RST) prior to any other initialization. Once the receiver module is initialized, the following steps are executed:

1. If required, assign interrupt sources to the necessary interrupt line.
2. If required, configure the ping watchdog to periodically check for an active link to the transmitter. See Section 7.5.9.3.3.4 for configuration details.
3. If required, configure the frame watchdog to make sure that each frame is received within a predetermined window. See Section 7.5.9.3.3.5 for configuration details.
4. Initialize the receive buffer pointer by writing to the RX_BUF_PTR_LOAD register. Received data is placed into the buffer starting with the address loaded in this register.
5. Make sure all errors and flags have been cleared from the RX_EVT_STS_ALT1_ register.

At this point the receiver is ready to receive any incoming frames. Software can now either poll on the RX_EVT_STS_ALT1_ register for various conditions. For example, when the RX_EVT_STS_ALT1_.FRAME_DONE and no other flags are set, the receiver has successfully received a frame without error.

Next, the application configures the various features such as the ping and frame watchdogs, DMA, external triggering, and so on. These features are described in subsequent sections. The receiver module is now ready to synchronize with the transmitter then begin reception. The synchronization sequence is described in Section 7.5.9.4.1.

7.5.9.3.3.2 FSI_RX Clocking

The receiver module registers and control logic are clocked by the device system clock (SYSCLK). The receiver state machine is clocked by the receiver input clock pin (RXCLK).

CAUTION

RXCLK must never be faster than SYSCLK.

7.5.9.3.3.3 Receiving Frames

Once the receiver has been properly configured and synchronized, incoming messages are handled as described below. Note that there is no equivalent to a chip-select signal to gate incoming data. Every valid clock edge latches data into the receiver.

The header information of the received frame is placed in their respective register fields.

- RX_FRAME_INFO.FRAME_TYPE contains the received frame type.
- RX_FRAME_TAG_UDATA.FRAME_TAG contains the received frame tag.
- RX_FRAME_TAG_UDATA.USER_DATA contains the received user data.

If any error conditions occur during reception such as a CRC mismatch, frame error, frame timeout, buffer overrun, or ping watchdog timeout, the corresponding flag is set in the RX_EVT_STS_ALT1_ register.

Note

If at any point during operation a frame error occurs, the receiver module must be reset and re-synchronized with the transmitter before the next frame can be successfully received. The follow errors are classified as frame errors:

- Type error
- CRC error
- End of frame error

7.5.9.3.3.3.1 Receiving Frames with DMA

The FSI receiver can continuously receive data and move the data from the receiver buffer with the DMA. A DMA trigger is generated every time a data frame has been received. This is concurrent with the FRAME_DONE signal that sets the RX_EVT_STS_ALT1_.FRAME_DONE flag. To receive continuous data with the DMA, some configurations need to be made on the receiver.

First, set RX_DMA_CTRL.DMA_EVT_EN to 1. This allows the DMA trigger to propagate to the DMA module. The receiver is now able to trigger a DMA event upon the reception of a data frame.

The DMA must also be configured properly for the FSI to receive the data. One way for using the receiver to continuously feed the DMA is:

- Set up two DMA channels to be triggered by the FSI Receiver DMA Trigger.
- Configure one DMA channel to copy data from the receive buffer to a larger data buffer.
- Configure the next DMA channel to copy the received frame tag and user data to another data buffer.
- Since the FSI receive buffer is a 16-word circular buffer, make sure the DMA channel servicing the data buffer wraps after 16 words are copied.

Unlike the transmitter, there is no requirement to have the DMA channel which is handling the data buffer, execute before the DMA channel handling the received tag and user data.

7.5.9.3.3.4 Ping Frame Watchdog

The ping frame watchdog is a hardware-enabled automatic error detection of the connection status to the transmitter. This watchdog monitors the time elapsed between ping frames. If the transmitter has been set up to periodically send out a ping frame, the receiver can be set up to monitor whether this frame has been received

within a specified amount of time. If the time between ping frames has exceeded the programmed number of clock cycles, an event is triggered that can generate an interrupt or be monitored by software.

This watchdog has a dedicated counter that is reset and restarted upon the successful reception of a ping frame. The watchdog counter is incremented at the rate of SYSCLK. Optionally, the watchdog can be configured to be reset upon the successful reception of any frame. This option allows the receiver to monitor for any successful frame to indicate that the connection is still alive and the transmitter is still functioning as expected.

To configure the ping frame watchdog for operation:

1. Reset the ping watchdog counter by setting RX_PING_WD_CTRL.PING_WD_RST to 1 and then subsequently clearing the bit to 0.
2. Set RX_OPER_CTRL.PING_WD_RST_MODE to the desired watchdog reset event, set to 0 for ping frames only or set to 1 for any frame.
3. Set RX_PING_WD_REF to the maximum time between frames. Add 10 additional SYSCLK cycles to account for clock synchronization.
4. Enable the ping watchdog by setting RX_PING_WD_CTRL.PING_EN to 1.

The ping watchdog is now enabled and can now monitor for ping frames.

If the RX_PING_WD_CNT value reaches the value programmed in RX_PING_WD_REF, the RX_EVT_STS.PING_WD_TO flag is set. If configured, an interrupt can be generated on this event.

7.5.9.3.3.5 Frame Watchdog

The frame watchdog is an additional feature the receiver can use to monitor for any error conditions. This dedicated watchdog monitors the duration for a single frame to be received. The watchdog starts incrementing at the time the receiver detects a proper start of frame condition. If the end of frame condition is not detected within the expected number of SYSCLK cycles, the frame watchdog is triggered that can generate an interrupt or be monitored by software.

This watchdog is automatically started and stopped at the start-of-frame and end-of-frame conditions, respectively. The frame watchdog is connected to SYSCLK.

To configure the frame watchdog for operation:

1. Reset the frame watchdog counter by setting RX_FRAME_WD_CTRL. FRAME_WD_CNT_RST to 1 and then subsequently clearing the bit to 0.
2. Set RX_FRAME_WD_REF.FRAME_WD_REF to the maximum number of SYSCLK cycles expected to be in the longest frame that can be received. Add an additional 10 SYSCLK cycles to account for clock synchronization.
3. Enable the frame watchdog by setting RX_FRAME_WD_CTRL.FRAME_WD_CNT_EN to 1.

The frame watchdog is now enabled and can detect a failed frame.

If the RX_FRAME_WD_CNT reaches the value programmed in RX_FRAME_WD_REF, the RX_EVT_STS_ALT1_.FRAME_WD_TO flag is set. If enabled, an interrupt can be generated on this event.

If the frame watchdog interrupt ever occurs, the receiver core is in an invalid state to receive a new transmission. The only way to recover from a frame watchdog time out is to undergo a soft reset, and subsequently resynchronizing with the transmitter.

7.5.9.3.3.6 Delay Line Control

The receiver module has a programmable delay line on each of the external signal inputs: RXCLK, RXD0, and RXD1. The delay elements introduce delays on the respective lines. This is to facilitate adjustment for signal delays introduced by system level components such as signal buffers, ferrite beads, isolators, and so on, or board delays such as uneven trace lengths, long cable length, and so on. The length of the delay is controlled by setting the RX_DLY_LINE_CTRL register values for each line. By default, no delay is introduced by the delay line elements. The delay values must only be adjusted while the FSIRX is held in soft reset, making sure that there are no active transmissions during this process. Figure 7-324 shows a representation of the delay line circuitry for the input signals. The implementation for RXCLK, RXD0, and RXD1 are replicas of this diagram. All circuits behave similarly.

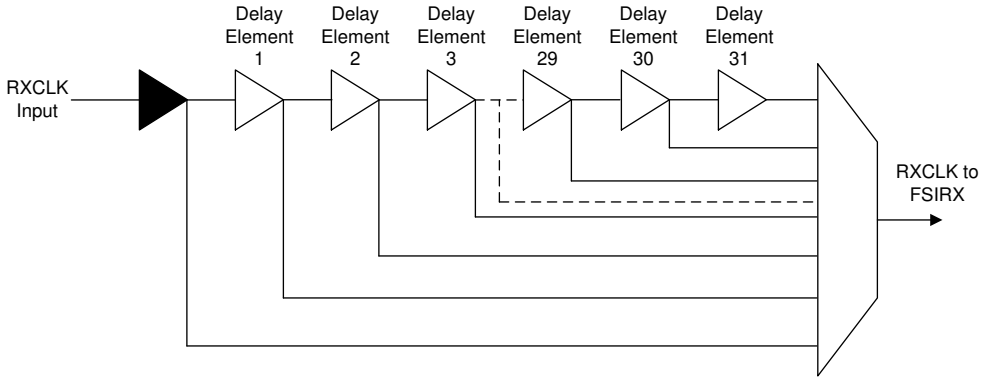


Figure 7-324. Delay Line Control Circuit

For more information on skew compensation, refer to *Fast Serial Interface (FSI) Skew Compensation*.

7.5.9.3.3.7 Buffer Management

The FSI receiver has a 16-word buffer that the data is copied to when the data has been received. This buffer is implemented as a circular buffer, not a FIFO, so some care must be taken to properly interpret buffer overrun and underrun as well as the RX_BUF_PTR_STS register. These flags and pointers work under the assumption that the software or DMA is using the buffer as a circular buffer. If the receiver state machine enters into an erroneous state, there is no way for software to cleanly handle this because there is no specified receive clock. For the receiver to detect a clean resynchronization, the state machine needs to be operational and not in the error state. The only way to recover from the error state is to reset the entire receiver module. For overrun and underrun, the receiver can no longer verify that values in the buffer are valid. As such, the best way to recover is to reset the FSI and resynchronize with the transmitter.

Due to the flexibility of the receive buffer, it is possible for software to implement a simple ping-pong buffer, or to randomly receive and read from any location of the buffer. If the buffer is used in this manner, these flags and status fields can be ignored without adversely affecting the receiver capability. Additionally, the CURR_WORD_CNT is also invalid if used in this way. The application can set the buffer pointer manually by writing the 4-bit index to RX_BUF_PTR_LOAD. This forces the receiver to start storing the received data starting at the indicated location in the buffer.

7.5.9.3.3.8 CRC Submodule

The receive module automatically calculates the CRC on the incoming data. The received CRC value is placed into RX_CRC_INFO.RX_CRC. The CRC value calculated by hardware on the received data is placed into RX_CRC_INFO.CALC_CRC. These values are compared by hardware and RX_EVT_STS_ALT1_.CRC_ERROR is set if there is a mismatch. The receiver can generate an interrupt based on RX_EVT_STS_ALT1_.CRC_ERROR if enabled.

Since the CRC is only used in data frames, the values found in RX_CRC_INFO.RX_CRC and RX_CRC_INFO.CALC_CRC are undefined during ping and error frames.

For more information on how the CRC is calculated, refer to Section 7.5.9.3.7.

If the transmitting module is sending a software-defined CRC value (FSITX.TX_OPER_CTRL_LO_ALT2_.SW_CRC = 1), the receiver module triggers a CRC error event if the received value does not match the hardware-calculated value. As this is an application-level decision, the FSIRX can safely disregard the CRC error event. Application software needs to calculate and verify the incoming CRC using the same custom algorithm used on the transmitter and act appropriately.

The CRC field can also be used as an application-specific value, not a CRC. The application can use the RX_CRC_INFO.RX_CRC as required. All CRC errors and flags can be ignored in this situation.

7.5.9.3.3.9 Using the Zero Bits of the Receiver Tag Registers

The receiver tag registers (receiver frame tag and user data (RX_FRAME_TAG_UDATA) register and receiver ping tag (RX_PING_TAG) register)) have the least-significant bit set to 0. The actual received tag is in the bit positions 4:1. The reason for this is to facilitate user software to create a table of functions that can be called depending on the tag value. A function pointer needs a 32-bit storage space and, hence, each successive pointer is offset by 2. If the first pointer is at address x, then the second pointer is at address x + 2, the third at address x + 4, and so on. By keeping the LSB to 0, the five bits of the tag register (bits 4:0) can now be directly used as an index into a table of function pointers.

7.5.9.3.3.10 Conditions in Which the Receiver Must Undergo a Soft Reset

The receiver receives data on every clock edge. While there are specific patterns that determine the a start of a frame, and denote the end of a frame, these patterns are able to occur at any point during normal operation inside of the frame. If there ever is a point at which the receiver fails to detect a successful frame, the module must be reset to make sure that subsequent frames are received properly.

When any of the following errors occur in a received frame, the receiver can be required to be reset and resynchronized with the transmitter:

- Frame type error
- End of frame error
- Ping frame watchdog timeout
- Frame watchdog timeout
- Receiver in an invalid state due to noisy clock

The receiver core status (RX_VIS_1.RX_CORE_STS) can be monitored to determine if the receiver core has entered into an error state requiring a soft reset to resume communication. Incorrect frame type and end of frame errors always cause this bit to become set. A soft reset is required in these cases. A frame watchdog timeout always requires a reset due to the fact that the receiver state machine is still expecting more information when the watchdog timed out. RX_CORE_STS can be used to determine if a noise event was the cause of the failed frame. The ping frame watchdog also does not cause RX_CORE_STS to be set. Similar to the frame watchdog, a corrupt receiver may not be the reason for the ping frame to have timed out. The transmitter could have gone offline and never sent a ping frame. Alternately, during idle time, a noise event could have occurred, thereby putting the receiver into a corrupt state. As the receiver is able to detect this during the ping frame watchdog timeout interrupt handler, this type of event is not lost and the application can act appropriately.

As the receiver is clocked by RXCLK, not SYSCLK, a noisy clock or data line can cause some internal design constraints to be violated, putting the receiver core logic into undefined states. Make sure that the clock and data lines satisfy the Electrical Characteristics and timing requirements of the FSI module found in the device data sheet. Failure to do so can cause the receiver state machine to go into an unrecoverable error state. The receiver can only be recovered by undergoing a soft reset. To determine the state of the receiver core after an unexpected frame error, the application must check the receiver core status bit.

In addition to the above errors, buffer overrun or underrun can warrant a soft reset to resynchronize with the local application software. Refer to Section 7.5.9.3.3.7 for more information on the receive buffers. The requirement of resetting the receiver due to overrun or underrun is up to the application.

After the receiver has been placed into soft reset, the application must notify the other device's transmitter to begin a new synchronization phase. The simplest way to achieve this is through a ping or error frame sent with a designated tag. If the application is not using the FSITX on the device with the detected error, some other method must be established. The other device must stop transmitting and begin a new synchronization phase.

7.5.9.3.3.11 FSI_RX Reset

The receiver module and the registers are reset by SYSRSn. The receiver core is reset by SYSRSn or by writing a 1 to RX_MAIN_CTRL_ALTC_.CORE_RST.

A module reset causes the registers to be reset to their default state. After a module reset, the receiver module must be re-initialized and the data link re-established.

7.5.9.3.4 Frame Format

The FSI module transmits and receives information in frames. Each frame contains multiple phases where different information can be found. The number of phases as well as the total length of the frame varies depending on the frame type being transmitted. Frames can be as short as 16-bits long for a ping or error frame or 288-bits long for a 16-word data frame.

In normal transmission mode, there are four preamble clock edges before the start of the frame and four post-frame clock edges (postamble). Data is transmitted on both edges of the clock (double data rate). The basic frame structure is shown in Table 7-144. Each phase of the frame (such as start-of-frame, frame type, and so on) is transmitted with the most-significant bit first. Table 7-144 describes the basic frame structure used by the FSI and adapted according to which frame type is transmitted.

Table 7-144. Basic Frame Structure

Idle State	Preamble	Start of Frame	Frame Type	User Data	Data Words	CRC Byte	Frame Tag	End of Frame	Postamble	Idle State
	1111	1001	4 bits	8 bits	1-16 words	8 bits	4 bits	0110	1111	

The FSI also supports a FSI-SPI compatibility mode. The SPI compatible frame structure is similar to a standard FSI frame, but there are differences. Refer to Section 7.5.9.3.10 for more information on how to configure and use the FSI-SPI compatibility mode.

Note

One word of the FSI refers to 16 bits.

The terms “frame” and “packet” can be used interchangeably to describe the signaling format of the FSI.

7.5.9.3.4.1 FSI Frame Phases

The different phases of the frame structure are described in detail.

- **Idle State:** During the idle state, the clock and data lines are driven high, the inactive state.
- **Preamble:** The preamble phase contains four clock edges (or two complete clock pulses) with the data signals held in the high state. These clock edges serve to flush the receiver logic and prepare the receiver logic for receiving a new frame. This phase is not present in SPI compatibility mode.
- **Start of Frame:** The start of frame phase contains two clock pulses with four bits, 1001, transmitted on the data lines.
- **Frame Type:** The frame type phase contains two clock pulses with the 4-bit frame type code being transmitted on the data lines. The different frame types are described in detail in Section 7.5.9.3.4.2. The transmitter must set the TX_FRAME_CTRL.FRAME_TYPE field before transmitting a frame. The received frame type is stored in the RX_FRAME_INFO.FRAME_TYPE.
- **User Data:** The user data phase contains a fully user-configurable data field. There are no restrictions on how this field is used. This phase is only available in data frames. The user data to be transmitted is set by writing to TX_FRAME_TAG_UDATA.USER_DATA. The received user data is stored in RX_FRAME_TAG_UDATA.USER_DATA.
- **Data:** The data phase contains the data that is being transmitted. The data is pulled from the transmit buffer of the transmitter and is placed in the receive buffer of the receiver. Word 0 is transmitted first. This phase is only present in data frames. Depending on the type of frame transmitted, this can contain anywhere between 1 and 16 words depending on the frame type selected. More information on data frames is found in Section 7.5.9.3.4.2.3.
- **CRC Byte:** The CRC byte contains the CRC of the transmitted data. The value present in this phase can be sourced from either hardware or software based on the TX_OPER_CTRL_LO_ALT2_.SW_CRC bit. Refer to the module-specific section of the CRC Submodule for more information on the CRC is generated or used, for the transmitter and receiver modules respectively. The CRC byte is only present in data frames.
- **Frame Tag:** The frame tag contains the 4-bit user-defined frame tag. There are no restrictions on how this field is used in an application. The transmitter supplies this tag into the TX_FRAME_TAG_UDATA.FRAME_TAG bits for data frames. Ping frames use the tag defined in TX_PING_TAG.TAG. The receiver can access the received frame tag in RX_FRAME_TAG_UDATA.FRAME_TAG.
- **End of Frame:** The end of frame contains four clock edges with four bits, 0110, transmitted on the data lines.
- **Postamble:** The postamble contains four additional clock edges with the data lines held in the high state. After the postamble, the clock and data lines are driven high, their inactive state. This phase is not present in FSI-SPI compatibility mode.

7.5.9.3.4.2 Frame Types

The FSI hardware can generate and handle many predefined frame types. The different frame types can be used by the application to signal different types of events or convey different information to the receiver. The different frame types influence which phases and data fields to include in the transmitted frames.

Table 7-145 provides a short overview of the different frame types used by the FSI. Each frame type is described in more detail in the following subsections.

Table 7-145. Frame Types and Their 4-bit Codes

Frame Type	4-bit Frame Code	Description
PING	0000	This is the ping frame that can be sent either by software or automatically by hardware.
ERROR	1111	This must be used typically during error conditions or any condition where one side wants to signal the other side for attention. However, the user software can use this for any purpose.
DATA_1_WORD	0100	1 word data packet (16 bits of data)
DATA_2_WORD	0101	2 word data packet (32 bits of data)
DATA_4_WORD	0110	4 word data packet (64 bits of data)
DATA_6_WORD	0111	6 word data packet (96 bits of data)
DATA_N_WORD	0011	N(1-16) word data packet where software has programmed the number of the data words in a designated register. Both transmitter and receiver modules must have the same value programmed.
Reserved	0001, 0010, and 1000-1110	Reserved

7.5.9.3.4.2.1 Ping Frames

Ping frames are one of the most basic frames that can be generated by the FSI. Table 7-146 shows the structure of the ping frames.

Table 7-146. Ping Frame

Idle State	Preamble	SOF	Frame Type	Frame Tag	EOF	Postamble	Idle State
	1111	1001	0000	xxxx	0110	1111	

The ping frame type is always 0000. The frame tag is defined by the application. Separate frame tags exist for timer and software initiated ping frames. No data or CRC is transmitted in a ping frame.

The main purpose of the ping frame is to periodically send a notification to the receiver to make sure an active connection between the transmitter and receiver. The transmitter and receiver cores implement different features to allow the ping frame to operate as a line break detect feature.

On the transmitter, the ping frame is the only frame that can be set up and transmitted without any further software or DMA intervention. Ping frames can be transmitted by any (or all) of the three sources: automatic ping timer, software, or external triggers. See Section 7.5.9.3.2.3.3 for information on how the transmitter configures and sends the ping frames.

The receiver has a ping watchdog that can detect if a ping frame has not been received in a predetermined window. This allows the receiver to know if the connection between the receiver and the transmitter has been broken. See Section 7.5.9.3.3.4 for information on how the receiver handles ping frames.

7.5.9.3.4.2.2 Error Frames

Error frames are similar to ping frames in that there are no data fields transmitted. Despite the naming of this frame as an “error frame,” the usage of it is up to the application, as no restrictions are placed on how and when this type of frame is transmitted. Table 7-147 shows the structure of an error frame.

Table 7-147. Error Frame

Idle State	Preamble	SOF	Frame Type	Frame Tag	EOF	Postamble	Idle State
	1111	1001	1111	xxxx	0110	1111	

The structure of the error frame is the same as a ping frame. No data or CRC values are transmitted. The frame type is 1111 for all error frames, and the frame tag is defined by software in the TX_FRAME_TAG_UDATA register.

The receiver can detect if an error frame has been received based on the frame type field. Because of this, the receiver can read the incoming frame tag from the RX_FRAME_TAG_UDATA register and act on up to 16 different conditions.

7.5.9.3.4.2.3 Data Frames

Data frames are the most complex frames. As the name indicates, these frames are used to transfer data. Table 7-148 shows the general structure of data frames.

Table 7-148. Data Frame

Idle State	Preamble	SOF	Frame Type	User Data	Data Words	CRC Byte	Frame Tag	EOF	Postamble	Idle State
	1111	1001	0xxx	xxxx xxxx	1-16 words	xxxx xxxx	xxxx	0110	1111	

The frame type field reflects the 4-bit code of the frame type. A list of frame types can be seen in Table 7-145. The number of the data words transmitted is determined by the frame type chosen.

There are four fixed-length data frames supported by the frame type: 1 word, 2 words, 4 words, and 6 words.

Additionally, there is a user-defined data length frame type where the number of data words is fixed by software. Anywhere from 1 to 16 words can be transmitted in this frame type. This length must be configured in the N_WORDS field of the transmitter’s TX_FRAME_CTRL register and receiver’s RX_OPER_CTRL register.

7.5.9.3.4.3 Multi-Lane Transmission

The FSI is capable of transmitting and receiving data on two parallel data lines. When enabled, data bits are split between the data lines while the start of frame, frame type, frame tag, and end of frame fields are identical and complete on each line. The user data, data, and CRC fields are split between the data lines. Starting with the most-significant bit, the odd-numbered bits appear on D0 and even-numbered bits appear on D1.

In the following example, assume the following:

8-bit user data: u7u6u5u4u3u2u1u0

16-bit data: d15d14d13d12…d1d0

8-bit CRC: c7c6c5c4c3c2c1c0

Table 7-149. Multi-Lane Frame Format

Idle State	Preamble	SOF	Frame Type	User Data	Data Words	CRC Byte	Frame Tag	EOF	Postamble	Idle State
TXD0	1111	1001	0011	$u^7u^5u^3u^1$	$d^{15}d^{13}…d^1$	$c^7c^5c^3c^1$	xxxx	0110	1111	
TXD1	1111	1001	0011	$u^6u^4u^2u^0$	$d^{14}d^{12}…d^0$	$c^6c^4c^2c^0$	xxxx	0110	1111	

7.5.9.3.5 Flush Sequence

Every time there is a soft reset of the receiver, the receiver requires a flush sequence from the transmitter before the receiver can receive and decode frames. The receiver core has an asynchronous reset mechanism that allows the receive module to be reset even in the absence of the receive clocks. However, due to the design, this reset is released synchronous to the receive clock (RXCLK). Thus, the receiver requires five full clock pulses to be able to come out of reset. Sending the flush pattern makes sure that these clock edges are received and any subsequent frames sent to the receiver are correctly interpreted.

The flush sequence consists of a single toggle on both of the data lines as well as five consecutive pulses on the clock line.

If the FSI receiver is receiving data from a standard SPI, a data word of 0xFFFF from the SPI has the same effect as a flush sequence.

Figure 7-325 shows a sample plot of the flush sequence.

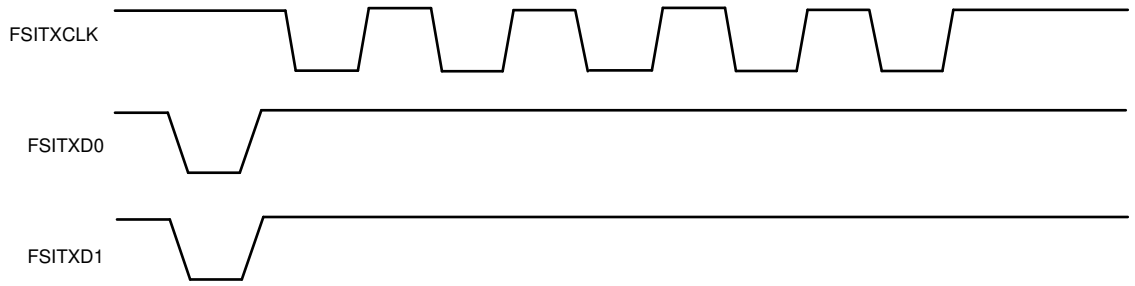


Figure 7-325. Flush Sequence Signals

7.5.9.3.6 Internal Loopback

The transmitter and receiver cores can be connected together internally to allow for development and debug. This is achieved by setting RX_MAIN_CTRL_ALTC_.INT_LOOPBACK to 1. Internal loopback routes the signals from the corresponding transmitter to the appropriate receiver pin. No configuration needs to be done in the transmitter.

Figure 7-326 shows the signal connections with internal loopback.

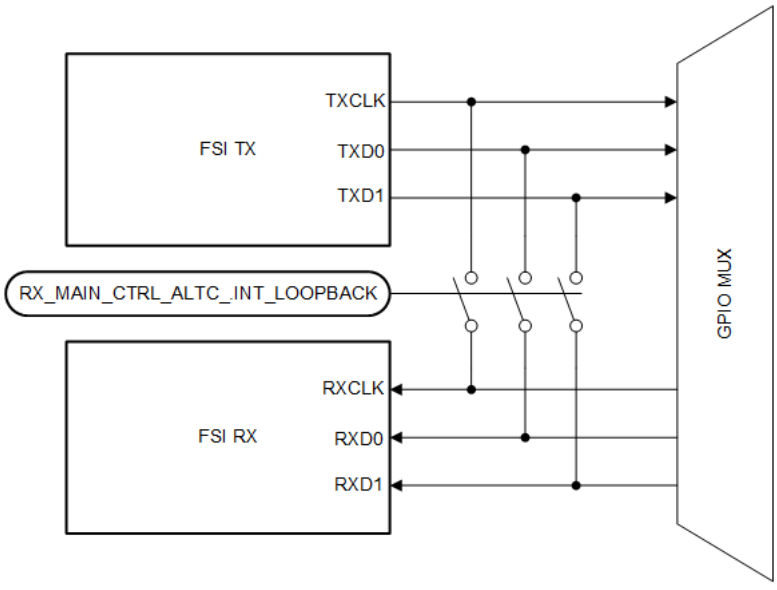


Figure 7-326. FSI with Internal Loopback

7.5.9.3.7 CRC Generation

The FSI uses CRC-8 with the polynomial 0x07 for the internal hardware CRC generation. This polynomial is also represented as x^8+x^2+x+1.

For example, for a 2-word data packet the following calculation occurs:

Data-1 = 0x4433

Data-0 = 0x2211

User Data = 0xAA

The CRC is computed with the bytes being taken in the following order (first to last):

0xAA – Byte 0, User Data

0x11 – Byte 1, Data-0, Least-significant byte

0x22 – Byte 2, Data-0, Most-significant byte

0x33 – Byte 3, Data-1, Least-significant byte

0x44 – Byte 4, Data-1, Most-significant byte

7.5.9.3.8 ECC Module

The FSI module comes with a 16-bit or 32-bit ECC computation module in both the transmitter and receiver. Use of this module is optional.

Note that the ECC is independent and unrelated to the hardware CRC computation module present in both the transmitter and receiver cores.

The following example shows a scenario in which the application requires ECC be calculated and transmitted on a 2-word data frame.

In the FSITX module:

1. Configure the ECC module for 32-bit data by setting TX_OPER_CTRL_HI_ALT1_.ECC_SEL to 1.
2. Write the data to the TX_ECC_DATA register as well as the transmit buffer.
3. Read TX_ECC_VAL Register. This register contains the 8-bit ECC value calculated on the data.
4. Copy the 8-bit data from TX_ECC_VAL to TX_FRAME_TAG_UDATA.USER_DATA.
5. Set the Start Condition to begin the transmission.

The reverse process is followed on the FSIRX module. Once the data frame is received, user software can do the following:

1. Copy the data from the receive buffer to the RX_ECC_DATA register.
2. Copy the received user data that contains the transmitted ECC value from RX_FRAME_TAG_UDATA.USER_DATA to the RX_ECC_VAL register.
3. Read the RX_ECC_LOG register. This contains the result of the ECC computation using the RX_ECC_DATA and RX_ECC_VAL registers.
   a. If no ECC errors were detected, RX_ECC_LOG is 0. The correct data is available in RX_ECC_SEC_DATA.
   b. If a single bit error was detected, RX_ECC_LOG.SBE is 1. The autocorrected data is available in RX_ECC_SEC_DATA.
   c. If multiple bit errors occurred, RX_ECC_LOG.MBE is 1. The data in RX_ECC_SEC_DATA is invalid and must not be used.

Using a 2-word data frame plus using the user data for the ECC is one possible implementation for ECC detection. Another option is to use a larger data frame a allocate one of the data words to be the ECC value.

7.5.9.3.9 FSI Trigger Generation

The RX_TRIGx external trigger can be used to initiate FSITX transmission. RX_TRIG0 must be used if TDM mode (multi-node configuration) is required. RX_TRIG0 must be used as the trigger source for start of transmission while the programmable stretch width RX_TRIG0 signal is used as the SEL_TDM_PATH signal (which decides whether the local FSITX is active or put in bypass mode).

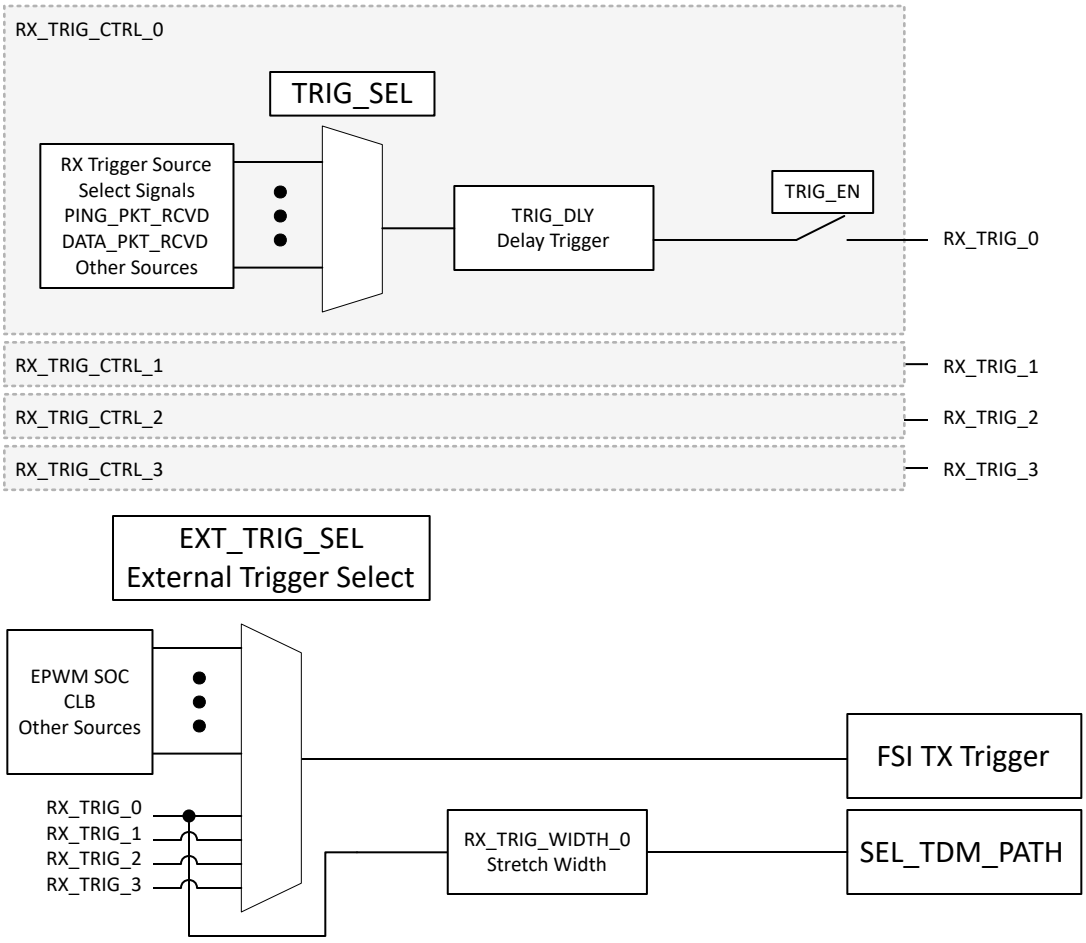


Figure 7-327. RX_TRIGx FSI Trigger

The signal source for the RX_TRIGx signal is selected through the RX_TRIG_CTRL_x.TRIG_SEL bits, as listed in Table 7-150.

Table 7-150. RX_TRIGx Trigger Select Signals

RX_TRIG_CTRLx.TRIG_SEL	Selected Signal
0	Ping Packet Received
1	Data Packet Received
2	Error Packet Received
3	Ping Frame Tag Match Occurred
4	Data Frame Tag Match Occurred
5	Error Frame Tag Match Occurred
6	Frame Done
7	Reserved
8 to 15	Reserved

The RX_TRIGx signals can optionally be delayed (this can be used in TDM scenarios) through the RX_TRIG_CTRL_x.TRIG_DLY.

7.5.9.3.10 FSI-SPI Compatibility Mode

The FSI supports a SPI compatibility mode. While the FSI can communicate with a standard SPI module, the FSI supports a limited configuration. The features of this compatibility mode are:

- Data transmits on rising edge and receive on falling edge of the clock.
- Only 16-bit word size is supported.
- TXD1 is driven like an active-low, chip-select signal. The signal is low for the duration for the full frame transmission.
- No receiver chip-select input is required. RXD1 is not used. Data is shifted into the receiver on every active clock edge.
- No preamble or postamble clocks are transmitted. All signals return to the IDLE state after the frame phase is finished.
- It is not possible to transmit in the SPI peripheral configuration because the FSI TXCLK cannot take an external clock source.

Table 7-151 lists the frame structure of the FSI-SPI compatibility mode. Each frame phase is present in this mode. If the FSI is transmitting to a standard SPI module, the SPI must decode the frame structure. Similarly, if the FSI is configured as a SPI peripheral, the standard SPI must encode the transmission to be sent.

Table 7-151. FSI-SPI Compatibility Frame Structure

Idle State	Start of Frame	Frame Type	User Data	Data Words	CRC byte(1)	Frame Tag	End of Frame	Idle State
	1001	4 bits	8 bits	1-16 words	8 bits	4 bits	0110	

(1) The CRC byte is present only in data frames.

Because of the requirement that the standard SPI module encodes the various frame data, this limits the type of modules that can be connected to the FSI in SPI mode. The paired SPI module must have enough functionality to encode and decode the frames.

If the FSI is transmitted to a standard 16-bit SPI, the data is arranged in the following manner. The example provided in Table 7-152 assumes a DATA_2_WORD frame has been sent.

Table 7-152. Contents of Data Received by a Standard SPI

SPI Data	Data Contents
SPI word 0	1001, 0100, 8-bit User Data
SPI word 1	Data word 1
SPI word 2	Data word 2
SPI word 3	8-bit CRC, 4-bit Frame Tag, 0110

7.5.9.3.10.1 Available SPI Modes

There are a few wiring schemes available for the FSI to use when communicating with an SPI module.

7.5.9.3.10.1.1 FSITX as SPI Controller, Transmit Only

The FSITX can operate as an independent SPI controller module. In this condition, TXCLK is connected to SPICLK, TXD0 is connected to SPIPICO, and TXD1 is connected to $\overline{\text{SPIPTE}}$, the chip select.

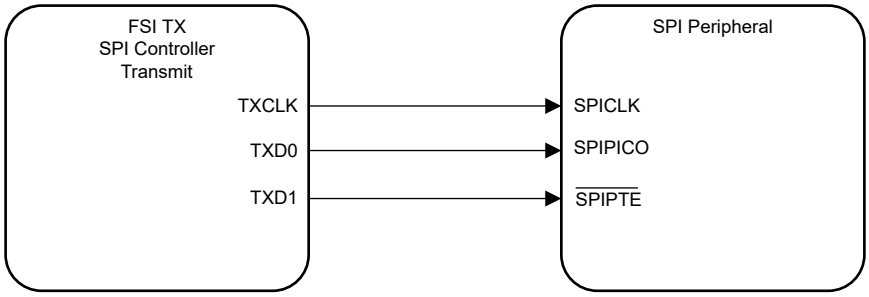


Figure 7-328. FSITX as SPI Controller, Transmit Only

When the FSI is an SPI transmitter, the application has the ability to check for frame errors, line breaks, CRC errors, and ECC checks on data. These are all encoded by hardware in every FSI frame. The SPI receiver requires some software to act upon this information.

Table 7-153. FSI as Controller Transmitter, SPI as Peripheral Receiver

Capability	Availability	Comment
Framing checks on the data frames	Yes	Can be implemented in software on the SPI receiver.
Ability to detect line breaks	Yes	Can be implemented in software on the SPI receiver but requires additional software overhead such as a timer or watchdog.
CRC check	Yes	Can be implemented in software on the SPI receiver. For devices that have , this is more efficient.
ECC on data	Yes	Can be implemented in software on the SPI receiver
Detection of abruptly terminated frames	No	
Double edge data rate	No	
Recovery from glitches on signal lines between frames	No	
Skew adjustment on signal lines	No	

7.5.9.3.10.1.1.1 Initialization

To configure the FSITX module to be an SPI controller for transmit only, proceed through the standard FSITX initialization procedure. Before releasing the FSITX from reset, set TX_OPER_CTRL_LO_ALT2_.SPI_MODE to 1. This enables the SPI clocking scheme and signaling structure.

7.5.9.3.10.1.1.2 Operation

The operation of the FSITX module in FSI-SPI Compatibility mode is the same as if the module is in standard FSI mode. The application can utilize the frame timer, ping frames, external frame triggers, and so on. Refer to Section 7.5.9.3.2 for more information on each of these features.

7.5.9.3.10.1.2 FSIRX as SPI Peripheral, Receive Only

The FSIRX can operate as an independent SPI peripheral module. In this usage, RXCLK is connected to SPICLK and RXD0 is connected to SPIPICO. RXD1 is unused. There is no requirement for a chip select signal to be used when connected to the FSIRX. This is because the FSIRX responds to any incoming clock edge. If there is any noise or unwanted clock transitions, a flush sequence is required to resynchronize the FSIRX module with the controller.

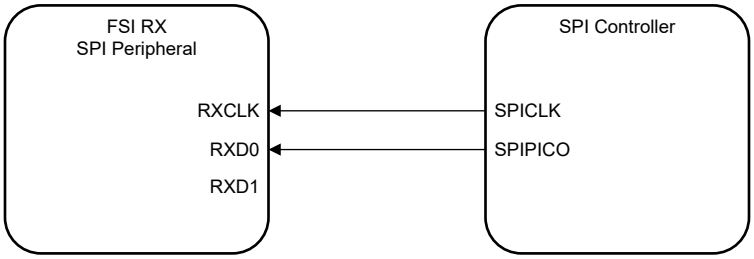


Figure 7-329. FSIRX as SPI Peripheral, Receive Only

When the FSI is an SPI receiver communicating with an SPI transmitter, the application has the ability to detect frame errors, line breaks, CRC errors, ECC checks on data, as well as abruptly terminated frames. Note that the FSI can handle all of this in hardware, but the SPI transmitter must encode the information into the data to be transmitted.

Table 7-154. SPI as Controller Transmitter, FSI as Peripheral Receiver

Capability	Availability	Comment
Framing checks on the data frames	Yes	Standard on FSI
Ability to detect line breaks	Yes	Can be implemented in software on the SPI transmitter but requires the use of a timer or watchdog in the transmitting SPI device.
CRC check	Yes	Can be implemented in software on the SPI transmitter.
ECC on data	Yes	Can be implemented in software on the SPI transmitter.
Detection of abruptly terminated frames	Yes	This is accomplished with the FSI setting up the frame watchdog counter.
Double edge data rate	No	
Recovery from glitches on signal lines between frames	Yes	Whenever glitches occur on either the clock or data lines in between transmissions, the initial flush pattern of a frame discards the effects of these glitches and causes the receiver to resynchronize when the real “start-of-frame” pattern is seen. So, the ability to reject glitches in between frames is very high.
Skew adjustment on signal lines	Yes	The FSI receiver has the ability to add delays to the incoming signal lines.

7.5.9.3.10.1.2.1 Initialization

To configure the FSIRX module to be an SPI peripheral for receiving only, proceed through the standard FSIRX initialization procedure. Before releasing the FSIRX from reset, set RX_OPER_CTRL.SPI_MODE to 1. This enables the SPI clocking scheme and signaling structure.

7.5.9.3.10.1.2.2 Operation

The operation of the FSIRX module in FSI-SPI compatibility mode is the same as if the module is in standard FSI mode. The application can utilize the Frame and Ping Watchdogs, CRC and ECC checks, and so on. Refer to Section 7.5.9.3.3 for more information on each of these features.

7.5.9.3.10.1.3 FSITX and FSIRX Emulating a Full Duplex SPI Controller

In this configuration, the FSITX is the controller clock. The FSITX module drives TXCLK (SPICLK), TXD0 (SPIPICO), and TXD1 (SPISTE/chip select) to the SPI peripheral. The SPIPOCI signal is connected back to the RXD0 signal. RXCLK can be applied either using the internal SPI pairing feature or externally wired, depending on the application requirements. Since the FSITX and RX modules are independent, the FSIRX can also be thought of as an additional SPI peripheral. Some software logic is required for the FSI to emulate an SPI controller fully.

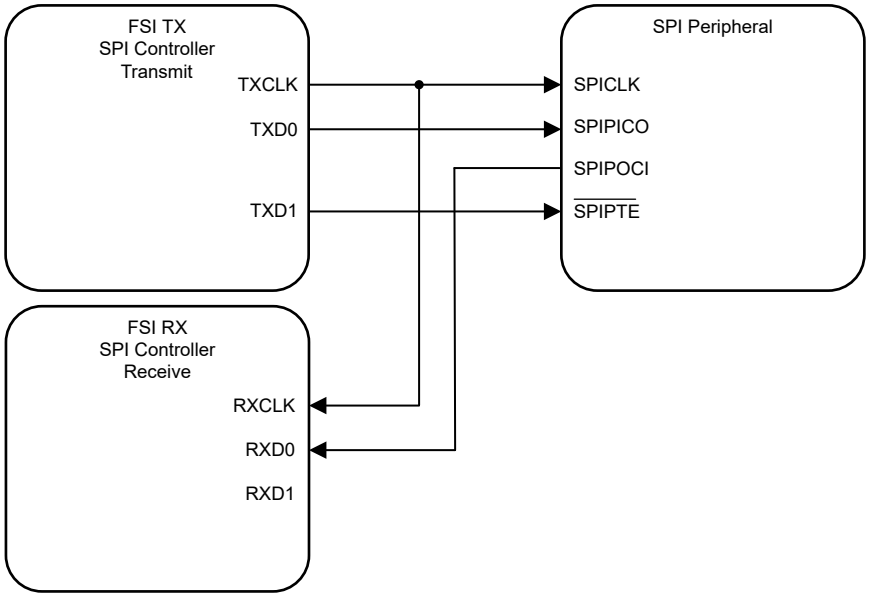


Figure 7-330. FSITX and FSIRX as SPI Controller, Full Duplex

7.5.9.3.10.1.3.1 Initialization

To configure both FSITX and RX modules for full duplex SPI controller operation, follow the initialization instructions for each module described in the preceding sections. Both FSITX and RX modules must set their respective SPI_MODE bits. This enables the SPI clocking scheme and signaling structures.

If internal clock loopback is desired, the FSIRX module must also set RX_MAIN_CTRL_ALTC_.SPI_PAIRING to 1. This internally connects TXCLK to RXCLK. If using internal clock loopback, the GPIO used for RXCLK can be reallocated to other application requirements.

If the application requires an external clock loopback, make sure that TXCLK is connected to RXCLK. This is required if the SPI peripheral is across an isolation barrier and there is latency between TXCLK being launched and SPIPOCI data being received on RXD0.

7.5.9.3.10.1.3.2 Operation

In this mode of operation, some higher level software must be written to emulate a full SPI controller module. There is no path for the transmit module to determine what the receive module received. Both the TX and RX

modules are still able to utilize the various other features available, such as the ping frame timer, ping frame and frame watchdogs, CRC and ECC error checkers, and so on. The procedure for configuring these features is described elsewhere in this document.

7.5.9.4 FSI Programing Guide

This section describes various operational sequences and features for the FSI.

7.5.9.4.1 Establishing the Communication Link

Once the transmitter and receiver modules have been configured, some synchronization must occur before the modules exchange data. Since the receiver accepts data on any clock transition, the receiver core logic must be flushed to properly interpret the start of a new, valid frame. This is especially true when the FSI modules reside on separate devices and are possibly isolated.

The following example provides a suggested approach for establishing a clean communication link on two separate devices that power up in an arbitrary order. Note that this is only a sample synchronization. Depending on application requirements, a different approach can be followed. The single, most important aspect of synchronization is to make sure that the receiver is properly flushed and ready to receive a complete frame without error. How to achieve this is up to the application.

Figure 7-331 shows the connection of the devices in this example. While there is no true concept of a main device or a remote device node in the FSI protocol, the example uses this nomenclature as a simple way to describe the data flow.

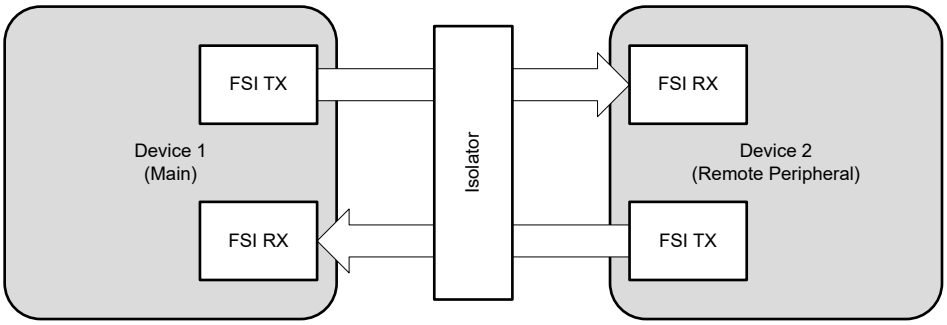


Figure 7-331. Point to Point Connection

Device 1 is the main node; it is the driver of the initialization sequence. Device 2 is the remote node; it responds to the main device commands. In this example, as well as in a real world use-case, neither the main device nor the remote device knows precisely when the other is ready to receive communication.

Sample sequences for both the main device and remote device are provided in the following subsections.

7.5.9.4.1.1 Establishing the Communication Link from the Main Device

The following sequence is an example of how the main device node establishes the communication link with the remote device without external signals outside of the standard communication link.

1. Assert the core reset to both the FSITX and FSIRX modules, and then deassert the resets.
2. Configure the transmitter and receiver for desired operation.
3. Set up the receiver interrupts to detect an incoming transmission.
4. Begin the ping loop:
   - Send the flush sequence.
   - Send a ping frame with the frame tag 0000.
   - Wait for some time. (determined by application)
   - If the FSIRX has received a valid ping frame, continue; else iterate the loop again.
   - If the received ping frame tag was 0001, continue; else iterate the loop again.
5. Send a ping frame with the frame tag 0001.

At this point, both the main transmit and receive channels have successfully received a frame from their remote counterparts. The link has been established and standard application communication can begin.

7.5.9.4.1.2 Establishing the Communication Link from the Remote Device

The following sequence is an example of how the remote device node establishes the communication link with the main device without external signals outside of the standard communication link.

1. Apply the core reset to both the FSITX and FSIRX modules, and then release the reset.
2. Configure the transmitter and receiver for desired operation.
3. Set up the receiver interrupts to detect an incoming transmission.
4. Wait for a receiver interrupt.
5. If the FSIRX has received a valid ping frame, continue; else return to step 4.
6. If the received frame tag was 0000, continue; else discard the transmission and return to step 4.
7. Send the flush sequence.
8. Send a ping frame with the frame tag 0001.
9. Wait for a receiver interrupt.
10. If the FSIRX has received a valid ping frame, continue; else return to step 4.
11. If the received ping frame tag was 0001, continue; else if the received frame tag was 0000, return to step 9. This can happen if a second ping frame was already in transit before receiving the remote device response in step 8.

At this point, both the transmit and receive modules have successfully received ping frames from their main counterparts. The link has been established and regular communication can now proceed. The application can configure periodic ping frames from the transmitter, initialize the receiver ping and frame watchdogs, and begin the communication required by the application.

7.5.9.4.2 Register Protection

Both the FSITX and FSIRX modules contain control registers that have embedded write protection. This is accomplished through register keys and a main register lock. These protections make sure that no spurious writes or unintentional modifications to these registers are accepted. .

Register Key Protection

bits in the FSI registers are protected by a key. To write to these bits, the key must be written at the same time. For example, to put the transmitter core into reset, TX_MAIN_CTRL.CORE_RST must be set. To do this, write 0xA501 to TX_MAIN_CTRL, where 0xA500 is the KEY value, and 0x0001 is the CORE_RST value. Refer to the *Registers* section for more information on which registers have write keys added.

Control Register Lock Protection

There also exists a main lock to prevent any modifications to the control registers. There is an independent lock for each FSI module. For the list of registers that are protected by this control register lock, refer to the *Registers* section. The control register lock prevents any writes to the control registers until the lock is released. To set the control register lock, write 0xA501 to RX_LOCK_CTRL and TX_LOCK_CTRL for the receiver and transmitter, respectively.

The control register lock cannot be disabled by the application until a SYSRSn has been asserted. This can occur at the device level, or by writing to the appropriate peripheral soft reset register (DEV_CFG_REGS.SOFTPRESx) for the FSI module.

7.5.9.4.3 Emulation Mode

There is no specific emulation mode or configuration supported. The FSI cores are always in free running mode. CPU halts do not have any effect on the operation of the FSI. Reads of registers and data buffers by the debugger do not affect any flags or status of the data buffers.

If you want to stop the operation of either FSI module when the debugger halts, the following steps are required:

1. Set the debugger to real-time emulation mode.
2. Mark the FSI interrupt group as a time-critical interrupt. That is, enable the corresponding bit in the DBGIER register.
3. The ISR can check the DSTAT register and to determine if the ISR was called when the debugger was halted.
4. FSI operations can be disabled and the ISR can branch to a debug-specific halt location.

7.5.10 Sigma Delta Filter Module (SDFM)

The sigma delta filter module (SDFM) is a four-channel digital filter designed specifically for current measurement and resolver position decoding in motor control applications. Each input channel can receive an independent delta-sigma (ΔΣ) modulator bit stream. The bit streams are processed by four individually-programmable digital decimation filters. The filter set includes a fast comparator (secondary filter) for immediate digital threshold comparisons for over-current and under-current monitoring, and zeros crossing detection.

7.5.10.1 Introduction

Figure 7-332 shows the SDFM CPU Interface.

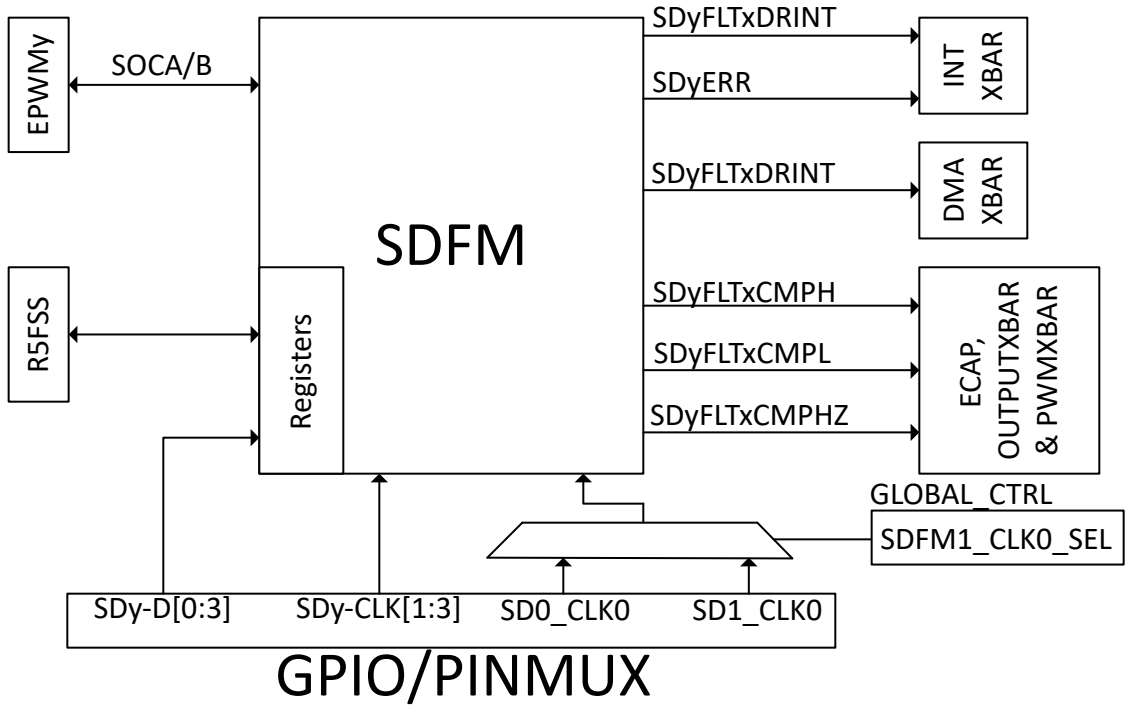


Figure 7-332. Sigma Delta Filter Module (SDFM) CPU Interface

Note

For this image and all images in *Sigma Delta Filter Module (SDFM)*, x represents 1-4 and y stands for SDFM1 and SDFM2.

Note

The SDFM Data signals are enumerated as SDFM_D[0:3] while the corresponding registers are enumerated as [1:4].

Note

SDFM1_CLK0_SEL signal is controlled by register in Global Control Register Space

7.5.10.1.1 Features

SDFM features include:

- Eight external pins per SDFM module
  - Four sigma-delta data input pins per SDFM module (SDFMy_Dx, where x = 0 to 3)
  - Four sigma-delta clock input pins per SDFM module (SDFMy_CLKx where x = 0 to 3)
- Modulator clock rate equals the modulator data rate
- Four independent, configurable secondary filter (comparator) units per SDFM module:
  - Four different filter type selection (Sinc1/Sinc2/SincFast/Sinc3) options available
  - Ability to detect over-value condition, under-value condition, and Threshold-crossing conditions
    1. Two independent Higher Threshold comparators (used to detect over-value condition)
    2. Two independent Lower Threshold comparators (used to detect under-value condition)
    3. One independent Threshold-Crossing comparator (used to measure duty cycle/frequency with eCAP)
  - OSR value for comparator filter unit (COSR) programmable from 1 to 32
- Four independent configurable primary filter (data filter) units per SDFM module:
  - Four different filter type selection (Sinc1/Sinc2/SincFast/Sinc3) options available
  - OSR value for data filter unit (DOSR) programmable from 1 to 256
  - Ability to enable or disable (or both) individual filter module
  - Ability to synchronize all four independent filters of an SDFM module by using the Main Filter Enable (MFE) bit or by using PWM signals
- Data filter output can be represented in either 16 bits or 32 bits.
- Data filter unit has a programmable mode FIFO to reduce interrupt overhead. The FIFO has the following features:
  - The primary filter (data filter) has a 16-deep x 32-bit FIFO.
  - The FIFO can interrupt the CPU after programmable number of data-ready events.
  - FIFO Wait-for-Sync feature: Ability to ignore data-ready events until the PWM synchronization signal (SDSYNC) is received. Once the SDSYNC event is received, the FIFO is populated on every data-ready event.
  - Data filter output can be represented in either 16 bits or 32 bits.
- PWMx.SOCA/SOCB can be configured to serve as SDSYNC source on a per-data-filter-channel basis.
- Configurable Input Qualification available for both SDFMy-CLKx and SDFMy-Dx
- Ability to use one filter channel clock (SDFMy-CLK0) to provide clock to other filter clock channels.
- Configurable digital filter available on comparator filter events to blankout comparator events caused by spurious noise

7.5.10.1.2 Block Diagram

Each SDFM module has four independent filter modules. These filter modules are identical and can be configured independently. Each individual filter module has the following units:

- Input control unit
- Primary filter (data filter) unit
- Secondary filter (comparator filter) unit with 4 independent comparators

Figure 7-333 shows the SDFM module block diagram. The SDFM port operation is configured and controlled by the registers listed in the *SDFM Registers* section of the Register Addendum.

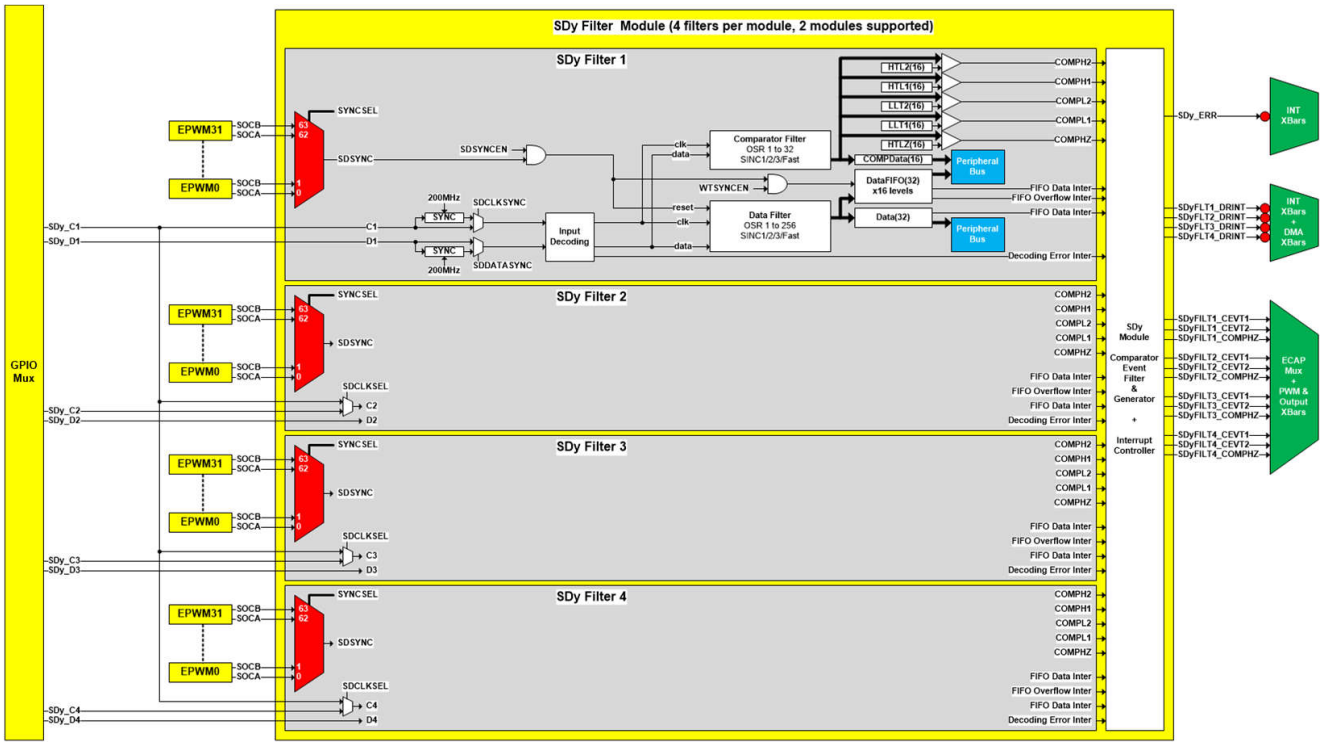


Figure 7-333. SDFM Integration Diagram

A. The Enumeration shown has each SDFM data and clock signal as [1:4]. The signal naming in the AM263Px datasheet for these signals is [0:3] and maps accordingly.

Each filter module shown in Figure 7-334 has a primary (data) filter and a secondary (comparator) filter pair that receives the same bit stream. Except for the input bit stream, both the primary and secondary filter are completely independent of each other. Each of these filter modules can be independently configured. So, in a SDFM module, there is a total of four primary filters and four secondary filters.

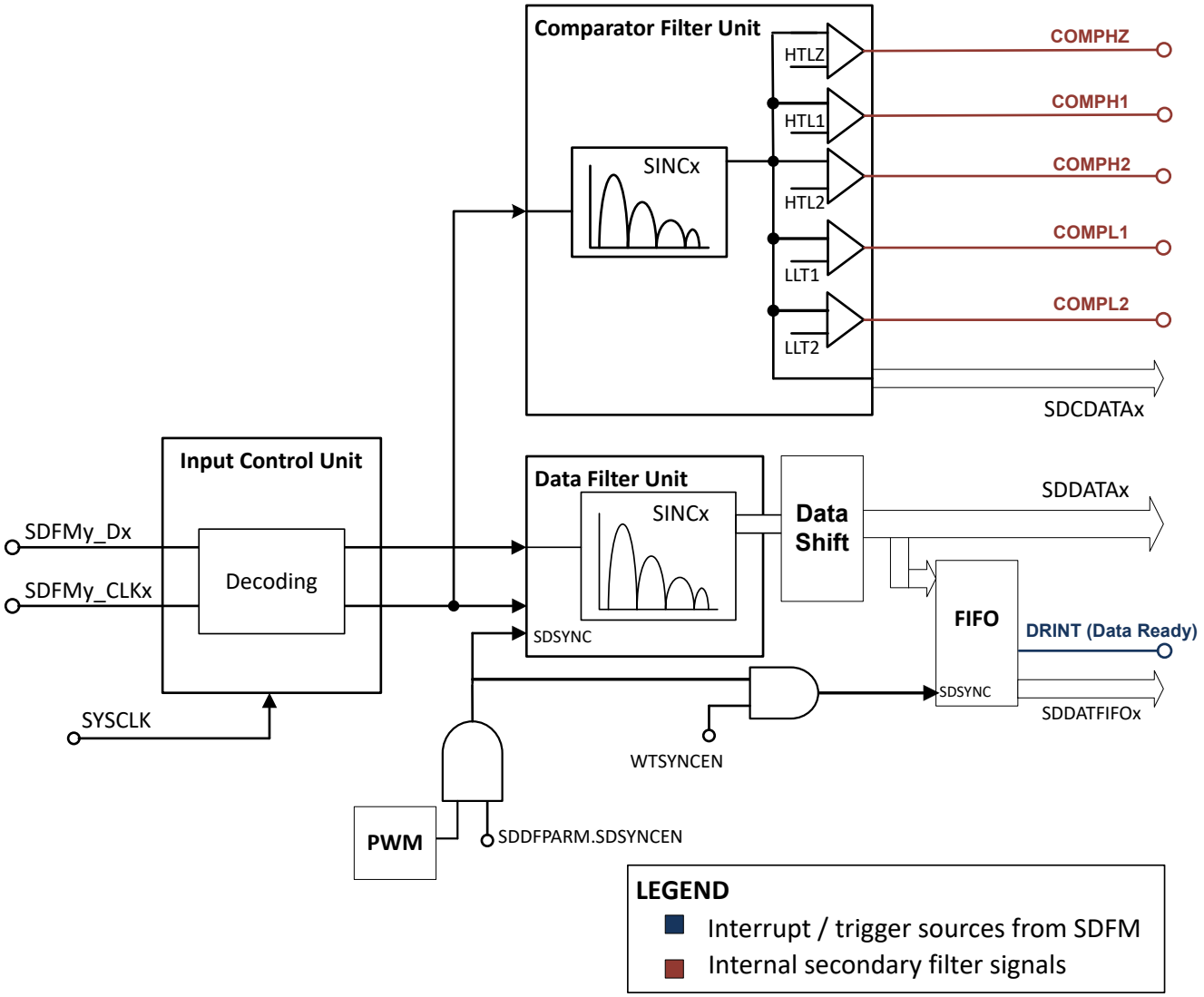


Figure 7-334. Block Diagram of One Filter Module

7.5.10.2 SDFM Integration

There are 4x SDFM modules integrated in the CONTROLSS. The diagrams below provides a visual representation of the device integration details.

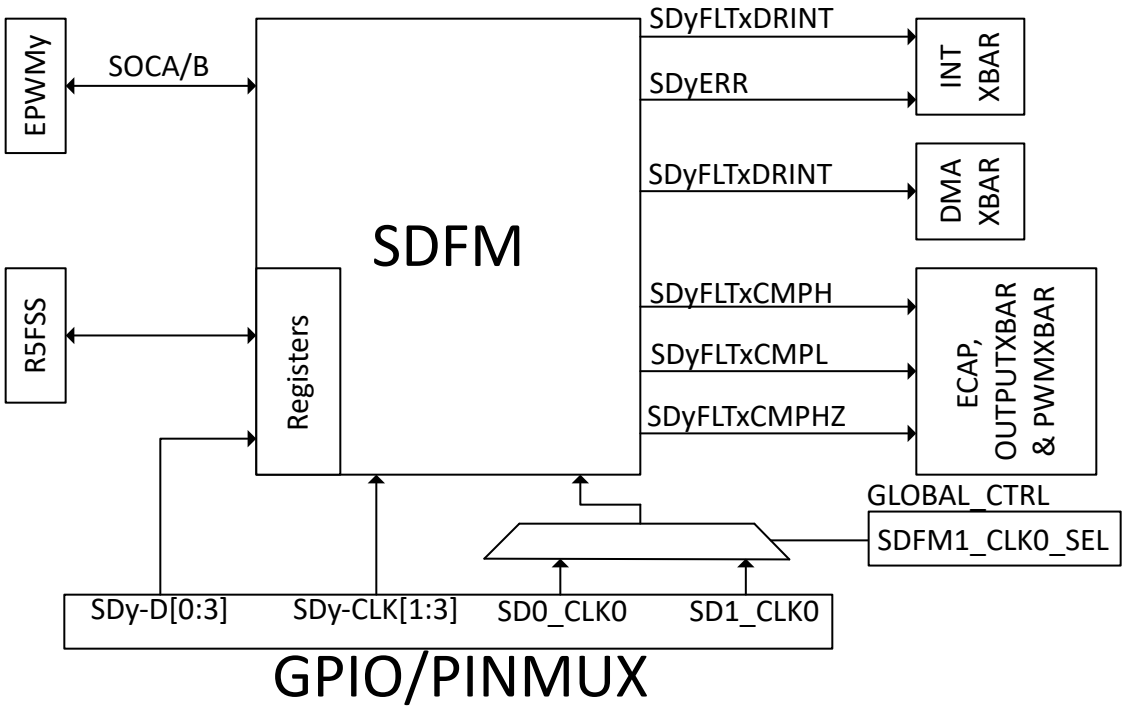


Figure 7-335. SDFM Integration Diagram (Simple)

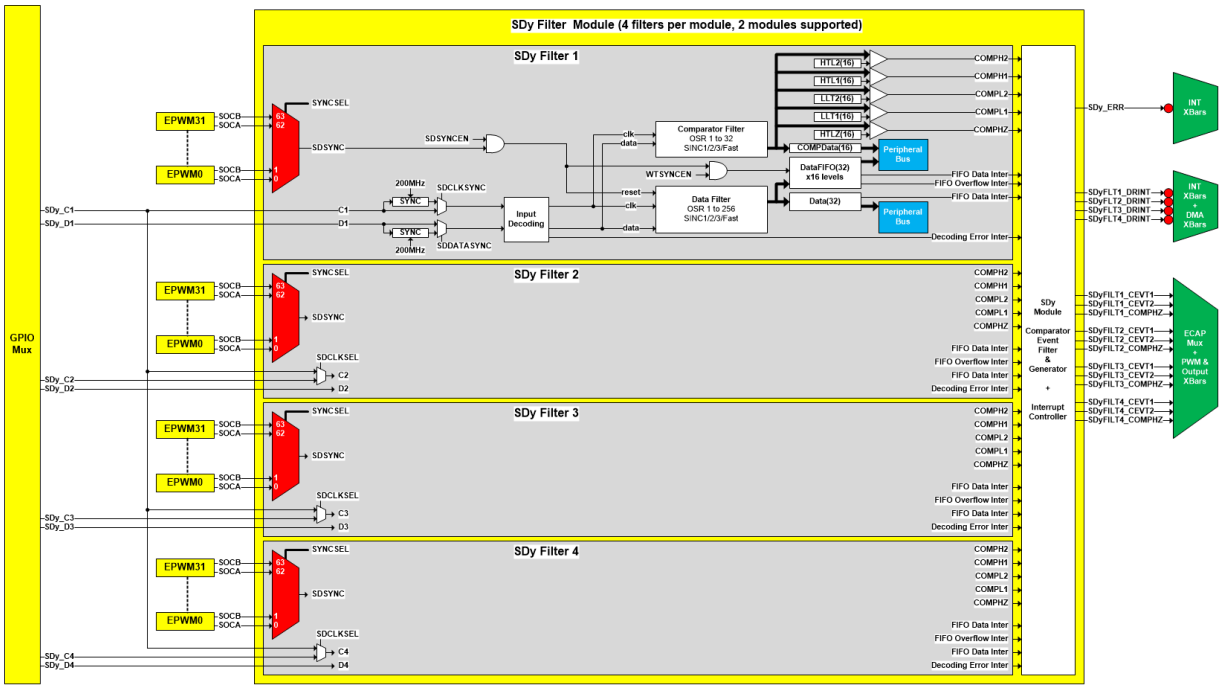


Figure 7-336. SDFM Integration Diagram (Detailed)

7.5.10.3 Configuring Device Pins

For proper SDFM operation, use the following GPIO input qualification. Other GPIO qualifications are not supported.

- GPIO Input qualification is ASYNC, make sure to check the SDFM Electrical Data and Timing (Using ASYNC) requirement is met and be aware of the following caution message. SDFM Input Qualification feature is used to provide protection against random noise glitches.

CAUTION

The SDFM clock inputs (SDFMy_CLKy pins) directly clock the SDFM module. Any glitches or ringing noise on these inputs can corrupt the SDFM module operation. Special precautions must be taken on these signals to make sure of a clean and noise-free signal that meets SDFM timing requirements. Precautions such as series termination for ringing due to any impedance mismatch of the clock driver and spacing of traces from other noisy signals are recommended.

Note

The SDFM module expects SDFMy-Dx to change on the falling edge of SDFMy_CLKx and strobes for SDFMy-Dx on the rising edge. But some SD-modulators in the market change SDFMy-Dx on the rising edge and expect SDFM to strobe for data on the falling edge. In such cases, the GPIO inversion feature (GPxINV) is used on SDFMy-CLKx pin to change polarity and make it compatible with the SDFM.

See the *General-Purpose Input/Output (GPIO)* chapter for more details on GPIO mux and settings.

7.5.10.4 Input Qualification

Impulse noise sources such as EMI, crosstalk, and so on, has the possibility of corrupting SDCLK and SDDATA bit streams, resulting in corrupted SDFM filtered data. This impulse noise effects can be mitigated when using the input qualification feature that synchronizes SDCLK/SDDATA signals with PLLCLK. By default, both SDCLK and SDDATA bit stream are not synchronized. SDCLK can be synchronized to PLLCLK by setting SDCTLPARMx.SDCLKSYNC = 1 and SDDATA can be synchronized to PLLCLK by setting SDCTLPARMx.SDDATASYNC = 1. Figure 7-337 shows optional Input Qualification option on SDCLK and SDDATA lines.

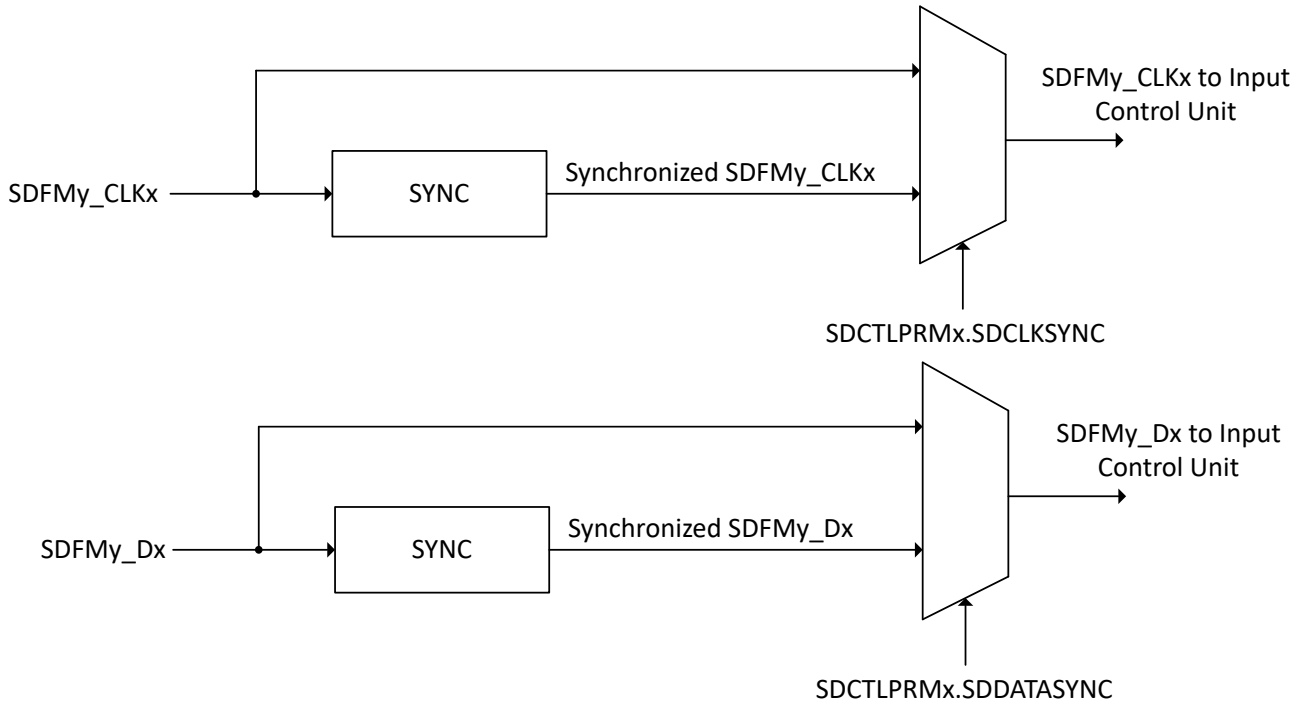


Figure 7-337. Input Qualification on SDFMy-CLKx and SDFMy-Dx

Note

SDFM PLL clock needs to be configured in case synchronizers inside SDFM are used.

7.5.10.5 Input Control Unit

The input control unit receives sigma delta modulated data and a sigma delta modulated clock. The modulated data received is captured and passed on to the data filter unit and comparator unit. This unit can be configured to receive the modulated data in only mode 0. Table 7-155 and Figure 7-338 show how SDCTLPARMx.MOD bits can be configured in mode 0.

Table 7-155. Modulator Clock Modes

Modulator Mode [MOD]	Description
0	The modulator clock is running with the modulator data rate. The modulator data is strobed at every rising edge of the modulator clock.
1	Reserved
2	Reserved
3	Reserved

Note

To achieve the maximum value, the sigma-delta modulator has to be operated at absolute maximum positive or negative full scale, which is outside of the recommended full scale range of 80% of most sigma-delta modulators.

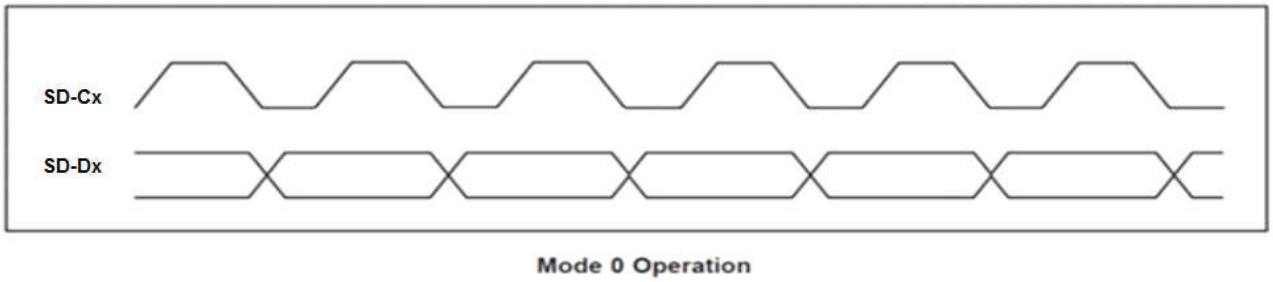


Figure 7-338. Different Modulator Modes Supported

7.5.10.6 SDFM Clock Control

In systems, the modulator clock can be generated using PWMs. Assuming all the SD-CLKs see the same delay on board traces, you can potentially use just one clock to clock multiple filters; thereby, saving on the number of pins used for SDFM. To enable this, Filter1 SDCLK (SDFM-CLK0) can possibly apply to other filter channels if required. The SDCTLPARAMx.SDCLKSEL register bit field can be configured to select filter channel SDCLK. It is also possible to drive SD0 FILTER0 clock from the pinmux to SD1 FILTER0 by configuring CONTROLSS_CTRL.SDFM1_CLK0_SEL. See Figure 7-339 to view this feature.

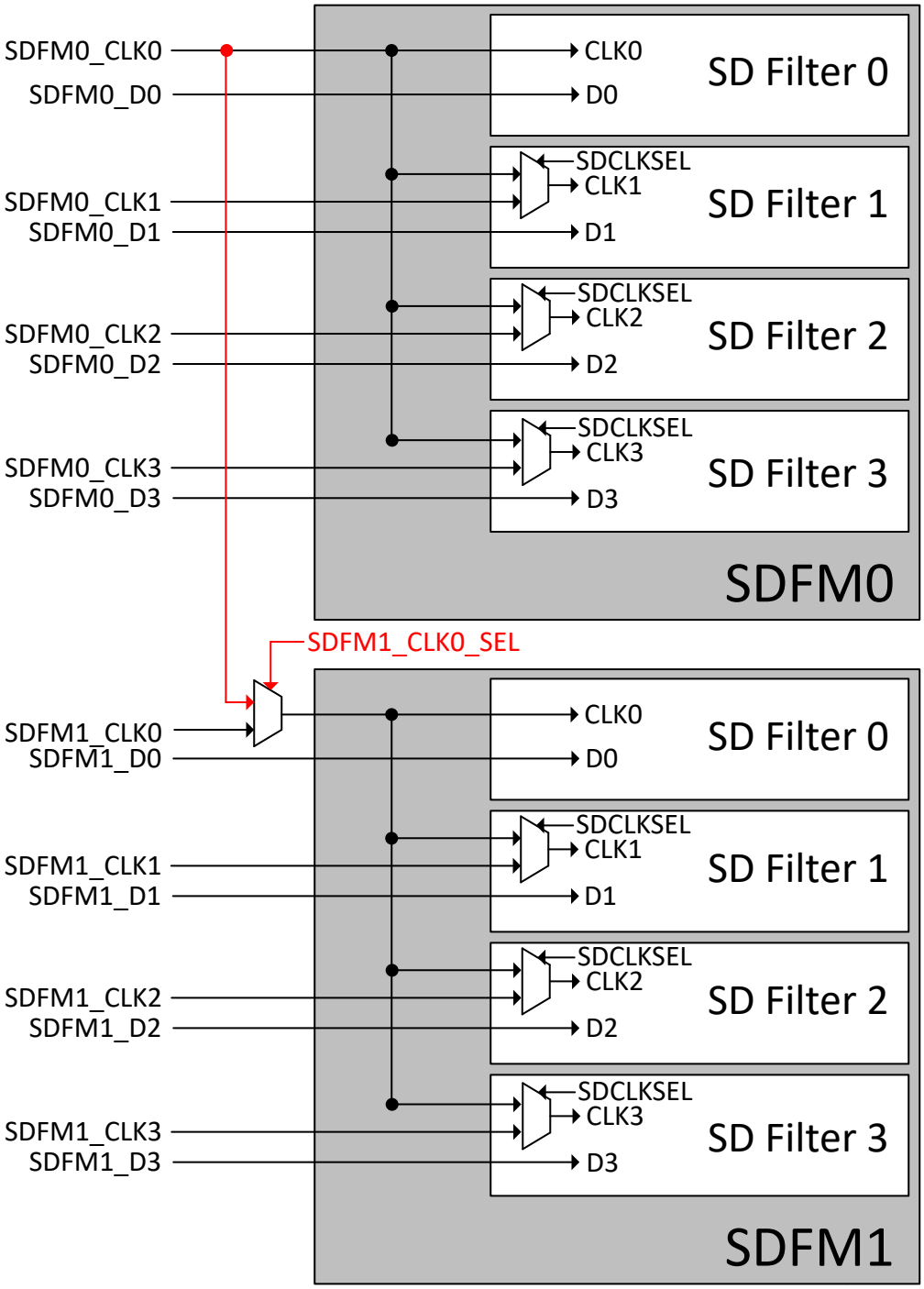


Figure 7-339. SDFM Clock Control

7.5.10.7 Sinc Filter

Both the comparator filter and data filter available in SDFM have the $Sinc^N$ filter as the core. The $Sinc^N$ filter is essentially a low-pass filter that converts the input bit stream into digital data by digital filtering and decimation. This filtered digital data represents analog input given to the sigma delta modulator. Simplified $Sinc^N$ architecture consists of cascaded integrators and differentiators separated by a down-sampler as shown in Figure 7-340. The Z-transfer function of the Sinc filter of order N is shown in Figure 7-341.

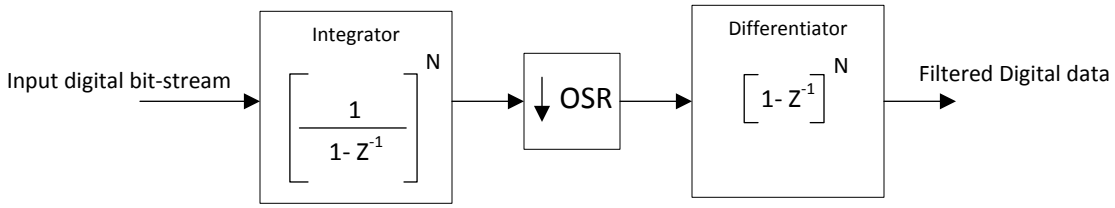


Figure 7-340. Simplified Sinc Filter Architecture

$$H(Z) = \left[\frac{1 - Z^{-OSR}}{1 - Z^{-1}} \right]^{N}$$

N = Order of Sinc filter
OSR = Over Sampling Ratio

Figure 7-341. Z-Transform of Sinc Filter of Order N

Effective resolution of the Sinc filter (ENOB) depends upon filter type, OSR and sigma-delta modulator frequency. Typically, higher resolution or ENOB can be achieved by higher OSR for a given filter type; however, the tradeoff is increased filter delay. It is important to choose the right sigma delta modulator by studying the optimal speed versus resolution tradeoff. Refer to the corresponding sigma delta modulator data sheet to determine the effective resolution for a given Sinc filter configuration. Figure 7-342 shows the frequency response of different filter structures when OSR = 32 and when the sigma delta modulator frequency is 10 MHz.

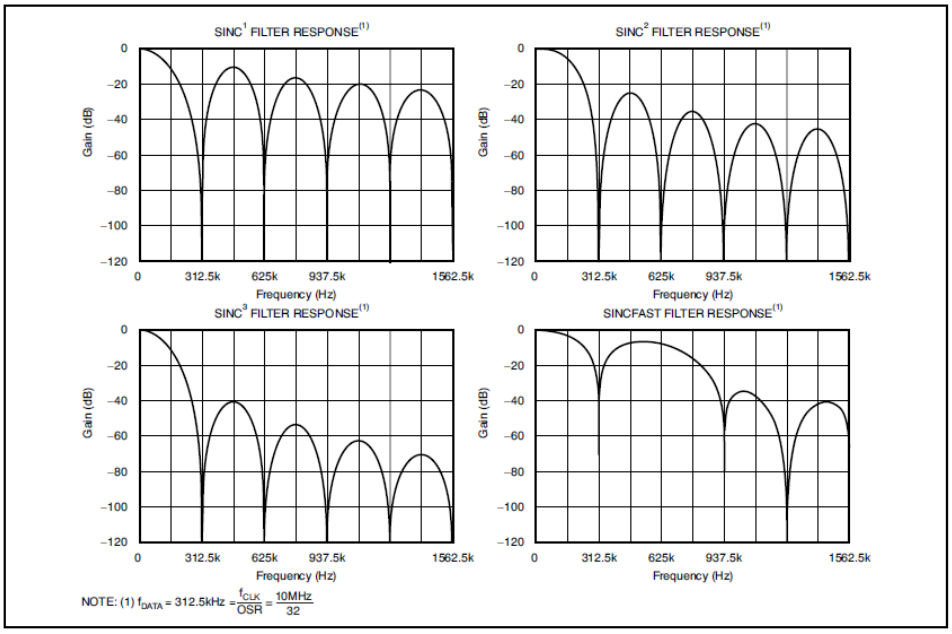


Figure 7-342. Frequency Response of Different Sinc Filters

The order of different sinc filter is shown in Table 7-156.

Table 7-156. Order of Sinc Filter

Filter Type	Order of Sinc Filter
Sinc1	1
Sinc2	2
Sinc3	3
SincFast	3

7.5.10.7.1 Data Rate and Latency of the Sinc Filter

The data rate of the sinc filter (filter throughput) represented in samples/sec is calculated by the following formula:

$$Data\ rate\ of\ Sinc\ filter = \frac{Modulator\ data\ rate}{OSR} \quad (16)$$

The latency of the sinc filter represented in secs is defined as the amount of time taken by a sinc filter type to deliver the correct filtered output upon initiation. For a given filter type, latency is calculated by the following formula:

$$Latency\ of\ Sinc\ filter = \frac{Order\ of\ Sinc\ filter}{Data\ rate\ of\ Sinc\ filter} \quad (17)$$

Example configuration:

Sinc filter type	= sinc3
Modulator data rate	= 10 MHz
OSR	= 256
Data rate of Sinc Filter	= 10 MHz / 256 = 39.1 K samples / sec
Sinc filter latency	= 76.8 µs
Sinc filter type	= sinc2
Modulator data rate	= 10 MHz
OSR	= 256
Data rate of Sinc Filter	= 10 MHz / 256 = 39.1 K samples / sec
Sinc filter latency	=51.2 µs

7.5.10.8 Data (Primary) Filter Unit

The data filter is a configurable Sinc filter which supports the following filter types: Sinc1, Sinc2, Sinc3, and SincFast. The data filter OSR (DOSR) settings can be configured from 1 to 256 and is independent of the comparator filter. Effective resolution of the data filter (ENOB) depends upon Data filter type, DOSR, and sigma-delta modulator frequency. By default, the data filter is disabled and setting of SDDFPARMx.FEN = 1 enables the data filter. The data filter output is represented in 26-bit signed integer in two's complement format. This filter unit translates a low input signal as '-1' and a high input signal as '1'. The resulting calculation gives both positive and negative values for the output of the data filter. Table 7-157 shows the different full scale values that the data filter can store using different OSRs.

See Section 7.5.10.7.1 to understand how to calculate data rate and latency of data filter.

Table 7-157. Peak Data Values for Different DOSR/Filter Combinations

DOSR	Sinc1	Sinc2	Sinc3	SincFast
x	x	x^2	x^3	$2x^2$
4	–4 to 4	–16 to 16	–64 to 64	–32 to 32
8	–8 to 8	–64 to 64	–512 to 512	–128 to 128
16	–16 to 16	–256 to 256	–4096 to 4096	–512 to 512
32	–32 to 32	–1024 to 1024	–32,768 to 32,768	–2048 to 2048
64	–64 to 64	–4096 to 4096	–262,144 to 262,144	–8192 to 8192
128	–128 to 128	–16,384 to 16,384	–2,097,152 to 2,097,152	–32,768 to 32,768
256	–256 to 256	–65,536 to 65,536	–16,777,216 to 16,777,216	–131,072 to 131,072

7.5.10.8.1 32-bit or 16-bit Data Filter Output Representation

The data filter output can be represented in either 32-bit or 16-bit format.

32-bit data filter representation:

- When SDDPARMx.DR = 1, data filter output is represented in 32-bit format. Writes to shift control bits do not have any bearing on the output of the data filter in this configuration.

16-bit data filter representation:

- By default, data filter output is represented in 16-bit format
- When SDDPARMx.DR = 0, data filter output is represented in 16-bit format. But it is the responsibility of the user to configure the corresponding shift control bits in the SDDPARMx register to control which 16-bit part of the 32-bit word is sent to the register map.

  For example, for the data filter configuration below:
  - Filter type = Sinc3
  - OSR = 128
  - SDDPARMx.DR = 0

  The data filter with a 26-bit signed output value can be in the range of –16,777,216 to 16,777,216. But, 16-bit signed output supports values only from –32,768 to 32,767. Therefore, it is required to configure shift control bits (SDDPARMx.SH) to 7 to represent the data filter output correctly in 16-bit format. Table 7-158 shows the configuration settings of shift control bits for different OSR and filter types.

Table 7-158. Shift Control Bit Configuration Settings

OSR	Sinc1	Sinc2	SincFast	Sinc3
1 to 31	0	0	0	0
32 to 40	0	0	0	1
41 to 50	0	0	0	2
51 to 63	0	0	0	3
64 to 80	0	0	0	4
81 to 101	0	0	0	5
102 to 127	0	0	0	6
128 to 161	0	0	1	7
162 to 181	0	0	1	8
182 to 203	0	1	2	8
204 to 255	0	1	2	9
256	0	2	3	10

CAUTION
Configuring shift control bits incorrectly results in getting an incorrect 16-bit data filter output.

7.5.10.8.2 Data FIFO

Each primary (data) filter channel has a 16-level deep, 32-bit FIFO.

FIFOs can be configured to collect a programmable number of data filter samples before issuing data-ready interrupt. This reduces the number of data-ready interrupts generated and resulting interrupt overhead for managed data flow.

By default, FIFO operation is disabled. FIFOs can be enabled by setting SDFIFOCTLx.FFEN = 1. When FIFO is enabled, each data-ready event from the data filter populates the FIFO, and the status of the FIFO at any given time is updated in the SDFIFOCTLx.SDFFST bit field.

Setting up FIFO to interrupt after receiving programmable number of data ready events:

- Enable SDFM FIFO (Set SDFIFOCTLx.FFEN = 1)
- Enable SDFM FIFO interrupt (Set SDFIFOCTLx.FFIEN = 1)
- Configure SDFIFOCTLx.SDFFIL bit field to any value between 0 to 16
- Configure SDFM data ready event to interrupt on FIFO interrupt (SDFFINT) (Set SDFFINTx = 1)
- Select data-ready interrupt source is SDFFINTx (DRINTx = SDFFINTx) (SDFIFOCTLx.DRINTSEL = 1)

When the SDFIFOCTLx.SDFFST >= SDFIFOCTLx.SDFFIL condition is met, the SDIFLG.SDFFINTx bit is set and an interrupt is generated on the DRINTx. SDIFLG.SDFFINTx flag can be cleared by setting the SDIFLGCLR. SDFFINTx bit field.

Wait for Sync feature:

The FIFO wait for sync feature can be used to ignore data-ready events from the data filter until the SDSYNC (from PWM) event is triggered.

By default, the Wait for Sync feature is disabled. This feature can be enabled by setting SDSYNCx.WTSYNCEN = 1

When the wait for sync feature is disabled:

FIFOs get populated on every data ready event until the FIFO gets full (or) when SDFIFOCTLx.SDFFST >= SDFIFOCTLx.SDFFIL.

When the wait for sync feature enabled:

FIFOs do not get populated on every data ready event until the FIFO receives a SDSYNC event. On a SYSYNC event, the FIFO sets SDSYNCx.WTSYNFLG = 1 and data ready events from the primary filter start populating the FIFO until either the FIFOs get full or when SDFIFOCTLx.SDFFST >= SDFIFOCTLx.SDFFIL. WTSYNFLG can be cleared either automatically or manually.

When WTSYNFLG = 0, FIFOs contents are frozen and subsequent data ready events do not populate FIFO until next SDSYNC event.

WTSYNFLG automatic clear mode:

By default, this mode is enabled. When SDSYNCx.WTSCLREN = 1, WTSYNFLG is automatically cleared on SDFFINT event.

WTSYNFLG manual clear mode:

Setting SDSYNCx.WTSYNCLR = 1 can be used to clear WTSYNFLG manually.

Clearing FIFO contents:

FIFO contents can cleared by any of the following methods:-

- Disabling FIFO clear FIFO contents. This can be done by clearing SDFIFOCTLx.FFEN = 0.
- Disabling Primary filter clear FIFO contents. This can done by either clearing SDDFPARMx.FEN = 0 (or) by clearing SDMFILEN.MFE = 0.
- FIFO contents can also be automatically cleared upon receiving the SDSYNC event. By default, this feature is disabled and this feature can be enabled by setting FIFO Clear-on-SDSYNC enable (SDSYNCx.FFSYNCCLREN = 1).

  Note: The above feature is only enabled when wait for sync feature is enabled (SDSYNCx. WTSYNCEN = 1).

FIFO debug access behavior:

Debug access of the SDDATFIFOx registers does not affect the FIFO pointers. On a CPU/RTDMA access to the SDDATFIFOx register, the FIFO read pointers advance to the next available entry in the FIFO.

7.5.10.8.3 SDSYNC Event

Primary (data) filters can be synchronized with respect to the PWM event (called SDSYNC event). The SDSYNC signal from the PWM module is used to reset the DOSR counter. This feature is by default disabled and can be enabled by setting SDDFPARMx.SDSYNCEN = 1. Each primary filter can be synchronized from any of the available PWMx SOCA/SOCB signals. The SDSYNCx.SDSYNCSEL bits allow the user to configure which PWM signal provides the SDSYNC pulse to the primary filter. Figure 7-343 shows how device PWM signals are connected to the SDFM modules.

Table 7-159. SDSYNCx.SYNCSEL

SYNCSEL[5:0]	SDSYNCx.SYNCSEL[5:0]			
	1	2	3	4
0	PWM0.SOCA	PWM0.SOCA	PWM0.SOCA	PWM0.SOCA
1	PWM0.SOCB	PWM0.SOCB	PWM0.SOCB	PWM0.SOCB
2	PWM1.SOCA	PWM1.SOCA	PWM1.SOCA	PWM1.SOCA
3	PWM1.SOCB	PWM1.SOCB	PWM1.SOCB	PWM1.SOCB
4	PWM2.SOCA	PWM2.SOCA	PWM2.SOCA	PWM2.SOCA
5	PWM2.SOCB	PWM2.SOCB	PWM2.SOCB	PWM2.SOCB
6	PWM3.SOCA	PWM3.SOCA	PWM3.SOCA	PWM3.SOCA
7	PWM3.SOCB	PWM3.SOCB	PWM3.SOCB	PWM3.SOCB
8	PWM4.SOCA	PWM4.SOCA	PWM4.SOCA	PWM4.SOCA
9	PWM4.SOCB	PWM4.SOCB	PWM4.SOCB	PWM4.SOCB
10	PWM5.SOCA	PWM5.SOCA	PWM5.SOCA	PWM5.SOCA
11	PWM5.SOCB	PWM5.SOCB	PWM5.SOCB	PWM5.SOCB
12	PWM6.SOCA	PWM6.SOCA	PWM6.SOCA	PWM6.SOCA
13	PWM6.SOCB	PWM6.SOCB	PWM6.SOCB	PWM6.SOCB
14	PWM7.SOCA	PWM7.SOCA	PWM7.SOCA	PWM7.SOCA
15	PWM7.SOCB	PWM7.SOCB	PWM7.SOCB	PWM7.SOCB
16	PWM8.SOCA	PWM8.SOCA	PWM8.SOCA	PWM8.SOCA
17	PWM8.SOCB	PWM8.SOCB	PWM8.SOCB	PWM8.SOCB
18	PWM9.SOCA	PWM9.SOCA	PWM9.SOCA	PWM9.SOCA
19	PWM9.SOCB	PWM9.SOCB	PWM9.SOCB	PWM9.SOCB
20	PWM10.SOCA	PWM10.SOCA	PWM10.SOCA	PWM10.SOCA
21	PWM10.SOCB	PWM10.SOCB	PWM10.SOCB	PWM10.SOCB
22	PWM11.SOCA	PWM11.SOCA	PWM11.SOCA	PWM11.SOCA
23	PWM11.SOCB	PWM11.SOCB	PWM11.SOCB	PWM11.SOCB
24	PWM12.SOCA	PWM12.SOCA	PWM12.SOCA	PWM12.SOCA
25	PWM12.SOCB	PWM12.SOCB	PWM12.SOCB	PWM12.SOCB
26	PWM13.SOCA	PWM13.SOCA	PWM13.SOCA	PWM13.SOCA
27	PWM13.SOCB	PWM13.SOCB	PWM13.SOCB	PWM13.SOCB
28	PWM14.SOCA	PWM14.SOCA	PWM14.SOCA	PWM14.SOCA
29	PWM14.SOCB	PWM14.SOCB	PWM14.SOCB	PWM14.SOCB
30	PWM15.SOCA	PWM15.SOCA	PWM15.SOCA	PWM15.SOCA
31	PWM15.SOCB	PWM15.SOCB	PWM15.SOCB	PWM15.SOCB
32	PWM16.SOCA	PWM16.SOCA	PWM16.SOCA	PWM16.SOCA
33	PWM16.SOCB	PWM16.SOCB	PWM16.SOCB	PWM16.SOCB
34	PWM17.SOCA	PWM17.SOCA	PWM17.SOCA	PWM17.SOCA
35	PWM17.SOCB	PWM17.SOCB	PWM17.SOCB	PWM17.SOCB
36	PWM18.SOCA	PWM18.SOCA	PWM18.SOCA	PWM18.SOCA
37	PWM18.SOCB	PWM18.SOCB	PWM18.SOCB	PWM18.SOCB
38	PWM19.SOCA	PWM19.SOCA	PWM19.SOCA	PWM19.SOCA

Table 7-159. SDSYNCx.SYNCSEL (continued)

SYNCSEL[5:0]	SDSYNCx.SYNCSEL[5:0]			
	1	2	3	4
39	PWM19.SOCB	PWM19.SOCB	PWM19.SOCB	PWM19.SOCB
40	PWM20.SOCA	PWM20.SOCA	PWM20.SOCA	PWM20.SOCA
41	PWM20.SOCB	PWM20.SOCB	PWM20.SOCB	PWM20.SOCB
42	PWM21.SOCA	PWM21.SOCA	PWM21.SOCA	PWM21.SOCA
43	PWM21.SOCB	PWM21.SOCB	PWM21.SOCB	PWM21.SOCB
44	PWM22.SOCA	PWM22.SOCA	PWM22.SOCA	PWM22.SOCA
45	PWM22.SOCB	PWM22.SOCB	PWM22.SOCB	PWM22.SOCB
46	PWM23.SOCA	PWM23.SOCA	PWM23.SOCA	PWM23.SOCA
47	PWM23.SOCB	PWM23.SOCB	PWM23.SOCB	PWM23.SOCB
48	PWM24.SOCA	PWM24.SOCA	PWM24.SOCA	PWM24.SOCA
49	PWM24.SOCB	PWM24.SOCB	PWM24.SOCB	PWM24.SOCB
50	PWM25.SOCA	PWM25.SOCA	PWM25.SOCA	PWM25.SOCA
51	PWM25.SOCB	PWM25.SOCB	PWM25.SOCB	PWM25.SOCB
52	PWM26.SOCA	PWM26.SOCA	PWM26.SOCA	PWM26.SOCA
53	PWM26.SOCB	PWM26.SOCB	PWM26.SOCB	PWM26.SOCB
54	PWM27.SOCA	PWM27.SOCA	PWM27.SOCA	PWM27.SOCA
55	PWM27.SOCB	PWM27.SOCB	PWM27.SOCB	PWM27.SOCB
56	PWM28.SOCA	PWM28.SOCA	PWM28.SOCA	PWM28.SOCA
57	PWM28.SOCB	PWM28.SOCB	PWM28.SOCB	PWM28.SOCB
58	PWM29.SOCA	PWM29.SOCA	PWM29.SOCA	PWM29.SOCA
59	PWM29.SOCB	PWM29.SOCB	PWM29.SOCB	PWM29.SOCB
60	PWM30.SOCA	PWM30.SOCA	PWM30.SOCA	PWM30.SOCA
61	PWM30.SOCB	PWM30.SOCB	PWM30.SOCB	PWM30.SOCB
62	PWM31.SOCA	PWM31.SOCA	PWM31.SOCA	PWM31.SOCA
63	PWM31.SOCB	PWM31.SOCB	PWM31.SOCB	PWM31.SOCB

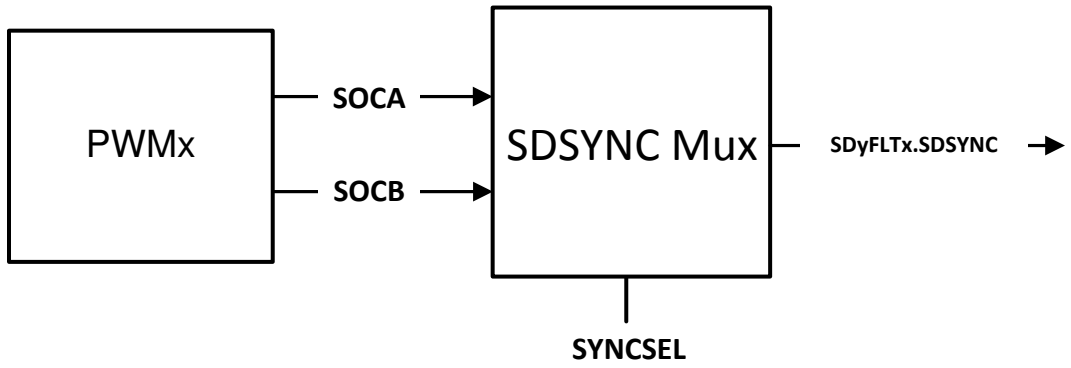


Figure 7-343. SDSYNC Event

Note

Make sure that only one SDSYNC event is generated per PWM timer period. Using PWM in up-count or down-count mode can automatically make sure that only SDSYNC events are generated. But, if up-down count mode is used, then make sure that only one SDSYNC event per PWM cycle is generated; otherwise, the filter synchronizer corrupts SDFM timing by providing two pulses per PWM cycle.

Because of the inherent architecture of the Sinc filter (Sinc1, Sinc2, Sinc3, SincFast), the first few samples, depending upon filter type, are incorrect. Table 7-160 shows the number of incorrect samples on the following conditions:

- When Sinc filter is enabled and configured for first time.
- When Sinc filter is disabled and re-enabled or reconfigured in the middle of operation.
- When data filter receives SDSYNC event from PWM.

Table 7-160. Number of Incorrect Samples Tabulated

Filter Type	Number of Incorrect Samples After the Filter is Enabled and Configured
Sinc1	No incorrect sample.
Sinc2	The first sample of the Sinc2 filter is incorrect.
SincFast	The first two samples of the SincFast filter are incorrect.
Sinc3	The first two samples of the Sinc3 filter are incorrect.

CAUTION

SDFM comparator interrupts can be enabled only after providing sufficient settling time to make sure the comparator filter does not trip on these incorrect samples. Therefore, SDFM comparator interrupts (CMPxH and CMPxL) can be enabled only after a sufficient delay is provided after the comparator filter is configured. This sufficient delay is calculated by adding the latency of the comparator filter and 5 SDFM-CLKx clock cycles.

7.5.10.9 Comparator (Secondary) Filter Unit

Most control systems require protection of the system by tripping the PWM in case the current or voltage goes out of bounds. The primary purpose of the secondary (comparator) filter is to allow the user to monitor input conditions with a fast settling time. This allows the user to trip PWMs to protect the system from potential damage.

Note

The secondary (comparator) filter cannot be synchronized with respect to the PWM event (SDSYNC event).

The comparator filter is a configurable Sinc filter that supports the following filter types: Sinc1, Sinc2, Sinc3, and SincFast. The comparator OSR (COSR) settings can be configured from 1 to 32 and is independent of the data filter. Effective resolution of the comparator filter (ENOB) depends upon the comparator filter type, COSR, and sigma-delta modulator frequency. By default, the comparator filter is disabled and setting SDCPARMx.CEN = 1 enables the comparator filter. The comparator filter output is represented in 16-bit unsigned format. This filter unit translates a low input signal as 0 and a high input signal as 1. The resulting calculations give only positive values for the output of the comparator filter. Table 7-161 shows the different full-scale values that the comparator filter can store using different OSRs.

Table 7-161. Peak Data Values for Different OSR/Filter Combinations

OSR	Sinc1	Sinc2	Sinc3	SincFast
x	0 to x	0 to x2	0 to x3	0 to 2x2
4	0 to 4	0 to 16	0 to 64	0 to 32
8	0 to 8	0 to 64	0 to 512	0 to 128
16	0 to 16	0 to 256	0 to 4096	0 to 512
32	0 to 32	0 to 1024	0 to 32,768	0 to 2048

See Section 7.5.10.7.1 to understand how to calculate data rate and latency of comparator filter.

The output of the comparator filter is memory-mapped and can be read in the SDCDATAx register. This register, SDCDATAx, is updated every COSR number of SDFM-CLKx cycles. The comparator filter digital output is connected to digital comparators explained below.

Note

The enumeration between the SDFM Data and Clock signals is described as [0:3] in the device datasheet but the Register Addendum maps the [0:3] Data and Clock signals to [1:4].

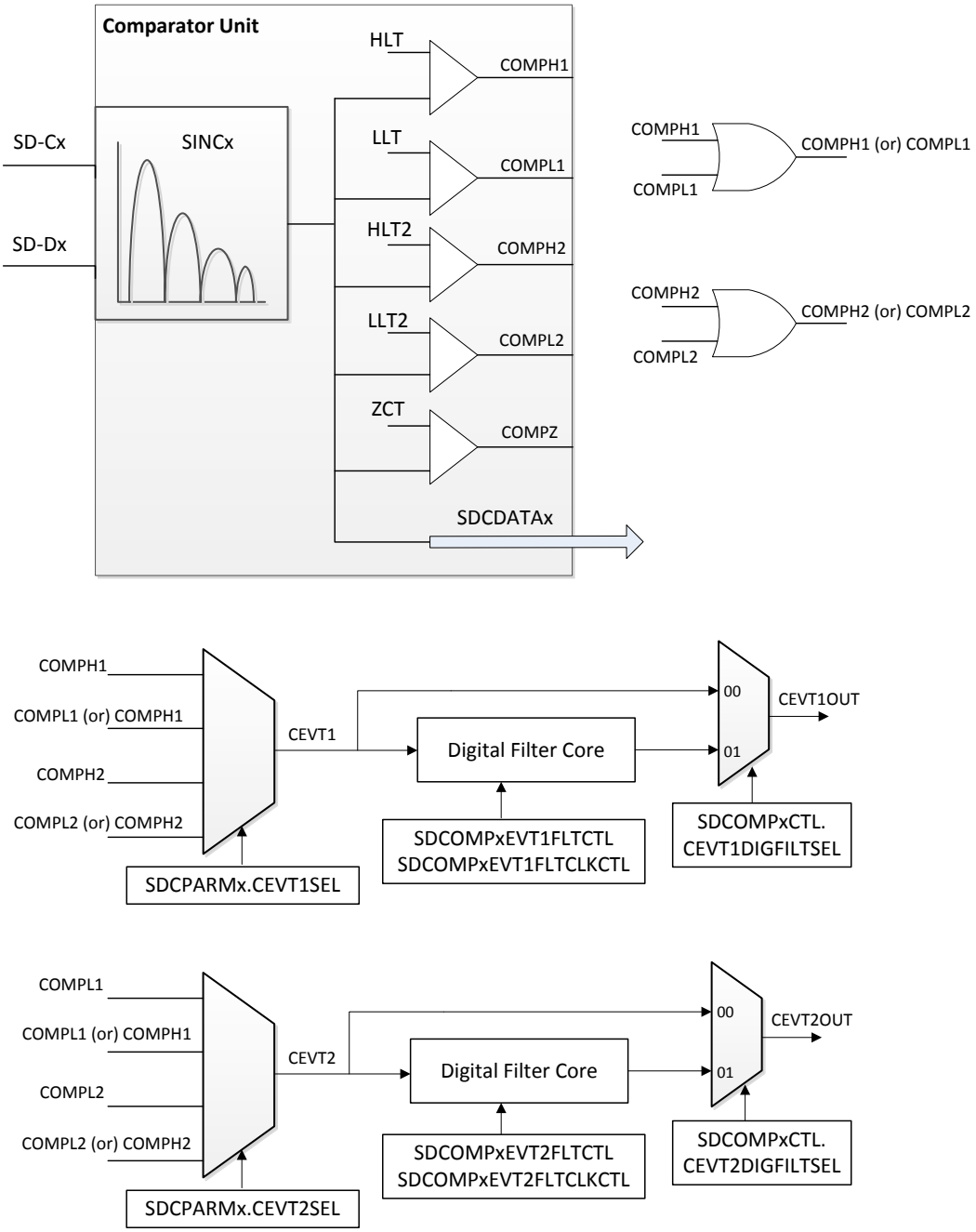


Figure 7-344. Comparator Unit Structure

7.5.10.9.1 Higher Threshold (HLT) Comparators

- High threshold comparator can be used to detect over-value condition.
- When comparator data > = higher threshold register, a high threshold event is generated.
- Higher threshold comparator events except for COMPHZx can be configured to trigger following events: CPU interrupt, PWM trip.
- This device has three high threshold comparators:
  - **Higher Threshold 1 (HLT1) Comparator**:
    - When comparator data > = (SDFLTxCMPH1.HLT), HLT1 comparator generates COMPH1 event.
    - The COMPH1 event is connected to both CEVT1 and CEVT2.
  - **Higher Threshold 2 (HLT2) Comparator**:
    - When comparator data > = (SDFLTxCMPH2.HLT), HLT2 comparator generates COMPH2 event.
    - The COMPH2 event is connected to both CEVT1 and CEVT2.
  - **Higher Threshold (HTLZ) Comparator**:
    - When comparator data > = (SDFLT1CMPHZ.CMPHZ), it can generate a Higher Threshold (B) event (COMPHZx) and sets the corresponding SDSTATUS.HZx flag. But, this event cannot be configured to generate SDFM interrupt (SDx_ERR).

7.5.10.9.2 Lower Threshold (LLT) Comparators

- The low threshold comparator can be used to detect under-value condition.
- When comparator data < = Lower Threshold register, a low threshold event is generated.
- Lower threshold comparator events can be configured to trigger following events: CPU interrupt, PWM trip.
- Lower threshold comparator events can be used in conjunction with ECAP to measure the frequency / duty cycle of Threshold crossing
- This device has two low threshold comparators. .
  - **Lower Threshold 1 (LLT1) Comparator**
    - When comparator data < = (SDFLTxCMPL1.LLT), the LLT1 comparator generates COMPL1 event.
    - The COMPL1 event is connected to both CEVT1 and CEVT2.
  - **Lower Threshold 2 (LLT2) Comparator**
    - When comparator data < = (SDFLTxCMPL2.LLT), LLT2 comparator generates COMPL2 event.
    - The COMPL2 event is connected to both CEVT1 and CEVT2.

7.5.10.9.3 Digital Filter

The digital filter works on a window of FIFO samples (SAMPWIN + 1) taken from the input. The filter output resolves to the majority value of the sample window, where majority is defined by the threshold (THRESH) value. If the majority threshold is not satisfied, the filter output remains unchanged.

For proper operation, the value of THRESH must be greater than SAMPWIN / 2.

A prescale function (CLKPRESCALE) determines the filter sampling rate, where the filter FIFO captures one sample every CLKPRESCALE system clocks. Old data from the FIFO is discarded.

A conceptual model of the digital filter is shown in Figure 7-345.

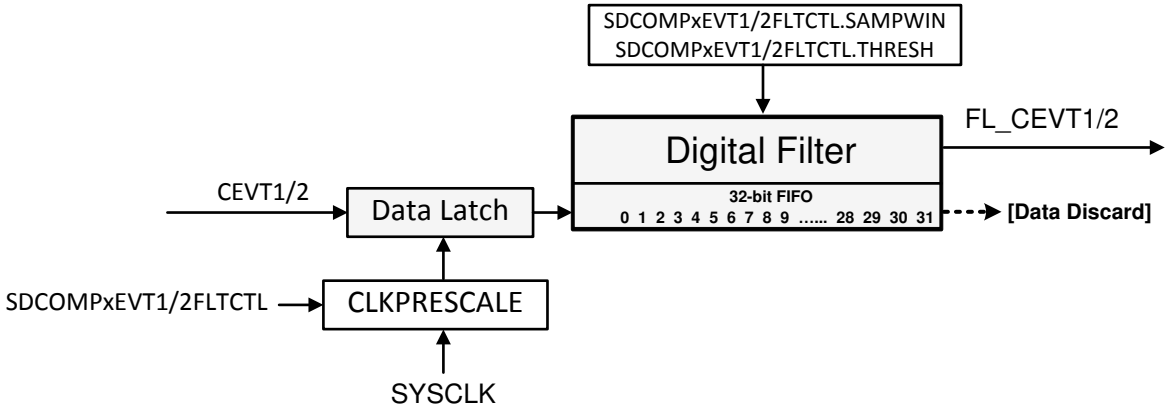


Figure 7-345. Digital Filter

Equivalent C code of the filter implementation is:

```
if (FILTER_OUTPUT == 0) {
    if (Num_1s_in_SAMPWIN >= THRESH) {
        FILTER_OUTPUT = 1;
    }
}
else {
    if (Num_0s_in_SAMPWIN >= THRESH) {
        FILTER_OUTPUT = 0;
    }
}
```

The configurable digital filter output is for filtering glitches. The application chooses between filtered or raw output of the comparator, and the output can reach the event flag register (SDIFLG.FLTx_FLG_CEVTx) and the CEVETxOUT event output of the SDFM module as show in *Digital Filter Outputs*. The figure also shows rise edge detection logic along the path from the filter to flag register.

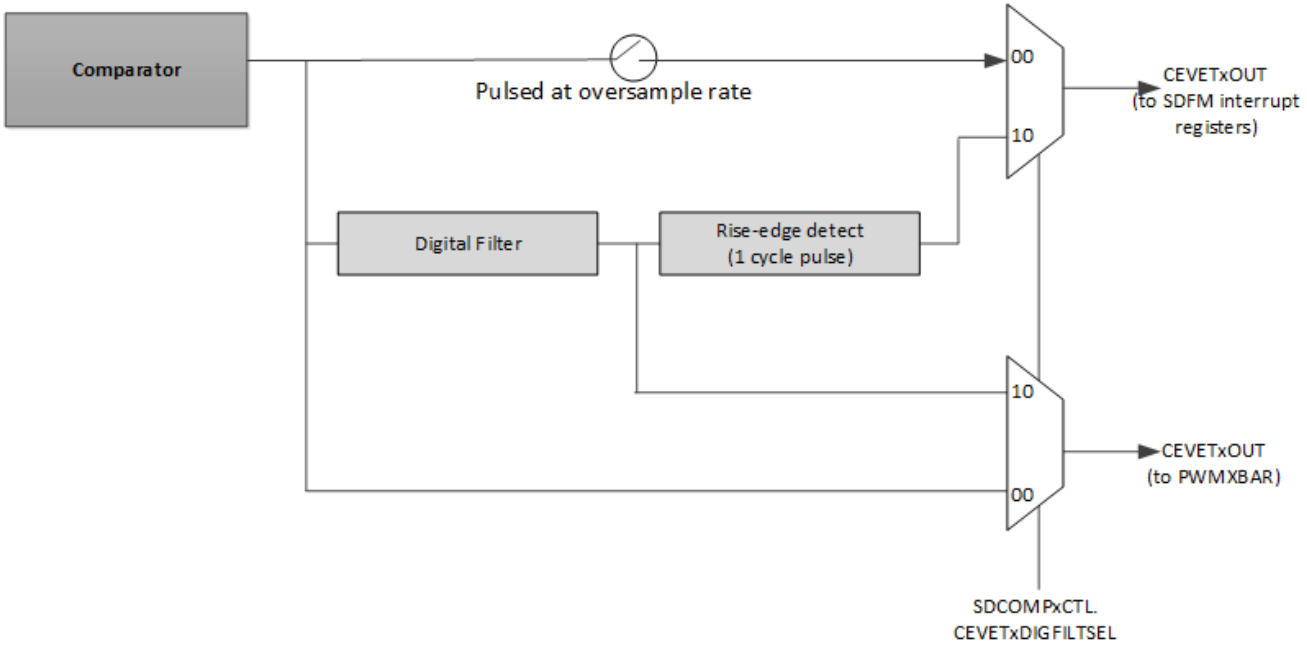


Figure 7-346. Digital Filter Outputs

When the digital filter path is chosen, the event flag register is set only once on the rise edge of digital filter output. If the event flag register is cleared, the flag is not set again even if the comparator output is maintained high. The issue is not present on the CEVETxOUT event going to XBAR nor if the raw output path is chosen (aka CEVTxDIGFILTSEL = 0).

Filter Initialization Sequence

To make sure of proper operation of the digital filter, the following initialization sequence is recommended:

1. Configure the digital filter parameters for operation:
   - Set SAMPWIN for the number of samples to monitor in the FIFO window.
   - Set THRESH for the threshold required for majority qualification.
   - Set CLKPRESCALE for the digital filter clock prescale value.
2. Initialize the sample values in the digital FIFO window by setting FILINIT = 1.

7.5.10.10 Theoretical SDFM Filter Output

The following equations can be used to derive a theoretical filter output of an SDFM filter output for both a comparator filter and a data filter.

$$\text{Density of ones in bitstream} = \frac{\text{Input Voltage} + \text{Vclipping}}{2 \times \text{Vclipping}} \tag{18}$$

Where:

- Vclipping = maximum differential voltage input range of modulator
- Input voltage = Differential input voltage applied to the modulator

$$\text{Comparator Filter Output}\left(\text{Theoretical}\right) = \text{Density of ones in bitstream} \times \text{Maximum Filter Output}\left(\text{FilterType, COSR}\right) \tag{19}$$

$$\text{FilterOutput} = \left\{\frac{\text{absolute}\left(\text{Input voltage}\right)}{\text{Vclipping}}\right\} \times \text{Maximum Filter Output}\left(\text{FilterType, DOSR}\right) \tag{20}$$

$$\text{Data Filter Output\_32bit}\left(\text{Theoretical}\right) = \begin{cases} \text{FilterOutput} & \text{if Input Voltage is +ve voltage} \\ \text{2's complement of FilterOutput} & \text{if input voltage is -ve voltage} \end{cases} \tag{21}$$

$$\text{Data Filter Output\_16bit}\left(\text{Theoretical}\right) = \text{Data Filter Output\_32bit}\left(\text{Theoretical}\right) >> \text{Shift value}\left(\text{FilterType, OSR}\right) \tag{22}$$

For example, when using the AMC1306x25 modulator:

AMC1306x25	Vclipping = Input voltage (AINP - AINN) =	320 mV 100 mV
SDFM filter settings	Filter type = Comparator OSR (COSR) = Data filter OSR (DOSR) =	3 32 100

Density of ones in bitstream	Using Equation 18	0.65625
Comparator filter output Filter type = Sinc3 COSR = 32	Using Equation 19	21504
Data filter output (32-bit) Filter type = Sinc3 DOSR = 100	Using Equation 20 and Equation 21	312500
Data filter output (32-bit) Filter type = Sinc3 DOSR = 100 (Right shift by 5)	Using Equation 22	9765

7.5.10.11 Interrupt Unit

Each SDFM can generate five CPU interrupts such as SDFM Error (SDy_ERR) and SDFM data ready (SDy_DRINTx) interrupts for each filter module.

7.5.10.11.1 SDFM (SDy_ERR) Interrupt Sources

Figure 7-347 shows the structure of SDy_ERR interrupt. SDy_ERR interrupt can be triggered by any of these 16 events.

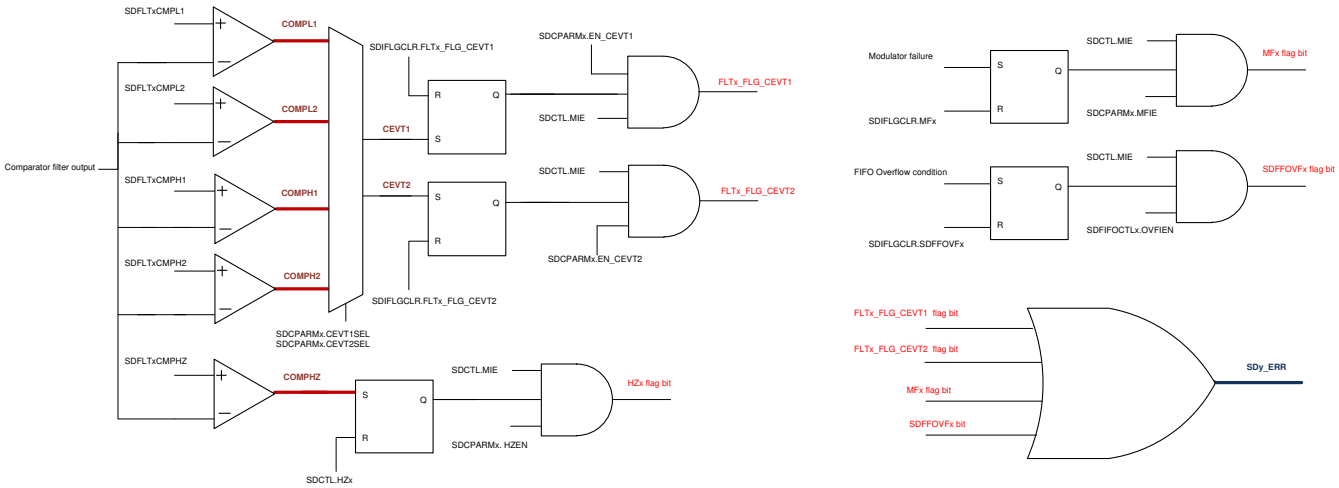


Figure 7-347. SDFM Error (SD_ERR) Interrupt Sources

1. Comparator Event1 (CEVT1)

CEVT1 events from any of the four comparator filter module can trigger CPU interrupt. This event can be configured to trigger SDy_ERR interrupt only if below configurations are made:

- Enable Main interrupt enable (SDCTL.MIE = 1)
- Enable comparator Event1 interrupt (SDCPARMx.EN_CEVT1 = 1)

On a CEVT1 event, SDIFLG.FLTx_FLG_CEVT1 flag bit is set. This flag bit can only be reset if the corresponding bit in SDIFLGCLR register is set and if the interrupt source is no longer active.

2. Comparator Event2 (CEVT2)

CEVT2 events from any of the four comparator filter module can trigger CPU interrupt. This event can be configured to trigger SDy_ERR interrupt only if below configurations are made:

- Enable Main interrupt enable (SDCTL.MIE = 1)
- Enable comparator event1 interrupt (SDCPARMx.EN_CEVT2 = 1)

On a CEVT2 event, SDIFLG.FLTx_FLG_CEVT2 flag bit is set. This flag bit can only be reset if the corresponding bit in SDIFLGCLR register is set and if the interrupt source is no longer active.

3. Modulator Failure (MFx) event

Modulator failures (MFx) are generated when SD-Cx goes missing. The modulator clock is considered missing if SD-Cx does not toggle for 64-SYSCLKs. MFx events from any of the four filter modules can trigger CPU interrupt. This event can be configured to trigger SDy_ERR interrupt only if below configurations are made:

- Enable Main Interrupt Enable (SDCTL.MIE = 1)
- Enable modulator clock failure interrupt source (SDCPARMx.MFIE = 1)

On a MFx event, SDIFLG.MFx flag bit is set. This flag bit can only be reset if the corresponding bit in SDIFLGCLR register is set and if the interrupt source is no longer active.

4. FIFO overflow (SDFFOVFx) event

The number of filter data available in FIFO at any given point can be tracked in SDFIFOCTLx.SDFFST. If the number of words received in FIFO is greater than Max FIFO depth (16), SDFFOVFx event is generated. SDFFOVFx events from any of the four filter modules can trigger CPU interrupt. This event can be configured to trigger SDy_ERR interrupt, only if below configurations are made:

- Enable SDFM FIFO (Set SDFIFOCTLx.FFEN = 1)
- Enable SDFM FIFO overflow interrupt (Set SDFIFOCTLx.OVFIEN = 1) and
- Enable Main interrupt enable (Set SDCTL.MIE = 1)

On a SDFFOVFx event, all subsequent data (primary) filter data is lost and is not stored in FIFO. SDIFLG.SDFFOVFx flag bit is set on a FIFO overflow event and this bit can be cleared if the corresponding bit in SDIFLGCLR register is set and if the interrupt source is no longer active.

7.5.10.11.2 Data Ready (DRINT) Interrupt Sources

Figure 7-348 shows the structure of interrupt SDy_DRINTx interrupt. Each SDy_DRINTx interrupt is triggered by corresponding Data Filter channel.

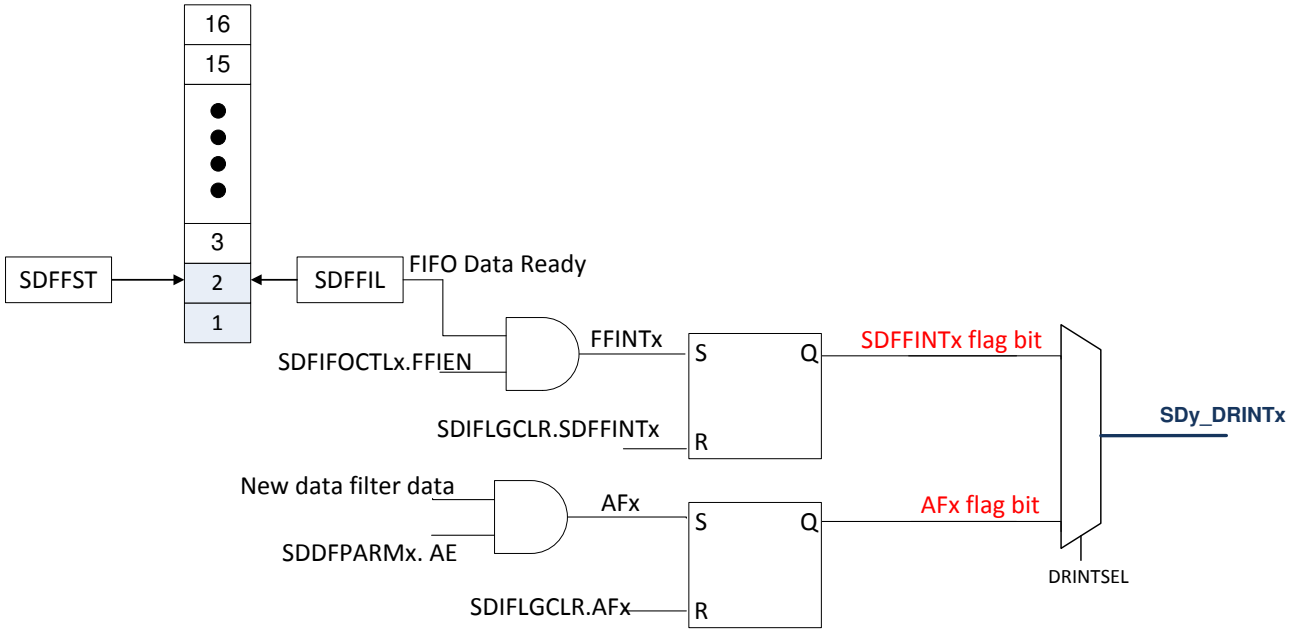


Figure 7-348. SDFM Data Ready (SDy_DRINTx) Interrupt

1. Data Acknowledge (AFx)

When the primary filter is ready with a new filter data, AFx event is generated. AFx events from each filter can generate their own SDy_DRINTx interrupt. This event can be configured to trigger SDy_DRINTx interrupt only if below configurations are made:

- Enable individual filter interrupts (SDDFPARMx. AE = 1)
- Select data-ready interrupt source AFx (DRINTx = AFx) (SDFIFOCTLx.DRINTSEL = 0)

On an AFx event, the SDIFLG.AFx flag bit is set. This flag bit can only be reset, if the corresponding bit in SDIFLGCLR register is set and if the interrupt source is no longer active.

2. Four FIFO Data ready interrupt (SDFFINTx)

FIFO Data Ready event is generated whenever SDFIFOCTLx.SDFFST >= SDFIFOCTLx.SDFFIL condition is met. FIFO data ready events from each filter can generate their own SDy_DRINTx interrupt. This event can be configured to trigger SDy_DRINTx interrupt only if below configurations are made:

Table 7-162 shows how the DRINTx output is selected.

- Enable SDFM FIFO (Set SDFIFOCTLx.FFEN = 1) and
- Enable SDFM FIFO interrupt (Set SDFIFOCTLx.FFIEN = 1)
- Select data-Ready interrupt source is SDFFINTx (DRINTx = SDFFINTx) (SDFIFOCTLx.DRINTSEL = 1)

Table 7-162. SDFM Data-Ready Interrupt (SDy_DRINTx) Output Selection

DRINTSEL	AE	FFIEN	FFEN	DRINTx
0	0	x	X	0
0	1	x	X	AFx
1	x	0	X	0
1	x	x	0	0
1	x	1	1	SDFFINTx

7.5.10.12 SDFM Programming Guide

Driver Information

Driver features are available at the SDFM driver page.

Software API Information

The SDFM driver provides an API to configure the SDFM module. Full documentation is located on APIs for SDFM

Example Usage

The below links shows an example on how to use SDFM

- SDFM EPWM sync CPU read
- SDFM Filter sync CPU read
- SDFM single channel filter sync CPU read

7.5.11 Crossbar (XBAR)

The crossbars (referred to as XBAR throughout this chapter) provide flexibility to connect device inputs, outputs, and internal resources in a variety of configurations.

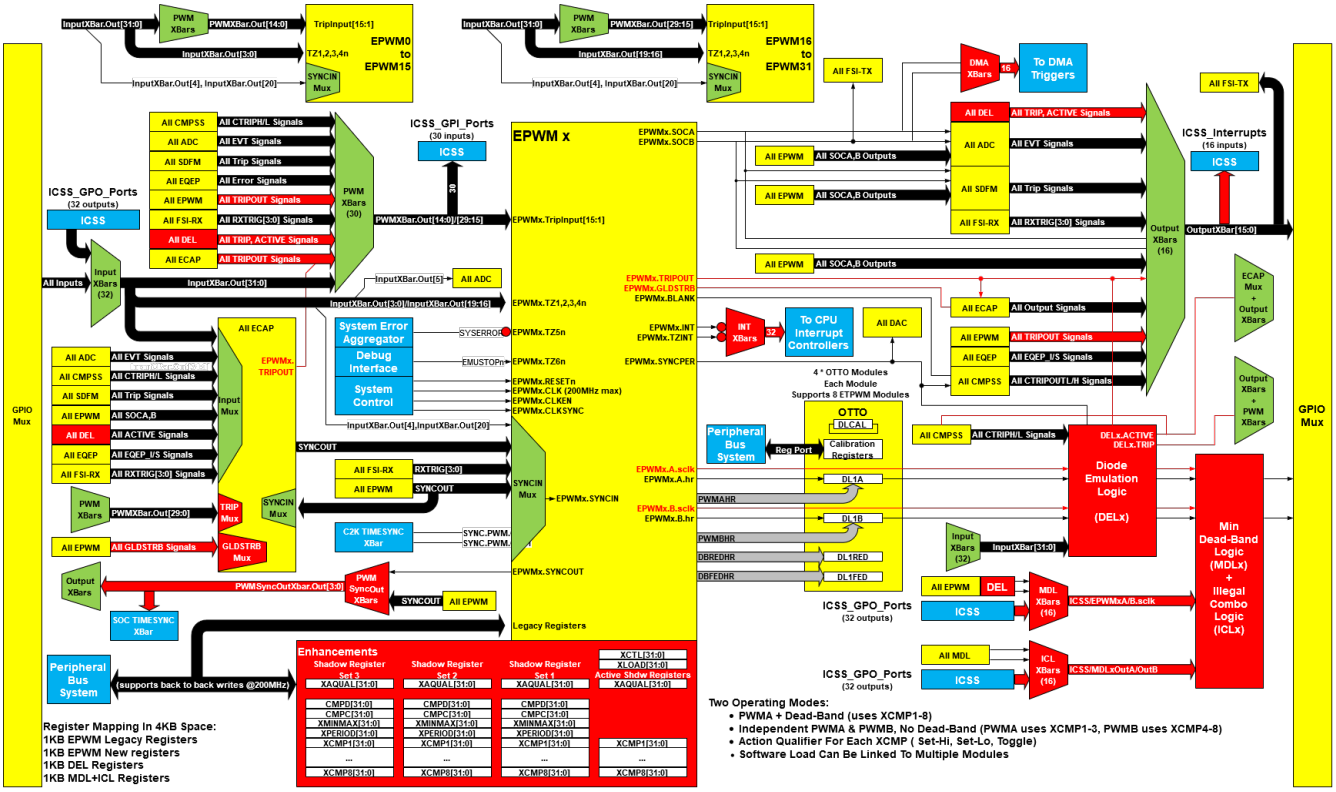


Figure 7-349. CONTROLSS XBAR Diagram

Further details about each of these XBARs can be found in the following sections.

7.5.11.1 INPUTXBAR

The INPUTXBAR routes the signals from any GPIO to different IP blocks such as the eCAP(s), ePWM(s), ICSS GPI(s), and the PWMXBAR. The INPUTXBAR has access to every GPIO and can route each signal to any (or multiple) of the IP blocks previously mentioned. This flexibility relieves some of the constraints on peripheral muxing by enabling any available GPIO pin to be used for slow changing I/O signals by the CONTROLSS. It is important to note that the function selected on the GPIO multiplexer does not affect the INPUTXBAR. The INPUTXBAR simply connects the signal on the input buffer to the selected destination. This flexibility enables routing the output of one peripheral to another (for example, measure the output of an ePWM with an eCAP for a frequency test). Apart from GPIOs, ICSS GPO(s), can also be used as inputs to the INPUTXBAR and be used in a similar fashion to any GPIO source.

The architecture of the INPUTXBAR is composed of multiple input unit XBARs which routes any of the INPUTXBAR sources to the single output of the XBAR. The two step multiplexer logic ensures that only one source is routed to the output.

The INPUTXBAR is configured by writing to the [INPUTXBAR[0-31]_G[0-1].SEL and INPUTXBAR[31:0].GSEL] registers. The Figure 7-350 shows all IP sources and destinations and Table 7-163 provides a comprehensive list of the destinations. For more information on configuration, see the CONTROLSS_INPUTXBAR register definitions in the XBAR register section.

Note

Please note INPUTXBAR routes GPIO pin to any of its 32 outputs and is not a OR implementation like most of CONTROLSSS XBARs

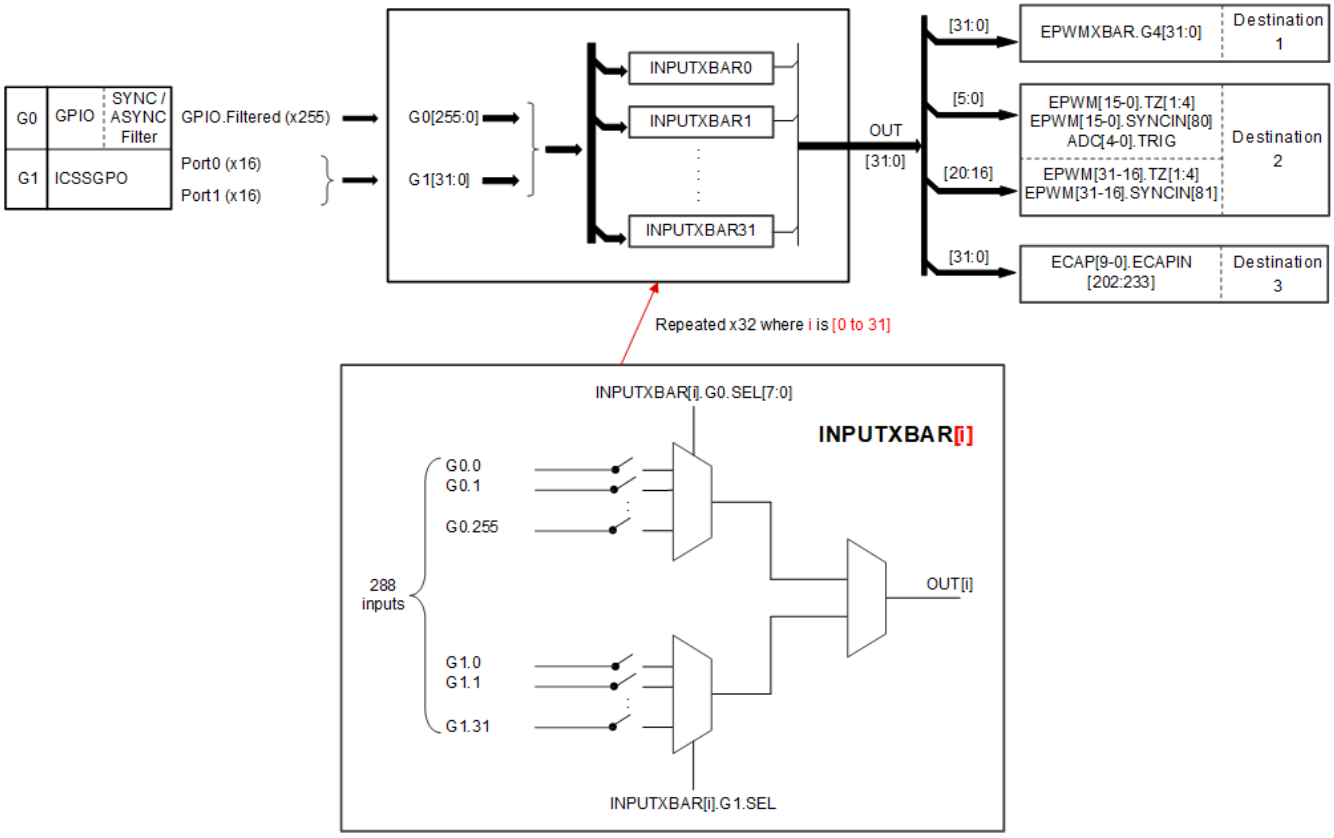


Figure 7-350. INPUTXBAR Functional Block Diagram

Note

Parameters for table below
w=[4:0]
x=[15:0]
y=[31:16]
z=[31:0]
u=[9:0]

Table 7-163. INPUTXBAR Output Destinations

INPUTXBAR Outputs	Destination-1	Destination-2	Destination-3
INPUTXBAR.Out0	PWMXBAR.G4.0	EPWMx.TZ1	ECAPu.ECAPIN.202
INPUTXBAR.Out1	PWMXBAR.G4.1	EPWMx.TZ2	ECAPu.ECAPIN.203
INPUTXBAR.Out2	PWMXBAR.G4.2	EPWMx.TZ3	ECAPu.ECAPIN.204
INPUTXBAR.Out3	PWMXBAR.G4.3	EPWMx.TZ4	ECAPu.ECAPIN.205
INPUTXBAR.Out4	PWMXBAR.G4.4	EPWMz.SYNCIN.80	ECAPu.ECAPIN.206
INPUTXBAR.Out5	PWMXBAR.G4.5	ADCw.TRIG.5	ECAPu.ECAPIN.207
INPUTXBAR.Out6	PWMXBAR.G4.6	Not Used	ECAPu.ECAPIN.208
INPUTXBAR.Out7	PWMXBAR.G4.7	Not Used	ECAPu.ECAPIN.209
INPUTXBAR.Out8	PWMXBAR.G4.8	Not Used	ECAPu.ECAPIN.210
INPUTXBAR.Out9	PWMXBAR.G4.9	Not Used	ECAPu.ECAPIN.211
INPUTXBAR.Out10	PWMXBAR.G4.10	Not Used	ECAPu.ECAPIN.212
INPUTXBAR.Out11	PWMXBAR.G4.11	Not Used	ECAPu.ECAPIN.213
INPUTXBAR.Out12	PWMXBAR.G4.12	Not Used	ECAPu.ECAPIN.214
INPUTXBAR.Out13	PWMXBAR.G4.13	Not Used	ECAPu.ECAPIN.215
INPUTXBAR.Out14	PWMXBAR.G4.14	Not Used	ECAPu.ECAPIN.216
INPUTXBAR.Out15	PWMXBAR.G4.15	Not Used	ECAPu.ECAPIN.217
INPUTXBAR.Out16	PWMXBAR.G4.16	EPWMy.TZ1	ECAPu.ECAPIN.218
INPUTXBAR.Out17	PWMXBAR.G4.17	EPWMy.TZ2	ECAPu.ECAPIN.219
INPUTXBAR.Out18	PWMXBAR.G4.18	EPWMy.TZ3	ECAPu.ECAPIN.220
INPUTXBAR.Out19	PWMXBAR.G4.19	EPWMy.TZ4	ECAPu.ECAPIN.221
INPUTXBAR.Out20	PWMXBAR.G4.20	EPWMz.SYNCIN.81	ECAPu.ECAPIN.222
INPUTXBAR.Out21	PWMXBAR.G4.21	Not Used	ECAPu.ECAPIN.223
INPUTXBAR.Out22	PWMXBAR.G4.22	Not Used	ECAPu.ECAPIN.224
INPUTXBAR.Out23	PWMXBAR.G4.23	Not Used	ECAPu.ECAPIN.225
INPUTXBAR.Out24	PWMXBAR.G4.24	Not Used	ECAPu.ECAPIN.226
INPUTXBAR.Out25	PWMXBAR.G4.25	Not Used	ECAPu.ECAPIN.227
INPUTXBAR.Out26	PWMXBAR.G4.26	Not Used	ECAPu.ECAPIN.228
INPUTXBAR.Out27	PWMXBAR.G4.27	Not Used	ECAPu.ECAPIN.229
INPUTXBAR.Out28	PWMXBAR.G4.28	Not Used	ECAPu.ECAPIN.230
INPUTXBAR.Out29	PWMXBAR.G4.29	Not Used	ECAPu.ECAPIN.231
INPUTXBAR.Out30	PWMXBAR.G4.30	Not Used	ECAPu.ECAPIN.232
INPUTXBAR.Out31	PWMXBAR.G4.31	Not Used	ECAPu.ECAPIN.233

For more information on configuration, see the INPUTXBAR register definitions.

7.5.11.2 PWMXBAR

The PWMXBAR routes the trip events of different real-time CONTROLSS instances to either different ePWM trip inputs or to ICSSM GPI inputs. The sources of trip events to ePWM can be any of the following: compare

subsystem trip high and low events, SDFM filter events, ADC events, INPUTXBAR outputs, ePWM tripout events, diode emulation trip/active signals, eQEP error events, FSIRX triggers, and eCAP trip outputs.

The architecture of the PWMXBAR includes unit XBARs which allow any of the PWMXBAR inputs to be routed to a single output of the XBAR. Multiple PWMXBAR outputs can have the same trip source routed to them. PWMXBAR outputs can also trigger ICSSM GPI inputs and can capture any inputs to the ePWM trip inputs. Each unit XBAR also has an associated set of PWMXBAR_STATUS and PWMXBAR_FLAG registers which can be used to inform the application of events. The PWMXBAR_FLAG_CLR register allows the application to clear the flags of captured events in a controlled fashion.

The PWMXBAR is configured by writing to the [PWMXBAR[0-29]_G[0-8].SEL] registers. The Figure 7-351 shows all IP sources and destinations and Table 7-164 provides a comprehensive list of the destinations. For more information on configuration, see the CONTROLSS_PWMXBAR register definitions in the XBAR register section.

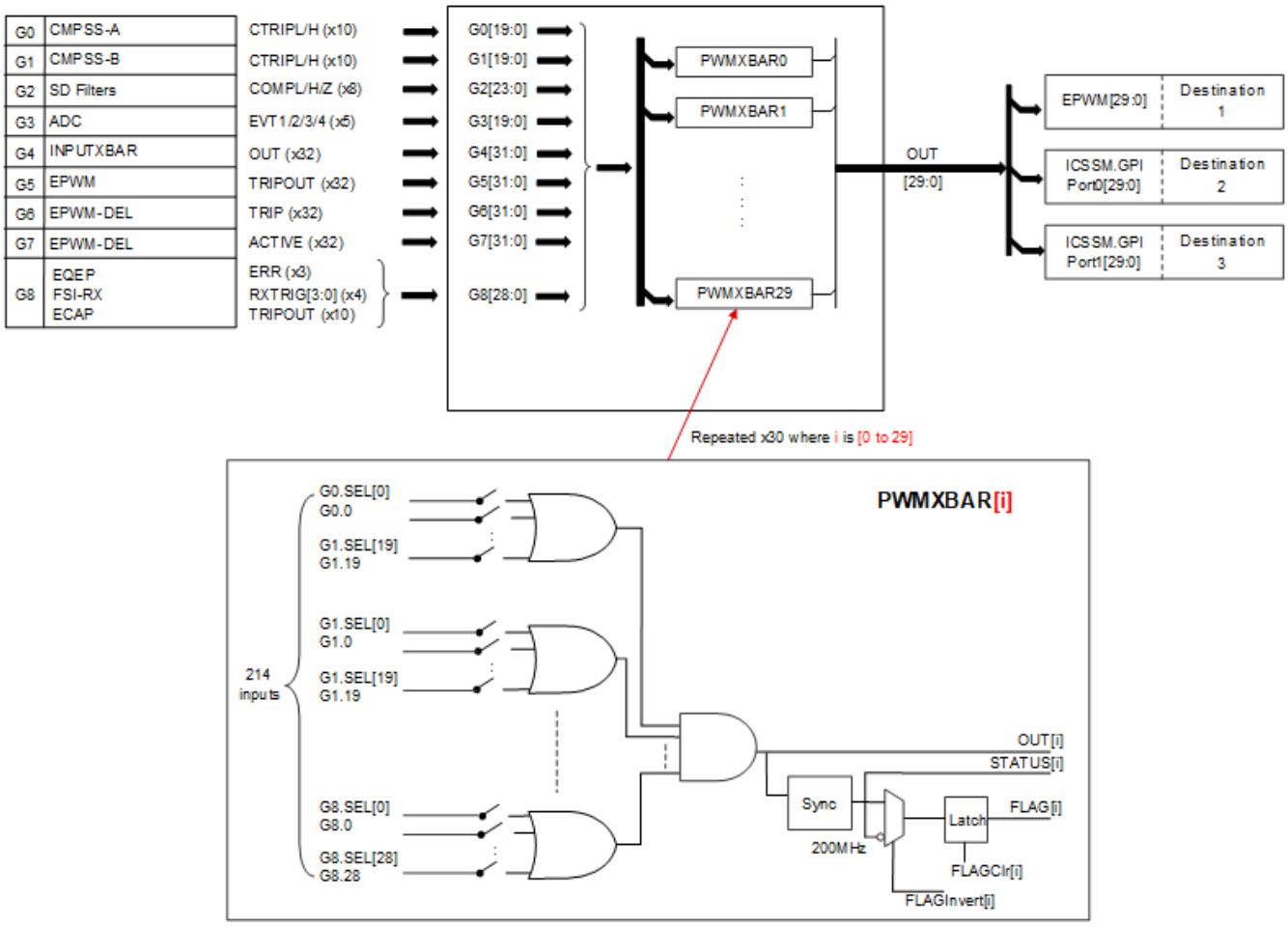


Figure 7-351. PWMXBAR Functional Block Diagram

Note

Parameters for table below
x=[15:0],y=[31:16]

Table 7-164. PWMXBAR Output Destinations

PWMXBAR Outputs	Destination-1	Destination-2	Destination-3
PWMXBAR.Out0	EPWMx.TripInput.1	ICSSM.GPI_Port0.0	ICSSM.GPI_Port1.0
PWMXBAR.Out1	EPWMx.TripInput.2	ICSSM.GPI_Port0.1	ICSSM.GPI_Port1.1

Table 7-164. PWMXBAR Output Destinations (continued)

PWMXBAR Outputs	Destination-1	Destination-2	Destination-3
PWMXBAR.Out2	EPWMx.TripInput.3	ICSSM.GPI_Port0.2	ICSSM.GPI_Port1.2
PWMXBAR.Out3	EPWMx.TripInput.4	ICSSM.GPI_Port0.3	ICSSM.GPI_Port1.3
PWMXBAR.Out4	EPWMx.TripInput.5	ICSSM.GPI_Port0.4	ICSSM.GPI_Port1.4
PWMXBAR.Out5	EPWMx.TripInput.6	ICSSM.GPI_Port0.5	ICSSM.GPI_Port1.5
PWMXBAR.Out6	EPWMx.TripInput.7	ICSSM.GPI_Port0.6	ICSSM.GPI_Port1.6
PWMXBAR.Out7	EPWMx.TripInput.8	ICSSM.GPI_Port0.7	ICSSM.GPI_Port1.7
PWMXBAR.Out8	EPWMx.TripInput.9	ICSSM.GPI_Port0.8	ICSSM.GPI_Port1.8
PWMXBAR.Out9	EPWMx.TripInput.10	ICSSM.GPI_Port0.9	ICSSM.GPI_Port1.9
PWMXBAR.Out10	EPWMx.TripInput.11	ICSSM.GPI_Port0.10	ICSSM.GPI_Port1.10
PWMXBAR.Out11	EPWMx.TripInput.12	ICSSM.GPI_Port0.11	ICSSM.GPI_Port1.11
PWMXBAR.Out12	EPWMx.TripInput.13	ICSSM.GPI_Port0.12	ICSSM.GPI_Port1.12
PWMXBAR.Out13	EPWMx.TripInput.14	ICSSM.GPI_Port0.13	ICSSM.GPI_Port1.13
PWMXBAR.Out14	EPWMx.TripInput.15	ICSSM.GPI_Port0.14	ICSSM.GPI_Port1.14
PWMXBAR.Out15	EPWMy.TripInput.1	ICSSM.GPI_Port0.15	ICSSM.GPI_Port1.15
PWMXBAR.Out16	EPWMy.TripInput.2	ICSSM.GPI_Port0.16	ICSSM.GPI_Port1.16
PWMXBAR.Out17	EPWMy.TripInput.3	ICSSM.GPI_Port0.17	ICSSM.GPI_Port1.17
PWMXBAR.Out18	EPWMy.TripInput.4	ICSSM.GPI_Port0.18	ICSSM.GPI_Port1.18
PWMXBAR.Out19	EPWMy.TripInput.5	ICSSM.GPI_Port0.19	ICSSM.GPI_Port1.19
PWMXBAR.Out20	EPWMy.TripInput.6	ICSSM.GPI_Port0.20	ICSSM.GPI_Port1.20
PWMXBAR.Out21	EPWMy.TripInput.7	ICSSM.GPI_Port0.21	ICSSM.GPI_Port1.21
PWMXBAR.Out22	EPWMy.TripInput.8	ICSSM.GPI_Port0.22	ICSSM.GPI_Port1.22
PWMXBAR.Out23	EPWMy.TripInput.9	ICSSM.GPI_Port0.23	ICSSM.GPI_Port1.23
PWMXBAR.Out24	EPWMy.TripInput.10	ICSSM.GPI_Port0.24	ICSSM.GPI_Port1.24
PWMXBAR.Out25	EPWMy.TripInput.11	ICSSM.GPI_Port0.25	ICSSM.GPI_Port1.25
PWMXBAR.Out26	EPWMy.TripInput.12	ICSSM.GPI_Port0.26	ICSSM.GPI_Port1.26
PWMXBAR.Out27	EPWMy.TripInput.13	ICSSM.GPI_Port0.27	ICSSM.GPI_Port1.27
PWMXBAR.Out28	EPWMy.TripInput.14	ICSSM.GPI_Port0.28	ICSSM.GPI_Port1.28
PWMXBAR.Out29	EPWMy.TripInput.15	ICSSM.GPI_Port0.29	ICSSM.GPI_Port1.29

7.5.11.3 MDLXBAR

The MDLXBAR is able to route one of three input signals to the Minimum Dead-Band submodule inside the ePWM module. The input signals comprise of either PWMA or PWMB after having passed through the Diode Emulation block, or the ICSSM GPO ports. For information on MDLXBAR use cases, refer to ePWM module specification.

The MDLXBAR architecture allows for each MDL unit XBAR to select from any of the three aforementioned signals and routes the signal to a single output of the XBAR. The output of MDLXBAR can be sourced to each of the 32 Minimum Dead-Band logic (MDL) submodules inside the ePWM module.

The MDLXBAR is configured by writing to the MDLXBAR[0-15].G[0-2].SEL registers. The Figure 7-352 shows all IP sources and destinations and Table 7-165 provides a comprehensive list of the destinations. For more information on configuration, see the CONTROLSS_MDLXBAR register definitions.

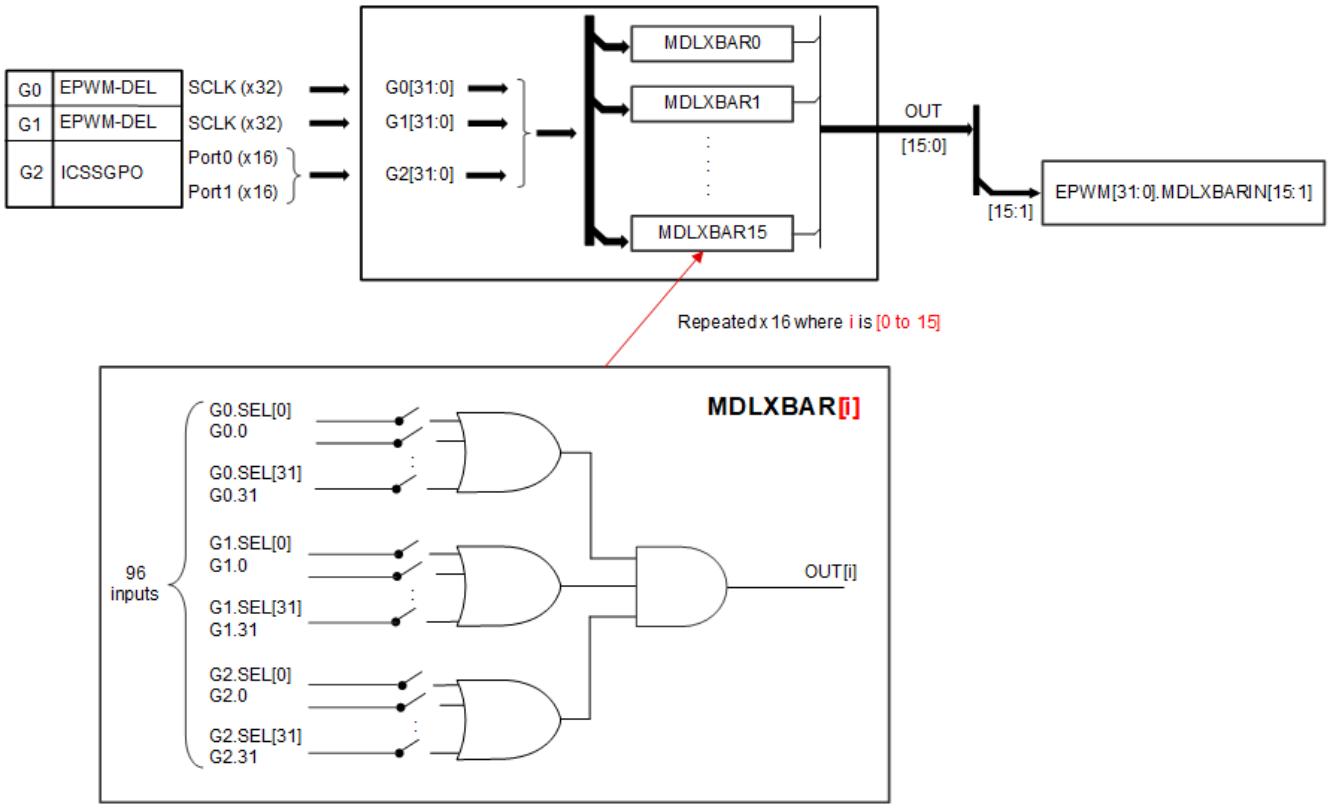


Figure 7-352. MDLXBAR Functional Block Diagram

Note

Parameters for table below
x=[31:0]

Table 7-165. MDL XBAR Output Destinations

MDLXBAR Outputs	Destination-1
MDLXBAR.Out0	Not Used
MDLXBAR.Out1	EPWMx.MDLXBARIN.1
MDLXBAR.Out2	EPWMx.MDLXBARIN.2
MDLXBAR.Out3	EPWMx.MDLXBARIN.3
MDLXBAR.Out4	EPWMx.MDLXBARIN.4
MDLXBAR.Out5	EPWMx.MDLXBARIN.5
MDLXBAR.Out6	EPWMx.MDLXBARIN.6
MDLXBAR.Out7	EPWMx.MDLXBARIN.7
MDLXBAR.Out8	EPWMx.MDLXBARIN.8
MDLXBAR.Out9	EPWMx.MDLXBARIN.9
MDLXBAR.Out10	EPWMx.MDLXBARIN.10
MDLXBAR.Out11	EPWMx.MDLXBARIN.11
MDLXBAR.Out12	EPWMx.MDLXBARIN.12
MDLXBAR.Out13	EPWMx.MDLXBARIN.13
MDLXBAR.Out14	EPWMx.MDLXBARIN.14

Table 7-165. MDL XBAR Output Destinations (continued)

MDLXBAR Outputs	Destination-1
MDLXBAR.Out15	EPWMx.MDLXBARIN.15

7.5.11.4 ICLXBAR

The ICLXBAR is able to route one of three input signals to the Illegal Combination Logic (ICL) submodules inside the PWM module. The input signals comprise of either PWMA or PWMB after having passed through the Minimum Dead-Band block, or the ICSSM GPO ports. For information on ICLXBAR use cases, refer to ePWM module specification.

The ICLXBAR architecture allows for each ICL unit XBAR to select from any of the three aforementioned signals and routes the signal to a single output of the XBAR. The output of ICLXBAR can be sourced to each of the 32 ICL submodules inside the ePWM module.

The ICLXBAR is configured by writing to the ICLXBAR[0-15].G[0-2].SEL registers. The Figure 7-353 shows all IP sources and destinations and Table 7-166 provides a comprehensive list of the destinations. For more information on configuration, see the CONTROLSS_ICLXBAR register definitions.

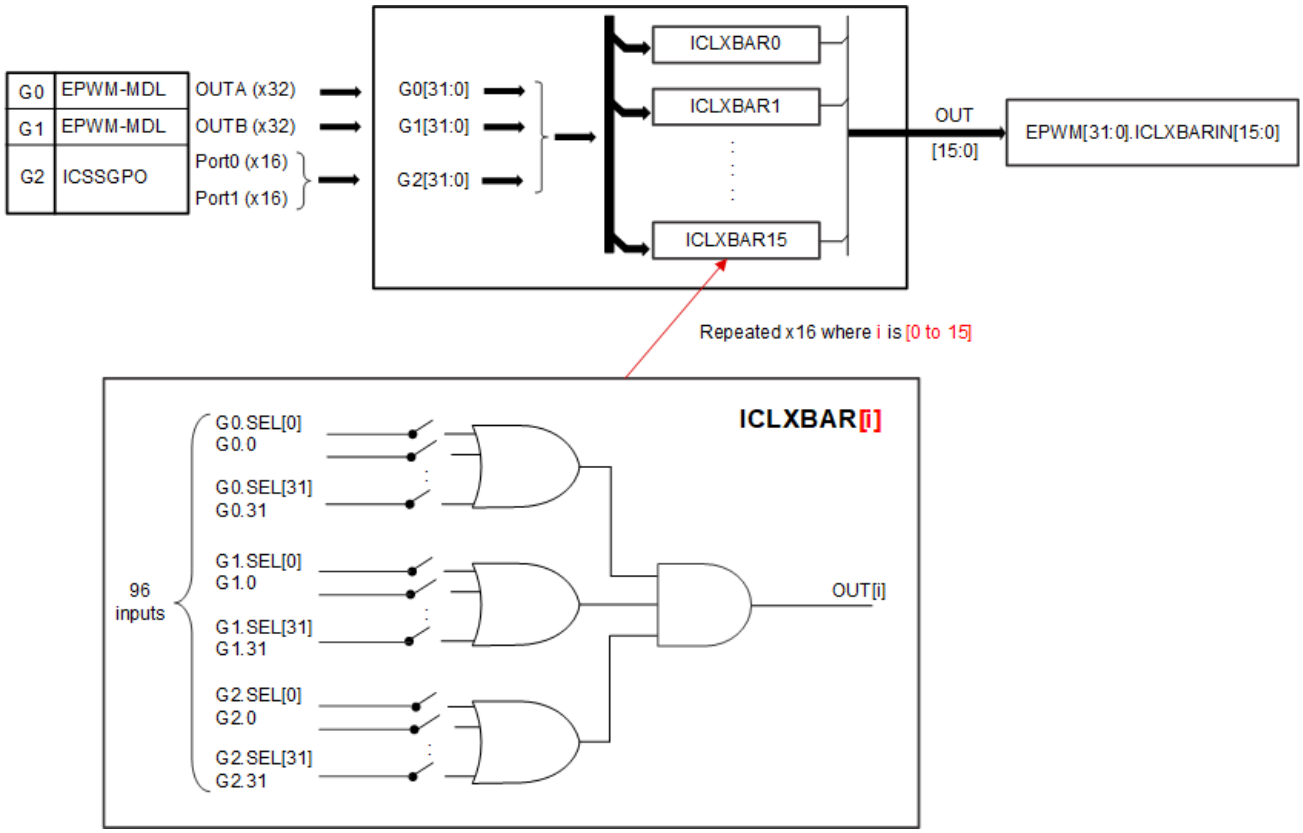


Figure 7-353. ICLXBAR Functional Block Diagram

Parameters for table below
x=[31:0]

Table 7-166. ICLXBAR Output Destinations

ICLXBAR outputs	Desination-1
ICLXBAR.Out0	EPWMx.ICLXBARIN.0
ICLXBAR.Out1	EPWMx.ICLXBARIN.1
ICLXBAR.Out2	EPWMx.ICLXBARIN.2

Table 7-166. ICLXBAR Output Destinations (continued)

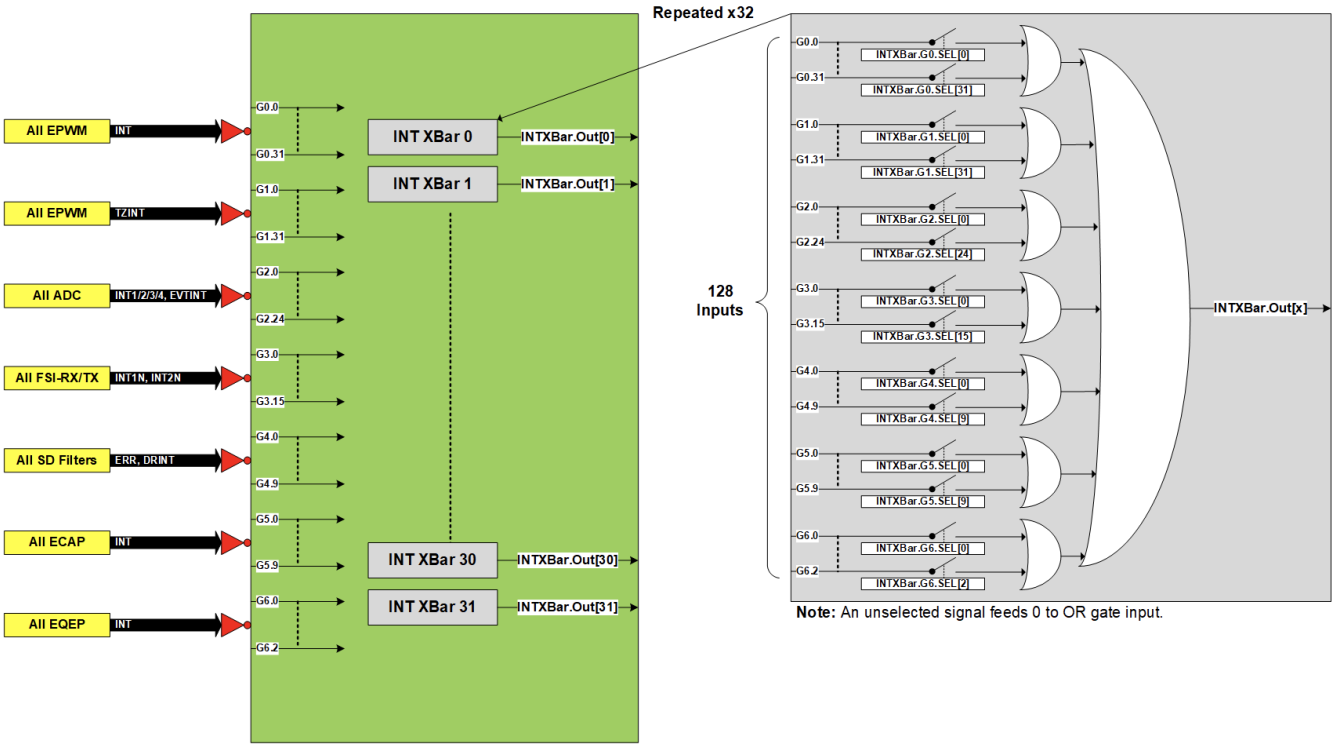
ICLXBAR outputs	Desination-1
ICLXBAR.Out3	EPWMx.ICLXBARIN.3
ICLXBAR.Out4	EPWMx.ICLXBARIN.4
ICLXBAR.Out5	EPWMx.ICLXBARIN.5
ICLXBAR.Out6	EPWMx.ICLXBARIN.6
ICLXBAR.Out7	EPWMx.ICLXBARIN.7
ICLXBAR.Out8	EPWMx.ICLXBARIN.8
ICLXBAR.Out9	EPWMx.ICLXBARIN.9
ICLXBAR.Out10	EPWMx.ICLXBARIN.10
ICLXBAR.Out11	EPWMx.ICLXBARIN.11
ICLXBAR.Out12	EPWMx.ICLXBARIN.12
ICLXBAR.Out13	EPWMx.ICLXBARIN.13
ICLXBAR.Out14	EPWMx.ICLXBARIN.14
ICLXBAR.Out15	EPWMx.ICLXBARIN.15

7.5.11.5 INTXBAR

On this device, the INT x-bar is used to route real-time CONTROLSS peripheral interrupts to the SoC interrupt controller. The idea is to limit the number of interrupts to 32 within real-time CONTROLSS before connecting to the SOC interrupt controller.

The INT x-bar is further made up of unit x-bars, its architecture allows multiple interrupt sources to be active. For ease of readability, the interrupt sources are grouped together based on IP generating the same. Interrupt sources are active low and before entering the x-bar, these are inverted to be active high.

The INT x-bar configured by way of the INTXBAR[31:0].G[6:0].SEL registers. The available IP source for each INPUTx is shown in INTXBAR figure below.



7.5.11.6 DMAXBAR

On this device, the DMA x-bar is used to route DMA requests from real-time control subsystem peripherals to the SoC EDMA. The idea is to limit the number of DMA requests to 16 within real-time control subsystem before connecting it to the SOC EDMA.

The DMA x-bar is further made up of unit x-bars, which allows 2 tier selection of any one of the DMA request sources. Except for the ePWM(s) SOCA and SOCB, rest of the DMA sources are active low and before entering the x-bar, these are inverted to be active high.

The DMA x-bar configured by way of the DMAXBar[31:0].G[5:0].SEL and DMAXBAR[5:0].GSEL registers. The available IP sources for each INPUTx is shown in DMA x-bar figure below

Note

Please note DMAXBAR routes DMA requests from CONTROLSS IPs to any of its 16 outputs and is not a OR implementation like most of CONTROLSSS XBARs

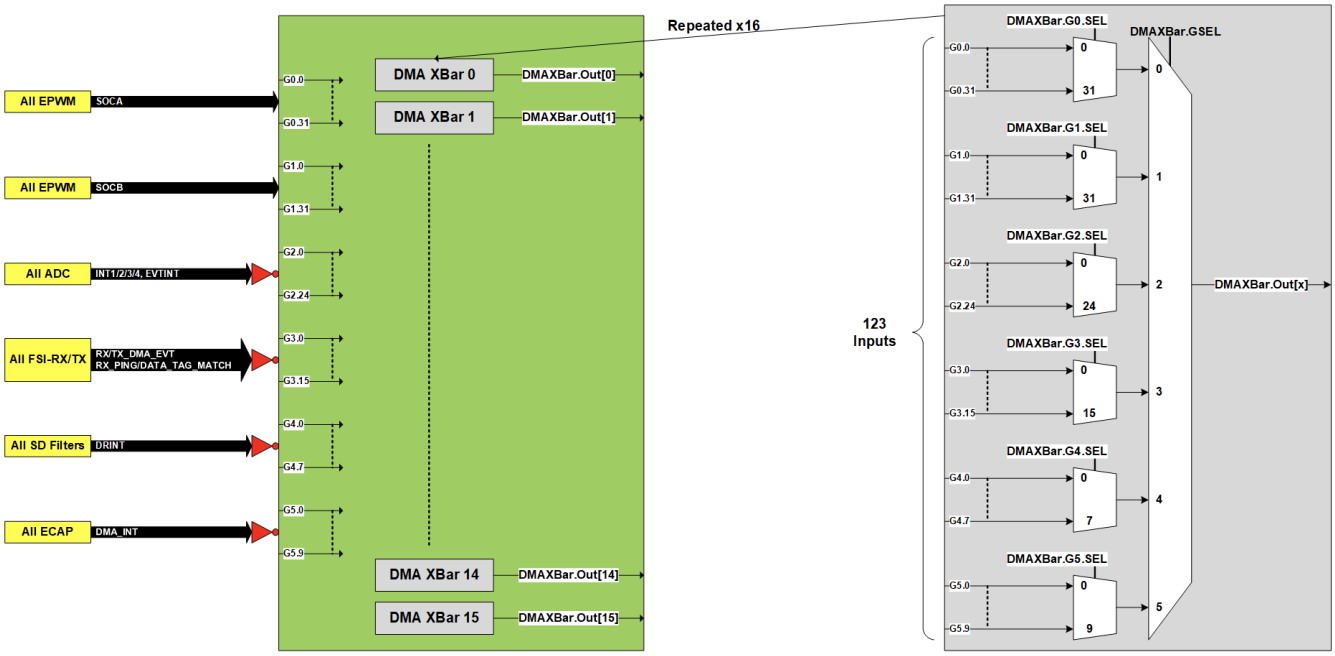


Figure 7-354. DMAXBAR Block Diagram

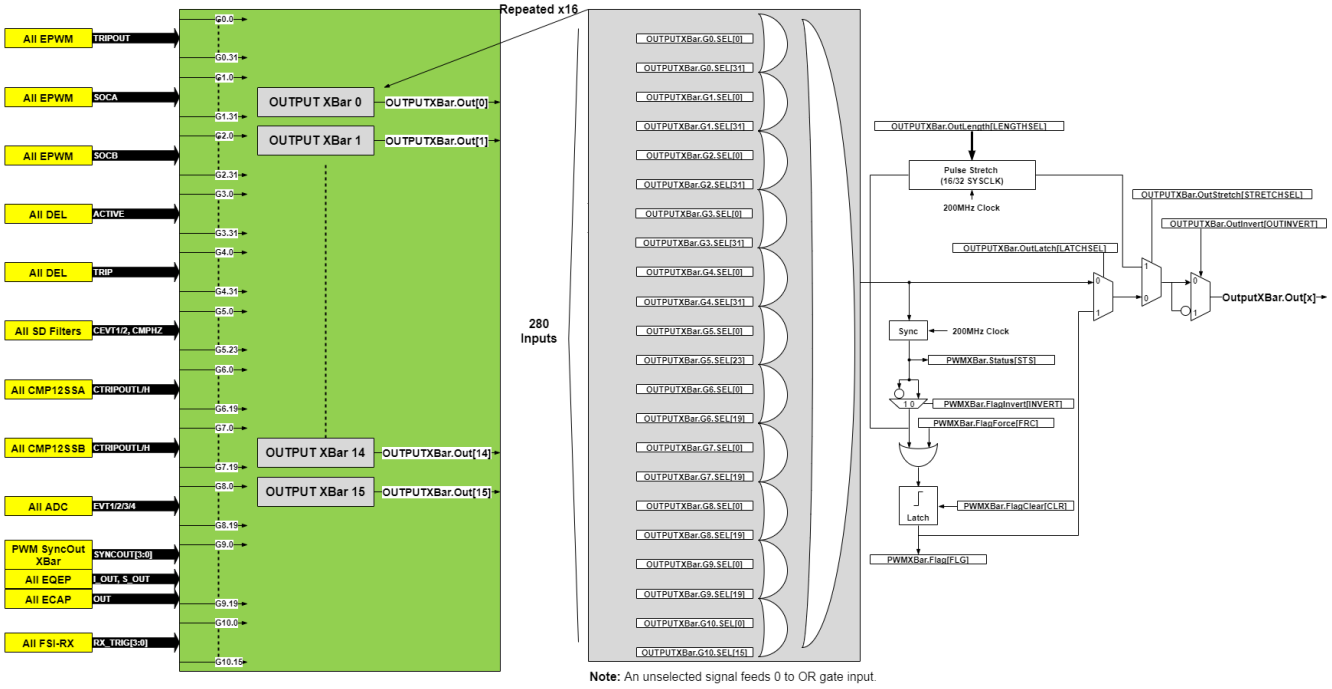
7.5.11.7 OUTPUTXBAR

On this device, the OUTPUT x-bar is used to route signals from all the control peripheral trip events to the output x-bar mapped pads or to the PRU-ICSS interrupts.

The source of these trip events is EPWM trip outputs, SOCA, SOCB, Diode Emulation Logic (DEL) generated active and trip events, SD filter events, compare subsystem high and low trips, ADC events, PWM syncout x-bar sync outputs, EQEP index and strobe, and ECAP outputs.

The architecture of the output x-bar is composed of output unit x-bars which routes any of the output x-bar inputs to the single output of the unit x-bar. If needed, the same trip source can be routed to multiple OUTPUT x-bar outputs by suitable programming. Each output unit x-bar also has an associated set of status and clear/flags which can be used by application to know about an event by reading the status bit. The clear flags allows to clear the captured events in a controlled fashion. Since the output x-bar is routed to GPIOs, the internal low width pulses to be stretched to 16 or 32 cycles of 200MHz real-time control subsystem clock. The polarity of the latched signal is controlled by the status registers.

The OUTPUTXBAR is configured by way of the OUTPUTxSELECT registers. The available IP sources for each INPUTx is shown in OUTPUTXBAR figure below. While the Output x-bar outputs destination is show in the table below.



Note: An unselected signal feeds 0 to OR gate input.

Figure 7-355. OUTPUTXBAR Block Diagram

Figure 7-356. OUTPUTXBAR Block Diagram

Parameters for table below
x=[3:0]

OUTPUTXBAR Outputs	Destination-1	Destination-2	Destination-3	Destination-3	Destination-4
OUTPUTXBAR.Out0	ICSSM_SYNC0_PAD	FSI_TXx.EXTTRIGGER63	FSI_TXx.EXTPINGTRIGGER63	FSI_TXx.EXTPINGTRIGGER63	ICSSM.PR1_TARGET_INTR.31
OUTPUTXBAR.Out1	ICSSM_LATCH0_PAD	FSI_TXx.EXTTRIGGER62	FSI_TXx.EXTPINGTRIGGER62	FSI_TXx.EXTPINGTRIGGER62	ICSSM.PR1_TARGET_INTR.32
OUTPUTXBAR.Out2	MMCSD_WP_PAD	FSI_TXx.EXTTRIGGER61	FSI_TXx.EXTPINGTRIGGER61	FSI_TXx.EXTPINGTRIGGER61	ICSSM.PR1_TARGET_INTR.33
OUTPUTXBAR.Out3	MMCSD_CD_PAD	FSI_TXx.EXTTRIGGER60	FSI_TXx.EXTPINGTRIGGER60	FSI_TXx.EXTPINGTRIGGER60	ICSSM.PR1_TARGET_INTR.34
OUTPUTXBAR.Out4	MMCSD_POW_PAD	FSI_TXx.EXTTRIGGER59	FSI_TXx.EXTPINGTRIGGER59	FSI_TXx.EXTPINGTRIGGER59	ICSSM.PR1_TARGET_INTR.35
OUTPUTXBAR.Out5	I2C0_SCL_PAD	FSI_TXx.EXTTRIGGER58	FSI_TXx.EXTPINGTRIGGER58	FSI_TXx.EXTPINGTRIGGER58	ICSSM.PR1_TARGET_INTR.36
OUTPUTXBAR.Out6	I2C0_SDA_PAD	FSI_TXx.EXTTRIGGER57	FSI_TXx.EXTPINGTRIGGER57	FSI_TXx.EXTPINGTRIGGER57	ICSSM.PR1_TARGET_INTR.37
OUTPUTXBAR.Out7	UART0_RTS_PAD	FSI_TXx.EXTTRIGGER56	FSI_TXx.EXTPINGTRIGGER56	FSI_TXx.EXTPINGTRIGGER56	ICSSM.PR1_TARGET_INTR.38
OUTPUTXBAR.Out8	UART0_CTS_PAD	FSI_TXx.EXTTRIGGER55	FSI_TXx.EXTPINGTRIGGER55	FSI_TXx.EXTPINGTRIGGER55	ICSSM.PR1_TARGET_INTR.39
OUTPUTXBAR.Out9	SPI1_CS_PAD	FSI_TXx.EXTTRIGGER54	FSI_TXx.EXTPINGTRIGGER54	FSI_TXx.EXTPINGTRIGGER54	ICSSM.PR1_TARGET_INTR.40
OUTPUTXBAR.Out10	SPI1_CLK_PAD	FSI_TXx.EXTTRIGGER53	FSI_TXx.EXTPINGTRIGGER53	FSI_TXx.EXTPINGTRIGGER53	ICSSM.PR1_TARGET_INTR.41
OUTPUTXBAR.Out11	SPI1_TX_PAD	FSI_TXx.EXTTRIGGER52	FSI_TXx.EXTPINGTRIGGER52	FSI_TXx.EXTPINGTRIGGER52	ICSSM.PR1_TARGET_INTR.42
OUTPUTXBAR.Out12	SPI1_RX_PAD	FSI_TXx.EXTTRIGGER51	FSI_TXx.EXTPINGTRIGGER51	FSI_TXx.EXTPINGTRIGGER51	ICSSM.PR1_TARGET_INTR.43
OUTPUTXBAR.Out13	SPI2_CS_PAD	FSI_TXx.EXTTRIGGER50	FSI_TXx.EXTPINGTRIGGER50	FSI_TXx.EXTPINGTRIGGER50	ICSSM.PR1_TARGET_INTR.44

OUTPUTXBAR Outputs	Destination-1	Destination-2	Destination-3	Destination-3	Destination-4
OUTPUTXBAR.Out14	SPI2_CLK_PAD	FSI_TXx.EXTTRIGGER49	FSI_TXx.EXTPINGTRIGGER49	FSI_TXx.EXTPINGTRIGGER49	ICSSM.PR1_TARGET_INTR.45
OUTPUTXBAR.Out15	ICSSM_MII1_TXD3_PAD	FSI_TXx.EXTTRIGGER48	FSI_TXx.EXTPINGTRIGGER48	FSI_TXx.EXTPINGTRIGGER48	ICSSM.PR1_TARGET_INTR.46

7.5.11.8 PWMSYNCOUTXBAR

On this device, the PWM SyncOut x-bar is used to route signals from ePWM(s) sync output(s) to Output x-bar and SoC time sync logic.

The PWM SyncOut x-bar configured by way of the PWMSyncOutXBar[3:0].G0.SEL registers. The available IP sources for each input is shown in PWMSYNCOUTXBAR figure below while the PWM SyncOut x-bar destinations is shown in the table below.

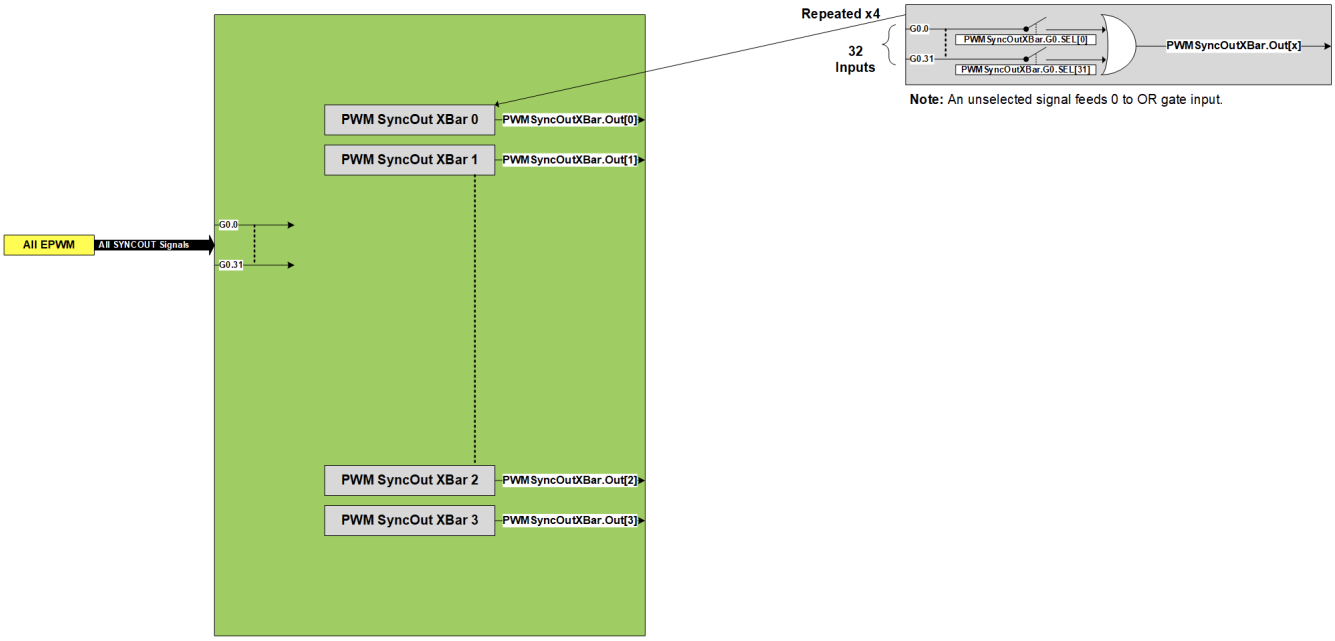


Figure 7-357. pwmsyncout x-bar

PWMSYNCOUTXBAROutputs	Destination-1	Destination-2
PWMSYNCOUTXBAR.Out0	OUTPUTXBAR.G9.0	TIMESYNCXBAR.IN_INTR6
PWMSYNCOUTXBAR.Out1	OUTPUTXBAR.G9.1	TIMESYNCXBAR.IN_INTR7
PWMSYNCOUTXBAR.Out2	OUTPUTXBAR.G9.2	TIMESYNCXBAR.IN_INTR8
PWMSYNCOUTXBAR.Out3	OUTPUTXBAR.G9.3	TIMESYNCXBAR.IN_INTR9

7.6 OptiFlash

This section describes the OptiFlash module in the device.

7.6.1 OptiFlash Overview

OptiFlash memory technology is a TI patented technology that enables cost effective and scalable high-performance Microcontrollers (MCU) with external flash. With traditional MCUs the ratio of Flash:SRAM typically ranges between 8:1 and 12:1. However, TI's AM26x MCUs equipped with OptiFlash, large on-chip SRAM (OCSRAM) and tightly coupled memory (TCM), it is possible to implement a cost optimized system, where the OCSRAM scales efficiently with external on-PCB Flash.

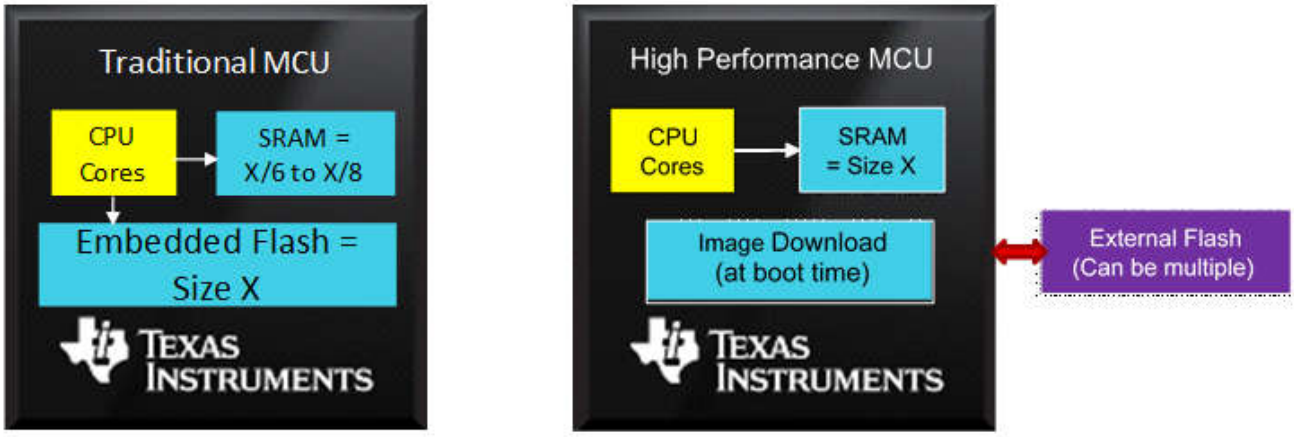


Figure 7-358. Typical MCU-Flash Architectures

Traditional MCUs with embedded flash on the same die, as shown in the block on the left in Figure 7-358 are limited in scale due to the high-voltage circuits needed for programming and erasing Flash. High-performance MCUs, as shown in the block on the right in Figure 7-358, use an external flash memory part and typically download the entire image to SRAM during boot. However, a common disadvantage of these MCUs is cost due to large SRAM size that must be greater than or equal to application software image size, and long boot and startup time.

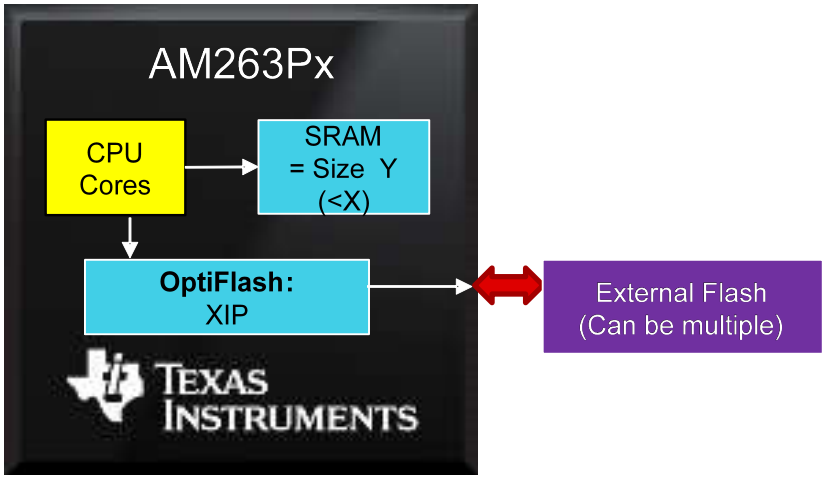


Figure 7-359. MCU-OptiFlash Architecture

OptiFlash Technology is a low-cost solution that enables high performance by mixing the correct ratio of on-chip SRAM and external Flash, as seen in Figure 7-359. An SoC with integrated OptiFlash Technology, such as AM263Px aims to solve the limitations faced by traditional High Performance MCUs with external Flash by proiding hybrid execution from internal SRAM and direct exectution from external Flash. This functionality is

referred to as Exectute in Place (XIP). XIP's goal is for exeution from external Flash to reach the performance of execution from internal SRAM.

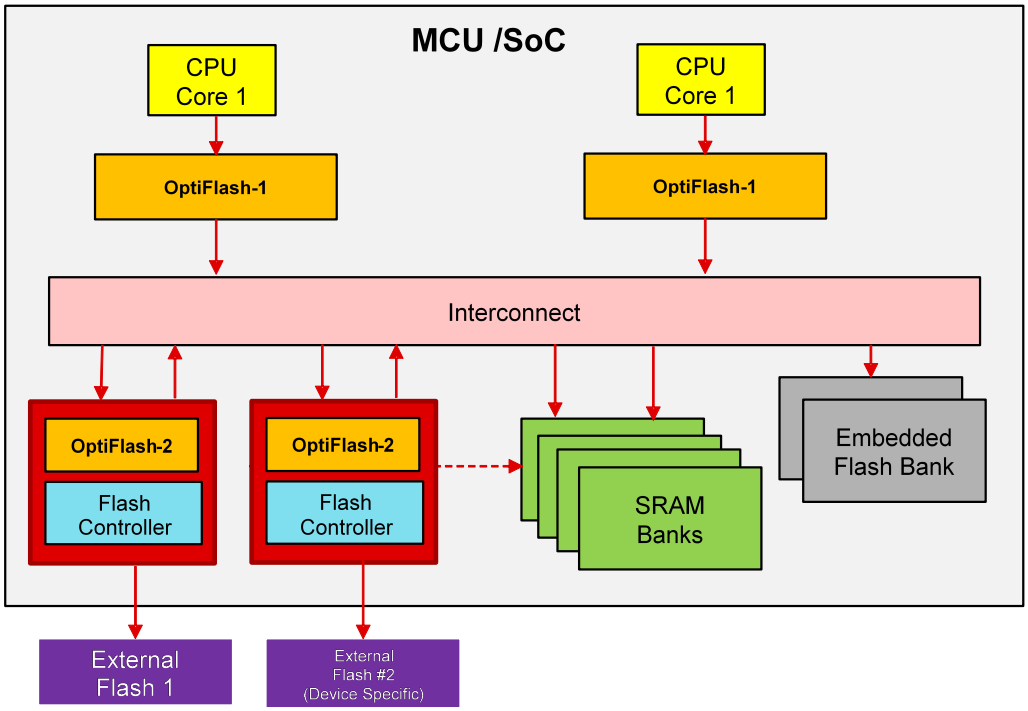


Figure 7-360. OptiFlash Technology - Key HW Components

The OptiFlash module has three hardware accelerator submodules inline to the **CPU data path** inside block *OptiFlash-1* in Figure 7-360 for improving flash performance:

- RAT - Region Based Address Translation.
- FLC - Fast Local Copy allowing code in slow access memory to be placed into on-chip SRAM for fast access.
- RL2 - Remote L2 is an L2 cache with remote cache data storage memory. That is, you can cache the system Flash into the SoC memory system.

These submodules have dependencies on each other. The space specified by the RAT is never within the FLC or RL2 ranges. Also, the space specified to be copied to internal SRAM by the FLC won't be cached by the RL2.

The OptiFlash module has three hardware accelerator submodules inline to the **Flash Controller and Interface** inside block *OptiFlash-2* as shown in Figure 7-360:

- ECC/Safety Engine
- OTFA - On-The-Fly Authentication
- FOTA - Firmware upgrade Over The Air

7.6.2 OptiFlash Components

7.6.2.1 Octal Serial Peripheral Interface (OSPI)

On the AM263Px SoC, the Octal-SPI (OSPI) flash controller runs at a maximum speed of 133 MHz DDR over 8 data lines.

For more information on the OSPI flash controller, see Section 13.3.3.6

7.6.2.2 Remote Layer 2 Cache (RL2)

OptiFlash supports a Remote L2 controller (RL2) for caching that is customized for optimized flash and application system performance. It can reduce external Flash access by up to 65-95% based on application profile. It acts as a Level 2 cache controller as it provides additional caching - specific to Flash storage - beyond the CPU Core's L1 caching. The cache is remote, meaning the actual cache memory can be part of any system memory, such as On-chip Memory (remote cache data storage memory) instead of a dedicated cache storage within the controller. The user has flexibility to specify the size of cache based on target application needs.

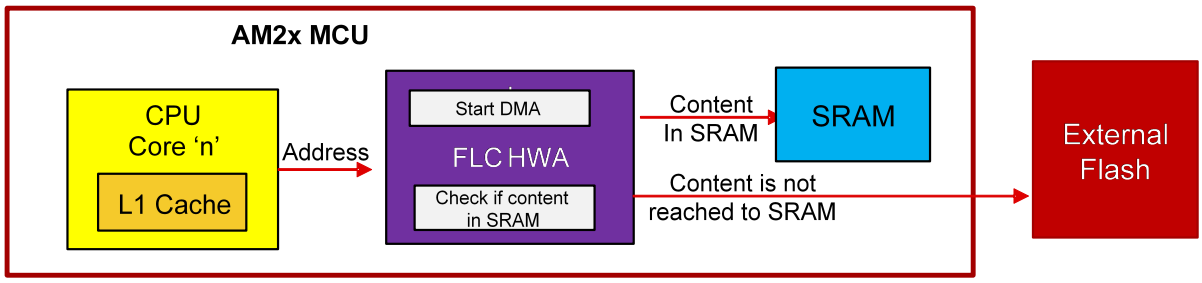
Because external flash holds only read-only data, the L2 cache is effectively caching code and read-only data only.

For more information on the RL2 module, see Section 13.3.2.1.

7.6.2.3 Fast Local Copy (FLC)

OptiFlash supports a Fast Local Copy (FLC) engine for image download acceleration during boot or run time to enable code download along with CPU execution in parallel. It redirects the CPU access to Flash when the copy is in progress and redirects the access to SRAM when the content is in valid in SRAM, as shown in FIGURE. This enables CPU to execute immediately without waiting for the copy to complete. The operation is transparent to CPU, and results in reduced boot time, in order to meet startup time goals similar to an embedded flash and also provide dynamic overlay for run time performance.

Figure 7-361. FLC Block Diagram



When the CPU requests an address, the FLC HWA looks for the copied address in the internal SRAM, and returns the data corresponding to that address. Otherwise, the data is fetched from the external flash. Because the data is returned from internal SRAM rather than external flash, the fetch time is reduced, thus providing a boost in overall performance.

For more information on FLC, see Section 13.3.2.2.

7.6.2.4 On-the-Fly Authentication (OTFA)

On-the-fly encryption and Authentication (OTFA) enables external memory IP protection at runtime during XIP. The device has an option to disable OTFA. For more information on OTFA, see Section 13.3.3.8.

7.6.2.5 Region Address Translation (RAT)

The Region based Address Translation (RAT) allows segments of the memory map to be relocated to a different address in the system. It is a dynamic address translator for common code (placed in SRAM) instead of duplicated/redundant code across multiple cores that is downloaded to on-chip SRAM during boot-time.

For more information on RAT, see Section 13.3.2.3

7.6.2.6 Firmware Upgrade Over the Air (FOTA)

The Firmware Upgrade Over the Air (FOTA) module is a hardware accelerator that is a component of the OptiFlash technology to schedule reads and writes over the same 8 data lines of the OSPI module. The purpose of this hardware accelerator is to minimize XIP downtime.

For more information on FOTA, see Section 13.3.3.7.

8 Interprocessor Communication (IPC)

This chapter describes the interprocessor communication (IPC) modules in the device.

8.1 Mailbox

The device provides a mailbox mechanism to asynchronously exchange the messages between any two processors.

Each processor has a mailbox memory space, and registers designated to be used by other processor that wishes to communicate.

Note

There is an MPU at every Mailbox that can be used to partition the mailbox memory between the Controllers/cores. This gives some flexibility over a fixed allocation scheme.

8.1.1 Mailbox

The device provides a mailbox mechanism to asynchronously exchange the messages between any two processors. Mailbox mechanism is supported across the following processor cores in the device.

Table 8-1. Processor Cores

PROCESSOR NUMBER	PROCESSOR
PROC0	R5FSS0_CORE0
PROC1	R5FSS0_CORE1
PROC2	R5FSS1_CORE0
PROC3	R5FSS1_CORE1
PROC4	ICSS-PRU0
PROC5	ICSS-PRU1
PROC6	HSM

The mailbox mechanism is achieved by using hardware interrupts generated by the controller processor to the target processor. The message payload is placed in shared memory accessible by both the processors.

The device supports two shared memory banks recommended for the purpose of mailbox message payload.

MBOX_SRAM	0x7200 0000	16KB
HSM_MBOX_SRAM	0x4400 0000	2KB

Note

It is not necessary to use the above memory for mailbox. Any shared memory including L2 OCMRAM /TCM can be used for Mailbox payload.

Note

Similar to L2 OCMRAM, there is an MPU in front of MBOX_SRAM that may be used to partition the mailbox memory between the Controllers/cores.

8.1.2 Maibox Message Scheme

The processor which wishes to send a message to another processor writes the message to the mailbox memory space, then interrupts the receiver processor. The receiver processor acknowledges the interrupt, then reads the message from the mailbox memory space. The receiver informs the sender that the message is read by an interrupt, which is acknowledged back by the sender. The sender must not initiate another message to the same receiver until the previously initiated mailbox interaction with the same receiver is complete.

The following sequence is followed for performing a mailbox communication.

1. SENDER Processor writes the message in a shared SRAM space accessible by the RECEIVER Processor.
2. SENDER Processor triggers an interrupt to RECEIVER by writing 1 to <SENDER>_MBOX_WRITE_DONE[RECEIVER].
3. RECEIVER Processor gets a single aggregated interrupt “MBOX_READ_REQ” for all interprocessor communication from all senders.
4. RECEIVER Processor reads the register <RECEIVER>_MBOX_READ_REQ and sees bit [SENDER] is set to 0x1.
5. RECEIVER Processor writes 0x1 to <RECEIVER>_MBOX_READ_REQ[SENDER] to clear the interrupt.
6. RECEIVER Processor reads the message from shared memory.
7. RECEIVER Processor generates an acknowledge interrupt to SENDER Processor by writing 0x1 to RECEIVER>_MBOX_READ_DONE_ACK[SENDER]
8. SENDER Processor gets a single aggregated interrupt “MBOX_READ_DONE” for all interprocessor communication from all Receivers.
9. SENDER Processor reads the register <SENDER>_MBOX_READ_DONE and sees bit [RECEIVER] is 0x1.

10. SENDER Processor writes 0x1 to <SENDER>_MBOX_READ_DONE [RECEIVER] to clear the interrupt.

Note

The mailbox scheme ensures the number of mailbox interrupts to a processor is always only 2, regardless of the number of processors in the SoC. (**MBOX_READ_REQ** and **MBOX_READ_DONE)**

Note

Every processor is always writing to its own designated mailbox registers.

8.1.3 Maibox Message Example

Figure 8-1 shows the steps for sending a mailbox message from R5SS0_CORE0 to R5SS1_CORE1:

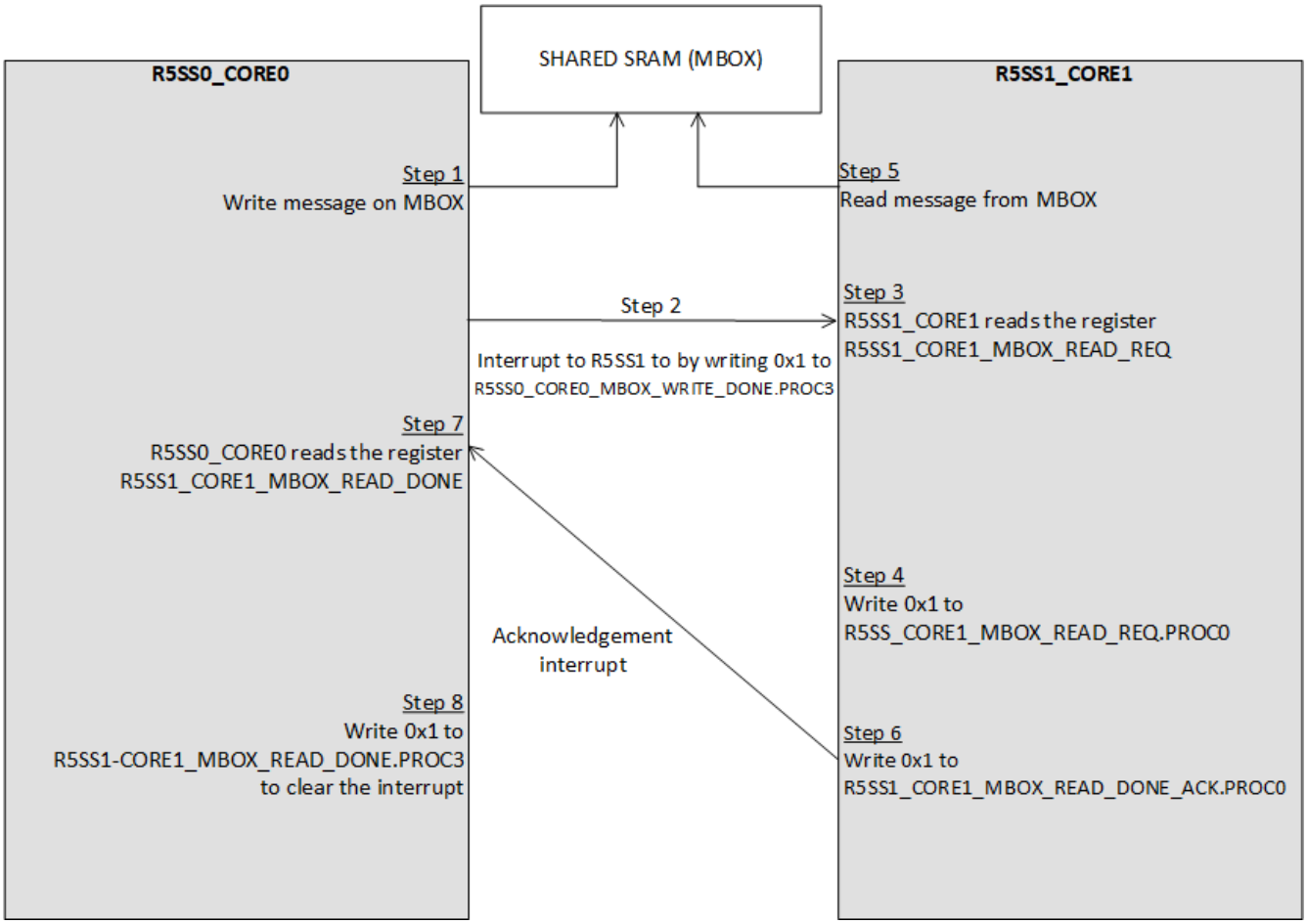


Figure 8-1. Mailbox Message Example

1. R5SS0_CORE0 writes the message in appropriate shared SRAM (Eg: MBOX_SRAM).
2. R5SS0_CORE0 interrupt to R5SS1_CORE1 by writing 1 to R5SS0_CORE0_MBOX_WRITE_DONE.PROC3.
3. R5SS1_CORE1 gets the interrupt MBOX_READ_REQ. R5SS1_CORE1 reads the register R5SS1_CORE1_MBOX_READ_REQ. and sees the bit PROC0 is 0x1.
4. R5SS1_CORE1 writes to 0x1 to R5SS1_CORE1_MBOX_READ_REQ.PROC0 .
5. R5SS1_CORE1 reads the message.
6. R5SS1_CORE1 writes 0x1 to R5SS1_CORE1_MBOX_READ_DONE_ACK.PROC0 to generate an acknowledgment interrupt to R5SS0_CORE0 .
7. R5SS0_CORE0 gets the interrupt MBOX_READ_DONE. R5SS0_CORE0 reads the register R5SS1_CORE1_MBOX_READ_DONE and sees bit PROC3 is 0x1.

8. R5SS0_CORE0 writes 0x1 to R5SS1_CORE1_MBOX_READ_DONE. PROC3 to clear the interrupt.

8.1.4 Maibox Registers

Each processor has registers designated for sending and receiving mailbox events. The registers for R5 Cores and ICSSM Cores are present as part of the MSS_CTRL and registers for HSM M4 is present inside HSM.

8.1.4.1 R5SS0_CORE0 Mailbox Registers

The mailbox registers designated for R5SS0_CORE0 are shown in Table 8-2.

Table 8-2. R5SS0_CORE0 Mailbox Registers

REGISTER	Description
R5SS0_CORE0_MBOX_WRITE_DONE	Sender Write Done Register
R5SS0_CORE0_MBOX_READ_REQ	Receiver Read Request Register
R5SS0_CORE0_MBOX_READ_DONE_ACK	Receiver Read Acknowledge Register
R5SS0_CORE0_MBOX_READ_DONE	Sender Read completed Register

8.1.4.2 R5SS0_CORE1 Mailbox Registers

The mailbox registers designated for R5SS0_CORE1 are shown in Table 8-3.

Table 8-3. R5SS0_CORE1 Mailbox Registers

REGISTER	Description
R5SS0_CORE1_MBOX_WRITE_DONE	Sender Write Done Register
R5SS0_CORE1_MBOX_READ_REQ	Receiver Read Request Register
R5SS0_CORE1_MBOX_READ_DONE_ACK	Receiver Read Acknowledge Register
R5SS0_CORE1_MBOX_READ_DONE	Sender Read completed Register

8.1.4.3 R5SS1_CORE0 Mailbox Registers

The mailbox registers designated for R5SS1_CORE0 is shown in Table 8-4.

Table 8-4. R5SS1_CORE0 Mailbox Registers

REGISTER	Description
R5SS1_CORE0_MBOX_WRITE_DONE	Sender Write Done Register
R5SS1_CORE0_MBOX_READ_REQ	Receiver Read Request Register
R5SS1_CORE0_MBOX_READ_DONE_ACK	Receiver Read Acknowledge Register
R5SS1_CORE0_MBOX_READ_DONE	Sender Read completed Register

8.1.4.4 R5SS1_CORE1 Mailbox Registers

The mailbox registers designated for R5SS1_CORE1 are shown in Table 8-5.

Table 8-5. R5SS1_CORE1 Mailbox Registers

REGISTER	Description
R5SS1_CORE1_MBOX_WRITE_DONE	Sender Write Done Register
R5SS1_CORE1_MBOX_READ_REQ	Receiver Read Request Register
R5SS1_CORE1_MBOX_READ_DONE_ACK	Receiver Read Acknowledge Register
R5SS1_CORE1_MBOX_READ_DONE	Sender Read completed Register

8.1.4.5 ICSSM_PRU0 Mailbox Registers

The mailbox registers designated for ICSSM_PRU0 is shown in Table 8-6.

Table 8-6. ICSSM_PRU0 Mailbox Registers

REGISTER	Description
ICSSM_PRU0_MBOX_WRITE_DONE	Sender Write Done Register
ICSSM_PRU0_MBOX_READ_REQ	Receiver Read Request Register

Table 8-6. ICSSM_PRU0 Mailbox Registers (continued)

REGISTER	Description
ICSSM_PRU0_MBOX_READ_DONE_ACK	Receiver Read Acknowledge Register
ICSSM_PRU0_MBOX_READ_DONE	Sender Read completed Register

8.1.4.6 ICSSM_PRU1 Mailbox Registers

The mailbox registers designated for ICSSM_PRU1 is shown in Table 8-7.

Table 8-7. ICSSM_PRU1 Mailbox Registers

REGISTER	Description
ICSSM_PRU1_MBOX_WRITE_DONE	Sender Write Done Register
ICSSM_PRU1_MBOX_READ_REQ	Receiver Read Request Register
ICSSM_PRU1_MBOX_READ_DONE_ACK	Receiver Read Acknowledge Register
ICSSM_PRU1_MBOX_READ_DONE	Sender Read completed Register

8.1.4.7 HSM Mailbox Registers

The mailbox registers designated for HSM are shown in Table 8-8.

Table 8-8. HSM Mailbox Registers

REGISTER	Description
HSM_MBOX_WRITE_DONE	Sender Write Done Register
HSM_MBOX_READ_REQ	Receiver Read Request Register
HSM_MBOX_READ_DONE_ACK	Receiver Read Acknowledge Register
HSM_MBOX_READ_DONE	Sender Read completed Register

8.2 Spinlock

This chapter describes the Spinlock module of the device.

8.2.1 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.

Figure 8-2 shows an overview of the Spinlock module.

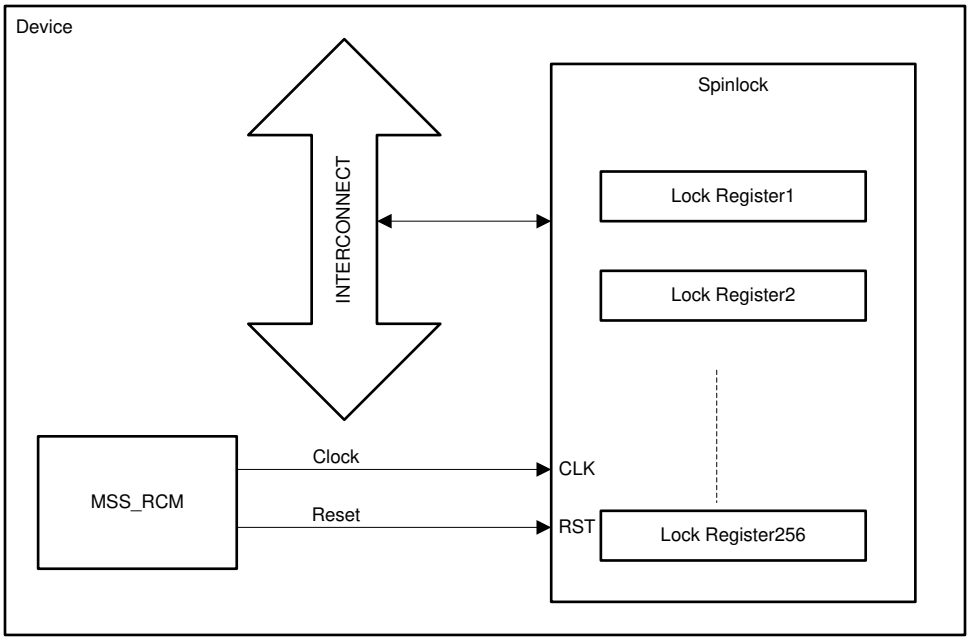


Figure 8-2. Spinlock Overview

8.2.1.1 Spinlock Not Supported Features

The following features are not supported by the module:

- Ownership enforcement. There is no support to ensure that a lock register is locked and unlocked by the same process
- There is no support for checking that the same VBUS initiator that acquired the lock is the one that is freeing the lock
- There is no support for fairness or congestion control.

8.2.2 Spinlock Integration

This section describes module integration in the device, including information about clocks, resets, and hardware requests.

Figure 8-3 shows the integration of the Spinlock module.

Spinlock Integration

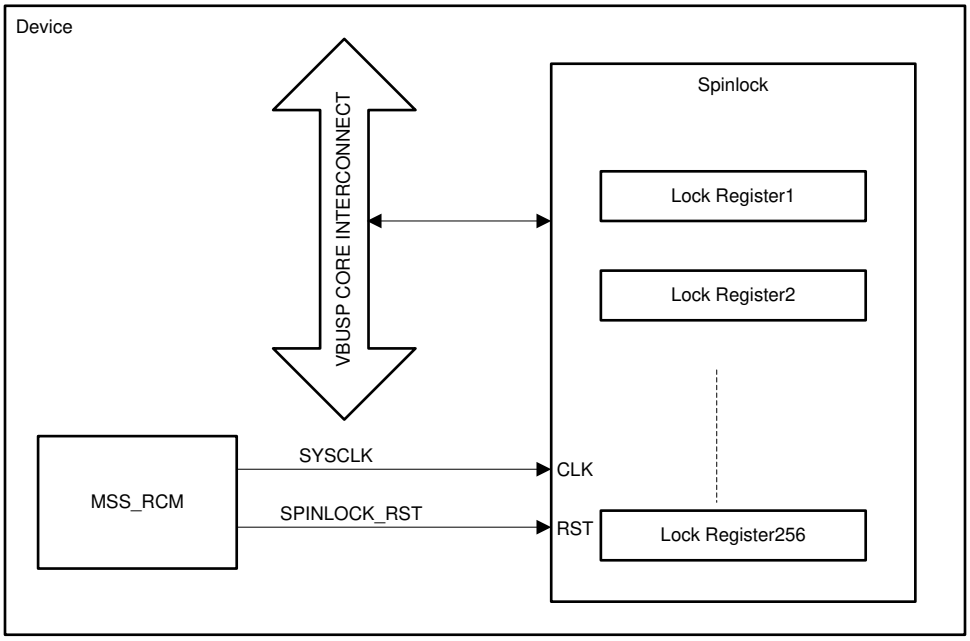


Figure 8-3. SPINLOCK Integration

Table 8-9. SPINLOCK Integration Attributes

Module Instance	SoC Interconnect
SPINLOCK	VBUSP CORE Interconnect

Table 8-10. SPINLOCK Clocks

Module Instance	Module Clock Input	Source Clock Signal	Source	Description
SPINLOCK	CLK	SYSCLK	MSS_RCM	SPINLOCK Functional and Interface clock

Table 8-11. SPINLOCK Resets

Module Instance	Module Reset Input	Source Reset Signal	Source	Description
SPINLOCK	RST	SPINLOCK_RSTN	MSS_RCM	SPINLOCK Reset

Table 8-12. SPINLOCK Interrupt Requests

Module Instance	Module Interrupt Signal	Destination Interrupt Input	Destination	Description	Type
SPINLOCK0	-	-	-	No interrupts to external processors	-

Table 8-13. SPINLOCK DMA Requests

Module Instance	Module DMA Event	Destination DMA Event Input	Destination	Description	Type
SPINLOCK0	-	-	-	No EDMA Requests	-

8.2.3 Spinlock Functional Description

8.2.3.1 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. The SPINLOCK_SYSTATUS[0] RESETDONE bit can be polled to check the reset status (reading 1 indicates that reset sequence is done; reading 0 indicates that reset sequence is in progress). The software must ensure that the software reset completes before doing Spinlock operations.

8.2.3.2 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.

8.2.3.3 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 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 8-4 shows the SPINLOCK_LOCK_REG_y register state diagram.

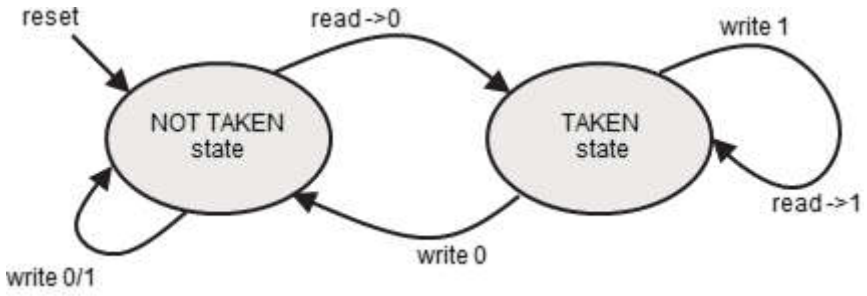


Figure 8-4. Lock Register State Diagram

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.

8.2.4 Spinlock Programming Guide

8.2.4.1 Spinlock Low-level Programming Models

This section covers the low-level hardware programming sequences for configuration and usage of the module.

8.2.4.1.1 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

8.2.4.1.1.1 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 8-14. 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		

8.2.4.1.1.2 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 8-5 shows the described above procedure.

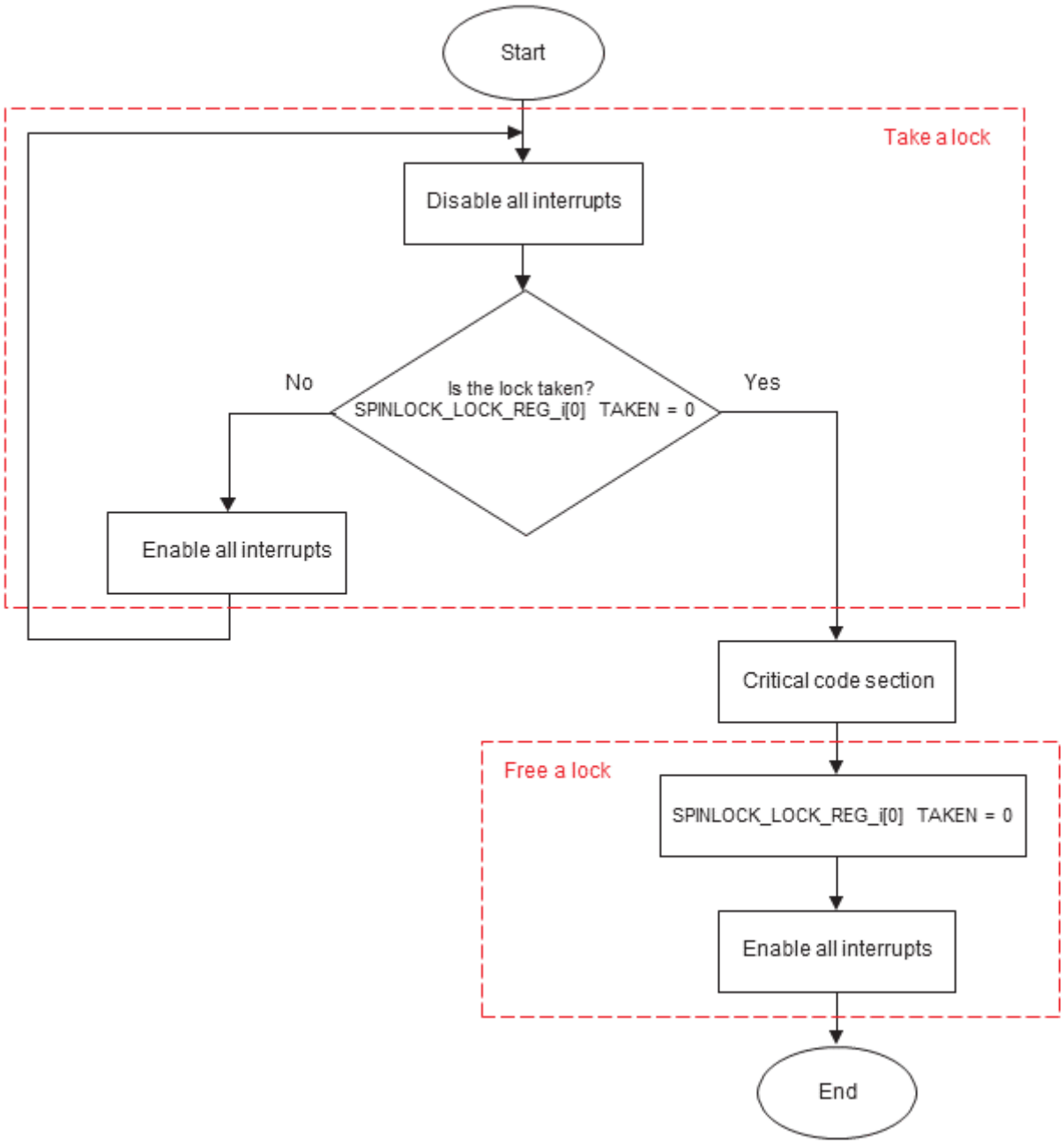


Figure 8-5. Take and Release Spinlock

Table 8-15. Register Call Summary

Register Name
SPINLOCK_LOCK_REG_y[0] TAKEN

Table 8-16. 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	

9 Memory Controllers

This chapter describes the memory controllers in the device.

9.1 Memory Controllers Overview

The AM263Px family of devices utilize an integrated On-Chip Static Random Access Memory (OCSRAM). Controllers for external memory sources such as DDR are not included. The key functionality of the OCSRAM module includes:

- Total of up to 3MB of RAM
  - Can be used as code and data memory by the R5FSS cores
  - Can be used as data buffers accessible by EDMA
- Four 64-bit wide independent banks of size 512KB with 200Mhz operating frequency
- Accessible by all initiator modules via the CORE_VBUSM interconnect as detailed in Section 3.2.
- Protected with MPU firewalls as detailed in MPU Firewalls
- Loadable space for the Secondary Bootloader (SBL) as detailed in Section 5.1
- 64-bit ECC Support
  - Read-Modify-Write mechanism to support ECC update on memory writes less then 64-bits. Read-Modify-Write operation works independently per bank and requires one additional cycle to update.

10 Interrupts

This chapter describes the interrupts in the device.

10.1 Interrupt Architecture

The SoC has many peripherals and a large number of event sources including interrupts, time sync events, and DMA requests. The use of events is completely dependent on a user's specific application, which drives a need for maximum flexibility on which event sources are used in the system. Software control must be used to service these events.

The SoC includes the following interrupt servicing modules (hosts):

- 2x Real-time microcontroller units (R5FSS0, R5FSS1), each supporting:
  - Dual-R5F Cluster
  - Vectored interrupt manager (VIM)
- Hardware Security Module (HSM)
  - Single Arm Cortex-M4F core
  - Nested vectored interrupt controller (NVIC)
- Industrial communications subsystem (PRU-ICSS):
  - Two programmable real-time units (PRUs)
  - Local interrupt controller (INTC)

Most of the system events are routed directly to the various processing 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 (implemented via 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:

- GPIO XBAR Interrupt Router (GPIO_XBAR_INTRTR0):
  - Provides selection of active GPIO[0:3] module interrupts
  - Supported by dedicated device GPIO muxing that provides virtualization
- PRU-ICSS XBAR Interrupt Router (PRU_ICSS_XBAR_INTRTR0):
  - Provides selection of active PRU-ICSS XBAR events for routing as processor interrupts or DMA events
- EDMA Trigger XBAR Interrupt Router (EDMA_XBAR_INTRTR0):
  - Provides selection of DMA Trigger events from various device peripherals

In addition, the following interrupt router instances are part of the SoC time sync architecture:

- Time Sync Event Router0 (SOC_TIMESYNC_XBAR0):
  - See SOC_TIMESYNC_XBAR0 Overview
- Time Sync Event Router1 (CONTROLSS_TIMESYNC_XBAR1):
  - See SOC_TIMESYNC_XBAR1 Overview

10.2 Interrupt Controllers

10.2.1 Vectored Interrupt Manager (VIM)

10.2.1.1 VIM Overview

The VIM aggregates device interrupts and sends them to the R5F CPU(s). It can be used in either split or single-core 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:

- 256 interrupt inputs per R5F core
- Each interrupt has its own 4-bit programmable priority
  - Defined via the VIM_INTPRIORITY register
  - The VIM provides support for priority interruption of interrupts
- Each interrupt has its own enable mask

  - Interrupt enable is done via the MSS_VIM_INTR_EN_SET_j register
  -
  - Interrupt disable is done via the MSS_VIM_INTR_EN_CLR_j register
- Each interrupt can be programmed as either an IRQ or FIQ
  - Defined via the MSS_VIM_INTMAP_j register
- Each interrupt has its own programmable 32-bit vector address associated with it
  - Defined via the MSS_VIM_INTVECTOR 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 single-core capable
  - In single-core mode, only interrupts connected to VIM interrupt core 0 are available
- Software interrupt generation

10.2.1.2 VIM Interrupt Inputs

The VIM supports 256 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*.

10.2.1.3 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 MSS_VIM_INTMAP_j register) and is enabled (via the MSS_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

10.2.1.4 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 (MSS_VIM_INTVECTOR). Hence, the interrupt vector table is organized in 256 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 10-1 shows the VIM RAM interrupt vector map.

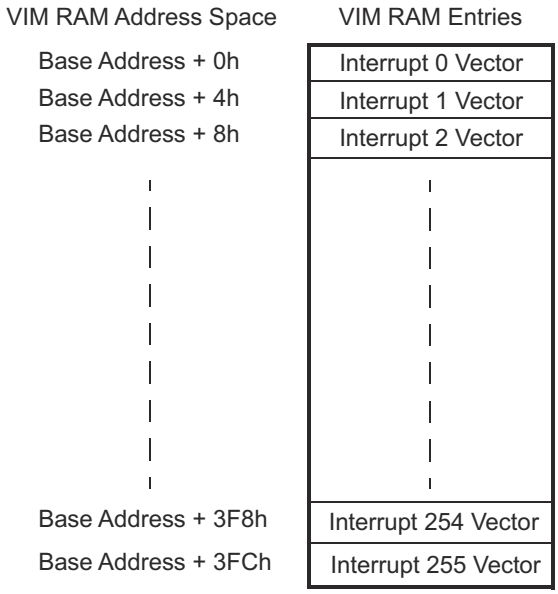


Figure 10-1. VIM RAM Interrupt Vector Map

The interrupt vector table has protection by ECC to indicate corruption due to soft errors. The ECC logic inside VIM supports SECDED. See Table 7-8 for the VIM RAM ID in the ECC aggregator map.

10.2.1.5 VIM Interrupt Prioritization

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.

Note

VIM priority scheme: 00 = Highest Priority - 15 = Lowest Priority

When an interrupt goes from pending to active (FIQ: reading the MSS_VIM_FIQVEC register; IRQ: reading the MSS_VIM_IRQVEC register, or the *coreN_IRQACK* going high), then the interrupt is loaded into the corresponding active register (MSS_VIM_ACTFIQ / MSS_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 MSS_VIM_FIQVEC register, or MSS_VIM_IRQVEC register) another interrupt of higher priority arrives, then the FIQn/IRQn is asserted and that interrupt made pending as normal. If the CPU switches this interrupt to active (by reading the MSS_VIM_FIQVEC / MSS_VIM_IRQVEC register), then the currently active interrupt is pushed onto a stack. When an interrupt is cleared by reading the MSS_VIM_FIQVEC / MSS_VIM_IRQVEC register, if there are any interrupts on the stack, the first entry is popped off and put back into the MSS_VIM_ACTFIQ / MSS_VIM_ACTIRQ register, so that software retains original context and continues previous operation.

Note

"Masked off" means that the registers are masked off from priority arbitration to interrupt the currently active interrupt, this does *not* mean that the status bits in the registers are masked off. That is, this priority masking has NO EFFECT on whether the status bits are visible in the masked registers such as the Group M Interrupt Enabled Status/Clear Register.

10.2.1.6 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 MSS_VIM_DEDVEC register is used to provide the default vector for the *coreN_IRQADDRV* signal, the MSS_VIM_IRQVEC register, and the MSS_VIM_FIQVEC register. The MSS_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 (that is, for masking purposes). 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. When in lockstep mode, only the RAM for CPU0 is used.

10.2.1.7 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.

10.2.1.8 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 10.2.1.8.1, but the procedures in Section 10.2.1.8.2 or Section 10.2.1.8.3 (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 10.2.1.8.4, but the procedure in Section 10.2.1.8.5 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.

10.2.1.8.1 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 MSS_VIM_PRIIRQ[9:0] NUM bit field (which corresponds to the vector address) into the MSS_VIM_ACTIRQ[9:0] NUM bit field, which causes the MSS_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 MSS_VIM_ACTIRQ[9:0] NUM bit field to determine number, and reading the appropriate bit in the MSS_VIM_INTTYPE_j register to determine type)
   a. Pulse
      i. Clear the status by writing a '1' to the appropriate bit in the MSS_VIM_IRQSTS_j register, or MSS_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 MSS_VIM_IRQSTS_j register, or MSS_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 MSS_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 MSS_VIM_ACTIRQ[31] VALID bit

10.2.1.8.2 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 MSS_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 MSS_VIM_PRIIRQ[9:0] NUM bit field to be loaded into the MSS_VIM_ACTIRQ[9:0] NUM bit field, and the MSS_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 MSS_VIM_STS_j register, or MSS_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 MSS_VIM_STS_j register, or MSS_VIM_IRQSTS_j register
4. Write any value to the MSS_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 MSS_VIM_ACTIRQ[31] VALID bit

10.2.1.8.3 Servicing IRQ Through MMR Interface (Alternative)

If a user does not want to use the MSS_VIM_IRQVEC 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 MSS_VIM_IRQVEC register is never read). Software would be responsible for doing all priority operations.

1. Determine which interrupt to service
   a. Read the MSS_VIM_PRIIRQ register to determine which interrupt is the highest priority IRQ currently asserted, OR
   b. Optionally read the MSS_VIM_IRQGSTS register to determine which groups have IRQs pending, then read the MSS_VIM_IRQSTS_j register and use a software prioritization scheme to determine which IRQ 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 MSS_VIM_STS_j register, or MSS_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 MSS_VIM_STS_j register, or MSS_VIM_IRQSTS_j register

10.2.1.8.4 Servicing FIQ

When an FIQ interrupt is received, the CPU should follow these steps:

1. Read the MSS_VIM_FIQVEC 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_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 MSS_VIM_PRIFIQ[9:0] NUM bit field to be loaded into the MSS_VIM_ACTFIQ[9:0] NUM bit field, and the MSS_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 MSS_VIM_ACTFIQ[9:0] NUM bit field to determine number, and reading the appropriate bit in the MSS_VIM_INTTYPE_j register to determine type)
   a. Pulse
      i. Clear the status by writing a '1' to the appropriate bit in the MSS_VIM_STS_j register, or MSS_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 MSS_VIM_STS_j register, or MSS_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. This will also clear the MSS_VIM_ACTFIQ[31] VALID bit
   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 MSS_VIM_ACTFIQ[31] VALID bit

10.2.1.8.5 Servicing FIQ (Alternative)

If a user does not want to use the MSS_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 MSS_VIM_FIQVEC register is never read). Software would be responsible for doing all priority operations.

1. Determine which interrupt to service
   a. Read the MSS_VIM_PRIFIQ register to determine which interrupt is the highest priority FIQ currently asserted, OR
   b. Optionally read the MSS_VIM_FIQGSTS register to determine which groups have IRQs pending, then read the MSS_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 MSS_VIM_STS_j register, or MSS_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 MSS_VIM_STS_j register, or MSS_VIM_FIQSTS_j register.

10.2.2 Other Interrupt Controllers

All other device interrupt controllers are described in their respective chapters and/or third party documentation.

10.3 Interrupt Routers

10.3.1 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 Section 10.1, *Interrupt Architecture*. Table 10-1 summarizes the configuration details for the various interrupt routers.

Table 10-1. INTRTR Modules Configuration

Module	Number of Inputs	Number of Outputs	Interrupt Type
GPIO_XBAR_INTRTR0	180	30[1]	Pulse
PRU_ICSS_XBAR_INTRTR0	74	16	Pulse
EDMA_XBAR_INTRTR0	260	64	Pulse
SOC_TIMESYNC_XBAR0	16	26	Pulse
SOC_TIMESYNC_XBAR1	28	20[1]	Pulse
CONTROLSS_INTXBAR	186	32	Pulse

[1] - Only 4 outputs from GPIO_XBAR_INTRTR0 & SOC_TIMESYNC_XBAR1 connect to each VIM[3:0] instance

CONTROLSS_INTXBAR is described in the CONTROLSS chapter and SOC_TIMESYNC_XBAR0, SOC_TIMESYNC_XBAR1 are captured in TimeSync Event Sources.

The user should take the following into account when programming the MUXCNTL_N register:

- Avoid programming this register when input interrupts are active. This can lead to spurious asynchronous output toggles which can lead to unpredictable behavior.
- All mux control settings default to '0', which means that at reset no input interrupt will be propagated to any of the INTRTR outputs. This is due to the fact that the 0th input of all internal muxes is unused in the current implementation

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.

10.3.2 INTRTR Integration

This section describes the INTRTR integration in the device, including information about clocks, resets, and hardware requests.

10.3.2.1 PRU-ICSS XBAR INTRTR0

There is 1x PRU-ICSS XBAR Interrupt Router module integrated in the device. The diagram below provides a visual representation of the device integration details for the PRU-ICSS XBAR Interrupt Router.

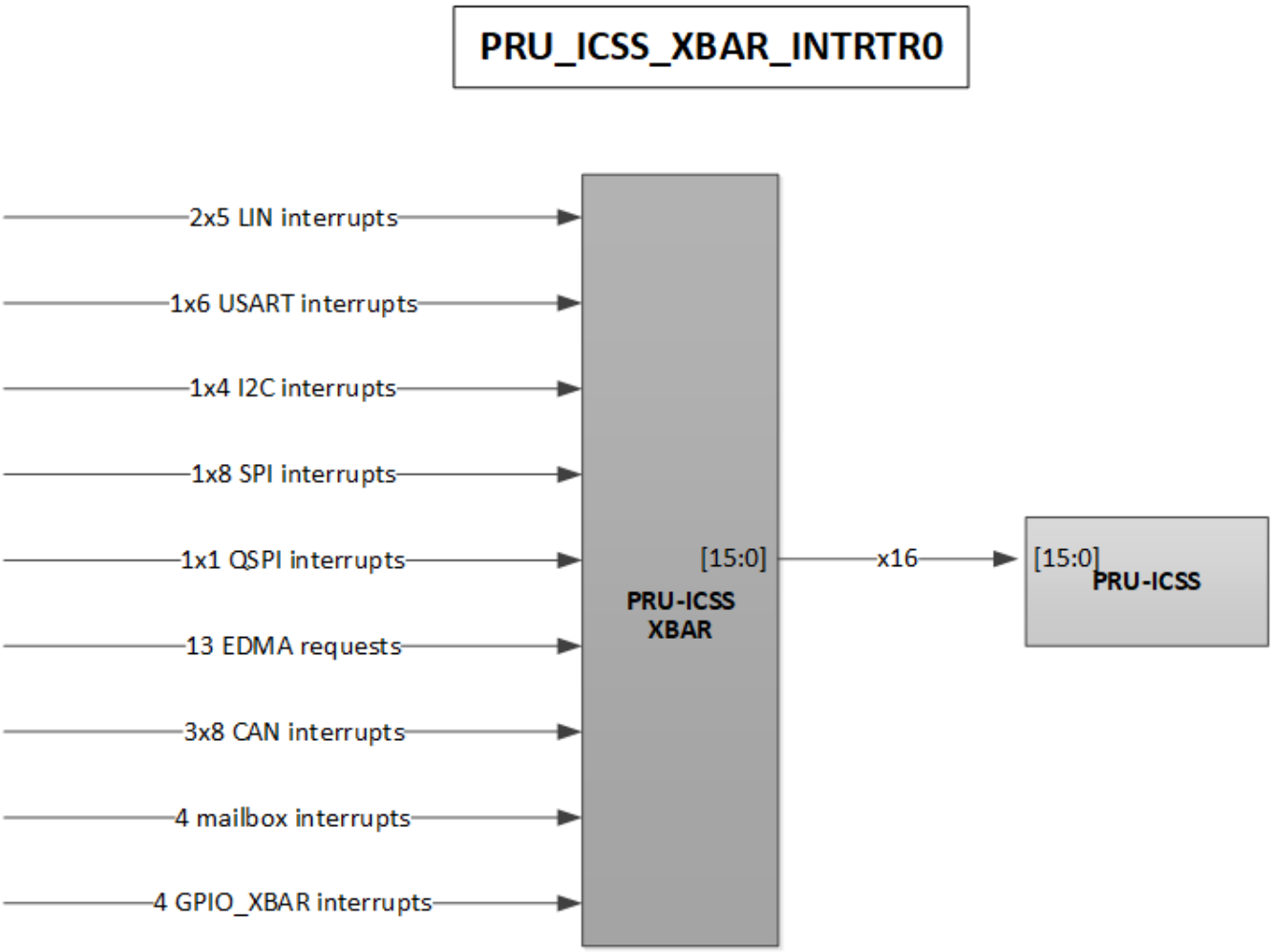


Figure 10-2. PRU-ICSS XBAR Interrupt Router Integration Diagram

The tables below summarize the device integration details of PRU-ICSS XBAR Interrupt router.

Table 10-2. PRU-ICSS XBAR Interrupt router Device Integration

Module Instance	Device Allocation	SoC Interconnect
PRU_ICSS_XBAR_INTRTR0	✓	INFRA0 VBUSP Interconnect

Table 10-3. PRU-ICSS_XBAR_INTRTR0 Clocks

Module Instance	Module Clock in_intr	Source Clock Signal	Source	Default Freq	Description
PRU_ICSS_XBAR_INTRTR0	SYSCLK	SYS_CLK	MSS_RCM	200 MHz	PRU_ICSS_XBAR_INTRTR0 Fnctional and Interface clock

Table 10-4. PRU-ICSS_XBAR_INTRTR0 Resets

Module Instance	Module Reset in_intr	Source Reset Signal	Source	Description
PRU_ICSS_XBAR_INTRTR0	RST	SYS_RST	MSS_RCM	PRU_ICSS_XBAR_INTRTR0 Reset

Table 10-5. PRU-ICSS_XBAR_INTRTR0 Output Hardware Requests

Module Instance	Module XBAR Output	Destination XBAR signal	Destination	Type	Description
PRU_ICSS_XBAR_INTRTR0	outl_intr_0	PR1_SLV_INTR_0	PRU-ICSS	Pulse	Selectable Hardware Request 0
	outl_intr_1	PR1_SLV_INTR_1			Selectable Hardware Request 1
	outl_intr_2	PR1_SLV_INTR_2			Selectable Hardware Request 2
	outl_intr_3	PR1_SLV_INTR_3			Selectable Hardware Request 3
	outl_intr_4	PR1_SLV_INTR_4			Selectable Hardware Request 4
	outl_intr_5	PR1_SLV_INTR_5			Selectable Hardware Request 5
	outl_intr_6	PR1_SLV_INTR_6			Selectable Hardware Request 6
	outl_intr_7	PR1_SLV_INTR_7			Selectable Hardware Request 7
	outl_intr_8	PR1_SLV_INTR_8			Selectable Hardware Request 8
	outl_intr_9	PR1_SLV_INTR_9			Selectable Hardware Request 9
	outl_intr_10	PR1_SLV_INTR_10			Selectable Hardware Request 10
	outl_intr_11	PR1_SLV_INTR_11			Selectable Hardware Request 11
	outl_intr_12	PR1_SLV_INTR_12			Selectable Hardware Request 12
	outl_intr_13	PR1_SLV_INTR_13			Selectable Hardware Request 13
	outl_intr_14	PR1_SLV_INTR_14			Selectable Hardware Request 14
	outl_intr_15	PR1_SLV_INTR_15			Selectable Hardware Request 15

Table 10-6. PRU_ICSS_XBAR_INTRTR0 in_intr Hardware Requests

Module Instance	Source Module	Source in_intr signal	XBAR Module in_intr	Type	Description	
PRU_ICSS_XBAR_INTRTR0	LIN0	lin0_intr_req0	IN_INTR0	Level	LIN0 Interrupt Request 0	
	LIN0	lin0_intr_req1	IN_INTR1	Level	LIN0 Interrupt Request 1	
	LIN1	lin1_intr_req0	IN_INTR2	Level	LIN1 Interrupt Request 0	
	LIN1	lin1_intr_req1	IN_INTR3	Level	LIN1 Interrupt Request 1	
	LIN2	lin2_intr_req0	IN_INTR4	Level	LIN2 Interrupt Request 0	
	LIN2	lin2_intr_req1	IN_INTR5	Level	LIN2 Interrupt Request 1	
	LIN3	lin3_intr_req0	IN_INTR6	Level	LIN3 Interrupt Request 0	
	LIN3	lin3_intr_req1	IN_INTR7	Level	LIN3 Interrupt Request 1	
	LIN4	lin4_intr_req0	IN_INTR8	Level	LIN4 Interrupt Request 0	
	LIN4	lin4_intr_req1	IN_INTR9	Level	LIN4 Interrupt Request 1	
	UART0	uart0_irq	IN_INTR10	Level	UART0 Interrupt	
	UART1	uart1_irq	IN_INTR11	Level	UART1 Interrupt	
	UART2	uart2_irq	IN_INTR12	Level	UART2 Interrupt	
	UART3	uart3_irq	IN_INTR13	Level	UART3 Interrupt	
	UART4	uart4_irq	IN_INTR14	Level	UART4 Interrupt	
	UART5	uart5_irq	IN_INTR15	Level	UART5 Interrupt	
	I2C0	I2C0_IRQ	IN_INTR16	Pulse	I2C0 Interrupt	
	I2C1	I2C1_IRQ	IN_INTR17	Pulse	I2C1 Interrupt	
	I2C2	I2C2_IRQ	IN_INTR18	Pulse	I2C2 Interrupt	
	I2C3	I2C3_IRQ	IN_INTR19	Pulse	I2C3 Interrupt	
	SPI0	SPI0_intr	IN_INTR20	Level	SPI0 Interrupt	
	SPI1	SPI1_intr	IN_INTR21	Level	SPI1 Interrupt	
	SPI2	SPI2_intr	IN_INTR22	Level	SPI2 Interrupt	
	SPI3	SPI3_intr	IN_INTR23	Level	SPI3 Interrupt	
	SPI4	SPI4_intr	IN_INTR24	Level	SPI4 Interrupt	
	OSPI	OSPI_intr	IN_INTR25	Level	OSPI Interrupt	
	SOC_EDMA0	TPCC_intg	IN_INTR26	Pulse	TPCC Global Interrupt	
	SOC_EDMA0	TPCC_int0	IN_INTR27	Pulse	TPCC Region0 Interrupt	
	SOC_EDMA0	TPCC_int1	IN_INTR28	Pulse	TPCC Region1 Interrupt	
	SOC_EDMA0	TPCC_int2	IN_INTR29	Pulse	TPCC Region2 Interrupt	
	SOC_EDMA0	TPCC_int3	IN_INTR30	Pulse	TPCC Region3 Interrupt	
	SOC_EDMA0	TPCC_int4	IN_INTR31	Pulse	TPCC Region4 Interrupt	
	SOC_EDMA0	TPCC_int5	IN_INTR32	Pulse	TPCC Region5 Interrupt	
	SOC_EDMA0	TPCC_int6	IN_INTR33	Pulse	TPCC Region6 Interrupt	
	SOC_EDMA0	TPCC_int7	IN_INTR34	Pulse	TPCC Region7 Interrupt	
	SOC_EDMA0	TPCC_errint	IN_INTR35	Pulse	TPCC Error Interrupt	
	SOC_EDMA0	tpcc_mpint	IN_INTR36	Pulse	TPCC Memory Protection Violation Interrupt	
	SOC_EDMA0	tptc_erint0	IN_INTR37	Pulse	TPCC Interrupt	
	SOC_EDMA0	tptc_erint1	IN_INTR38	Pulse	TPCC Interrupt	
		MCAN0	mcanss0_ext_ts_rollover_lvl_int	IN_INTR39	Level	MCAN0 External TimeSync Rollover Interrupt
MCAN0		mcanss0_mcan_lvl_int_0	IN_INTR40	Level	MCAN0 Interrupt 0	
MCAN0		mcanss0_mcan_lvl_int_1	IN_INTR41	Level	MCAN0 Interrupt 1	
MCAN1		mcanss1_ext_ts_rollover_lvl_int	IN_INTR42	Level	MCAN1 External TimeSync Rollover Interrupt	
MCAN1		mcanss1_mcan_lvl_int_0	IN_INTR43	Level	MCAN1 Interrupt 0	

10.3.2.2 EDMA XBAR INTRTR0

There is 1x EDMA XBAR Interrupt Router module integrated in the device. The diagram below provides a visual representation of the device integration details for EDMA XBAR Interrupt Router.

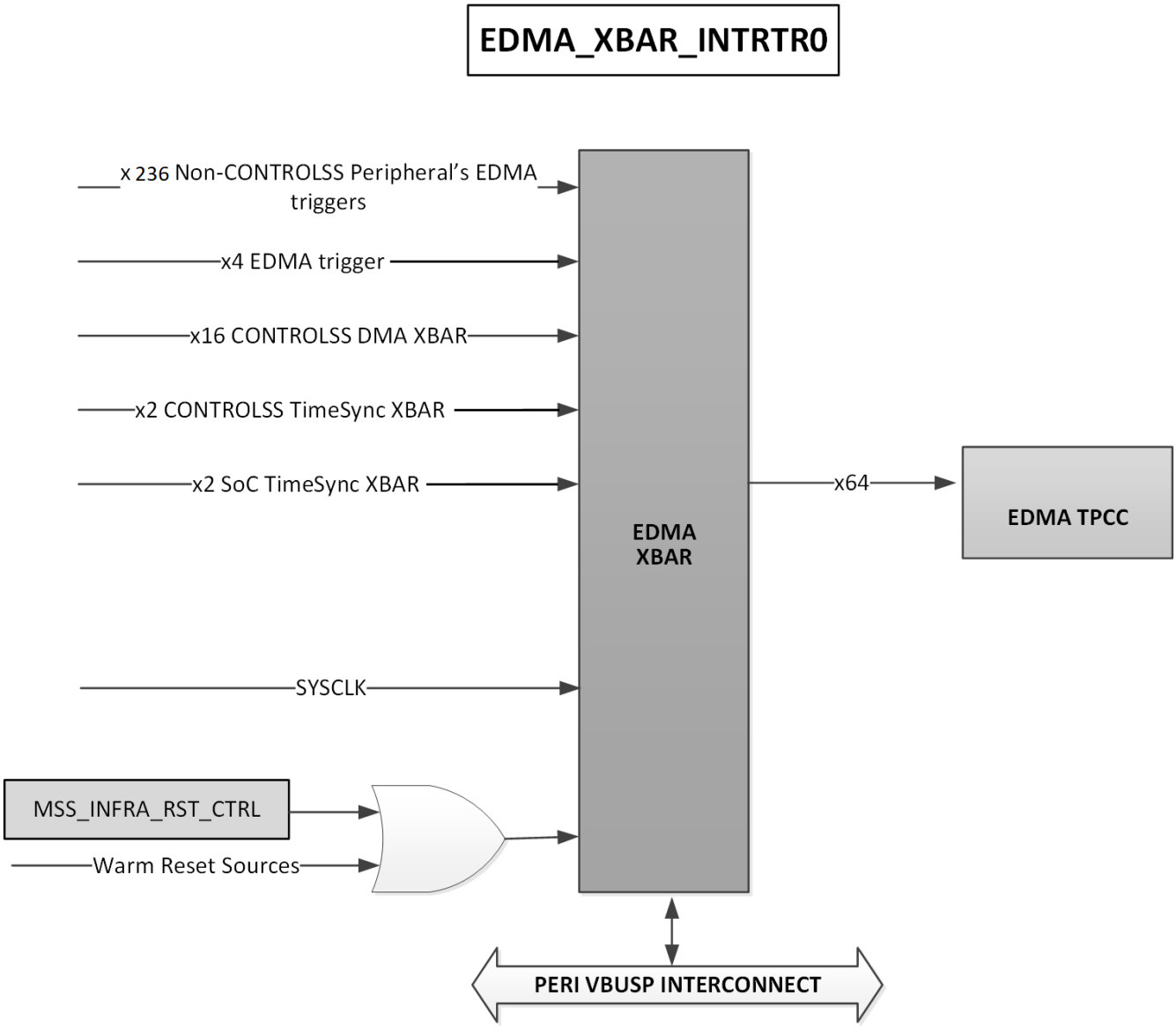


Figure 10-3. EDMA XBAR Interrupt Router Integration Diagram

The tables below summarize the device integration details of EDMA XBAR Interrupt router.

Table 10-7. EDMA XBAR Intrrupt router Device Integration

Module Instance	Device Allocation	SoC Interconnect
EDMA_XBAR_INTRTR0	✓	INFRA0 VBUSP Interconnect

Table 10-8. EDMA_XBAR_INTRTR0 Clocks

Module Instance	Module Clock in_intr	Source Clock Signal	Source	Default Freq	Description
EDMA_XBAR_INTRTR0	SYSCLK	SYS_CLK	MSS_RCM	200 MHz	EDMA_XBAR_INTRTR0 Functional and Interface clock

Table 10-9. EDMA_XBAR_INTRTR0 Resets

Module Instance	Module Reset in_intr	Source Reset Signal	Source	Description
EDMA_XBAR_INTRTR0	RST	SYS_RST	MSS_RCM	EDMA_XBAR_INTRTR0 Reset

Table 10-10. EDMA_XBAR_INTRTR0 Output Hardware Requests

Module Instance	Module XBAR Output	Destination XBAR signal	Destination	Description	Type
EDMA_XBAR_INTRTR0	outl_intr_0	EDMA_Trigger_XBAROut_0	TPCC	Selectable Hardware Request 0	Pulse
	outl_intr_1	EDMA_Trigger_XBAROut_1		Selectable Hardware Request 1	
	outl_intr_2	EDMA_Trigger_XBAROut_2		Selectable Hardware Request 2	
	outl_intr_3	EDMA_Trigger_XBAROut_3		Selectable Hardware Request 3	
	outl_intr_4	EDMA_Trigger_XBAROut_4		Selectable Hardware Request 4	
	outl_intr_5	EDMA_Trigger_XBAROut_5		Selectable Hardware Request 5	
	outl_intr_6	EDMA_Trigger_XBAROut_6		Selectable Hardware Request 6	
	outl_intr_7	EDMA_Trigger_XBAROut_7		Selectable Hardware Request 7	
	outl_intr_8	EDMA_Trigger_XBAROut_8		Selectable Hardware Request 8	
	outl_intr_9	EDMA_Trigger_XBAROut_9		Selectable Hardware Request 9	
	outl_intr_10	EDMA_Trigger_XBAROut_10		Selectable Hardware Request 10	
	outl_intr_11	EDMA_Trigger_XBAROut_11		Selectable Hardware Request 11	
	outl_intr_12	EDMA_Trigger_XBAROut_12		Selectable Hardware Request 12	
	outl_intr_13	EDMA_Trigger_XBAROut_13		Selectable Hardware Request 13	
	outl_intr_14	EDMA_Trigger_XBAROut_14		Selectable Hardware Request 14	
	outl_intr_15	EDMA_Trigger_XBAROut_15		Selectable Hardware Request 15	
	outl_intr_16	EDMA_Trigger_XBAROut_16		Selectable Hardware Request 16	
	outl_intr_17	EDMA_Trigger_XBAROut_17		Selectable Hardware Request 17	
	outl_intr_18	EDMA_Trigger_XBAROut_18		Selectable Hardware Request 18	
	outl_intr_19	EDMA_Trigger_XBAROut_19		Selectable Hardware Request 19	
	outl_intr_20	EDMA_Trigger_XBAROut_20		Selectable Hardware Request 20	
	outl_intr_21	EDMA_Trigger_XBAROut_21		Selectable Hardware Request 21	
	outl_intr_22	EDMA_Trigger_XBAROut_22		Selectable Hardware Request 22	
	outl_intr_23	EDMA_Trigger_XBAROut_23		Selectable Hardware Request 23	
	outl_intr_24	EDMA_Trigger_XBAROut_24		Selectable Hardware Request 24	
	outl_intr_25	EDMA_Trigger_XBAROut_25		Selectable Hardware Request 25	
	outl_intr_26	EDMA_Trigger_XBAROut_26		Selectable Hardware Request 26	
	outl_intr_27	EDMA_Trigger_XBAROut_27		Selectable Hardware Request 27	
	outl_intr_28	EDMA_Trigger_XBAROut_28		Selectable Hardware Request 28	
	outl_intr_29	EDMA_Trigger_XBAROut_29		Selectable Hardware Request 29	
	outl_intr_30	EDMA_Trigger_XBAROut_30		Selectable Hardware Request 30	
	outl_intr_31	EDMA_Trigger_XBAROut_31		Selectable Hardware Request 31	
	outl_intr_32	EDMA_Trigger_XBAROut_32		Selectable Hardware Request 32	
	outl_intr_33	EDMA_Trigger_XBAROut_33		Selectable Hardware Request 33	
	outl_intr_34	EDMA_Trigger_XBAROut_34		Selectable Hardware Request 34	
	outl_intr_35	EDMA_Trigger_XBAROut_35		Selectable Hardware Request 35	
	outl_intr_36	EDMA_Trigger_XBAROut_36		Selectable Hardware Request 36	
	outl_intr_37	EDMA_Trigger_XBAROut_37		Selectable Hardware Request 37	
	outl_intr_38	EDMA_Trigger_XBAROut_38		Selectable Hardware Request 38	
	outl_intr_39	EDMA_Trigger_XBAROut_39		Selectable Hardware Request 39	
	outl_intr_40	EDMA_Trigger_XBAROut_40		Selectable Hardware Request 40	
	outl_intr_41	EDMA_Trigger_XBAROut_41		Selectable Hardware Request 41	
	outl_intr_42	EDMA_Trigger_XBAROut_42		Selectable Hardware Request 42	

Table 10-10. EDMA_XBAR_INTRTR0 Output Hardware Requests (continued)

Module Instance	Module XBAR Output	Destination XBAR signal	Destination	Description	Type
	outl_intr_43	EDMA_Trigger_XBAROut_43		Selectable Hardware Request 43	
	outl_intr_44	EDMA_Trigger_XBAROut_44		Selectable Hardware Request 44	
	outl_intr_45	EDMA_Trigger_XBAROut_45		Selectable Hardware Request 45	
	outl_intr_46	EDMA_Trigger_XBAROut_46		Selectable Hardware Request 46	
	outl_intr_47	EDMA_Trigger_XBAROut_47		Selectable Hardware Request 47	
	outl_intr_48	EDMA_Trigger_XBAROut_48		Selectable Hardware Request 48	
	outl_intr_49	EDMA_Trigger_XBAROut_49		Selectable Hardware Request 49	
	outl_intr_50	EDMA_Trigger_XBAROut_50		Selectable Hardware Request 50	
	outl_intr_51	EDMA_Trigger_XBAROut_51		Selectable Hardware Request 51	
	outl_intr_52	EDMA_Trigger_XBAROut_52		Selectable Hardware Request 52	
	outl_intr_53	EDMA_Trigger_XBAROut_53		Selectable Hardware Request 53	
	outl_intr_54	EDMA_Trigger_XBAROut_54		Selectable Hardware Request 54	
	outl_intr_55	EDMA_Trigger_XBAROut_55		Selectable Hardware Request 55	
	outl_intr_56	EDMA_Trigger_XBAROut_56		Selectable Hardware Request 56	
	outl_intr_57	EDMA_Trigger_XBAROut_57		Selectable Hardware Request 57	
	outl_intr_58	EDMA_Trigger_XBAROut_58		Selectable Hardware Request 58	
	outl_intr_59	EDMA_Trigger_XBAROut_59		Selectable Hardware Request 59	
	outl_intr_60	EDMA_Trigger_XBAROut_60		Selectable Hardware Request 60	
	outl_intr_61	EDMA_Trigger_XBAROut_61		Selectable Hardware Request 61	
	outl_intr_62	EDMA_Trigger_XBAROut_62		Selectable Hardware Request 62	
	outl_intr_63	EDMA_Trigger_XBAROut_63		Selectable Hardware Request 63	

Table 10-11. EDMA_XBAR_INTRTR0 in_intr Hardware Requests

Module Instance	Source Module	Source in_intr signal	XBAR Module in_intr	Desc ription	Type
EDMA_XBAR_INTRTR0	LIN0	lin0_RXDMA	IN_INTR0	LIN0 RX DMA Request	Pulse
	LIN0	lin0_TXDMA	IN_INTR1	LIN0 TX DMA Request	Pulse
	LIN1	lin1_RXDMA	IN_INTR2	LIN1 RX DMA Request	Pulse
	LIN1	lin0_TXDMA	IN_INTR3	LIN1 TX DMA Request	Pulse
	LIN2	lin2_RXDMA	IN_INTR4	LIN2 RX DMA Request	Pulse
	LIN2	lin2_TXDMA	IN_INTR5	LIN2 TX DMA Request	Pulse
	LIN3	lin3_RXDMA	IN_INTR6	LIN3 RX DMA Request	Pulse
	LIN3	lin3_TXDMA	IN_INTR7	LIN3 TX DMA Request	Pulse
	LIN4	lin4_RXDMA	IN_INTR8	LIN4 RX DMA Request	Pulse
	LIN4	lin4_TXDMA	IN_INTR9	LIN4 TX DMA Request	Pulse
	I2C0	I2C0_TX	IN_INTR10	I2C0 RX DMA Request	Pulse
	I2C0	I2C0_RX	IN_INTR11	I2C0 TX DMA Request	Pulse
	I2C1	I2C1_TX	IN_INTR12	I2C1 RX DMA Request	Pulse
	I2C1	I2C1_RX	IN_INTR13	I2C1 TX DMA Request	Pulse
	I2C2	I2C2_TX	IN_INTR14	I2C2 RX DMA Request	Pulse
	I2C2	I2C2_RX	IN_INTR15	I2C2 TX DMA Request	Pulse
	I2C3	I2C3_TX	IN_INTR16	I2C3 RX DMA Request	Pulse
	I2C3	I2C3_RX	IN_INTR17	I2C3 TX DMA Request	Pulse
	SPI0	SPI0_dma_Read_req0	IN_INTR18	SPI0 DMA Read Request 0	Pulse
	SPI0	SPI0_dma_Read_req1	IN_INTR19	SPI0 DMA Read Request 1	Pulse
	SPI0	SPI0_dma_Read_req2	IN_INTR20	SPI0 DMA Read Request 2	Pulse

Table 10-11. EDMA_XBAR_INTRTR0 in_intr Hardware Requests (continued)

Module Instance	Source Module	Source in_intr signal	XBAR Module in_intr	Desc ription	Type
	SPI0	SPI0_dma_Read_req3	IN_INTR21	SPI0 DMA Read Request 3	Pulse
	SPI0	SPI0_dma_Write_req0	IN_INTR22	SPI0 DMA Write Request 0	Pulse
	SPI0	SPI0_dma_Write_req1	IN_INTR23	SPI0 DMA Write Request 1	Pulse
	SPI0	SPI0_dma_Write_req2	IN_INTR24	SPI0 DMA Write Request 2	Pulse
	SPI0	SPI0_dma_Write_req3	IN_INTR25	SPI0 DMA Write Request 3	Pulse
	SPI1	SPI1_dma_Read_req0	IN_INTR26	SPI1 DMA Read Request 0	Pulse
	SPI1	SPI1_dma_Read_req1	IN_INTR27	SPI1 DMA Read Request 1	Pulse
	SPI1	SPI1_dma_Read_req2	IN_INTR28	SPI1 DMA Read Request 2	Pulse
	SPI1	SPI1_dma_Read_req3	IN_INTR29	SPI1 DMA Read Request 3	Pulse
	SPI1	SPI1_dma_Write_req0	IN_INTR30	SPI1 DMA Write Request 0	Pulse
	SPI1	SPI1_dma_Write_req1	IN_INTR31	SPI1 DMA Write Request 1	Pulse
	SPI1	SPI1_dma_Write_req2	IN_INTR32	SPI1 DMA Write Request 2	Pulse
	SPI1	SPI1_dma_Write_req3	IN_INTR33	SPI1 DMA Write Request 3	Pulse
	SPI2	SPI2_dma_Read_req0	IN_INTR34	SPI2 DMA Read Request 0	Pulse
	SPI2	SPI2_dma_Read_req1	IN_INTR35	SPI2 DMA Read Request 1	Pulse
	SPI2	SPI2_dma_Read_req2	IN_INTR36	SPI2 DMA Read Request 2	Pulse
	SPI2	SPI2_dma_Read_req3	IN_INTR37	SPI2 DMA Read Request 3	Pulse
	SPI2	SPI2_dma_Write_req0	IN_INTR38	SPI2 DMA Write Request 0	Pulse
	SPI2	SPI2_dma_Write_req1	IN_INTR39	SPI2 DMA Write Request 1	Pulse
	SPI2	SPI2_dma_Write_req2	IN_INTR40	SPI2 DMA Write Request 2	Pulse
	SPI2	SPI2_dma_Write_req3	IN_INTR41	SPI2 DMA Write Request 3	Pulse
	SPI3	SPI3_dma_Read_req0	IN_INTR42	SPI3 DMA Read Request 0	Pulse
	SPI3	SPI3_dma_Read_req1	IN_INTR43	SPI3 DMA Read Request 1	Pulse
	SPI3	SPI3_dma_Read_req2	IN_INTR44	SPI3 DMA Read Request 2	Pulse
	SPI3	SPI3_dma_Read_req3	IN_INTR45	SPI3 DMA Read Request 3	Pulse
	SPI3	SPI3_dma_Write_req0	IN_INTR46	SPI3 DMA Write Request 0	Pulse
	SPI3	SPI3_dma_Write_req1	IN_INTR47	SPI3 DMA Write Request 1	Pulse
	SPI3	SPI3_dma_Write_req2	IN_INTR48	SPI3 DMA Write Request 2	Pulse
	SPI3	SPI3_dma_Write_req3	IN_INTR49	SPI3 DMA Write Request 3	Pulse
	SPI4	SPI4_dma_Read_req0	IN_INTR50	SPI4 DMA Read Request 0	Pulse
	SPI4	SPI4_dma_Read_req1	IN_INTR51	SPI4 DMA Read Request 1	Pulse
	SPI4	SPI4_dma_Read_req2	IN_INTR52	SPI4 DMA Read Request 2	Pulse
	SPI4	SPI4_dma_Read_req3	IN_INTR53	SPI4 DMA Read Request 3	Pulse
	SPI4	SPI4_dma_Write_req0	IN_INTR54	SPI4 DMA Write Request 0	Pulse
	SPI4	SPI4_dma_Write_req1	IN_INTR55	SPI4 DMA Write Request 1	Pulse
	SPI4	SPI4_dma_Write_req2	IN_INTR56	SPI4 DMA Write Request 2	Pulse
	SPI4	SPI4_dma_Write_req3	IN_INTR57	SPI4 DMA Write Request 3	Pulse
	RTI0	RTI0_DMA_0	IN_INTR58	RTI0 DMA Request 0	Pulse
	RTI0	RTI0_DMA_1	IN_INTR59	RTI0 DMA Request 1	Pulse
	RTI0	RTI0_DMA_2	IN_INTR60	RTI0 DMA Request 2	Pulse
	RTI0	RTI0_DMA_3	IN_INTR61	RTI0 DMA Request 3	Pulse
	RTI1	RTI1_DMA_0	IN_INTR62	RTI1 DMA Request 0	Pulse
	RTI1	RTI1_DMA_1	IN_INTR63	RTI1 DMA Request 1	Pulse
	RTI1	RTI1_DMA_2	IN_INTR64	RTI1 DMA Request 2	Pulse
	RTI1	RTI1_DMA_3	IN_INTR65	RTI1 DMA Request 3	Pulse
	RTI2	RTI2_DMA_0	IN_INTR66	RTI2 DMA Request 0	Pulse

Table 10-11. EDMA_XBAR_INTRTR0 in_intr Hardware Requests (continued)

Module Instance	Source Module	Source in_intr signal	XBAR Module in_intr	Desc ription	Type
	RTI2	RTI2_DMA_1	IN_INTR67	RTI2 DMA Request 1	Pulse
	RTI2	RTI2_DMA_2	IN_INTR68	RTI2 DMA Request 2	Pulse
	RTI2	RTI2_DMA_3	IN_INTR69	RTI2 DMA Request 3	Pulse
	RTI3	RTI3_DMA_0	IN_INTR70	RTI3 DMA Request 0	Pulse
	RTI3	RTI3_DMA_1	IN_INTR71	RTI3 DMA Request 1	Pulse
	RTI3	RTI3_DMA_2	IN_INTR72	RTI3 DMA Request 2	Pulse
	RTI3	RTI3_DMA_3	IN_INTR73	RTI3 DMA Request 3	Pulse
	MCAN0	mcanss0_tx_dma_0	IN_INTR74	MCAN0 TX DMA Request 0	Pulse
	MCAN0	mcanss0_tx_dma_1	IN_INTR75	MCAN0 TX DMA Request 1	Pulse
	MCAN0	mcanss0_tx_dma_2	IN_INTR76	MCAN0 TX DMA Request 2	Pulse
	MCAN0	mcanss0_tx_dma_3	IN_INTR77	MCAN0 TX DMA Request 3	Pulse
	MCAN1	mcanss1_tx_dma_0	IN_INTR78	MCAN1 TX DMA Request 0	Pulse
	MCAN1	mcanss1_tx_dma_1	IN_INTR79	MCAN1 TX DMA Request 1	Pulse
	MCAN1	mcanss1_tx_dma_2	IN_INTR80	MCAN1 TX DMA Request 2	Pulse
	MCAN1	mcanss1_tx_dma_3	IN_INTR81	MCAN1 TX DMA Request 3	Pulse
	MCAN2	mcanss2_tx_dma_0	IN_INTR82	MCAN2 TX DMA Request 0	Pulse
	MCAN2	mcanss2_tx_dma_1	IN_INTR83	MCAN2 TX DMA Request 1	Pulse
	MCAN2	mcanss2_tx_dma_2	IN_INTR84	MCAN2 TX DMA Request 2	Pulse
	MCAN2	mcanss2_tx_dma_3	IN_INTR85	MCAN2 TX DMA Request 3	Pulse
	MCAN3	mcanss3_tx_dma_0	IN_INTR86	MCAN3 TX DMA Request 0	Pulse
	MCAN3	mcanss3_tx_dma_1	IN_INTR87	MCAN3 TX DMA Request 1	Pulse
	MCAN3	mcanss3_tx_dma_2	IN_INTR88	MCAN3 TX DMA Request 2	Pulse
	MCAN3	mcanss3_tx_dma_3	IN_INTR89	MCAN3 TX DMA Request 3	Pulse
	UART0	usart0_dma_0	IN_INTR90	UART0 DMA Request 0	Pulse
	UART0	usart0_dma_1	IN_INTR91	UART0 DMA Request 1	Pulse
	UART1	usart1_dma_0	IN_INTR92	UART1 DMA Request 0	Pulse
	UART1	usart1_dma_1	IN_INTR93	UART1 DMA Request 1	Pulse
	UART2	usart2_dma_0	IN_INTR94	UART2 DMA Request 0	Pulse
	UART2	usart2_dma_1	IN_INTR95	UART2 DMA Request 1	Pulse
	UART3	usart0_dma_0	IN_INTR96	UART3 DMA Request 0	Pulse
	UART3	usart3_dma_1	IN_INTR97	UART3 DMA Request 1	Pulse
	UART4	usart4_dma_0	IN_INTR98	UART4 DMA Request 0	Pulse
	UART4	usart4_dma_1	IN_INTR99	UART4 DMA Request 1	Pulse
	UART5	usart5_dma_0	IN_INTR100	UART5 DMA Request 0	Pulse
	UART5	usart5_dma_1	IN_INTR101	UART5 DMA Request 1	Pulse
	MCRC	mcrc_DMA_Event_0	IN_INTR102	MCRC DMA Event 0	Pulse
	MCRC	mcrc_DMA_Event_1	IN_INTR103	MCRC DMA Event 1	Pulse
	MCRC	mcrc_DMA_Event_2	IN_INTR104	MCRC DMA Event 2	Pulse
	MCRC	mcrc_DMA_Event_3	IN_INTR105	MCRC DMA Event 3	Pulse
	QSPI	qSPI_intr	IN_INTR106	QSPI Interrupt	Pulse
	GPIO_XBAR	GPIO_xbarout_4	IN_INTR107	GPIO XBAR Out 4	Pulse
	GPIO_XBAR	GPIO_xbarout_5	IN_INTR108	GPIO XBAR Out 5	Pulse
	GPIO_XBAR	GPIO_xbarout_6	IN_INTR109	GPIO XBAR Out 6	Pulse
	GPIO_XBAR	GPIO_xbarout_7	IN_INTR110	GPIO XBAR Out 7	Pulse
	SOC_TimeSync_XBAR	Sync_Xbarout_0	IN_INTR111	SOC TimeSync XBAR Out 0	Pulse

Table 10-11. EDMA_XBAR_INTRTR0 in_intr Hardware Requests (continued)

Module Instance	Source Module	Source in_intr signal	XBAR Module in_intr	Desc ription	Type
	SOC_TimeSync_XBAR	Sync_Xbarout_1	IN_INTR112	SOC TimeSync XBAR Out 1	Pulse
	CONTROLSS_TimeSync_XBAR	C2k_timesync_xbar.out10	IN_INTR113	CONTROLSS TimeSync XBAR Out 0	Pulse
	CONTROLSS_TimeSync_XBAR	C2k_timesync_xbar.out11	IN_INTR114	CONTROLSS TimeSync XBAR Out 1	Pulse
	CONTROLSS_DMA_XBAR	CCSS_DMA_0	IN_INTR115	CONTROLSS EDMA_XBAR Out 0	Pulse
	CONTROLSS_DMA_XBAR	CCSS_DMA_1	IN_INTR116	CONTROLSS EDMA_XBAR Out 1	Pulse
	CONTROLSS_DMA_XBAR	CCSS_DMA_2	IN_INTR117	CONTROLSS EDMA_XBAR Out 2	Pulse
	CONTROLSS_DMA_XBAR	CCSS_DMA_3	IN_INTR118	CONTROLSS EDMA_XBAR Out 3	Pulse
	CONTROLSS_DMA_XBAR	CCSS_DMA_4	IN_INTR119	CONTROLSS EDMA_XBAR Out 4	Pulse
	CONTROLSS_DMA_XBAR	CCSS_DMA_5	IN_INTR120	CONTROLSS EDMA_XBAR Out 5	Pulse
	CONTROLSS_DMA_XBAR	CCSS_DMA_6	IN_INTR121	CONTROLSS EDMA_XBAR Out 6	Pulse
	CONTROLSS_DMA_XBAR	CCSS_DMA_7	IN_INTR122	CONTROLSS EDMA_XBAR Out 7	Pulse
	CONTROLSS_DMA_XBAR	CCSS_DMA_8	IN_INTR123	CONTROLSS EDMA_XBAR Out 8	Pulse
	CONTROLSS_DMA_XBAR	CCSS_DMA_9	IN_INTR124	CONTROLSS EDMA_XBAR Out 9	Pulse
	CONTROLSS_DMA_XBAR	CCSS_DMA_10	IN_INTR125	CONTROLSS EDMA_XBAR Out 10	Pulse
	CONTROLSS_DMA_XBAR	CCSS_DMA_11	IN_INTR126	CONTROLSS EDMA_XBAR Out 11	Pulse
	CONTROLSS_DMA_XBAR	CCSS_DMA_12	IN_INTR127	CONTROLSS EDMA_XBAR Out 12	Pulse
	CONTROLSS_DMA_XBAR	CCSS_DMA_13	IN_INTR128	CONTROLSS EDMA_XBAR Out 13	Pulse
	CONTROLSS_DMA_XBAR	CCSS_DMA_14	IN_INTR129	CONTROLSS EDMA_XBAR Out 14	Pulse
	CONTROLSS_DMA_XBAR	CCSS_DMA_15	IN_INTR130	CONTROLSS EDMA_XBAR Out 15	Pulse
	MMCSD	mmc_DMA_RD	IN_INTR131	MMCSD DMA Read Request	Pulse
	MMCSD	mmc_DMA_WR	IN_INTR132	MMCSD DMA Write Request	Pulse
	DTHE	DTHE_SHA_DMA_REQ0	IN_INTR133	DTHE SHA DMA Request 0	Pulse
	DTHE	DTHE_SHA_DMA_REQ1	IN_INTR134	DTHE SHA DMA Request 1	Pulse
	DTHE	DTHE_SHA_DMA_REQ2	IN_INTR135	DTHE SHA DMA Request 2	Pulse
	DTHE	DTHE_SHA_DMA_REQ3	IN_INTR136	DTHE SHA DMA Request 3	Pulse
	DTHE	DTHE_SHA_DMA_REQ4	IN_INTR137	DTHE SHA DMA Request 4	Pulse
	DTHE	DTHE_SHA_DMA_REQ5	IN_INTR138	DTHE SHA DMA Request 5	Pulse
	DTHE	DTHE_AES_DMA_REQ0	IN_INTR139	DTHE AES DMA Request 0	Pulse
	DTHE	DTHE_AES_DMA_REQ1	IN_INTR140	DTHE AES DMA Request 1	Pulse
	DTHE	DTHE_AES_DMA_REQ2	IN_INTR141	DTHE AES DMA Request 2	Pulse
	DTHE	DTHE_AES_DMA_REQ3	IN_INTR142	DTHE AES DMA Request 3	Pulse
	DTHE	DTHE_AES_DMA_REQ4	IN_INTR143	DTHE AES DMA Request 4	Pulse
	DTHE	DTHE_AES_DMA_REQ5	IN_INTR144	DTHE AES DMA Request 5	Pulse

Table 10-11. EDMA_XBAR_INTRTR0 in_intr Hardware Requests (continued)

Module Instance	Source Module	Source in_intr signal	XBAR Module in_intr	Desc ription	Type
	DTHE	DTHE_AES_DMA_REQ6	IN_INTR145	DTHE AES DMA Request 6	Pulse
	DTHE	DTHE_AES_DMA_REQ7	IN_INTR146	DTHE AES DMA Request 7	Pulse
	MCAN0	mcanss0_fe_0	IN_INTR147	MCAN0 Request 0	Pulse
	MCAN0	mcanss0_fe_1	IN_INTR148	MCAN0 Request 1	Pulse
	MCAN0	mcanss0_fe_2	IN_INTR149	MCAN0 Request 2	Pulse
	MCAN0	mcanss0_fe_3	IN_INTR150	MCAN0 Request 3	Pulse
	MCAN0	mcanss0_fe_4	IN_INTR151	MCAN0 Request 4	Pulse
	MCAN0	mcanss0_fe_5	IN_INTR152	MCAN0 Request 5	Pulse
	MCAN0	mcanss0_fe_6	IN_INTR153	MCAN0 Request 6	Pulse
	MCAN1	mcanss1_fe_0	IN_INTR154	MCAN1 Request 0	Pulse
	MCAN1	mcanss1_fe_1	IN_INTR155	MCAN1 Request 1	Pulse
	MCAN1	mcanss1_fe_2	IN_INTR156	MCAN1 Request 2	Pulse
	MCAN1	mcanss1_fe_3	IN_INTR157	MCAN1 Request 3	Pulse
	MCAN1	mcanss1_fe_4	IN_INTR158	MCAN1 Request 4	Pulse
	MCAN1	mcanss1_fe_5	IN_INTR159	MCAN1 Request 5	Pulse
	MCAN1	mcanss1_fe_6	IN_INTR160	MCAN1 Request 6	Pulse
	MCAN2	mcanss2_fe_0	IN_INTR161	MCAN2 Request 0	Pulse
	MCAN2	mcanss2_fe_1	IN_INTR162	MCAN2 Request 1	Pulse
	MCAN2	mcanss2_fe_2	IN_INTR163	MCAN2 Request 2	Pulse
	MCAN2	mcanss2_fe_3	IN_INTR164	MCAN2 Request 3	Pulse
	MCAN2	mcanss2_fe_4	IN_INTR165	MCAN2 Request 4	Pulse
	MCAN2	mcanss2_fe_5	IN_INTR166	MCAN2 Request 5	Pulse
	MCAN2	mcanss2_fe_6	IN_INTR167	MCAN2 Request 6	Pulse
	MCAN3	mcanss3_fe_0	IN_INTR168	MCAN3 Request 0	Pulse
	MCAN3	mcanss3_fe_1	IN_INTR169	MCAN3 Request 1	Pulse
	MCAN3	mcanss3_fe_2	IN_INTR170	MCAN3 Request 2	Pulse
	MCAN3	mcanss3_fe_3	IN_INTR171	MCAN3 Request 3	Pulse
	MCAN3	mcanss3_fe_4	IN_INTR172	MCAN3 Request 4	Pulse
	MCAN3	mcanss3_fe_5	IN_INTR173	MCAN3 Request 5	Pulse
	MCAN3	mcanss3_fe_6	IN_INTR174	MCAN3 Request 6	Pulse
	RESERVED	Reserved	IN_INTR175	Reserved	-
	MCAN4	mcanss4_tx_dma_0	IN_INTR176	MCAN4 TX DMA Request 0	Pulse
	MCAN4	mcanss4_tx_dma_1	IN_INTR177	MCAN4 TX DMA Request 1	Pulse
	MCAN4	mcanss4_tx_dma_2	IN_INTR178	MCAN4 TX DMA Request 2	Pulse
	MCAN4	mcanss4_tx_dma_3	IN_INTR179	MCAN4 TX DMA Request 3	Pulse
	MCAN5	mcanss5_tx_dma_0	IN_INTR180	MCAN5 TX DMA Request 0	Pulse
	MCAN5	mcanss5_tx_dma_1	IN_INTR181	MCAN5 TX DMA Request 1	Pulse
	MCAN5	mcanss5_tx_dma_2	IN_INTR182	MCAN5 TX DMA Request 2	Pulse
	MCAN5	mcanss5_tx_dma_3	IN_INTR183	MCAN5 TX DMA Request 3	Pulse
	MCAN6	mcanss6_tx_dma_0	IN_INTR184	MCAN6 TX DMA Request 0	Pulse
	MCAN6	mcanss6_tx_dma_1	IN_INTR185	MCAN6 TX DMA Request 1	Pulse
	MCAN6	mcanss6_tx_dma_2	IN_INTR186	MCAN6 TX DMA Request 2	Pulse
	MCAN6	mcanss6_tx_dma_3	IN_INTR187	MCAN6 TX DMA Request 3	Pulse
	MCAN7	mcanss7_tx_dma_0	IN_INTR188	MCAN7 TX DMA Request 0	Pulse
	MCAN7	mcanss7_tx_dma_1	IN_INTR189	MCAN7 TX DMA Request 1	Pulse
	MCAN7	mcanss7_tx_dma_2	IN_INTR190	MCAN7 TX DMA Request 2	Pulse

Table 10-11. EDMA_XBAR_INTRTR0 in_intr Hardware Requests (continued)

Module Instance	Source Module	Source in_intr signal	XBAR Module in_intr	Desc ription	Type
	MCAN7	mcanss7_tx_dma_3	IN_INTR191	MCAN7 TX DMA Request 3	Pulse
	MCAN4	mcanss4_fe_0	IN_INTR192	MCAN4 Request 0	Pulse
	MCAN4	mcanss4_fe_1	IN_INTR193	MCAN4 Request 1	Pulse
	MCAN4	mcanss4_fe_2	IN_INTR194	MCAN4 Request 2	Pulse
	MCAN4	mcanss4_fe_3	IN_INTR195	MCAN4 Request 3	Pulse
	MCAN4	mcanss4_fe_4	IN_INTR196	MCAN4 Request 4	Pulse
	MCAN4	mcanss4_fe_5	IN_INTR197	MCAN4 Request 5	Pulse
	MCAN4	mcanss4_fe_6	IN_INTR198	MCAN4 Request 6	Pulse
	MCAN5	mcanss5_fe_0	IN_INTR199	MCAN5 Request 0	Pulse
	MCAN5	mcanss5_fe_1	IN_INTR200	MCAN5 Request 1	Pulse
	MCAN5	mcanss5_fe_2	IN_INTR201	MCAN5 Request 2	Pulse
	MCAN5	mcanss5_fe_3	IN_INTR202	MCAN5 Request 3	Pulse
	MCAN5	mcanss5_fe_4	IN_INTR203	MCAN5 Request 4	Pulse
	MCAN5	mcanss5_fe_5	IN_INTR204	MCAN5 Request 5	Pulse
	MCAN5	mcanss5_fe_6	IN_INTR205	MCAN5 Request 6	Pulse
	MCAN6	mcanss6_fe_0	IN_INTR206	MCAN6 Request 0	Pulse
	MCAN6	mcanss6_fe_1	IN_INTR207	MCAN6 Request 1	Pulse
	MCAN6	mcanss6_fe_2	IN_INTR208	MCAN6 Request 2	Pulse
	MCAN6	mcanss6_fe_3	IN_INTR209	MCAN6 Request 3	Pulse
	MCAN6	mcanss6_fe_4	IN_INTR210	MCAN6 Request 4	Pulse
	MCAN6	mcanss6_fe_5	IN_INTR211	MCAN6 Request 5	Pulse
	MCAN6	mcanss6_fe_6	IN_INTR212	MCAN6 Request 6	Pulse
	MCAN7	mcanss7_fe_0	IN_INTR213	MCAN7 Request 0	Pulse
	MCAN7	mcanss7_fe_1	IN_INTR214	MCAN7 Request 1	Pulse
	MCAN7	mcanss7_fe_2	IN_INTR215	MCAN7 Request 2	Pulse
	MCAN7	mcanss7_fe_3	IN_INTR216	MCAN7 Request 3	Pulse
	MCAN7	mcanss7_fe_4	IN_INTR217	MCAN7 Request 4	Pulse
	MCAN7	mcanss7_fe_5	IN_INTR218	MCAN7 Request 5	Pulse
	MCAN7	mcanss7_fe_6	IN_INTR219	MCAN7 Request 6	Pulse
	SPI5	SPI5_dma_Read_req0	IN_INTR220	SPI5 DMA Read Request 0	Pulse
	SPI5	SPI5_dma_Read_req1	IN_INTR221	SPI5 DMA Read Request 1	Pulse
	SPI5	SPI5_dma_Read_req2	IN_INTR222	SPI5 DMA Read Request 2	Pulse
	SPI5	SPI5_dma_Read_req3	IN_INTR223	SPI5 DMA Read Request 3	Pulse
	SPI5	SPI5_dma_Write_req0	IN_INTR224	SPI5 DMA Write Request 0	Pulse
	SPI5	SPI5_dma_Write_req1	IN_INTR225	SPI5 DMA Write Request 1	Pulse
	SPI5	SPI5_dma_Write_req2	IN_INTR226	SPI5 DMA Write Request 2	Pulse
	SPI5	SPI5_dma_Write_req3	IN_INTR227	SPI5 DMA Write Request 3	Pulse
	SPI6	SPI6_dma_Read_req0	IN_INTR228	SPI6 DMA Read Request 0	Pulse
	SPI6	SPI6_dma_Read_req1	IN_INTR229	SPI6 DMA Read Request 1	Pulse
	SPI6	SPI6_dma_Read_req2	IN_INTR230	SPI6 DMA Read Request 2	Pulse
	SPI6	SPI6_dma_Read_req3	IN_INTR231	SPI6 DMA Read Request 3	Pulse
	SPI6	SPI6_dma_Write_req0	IN_INTR232	SPI6 DMA Write Request 0	Pulse
	SPI6	SPI6_dma_Write_req1	IN_INTR233	SPI6 DMA Write Request 1	Pulse
	SPI6	SPI6_dma_Write_req2	IN_INTR234	SPI6 DMA Write Request 2	Pulse
	SPI6	SPI6_dma_Write_req3	IN_INTR235	SPI6 DMA Write Request 3	Pulse
	SPI7	SPI7_dma_Read_req0	IN_INTR236	SPI7 DMA Read Request 0	Pulse

Table 10-11. EDMA_XBAR_INTRTR0 in_intr Hardware Requests (continued)

Module Instance	Source Module	Source in_intr signal	XBAR Module in_intr	Desc ription	Type
	SPI7	SPI7_dma_Read_req1	IN_INTR237	SPI7 DMA Read Request 1	Pulse
	SPI7	SPI7_dma_Read_req2	IN_INTR238	SPI7 DMA Read Request 2	Pulse
	SPI7	SPI7_dma_Read_req3	IN_INTR239	SPI7 DMA Read Request 3	Pulse
	SPI7	SPI7_dma_Write_req0	IN_INTR240	SPI7 DMA Write Request 0	Pulse
	SPI7	SPI7_dma_Write_req1	IN_INTR241	SPI7 DMA Write Request 1	Pulse
	SPI7	SPI7_dma_Write_req2	IN_INTR242	SPI7 DMA Write Request 2	Pulse
	SPI7	SPI7_dma_Write_req3	IN_INTR243	SPI7 DMA Write Request 3	Pulse
	RTI4	RTI4_DMA_0	IN_INTR244	RTI4 DMA Request 0	Pulse
	RTI4	RTI4_DMA_1	IN_INTR245	RTI4 DMA Request 1	Pulse
	RTI4	RTI4_DMA_2	IN_INTR246	RTI4 DMA Request 2	Pulse
	RTI4	RTI4_DMA_3	IN_INTR247	RTI4 DMA Request 3	Pulse
	RTI5	RTI5_DMA_0	IN_INTR248	RTI5 DMA Request 0	Pulse
	RTI5	RTI5_DMA_1	IN_INTR249	RTI5 DMA Request 1	Pulse
	RTI5	RTI5_DMA_2	IN_INTR250	RTI5 DMA Request 2	Pulse
	RTI5	RTI5_DMA_3	IN_INTR251	RTI5 DMA Request 3	Pulse
	RTI6	RTI6_DMA_0	IN_INTR252	RTI6 DMA Request 0	Pulse
	RTI6	RTI6_DMA_1	IN_INTR253	RTI6 DMA Request 1	Pulse
	RTI6	RTI6_DMA_2	IN_INTR254	RTI6 DMA Request 2	Pulse
	RTI6	RTI6_DMA_3	IN_INTR255	RTI6 DMA Request 3	Pulse
	RTI7	RTI7_DMA_0	IN_INTR256	RTI7 DMA Request 0	Pulse
	RTI7	RTI7_DMA_1	IN_INTR257	RTI7 DMA Request 1	Pulse
	RTI7	RTI7_DMA_2	IN_INTR258	RTI7 DMA Request 2	Pulse
	RTI7	RTI7_DMA_3	IN_INTR259	RTI7 DMA Request 3	Pulse

10.3.2.3 GPIO XBAR INTRTR0

There is 1x GPIO XBAR Interrupt Router module integrated in the device. The diagram below provides a visual representation of the device integration details for GPIO XBAR Interrupt router.

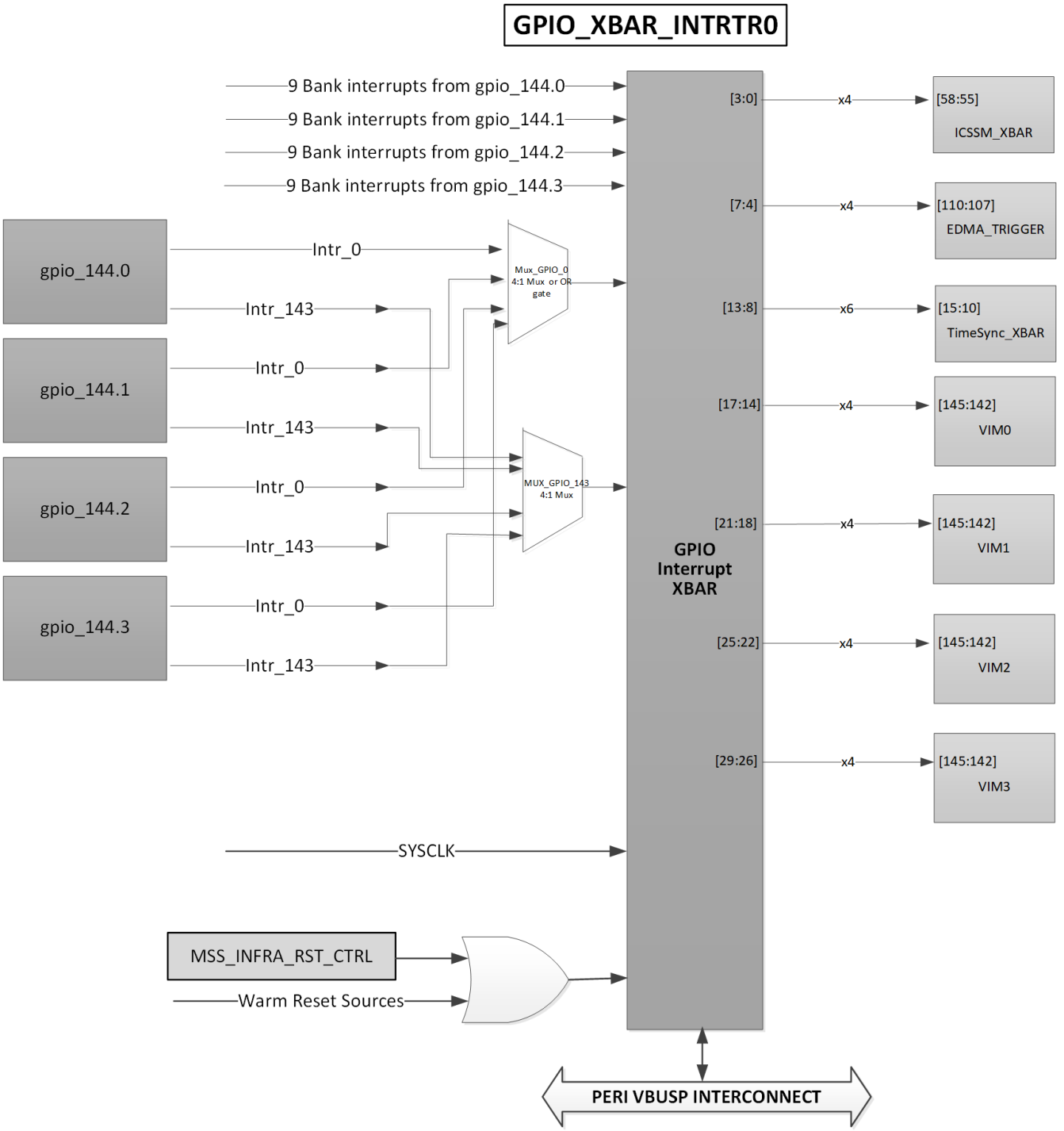


Figure 10-4. GPIO XBAR Interrupt Router Integration Diagram

The tables below summarize the device integration details of GPIO XBAR Interrupt router.

Table 10-12. GPIO XBAR Intrrupt router Device Integration

Module Instance	Device Allocation	SoC Interconnect
GPIO_XBAR_INTRTR0	✓	INFRA0 VBUSP Interconnect

Table 10-13. GPIO_XBAR_INTRTR0 Clocks

Module Instance	Module Clock in_intr	Source Clock Signal	Source	Default Freq	Description
GPIO_XBAR_INTRTR0	SYSCLK	SYS_CLK	MSS_RCM	200 MHz	GPIO_XBAR_INTRTR0 Functional and Interface clock

Table 10-14. GPIO_XBAR_INTRTR0 Resets

Module Instance	Module Reset in_intr	Source Reset Signal	Source	Description
GPIO_XBAR_INTRTR0	RST	SYS_RST	MSS_RCM	GPIO_XBAR_INTRTR0 Reset

Table 10-15. GPIO_XBAR_INTRTR0 Ouput Hardware Requests

Module Instance	Module XBAR Output	Destination XBAR signal	Destination	Description	Type
GPIO_XBAR_INTRTR0	outl_intr_0	GPIO_XBAR_ICSSM_out_0	ICSSM_XBAR	Selectable Hardware Request0	Pulse
	outl_intr_1	GPIO_XBAR_ICSSM_out_1	ICSSM_XBAR	Selectable Hardware Request1	
	outl_intr_2	GPIO_XBAR_ICSSM_out_2	ICSSM_XBAR	Selectable Hardware Request2	
	outl_intr_3	GPIO_XBAR_ICSSM_out_3	ICSSM_XBAR	Selectable Hardware Request3	
	outl_intr_4	GPIO_XBAR_EDMA_out_0	EDMA_XBAR	Selectable Hardware Request4	
	outl_intr_5	GPIO_XBAR_EDMA_out_1	EDMA_XBAR	Selectable Hardware Request5	
	outl_intr_6	GPIO_XBAR_EDMA_out_2	EDMA_XBAR	Selectable Hardware Request6	
	outl_intr_7	GPIO_XBAR_EDMA_out_3	EDMA_XBAR	Selectable Hardware Request7	
	outl_intr_8	GPIO_XBAR_TimeSync_out_0	TimeSync_XBAR	Selectable Hardware Request8	
	outl_intr_9	GPIO_XBAR_TimeSync_out_1	TimeSync_XBAR	Selectable Hardware Request9	
	outl_intr_10	GPIO_XBAR_TimeSync_out_2	TimeSync_XBAR	Selectable Hardware Request10	
	outl_intr_11	GPIO_XBAR_TimeSync_out_3	TimeSync_XBAR	Selectable Hardware Request11	
	outl_intr_12	GPIO_XBAR_TimeSync_out_4	TimeSync_XBAR	Selectable Hardware Request12	
	outl_intr_13	GPIO_XBAR_TimeSync_out_5	TimeSync_XBAR	Selectable Hardware Request13	
	outl_intr_14	GPIO_XBAR_VIM0_out_0	VIM_0	Selectable Hardware Request14	
	outl_intr_15	GPIO_XBAR_VIM0_out_1	VIM_0	Selectable Hardware Request15	
	outl_intr_16	GPIO_XBAR_VIM0_out_2	VIM_0	Selectable Hardware Request16	
	outl_intr_17	GPIO_XBAR_VIM0_out_3	VIM_0	Selectable Hardware Request17	
	outl_intr_18	GPIO_XBAR_VIM1_out_0	VIM_1	Selectable Hardware Request18	
	outl_intr_19	GPIO_XBAR_VIM1_out_1	VIM_1	Selectable Hardware Request19	
	outl_intr_20	GPIO_XBAR_VIM1_out_2	VIM_1	Selectable Hardware Request20	
	outl_intr_21	GPIO_XBAR_VIM1_out_3	VIM_1	Selectable Hardware Request21	
	outl_intr_22	GPIO_XBAR_VIM2_out_0	VIM_2	Selectable Hardware Request22	
	outl_intr_23	GPIO_XBAR_VIM2_out_1	VIM_2	Selectable Hardware Request23	
	outl_intr_24	GPIO_XBAR_VIM2_out_2	VIM_2	Selectable Hardware Request24	
	outl_intr_25	GPIO_XBAR_VIM2_out_3	VIM_2	Selectable Hardware Request25	
	outl_intr_26	GPIO_XBAR_VIM3_out_0	VIM_3	Selectable Hardware Request26	
	outl_intr_27	GPIO_XBAR_VIM3_out_1	VIM_3	Selectable Hardware Request27	
	outl_intr_28	GPIO_XBAR_VIM3_out_2	VIM_3	Selectable Hardware Request28	
	outl_intr_29	GPIO_XBAR_VIM3_out_3	VIM_3	Selectable Hardware Request29	

Table 10-16. GPIO_XBAR_INTRTR0 in_intr Hardware Requests

Module Instance	Source Module	Source in_intr signal	XBAR Module in_intr	Description	Type
GPIO_XBAR_INTRTR0	GPIO_0	mux_GPIO_0	in_intr_0	gpio_144x.intr_0 in_intr	Pulse
	GPIO_1	mux_GPIO_1	in_intr_1	gpio_144x.intr_1 in_intr	
	GPIO_2	mux_GPIO_2	in_intr_2	gpio_144x.intr_2 in_intr	
	GPIO_3	mux_GPIO_3	in_intr_3	gpio_144x.intr_3 in_intr	
	GPIO_4	mux_GPIO_4	in_intr_4	gpio_144x.intr_4 in_intr	
	GPIO_5	mux_GPIO_5	in_intr_5	gpio_144x.intr_5 in_intr	
	GPIO_6	mux_GPIO_6	in_intr_6	gpio_144x.intr_6 in_intr	
	GPIO_7	mux_GPIO_7	in_intr_7	gpio_144x.intr_7 in_intr	
	GPIO_8	mux_GPIO_8	in_intr_8	gpio_144x.intr_8 in_intr	
	GPIO_9	mux_GPIO_9	in_intr_9	gpio_144x.intr_9 in_intr	
	GPIO_10	mux_GPIO_10	in_intr_10	gpio_144x.intr_10 in_intr	
	GPIO_11	mux_GPIO_11	in_intr_11	gpio_144x.intr_11 in_intr	
	GPIO_12	mux_GPIO_12	in_intr_12	gpio_144x.intr_12 in_intr	
	GPIO_13	mux_GPIO_13	in_intr_13	gpio_144x.intr_13 in_intr	
	GPIO_14	mux_GPIO_14	in_intr_14	gpio_144x.intr_14 in_intr	
	GPIO_15	mux_GPIO_15	in_intr_15	gpio_144x.intr_15 in_intr	
	GPIO_16	mux_GPIO_16	in_intr_16	gpio_144x.intr_16 in_intr	
	GPIO_17	mux_GPIO_17	in_intr_17	gpio_144x.intr_17 in_intr	
	GPIO_18	mux_GPIO_18	in_intr_18	gpio_144x.intr_18 in_intr	
	GPIO_19	mux_GPIO_19	in_intr_19	gpio_144x.intr_19 in_intr	
	GPIO_20	mux_GPIO_20	in_intr_20	gpio_144x.intr_20 in_intr	
	GPIO_21	mux_GPIO_21	in_intr_21	gpio_144x.intr_21 in_intr	
	GPIO_22	mux_GPIO_22	in_intr_22	gpio_144x.intr_22 in_intr	
	GPIO_23	mux_GPIO_23	in_intr_23	gpio_144x.intr_23 in_intr	
	GPIO_24	mux_GPIO_24	in_intr_24	gpio_144x.intr_24 in_intr	
	GPIO_25	mux_GPIO_25	in_intr_25	gpio_144x.intr_25 in_intr	
	GPIO_26	mux_GPIO_26	in_intr_26	gpio_144x.intr_26 in_intr	
	GPIO_27	mux_GPIO_27	in_intr_27	gpio_144x.intr_27 in_intr	
	GPIO_28	mux_GPIO_28	in_intr_28	gpio_144x.intr_28 in_intr	
	GPIO_29	mux_GPIO_29	in_intr_29	gpio_144x.intr_29 in_intr	
	GPIO_30	mux_GPIO_30	in_intr_30	gpio_144x.intr_30 in_intr	
	GPIO_31	mux_GPIO_31	in_intr_31	gpio_144x.intr_31 in_intr	
	GPIO_32	mux_GPIO_32	in_intr_32	gpio_144x.intr_32 in_intr	
	GPIO_33	mux_GPIO_33	in_intr_33	gpio_144x.intr_33 in_intr	
	GPIO_34	mux_GPIO_34	in_intr_34	gpio_144x.intr_34 in_intr	
	GPIO_35	mux_GPIO_35	in_intr_35	gpio_144x.intr_35 in_intr	
	GPIO_36	mux_GPIO_36	in_intr_36	gpio_144x.intr_36 in_intr	
	GPIO_37	mux_GPIO_37	in_intr_37	gpio_144x.intr_37 in_intr	
	GPIO_38	mux_GPIO_38	in_intr_38	gpio_144x.intr_38 in_intr	
	GPIO_39	mux_GPIO_39	in_intr_39	gpio_144x.intr_38 in_intr	
	GPIO_40	mux_GPIO_40	in_intr_40	gpio_144x.intr_40 in_intr	
	GPIO_41	mux_GPIO_41	in_intr_41	gpio_144x.intr_41 in_intr	
	GPIO_42	mux_GPIO_42	in_intr_42	gpio_144x.intr_42 in_intr	
	GPIO_43	mux_GPIO_43	in_intr_43	gpio_144x.intr_43 in_intr	

Table 10-16. GPIO_XBAR_INTRTR0 in_intr Hardware Requests (continued)

Module Instance	Source Module	Source in_intr signal	XBAR Module in_intr	Description	Type
	GPIO_44	mux_GPIO_44	in_intr_44	gpio_144x.intr_44 in_intr	
	GPIO_45	mux_GPIO_45	in_intr_45	gpio_144x.intr_45 in_intr	
	GPIO_46	mux_GPIO_46	in_intr_46	gpio_144x.intr_46 in_intr	
	GPIO_47	mux_GPIO_47	in_intr_47	gpio_144x.intr_47 in_intr	
	GPIO_48	mux_GPIO_48	in_intr_48	gpio_144x.intr_48 in_intr	
	GPIO_49	mux_GPIO_49	in_intr_49	gpio_144x.intr_49 in_intr	
	GPIO_50	mux_GPIO_50	in_intr_50	gpio_144x.intr_50 in_intr	
	GPIO_51	mux_GPIO_51	in_intr_51	gpio_144x.intr_51 in_intr	
	GPIO_52	mux_GPIO_52	in_intr_52	gpio_144x.intr_52 in_intr	
	GPIO_53	mux_GPIO_53	in_intr_53	gpio_144x.intr_53 in_intr	
	GPIO_54	mux_GPIO_54	in_intr_54	gpio_144x.intr_54 in_intr	
	GPIO_55	mux_GPIO_55	in_intr_55	gpio_144x.intr_55 in_intr	
	GPIO_56	mux_GPIO_56	in_intr_56	gpio_144x.intr_56 in_intr	
	GPIO_57	mux_GPIO_57	in_intr_57	gpio_144x.intr_57 in_intr	
	GPIO_58	mux_GPIO_58	in_intr_58	gpio_144x.intr_58 in_intr	
	GPIO_59	mux_GPIO_59	in_intr_59	gpio_144x.intr_59 in_intr	
	GPIO_60	mux_GPIO_60	in_intr_60	gpio_144x.intr_60 in_intr	
	GPIO_61	mux_GPIO_61	in_intr_61	gpio_144x.intr_61 in_intr	
	GPIO_62	mux_GPIO_62	in_intr_62	gpio_144x.intr_62 in_intr	
	GPIO_63	mux_GPIO_63	in_intr_63	gpio_144x.intr_63 in_intr	
	GPIO_64	mux_GPIO_64	in_intr_64	gpio_144x.intr_64 in_intr	
	GPIO_65	mux_GPIO_65	in_intr_65	gpio_144x.intr_65 in_intr	
	GPIO_66	mux_GPIO_66	in_intr_66	gpio_144x.intr_66 in_intr	
	GPIO_67	mux_GPIO_67	in_intr_67	gpio_144x.intr_67 in_intr	
	GPIO_68	mux_GPIO_68	in_intr_68	gpio_144x.intr_68 in_intr	
	GPIO_69	mux_GPIO_69	in_intr_69	gpio_144x.intr_69 in_intr	
	GPIO_70	mux_GPIO_70	in_intr_70	gpio_144x.intr_70 in_intr	
	GPIO_71	mux_GPIO_71	in_intr_71	gpio_144x.intr_71 in_intr	
	GPIO_12	mux_GPIO_72	in_intr_72	gpio_144x.intr_72 in_intr	
	GPIO_73	mux_GPIO_73	in_intr_73	gpio_144x.intr_73 in_intr	
	GPIO_74	mux_GPIO_74	in_intr_74	gpio_144x.intr_74 in_intr	
	GPIO_75	mux_GPIO_75	in_intr_75	gpio_144x.intr_75 in_intr	
	GPIO_76	mux_GPIO_76	in_intr_76	gpio_144x.intr_76 in_intr	
	GPIO_77	mux_GPIO_77	in_intr_77	gpio_144x.intr_77 in_intr	
	GPIO_78	mux_GPIO_78	in_intr_78	gpio_144x.intr_78 in_intr	
	GPIO_79	mux_GPIO_79	in_intr_79	gpio_144x.intr_79 in_intr	
	GPIO_80	mux_GPIO_80	in_intr_80	gpio_144x.intr_80 in_intr	
	GPIO_81	mux_GPIO_81	in_intr_81	gpio_144x.intr_81 in_intr	
	GPIO_82	mux_GPIO_82	in_intr_82	gpio_144x.intr_82 in_intr	
	GPIO_83	mux_GPIO_83	in_intr_83	gpio_144x.intr_83 in_intr	
	GPIO_84	mux_GPIO_84	in_intr_84	gpio_144x.intr_84 in_intr	
	GPIO_85	mux_GPIO_85	in_intr_85	gpio_144x.intr_85 in_intr	
	GPIO_86	mux_GPIO_86	in_intr_86	gpio_144x.intr_86 in_intr	
	GPIO_87	mux_GPIO_87	in_intr_87	gpio_144x.intr_87 in_intr	

Table 10-16. GPIO_XBAR_INTRTR0 in_intr Hardware Requests (continued)

Module Instance	Source Module	Source in_intr signal	XBAR Module in_intr	Description	Type
	GPIO_88	mux_GPIO_88	in_intr_88	gpio_144x.intr_88 in_intr	
	GPIO_89	mux_GPIO_89	in_intr_89	gpio_144x.intr_89 in_intr	
	GPIO_90	mux_GPIO_90	in_intr_90	gpio_144x.intr_90 in_intr	
	GPIO_91	mux_GPIO_91	in_intr_91	gpio_144x.intr_91 in_intr	
	GPIO_92	mux_GPIO_92	in_intr_92	gpio_144x.intr_92 in_intr	
	GPIO_93	mux_GPIO_93	in_intr_93	gpio_144x.intr_93 in_intr	
	GPIO_94	mux_GPIO_94	in_intr_94	gpio_144x.intr_94 in_intr	
	GPIO_95	mux_GPIO_95	in_intr_95	gpio_144x.intr_95 in_intr	
	GPIO_96	mux_GPIO_96	in_intr_96	gpio_144x.intr_96 in_intr	
	GPIO_97	mux_GPIO_97	in_intr_97	gpio_144x.intr_97 in_intr	
	GPIO_98	mux_GPIO_98	in_intr_98	gpio_144x.intr_98 in_intr	
	GPIO_99	mux_GPIO_99	in_intr_99	gpio_144x.intr_99 in_intr	
	GPIO_100	mux_GPIO_100	in_intr_100	gpio_144x.intr_100 in_intr	
	GPIO_101	mux_GPIO_101	in_intr_101	gpio_144x.intr_101 in_intr	
	GPIO_102	mux_GPIO_102	in_intr_102	gpio_144x.intr_102 in_intr	
	GPIO_103	mux_GPIO_103	in_intr_103	gpio_144x.intr_103 in_intr	
	GPIO_104	mux_GPIO_104	in_intr_104	gpio_144x.intr_104 in_intr	
	GPIO_105	mux_GPIO_105	in_intr_105	gpio_144x.intr_105 in_intr	
	GPIO_106	mux_GPIO_106	in_intr_106	gpio_144x.intr_106 in_intr	
	GPIO_107	mux_GPIO_107	in_intr_107	gpio_144x.intr_107 in_intr	
	GPIO_108	mux_GPIO_108	in_intr_108	gpio_144x.intr_108 in_intr	
	GPIO_109	mux_GPIO_109	in_intr_109	gpio_144x.intr_109 in_intr	
	GPIO_110	mux_GPIO_110	in_intr_110	gpio_144x.intr_110 in_intr	
	GPIO_111	mux_GPIO_111	in_intr_111	gpio_144x.intr_111 in_intr	
	GPIO_112	mux_GPIO_112	in_intr_112	gpio_144x.intr_112 in_intr	
	GPIO_113	mux_GPIO_113	in_intr_113	gpio_144x.intr_113 in_intr	
	GPIO_114	mux_GPIO_114	in_intr_114	gpio_144x.intr_114 in_intr	
	GPIO_115	mux_GPIO_115	in_intr_115	gpio_144x.intr_115 in_intr	
	GPIO_116	mux_GPIO_116	in_intr_116	gpio_144x.intr_116 in_intr	
	GPIO_117	mux_GPIO_117	in_intr_117	gpio_144x.intr_117 in_intr	
	GPIO_118	mux_GPIO_118	in_intr_118	gpio_144x.intr_118 in_intr	
	GPIO_119	mux_GPIO_119	in_intr_119	gpio_144x.intr_119 in_intr	
	GPIO_120	mux_GPIO_120	in_intr_120	gpio_144x.intr_120 in_intr	
	GPIO_121	mux_GPIO_121	in_intr_121	gpio_144x.intr_121 in_intr	
	GPIO_122	mux_GPIO_122	in_intr_122	gpio_144x.intr_122 in_intr	
	GPIO_123	mux_GPIO_123	in_intr_123	gpio_144x.intr_123 in_intr	

Table 10-16. GPIO_XBAR_INTRTR0 in_intr Hardware Requests (continued)

Module Instance	Source Module	Source in_intr signal	XBAR Module in_intr	Description	Type
	GPIO_124	mux_GPIO_124	in_intr_124	gpio_144x.intr_124 in_intr	
	GPIO_125	mux_GPIO_125	in_intr_125	gpio_144x.intr_125 in_intr	
	GPIO_126	mux_GPIO_126	in_intr_126	gpio_144x.intr_126 in_intr	
	GPIO_127	mux_GPIO_127	in_intr_127	gpio_144x.intr_127 in_intr	
	GPIO_128	mux_GPIO_128	in_intr_128	gpio_144x.intr_128 in_intr	
	GPIO_129	mux_GPIO_129	in_intr_129	gpio_144x.intr_129 in_intr	
	GPIO_130	mux_GPIO_130	in_intr_130	gpio_144x.intr_130 in_intr	
	GPIO_131	mux_GPIO_131	in_intr_131	gpio_144x.intr_131 in_intr	
	GPIO_132	mux_GPIO_132	in_intr_132	gpio_144x.intr_132 in_intr	
	GPIO_133	mux_GPIO_133	in_intr_133	gpio_144x.intr_133 in_intr	
	GPIO_134	mux_GPIO_134	in_intr_134	gpio_144x.intr_134 in_intr	
	GPIO_135	mux_GPIO_135	in_intr_135	gpio_144x.intr_135 in_intr	
	GPIO_136	mux_GPIO_136	in_intr_136	gpio_144x.intr_136 in_intr	
	GPIO_137	mux_GPIO_137	in_intr_137	gpio_144x.intr_137 in_intr	
	GPIO_138	mux_GPIO_138	in_intr_138	gpio_144x.intr_138 in_intr	
	GPIO_139	mux_GPIO_139	in_intr_139	gpio_144x.intr_139 in_intr	
	GPIO_140	mux_GPIO_140	in_intr_140	gpio_144x.intr_140 in_intr	
	GPIO_141	mux_GPIO_141	in_intr_141	gpio_144x.intr_141 in_intr	
	GPIO_142	mux_GPIO_142	in_intr_142	gpio_144x.intr_142 in_intr	
	GPIO_143	mux_GPIO_143	in_intr_143	gpio_144x.intr_143 in_intr	
	gpio_144_0_bank_intr_0	mux_GPIO_144	in_intr_144	gpio_144x.intr_144 in_intr	
	gpio_144_0_bank_intr_1	mux_GPIO_145	in_intr_145	gpio_144x.intr_145 in_intr	
	gpio_144_0_bank_intr_2	mux_GPIO_146	in_intr_146	gpio_144x.intr_146 in_intr	
	gpio_144_0_bank_intr_3	mux_GPIO_147	in_intr_147	gpio_144x.intr_147 in_intr	
	gpio_144_0_bank_intr_4	mux_GPIO_148	in_intr_148	gpio_144x.intr_148 in_intr	
	gpio_144_0_bank_intr_5	mux_GPIO_149	in_intr_149	gpio_144x.intr_149 in_intr	
	gpio_144_0_bank_intr_6	mux_GPIO_150	in_intr_150	gpio_144x.intr_150 in_intr	
	gpio_144_0_bank_intr_7	mux_GPIO_151	in_intr_151	gpio_144x.intr_151 in_intr	
	gpio_144_0_bank_intr_8	mux_GPIO_152	in_intr_152	gpio_144x.intr_152 in_intr	
	gpio_144_1_bank_intr_0	mux_GPIO_153	in_intr_153	gpio_144x.intr_153 in_intr	
	gpio_144_1_bank_intr_1	mux_GPIO_154	in_intr_154	gpio_144x.intr_154 in_intr	
	gpio_144_1_bank_intr_2	mux_GPIO_155	in_intr_155	gpio_144x.intr_155 in_intr	
	gpio_144_1_bank_intr_3	mux_GPIO_156	in_intr_156	gpio_144x.intr_156 in_intr	

Table 10-16. GPIO_XBAR_INTRTR0 in_intr Hardware Requests (continued)

Module Instance	Source Module	Source in_intr signal	XBAR Module in_intr	Description	Type
	gpio_144_1_bank_intr_4	mux_GPIO_157	in_intr_157	gpio_144x.intr_157 in_intr	
	gpio_144_1_bank_intr_5	mux_GPIO_158	in_intr_158	gpio_144x.intr_158 in_intr	
	gpio_144_1_bank_intr_6	mux_GPIO_159	in_intr_159	gpio_144x.intr_159 in_intr	
	gpio_144_1_bank_intr_7	mux_GPIO_160	in_intr_160	gpio_144x.intr_160 in_intr	
	gpio_144_1_bank_intr_8	mux_GPIO_161	in_intr_161	gpio_144x.intr_161 in_intr	
	gpio_144_2_bank_intr_0	mux_GPIO_162	in_intr_162	gpio_144x.intr_162 in_intr	
	gpio_144_2_bank_intr_1	mux_GPIO_163	in_intr_163	gpio_144x.intr_163 in_intr	
	gpio_144_2_bank_intr_2	mux_GPIO_164	in_intr_164	gpio_144x.intr_164 in_intr	
	gpio_144_2_bank_intr_3	mux_GPIO_165	in_intr_165	gpio_144x.intr_165 in_intr	
	gpio_144_0_bank_intr_4	mux_GPIO_166	in_intr_166	gpio_144x.intr_166 in_intr	
	gpio_144_2_bank_intr_5	mux_GPIO_167	in_intr_167	gpio_144x.intr_167 in_intr	
	gpio_144_2_bank_intr_6	mux_GPIO_168	in_intr_168	gpio_144x.intr_168 in_intr	
	gpio_144_2_bank_intr_7	mux_GPIO_169	in_intr_169	gpio_144x.intr_169 in_intr	
	gpio_144_2_bank_intr_8	mux_GPIO_170	in_intr_170	gpio_144x.intr_170 in_intr	
	gpio_144_3_bank_intr_0	mux_GPIO_171	in_intr_171	gpio_144x.intr_171 in_intr	
	gpio_144_3_bank_intr_1	mux_GPIO_172	in_intr_172	gpio_144x.intr_172 in_intr	
	gpio_144_3_bank_intr_2	mux_GPIO_173	in_intr_173	gpio_144x.intr_173 in_intr	
	gpio_144_3_bank_intr_3	mux_GPIO_174	in_intr_174	gpio_144x.intr_174 in_intr	
	gpio_144_3_bank_intr_4	mux_GPIO_175	in_intr_175	gpio_144x.intr_175 in_intr	
	gpio_144_3_bank_intr_5	mux_GPIO_176	in_intr_176	gpio_144x.intr_176 in_intr	
	gpio_144_3_bank_intr_6	mux_GPIO_177	in_intr_177	gpio_144x.intr_177 in_intr	
	gpio_144_3_bank_intr_7	mux_GPIO_178	in_intr_178	gpio_144x.intr_178 in_intr	
	gpio_144_3_bank_intr_8	mux_GPIO_179	in_intr_179	gpio_144x.intr_179 in_intr	

10.4 Interrupt Sources

10.4.1 R5FSS0_CORE0 Interrupt Map

Table 10-17 shows the mapping of events to the R5FSS0_CORE0.

Both R5FSS0_CORE0 and R5FSS0_CORE1 use the R5FSS0_CORE0 interrupt map when operating in lockstep mode.

Table 10-17. R5FSS0_CORE0 Interrupt Map

Interrupt Input Line	Interrupt ID	Source Interrupt
R5FSS0_CORE0_INTR_IN_0	0	R5FSS0_CORE0_INTR_PRU_ICSSM0_PR1_HOST_INTR_PEND_0
R5FSS0_CORE0_INTR_IN_1	1	R5FSS0_CORE0_INTR_PRU_ICSSM0_PR1_HOST_INTR_PEND_1
R5FSS0_CORE0_INTR_IN_2	2	R5FSS0_CORE0_INTR_PRU_ICSSM0_PR1_HOST_INTR_PEND_2
R5FSS0_CORE0_INTR_IN_3	3	R5FSS0_CORE0_INTR_PRU_ICSSM0_PR1_HOST_INTR_PEND_3
R5FSS0_CORE0_INTR_IN_4	4	R5FSS0_CORE0_INTR_PRU_ICSSM0_PR1_HOST_INTR_PEND_4
R5FSS0_CORE0_INTR_IN_5	5	R5FSS0_CORE0_INTR_PRU_ICSSM0_PR1_HOST_INTR_PEND_5
R5FSS0_CORE0_INTR_IN_6	6	R5FSS0_CORE0_INTR_PRU_ICSSM0_PR1_HOST_INTR_PEND_6
R5FSS0_CORE0_INTR_IN_7	7	R5FSS0_CORE0_INTR_PRU_ICSSM0_PR1_HOST_INTR_PEND_7
R5FSS0_CORE0_INTR_IN_8	8	R5FSS0_CORE0_INTR_PRU_ICSSM0_PR1_RX_SOF_INTR_REQ_0
R5FSS0_CORE0_INTR_IN_9	9	R5FSS0_CORE0_INTR_PRU_ICSSM0_PR1_RX_SOF_INTR_REQ_1
R5FSS0_CORE0_INTR_IN_10	10	R5FSS0_CORE0_INTR_PRU_ICSSM0_PR1_TX_SOF_INTR_REQ_0
R5FSS0_CORE0_INTR_IN_11	11	R5FSS0_CORE0_INTR_PRU_ICSSM0_PR1_TX_SOF_INTR_REQ_1
R5FSS0_CORE0_INTR_IN_12	12	R5FSS0_CORE0_INTR_CPSW0_FH_INTR
R5FSS0_CORE0_INTR_IN_13	13	R5FSS0_CORE0_INTR_CPSW0_TH_INTR
R5FSS0_CORE0_INTR_IN_14	14	R5FSS0_CORE0_INTR_CPSW0_TH_THRESH_INTR
R5FSS0_CORE0_INTR_IN_15	15	R5FSS0_CORE0_INTR_CPSW0_MISC_INTR
R5FSS0_CORE0_INTR_IN_16	16	R5FSS0_CORE0_INTR_LIN0_INTR_0
R5FSS0_CORE0_INTR_IN_17	17	R5FSS0_CORE0_INTR_LIN0_INTR_1
R5FSS0_CORE0_INTR_IN_18	18	R5FSS0_CORE0_INTR_LIN1_INTR_0
R5FSS0_CORE0_INTR_IN_19	19	R5FSS0_CORE0_INTR_LIN1_INTR_1
R5FSS0_CORE0_INTR_IN_20	20	R5FSS0_CORE0_INTR_LIN2_INTR_0
R5FSS0_CORE0_INTR_IN_21	21	R5FSS0_CORE0_INTR_LIN2_INTR_1
R5FSS0_CORE0_INTR_IN_22	22	R5FSS0_CORE0_INTR_LIN3_INTR_0
R5FSS0_CORE0_INTR_IN_23	23	R5FSS0_CORE0_INTR_LIN3_INTR_1
R5FSS0_CORE0_INTR_IN_24	24	R5FSS0_CORE0_INTR_LIN4_INTR_0
R5FSS0_CORE0_INTR_IN_25	25	R5FSS0_CORE0_INTR_LIN4_INTR_1
R5FSS0_CORE0_INTR_IN_26	26	R5FSS0_CORE0_INTR_MCAN0_EXT_TS_ROLLOVER_LVL_INT_0
R5FSS0_CORE0_INTR_IN_27	27	R5FSS0_CORE0_INTR_MCAN0_MCAN_LVL_INT_0
R5FSS0_CORE0_INTR_IN_28	28	R5FSS0_CORE0_INTR_MCAN0_MCAN_LVL_INT_1
R5FSS0_CORE0_INTR_IN_29	29	R5FSS0_CORE0_INTR_MCAN1_EXT_TS_ROLLOVER_LVL_INT_0
R5FSS0_CORE0_INTR_IN_30	30	R5FSS0_CORE0_INTR_MCAN1_MCAN_LVL_INT_0
R5FSS0_CORE0_INTR_IN_31	31	R5FSS0_CORE0_INTR_MCAN1_MCAN_LVL_INT_1
R5FSS0_CORE0_INTR_IN_32	32	R5FSS0_CORE0_INTR_MCAN2_EXT_TS_ROLLOVER_LVL_INT_0
R5FSS0_CORE0_INTR_IN_33	33	R5FSS0_CORE0_INTR_MCAN2_MCAN_LVL_INT_0
R5FSS0_CORE0_INTR_IN_34	34	R5FSS0_CORE0_INTR_MCAN2_MCAN_LVL_INT_1
R5FSS0_CORE0_INTR_IN_35	35	R5FSS0_CORE0_INTR_MCAN3_EXT_TS_ROLLOVER_LVL_INT_0
R5FSS0_CORE0_INTR_IN_36	36	R5FSS0_CORE0_INTR_MCAN3_MCAN_LVL_INT_0
R5FSS0_CORE0_INTR_IN_37	37	R5FSS0_CORE0_INTR_MCAN3_MCAN_LVL_INT_1
R5FSS0_CORE0_INTR_IN_38	38	R5FSS0_CORE0_INTR_UART0_IRQ
R5FSS0_CORE0_INTR_IN_39	39	R5FSS0_CORE0_INTR_UART1_IRQ
R5FSS0_CORE0_INTR_IN_40	40	R5FSS0_CORE0_INTR_UART2_IRQ
R5FSS0_CORE0_INTR_IN_41	41	R5FSS0_CORE0_INTR_UART3_IRQ

Table 10-17. R5FSS0_CORE0 Interrupt Map (continued)

Interrupt Input Line	Interrupt ID	Source Interrupt
R5FSS0_CORE0_INTR_IN_42	42	R5FSS0_CORE0_INTR_UART4_IRQ
R5FSS0_CORE0_INTR_IN_43	43	R5FSS0_CORE0_INTR_UART5_IRQ
R5FSS0_CORE0_INTR_IN_44	44	R5FSS0_CORE0_INTR_I2C0_IRQ
R5FSS0_CORE0_INTR_IN_45	45	R5FSS0_CORE0_INTR_I2C1_IRQ
R5FSS0_CORE0_INTR_IN_46	46	R5FSS0_CORE0_INTR_I2C2_IRQ
R5FSS0_CORE0_INTR_IN_47	47	R5FSS0_CORE0_INTR_I2C3_IRQ
R5FSS0_CORE0_INTR_IN_48	48	R5FSS0_CORE0_INTR_DTHE_SHA_S_INT
R5FSS0_CORE0_INTR_IN_49	49	R5FSS0_CORE0_INTR_DTHE_SHA_P_INT
R5FSS0_CORE0_INTR_IN_50	50	R5FSS0_CORE0_INTR_DTHE_TRNG_INT
R5FSS0_CORE0_INTR_IN_51	51	R5FSS0_CORE0_INTR_DTHE_PKAE_INT
R5FSS0_CORE0_INTR_IN_52	52	R5FSS0_CORE0_INTR_DTHE_AES_S_INT
R5FSS0_CORE0_INTR_IN_53	53	R5FSS0_CORE0_INTR_DTHE_AES_P_INT
R5FSS0_CORE0_INTR_IN_54	54	R5FSS0_CORE0_INTR_OSPI0_INT
R5FSS0_CORE0_INTR_IN_55	55	R5FSS0_CORE0_INTR_TPCC0_INTG
R5FSS0_CORE0_INTR_IN_56	56	R5FSS0_CORE0_INTR_TPCC0_INT_0
R5FSS0_CORE0_INTR_IN_57	57	R5FSS0_CORE0_INTR_TPCC0_INT_1
R5FSS0_CORE0_INTR_IN_58	58	R5FSS0_CORE0_INTR_TPCC0_INT_2
R5FSS0_CORE0_INTR_IN_59	59	R5FSS0_CORE0_INTR_TPCC0_INT_3
R5FSS0_CORE0_INTR_IN_60	60	R5FSS0_CORE0_INTR_TPCC0_INT_4
R5FSS0_CORE0_INTR_IN_61	61	R5FSS0_CORE0_INTR_TPCC0_INT_5
R5FSS0_CORE0_INTR_IN_62	62	R5FSS0_CORE0_INTR_TPCC0_INT_6
R5FSS0_CORE0_INTR_IN_63	63	R5FSS0_CORE0_INTR_TPCC0_INT_7
R5FSS0_CORE0_INTR_IN_64	64	R5FSS0_CORE0_INTR_TPCC0_ERRINT
R5FSS0_CORE0_INTR_IN_65	65	R5FSS0_CORE0_INTR_TPCC0_MPINT
R5FSS0_CORE0_INTR_IN_66	66	R5FSS0_CORE0_INTR_TPTC0_ERINT_0
R5FSS0_CORE0_INTR_IN_67	67	R5FSS0_CORE0_INTR_TPTC0_ERINT_1
R5FSS0_CORE0_INTR_IN_68	68	R5FSS0_CORE0_INTR_MCRC0_INT
R5FSS0_CORE0_INTR_IN_69	69	R5FSS0_CORE0_INTR_MPU_ADDR_ERRAGG
R5FSS0_CORE0_INTR_IN_70	70	R5FSS0_CORE0_INTR_MPU_PROT_ERRAGG
R5FSS0_CORE0_INTR_IN_71	71	R5FSS0_CORE0_INTR_PBIST_DONE
R5FSS0_CORE0_INTR_IN_72	72	R5FSS0_CORE0_INTR_TPCC0_INTAGGR
R5FSS0_CORE0_INTR_IN_73	73	R5FSS0_CORE0_INTR_TPCC0_ERRAGGR
R5FSS0_CORE0_INTR_IN_74	74	R5FSS0_CORE0_INTR_DCC0_DONE
R5FSS0_CORE0_INTR_IN_75	75	R5FSS0_CORE0_INTR_DCC1_DONE
R5FSS0_CORE0_INTR_IN_76	76	R5FSS0_CORE0_INTR_DCC2_DONE
R5FSS0_CORE0_INTR_IN_77	77	R5FSS0_CORE0_INTR_DCC3_DONE
R5FSS0_CORE0_INTR_IN_78	78	R5FSS0_CORE0_INTR_MCSPI0_INTR
R5FSS0_CORE0_INTR_IN_79	79	R5FSS0_CORE0_INTR_MCSPI1_INTR
R5FSS0_CORE0_INTR_IN_80	80	R5FSS0_CORE0_INTR_MCSPI2_INTR
R5FSS0_CORE0_INTR_IN_81	81	R5FSS0_CORE0_INTR_MCSPI3_INTR
R5FSS0_CORE0_INTR_IN_82	82	R5FSS0_CORE0_INTR_MCSPI4_INTR
R5FSS0_CORE0_INTR_IN_83	83	R5FSS0_CORE0_INTR_MMC0_INTR
R5FSS0_CORE0_INTR_IN_84	84	R5FSS0_CORE0_INTR_RTI0_INTR_0
R5FSS0_CORE0_INTR_IN_85	85	R5FSS0_CORE0_INTR_RTI0_INTR_1
R5FSS0_CORE0_INTR_IN_86	86	R5FSS0_CORE0_INTR_RTI0_INTR_2
R5FSS0_CORE0_INTR_IN_87	87	R5FSS0_CORE0_INTR_RTI0_INTR_3
R5FSS0_CORE0_INTR_IN_88	88	R5FSS0_CORE0_INTR_RESERVED

Table 10-17. R5FSS0_CORE0 Interrupt Map (continued)

Interrupt Input Line	Interrupt ID	Source Interrupt
R5FSS0_CORE0_INTR_IN_89	89	R5FSS0_CORE0_INTR_RTI0_OVERFLOW_INT0
R5FSS0_CORE0_INTR_IN_90	90	R5FSS0_CORE0_INTR_RTI0_OVERFLOW_INT1
R5FSS0_CORE0_INTR_IN_91	91	R5FSS0_CORE0_INTR_RTI1_INTR_0
R5FSS0_CORE0_INTR_IN_92	92	R5FSS0_CORE0_INTR_RTI1_INTR_1
R5FSS0_CORE0_INTR_IN_93	93	R5FSS0_CORE0_INTR_RTI1_INTR_2
R5FSS0_CORE0_INTR_IN_94	94	R5FSS0_CORE0_INTR_RTI1_INTR_3
R5FSS0_CORE0_INTR_IN_95	95	R5FSS0_CORE0_INTR_RESERVED
R5FSS0_CORE0_INTR_IN_96	96	R5FSS0_CORE0_INTR_RTI1_OVERFLOW_INT0
R5FSS0_CORE0_INTR_IN_97	97	R5FSS0_CORE0_INTR_RTI1_OVERFLOW_INT1
R5FSS0_CORE0_INTR_IN_98	98	R5FSS0_CORE0_INTR_RTI2_INTR_0
R5FSS0_CORE0_INTR_IN_99	99	R5FSS0_CORE0_INTR_RTI2_INTR_1
R5FSS0_CORE0_INTR_IN_100	100	R5FSS0_CORE0_INTR_RTI2_INTR_2
R5FSS0_CORE0_INTR_IN_101	101	R5FSS0_CORE0_INTR_RTI2_INTR_3
R5FSS0_CORE0_INTR_IN_102	102	R5FSS0_CORE0_INTR_RESERVED
R5FSS0_CORE0_INTR_IN_103	103	R5FSS0_CORE0_INTR_RTI2_OVERFLOW_INT0
R5FSS0_CORE0_INTR_IN_104	104	R5FSS0_CORE0_INTR_RTI2_OVERFLOW_INT1
R5FSS0_CORE0_INTR_IN_105	105	R5FSS0_CORE0_INTR_RTI3_INTR_0
R5FSS0_CORE0_INTR_IN_106	106	R5FSS0_CORE0_INTR_RTI3_INTR_1
R5FSS0_CORE0_INTR_IN_107	107	R5FSS0_CORE0_INTR_RTI3_INTR_2
R5FSS0_CORE0_INTR_IN_108	108	R5FSS0_CORE0_INTR_RTI3_INTR_3
R5FSS0_CORE0_INTR_IN_109	109	R5FSS0_CORE0_INTR_RESERVED
R5FSS0_CORE0_INTR_IN_110	110	R5FSS0_CORE0_INTR_RTI3_OVERFLOW_INT0
R5FSS0_CORE0_INTR_IN_111	111	R5FSS0_CORE0_INTR_RTI3_OVERFLOW_INT1
R5FSS0_CORE0_INTR_IN_112	112	R5FSS0_CORE0_INTR_RESERVED
R5FSS0_CORE0_INTR_IN_113	113	R5FSS0_CORE0_INTR_ESM0_ESM_INT_CFG
R5FSS0_CORE0_INTR_IN_114	114	R5FSS0_CORE0_INTR_ESM0_ESM_INT_HI
R5FSS0_CORE0_INTR_IN_115	115	R5FSS0_CORE0_INTR_ESM0_ESM_INT_LOW
R5FSS0_CORE0_INTR_IN_116	116	R5FSS0_CORE0_INTR_R5SS0_COMMRX_0
R5FSS0_CORE0_INTR_IN_117	117	R5FSS0_CORE0_INTR_R5SS0_COMMTX_0
R5FSS0_CORE0_INTR_IN_118	118	R5FSS0_CORE0_INTR_R5SS0_CPU0_CTI_INT
R5FSS0_CORE0_INTR_IN_119	119	R5FSS0_CORE0_INTR_R5SS0_CPU0_VALFIQ
R5FSS0_CORE0_INTR_IN_120	120	R5FSS0_CORE0_INTR_R5SS0_CPU0_VALIRQ
R5FSS0_CORE0_INTR_IN_121	121	R5FSS0_CORE0_INTR_R5SS0_CPU1_CTI_INT
R5FSS0_CORE0_INTR_IN_122	122	R5FSS0_CORE0_INTR_R5SS1_CPU0_PMU_INT
R5FSS0_CORE0_INTR_IN_123	123	R5FSS0_CORE0_INTR_R5SS1_CPU1_PMU_INT
R5FSS0_CORE0_INTR_IN_124	124	R5FSS0_CORE0_INTR_MMR_ACC_ERRAGG
R5FSS0_CORE0_INTR_IN_125	125	R5FSS0_CORE0_INTR_R5SS0_LIVELOCK_1
R5FSS0_CORE0_INTR_IN_126	126	R5FSS0_CORE0_INTR_R5SS1_LIVELOCK_0
R5FSS0_CORE0_INTR_IN_127	127	R5FSS0_CORE0_INTR_R5SS1_LIVELOCK_1
R5FSS0_CORE0_INTR_IN_128	128	R5FSS0_CORE0_INTR_RTI_WDT0_NMI
R5FSS0_CORE0_INTR_IN_129	129	R5FSS0_CORE0_INTR_SW_IRQ
R5FSS0_CORE0_INTR_IN_130	130	R5FSS0_CORE0_INTR_R5SS0_CORE0_FPU_EXP
R5FSS0_CORE0_INTR_IN_131	131	R5FSS0_CORE0_INTR_DEBUGSS_TXDATA_AVAIL
R5FSS0_CORE0_INTR_IN_132	132	R5FSS0_CORE0_INTR_DEBUGSS_R5SS1_STC_DONE
R5FSS0_CORE0_INTR_IN_133	133	R5FSS0_CORE0_INTR_TSENSE_H
R5FSS0_CORE0_INTR_IN_134	134	R5FSS0_CORE0_INTR_TSENSE_L
R5FSS0_CORE0_INTR_IN_135	135	R5FSS0_CORE0_INTR_AHB_WRITE_ERR

Table 10-17. R5FSS0_CORE0 Interrupt Map (continued)

Interrupt Input Line	Interrupt ID	Source Interrupt
R5FSS0_CORE0_INTR_IN_136	136	R5FSS0_CORE0_INTR_MBOX_READ_REQ
R5FSS0_CORE0_INTR_IN_137	137	R5FSS0_CORE0_INTR_MBOX_READ_ACK
R5FSS0_CORE0_INTR_IN_138	138	R5FSS0_CORE0_INTR_SOC_TIMESYNCXBAR1_OUT_2
R5FSS0_CORE0_INTR_IN_139	139	R5FSS0_CORE0_INTR_SOC_TIMESYNCXBAR1_OUT_3
R5FSS0_CORE0_INTR_IN_140	140	R5FSS0_CORE0_INTR_SOC_TIMESYNCXBAR1_OUT_4
R5FSS0_CORE0_INTR_IN_141	141	R5FSS0_CORE0_INTR_SOC_TIMESYNCXBAR1_OUT_5
R5FSS0_CORE0_INTR_IN_142	142	R5FSS0_CORE0_INTR_GPIO_INTRXBAR_OUT_14
R5FSS0_CORE0_INTR_IN_143	143	R5FSS0_CORE0_INTR_GPIO_INTRXBAR_OUT_15
R5FSS0_CORE0_INTR_IN_144	144	R5FSS0_CORE0_INTR_GPIO_INTRXBAR_OUT_16
R5FSS0_CORE0_INTR_IN_145	145	R5FSS0_CORE0_INTR_GPIO_INTRXBAR_OUT_17
R5FSS0_CORE0_INTR_IN_146	146	R5FSS0_CORE0_CONTROLSS_INTRXBAR0_OUT_0
R5FSS0_CORE0_INTR_IN_147	147	R5FSS0_CORE0_CONTROLSS_INTRXBAR0_OUT_1
R5FSS0_CORE0_INTR_IN_148	148	R5FSS0_CORE0_CONTROLSS_INTRXBAR0_OUT_2
R5FSS0_CORE0_INTR_IN_149	149	R5FSS0_CORE0_CONTROLSS_INTRXBAR0_OUT_3
R5FSS0_CORE0_INTR_IN_150	150	R5FSS0_CORE0_CONTROLSS_INTRXBAR0_OUT_4
R5FSS0_CORE0_INTR_IN_151	151	R5FSS0_CORE0_CONTROLSS_INTRXBAR0_OUT_5
R5FSS0_CORE0_INTR_IN_152	152	R5FSS0_CORE0_CONTROLSS_INTRXBAR0_OUT_6
R5FSS0_CORE0_INTR_IN_153	153	R5FSS0_CORE0_CONTROLSS_INTRXBAR0_OUT_7
R5FSS0_CORE0_INTR_IN_154	154	R5FSS0_CORE0_CONTROLSS_INTRXBAR0_OUT_8
R5FSS0_CORE0_INTR_IN_155	155	R5FSS0_CORE0_CONTROLSS_INTRXBAR0_OUT_9
R5FSS0_CORE0_INTR_IN_156	156	R5FSS0_CORE0_CONTROLSS_INTRXBAR0_OUT_10
R5FSS0_CORE0_INTR_IN_157	157	R5FSS0_CORE0_CONTROLSS_INTRXBAR0_OUT_11
R5FSS0_CORE0_INTR_IN_158	158	R5FSS0_CORE0_CONTROLSS_INTRXBAR0_OUT_12
R5FSS0_CORE0_INTR_IN_159	159	R5FSS0_CORE0_CONTROLSS_INTRXBAR0_OUT_13
R5FSS0_CORE0_INTR_IN_160	160	R5FSS0_CORE0_CONTROLSS_INTRXBAR0_OUT_14
R5FSS0_CORE0_INTR_IN_161	161	R5FSS0_CORE0_CONTROLSS_INTRXBAR0_OUT_15
R5FSS0_CORE0_INTR_IN_162	162	R5FSS0_CORE0_CONTROLSS_INTRXBAR0_OUT_16
R5FSS0_CORE0_INTR_IN_163	163	R5FSS0_CORE0_CONTROLSS_INTRXBAR0_OUT_17
R5FSS0_CORE0_INTR_IN_164	164	R5FSS0_CORE0_CONTROLSS_INTRXBAR0_OUT_18
R5FSS0_CORE0_INTR_IN_165	165	R5FSS0_CORE0_CONTROLSS_INTRXBAR0_OUT_19
R5FSS0_CORE0_INTR_IN_166	166	R5FSS0_CORE0_CONTROLSS_INTRXBAR0_OUT_20
R5FSS0_CORE0_INTR_IN_167	167	R5FSS0_CORE0_CONTROLSS_INTRXBAR0_OUT_21
R5FSS0_CORE0_INTR_IN_168	168	R5FSS0_CORE0_CONTROLSS_INTRXBAR0_OUT_22
R5FSS0_CORE0_INTR_IN_169	169	R5FSS0_CORE0_CONTROLSS_INTRXBAR0_OUT_23
R5FSS0_CORE0_INTR_IN_170	170	R5FSS0_CORE0_CONTROLSS_INTRXBAR0_OUT_24
R5FSS0_CORE0_INTR_IN_171	171	R5FSS0_CORE0_CONTROLSS_INTRXBAR0_OUT_25
R5FSS0_CORE0_INTR_IN_172	172	R5FSS0_CORE0_CONTROLSS_INTRXBAR0_OUT_26
R5FSS0_CORE0_INTR_IN_173	173	R5FSS0_CORE0_CONTROLSS_INTRXBAR0_OUT_27
R5FSS0_CORE0_INTR_IN_174	174	R5FSS0_CORE0_CONTROLSS_INTRXBAR0_OUT_28
R5FSS0_CORE0_INTR_IN_175	175	R5FSS0_CORE0_CONTROLSS_INTRXBAR0_OUT_29
R5FSS0_CORE0_INTR_IN_176	176	R5FSS0_CORE0_CONTROLSS_INTRXBAR0_OUT_30
R5FSS0_CORE0_INTR_IN_177	177	R5FSS0_CORE0_CONTROLSS_INTRXBAR0_OUT_31
R5FSS0_CORE0_INTR_IN_178	178	R5FSS0_CORE0_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_0
R5FSS0_CORE0_INTR_IN_179	179	R5FSS0_CORE0_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_1
R5FSS0_CORE0_INTR_IN_180	180	R5FSS0_CORE0_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_2
R5FSS0_CORE0_INTR_IN_181	181	R5FSS0_CORE0_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_3
R5FSS0_CORE0_INTR_IN_182	182	R5FSS0_CORE0_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_4

Table 10-17. R5FSS0_CORE0 Interrupt Map (continued)

Interrupt Input Line	Interrupt ID	Source Interrupt
R5FSS0_CORE0_INTR_IN_183	183	R5FSS0_CORE0_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_5
R5FSS0_CORE0_INTR_IN_184	184	R5FSS0_CORE0_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_6
R5FSS0_CORE0_INTR_IN_185	185	R5FSS0_CORE0_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_7
R5FSS0_CORE0_INTR_IN_186	186	R5FSS0_CORE0_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_8
R5FSS0_CORE0_INTR_IN_187	187	R5FSS0_CORE0_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_9
R5FSS0_CORE0_INTR_IN_188	188	R5FSS0_CORE0_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_10
R5FSS0_CORE0_INTR_IN_189	189	R5FSS0_CORE0_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_11
R5FSS0_CORE0_INTR_IN_190	190	R5FSS0_CORE0_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_12
R5FSS0_CORE0_INTR_IN_191	191	R5FSS0_CORE0_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_13
R5FSS0_CORE0_INTR_IN_192	192	R5FSS0_CORE0_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_14
R5FSS0_CORE0_INTR_IN_193	193	R5FSS0_CORE0_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_15
R5FSS0_CORE0_INTR_IN_194	194	R5FSS0_CORE0_CPSW0_CPTS_COMP
R5FSS0_CORE0_INTR_IN_195	195	R5FSS0_CORE0_INTR_RESERVED
R5FSS0_CORE0_INTR_IN_196	196	R5FSS0_CORE0_INTR_RESERVED
R5FSS0_CORE0_INTR_IN_197	197	R5FSS0_CORE0_INTR_MCAN4_EXT_TS_ROLLOVER_LVL_INT_0
R5FSS0_CORE0_INTR_IN_198	198	R5FSS0_CORE0_INTR_MCAN4_MCAN_LVL_INT_0
R5FSS0_CORE0_INTR_IN_199	199	R5FSS0_CORE0_INTR_MCAN4_MCAN_LVL_INT_1
R5FSS0_CORE0_INTR_IN_200	200	R5FSS0_CORE0_INTR_MCAN5_EXT_TS_ROLLOVER_LVL_INT_0
R5FSS0_CORE0_INTR_IN_201	201	R5FSS0_CORE0_INTR_MCAN5_MCAN_LVL_INT_0
R5FSS0_CORE0_INTR_IN_202	202	R5FSS0_CORE0_INTR_MCAN5_MCAN_LVL_INT_1
R5FSS0_CORE0_INTR_IN_203	203	R5FSS0_CORE0_INTR_MCAN6_EXT_TS_ROLLOVER_LVL_INT_0
R5FSS0_CORE0_INTR_IN_204	204	R5FSS0_CORE0_INTR_MCAN6_MCAN_LVL_INT_0
R5FSS0_CORE0_INTR_IN_205	205	R5FSS0_CORE0_INTR_MCAN6_MCAN_LVL_INT_1
R5FSS0_CORE0_INTR_IN_206	206	R5FSS0_CORE0_INTR_MCAN7_EXT_TS_ROLLOVER_LVL_INT_0
R5FSS0_CORE0_INTR_IN_207	207	R5FSS0_CORE0_INTR_MCAN7_MCAN_LVL_INT_0
R5FSS0_CORE0_INTR_IN_208	208	R5FSS0_CORE0_INTR_MCAN7_MCAN_LVL_INT_1
R5FSS0_CORE0_INTR_IN_209	209	R5FSS0_CORE0_INTR_R5SS0_CPU0_TMU_LVF
R5FSS0_CORE0_INTR_IN_210	210	R5FSS0_CORE0_INTR_R5SS0_CPU0_TMU_LUF
R5FSS0_CORE0_INTR_IN_211	211	R5FSS0_CORE0_INTR_HW_RESOLVER
R5FSS0_CORE0_INTR_IN_212	212	R5FSS0_CORE0_INTR_FSS_VBUSM_TIMEOUT
R5FSS0_CORE0_INTR_IN_213	213	R5FSS0_CORE0_INTR_OTFA_ERROR
R5FSS0_CORE0_INTR_IN_214	214	R5FSS0_CORE0_INTR_FOTA_STAT
R5FSS0_CORE0_INTR_IN_215	215	R5FSS0_CORE0_INTR_FOTA_STAT_ERR
R5FSS0_CORE0_INTR_IN_216	216	R5FSS0_CORE0_INTR_MCSPI5_INTR
R5FSS0_CORE0_INTR_IN_217	217	R5FSS0_CORE0_INTR_MCSPI6_INTR
R5FSS0_CORE0_INTR_IN_218	218	R5FSS0_CORE0_INTR_MCSPI7_INTR
R5FSS0_CORE0_INTR_IN_219	219	R5FSS0_CORE0_INTR_RTI4_INTR_0
R5FSS0_CORE0_INTR_IN_220	220	R5FSS0_CORE0_INTR_RTI4_INTR_1
R5FSS0_CORE0_INTR_IN_221	221	R5FSS0_CORE0_INTR_RTI4_INTR_2
R5FSS0_CORE0_INTR_IN_222	222	R5FSS0_CORE0_INTR_RTI4_INTR_3
R5FSS0_CORE0_INTR_IN_223	223	R5FSS0_CORE0_INTR_RTI4_OVERFLOW_INT0
R5FSS0_CORE0_INTR_IN_224	224	R5FSS0_CORE0_INTR_RTI4_OVERFLOW_INT1
R5FSS0_CORE0_INTR_IN_225	225	R5FSS0_CORE0_INTR_RTI5_INTR_0
R5FSS0_CORE0_INTR_IN_226	226	R5FSS0_CORE0_INTR_RTI5_INTR_1
R5FSS0_CORE0_INTR_IN_227	227	R5FSS0_CORE0_INTR_RTI5_INTR_2
R5FSS0_CORE0_INTR_IN_228	228	R5FSS0_CORE0_INTR_RTI5_INTR_3
R5FSS0_CORE0_INTR_IN_229	229	R5FSS0_CORE0_INTR_RTI5_OVERFLOW_INT0

Table 10-17. R5FSS0_CORE0 Interrupt Map (continued)

Interrupt Input Line	Interrupt ID	Source Interrupt
R5FSS0_CORE0_INTR_IN_230	230	R5FSS0_CORE0_INTR_RTI5_OVERFLOW_INT1
R5FSS0_CORE0_INTR_IN_231	231	R5FSS0_CORE0_INTR_RTI6_INTR_0
R5FSS0_CORE0_INTR_IN_232	232	R5FSS0_CORE0_INTR_RTI6_INTR_1
R5FSS0_CORE0_INTR_IN_233	233	R5FSS0_CORE0_INTR_RTI6_INTR_2
R5FSS0_CORE0_INTR_IN_234	234	R5FSS0_CORE0_INTR_RTI6_INTR_3
R5FSS0_CORE0_INTR_IN_235	235	R5FSS0_CORE0_INTR_RTI6_OVERFLOW_INT0
R5FSS0_CORE0_INTR_IN_236	236	R5FSS0_CORE0_INTR_RTI6_OVERFLOW_INT1
R5FSS0_CORE0_INTR_IN_237	237	R5FSS0_CORE0_INTR_RTI7_INTR_0
R5FSS0_CORE0_INTR_IN_238	238	R5FSS0_CORE0_INTR_RTI7_INTR_1
R5FSS0_CORE0_INTR_IN_239	239	R5FSS0_CORE0_INTR_RTI7_INTR_2
R5FSS0_CORE0_INTR_IN_240	240	R5FSS0_CORE0_INTR_RTI7_INTR_3
R5FSS0_CORE0_INTR_IN_241	241	R5FSS0_CORE0_INTR_RTI7_OVERFLOW_INT0
R5FSS0_CORE0_INTR_IN_242	242	R5FSS0_CORE0_INTR_RTI7_OVERFLOW_INT1
R5FSS0_CORE0_INTR_IN_243	243	R5FSS0_CORE0_INTR_R5SS0_CPU0_RL2_ERR
R5FSS0_CORE0_INTR_IN_244	244	R5FSS0_CORE0_INTR_R5SS0_CPU1_RL2_ERR
R5FSS0_CORE0_INTR_IN_245	245	R5FSS0_CORE0_INTR_R5SS1_CPU0_RL2_ERR
R5FSS0_CORE0_INTR_IN_246	246	R5FSS0_CORE0_INTR_R5SS1_CPU1_RL2_ERR

10.4.2 R5FSS0_CORE1 Interrupt Map

Table 10-18 shows the mapping of events to the R5FSS0_CORE1.

Both R5FSS0_CORE1 and R5FSS0_CORE0 use the R5FSS0_CORE0 interrupt map when operating in lockstep mode.

Table 10-18. R5FSS0_CORE1 Interrupt Map

Interrupt Input Line	Interrupt ID	Source Interrupt
R5FSS0_CORE1_INTR_IN_0	0	R5FSS0_CORE1_INTR_PRU_ICSSM0_PR1_HOST_INTR_PEND_0
R5FSS0_CORE1_INTR_IN_1	1	R5FSS0_CORE1_INTR_PRU_ICSSM0_PR1_HOST_INTR_PEND_1
R5FSS0_CORE1_INTR_IN_2	2	R5FSS0_CORE1_INTR_PRU_ICSSM0_PR1_HOST_INTR_PEND_2
R5FSS0_CORE1_INTR_IN_3	3	R5FSS0_CORE1_INTR_PRU_ICSSM0_PR1_HOST_INTR_PEND_3
R5FSS0_CORE1_INTR_IN_4	4	R5FSS0_CORE1_INTR_PRU_ICSSM0_PR1_HOST_INTR_PEND_4
R5FSS0_CORE1_INTR_IN_5	5	R5FSS0_CORE1_INTR_PRU_ICSSM0_PR1_HOST_INTR_PEND_5
R5FSS0_CORE1_INTR_IN_6	6	R5FSS0_CORE1_INTR_PRU_ICSSM0_PR1_HOST_INTR_PEND_6
R5FSS0_CORE1_INTR_IN_7	7	R5FSS0_CORE1_INTR_PRU_ICSSM0_PR1_HOST_INTR_PEND_7
R5FSS0_CORE1_INTR_IN_8	8	R5FSS0_CORE1_INTR_PRU_ICSSM0_PR1_RX_SOF_INTR_REQ_0
R5FSS0_CORE1_INTR_IN_9	9	R5FSS0_CORE1_INTR_PRU_ICSSM0_PR1_RX_SOF_INTR_REQ_1
R5FSS0_CORE1_INTR_IN_10	10	R5FSS0_CORE1_INTR_PRU_ICSSM0_PR1_TX_SOF_INTR_REQ_0
R5FSS0_CORE1_INTR_IN_11	11	R5FSS0_CORE1_INTR_PRU_ICSSM0_PR1_TX_SOF_INTR_REQ_1
R5FSS0_CORE1_INTR_IN_12	12	R5FSS0_CORE1_INTR_CPSW0_FH_INTR
R5FSS0_CORE1_INTR_IN_13	13	R5FSS0_CORE1_INTR_CPSW0_TH_INTR
R5FSS0_CORE1_INTR_IN_14	14	R5FSS0_CORE1_INTR_CPSW0_TH_THRESH_INTR
R5FSS0_CORE1_INTR_IN_15	15	R5FSS0_CORE1_INTR_CPSW0_MISC_INTR
R5FSS0_CORE1_INTR_IN_16	16	R5FSS0_CORE1_INTR_LIN0_INTR_0
R5FSS0_CORE1_INTR_IN_17	17	R5FSS0_CORE1_INTR_LIN0_INTR_1
R5FSS0_CORE1_INTR_IN_18	18	R5FSS0_CORE1_INTR_LIN1_INTR_0
R5FSS0_CORE1_INTR_IN_19	19	R5FSS0_CORE1_INTR_LIN1_INTR_1
R5FSS0_CORE1_INTR_IN_20	20	R5FSS0_CORE1_INTR_LIN2_INTR_0
R5FSS0_CORE1_INTR_IN_21	21	R5FSS0_CORE1_INTR_LIN2_INTR_1
R5FSS0_CORE1_INTR_IN_22	22	R5FSS0_CORE1_INTR_LIN3_INTR_0
R5FSS0_CORE1_INTR_IN_23	23	R5FSS0_CORE1_INTR_LIN3_INTR_1
R5FSS0_CORE1_INTR_IN_24	24	R5FSS0_CORE1_INTR_LIN4_INTR_0
R5FSS0_CORE1_INTR_IN_25	25	R5FSS0_CORE1_INTR_LIN4_INTR_1
R5FSS0_CORE1_INTR_IN_26	26	R5FSS0_CORE1_INTR_MCAN0_EXT_TS_ROLLOVER_LVL_INT_0
R5FSS0_CORE1_INTR_IN_27	27	R5FSS0_CORE1_INTR_MCAN0_MCAN_LVL_INT_0
R5FSS0_CORE1_INTR_IN_28	28	R5FSS0_CORE1_INTR_MCAN0_MCAN_LVL_INT_1
R5FSS0_CORE1_INTR_IN_29	29	R5FSS0_CORE1_INTR_MCAN1_EXT_TS_ROLLOVER_LVL_INT_0
R5FSS0_CORE1_INTR_IN_30	30	R5FSS0_CORE1_INTR_MCAN1_MCAN_LVL_INT_0
R5FSS0_CORE1_INTR_IN_31	31	R5FSS0_CORE1_INTR_MCAN1_MCAN_LVL_INT_1
R5FSS0_CORE1_INTR_IN_32	32	R5FSS0_CORE1_INTR_MCAN2_EXT_TS_ROLLOVER_LVL_INT_0
R5FSS0_CORE1_INTR_IN_33	33	R5FSS0_CORE1_INTR_MCAN2_MCAN_LVL_INT_0
R5FSS0_CORE1_INTR_IN_34	34	R5FSS0_CORE1_INTR_MCAN2_MCAN_LVL_INT_1
R5FSS0_CORE1_INTR_IN_35	35	R5FSS0_CORE1_INTR_MCAN3_EXT_TS_ROLLOVER_LVL_INT_0
R5FSS0_CORE1_INTR_IN_36	36	R5FSS0_CORE1_INTR_MCAN3_MCAN_LVL_INT_0
R5FSS0_CORE1_INTR_IN_37	37	R5FSS0_CORE1_INTR_MCAN3_MCAN_LVL_INT_1
R5FSS0_CORE1_INTR_IN_38	38	R5FSS0_CORE1_INTR_UART0_IRQ
R5FSS0_CORE1_INTR_IN_39	39	R5FSS0_CORE1_INTR_UART1_IRQ
R5FSS0_CORE1_INTR_IN_40	40	R5FSS0_CORE1_INTR_UART2_IRQ
R5FSS0_CORE1_INTR_IN_41	41	R5FSS0_CORE1_INTR_UART3_IRQ

Table 10-18. R5FSS0_CORE1 Interrupt Map (continued)

Interrupt Input Line	Interrupt ID	Source Interrupt
R5FSS0_CORE1_INTR_IN_42	42	R5FSS0_CORE1_INTR_UART4_IRQ
R5FSS0_CORE1_INTR_IN_43	43	R5FSS0_CORE1_INTR_UART5_IRQ
R5FSS0_CORE1_INTR_IN_44	44	R5FSS0_CORE1_INTR_I2C0_IRQ
R5FSS0_CORE1_INTR_IN_45	45	R5FSS0_CORE1_INTR_I2C1_IRQ
R5FSS0_CORE1_INTR_IN_46	46	R5FSS0_CORE1_INTR_I2C2_IRQ
R5FSS0_CORE1_INTR_IN_47	47	R5FSS0_CORE1_INTR_I2C3_IRQ
R5FSS0_CORE1_INTR_IN_48	48	R5FSS0_CORE1_INTR_DTHE_SHA_S_INT
R5FSS0_CORE1_INTR_IN_49	49	R5FSS0_CORE1_INTR_DTHE_SHA_P_INT
R5FSS0_CORE1_INTR_IN_50	50	R5FSS0_CORE1_INTR_DTHE_TRNG_INT
R5FSS0_CORE1_INTR_IN_51	51	R5FSS0_CORE1_INTR_DTHE_PKAE_INT
R5FSS0_CORE1_INTR_IN_52	52	R5FSS0_CORE1_INTR_DTHE_AES_S_INT
R5FSS0_CORE1_INTR_IN_53	53	R5FSS0_CORE1_INTR_DTHE_AES_P_INT
R5FSS0_CORE1_INTR_IN_54	54	R5FSS0_CORE1_INTR_OSPI0_INT
R5FSS0_CORE1_INTR_IN_55	55	R5FSS0_CORE1_INTR_TPCC0_INTG
R5FSS0_CORE1_INTR_IN_56	56	R5FSS0_CORE1_INTR_TPCC0_INT_0
R5FSS0_CORE1_INTR_IN_57	57	R5FSS0_CORE1_INTR_TPCC0_INT_1
R5FSS0_CORE1_INTR_IN_58	58	R5FSS0_CORE1_INTR_TPCC0_INT_2
R5FSS0_CORE1_INTR_IN_59	59	R5FSS0_CORE1_INTR_TPCC0_INT_3
R5FSS0_CORE1_INTR_IN_60	60	R5FSS0_CORE1_INTR_TPCC0_INT_4
R5FSS0_CORE1_INTR_IN_61	61	R5FSS0_CORE1_INTR_TPCC0_INT_5
R5FSS0_CORE1_INTR_IN_62	62	R5FSS0_CORE1_INTR_TPCC0_INT_6
R5FSS0_CORE1_INTR_IN_63	63	R5FSS0_CORE1_INTR_TPCC0_INT_7
R5FSS0_CORE1_INTR_IN_64	64	R5FSS0_CORE1_INTR_TPCC0_ERRINT
R5FSS0_CORE1_INTR_IN_65	65	R5FSS0_CORE1_INTR_TPCC0_MPINT
R5FSS0_CORE1_INTR_IN_66	66	R5FSS0_CORE1_INTR_TPTC0_ERINT_0
R5FSS0_CORE1_INTR_IN_67	67	R5FSS0_CORE1_INTR_TPTC0_ERINT_1
R5FSS0_CORE1_INTR_IN_68	68	R5FSS0_CORE1_INTR_MCRC0_INT
R5FSS0_CORE1_INTR_IN_69	69	R5FSS0_CORE1_INTR_MPU_ADDR_ERRAGG
R5FSS0_CORE1_INTR_IN_70	70	R5FSS0_CORE1_INTR_MPU_PROT_ERRAGG
R5FSS0_CORE1_INTR_IN_71	71	R5FSS0_CORE1_INTR_PBIST_DONE
R5FSS0_CORE1_INTR_IN_72	72	R5FSS0_CORE1_INTR_TPCC0_INTAGGR
R5FSS0_CORE1_INTR_IN_73	73	R5FSS0_CORE1_INTR_TPCC0_ERRAGGR
R5FSS0_CORE1_INTR_IN_74	74	R5FSS0_CORE1_INTR_DCC0_DONE
R5FSS0_CORE1_INTR_IN_75	75	R5FSS0_CORE1_INTR_DCC1_DONE
R5FSS0_CORE1_INTR_IN_76	76	R5FSS0_CORE1_INTR_DCC2_DONE
R5FSS0_CORE1_INTR_IN_77	77	R5FSS0_CORE1_INTR_DCC3_DONE
R5FSS0_CORE1_INTR_IN_78	78	R5FSS0_CORE1_INTR_MCSPI0_INTR
R5FSS0_CORE1_INTR_IN_79	79	R5FSS0_CORE1_INTR_MCSPI1_INTR
R5FSS0_CORE1_INTR_IN_80	80	R5FSS0_CORE1_INTR_MCSPI2_INTR
R5FSS0_CORE1_INTR_IN_81	81	R5FSS0_CORE1_INTR_MCSPI3_INTR
R5FSS0_CORE1_INTR_IN_82	82	R5FSS0_CORE1_INTR_MCSPI4_INTR
R5FSS0_CORE1_INTR_IN_83	83	R5FSS0_CORE1_INTR_MMC0_INTR
R5FSS0_CORE1_INTR_IN_84	84	R5FSS0_CORE1_INTR_RTI0_INTR_0
R5FSS0_CORE1_INTR_IN_85	85	R5FSS0_CORE1_INTR_RTI0_INTR_1
R5FSS0_CORE1_INTR_IN_86	86	R5FSS0_CORE1_INTR_RTI0_INTR_2
R5FSS0_CORE1_INTR_IN_87	87	R5FSS0_CORE1_INTR_RTI0_INTR_3
R5FSS0_CORE1_INTR_IN_88	88	R5FSS0_CORE1_INTR_RESERVED

Table 10-18. R5FSS0_CORE1 Interrupt Map (continued)

Interrupt Input Line	Interrupt ID	Source Interrupt
R5FSS0_CORE1_INTR_IN_89	89	R5FSS0_CORE1_INTR_RTI0_OVERFLOW_INT0
R5FSS0_CORE1_INTR_IN_90	90	R5FSS0_CORE1_INTR_RTI0_OVERFLOW_INT1
R5FSS0_CORE1_INTR_IN_91	91	R5FSS0_CORE1_INTR_RTI1_INTR_0
R5FSS0_CORE1_INTR_IN_92	92	R5FSS0_CORE1_INTR_RTI1_INTR_1
R5FSS0_CORE1_INTR_IN_93	93	R5FSS0_CORE1_INTR_RTI1_INTR_2
R5FSS0_CORE1_INTR_IN_94	94	R5FSS0_CORE1_INTR_RTI1_INTR_3
R5FSS0_CORE1_INTR_IN_95	95	R5FSS0_CORE1_INTR_RESERVED
R5FSS0_CORE1_INTR_IN_96	96	R5FSS0_CORE1_INTR_RTI1_OVERFLOW_INT0
R5FSS0_CORE1_INTR_IN_97	97	R5FSS0_CORE1_INTR_RTI1_OVERFLOW_INT1
R5FSS0_CORE1_INTR_IN_98	98	R5FSS0_CORE1_INTR_RTI2_INTR_0
R5FSS0_CORE1_INTR_IN_99	99	R5FSS0_CORE1_INTR_RTI2_INTR_1
R5FSS0_CORE1_INTR_IN_100	100	R5FSS0_CORE1_INTR_RTI2_INTR_2
R5FSS0_CORE1_INTR_IN_101	101	R5FSS0_CORE1_INTR_RTI2_INTR_3
R5FSS0_CORE1_INTR_IN_102	102	R5FSS0_CORE1_INTR_RESERVED
R5FSS0_CORE1_INTR_IN_103	103	R5FSS0_CORE1_INTR_RTI2_OVERFLOW_INT0
R5FSS0_CORE1_INTR_IN_104	104	R5FSS0_CORE1_INTR_RTI2_OVERFLOW_INT1
R5FSS0_CORE1_INTR_IN_105	105	R5FSS0_CORE1_INTR_RTI3_INTR_0
R5FSS0_CORE1_INTR_IN_106	106	R5FSS0_CORE1_INTR_RTI3_INTR_1
R5FSS0_CORE1_INTR_IN_107	107	R5FSS0_CORE1_INTR_RTI3_INTR_2
R5FSS0_CORE1_INTR_IN_108	108	R5FSS0_CORE1_INTR_RTI3_INTR_3
R5FSS0_CORE1_INTR_IN_109	109	R5FSS0_CORE1_INTR_RESERVED
R5FSS0_CORE1_INTR_IN_110	110	R5FSS0_CORE1_INTR_RTI3_OVERFLOW_INT0
R5FSS0_CORE1_INTR_IN_111	111	R5FSS0_CORE1_INTR_RTI3_OVERFLOW_INT1
R5FSS0_CORE1_INTR_IN_112	112	R5FSS0_CORE1_INTR_RESERVED
R5FSS0_CORE1_INTR_IN_113	113	R5FSS0_CORE1_INTR_ESM0_ESM_INT_CFG
R5FSS0_CORE1_INTR_IN_114	114	R5FSS0_CORE1_INTR_ESM0_ESM_INT_HI
R5FSS0_CORE1_INTR_IN_115	115	R5FSS0_CORE1_INTR_ESM0_ESM_INT_LOW
R5FSS0_CORE1_INTR_IN_116	116	R5FSS0_CORE1_INTR_R5SS0_COMMRX_1
R5FSS0_CORE1_INTR_IN_117	117	R5FSS0_CORE1_INTR_R5SS0_COMMTX_1
R5FSS0_CORE1_INTR_IN_118	118	R5FSS0_CORE1_INTR_R5SS0_CPU0_CTI_INT
R5FSS0_CORE1_INTR_IN_119	119	R5FSS0_CORE1_INTR_R5SS0_CPU1_CTI_INT
R5FSS0_CORE1_INTR_IN_120	120	R5FSS0_CORE1_INTR_R5SS0_CPU1_VALFIQ
R5FSS0_CORE1_INTR_IN_121	121	R5FSS0_CORE1_INTR_R5SS0_CPU1_VALIRQ
R5FSS0_CORE1_INTR_IN_122	122	R5FSS0_CORE1_INTR_R5SS1_CPU0_PMU_INT
R5FSS0_CORE1_INTR_IN_123	123	R5FSS0_CORE1_INTR_R5SS1_CPU1_PMU_INT
R5FSS0_CORE1_INTR_IN_124	124	R5FSS0_CORE1_INTR_MMR_ACC_ERRAGG
R5FSS0_CORE1_INTR_IN_125	125	R5FSS0_CORE1_INTR_R5SS0_LIVELOCK_0
R5FSS0_CORE1_INTR_IN_126	126	R5FSS0_CORE1_INTR_R5SS1_LIVELOCK_0
R5FSS0_CORE1_INTR_IN_127	127	R5FSS0_CORE1_INTR_R5SS1_LIVELOCK_1
R5FSS0_CORE1_INTR_IN_128	128	R5FSS0_CORE1_INTR_RTI_WDT1_NMI
R5FSS0_CORE1_INTR_IN_129	129	R5FSS0_CORE1_INTR_SW_IRQ
R5FSS0_CORE1_INTR_IN_130	130	R5FSS0_CORE1_INTR_R5SS0_CORE1_FPU_EXP
R5FSS0_CORE1_INTR_IN_131	131	R5FSS0_CORE1_INTR_DEBUGSS_TXDATA_AVAIL
R5FSS0_CORE1_INTR_IN_132	132	R5FSS0_CORE1_INTR_DEBUGSS_R5SS1_STC_DONE
R5FSS0_CORE1_INTR_IN_133	133	R5FSS0_CORE1_INTR_TSENSE_H
R5FSS0_CORE1_INTR_IN_134	134	R5FSS0_CORE1_INTR_TSENSE_L
R5FSS0_CORE1_INTR_IN_135	135	R5FSS0_CORE1_INTR_AHB_WRITE_ERR

Table 10-18. R5FSS0_CORE1 Interrupt Map (continued)

Interrupt Input Line	Interrupt ID	Source Interrupt
R5FSS0_CORE1_INTR_IN_136	136	R5FSS0_CORE1_INTR_MBOX_READ_REQ
R5FSS0_CORE1_INTR_IN_137	137	R5FSS0_CORE1_INTR_MBOX_READ_ACK
R5FSS0_CORE1_INTR_IN_138	138	R5FSS0_CORE1_INTR_SOC_TIMESYNCXBAR1_OUT_6
R5FSS0_CORE1_INTR_IN_139	139	R5FSS0_CORE1_INTR_SOC_TIMESYNCXBAR1_OUT_7
R5FSS0_CORE1_INTR_IN_140	140	R5FSS0_CORE1_INTR_SOC_TIMESYNCXBAR1_OUT_8
R5FSS0_CORE1_INTR_IN_141	141	R5FSS0_CORE1_INTR_SOC_TIMESYNCXBAR1_OUT_9
R5FSS0_CORE1_INTR_IN_142	142	R5FSS0_CORE1_INTR_GPIO_INTRXBAR_OUT_18
R5FSS0_CORE1_INTR_IN_143	143	R5FSS0_CORE1_INTR_GPIO_INTRXBAR_OUT_19
R5FSS0_CORE1_INTR_IN_144	144	R5FSS0_CORE1_INTR_GPIO_INTRXBAR_OUT_20
R5FSS0_CORE1_INTR_IN_145	145	R5FSS0_CORE1_INTR_GPIO_INTRXBAR_OUT_21
R5FSS0_CORE1_INTR_IN_146	146	R5FSS0_CORE1_CONTROLSS_INTRXBAR0_OUT_0
R5FSS0_CORE1_INTR_IN_147	147	R5FSS0_CORE1_CONTROLSS_INTRXBAR0_OUT_1
R5FSS0_CORE1_INTR_IN_148	148	R5FSS0_CORE1_CONTROLSS_INTRXBAR0_OUT_2
R5FSS0_CORE1_INTR_IN_149	149	R5FSS0_CORE1_CONTROLSS_INTRXBAR0_OUT_3
R5FSS0_CORE1_INTR_IN_150	150	R5FSS0_CORE1_CONTROLSS_INTRXBAR0_OUT_4
R5FSS0_CORE1_INTR_IN_151	151	R5FSS0_CORE1_CONTROLSS_INTRXBAR0_OUT_5
R5FSS0_CORE1_INTR_IN_152	152	R5FSS0_CORE1_CONTROLSS_INTRXBAR0_OUT_6
R5FSS0_CORE1_INTR_IN_153	153	R5FSS0_CORE1_CONTROLSS_INTRXBAR0_OUT_7
R5FSS0_CORE1_INTR_IN_154	154	R5FSS0_CORE1_CONTROLSS_INTRXBAR0_OUT_8
R5FSS0_CORE1_INTR_IN_155	155	R5FSS0_CORE1_CONTROLSS_INTRXBAR0_OUT_9
R5FSS0_CORE1_INTR_IN_156	156	R5FSS0_CORE1_CONTROLSS_INTRXBAR0_OUT_10
R5FSS0_CORE1_INTR_IN_157	157	R5FSS0_CORE1_CONTROLSS_INTRXBAR0_OUT_11
R5FSS0_CORE1_INTR_IN_158	158	R5FSS0_CORE1_CONTROLSS_INTRXBAR0_OUT_12
R5FSS0_CORE1_INTR_IN_159	159	R5FSS0_CORE1_CONTROLSS_INTRXBAR0_OUT_13
R5FSS0_CORE1_INTR_IN_160	160	R5FSS0_CORE1_CONTROLSS_INTRXBAR0_OUT_14
R5FSS0_CORE1_INTR_IN_161	161	R5FSS0_CORE1_CONTROLSS_INTRXBAR0_OUT_15
R5FSS0_CORE1_INTR_IN_162	162	R5FSS0_CORE1_CONTROLSS_INTRXBAR0_OUT_16
R5FSS0_CORE1_INTR_IN_163	163	R5FSS0_CORE1_CONTROLSS_INTRXBAR0_OUT_17
R5FSS0_CORE1_INTR_IN_164	164	R5FSS0_CORE1_CONTROLSS_INTRXBAR0_OUT_18
R5FSS0_CORE1_INTR_IN_165	165	R5FSS0_CORE1_CONTROLSS_INTRXBAR0_OUT_19
R5FSS0_CORE1_INTR_IN_166	166	R5FSS0_CORE1_CONTROLSS_INTRXBAR0_OUT_20
R5FSS0_CORE1_INTR_IN_167	167	R5FSS0_CORE1_CONTROLSS_INTRXBAR0_OUT_21
R5FSS0_CORE1_INTR_IN_168	168	R5FSS0_CORE1_CONTROLSS_INTRXBAR0_OUT_22
R5FSS0_CORE1_INTR_IN_169	169	R5FSS0_CORE1_CONTROLSS_INTRXBAR0_OUT_23
R5FSS0_CORE1_INTR_IN_170	170	R5FSS0_CORE1_CONTROLSS_INTRXBAR0_OUT_24
R5FSS0_CORE1_INTR_IN_171	171	R5FSS0_CORE1_CONTROLSS_INTRXBAR0_OUT_25
R5FSS0_CORE1_INTR_IN_172	172	R5FSS0_CORE1_CONTROLSS_INTRXBAR0_OUT_26
R5FSS0_CORE1_INTR_IN_173	173	R5FSS0_CORE1_CONTROLSS_INTRXBAR0_OUT_27
R5FSS0_CORE1_INTR_IN_174	174	R5FSS0_CORE1_CONTROLSS_INTRXBAR0_OUT_28
R5FSS0_CORE1_INTR_IN_175	175	R5FSS0_CORE1_CONTROLSS_INTRXBAR0_OUT_29
R5FSS0_CORE1_INTR_IN_176	176	R5FSS0_CORE1_CONTROLSS_INTRXBAR0_OUT_30
R5FSS0_CORE1_INTR_IN_177	177	R5FSS0_CORE1_CONTROLSS_INTRXBAR0_OUT_31
R5FSS0_CORE1_INTR_IN_178	178	R5FSS0_CORE1_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_0
R5FSS0_CORE1_INTR_IN_179	179	R5FSS0_CORE1_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_1
R5FSS0_CORE1_INTR_IN_180	180	R5FSS0_CORE1_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_2
R5FSS0_CORE1_INTR_IN_181	181	R5FSS0_CORE1_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_3
R5FSS0_CORE1_INTR_IN_182	182	R5FSS0_CORE1_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_4

Table 10-18. R5FSS0_CORE1 Interrupt Map (continued)

Interrupt Input Line	Interrupt ID	Source Interrupt
R5FSS0_CORE1_INTR_IN_183	183	R5FSS0_CORE1_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_5
R5FSS0_CORE1_INTR_IN_184	184	R5FSS0_CORE1_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_6
R5FSS0_CORE1_INTR_IN_185	185	R5FSS0_CORE1_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_7
R5FSS0_CORE1_INTR_IN_186	186	R5FSS0_CORE1_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_8
R5FSS0_CORE1_INTR_IN_187	187	R5FSS0_CORE1_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_9
R5FSS0_CORE1_INTR_IN_188	188	R5FSS0_CORE1_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_10
R5FSS0_CORE1_INTR_IN_189	189	R5FSS0_CORE1_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_11
R5FSS0_CORE1_INTR_IN_190	190	R5FSS0_CORE1_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_12
R5FSS0_CORE1_INTR_IN_191	191	R5FSS0_CORE1_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_13
R5FSS0_CORE1_INTR_IN_192	192	R5FSS0_CORE1_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_14
R5FSS0_CORE1_INTR_IN_193	193	R5FSS0_CORE1_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_15
R5FSS0_CORE1_INTR_IN_194	194	R5FSS0_CORE1_CPSW0_CPTS_COMP
R5FSS0_CORE1_INTR_IN_195	195	R5FSS0_CORE1_INTR_RESERVED
R5FSS0_CORE1_INTR_IN_196	196	R5FSS0_CORE1_INTR_RESERVED
R5FSS0_CORE1_INTR_IN_197	197	R5FSS0_CORE1_INTR_MCAN4_EXT_TS_ROLLOVER_LVL_INT_0
R5FSS0_CORE1_INTR_IN_198	198	R5FSS0_CORE1_INTR_MCAN4_MCAN_LVL_INT_0
R5FSS0_CORE1_INTR_IN_199	199	R5FSS0_CORE1_INTR_MCAN4_MCAN_LVL_INT_1
R5FSS0_CORE1_INTR_IN_200	200	R5FSS0_CORE1_INTR_MCAN5_EXT_TS_ROLLOVER_LVL_INT_0
R5FSS0_CORE1_INTR_IN_201	201	R5FSS0_CORE1_INTR_MCAN5_MCAN_LVL_INT_0
R5FSS0_CORE1_INTR_IN_202	202	R5FSS0_CORE1_INTR_MCAN5_MCAN_LVL_INT_1
R5FSS0_CORE1_INTR_IN_203	203	R5FSS0_CORE1_INTR_MCAN6_EXT_TS_ROLLOVER_LVL_INT_0
R5FSS0_CORE1_INTR_IN_204	204	R5FSS0_CORE1_INTR_MCAN6_MCAN_LVL_INT_0
R5FSS0_CORE1_INTR_IN_205	205	R5FSS0_CORE1_INTR_MCAN6_MCAN_LVL_INT_1
R5FSS0_CORE1_INTR_IN_206	206	R5FSS0_CORE1_INTR_MCAN7_EXT_TS_ROLLOVER_LVL_INT_0
R5FSS0_CORE1_INTR_IN_207	207	R5FSS0_CORE1_INTR_MCAN7_MCAN_LVL_INT_0
R5FSS0_CORE1_INTR_IN_208	208	R5FSS0_CORE1_INTR_MCAN7_MCAN_LVL_INT_1
R5FSS0_CORE1_INTR_IN_209	209	R5FSS0_CORE1_INTR_R5SS0_CPU0_TMU_LVF
R5FSS0_CORE1_INTR_IN_210	210	R5FSS0_CORE1_INTR_R5SS0_CPU0_TMU_LUF
R5FSS0_CORE1_INTR_IN_211	211	R5FSS0_CORE1_INTR_HW_RESOLVER
R5FSS0_CORE1_INTR_IN_212	212	R5FSS0_CORE1_INTR_FSS_VBUSM_TIMEOUT
R5FSS0_CORE1_INTR_IN_213	213	R5FSS0_CORE1_INTR_OTFA_ERROR
R5FSS0_CORE1_INTR_IN_214	214	R5FSS0_CORE1_INTR_FOTA_STAT
R5FSS0_CORE1_INTR_IN_215	215	R5FSS0_CORE1_INTR_FOTA_STAT_ERR
R5FSS0_CORE1_INTR_IN_216	216	R5FSS0_CORE1_INTR_MCSPI5_INTR
R5FSS0_CORE1_INTR_IN_217	217	R5FSS0_CORE1_INTR_MCSPI6_INTR
R5FSS0_CORE1_INTR_IN_218	218	R5FSS0_CORE1_INTR_MCSPI7_INTR
R5FSS0_CORE1_INTR_IN_219	219	R5FSS0_CORE1_INTR_RTI4_INTR_0
R5FSS0_CORE1_INTR_IN_220	220	R5FSS0_CORE1_INTR_RTI4_INTR_1
R5FSS0_CORE1_INTR_IN_221	221	R5FSS0_CORE1_INTR_RTI4_INTR_2
R5FSS0_CORE1_INTR_IN_222	222	R5FSS0_CORE1_INTR_RTI4_INTR_3
R5FSS0_CORE1_INTR_IN_223	223	R5FSS0_CORE1_INTR_RTI4_OVERFLOW_INT0
R5FSS0_CORE1_INTR_IN_224	224	R5FSS0_CORE1_INTR_RTI4_OVERFLOW_INT1
R5FSS0_CORE1_INTR_IN_225	225	R5FSS0_CORE1_INTR_RTI5_INTR_0
R5FSS0_CORE1_INTR_IN_226	226	R5FSS0_CORE1_INTR_RTI5_INTR_1
R5FSS0_CORE1_INTR_IN_227	227	R5FSS0_CORE1_INTR_RTI5_INTR_2
R5FSS0_CORE1_INTR_IN_228	228	R5FSS0_CORE1_INTR_RTI5_INTR_3
R5FSS0_CORE1_INTR_IN_229	229	R5FSS0_CORE1_INTR_RTI5_OVERFLOW_INT0

Table 10-18. R5FSS0_CORE1 Interrupt Map (continued)

Interrupt Input Line	Interrupt ID	Source Interrupt
R5FSS0_CORE1_INTR_IN_230	230	R5FSS0_CORE1_INTR_RTI5_OVERFLOW_INT1
R5FSS0_CORE1_INTR_IN_231	231	R5FSS0_CORE1_INTR_RTI6_INTR_0
R5FSS0_CORE1_INTR_IN_232	232	R5FSS0_CORE1_INTR_RTI6_INTR_1
R5FSS0_CORE1_INTR_IN_233	233	R5FSS0_CORE1_INTR_RTI6_INTR_2
R5FSS0_CORE1_INTR_IN_234	234	R5FSS0_CORE1_INTR_RTI6_INTR_3
R5FSS0_CORE1_INTR_IN_235	235	R5FSS0_CORE1_INTR_RTI6_OVERFLOW_INT0
R5FSS0_CORE1_INTR_IN_236	236	R5FSS0_CORE1_INTR_RTI6_OVERFLOW_INT1
R5FSS0_CORE1_INTR_IN_237	237	R5FSS0_CORE1_INTR_RTI7_INTR_0
R5FSS0_CORE1_INTR_IN_238	238	R5FSS0_CORE1_INTR_RTI7_INTR_1
R5FSS0_CORE1_INTR_IN_239	239	R5FSS0_CORE1_INTR_RTI7_INTR_2
R5FSS0_CORE1_INTR_IN_240	240	R5FSS0_CORE1_INTR_RTI7_INTR_3
R5FSS0_CORE1_INTR_IN_241	241	R5FSS0_CORE1_INTR_RTI7_OVERFLOW_INT0
R5FSS0_CORE1_INTR_IN_242	242	R5FSS0_CORE1_INTR_RTI7_OVERFLOW_INT1
R5FSS0_CORE1_INTR_IN_243	243	R5FSS0_CORE1_INTR_R5SS0_CPU0_RL2_ERR
R5FSS0_CORE1_INTR_IN_244	244	R5FSS0_CORE1_INTR_R5SS0_CPU1_RL2_ERR
R5FSS0_CORE1_INTR_IN_245	245	R5FSS0_CORE1_INTR_R5SS1_CPU0_RL2_ERR
R5FSS0_CORE1_INTR_IN_246	246	R5FSS0_CORE1_INTR_R5SS1_CPU1_RL2_ERR

10.4.3 R5FSS1_CORE0 Interrupt Map

Table 10-19 shows the mapping of events to the R5FSS1_CORE0.

Both R5FSS1_CORE0 and R5FSS1_CORE1 use the R5FSS1_CORE0 interrupt map when operating in lockstep mode.

Table 10-19. R5FSS1_CORE0 Interrupt Map

Interrupt Input Line	Interrupt ID	Source Interrupt
R5FSS1_CORE0_INTR_IN_0	0	R5FSS1_CORE0_INTR_PRU_ICSSM0_PR1_HOST_INTR_PEND_0
R5FSS1_CORE0_INTR_IN_1	1	R5FSS1_CORE0_INTR_PRU_ICSSM0_PR1_HOST_INTR_PEND_1
R5FSS1_CORE0_INTR_IN_2	2	R5FSS1_CORE0_INTR_PRU_ICSSM0_PR1_HOST_INTR_PEND_2
R5FSS1_CORE0_INTR_IN_3	3	R5FSS1_CORE0_INTR_PRU_ICSSM0_PR1_HOST_INTR_PEND_3
R5FSS1_CORE0_INTR_IN_4	4	R5FSS1_CORE0_INTR_PRU_ICSSM0_PR1_HOST_INTR_PEND_4
R5FSS1_CORE0_INTR_IN_5	5	R5FSS1_CORE0_INTR_PRU_ICSSM0_PR1_HOST_INTR_PEND_5
R5FSS1_CORE0_INTR_IN_6	6	R5FSS1_CORE0_INTR_PRU_ICSSM0_PR1_HOST_INTR_PEND_6
R5FSS1_CORE0_INTR_IN_7	7	R5FSS1_CORE0_INTR_PRU_ICSSM0_PR1_HOST_INTR_PEND_7
R5FSS1_CORE0_INTR_IN_8	8	R5FSS1_CORE0_INTR_PRU_ICSSM0_PR1_RX_SOF_INTR_REQ_0
R5FSS1_CORE0_INTR_IN_9	9	R5FSS1_CORE0_INTR_PRU_ICSSM0_PR1_RX_SOF_INTR_REQ_1
R5FSS1_CORE0_INTR_IN_10	10	R5FSS1_CORE0_INTR_PRU_ICSSM0_PR1_TX_SOF_INTR_REQ_0
R5FSS1_CORE0_INTR_IN_11	11	R5FSS1_CORE0_INTR_PRU_ICSSM0_PR1_TX_SOF_INTR_REQ_1
R5FSS1_CORE0_INTR_IN_12	12	R5FSS1_CORE0_INTR_CPSW0_FH_INTR
R5FSS1_CORE0_INTR_IN_13	13	R5FSS1_CORE0_INTR_CPSW0_TH_INTR
R5FSS1_CORE0_INTR_IN_14	14	R5FSS1_CORE0_INTR_CPSW0_TH_THRESH_INTR
R5FSS1_CORE0_INTR_IN_15	15	R5FSS1_CORE0_INTR_CPSW0_MISC_INTR
R5FSS1_CORE0_INTR_IN_16	16	R5FSS1_CORE0_INTR_LIN0_INTR_0
R5FSS1_CORE0_INTR_IN_17	17	R5FSS1_CORE0_INTR_LIN0_INTR_1
R5FSS1_CORE0_INTR_IN_18	18	R5FSS1_CORE0_INTR_LIN1_INTR_0
R5FSS1_CORE0_INTR_IN_19	19	R5FSS1_CORE0_INTR_LIN1_INTR_1
R5FSS1_CORE0_INTR_IN_20	20	R5FSS1_CORE0_INTR_LIN2_INTR_0
R5FSS1_CORE0_INTR_IN_21	21	R5FSS1_CORE0_INTR_LIN2_INTR_1
R5FSS1_CORE0_INTR_IN_22	22	R5FSS1_CORE0_INTR_LIN3_INTR_0
R5FSS1_CORE0_INTR_IN_23	23	R5FSS1_CORE0_INTR_LIN3_INTR_1
R5FSS1_CORE0_INTR_IN_24	24	R5FSS1_CORE0_INTR_LIN4_INTR_0
R5FSS1_CORE0_INTR_IN_25	25	R5FSS1_CORE0_INTR_LIN4_INTR_1
R5FSS1_CORE0_INTR_IN_26	26	R5FSS1_CORE0_INTR_MCAN0_EXT_TS_ROLLOVER_LVL_INT_0
R5FSS1_CORE0_INTR_IN_27	27	R5FSS1_CORE0_INTR_MCAN0_MCAN_LVL_INT_0
R5FSS1_CORE0_INTR_IN_28	28	R5FSS1_CORE0_INTR_MCAN0_MCAN_LVL_INT_1
R5FSS1_CORE0_INTR_IN_29	29	R5FSS1_CORE0_INTR_MCAN1_EXT_TS_ROLLOVER_LVL_INT_0
R5FSS1_CORE0_INTR_IN_30	30	R5FSS1_CORE0_INTR_MCAN1_MCAN_LVL_INT_0
R5FSS1_CORE0_INTR_IN_31	31	R5FSS1_CORE0_INTR_MCAN1_MCAN_LVL_INT_1
R5FSS1_CORE0_INTR_IN_32	32	R5FSS1_CORE0_INTR_MCAN2_EXT_TS_ROLLOVER_LVL_INT_0
R5FSS1_CORE0_INTR_IN_33	33	R5FSS1_CORE0_INTR_MCAN2_MCAN_LVL_INT_0
R5FSS1_CORE0_INTR_IN_34	34	R5FSS1_CORE0_INTR_MCAN2_MCAN_LVL_INT_1
R5FSS1_CORE0_INTR_IN_35	35	R5FSS1_CORE0_INTR_MCAN3_EXT_TS_ROLLOVER_LVL_INT_0
R5FSS1_CORE0_INTR_IN_36	36	R5FSS1_CORE0_INTR_MCAN3_MCAN_LVL_INT_0
R5FSS1_CORE0_INTR_IN_37	37	R5FSS1_CORE0_INTR_MCAN3_MCAN_LVL_INT_1
R5FSS1_CORE0_INTR_IN_38	38	R5FSS1_CORE0_INTR_UART0_IRQ
R5FSS1_CORE0_INTR_IN_39	39	R5FSS1_CORE0_INTR_UART1_IRQ
R5FSS1_CORE0_INTR_IN_40	40	R5FSS1_CORE0_INTR_UART2_IRQ
R5FSS1_CORE0_INTR_IN_41	41	R5FSS1_CORE0_INTR_UART3_IRQ

Table 10-19. R5FSS1_CORE0 Interrupt Map (continued)

Interrupt Input Line	Interrupt ID	Source Interrupt
R5FSS1_CORE0_INTR_IN_42	42	R5FSS1_CORE0_INTR_UART4_IRQ
R5FSS1_CORE0_INTR_IN_43	43	R5FSS1_CORE0_INTR_UART5_IRQ
R5FSS1_CORE0_INTR_IN_44	44	R5FSS1_CORE0_INTR_I2C0_IRQ
R5FSS1_CORE0_INTR_IN_45	45	R5FSS1_CORE0_INTR_I2C1_IRQ
R5FSS1_CORE0_INTR_IN_46	46	R5FSS1_CORE0_INTR_I2C2_IRQ
R5FSS1_CORE0_INTR_IN_47	47	R5FSS1_CORE0_INTR_I2C3_IRQ
R5FSS1_CORE0_INTR_IN_48	48	R5FSS1_CORE0_INTR_DTHE_SHA_S_INT
R5FSS1_CORE0_INTR_IN_49	49	R5FSS1_CORE0_INTR_DTHE_SHA_P_INT
R5FSS1_CORE0_INTR_IN_50	50	R5FSS1_CORE0_INTR_DTHE_TRNG_INT
R5FSS1_CORE0_INTR_IN_51	51	R5FSS1_CORE0_INTR_DTHE_PKAE_INT
R5FSS1_CORE0_INTR_IN_52	52	R5FSS1_CORE0_INTR_DTHE_AES_S_INT
R5FSS1_CORE0_INTR_IN_53	53	R5FSS1_CORE0_INTR_DTHE_AES_P_INT
R5FSS1_CORE0_INTR_IN_54	54	R5FSS1_CORE0_INTR_OSPI0_INT
R5FSS1_CORE0_INTR_IN_55	55	R5FSS1_CORE0_INTR_TPCC0_INTG
R5FSS1_CORE0_INTR_IN_56	56	R5FSS1_CORE0_INTR_TPCC0_INT_0
R5FSS1_CORE0_INTR_IN_57	57	R5FSS1_CORE0_INTR_TPCC0_INT_1
R5FSS1_CORE0_INTR_IN_58	58	R5FSS1_CORE0_INTR_TPCC0_INT_2
R5FSS1_CORE0_INTR_IN_59	59	R5FSS1_CORE0_INTR_TPCC0_INT_3
R5FSS1_CORE0_INTR_IN_60	60	R5FSS1_CORE0_INTR_TPCC0_INT_4
R5FSS1_CORE0_INTR_IN_61	61	R5FSS1_CORE0_INTR_TPCC0_INT_5
R5FSS1_CORE0_INTR_IN_62	62	R5FSS1_CORE0_INTR_TPCC0_INT_6
R5FSS1_CORE0_INTR_IN_63	63	R5FSS1_CORE0_INTR_TPCC0_INT_7
R5FSS1_CORE0_INTR_IN_64	64	R5FSS1_CORE0_INTR_TPCC0_ERRINT
R5FSS1_CORE0_INTR_IN_65	65	R5FSS1_CORE0_INTR_TPCC0_MPINT
R5FSS1_CORE0_INTR_IN_66	66	R5FSS1_CORE0_INTR_TPTC0_ERINT_0
R5FSS1_CORE0_INTR_IN_67	67	R5FSS1_CORE0_INTR_TPTC0_ERINT_1
R5FSS1_CORE0_INTR_IN_68	68	R5FSS1_CORE0_INTR_MCRC0_INT
R5FSS1_CORE0_INTR_IN_69	69	R5FSS1_CORE0_INTR_MPU_ADDR_ERRAGG
R5FSS1_CORE0_INTR_IN_70	70	R5FSS1_CORE0_INTR_MPU_PROT_ERRAGG
R5FSS1_CORE0_INTR_IN_71	71	R5FSS1_CORE0_INTR_PBIST_DONE
R5FSS1_CORE0_INTR_IN_72	72	R5FSS1_CORE0_INTR_TPCC0_INTAGGR
R5FSS1_CORE0_INTR_IN_73	73	R5FSS1_CORE0_INTR_TPCC0_ERRAGGR
R5FSS1_CORE0_INTR_IN_74	74	R5FSS1_CORE0_INTR_DCC0_DONE
R5FSS1_CORE0_INTR_IN_75	75	R5FSS1_CORE0_INTR_DCC1_DONE
R5FSS1_CORE0_INTR_IN_76	76	R5FSS1_CORE0_INTR_DCC2_DONE
R5FSS1_CORE0_INTR_IN_77	77	R5FSS1_CORE0_INTR_DCC3_DONE
R5FSS1_CORE0_INTR_IN_78	78	R5FSS1_CORE0_INTR_MCSPI0_INTR
R5FSS1_CORE0_INTR_IN_79	79	R5FSS1_CORE0_INTR_MCSPI1_INTR
R5FSS1_CORE0_INTR_IN_80	80	R5FSS1_CORE0_INTR_MCSPI2_INTR
R5FSS1_CORE0_INTR_IN_81	81	R5FSS1_CORE0_INTR_MCSPI3_INTR
R5FSS1_CORE0_INTR_IN_82	82	R5FSS1_CORE0_INTR_MCSPI4_INTR
R5FSS1_CORE0_INTR_IN_83	83	R5FSS1_CORE0_INTR_MMC0_INTR
R5FSS1_CORE0_INTR_IN_84	84	R5FSS1_CORE0_INTR_RTI0_INTR_0
R5FSS1_CORE0_INTR_IN_85	85	R5FSS1_CORE0_INTR_RTI0_INTR_1
R5FSS1_CORE0_INTR_IN_86	86	R5FSS1_CORE0_INTR_RTI0_INTR_2
R5FSS1_CORE0_INTR_IN_87	87	R5FSS1_CORE0_INTR_RTI0_INTR_3
R5FSS1_CORE0_INTR_IN_88	88	R5FSS1_CORE0_INTR_RESERVED

Table 10-19. R5FSS1_CORE0 Interrupt Map (continued)

Interrupt Input Line	Interrupt ID	Source Interrupt
R5FSS1_CORE0_INTR_IN_89	89	R5FSS1_CORE0_INTR_RTI0_OVERFLOW_INT0
R5FSS1_CORE0_INTR_IN_90	90	R5FSS1_CORE0_INTR_RTI0_OVERFLOW_INT1
R5FSS1_CORE0_INTR_IN_91	91	R5FSS1_CORE0_INTR_RTI1_INTR_0
R5FSS1_CORE0_INTR_IN_92	92	R5FSS1_CORE0_INTR_RTI1_INTR_1
R5FSS1_CORE0_INTR_IN_93	93	R5FSS1_CORE0_INTR_RTI1_INTR_2
R5FSS1_CORE0_INTR_IN_94	94	R5FSS1_CORE0_INTR_RTI1_INTR_3
R5FSS1_CORE0_INTR_IN_95	95	R5FSS1_CORE0_INTR_RESERVED
R5FSS1_CORE0_INTR_IN_96	96	R5FSS1_CORE0_INTR_RTI1_OVERFLOW_INT0
R5FSS1_CORE0_INTR_IN_97	97	R5FSS1_CORE0_INTR_RTI1_OVERFLOW_INT1
R5FSS1_CORE0_INTR_IN_98	98	R5FSS1_CORE0_INTR_RTI2_INTR_0
R5FSS1_CORE0_INTR_IN_99	99	R5FSS1_CORE0_INTR_RTI2_INTR_1
R5FSS1_CORE0_INTR_IN_100	100	R5FSS1_CORE0_INTR_RTI2_INTR_2
R5FSS1_CORE0_INTR_IN_101	101	R5FSS1_CORE0_INTR_RTI2_INTR_3
R5FSS1_CORE0_INTR_IN_102	102	R5FSS1_CORE0_INTR_RESERVED
R5FSS1_CORE0_INTR_IN_103	103	R5FSS1_CORE0_INTR_RTI2_OVERFLOW_INT0
R5FSS1_CORE0_INTR_IN_104	104	R5FSS1_CORE0_INTR_RTI2_OVERFLOW_INT1
R5FSS1_CORE0_INTR_IN_105	105	R5FSS1_CORE0_INTR_RTI3_INTR_0
R5FSS1_CORE0_INTR_IN_106	106	R5FSS1_CORE0_INTR_RTI3_INTR_1
R5FSS1_CORE0_INTR_IN_107	107	R5FSS1_CORE0_INTR_RTI3_INTR_2
R5FSS1_CORE0_INTR_IN_108	108	R5FSS1_CORE0_INTR_RTI3_INTR_3
R5FSS1_CORE0_INTR_IN_109	109	R5FSS1_CORE0_INTR_RESERVED
R5FSS1_CORE0_INTR_IN_110	110	R5FSS1_CORE0_INTR_RTI3_OVERFLOW_INT0
R5FSS1_CORE0_INTR_IN_111	111	R5FSS1_CORE0_INTR_RTI3_OVERFLOW_INT1
R5FSS1_CORE0_INTR_IN_112	112	R5FSS1_CORE0_INTR_RESERVED
R5FSS1_CORE0_INTR_IN_113	113	R5FSS1_CORE0_INTR_ESM0_ESM_INT_CFG
R5FSS1_CORE0_INTR_IN_114	114	R5FSS1_CORE0_INTR_ESM0_ESM_INT_HI
R5FSS1_CORE0_INTR_IN_115	115	R5FSS1_CORE0_INTR_ESM0_ESM_INT_LOW
R5FSS1_CORE0_INTR_IN_116	116	R5FSS1_CORE0_INTR_R5SS0_CPU0_PMU_INT
R5FSS1_CORE0_INTR_IN_117	117	R5FSS1_CORE0_INTR_R5SS0_CPU1_PMU_INT
R5FSS1_CORE0_INTR_IN_118	118	R5FSS1_CORE0_INTR_R5SS1_COMMRX_0
R5FSS1_CORE0_INTR_IN_119	119	R5FSS1_CORE0_INTR_R5SS1_COMMTX_0
R5FSS1_CORE0_INTR_IN_120	120	R5FSS1_CORE0_INTR_R5SS1_CPU0_CTI_INT
R5FSS1_CORE0_INTR_IN_121	121	R5FSS1_CORE0_INTR_R5SS1_CPU0_VALFIQ
R5FSS1_CORE0_INTR_IN_122	122	R5FSS1_CORE0_INTR_R5SS1_CPU0_VALIRQ
R5FSS1_CORE0_INTR_IN_123	123	R5FSS1_CORE0_INTR_R5SS1_CPU1_CTI_INT
R5FSS1_CORE0_INTR_IN_124	124	R5FSS1_CORE0_INTR_MMR_ACC_ERRAGG
R5FSS1_CORE0_INTR_IN_125	125	R5FSS1_CORE0_INTR_R5SS1_LIVELOCK_1
R5FSS1_CORE0_INTR_IN_126	126	R5FSS1_CORE0_INTR_R5SS0_LIVELOCK_0
R5FSS1_CORE0_INTR_IN_127	127	R5FSS1_CORE0_INTR_R5SS0_LIVELOCK_1
R5FSS1_CORE0_INTR_IN_128	128	R5FSS1_CORE0_INTR_RTI_WDT2_NMI
R5FSS1_CORE0_INTR_IN_129	129	R5FSS1_CORE0_INTR_SW_IRQ
R5FSS1_CORE0_INTR_IN_130	130	R5FSS1_CORE0_INTR_R5SS1_CORE0_FPU_EXP
R5FSS1_CORE0_INTR_IN_131	131	R5FSS1_CORE0_INTR_DEBUGSS_TXDATA_AVAIL
R5FSS1_CORE0_INTR_IN_132	132	R5FSS1_CORE0_INTR_DEBUGSS_R5SS0_STC_DONE
R5FSS1_CORE0_INTR_IN_133	133	R5FSS1_CORE0_INTR_TSENSE_H
R5FSS1_CORE0_INTR_IN_134	134	R5FSS1_CORE0_INTR_TSENSE_L
R5FSS1_CORE0_INTR_IN_135	135	R5FSS1_CORE0_INTR_AHB_WRITE_ERR

Table 10-19. R5FSS1_CORE0 Interrupt Map (continued)

Interrupt Input Line	Interrupt ID	Source Interrupt
R5FSS1_CORE0_INTR_IN_136	136	R5FSS1_CORE0_INTR_MBOX_READ_REQ
R5FSS1_CORE0_INTR_IN_137	137	R5FSS1_CORE0_INTR_MBOX_READ_ACK
R5FSS1_CORE0_INTR_IN_138	138	R5FSS1_CORE0_INTR_SOC_TIMESYNCXBAR1_OUT_26
R5FSS1_CORE0_INTR_IN_139	139	R5FSS1_CORE0_INTR_SOC_TIMESYNCXBAR1_OUT_27
R5FSS1_CORE0_INTR_IN_140	140	R5FSS1_CORE0_INTR_SOC_TIMESYNCXBAR1_OUT_28
R5FSS1_CORE0_INTR_IN_141	141	R5FSS1_CORE0_INTR_SOC_TIMESYNCXBAR1_OUT_29
R5FSS1_CORE0_INTR_IN_142	142	R5FSS1_CORE0_INTR_GPIO_INTRXBAR_OUT_22
R5FSS1_CORE0_INTR_IN_143	143	R5FSS1_CORE0_INTR_GPIO_INTRXBAR_OUT_23
R5FSS1_CORE0_INTR_IN_144	144	R5FSS1_CORE0_INTR_GPIO_INTRXBAR_OUT_24
R5FSS1_CORE0_INTR_IN_145	145	R5FSS1_CORE0_INTR_GPIO_INTRXBAR_OUT_25
R5FSS1_CORE0_INTR_IN_146	146	R5FSS1_CORE0_CONTROLSS_INTRXBAR0_OUT_0
R5FSS1_CORE0_INTR_IN_147	147	R5FSS1_CORE0_CONTROLSS_INTRXBAR0_OUT_1
R5FSS1_CORE0_INTR_IN_148	148	R5FSS1_CORE0_CONTROLSS_INTRXBAR0_OUT_2
R5FSS1_CORE0_INTR_IN_149	149	R5FSS1_CORE0_CONTROLSS_INTRXBAR0_OUT_3
R5FSS1_CORE0_INTR_IN_150	150	R5FSS1_CORE0_CONTROLSS_INTRXBAR0_OUT_4
R5FSS1_CORE0_INTR_IN_151	151	R5FSS1_CORE0_CONTROLSS_INTRXBAR0_OUT_5
R5FSS1_CORE0_INTR_IN_152	152	R5FSS1_CORE0_CONTROLSS_INTRXBAR0_OUT_6
R5FSS1_CORE0_INTR_IN_153	153	R5FSS1_CORE0_CONTROLSS_INTRXBAR0_OUT_7
R5FSS1_CORE0_INTR_IN_154	154	R5FSS1_CORE0_CONTROLSS_INTRXBAR0_OUT_8
R5FSS1_CORE0_INTR_IN_155	155	R5FSS1_CORE0_CONTROLSS_INTRXBAR0_OUT_9
R5FSS1_CORE0_INTR_IN_156	156	R5FSS1_CORE0_CONTROLSS_INTRXBAR0_OUT_10
R5FSS1_CORE0_INTR_IN_157	157	R5FSS1_CORE0_CONTROLSS_INTRXBAR0_OUT_11
R5FSS1_CORE0_INTR_IN_158	158	R5FSS1_CORE0_CONTROLSS_INTRXBAR0_OUT_12
R5FSS1_CORE0_INTR_IN_159	159	R5FSS1_CORE0_CONTROLSS_INTRXBAR0_OUT_13
R5FSS1_CORE0_INTR_IN_160	160	R5FSS1_CORE0_CONTROLSS_INTRXBAR0_OUT_14
R5FSS1_CORE0_INTR_IN_161	161	R5FSS1_CORE0_CONTROLSS_INTRXBAR0_OUT_15
R5FSS1_CORE0_INTR_IN_162	162	R5FSS1_CORE0_CONTROLSS_INTRXBAR0_OUT_16
R5FSS1_CORE0_INTR_IN_163	163	R5FSS1_CORE0_CONTROLSS_INTRXBAR0_OUT_17
R5FSS1_CORE0_INTR_IN_164	164	R5FSS1_CORE0_CONTROLSS_INTRXBAR0_OUT_18
R5FSS1_CORE0_INTR_IN_165	165	R5FSS1_CORE0_CONTROLSS_INTRXBAR0_OUT_19
R5FSS1_CORE0_INTR_IN_166	166	R5FSS1_CORE0_CONTROLSS_INTRXBAR0_OUT_20
R5FSS1_CORE0_INTR_IN_167	167	R5FSS1_CORE0_CONTROLSS_INTRXBAR0_OUT_21
R5FSS1_CORE0_INTR_IN_168	168	R5FSS1_CORE0_CONTROLSS_INTRXBAR0_OUT_22
R5FSS1_CORE0_INTR_IN_169	169	R5FSS1_CORE0_CONTROLSS_INTRXBAR0_OUT_23
R5FSS1_CORE0_INTR_IN_170	170	R5FSS1_CORE0_CONTROLSS_INTRXBAR0_OUT_24
R5FSS1_CORE0_INTR_IN_171	171	R5FSS1_CORE0_CONTROLSS_INTRXBAR0_OUT_25
R5FSS1_CORE0_INTR_IN_172	172	R5FSS1_CORE0_CONTROLSS_INTRXBAR0_OUT_26
R5FSS1_CORE0_INTR_IN_173	173	R5FSS1_CORE0_CONTROLSS_INTRXBAR0_OUT_27
R5FSS1_CORE0_INTR_IN_174	174	R5FSS1_CORE0_CONTROLSS_INTRXBAR0_OUT_28
R5FSS1_CORE0_INTR_IN_175	175	R5FSS1_CORE0_CONTROLSS_INTRXBAR0_OUT_29
R5FSS1_CORE0_INTR_IN_176	176	R5FSS1_CORE0_CONTROLSS_INTRXBAR0_OUT_30
R5FSS1_CORE0_INTR_IN_177	177	R5FSS1_CORE0_CONTROLSS_INTRXBAR0_OUT_31
R5FSS1_CORE0_INTR_IN_178	178	R5FSS1_CORE0_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_0
R5FSS1_CORE0_INTR_IN_179	179	R5FSS1_CORE0_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_1
R5FSS1_CORE0_INTR_IN_180	180	R5FSS1_CORE0_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_2
R5FSS1_CORE0_INTR_IN_181	181	R5FSS1_CORE0_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_3
R5FSS1_CORE0_INTR_IN_182	182	R5FSS1_CORE0_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_4

Table 10-19. R5FSS1_CORE0 Interrupt Map (continued)

Interrupt Input Line	Interrupt ID	Source Interrupt
R5FSS1_CORE0_INTR_IN_183	183	R5FSS1_CORE0_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_5
R5FSS1_CORE0_INTR_IN_184	184	R5FSS1_CORE0_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_6
R5FSS1_CORE0_INTR_IN_185	185	R5FSS1_CORE0_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_7
R5FSS1_CORE0_INTR_IN_186	186	R5FSS1_CORE0_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_8
R5FSS1_CORE0_INTR_IN_187	187	R5FSS1_CORE0_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_9
R5FSS1_CORE0_INTR_IN_188	188	R5FSS1_CORE0_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_10
R5FSS1_CORE0_INTR_IN_189	189	R5FSS1_CORE0_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_11
R5FSS1_CORE0_INTR_IN_190	190	R5FSS1_CORE0_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_12
R5FSS1_CORE0_INTR_IN_191	191	R5FSS1_CORE0_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_13
R5FSS1_CORE0_INTR_IN_192	192	R5FSS1_CORE0_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_14
R5FSS1_CORE0_INTR_IN_193	193	R5FSS1_CORE0_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_15
R5FSS1_CORE0_INTR_IN_194	194	R5FSS1_CORE0_CPSW0_CPTS_COMP
R5FSS1_CORE0_INTR_IN_195	195	R5FSS1_CORE0_INTR_RESERVED
R5FSS1_CORE0_INTR_IN_196	196	R5FSS1_CORE0_INTR_RESERVED
R5FSS1_CORE0_INTR_IN_197	197	R5FSS1_CORE0_INTR_MCAN4_EXT_TS_ROLLOVER_LVL_INT_0
R5FSS1_CORE0_INTR_IN_198	198	R5FSS1_CORE0_INTR_MCAN4_MCAN_LVL_INT_0
R5FSS1_CORE0_INTR_IN_199	199	R5FSS1_CORE0_INTR_MCAN4_MCAN_LVL_INT_1
R5FSS1_CORE0_INTR_IN_200	200	R5FSS1_CORE0_INTR_MCAN5_EXT_TS_ROLLOVER_LVL_INT_0
R5FSS1_CORE0_INTR_IN_201	201	R5FSS1_CORE0_INTR_MCAN5_MCAN_LVL_INT_0
R5FSS1_CORE0_INTR_IN_202	202	R5FSS1_CORE0_INTR_MCAN5_MCAN_LVL_INT_1
R5FSS1_CORE0_INTR_IN_203	203	R5FSS1_CORE0_INTR_MCAN6_EXT_TS_ROLLOVER_LVL_INT_0
R5FSS1_CORE0_INTR_IN_204	204	R5FSS1_CORE0_INTR_MCAN6_MCAN_LVL_INT_0
R5FSS1_CORE0_INTR_IN_205	205	R5FSS1_CORE0_INTR_MCAN6_MCAN_LVL_INT_1
R5FSS1_CORE0_INTR_IN_206	206	R5FSS1_CORE0_INTR_MCAN7_EXT_TS_ROLLOVER_LVL_INT_0
R5FSS1_CORE0_INTR_IN_207	207	R5FSS1_CORE0_INTR_MCAN7_MCAN_LVL_INT_0
R5FSS1_CORE0_INTR_IN_208	208	R5FSS1_CORE0_INTR_MCAN7_MCAN_LVL_INT_1
R5FSS1_CORE0_INTR_IN_209	209	R5FSS1_CORE0_INTR_R5SS0_CPU0_TMU_LVF
R5FSS1_CORE0_INTR_IN_210	210	R5FSS1_CORE0_INTR_R5SS0_CPU0_TMU_LUF
R5FSS1_CORE0_INTR_IN_211	211	R5FSS1_CORE0_INTR_HW_RESOLVER
R5FSS1_CORE0_INTR_IN_212	212	R5FSS1_CORE0_INTR_FSS_VBUSM_TIMEOUT
R5FSS1_CORE0_INTR_IN_213	213	R5FSS1_CORE0_INTR_OTFA_ERROR
R5FSS1_CORE0_INTR_IN_214	214	R5FSS1_CORE0_INTR_FOTA_STAT
R5FSS1_CORE0_INTR_IN_215	215	R5FSS1_CORE0_INTR_FOTA_STAT_ERR
R5FSS1_CORE0_INTR_IN_216	216	R5FSS1_CORE0_INTR_MCSPI5_INTR
R5FSS1_CORE0_INTR_IN_217	217	R5FSS1_CORE0_INTR_MCSPI6_INTR
R5FSS1_CORE0_INTR_IN_218	218	R5FSS1_CORE0_INTR_MCSPI7_INTR
R5FSS1_CORE0_INTR_IN_219	219	R5FSS1_CORE0_INTR_RTI4_INTR_0
R5FSS1_CORE0_INTR_IN_220	220	R5FSS1_CORE0_INTR_RTI4_INTR_1
R5FSS1_CORE0_INTR_IN_221	221	R5FSS1_CORE0_INTR_RTI4_INTR_2
R5FSS1_CORE0_INTR_IN_222	222	R5FSS1_CORE0_INTR_RTI4_INTR_3
R5FSS1_CORE0_INTR_IN_223	223	R5FSS1_CORE0_INTR_RTI4_OVERFLOW_INT0
R5FSS1_CORE0_INTR_IN_224	224	R5FSS1_CORE0_INTR_RTI4_OVERFLOW_INT1
R5FSS1_CORE0_INTR_IN_225	225	R5FSS1_CORE0_INTR_RTI5_INTR_0
R5FSS1_CORE0_INTR_IN_226	226	R5FSS1_CORE0_INTR_RTI5_INTR_1
R5FSS1_CORE0_INTR_IN_227	227	R5FSS1_CORE0_INTR_RTI5_INTR_2
R5FSS1_CORE0_INTR_IN_228	228	R5FSS1_CORE0_INTR_RTI5_INTR_3
R5FSS1_CORE0_INTR_IN_229	229	R5FSS1_CORE0_INTR_RTI5_OVERFLOW_INT0

Table 10-19. R5FSS1_CORE0 Interrupt Map (continued)

Interrupt Input Line	Interrupt ID	Source Interrupt
R5FSS1_CORE0_INTR_IN_230	230	R5FSS1_CORE0_INTR_RTI5_OVERFLOW_INT1
R5FSS1_CORE0_INTR_IN_231	231	R5FSS1_CORE0_INTR_RTI6_INTR_0
R5FSS1_CORE0_INTR_IN_232	232	R5FSS1_CORE0_INTR_RTI6_INTR_1
R5FSS1_CORE0_INTR_IN_233	233	R5FSS1_CORE0_INTR_RTI6_INTR_2
R5FSS1_CORE0_INTR_IN_234	234	R5FSS1_CORE0_INTR_RTI6_INTR_3
R5FSS1_CORE0_INTR_IN_235	235	R5FSS1_CORE0_INTR_RTI6_OVERFLOW_INT0
R5FSS1_CORE0_INTR_IN_236	236	R5FSS1_CORE0_INTR_RTI6_OVERFLOW_INT1
R5FSS1_CORE0_INTR_IN_237	237	R5FSS1_CORE0_INTR_RTI7_INTR_0
R5FSS1_CORE0_INTR_IN_238	238	R5FSS1_CORE0_INTR_RTI7_INTR_1
R5FSS1_CORE0_INTR_IN_239	239	R5FSS1_CORE0_INTR_RTI7_INTR_2
R5FSS1_CORE0_INTR_IN_240	240	R5FSS1_CORE0_INTR_RTI7_INTR_3
R5FSS1_CORE0_INTR_IN_241	241	R5FSS1_CORE0_INTR_RTI7_OVERFLOW_INT0
R5FSS1_CORE0_INTR_IN_242	242	R5FSS1_CORE0_INTR_RTI7_OVERFLOW_INT1
R5FSS1_CORE0_INTR_IN_243	243	R5FSS1_CORE0_INTR_R5SS0_CPU0_RL2_ERR
R5FSS1_CORE0_INTR_IN_244	244	R5FSS1_CORE0_INTR_R5SS0_CPU1_RL2_ERR
R5FSS1_CORE0_INTR_IN_245	245	R5FSS1_CORE0_INTR_R5SS1_CPU0_RL2_ERR
R5FSS1_CORE0_INTR_IN_246	246	R5FSS1_CORE0_INTR_R5SS1_CPU1_RL2_ERR

10.4.4 R5FSS1_CORE1 Interrupt Map

Table 10-20 shows the mapping of events to the R5FSS1_CORE1.

Both R5FSS1_CORE1 and R5FSS1_CORE0 use the R5FSS1_CORE0 interrupt map when operating in lockstep mode.

Table 10-20. R5FSS1_CORE1 Interrupt Map

Interrupt Input Line	Interrupt ID	Source Interrupt
R5FSS1_CORE1_INTR_IN_0	0	R5FSS1_CORE1_INTR_PRU_ICSSM0_PR1_HOST_INTR_PEND_0
R5FSS1_CORE1_INTR_IN_1	1	R5FSS1_CORE1_INTR_PRU_ICSSM0_PR1_HOST_INTR_PEND_1
R5FSS1_CORE1_INTR_IN_2	2	R5FSS1_CORE1_INTR_PRU_ICSSM0_PR1_HOST_INTR_PEND_2
R5FSS1_CORE1_INTR_IN_3	3	R5FSS1_CORE1_INTR_PRU_ICSSM0_PR1_HOST_INTR_PEND_3
R5FSS1_CORE1_INTR_IN_4	4	R5FSS1_CORE1_INTR_PRU_ICSSM0_PR1_HOST_INTR_PEND_4
R5FSS1_CORE1_INTR_IN_5	5	R5FSS1_CORE1_INTR_PRU_ICSSM0_PR1_HOST_INTR_PEND_5
R5FSS1_CORE1_INTR_IN_6	6	R5FSS1_CORE1_INTR_PRU_ICSSM0_PR1_HOST_INTR_PEND_6
R5FSS1_CORE1_INTR_IN_7	7	R5FSS1_CORE1_INTR_PRU_ICSSM0_PR1_HOST_INTR_PEND_7
R5FSS1_CORE1_INTR_IN_8	8	R5FSS1_CORE1_INTR_PRU_ICSSM0_PR1_RX_SOF_INTR_REQ_0
R5FSS1_CORE1_INTR_IN_9	9	R5FSS1_CORE1_INTR_PRU_ICSSM0_PR1_RX_SOF_INTR_REQ_1
R5FSS1_CORE1_INTR_IN_10	10	R5FSS1_CORE1_INTR_PRU_ICSSM0_PR1_TX_SOF_INTR_REQ_0
R5FSS1_CORE1_INTR_IN_11	11	R5FSS1_CORE1_INTR_PRU_ICSSM0_PR1_TX_SOF_INTR_REQ_1
R5FSS1_CORE1_INTR_IN_12	12	R5FSS1_CORE1_INTR_CPSW0_FH_INTR
R5FSS1_CORE1_INTR_IN_13	13	R5FSS1_CORE1_INTR_CPSW0_TH_INTR
R5FSS1_CORE1_INTR_IN_14	14	R5FSS1_CORE1_INTR_CPSW0_TH_THRESH_INTR
R5FSS1_CORE1_INTR_IN_15	15	R5FSS1_CORE1_INTR_CPSW0_MISC_INTR
R5FSS1_CORE1_INTR_IN_16	16	R5FSS1_CORE1_INTR_LIN0_INTR_0
R5FSS1_CORE1_INTR_IN_17	17	R5FSS1_CORE1_INTR_LIN0_INTR_1
R5FSS1_CORE1_INTR_IN_18	18	R5FSS1_CORE1_INTR_LIN1_INTR_0
R5FSS1_CORE1_INTR_IN_19	19	R5FSS1_CORE1_INTR_LIN1_INTR_1
R5FSS1_CORE1_INTR_IN_20	20	R5FSS1_CORE1_INTR_LIN2_INTR_0
R5FSS1_CORE1_INTR_IN_21	21	R5FSS1_CORE1_INTR_LIN2_INTR_1
R5FSS1_CORE1_INTR_IN_22	22	R5FSS1_CORE1_INTR_LIN3_INTR_0
R5FSS1_CORE1_INTR_IN_23	23	R5FSS1_CORE1_INTR_LIN3_INTR_1
R5FSS1_CORE1_INTR_IN_24	24	R5FSS1_CORE1_INTR_LIN4_INTR_0
R5FSS1_CORE1_INTR_IN_25	25	R5FSS1_CORE1_INTR_LIN4_INTR_1
R5FSS1_CORE1_INTR_IN_26	26	R5FSS1_CORE1_INTR_MCAN0_EXT_TS_ROLLOVER_LVL_INT_0
R5FSS1_CORE1_INTR_IN_27	27	R5FSS1_CORE1_INTR_MCAN0_MCAN_LVL_INT_0
R5FSS1_CORE1_INTR_IN_28	28	R5FSS1_CORE1_INTR_MCAN0_MCAN_LVL_INT_1
R5FSS1_CORE1_INTR_IN_29	29	R5FSS1_CORE1_INTR_MCAN1_EXT_TS_ROLLOVER_LVL_INT_0
R5FSS1_CORE1_INTR_IN_30	30	R5FSS1_CORE1_INTR_MCAN1_MCAN_LVL_INT_0
R5FSS1_CORE1_INTR_IN_31	31	R5FSS1_CORE1_INTR_MCAN1_MCAN_LVL_INT_1
R5FSS1_CORE1_INTR_IN_32	32	R5FSS1_CORE1_INTR_MCAN2_EXT_TS_ROLLOVER_LVL_INT_0
R5FSS1_CORE1_INTR_IN_33	33	R5FSS1_CORE1_INTR_MCAN2_MCAN_LVL_INT_0
R5FSS1_CORE1_INTR_IN_34	34	R5FSS1_CORE1_INTR_MCAN2_MCAN_LVL_INT_1
R5FSS1_CORE1_INTR_IN_35	35	R5FSS1_CORE1_INTR_MCAN3_EXT_TS_ROLLOVER_LVL_INT_0
R5FSS1_CORE1_INTR_IN_36	36	R5FSS1_CORE1_INTR_MCAN3_MCAN_LVL_INT_0
R5FSS1_CORE1_INTR_IN_37	37	R5FSS1_CORE1_INTR_MCAN3_MCAN_LVL_INT_1
R5FSS1_CORE1_INTR_IN_38	38	R5FSS1_CORE1_INTR_UART0_IRQ
R5FSS1_CORE1_INTR_IN_39	39	R5FSS1_CORE1_INTR_UART1_IRQ
R5FSS1_CORE1_INTR_IN_40	40	R5FSS1_CORE1_INTR_UART2_IRQ
R5FSS1_CORE1_INTR_IN_41	41	R5FSS1_CORE1_INTR_UART3_IRQ

Table 10-20. R5FSS1_CORE1 Interrupt Map (continued)

Interrupt Input Line	Interrupt ID	Source Interrupt
R5FSS1_CORE1_INTR_IN_42	42	R5FSS1_CORE1_INTR_UART4_IRQ
R5FSS1_CORE1_INTR_IN_43	43	R5FSS1_CORE1_INTR_UART5_IRQ
R5FSS1_CORE1_INTR_IN_44	44	R5FSS1_CORE1_INTR_I2C0_IRQ
R5FSS1_CORE1_INTR_IN_45	45	R5FSS1_CORE1_INTR_I2C1_IRQ
R5FSS1_CORE1_INTR_IN_46	46	R5FSS1_CORE1_INTR_I2C2_IRQ
R5FSS1_CORE1_INTR_IN_47	47	R5FSS1_CORE1_INTR_I2C3_IRQ
R5FSS1_CORE1_INTR_IN_48	48	R5FSS1_CORE1_INTR_DTHE_SHA_S_INT
R5FSS1_CORE1_INTR_IN_49	49	R5FSS1_CORE1_INTR_DTHE_SHA_P_INT
R5FSS1_CORE1_INTR_IN_50	50	R5FSS1_CORE1_INTR_DTHE_TRNG_INT
R5FSS1_CORE1_INTR_IN_51	51	R5FSS1_CORE1_INTR_DTHE_PKAE_INT
R5FSS1_CORE1_INTR_IN_52	52	R5FSS1_CORE1_INTR_DTHE_AES_S_INT
R5FSS1_CORE1_INTR_IN_53	53	R5FSS1_CORE1_INTR_DTHE_AES_P_INT
R5FSS1_CORE1_INTR_IN_54	54	R5FSS1_CORE1_INTR_OSPI0_INT
R5FSS1_CORE1_INTR_IN_55	55	R5FSS1_CORE1_INTR_TPCC0_INTG
R5FSS1_CORE1_INTR_IN_56	56	R5FSS1_CORE1_INTR_TPCC0_INT_0
R5FSS1_CORE1_INTR_IN_57	57	R5FSS1_CORE1_INTR_TPCC0_INT_1
R5FSS1_CORE1_INTR_IN_58	58	R5FSS1_CORE1_INTR_TPCC0_INT_2
R5FSS1_CORE1_INTR_IN_59	59	R5FSS1_CORE1_INTR_TPCC0_INT_3
R5FSS1_CORE1_INTR_IN_60	60	R5FSS1_CORE1_INTR_TPCC0_INT_4
R5FSS1_CORE1_INTR_IN_61	61	R5FSS1_CORE1_INTR_TPCC0_INT_5
R5FSS1_CORE1_INTR_IN_62	62	R5FSS1_CORE1_INTR_TPCC0_INT_6
R5FSS1_CORE1_INTR_IN_63	63	R5FSS1_CORE1_INTR_TPCC0_INT_7
R5FSS1_CORE1_INTR_IN_64	64	R5FSS1_CORE1_INTR_TPCC0_ERRINT
R5FSS1_CORE1_INTR_IN_65	65	R5FSS1_CORE1_INTR_TPCC0_MPINT
R5FSS1_CORE1_INTR_IN_66	66	R5FSS1_CORE1_INTR_TPTC0_ERINT_0
R5FSS1_CORE1_INTR_IN_67	67	R5FSS1_CORE1_INTR_TPTC0_ERINT_1
R5FSS1_CORE1_INTR_IN_68	68	R5FSS1_CORE1_INTR_MCRC0_INT
R5FSS1_CORE1_INTR_IN_69	69	R5FSS1_CORE1_INTR_MPU_ADDR_ERRAGG
R5FSS1_CORE1_INTR_IN_70	70	R5FSS1_CORE1_INTR_MPU_PROT_ERRAGG
R5FSS1_CORE1_INTR_IN_71	71	R5FSS1_CORE1_INTR_PBIST_DONE
R5FSS1_CORE1_INTR_IN_72	72	R5FSS1_CORE1_INTR_TPCC0_INTAGGR
R5FSS1_CORE1_INTR_IN_73	73	R5FSS1_CORE1_INTR_TPCC0_ERRAGGR
R5FSS1_CORE1_INTR_IN_74	74	R5FSS1_CORE1_INTR_DCC0_DONE
R5FSS1_CORE1_INTR_IN_75	75	R5FSS1_CORE1_INTR_DCC1_DONE
R5FSS1_CORE1_INTR_IN_76	76	R5FSS1_CORE1_INTR_DCC2_DONE
R5FSS1_CORE1_INTR_IN_77	77	R5FSS1_CORE1_INTR_DCC3_DONE
R5FSS1_CORE1_INTR_IN_78	78	R5FSS1_CORE1_INTR_MCSPI0_INTR
R5FSS1_CORE1_INTR_IN_79	79	R5FSS1_CORE1_INTR_MCSPI1_INTR
R5FSS1_CORE1_INTR_IN_80	80	R5FSS1_CORE1_INTR_MCSPI2_INTR
R5FSS1_CORE1_INTR_IN_81	81	R5FSS1_CORE1_INTR_MCSPI3_INTR
R5FSS1_CORE1_INTR_IN_82	82	R5FSS1_CORE1_INTR_MCSPI4_INTR
R5FSS1_CORE1_INTR_IN_83	83	R5FSS1_CORE1_INTR_MMC0_INTR
R5FSS1_CORE1_INTR_IN_84	84	R5FSS1_CORE1_INTR_RTI0_INTR_0
R5FSS1_CORE1_INTR_IN_85	85	R5FSS1_CORE1_INTR_RTI0_INTR_1
R5FSS1_CORE1_INTR_IN_86	86	R5FSS1_CORE1_INTR_RTI0_INTR_2
R5FSS1_CORE1_INTR_IN_87	87	R5FSS1_CORE1_INTR_RTI0_INTR_3
R5FSS1_CORE1_INTR_IN_88	88	R5FSS1_CORE1_INTR_RESERVED

Table 10-20. R5FSS1_CORE1 Interrupt Map (continued)

Interrupt Input Line	Interrupt ID	Source Interrupt
R5FSS1_CORE1_INTR_IN_89	89	R5FSS1_CORE1_INTR_RTI0_OVERFLOW_INT0
R5FSS1_CORE1_INTR_IN_90	90	R5FSS1_CORE1_INTR_RTI0_OVERFLOW_INT1
R5FSS1_CORE1_INTR_IN_91	91	R5FSS1_CORE1_INTR_RTI1_INTR_0
R5FSS1_CORE1_INTR_IN_92	92	R5FSS1_CORE1_INTR_RTI1_INTR_1
R5FSS1_CORE1_INTR_IN_93	93	R5FSS1_CORE1_INTR_RTI1_INTR_2
R5FSS1_CORE1_INTR_IN_94	94	R5FSS1_CORE1_INTR_RTI1_INTR_3
R5FSS1_CORE1_INTR_IN_95	95	R5FSS1_CORE1_INTR_RESERVED
R5FSS1_CORE1_INTR_IN_96	96	R5FSS1_CORE1_INTR_RTI1_OVERFLOW_INT0
R5FSS1_CORE1_INTR_IN_97	97	R5FSS1_CORE1_INTR_RTI1_OVERFLOW_INT1
R5FSS1_CORE1_INTR_IN_98	98	R5FSS1_CORE1_INTR_RTI2_INTR_0
R5FSS1_CORE1_INTR_IN_99	99	R5FSS1_CORE1_INTR_RTI2_INTR_1
R5FSS1_CORE1_INTR_IN_100	100	R5FSS1_CORE1_INTR_RTI2_INTR_2
R5FSS1_CORE1_INTR_IN_101	101	R5FSS1_CORE1_INTR_RTI2_INTR_3
R5FSS1_CORE1_INTR_IN_102	102	R5FSS1_CORE1_INTR_RESERVED
R5FSS1_CORE1_INTR_IN_103	103	R5FSS1_CORE1_INTR_RTI2_OVERFLOW_INT0
R5FSS1_CORE1_INTR_IN_104	104	R5FSS1_CORE1_INTR_RTI2_OVERFLOW_INT1
R5FSS1_CORE1_INTR_IN_105	105	R5FSS1_CORE1_INTR_RTI3_INTR_0
R5FSS1_CORE1_INTR_IN_106	106	R5FSS1_CORE1_INTR_RTI3_INTR_1
R5FSS1_CORE1_INTR_IN_107	107	R5FSS1_CORE1_INTR_RTI3_INTR_2
R5FSS1_CORE1_INTR_IN_108	108	R5FSS1_CORE1_INTR_RTI3_INTR_3
R5FSS1_CORE1_INTR_IN_109	109	R5FSS1_CORE1_INTR_RESERVED
R5FSS1_CORE1_INTR_IN_110	110	R5FSS1_CORE1_INTR_RTI3_OVERFLOW_INT0
R5FSS1_CORE1_INTR_IN_111	111	R5FSS1_CORE1_INTR_RTI3_OVERFLOW_INT1
R5FSS1_CORE1_INTR_IN_112	112	R5FSS1_CORE1_INTR_RESERVED
R5FSS1_CORE1_INTR_IN_113	113	R5FSS1_CORE1_INTR_ESM0_ESM_INT_CFG
R5FSS1_CORE1_INTR_IN_114	114	R5FSS1_CORE1_INTR_ESM0_ESM_INT_HI
R5FSS1_CORE1_INTR_IN_115	115	R5FSS1_CORE1_INTR_ESM0_ESM_INT_LOW
R5FSS1_CORE1_INTR_IN_116	116	R5FSS1_CORE1_INTR_R5SS0_CPU0_PMU_INT
R5FSS1_CORE1_INTR_IN_117	117	R5FSS1_CORE1_INTR_R5SS0_CPU1_PMU_INT
R5FSS1_CORE1_INTR_IN_118	118	R5FSS1_CORE1_INTR_R5SS1_COMMRX_1
R5FSS1_CORE1_INTR_IN_119	119	R5FSS1_CORE1_INTR_R5SS1_COMMTX_1
R5FSS1_CORE1_INTR_IN_120	120	R5FSS1_CORE1_INTR_R5SS1_CPU0_CTI_INT
R5FSS1_CORE1_INTR_IN_121	121	R5FSS1_CORE1_INTR_R5SS1_CPU1_CTI_INT
R5FSS1_CORE1_INTR_IN_122	122	R5FSS1_CORE1_INTR_R5SS1_CPU1_VALFIQ
R5FSS1_CORE1_INTR_IN_123	123	R5FSS1_CORE1_INTR_R5SS1_CPU1_VALIRQ
R5FSS1_CORE1_INTR_IN_124	124	R5FSS1_CORE1_INTR_MMR_ACC_ERRAGG
R5FSS1_CORE1_INTR_IN_125	125	R5FSS1_CORE1_INTR_R5SS1_LIVELOCK_0
R5FSS1_CORE1_INTR_IN_126	126	R5FSS1_CORE1_INTR_R5SS0_LIVELOCK_0
R5FSS1_CORE1_INTR_IN_127	127	R5FSS1_CORE1_INTR_R5SS0_LIVELOCK_1
R5FSS1_CORE1_INTR_IN_128	128	R5FSS1_CORE1_INTR_RTI_WDT3_NMI
R5FSS1_CORE1_INTR_IN_129	129	R5FSS1_CORE1_INTR_SW_IRQ
R5FSS1_CORE1_INTR_IN_130	130	R5FSS1_CORE1_INTR_R5SS1_CORE1_FPU_EXP
R5FSS1_CORE1_INTR_IN_131	131	R5FSS1_CORE1_INTR_DEBUGSS_TXDATA_AVAIL
R5FSS1_CORE1_INTR_IN_132	132	R5FSS1_CORE1_INTR_DEBUGSS_R5SS0_STC_DONE
R5FSS1_CORE1_INTR_IN_133	133	R5FSS1_CORE1_INTR_TSENSE_H
R5FSS1_CORE1_INTR_IN_134	134	R5FSS1_CORE1_INTR_TSENSE_L
R5FSS1_CORE1_INTR_IN_135	135	R5FSS1_CORE1_INTR_AHB_WRITE_ERR

Table 10-20. R5FSS1_CORE1 Interrupt Map (continued)

Interrupt Input Line	Interrupt ID	Source Interrupt
R5FSS1_CORE1_INTR_IN_136	136	R5FSS1_CORE1_INTR_MBOX_READ_REQ
R5FSS1_CORE1_INTR_IN_137	137	R5FSS1_CORE1_INTR_MBOX_READ_ACK
R5FSS1_CORE1_INTR_IN_138	138	R5FSS1_CORE1_INTR_SOC_TIMESYNCXBAR1_OUT_30
R5FSS1_CORE1_INTR_IN_139	139	R5FSS1_CORE1_INTR_SOC_TIMESYNCXBAR1_OUT_31
R5FSS1_CORE1_INTR_IN_140	140	R5FSS1_CORE1_INTR_SOC_TIMESYNCXBAR1_OUT_32
R5FSS1_CORE1_INTR_IN_141	141	R5FSS1_CORE1_INTR_SOC_TIMESYNCXBAR1_OUT_33
R5FSS1_CORE1_INTR_IN_142	142	R5FSS1_CORE1_INTR_GPIO_INTRXBAR_OUT_26
R5FSS1_CORE1_INTR_IN_143	143	R5FSS1_CORE1_INTR_GPIO_INTRXBAR_OUT_27
R5FSS1_CORE1_INTR_IN_144	144	R5FSS1_CORE1_INTR_GPIO_INTRXBAR_OUT_28
R5FSS1_CORE1_INTR_IN_145	145	R5FSS1_CORE1_INTR_GPIO_INTRXBAR_OUT_29
R5FSS1_CORE1_INTR_IN_146	146	R5FSS1_CORE1_CONTROLSS_INTRXBAR0_OUT_0
R5FSS1_CORE1_INTR_IN_147	147	R5FSS1_CORE1_CONTROLSS_INTRXBAR0_OUT_1
R5FSS1_CORE1_INTR_IN_148	148	R5FSS1_CORE1_CONTROLSS_INTRXBAR0_OUT_2
R5FSS1_CORE1_INTR_IN_149	149	R5FSS1_CORE1_CONTROLSS_INTRXBAR0_OUT_3
R5FSS1_CORE1_INTR_IN_150	150	R5FSS1_CORE1_CONTROLSS_INTRXBAR0_OUT_4
R5FSS1_CORE1_INTR_IN_151	151	R5FSS1_CORE1_CONTROLSS_INTRXBAR0_OUT_5
R5FSS1_CORE1_INTR_IN_152	152	R5FSS1_CORE1_CONTROLSS_INTRXBAR0_OUT_6
R5FSS1_CORE1_INTR_IN_153	153	R5FSS1_CORE1_CONTROLSS_INTRXBAR0_OUT_7
R5FSS1_CORE1_INTR_IN_154	154	R5FSS1_CORE1_CONTROLSS_INTRXBAR0_OUT_8
R5FSS1_CORE1_INTR_IN_155	155	R5FSS1_CORE1_CONTROLSS_INTRXBAR0_OUT_9
R5FSS1_CORE1_INTR_IN_156	156	R5FSS1_CORE1_CONTROLSS_INTRXBAR0_OUT_10
R5FSS1_CORE1_INTR_IN_157	157	R5FSS1_CORE1_CONTROLSS_INTRXBAR0_OUT_11
R5FSS1_CORE1_INTR_IN_158	158	R5FSS1_CORE1_CONTROLSS_INTRXBAR0_OUT_12
R5FSS1_CORE1_INTR_IN_159	159	R5FSS1_CORE1_CONTROLSS_INTRXBAR0_OUT_13
R5FSS1_CORE1_INTR_IN_160	160	R5FSS1_CORE1_CONTROLSS_INTRXBAR0_OUT_14
R5FSS1_CORE1_INTR_IN_161	161	R5FSS1_CORE1_CONTROLSS_INTRXBAR0_OUT_15
R5FSS1_CORE1_INTR_IN_162	162	R5FSS1_CORE1_CONTROLSS_INTRXBAR0_OUT_16
R5FSS1_CORE1_INTR_IN_163	163	R5FSS1_CORE1_CONTROLSS_INTRXBAR0_OUT_17
R5FSS1_CORE1_INTR_IN_164	164	R5FSS1_CORE1_CONTROLSS_INTRXBAR0_OUT_18
R5FSS1_CORE1_INTR_IN_165	165	R5FSS1_CORE1_CONTROLSS_INTRXBAR0_OUT_19
R5FSS1_CORE1_INTR_IN_166	166	R5FSS1_CORE1_CONTROLSS_INTRXBAR0_OUT_20
R5FSS1_CORE1_INTR_IN_167	167	R5FSS1_CORE1_CONTROLSS_INTRXBAR0_OUT_21
R5FSS1_CORE1_INTR_IN_168	168	R5FSS1_CORE1_CONTROLSS_INTRXBAR0_OUT_22
R5FSS1_CORE1_INTR_IN_169	169	R5FSS1_CORE1_CONTROLSS_INTRXBAR0_OUT_23
R5FSS1_CORE1_INTR_IN_170	170	R5FSS1_CORE1_CONTROLSS_INTRXBAR0_OUT_24
R5FSS1_CORE1_INTR_IN_171	171	R5FSS1_CORE1_CONTROLSS_INTRXBAR0_OUT_25
R5FSS1_CORE1_INTR_IN_172	172	R5FSS1_CORE1_CONTROLSS_INTRXBAR0_OUT_26
R5FSS1_CORE1_INTR_IN_173	173	R5FSS1_CORE1_CONTROLSS_INTRXBAR0_OUT_27
R5FSS1_CORE1_INTR_IN_174	174	R5FSS1_CORE1_CONTROLSS_INTRXBAR0_OUT_28
R5FSS1_CORE1_INTR_IN_175	175	R5FSS1_CORE1_CONTROLSS_INTRXBAR0_OUT_29
R5FSS1_CORE1_INTR_IN_176	176	R5FSS1_CORE1_CONTROLSS_INTRXBAR0_OUT_30
R5FSS1_CORE1_INTR_IN_177	177	R5FSS1_CORE1_CONTROLSS_INTRXBAR0_OUT_31
R5FSS1_CORE1_INTR_IN_178	178	R5FSS1_CORE1_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_0
R5FSS1_CORE1_INTR_IN_179	179	R5FSS1_CORE1_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_1
R5FSS1_CORE1_INTR_IN_180	180	R5FSS1_CORE1_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_2
R5FSS1_CORE1_INTR_IN_181	181	R5FSS1_CORE1_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_3
R5FSS1_CORE1_INTR_IN_182	182	R5FSS1_CORE1_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_4

Table 10-20. R5FSS1_CORE1 Interrupt Map (continued)

Interrupt Input Line	Interrupt ID	Source Interrupt
R5FSS1_CORE1_INTR_IN_183	183	R5FSS1_CORE1_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_5
R5FSS1_CORE1_INTR_IN_184	184	R5FSS1_CORE1_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_6
R5FSS1_CORE1_INTR_IN_185	185	R5FSS1_CORE1_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_7
R5FSS1_CORE1_INTR_IN_186	186	R5FSS1_CORE1_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_8
R5FSS1_CORE1_INTR_IN_187	187	R5FSS1_CORE1_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_9
R5FSS1_CORE1_INTR_IN_188	188	R5FSS1_CORE1_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_10
R5FSS1_CORE1_INTR_IN_189	189	R5FSS1_CORE1_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_11
R5FSS1_CORE1_INTR_IN_190	190	R5FSS1_CORE1_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_12
R5FSS1_CORE1_INTR_IN_191	191	R5FSS1_CORE1_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_13
R5FSS1_CORE1_INTR_IN_192	192	R5FSS1_CORE1_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_14
R5FSS1_CORE1_INTR_IN_193	193	R5FSS1_CORE1_INTR_PRU_ICSSM0_PR1_IEP0_CMP_INTR_REQ_15
R5FSS1_CORE1_INTR_IN_194	194	R5FSS1_CORE1_CPSW0_CPTS_COMP
R5FSS1_CORE1_INTR_IN_195	195	R5FSS1_CORE1_INTR_RESERVED
R5FSS1_CORE1_INTR_IN_196	196	R5FSS1_CORE1_INTR_RESERVED
R5FSS1_CORE1_INTR_IN_197	197	R5FSS1_CORE1_INTR_MCAN4_EXT_TS_ROLLOVER_LVL_INT_0
R5FSS1_CORE1_INTR_IN_198	198	R5FSS1_CORE1_INTR_MCAN4_MCAN_LVL_INT_0
R5FSS1_CORE1_INTR_IN_199	199	R5FSS1_CORE1_INTR_MCAN4_MCAN_LVL_INT_1
R5FSS1_CORE1_INTR_IN_200	200	R5FSS1_CORE1_INTR_MCAN5_EXT_TS_ROLLOVER_LVL_INT_0
R5FSS1_CORE1_INTR_IN_201	201	R5FSS1_CORE1_INTR_MCAN5_MCAN_LVL_INT_0
R5FSS1_CORE1_INTR_IN_202	202	R5FSS1_CORE1_INTR_MCAN5_MCAN_LVL_INT_1
R5FSS1_CORE1_INTR_IN_203	203	R5FSS1_CORE1_INTR_MCAN6_EXT_TS_ROLLOVER_LVL_INT_0
R5FSS1_CORE1_INTR_IN_204	204	R5FSS1_CORE1_INTR_MCAN6_MCAN_LVL_INT_0
R5FSS1_CORE1_INTR_IN_205	205	R5FSS1_CORE1_INTR_MCAN6_MCAN_LVL_INT_1
R5FSS1_CORE1_INTR_IN_206	206	R5FSS1_CORE1_INTR_MCAN7_EXT_TS_ROLLOVER_LVL_INT_0
R5FSS1_CORE1_INTR_IN_207	207	R5FSS1_CORE1_INTR_MCAN7_MCAN_LVL_INT_0
R5FSS1_CORE1_INTR_IN_208	208	R5FSS1_CORE1_INTR_MCAN7_MCAN_LVL_INT_1
R5FSS1_CORE1_INTR_IN_209	209	R5FSS1_CORE1_INTR_R5SS0_CPU0_TMU_LVF
R5FSS1_CORE1_INTR_IN_210	210	R5FSS1_CORE1_INTR_R5SS0_CPU0_TMU_LUF
R5FSS1_CORE1_INTR_IN_211	211	R5FSS1_CORE1_INTR_HW_RESOLVER
R5FSS1_CORE1_INTR_IN_212	212	R5FSS1_CORE1_INTR_FSS_VBUSM_TIMEOUT
R5FSS1_CORE1_INTR_IN_213	213	R5FSS1_CORE1_INTR_OTFA_ERROR
R5FSS1_CORE1_INTR_IN_214	214	R5FSS1_CORE1_INTR_FOTA_STAT
R5FSS1_CORE1_INTR_IN_215	215	R5FSS1_CORE1_INTR_FOTA_STAT_ERR
R5FSS1_CORE1_INTR_IN_216	216	R5FSS1_CORE1_INTR_MCSPI5_INTR
R5FSS1_CORE1_INTR_IN_217	217	R5FSS1_CORE1_INTR_MCSPI6_INTR
R5FSS1_CORE1_INTR_IN_218	218	R5FSS1_CORE1_INTR_MCSPI7_INTR
R5FSS1_CORE1_INTR_IN_219	219	R5FSS1_CORE1_INTR_RTI4_INTR_0
R5FSS1_CORE1_INTR_IN_220	220	R5FSS1_CORE1_INTR_RTI4_INTR_1
R5FSS1_CORE1_INTR_IN_221	221	R5FSS1_CORE1_INTR_RTI4_INTR_2
R5FSS1_CORE1_INTR_IN_222	222	R5FSS1_CORE1_INTR_RTI4_INTR_3
R5FSS1_CORE1_INTR_IN_223	223	R5FSS1_CORE1_INTR_RTI4_OVERFLOW_INT0
R5FSS1_CORE1_INTR_IN_224	224	R5FSS1_CORE1_INTR_RTI4_OVERFLOW_INT1
R5FSS1_CORE1_INTR_IN_225	225	R5FSS1_CORE1_INTR_RTI5_INTR_0
R5FSS1_CORE1_INTR_IN_226	226	R5FSS1_CORE1_INTR_RTI5_INTR_1
R5FSS1_CORE1_INTR_IN_227	227	R5FSS1_CORE1_INTR_RTI5_INTR_2
R5FSS1_CORE1_INTR_IN_228	228	R5FSS1_CORE1_INTR_RTI5_INTR_3
R5FSS1_CORE1_INTR_IN_229	229	R5FSS1_CORE1_INTR_RTI5_OVERFLOW_INT0

Table 10-20. R5FSS1_CORE1 Interrupt Map (continued)

Interrupt Input Line	Interrupt ID	Source Interrupt
R5FSS1_CORE1_INTR_IN_230	230	R5FSS1_CORE1_INTR_RTI5_OVERFLOW_INT1
R5FSS1_CORE1_INTR_IN_231	231	R5FSS1_CORE1_INTR_RTI6_INTR_0
R5FSS1_CORE1_INTR_IN_232	232	R5FSS1_CORE1_INTR_RTI6_INTR_1
R5FSS1_CORE1_INTR_IN_233	233	R5FSS1_CORE1_INTR_RTI6_INTR_2
R5FSS1_CORE1_INTR_IN_234	234	R5FSS1_CORE1_INTR_RTI6_INTR_3
R5FSS1_CORE1_INTR_IN_235	235	R5FSS1_CORE1_INTR_RTI6_OVERFLOW_INT0
R5FSS1_CORE1_INTR_IN_236	236	R5FSS1_CORE1_INTR_RTI6_OVERFLOW_INT1
R5FSS1_CORE1_INTR_IN_237	237	R5FSS1_CORE1_INTR_RTI7_INTR_0
R5FSS1_CORE1_INTR_IN_238	238	R5FSS1_CORE1_INTR_RTI7_INTR_1
R5FSS1_CORE1_INTR_IN_239	239	R5FSS1_CORE1_INTR_RTI7_INTR_2
R5FSS1_CORE1_INTR_IN_240	240	R5FSS1_CORE1_INTR_RTI7_INTR_3
R5FSS1_CORE1_INTR_IN_241	241	R5FSS1_CORE1_INTR_RTI7_OVERFLOW_INT0
R5FSS1_CORE1_INTR_IN_242	242	R5FSS1_CORE1_INTR_RTI7_OVERFLOW_INT1
R5FSS1_CORE1_INTR_IN_243	243	R5FSS1_CORE1_INTR_R5SS0_CPU0_RL2_ERR
R5FSS1_CORE1_INTR_IN_244	244	R5FSS1_CORE1_INTR_R5SS0_CPU1_RL2_ERR
R5FSS1_CORE1_INTR_IN_245	245	R5FSS1_CORE1_INTR_R5SS1_CPU0_RL2_ERR
R5FSS1_CORE1_INTR_IN_246	246	R5FSS1_CORE1_INTR_R5SS1_CPU1_RL2_ERR

10.4.5 PRU-ICSS Interrupt Map

Table 10-21 shows the mapping of external events to the PRU-ICSS.

Table 10-21. PRU-ICSS external events mapping

Interrupt Input Line	Interrupt Signal name	Interrupt Source	Interrupt Signal
PRU_ICSSM0_INTR_IN_0	PR1_SLV_INTR_0	PRU-ICSS XBAR	PRU-ICSS_XBAROUT_0
PRU_ICSSM0_INTR_IN_1	PR1_SLV_INTR_1	PRU-ICSS XBAR	PRU-ICSS_XBAROUT_1
PRU_ICSSM0_INTR_IN_2	PR1_SLV_INTR_2	PRU-ICSS XBAR	PRU-ICSS_XBAROUT_2
PRU_ICSSM0_INTR_IN_3	PR1_SLV_INTR_3	PRU-ICSS XBAR	PRU-ICSS_XBAROUT_3
PRU_ICSSM0_INTR_IN_4	PR1_SLV_INTR_4	PRU-ICSS XBAR	PRU-ICSS_XBAROUT_4
PRU_ICSSM0_INTR_IN_5	PR1_SLV_INTR_5	PRU-ICSS XBAR	PRU-ICSS_XBAROUT_5
PRU_ICSSM0_INTR_IN_6	PR1_SLV_INTR_6	PRU-ICSS XBAR	PRU-ICSS_XBAROUT_6
PRU_ICSSM0_INTR_IN_7	PR1_SLV_INTR_7	PRU-ICSS XBAR	PRU-ICSS_XBAROUT_7
PRU_ICSSM0_INTR_IN_8	PR1_SLV_INTR_8	PRU-ICSS XBAR	PRU-ICSS_XBAROUT_8
PRU_ICSSM0_INTR_IN_9	PR1_SLV_INTR_9	PRU-ICSS XBAR	PRU-ICSS_XBAROUT_9
PRU_ICSSM0_INTR_IN_10	PR1_SLV_INTR_10	PRU-ICSS XBAR	PRU-ICSS_XBAROUT_10
PRU_ICSSM0_INTR_IN_11	PR1_SLV_INTR_11	PRU-ICSS XBAR	PRU-ICSS_XBAROUT_11
PRU_ICSSM0_INTR_IN_12	PR1_SLV_INTR_12	PRU-ICSS XBAR	PRU-ICSS_XBAROUT_12
PRU_ICSSM0_INTR_IN_13	PR1_SLV_INTR_13	PRU-ICSS XBAR	PRU-ICSS_XBAROUT_13
PRU_ICSSM0_INTR_IN_14	PR1_SLV_INTR_14	PRU-ICSS XBAR	PRU-ICSS_XBAROUT_14
PRU_ICSSM0_INTR_IN_15	PR1_SLV_INTR_15	PRU-ICSS XBAR	PRU-ICSS_XBAROUT_15
PRU_ICSSM0_INTR_IN_16	PR1_SLV_INTR_16	CONTROLSS_INTXBAR	OUTPUTXBAR.Out0
PRU_ICSSM0_INTR_IN_17	PR1_SLV_INTR_17	CONTROLSS_INTXBAR	OUTPUTXBAR.Out1
PRU_ICSSM0_INTR_IN_18	PR1_SLV_INTR_18	CONTROLSS_INTXBAR	OUTPUTXBAR.Out2
PRU_ICSSM0_INTR_IN_19	PR1_SLV_INTR_19	CONTROLSS_INTXBAR	OUTPUTXBAR.Out3

Table 10-21. PRU-ICSS external events mapping (continued)

Interrupt Input Line	Interrupt Signal name	Interrupt Source	Interrupt Signal
PRU_ICSSM0_INTR_IN_20	PR1_SLV_INTR_20	CONTROLSS_INTXBAR	OUTPUTXBAR.Out4
PRU_ICSSM0_INTR_IN_21	PR1_SLV_INTR_21	CONTROLSS_INTXBAR	OUTPUTXBAR.Out5
PRU_ICSSM0_INTR_IN_22	PR1_SLV_INTR_22	CONTROLSS_INTXBAR	OUTPUTXBAR.Out6
PRU_ICSSM0_INTR_IN_23	PR1_SLV_INTR_23	CONTROLSS_INTXBAR	OUTPUTXBAR.Out7
PRU_ICSSM0_INTR_IN_24	PR1_SLV_INTR_24	CONTROLSS_INTXBAR	OUTPUTXBAR.Out8
PRU_ICSSM0_INTR_IN_25	PR1_SLV_INTR_25	CONTROLSS_INTXBAR	OUTPUTXBAR.Out9
PRU_ICSSM0_INTR_IN_26	PR1_SLV_INTR_26	CONTROLSS_INTXBAR	OUTPUTXBAR.Out10
PRU_ICSSM0_INTR_IN_27	PR1_SLV_INTR_27	CONTROLSS_INTXBAR	OUTPUTXBAR.Out11
PRU_ICSSM0_INTR_IN_28	PR1_SLV_INTR_28	CONTROLSS_INTXBAR	OUTPUTXBAR.Out12
PRU_ICSSM0_INTR_IN_29	PR1_SLV_INTR_29	CONTROLSS_INTXBAR	OUTPUTXBAR.Out13
PRU_ICSSM0_INTR_IN_30	PR1_SLV_INTR_30	CONTROLSS_INTXBAR	OUTPUTXBAR.Out14
PRU_ICSSM0_INTR_IN_31	PR1_SLV_INTR_31	CONTROLSS_INTXBAR	OUTPUTXBAR.Out15
PRU_ICSSM0_INTR_IN_32	ICSSM0_EDC_LATCH0_IN	SOC_TSXBAR_INTR	SOC_TIMESYNC_XBAR1_SYNCEVENT_OUT10
PRU_ICSSM0_INTR_IN_33	ICSSM0_EDC_LATCH1_IN	SOC_TSXBAR_INTR	SOC_TIMESYNC_XBAR1_SYNCEVENT_OUT11
PRU_ICSSM0_INTR_IN_34	ICSSM0_IEP_CAP_INTR0	SOC_TSXBAR_INTR	SOC_TIMESYNC_XBAR1_SYNCEVENT_OUT12
PRU_ICSSM0_INTR_IN_35	ICSSM0_IEP_CAP_INTR1	SOC_TSXBAR_INTR	SOC_TIMESYNC_XBAR1_SYNCEVENT_OUT13
PRU_ICSSM0_INTR_IN_36	ICSSM0_IEP_CAP_INTR2	SOC_TSXBAR_INTR	SOC_TIMESYNC_XBAR1_SYNCEVENT_OUT14
PRU_ICSSM0_INTR_IN_37	ICSSM0_IEP_CAP_INTR3	SOC_TSXBAR_INTR	SOC_TIMESYNC_XBAR1_SYNCEVENT_OUT15
PRU_ICSSM0_INTR_IN_38	ICSSM0_IEP_CAP_INTR4	SOC_TSXBAR_INTR	SOC_TIMESYNC_XBAR1_SYNCEVENT_OUT16
PRU_ICSSM0_INTR_IN_39	ICSSM0_IEP_CAP_INTR5	SOC_TSXBAR_INTR	SOC_TIMESYNC_XBAR1_SYNCEVENT_OUT17

Note

See tables **PRU-ICSS IP Internal Interrupts** and **AM263Px-Specific PRU-ICSS Interrupt Mapping** in the Programmable Real-Time Unit Subsystem (PRU-ICSS) chapter of the TRM for the mapping of external/internal events to the PRU-ICSS INTC interrupt lines 0 through 63.

10.4.6 ESM0 Interrupt Map

Table 10-22 shows the mapping of events to the ESM0.

Table 10-22. ESM0 Interrupt Map (Level)

Interrupt Input Line	Interrupt ID	Interrupt Source	Interrupt Signal
ESM_LVL_EVENT_0	0	EFUSE	efc_error
ESM_LVL_EVENT_1	1	EFUSE	efs_autoload_error
ESM_LVL_EVENT_2	2	MCAN0	MCAN0_ecc_corr_lvl_int
ESM_LVL_EVENT_3	3	MCAN0	MCAN0_ecc_uncorr_lvl_int
ESM_LVL_EVENT_4	4	MCAN1	MCAN1_ecc_corr_lvl_int
ESM_LVL_EVENT_5	5	MCAN1	MCAN1_ecc_uncorr_lvl_int
ESM_LVL_EVENT_6	6	MCAN2	MCAN2_ecc_corr_lvl_int
ESM_LVL_EVENT_7	7	MCAN2	MCAN2_ecc_uncorr_lvl_int
ESM_LVL_EVENT_8	8	MCAN3	MCAN3_ecc_corr_lvl_int
ESM_LVL_EVENT_9	9	MCAN3	MCAN3_ecc_uncorr_lvl_int
ESM_LVL_EVENT_10	10	R5FSS0_CORE0	R5FSS0_livelock_0
ESM_LVL_EVENT_11	11	R5FSS0_CORE1	R5FSS0_livelock_1
ESM_LVL_EVENT_12	12	R5FSS1_CORE0	R5FSS1_livelock_0
ESM_LVL_EVENT_13	13	R5FSS1_CORE1	R5FSS1_livelock_1
ESM_LVL_EVENT_14	14	R5FSS0_CORE0	R5FSS0_CORE0_TCMADDR_err
ESM_LVL_EVENT_15	15	R5FSS0_CORE1	R5FSS0_CORE1_TCMADDR_err
ESM_LVL_EVENT_16	16	R5FSS1_CORE0	R5FSS1_CORE0_TCMADDR_err
ESM_LVL_EVENT_17	17	R5FSS1_CORE1	R5FSS1_CORE1_TCMADDR_err
ESM_LVL_EVENT_18	18	Reserved	Reserved
ESM_LVL_EVENT_19	19	ECC_AGGREGATOR	soc_eccagg_corr_level
ESM_LVL_EVENT_20	20	ECC_AGGREGATOR	soc_eccagg_uncorr_level
ESM_LVL_EVENT_21	21	DCC0	DCC0_err
ESM_LVL_EVENT_22	22	DCC1	DCC1_err
ESM_LVL_EVENT_23	23	DCC2	DCC2_err
ESM_LVL_EVENT_24	24	DCC3	DCC3_err
ESM_LVL_EVENT_25	25	CORE_PLL	pll_core_lockloss
ESM_LVL_EVENT_26	26	PERI_PLL	pll_per_lockloss
ESM_LVL_EVENT_27	27	RCOSC	rcref_clk_loss_detect
ESM_LVL_EVENT_28	28	HSM	HSM_ESM_high_intr
ESM_LVL_EVENT_29	29	HSM	HSM_ESM_low_intr
ESM_LVL_EVENT_30	30	XTAL	crystal_clockloss
ESM_LVL_EVENT_31	31	Aggregated VBUSP Error	Aggregated_VBUSP_error_H
ESM_LVL_EVENT_32	32	OPTI_FLASH	FOTA_STAT_ERR_INTR
ESM_LVL_EVENT_33	33	Aggregated VBUSM Error	Aggregated_VBUSM_error_H
ESM_LVL_EVENT_34	34	Aggregated VBUSM Error	Aggregated_VBUSM_error_L
ESM_LVL_EVENT_35	35	Reserved	Reserved
ESM_LVL_EVENT_36	36	Reserved	Reserved
ESM_LVL_EVENT_37	37	OPTI_FLASH	FSS_VBUSM_TIMEOUT
ESM_LVL_EVENT_38	38	OPTI_FLASH	OTFA_ERROR
ESM_LVL_EVENT_39	39	OPTI_FLASH	OSPI_ECC_CORR
ESM_LVL_EVENT_40	40	OPTI_FLASH	OSPI_ECC_UNCORR
ESM_LVL_EVENT_41	41	VMON_ERR_H	voltage_monitor_err_H
ESM_LVL_EVENT_42	42	VMON_ERR_L	voltage_monitor_err_L

Table 10-22. ESM0 Interrupt Map (Level) (continued)

Interrupt Input Line	Interrupt ID	Interrupt Source	Interrupt Signal
ESM_LVL_EVENT_43	43	Reserved	Reserved
ESM_LVL_EVENT_44	44	THERMAL_MONITOR	thermal_monitor_critical
ESM_LVL_EVENT_45	45	CPSW	CPSW_ECC_SEC_PEND_INTR
ESM_LVL_EVENT_46	46	CPSW	CPSW_ECC_DED_PEND_INTR
ESM_LVL_EVENT_47	47	R5FSS0_CORE0	R5FSS0_CORE0_ecc_corrected_level.0
ESM_LVL_EVENT_48	48	R5FSS0_CORE0	R5FSS0_CORE0_ecc_uncorrected_level.0
ESM_LVL_EVENT_49	49	R5FSS0_CORE1	R5FSS0_CORE1_ecc_corrected_level.0
ESM_LVL_EVENT_50	50	R5FSS0_CORE1	R5FSS0_CORE1_ecc_uncorrected_level.0
ESM_LVL_EVENT_51	51	R5FSS0_CORE0	R5FSS0_ecc_de_to_esm_0.0
ESM_LVL_EVENT_52	52	R5FSS0_CORE1	R5FSS0_ecc_de_to_esm_1.0
ESM_LVL_EVENT_53	53	R5FSS0_CORE0	R5FSS0_ecc_se_to_esm_0.0
ESM_LVL_EVENT_54	54	R5FSS0_CORE1	R5FSS0_ecc_se_to_esm_1.0
ESM_LVL_EVENT_55	55	R5FSS1_CORE0	R5FSS1_CORE0_ecc_corrected_level.0
ESM_LVL_EVENT_56	56	R5FSS1_CORE0	R5FSS1_CORE0_ecc_uncorrected_level.0
ESM_LVL_EVENT_57	57	R5FSS1_CORE1	R5FSS1_CORE1_ecc_corrected_level.0
ESM_LVL_EVENT_58	58	R5FSS1_CORE1	R5FSS1_CORE1_ecc_uncorrected_level.0
ESM_LVL_EVENT_59	59	R5FSS0_CORE0	R5FSS1_ecc_de_to_esm_0.0
ESM_LVL_EVENT_60	60	R5FSS0_CORE1	R5FSS1_ecc_de_to_esm_1.0
ESM_LVL_EVENT_61	61	R5FSS0_CORE0	R5FSS1_ecc_se_to_esm_0.0
ESM_LVL_EVENT_62	62	R5FSS0_CORE1	R5FSS1_ecc_se_to_esm_1.0
ESM_LVL_EVENT_63	63	EDMA0	tpcc0_err_intagg

Table 10-23. ESM0 Interrupt Map (Pulse)

Interrupt Input Line	Interrupt ID	Interrupt Source	Interrupt Signal
ESM_PLS_EVENT_0	0	WWDT0	RTI0_WWD_NMI
ESM_PLS_EVENT_1	1	WWDT1	RTI1_WWD_NMI
ESM_PLS_EVENT_2	2	WWDT2	RTI2_WWD_NMI
ESM_PLS_EVENT_3	3	WWDT3	RTI3_WWD_NMI
ESM_PLS_EVENT_4	4	EDMA0	TPCC_errint
ESM_PLS_EVENT_5	5	R5FSS0	R5FSS0_bus_monitor_err_pulse.0
ESM_PLS_EVENT_6	6	R5FSS0	R5FSS0_compare_err_pulse.0
ESM_PLS_EVENT_7	7	R5FSS0	R5FSS0_vim_compare_err_pulse.0
ESM_PLS_EVENT_8	8	R5FSS0	R5FSS0_cpu_miscompare_pulse.0
ESM_PLS_EVENT_9	9	R5FSS1	R5FSS1_bus_monitor_err_pulse.0
ESM_PLS_EVENT_10	10	R5FSS1	R5FSS1_compare_err_pulse.0
ESM_PLS_EVENT_11	11	R5FSS1	R5FSS1_vim_compare_err_pulse.0
ESM_PLS_EVENT_12	12	R5FSS1	R5FSS1_cpu_miscompare_pulse.0
ESM_PLS_EVENT_13	13	PRU_ICSSM0	pr1_ecc_ded_err_req
ESM_PLS_EVENT_14	14	PRU_ICSSM0	pr1_ecc_sec_err_req
ESM_PLS_EVENT_15	15	SRAM Bank 0	sram0_ecc_uncorr_pulse
ESM_PLS_EVENT_16	16	SRAM Bank 1	sram1_ecc_uncorr_pulse
ESM_PLS_EVENT_17	17	SRAM Bank 2	sram2_ecc_uncorr_pulse
ESM_PLS_EVENT_18	18	SRAM Bank 3	sram3_ecc_uncorr_pulse
ESM_PLS_EVENT_19	19	CCM0	CCM_0_selftest_err
ESM_PLS_EVENT_20	20	CCM0	CCM_0_lockstep_compare_err
ESM_PLS_EVENT_21	21	CCM1	CCM_1_selftest_err

Table 10-23. ESM0 Interrupt Map (Pulse) (continued)

Interrupt Input Line	Interrupt ID	Interrupt Source	Interrupt Signal
ESM_PLS_EVENT_22	22	CCM1	CCM_1_lockstep_compare_err
ESM_PLS_EVENT_23	23	R5SS0	R5SS0_TMU_COMP_ERR
ESM_PLS_EVENT_24	24	R5SS0	R5SS0_CPU0_TMU_PARITY_ERR
ESM_PLS_EVENT_25	25	R5SS0	R5SS0_CPU1_TMU_PARITY_ERR
ESM_PLS_EVENT_26	26	R5SS1	R5SS1_TMU_COMP_ERR
ESM_PLS_EVENT_27	27	R5SS1	R5SS1_CPU0_TMU_PARITY_ERR
ESM_PLS_EVENT_28	28	R5SS1	R5SS1_CPU1_TMU_PARITY_ERR
ESM_PLS_EVENT_29	29	R5SS0	R5SS0_RL2_COMP_ERR
ESM_PLS_EVENT_30	30	R5SS1	R5SS1_RL2_COMP_ERR
ESM_PLS_EVENT_31	31	ADC_SAFETY	ADC_SAFETY_CHECKEVENT0
ESM_PLS_EVENT_32	32	ADC_SAFETY	ADC_SAFETY_CHECKEVENT1
ESM_PLS_EVENT_33	33	ADC_SAFETY	ADC_SAFETY_CHECKEVENT2
ESM_PLS_EVENT_34	34	ADC_SAFETY	ADC_SAFETY_CHECKEVENT3
ESM_PLS_EVENT_35	35	OPTI_FLASH	OTFA_ECC_UNCORR
ESM_PLS_EVENT_36	36	OPTI_FLASH	OTFA_ECC_CORR
ESM_PLS_EVENT_37	37	SRAM Bank 4	sram4_ecc_uncorr_pulse
ESM_PLS_EVENT_38	38	SRAM Bank 5	sram5_ecc_uncorr_pulse
ESM_PLS_EVENT_39	39	MCAN4	mcanss4_ecc_corr_pls_int
ESM_PLS_EVENT_40	40	MCAN4	mcanss4_ecc_uncorr_pls_int
ESM_PLS_EVENT_41	41	MCAN5	mcanss5_ecc_corr_pls_int
ESM_PLS_EVENT_42	42	MCAN5	mcanss5_ecc_uncorr_pls_int
ESM_PLS_EVENT_43	43	MCAN6	mcanss6_ecc_corr_pls_int
ESM_PLS_EVENT_44	44	MCAN6	mcanss6_ecc_uncorr_pls_int
ESM_PLS_EVENT_45	45	MCAN7	mcanss7_ecc_corr_pls_int
ESM_PLS_EVENT_46	46	MACN7	mcanss7_ecc_uncorr_pls_int

11 Data Movement Architecture

This chapter describes the data movement architecture of the device.

11.1 Data Movement Architecture Overview

This chapter is a high-level summary of the data movement architecture implemented in the device.

11.1.1 Overview

The primary goal of the device Data Movement Architecture and related Subsystems is to ensure that data can be efficiently transferred from a producer to a consumer while meeting the real time requirements of the system.

The Enhanced Data Movement Architecture (EDMA) module aims to facilitate direct memory access (DMA) and provides a consistent Application Programming Interface (API) to the host software.

Data movement tasks are commonly offloaded from the host processor to peripheral hardware to increase system performance. Significant performance gains result from careful design of the interface between the host software and the underlying acceleration hardware. In networking applications, packet transmission and reception are critical tasks. In general purpose compute, ping pong buffer prefetch and store are critical tasks as well as general mis-aligned block copy operations.

The design goals for the device Data Movement Architecture and are as follows:

- Minimize cost
- Minimize host overhead
- Maximize memory use efficiency
- Maximize bus burst efficiency
- Maximize symmetry between transmit/receive operations
- Maximize scalability for number of connections / buffer sizes / queue sizes / protocols supported
- Minimize protocol specific features
- Minimize complexity

11.1.2 Definition of Terms

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 DMA whenever communication is required between entities. Queues can have multiple different implementations and DMA 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.

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.

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.

11.2 Enhanced Direct Memory Access (EDMA)

This section describes the Enhanced Direct Memory Access (EDMA) controller. For features applicable to the EDMA instances in the device, see the device-specific Integration section. The primary purpose of the EDMA controller is to service data transfers programmed between two memory-mapped follower endpoints on the device. The EDMA controller consists of two principle blocks:

- EDMA channel controllers: EDMA_TPCC
- EDMA transfer controllers: EDMA_TPTC

Devices can have multiple instances of EDMA channel controllers, each associated with multiple EDMA transfer controllers.

The EDMA channel controller serves as the user interface for the EDMA controller. The EDMA_TPCC includes parameter RAM (PaRAM), channel control registers, and interrupt control registers. The EDMA_TPCC serves to prioritize incoming software requests or events from peripherals, and submits transfer requests (TR) to the EDMA transfer controller.

The EDMA transfer controllers are responsible for data movement. The transfer request packets (TRP) submitted by the EDMA_TPCC contain the transfer context, based on which the transfer controller issues read/write commands to the source and destination addresses programmed for a given transfer.

11.2.1 EDMA Module Overview

The enhanced direct memory access module, also called EDMA, performs high-performance data transfers between two target endpoints, memories and peripheral devices without microprocessor unit (MPU) or digital signal processor (DSP) support during transfer. EDMA transfer is programmed through a logical EDMA channel, which allows the transfer to be optimally tailored to the requirements of the application.

The EDMA controller is based on two major principal blocks:

- EDMA third-party channel controller (EDMA_TPCC)
- EDMA third-party transfer controller (EDMA_TPTC)

Figure 11-1 shows an overview of the EDMA module.

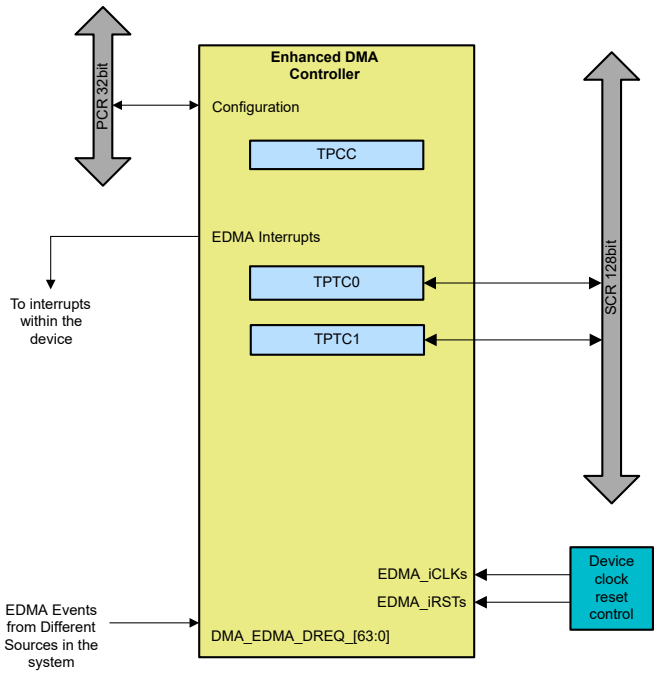


Figure 11-1. EDMA Module Overview

For EDMA instances available on the device, see the device-specific integration section.

The **TPCC** is a high flexible channel controller that serves as both a user interface and an event interface for the EDMA controller. The EDMA_TPCC serves to prioritize incoming software requests or events from peripherals, and submits transfer requests (TRs) to the transfer controller.

The **TPTC** performs read and write transfers by EDMA ports to the target peripherals, as programmed in the Active and Pending set of the registers. The transfer controllers are responsible for data movement, and issue read/write commands to the source and destination addresses programmed for a given transfer in the EDMA_TPCC.

11.2.1.1 EDMA Features

This section shows generic EDMA features. For features applicable to the EDMA instances in the device, see the device-specific Integration section.

The EDMA_TPCC channel controller has the following features:

- Fully orthogonal transfer description:
  - Three transfer dimensions
  - A-synchronized transfers: one dimension serviced per event
  - AB-synchronized transfers: two dimensions serviced per event
  - Independent indexes on source and destination
  - Chaining feature allowing a 3-D transfer based on a single event.
- Flexible transfer definition:
  - Increment or FIFO transfer addressing modes
  - Linking mechanism allows automatic PaRAM set update
  - Chaining allows multiple transfers to execute with one event
- Interrupt generation for the following:
  - Transfer completion
  - Error conditions
- Debug visibility:
  - Queue water marking/threshold
  - Error and status recording to facilitate debug
- 64 DMA request channels:
  - Event synchronization
  - Manual synchronization (CPUs write to event set registers EDMA_TPCC_ESR and EDMA_TPCC_ESRH).
  - Chain synchronization (completion of one transfer triggers another transfer).
- Eight QDMA channels:
  - QDMA channels trigger automatically upon writing to a parameter RAM (PaRAM) set entry.
  - Support for programmable QDMA channel to PaRAM mapping.
- Each PaRAM set can be used for a DMA channel, QDMA channel, or link set.
- Multiple transfer controllers/event queues.
- 16 event entries per event queue.

The **EDMA_TPTC** transfer controller has the following features:

- 64-bit wide read and write ports per TC
- Supports two-dimensional transfers with independent indexes on source and destination (EDMA_TPCC manages the third dimension)
- Support for increment or constant addressing mode transfers
- Interrupt and error support
- Memory-Mapped Register (MMR) bit fields are fixed position in 32-bit MMR regardless of endianness

11.2.2 EDMA Integration

This section describes modules integration in the device, including information about clocks, resets, and hardware requests.

11.2.2.1 EDMA Integration

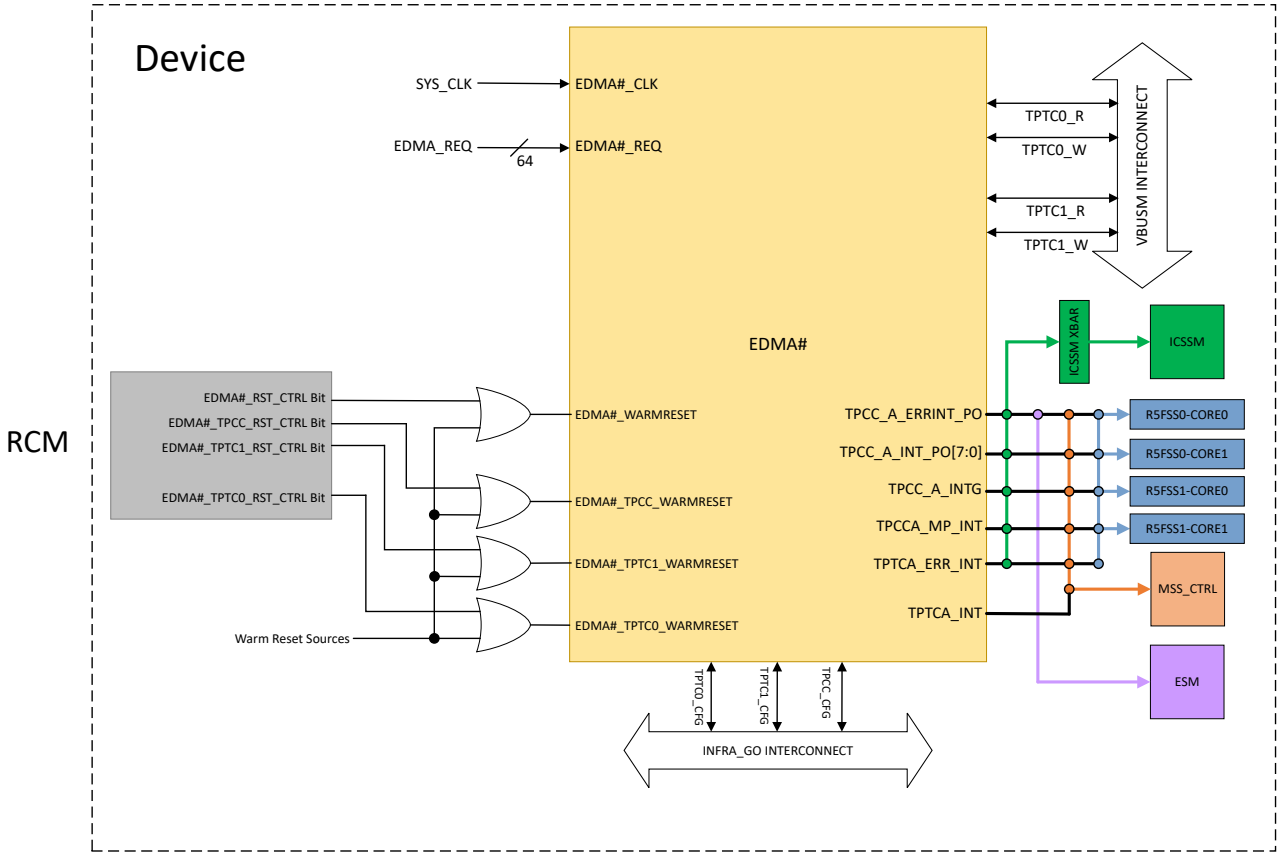


Figure 11-2. EDMA Integration Block Diagram

Note

For more information on the interconnects, see the *System Interconnect* chapter.

For more information on power, reset, and clock management, see the corresponding sections within the *Device Configuration* chapter.

For more information on the device interrupt controllers, see the *Interrupt Controllers* chapter.

11.2.2.2 EDMA Interrupt Aggregator

The following EDMA interrupts are aggregated and sent to the processor:

- TPCC Completion Interrupt
- TPCC Completion Region Interrupts
- TPTCs Completion Interrupt

Table 11-1 shows the associated interrupt and registers for each TPCC instance.

Table 11-1. TPCC Interrupts

TPCC	Interrupt	Registers Space
TPCC0	TPCC0_INTAGG	*_INTAGG_MASK *_INTAGG_STATUS *_INTAGG_STATUS_RAW

For an event to generate an interrupt to the processor, the corresponding bit field must be unmasked in TPCC_x_INTAGG_MASK.

Only an interrupt processor can read the TPCC_x_INTAGG_STATUS register to detect which event triggered the interrupt.

The interrupt can be cleared by writing 0x1 to the corresponding bit in TPCC_x_INTAGG_STATUS.

The software must verfiy that all the aggregated interrupts are cleared so that the level interrupt is de-asserted before exiting the ISR. Only then the software can provide a new pulse interrupt to the processor. Thus, after clearing the software can read the register to confirm a value of 0x0.

The register TPCC_x_INTAGG_STATUS_RAW is set on an event irrespective of the value in TPCC_x_INTAGG_MASK. This field can be cleared by writing 0x1 to the corresponding bit in TPCC_x_INTAGG_STATUS_RAW.

11.2.2.3 EDMA Error Interrupt Aggregator

The following interrupts are aggregated and sent to the processor:

- TPCC Error
- TPCC MPU Error
- TPTCs Error
- TPCC Read and Write Config Space Access error
- TPTCs Read and Write Config Space Access error

Table 11-2. TPCC Error Interrupt Aggregators

TPCC	Interrupt	Registers Space
TPCC0	TPCC0_ERRAGG	*_ERRAGG_MASK *_ERRAGG_STATUS *_ERRAGG_STATUS_RAW

For an event to generate an interrupt to the processor, the corresponding bit field must be unmasked in TPCC_x_ERRAGG_MASK.

Only an interrupt processor can read the TPCC_x_ERRAGG_STATUS register to detect which event triggered the interrupt.

The interrupt can be cleared by writing 0x1 to the corresponding bit in TPCC_x_ERRAGG_STATUS.

The software must ensure that all the aggregated interrupts are cleared so that the level interrupt is de-asserted before exiting the ISR. Only then is it ensured that a new pulse interrupt is generated to the processor. Thus, after clearing the software should read the register to confirm a value of 0x0

The register TPCC_x_ERRAGG_STATUS_RAW is set on an event irrespective of the value in TPCC_x_ERRAGG_MASK. This field can be cleared by writing 0x1 to the corresponding bit in TPCC_x_ERRAGG_STATUS_RAW.

11.2.2.4 EDMA Configuration

- The device has 1 channel controller: TPCC0 and two transfer controllers: TPTC0 and TPTC1.

Table 11-3. EDMA Channel Controller Configuration

Parameters	TPCC
DMA Channel	64
PaRAM Entires	256
QDMA Channel	8
Event queues	2
Mem Protection	Yes
Channel Mapping	Yes

Table 11-3. EDMA Channel Controller Configuration (continued)

Parameters	TPCC
Num TCs	2
Num Interrupt Channel	64
Num Regions	8

Table 11-4. EDMA Transfer Controller Configuration

Parameters	TPTC[0/1]
FIFO Size	512
TR Pipe Depth	4
Bus Width	8
Read Cmd Num	8
Write Cmd Num	8
RAM ECC	Yes

Default Burst Size configuration (DBS)

All TPTCs in the device support four different configurable default-burst-sizes. Table 11-5 shows the config-value to DBS mapping.

Table 11-5. Config Value to DBS Mapping

Config value	Burst size
2'b00	16 bytes
2'b01	32 bytes
2'b10	64 bytes
2'b11	128 bytes

Table 11-6. TPTC DBS Configuration Registers

TPTC instance	Corresponding Register
TPTC[0/1]	TPTC_DBS_CONFIG::TPTC_DBS_CONFIG_TPTC[0/1]

11.2.3 EDMA Controller Functional Description

This chapter discusses the architecture of the EDMA controller. The description contained in this section is generic to the EDMA module, and not all features mentioned here are supported by the device. See the EDMA integration section of the device to determine the applicability of these features.

11.2.3.1 Block Diagram

Figure 11-3 shows the functional block diagram of the EDMA controller.

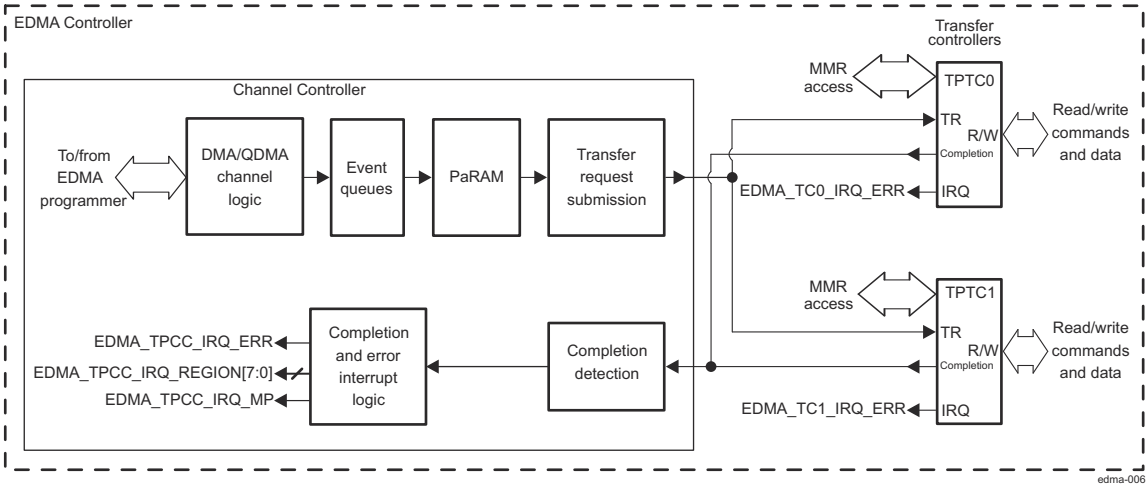


Figure 11-3. EDMA Controller Block Diagram

11.2.3.1.1 Third-Party Channel Controller

The TPCC is the EDMA transfer scheduler responsible for scheduling, arbitrating, and issuing user programmed transfers to the two TPTCs.

The functional block diagram below describes EDMA channel controller (EDMA_TPCC).

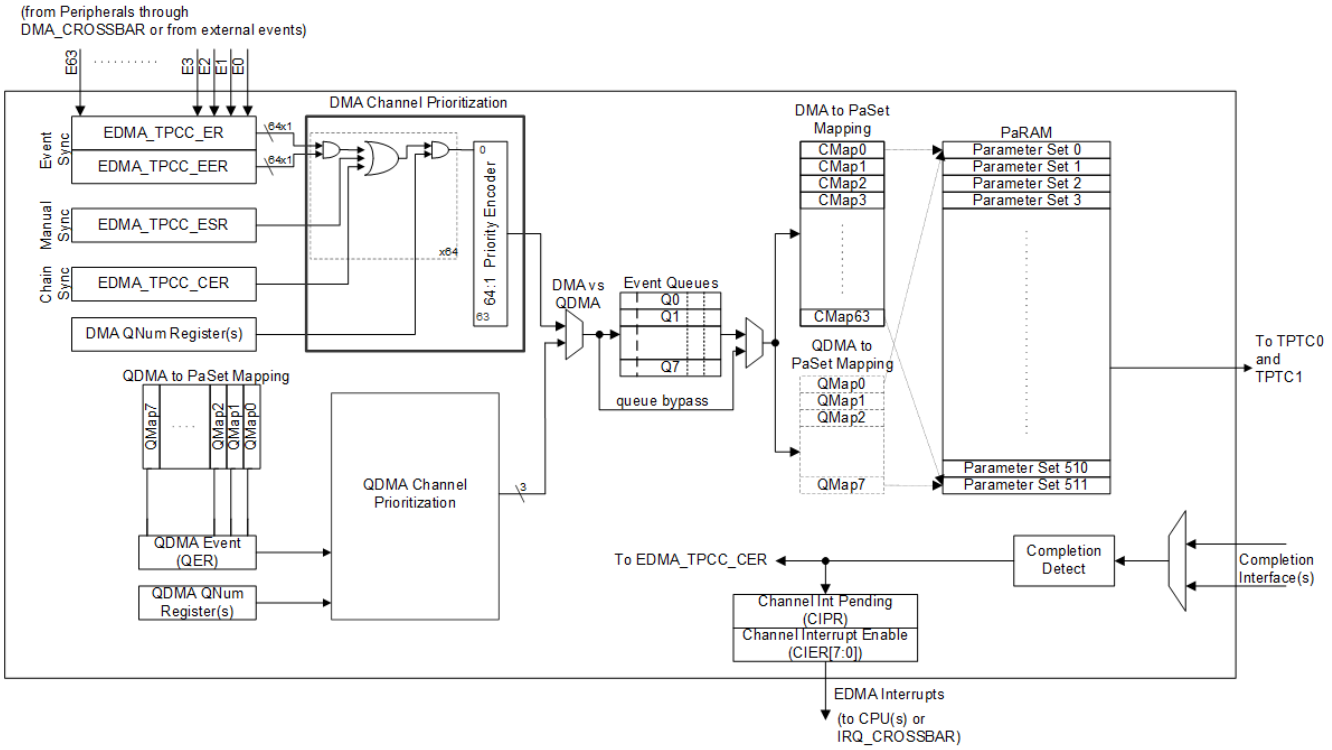


Figure 11-4. EDMA Channel Controller Block Diagram

A. Although the block is depicted twice in EDMA Channel Controller Block Diagram, there is only one physical register set for the QDMA to

PaRAM set mapping block.

The main blocks of the EDMA_TPCC are as follows:

- Parameter RAM (PaRAM): The PaRAM maintains parameter sets for channel and reload parameter sets. The PaRAM must be written with the transfer context for the desired channels and link parameter sets. EDMA_TPCC processes and sets based on a trigger event and submits a transfer request (TR) to the transfer controllers.
- EDMA event and interrupt processing registers: Allows mapping of events to parameter sets, enable/disable events, enable/disable interrupt conditions, and clearing interrupts.
- Completion detection: The completion detect block detects completion of transfers by the EDMA_TPTCs or follower peripherals. The completion of transfers can be used optionally to chain trigger new transfers or to assert interrupts.
- Event queues: Event queues form the interface between the event detection logic and the transfer request submission logic.
- Memory protection registers: Memory protection registers define the accesses (privilege level and requestor(s)) that are allowed to access the DMA channel shadow region view(s) and regions of PaRAM.

Other functions include the following:

- Region registers: Region registers allow DMA resources (DMA channels and interrupts) to be assigned to unique regions that different EDMA programmers own (for example, DSPs).
- Debug registers: Debug registers allow debug visibility by providing registers to read the queue status, controller status, and missed event status.

The EDMA_TPCC includes two channel types: DMA channels (64 channels) and QDMA channels (8 channels).

Each channel is associated with a given event queue/transfer controller and with a given PaRAM set. These channels are identical. The main difference between a DMA channel and a QDMA channel is the method that the system uses to trigger transfers.

- DMA channels are triggered by external events by the event set registers EDMA_TPCC_ESR and EDMA_TPCC_ESRH, or through chaining register EDMA_TPCC_CER.
- QDMA channels are triggered automatically (auto-triggered) by the CPU. QDMAs allow a minimum number of linear writes to be issued to the TPCC to force a series of transfers to occur.

The TPCC arbitrates among pending DMA and QDMA events with a fixed [64:1] and [8:1] priority encoder for these events, respectively (a low channel number corresponds to a high priority).

DMA events are always higher priority than QDMA events. The higher-priority event is placed in the event queue to await submission to the transfer controllers, which occurs at the earliest opportunity. Each event queue is serviced in FIFO order, with a maximum of 16 queued events per event queue. If more than one TPTC is ready to be programmed with a transmission request (TR), the event queues are serviced with fixed priority: Q0 is higher than Q1. When an event is ready to be queued and the event queue and the TC channel are empty, the event bypasses the event queue and goes directly to the PaRAM processing logic for submission to the appropriate TC. If the transfer request TR bus or PaRAM processing are busy, the bypass path is not used. The bypass is not used to dequeue for a higher-priority event.

Events are extracted from the event queue when the EDMA_TPTC is available for a new TR to be programmed into the EDMA_TPTC (signaled with the empty signal, indicating an empty program register set). As an event is extracted from the event queue, the associated PaRAM entry is processed and submitted to the TPTC as a TR. The TPCC updates the appropriate counts and addresses in the PaRAM entry in anticipation of the next trigger event for that PaRAM entry.

The EDMA_TPCC also has an error detection logic that causes an error interrupt generation on various error conditions (for example: missed events EDMA_TPCC_EMR and EDMA_TPCC_EMRH registers, exceeding event queue thresholds in EDMA_TPCC_CCERR register, etc.).

11.2.3.1.2 Third-Party Transfer Controller

The TPTC module is the EDMA transfer engine that generates transfers as programmed in dedicated working registers, using two dedicated controller ports: a read-only port and a write-only port.

Figure 11-5 shows a functional block diagram and of the EDMA transfer controller (EDMA_TPTC) and its connection to the EDMA_TPCC.

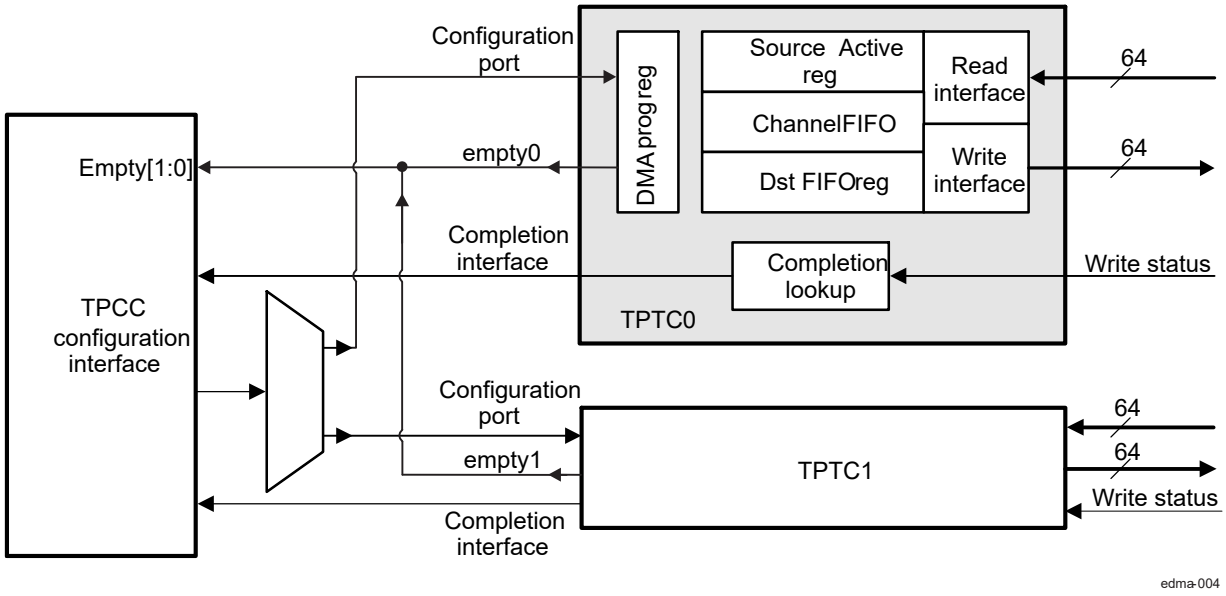


Figure 11-5. TPTC Block Diagram

Note

The port data bus width of the instances of the TPTC is fixed at 64 bits.

Two instances of the EDMA_TPTC generate concurrent traffic on the L3_VBUSM interconnect. Each TC controller consists of the following components:

- DMA Program Register Set: Stores the context for the DMA transfer that is loaded into the active register set when the current active register set completes. The CPU or TPCC programs the Program Register Set, not the active register set. For typical standalone operation, the CPU programs the Program Register while the TC services the Active register set. The Program Register set includes ownership control such that CPU software and the EDMA stay synchronized relative to one another.
- Source Active Register Set : Stores the context (src/dst/cnt/etc) for the DMA Transfer Request (TR) in progress in the Read Controller. The Active register set is split into independent Source and Destination, because the source interconnect controller and the destination interconnect controller operate independently of one another.
- Destination FIFO Register Set: Stores the context (src/dst/cnt/etc) for the DMA Transfer Request (TR) in progress, or pending, in the Write Controller. The pending register must allow the source controller to begin processing a new TR while the destination register set processes the previous TR.
- Channel FIFO: Temporary holding buffer for in-flight data. The read return data of the source peripheral is stored in the Data FIFO, and then is written to the destination peripheral by the write command/data bus.
- Read Controller/Interconnect Read Interface: The Interconnect read interface issues optimally sized read commands to the source peripheral, based on a burst size of 32 bytes and available landing space in the channel FIFO.
- Write controller/Interconnect Write interface: The local interconnect write interface issues optimally sized write commands to the destination peripheral, based on a burst size of 32 bytes and available data in the channel FIFO.
- Completion interface: sends completion codes to the EDMA_TPCC when a transfer completes and generates interrupts and chained events in the TPCC module.

- Configuration port: Target interface that provides read/write access to program registers and read access to all memory-mapped TPTC registers.

When one EDMA_TPTC module is idle and receive its first TR, DMA program register set receives the TR, where it transitions to the DMA source active set and the destination FIFO register set immediately. The second TR (if pending from EDMA_TPCC) is loaded into the DMA program set, ensuring it can start as soon as possible when the active transfer completes. As soon as the current active set is exhausted, the TR is loaded from the DMA program register set into the DMA source active register set as well as to the appropriate entry in the destination FIFO register set.

The read controller issues read commands controlled by the rules of command fragmentation and optimization. These are issued only when the data FIFO has space available for the data read. When sufficient data is in the data FIFO, the write controller starts issuing a write command again following the rules for command fragmentation and optimization.

Depending on the number of entries, the read controller can process up to two or four transfer requests ahead of the destination subject to the amount of free data FIFO.

11.2.3.2 Types of EDMA Controller Transfers

An EDMA transfer is always defined in terms of three dimensions. Figure 11-6 shows the three dimensions used by EDMA controller transfers. These three dimensions are defined as:

- 1st Dimension or Array (A): The 1st dimension in a transfer consists of EDMA_TPCC_ABCNT_n[15:0] ACNT contiguous bytes.
- 2nd Dimension or Frame (B): The 2nd dimension in a transfer consists of EDMA_TPCC_ABCNT_n[31:16] BCNT arrays of ACNT bytes. Each array transfer in the 2nd dimension is separated from each other by an index programmed using bit-fields EDMA_TPCC_BIDX_n[15:0] SBIDX or EDMA_TPCC_BIDX_n[31:16] DBIDX.
- 3rd Dimension or Block (C): The 3rd dimension in a transfer consists of CCNT frames of BCNT arrays of ACNT bytes. The Count for 3rd Dimension is defined in PaRAM memory EDMA_TPCC_CCNT_n[15:0] CCNT. Each transfer in the 3rd dimension is separated from the previous by an index programmed using EDMA_TPCC_CIDX_n[15:0] SCIDX or EDMA_TPCC_CIDX_n[31:16] DCIDX.

Note

The reference point for the index depends on the synchronization type. The amount of data transferred upon receipt of a trigger/synchronization event is controlled by the synchronization types (EDMA_TPCC_OPT_n[2] SYNCDIM bit). For these three dimensions, only two synchronization types are supported: A-synchronized transfers and AB-synchronized transfers.

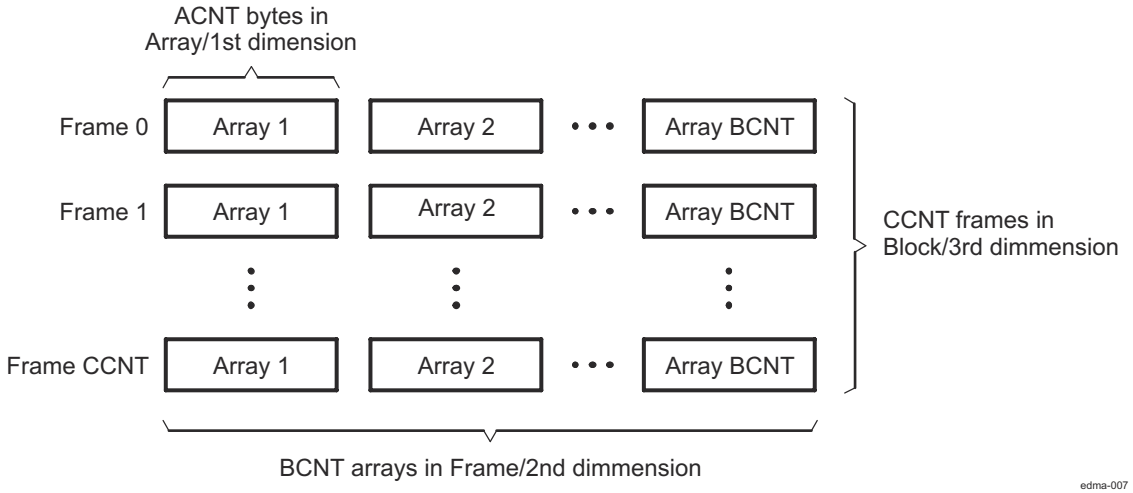


Figure 11-6. Definition of ACNT, BCNT, and CCNT

11.2.3.2.1 A-Synchronized Transfers

In an A-synchronized transfer, each EDMA sync event initiates the transfer of the 1st dimension of EDMA_TPCC_ABCNT_n[15:0] ACNT bytes, or one array of ACNT bytes. Each event/TR packet conveys the transfer information for one array only. Thus, BCNT × CCNT events are needed to completely service a PaRAM set.

Arrays are always separated by EDMA_TPCC_BIDX_n[15:0]SBIDX and EDMA_TPCC_BIDX_n[31:16] DBIDX, as shown in Figure 11-7, where the start address of Array N is equal to the start address of Array N – 1 plus source (SRC) or destination (DST) in EDMA_TPCC_BIDX_n register.

Frames are always separated by EDMA_TPCC_CIDX_n[15:0] SCIDX and EDMA_TPCC_CIDX_n[31:16] DCIDX. For A-synchronized transfers, after the frame is exhausted, the address is updated by adding SRCCIDX/ DSTCIDX to the beginning address of the last array in the frame. As in Figure 11-7, SRCCIDX / DSTCIDX is the difference between the start of Frame 0 Array 3 to the start of Frame 1 Array 0.

Figure 11-7 shows an A-synchronized transfer of 3 (CCNT) frames of 4 (BCNT) arrays of n (ACNT) bytes. In this example, a total of 12 sync events (BCNT × CCNT) exhaust a PaRAM set. See Figure 11-7 for details on parameter set updates.

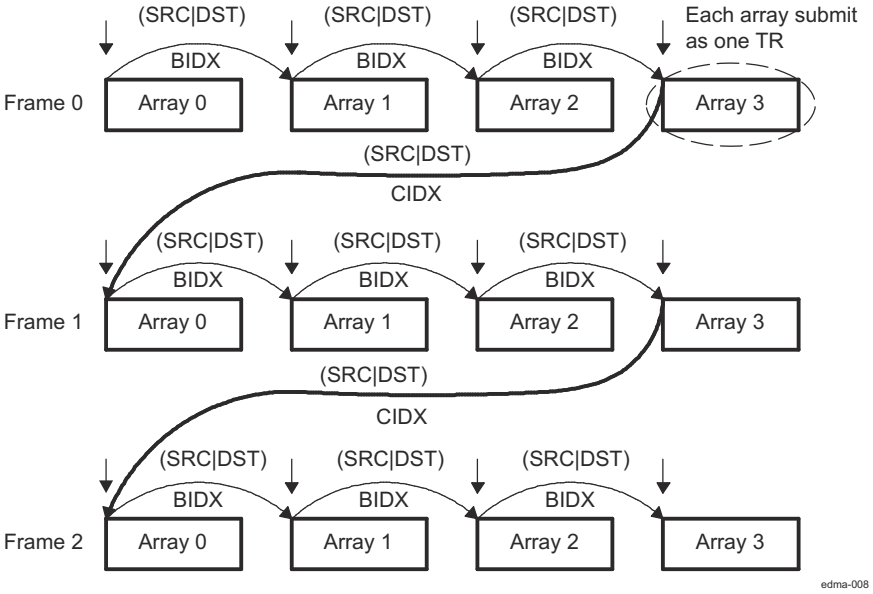


Figure 11-7. A-Synchronized Transfers (ACNT = n, BCNT = 4, CCNT = 3)

11.2.3.2.2 AB-Synchronized Transfers

In a AB-synchronized transfer, each EDMA sync event initiates the transfer of 2 dimensions or one frame. Each event/TR packet conveys information for one entire frame of BCNT_n arrays of ACNT_n bytes. Thus, EDMA_TPCC_CCNT_n events are needed to completely service a PaRAM set.

Arrays are always separated by EDMA_TPCC_BIDX_n[15:0] SBIDX and EDMA_TPCC_BIDX_n[31:16] DBIDX as shown in Figure 11-8. Frames are always separated by SRCCIDX and DSTCIDX.

Note that for AB-synchronized transfers, after a TR for the frame is submitted, the address update is to add EDMA_TPCC_CIDX_n[15:0] SCIDX / EDMA_TPCC_CIDX_n[31:16] DCIDX to the beginning address of the beginning array in the frame. This is different from A-synchronized transfers where the address is updated by adding SRCCIDX/DSTCIDX to the start address of the last array in the frame. See Section 11.2.3.3.6 for details on parameter set updates.

Figure 11-8 shows an AB-synchronized transfer of 3 (CCNT) frames of 4 (BCNT) arrays of *n* (ACNT) bytes. In this example, a total of 3 sync events (CCNT) exhaust a PaRAM set; that is, a total of 3 transfers of 4 arrays each completes the transfer.

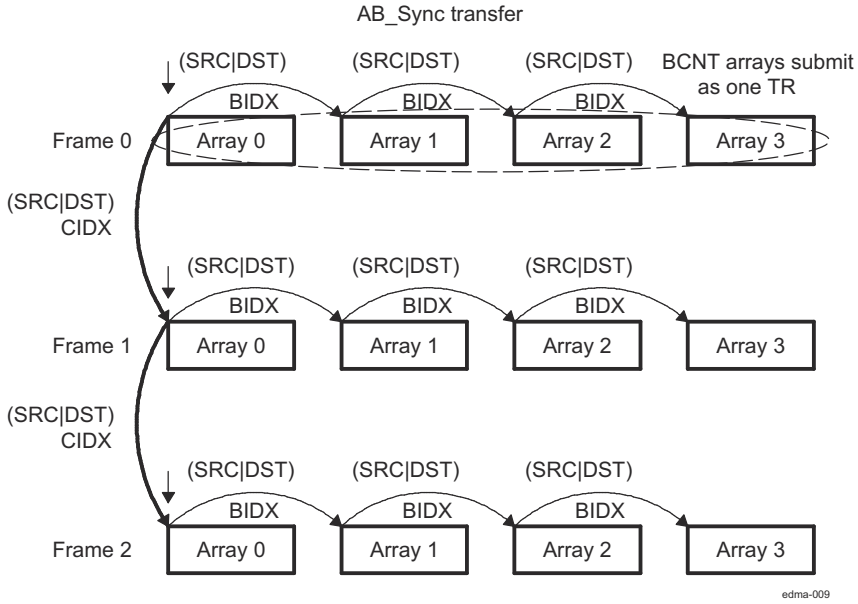


Figure 11-8. AB-Synchronized Transfers (ACNT = n, BCNT = 4, CCNT = 3)

Note

ABC-synchronized transfers are not directly supported. It can be logically achieved by chaining between multiple AB-synchronized transfers.

11.2.3.3 Parameter RAM (PaRAM)

The EDMA controller is a RAM-based architecture. The transfer context (source/destination addresses, count, indexes, etc.) for DMA or QDMA channels is programmed in a parameter RAM table in EDMA_TPCC. The PaRAM table is segmented into multiple PaRAM sets. Each PaRAM set includes eight four-byte PaRAM set entries (32-bytes total per PaRAM set), which includes typical DMA transfer parameters such as source address, destination address, transfer counts, indexes, options, etc.

The PaRAM structure supports flexible ping-pong, circular buffering, channel chaining, and auto-reloading (linking).

The contents of the PaRAM include the following:

- 256 PaRAM sets
- 64 channels that are direct mapped and can be used as link for QDMA sets if not used for DMA channels

- 8 channels remain for link or QDMA sets

By default, all channels map to PaRAM set to 0 and should be remapped before use by EDMA_TPCC_DCHMAPN_m and EDMA_TPCC_QCHMAPN_j registers. This can be done in the device boot flow.

Table 11-7. EDMA Parameter RAM Contents

PaRAM Set Number	Base Address	Parameters(1)
0	EDMA Base Address + 4000h to EDMA Base Address + 401Fh	PaRAM set 0
1	EDMA Base Address + 4020h to EDMA Base Address + 403Fh	PaRAM set 1
2	EDMA Base Address + 4040h to EDMA Base Address + 405Fh	PaRAM set 2
3	EDMA Base Address + 4060h to EDMA Base Address + 407Fh	PaRAM set 3
4	EDMA Base Address + 4080h to EDMA Base Address + 409Fh	PaRAM set 4
5	EDMA Base Address + 40A0h to EDMA Base Address + 40BFh	PaRAM set 5
6	EDMA Base Address + 40C0h to EDMA Base Address + 40DFh	PaRAM set 6
7	EDMA Base Address + 40E0h to EDMA Base Address + 40FFh	PaRAM set 7
8	EDMA Base Address + 4100h to EDMA Base Address + 411Fh	PaRAM set 8
9	EDMA Base Address + 4120h to EDMA Base Address + 413Fh	PaRAM set 9
...	...	...
63	EDMA Base Address + 47E0h to EDMA Base Address + 47FFh	PaRAM set 63
64	EDMA Base Address + 4800h to EDMA Base Address + 481Fh	PaRAM set 64
65	EDMA Base Address + 4820h to EDMA Base Address + 483Fh	PaRAM set 65
...	...	...
127	EDMA Base Address + 5000h to EDMA Base Address + 4FE0h	PaRAM set 127

(1) The device has 8 QDMA channels that can be mapped to any parameter set number from 0 to .

Note

11.2.3.3.1 PaRAM

Each parameter set of PaRAM is organized into eight 32-bit words or 32 bytes, as shown in PaRAM Set and described in EDMA Channel Parameter Description. Each PaRAM set consists of 16-bit and 32-bit parameters.

Figure 11-9. PaRAM Set

Note

Figure above is a representation of 128 bit entries. For device specific details please refer to EDMA configuration chapter.

Table 11-8. EDMA Channel Parameter Description

Offset Address (bytes)	Acronym	Parameter	Description
0h	OPT	Channel Options EDMA_TPCC_OPT_n register	Transfer configuration options
4h	SRC	Channel Source Address EDMA_TPCC_SRC_n register	The byte address from which data is transferred
8h(1)	ACNT	Count for 1st Dimension EDMA_TPCC_ABCNT_n[15:0] ACNT bit-field.	Unsigned value specifying the number of contiguous bytes within an array (first dimension of the transfer). Valid values range from 1 to 65 535.
	BCNT	Count for 2nd Dimension EDMA_TPCC_ABCNT_n[31:16] BCNT bit-field.	Unsigned value specifying the number of arrays in a frame, where an array is ACNT bytes. Valid values range from 1 to 65 535.
Ch	DST	Channel Destination Address EDMA_TPCC_DST_n register	The byte address to which data is transferred
10h(1)	SBIDX	Source BCNT Index EDMA_TPCC_BIDX_n[15:0] SBIDX bit-field.	Signed value specifying the byte address offset between source arrays within a frame (2nd dimension). Valid values range from –32 768 and 32 767.
	DBIDX	Destination BCNT Index EDMA_TPCC_BIDX_n[31:16] DBIDX bit-field.	Signed value specifying the byte address offset between destination arrays within a frame (2nd dimension). Valid values range from –32 768 and 32 767.
14h(1)	LINK	Link Address EDMA_TPCC_LNK_n[15:0] LINK bit-field	The PaRAM address containing the PaRAM set to be linked (copied from) when the current PaRAM set is exhausted. A value of FFFFh specifies a null link.
	BCNTRLD	BCNT Reload EDMA_TPCC_LNK_n[31:16] BCNTRLD bit-field	The count value used to reload BCNT when BCNT decrements to 0 (TR is submitted for the last array in 2nd dimension). Only relevant in A-synchronized transfers.
18h(1)	SCIDX	Source CCNT index. EDMA_TPCC_CIDX_n[15:0] SCIDX bit-field.	Signed value specifying the byte address offset between frames within a block (3rd dimension). Valid values range from –32 768 and 32 767. A-synchronized transfers: The byte address offset from the beginning of the last source array in a frame to the beginning of the first source array in the next frame. AB-synchronized transfers: The byte address offset from the beginning of the first source array in a frame to the beginning of the first source array in the next frame.
	DCIDX	Destination CCNT index. EDMA_TPCC_CIDX_n[31:16] DCIDX bit-field.	Signed value specifying the byte address offset between frames within a block (3rd dimension). Valid values range from –32 768 and 32 767. A-synchronized transfers: The byte address offset from the beginning of the last destination array in a frame to the beginning of the first destination array in the next frame. AB-synchronized transfers: The byte address offset from the beginning of the first destination array in a frame to the beginning of the first destination array in the next frame.
1Ch	CCNT	Count for 3rd Dimension. EDMA_TPCC_CCNT_n[15:0] CCNT bit-field.	Unsigned value specifying the number of frames in a block, where a frame is BCNT arrays of ACNT bytes. Valid values range from 1 to 65 535.
	Reserved	Reserved	Reserved. Always write 0 to this bit; writes of 1 to this bit are not supported and attempts cam result in undefined behavior.

(1) If OPT, SRC, or DST is the trigger word for a QDMA transfer, then it is required to do a 32-bit access to that field. Furthermore, it is recommended to perform only 32-bit accesses on the parameter RAM for best code compatibility. For example, switching the endianness of the processor swaps addresses of the 16-bit fields, but 32-bit accesses avoid the issue entirely.

11.2.3.3.2 EDMA Channel PaRAM Set Entry Fields

11.2.3.3.2.1 Channel Options Parameter (OPT)

This is the control register for TPCC channel configuration options. Refer to the EDMA_TPCC_OPT_n register bitfield description in the AM263Px Register Addendum for additional details.

11.2.3.3.2.2 Channel Source Address (SRC)

The 32-bit source address parameter specifies the starting byte address of the source. For SAM in increment mode, there are no alignment restrictions imposed by EDMA. For SAM in FIFO addressing mode, it must program the source address to be aligned to a 256-bit aligned address (5 LSBs of address must be 0). If this rule is not observed, the EDMA_TPTC returns an error. Refer to Section 11.2.3.12.3 *Error Generation* for additional details.

11.2.3.3.2.3 Channel Destination Address (DST)

The 32-bit destination address parameter specifies the starting byte address of the destination. For DAM in increment mode, there are no alignment restrictions imposed by EDMA. For DAM in FIFO addressing mode, it must program the destination address to be aligned to a 256-bit aligned address (5 LSBs of address must be 0). If this rule is not observed, the EDMA_TPTC returns an error. Refer to *Error Generation* for additional details.

11.2.3.3.2.4 Count for 1st Dimension (ACNT)

EDMA_TPCC_ABCNT_n[15:0] ACNT represents the number of bytes within the 1st dimension of a transfer. ACNT is a 16-bit unsigned value with valid values between 1 and 65535. Therefore, the maximum number of bytes in an array is 65 535 bytes (64K – 1 bytes). ACNT must be greater than or equal to 1 for a TR to be submitted to EDMA_TPTC. A transfer with ACNT equal to 0 is considered either a null or dummy transfer. A dummy or null transfer generates a completion code depending on the settings of the completion bit fields in EDMA_TPCC_OPT_n.

Refer to Section 11.2.3.3.5 *Dummy Versus Null Transfer Comparison* and Section 11.2.3.5.3 *Dummy or Null Completion* for details on dummy/null completion conditions.

11.2.3.3.2.5 Count for 2nd Dimension (BCNT)

EDMA_TPCC_ABCNT_n[15:0] BCNT is a 16-bit unsigned value that specifies the number of arrays of length ACNT. For normal operation, valid values for BCNT are between 1 and 65 535. Therefore, the maximum number of arrays in a frame is 65 535 (64K – 1 arrays). A transfer with BCNT equal to 0 is considered either a null or dummy transfer. A dummy or null transfer generates a completion code depending on the settings of the completion bit fields in EDMA_TPCC_OPT_n.

Refer to Section 11.2.3.3.5 *Dummy Versus Null Transfer Comparison* and Section 11.2.3.5.3 *Dummy or Null Completion* for details on dummy/null completion conditions.

11.2.3.3.2.6 Count for 3rd Dimension (CCNT)

EDMA_TPCC_CCNT_n[15:0] CCNT is a 16-bit unsigned value that specifies the number of frames in a block. Valid values for CCNT are between 1 and 65 535. Therefore, the maximum number of frames in a block is 65 535 (64K – 1 frames). A transfer with CCNT equal to 0 is considered either a null or dummy transfer. A dummy or null transfer generates a completion code depending on the settings of the completion bit fields in EDMA_TPCC_OPT_n.

A CCNT value of 0 is considered either a null or dummy transfer.

Refer to Section 11.2.3.3.5 *Dummy Versus Null Transfer Comparison* and Section 11.2.3.5.3 *Dummy or Null Completion* for details on dummy/null completion conditions.

11.2.3.3.2.7 BCNT Reload (BCNTRLD)

EDMA_TPCC_LNK_n[31:16] BCNTRLD is a 16-bit unsigned value used to reload the EDMA_TPCC_ABCNT_n[15:0] BCNT field once the last array in the 2nd dimension is transferred. This field is only used for A-synchronized transfers. In this case, the EDMA_TPCC decrements the BCNT value by 1 on each TR submission. When BCNT reaches 0, the EDMA_TPCC decrements CCNT and uses the BCNTRLD value to reinitialize the BCNT value.

For AB-synchronized transfers, the EDMA_TPCC submits the BCNT in the TR and the EDMA_TPTC decrements BCNT appropriately. For AB-synchronized transfers, BCNTRLD is not used.

11.2.3.3.2.8 Source B Index (SBIDX)

EDMA_TPCC_BIDX_n[15:0] SBIDX is a 16-bit signed value (2s complement) used for source address modification between each array in the 2nd dimension. Valid values for EDMA_TPCC_BIDX_n[15:0] SBIDX are between –32 768 and 32 767. It provides a byte address offset from the beginning of the source array to the beginning of the next source array. It applies to both A-synchronized and AB-synchronized transfers. Some examples:

- EDMA_TPCC_BIDX_n[15:0] SBIDX = 0000h (0): no address offset from the beginning of an array to the beginning of the next array. All arrays are fixed to the same beginning address.
- EDMA_TPCC_BIDX_n[15:0] SBIDX = 0003h (+3): the address offset from the beginning of an array to the beginning of the next array in a frame is 3 bytes. For example, if the current array begins at address 1000h, the next array begins at 1003h.
- EDMA_TPCC_BIDX_n[15:0] SBIDX = FFFFh (–1): the address offset from the beginning of an array to the beginning of the next array in a frame is –1 byte. For example, if the current array begins at address 5054h, the next array begins at 5053h.

11.2.3.3.2.9 Destination B Index (DBIDX)

EDMA_TPCC_BIDX_n[31:16] DBIDX is a 16-bit signed value (2s complement) used for destination address modification between each array in the 2nd dimension. Valid values for EDMA_TPCC_BIDX_n[31:16] DBIDX are between –32 768 and 32 767. It provides a byte address offset from the beginning of the destination array to the beginning of the next destination array within the current frame. It applies to both A-synchronized and AB-synchronized transfers. Refer to Section 11.2.3.3.2.8 *Source B Index (SBIDX)* for examples.

11.2.3.3.2.10 Source C Index (SCIDX)

EDMA_TPCC_CIDX_n[15:0] SCIDX is a 16-bit signed value (2s complement) used for source address modification in the 3rd dimension. Valid values for EDMA_TPCC_CIDX_n[15:0] SCIDX are between –32 768 and 32 767. It provides a byte address offset from the beginning of the current array (pointed to by SRC address) to the beginning of the first source array in the next frame. It applies to both A-synchronized and AB-synchronized transfers.

Note

When SCIDX is applied, the current array in an A-synchronized transfer is the last array in the frame (Figure 11-7), while the current array in an AB-synchronized transfer is the first array in the frame (Figure 11-8).

11.2.3.3.2.11 Destination C Index (DCIDX)

EDMA_TPCC_CIDX_n[31:16] DCIDX is a 16-bit signed value (2s complement) used for destination address modification in the 3rd dimension. Valid values are between –32 768 and 32 767. It provides a byte address offset from the beginning of the current array (pointed to by DST address) to the beginning of the first destination array TR in the next frame. It applies to both A-synchronized and AB-synchronized transfers.

Note

When DCIDX is applied, the current array in an A-synchronized transfer is the last array in the frame (Figure 11-7), while the current array in a AB-synchronized transfer is the first array in the frame (Figure 11-8).

11.2.3.3.2.12 Link Address (LINK)

The EDMA_TPCC provides a mechanism, called linking, to reload the current PaRAM set upon its natural termination (that is, after the count fields are decremented to 0) with a new PaRAM set. The 16-bit parameter EDMA_TPCC_LNK_n[15:0] LINK specifies the byte address offset in the PaRAM from which the EDMA_TPCC loads/reloads the next PaRAM set during linking.

It must program the link address to point to a valid aligned 32-byte PaRAM set. The 5 LSBs of the LINK field should be cleared to 0.

The EDMA_TPCC ignores the upper 2 bits of the LINK entry, allowing the flexibility of programming the link address as either an absolute/literal byte address or use the PaRAM-base-relative offset address. Therefore, if it use the literal address with a range from 4000h to 7FFFh, it will be treated as a PaRAM-base-relative value of 0000h to 3FFFh.

It should check that the programed value in the EDMA_TPCC_LNK_n[15:0] LINK field is correctly, so that link update is requested from a PaRAM address that falls in the range of the available PaRAM addresses on the device.

Value of FFFFh in EDMA_TPCC_LNK_n[15:0] LINK bit-field is referred to as a NULL link that should cause the EDMA_TPCC to perform an internal write of 0 to all entries of the current PaRAM set, except for the EDMA_TPCC_LNK_n[15:0] LINK field is set to FFFFh. Also, see Section 11.2.3.5 *Completion of a DMA Transfer* for details on terminating a transfer.

11.2.3.3.3 Null PaRAM Set

A null PaRAM set is defined as a PaRAM set where all count fields (EDMA_TPCC_ABCNT_n[15:0] ACNT, EDMA_TPCC_ABCNT_n[31:16] BCNT, and EDMA_TPCC_CCNT_n[15:0] CCNT) are cleared to 0. If a PaRAM set associated with a channel is a NULL set, then when serviced by the EDMA_TPCC, the bit corresponding to the channel is set in the associated event missed register (EDMA_TPCC_EMR, EDMA_TPCC_EMRH, or EDMA_TPCC_QEMR). This bit remains set in the associated secondary event register (EDMA_TPCC_SER, EDMA_TPCC_SERH, or EDMA_TPCC_QSER).
This implies that any future events on the same channel are ignored by the EDMA_TPCC and it is required to clear the bit in EDMA_TPCC_SER, EDMA_TPCC_SERH, or EDMA_TPCC_QSER for the channel. This is considered an error condition, since events are not expected on a channel that is configured as a null transfer.

11.2.3.3.4 Dummy PaRAM Set

A dummy PaRAM set is defined as a PaRAM set where at least one of the count fields (EDMA_TPCC_ABCNT_n[15:0] ACNT, EDMA_TPCC_ABCNT_n[31:16] BCNT, or EDMA_TPCC_CCNT_n[15:0] CCNT) is cleared to 0 and at least one of the count fields is nonzero.

If a PaRAM set associated with a channel is a dummy set, then when serviced by the EDMA_TPCC, it will not set the bit corresponding to the channel (DMA/QDMA) in the event missed register (EDMA_TPCC_EMR, EDMA_TPCC_EMRH, or EDMA_TPCC_QEMR) and the secondary event register (EDMA_TPCC_SER, EDMA_TPCC_SERH, or EDMA_TPCC_QSER) bit gets cleared similar to a normal transfer. Future events on that channel are serviced. A dummy transfer is a legal transfer of 0 bytes.

11.2.3.3.5 Dummy Versus Null Transfer Comparison

There are some differences in the way the EDMA_TPCC logic treats a dummy versus a null transfer request. A null transfer request is an error condition, but a dummy transfer is a legal transfer of 0 bytes. A null transfer causes an error bit (E*n*) in EDMA_TPCC_EMR to get set and the E*n* bit in EDMA_TPCC_SER remains set, essentially preventing any further transfers on that channel without clearing the associated error registers.

Table 11-9 summarizes the conditions and effects of null and dummy transfer requests.

Table 11-9. Dummy and Null Transfer Request

Feature	Null TR	Dummy TR
EDMA_TPCC_EMR / EDMA_TPCC_EMRH / EDMA_TPCC_QEMR is set	Yes	No
EDMA_TPCC_SER / EDMA_TPCC_SERH / EDMA_TPCC_QSER remains set	Yes	No
Link update (STATIC = 0 in EDMA_TPCC_OPT_n)	Yes	Yes
EDMA_TPCC_QER is set	Yes	Yes
EDMA_TPCC_IPR / EDMA_TPCC_IPRH, EDMA_TPCC_CER / EDMA_TPCC_CERH is set using early completion	Yes	Yes

11.2.3.3.6 Parameter Set Updates

When a TR is submitted for a given DMA/QDMA channel and its corresponding PaRAM set, the EDMA_TPCC is responsible for updating the PaRAM set in anticipation of the next trigger event. For events that are not final, this includes address and count updates; for final events, this includes the link update.

The specific PaRAM set entries that are updated depend on the channel's synchronization type (A-synchronized or AB-synchronized) and the current state of the PaRAM set. A B-update refers to the decrementing of EDMA_TPCC_ABCNT_n[31:16] BCNT in the case of A-synchronized transfers after the submission of successive TRs. A C-update refers to the decrementing of CCNT in the case of A-synchronized transfers after BCNT TRs for EDMA_TPCC_ABCNT_n[15:0] ACNT byte transfers have submitted. For AB-synchronized transfers, a C-update refers to the decrementing of EDMA_TPCC_CCNT_n[15:0] CCNT after submission of every transfer request.

Refer to Table 11-10 for details and conditions on the parameter updates. A link update occurs when the PaRAM set is exhausted, as described in Section 11.2.3.3.7 *Linking Transfers*.

After the TR is read from the PaRAM (and is in process of being submitted to EDMA_TPTC), the following fields are updated if needed:

- A-synchronized: BCNT, CCNT, SRC, DST.
- AB-synchronized: CCNT, SRC, DST.

The following fields are not updated (except for during linking, where all fields are overwritten by the link PaRAM set):

- A-synchronized: EDMA_TPCC_ABCNT_n[15:0] ACNT, EDMA_TPCC_LNK_n[31:16] BCNTRLD, EDMA_TPCC_BIDX_n[15:0] SBIDX, EDMA_TPCC_BIDX_n[31:16] DBIDX, EDMA_TPCC_CIDX_n[15:0] SCIDX, EDMA_TPCC_CIDX_n[31:16] DCIDX, EDMA_TPCC_OPT_n, EDMA_TPCC_LNK_n[15:0]LINK.
- AB-synchronized: EDMA_TPCC_ABCNT_n[15:0] ACNT, EDMA_TPCC_ABCNT_n[31:16] BCNT, EDMA_TPCC_LNK_n[31:16] BCNTRLD, EDMA_TPCC_BIDX_n[15:0] SBIDX, EDMA_TPCC_BIDX_n[31:16] DBIDX, EDMA_TPCC_CIDX_n[15:0] SCIDX, EDMA_TPCC_CIDX_n[31:16] DCIDX, EDMA_TPCC_OPT_n, EDMA_TPCC_LNK_n[15:0]LINK.

Note

PaRAM updates only pertain to the information that is needed to properly submit the next transfer request to the EDMA_TPTC. Updates that occur while data is moved within a transfer request are tracked within the transfer controller, and is detailed in Section 11.2.3.12 *EDMA Transfer Controller (EDMA_TPTC)*. For A-synchronized transfers, the EDMA_TPCC always submits a TRP for EDMA_TPCC_ABCNT_n[15:0] ACNT bytes (EDMA_TPCC_ABCNT_n[31:16] BCNT = 1 and EDMA_TPCC_CCNT_n[15:0] CCNT = 1). For AB-synchronized transfers, the EDMA_TPCC always submits a TRP for EDMA_TPCC_ABCNT_n[15:0] ACNT bytes of BCNT arrays (EDMA_TPCC_CCNT_n[15:0] CCNT = 1). The EDMA_TPTC is responsible for updating source and destination addresses within the array based on EDMA_TPCC_ABCNT_n[15:0] ACNT and EDMA_TPCC_OPT_n[10:8] FWID. For AB-synchronized transfers, the EDMA_TPTC is also responsible to update source and destination addresses between arrays based on EDMA_TPCC_BIDX_n[15:0] SBIDX and EDMA_TPCC_BIDX_n[31:16] DBIDX.

Table 11-10 shows the details of parameter updates that occur within EDMA_TPCC for A-synchronized and AB-synchronized transfers.

Table 11-10. Parameter Updates in EDMA_TPCC (for Non-Null, Non-Dummy PaRAM Set)

	A-Synchronized Transfer			AB-Synchronized Transfer		
	B-Update	**C-Update**	**Link Update**	**B-Update**	**C-Update**	**Link Update**
Condition:	**BCNT > 1**	**BCNT == 1 && CCNT > 1**	**BCNT == 1 && CCNT == 1**	**N/A**	**EDMA_TPCC_CCNT_n[15:0] CCNT > 1**	**EDMA_TPCC_CCNT_n[15:0] CCNT == 1**
SRC	+= SBIDX	+= SCIDX	= Link.EDMA_TPCC_SRC_n	in EDMA_TPTC	+= SCIDX	= Link.EDMA_TPCC_SRC_n

Table 11-10. Parameter Updates in EDMA_TPCC (for Non-Null, Non-Dummy PaRAM Set) (continued)

	A-Synchronized Transfer			AB-Synchronized Transfer		
	B-Update	C-Update	Link Update	B-Update	C-Update	Link Update
Condition:	BCNT > 1	BCNT == 1 && CCNT > 1	BCNT == 1 && CCNT == 1	N/A	EDMA_TPCC_CCNT_n[15:0] CCNT > 1	EDMA_TPCC_CCNT_n[15:0] CCNT == 1
DST	+= DBIDX	+= DCIDX	= Link.EDMA_TPCC_DST_n	in EDMA_TPTC	+= DCIDX	= Link.EDMA_TPCC_DST_n
ACNT	None	None	= Link.EDMA_TPCC_ABCNT_n[15:0] ACNT	None	None	= Link.EDMA_TPCC_ABCNT_n[15:0] ACNT
BCNT	–= 1	= BCNTRLD	= Link.EDMA_TPCC_ABCNT_n[31:16] BCNT	in EDMA_TPTC	N/A	= Link.EDMA_TPCC_ABCNT_n[31:16] BCNT
CCNT	None	–= 1	= Link.EDMA_TPCC_CCNT_n[15:0] CCNT	in EDMA_TPTC	-=1	= Link.EDMA_TPCC_CCNT_n[15:0] CCNT
SBIDX	None	None	= Link.EDMA_TPCC_BIDX_n[15:0] SBIDX	in EDMA_TPTC	None	= Link.EDMA_TPCC_BIDX_n[15:0] SBIDX
DBIDX	None	None	= Link.EDMA_TPCC_BIDX_n[31:16] DBIDX	None	None	= Link.EDMA_TPCC_BIDX_n[31:16] DBIDX
SCIDX	None	None	= Link.EDMA_TPCC_BIDX_n[15:0] SBIDX	in EDMA_TPTC	None	= Link.EDMA_TPCC_BIDX_n[15:0] SBIDX
DCIDX	None	None	= Link.EDMA_TPCC_BIDX_n[31:16] DBIDX	None	None	= Link.EDMA_TPCC_BIDX_n[31:16] DBIDX
LINK	None	None	= Link.EDMA_TPCC_LNK_n[15:0]LINK	None	None	= Link.EDMA_TPCC_LNK_n[15:0]LINK
BCNTRLD	None	None	= Link.EDMA_TPCC_LNK_n[31:16] BCNTRLD	None	None	= Link.EDMA_TPCC_LNK_n[31:16] BCNTRLD
OPT(1)	None	None	= LINK.EDMA_TPCC_OPT_n	None	None	= LINK.EDMA_TPCC_OPT_n

(1) In all cases, no updates occur if EDMA_TPCC_OPT_n[3] STATIC == 1 for the current PaRAM set.

Note

The EDMA_TPCC includes no special hardware to detect when an indexed address update calculation overflows/underflows. The address update will wrap across boundaries as programmed by the user. It should ensure that no transfer is allowed to cross internal port boundaries between peripherals. A single TR must target a single source/destination slave endpoint.

11.2.3.3.7 Linking Transfers

The EDMA_TPCC provides a mechanism known as linking, which allows the entire PaRAM set to be reloaded from a location within the PaRAM memory map (for both DMA and QDMA channels). Linking is especially useful for maintaining ping-pong buffers, circular buffering, and repetitive/continuous transfers with no CPU intervention. Upon completion of a transfer, the current transfer parameters are reloaded with the parameter set pointed that the 16-bit link address field of the current parameter set points to. Linking only occurs when the EDMA_TPCC_OPT_n[3] STATIC bit is cleared.

Note

It should always link a transfer (EDMA or QDMA) to another useful transfer. If it must terminate a transfer, then link the transfer to a NULL parameter set. Refer to Section 11.2.3.3.3 *Null PaRAM Set*.

The link update occurs after the current PaRAM set event parameters have been exhausted. An event's parameters are exhausted when the EDMA channel controller has submitted all of the transfers that are associated with the PaRAM set.

A link update occurs for null and dummy transfers depending on the state of the EDMA_TPCC_OPT_n[3] STATIC bit and the EDMA_TPCC_LNK_n[15:0] LINK field. In both cases (null or dummy), if the value of EDMA_TPCC_LNK_n[15:0] LINK is FFFFh, then a null PaRAM set (with all 0s and EDMA_TPCC_LNK_n[15:0] LINK set to FFFFh) is written to the current PaRAM set.
Similarly, if EDMA_TPCC_LNK_n[15:0] LINK is set to a value other than FFFFh, then the appropriate PaRAM location that EDMA_TPCC_LNK_n[15:0] LINK points to is copied to the current PaRAM set.

Once the channel completion conditions are met for an event, the transfer parameters that are located at the link address are loaded into the current DMA or QDMA channel's associated parameter set. This indicates that the EDMA_TPCC reads the entire set (eight words) from the PaRAM set specified by EDMA_TPCC_LNK_n[15:0] LINK and writes all eight words to the PaRAM set that is associated with the current channel. Figure 11-10 shows an example of a linked transfer.

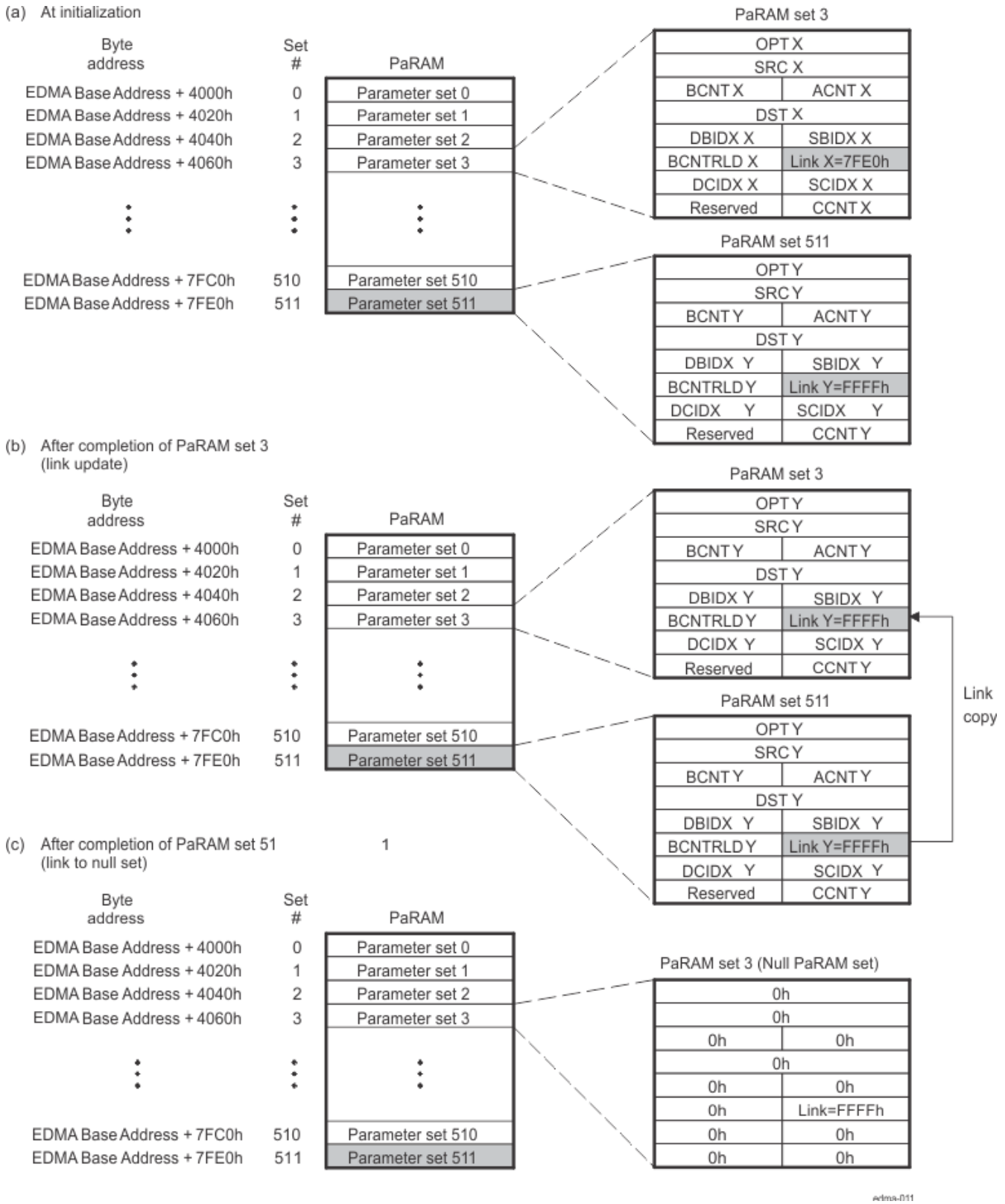


Figure 11-10. Linked Transfer

Note

AM263/Px has a maximum of 256 PaRAM sets. Additional tables and diagrams in this chapter may show a larger number (up to 511), however 256 is the maximum allowed number of entries.

Any PaRAM set in the PaRAM can be used as a link/reload parameter set. The PaRAM sets associated with peripheral synchronization events (refer to Section 11.2.3.6 *Event, Channel, and PaRAM Mapping*) only use for linking if the corresponding events are disabled.

If a PaRAM set location is defined as a QDMA channel PaRAM set (by EDMA_TPCC_QCHMAPN_j register), then copying the link PaRAM set into the current QDMA channel PaRAM set is recognized as a trigger event. It is latched in EDMA_TPCC_QER because a write to the trigger word was performed. This feature is used to create a linked list of transfers using a single QDMA channel and multiple PaRAM sets. Refer to Section 11.2.3.4.2 *QDMA Channels*.

Linking to itself replicates the behavior of auto-initialization, thus facilitating the use of circular buffering and repetitive transfers. After an EDMA channel exhausts its current PaRAM set, it reloads all of the parameter set entries from another PaRAM set, which is initialized with values that are identical to the original PaRAM set. Figure 11-11 shows an example of a linked to self transfer. Here, the PaRAM set 511 has the link field pointing to the address of parameter set 511 (linked to self). For AM263/Px devices, this would be PaRAM set 255 with the link field pointing to the address of parameter set 255 (linked to self).

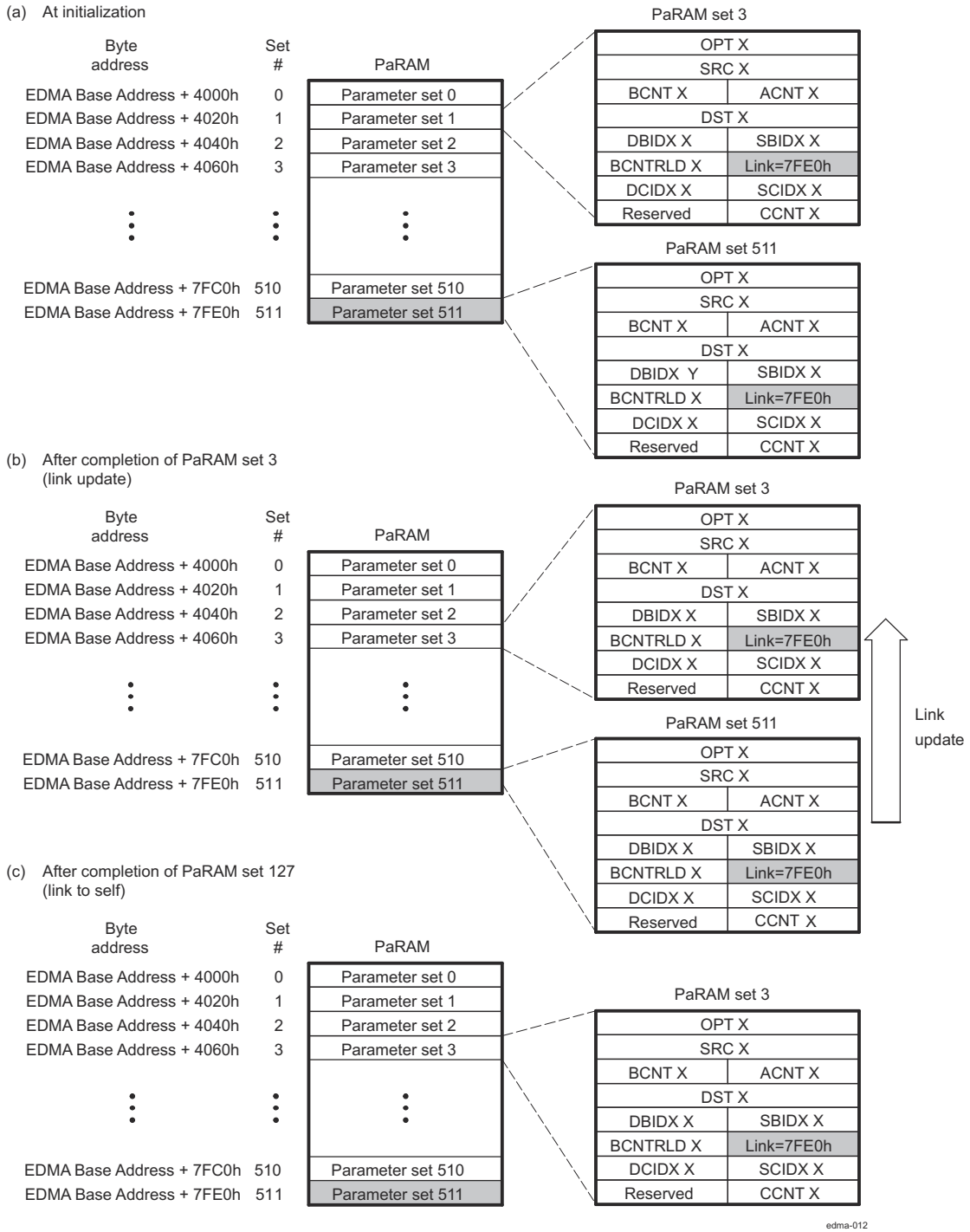


Figure 11-11. Link-to-Self Transfer

Note

If the in EDMA_TPCC_OPT_n[3] STATIC bit is set for a PaRAM set, then link updates are not performed.

11.2.3.3.8 Constant Addressing Mode Transfers/Alignment Issues

If either EDMA_TPCC_OPT_n[0] SAM or EDMA_TPCC_OPT_n[1] DAM is set (constant addressing mode), then the source or destination address must be aligned to a 256-bit aligned address, respectively, and the corresponding EDMA_TPCC_BIDX_n is an even multiple of 32 bytes (256 bits). The EDMA_TPCC does not

recognize errors here, but the EDMA_TPTC asserts an error if this is not true. Refer to Section 11.2.3.12.3 *Error Generation*.

Note

The constant addressing (CONST) mode has limited applicability. The EDMA is configured for the constant addressing mode (EDMA_TPCC_OPT_n[0] SAM / EDMA_TPCC_OPT_n[1] DAM = 1) only if the transfer source or destination (on-chip memory, off-chip memory controllers, slave peripherals) support the constant addressing mode. If the constant addressing mode is not supported, the similar logical transfer can be achieved using the increment (INCR) mode (EDMA_TPCC_OPT_n[0] SAM / EDMA_TPCC_OPT_n[1] DAM =0) by appropriately programming the count and indices values.

11.2.3.3.9 Element Size

The EDMA controller does not use element-size and element-indexing. Instead, all transfers are defined in terms of all three dimensions: EDMA_TPCC_ABCNT_n[15:0] ACNT, EDMA_TPCC_ABCNT_n[31:16] BCNT, and EDMA_TPCC_CCNT_n[15:0] CCNT. An element-indexed transfer is logically achieved by programming EDMA_TPCC_ABCNT_n[15:0] ACNT to the size of the element and EDMA_TPCC_ABCNT_n[31:16] BCNT to the number of elements that need to be transferred. For example: If there are 16-bit audio data and 256 audio samples that must be transferred to a serial port, therefore the EDMA_TPCC_ABCNT_n[15:0] ACNT = 2 (2 bytes) and EDMA_TPCC_ABCNT_n[31:16] BCNT = 256.

11.2.3.4 Initiating a DMA Transfer

There are multiple ways to initiate a programmed data transfer using the EDMA_TPCC channel controller. Transfers on DMA channels are initiated by three sources.

They are listed as follows:

- **Event-triggered transfer request** (this is the typical usage of EDMA controller): A peripheral, system, or externally-generated event triggers a transfer request.
- **Manually-triggered transfer request:**The CPU manually triggers a transfer by writing a 1 to the corresponding bit in the event set registers (EDMA_TPCC_ESR / EDMA_TPCC_ESRH).
- **Chain-triggered transfer request:** A transfer is triggered on the completion of another transfer or sub-transfer.

Transfers on QDMA channels are initiated by two sources. They are as follows:

- **Auto-triggered transfer request:** Writing to the programmed trigger word triggers a transfer.
- **Link-triggered transfer requests:** Writing to the trigger word triggers the transfer when linking occurs.

11.2.3.4.1 DMA Channels

11.2.3.4.1.1 Event-Triggered Transfer Request

When an event is asserted from a peripheral or device pins, it gets latched in the corresponding bit of the event register (EDMA_TPCC_ER[31:0] E*n* = 1). For more information about peripheral events to EDMA events mapping, refer to *the device data manual*.
If the corresponding event in the event enable register (EDMA_TPCC_EER) is enabled (EDMA_TPCC_EER[31:0] E*n* = 1), then the EDMA_TPCC prioritizes and queues the event in the appropriate event queue. When the event reaches the head of the queue, it is evaluated for submission as a transfer request to the transfer controller.

If the PaRAM set is valid (not a NULL set), then a transfer request packet (TRP) is submitted to the EDMA_TPTC and the EDMA_TPCC_ER[31:0] E*n* bit is cleared. At this point, a new event can be safely received by the EDMA_TPCC.

If the PaRAM set associated with the channel is a NULL set (see Section 11.2.3.3.3 *Null PaRAM Set*), then no transfer request (TR) is submitted and the corresponding EDMA_TPCC_ER[31:0] E*n* bit is cleared and simultaneously the corresponding channel bit is set in the event miss register (EDMA_TPCC_EMR[31:0] E*n* = 1) to indicate that the event was discarded due to a null TR being serviced. Good programming practices should include cleaning the event missed error before re-triggering the DMA channel.

When an event is received, the corresponding event bit in the event register is set (EDMA_TPCC_ER[31:0] E*n* = 1), regardless of the state of EDMA_TPCC_EER[31:0] E*n*. If the event is disabled when an external event is received (EDMA_TPCC_ER[31:0] E*n* = 1 and EDMA_TPCC_EER[31:0] E*n* = 0), the EDMA_TPCC_ER[31:0] E*n* bit remains set. If the event is subsequently enabled (EDMA_TPCC_EER[31:0] E*n* = 1), then the pending event is processed by the EDMA_TPCC and the TR is processed/submitted, after which the EDMA_TPCC_ER[31:0] E*n* bit is cleared.

If an event is being processed (prioritized or is in the event queue) and another sync event is received for the same channel prior to the original being cleared (EDMA_TPCC_ER[31:0] E*n* != 0), then the second event is registered as a missed event in the corresponding bit of the event missed register (EDMA_TPCC_EMR[31:0] E*n* = 1).

11.2.3.4.1.2 Manually-Triggered Transfer Request

The CPU or any peripheral device module initiates a DMA transfer by writing to the event set register EDMA_TPCC_ESR. Writing a 1 to an event bit in the EDMA_TPCC_ESR results in the event being prioritized/queued in the appropriate event queue, regardless of the state of the EDMA_TPCC_EER[31:0] E*n* bit. When the event reaches the head of the queue, it is evaluated for submission as a transfer request to the transfer controller.

As in the event-triggered transfers, if the PaRAM set associated with the channel is valid (it is not a null set) then the TR is submitted to the associated EDMA_TPTC and the channel can be triggered again.

If the PaRAM set associated with the channel is a NULL set (see Section 11.2.3.3.3 *Null PaRAM Set*), then no transfer request (TR) is submitted and the corresponding EDMA_TPCC_ER[31:0] E*n* bit is cleared and simultaneously the corresponding channel bit is set in the event miss register EDMA_TPCC_EMR[31:0] E*n* = 1 to indicate that the event was discarded due to a null TR being serviced. Good programming practices should include clearing the event missed error before re-triggering the DMA channel.

If an event is being processed (prioritized or is in the event queue) and the same channel is manually set by a write to the corresponding channel bit of the event set register EDMA_TPCC_ESR[31:0] E*n* = 1 prior to the original being cleared EDMA_TPCC_ESR[31:0] E*n* = 0, then the second event is registered as a missed event in the corresponding bit of the event missed register EDMA_TPCC_EMR[31:0] E*n* = 1.

11.2.3.4.1.3 Chain-Triggered Transfer Request

Chaining is a mechanism by which the completion of one transfer automatically sets the event for another channel. When a chained completion code is detected, the value of which is dictated by the transfer completion code EDMA_TPCC_OPT_n[17:12] TCC of the PaRAM set associated with the channel, it results in the corresponding bit in the chained event register EDMA_TPCC_CER to be set EDMA_TPCC_CER[31:0] E[TCC] = 1).

Once a bit is set in EDMA_TPCC_CER, the EDMA_TPCC prioritizes and queues the event in the appropriate event queue. When the event reaches the head of the queue, it is evaluated for submission as a transfer request to the transfer controller.

As in the event-triggered transfers, if the PaRAM set associated with the channel is valid (it is not a null set) then the TR is submitted to the associated EDMA_TPTC and the channel can be triggered again.

If the PaRAM set associated with the channel is a NULL set (see Section 11.2.3.3.3 *Null PaRAM Set*), then no transfer request (TR) is submitted and the corresponding EDMA_TPCC_CER[31:0] E*n* bit is cleared and simultaneously the corresponding channel bit is set in the event miss register EDMA_TPCC_EMR[31:0] E*n* = 1 to indicate that the event was discarded due to a null TR being serviced. In this case, the error condition must be cleared before the DMA channel can be re-triggered. Good programming practices might include clearing the event missed error before re-triggering the DMA channel.

If a chaining event is being processed (prioritized or queued) and another chained event is received for the same channel prior to the original being cleared EDMA_TPCC_CER[31:0] E*n* != 0), then the second chained event is registered as a missed event in the corresponding channel bit of the event missed register EDMA_TPCC_EMR[31:0] E*n* = 1.

Note

Chained event registers EDMA_TPCC_CER, event registers EDMA_TPCC_ER, and event set registers EDMA_TPCC_ESR operate independently. An event E*n* can be triggered by any of the trigger sources (event-triggered, manually-triggered, or chain-triggered).

11.2.3.4.2 QDMA Channels

11.2.3.4.2.1 Auto-Triggered and Link-Triggered Transfer Request

QDMA-based transfer requests are issued when a QDMA event gets latched in the QDMA event register EDMA_TPCC_QER[31:0] E*n* = 1. A bit corresponding to a QDMA channel is set in the QDMA event register EDMA_TPCC_QER when the following occurs:

- A CPU (or any device module) write occurs to a PaRAM address that is defined as a QDMA channel trigger word (programmed in the QDMA channel mapping register EDMA_TPCC_QCHMAPN_j for the particular QDMA channel and the QDMA channel is enabled via the QDMA event enable register EDMA_TPCC_QEER[31:0] E*n* = 1.
- EDMA_TPCC performs a link update on a PaRAM set address that is configured as a QDMA channel matches EDMA_TPCC_QCHMAPN_j settings and the corresponding channel is enabled via the QDMA event enable register EDMA_TPCC_QEER[31:0] E*n* = 1.

Once a bit is set in EDMA_TPCC_QER, the EDMA_TPCC prioritizes and queues the event in the appropriate event queue. When the event reaches the head of the queue, it is evaluated for submission as a transfer request to the transfer controller.

As in the event-triggered transfers, if the PaRAM set associated with the channel is valid (it is not a null set) then the TR is submitted to the associated EDMA_TPTC and the channel can be triggered again.

If a bit is already set in EDMA_TPCC_QER[31:0] E*n* = 1 and a second QDMA event for the same QDMA channel occurs prior to the original being cleared, the second QDMA event gets captured in the QDMA event miss register EDMA_TPCC_QEMR[7:0] E*n* = 1.

11.2.3.4.3 Comparison Between DMA and QDMA Channels

The primary difference between DMA and QDMA channels is the event/channel synchronization.

QDMA events are either auto-triggered or link triggered. Auto-triggering allows QDMA channels to be triggered by CPU(s) with a minimum number of linear writes to PaRAM. Link triggering allows a linked list of transfers to be executed, using a single QDMA PaRAM set and multiple link PaRAM sets.

A QDMA transfer is triggered when a CPU (or other device modules) writes to the trigger word of the QDMA channel parameter set (auto-triggered) or when the EDMA_TPCC performs a link update on a PaRAM set that has been mapped to a QDMA channel (link triggered).

Note

The CPUs triggered (manually triggered) DMA channels, in addition to writing to the PaRAM set, it is required to write to the event set register EDMA_TPCC_ESR to kick-off the transfer.

QDMA channels are typically for cases where a single event accomplishes a complete transfer since the CPU (or other device modules) must reprogram some portion of the QDMA PaRAM set in order to re-trigger the channel. QDMA transfers are programmed with EDMA_TPCC_ABCNT_n[31:0] BCNT =1 and EDMA_TPCC_CCNT_n[15:0] CCNT = 1 for A-synchronized transfers, and EDMA_TPCC_CCNT_n[15:0] CCNT = 1 for AB-synchronized transfers.

Additionally, since linking is also supported (if EDMA_TPCC_OPT_n[3] STATIC = 0) for QDMA transfers, it allows to initiate a linked list of QDMAs, so when EDMA_TPCC copies over a link PaRAM set (including the write to the trigger word), the current PaRAM set mapped to the QDMA channel automatically recognizes as a valid QDMA event and initiate another set of transfers as specified by the linked set.

11.2.3.5 Completion of a DMA Transfer

A parameter set for a given channel is complete when the required number of transfer requests is submitted (based on receiving the number of synchronization events). The expected number of TRs for a non-null/non-dummy transfer is shown in Table 11-11 for both synchronization types along with state of the PaRAM set prior to the final TR being submitted. When the counts (EDMA_TPCC_ABCNT_n[31:0] BCNT and/or EDMA_TPCC_CCNT_n[15:0] CCNT) are this value, the next TR results in:

- Final chaining or interrupt codes sent by the transfer controllers (instead of intermediate).
- Link updates (linking to either null or another valid link set).

Table 11-11. Expected Number of Transfers for Non-Null Transfer

Sync Mode	Counts at time 0	Total # Transfers	Counts prior to final TR
A-synchronized	ACNT BCNT CCNT	(BCNT × CCNT) TRs of ACNT bytes each	EDMA_TPCC_ABCNT_n[31:0] BCNT == 1 && EDMA_TPCC_CCNT_n[15:0] CCNT == 1
AB-synchronized	ACNT BCNT CCNT	CCNT TRs for ACNT × BCNT bytes each	EDMA_TPCC_CCNT_n[15:0] CCNT == 1

The PaRAM OPT field must program with a specific transfer completion code TCC or EDMA_TPCC_OPT_n[17:12] TCC along with the other EDMA_TPCC_OPT_n fields ([22] TCCHEN, [20] TCINTEN, [23] ITCCHEN, and [21] ITCINTEN bits) to indicate whether the completion code is to be used for generating a chained event or/and for generating an interrupt upon completion of a transfer.

The specific EDMA_TPCC_OPT_n[17:12] TCC value (6-bit binary value) programmed dictates which of the 64-bits in the chain event register EDMA_TPCC_CER [TCC] and/or interrupt pending register EDMA_TPCC_IPR [TCC] is set.

It can selectively program whether the transfer controller sends back completion codes on completion of the final transfer request (TR) of a parameter set EDMA_TPCC_OPT_n[22] TCCHEN or EDMA_TPCC_OPT_n[20] TCINTEN, for all but the final transfer request (TR) of a parameter set EDMA_TPCC_OPT_n[23] ITCCHEN or EDMA_TPCC_OPT_n[21] ITCINTEN), or for all TRs of a parameter set (both). Refer to Section 11.2.3.8 *Chaining EDMA Channels* for details on chaining (intermediate/final chaining) and Section 11.2.3.9 *EDMA Interrupts* for details on intermediate/final interrupt completion.

A completion detection interface exists between the EDMA channel controller and transfer controller(s). This interface sends back information from the transfer controller to the channel controller to indicate that a specific transfer is completed. Completion of a transfer is used for generating chained events and/or generating interrupts to the CPU(s).

All DMA/QDMA PaRAM sets must also specify a link address value. For repetitive transfers such as ping-pong buffers, the link address value must point to another predefined PaRAM set. Alternatively, a non-repetitive transfer must set the link address value to the null link value. The null link value is defined as FFFFh. Refer to Section 11.2.3.3.7 *Linking Transfers* for more details.

Note

Any incoming events that are mapped to a null PaRAM set results in an error condition. The error condition must clear before the corresponding channel is used again. Refer to Section 11.2.3.3.5 *Dummy Versus Null Transfer Comparison*.

There are three ways the EDMA_TPCC gets updated/informed about a transfer completion: normal completion, early completion, and dummy/null completion. This applies to both chained events and completion interrupt generation.

11.2.3.5.1 Normal Completion

In normal completion mode EDMA_TPCC_OPT_n[11] TCCMODE = 0, the transfer or sub-transfer is considered to be complete when the EDMA channel controller receives the completion codes from the EDMA transfer

controller. In this mode, the completion code to the channel controller is posted by the transfer controller after it receives a signal from the destination peripheral. Normal completion is typically used to generate an interrupt to inform the CPU that a set of data is ready for processing.

11.2.3.5.2 Early Completion

In early completion mode EDMA_TPCC_OPT_n[11] TCCMODE = 1, the transfer is considered to be complete when the EDMA channel controller submits the transfer request (TR) to the EDMA transfer controller. In this mode, the channel controller generates the completion code internally. Early completion is typically useful for chaining, as it allows subsequent transfers to be chained-triggered while the previous transfer is still in progress within the transfer controller, maximizing the overall throughput of the set of the transfers.

11.2.3.5.3 Dummy or Null Completion

This is a variation of early completion. Dummy or null completion is associated with a dummy set Section 11.2.3.3.4 or null set Section 11.2.3.3.3. In both cases, the EDMA channel controller does not submit the associated transfer request to the EDMA transfer controller(s). However, if the set (dummy/null) has the OPT field programmed to return completion code (intermediate/final interrupt/chaining completion), then it sets the appropriate bits in the interrupt pending registers EDMA_TPCC_IPR and EDMA_TPCC_IPRH or chained event register EDMA_TPCC_CER and EDMA_TPCC_CERH. The internal early completion path is used by the channel controller to return the completion codes internally (that is, EDMA_TPCC generates the completion code).

11.2.3.6 Event, Channel, and PaRAM Mapping

Several of the 64 DMA channels are tied to a specific hardware event, thus allowing events from device peripherals or external hardware (via the dma_evt[3:0] pins) to trigger transfers. A DMA channel typically requests a data transfer when it receives its event (apart from manually-triggered, chain-triggered, and other transfers). The amount of data transferred per synchronization event depends on the channel's configuration (EDMA_TPCC_ABCNT_n[15:0] ACNT, EDMA_TPCC_ABCNT_n[31:16] BCNT, EDMA_TPCC_CCNT_n[15:0] CCNT, etc.) and the synchronization type (A-synchronized or AB-synchronized).

The association of an event to a channel is fixed within the EDMA Channel Controller, that is, each DMA channel has one specific event associated with it.

In an application, if a channel does not use the associated synchronization event or if it does not have an associated synchronization event (unused), that channel can be used for manually-triggered or chained-triggered transfers, for linking/reloading, or as a QDMA channel.

11.2.3.6.1 DMA Channel to PaRAM Mapping

The mapping between the DMA channel numbers and the PaRAM sets is programmable (see Section 11.2.3.3). The DMA channel mapping registers EDMA_TPCC_DCHMAPN_m in the EDMA_TPCC provide programmability that allows the DMA channels to be mapped to any of the PaRAM sets in the PaRAM memory map. Figure 11-12 illustrates the use of EDMA_TPCC_DCHMAPN_m. There is one EDMA_TPCC_DCHMAPN_m register per channel.

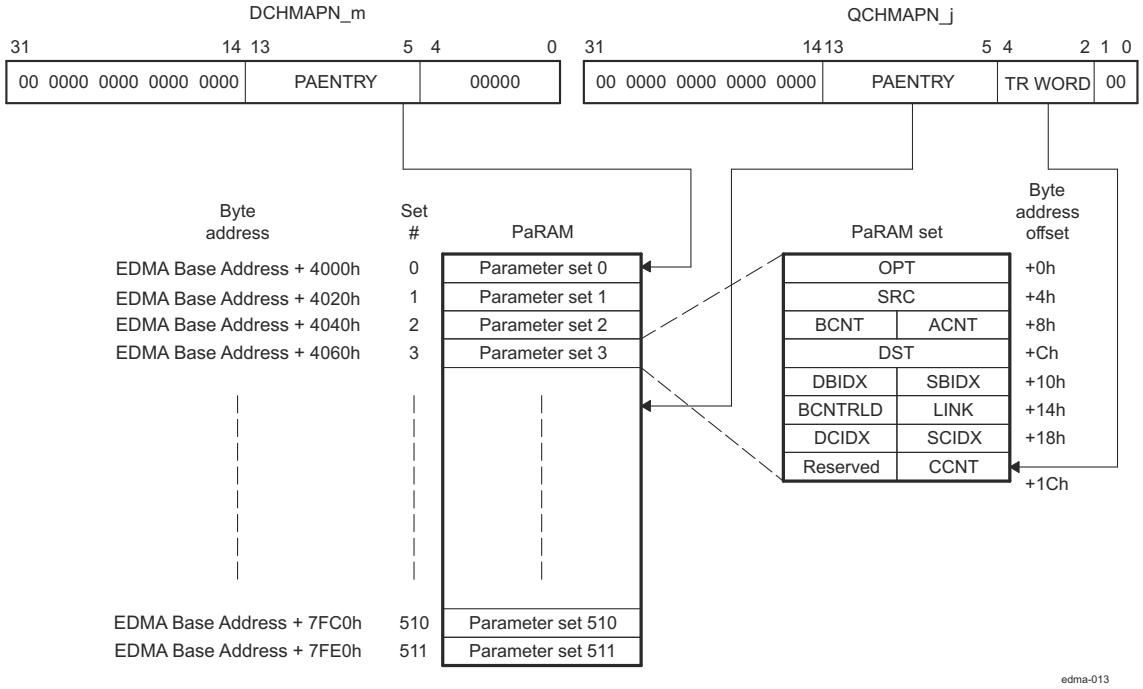


Figure 11-12. DMA Channel and QDMA Channel to PaRAM Mapping

Note

11.2.3.6.2 QDMA Channel to PaRAM Mapping

The mapping between the QDMA channels and the PaRAM sets is programmable. The QDMA channel mapping register EDMA_TPCC_QCHMAPN_j in the EDMA_TPCC allows to map the QDMA channels to any of the PaRAM sets in the PaRAM memory map. Figure 11-13 illustrates the use of EDMA_TPCC_QCHMAPN_j.

EDMA_TPCC_QCHMAPN_j[4:2] TRWORD bit-field allows to program the trigger word in the PaRAM set for the QDMA channel. A trigger word is one of the eight words in the PaRAM set. For a QDMA transfer to occur, a valid TR synchronization event for EDMA_TPCC is a write to the trigger word in the PaRAM set pointed to by EDMA_TPCC_QCHMAPN_j for a particular QDMA channel. By default, QDMA channels are mapped to PaRAM set 0.

It must appropriately re-map PaRAM set 0 before use.

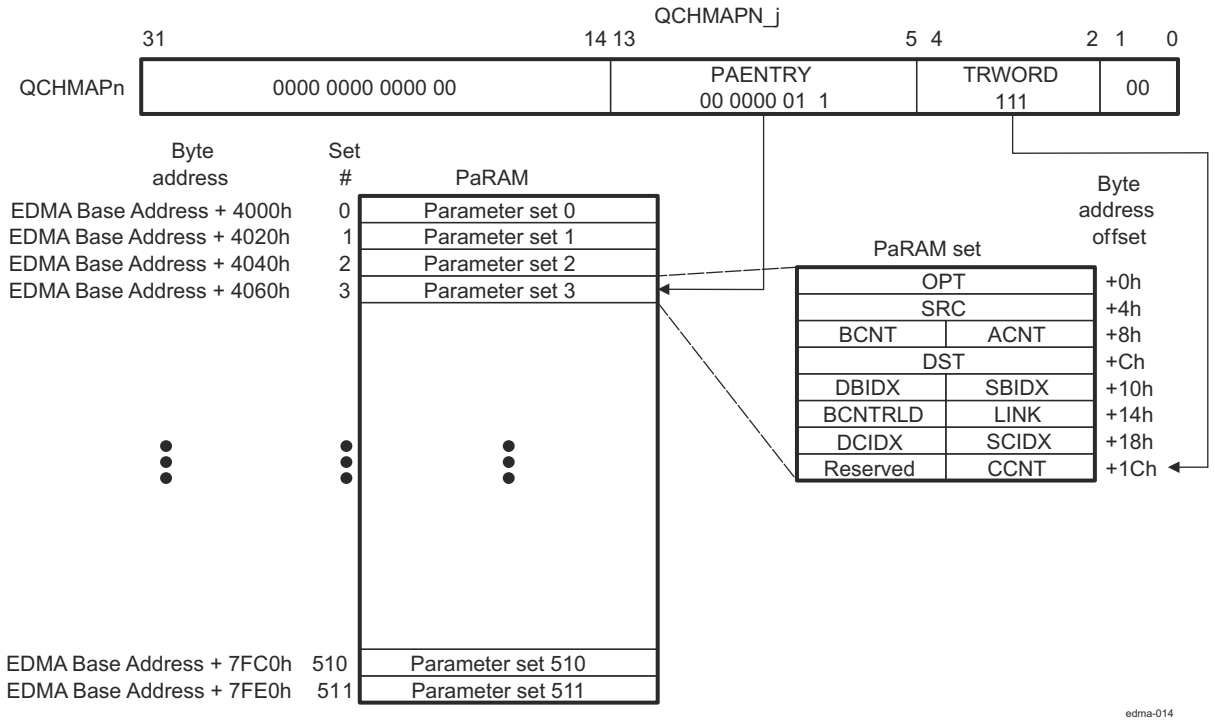


Figure 11-13. QDMA Channel to PaRAM Mapping

11.2.3.7 EDMA Channel Controller Regions

The EDMA channel controller divides its address space into eight regions. Individual channel resources are assigned to a specific region, where each region is typically assigned to a specific device module uses the EDMA controller.

Application software can use regions or to ignore them altogether. It can be used active memory protection in conjunction with regions so that only a specific device module which uses the EDMA (for example, privilege identification) or privilege level (for example, user vs. supervisor) is allowed access to a given region, and thus to a given DMA or QDMA channel. This allows robust system-level DMA code where each EDMA initiator only modifies the state of the assigned resources. Memory protection is described in Section 11.2.3.10 *Memory Protection*.

11.2.3.7.1 Region Overview

The EDMA channel controller memory-mapped registers are divided in three main categories:

1. Global registers
2. Global region channel registers
3. Shadow region channel registers

The global registers are located at a single/fixed location in the EDMA_TPCC memory map. These registers control EDMA resource mapping and provide debug visibility and error tracking information.

The channel registers (including DMA, QDMA, and interrupt registers) are accessible via the global channel region address range, or in the shadow *n* channel region address range(s). For example, the event enable register EDMA_TPCC_EER is visible at the global address of EDMA Base Address + 1020h or region addresses of EDMA Base Address + 2020h for region 0, EDMA Base Address + 2220h for region 1, … EDMA Base Address + 2E20h for region 7.

The DMA region access enable registers EDMA_TPCC_DRAEM_k and the QDMA region access enable registers EDMA_TPCC_QRAEN_k control the underlying control register bits that are accessible via the shadow region address space (except for EDMA_TPCC_IEVAL and EDMA_TPCC_IEVAL_RN_k registers). Table 11-12 lists the registers in the shadow region memory map. Refer to *EDMA_TPCC register mapping summary* for the complete global and shadow region memory maps.

Table 11-12. Shadow Region Registers

EDMA_TPCC_DRAEM_k	EDMA_TPCC_DRAEHM_k	EDMA_TPCC_QRAEN_k
EDMA_TPCC_ER	EDMA_TPCC_ERH	EDMA_TPCC_QER
EDMA_TPCC_ECR	EDMA_TPCC_ECRH	EDMA_TPCC_QEER
EDMA_TPCC_ESR	EDMA_TPCC_ESRH	EDMA_TPCC_QEECR
EDMA_TPCC_CER	EDMA_TPCC_CERH	EDMA_TPCC_QEESR
EDMA_TPCC_EER	EDMA_TPCC_EERH	
EDMA_TPCC_EECR	EDMA_TPCC_EECRH	
EDMA_TPCC_EESR	EDMA_TPCC_EESRH	
EDMA_TPCC_SER	EDMA_TPCC_SERH	
EDMA_TPCC_SECR	EDMA_TPCC_SECRH	
EDMA_TPCC_IER	EDMA_TPCC_IERH	
EDMA_TPCC_IECR	EDMA_TPCC_IECRH	
EDMA_TPCC_IESR	EDMA_TPCC_IESRH	
EDMA_TPCC_IPR	EDMA_TPCC_IPRH	
EDMA_TPCC_ICR	EDMA_TPCC_ICRH	
Register not affected by DRAE\DRAEH		
EDMA_TPCC_IEVAL		
EDMA_TPCC_IEVAL_RN_k		

Figure 11-14 illustrates the conceptual view of the regions.

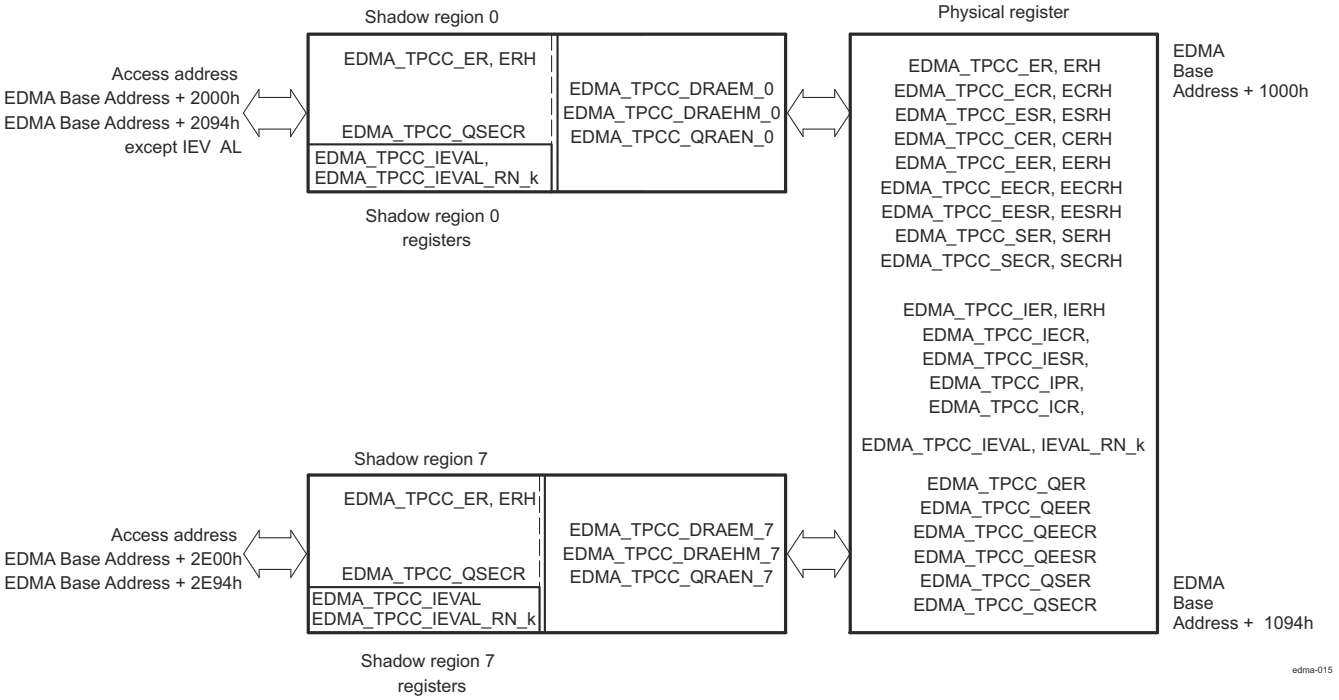


Figure 11-14. Shadow Region Registers

11.2.3.7.2 Channel Controller Regions

There are eight EDMA shadow regions (and associated memory maps). Associated with each shadow region are a set of registers defining which channels and interrupt completion codes belong to that region. These registers are user-programmed per region to assign ownership of the DMA/QDMA channels to a region.

- EDMA_TPCC_DRAEM_k and EDMA_TPCC_DRAEHM_k: One register pair exists for each of the shadow regions. The number of bits in each register pair matches the number of DMA channels (64 DMA channels). These registers need to be programmed to assign ownership of DMA channels and interrupt (or EDMA_TPCC_OPT_n[17:12] TCC codes) to the respective region. Accesses to DMA and interrupt registers via the shadow region address view are filtered through the DRAEM/DRAEHM pair. A value of 1 in the corresponding EDMA_TPCC_DRAEM_k[31:0] / EDMA_TPCC_DRAEHM_k[31:0] bit implies that the corresponding DMA interrupt channel is accessible; a value of 0 in the corresponding EDMA_TPCC_DRAEM_k[31:0] / EDMA_TPCC_DRAEHM_k[31:0] bit forces writes to be discarded and returns a value of 0 for reads.
- EDMA_TPCC_QRAEN_k: One register exists for every region. The number of bits in each register matches the number of QDMA channels (8 QDMA channels). These registers must be programmed to assign ownership of QDMA channels to the respective region. To enable a channel in a shadow region using shadow region 0 EDMA_TPCC_QEER, the corresponding bits in QRAE must be set or writing into EDMA_TPCC_QEESR there will be no the desired effect.
- EDMA_TPCC_MPPAN_k and EDMA_TPCC_MPPAG: One register exists for every region. This register defines the privilege level, requestor, and types of accesses allowed to a region's memory-mapped registers.

It is typical for an application to have a unique assignment of QDMA/DMA channels (and, therefore, a given bit position) to a given region.

The use of shadow regions allows restricted access to EDMA resources (DMA channels, QDMA channels, TCC, interrupts) by tasks in a system by setting or clearing bits in the EDMA_TPCC_DRAEM_k / EDMA_TPCC_QRAEN_k registers.

If exclusive access to any given channel / TCC code is required for a region, then only that region's EDMA_TPCC_DRAEM_k / EDMA_TPCC_QRAEN_k have the associated bit set.

Example 11-1. Resource Pool Division Across Two Regions

This example illustrates a resource pool division across two regions, assuming region 0 must be allocated 16 DMA channels (0-15) and 1 QDMA channel (0) and 32 TCC codes (0-15 and 48-63).

Region 1 needs to be allocated 16 DMA channels (16-32) and the remaining 7 QDMA channels (1-7) and TCC codes (16-47).

EDMA_TPCC_DRAEM_k should be equal to the OR of the bits that are required for the DMA channels and the TCC codes:

```
Region 0: DRAEHM, DRAEM = 0xFFFF0000, 0x0000FFFF QRAEN = 0x0000001
Region 1: DRAEHM, DRAEM = 0x0000FFFF, 0xFFFF0000 QRAEN = 0x00000FE
```

11.2.3.7.3 Region Interrupts

In addition to the EDMA_TPCC global completion interrupt, there is an additional completion interrupt line that is associated with every shadow region. Along with the interrupt enable register EDMA_TPCC_IER, DRAEM acts as a secondary interrupt enable for the respective shadow region interrupts. Refer to *Hardware Request* for more information about EDMA Interrupts.

11.2.3.8 Chaining EDMA Channels

The channel chaining capability for the EDMA allows the completion of an EDMA channel transfer to trigger another EDMA channel transfer. The purpose is to allow the ability to chain several events through one event occurrence.

Chaining is different from linking (Section 11.2.3.3.7 *Linking Transfers*). The EDMA link feature reloads the current channel parameter set with the linked parameter set. The EDMA chaining feature does not modify or

update any channel parameter set. It provides a synchronization event to the chained channel (see Section 11.2.3.4.1.3 *Chain-Triggered Transfer Request*).

Chaining is achieved at either final transfer completion or intermediate transfer completion, or both, of the current channel. Consider a channel *m* (DMA/QDMA) required to chain to channel *n*. Channel number *n* (0-63) needs to be programmed into the EDMA_TPCC_OPT_n[17:12] TCC bit-field of channel *m* channel options parameter (OPT) set.

- If final transfer completion chaining EDMA_TPCC_OPT_n[22] TCCHEN = 1 is enabled, the chain-triggered event occurs after the submission of the last transfer request of channel *m* is either submitted or completed (depending on early or normal completion).

- If intermediate transfer completion chaining EDMA_TPCC_OPT_n[23] ITCCHEN = 1 is enabled, the chain-triggered event occurs after every transfer request, except the last of channel *m* is either submitted or completed (depending on early or normal completion).
- If both final and intermediate transfer completion chaining (EDMA_TPCC_OPT_n[22] TCCHEN = 1 and EDMA_TPCC_OPT_n[23] ITCCHEN = 1) are enabled, then the chain-trigger event occurs after every transfer request is submitted or completed (depending on early or normal completion).

Table 11-13 illustrates the number of chain event triggers occurring in different synchronized scenarios. Consider channel 31 programmed with EDMA_TPCC_ABCNT_n[15:0] ACNT = 3, EDMA_TPCC_ABCNT_n[31:16] BCNT = 4, EDMA_TPCC_CCNT_n[15:0] CCNT = 5, and EDMA_TPCC_OPT_n[17:12] TCC = 30.

Table 11-13. Chain Event Triggers

	(Number of chained event triggers on channel 30)	
Options	**A-Synchronized**	**AB-Synchronized**
EDMA_TPCC_OPT_n[22] TCCHEN = 1, EDMA_TPCC_OPT_n[23] ITCCHEN = 0	1 (Owing to the last TR)	1 (Owing to the last TR)
EDMA_TPCC_OPT_n[22] TCCHEN = 0, EDMA_TPCC_OPT_n[23] ITCCHEN = 1	19 (Owing to all but the last TR)	4 (Owing to all but the last TR)
EDMA_TPCC_OPT_n[22] TCCHEN = 1, EDMA_TPCC_OPT_n[23] ITCCHEN = 1	20 (Owing to a total of 20 TRs)	5 (Owing to a total of 5 TRs)

11.2.3.9 EDMA Interrupts

The EDMA interrupts are divided into 2 categories: transfer completion interrupts and error interrupts.

There are nine region interrupts, eight shadow regions and one global region. The transfer completion interrupts are listed in Table 11-14. The transfer completion interrupts and the error interrupts from the transfer controllers are all routed to the device interrupt controllers INTCs.

Table 11-14. EDMA Transfer Completion Interrupts

Name	Description
EDMA_TPCC_INT0	EDMA_TPCC Transfer Completion Interrupt Shadow Region 0
EDMA_TPCC_INT1	EDMA_TPCC Transfer Completion Interrupt Shadow Region 1
EDMA_TPCC_INT2	EDMA_TPCC Transfer Completion Interrupt Shadow Region 2
EDMA_TPCC_INT3	EDMA_TPCC Transfer Completion Interrupt Shadow Region 3
EDMA_TPCC_INT4	EDMA_TPCC Transfer Completion Interrupt Shadow Region 4
EDMA_TPCC_INT5	EDMA_TPCC Transfer Completion Interrupt Shadow Region 5
EDMA_TPCC_INT6	EDMA_TPCC Transfer Completion Interrupt Shadow Region 6
EDMA_TPCC_INT7	EDMA_TPCC Transfer Completion Interrupt Shadow Region 7

Table 11-15. EDMA Error Interrupts

Name	Description
EDMA_TPCC_ERRINT	EDMA_TPCC Error Interrupt
EDMA_TPCC_MPINT	EDMA_TPCC Memory Protection Interrupt
EDMA_TC0_ERRINT	TC0 Error Interrupt
EDMA_TC1_ERRINT	TC1 Error Interrupt

11.2.3.9.1 Transfer Completion Interrupts

The EDMA_TPCC is responsible for generating transfer completion interrupts to the CPU(s) (and other EDMA controllers). The EDMA generates a single completion interrupt per shadow region, as well as one for the global region on behalf of all 64 channels. The various control registers and bit fields facilitate EDMA interrupt generation.

The software architecture must either use the global interrupt or the shadow interrupts, but not both.

The transfer completion code EDMA_TPCC_OPT_n[17:12] TCC value is directly mapped to the bits of the interrupt pending register EDMA_TPCC_IPR / EDMA_TPCC_IPRH.

For example, if EDMA_TPCC_OPT_n[17:12] TCC = 10 0001b, EDMA_TPCC_IPRH[1] is set after transfer completion, and results in interrupt generation to the CPU(s) if the completion interrupt is enabled for the CPU. See Section 11.2.3.9.1.1 *Enabling Transfer Completion Interrupts* for details about enabling EDMA transfer completion interrupts.

When a completion code is returned (as a result of early or normal completions), the corresponding bit in EDMA_TPCC_IPR / EDMA_TPCC_IPRH registers is set if transfer completion interrupt (final/intermediate) is enabled in the channel options parameter (OPT) for a PaRAM set associated with the transfer.

Table 11-16. Transfer Complete Code (TCC) to EDMA_TPCC Interrupt Mapping

TCC values in EDMA_TPCC_OPT_n[17:12] TCC (EDMA_TPCC_OPT_n[20] TCINTEN / EDMA_TPCC_OPT_n[21] ITCINTEN = 1)	EDMA_TPCC_IPR Bit Set	TCC values in EDMA_TPCC_OPT_n[17:12] TCC (EDMA_TPCC_OPT_n[20] TCINTEN / EDMA_TPCC_OPT_n[21] ITCINTEN = 1)	EDMA_TPCC_IPRH Bit Set(1)
0	EDMA_TPCC_IPR[0]	20h	EDMA_TPCC_IPR[32] / EDMA_TPCC_IPRH[0]
1	EDMA_TPCC_IPR[1]	21h	EDMA_TPCC_IPR[33] / EDMA_TPCC_IPRH[1]
2h	EDMA_TPCC_IPR[2]	22h	EDMA_TPCC_IPR[34] / EDMA_TPCC_IPRH[2]
3h	EDMA_TPCC_IPR[3]	23h	EDMA_TPCC_IPR[35] / EDMA_TPCC_IPRH[3]
4h	EDMA_TPCC_IPR[4]	24h	EDMA_TPCC_IPR[36] / EDMA_TPCC_IPRH[4]
...	...	...	...
1Eh	EDMA_TPCC_IPR[30]	3Eh	EDMA_TPCC_IPR[62] / EDMA_TPCC_IPRH[30]
1Fh	EDMA_TPCC_IPR[31]	3Fh	EDMA_TPCC_IPR[63] / EDMA_TPCC_IPRH[31]

(1) Bit fields EDMA_TPCC_IPR [32-63] correspond to bits 0 to 31 in EDMA_TPCC_IPRH, respectively.

The transfer completion code (TCC) can program to any value for a DMA/QDMA channel. A direct relation between the channel number and the transfer completion code value does not need to exist. This allows multiple channels having the same transfer completion code value to cause a CPU to execute the same interrupt service routine (ISR) for different channels.

If the channel is used in the context of a shadow region and it intends for the shadow region interrupt to be asserted, then ensure that the bit corresponding to the TCC code is enabled in EDMA_TPCC_IER / EDMA_TPCC_IERH and in the corresponding shadow region's DMA region access registers (EDMA_TPCC_DRAEM_k / EDMA_TPCC_DRAEHM_k).

Interrupt generation can be enabled at either final transfer completion or intermediate transfer completion, or both. Consider channel *m* as an example.

- If the final transfer interrupt (EDMA_TPCC_OPT_n[20] TCINTEN = 1) is enabled, the interrupt occurs after the last transfer request of channel *m* is either submitted or completed (depending on early or normal completion).
- If the intermediate transfer interrupt (EDMA_TPCC_OPT_n[21] ITCINTEN = 1) is enabled, the interrupt occurs after every transfer request, except the last TR of channel *m* is either submitted or completed (depending on early or normal completion).
- If both final and intermediate transfer completion interrupts (EDMA_TPCC_OPT_n[20] TCINTEN = 1, and EDMA_TPCC_OPT_n[21] ITCINTEN = 1) are enabled, then the interrupt occurs after every transfer request is submitted or completed (depending on early or normal completion).

Table 11-17 shows the number of interrupts that occur in different synchronized scenarios. Consider channel 31, programmed with ABCNT_n[15:0] ACNT = 3, EDMA_TPCC_ABCNT_n[31:16] BCNT = 4, EDMA_TPCC_CCNT_n[15:0]CCNT = 5, and EDMA_TPCC_OPT_n[17:12] TCC = 30.

Table 11-17. Number of Interrupts

Options	A-Synchronized	AB-Synchronized
EDMA_TPCC_OPT_n[20] TCINTEN = 1, EDMA_TPCC_OPT_n[21] ITCINTEN = 0	1 (Last TR)	1 (Last TR)
EDMA_TPCC_OPT_n[20] TCINTEN = 0, EDMA_TPCC_OPT_n[21] ITCINTEN = 1	19 (All but the last TR)	4 (All but the last TR)
EDMA_TPCC_OPT_n[20] TCINTEN = 1, EDMA_TPCC_OPT_n[21] ITCINTEN = 1	20 (All TRs)	5 (All TRs)

11.2.3.9.1.1 Enabling Transfer Completion Interrupts

For the EDMA channel controller to assert a transfer completion to the external environment, the interrupts must be enabled in the EDMA_TPCC. This is in addition to setting up the EDMA_TPCC_OPT_n[20] TCINTEN and EDMA_TPCC_OPT_n[21] ITCINTEN bits of the associated PaRAM set.

The EDMA channel controller has interrupt enable registers EDMA_TPCC_IER / EDMA_TPCC_IERH and each bit location in EDMA_TPCC_IER / EDMA_TPCC_IERH serves as a primary enable for the corresponding interrupt pending registers EDMA_TPCC_IPR / EDMA_TPCC_IPRH.

All of the interrupt registers (EDMA_TPCC_IER, EDMA_TPCC_IESR, EDMA_TPCC_IECR, and EDMA_TPCC_IPR) are either manipulated from the global DMA channel region, or by the DMA channel shadow regions. The shadow regions provide a view to the same set of physical registers that are in the global region.

The EDMA channel controller has a hierarchical completion interrupt scheme that uses a single set of interrupt pending registers EDMA_TPCC_IPR / EDMA_TPCC_IPRH and single set of interrupt enable registers EDMA_TPCC_IER / EDMA_TPCC_IERH. The programmable DMA region access enable registers EDMA_TPCC_DRAEM_k / EDMA_TPCC_DRAEHM_k provides a second level of interrupt masking. The global region interrupt output is gated based on the enable mask that is provided by EDMA_TPCC_IER / EDMA_TPCC_IERH, see Figure 11-15

The region interrupt outputs are gated by EDMA_TPCC_IER and the specific EDMA_TPCC_DRAEM_k / EDMA_TPCC_DRAEHM_k associated with the region.

Figure 11-15 shows the Interrupt diagram of the EDMA controller.

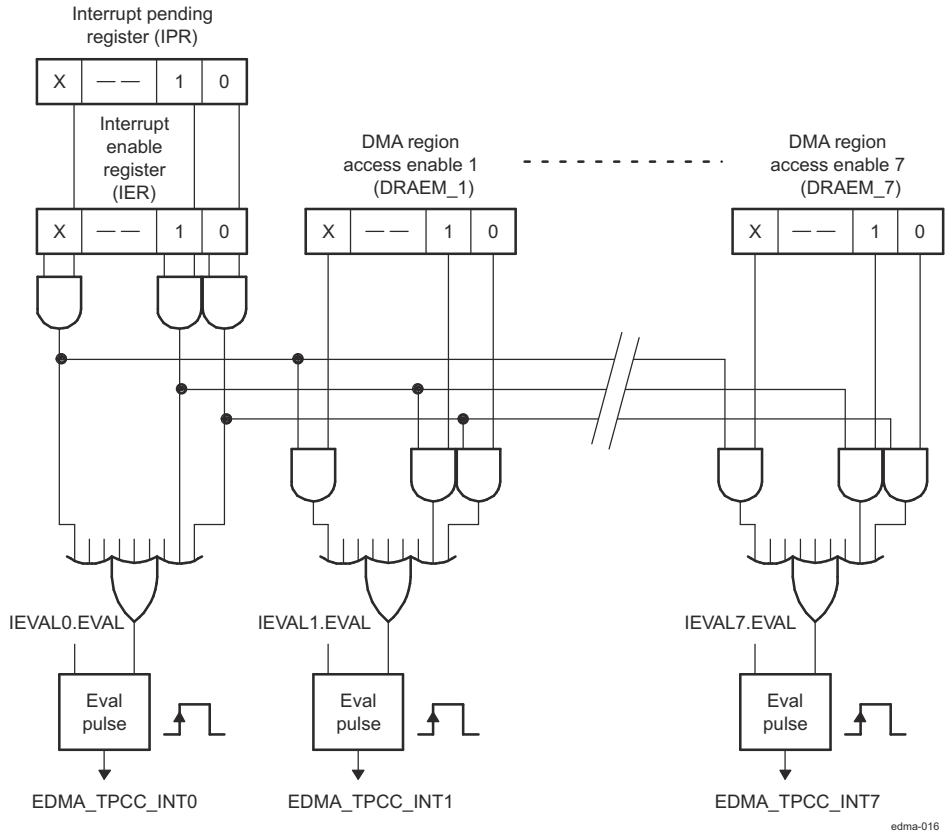


Figure 11-15. Interrupt Diagram

The EDMA_TPCC generates the transfer completion interrupts that are associated with each shadow region, the following conditions must be true:

- EDMA_TPCC_INT0: (EDMA_TPCC_IPR[0] E0 & EDMA_TPCC_IER[0] E0 & EDMA_TPCC_DRAEM_k.DRAEM_0[0] E0) | (EDMA_TPCC_IPR[1] E1 & EDMA_TPCC_IER[1] E1 & EDMA_TPCC_DRAEM_k.DRAEM_0[1] E1) | …|(EDMA_TPCC_IPRH[31] E63 & EDMA_TPCC_IERH[31] E63 & EDMA_TPCC_DRAEHM_k.DRAEHM_0[31] E63)
- EDMA_TPCC_INT1: (EDMA_TPCC_IPR[0] E0 & EDMA_TPCC_IER[0] E0 & EDMA_TPCC_DRAEM_k.DRAEM_1[0] E0) | (EDMA_TPCC_IPR[1] E1 & EDMA_TPCC_IER[1] E1 & EDMA_TPCC_DRAEM_k.DRAEM_1[1] E1) | …| (EDMA_TPCC_IPRH[31] E63 & EDMA_TPCC_IERH[31] E63 & EDMA_TPCC_DRAEHM_k.DRAEHM_1[31] E63)
- EDMA_TPCC_INT2: (EDMA_TPCC_IPR[0] E0 & EDMA_TPCC_IER[0] E0 & EDMA_TPCC_DRAEM_k.DRAEM_2[0] E0) | (EDMA_TPCC_IPR[1] E1 & EDMA_TPCC_IER[1] E1 & EDMA_TPCC_DRAEM_k.DRAEM_2[1] E1) | …|(EDMA_TPCC_IPRH[31] E63 & EDMA_TPCC_IERH[31] E63 & EDMA_TPCC_DRAEHM_k.DRAEHM_2[31] E63)....
- Up to EDMA_TPCC_INT7: (EDMA_TPCC_IPR[0] E0 & EDMA_TPCC_IER[0] E0 & EDMA_TPCC_DRAEM_k.DRAEM_7[0] E0) | (EDMA_TPCC_IPR[1] E1 & EDMA_TPCC_IER[1] E1 & EDMA_TPCC_DRAEM_k.DRAEM_7[1] E1) | …|(EDMA_TPCC_IPRH[31] E63 & EDMA_TPCC_IERH[31] E63 & EDMA_TPCC_DRAEHM_k.DRAEHM_7[31] E63)

Note

The EDMA_TPCC_DRAEM_k / EDMA_TPCC_DRAEHM_k for all regions are expected to be set up at system initialization and to remain static for an extended period of time. The interrupt enable registers are used for dynamic enable/disable of individual interrupts.

Because there is no relation between the EDMA_TPCC_OPT_n[17:12] TCC value and the DMA/QDMA channel, it is possible, the DMA channel 0 to have the EDMA_TPCC_OPT_n[17:12] TCC = 63 in its associated PaRAM set. This mean that if a transfer completion interrupt is enabled (EDMA_TPCC_OPT_n[20] TCINTEN or EDMA_TPCC_OPT_n[21] ITCINTEN is set), then based on the TCC value, EDMA_TPCC_IPRH[31] E63 is set up on completion. For proper channel operations and interrupt generation using the shadow region map - program the EDMA_TPCC_DRAEM_k / EDMA_TPCC_DRAEHM_k that is associated with the shadow region to have read/write access to both bit 0 (corresponding to channel 0) and bit 63 (corresponding to EDMA_TPCC_IPRH bit that is set upon completion).

11.2.3.9.1.2 Clearing Transfer Completion Interrupts

Transfer completion interrupts that are latched to the interrupt pending registers (EDMA_TPCC_IPR / EDMA_TPCC_IPRH) are cleared by writing a 1 to the corresponding bit in the interrupt pending clear register (EDMA_TPCC_ICR / EDMA_TPCC_ICRH). For example, a write of 1 to EDMA_TPCC_ICR[0] E0 clears a pending interrupt in EDMA_TPCC_IPR[0] E0.

If an incoming transfer completion code TCC (EDMA_TPCC_OPT_n[17:12] TCC) gets latched to a bit in EDMA_TPCC_IPR / EDMA_TPCC_IPRH, then additional bits that get set due to a subsequent transfer completion does not result in asserting the EDMA_TPCC completion interrupt. In order for the completion interrupt to be pulsed, the required transition is from a state where no enabled interrupts are set to a state where at least one enabled interrupt is set.

11.2.3.9.2 EDMA Interrupt Servicing

Upon completion of a transfer (early or normal completion), the EDMA channel controller sets the appropriate bit in the interrupt pending registers (EDMA_TPCC_IPR / EDMA_TPCC_IPRH), as the transfer completion codes specify. If the completion interrupts are appropriately enabled, then the CPU enters the interrupt service routine (ISR) when the completion interrupt is asserted.

After servicing the interrupt, the ISR should clear the corresponding bit in EDMA_TPCC_IPR/ EDMA_TPCC_IPRH, thereby enabling recognition of future interrupts. The EDMA_TPCC only asserts additional completion interrupts when all EDMA_TPCC_IPR / EDMA_TPCC_IPRH bits clear.

When one interrupt is serviced many other transfer completions may result in additional bits being set in EDMA_TPCC_IPR / EDMA_TPCC_IPRH, thereby resulting in additional interrupts. Each of the bits in EDMA_TPCC_IPR / EDMA_TPCC_IPRH may need different types of service therefore, the ISR must check all pending interrupts and continue until all of the posted interrupts are serviced appropriately.

Examples of pseudo code for a CPU interrupt service routine for an EDMA_TPCC completion interrupt are shown in Example 11-2 and Example 11-3.

The ISR routine in Example 11-2 is more exhaustive and incurs a higher latency.

Example 11-2. Interrupt Servicing

The pseudo code:

1. Reads the interrupt pending register EDMA_TPCC_IPR / EDMA_TPCC_IPRH.
2. Performs the operations needed.
3. Writes to the interrupt pending clear register EDMA_TPCC_ICR / EDMA_TPCC_ICRH to clear the corresponding EDMA_TPCC_IPR / EDMA_TPCC_IPRH bit(s).
4. Reads EDMA_TPCC_IPR / EDMA_TPCC_IPRH again:

a. If EDMA_TPCC_IPR / EDMA_TPCC_IPRH is not equal to 0, repeat from step 2 (implies occurrence of new event between step 2 to step 4).
b. If EDMA_TPCC_IPR / EDMA_TPCC_IPRH is equal to 0, assure that all of the enabled interrupts are inactive.

Note

An event may occur during step 4 while the EDMA_TPCC_IPR / EDMA_TPCC_IPRH bits are read as 0 and the application is still in the interrupt service routine. If this happens, a new interrupt is recorded in the device interrupt controller and a new interrupt generates as soon as the application exits in the interrupt service routine.

11.2.3.9.3

Example 11-3 is less rigorous, with less burden on the software in polling for set interrupt bits, but can occasionally cause a race condition as mentioned above.

Example 11-3. Interrupt Servicing

If any enabled and pending (possibly lower priority) interrupts are left, force the interrupt logic to reassert the interrupt pulse by setting the EDMA_TPCC_IEVAL[0] EVAL bit in the interrupt evaluation register.

The pseudo code is as follows:

1. Enters ISR.
2. Reads EDMA_TPCC_IPR / EDMA_TPCC_IPRH.
3. For the condition that is set in EDMA_TPCC_IPR / EDMA_TPCC_IPRH:
   a. Service interrupt as the application requires.
   b. Clear the bit for serviced conditions (others may still be set, and other transfers may have resulted in returning the TCC to EDMA_TPCC after step 2).
4. Reads EDMA_TPCC_IPR / EDMA_TPCC_IPRH prior to exiting the ISR:
   a. If EDMA_TPCC_IPR / EDMA_TPCC_IPRH is equal to 0, then exit the ISR.
   b. If EDMA_TPCC_IPR / EDMA_TPCC_IPRH is not equal to 0, then set EDMA_TPCC_IEVAL so that upon exit of ISR, a new interrupt triggers if any enabled interrupts are still pending.

11.2.3.9.4 Interrupt Evaluation Operations

The EDMA_TPCC has interrupt evaluate registers EDMA_TPCC_IEVAL that exist in the global region and in each shadow region. The registers in the shadow region are the only registers in the DMA channel shadow region memory map that are not affected by the settings for the DMA region access enable registers EDMA_TPCC_DRAEM_k / EDMA_TPCC_DRAEHM_k. Writing a 1 to the EDMA_TPCC_IEVAL[0] EVAL bit in the registers that are associated with a particular shadow region results in pulsing the associated region interrupt (global or shadow), if any enabled interrupt (via EDMA_TPCC_IER / EDMA_TPCC_IERH) is still pending EDMA_TPCC_IPR / EDMA_TPCC_IPRH. This register assures that the CPU does not miss the interrupts (or the EDMA controller associated with the shadow region) if the software architecture chooses not to use all interrupts. Refer to Example 11-3 about the use of EDMA_TPCC_IEVAL in the EDMA interrupt service routine (ISR).

Similarly an error evaluation register EDMA_TPCC_EEVAL exists in the global region. Writing a 1 to the EDMA_TPCC_EEVAL[0] EVAL bit causes the pulsing of the error interrupt if any pending errors are in EDMA_TPCC_EMR / EDMA_TPCC_EMRH, EDMA_TPCC_QEMR, or EDMA_TPCC_CCERR. See Section 11.2.3.9.5 *Error Interrupts* for additional information regarding error interrupts.

Note

While using EDMA_TPCC_IEVAL for shadow region completion interrupts, check that the EDMA_TPCC_IEVAL operated upon is from that particular shadow region memory map.

11.2.3.9.5 Error Interrupts

The EDMA_TPCC error registers provide the capability to differentiate error conditions (event missed, threshold exceed, etc.). Additionally, setting the error bits in these registers results in asserting the EDMA_TPCC error interrupt. If the EDMA_TPCC error interrupt is enabled in the device interrupt controller(s), then it allows the CPU(s) to handle the error conditions.

The EDMA_TPCC has a single error interrupt (EDMA_TPCC_ERRINT) that is asserted for all EDMA_TPCC error conditions. There are four conditions that cause the error interrupt:

- DMA missed events: for all 64 DMA channels. DMA missed events are latched in the event missed registers EDMA_TPCC_EMR / EDMA_TPCC_EMRH.
- QDMA missed events: for all 8 QDMA channels. QDMA missed events are latched in the QDMA event missed register EDMA_TPCC_QEMR.
- Threshold exceed: for all event queues. These are latched in EDMA_TPCC error register EDMA_TPCC_CCERR.
- TCC error: for outstanding transfer requests that are expected to return completion code EDMA_TPCC_OPT_n[22] TCCHEN or EDMA_TPCC_OPT_n[23] TCINTEN bit is set to 1, exceeding the maximum limit of 63. This is also latched in the EDMA_TPCC error register EDMA_TPCC_CCERR.

Figure 11-16 illustrates the EDMA_TPCC error interrupt generation operation.

If any of the bits are set in the error registers due to any error condition, the EDMA_TPCC_ERRINT is always asserted, as there are no enables for masking these error events. Similar to transfer completion interrupts (EDMA_TPCC_INT), the error interrupt also only pulses when the error interrupt condition transitions from no errors being set to at least one error being set. If additional error events are latched prior to the original error bits clearing, the EDMA_TPCC does not generate additional interrupt.

To reduce the burden on the software, there is an error evaluate register EDMA_TPCC_EEVAL that allows re-evaluation of pending set error events/bits, similar to the interrupt evaluate register EDMA_TPCC_IEVAL. Unlike the EDMA_TPCC_IEVAL functionality, the EDMA_TPCC_EEVAL register must be written with ‘1’ after any error interrupts are serviced (even when all pending errors are cleared) in order for subsequent errors to trigger a new interrupt.

Note

It is good practice to enable the error interrupt in the device interrupt controller and to associate an interrupt service routine with it to address the various error conditions appropriately. Doing so puts less burden on the software (polling for error status), it provides a good debug mechanism for unexpected error conditions.

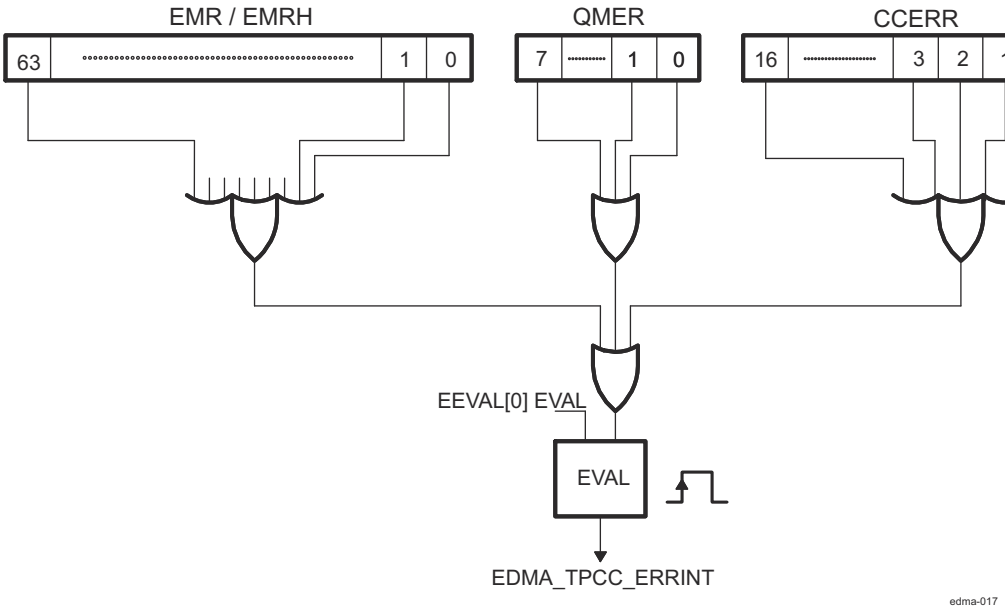


Figure 11-16. Error Interrupt Operation

11.2.3.10 Memory Protection

The EDMA channel controller supports two kinds of memory protection: active and proxy.

11.2.3.10.1 Active Memory Protection

Active memory protection is a feature that allows or prevents read and write accesses to the EDMA_TPCC registers. Active memory protection is achieved by a set of memory protection permissions attribute EDMA_TPCC_MPPAN_k registers.

The EDMA_TPCC register map is divided into three categories:

- a global region.
- a global channel region.
- eight shadow regions.

Each shadow region consists of the respective shadow region registers and the associated PaRAM. For more detailed information regarding the contents of a shadow region, refer to the associated Register Addendum.

Each of the eight shadow regions has an associated EDMA_TPCC_MPPAN_k registers that defines the specific requestor(s) and types of requests that are allowed to the regions resources.

The global channel region is also protected with a memory-mapped register EDMA_TPCC_MPPAG. The EDMA_TPCC_MPPAG applies to the global region and to the global channel region, except the other EDMA_TPCC_MPPAN_k registers themselves.

Table 11-18 shows the accesses that are allowed or not allowed to the EDMA_TPCC_MPPAG and EDMA_TPCC_MPPAN_k. The active memory protection uses the EDMA_TPCC_OPT_n[31] PRIV and EDMA_TPCC_OPT_n[27:24] PRIVID attributes of the EDMA peripheral modules. TheEDMA_TPCC_OPT_n[31] PRIV is the privilege level (i.e., user vs. supervisor).

The EDMA_TPCC_OPT_n[27:24] PRIVID refers to a privilege ID with a number that is associated with an EDMA peripheral modules.

Table 11-18. Allowed Accesses

Access	Supervisor	User
Read	Yes	Yes
Write	Yes	No

Table 11-19 describes the EDMA_TPCC_MPPAN_k register mapping for the shadow regions (which includes shadow region registers and PaRAM addresses).

The region-based EDMA_TPCC_MPPAN_k registers are used to protect accesses to the DMA shadow regions and the associated region PaRAM. Because there are eight regions, there are eight EDMA_TPCC_MPPAN_k region registers (MPPA[0-7]).

Table 11-19. MPPA Registers to Region Assignment

Register	Registers Protect	Address Range	PaRAM Protect(1)	Address Range
EDMA_TPCC_MPPAG	Global Range	0000h-1FFCh	N/A	N/A
EDMA_TPCC_MPPAN_k. MPPAN_0	DMA Shadow 0	2000h-21FCh	1st octant	4000h-47FCh
MPPAN_1	DMA Shadow 1	2200h-23FCh	2nd octant	4800h-4FFCh
MPPAN_2	DMA Shadow 2	2400h-25FCh	3rd octant	5000h-57FCh
MPPAN_3	DMA Shadow 3	2600h-27FCh	4th octant	5800h-5FFCh
MPPAN_4	DMA Shadow 4	2800h-29FCh	5th octant	6000h-67FCh
MPPAN_5	DMA Shadow 5	2A00h-2BFCh	6th octant	6800h-6FFCh
MPPAN_6	DMA Shadow 6	2C00h-2DFCh	7th octant	7000h-77FCh
MPPAN_7	DMA Shadow 7	2E00h-2FFCh	8th octant	7800h-7FFCh

(1) The PARAM region is divided into 8 regions referred to as an octant.

Example *Access denied.*

Write access to shadow region 7's event enable set register EDMA_TPCC_EESR:

1. The original value of the event enable register EDMA_TPCC_EER at address offset 0x1020 is 0x0.
2. The EDMA_TPCC_MPPAN_k. EDMA_TPCC_MPPAN_7[7] NS is set to prevent user level accesses (EDMA_TPCC_MPPAN_k. EDMA_TPCC_MPPAN_7[1] UW = 0, EDMA_TPCC_MPPAN_k. EDMA_TPCC_MPPAN_7[2] UR = 0), but it allows supervisor level accesses (EDMA_TPCC_MPPAN_k. EDMA_TPCC_MPPAN_7[4] SW = 1, EDMA_TPCC_MPPAN_k. EDMA_TPCC_MPPAN_7[5] SR = 1) with a privilege ID of 0. (EDMA_TPCC_MPPAN_k. EDMA_TPCC_MPPAN_7[10] AID0 = 1).
3. EDMA peripheral modules with a privilege ID of 0 attempts to perform a user-level write of a value of 0xFF00FF00 to shadow region 7's event enable set register EDMA_TPCC_EESR at address offset 0x2E30.

Note

The EDMA_TPCC_EER is a read-only register and the only way that write to it is by writing to the EDMA_TPCC_EESR. There is only one physical register for EDMA_TPCC_EER, EDMA_TPCC_EESR, etc. and that the shadow regions only provide to the same physical set.

4. Since the EDMA_TPCC_MPPAN_k. EDMA_TPCC_MPPAN_7[1] UW = 0, though the privilege ID of the write access is set to 0, the access is not allowed and the EDMA_TPCC_EER is not written too.

Example *Access Allowed*

Write access to shadow region 7's event enable set register EDMA_TPCC_EESR:

1. The original value of the event enable register EDMA_TPCC_EER at address offset 0x1020 is 0x0.
2. The EDMA_TPCC_MPPAN_k.EDMA_TPCC_MPPAN_7 is set to allow user-level accesses (EDMA_TPCC_MPPAN_k.EDMA_TPCC_MPPAN_7[1] UW = 1, EDMA_TPCC_MPPAN_k. EDMA_TPCC_MPPAN_7[2] UR = 1) and supervisor-level accesses (EDMA_TPCC_MPPAN_k. EDMA_TPCC_MPPAN_7[4] SW = 1, EDMA_TPCC_MPPAN_k. EDMA_TPCC_MPPAN_7[5] SR = 1) with a privilege ID of 0. (EDMA_TPCC_MPPAN_k. EDMA_TPCC_MPPAN_7[10] AID0 = 1).
3. EDMA peripheral modules with a privilege ID of 0, attempts to perform a user-level write of a value of 0xABCD0123 to shadow region 7's event enable set register EDMA_TPCC_EESR at address offset 0x2E30.

Note

The EDMA_TPCC_EER is a read-only register and the only way that write to it is by writing to the EDMA_TPCC_EESR. There is only one physical register for EDMA_TPCC_EER, EDMA_TPCC_EESR, etc. and that the shadow regions only provide to the same physical set.

4. Since the EDMA_TPCC_MPPAN_k. EDMA_TPCC_MPPAN_7[1] UW = 1 and EDMA_TPCC_MPPAN_k. MPPAN_7[10] AID0 = 1, the user-level write access is allowed.
5. The accesse to shadow region registers are masked by their respective EDMA_TPCC_DRAEM_k register. In this example, the EDMA_TPCC_DRAEM_k. EDMA_TPCC_DRAEM_7 is set of 0x9FF00FC2.
6. The value finally written to EDMA_TPCC_EER is 0x8BC00102.

11.2.3.10.2 Proxy Memory Protection

Proxy memory protection allows an EDMA transfer programmed by a given peripheral module connected to EDMA, to have its permissions travel with the transfer through the EDMA_TPTC. The permissions travel along with the read transactions to the source and the write transactions to the destination endpoints. The EDMA_TPCC_OPT_n[31] PRIV bit and EDMA_TPCC_OPT_n[27:24] PRIVID bit is set with the peripheral module's PRIV value and PRIVID values, respectively, when any part of the PaRAM set is written.

The EDMA_TPCC_OPT_n[31] PRIV is the privilege level (i.e., user vs. supervisor). The EDMA_TPCC_OPT_n[27:24] PRIVID refers to a privilege ID with a number that is associated with an peripheral module connected to EDMA.

These options are part of the TR that are submitted to the transfer controller. The transfer controller uses the above values on their respective read and write command bus so that the target endpoints can perform memory protection checks based on these values.

Consider a parameter set that is programmed by a CPU in user privilege level for a simple transfer with the source buffer on an L2 page and the destination buffer on an L1D page. The EDMA_TPCC_OPT_n[31] PRIV is 0 for user-level and the CPU has a EDMA_TPCC_OPT_n[27:24] PRIVID to 0.

The PaRAM set is shown in Figure 11-17.

Figure 11-17. PaRAM Set Content for Proxy Memory Protection Example

(a) EDMA Parameters

Parameter Contents		Parameter	
0010 0007h		Channel Options Parameter (OPT)	
009F 0000h		Channel Source Address (SRC)	
0001h	0004h	Count for 2nd Dimension (BCNT)	Count for 1st Dimension (ACNT)
00F0 7800h		Channel Destination Address (DST)	
0001h	0001h	Destination BCNT Index (DBIDX)	Source BCNT Index (SBIDX)
0000h	FFFFh	BCNT Reload (BCNTRLD)	Link Address (LINK)
0001h	1000h	Destination CCNT Index (DCIDX)	Source CCNT Index (SCIDX)
0000h	0001h	Reserved	Count for 3rd Dimension (CCNT)

(b) Channel Options Parameter (OPT_n) Content

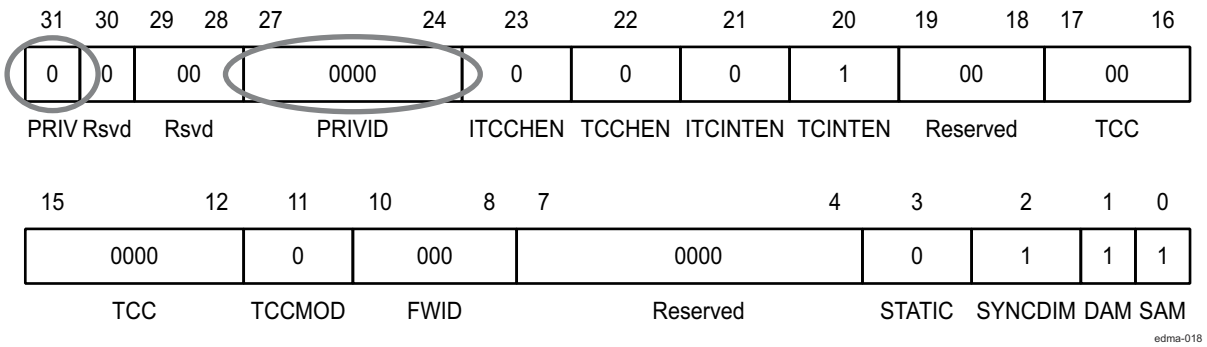


Figure 11-18. Channel Options Parameter (OPT) Example

The EDMA_TPCC_OPT_n[31] PRIV and EDMA_TPCC_OPT_n[27:24] PRIVID information travels along with the read and write requests that are issued to the source and destination memories.

For example, if the access attributes that are associated with the L2 page with the source buffer only allow supervisor read, write accesses (EDMA_TPCC_MPPAN_k[4] SW and EDMA_TPCC_MPPAN_k[5] SR), the user-level read request above is refused. Similarly, if the access attributes that are associated with the L1D page with the destination buffer only allow supervisor read and write accesses (EDMA_TPCC_MPPAN_k[4] SW, EDMA_TPCC_MPPAN_k[5] SR), the user-level write request above is refused. For the transfer to succeed, the source and destination pages must have user-read and user-write permissions, respectively, along with allowing accesses from a PRIVID = 0.

Because the privilege level and privilege identification travel with the read and write requests, EDMA acts as a proxy.

Figure 11-19 illustrates the propagation of EDMA_TPCC_OPT_n[31] PRIV and EDMA_TPCC_OPT_n[27:24] PRIVID at the boundaries of all the interacting entities (CPU, EDMA_TPCC, EDMA_TPTCs, and slave memories).

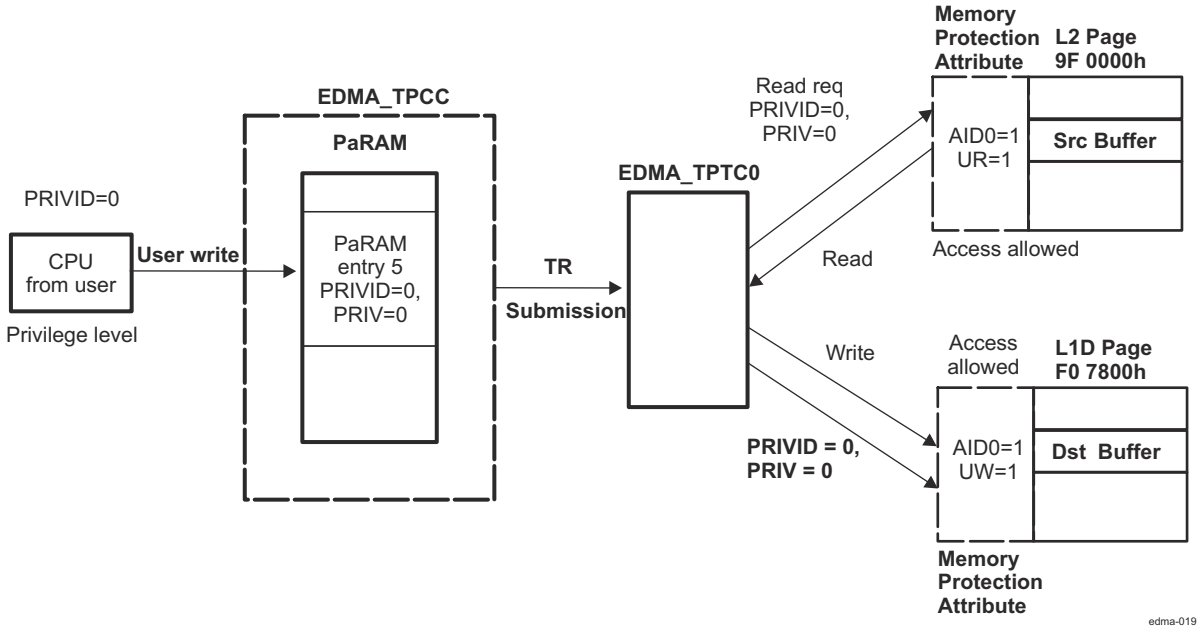


Figure 11-19. Proxy Memory Protection Example

11.2.3.11 Event Queue(s)

Event queues are a part of the EDMA channel controller. Event queues form the interface between the event detection logic in the EDMA_TPCC and the transfer request (TR) submission logic of the EDMA_TPCC. Each queue is 16 entries deep. Each event queue can queue a maximum of 16 events. If there are more than 16 events, then the events that cannot find a place in the event queue remain set in the associated event register and the CPU does not stall.

There are two event queues for the device: Queue0, Queue1. Events in Queue0 result in submission of its associated transfer requests (TRs) to TC0. The transfer requests that are associated with events in Queue1 are submitted to TC1.

An event that wins prioritization against other DMA and/or QDMA pending events is placed at the tail of the appropriate event queue. Each event queue is serviced in FIFO order. Once the event reaches the head of its queue and the corresponding transfer controller is ready to receive another TR, the event is de-queued and the PaRAM set corresponding to the de-queued event is processed and submitted as a transfer request packet (TRP) to the associated EDMA transfer controller.

Queue0 has highest priority and Queue1 has the lowest priority, if Queue0 and Queue1 both have at least one event entry and if both TC0 and TC1 can accept transfer requests, then the event in Queue0 is de-queued first and its associated PaRAM set is processed and submitted as a transfer request (TR) to TC0.

Refer to *Performance Considerations* for system-level performance considerations. All of the event entries in all of the event queues are software readable (not writeable) by accessing the event entry registers EDMA_TPCC_Q0E_p and EDMA_TPCC_Q1E_p. Each event entry register characterizes the queued event in terms of the type of event (manual, event, chained or auto-triggered) and the event number. Refer to the associated Register Addendum for EDMA_TPCC_Q0E_p / EDMA_TPCC_Q1E_p descriptions of the bit fields.

11.2.3.11.1 DMA/QDMA Channel to Event Queue Mapping

Each of the 64 DMA channels and eight QDMA channels are programmed independently to map to a specific queue, using the DMA queue number register EDMA_TPCC_DMAQNUMN_k and the QDMA queue number register EDMA_TPCC_QDMAQNUM. The mapping of DMA/QDMA channels is critical to achieving the desired performance level for the EDMA and most importantly, in meeting real-time deadlines. Refer to *System-level Performance Considerations*.

Note

If an event is ready to be queued and both the event queue and the EDMA transfer controller that is associated to the event queue are empty, then the event bypasses the event queue, and moves the PaRAM processing logic, and eventually to the transfer request submission logic for submission to the EDMA_TPTC. In this case, the event is not logged in the event queue status registers.

11.2.3.11.2 Queue RAM Debug Visibility

There are two event queues and each queue has 16 entries. These 16 entries are managed in a circular FIFO. There is a queue status register EDMA_TPCC_QSTATN_i associated with each queue. These along with all of the 16 entries per queue can be read via registers EDMA_TPCC_QSTATN_i and Q0E_p / Q1E_p, respectively.

These registers provide user visibility.

The event queue entry register (Q*x*E*y* Q0E_p / Q1E_p) uniquely identifies the specific event type (event-triggered, manually-triggered, chain-triggered, and QDMA events) along with the event number (for all DMA/QDMA event channels) that are in the queue or have been de-queued (passed through the queue).

Each of the 16 entries in the event queue are read using the EDMA_TPCC memory-mapped register. To see the history of the last 16 TRs that have been processed by the EDMA on a given queue, read the event queue registers. This provides user/software visibility and is helpful for debugging real-time issues (typically post-mortem), involving multiple events and event sources.

The queue status register (QSTAT*n* EDMA_TPCC_QSTATN_i) includes fields for the start pointer EDMA_TPCC_QSTATN_i[3:0] STRTPTR which provides the offset to the head entry of an event. It also includes a field called EDMA_TPCC_QSTATN_i[12:8] NUMVAL that provides the total number of valid entries residing in the event queue at a given instance of time. The EDMA_TPCC_QSTATN_i[3:0] STRTPTR is used to index appropriately into the 16 event entries. EDMA_TPCC_QSTATN_i[12:8] NUMVAL number of entries starting from STRTPTR are indicative of events still queued in the respective queue. The remaining entry must be read to determine what's already de-queued and submitted to the associated transfer controller.

11.2.3.11.3 Queue Resource Tracking

The EDMA_TPCC event queue includes watermarking/threshold logic that allows to keep track of maximum usage of all event queues. This is useful for debugging real-time deadline violations that may result from head-of-line blocking on a given EDMA event queue.

The maximum number of events are programed that the queue up in an event queue by programming the threshold value (between 0 to 15) in the queue watermark threshold A register EDMA_TPCC_QWMTHRA. The maximum queue usage is recorded actively in the watermark EDMA_TPCC_QSTATN_i[20:16] WM field of the queue status register, that keeps getting updated based on a comparison of number of valid entries, which is also visible in the EDMA_TPCC_QSTATN_i[12:8] NUMVAL bit and the maximum number of entries.

If the queue usage is exceeded, this status is visible in the EDMA_TPCC registers: the QTHRXCD*n* bits in the channel controller error register EDMA_TPCC_CCERR[7:0] and the EDMA_TPCC_QSTATN_i[24] THRXCD bit, where *n* stands for the event queue number. Any bits that are set in EDMA_TPCC_CCERR also generate an EDMA_TPCC error interrupt.

11.2.3.11.4 Performance Considerations

The device system bus infrastructure arbitrates bus requests from all of the controllers (TCs, CPU(S), and other bus controllers) to the shared target resources (peripherals and memories).

Therefore, the priority of unloading queues has a secondary affect compared to the priority of the transfers as they are executed by the EDMA_TPTC.

11.2.3.12 EDMA Transfer Controller (EDMA_TPTC)

The EDMA channel controller is the user-interface of the EDMA and the EDMA transfer controller (EDMA_TPTC) is the data movement engine of the EDMA controller. The EDMA_TPCC submits transfer requests (TR) to the EDMA_TPTC and the EDMA_TPTC performs the data transfers dictated by the TR, so the EDMA_TPTC is a slave to the EDMA_TPCC.

11.2.3.12.1 Architecture Details

11.2.3.12.1.1 Command Fragmentation

The TC read and write controllers in conjunction with the source and destination register sets are responsible for issuing optimally-sized reads and writes to the target endpoints. An optimally-sized command is defined by the transfer controller default burst size (DBS), which is defined in the *TPTC DBS Configuration registers*.

The EDMA_TPTC attempts to issue the largest possible command size as limited by the DBS value or the EDMA_TPCC_ABCNT_n[15:0] ACNT and EDMA_TPCC_ABCNT_n[31:16] BCNT value of the TR. EDMA_TPTC obeys the following rules:

- The read/write controllers always issue commands less than or equal to the DBS value.
- The first command of a 1D transfer command always aligns the address of subsequent commands to the DBS value.

Table 11-20 lists the TR segmentation rules that are followed by the EDMA_TPTC. In summary, if the EDMA_TPCC_ABCNT_n[15:0] ACNT value is larger than the DBS value, then the EDMA_TPTC breaks the EDMA_TPCC_ABCNT_n[15:0] ACNT array into DBS-sized commands to the source/destination addresses. Each EDMA_TPCC_ABCNT_n[31:16] BCNT number of arrays are then serviced in succession.

For BCNT arrays of ACNT bytes (that is, a 2D transfer), if the EDMA_TPCC_ABCNT_n[15:0] ACNT value is less than or equal to the DBS value, then the TR may be optimized into a 1D-transfer in order to maximize efficiency. The optimization takes place if the EDMA_TPTC recognizes that the 2D-transfer is organized as a single dimension (EDMA_TPCC_ABCNT_n[15:0] ACNT == EDMA_TPCC_BIDX_n) and the ACNT value is a power of 2.

Table 11-20 lists conditions in which the optimizations are performed.

Table 11-20. Read/Write Command Optimization Rules

ACNT ≤ DBS	ACNT is power of 2	BIDX = ACNT	BCNT ≤ 1023	SAM/DAM = Increment	Description
Yes	Yes	Yes	Yes	Yes	Optimized
No	x	x	x	x	Not Optimized
x	No	x	x	x	Not Optimized
x	x	No	x	x	Not Optimized
x	x	x	No	x	Not Optimized
x	x	x	x	No	Not Optimized

11.2.3.12.1.2 TR Pipelining

TR pipelining refers to the ability of the source active set to proceed ahead of the destination active set. Essentially, the reads for a given TR may already be in progress while the writes of a previous TR may not have completed.

The number of outstanding TRs is limited by the number of destination FIFO register entries.

TR pipelining is useful for maintaining throughput on back-to-back small TRs. It minimizes the startup overhead because reads start in the background of a previous TR writes.

Example 11-4. Command Fragmentation (DBS = 64)

The pseudo code:

1. EDMA_TPTCn_PCNT[15:0] ACNT = 8, EDMA_TPTCn_PCNT[31:16] BCNT = 8, EDMA_TPTCn_PBIDX[15:0] SBIDX = 8, EDMA_TPTCn_PBIDX[31:16] DBIDX = 10, EDMA_TPTCn_PSRC[31:0] SADDR = 64, EDMA_TPTCn_SADST[31:0] DADDR = 191

   Read Controller: This is optimized from a 2D-transfer to a 1D-transfer such that the read side is equivalent to EDMA_TPTCn_PCNT[15:0] ACNT = 64, EDMA_TPTCn_PCNT[31:16] BCNT = 1.

   Cmd0 = 64 byte

   Write Controller: Because DBIDX != ACNT, it is not optimized.

   Cmd0 = 8 byte, Cmd1 = 8 byte, Cmd2 = 8 byte, Cmd3 = 8 byte, Cmd4 = 8 byte, Cmd5 = 8 byte, Cmd6 = 8 byte, Cmd7 = 8 byte.
2. EDMA_TPTCn_PCNT[15:0] ACNT=128, EDMA_TPTCn_PCNT[31:16] BCNT = 1, EDMA_TPTCn_PSRC[31:0] SADDR = 63, EDMA_TPTCn_SADST[31:0] DADDR = 513

   Read Controller: Read address is not aligned.

   Cmd0 = 1 byte, (now the SADDR is aligned to 64 for the next command)

   Cmd1 = 64 bytes

   Cmd2 = 63 bytes

   Write Controller: The write address is also not aligned.

   Cmd0 = 63 bytes, (now the DADDR is aligned to 64 for the next command)

   Cmd1 = 64 bytes

   Cmd2 = 1 byte

11.2.3.12.1.3 Performance Tuning

By default, reads are as issued as fast as possible. In some cases, the reads issued by the EDMA_TPTC could fill the available command buffering for a target, delaying other (potentially higher priority) controllers from successfully submitting commands to that target. The rate at which read commands are issued by the EDMA_TPTC is controlled by the EDMA_TPTCn_RDRATE register. The EDMA_TPTCn_RDRATE register defines the number of cycles that the EDMA_TPTC read controller waits before issuing subsequent commands for a given TR, thus minimizing the chance of the EDMA_TPTC consuming all available target resources. The EDMA_TPTCn_RDRATE[2:0] RDRATE value must be set to a relatively small value if the transfer controller is targeted for high priority transfers and to a higher value if the transfer controller is targeted for low priority transfers.

In contrast, the Write Interface does not have any performance turning knobs because writes always have an interval between commands as write commands are submitted along with the associated write data.

11.2.3.12.2 Memory Protection

The transfer controller plays an important role in handling proxy memory protection. There are two access properties associated with a transfer: for instance, the privilege id (system-wide identification assigned to a controller) of the controller initiating the transfer, and the privilege level (user versus supervisor) used to program the transfer. This information is maintained in the PaRAM set when it is programmed in the channel controller. When a TR is submitted to the transfer controller, this information is made available to the EDMA_TPTC and used by the EDMA_TPTC while issuing read and write commands. The read or write commands have the same privilege identification, and privilege level as that programmed in the EDMA transfer in the channel controller.

11.2.3.12.3 Error Generation

Errors are generated if enabled under three conditions:

- EDMA_TPTC detection of an error signaled by the source or destination address.
- Attempt to read or write to an invalid address in the configuration memory map.
- Detection of a constant addressing mode TR violating the constant addressing mode transfer rules (the source/destination addresses and source/destination indexes must be aligned to 32 bytes).

Either or all error types may be disabled. If an error bit is set and enabled, the error interrupt for the concerned transfer controller is generated.

11.2.3.12.4 Debug Features

The DMA program register set, DMA source active register set, and the destination FIFO register set are used to derive a brief history of TRs serviced through the transfer controller.

Additionally, the EDMA_TPTC status register EDMA_TPTCn_TCSTAT has dedicated bit fields to indicate the ongoing activity within different parts of the transfer controller:

- The EDMA_TPTCn_TCSTAT[1] SRCACTV bit indicates whether the source active set is active.
- The EDMA_TPTCn_TCSTAT[6:4] DSTACTV bit indicates the number of TRs resident in the destination register active set at a given instance.
- The EDMA_TPTCn_TCSTAT[0] PROGBUSY bit indicates whether a valid TR is present in the DMA program set.

Note

If the TRs are in progression, it must realize that there is a chance that the values read from the EDMA_TPTC status registers will be inconsistent since the EDMA_TPTC changes the values of these registers due to ongoing activities.

It is recommended that to ensure no additional submission of TRs to the EDMA_TPTC in order to facilitate ease of debug.

11.2.3.12.4.1 Destination FIFO Register Pointer

The destination FIFO register pointer is implemented as a circular buffer with the start pointer being EDMA_TPTCn_TCSTAT[12:11] DFSTRTPTR and a buffer depth of usually 2 or 4. The EDMA_TPTC maintains two important status details in EDMA_TPTCn_TCSTAT that are used during advanced debugging, if necessary. The EDMA_TPTCn_TCSTAT[12:11] DFSTRTPTR is a start pointer, the index to the head of the destination FIFO register. The EDMA_TPTCn_TCSTAT[6:4] DSTACTV is a counter for the number of valid (occupied) entries. These registers are used to get a brief history of transfers.

Examples of some register field values and their interpretation:

- EDMA_TPTCn_TCSTAT[12:11] DFSTRTPTR = 0x0 and EDMA_TPTCn_TCSTAT[6:4] DSTACTV = 0x0 implies that no TRs are stored in the destination FIFO register.
- EDMA_TPTCn_TCSTAT[12:11] DFSTRTPTR = 0x1 and EDMA_TPTCn_TCSTAT[6:4] DSTACTV = 0x2 implies that two TRs are present. The first pending TR is read from the destination FIFO register entry 1 and the second pending TR is read from the destination FIFO register entry 2.
- EDMA_TPTCn_TCSTAT[12:11] DFSTRTPTR = 0x3 and EDMA_TPTCn_TCSTAT[6:4] DSTACTV = 0x2 implies that two TRs are present. The first pending TR is read from the destination FIFO register entry 3 and the second pending TR is read from the destination FIFO register entry 0.

11.2.3.13 Event Dataflow

This section summarizes the data flow of a single event, from the time the event is latched to the channel controller to the time the transfer completion code is returned. The following steps list the sequence of EDMA_TPCC activity:

1. Event is asserted from an external source (peripheral or external interrupt). This also is similar for a manually-triggered, chained-triggered, or QDMA-triggered event. The event is latched into the EDMA_TPCC_ER[31:0]E*n* / EDMA_TPCC_ERH[31:0] E*n* (or EDMA_TPCC_CER[31:0] E*n* / EDMA_TPCC_CERH[31:0] E*n*, EDMA_TPCC_ESR[31:0] E*n* / EDMA_TPCC_ESRH[31:0] E*n*, EDMA_TPCC_QER[7:0] E*n*) bit.

2. Once an event is prioritized and queued into the appropriate event queue, the EDMA_TPCC_SER[31:0] E*n* \ EDMA_TPCC_SERH[31:0] E*n* (or EDMA_TPCC_QSER[7:0] E*n*) bit is set to inform the event prioritization / processing logic to disregard this event since it is already in the queue. Alternatively, if the transfer controller and the event queue are empty, then the event bypasses the queue.
3. The EDMA_TPCC processing and the submission logic evaluates the appropriate PaRAM set and determines whether it is a non-null and non-dummy transfer request (TR).
4. The EDMA_TPCC clears the EDMA_TPCC_ER[31:0] E*n*/ EDMA_TPCC_ERH[31:0] E*n* (or EDMA_TPCC_CER[31:0] E*n* / EDMA_TPCC_CERH[31:0] E*n*, EDMA_TPCC_ESR[31:0]E*n* / EDMA_TPCC_ESRH[31:0] E*n*, EDMA_TPCC_QER[31:0] E*n*) bit and the EDMA_TPCC_SER[31:0] E*n*/ EDMA_TPCC_SERH[31:0] E*n* bit as soon as it determines the TR is non-null. In the case of a null set, the EDMA_TPCC_SER[31:0] E*n*/ EDMA_TPCC_SERH[31:0] E*n* bit remains set. It submits the non-null/non-dummy TR to the associated transfer controller. If the TR was programmed for early completion, the EDMA_TPCC immediately sets the interrupt pending register (EDMA_TPCC_IPR[31:0] I[TCC] / EDMA_TPCC_IPRH[31:0] I[TCC] - 32).
5. If the TR was programmed for normal completion, the EDMA_TPCC sets the interrupt pending register (EDMA_TPCC_IPR[31:0] I[TCC] / EDMA_TPCC_IPRH[31:0] I[TCC]) when the EDMA_TPTC informs the EDMA_TPCC about completion of the transfer (returns transfer completion codes).
6. The EDMA_TPCC programs the associated EDMA_TPTC's Program Register Set with the TR.
7. The TR is then passed to the Source Active set and the DST FIFO Register Set, if both the register sets are available.
8. The Read Controller processes the TR by issuing read commands to the source slave endpoint. The Read Data lands in the Data FIFO of the EDMA_TPTC*n*.
9. As soon as sufficient data is available, the Write Controller begins processing the TR by issuing write commands to the destination slave endpoint.
10. This continues until the TR completes and the EDMA_TPTC*n* then signals completion status to the EDMA_TPCC.

11.2.3.14 EDMA Controller Prioritization

The EDMA controller has many implementation rules to deal with concurrent events/channels, transfers, etc. The following subsections detail various arbitration details whenever there might be occurrence of concurrent activity. Figure 11-20 shows the different places EDMA priorities come into play.

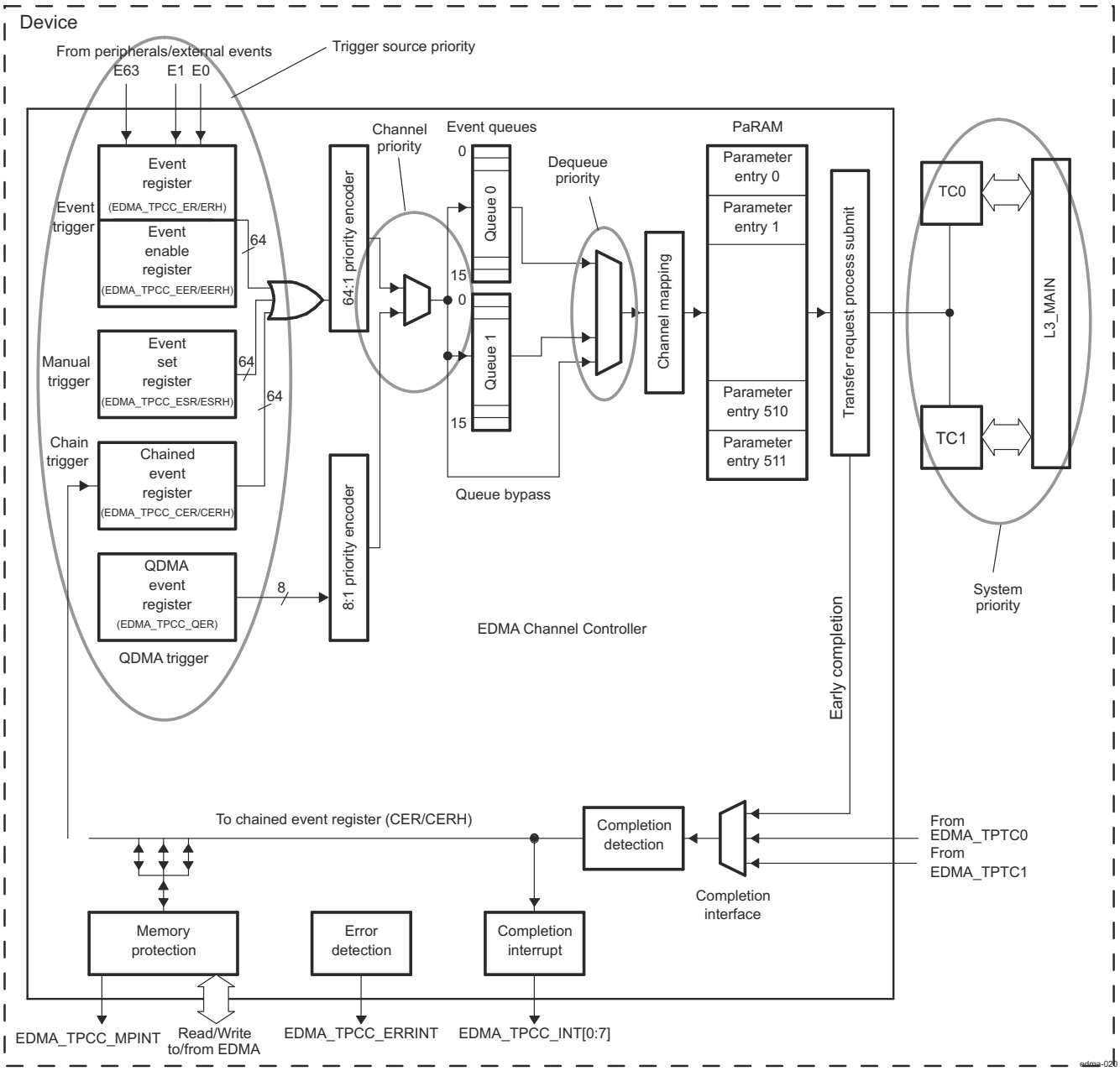


Figure 11-20. EDMA Prioritization

11.2.3.14.1 Channel Priority

The EDMA event registers EDMA_TPCC_ER and EDMA_TPCC_ERH capture up to 64 events, the QDMA event register EDMA_TPCC_QER captures QDMA events for all QDMA channels therefore, it is possible for events to occur simultaneously on the DMA/QDMA event inputs. For events arriving simultaneously, the event associated with the lowest channel number is prioritized for submission to the event queues (for DMA events, channel 0 has the highest priority and channel 63 has the lowest priority, for QDMA events, channel 0 has the highest priority and channel 7 has the lowest priority). This mechanism only sorts simultaneous events for submission to the event queues.

If a DMA and QDMA event occurs simultaneously, the DMA event always has prioritization against the QDMA event for submission to the event queues.

11.2.3.14.2 Trigger Source Priority

If a EDMA channel is associated with more than one trigger source (event trigger, manual trigger, and chain trigger), and if multiple events are set simultaneously for the same channel (EDMA_TPCC_ER[31:0] E*n* = 1, EDMA_TPCC_ESR[31:0] E*n* = 1 , EDMA_TPCC_CER[31:0] E*n* = 1) , then the EDMA_TPCC always services these events in the following priority order: event trigger (via EDMA_TPCC_ER) is higher priority than chain trigger (via EDMA_TPCC_CER) and chain trigger is higher priority than manual trigger (via EDMA_TPCC_ESR).

This implies that if for channel 0, both EDMA_TPCC_ER[0] E0 = 1 and EDMA_TPCC_CER[0] E0 = 1 at the same time, then the EDMA_TPCC_ER[0] E0 event is always queued before the EDMA_TPCC_CER[0] E0 event.

11.2.3.14.3 Dequeue Priority

The priority of the associated transfer request (TR) is further mitigated by which event queue is being used for event submission (dictated by EDMA_TPCC_DMAQNUMN_k and EDMA_TPCC_QDMAQNUM). For submission of a TR to the transfer request, events need to be de-queued from the event queues. Queue 0 has the highest dequeue priority and queue 1 the lowest.

11.2.3.15 Emulation Considerations

During debug when using the emulator, the CPU(s) may be halted on an execute packet boundary for single-stepping, benchmarking, profiling, or other debug purposes. During an emulation halt, the EDMA channel controller and transfer controller operations continue. Events continue to be latched and processed and transfer requests continue to be submitted and serviced.

Since EDMA is involved in servicing multiple controller and target peripherals, it is not feasible to have an independent behavior of the EDMA for emulation halts. EDMA functionality would be coupled with the peripherals it is servicing, which might have different behavior during emulation halts.

11.2.4 EDMA Transfer Examples

The EDMA channel controller performs a variety of transfers depending on the parameter configuration. The following sections provide a description and PaRAM configuration for some typical use case scenarios.

11.2.4.1 Block Move Example

The most basic transfer performed by the EDMA is a block move. During device operation it is often necessary to transfer a block of data from one location to another, usually between on-chip and off-chip memory.

In this example, a section of data is to be copied from external memory to internal L2 SRAM as shown in Figure 11-21.

The source address for the transfer is set to the start of the data block in external memory, and the destination address is set to the start of the data block in L2. If the data block is less than 64K bytes, the PaRAM configuration shown in Figure 11-22 holds true with the synchronization type set to A-synchronized and indexes cleared to 0. If the amount of data is greater than or equal to 64K bytes, EDMA_TPCC_ABCNT_n[31:16] BCNT and the B-indexes need to be set appropriately with the synchronization type set to AB-synchronized. The EDMA_TPCC_OPT_n[3] STATIC bit is set to prevent linking.

This transfer example may also be set up using QDMA. For successive transfer submissions, of a similar nature, the number of cycles used to submit the transfer are fewer depending on the number of changing transfer parameters. The QDMA trigger word must be programed to be the highest numbered offset in the PaRAM set that undergoes change.

Figure 11-22 shows the parameters Block Move transfer.

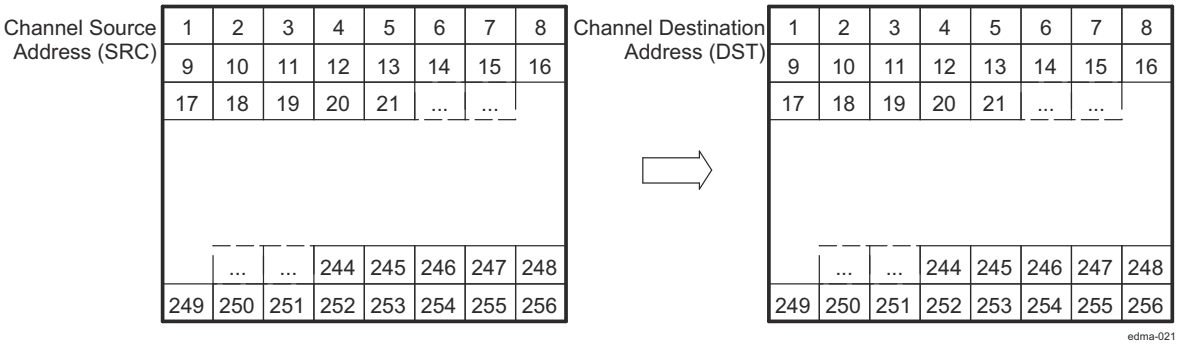


Figure 11-21. Block Move Example

Figure 11-22. Block Move Example PaRAM Configuration

(a) EDMA Parameters

Parameter Contents		Parameter	
0010 0008h		Channel Options Parameter (OPT)	
Channel Source Address (SRC)		Channel Source Address (SRC)	
0001h	FFFFh	Count for 2nd Dimension (BCNT)	Count for 1st Dimension (ACNT)
Channel Destination Address (DST)		Channel Destination Address (DST)	
0000h	0000h	Destination BCNT Index (DBIDX)	Source BCNT Index (SBIDX)
0000h	FFFFh	BCNT Reload (BCNTRLD)	Link Address (LINK)
0000h	0000h	Destination CCNT Index (DCIDX)	Source CCNT Index (SCIDX)
0000h	0001h	Reserved	Count for 3rd Dimension (CCNT)

(b)Channel Options Parameter (OPT) Content

- EDMA_TPCC_OPT_n[3] STATIC = 0x1
- EDMA_TPCC_OPT_n[20] TCINTEN = 0x1

11.2.4.2 Subframe Extraction Example

The EDMA can efficiently extract a small frame of data from a larger frame of data. By performing a 2D-to-1D transfer, the EDMA retrieves a portion of data for the CPU to process. In this example, a 640 × 480-pixel frame of video data is stored in external memory. Each pixel is represented by a 16-bit halfword. The CPU extracts a 16 × 12-pixel subframe of the image for processing. To facilitate more efficient processing time by the CPU, the EDMA places the subframe in internal L2 SRAM. Figure 11-23 shows the transfer of a subframe from external memory to L2.

The same PaRAM entry options are used for QDMA channels, as well as DMA channels. The EDMA_TPCC_OPT_n[3] STATIC bit is set to prevent linking. For successive transfers, only changed parameters need to be programmed before triggering the channel.

Figure 11-24 shows the parameters for Subframe Extraction transfer.

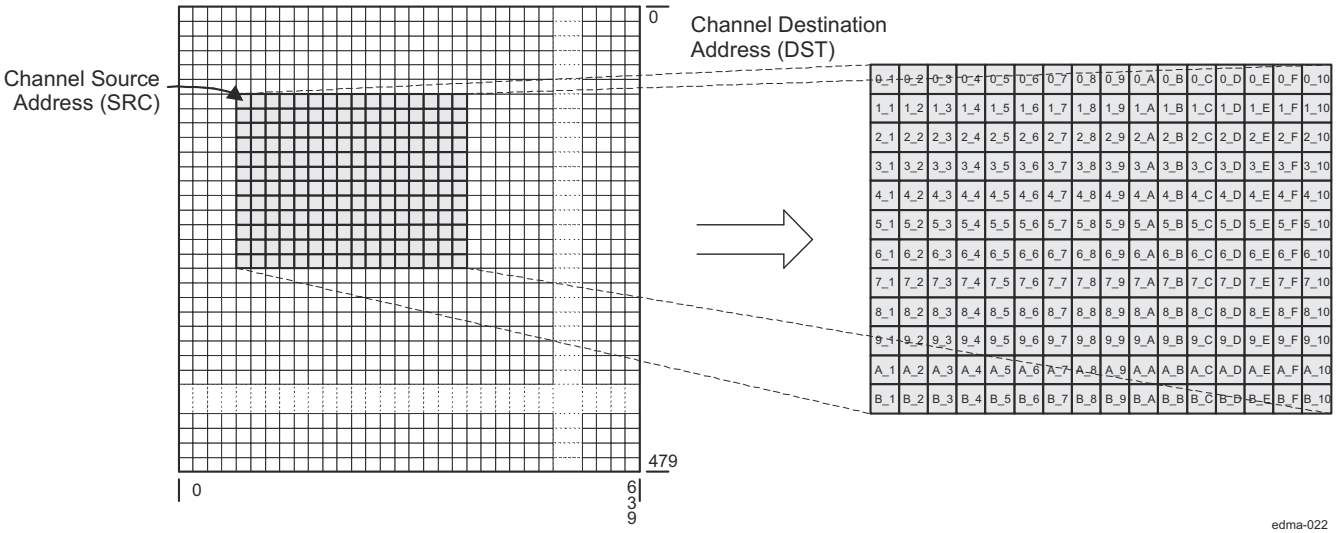


Figure 11-23. Subframe Extraction Transfer

Figure 11-24. Subframe Extraction Example PaRAM Configuration

(a) EDMA Parameters

Parameter Contents			Parameter	
0010 000Ch			Channel Options Parameter (OPT)	
Channel Source Address (SRC)			Channel Source Address (SRC)	
000Ch	0020h		Count for 2nd Dimension (BCNT)	Count for 1st Dimension (ACNT)
Channel Destination Address (DST)			Channel Destination Address (DST)	
0020h	0500h		Destination BCNT Index (DBIDX)	Source BCNT Index (SBIDX)
0000h	FFFFh		BCNT Reload (BCNTRLD)	Link Address (LINK)
0000h	0000h		Destination CCNT Index (DCIDX)	Source CCNT Index (SCIDX)
0000h	0001h		Reserved	Count for 3rd Dimension (CCNT)

(b)Channel Options Parameter (OPT) Content

- EDMA_TPCC_OPT_n[2] SYNCDIM = 0x1
- EDMA_TPCC_OPT_n[3] STATIC = 0x1
- EDMA_TPCC_OPT_n[20] TCINTEN = 0x1

11.2.4.3 Data Sorting Example

Many applications require the use of multiple data arrays, it is often desirable to have the arrays arranged such that the first elements of each array are adjacent, the second elements are adjacent, and so on. Often this is not how the data is presented to the device. Either data is transferred via a peripheral with the data arrays arriving one after the other or the arrays are located in memory with each array occupying a portion of contiguous memory spaces. For these instances, the EDMA can reorganize the data into the desired format.

To determine the parameter set values, the following need to be considered:

- ACNT - Program this to be the size in bytes of an element.
- BCNT - Program this to be the number of elements in a frame.
- CCNT - Program this to be the number of frames.
- SBIDX - Program this to be the size of the element or ACNT.
- DBIDX - CCNT × ACNT
- SCIDX - ACNT × BCNT
- DCIDX - ACNT

The synchronization type needs to be AB-synchronized and the EDMA_TPCC_OPT_n[3] STATIC bit is 0 to allow updates to the parameter set. It is advised to use normal EDMA channels for sorting.

It is not possible to sort this with a single trigger event. Instead, the channel can be programmed to be chained to itself. After BCNT elements get sorted, intermediate chaining could be used to trigger the channel again causing the transfer of the next BCNT elements and so on. Figure 11-26 shows the parameter set programming for this transfer, assuming channel 0 and an element size of 4 bytes.

Figure 11-25 shows the Data Sorting transfer

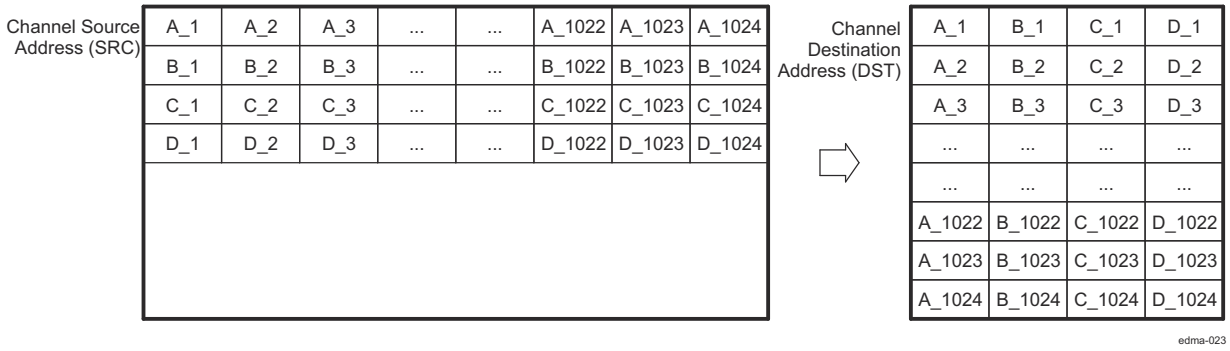


Figure 11-25. Data Sorting Example

Figure 11-26. Data Sorting Example PaRAM Configuration

(a) EDMA Parameters

Parameter Contents		Parameter	
0090 0004h		Channel Options Parameter (OPT)	
Channel Source Address (SRC)		Channel Source Address (SRC)	
0400h	0004h	Count for 2nd Dimension (BCNT)	Count for 1st Dimension (ACNT)
Channel Destination Address (DST)		Channel Destination Address (DST)	
0010h	0001h	Destination BCNT Index (DSTBIDX)	Source BCNT Index (SRCBIDX)
0000h	FFFFh	BCNT Reload (BCNTRLD)	Link Address (LINK)
0001h	1000h	Destination CCNT Index (DSTCIDX)	Source CCNT Index (SRCCIDX)
0000h	0004h	Reserved	Count for 3rd Dimension (CCNT)

(b) Channel Options Parameter (OPT) Content

- EDMA_TPCC_OPT_n[2] SYNCDIM = 0x1
- EDMA_TPCC_OPT_n[20] TCINTEN = 0x1
- EDMA_TPCC_OPT_n[23] ITCCHEN = 0x1

11.2.4.4 Setting Up an EDMA Transfer

The following list provides a quick guide for the typical steps involved in setting up a transfer.

1. Initiating a DMA/QDMA channel
   a. Determine the type of channel (QDMA or DMA) to be used.
   b. Channel mapping
      i. If using a QDMA channel, program the EDMA_TPCC_QCHMAPN_j with the parameter set number to which the channel maps and the trigger word.
      ii. If using a DMA channel, program the EDMA_TPCC_DCHMAPN_m with the parameter set number to which the channel maps.
   c. If the channel is being used in the context of a shadow region, ensure the EDMA_TPCC_DRAEM_k / EDMA_TPCC_DRAEHM_k for the region is properly set up to allow read write accesses to bits in the event registers and interrupt registers in the Shadow region memory map. The subsequent steps in this process should be done using the respective shadow region registers. (Shadow region descriptions and usage are provided in Section 11.2.3.7.1.)
   d. Determine the type of triggering used.
      i. If external events are used for triggering (DMA channels), enable the respective event in EDMA_TPCC_EER / EDMA_TPCC_EERH by writing into EDMA_TPCC_EESR / EDMA_TPCC_EESRH.
      ii. If QDMA Channel is used, enable the channel in EDMA_TPCC_QEER by writing into EDMA_TPCC_QEESR.
   e. Queue setup
      i. If a QDMA channel is used, set up the EDMA_TPCC_QDMAQNUM to map the channel to the respective event queue.
      ii. If a DMA channel is used, set up the EDMA_TPCC_DMAQNUMN_k to map the event to the respective event queue.
2. Parameter set setup
   a. Program the PaRAM set number associated with the channel. Note that

Note

If it is a QDMA channel, the PaRAM entry that is configured as trigger word is written to last. Alternatively, enable the QDMA channel (step 1-d-ii above) just before the write to the trigger word.

3. Interrupt setup
   a. Enable the interrupt in the EDMA_TPCC_IER / EDMA_TPCC_IERH by writing into EDMA_TPCC_IESR / EDMA_TPCC_IESRH.
   b. Ensure the EDMA_TPCC completion interrupt (this refers to either the Global interrupt or the shadow region interrupt) is enabled properly in the Device Interrupt controller.
   c. Set up the interrupt controller properly to receive the expected EDMA interrupt.
4. Initiate transfer
   a. This step is highly dependent on the event trigger source:
      i. If the source is an external event coming from a peripheral, the peripheral will be enabled to start generating relevant EDMA events that can be latched to the EDMA_TPCC_ER transfer.
      ii. For QDMA events, writes to the trigger word (step 2-a above) will initiate the transfer.
      iii. Manually triggered transfers will be initiated by writes to the Event Set Registers EDMA_TPCC_ESR / EDMA_TPCC_ESRH.
      iv. Chained-trigger events initiate when a previous transfer returns a transfer completion code equal to the chained channel number.
5. Wait for completion
   a. If the interrupts are enabled as mentioned in step 3 above, then the EDMA_TPCC will generate a completion interrupt to the CPU whenever transfer completion results in setting the corresponding bits in the interrupt pending register EDMA_TPCC_IPR / EDMA_TPCC_IPRH. The set bits must be cleared in the EDMA_TPCC_IPR / EDMA_TPCC_IPRH by writing to corresponding bit in EDMA_TPCC_ICR / EDMA_TPCC_ICRH.
   b. If polling for completion (interrupts not enabled in the device controller), then the application code can wait on the expected bits to be set in the EDMA_TPCC_IPR / EDMA_TPCC_IPRH. Again, the set

bits in the EDMA_TPCC_IPR / EDMA_TPCC_IPRH must be manually cleared via EDMA_TPCC_ICR / EDMA_TPCC_ICRH before the next set of transfers is performed for the same transfer completion code values.

11.2.5 EDMA Debug Checklist and Programming Tips

This section lists some tips to keep in mind while debugging applications using the EDMA controller.

11.2.5.1 EDMA Debug Checklist

Table 11-21 provides some common issues and their probable causes and resolutions.

Table 11-21. Debug Checklist

Issue	Description/Solution
The transfer associated with the channel does not happen. The channel does not get serviced.	The EDMA_TPCC may not service a transfer request, even though the associated PaRAM set is programmed appropriately. Check for the following: 1) Verify that events are enabled, i.e., if an external/peripheral event is latched in Event Registers EDMA_TPCC_ER / EDMA_TPCC_ERH, check that the event is enabled in the Event Enable Registers EDMA_TPCC_EER / EDMA_TPCC_EERH. Similarly, for QDMA channels, check that QDMA events are appropriately enabled in the QDMA Event Enable Register EDMA_TPCC_QEER. 2) Verify that the DMA or QDMA Secondary Event Register EDMA_TPCC_SER / EDMA_TPCC_SERH / EDMA_TPCC_QSER bits corresponding to the particular event or channel are not set.
The Secondary Event Registers bits are set, not allowing additional transfers to occur on a channel.	It is possible that a trigger event was received when the parameter set associated with the channel/event was a NULL set for a previous transfer on the channel. This is typical in two cases: 1) QDMA channels: Typically if the parameter set is non-static and expected to be terminated by a NULL set (i.e., EDMA_TPCC_OPT_n[3] STATIC = 0x0, EDMA_TPCC_LNK_n[15:0] LINK = 0xFFFF), the parameter set is updated with a NULL set after submission of the last TR. Because QDMA channels are auto-triggered, this update caused the generation of an event. An event generated for a NULL set causes an error condition and results in setting the bits corresponding to the QDMA channel in the EDMA_TPCC_QEMR and EDMA_TPCC_QSER. This will disable further prioritization of the channel. 2) DMA channels used in a continuous mode: The peripheral may be set up to continuously generate infinite events . The parameter set may be programmed to expect only a finite number of events and to be terminated by a NULL link. After the expected number of events, the parameter set is reloaded with a NULL parameter set. Because the peripheral will generate additional events, an error condition is set in the EDMA_TPCC_SER[31:0] E*n* and EDMA_TPCC_EMR[31:0] E*n* set, preventing further event prioritization. Check the number of events received is limited to the expected number of events for which the parameter set is programmed, or check the bits corresponding to particular channel or event are not set in the Secondary event registers (EDMA_TPCC_SER / EDMA_TPCC_SERH / EDMA_TPCC_QSER) and Event Missed Registers (EDMA_TPCC_EMR / EDMA_TPCC_EMRH / EDMA_TPCC_QEMR) before trying to perform subsequent transfers for the event/channel.
Completion interrupts are not asserted, or no further interrupts are received after the first completion interrupt.	Check the following: 1) The interrupt generation is enabled in the EDMA_TPCC_OPT_n of the associated PaRAM set (EDMA_TPCC_OPT_n[20] TCINTEN = 0x1 and/or EDMA_TPCC_OPT_n[20] ITCINTEN = 0x1). 2) The interrupts are enabled in the EDMA Channel Controller, via the Interrupt Enable Registers (EDMA_TPCC_IER / EDMA_TPCC_IERH). 3) The corresponding interrupts are enabled in the device interrupt controller. 4) The set interrupts are cleared in the interrupt pending registers (EDMA_TPCC_IPR / EDMA_TPCC_IPRH) before exiting the transfer completion interrupt service routine (ISR). See Section 11.2.3.9.1.2 *Clearing Transfer Complerion Interrupts* for details on writing EDMA ISRs. 5) If working with shadow region interrupts, make sure that the DMA Region Access registers (EDMA_TPCC_DRAEM_k / EDMA_TPCC_DRAEHM_k) are set up properly, because the EDMA_TPCC_DRAEM_k / EDMA_TPCC_DRAEHM_k registers act as secondary enables for shadow region completion interrupts, along with the EDMA_TPCC_IER / EDMA_TPCC_IERH registers. If working with shadow region interrupts, make sure that the bits corresponding to the transfer completion code EDMA_TPCC_OPT_n[17:12] TCC value are also enabled in the EDMA_TPCC_DRAEM_k / EDMA_TPCC_DRAEHM_k registers. For instance, if the PaRAM set associated with Channel 0 returns a completion code of 63 EDMA_TPCC_OPT_n[17:12] TCC = 63, ensure that EDMA_TPCC_DRAEHM_k[31] E63 is also set for a shadow region completion interrupt because the interrupt pending register bit set will be EDMA_TPCC_IPRH[31] I63 (not EDMA_TPCC_IPR[0] I0).

11.2.5.2 EDMA Programming Tips

1. For several registers, the setting and clearing of bits needs to be done via separate dedicated registers. For example, the Event Register (EDMA_TPCC_ER / EDMA_TPCC_ERH) can only be cleared by writing a 1 to the corresponding bits in the Event Clear Registers (EDMA_TPCC_ECR / EDMA_TPCC_ECRH). Similarly, the Event Enable Register (EDMA_TPCC_EER / EDMA_TPCC_EERH) bits can only be set with writing of 0x1 to the Event Enable Set Registers (EDMA_TPCC_EESR / EDMA_TPCC_EESRH) and cleared with writing of 0x1 to the corresponding bits in the Event Enable Clear Register (EDMA_TPCC_EECR / EDMA_TPCC_EECRH).
2. Writes to the shadow region memory maps are governed by region access registers (EDMA_TPCC_DRAE / EDMA_TPCC_DRAEHM_k / EDMA_TPCC_QRAEN_k). If the appropriate channels are not enabled in these registers, read/write access to the shadow region memory map is not enabled.
3. When working with shadow region completion interrupts, ensure that the DMA Region Access Registers (EDMA_TPCC_DRAEM_k / EDMA_TPCC_DRAEHM_k) for every region are set in a mutually exclusive way (unless it is a requirement for an application). If there is an overlap in the allocated channels and transfer completion codes (setting of Interrupt Pending Register bits) in the region resource allocation, it results in multiple shadow region completion interrupts.
   For example, if EDMA_TPCC_DRAEM_k.DRAEM_0[0] E0 and EDMA_TPCC_DRAEM_k.DRAEM_1[0] E0 are both set, then on completion of a transfer that returns a TCC = 0x0, they will generate both shadow region 0 and 1 completion interrupts.
4. While programming a non-dummy parameter set, ensure the EDMA_TPCC_CCNT_n[15:0] CCNT is not left to zero.
5. Enable the EDMA_TPCC error interrupt in the device controller and attach an interrupt service routine (ISR) to ensure that error conditions are not missed in an application and are appropriately addressed with the ISR.
6. Depending on the application, it can want to break large transfers into smaller transfers and use self-chaining to prevent starvation of other events in an event queue.
7. In applications where a large transfer is broken into sets of small transfers using chaining or other methods, it chooses to use the early chaining option to reduce the time between the sets of transfers and increase the throughput.
   However, keep in mind that with early completion, all data might have not been received at the end point when completion is reported because the EDMA_TPCC internally signals completion when the TR is submitted to the EDMA_TPTC, potentially before any data has been transferred.
8. The event queue entries can be observed to determine the last few events if there is a system failure (provided the entries were not bypassed).

11.2.6 EDMA Event Map

Events are mapped through DMA Trigger XBAR.

See EDMA XBAR INTRTR0

12 Time Sync

This chapter describes the time sync modules in the device.

12.1 Time Sync Architecture

This section provides a high-level overview of the SoC time synchronization architecture.

12.1.1 Time Sync Architecture Overview

Table 12-1 shows the time synchronization functions supported by the device.

Table 12-1. Time Synchronization Functions in the SOC

Interface	Time Sync Functions	Supported by
PRU-ICSS	IEEE 1588-2008 (1/2-step), 802.1AS, TSN	PRU-ICSS firmware
CPSW0	IEEE 1588-2008 (2-step), 802.1AS	CPTS in CPSW0
EPWMx.SYNCOUT	PWM SYNCOUT	EPWMx.SYNCOUT

Note

These time sync functions are described in detail in each respective chapter.

Any of these functions can be a time sync controller in the system. Sync routers (SOC_TIMESYNC_XBAR0, SOC_TIMESYNC_XBAR1) provide flexibility for each time domain to choose a synch controller independently. In addition these routers also provide selection of sync and compare events for routing as CPU or DMA events.

Figure 12-1 shows a high-level overview of the SoC time sync architecture.

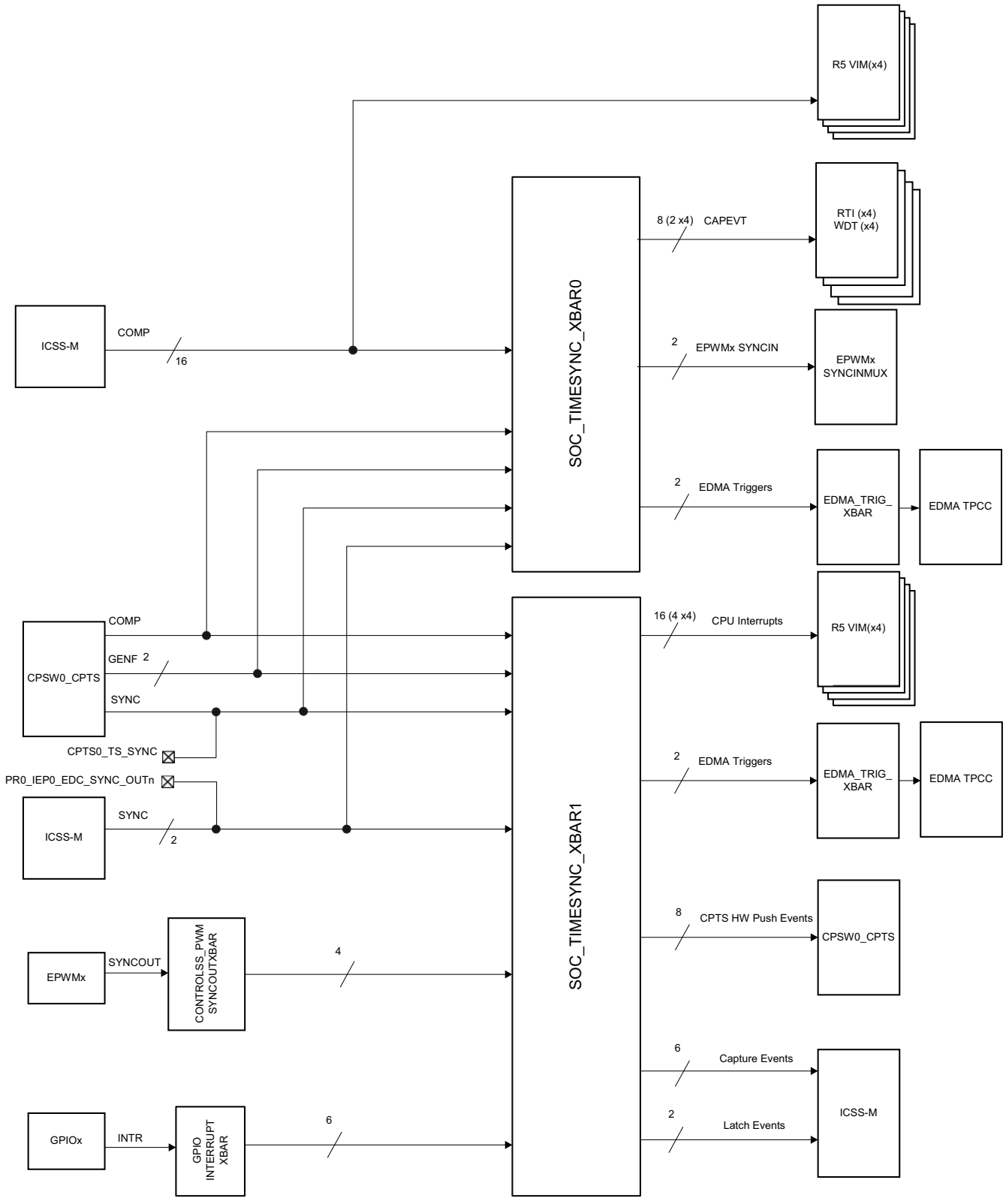


Figure 12-1. SoC Time Sync Architecture

12.2 Time Sync Routers

12.2.1 Time Sync Routers Overview

12.2.1.1 SOC_TIMESYNC_XBAR0 Overview

The Time Sync Event Router0 (**SOC_TIMESYNC_XBAR0**) implements a set of multiplexers to provide selection of various events to EPWM SYNCIN, RTI Capture and DMA Trigger. There is one register per output that controls the selection (SOC_TIMESYNC_XBAR0_MUXCNTL_y).

The SOC_TIMESYNC_XBAR0 module has the following configuration:

- Number of input events: 22
- Number of output events: 12
- Event input type: Pulse

12.2.1.2 SOC_TIMESYNC_XBAR1 Overview

The Time Sync Event Router1 (**SOC_TIMESYNC_XBAR1**) implements a set of multiplexers to provide selection of various events to CPU Interrupts, DMA Triggers, CPTS Push events, ICSS Latch and capture events. There is one register per output that controls the selection (SOC_TIMESYNC_XBAR1_MUXCNTL_y).

The SOC_TIMESYNC_XBAR1 module has the following configuration:

- Number of input events: 16
- Number of output events: 34
- Event input type: Pulse

12.2.2 Time Sync Routers Integration

This section describes the Time Sync Routers integration in the device, including information about clocks, resets, and hardware requests.

12.2.2.1 SOC_TIMESYNC_XBAR0 Integration

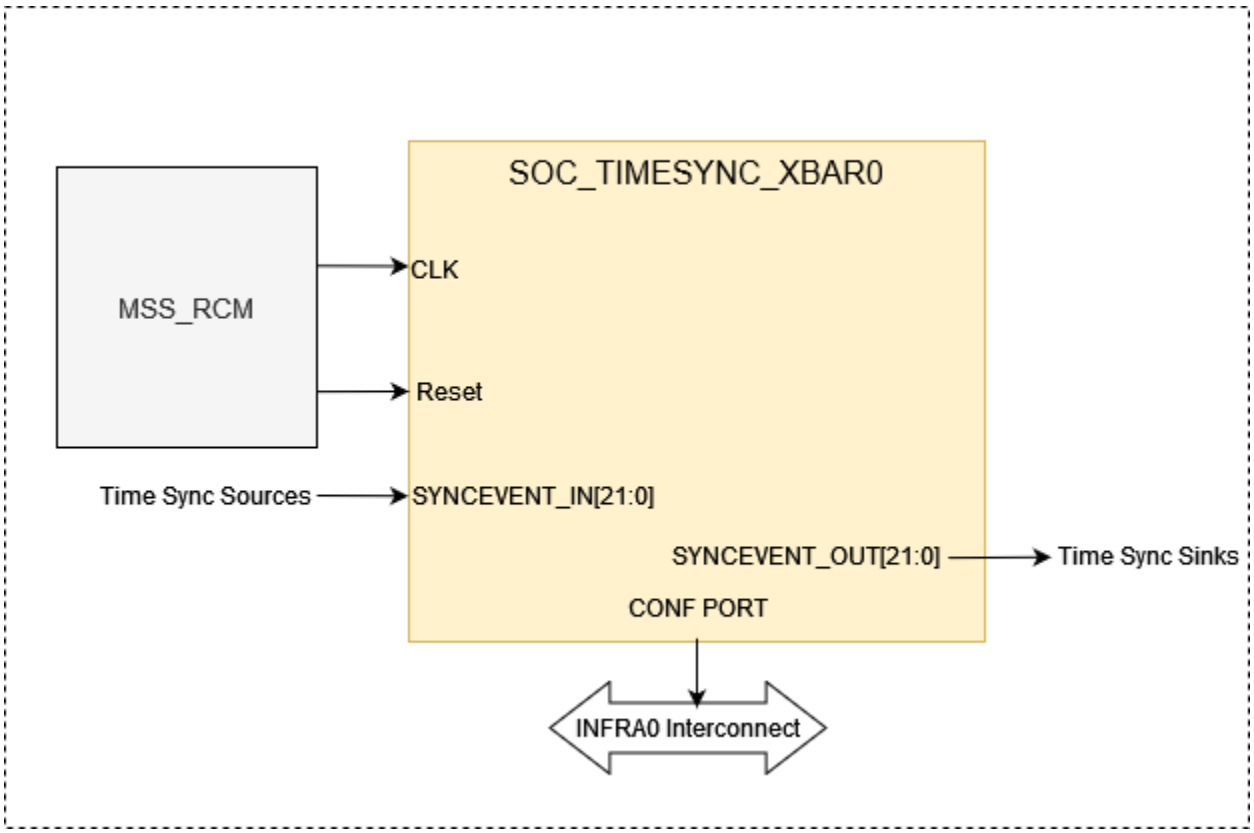


Figure 12-2. SOC_TIMESYNC_XBAR0 Integration

Table 12-2. SOC_TIMESYNC_XBAR0 Device Integration

Module Instance	Device Allocation	SoC Interconnect
SOC_TIMESYNC_XBAR0	✓	VBUSP INFRA0 Interconnect

Table 12-3. SOC_TIMESYNC_XBAR0 Clocks

Module Instance	Module Clock Input	Source Clock Signal	Source	Default Freq	Description
SOC_TIMESYNC_XBAR0	CLK	SYSCLK	MSS_RCM	200 MHz	SOC_TIMESYNC_XBAR0 Functional and Interface clock

Table 12-4. SOC_TIMESYNC_XBAR0 Resets

Module Instance	Module Reset Input	Source Reset Signal	Source	Description
SOC_TIMESYNC_XBAR0	RST	SYS_RST	RCM + Warm Reset Sources	SOC_TIMESYNC_XBAR0 Reset

Table 12-5. SOC_TIMESYNC_XBAR0 Time Sync Output Events

Module Instance	Module Sync Output	Destination Sync Signal	Destination	Type	Description
SOC_TIMESYNC_XBAR0	SYNCEVENT_OUT0	EPWMx_SYNCIN58	EPWMx	Edge	Selectable sync event 0
	SYNCEVENT_OUT1	EPWMx_SYNCIN59	EPWMx		Selectable sync event 1
	SYNCEVENT_OUT2	CAPEVT0	RTI0,WDT0		Selectable sync event 2
	SYNCEVENT_OUT3	CAPEVT1	RTI0,WDT0		Selectable sync event 3
	SYNCEVENT_OUT4	CAPEVT0	RTI1,WDT1		Selectable sync event 4
	SYNCEVENT_OUT5	CAPEVT1	RTI1,WDT1		Selectable sync event 5
	SYNCEVENT_OUT6	CAPEVT0	RTI2,WDT2		Selectable sync event 6

Module Instance	Module Sync Input	TimeSync Event Sources
SOC_TIMESYNC_XBAR0	SYNCEVENT_IN[21:0]	See SOC_TIMESYNC_XBAR0 Event Map table for time sync event mapping.

12.2.2.2 SOC_TIMESYNC_XBAR1 Integration

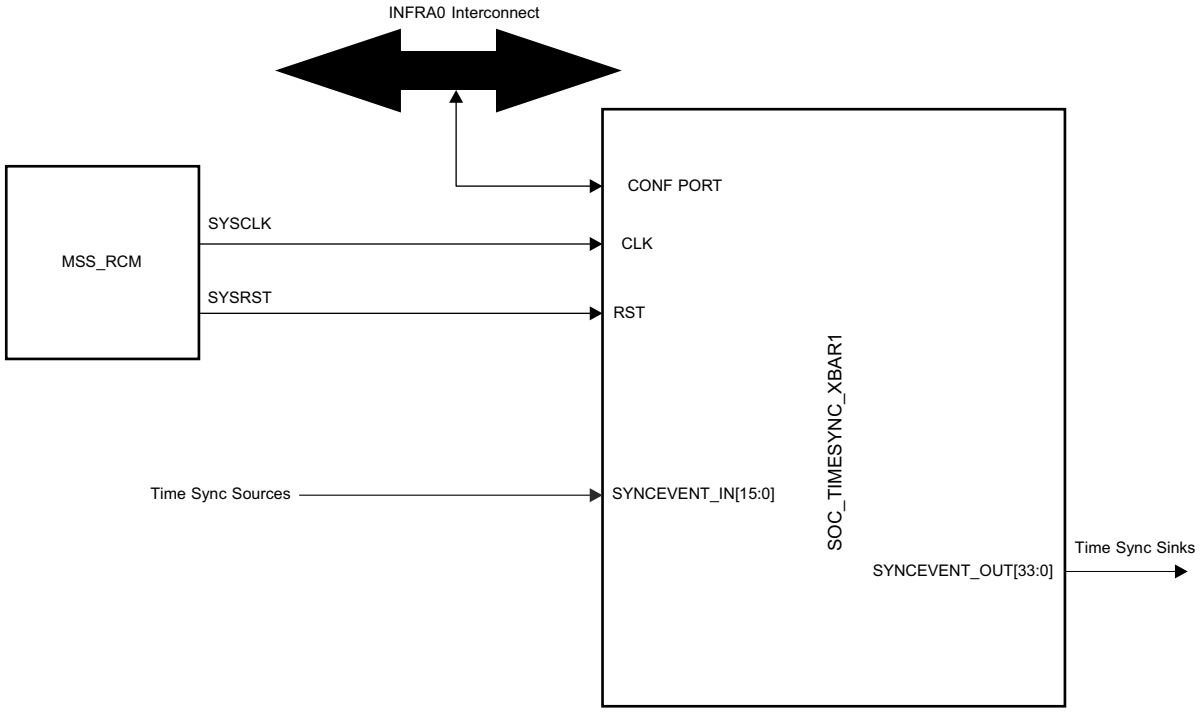


Figure 12-3. SOC_TIMESYNC_XBAR1 Integration

Table 12-6. SOC_TIMESYNC_XBAR1 Device Integration

Module Instance	Device Allocation	SoC Interconnect
SOC_TIMESYNC_XBAR1	✓	VBUSP INFRA Interconnect

Table 12-7. SOC_TIMESYNC_XBAR1 Clocks

Module Instance	Module Clock Input	Source Clock Signal	Source	Default Freq	Description
SOC_TIMESYNC_XBAR1	CLK	SYSCLK	MSS_RCM	200 MHz	SOC_TIMESYNC_XBAR1 Functional and Interface clock

Table 12-8. SOC_TIMESYNC_XBAR1 Resets

Module Instance	Module Reset Input	Source Reset Signal	Source	Description
SOC_TIMESYNC_XBAR1	RST	SYS_RST	RCM + Warm Reset Sources	SOC_TIMESYNC_XBAR1 Reset

Table 12-9. SOC_TIMESYNC_XBAR1 Time Sync Output Events

Module Instance	Module Sync Output	Destination Sync Signal	Destination	Type	Description
SOC_TIMESYNC_XBAR1	SYNCEVENT_OUT0	EDMA_TRIGGERXBAR_IN111	EDMA_TRIGGERXBAR	Edge	Selectablesync event 0
	SYNCEVENT_OUT1	EDMA_TRIGGERXBAR_IN112	EDMA_TRIGGERXBAR		Selectablesync event 1
	SYNCEVENT_OUT2	R5SS0_CORE0_INTR138	R5SS0_CORE0_VIM		Selectablesync event 2
	SYNCEVENT_OUT3	R5SS0_CORE0_INTR139	R5SS0_CORE0_VIM		Selectablesync event 3
	SYNCEVENT_OUT4	R5SS0_CORE0_INTR140	R5SS0_CORE0_VIM		Selectablesync event 4
	SYNCEVENT_OUT5	R5SS0_CORE0_INTR141	R5SS0_CORE0_VIM		Selectablesync event 5
	SYNCEVENT_OUT6	R5SS0_CORE1_INTR138	R5SS0_CORE1_VIM		Selectablesync event 6
	SYNCEVENT_OUT7	R5SS0_CORE1_INTR139	R5SS0_CORE1_VIM		Selectablesync event 7
	SYNCEVENT_OUT8	R5SS0_CORE1_INTR140	R5SS0_CORE1_VIM		Selectablesync event 8
	SYNCEVENT_OUT9	R5SS0_CORE1_INTR141	R5SS0_CORE1_VIM		Selectablesync event 9
	SYNCEVENT_OUT10	ICSSM0_EDC_LATCH0_IN	PRU_ICSSM0		Selectablesync event 10
	SYNCEVENT_OUT11	ICSSM0_EDC_LATCH1_IN	PRU_ICSSM0		Selectablesync event 11
	SYNCEVENT_OUT12	ICSSM0_IEP_CAP_INT R0	PRU_ICSSM0		Selectablesync event 12
	SYNCEVENT_OUT13	ICSSM0_IEP_CAP_INT R1	PRU_ICSSM0		Selectablesync event 13
	SYNCEVENT_OUT14	ICSSM0_IEP_CAP_INT R2	PRU_ICSSM0		Selectablesync event 14
	SYNCEVENT_OUT15	ICSSM0_IEP_CAP_INT R3	PRU_ICSSM0		Selectablesync event 15
	SYNCEVENT_OUT16	ICSSM0_IEP_CAP_INT R4	PRU_ICSSM0		Selectablesync event 16

Table 12-9. SOC_TIMESYNC_XBAR1 Time Sync Output Events (continued)

Module Instance	Module Sync Output	Destination Sync Signal	Destination	Type	Description
SOC_TIMESYNC_XBAR1	SYNCEVENT_OUT17	ICSSM0_IEP_CAP_INT R5	PRU_ICSSM0	Edge	Selectablesync event 17
	SYNCEVENT_OUT18	CPTS_HW1_TS_PUSH	CPSW0_CPTS		Selectablesync event 18
	SYNCEVENT_OUT19	CPTS_HW2_TS_PUSH	CPSW0_CPTS		Selectablesync event 19
	SYNCEVENT_OUT20	CPTS_HW3_TS_PUSH	CPSW0_CPTS		Selectablesync event 20
	SYNCEVENT_OUT21	CPTS_HW4_TS_PUSH	CPSW0_CPTS		Selectablesync event 21
	SYNCEVENT_OUT22	CPTS_HW5_TS_PUSH	CPSW0_CPTS		Selectablesync event 22
	SYNCEVENT_OUT23	CPTS_HW6_TS_PUSH	CPSW0_CPTS		Selectablesync event 23
	SYNCEVENT_OUT24	CPTS_HW7_TS_PUSH	CPSW0_CPTS		Selectablesync event 24
	SYNCEVENT_OUT25	CPTS_HW8_TS_PUSH	CPSW0_CPTS		Selectablesync event 25
	SYNCEVENT_OUT26	R5SS1_CORE0_INTR138	R5SS1_CORE0_VIM		Selectablesync event 26
	SYNCEVENT_OUT27	R5SS1_CORE0_INTR139	R5SS1_CORE0_VIM		Selectablesync event 27
	SYNCEVENT_OUT28	R5SS1_CORE0_INTR140	R5SS1_CORE0_VIM		Selectablesync event 28
	SYNCEVENT_OUT29	R5SS1_CORE0_INTR141	R5SS1_CORE0_VIM		Selectablesync event 29
	SYNCEVENT_OUT30	R5SS1_CORE1_INTR138	R5SS1_CORE1_VIM		Selectablesync event 30
	SYNCEVENT_OUT31	R5SS1_CORE1_INTR139	R5SS1_CORE1_VIM		Selectablesync event 31
	SYNCEVENT_OUT32	R5SS1_CORE1_INTR140	R5SS1_CORE1_VIM		Selectablesync event 32
	SYNCEVENT_OUT33	R5SS1_CORE1_INTR141	R5SS1_CORE1_VIM		Selectablesync event 33

Table 12-10. SOC_TIMESYNC_XBAR1 Time Sync Input Events

Module Instance	Module Sync Input	TimeSync Event Sources
SOC_TIMESYNC_XBAR1	SYNCEVENT_IN[15:0]	See SOC_TIMESYNC_XBAR1 Event Map table for time sync event mapping.

12.2.3 Time Sync Routers Registers

12.2.3.1 SOC_TIMESYNC_XBAR0 Registers

Table 12-11. SOC_TIMESYNC_XBAR0Instances

Instance	BaseAddress
SOC_TIMESYNC_XBAR0	52E00000h

Table 12-12. SOC_TIMESYNC_XBAR0 Registers

Offset	Acronym	Register Name	Physical Address
0h	SOC_TIMESYNC_XBAR0_PID	Peripheral identification register	52E00000h
4h+ (*n**0x4) where *n* goes from 0 -11	SOC_TIMESYNC_XBAR0_MUXCNTL_y	Eventmux control register	52E00004h+ (*n**0x4) where *n* goes from 0 -11

12.2.3.2 SOC_TIMESYNC_XBAR1 Registers

Table 12-13. SOC_TIMESYNC_XBAR1Instances

Instance	Base Address
SOC_TIMESYNC_XBAR1	52E04000h

Table 12-14. SOC_TIMESYNC_XBAR1Registers

Offset	Acronym	RegisterName	Physical Address
0h	SOC_TIMESYNC_XBAR1_PID	Peripheral identification register	52E04000h
4h+ (*n**0x4) where *n* goes from 0 -11	SOC_TIMESYNC_XBAR1_MUXCNTL_y	Eventmux control register	52E04004h+ (*n**0x4) where *n* goes from 0 -33

12.3 Time Sync and Compare Events

12.3.1 TimeSync Event Sources

12.3.1.1 SOC_TIMESYNC_XBAR0 Event Map

Table 12-15 shows the mapping of Events to the SOC_TIMESYNC_XBAR0. The router allows any of the inputs to be routed to the output.

Table 12-15. SOC_TIMESYNC_XBAR0 Events

Input Event	Event #	Event Name	Description	Type
SYNCEVENT_IN0	0	CPSW0_CPTS_COMP	CPTS Compare Event	Pulse
SYNCEVENT_IN1	1	CPSW0_CPTS_GENF0	CPTS Generate Function 0	Pulse
SYNCEVENT_IN2	2	CPSW0_CPTS_GENF1	CPTS Generate Function 1	Pulse
SYNCEVENT_IN3	3	CPSW0_CPTS_SYNC	CPTS SYNC Event	Pulse
SYNCEVENT_IN4	4	ICSSM_EDC_SYNC0	ICSS IEP Sync Event0	Pulse
SYNCEVENT_IN5	5	ICSSM_EDC_SYNC1	ICSS IEP Sync Event1	Pulse
SYNCEVENT_IN6	6	ICSSM_IEP_CMP_EVT0	ICSS IEP Compare Event 0	Pulse
SYNCEVENT_IN7	7	ICSSM_IEP_CMP_EVT1	ICSS IEP Compare Event 1	Pulse
SYNCEVENT_IN8	8	ICSSM_IEP_CMP_EVT2	ICSS IEP Compare Event 2	Pulse
SYNCEVENT_IN9	9	ICSSM_IEP_CMP_EVT3	ICSS IEP Compare Event 3	Pulse
SYNCEVENT_IN10	10	ICSSM_IEP_CMP_EVT4	ICSS IEP Compare Event 4	Pulse
SYNCEVENT_IN11	11	ICSSM_IEP_CMP_EVT5	ICSS IEP Compare Event 5	Pulse
SYNCEVENT_IN12	12	ICSSM_IEP_CMP_EVT6	ICSS IEP Compare Event 6	Pulse
SYNCEVENT_IN13	13	ICSSM_IEP_CMP_EVT7	ICSS IEP Compare Event 7	Pulse
SYNCEVENT_IN14	14	ICSSM_IEP_CMP_EVT8	ICSS IEP Compare Event 8	Pulse
SYNCEVENT_IN15	15	ICSSM_IEP_CMP_EVT9	ICSS IEP Compare Event 9	Pulse
SYNCEVENT_IN16	16	ICSSM_IEP_CMP_EVT10	ICSS IEP Compare Event 10	Pulse
SYNCEVENT_IN17	17	ICSSM_IEP_CMP_EVT11	ICSS IEP Compare Event 11	Pulse
SYNCEVENT_IN18	18	ICSSM_IEP_CMP_EVT12	ICSS IEP Compare Event 12	Pulse
SYNCEVENT_IN19	19	ICSSM_IEP_CMP_EVT13	ICSS IEP Compare Event 13	Pulse
SYNCEVENT_IN20	20	ICSSM_IEP_CMP_EVT14	ICSS IEP Compare Event 14	Pulse
SYNCEVENT_IN21	21	ICSSM_IEP_CMP_EVT15	ICSS IEP Compare Event 15	Pulse

12.3.1.2 SOC_TIMESYNC_XBAR1 Event Map

Table 12-16 shows the mapping of Events to the SOC_TIMESYNC_XBAR1. The router allows any of the inputs to be routed to the output.

Table 12-16. SOC_TIMESYNC_XBAR1 Events

Input Event	Event Number	Event Name	Description	Type
SYNCEVENT_IN0	0	CPSW0_CPTS_COMP	CPTS Compare Event	Pulse
SYNCEVENT_IN1	1	CPSW0_CPTS_GENF0	CPTS Generate Function 0	Pulse
SYNCEVENT_IN2	2	CPSW0_CPTS_GENF1	CPTS Generate Function 1	Pulse
SYNCEVENT_IN3	3	CPSW0_CPTS_SYNC	CPTS SYNC Event	Pulse
SYNCEVENT_IN4	4	ICSSM_EDC_SYNC0	ICSS IEP Sync Event0	Pulse
SYNCEVENT_IN5	5	ICSSM_EDC_SYNC1	ICSS IEP Sync Event1	Pulse
SYNCEVENT_IN6	6	CONTROLSS_PWMSYNCOUT XBAR_OUT0	EPWMSYNCOUT0 from PWMSYNCOUTXBAR	Pulse
SYNCEVENT_IN7	7	CONTROLSS_PWMSYNCOUT XBAR_OUT1	EPWMSYNCOUT1 from PWMSYNCOUTXBAR	Pulse
SYNCEVENT_IN8	8	CONTROLSS_PWMSYNCOUT XBAR_OUT2	EPWMSYNCOUT2 from PWMSYNCOUTXBAR	Pulse

Table 12-16. SOC_TIMESYNC_XBAR1 Events (continued)

Input Event	Event Number	Event Name	Description	Type
SYNCEVENT_IN9	9	CONTROLSS_PWMSYNCOUT XBAR_OUT3	EPWMSYNCOUT3 from PWMSYNCOUTXBAR	Pulse
SYNCEVENT_IN10	10	GPIO_INTER_XBAR_OUT8	GPIOINTERRUPT8 from GPIOINTXBAR	Pulse
SYNCEVENT_IN11	11	GPIO_INTER_XBAR_OUT9	GPIOINTERRUPT9 from GPIOINTXBAR	Pulse
SYNCEVENT_IN12	12	GPIO_INTER_XBAR_OUT10	GPIOINTERRUPT10 from GPIOINTXBAR	Pulse
SYNCEVENT_IN13	13	GPIO_INTER_XBAR_OUT11	GPIOINTERRUPT11 from GPIOINTXBAR	Pulse
SYNCEVENT_IN14	14	GPIO_INTER_XBAR_OUT12	GPIOINTERRUPT12 from GPIOINTXBAR	Pulse
SYNCEVENT_IN15	15	GPIO_INTER_XBAR_OUT13	GPIOINTERRUPT13 from GPIOINTXBAR	Pulse

12.3.1.3 PRU-ICSS Event Map

Table 12-17 shows the mapping of events to the PRU-ICSS Latch and Compare inputs.

Table 12-17. PRU-ICSS Event Map

Input Event	Event Name	Description	Type
ICSSM0_EDC_LATCH0_IN	SOC_TIMESYNC_XBAR1_SYNCEVENT_OUT10	Selectable sync event 10	Edge
ICSSM0_EDC_LATCH1_IN	SOC_TIMESYNC_XBAR1_SYNCEVENT_OUT11	Selectable sync event 11	Edge
ICSSM0_IEP_CAP_INTR0	SOC_TIMESYNC_XBAR1_SYNCEVENT_OUT12	Selectable sync event 12	Pulse
ICSSM0_IEP_CAP_INTR1	SOC_TIMESYNC_XBAR1_SYNCEVENT_OUT13	Selectable sync event 13	Pulse
ICSSM0_IEP_CAP_INTR2	SOC_TIMESYNC_XBAR1_SYNCEVENT_OUT14	Selectable sync event 14	Pulse
ICSSM0_IEP_CAP_INTR3	SOC_TIMESYNC_XBAR1_SYNCEVENT_OUT15	Selectable sync event 15	Pulse
ICSSM0_IEP_CAP_INTR4	SOC_TIMESYNC_XBAR1_SYNCEVENT_OUT16	Selectable sync event 16	Pulse
ICSSM0_IEP_CAP_INTR5	SOC_TIMESYNC_XBAR1_SYNCEVENT_OUT17	Selectable sync event 17	Pulse

12.3.1.4 CPSW0_CPTS Event Map

Table 12-18 shows the mapping of events to the CPSW0_CPTS Hardware push inputs.

Table 12-18. CPSW0_CPTS Event Map

Input Event	Event Name	Description	Type
CPTS_HW1_TS_PUSH	SOC_TIMESYNC_XBAR1_SYNCEVENT_OUT18	Selectable sync event 18	Pulse
CPTS_HW2_TS_PUSH	SOC_TIMESYNC_XBAR1_SYNCEVENT_OUT19	Selectable sync event 19	Pulse
CPTS_HW3_TS_PUSH	SOC_TIMESYNC_XBAR1_SYNCEVENT_OUT20	Selectable sync event 20	Pulse
CPTS_HW4_TS_PUSH	SOC_TIMESYNC_XBAR1_SYNCEVENT_OUT21	Selectable sync event 21	Pulse
CPTS_HW5_TS_PUSH	SOC_TIMESYNC_XBAR1_SYNCEVENT_OUT22	Selectable sync event 22	Pulse
CPTS_HW6_TS_PUSH	SOC_TIMESYNC_XBAR1_SYNCEVENT_OUT23	Selectable sync event 23	Pulse
CPTS_HW7_TS_PUSH	SOC_TIMESYNC_XBAR1_SYNCEVENT_OUT24	Selectable sync event 24	Pulse
CPTS_HW8_TS_PUSH	SOC_TIMESYNC_XBAR1_SYNCEVENT_OUT25	Selectable sync event 25	Pulse

13 Peripherals

This chapter describes the various peripheral modules instantiated in the device.

13.1 General Connectivity Peripherals

This section describes the general connectivity peripherals in the device.

13.1.1 General-Purpose Interface (GPIO)

This chapter describes the General-Purpose Input/Output (GPIO) for the device.

13.1.1.1 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.

The device has four instances of the GPIO module, one per R5FSS processor core. 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.

Note

Out of the 144 available GPIOs, only 139 GPIOs were connected to PADs and 5 GPIO Pins are grounded.

Table 13-1 shows GPIO modules allocation within device domains.

Table 13-1. GPIO Modules Allocation within Device Domains

Module Instance	Device
GPIO0	✓ (R5FSS0-CORE0)
GPIO1	✓ (R5FSS0-CORE1)
GPIO2	✓ (R5FSS1-CORE0)
GPIO3	✓ (R5FSS1-CORE1)

13.1.1.2 GPIO Environment

The GPIO[0-3] modules are hereinafter referred to as GPIO module.

This section describes the GPIO external connections (environment).

The general-purpose interface combines four GPIO modules for a flexible, user-programmable, general-purpose input/output (I/O) controller. The general-purpose interface implements functions that are not implemented with the dedicated controllers in the device and require simple input and/or output software-controlled signals. The GPIO allows a variety of custom connections and expands the I/O capabilities of the system to the real world.

The general-purpose interface can physically connect the device to a keyboard matrix and peripheral integrated circuits (ICs).

13.1.1.3 GPIO Integration

There are 4x GPIO modules integrated in the device. The diagram below provides a visual representation of the device integration details.

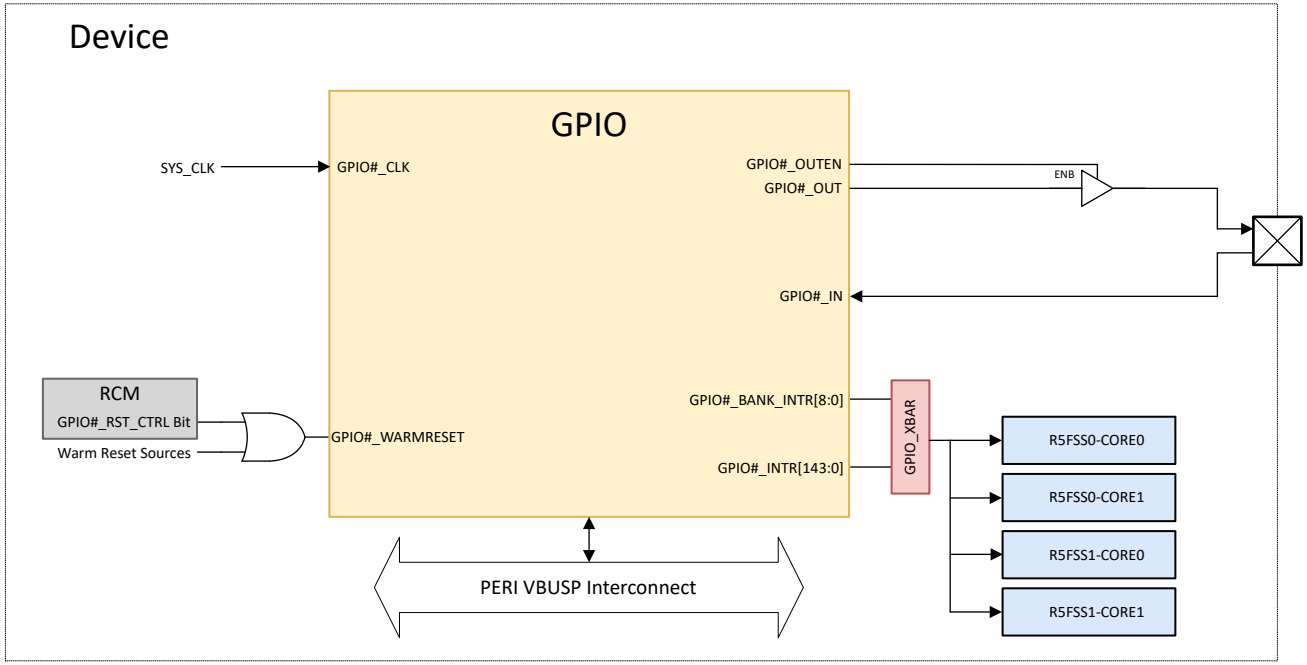


Figure 13-1. GPIO Integration Diagram

Note

There is a designated GPIO module per R5FSS core. Each R5FSS core has access to all GPI signals. The GPO signals can be assigned to a specific R5FSS core by configuring the MSS_IOMUX.*PAD*_CFG_REG.GPIO_SEL[17:16] of the associated IOMUX Pad Configuration register.

This diagram describes the GPIO multiplexor connectivity.

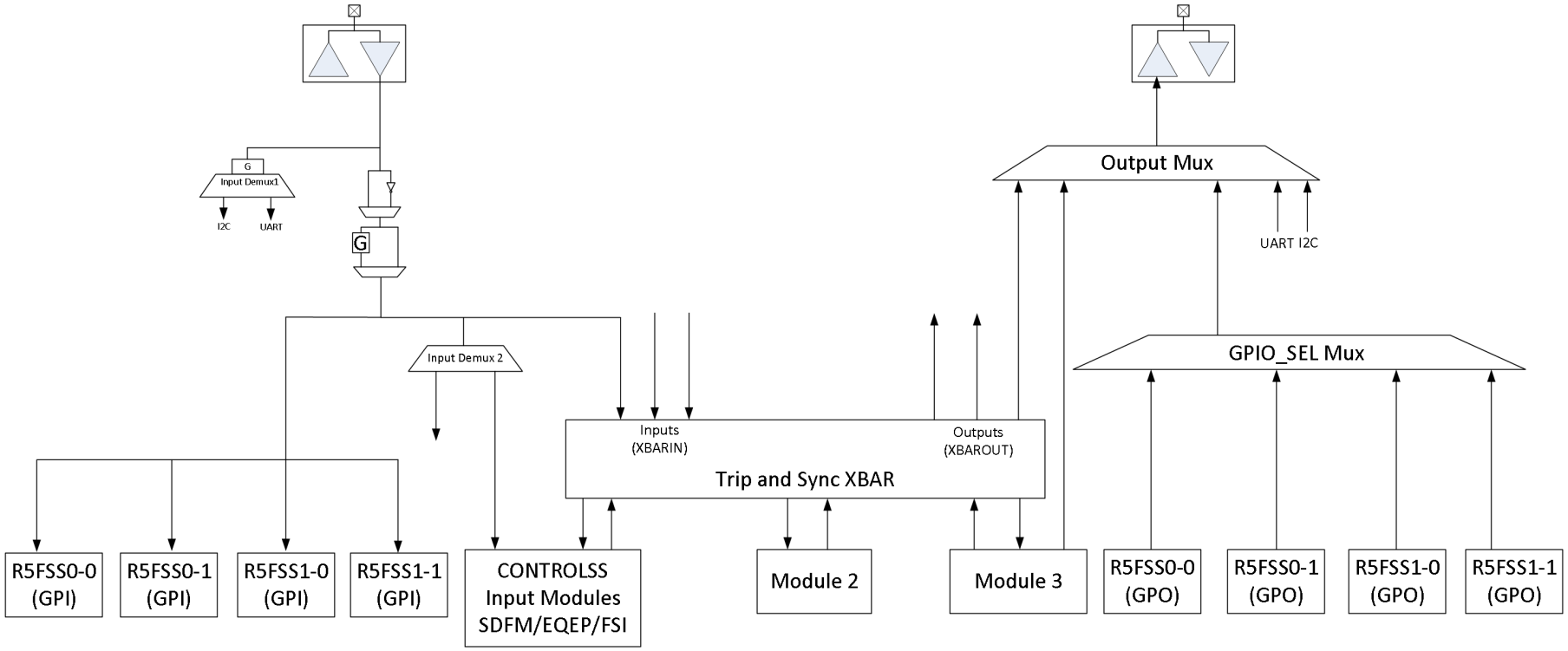


Figure 13-2. GPIO Mux Diagram

The tables below summarize the device integration details of GPIO# (where # = 0 to 4).

Table 13-2. *GPIO* Device Integration

This table describes the module device integration details.

Module Instance	Device Allocation	SoC Interconnect
GPIO0	✓	PERI VBUSP Interconnect
GPIO1	✓	PERI VBUSP Interconnect
GPIO2	✓	PERI VBUSP Interconnect
GPIO3	✓	PERI VBUSP Interconnect

Table 13-3. *GPIO* Clocks

This table describes the module clocking signals.

Module Instance	Module Clock Input	Source Clock Signal	Source	Default Freq	Description
GPIO0	GPIO0_VBUS_FICLK	SYS_CLK	PLL_CORE_CLK:HSDIV0_CLKOUT 0	200 MHz	GPIO0 Functional and Interface Clock
GPIO1	GPIO1_VBUS_FICLK	SYS_CLK	PLL_CORE_CLK:HSDIV0_CLKOUT 0	200 MHz	GPIO1 Functional and Interface Clock
GPIO2	GPIO2_VBUS_FICLK	SYS_CLK	PLL_CORE_CLK:HSDIV0_CLKOUT 0	200 MHz	GPIO2 Functional and Interface Clock
GPIO3	GPIO3_VBUS_FICLK	SYS_CLK	PLL_CORE_CLK:HSDIV0_CLKOUT 0	200 MHz	GPIO3 Functional and Interface Clock

Table 13-4. *GPIO* Resets

This table describes the module reset signals.

Module Instance	Module Reset Input	Source Reset Signal	Source	Description
GPIO0	GPIO0_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Source	GPIO0 Reset
GPIO1	GPIO1_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Source	GPIO1 Reset
GPIO2	GPIO2_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Source	GPIO2 Reset
GPIO3	GPIO3_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Source	GPIO3 Reset

Table 13-5. *GPIO* Interrupt Requests

This table describes the module interrupt requests.

Module Instance	Module Interrupt Signal	Destination Interrupt Input	Destination	Type	Description
GPIO#	GPIO#_[0:137]	Programmable via GPIO_XBAR_INTR0	GPIO_XBAR_INTR0	Pulse	GPIO#_[0:137] interrupt request
GPIO#	GPIO#_BANK0_INT	Programmable via GPIO_XBAR_INTR0	GPIO_XBAR_INTR0	Pulse	GPIO# BANK0 interrupt request
GPIO#	GPIO#_BANK1_INT	Programmable via GPIO_XBAR_INTR0	GPIO_XBAR_INTR0	Pulse	GPIO# BANK1 interrupt request
GPIO#	GPIO#_BANK2_INT	Programmable via GPIO_XBAR_INTR0	GPIO_XBAR_INTR0	Pulse	GPIO# BANK2 interrupt request

Table 13-5. *GPIO* Interrupt Requests (continued)

This table describes the module interrupt requests.

Module Instance	Module Interrupt Signal	Destination Interrupt Input	Destination	Type	Description
GPIO#	GPIO#_BANK3_INT	Programmable via GPIO_XBAR_INTR0	GPIO_XBAR_INTR0	Pulse	GPIO# BANK3 interrupt request
GPIO#	GPIO#_BANK4_INT	Programmable via GPIO_XBAR_INTR0	GPIO_XBAR_INTR0	Pulse	GPIO# BANK4 interrupt request
GPIO#	GPIO#_BANK5_INT	Programmable via GPIO_XBAR_INTR0	GPIO_XBAR_INTR0	Pulse	GPIO# BANK5 interrupt request
GPIO#	GPIO#_BANK6_INT	Programmable via GPIO_XBAR_INTR0	GPIO_XBAR_INTR0	Pulse	GPIO# BANK6 interrupt request
GPIO#	GPIO#_BANK7_INT	Programmable via GPIO_XBAR_INTR0	GPIO_XBAR_INTR0	Pulse	GPIO# BANK7 interrupt request
GPIO#	GPIO#_BANK8_INT	Programmable via GPIO_XBAR_INTR0	GPIO_XBAR_INTR0	Pulse	GPIO# BANK8 interrupt request

Note

Where # = 0 to 3

Table 13-6. *GPIO* DMA Requests

This table describes the module DMA requests.

Module Instance	Module DMA Event	Destination DMA Event Input	Destination	Type	Description
GPIO#	N/A	N/A	N/A	N/A	The GPIO module does not support DMA requests.

Table 13-7. *GPIO* Capture Event Inputs

This table describes the module capture event inputs.

Module Instance	Module Capture Event Input	Capture Event Source Signal	Source	Type	Description
GPIO#	N/A	N/A	N/A	N/A	The GPIO module does not support Capture Event Inputs

Note

For more information on the interconnects, see the *System Interconnect* chapter.

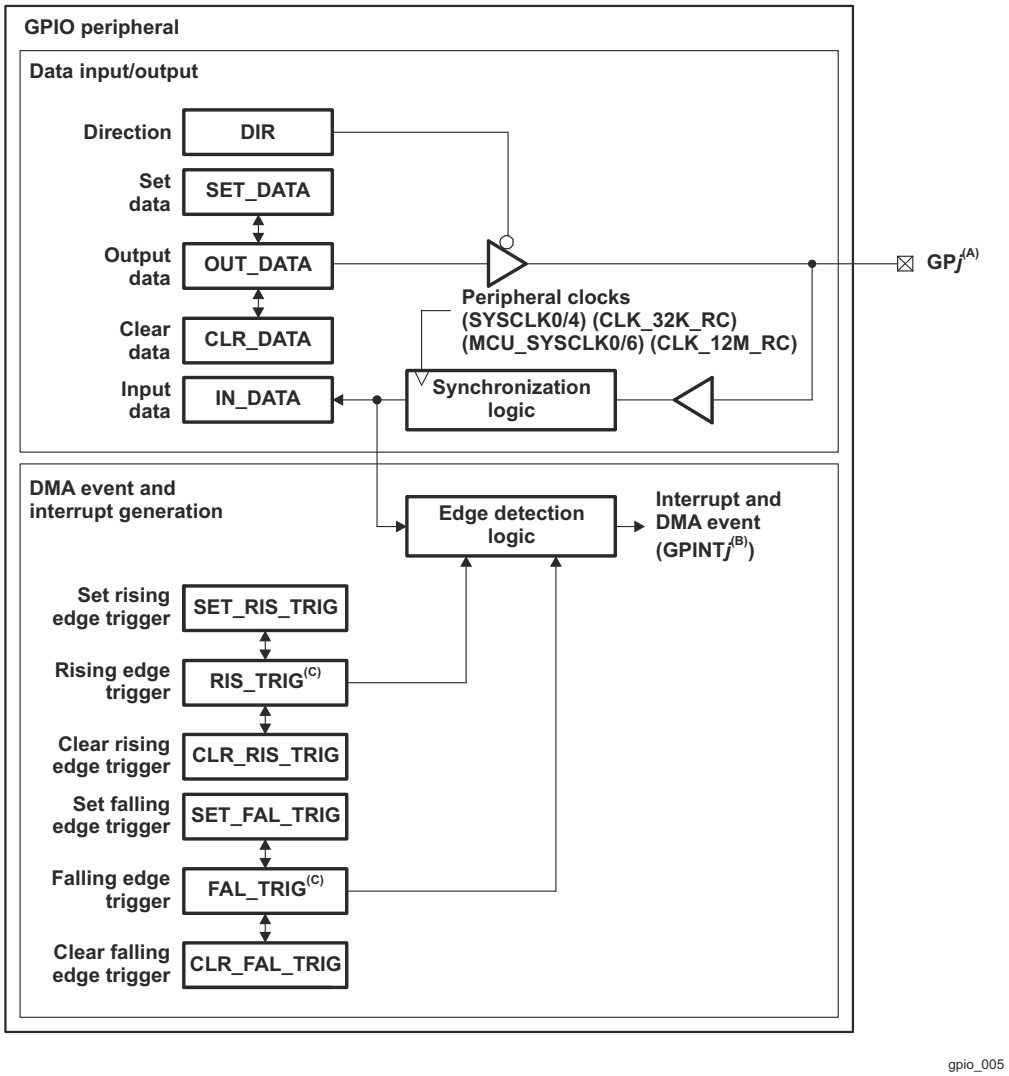
For more information on power, reset, and clock management, see the corresponding sections within the *Device Configuration* chapter.

For more information on the device interrupt controllers, see the *Interrupt Controllers* chapter.

13.1.1.4 GPIO Functional Description

13.1.1.4.1 GPIO Block Diagram

Figure 13-3 shows the general-purpose interface block diagram.



A. Where j = [0:143]
B. Some of the GPj pins are muxed with other device signals. For details, see the device-specific Datasheet.
C. All GPINTj can be used as host CPU interrupts via GPIO XBAR and synchronization events to the DMA.
D. The RIS_TRIG and FAL_TRIG registers are internal to the GPIO module and are not visible to the host CPU.

Figure 13-3. GPIO Block Diagram

13.1.1.4.2 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.
- 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.

13.1.1.4.3 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 and can be routed as a DMA event through the GPIO XBAR. 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.

13.1.1.4.3.1 Interrupt Enable (per Bank)

The GPIO_BINTEN register provides interrupt enable/disable feature for each bank of 16 GPINT signals.

13.1.1.4.3.2 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 13-8 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 13-8. 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

Table 13-8. GPIO Interrupt and DMA Event Configuration Options (continued)

RIS_TRIG Bit n	FAL_TRIG Bit n	Host CPU Interrupt and DMA Event Generation
1	1	GPINTj interrupt and DMA event is triggered on both rising and falling edge of GPj signal

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.

13.1.1.4.3.3 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.

13.1.1.4.4 Input Qualification

The input qualification scheme has been designed to be very flexible. Select the type of input qualification for each GPIO pin by configuring the MSS_IOMUX_<PAD>_CFG_REG[QUAL_SEL] registers. In the case of a GPIO input pin, the qualification can be specified as only synchronized to SYSCLK or qualification by a sampling window. For pins that are configured as peripheral inputs, the input can also be asynchronous in addition to synchronized to SYSCLK or qualified by a sampling window. The remainder of this section describes the options available.

13.1.1.4.4.1 No Synchronization (Asynchronous Input)

This mode is used for peripherals where input synchronization is not required or the peripheral performs the synchronization. Examples include communication ports SPI, and I^2C.

Note

Using input synchronization when the peripheral performs the synchronization can cause unexpected results. The user must make sure that the GPIO pin is configured for asynchronous in this case.

13.1.1.4.4.2 Synchronization to SYSCLK Only

This is the default qualification mode of all the pins at reset. In this mode, the input signal is only synchronized to the system clock (SYSCLK). Because the incoming signal is asynchronous, a SYSCLK period of delay is needed for the input to the device to be changed. No further qualification is performed on the signal.

13.1.1.4.4.3 Qualification Using a Sampling Window

In this mode, the signal is first synchronized to the system clock (SYSCLK) and then qualified by a specified number of cycles before the input is allowed to change. Figure 13-4 and Figure 13-5 show how the input qualification is performed to eliminate unwanted noise. Two parameters are specified by the user for this type of qualification: 1) the sampling period, or how often the signal is sampled, and 2) the number of samples to be taken.

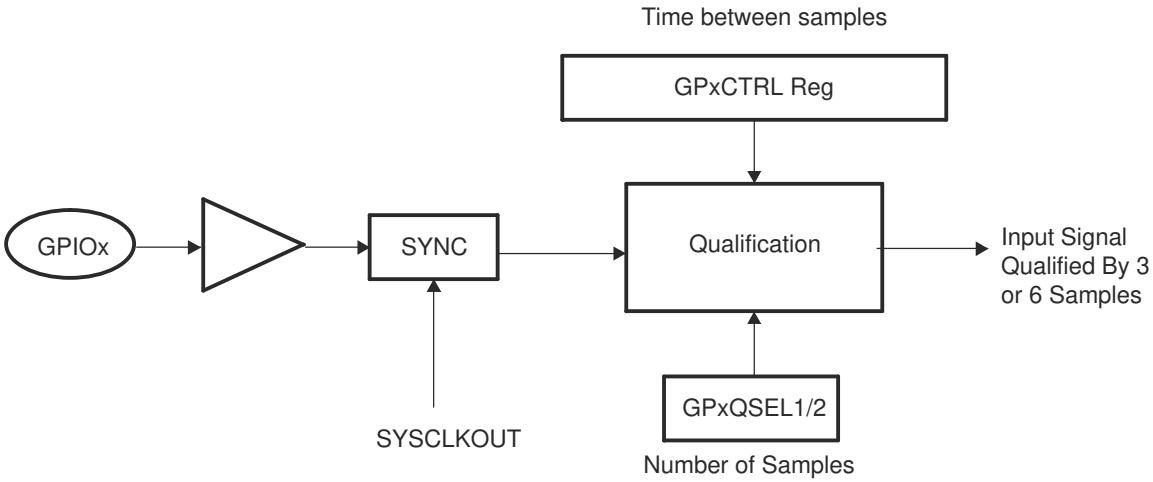


Figure 13-4. Input Qualification Using a Sampling Window

Note

For AM263Px, GPxCTRL Reg = MSS_IOMUX.**QUAL_GRP_*_CFG_REG**[QUAL_PERIOD_PER_SAMPLE] and GPxQSEL 1/2 = IOMUX.*_CFG_REG[QUAL_SEL] in the diagram above.

Time between samples (sampling period):

To qualify the signal, the input signal is sampled at a regular period. The sampling period is specified by the user and determines the time duration between samples, or how often the signal is sampled, relative to the CPU clock (SYSCLK).

The sampling period is specified by the qualification period (QUALPRDn) bits in the GPxCTRL register. The sampling period is configurable in groups of 8 input signals. For example, GPIO0 to GPIO7 use MSS_IOMUX.**QUAL_GRP_0_CFG_REG**[QUAL_PERIOD_PER_SAMPLE] setting and GPIO8 to GPIO15 use MSS_IOMUX.**QUAL_GRP_1_CFG_REG**[QUAL_PERIOD_PER_SAMPLE]. Table 13-9 and Table 13-10 show the relationship between the sampling period or sampling frequency and the MSS_IOMUX.**QUAL_GRP_*_CFG_REG**[QUAL_PERIOD_PER_SAMPLE] setting.

Table 13-9. Sampling Period

	Sampling Period
If IOMUX.**QUAL_GRP_*_CFG_REG**[QUAL_PERIOD_PER_SAMPLE] = 0	1 × T_{SYSCLK}
If IOMUX.**QUAL_GRP_*_CFG_REG**[QUAL_PERIOD_PER_SAMPLE] ≠ 0	2 × IOMUX.**QUAL_GRP_*_CFG_REG**[QUAL_PERIOD_PER_SAMPLE] × T_{SYSCLK}
	Where T_{SYSCLK} is the period in time of SYSCLK

Table 13-10. Sampling Frequency

	Sampling Frequency
If IOMUX.**QUAL_GRP_*_CFG_REG**[QUAL_PERIOD_PER_SAMPLE] = 0	f_{SYSCLK}
If IOMUX.**QUAL_GRP_*_CFG_REG**[QUAL_PERIOD_PER_SAMPLE] ≠ 0	$f_{SYSCLK} \times 1 \div (2 \times$ IOMUX.**QUAL_GRP_*_CFG_REG**[QUAL_PERIOD_PER_SAMPLE])
	Where f_{SYSCLK} is the frequency of SYSCLK

From these equations, the minimum and maximum time between samples can be calculated for a given SYSCLK frequency:

Example: Maximum Sampling Frequency:

If IOMUX.**QUAL_GRP_*_CFG_REG**[QUAL_PERIOD_PER_SAMPLE] = 0

then the sampling frequency is f_{SYSCLK}

If, for example, f_{SYSCLK} = 200MHz

then the signal is sampled at 200MHz or one sample every 5ns.

Example: Minimum Sampling Frequency:

If IOMUX.**QUAL_GRP_*_CFG_REG**[QUAL_PERIOD_PER_SAMPLE] = 0xFF (255)

then the sampling frequency is $f_{SYSCLK} \times 1 \div (2 \times$ IOMUX.**QUAL_GRP_*_CFG_REG**[QUAL_PERIOD_PER_SAMPLE])

If, for example, f_{SYSCLK} = 200MHz

then the signal is sampled at 200MHz × 1 ÷ (2 × 255) (392.157kHz) or one sample every 2.5µs.

Number of samples:

The number of times the signal is sampled is either three samples or six samples as specified in the qualification selection IOMUX.*_CFG_REG[QUAL_SEL] registers. When three or six consecutive cycles are the same, then the input change is passed through to the device.

Total Sampling-Window Width:

The sampling window is the time during which the input signal is sampled as shown in Figure 13-5. By using the equation for the sampling period, along with the number of samples to be taken, the total width of the window can be determined.

For the input qualifier to detect a change in the input, the level of the signal must be stable for the duration of the sampling-window width or longer.

The number of sampling periods within the window is always one less than the number of samples taken. For a three-sample window, the sampling-window width is two sampling-periods wide where the sampling period is defined in Table 13-9. Likewise, for a six-sample window, the sampling-window width is five sampling-periods wide. and show the calculations used to determine the total sampling-window width based on IOMUX.**QUAL_GRP_*_CFG_REG**[QUAL_PERIOD_PER_SAMPLE] and the number of samples taken.

Table 13-11. Case 1: Three-Sample Sampling-Window Width

	Total Sampling-Window Width
If IOMUX.**QUAL_GRP_*_CFG_REG**[QUAL_PERIOD_PER_SAMPLE] = 0	2 × T_{SYSCLK}
If IOMUX.**QUAL_GRP_*_CFG_REG**[QUAL_PERIOD_PER_SAMPLE] ≠ 0	2 × 2 × IOMUX.**QUAL_GRP_*_CFG_REG**[QUAL_PERIOD_PER_SAMPLE] × T_{SYSCLK}
	Where T_{SYSCLK} is the period in time of SYSCLK

Table 13-12. Case 2: Six-Sample Sampling-Window Width

	Total Sampling-Window Width
If IOMUX.**QUAL_GRP_*_CFG_REG**[QUAL_PERIOD_PER_SAMPLE] = 0	5 × T_{SYSCLK}
If IOMUX.**QUAL_GRP_*_CFG_REG**[QUAL_PERIOD_PER_SAMPLE] ≠ 0	5 × 2 × IOMUX.**QUAL_GRP_*_CFG_REG**[QUAL_PERIOD_PER_SAMPLE] × T_{SYSCLK}

Table 13-12. Case 2: Six-Sample Sampling-Window Width (continued)

Total Sampling-Window Width
Where T_{SYSCLK} is the period in time of SYSCLK

Note

The external signal change is asynchronous with respect to both the sampling period and SYSCLK. Due to the asynchronous nature of the external signal, the input must be held stable for a time greater than the sampling-window width to make sure the logic detects a change in the signal. The extra time required can be up to an additional sampling period + T_{SYSCLK}.

The required duration for an input signal to be stable for the qualification logic to detect a change is described in the data sheet.

Example Qualification Window:

For the example shown in Figure 13-5, the input qualification has been configured as follows:

- IOMUX.*_CFG_REG[QUAL_SEL] = 1,0. This indicates a six-sample qualification.
- IOMUX.**QUAL_GRP_*_CFG_REG**[QUAL_PERIOD_PER_SAMPLE] = 1. The sampling period is $t_w(SP)$ = 2 ×IOMUX.**QUAL_GRP_*_CFG_REG**[QUAL_PERIOD_PER_SAMPLE] × T_{SYSCLK} = 2 x T_{SYSCLK}.

This configuration results in the following:

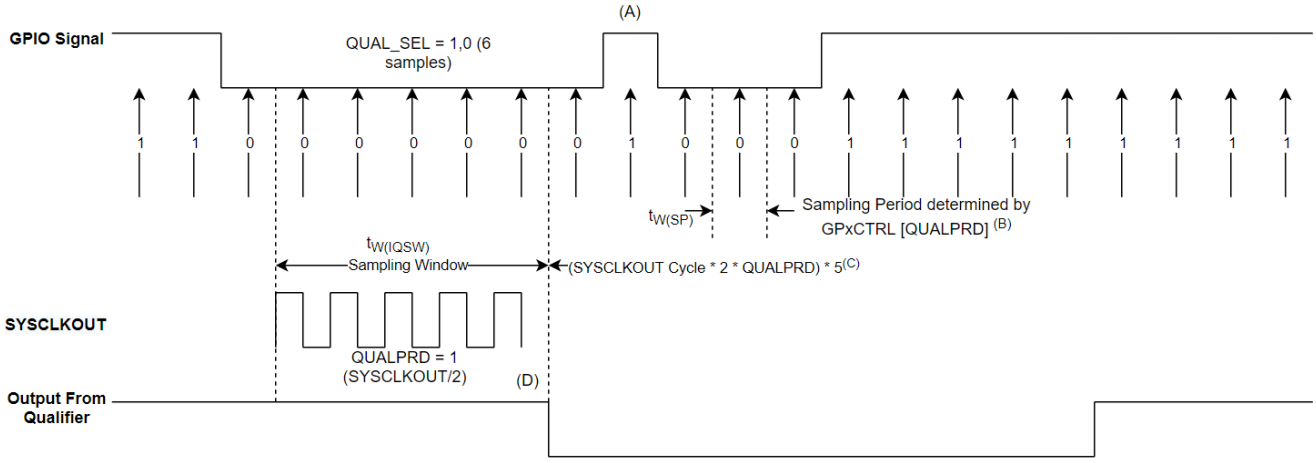
- The width of the sampling window is:

  $t_w(IQSW) = 5 \times t_w(SP) = 5 \times 2 \times$ IOMUX.**QUAL_GRP_*_CFG_REG**[QUAL_PERIOD_PER_SAMPLE] × $T_{SYSCLK} = 5 \times 2 \times T_{SYSCLK}$
- If, for example, T_{SYSCLK} = 5ns, then the duration of the sampling window is:

  Sampling period, $t_w(SP) = 2 \times T_{SYSCLK} = 2 \times 5ns = 10ns$

  Sampling window, $t_w(IQSW) = 5 \times t_w(SP) = 5 \times 5ns = 25ns$
- To account for the asynchronous nature of the input relative to the sampling period and SYSCLK, up to a single additional sampling period and SYSCLK period is required to detect a change in the input signal. For this example:

  $t_w(IQSW) + t_w(SP) + T_{SYSCLK} = 25ns + 10ns + 5ns = 40ns$
- In Figure 13-5, the glitch (A) is shorter then the qualification window and is ignored by the input qualifier.



A. This glitch will be ignored by the input qualifier. The QUALPRD bit field specifies the qualification sampling period. It can vary from 00 to 0xFF. If QUALPRD = 00, then the sampling period is 1 SYSCLKOUT cycle. For any other value "n", the qualification sampling period in 2n SYSCLKOUT cycles (i.e., at every 2n SYSCLKOUT cycles, the GPIO pin will be sampled).
B. The qualification period selected via the GPxCTRL register applies to groups of 8 GPIO pins.
C. The qualification block can take either three or six samples. The QUAL_SEL Register selects which sample mode is used.
D. In the example shown, for the qualifier to detect the change, the input should be stable for 10 SYSCLKOUT cycles or greater. In other words, the inputs should be stable for (5 x QUALPRD x 2) SYSCLKOUT cycles. That would ensure 5 sampling periods for detection to occur. Since external signals are driven asynchronously, an 13-SYSCLKOUT-wide pulse ensures reliable recognition.

Figure 13-5. Input Qualifier Clock Cycles

Note

For AM263Px, SYSCLKOUT = SYSCLK in the diagram above.

13.1.1.4.5 GPIO Emulation Halt Operation

The GPIO peripheral is not affected by emulation halts.

13.1.1.5 GPIO Programming Guide

13.1.1.5.1 GPIO Low-Level Programming Models

13.1.1.5.1.1 Global Initialization

13.1.1.5.1.1.1 GPIO Module Global Initialization

This procedure initializes the general-purpose Interface module after a power-on reset (POR) or software reset.

Table 13-13. GPIO Global Initialization

Step	Register/Bit Field/Programming Model	Value
Configure GPIO channels as input or output of the corresponding bank	DIR	-h
Interrupt requests configuration		
Configure detection events	SET_RIS_TRIG and/or SET_FAL_TRIG	-h
Clear interrupt status	INTSTAT	FFFFh
Enable interrupts for desired banks [0:8]	GPIO_BINTEN[8-0]	-h

13.1.1.5.1.2 GPIO Operational Modes Configuration

13.1.1.5.1.2.1 GPIO Read Input Register

Table 13-14. 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

13.1.1.5.1.2.2 GPIO Set Bit Function

Table 13-15. 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

13.1.1.5.1.2.3 GPIO Clear Bit Function

Table 13-16. 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

13.1.2 Inter-Integrated Circuit (I2C) Interface

This section describes the Inter-Integrated Circuit (I2C) module in the device.

13.1.2.1 I2C Overview

The I2C module is a serial bus that supports multiple controller devices. In multicontroller mode, one or more devices can be connected to the same bus and are capable of controlling the bus. Each I2C device on the bus is recognized by a unique address and can operate as either a transmitter or a receiver, depending on the function of the device. In addition to being a transmitter or receiver, a device connected to the I2C bus can also be considered a controller or a peripheral when performing data transfers.

Note

A controller device is the device that initiates the data transfer on a bus and generates the clock signal that permits the transfer. During the transmission, any device addressed by the controller is considered the peripheral.

Data is communicated to devices interfacing to the I2C module using the serial data pin (SDA) and the serial clock pin (SCL) as shown in Figure 13-6. These two wires carry information between the device and the other devices connected to the I2C bus. Both SDA and SCL pins on the device are bidirectional. They must be connected to a positive supply voltage through a pull-up resistor. When the bus is free, both pins are high. The driver of these two pins has an open-drain configuration to perform the wired-AND function.

The device has a special mode that can be entered to ignore a NACK generated from non-compliant I2C devices that are incapable of generating an ACK.

The I2C module consists of the following primary blocks:

- A serial Interface: one data pin (SDA) and one clock pin (SCL)
- The device register interface:
  - Data registers to temporarily hold received data and transmitted data traveling between the SDA pin and the CPU or the DMA
  - Control and status registers
- A prescaler to divide down the input clock that is driven to the I2C module
- A peripheral bus interface to enable the CPU and DMA to access the I2C module registers
- An arbitrator to handle arbitration between the I2C module (when configured as a controller) and another controller
- Interrupt generation logic (interrupts can be sent to the CPU)
- A clock synchronizer that synchronizes the I2C input clock (from the system module) and the clock on the SCL pin, and synchronizes data transfers with controllers of different clock speeds.
- A noise filter on each of the two serial pins
- DMA event generation logic that synchronizes data reception and data transmission in the I2C module for DMA transmission

In Figure 13-6, the CPU or the DMA writes data for transmission to ICDXR and reads received data from ICDRR. When the I2C module is configured as a transmitter, data written to ICDXR is copied to ICXSR and shifted out one bit at a time. When the I2C module is configured as a receiver, received data is shifted into ICRSR and then copied to ICDRR.

When the I2C function is not needed, the pins may be controlled as general-purpose input/output (GPIO) pins. The I/O structure of each pin includes:

- Programmable slew rate control of the outputs
- Open drain mode
- Programmable pull enable/disable on the input
- Programmable pull up/pull down function on the input

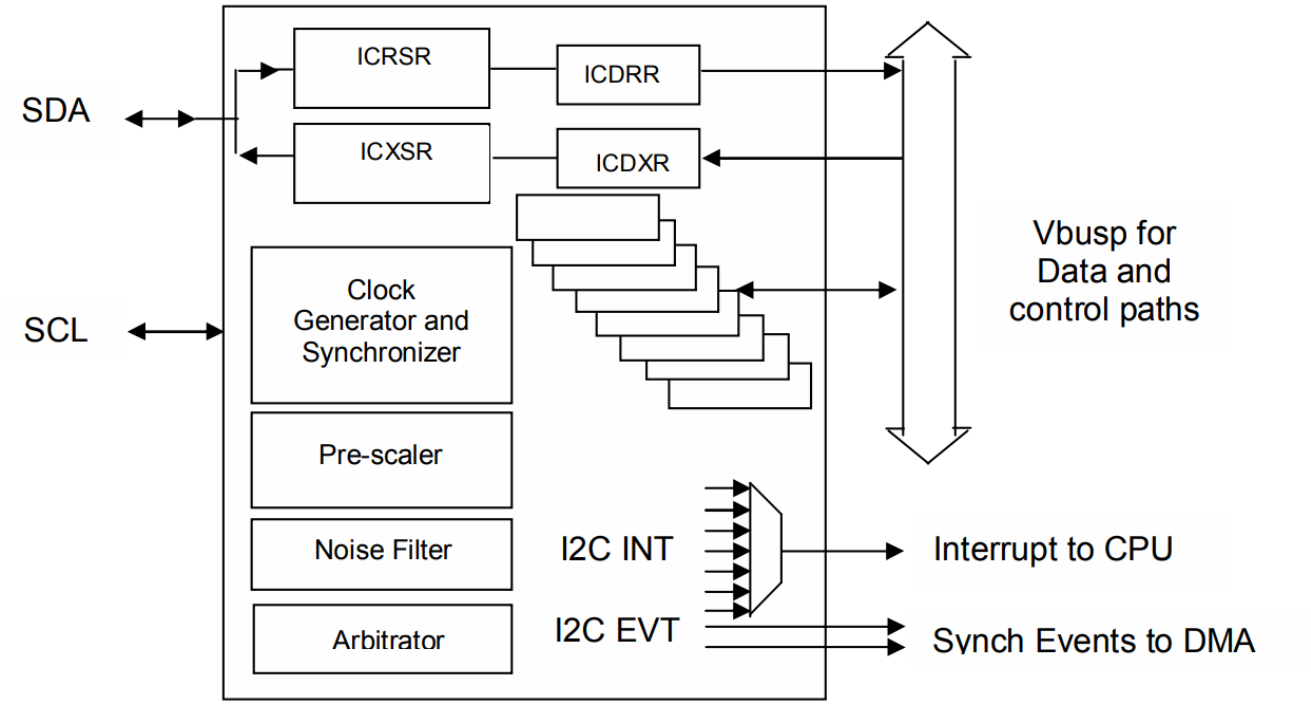


Figure 13-6. Simple I2C Block Diagram

13.1.2.1.1 I2C Features

Each multi controller I2C module has the following features:

- Compliance to the Philips I^{2}C bus specification, v2.1 (*The I2C Specification*, Philips document number 9398 393 40011)
  - Bit/Byte format transfer
  - 7-bit device addressing modes
  - General call
  - START byte
  - Multi-controller transmitter/ peripheral receiver mode
  - Multi-controller receiver/ peripheral transmitter mode
  - Combined controller transmit/receive and peripheral receive/transmit mode
  - Transfer rates from 10 kbps up to 100 kbps
- Free data format
- Two configurable DMA events (transmit and receive)
- Seven interrupts that can be used by the CPU
- Operates with VBUS frequency of 6.7 MHz and up
- Operates with module frequency between 6.7 MHz to 13.3 MHz
- Module enable/disable capability
- The SDA and SCL are optionally configurable as general purpose I/O
- Slew rate control of the outputs
- Open drain control of the outputs
- Programmable pullup/pulldown capability on the inputs
- Supports Ignore NACK mode

Note

Only the I2C0 instance is a true I2C Open Drain buffer. I2C[1-3] are implemented with the typical LVCMOS voltage buffer and should be properly configured to operate as an Input/Output Open Drain signal type.

13.1.2.1.2 I2C Not Supported Features

- High-speed (HS) and Fast-speed (FS) modes
- C-bus compatibility mode
- The combined format in 10-bit address mode (the I2C sends the peripheral address second byte every time it sends the peripheral address first byte)
- Low power clock enable

13.1.2.2 I2C Environment

This section describes the I2C external connections (environment).

13.1.2.2.1 I2C Typical Application

Figure 13-7 shows the multicontroller I2C controller and their related connections with I²C-compliant devices.

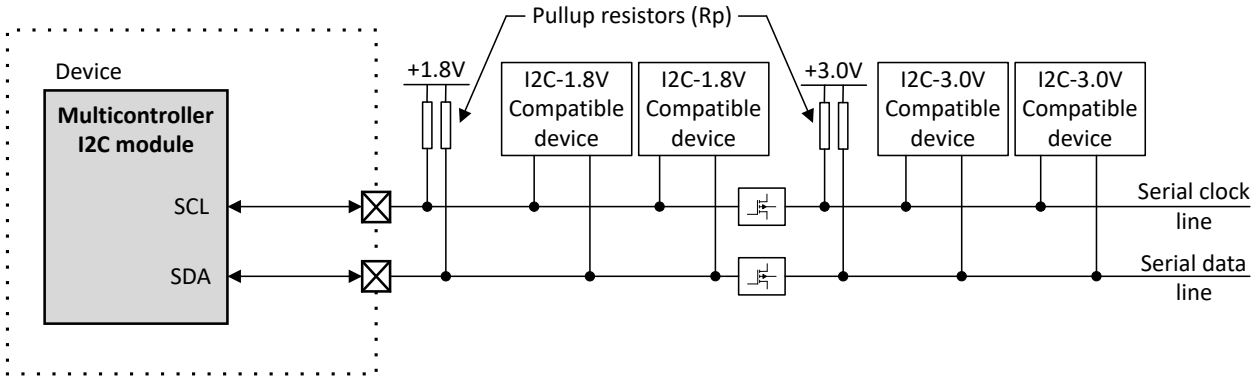


Figure 13-7. I2C and Typical Connections to I2C Devices

Table 13-17 describes the I2C I/O signals.

Table 13-17. I2C I/O Signals

		I2C[0-3]		
SCL	I2C[0]_SCL	I/O	I²C serial clock line. Open-drain output buffer.	1
SDA	I2C[0]_SDA	I/O	I²C serial data line. Open-drain output buffer.	1
SCL	I2C[1:3]_SCL	I/O	I²C serial clock line. Emulated open-drain output buffer.	1
SDA	I2C[1:3]_SDA	I/O	I²C serial data line. Emulated open-drain output buffer.	1

(1) I = Input; O = Output; I/O = Bidirectional

13.1.2.2.1.2

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.

13.1.2.2.2 I2C Typical Connection Protocol and Data Format

13.1.2.2.2.1 I2C Serial Data Formats

The I2C module operates in byte data format. Each message put on the SDA line is 2 to 8-bits long. The number of messages that can be transmitted or received is unrestricted. The data is transferred with the most significant bit (MSB) first (Figure 13-8). Each message is followed by an acknowledge bit from the I2C if it is in receiver mode. The I2C module does not support little endian systems.

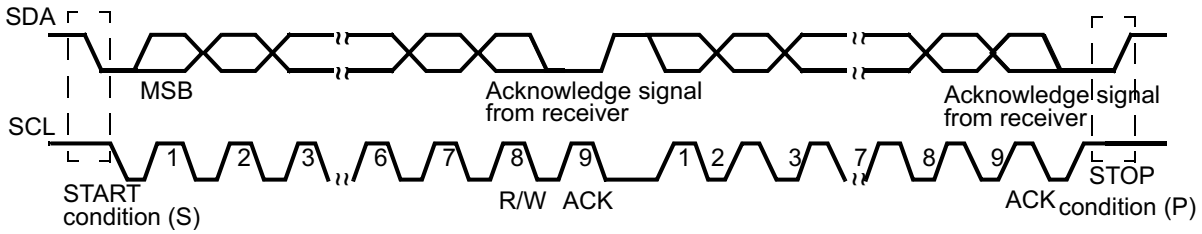


Figure 13-8. I2C Module Data Transfer

The first byte after a START condition (S) always consists of 8 bits that comprise either a 7-bit address plus the R/ $\overline{W}$ bit, or 8 data bits. The eighth bit, R/W, in the first byte determines the direction of the data. When the R/ $\overline{W}$ bit is 0, the controller writes (transmits) data to a selected peripheral device; when the R/ $\overline{W}$ bit is 1, the controller reads (receives) data from the peripheral device. In acknowledge mode, an extra bit dedicated for the acknowledgement (ACK) bit is inserted after each message.

The I2C module supports the following formats:

- 7-bit addressing format (Section 13.1.2.2.2.4.1)
- 10-bit addressing format (Section 13.1.2.2.2.4.2)
- 7-bit/10-bit addressing format with repeated START condition (Section 13.1.2.2.2.4.3)
- Free-data format (Section 13.1.2.2.2.4.4)

13.1.2.2.2.2 I2C Data Validity

The data on the serial data line (SDA) must be stable during the high period of the serial clock line. The high and low states of the data line can change only when the clock signal on the serial clock line (SCL) is low.

Figure 13-9 is an example of data validity requirements.

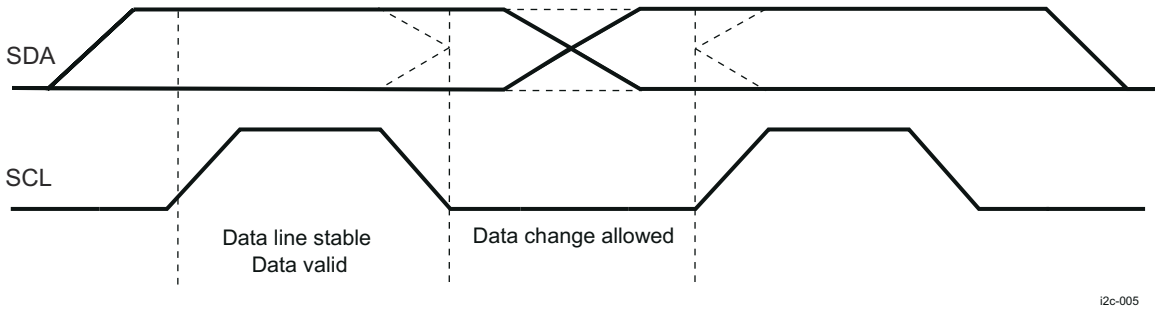


Figure 13-9. I2C Bit Transfer on the I2C Bus

13.1.2.2.2.3 I2C Start and Stop Conditions

START and STOP conditions are generated by a controller I2C module.

- The START condition is defined as a high-to-low transition on the SDA line while SCL is high. A controller drives this condition to indicate the start of data transfer. The bus is considered to be busy after the START condition, and the bus busy bit (BB) in ICSTR (Interrupt Status Register) is set to 1.
- The STOP condition is defined as a low-to-high transition on the SDA line while SCL is high. A controller drives this condition to indicate the end of data transfer. The bus is considered to be free after the STOP condition, therefore the BB bit in ICSTR (Interrupt Status Register) is cleared to 0.

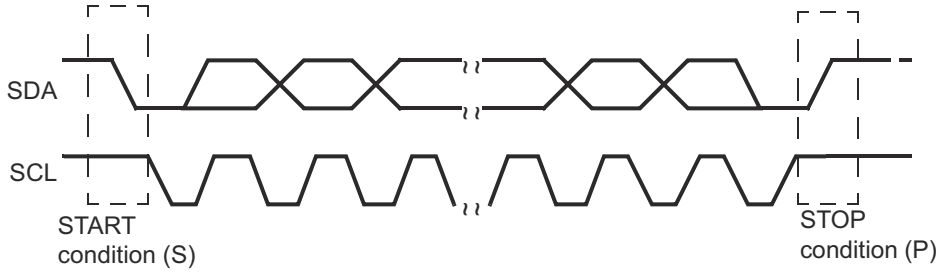


Figure 13-10. I2C Module START and STOP Conditions

For the I2C module to start a data transfer with a START condition, the controller mode bit (MST) and the START condition bit (STT) in the ICMDR must both be set to 1. For the I2C module to end a data transfer with a STOP condition, the STOP condition bit (STP) must be set to 1. When the BB bit is set to 1 and the STT bit is set to 1, a repeated START condition is generated.

13.1.2.2.2.4 I2C Addressing

The I2C module supports two data formats in fast/standard (F/S) mode:

- 7-bit/10-bit addressing format
- 7-bit/10-bit addressing format with repeated start (Sr) condition

13.1.2.2.2.4.1 7-Bit Addressing Format

In the 7-bit addressing format (Figure 13-11), the first byte after the START condition consists of a 7-bit peripheral address followed by the R/ $\overline{W}$ bit (in the LSB). The R/ $\overline{W}$ bit determines the direction of the data transfer:

- R/ $\overline{W}$ = 0: The controller writes (transmits) data to the addressed peripheral.
- R/ $\overline{W}$ = 1: The controller reads (receives) data from the peripheral.

An extra clock cycle dedicated for acknowledgment (ACK) is inserted after each byte. If the ACK is inserted by the peripheral after the first byte from the controller, it is followed by n bits of data from the transmitter (controller or peripheral, depending on the R/ $\overline{W}$ bit). The device I2C allows n to be a number between 2 to 8, programmable by the bit count (BC) field of ICMDR. After the data bits have been transferred, the receiver inserts an ACK bit.

To select the 7-bit addressing format, write 0 to the expanded address enable (XA) bit of I2CMDR and make sure the free data format mode is off (FDF = 0 in ICMDR).

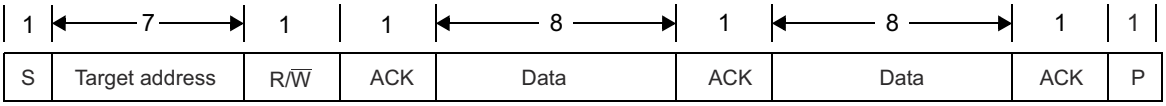


Figure 13-11. I2C Module 7-Bit Addressing Format

13.1.2.2.2.4.2 10-Bit Addressing Format

The 10-bit addressing format is similar to the 7-bit addressing format, but the controller sends the peripheral address in two separate byte transfers. In the 10-bit addressing format (Figure 13-12), the first byte is 11110b, the two MSBs of the 10-bit peripheral address, and the R/ $\overline{W}$ bit. The ACK bit is inserted after each byte. The second byte is the remaining 8 bits of the 10-bit peripheral address. The peripheral must send an acknowledgement after each of the two byte transfers. Once the controller has written the second byte to the peripheral, the controller can either write data or use repeated a START condition to change the data direction.

To select the 10-bit addressing format, write 1 to the expanded address enable (XA) bit of ICMDR and make sure the free data format mode is off (FDF = 0 in ICMDR).

Figure 13-12. I2C Module 10-bit Addressing Format

13.1.2.2.2.4.3 Using the Repeated START Condition

At the end of each byte, the controller can drive another START condition (Figure 13-13). Using this capability, a controller can transmit/receive any number of data bytes before generating a STOP condition. The length of a data byte can be from 2 to 8 bits. The repeated START condition can be used with the 7-bit addressing, 10-bit addressing, or the free data formats.

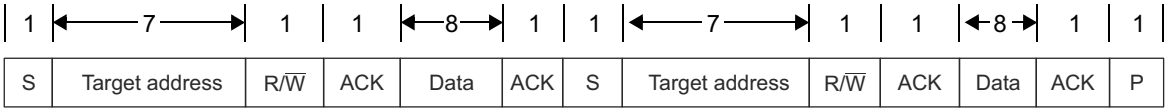


Figure 13-13. I2C Module 7-Bit Addressing Format with Repeated START

13.1.2.2.2.4.4 Free Data Format

In this format (Figure 13-14), the first byte after a START condition is a data byte. The ACK bit is inserted after each byte, followed by another 8 bits of data. No address or data direction bit is sent. Therefore, the transmitter and receiver must both support the free data format. The direction of data transmission (transmit or receive) remains constant throughout the transfer.

To select the free data format, write a 1 to the free data format (FDF) bit of the ICMDR. The free data format is not supported in the digital loop back mode.

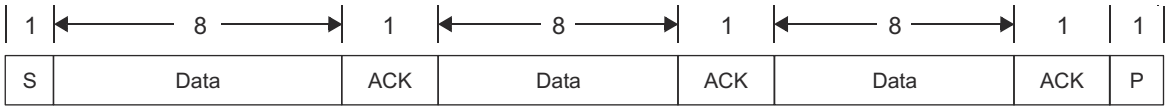


Figure 13-14. I2C Module in Free Data Format

13.1.2.2.2.5 I2C Controller Transmitter

All Controllers begin in this mode. The I2C module is a Controller and transmits control information and data to a target. In this mode, data assembled in any of the addressing formats shown in Figure 13-11, Figure 13-12, or Figure 13-13 is shifted out onto the SDA pin and synchronized with the self-generated clock pulses on the SCL pin. The clock pulses are inhibited and the SCL pin is held low when the intervention of the device is required ($\overline{\text{XSMT}}$ = 0) after a byte has been transmitted.

Note

If the I2C is configured for two simultaneous Controller transmissions, wait until the MST and BB have been reset before performing the second Controller transmission.

Failure to wait for the MST and BB to reset will prevent the start condition on the second transfer from being issued and the bus BB will not be set. Typically the end of the first transfer is handled by polling BB. However, the MST bit is not reset at the same instant as the BB bit. As a result, when the second Controller transmission is initiated before the resetting of the MST, the MST bit for the second transfer is reset. This prevents the I2C from recognizing itself as the Controller, thus failing to occupy the bus.

13.1.2.2.2.6 I2C Controller Receiver

In this mode, the I2C module is a controller and receives data from a target. This mode can only be entered from the controller transmitter mode (the I2C module must first transmit a command to the target). In any of the addressing formats shown in Section 13.1.2.2.2.4.1, Section 13.1.2.2.2.4.2, or Section 13.1.2.2.2.4.3, the controller receiver mode is entered after the target address byte and the R/ $\overline{\text{W}}$ bit have been transmitted (if the R/ $\overline{\text{W}}$ bit is 1). Serial data bits received on the SDA pin are shifted in with the self-generated clock pulses on the SCL pin. The clock pulses are inhibited and the SCL is held low when the intervention of the device is required (RSFULL = 1) after a byte has been received. At the end of the transfer, the controller-receiver signals the end of data to the target-transmitter by not generating an acknowledge on the last byte that was clocked out of the target. The target-transmitter then releases the data line allowing the controller-receiver to generate a STOP condition or a repeated START condition.

In many applications, the size of the message is in the initial bytes of the message itself. Since the size of the message is not known to the controller before the transmission/reception starts, the controller must use the repeat mode in order to force the stop condition when the reception is completed. The repeat mode is enabled by setting the RM bit to 1. Due to the double buffer implementation on the receive side, the controller must generate the stop condition (STP =1) after reading the (message size - 1)th data.

13.1.2.2.2.7 I2C Target Transmitter

In this mode, the I2C module is a target and transmits data to a controller. This mode can only be entered from the target receiver mode (The I2C module must first receive a command from the controller). In any of the addressing formats shown in Section 13.1.2.2.2.4.1, Section 13.1.2.2.2.4.2, or Section 13.1.2.2.2.4.3, the target transmitter mode is entered if the target address byte is the same as its own address and the R/ $\overline{\text{W}}$ bit has been transmitted (if the R/ $\overline{\text{W}}$ bit is set to 1). The target transmitter shifts the serial data out on the SDA pin with the clock pulses that are generated by the controller device. The target device does not generate the clock,

but it can hold the SCL pin low when intervention of the device is required ($\overline{XSMT}$ = 0) after a byte has been transmitted.

13.1.2.2.2.8 I2C Target Receiver

In this mode, the I2C module is a target and receives data from a controller. All targets begin in this mode. Serial data bits received on the SDA pin are shifted in with the clock pulses that are generated by the controller device. The target device does not generate the clock, but it can hold the SCL pin low while intervention of the device is required (RSFULL = 1) after a byte has been received.

13.1.2.2.2.9 I2C Bus Arbitration

If two or more controller transmitters simultaneously start a transmission on the same bus, an arbitration procedure is invoked. Figure 13-15 illustrates the arbitration procedure between two devices. The arbitration procedure uses the data presented on the SDA bus by the competing transmitters. The first controller transmitter that generates a high is overruled by the other controller that generates a low. The arbitration procedure gives priority to the device that transmits the serial data stream with the lowest binary value. The controller transmitter that loses the arbitration switches to the peripheral receiver mode, sets the arbitration lost (AL) flag, and generates the arbitration-lost interrupt. The data transmitted by the other controller module is salvaged, and the I2C continues to receive data from the controller module. Should two or more devices send identical first bytes, arbitration continues on the subsequent bytes.

If, during a serial transfer, the arbitration procedure is still in progress when a repeated START condition or STOP condition is transmitted to I2C bus, the controller transmitters involved must send the repeated START condition or STOP condition at the same position in the format frame. In other words, arbitration is not allowed between:

- A repeated START condition and a data bit
- A STOP condition and a data bit
- A repeated START condition and a STOP condition

Peripherals are not involved in the arbitration procedure.

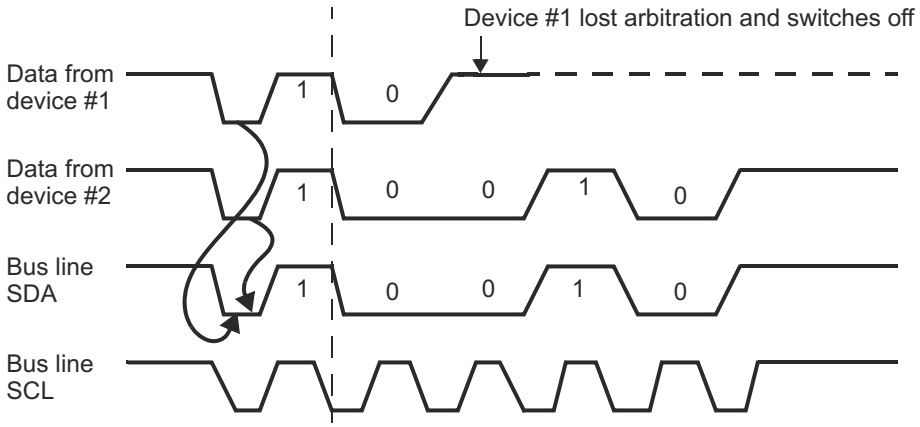


Figure 13-15. Arbitration Procedure Between Two Controller Transmitters

13.1.2.2.2.10 I2C Clock Generation and Synchronization

Under normal conditions only one controller device generates the clock signal; the SCL. During the arbitration procedure, however, there are two or more controller devices and the clock must be synchronized so that the data output can be compared. Figure 13-16 illustrates clock synchronization. The wired-AND property of the SCL line means that a device that first generates a low period on the SCL overrules the other devices. At this high-to-low transition, the clock generators of the other devices are forced to start their own low period. The SCL line is held low by the device with the longest low period. The other devices that finish their low periods must wait for the SCL line to be released before starting their high periods. A synchronized signal on the SCL is obtained where the slowest device determines the length of the low period and the fastest device determines the length of the high period.

If a device pulls down the clock line for a longer time, the result is that all clock generators must enter the wait state. In this way, a peripheral slows down a fast controller and the slow device creates enough time to store a received byte or to prepare a byte to be transmitted.

Note

I2C Protocol Fault

The following conditions violate the clock spec as defined in the Philips I^2C bus specification, v2.1 (*The I 2 C Specification,* Philips document number 9398 393 40011), and will result in an I2C protocol fault: I2CCLKH = 2, I2CCLKL = 2, I2CPSC = 2. This will cause the SDA data transition to occur while the SCL is high.

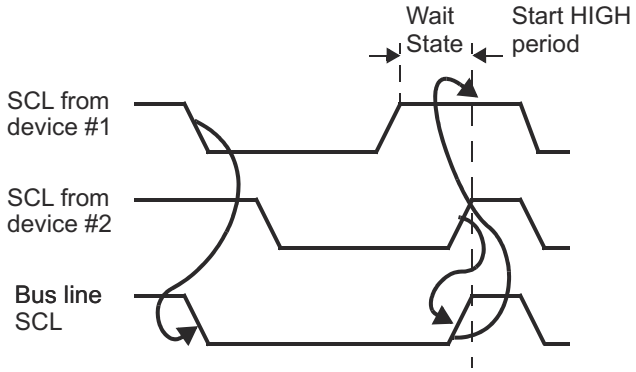


Figure 13-16. Synchronization of Two I2C Clock Generators During Arbitration

13.1.2.3 I2C Integration

There are 4x I2C module integrated in the device. The diagram below provides a visual representation of the device integration details.

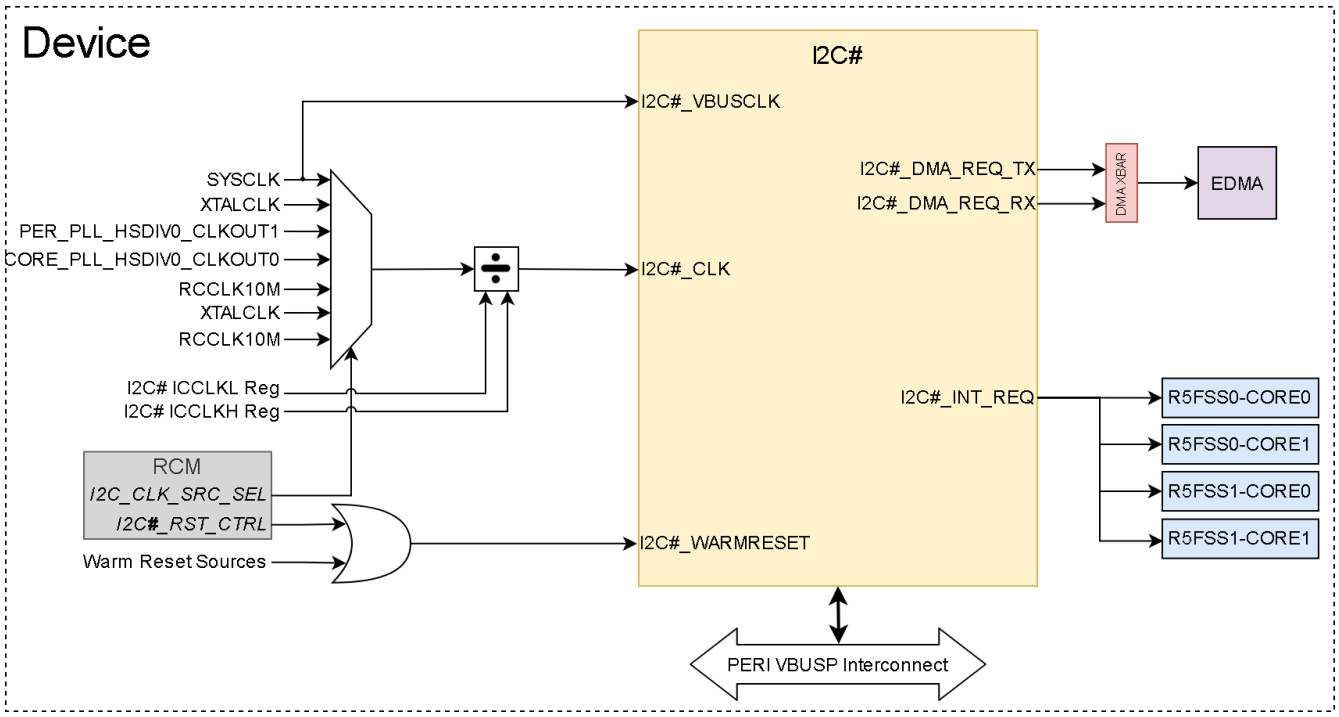


Figure 13-17. I2C Integration

The tables below summarize the device integration details of I2C# (where # = 0, 1, 2, 3).

Table 13-18. *I2C* Device Integration

This table describes the module device integration details.

Module Instance	Device Allocation	SoC Interconnect
I2C0	✓	PERI VBUSP Interconnect
I2C1	✓	PERI VBUSP Interconnect
I2C2	✓	PERI VBUSP Interconnect
I2C3	✓	PERI VBUSP Interconnect

Table 13-19. *I2C* Clocks

This table describes the module clocking signals.

Module Instance	Module Clock Input	Source Clock Signal	Source	Default Freq	Description
I2C[0:3]	I2C[0:3]_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	I2C[0:3] VBUS Clock
	I2C[0:3]_FCLK (I2C_CLK)	XTALCLK	External XTAL	25 MHz	I2C[0:3] Interface Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK: HSDIV0_CLKOUT0	500 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	

Table 13-20. *I2C* Resets

This table describes the module reset signals.

Module Instance	Module Reset Input	Source Reset Signal	Source	Description
I2C0	I2C0_RST(VBUSP_RSTn)	Warm Reset (SYS_NPRST)	RCM + Warm Reset Sources	I2C0 Asynchronous Reset
I2C1	I2C1_RST	Warm Reset (SYS_NPRST)	RCM + Warm Reset Sources	I2C1 Asynchronous Reset
I2C2	I2C2_RST	Warm Reset (SYS_NPRST)	RCM + Warm Reset Sources	I2C2 Asynchronous Reset
I2C3	I2C3_RST	Warm Reset (SYS_NPRST)	RCM + Warm Reset Sources	I2C3 Asynchronous Reset

Table 13-21. *I2C* Interrupt Requests

This table describes the module interrupt requests.

Module Instance	Module Interrupt Signal	Destination Interrupt Input	Destination	Type	Description
I2C0	i2c0_int_req	i2c0_int_req	ALL R5FSS Cores ICSSM Core	Pulse	I2C0 Status Event Interrupt
I2C1	i2c1_int_req	i2c1_int_req	ALL R5FSS Cores ICSSM Core	Pulse	I2C1 Status Event Interrupt
I2C2	i2c2_int_req	i2c2_int_req	ALL R5FSS Cores ICSSM Core	Pulse	I2C2 Status Event Interrupt
I2C3	i2c3_int_req	i2c3_int_req	ALL R5FSS Cores ICSSM Core	Pulse	I2C3 Status Event Interrupt

Table 13-22. *I2C* DMA Requests

This table describes the module DMA requests.

Module Instance	Module DMA Event	Destination DMA Event Input	Destination	Type	Description
I2C0	I2C0_TX	i2c0_dma_req_tx	EDMA Crossbar (EDMA_XBAR)	Pulse	I2C0 DMA Transmit Request
	I2C0_RX	i2c0_dma_req_rx			I2C0 DMA Receive Request
I2C1	I2C1_TX	i2c1_dma_req_tx	EDMA Crossbar (EDMA_XBAR)	Pulse	I2C1 DMA Transmit Request
	I2C1_RX	i2c1_dma_req_rx			I2C1 DMA Receive Request
IC2	I2C2_TX	i2c2_dma_req_tx	EDMA Crossbar (EDMA_XBAR)	Pulse	I2C2 DMA Transmit Request
	I2C2_RX	i2c2_dma_req_rx			I2C2 DMA Receive Request
I2C3	I2C3_TX	i2c3_dma_req_tx	EDMA Crossbar (EDMA_XBAR)	Pulse	I2C3 DMA Transmit Request
	I2C3_RX	i2c3_dma_req_rx			I2C3 DMA Receive Request

Note

For more information on the interconnects, see the *System Interconnect* chapter.

For more information on power, reset, and clock management, see the corresponding sections within the *Device Configuration* chapter.

For more information on the device interrupt controllers, see the *Interrupt Controllers* chapter.

13.1.2.4 I2C Functional Description

13.1.2.4.1 I2C Block Diagram

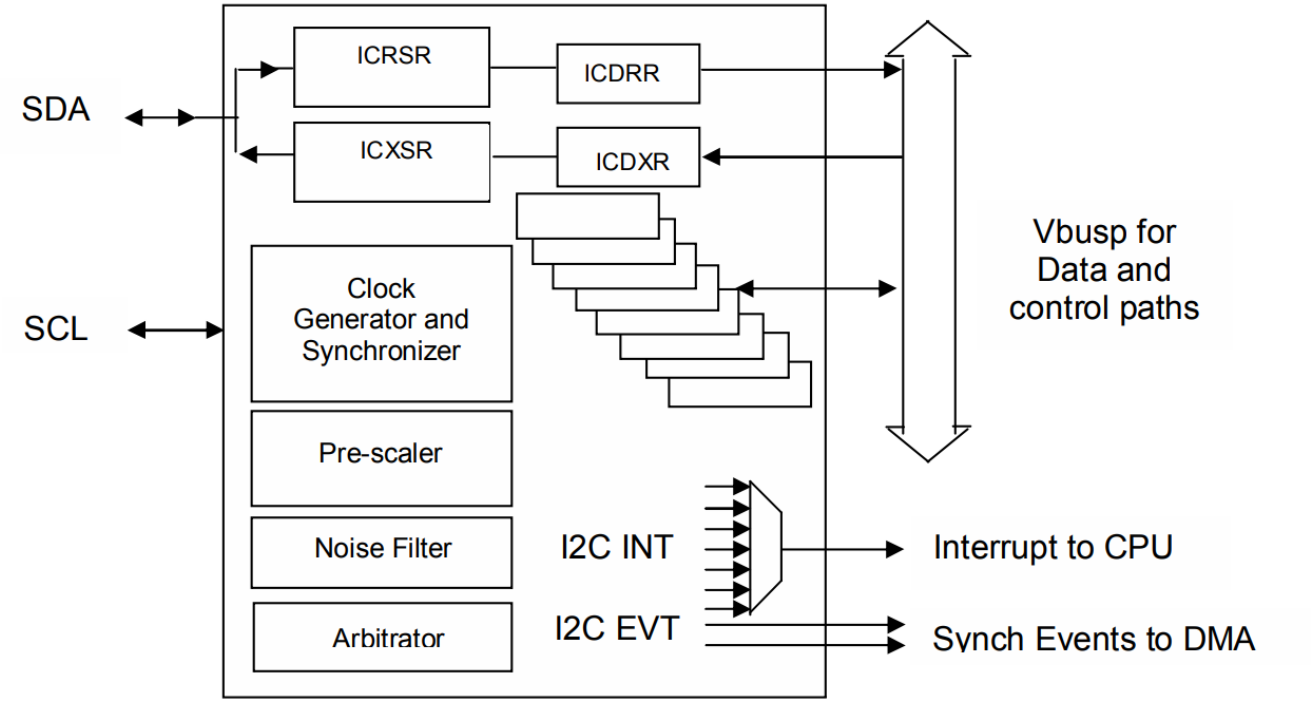


Figure 13-18. I2C Block Diagram

The I2C module consists of the following primary blocks:

- Serial I/F
- CPU Register Interface
- Pre-scaler
- VBUSP peripheral control I/F
- VBUSP peripheral dual data I/F (DMA and CPU with CPU has higher priority)
- Control/Status
- Data/Address
- I2C Clock generator

Data is communicated to devices interfacing the I2C via the serial data pin (SDA) and the serial clock pin (SCL). These two wires carry information between the TI device and others connected to the I2C bus. Both SDA and SCL are bi-directional pins. They must be connected to a positive supply voltage via a pull-up resistor. When the bus is free, both pins are high. The driver of these two pins has an open-drain to perform the required wired-AND function.

13.1.2.4.2 I2C Clocks

13.1.2.4.2.1 I2C Clocking

The I2C module is operated by the module clock. This clock is generated by way of the I2C prescaler block. The prescaler block consists of a 8-bit register, ICPSC, used for dividing down the device peripheral clock (VBUS_CLK) to obtain a module clock between 6.7 MHz and 13.3 MHz.

As shown in Figure 13-19, the I2C module uses the input clock generated from the device clock generator to generate the module clock and controller clock. The I2C input clock is the device peripheral clock (VBUS_CLK). The clock is then divided twice more inside the I2C module to produce the module clock and the controller clock.

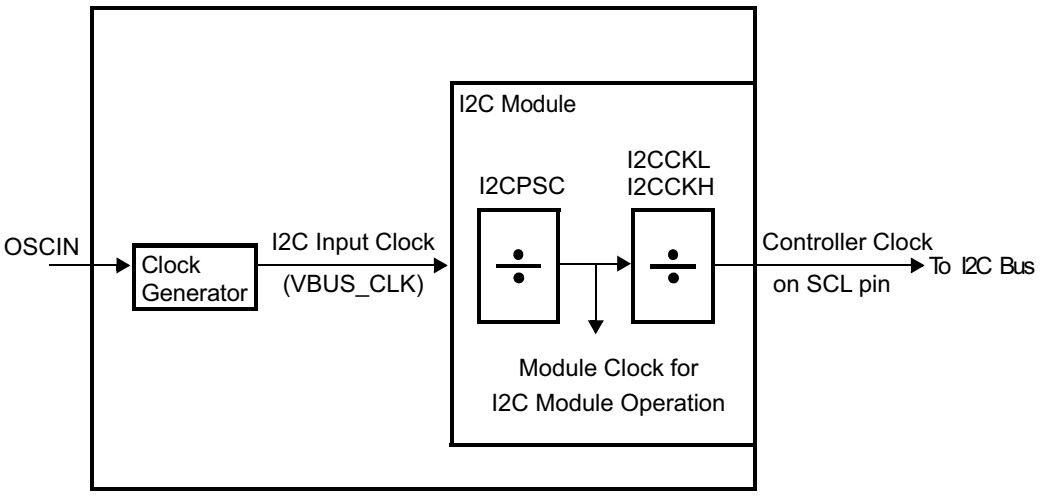


Figure 13-19. Clocking Diagram for the I2C Module

The module clock determines the frequency at which the I2C module operates. A programmable prescaler in the I2C module divides down the input clock to produce the module clock. To specify the divide-down value, initialize the IPSC7_IPSC0 bit field of the prescaler register, ICPSC. The resulting frequency is:

$$ModuleClockFrequency - \frac{I2CInputClockFrequency}{(ICPSC + 1)} \tag{23}$$

The module clock frequency must be between 6.7MHz and 13.3MHz. The prescaler can only be initialized while the I2C module is in the reset state (IRS = 0 in ICMDR). The prescaled frequency takes effect only when IRS is changed to 1. Changing the ICPSC value while IRS = 1 has no effect.

The controller clock appears on the SCL pin when the I2C module is configured to be a controller on the I2C bus. This clock controls the timing of the communication between the I2C module and a peripheral. As shown in Figure 13-19, a second clock divider in the I2C module divides down the module clock to produce the controller clock. The clock divider uses the ICCLKL to divide down the low portion of the module clock signal and uses the ICCLKH to divide down the high portion of the module clock signal.

The resulting frequency is:

$$ControllerClockFrequency = \frac{ModuleClockFrequency}{(ICCLKL + d) + (ICCLKH + d)} \tag{24}$$

$$ControllerClockFrequency = \frac{I2CInputClockFrequency}{(ICPSC + 1)((ICCLKL + d) + (ICCLKH + d))} \tag{25}$$

where *d* depends on the value of ICPSC:

ICPSC	d
0	7
1	6

ICPSC	d
Greater than 1	5

Note

The controller clock frequency defined above does not include rise/fall time and latency of the synchronizer inside the module. The actual transfer rate is slower than the value calculated from the formula above. Also, due to the nature of SCL synchronization, the SCL clock period can change if SCL synchronization is taking place.

13.1.2.4.3 I2C Software Reset

The I2C module can be reset in the following two ways:

- Through the global peripheral reset. A device reset causes a global peripheral reset.
- By clearing the $\overline{\text{IRS}}$ bit in the I2C mode register (ICMDR). When the global peripheral reset is removed, the $\overline{\text{IRS}}$ bit is cleared to 0, keeping the I2C module in the reset state.

13.1.2.4.4 I2C Interrupt Requests

The I2C module generates seven types of interrupts. These seven interrupts are accompanied with seven interrupt mask bits in the interrupt mask register (ICIMR) and with seven interrupt flag bits in the status register (ICSTR).

The I2C module generates the interrupt requests described below. All requests are multiplexed through an arbiter into a single I2C interrupt request to the CPU. Each interrupt request has a flag bit and an enable bit. Interrupts must be enabled prior to the occurrence of the expected interrupt condition. When one of the specified events occurs, the flag bit is set. If the corresponding enable bit is 0, the interrupt request is blocked. If the enable bit is 1, the interrupt request is forwarded to the CPU as an I2C interrupt request. As an alternative, the CPU can poll all of the bits shown in Table 13-23.

Table 13-23. Interrupt Requests Generated by I2C Module

Flag	Name	Generated
AL	Arbitration-lost interrupt	Generated when the I2C module has lost an arbitration contest with another controller-transmitter
NACK	No-acknowledge interrupt	Generated when the controller I2C does not receive an acknowledge from the receiver
ARDY	Register-access-ready interrupt	Generated when the previously programmed address, data and command have been performed and the status bits have been updated. The interrupt is used to notify the device that the I2C registers are ready to be accessed.
ICRRDY	Receive-data-ready interrupt	Generated when the received data in the receive-shift register (ICSR) has been copied into the data receive register (ICDRR). The RXRDY bit can also be polled by the device to determine when to read the received data in the ICDRR.
ICXRDY	Transmit-data-ready interrupt	Generated when the transmitted data has been copied from the data transmit register (ICDXR) into the transmit-shift register (ICXSR). The TXRDY bit can also be polled by the device to determine when to write the next data into ICDXR.
SCD	Stop-condition-detect interrupt	Generated when a STOP condition has been detected.
AAS	Address-as-peripheral interrupt	Generated when the I2C has recognized its own peripheral address or an address of all zeroes.

13.1.2.4.5 I2C Noise Filter

The noise filter is used to suppress any noises that are 50ns or less. It is designed to suppress noise with one module clock, assuming the lower and upper limits of the module clock are 6.7MHz and 13.3MHz, respectively.

13.1.2.5 I2C Programming Guide

13.1.2.5.1 I2C Low-Level Programming Models

13.1.2.5.1.1 I2C Programming Model

This section describes the programming model of the multicontroller I2C controllers configured in I2C mode. Register names begin with *I2Cn* where 'n' is the instance number of an I2C module. The instance number starts from 0 and increments up based on the number of I2C instances available on the device.

13.1.2.5.1.1.1 Main Program

13.1.2.5.1.1.1.1 Module State after Reset

Before enabling the I2C controller, perform the following steps:

1. Enable the functional and interface clocks.
2. Program the prescaler to obtain an approximately 12-MHz internal sampling clock by programming the corresponding value in the *I2Cn_ICPSC* I2C Prescaler Register IPSC7_IPSC0 bit field. This value depends on the frequency of the functional clock (SYS_CLK).
3. Take the I2C controller out of reset by setting the *I2Cn_ICMDR[5]* IRS bit to 1.
   a. If using interrupts for transmitting/receiving data, enable the interrupt masks in the *I2Cn_ICIMR* I2C Interrupt Mask Register.
   b. If using DMA for transmitting/receiving data, enable the DMA via the *I2Cn_ICDMAC* I2C DMA Control Register and then program the DMA controller.

13.1.2.5.1.1.2 Initialization Procedure

To initialize the I2C controller, use the *I2Cn_ICMDR* I2C Mode Register bits to configure the respective modes such as controller/peripheral, transmitter/receiver, repeat mode, bit count, expanded addressing, and free run mode.

13.1.2.5.1.1.3 Section

Program the *I2Cn_ICCLKL* I2C Clock Divider Low register and *I2Cn_ICCLKH* I2C Clock Divider High register to obtain a bit rate of 100 kbps or 400 kbps. These values depend on the internal sampling clock frequency (see I2C Clocking).

13.1.2.5.1.1.4 Configure Address Registers

In controller mode, configure the peripheral address register to transmit by programming the *I2Cn_ICSAR[9-0]* A9_A0 bit field.

Note

For a 10-bit address, set the *I2Cn_ICMDR[8]* XA bit to 1.

In peripheral mode, configure the peripheral address for other controllers to use by programming the *I2Cn_ICOAR[9-0]* A9_A0 bit field

13.1.2.5.1.1.5 Initiate a Transfer

If in controller transmitter mode, program the *I2Cn_ICDXR* I2C Data Transmit register with the transmit data first.

Poll the *I2Cn_ICSTR[12]* I2C Interrupt Status register for the Bus Busy (BB) bit. If it is '0', the busy is not busy and the transfer can be initiated by configuring the start/stop condition in the *I2Cn_ICMDR* I2C Mode Register with the STT and STP bits.

13.1.2.5.1.1.6 Receive Data

Poll the *I2Cn_ICSTR[3]* I2C Interrupt Status register ICRRDY bit, or use the RRDY interrupt (*I2Cn_ICIVR[2:0]* I2C Interrupt Vector register INTCODE = 0b100) to read the receive data in the *I2Cn_ICDRR* I2C Data Receive register. If the DMA is enabled, there are no I2C-specific registers for monitoring DMA activity.

Note

In receive mode only, the *I2Cn_ICSTR[11]* RSFULL 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.

13.1.2.5.1.1.1.7 Transmit Data

Poll the *I2Cn_ICSTR[4]* I2C Interrupt Status register ICXRDY bit, or use the XRDY interrupt (*I2Cn_ICIVR[2:0]* I2C Interrupt Vector register INTCODE = 0b101) to transmit data from the *I2Cn_ICDXR* I2C Data Transmit register. If the DMA is enabled, there are no I2C-specific registers for monitoring DMA activity.

Note

In transmit mode only, the *I2Cn_ICSTR[10]* XSMT bit indicates whether the transmitter has experienced underflow. Underflow occurs when the shift register is empty and the *I2Cn_ICDXR* register has not been loaded.

13.1.2.5.1.1.2 Interrupt Subroutine Sequence

Monitor the *I2Cn_ICIVR* I2C Interrupt Vector register INTCODE bits to determine which interrupt occurred in the following sequence.

1. Test for arbitration lost and resolve accordingly.
2. Test for no acknowledgment and resolve accordingly.
3. Test for register access ready and resolve accordingly.
4. Test for receive data ready and resolve accordingly.
5. Test for transmit data ready and resolve accordingly.
6. Test for stop condition and resolve accordingly.

13.1.3 Multichannel Serial Peripheral Interface (MCSPI)

This section describes the Multichannel Serial Peripheral Interface (MCSPI) modules for the device.

13.1.3.1 MCSPI Overview

The MCSPI (SPI) module is a multichannel transmit/receive, controller/peripheral synchronous serial bus. There are 8 SPI modules in the device.

13.1.3.1.1 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 two channels in controller mode, or single channel in receive mode
- Each SPI controller operates at up to 50 MHz
- SPI0 and SPI4 support 2 chip selects while SPI1, SPI2, SPI3, SPI5, SPI6, and SPI7 support 1 chip select
- Supports DMA access
- 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

13.1.3.1.2 SPI Not Supported Features

The following features are not supported on this family of devices:

- Peripheral mode wake-up
- Retention during power down
- In peripheral mode only channel 0 is used
- Local power management of clock activity

13.1.3.2 SPI Environment

The SPI[0:7] modules are hereinafter referred to as the SPI or MCSPI module.

This section describes the SPI external connections (environment).

13.1.3.2.1 MCSPI Protocol and Data Format

The synchronous MCSPI protocol allows a controller device to initiate serial data transfers to a peripheral device. A peripheral select line (SPIEN[i]) allows selection of an individual peripheral MCSPI device. Peripheral devices that are not selected do not interfere with MCSPI bus activities.

MCSPI offers the flexibility to modify the following parameters to adapt to the device features:

- Word length

  MCSPI supports any MCSPI word ranging from 4 bits to 32 bits long (the MCSPI_CHCONF_0/1/2/3[11-7] WL bit field).

  MCSPI word length can be changed between transmissions to allow the controller device to communicate with peripheral peripherals that have different requirements.
- MCSPI enable (SPIEN[i], for channel i)

  The polarity of the MCSPI enable signals is programmable (the MCSPI_CHCONF_0/1/2/3[6] EPOL bit). SPIEN[i] signals can be active high or low.

  Assertion of the SPIEN[i] signals is programmable and can be done manually or automatically. The manual assertion mode is available in single controller mode only. SPIEN[i] can be kept active between words with the MCSPI_CHCONF_0/1/2/3[20] FORCE bit.

  Two consecutive words for two different peripheral devices can go along with active SPIEN[i] signals with different polarity.
- Programmable start-bit

  In start-bit mode a start-bit is added before the MCSPI word length to indicate how the next MCSPI word must be handled. The start-bit is enabled by setting the MCSPI_CHCONF_0/1/2/3[23] SBE bit to 1. The MCSPI_CHCONF_0/1/2/3[24] SBPOL bit defines the polarity of the start-bit.
- Programmable MCSPI clock
  - Bit rate

    In controller mode, the baud rate of the MCSPI serial clock is programmable using the 50-MHz reference clock (from the device clock management module). Table 13-24 lists the SPICLK bit rates obtained for data transfer when programming the clock divider (the MCSPI_CHCONF_0/1/2/3[5-2] CLKD bit field).

Table 13-24. MCSPI Controller Clock Rates

Divider	Clock Rate
1	50 MHz(1)
2	25 MHz(1)
4	12.5 MHz
8	6.25 MHz
16	3.125 MHz
32	1.5625 MHz
64	781.25 kHz
128	390.625 kHz
256	~195 kHz
512	~97.7 kHz
1024	~48.8 kHz
2048	~24.4 kHz
4096	~12.2 kHz

(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 Datasheet.

– Polarity and phase

The polarity (the MCSPI_CHCONF_0/1/2/3[1] POL bit) and the phase (the MCSPI_CHCONF_0/1/2/3[0] PHA bit) of the MCSPI serial clock (SPICLK) are configurable to offer four combinations. Software selects the right combination, depending on the device. See Table 13-25 and Figure 13-20.

Table 13-25. Phase and Polarity Combinations

Polarity (POL)	Phase (PHA)	MCSPI Mode	Description
0	0	Mode 0	SPICLK is inactive low and sampling occurs at the rising edge.
0	1	Mode 1	SPICLK is inactive low and sampling occurs at the falling edge.
1	0	Mode 2	SPICLK is inactive high and sampling occurs at the falling edge.
1	1	Mode 3	SPICLK is inactive high and sampling occurs at the rising edge.

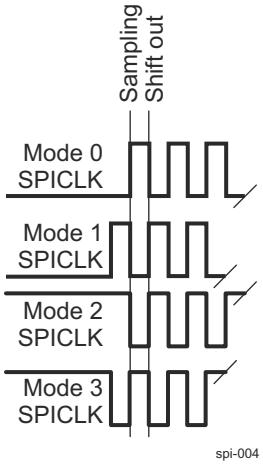


Figure 13-20. Phase and Polarity Combinations

13.1.3.2.1.1 Transfer Format

In controller and peripheral modes, the MCSPI drives the data lines when SPIEN[i] is asserted.

Each word is transmitted starting with the most-significant bit (MSB).

This section explains the two cases of data transmission determined by the clock phase (PHA) and the type of data transmission using a start-bit (SBE) called the start-bit mode:

- Transmission in mode 0 and mode 2 (PHA = 0)

  When PHA = 0, the first bit of the MCSPI word to transmit (on the controller or the peripheral data output pin) is valid one-half cycle of SPICLK after the assertion of SPIEN[i].

  Therefore, the first edge of the SPICLK line is used by the controller to sample the first data bit sent by the peripheral. On the same edge, the first data bit sent by the controller is sampled by the peripheral.

  On the next SPICLK edge, the received data bit is shifted into the receive shift register and a new data bit is transmitted on the serial data line.

  This process continues for a number of pulses on the SPICLK line defined by the MCSPI word length programmed in the controller device, with data being latched on odd-numbered edges and shifted on even-numbered edges, see Figure 13-21.

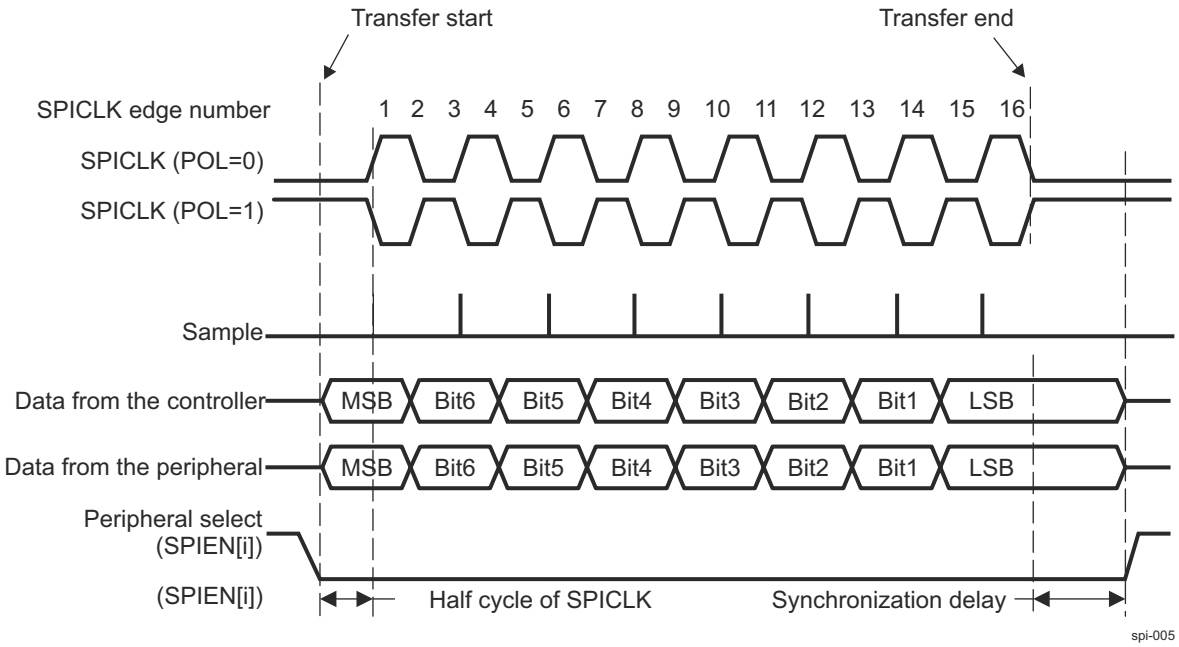


Figure 13-21. Full-Duplex Transfer Format With PHA = 0

- Transmission in mode 1 and mode 3 (PHA = 1)

  When PHA = 1, the first bit of the MCSPI word to transmit (on the controller or the peripheral data output pin) is valid on the following SPICLK edge (one-half cycle later). This is the sampling edge for the controller and peripheral. A synchronization delay is added between the activation of SPIEN[i] and the first SPICLK edge.

  The received data bit is shifted into the shift register on the third SPICLK edge.

  This process continues for a number of pulses on the SPICLK line defined by the MCSPI word length programmed in the controller device, with data being latched on even-numbered edges and shifted on odd-numbered edges.

  Note

  The minimum synchronization delay is one cycle of SPICLK, if the frequency of SPICLK equals the frequency of MCSPI_FCLK (MCSPI functional clock) in controller mode. The minimum synchronization delay is one-half cycle of SPICLK, if the frequency of SPICLK is lower than the frequency of MCSPI_FCLK in the controller and peripheral modes.

- Transmission with a start-bit (SBE = 1)

  When the MCSPI_CHCONF_0/1/2/3[23] SBE bit is set to 1, a start-bit is added before the MSB to indicate whether the next MCSPI word must be handled as a command or as data.

  Figure 13-22 shows an example of a data transfer with an extra start-bit.

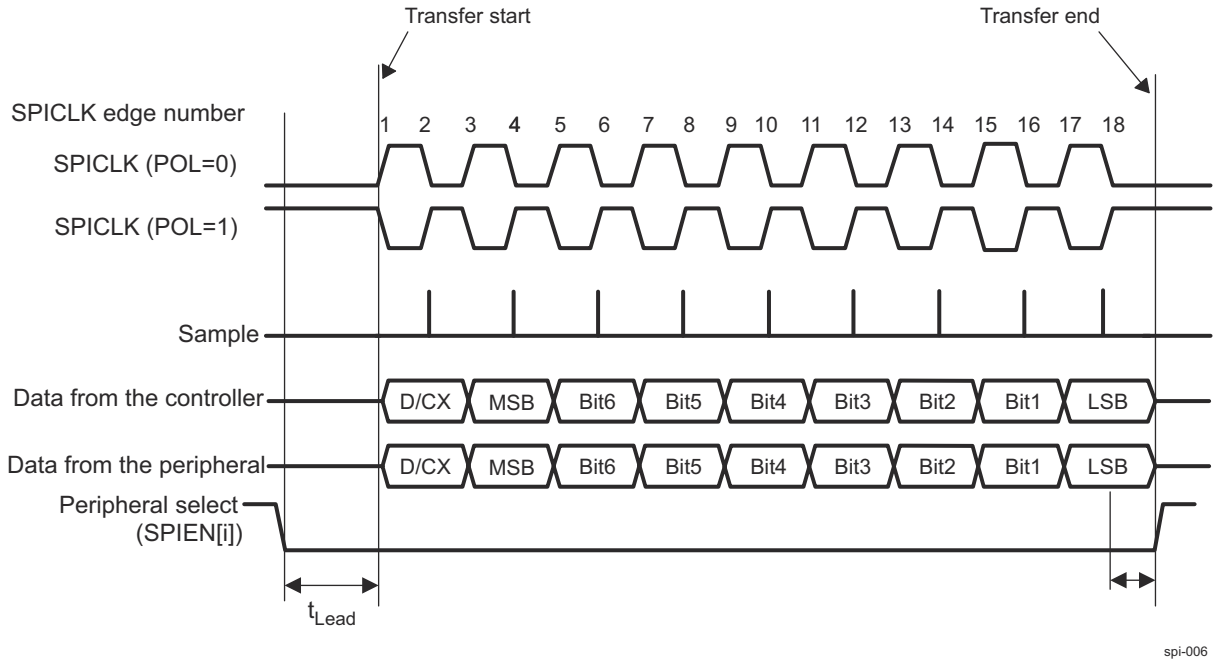
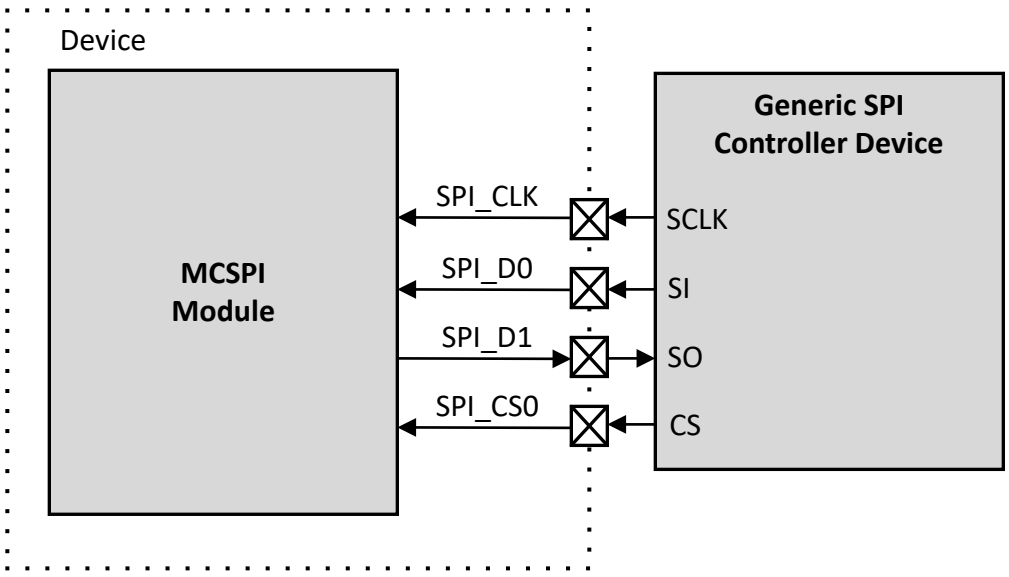


Figure 13-22. Extended MCSPI Transfer With a Start-Bit (SBE = 1)

13.1.3.2.2 MCSPI in Controller Mode

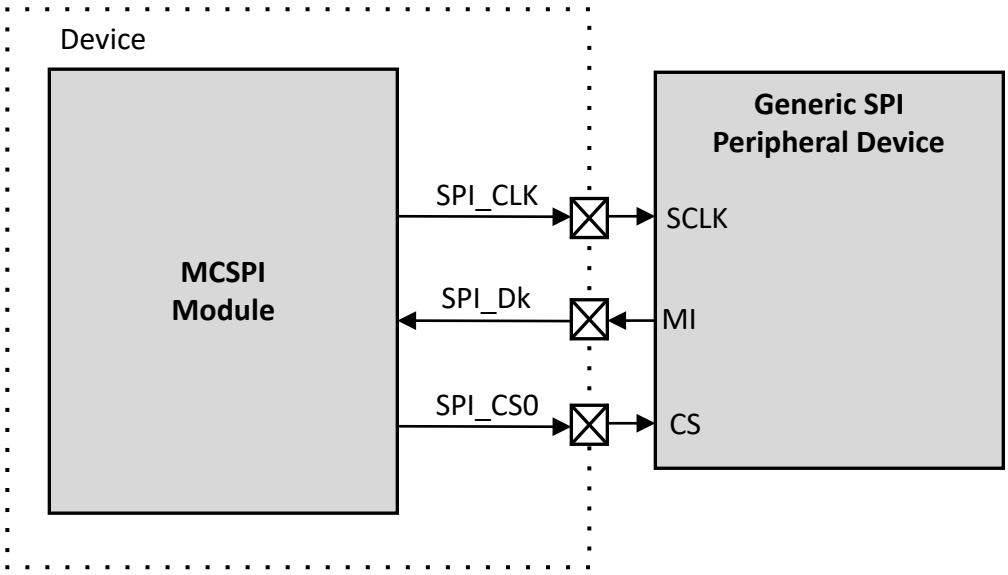
Figure 13-23 shows a case in controller mode (full-duplex) where the MCSPI module is connected with two peripheral devices.



A. Direction of D0 and D1 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 13-23. MCSPI Controller Mode (Full Duplex)

Figure 13-24 shows the controller single mode, which can also be configured in receive-only mode.



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

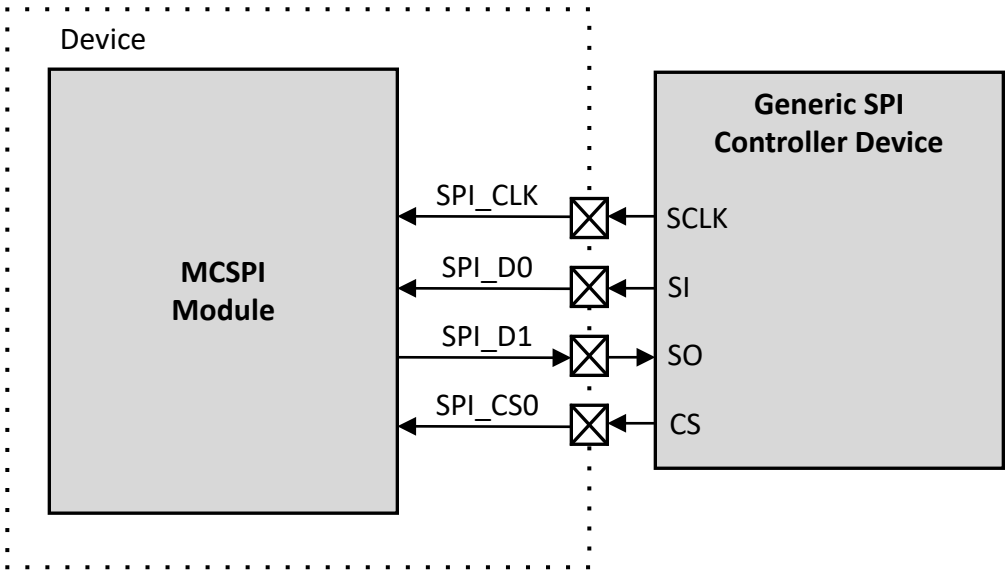
Figure 13-24. MCSPI Controller Single Mode (Receive Only)

13.1.3.2.3 MCSPI in Peripheral Mode

Figure 13-25 shows a case in peripheral mode (full-duplex).

Note

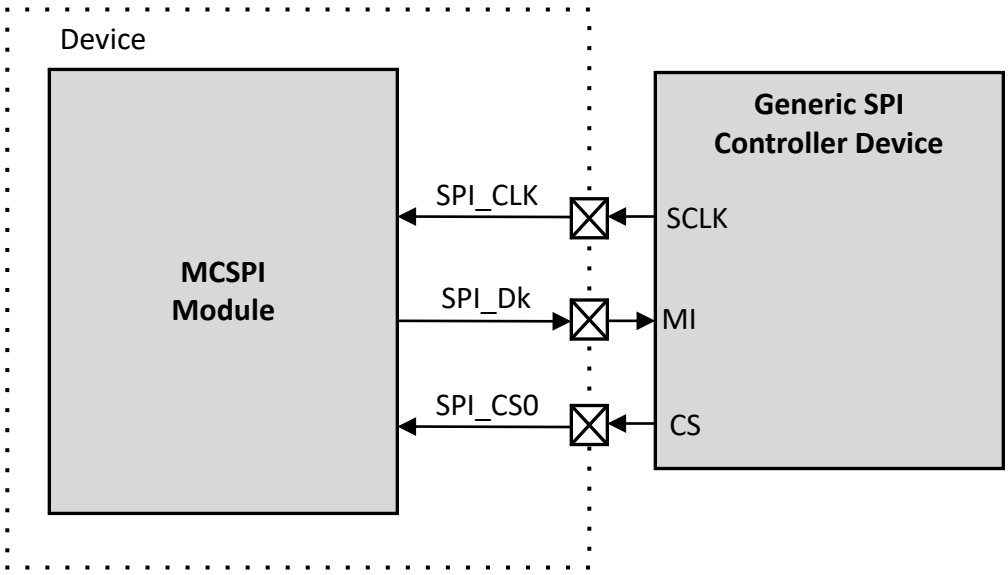
Only channel 0 can be configured as peripheral, but the chip-enable signal can be connected to any SPIEN[i] pin and then rerouted internally to channel 0 (the MCSPI_CHCONF_0[22-21] SPIENSLV bit field). For more information, see MCSPI Peripheral Mode.



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 13-25. MCSPI Peripheral Mode (Full Duplex)

Figure 13-26 shows the peripheral single mode, which can also be configured in transmit-only mode.



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 13-26. MCSPI Peripheral Single Mode (Transmit Only)

13.1.3.3 SPI Integration

There are 8x SPI modules integrated in the device. The diagram below provides a visual representation of the device integration details.

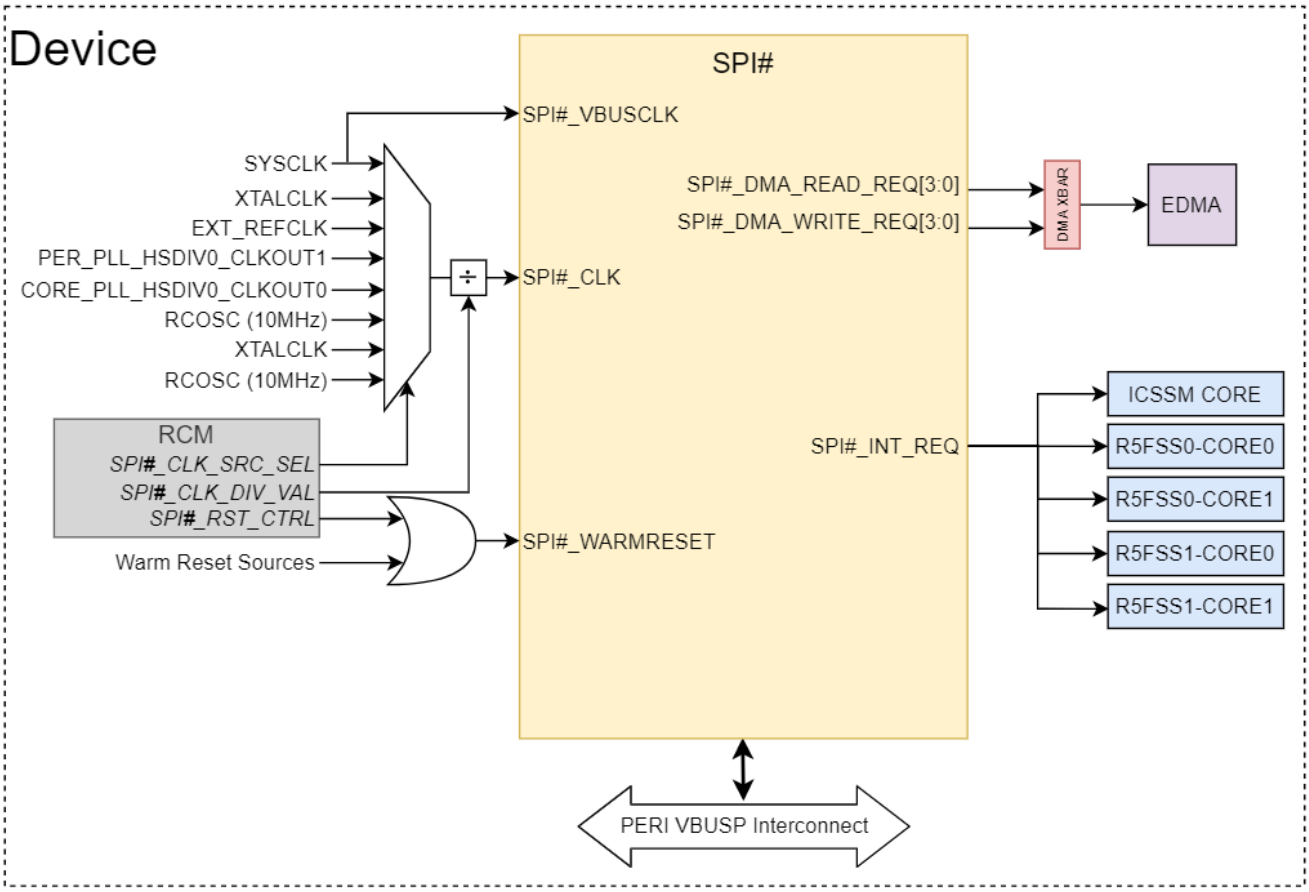


Figure 13-27. SPI Integration

The tables below summarize the device integration details of SPI# (where # = 0 to 7).

Table 13-26. *SPI* Device Integration

This table describes the module device integration details.

Module Instance	Device Allocation	SoC Interconnect
SPI0	✓	PERI VBUSP Interconnect
SPI1	✓	PERI VBUSP Interconnect
SPI2	✓	PERI VBUSP Interconnect
SPI3	✓	PERI VBUSP Interconnect
SPI4	✓	PERI VBUSP Interconnect
SPI5	✓	PERI VBUSP Interconnect
SPI6	✓	PERI VBUSP Interconnect
SPI7	✓	PERI VBUSP Interconnect

Table 13-27. *SPI* Clocks

This table describes the module clocking signals.

Module Instance	Module Clock Input	Source Clock Signal	Source	Default Freq	Description
SPI0	SPI0_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	SPI0 VBUS Clock
	SPI0_FCLK (SPI_CLK)	XTALCLK	External XTAL	25 MHz	SPI0 Interface Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK: HSDIV0_CLKOUT1	400 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	

Table 13-27. *SPI* Clocks (continued)

This table describes the module clocking signals.

Module Instance	Module Clock Input	Source Clock Signal	Source	Default Freq	Description
SPI1	SPI1_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	SPI1 VBUS Clock
	SPI1_FCLK (SPI_CLK)	XTALCLK	External XTAL	25 MHz	SPI1 Interface Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK: HSDIV0_CLKOUT1	400 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
SPI2	SPI2_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	SPI2 VBUS Clock
	SPI2_FCLK (SPI_CLK)	XTALCLK	External XTAL	25 MHz	SPI2 Interface Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK: HSDIV0_CLKOUT1	400 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	

Table 13-27. *SPI* Clocks (continued)

This table describes the module clocking signals.

Module Instance	Module Clock Input	Source Clock Signal	Source	Default Freq	Description
SPI3	SPI3_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	SPI3 VBUS Clock
	SPI3_FCLK (SPI_CLK)	XTALCLK	External XTAL	25 MHz	SPI3 Interface Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK: HSDIV0_CLKOUT1	400 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
SPI4	SPI4_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	SPI4 VBUS Clock
	SPI4_FCLK (SPI_CLK)	XTALCLK	External XTAL	25 MHz	SPI4 Interface Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK: HSDIV0_CLKOUT1	400 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	

Table 13-27. *SPI* Clocks (continued)

This table describes the module clocking signals.

Module Instance	Module Clock Input	Source Clock Signal	Source	Default Freq	Description
SPI5	SPI5_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	SPI5 VBUS Clock
	SPI5_FCLK (SPI_CLK)	XTALCLK	External XTAL	25 MHz	SPI5 Interface Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK: HSDIV0_CLKOUT1	400 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
SPI6	SPI6_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	SPI6 VBUS Clock
	SPI6_FCLK (SPI_CLK)	XTALCLK	External XTAL	25 MHz	SPI6 Interface Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK: HSDIV0_CLKOUT1	400 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	

Table 13-27. *SPI* Clocks (continued)

This table describes the module clocking signals.

Module Instance	Module Clock Input	Source Clock Signal	Source	Default Freq	Description
SPI7	SPI7_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	SPI7 VBUS Clock
	SPI7_FCLK (SPI_CLK)	XTALCLK	External XTAL	25 MHz	SPI7 Interface Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK: HSDIV0_CLKOUT1	400 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	

Table 13-28. *SPI* Resets

This table describes the module reset signals.

Module Instance	Module Reset Input	Source Reset Signal	Source	Description
SPI0	SPI0_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	SPI0 Asynchronous Reset
SPI1	SPI1_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	SPI1 Asynchronous Reset
SPI2	SPI2_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	SPI2 Asynchronous Reset
SPI3	SPI3_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	SPI3 Asynchronous Reset
SPI4	SPI4_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	SPI4 Asynchronous Reset
SPI5	SPI5_RST	Warm Reset (MOD_G_RST)	RCM+ Warm Reset Sources	SPI5 Asynchronous Reset
SPI6	SPI6_RST	Warm Reset (MOD_G_RST)	RCM+ Warm Reset Sources	SPI6 Asynchronous Reset
SPI7	SPI7_RST	Warm Reset (MOD_G_RST)	RCM+ Warm Reset Sources	SPI7 Asynchronous Reset

Table 13-29. *SPI* Interrupt Requests

This table describes the module interrupt requests.

Module Instance	Module Interrupt Signal	Destination Interrupt Input	Destination	Type	Description
SPI0	spi0_int_req	spi0_int_req	ALL R5FSS Cores PRU-ICSS Core	Level	SPI0 IP Status Information
SPI1	spi1_int_req	spi1_int_req	ALL R5FSS Cores PRU-ICSS Core	Level	SPI1 IP Status Information

Table 13-29. *SPI* Interrupt Requests (continued)

This table describes the module interrupt requests.

Module Instance	Module Interrupt Signal	Destination Interrupt Input	Destination	Type	Description
SPI2	spi2_int_req	spi2_int_req	ALL R5FSS Cores PRU-ICSS Core	Level	SPI2 IP Status Information
SPI3	spi3_int_req	spi3_int_req	ALL R5FSS Cores PRU-ICSS Core	Level	SPI3 IP Status Information
SPI4	spi4_int_req	spi4_int_req	ALL R5FSS Cores PRU-ICSS Core	Level	SPI4 IP Status Information
SPI5	spi5_int_req	spi5_int_req	ALL R5FSS Cores PRU-ICSS Core	Level	SPI5 IP Status Information
SPI6	spi6_int_req	spi6_int_req	ALL R5FSS Cores PRU-ICSS Core	Level	SPI6 IP Status Information
SPI7	spi7_int_req	spi7_int_req	ALL R5FSS Cores PRU-ICSS Core	Level	SPI7 IP Status Information

Table 13-30. *SPI* DMA Requests

This table describes the module DMA requests.

Module Instance	Module DMA Event	Destination DMA Event Input	Destination	Type	Description
SPI0	SPI0_DMA_READ_0	spi0_dma_read_req[0]	EDMA Crossbar (EDMA_XBAR)	Pulse	SPI0 DMA Read Request
	SPI0_DMA_READ_1	spi0_dma_read_req[1]			
	SPI0_DMA_READ_2	spi0_dma_read_req[2]			
	SPI0_DMA_READ_3	spi0_dma_read_req[3]			
	SPI0_DMA_WRITE_0	spi0_dma_write_req[0]			SPI0 DMA Write Request
	SPI0_DMA_WRITE_1	spi0_dma_write_req[1]			
	SPI0_DMA_WRITE_2	spi0_dma_write_req[2]			
	SPI0_DMA_WRITE_3	spi0_dma_write_req[3]			
SPI1	SPI1_DMA_READ_0	spi1_dma_read_req[0]	EDMA Crossbar (EDMA_XBAR)	Pulse	SPI1 DMA Read Request
	SPI1_DMA_READ_1	spi1_dma_read_req[1]			
	SPI1_DMA_READ_2	spi1_dma_read_req[2]			
	SPI1_DMA_READ_3	spi1_dma_read_req[3]			
	SPI1_DMA_WRITE_0	spi1_dma_write_req[0]			SPI1 DMA Write Request
	SPI1_DMA_WRITE_1	spi1_dma_write_req[1]			
	SPI1_DMA_WRITE_2	spi1_dma_write_req[2]			
	SPI1_DMA_WRITE_3	spi1_dma_write_req[3]			
SPI2	SPI2_DMA_READ_0	spi2_dma_read_req[0]	EDMA Crossbar (EDMA_XBAR)	Pulse	SPI2 DMA Read Request
	SPI2_DMA_READ_1	spi2_dma_read_req[1]			
	SPI2_DMA_READ_2	spi2_dma_read_req[2]			
	SPI2_DMA_READ_3	spi2_dma_read_req[3]			
	SPI2_DMA_WRITE_0	spi2_dma_write_req[0]			SPI2 DMA Write Request
	SPI2_DMA_WRITE_1	spi2_dma_write_req[1]			
	SPI2_DMA_WRITE_2	spi2_dma_write_req[2]			
	SPI2_DMA_WRITE_3	spi2_dma_write_req[3]			

Table 13-30. *SPI* DMA Requests (continued)

This table describes the module DMA requests.

Module Instance	Module DMA Event	Destination DMA Event Input	Destination	Type	Description
SPI3	SPI3_DMA_READ_0	spi3_dma_read_req[0]	EDMA Crossbar (EDMA_XBAR)	Pulse	SPI3 DMA Read Request
	SPI3_DMA_READ_1	spi3_dma_read_req[1]			
	SPI3_DMA_READ_2	spi3_dma_read_req[2]			
	SPI3_DMA_READ_3	spi3_dma_read_req[3]			
	SPI3_DMA_WRITE_0	spi3_dma_write_req[0]			SPI3 DMA Write Request
	SPI3_DMA_WRITE_1	spi3_dma_write_req[1]			
	SPI3_DMA_WRITE_2	spi3_dma_write_req[2]			
	SPI3_DMA_WRITE_3	spi3_dma_write_req[3]			
SPI4	SPI4_DMA_READ_0	spi4_dma_read_req[0]	EDMA Crossbar (EDMA_XBAR)	Pulse	SPI4 DMA Read Request
	SPI4_DMA_READ_1	spi4_dma_read_req[1]			
	SPI4_DMA_READ_2	spi4_dma_read_req[2]			
	SPI4_DMA_READ_3	spi4_dma_read_req[3]			
	SPI4_DMA_WRITE_0	spi4_dma_write_req[0]			SPI4 DMA Write Request
	SPI4_DMA_WRITE_1	spi4_dma_write_req[1]			
	SPI4_DMA_WRITE_2	spi4_dma_write_req[2]			
	SPI4_DMA_WRITE_3	spi4_dma_write_req[3]			
SPI5	SPI5_DMA_READ_0	spi5_dma_read_req[0]	EDMA Crossbar (EDMA_XBAR)	Pulse	SPI5 DMA Read Request
	SPI5_DMA_READ_1	spi5_dma_read_req[1]			
	SPI5_DMA_READ_2	spi5_dma_read_req[2]			
	SPI5_DMA_READ_3	spi5_dma_read_req[3]			
	SPI5_DMA_WRITE_0	spi5_dma_write_req[0]			SPI5 DMA Write Request
	SPI5_DMA_WRITE_1	spi5_dma_write_req[1]			
	SPI5_DMA_WRITE_2	spi5_dma_write_req[2]			
	SPI5_DMA_WRITE_3	spi5_dma_write_req[3]			
SPI6	SPI6_DMA_READ_0	spi6_dma_read_req[0]	EDMA Crossbar (EDMA_XBAR)	Pulse	SPI6 DMA Read Request
	SPI6_DMA_READ_1	spi6_dma_read_req[1]			
	SPI6_DMA_READ_2	spi6_dma_read_req[2]			
	SPI6_DMA_READ_3	spi6_dma_read_req[3]			
	SPI6_DMA_WRITE_0	spi6_dma_write_req[0]			SPI6 DMA Write Request
	SPI6_DMA_WRITE_1	spi6_dma_write_req[1]			
	SPI6_DMA_WRITE_2	spi6_dma_write_req[2]			
	SPI6_DMA_WRITE_3	spi6_dma_write_req[3]			
SPI7	SPI7_DMA_READ_0	spi7_dma_read_req[0]	EDMA Crossbar (EDMA_XBAR)	Pulse	SPI7 DMA Read Request
	SPI7_DMA_READ_1	spi7_dma_read_req[1]			
	SPI7_DMA_READ_2	spi7_dma_read_req[2]			
	SPI7_DMA_READ_3	spi7_dma_read_req[3]			
	SPI7_DMA_WRITE_0	spi7_dma_write_req[0]			SPI7 DMA Write Request
	SPI7_DMA_WRITE_1	spi7_dma_write_req[1]			
	SPI7_DMA_WRITE_2	spi7_dma_write_req[2]			
	SPI7_DMA_WRITE_3	spi7_dma_write_req[3]			

Note

For more information on the interconnects, see the *System Interconnect* chapter.

For more information on power, reset, and clock management, see the corresponding sections within the *Device Configuration* chapter.

For more information on the device interrupt controllers, see the *Interrupt Controllers* chapter.

13.1.3.4 MCSPI Functional Description

13.1.3.4.1 SPI Block Diagram

Figure 13-28 shows the SPI module.

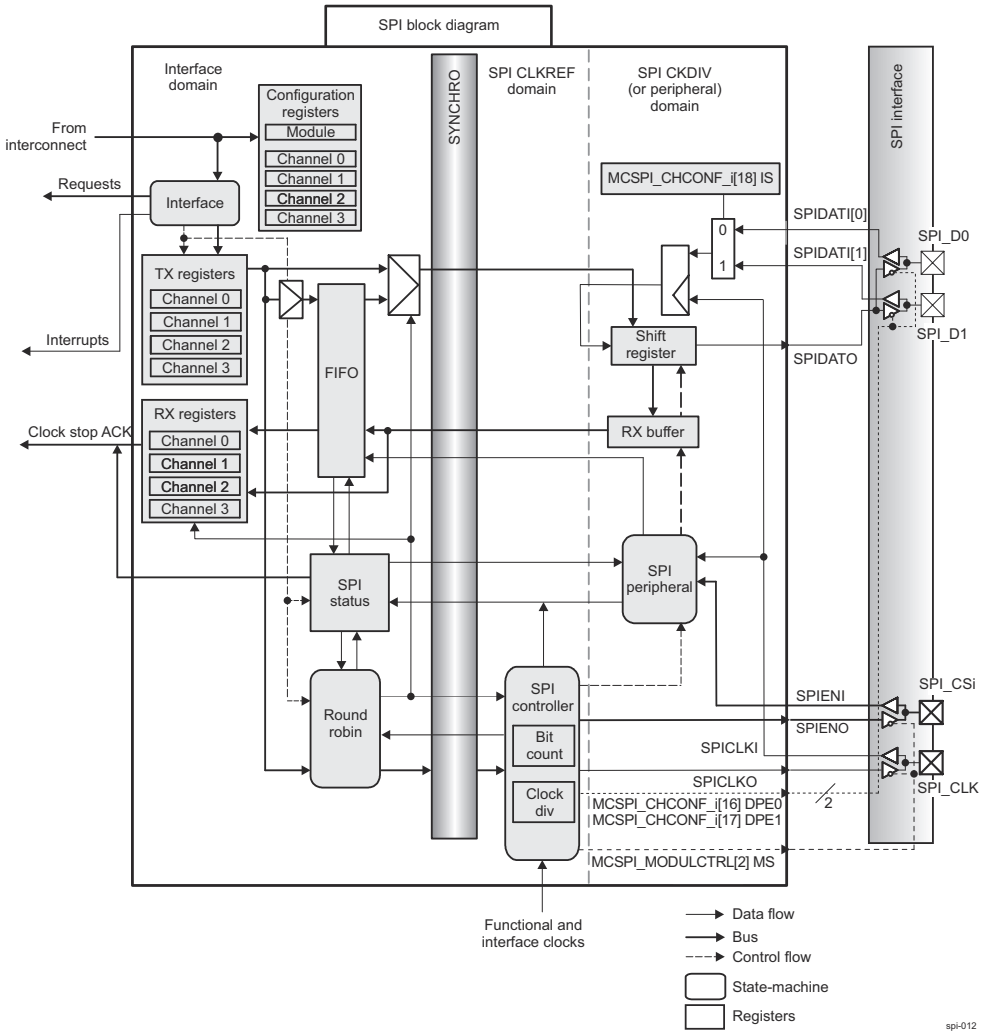


Figure 13-28. SPI Block Diagram

13.1.3.4.2 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.

13.1.3.4.3 MCSPI Controller Mode

13.1.3.4.3.1 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)

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*).

13.1.3.4.3.2 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 13-29 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.

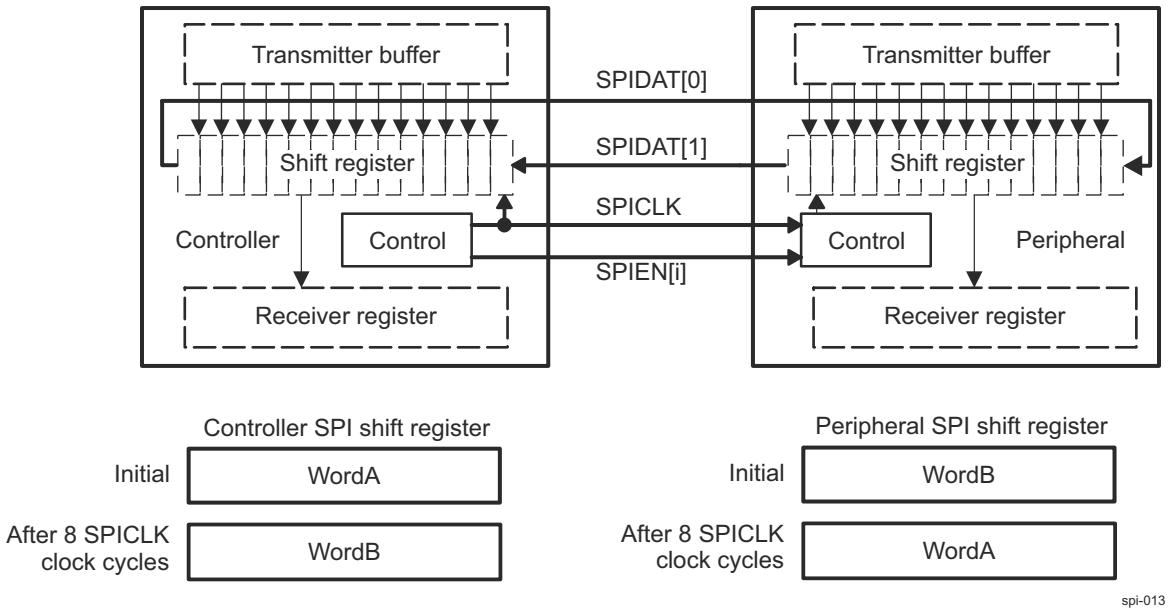


Figure 13-29. MCSPI Full-Duplex Transmission (Example)

13.1.3.4.3.3 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 13.1.3.4.3.2, apply in this mode.

Rule 3, defined in Section 13.1.3.4.3.2, 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.

13.1.3.4.3.4 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 13.1.3.4.3.2, apply in this mode.

Rule 2, defined in Section 13.1.3.4.3.2, 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.

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.

13.1.3.4.3.5 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.

13.1.3.4.3.5.1 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.

13.1.3.4.3.5.2 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.

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 13-30 and Figure 13-31 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.

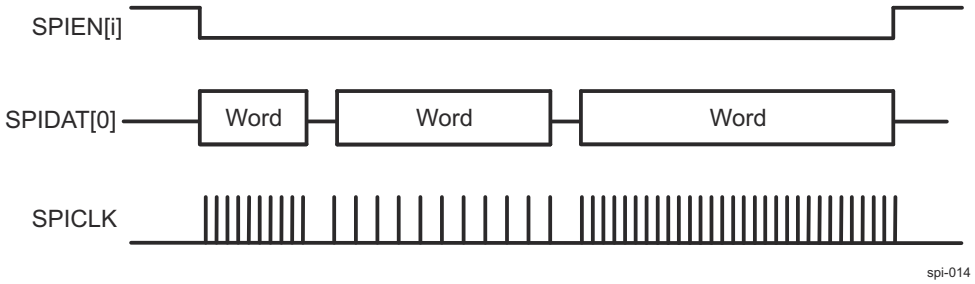


Figure 13-30. Continuous Transfers With SPIEN[i] Maintained Active (Single-Data-Pin Interface Mode)

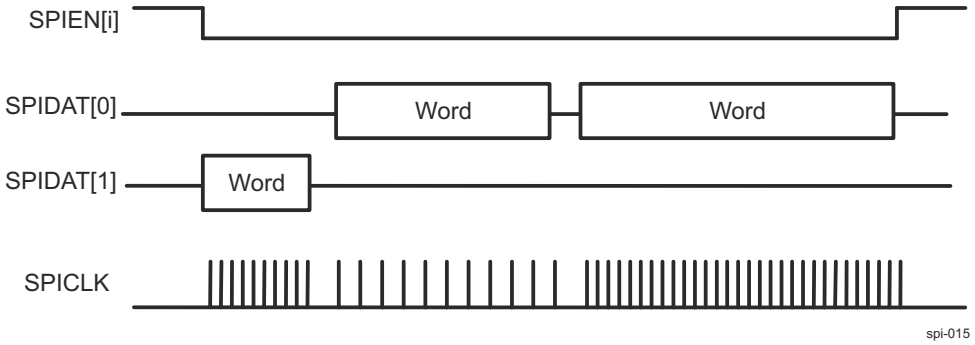


Figure 13-31. Continuous Transfers With SPIEN[i] Maintained Active (Dual-Data-Pin Interface Mode)

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.

13.1.3.4.3.5.3 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 13.1.3.4.3.2, *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.

13.1.3.4.3.6 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

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.

13.1.3.4.3.7 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 13-32 shows the CS SPIEN timing controls.

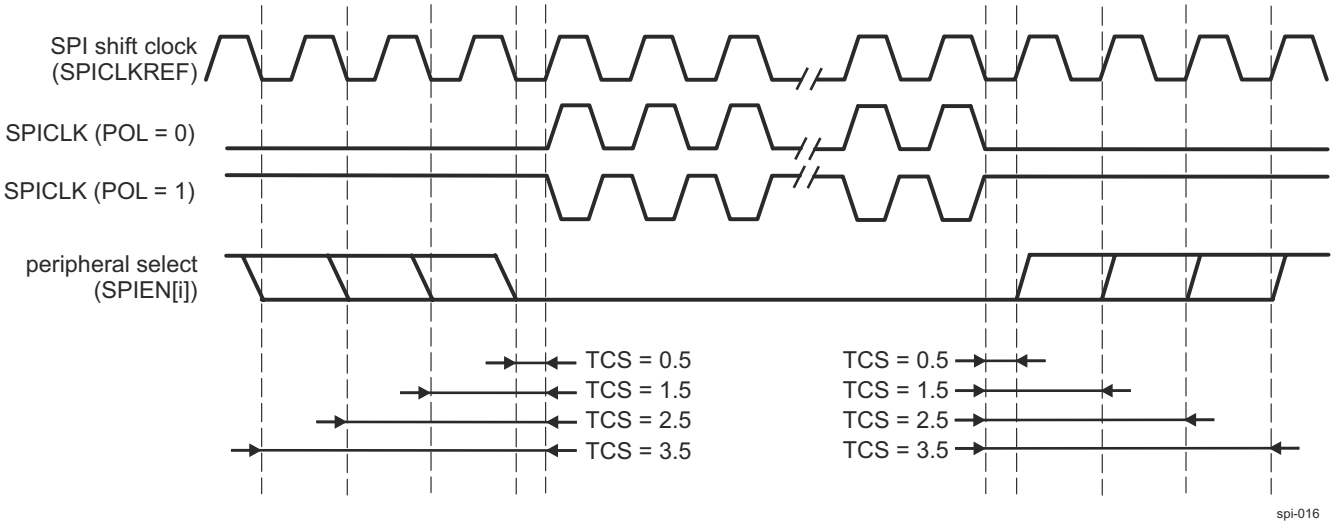


Figure 13-32. CS SPIEN Timing Controls

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.

13.1.3.4.3.8 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 13-31 summarizes the supported divisor values.

Table 13-31. MCSPI Controller Clock Rates

Divider	Clock Rate
1	50 MHz(1)
2	25 MHz(1)
4	12.5 MHz
8	6.25 MHz

Table 13-31. 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

13.1.3.4.3.8.1 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 13-32).

Table 13-32. 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.

For odd ratio values, the duty cycle is calculated as follows:

Duty_cycle = $(1 – 1/F_{RATIO})/2$

Table 13-33 shows examples of clock granularity with a clock source frequency of 50 MHz.

Table 13-33. 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

13.1.3.4.4 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.

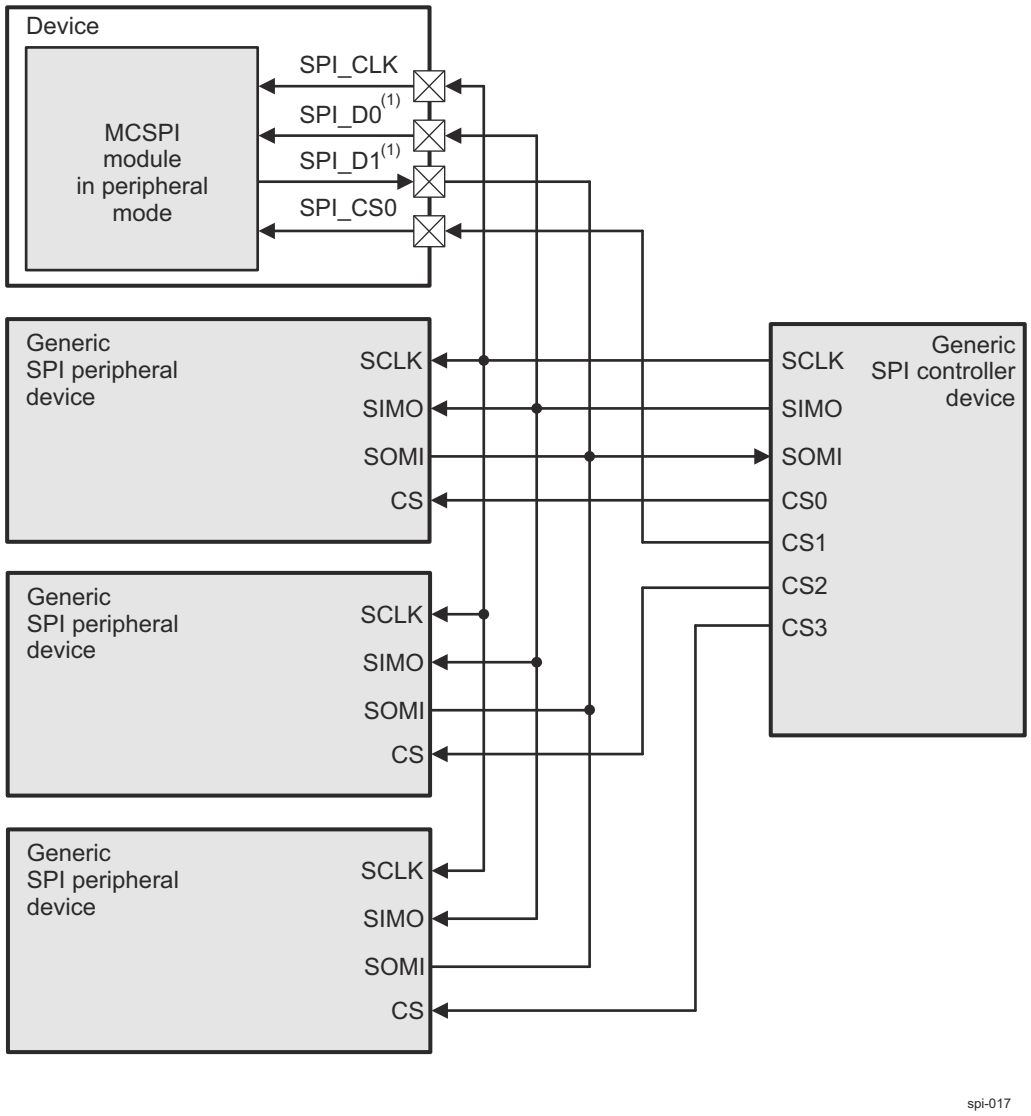
When the MCSPI_MODULCTRL[1] PIN34 is set to 0x0, the MCSPI does not support SPIEN[i] active between MCSPI words. In this case, the MCSPI uses the edge to detect word length.

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.

13.1.3.4.4.1 Dedicated Resources

Only channel 0 can be enabled in peripheral mode.

Figure 13-33 shows an example of four peripherals wired on a single controller device.



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 13-33. 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.

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 13.1.3.4.7.2, *Interrupt Events in Peripheral Mode*).

13.1.3.4.4.2 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 13.1.3.4.7.2, *Interrupt Events in Peripheral Mode*).

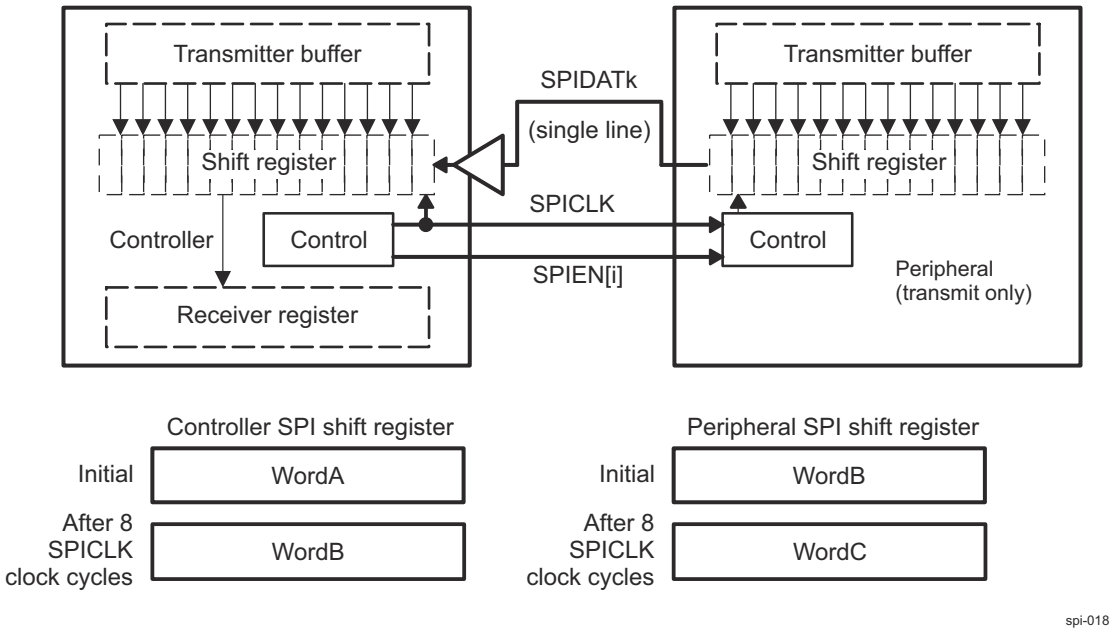
13.1.3.4.4.3 Peripheral Transmit-Only Mode

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 13-34 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.



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 13-34. MCSPI Half-Duplex Transmission (Transmit-Only peripheral)

13.1.3.4.4.4 Peripheral Receive-Only Mode

The peripheral receive mode is programmable (set the MCSPI_CHCONF_0[13-12] TRM bit field to 0x1).

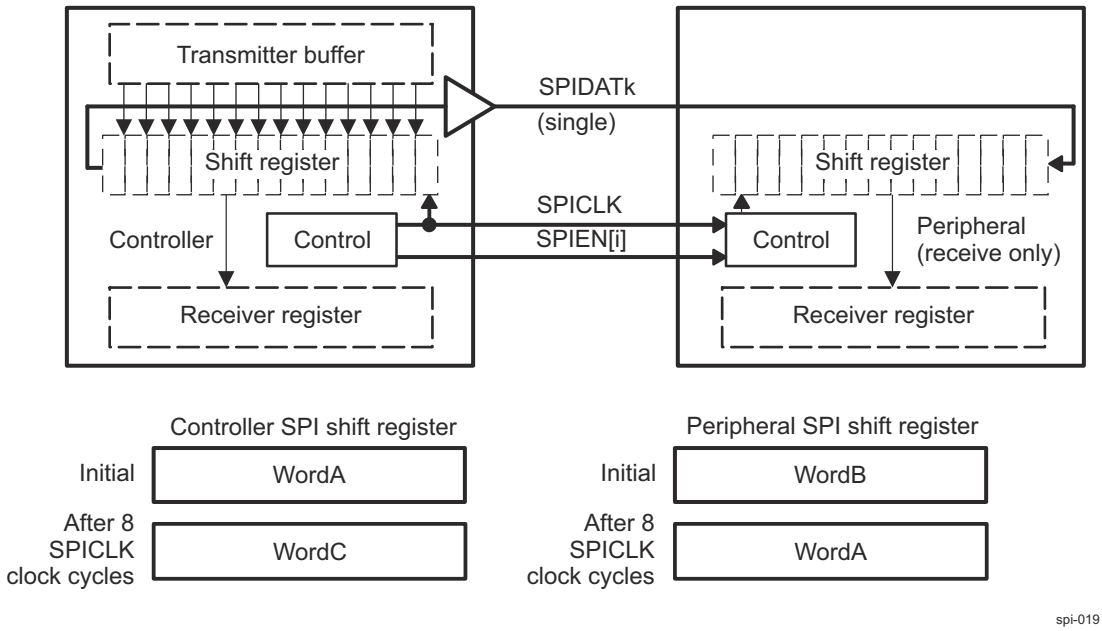
In receive-only mode, the MCSPI_TX_0 register must be loaded before the MCSPI is selected by an external MCSPI controller device. The MCSPI_TX_0 register content is always loaded into the shift register whether it is updated or not. The TX0_UNDERFLOW event is activated accordingly and does not prevent 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 13-35 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.



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 13-35. MCSPI Half-Duplex Transmission (Receive-Only Peripheral)

13.1.3.4.5 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.

13.1.3.4.6 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 13-36) or receive (see Figure 13-37) 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 13-38).

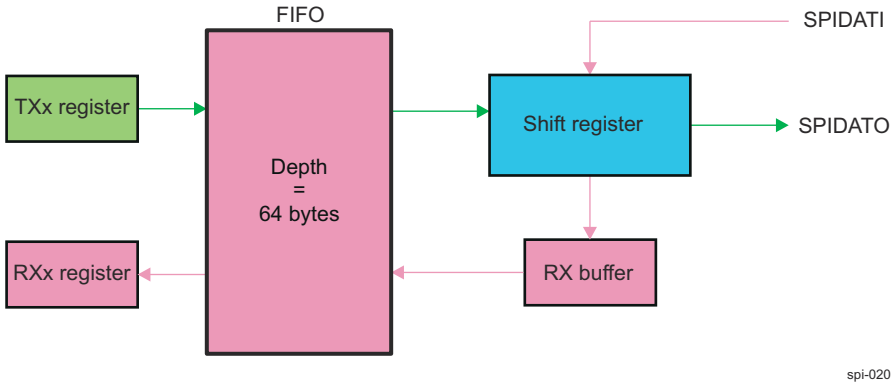


Figure 13-36. Buffer Used in Transmit Direction Only

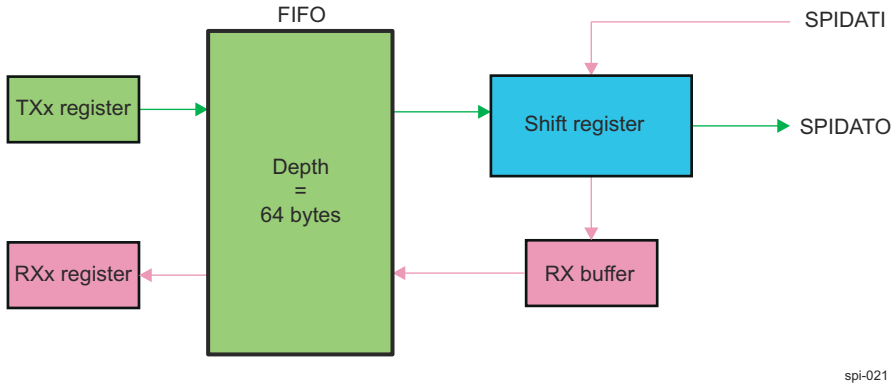


Figure 13-37. Buffer Used in Receive Direction Only

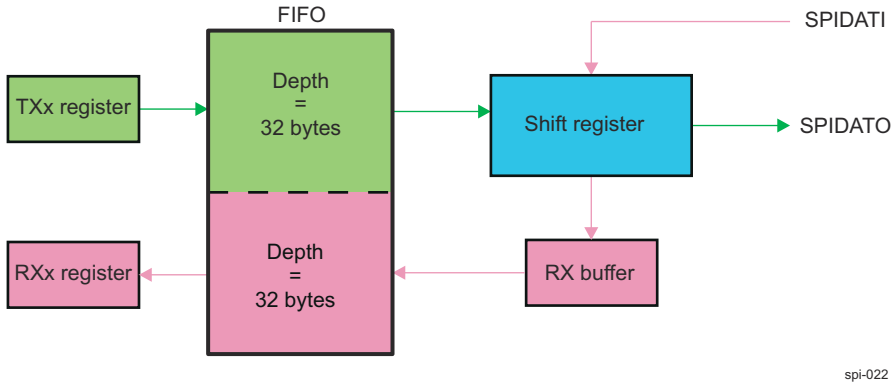


Figure 13-38. Buffer Used for Transmit and Receive Directions

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 13-34 lists the number of bytes written in the FIFO, depending on the word length.

Table 13-34. 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.

13.1.3.4.6.1 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 13-39.

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.

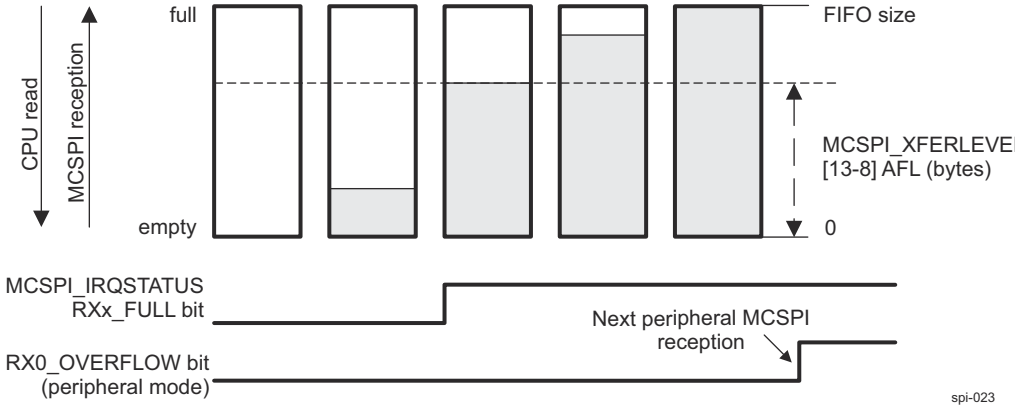


Figure 13-39. Buffer Almost Full Level (AFL)

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.

13.1.3.4.6.2 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 13-40.

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.

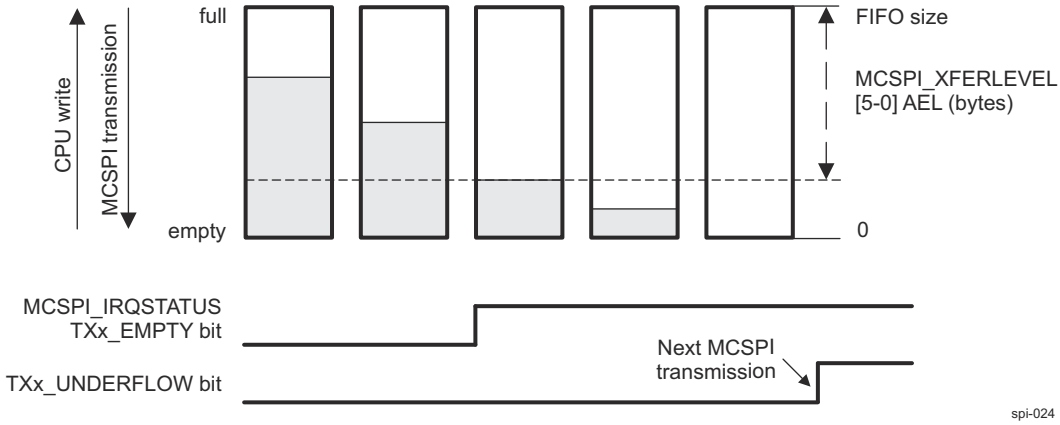


Figure 13-40. Buffer Almost Empty Level (AEL)

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.

13.1.3.4.6.3 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.

13.1.3.4.6.4 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.

- 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.

13.1.3.4.6.5 First MCSPI Word Delay

Figure 13-41 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.

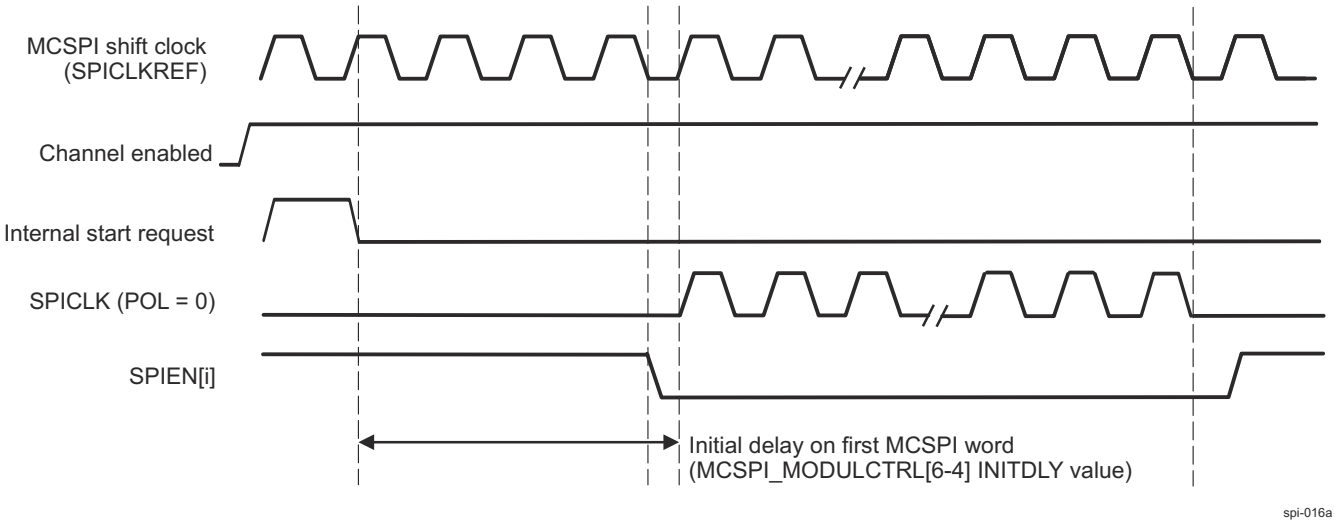


Figure 13-41. Controller Single Channel Initial Delay

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.

13.1.3.4.7 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.

13.1.3.4.7.1 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.

13.1.3.4.7.1.1 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 13.1.3.4.3.4, *Controller Receive-Only Mode*). When the FIFO buffer is enabled (the

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.

13.1.3.4.7.1.2 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).

13.1.3.4.7.1.3 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.

13.1.3.4.7.1.4 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).

13.1.3.4.7.2 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.

13.1.3.4.7.2.1 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

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.

13.1.3.4.7.2.2 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).

13.1.3.4.7.2.3 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.

13.1.3.4.7.2.4 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).

13.1.3.4.7.2.5 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.

The MCSPI_IRQSTATUS[17] EOW interrupt status bit must be cleared for interrupt line deassertion (if the event is enabled as the interrupt source).

13.1.3.4.7.3 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.

13.1.3.4.7.4 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.

13.1.3.4.8 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.

13.1.3.4.9 MCSPI Power Saving Management

Power consumption can be optimized by switching off internal clocks (interface and functional clock) when there is no activity.

13.1.3.4.9.1 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.

- 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.

13.1.3.4.9.2 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.

13.1.3.4.9.2.1 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.

13.1.3.5 MCSPI Programming Guide

This section describes the low-level hardware programming sequences for the configuration and use of the MCSPI module.

13.1.3.5.1 MCSPI Global Initialization

13.1.3.5.1.1 MCSPI Global Initialization

13.1.3.5.1.1.1 Main Sequence – MCSPI Global Initialization

The procedure in Table 13-35 can be used to initialize MCSPI when performing software reset.

Table 13-35. MCSPI Global Initialization

Step	Register/Bit Field/Programming Model	Value
Perform a software reset.	MCSPI_SYSCONFIG[1] SOFTRESET	1
Wait until reset is finished?	MCSPI_SYSSTATUS[0] RESETDONE	=1
Configure static settings (such as SPI controller or peripheral) as required.	MCSPI_MODULCTRL[8-0]	0x-
Write MCSPI_SYSCONFIG	MCSPI_SYSCONFIG	0x-

13.1.3.5.2 MCSPI Operational Mode Configuration

13.1.3.5.2.1 MCSPI Operational Modes

The selection of the working mode is done with the MCSPI_CHCONF_0/1/2/3 register.

Table 13-36. 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 13-37. 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 13-38. 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

13.1.3.5.2.1.1 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 13-39 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 13-39. 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		
Stop the channel	MCSPI_CHCTRL_0/1/2/3[0] EN	0

13.1.3.5.2.1.2 End of Transfer Sequences

The end of transfer depends on the transfer mode. Table 13-40 summarizes the type of end of transfer per transfer mode and gives a reference to the appropriate section for details.

Table 13-40. 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 13.1.3.5.2.1.3		See Section 13.1.3.5.2.1.4.1	See Section 13.1.3.5.2.1.4.2	See Section 13.1.3.5.2.1.5.1	See Section 13.1.3.5.2.1.5.2
	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 13.1.3.5.2.1.3		See Section 13.1.3.5.2.1.4.1	See Section 13.1.3.5.2.1.4.2	See Section 13.1.3.5.2.1.6.1	See Section 13.1.3.5.2.1.6.2
	Minimum number of word	1	1	1	1	2	3
	DMA transfer size		N		N		N-2
PERIPHERAL	End of transfer sequence	See Section 13.1.3.5.2.1.3		See Section 13.1.3.5.2.1.4.1	See Section 13.1.3.5.2.1.4.2	See Section 13.1.3.5.2.1.7	
	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.

13.1.3.5.2.1.3 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 13-41. 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 13-42. 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
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		

13.1.3.5.2.1.4 Transmit-Only (Controller and Peripheral)

13.1.3.5.2.1.4.1 Based on Interrupt Requests

Table 13-43. 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 13-44. 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		

13.1.3.5.2.1.4.2 Based on DMA Write Requests

When the DMA handler has completed its ‘N’ CBASS0 accesses, write_count is assigned with ‘N’.

Table 13-45. 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 13-46. 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		

13.1.3.5.2.1.5 Controller Normal Receive-Only

13.1.3.5.2.1.5.1 Based on Interrupt Requests

Table 13-47. 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 13-48. 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		

13.1.3.5.2.1.5.2 Based on DMA Read Requests

When the DMA handler has completed its ‘N-1’ CBASS0 accesses, read_count is assigned with ‘N-1’.

Table 13-49. 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 13-50. 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		

13.1.3.5.2.1.6 Controller Turbo Receive-Only

13.1.3.5.2.1.6.1 Based on Interrupt Requests

Table 13-51. 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
Stop the channel	MCSPI_CHCTRL_0/1/2/3[0] EN	0
Wait until channel_enable = FALSE		

Table 13-52. 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		

13.1.3.5.2.1.6.2 Based on DMA Read Requests

When the DMA handler has completed its ‘N-2’ CBASS0 accesses read_count is assigned with ‘N-2‘.

Table 13-53. 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 13-54. 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		

Table 13-54. Receive-Only With DMA (Controller Turbo) (Interrupt Routine) (continued)

Step	Register/Bit Field/Programming Model	Value
Read the receiver register	MCSPI_RX_0/1/2/3	0x-
Increment read_count +1		
ENDIF		
ENDIF		

13.1.3.5.2.1.7 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 13-55. 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 13-56. 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		

13.1.3.5.2.1.8 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 13.1.3.4.6, *MCSPI FIFO Buffer Management*.

Table 13-57. 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

13.1.3.5.2.1.8.1 Common Transfer Sequence in FIFO Mode

This flow describes the host sequence for a transfer of any type defined in Section 13.1.3.5.2.1.8, *Transfer Procedures With FIFO*.

In multi-channel, only one channel can use the FIFO.

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).

13.1.3.5.2.1.8.2 End of Transfer Sequences in FIFO Mode

Table 13-58 summarizes the type of end of transfer per transfer mode and gives a reference to the appropriate section for details.

Table 13-58. End of Transfer Sequences in FIFO Mode

Word count	TRANSMIT AND RECEIVE	TRANSMIT-ONLY	RECEIVE-ONLY
Yes	See Figure 13-42	See Figure 13-44	See Figure 13-45
No	See Figure 13-43	See Figure 13-44	See Figure 13-46

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.

13.1.3.5.2.1.8.3 Transmit-and-Receive With Word Count

Figure 13-42 shows the flow of a transfer in transmit-and-receive mode, with word count.

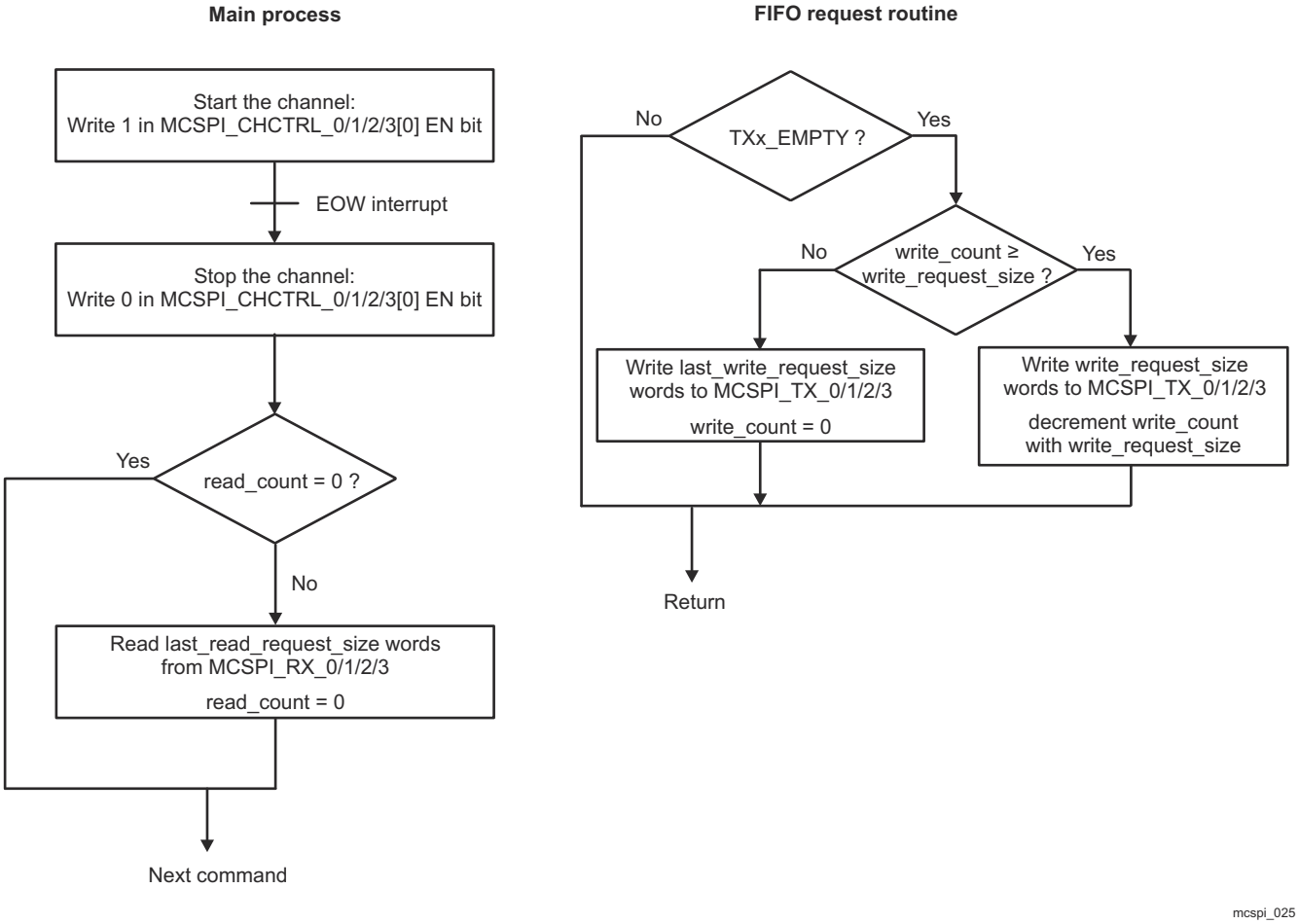


Figure 13-42. FIFO Mode Transmit-and-Receive With Word Count (Controller)

13.1.3.5.2.1.8.4 Transmit-and-Receive Without Word Count

Figure 13-43 shows the flow of a transfer in transmit-and-receive mode, without word count.

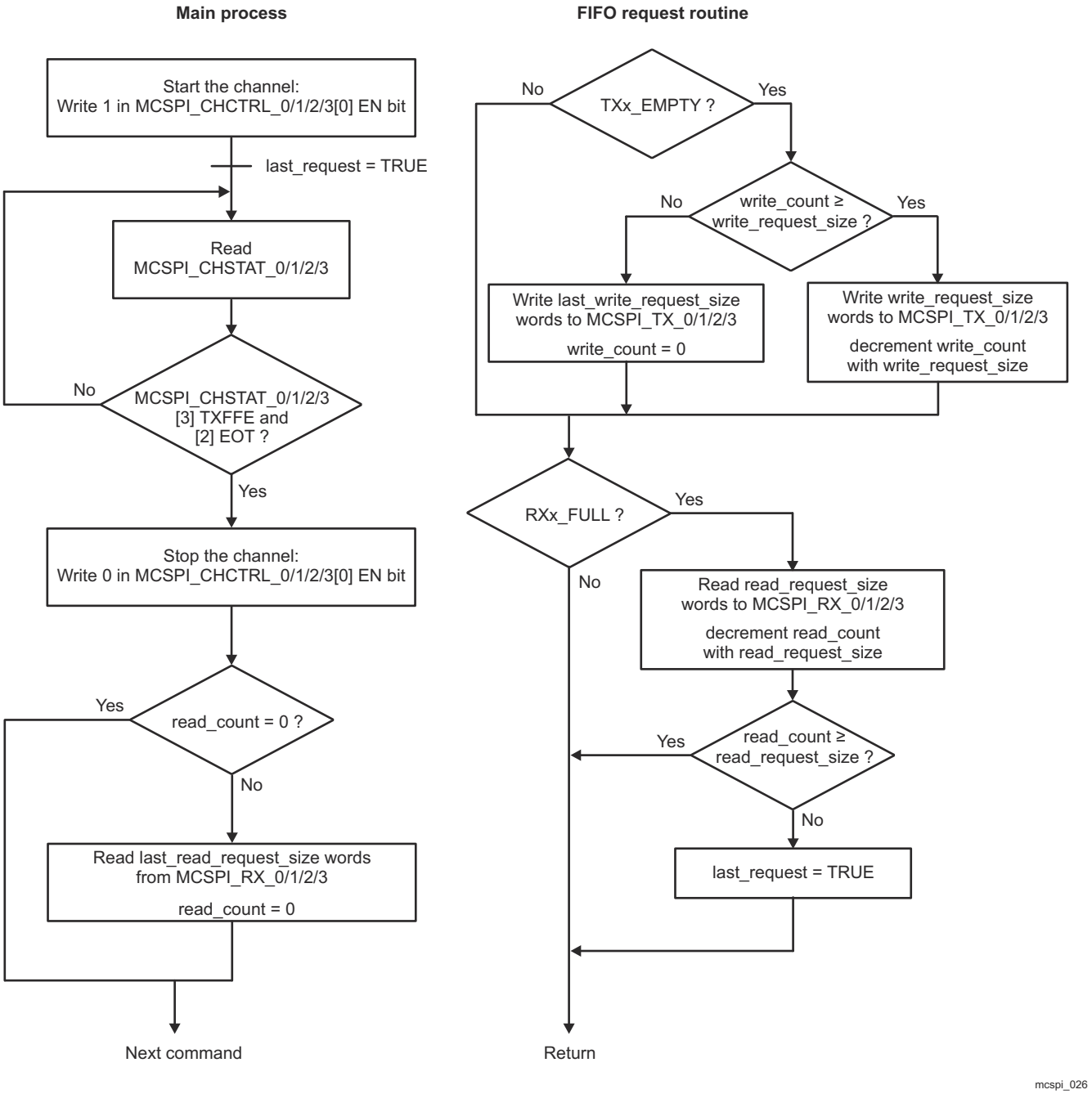


Figure 13-43. FIFO Mode Transmit-and-Receive Without Word Count (Controller)

13.1.3.5.2.1.8.5 Transmit-Only

Figure 13-44 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

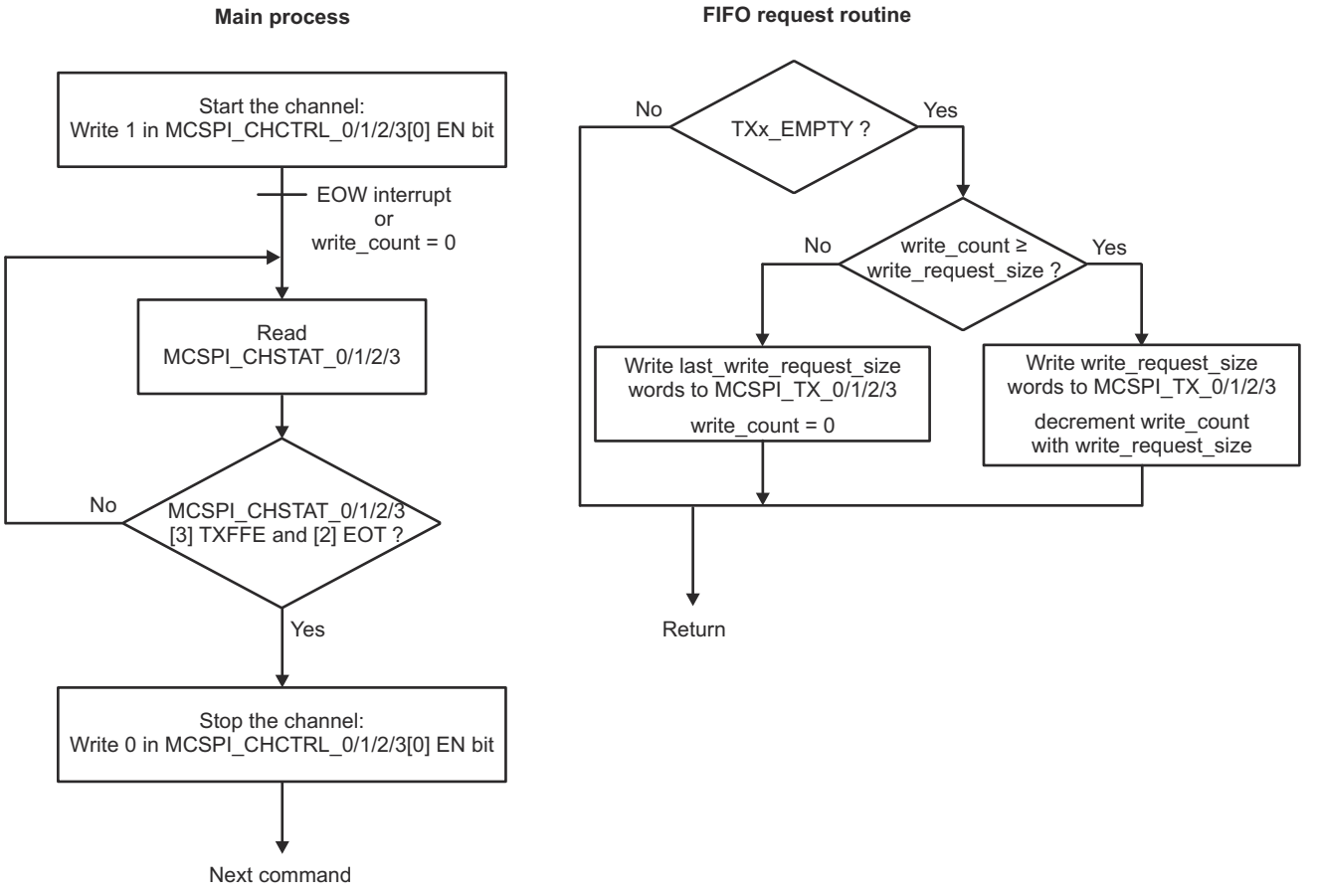


Figure 13-44. FIFO Mode Transmit-Only (Controller)

13.1.3.5.2.1.8.6 Receive-Only With Word Count

Figure 13-45 shows the flow of a transfer in receive-only mode, with word count.

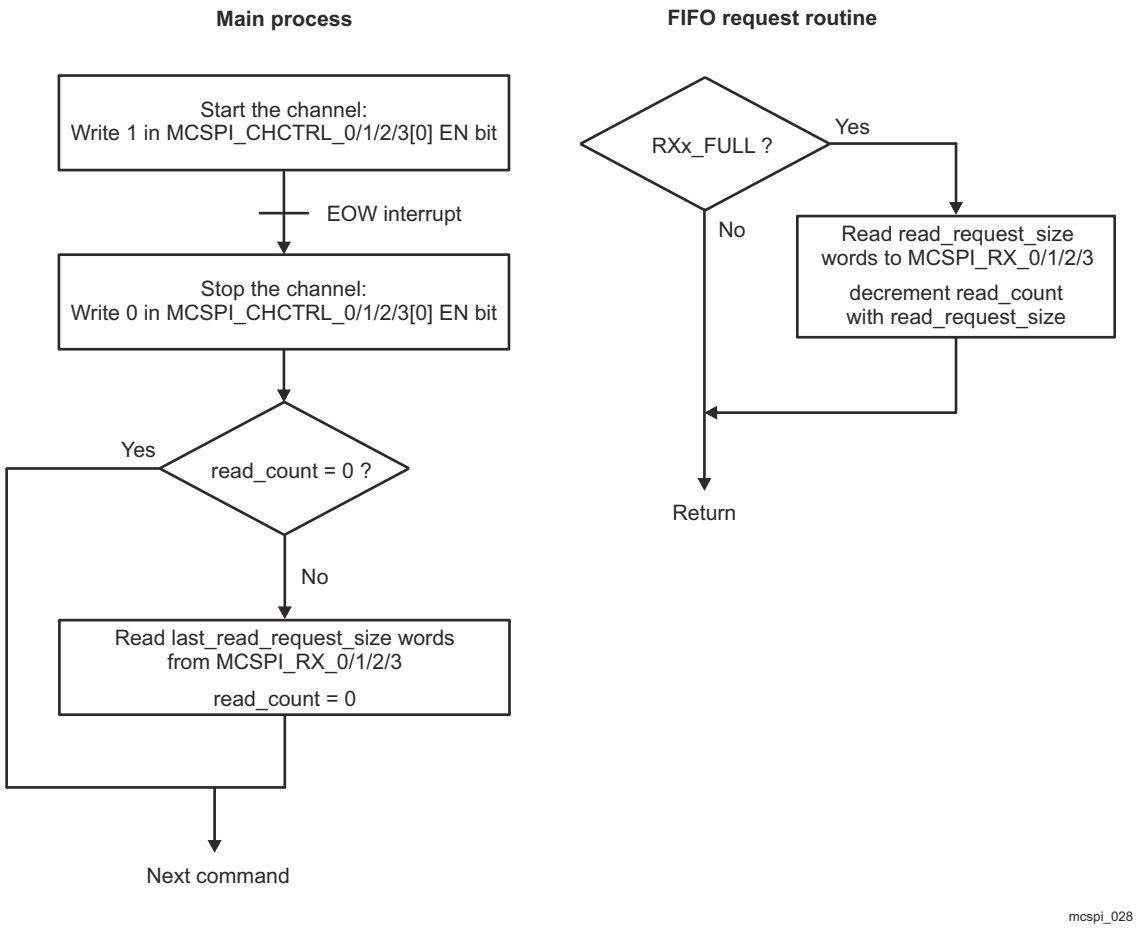


Figure 13-45. FIFO Mode Receive-Only With Word Count (Controller)

13.1.3.5.2.1.8.7 Receive-Only Without Word Count

Figure 13-46 shows the flow of a transfer in receive-only mode, with word count.

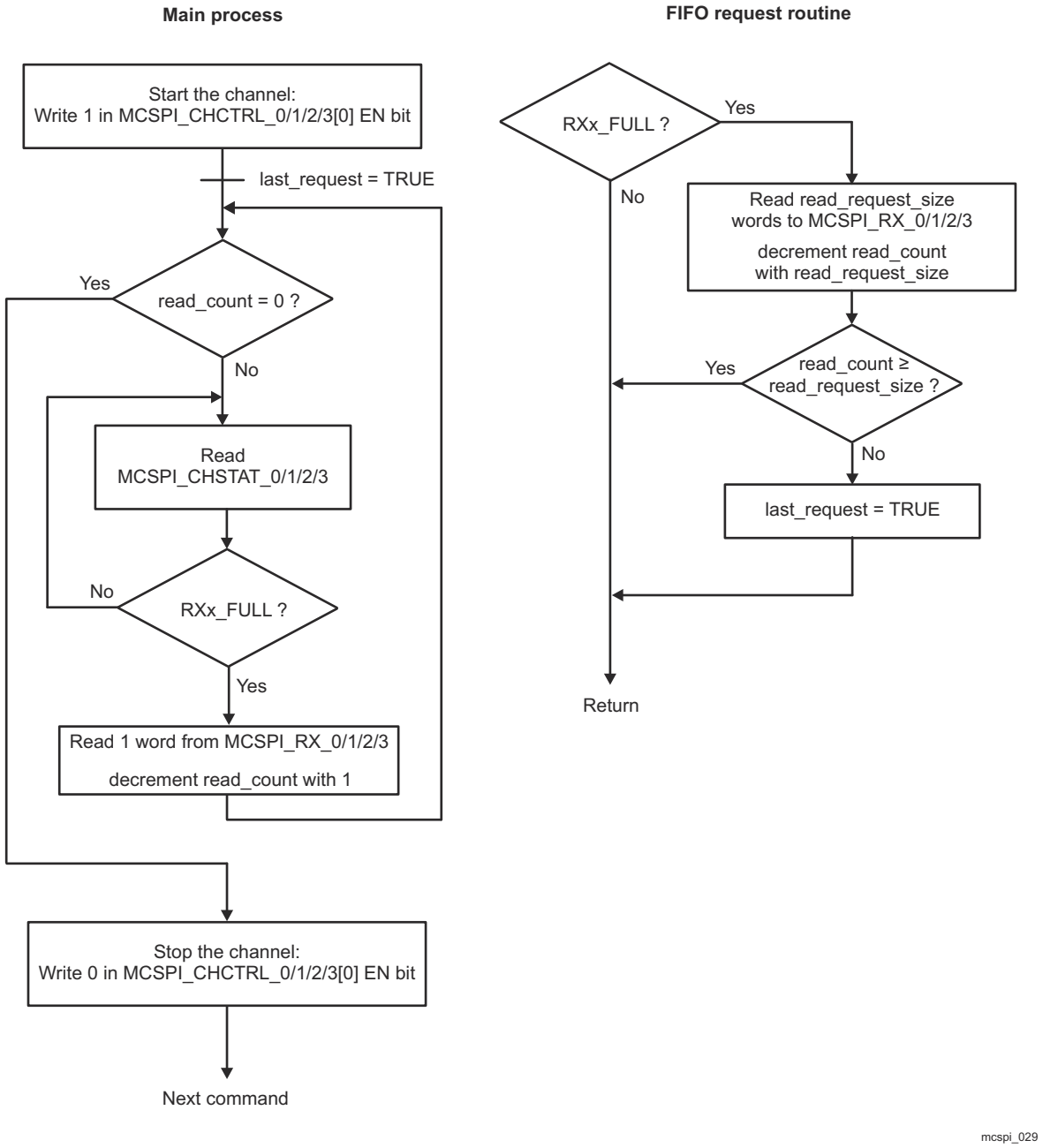


Figure 13-46. FIFO Mode Receive-Only Without Word Count (Controller)

13.1.3.5.2.1.9 Common Transfer Procedures Without FIFO – Polling Method

13.1.3.5.2.1.9.1 Receive-Only Procedure – Polling Method

Table 13-59 lists the receive-only procedure using the polling method.

Table 13-59. Receive-Only Procedure – Polling Method

Step	Register/Bit Field/Programming Model	Value
Configure the channel according to the mode.	See Table 13-36.	
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

13.1.3.5.2.1.9.2 Receive-Only Procedure – Interrupt Method

Table 13-60 lists the receive-only procedure using the interrupt method.

Table 13-60. Receive-Only Procedure – Interrupt Method

Step	Register/Bit Field/Programming Model	Value
Configure the channel according to the mode.	See Table 13-36.	
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-

13.1.3.5.2.1.9.3 Transmit-Only Procedure – Polling Method

Table 13-61 lists the transmit-only procedure using the polling method.

Table 13-61. Transmit-Only Procedure – Polling Method

Step	Register/Bit Field/Programming Model	Value
Configure the channel according to the mode.	See Table 13-37.	
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

13.1.3.5.2.1.9.4 Transmit-and-Receive Procedure – Polling Method

Table 13-62 lists the transmit-and-receive procedure using the polling method.

Table 13-62. Transmit-and-Receive Procedure – Polling Method

Step	Register/Bit Field/Programming Model	Value
Configure the channel according to the mode.	See Table 13-38.	
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-

13.1.3.5.3 Common Transfer Procedures Without FIFO – Polling Method

13.1.3.5.3.1 Receive-Only Procedure – Polling Method

Table 13-63 lists the receive-only procedure using the polling method.

Table 13-63. Receive-Only Procedure – Polling Method

Step	Register/Bit Field/Programming Model	Value
Configure the channel according to the mode.	See Table 13-36.	
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

13.1.3.5.3.2 Receive-Only Procedure – Interrupt Method

Table 13-64 lists the receive-only procedure using the interrupt method.

Table 13-64. Receive-Only Procedure – Interrupt Method

Step	Register/Bit Field/Programming Model	Value
Configure the channel according to the mode.	See Table 13-36.	
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-

13.1.3.5.3.3 Transmit-Only Procedure – Polling Method

Table 13-65 lists the transmit-only procedure using the polling method.

Table 13-65. Transmit-Only Procedure – Polling Method

Step	Register/Bit Field/Programming Model	Value
Configure the channel according to the mode.	See Table 13-37.	
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

13.1.3.5.3.4 Transmit-and-Receive Procedure – Polling Method

Table 13-66 lists the transmit-and-receive procedure using the polling method.

Table 13-66. Transmit-and-Receive Procedure – Polling Method

Step	Register/Bit Field/Programming Model	Value
Configure the channel according to the mode.	See Table 13-38.	
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-

13.1.4 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.

13.1.4.1 UART Overview

The UART is a target peripheral that utilizes the DMA for data transfer or interrupt polling via host CPU. There are six UART modules in the device. Each module can be used in the following modes: UART, IrDA, CIR or ISO 7816. The UART modules support IrDA and CIR modes when a 48 MHz functional clock frequency is used. Each UART can be used for configuration and data exchange with a number of external peripheral devices or interprocessor communication between devices.

13.1.4.1.1 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:
  - Data bit: 5, 6, 7, or 8 bits
  - 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 UART1 module instance has extended modem control signals (DCD, RI, DTR, DSR)
- Independent TX/RX
- Supports both little and big Endian operating mode
- Internal test and loopback capabilities

13.1.4.1.2 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

13.1.4.1.3 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

13.1.4.1.4 ISO 7816 (Smartcard) Functions

ISO 7816 mode is a half duplex protocol that uses the TX line of the UART peripheral in a bidirectional mode. It is a low-level interface supporting protocols T=0 and T=1. The features of this mode are listed below:

- Includes features to control repetition
- Acknowledges cases of parity mismatch in T=0 mode

13.1.4.2 UART Environment

The UART[0-5] modules are hereinafter referred to as UART module.

This section describes the UART/RS-485/IrDA/CIR external connections (environment).

- The UART interface is described in Section 13.1.4.2.1, *UART Functional Interfaces*.
- The RS-485 interface is described in Section 13.1.4.2.2, *RS-485 Functional Interfaces*.
- The IrDA interface is described in Section 13.1.4.2.3, *IrDA Functional Interfaces*.
- The CIR interface is described in Section 13.1.4.2.4, *CIR Functional Interfaces*.

13.1.4.2.1 UART Functional Interfaces

13.1.4.2.1.1 System Using UART Communication With Hardware Handshake

Each UART instance can be easily connected to the UART port of an external IC (see Figure 13-47).

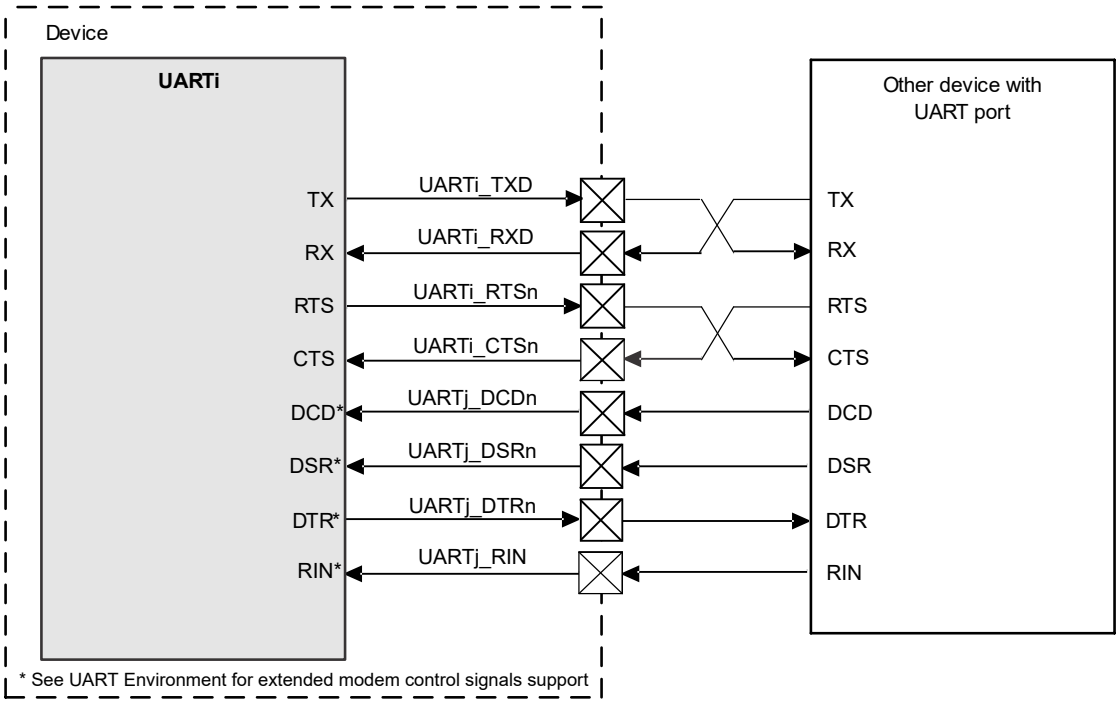


Figure 13-47. UART Mode Interface Signals

13.1.4.2.1.2 UART Interface Description

Table 13-67 lists the UART interface input/output (I/O) signals.

Table 13-67. UART I/O Signals

Module Pin Name	Device Level Signal Name	I/O[1]	Description	Module Pin Reset Value[2]
UART[0-5]				
RX	UART[0-5]_RXD	I	Serial data input	HiZ
TX	UART[0-5]_TXD	O	Serial data output[3]	1
CTS	UART[0-5]_CTSn	I	Clear to send[4]	HiZ
RTS	UART[0-5]_RTSn	O	Request to send[5]	1
UART1 Modem Signals				
DCD	UART1_DCDn	I	Data Carrier Detect[6]	HiZ
DSR	UART1_DSRn	I	Data Set Ready[7]	HiZ
DTR	UART1_DTRn	O	Data Terminal Ready[8]	1
RIN	UART1_RIN	I	Ring Indicator[9]	HiZ

(1) I = Input; O = Output
(2) HiZ = High Impedance
(3) 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).
(4) 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.
(5) 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.
(6) 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.
(7) 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.
(8) 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.
(9) 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.

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.

13.1.4.2.1.3 UART Protocol and Data Format

The UART device operates in three modes:

- UART 16× (<= 230.4 kbps)
- UART 16× with autobauding (>= 1200 bps and <= 115.2 kbps)
- UART 13× (>= 460.8 kbps)

CAUTION

To be used as a UART, the operating mode must be programmed appropriately in the UART_MDR1[2-0] MODE_SELECT bit field to select UART, IrDA, or CIR mode, and the UART_MDR3[4] DIR_EN bit field to select RS-485 mode.

The UART uses a wired interface for serial communication with a remote device.

The UART is functionally compatible with the TL16C750 UART and earlier designs such as the TL16C550.

Figure 13-48 shows the UART frame data format.

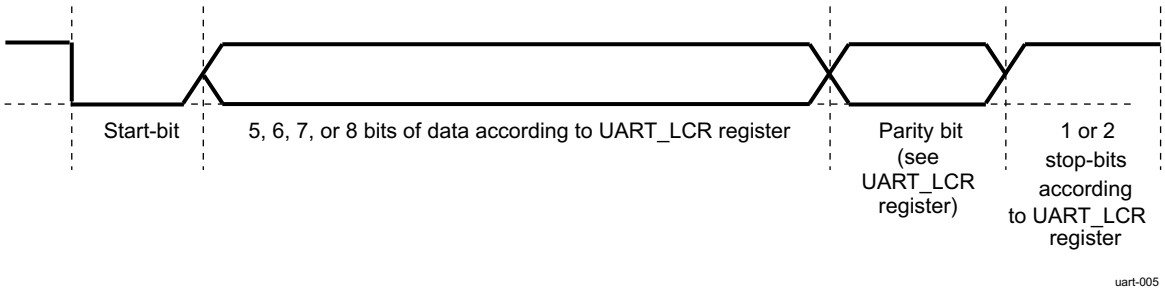


Figure 13-48. UART Frame Data Format

13.1.4.2.2 RS-485 Functional Interfaces

13.1.4.2.2.1 System Using RS-485 Communication

The RS-485 network physical layer consists of two-wire differential bus, usually twisted pair. External RS-485 transceiver IC is needed to access a RS-485 bus by the RS-485 mode. Figure 13-49 shows an example connection of UART in RS-485 mode.

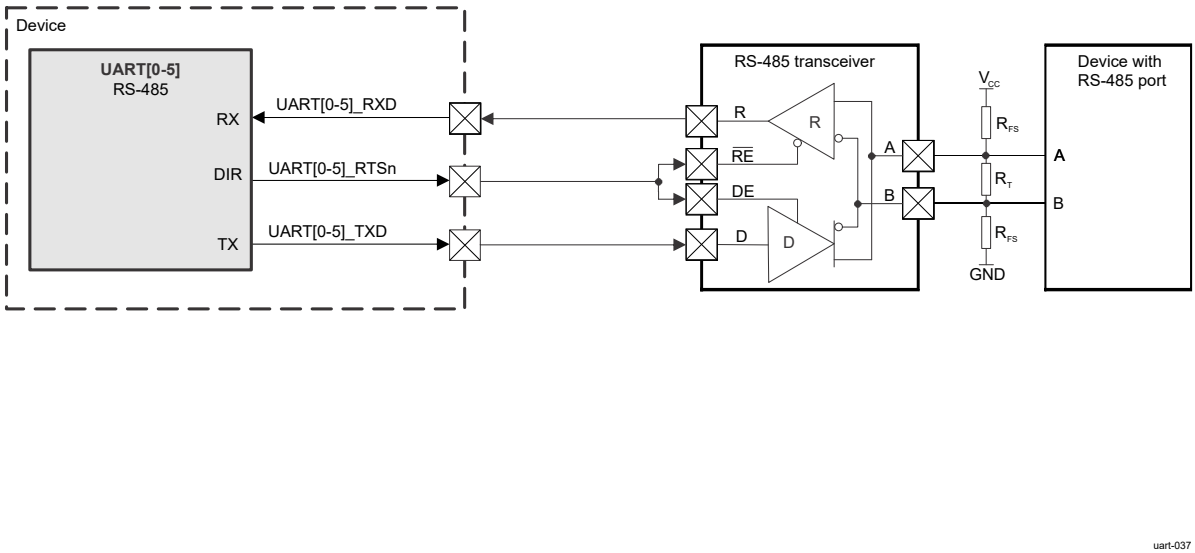


Figure 13-49. RS-485 Mode Interface Signals

13.1.4.2.2.2 RS-485 Interface Description

Table 13-68 lists the RS-485 interface input/output (I/O) signals.

Table 13-68. UART I/O Signals (RS-485 Mode)

Module Pin	Device Level Signal	I/O[1]	Description	Module Pin Reset Value[2]
			UART[0-5]	

Table 13-68. UART I/O Signals (RS-485 Mode) (continued)

Module Pin	Device Level Signal	I/O(1)	Description	Module Pin Reset Value(2)
RX	UART[0-5]_RXD	I	Serial data input	HiZ
TX	UART[0-5]_TXD	O	Serial data output	1
DIR	UART[0-5]_RTSn	O	RS-485 Direction	1

(1) I = Input; O = Output
(2) HiZ = High Impedance

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.

13.1.4.2.3 IrDA Functional Interfaces

13.1.4.2.3.1 System Using IrDA Communication Protocol

Figure 13-50 shows an example connection of UART0 to an external infrared transceiver in the IrDA modes (FIR, SIR, and MIR).

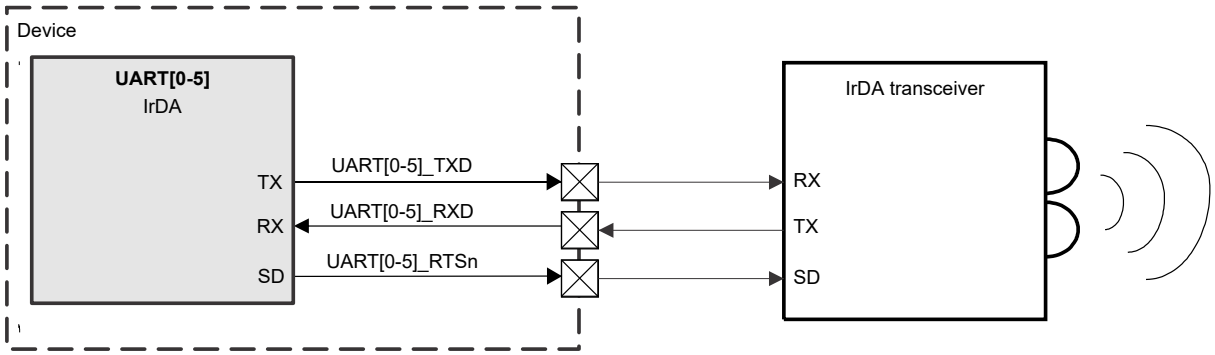


Figure 13-50. IrDA Mode Interface Signals

13.1.4.2.3.2 IrDA Interface Description

Table 13-69 lists the IrDA interface I/O signals.

Table 13-69. UART I/O Signals (IrDA Mode)

Module Pin	Device Level Signal	I/O[1]	Description	Module Pin Reset Value[2]
		UART[0-5]		
RX	UART[0-5]_RXD	I	Serial data input	HiZ
TX	UART[0-5]_TXD	O	Serial data output in IrDA modes (SIR, MIR, and FIR).[3]	0
SD	UART[0-5]_RTSn	O	SD mode is used to configure the transceivers.[4]	1

(1) I = Input; O = Output
(2) HiZ = High Impedance
(3) In other modes, this pin is set to the reset value (inactive state).
(4) The SD pinout (see UART_ACREG[6] SD_MOD bit).

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.

13.1.4.2.3.3 IrDA Protocol and Data Format

13.1.4.2.3.3.1 SIR Mode

In SIR mode, data is transferred between the Host CPU and peripheral devices at speeds of up to 115200 baud. A SIR transmit frame begins with start flags (a single 0xC0, a multiple 0xC0, or a single 0xC0 preceded by a number of 0xFF flags), followed by frame data and a CRC-16, and ends with a stop flag (0xC1).

The bit format for a single word uses 1 start-bit, 8 data bits, and 1 stop-bit, and is unaffected by the use and settings of the UART_LCR register.

The UART_BLR[6] XBOF_TYPE bit selects whether the 0xC0 or 0xFF start patterns are used when multiple start flags are required.

The SIR transmit state-machine attaches start flags, CRC-16, and stop flags, and checks the outgoing data to establish whether data transparency is required.

The SIR transparency is carried out if the outgoing data between the start and stop flags contains 0xC0, 0xC1, or 0x7D. If one of these start flags is about to be transmitted, the SIR state-machine sends an escape character (0x7D), inverts the fifth bit of the real data to be sent, and then sends this data immediately after the 0x7D character.

The SIR receive state-machine recovers the receive clock, removes the start flags and any transparency from the incoming data, and determines the frame boundary with reception of the stop flag. The SIR state-machine also checks for errors such as a frame abort (0x7D character followed immediately by a 0xC1 stop flag without transparency), a CRC error, or a frame-length error. At the end of a frame reception, the Host CPU reads the line status register (UART_LSR_IRDA) to find possible errors of the received frame.

Note

The module can transmit and receive data, but when the device is transmitting, the IR RX circuitry is automatically disabled by hardware. See the description of the UART_ACREG[5] DIS_IR_RX bit. This applies to all three modes: SIR, MIR, and FIR.

Infrared output in SIR mode can be 1.6-µs or 3/16 encoding, selected by the UART_ACREG[7] PULSE_TYPE bit. In 1.6-µs encoding, the infrared pulse width is 1.6 µs; and in 3/16th encoding, the infrared pulse width is 3/16th of a bit duration (1/baud rate).

For back-to-back frames, the transmitting device must send at least two start flags at the start of each frame.

Note

Reception supports variable-length stop-bits.

13.1.4.2.3.3.1.1 Frame Format

Figure 13-51 shows the IrDA SIR frame format.

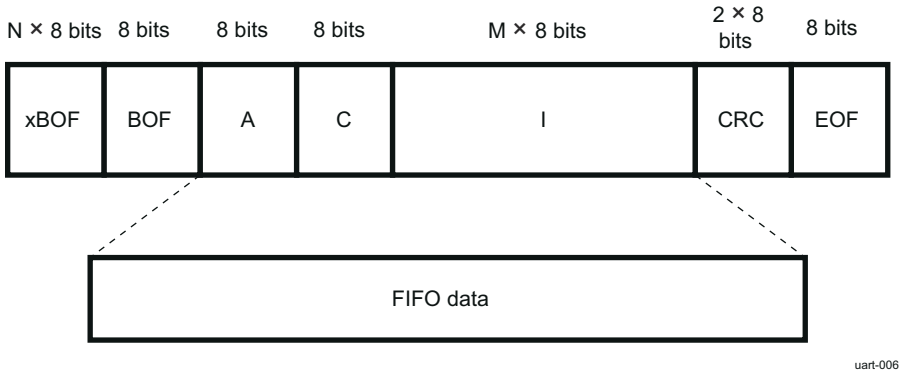


Figure 13-51. IrDA SIR Frame Format

The CRC is applied on the address (A), control (C), and information (I) bytes.

Note

The two words of CRC are written to the FIFO in reception.

13.1.4.2.3.3.1.2 Asynchronous Transparency

Before transmitting a byte, the UART IrDA controller examines each byte of the payload and the CRC field (between BOF and EOF). For each byte equal to 0xC0 (BOF), 0xC1 (EOF), or 0x7D (control escape), the controller performs certain tasks:

- In transmission:
  - Inserts a control escape (CE) byte preceding the byte
  - Complements bit 5 of the byte (that is, exclusive ORs the byte with 0x20)

  The byte sent for the CRC computation is the initial byte written in the TX FIFO (before the XOR with 0x20).
- In reception:

  For the A, C, I, and CRC fields:
  - Compares the byte with the CE byte; if they are not equal, sends the byte to the CRC detector and stores it in the RX FIFO.
  - If the byte is equal to the CE byte, discards the CE byte
  - Complements bit 5 of the byte following the CE
  - Sends the complemented byte to the CRC detector and stores it in the RX FIFO

13.1.4.2.3.3.1.3 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.

When a 0x7D character that is followed immediately by a 0xC1 character is received without transparency, the receiver treats the frame as an aborted frame.

13.1.4.2.3.3.1.4 Pulse Shaping

The SIR mode supports the 3/16 and the 1.6-µs pulse duration methods. The UART_ACREG[7] PULSE_TYPE bit selects the pulse-width method in transmit mode.

13.1.4.2.3.3.1.5 Encoder

Serial data from the transmit state-machine are encoded to transmit data to the optoelectronics. While the TX FIFO output is high, the TX line is always low, and the counter used to form a pulse on TX is cleared continuously.

After the TX FIFO output resets to 0, TX rises on the falling edge of the seventh 16XCLK. On the falling edge of the tenth 16XCLK pulse, TX falls, creating a 3-clock-wide pulse. While the TX FIFO output stays low, a pulse is transmitted during the seventh clock to the tenth clock of each 16-clock bit cycle.

Figure 13-52 shows the IrDA SIR encoding mechanism.

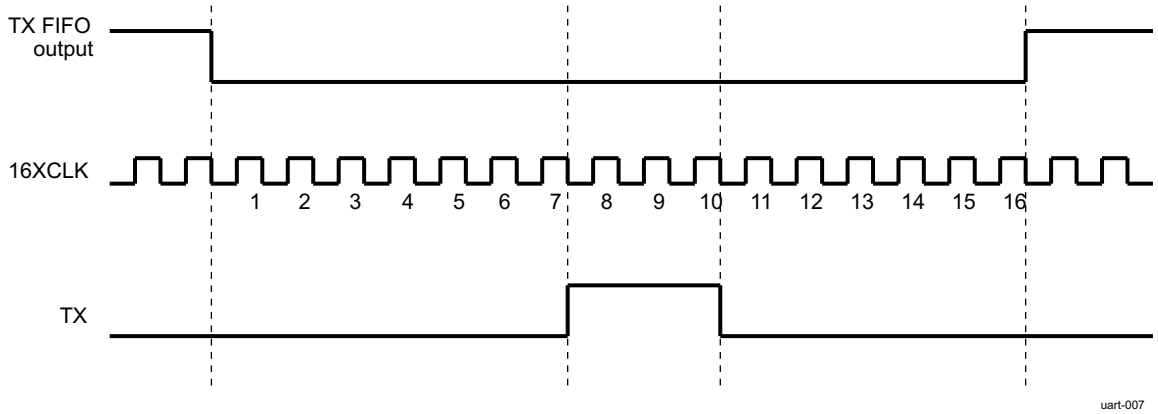


Figure 13-52. IrDA SIR Encoding Mechanism

13.1.4.2.3.3.1.6 Decoder

After reset, the RX FIFO input is high and the 4-bit counter is cleared. When a rising edge is detected on RX, the RX FIFO input falls on the next rising edge of 16XCLK with sufficient setup time. The RX FIFO input stays low for 16 cycles (16XCLK) and then returns to high as required by the IrDA specification. As long as no pulses (rising edges) are detected on the RX, the RX FIFO input remains high.

Figure 13-53 shows the IrDA SIR decoding mechanism.

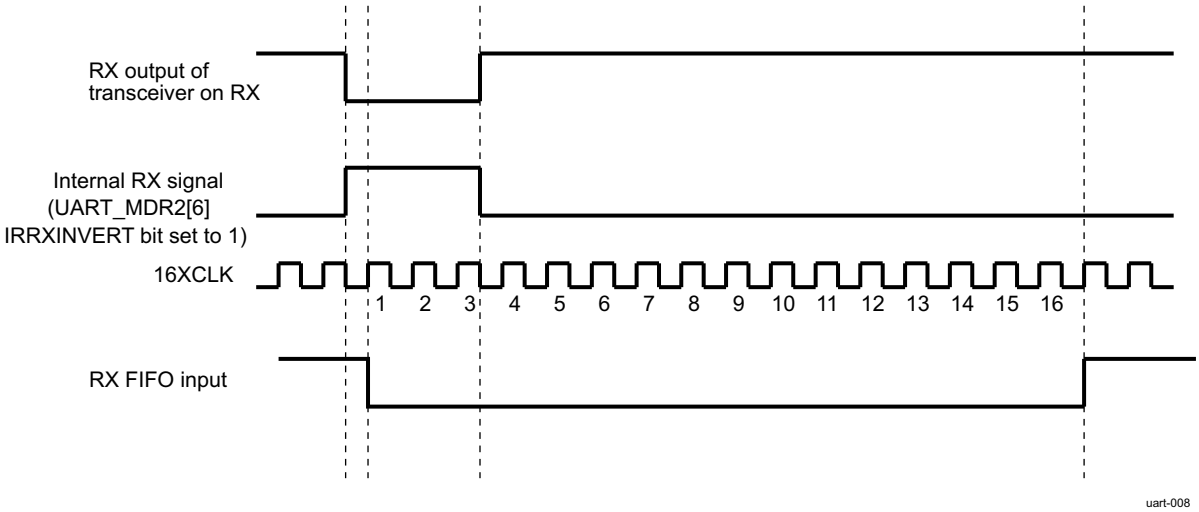


Figure 13-53. IrDA SIR Decoding Mechanism

The module can transmit and receive data, but when the device is transmitting, the IR RX circuitry is automatically disabled by hardware. The operation of the RX input can be disabled using the UART_ACREG[5] DIS_IR_RX bit. The UART_MDR2[6] IRRXINVERT bit can invert the signal from the transceiver (RX) pin to the IR RX logic in the UART. This inversion is performed by default.

13.1.4.2.3.3.1.7 IR Address Checking

In all IR modes, when address checking is enabled by setting the UART_EFR[1-0] bit field (see Table 13-70), only frames intended for the device are written to the RX FIFO. This is to avoid receiving frames not meant for this device in a multipoint infrared environment. To program two frame addresses that the UARTi receives in IrDA mode, use the UART_XON1_ADDR1[7-0] and UART_XON2_ADDR2[7-0] bit fields.

Table 13-70. UART_EFR[1-0] IR Address Checking Options

UART_EFR[1]	UART_EFR[0]	IR Address Checking
0	0	All address-checking operations disabled
0	1	Only address 1 checking enabled
1	0	Only address 2 checking enabled
1	1	All address-checking operations enabled

13.1.4.2.3.3.2 SIR Free-Format Mode

To allow complete software flexibility when transmitting and receiving infrared data packets, the SIR free-format (FF) mode is a subfunction of the existing SIR mode. In FF mode, all frames going to and from the FIFO buffers are untouched with respect to appending and removing control characters and CRC values.

The FF mode corresponds to a UART mode with a pulse modulation of 3/16 of baud rate pulse width.

For example, a normal SIR packet has BOF control and CRC error-checking data appended (transmitting) or removed (receiving) from the data going to and from the FIFOs.

Figure 13-54 shows SIR FF mode.

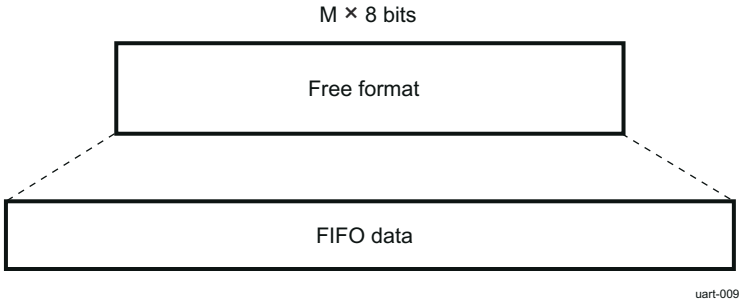


Figure 13-54. SIR FF Mode

In SIR FF mode, the Host CPU software must construct (that is, encode and decode) the entire FIFO data packet.

The SIR Free Format mode is selected by setting the module in UART mode (MDR1[2:0] = 000) and the MDR2[3] register bit to one to allow the pulse shaping. As the bit format is to remain the same, some UART mode configuration registers need to be set at specific value:

- LCR[1:0] = "11" (8 data bits)
- LCR[2] = 0 (2 stop bit)
- LCR[3] = 0 (no parity)
- ACREG[7] = 0 (3/16 of baud-rate pulse width)

The features defined through MDR2[6] and ACREG[5] are also supported, however:

- All other configuration registers need to be at the reset value
- The same UART mode interrupts used for the SIR FF mode are not necessarily relevant (XOFF, RTS, CETS, Modem Status Register)

13.1.4.2.3.3.3 MIR Mode

In MIR mode, data is transferred between the Host CPU and the peripheral devices at 0.576 Mbps or 1.152 Mbps. A MIR transmit frame starts with at least two start flags, followed by a frame data and a CRC-16, and ends with a stop flag (see Figure 13-55).

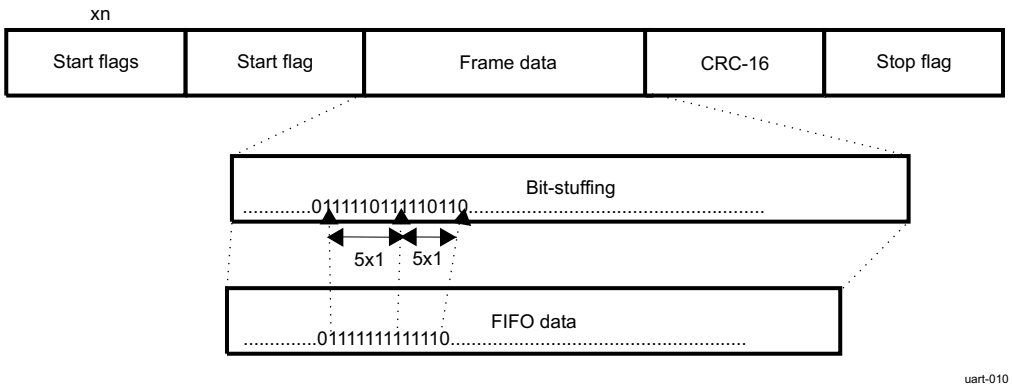


Figure 13-55. MIR Transmit Frame Format

On transmit, the MIR state-machine attaches start flags, a CRC-16, and stop flags, as in SIR mode. All fields are transmitted least-significant bit (LSB) of each byte first.

In MIR mode:

- The state-machine looks for consecutive 1s in the frame data and automatically inserts 0 after five consecutive 1s (this is called bit-stuffing).
- 0x7E is used for start and stop flags (unambiguously, not data, because of bit-stuffing).
- An abort sequence requires a minimum of seven consecutive 1s (unambiguously, not data, because of bit-stuffing).
- Back-to-back frames are allowed with three or more stop flags between them. If two consecutive frames are not back to back, the gap between the last stop flag of the first frame and the start flag of the second frame must be separated by at least seven bit durations.

On receive, the MIR receive state-machine recovers the receive clock, removes the start flags, destuffs the incoming data, and determines the frame boundary with reception of the stop flag. The state-machine also checks for errors such as frame abort, CRC error, and frame-length error. At the end of a frame reception, the Host CPU reads the line status register (UART_LSR_IRDA) to detect errors of the received frame.

The module can transmit and receive data, but when the device is transmitting, the IR RX circuitry is automatically disabled by hardware.

13.1.4.2.3.3.3.1 MIR Encoder/Decoder

To meet the MIR baud rate tolerance of 0.1 percent with a 48-MHz clock input, a 42-41-42 encoding/decoding adjustment is performed. The reference start point is the first start flag, and the 42-41-42 cyclic pattern is repeated until the stop flag is sent or detected.

The jitter created this way is within MIR tolerances. The pulse width is not exactly 1/4, but it is within the tolerances defined by IrDA specifications.

Figure 13-56 shows the MIR baud rate adjustment mechanism.

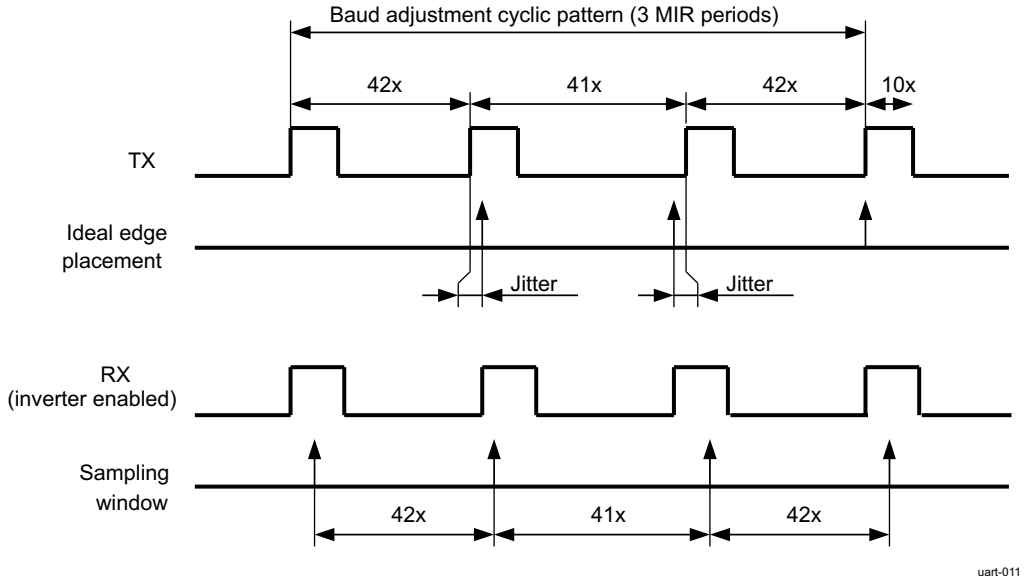


Figure 13-56. MIR Baud Rate Adjustment Mechanism

13.1.4.2.3.3.3.2 SIP Generation

In the MIR and FIR operation modes, the transmitter must send a serial infrared interaction pulse (SIP) at least once every 500 ms. The SIP informs slow devices (operating in SIR mode) that the medium is occupied.

Figure 13-57 shows the SIP.

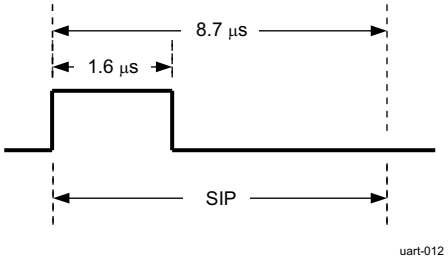


Figure 13-57. SIP

When the SIP_MODE bit of the Mode Definition Register 1 is equal to 1 (MDR1[6]), the TX state machine will always send one SIP at the end of the transmission frame. When MDR1[6] is equal to 0, the transmission of the SIP depends on the SEND_SIP bit of the Auxiliary Control Register (ACREG[3]). The system (Host CPU) can set ACREG[3] at least once every 500ms. The advantage of this approach over the default approach is that the TX state machine does not need to send the SIP at the end of each frame, which may reduce the overhead required.

13.1.4.2.3.3.4 FIR Mode

In FIR mode, data is transferred between the Host CPU and the peripheral devices at 4 Mbps. A FIR transmit frame starts with a preamble that is followed by a start flag, frame data, CRC-32, and ends with a stop flag.

Figure 13-58 shows the FIR transmit frame format.

Figure 13-58. FIR Transmit Frame Format

Preamble (16x)	Start flag	Frame data	CRC-32	Stop flag

On transmit, the FIR transmit state-machine attaches the preamble, start flag, CRC-32, and stop flag. An abort sequence requires at least two transmissions of 0000. Back-to-back frames are allowed, but each frame must be complete.

The state-machine also encodes the transmit data into 4-PPM format (see Table 13-71) and generates the SIP (see Section 13.1.4.2.3.3.3.2, *SIP Generation*).

Table 13-71. 4-PPM Format

Data Bit Pair (Bin)	4-PPM Data Symbol (Bin)
00	1000
01	0100
10	0010
11	0001

The four symbols described in Table 13-71 are the legal, encoded data symbols. All other combinations are illegal for encoding data. Some of these illegal symbols are used in the definition of the preamble, start flag, and stop flag because they are unambiguously not data (see Table 13-72).

Table 13-72. FIR Preamble, Start Flag, and Stop Flag

Frame Part	Transmitted Frame (Bin)
Preamble	1000 0000 1010 1000 (16 repeated transmissions)
Start flag	0000 1100 0000 1100 0110 0000 0110 0000
Stop flag	0000 1100 0000 1100 0000 0110 0000 0110

All fields are transmitted LSBs of each byte first (see Table 13-73).

Table 13-73. FIR Data Byte Transmission Order Example

Data Byte (Hex)	Data Byte Pair (Bin)	4-PPM Data Symbol (Bin)	Transmission Order
0x0B	00	1000	4
	00	1000	3
	10	0010	2
	11	0001	1

On receive, the FIR receive state-machine recovers the receive clock, removes the preamble and the start flag, decodes the 4-PPM incoming data, and determines the frame boundary with reception of the stop flag. The state-machine also checks for errors such as illegal symbol, CRC error, and frame-length error. At the end of a frame reception, the Host CPU reads the line status register (UART_LSR_IRDA) to detect errors of the received frame.

The module can transmit and receive data, but when the device is transmitting, the IR RX circuitry is automatically disabled by hardware.

13.1.4.2.4 CIR Functional Interfaces

13.1.4.2.4.1 System Using CIR Communication Protocol With Remote Control

All UART modules can be connected to an external infrared transceiver in CIR mode. Figure 13-59 shows an example connection of UART0 in CIR mode.

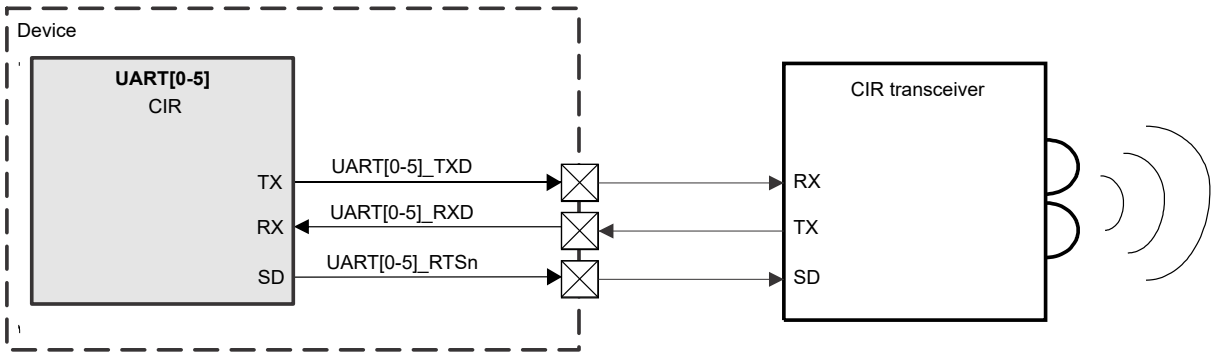


Figure 13-59. CIR Mode Interface Signals

13.1.4.2.4.2 CIR Interface Description

Table 13-74 lists the CIR interface I/O signals.

Table 13-74. UART I/O Signals (CIR Mode)

Module Pin	Device Level Signal	I/O(1)	Description	Module Pin Reset Value(2)
		UART[0-5]		
RX	UART[0-5]_RXD	I	Serial data input	HiZ
TX	UART[0-5]_TXD	O	Serial data output in CIR mode. (3)	0
SD	UART[0-5]_RTSn	O	SD mode is used to configure the transceivers.(4)	1

(1) I = Input; O = Output
(2) HiZ = High Impedance
(3) In other modes, this pin is set to the reset value (inactive state).
(4) The SD pinout is an inverted value of the UART_ACREG[6] SD_MOD bit.

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.

13.1.4.2.4.3 CIR Protocol and Data Format

In CIR mode, the infrared operation functions as a programmable (universal) remote control.

The CIR mode uses a variable 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 user-defined frame structure and packet content.

13.1.4.2.4.3.1 Carrier Modulation

Each modulated pulse that constitutes a digit is a train of on/off pulses (see Figure 13-60).

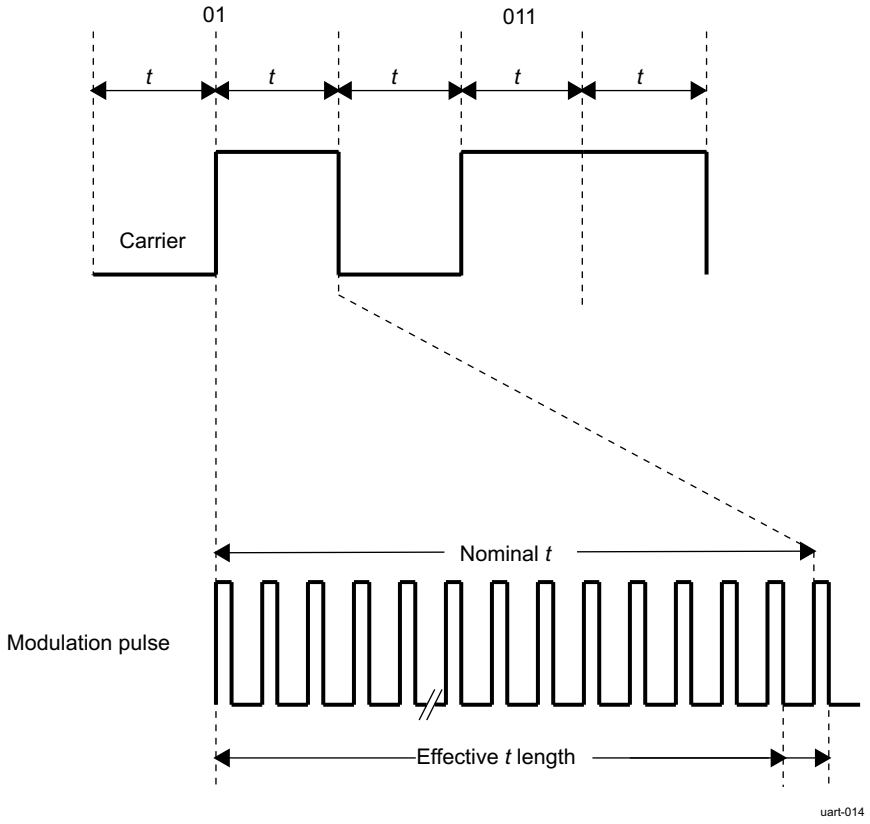


Figure 13-60. CIR Pulse Modulation

13.1.4.2.4.3.2 Pulse Duty Cycle

The programmer can choose one of four duty cycles for modulation pulses by setting the appropriate value in the UART_MDR2[5-4] CIR_PULSE_MODE bit field (1/4, 1/3, 5/12, or 1/2).

Figure 13-61 shows the CIR modulation duty cycles.

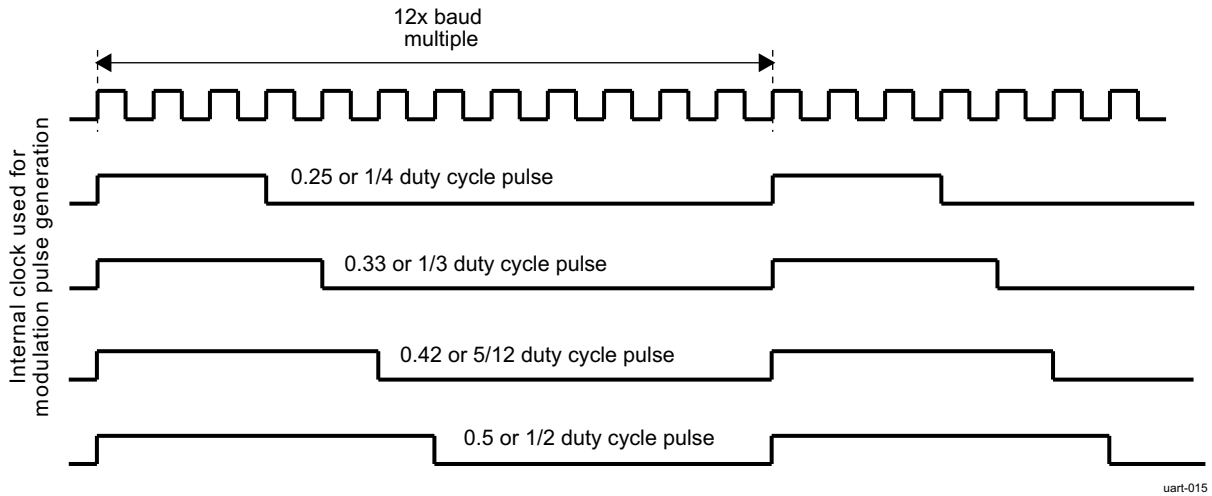


Figure 13-61. CIR Modulation Duty Cycle

The transmission logic ensures that all pulses are transmitted completely (no cutoff during transmission). While transmitting continuous bytes back-to-back, no delay is inserted between 2 transmitted bytes. Thus, software must handle the delay between consecutively transmitted bytes if the receiving end requires it.

13.1.4.2.4.3.3 Consumer IR Encoding/Decoding

There are two methods of encoding for remote-control applications:

- Pulse duration encoding (time-extended bit forms): A variable pulse distance, or duration, in which the difference between logic 1 and logic 0 is the length of the pulse width
- Biphase encoding: The encoding of logic 0 and logic 1 is in the change of signal level from 1 to 0 or 0 to 1, respectively.

Japanese manufacturers favor pulse duration encoding; European manufacturers favor biphase encoding.

CIR mode uses a completely flexible free-format encoding in which 1 is transmitted from the TX FIFO as a modulated pulse with duration *t*.

Similarly, 0 is transmitted as a blank duration T. The Host CPU constructs and deciphers the protocol of the data. For example, the RC-5 protocol using Manchester encoding can be emulated as using a 01 pair for 1 and a 10 pair for 0 (see Figure 13-62).

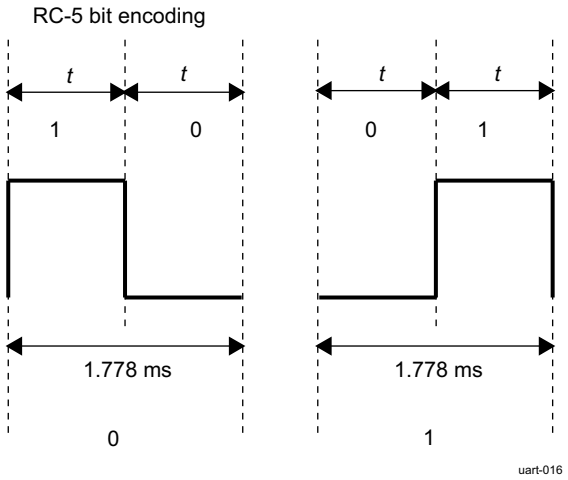


Figure 13-62. UART RC-5 Bit Encoding

Because CIR mode logic does not impose a fixed format for infrared packets of data, the Host CPU software can define the format using simple data structures that are then modulated into an industry standard, such as RC-5 or SIRC. To send a sequence of 0101 in RC-5, the Host CPU software must write an 8-bit binary character of 10011001 to the data FIFO of the UART.

For SIRC, the modulation length (multiples of *t*) is used to distinguish between 1 and 0. The subsequent SIRC digits show the difference in encoding between this and, for example, RC-5. The pulse width is extended for one digit.

Figure 13-63 shows SIRC bit encoding.

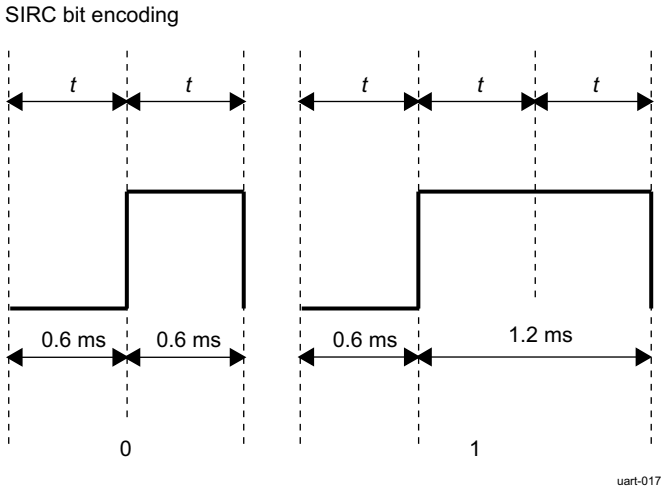


Figure 13-63. UART SIRC Bit Encoding

To construct comprehensive packets constituting remote-control commands, the Host CPU software must combine a number of 8-bit data characters in a sequence that follows one of the universally accepted formats.

Figure 13-64 shows a standard RC-5 frame as detected by UART in CIR mode (the SIRC format follows this). Each field in RC-5 can be considered as two *t* pulses (digital bits) from the TX FIFO.

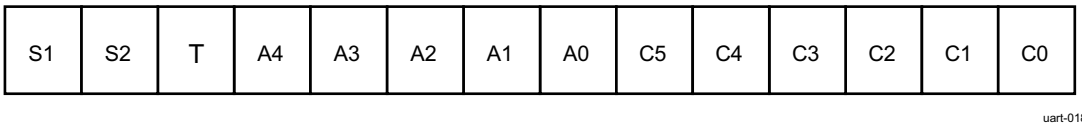


Figure 13-64. UART RC-5 Standard Packet Format

Where:

S1, S2:	Start-bits (always 1)
T:	Toggle bit
A4..A0:	Address (or system) bits
C5..C0:	Command bits

The toggle bit T changes when a new command is transmitted to detect when the same key is pressed twice (effectively receiving the same data from the host consecutively). A brief delay in the transmission of the same command is detected by the use of the toggle bit because a code is sent while the Host CPU transmits characters to the UART for transmission. The address bits define the machine or device for which the infrared transmission is intended, and the command defines the operation.

To accommodate an extended RC-5 format, the S2 bit is replaced by an additional command bit (C6) that lets the command range increase to 7 bits. This format is known as the extended RC-5 format.

The SIRC encoding uses the duration of modulation for mark and space; therefore, the duration of data bits in the standard frame length varies.

Figure 13-65 shows the packet format and bit encoding. As Figure 13-66 shows, 1 start-bit of 2.4 ms and control codes are followed by data that constitute the entire frame.

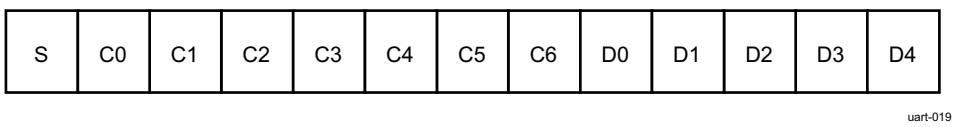


Figure 13-65. UART SIRC Packet Format

Note

The encoding must take a standard duration, but the contents of the data can vary. This implies that the control software for sending and receiving data packets must exercise a scheme of interpacket delay, where successive packets can be sent only after a real-time delay expires.

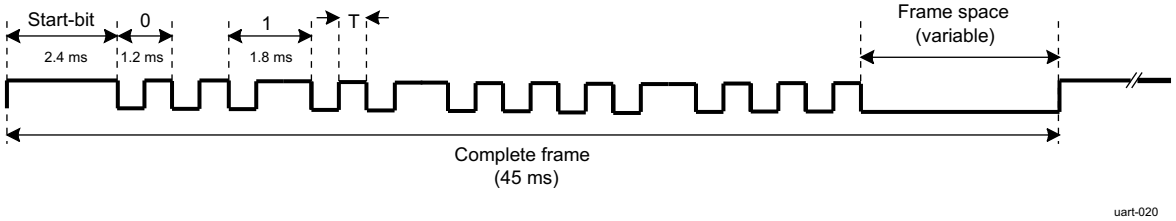


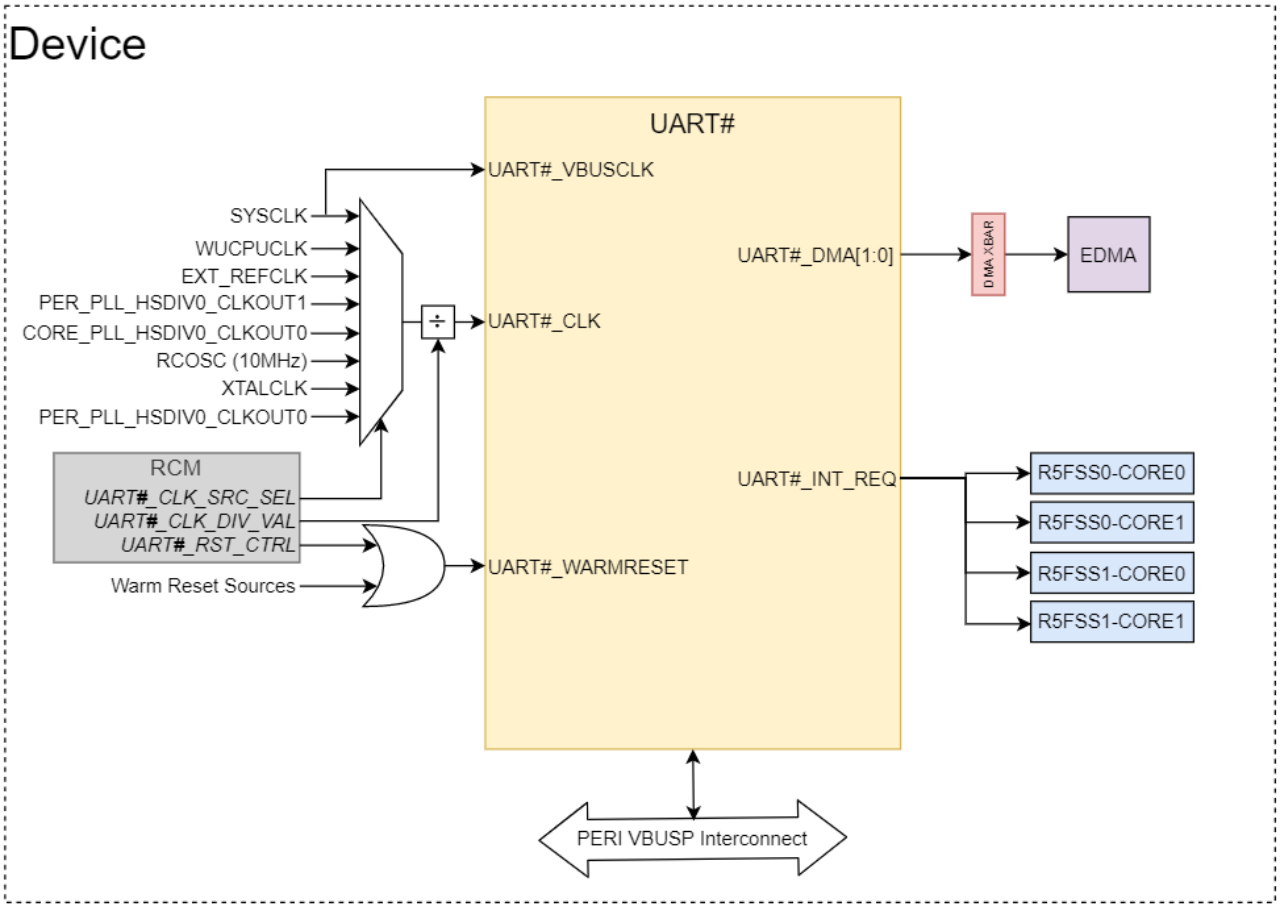
Figure 13-66. UART SIRC Bit Transmission Example

Note

This document does not describe all encoding methods and techniques; the previous information discusses the considerations required to employ different encoding methods for different industry-standard protocols. See industry-standard documentation for specific methods of encoding and protocol use.

13.1.4.3 UART Integration

There are 6x UART modules integrated in the device. The diagram below provides a visual representation of the device integration details.



= 0, 1, 2, 3, 4, 5

Figure 13-67. UART Integration

The tables below summarize the device integration details of UART# (where # = 0, 1, 2, 3, 4, 5) in the device.

Table 13-75. *UART* Device Integration

This table describes the module device integration details.

Module Instance	Device Allocation	SoC Interconnect
UART0	✓	PERI VBUSP Interconnect
UART1	✓	PERI VBUSP Interconnect
UART2	✓	PERI VBUSP Interconnect
UART3	✓	PERI VBUSP Interconnect
UART4	✓	PERI VBUSP Interconnect
UART5	✓	PERI VBUSP Interconnect

Table 13-76. *UART* Clocks

This table describes the module clocking signals.

Module Instance	Module Clock Input	Source Clock Signal	Source	Default Freq	Description
UART0	UART0_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	UART0 VBUS Clock
	UART0_FCLK (UART_CLK)	XTALCLK	External XTAL	25 MHz	UART0 Interface Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK: HSDIV0_CLKOUT0	400 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
		DPLL_PER_HSDIV0_CLKOUT0	PLL_PER_CLK:HSDIV0_CLKOUT0	160 MHz	
UART1	UART1_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	UART1 VBUS Clock
	UART1_FCLK (UART_CLK)	XTALCLK	External XTAL	25 MHz	UART1 Interface Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK: HSDIV0_CLKOUT0	400 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
		DPLL_PER_HSDIV0_CLKOUT0	PLL_PER_CLK:HSDIV0_CLKOUT0	160 MHz	

Table 13-76. *UART* Clocks (continued)

This table describes the module clocking signals.

Module Instance	Module Clock Input	Source Clock Signal	Source	Default Freq	Description
UART2	UART2_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	UART2 VBUS Clock
	UART2_FCLK (UART_CLK)	XTALCLK	External XTAL	25 MHz	UART2 Interface Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK: HSDIV0_CLKOUT0	400 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
		DPLL_PER_HSDIV0_CLKOUT0	PLL_PER_CLK:HSDIV0_CLKOUT0	160 MHz	
UART3	UART3_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	UART3 VBUS Clock
	UART3_FCLK (UART_CLK)	XTALCLK	External XTAL	25 MHz	UART3 Interface Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK: HSDIV0_CLKOUT0	400 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
		DPLL_PER_HSDIV0_CLKOUT0	PLL_PER_CLK:HSDIV0_CLKOUT0	160 MHz	

Table 13-76. *UART* Clocks (continued)

This table describes the module clocking signals.

Module Instance	Module Clock Input	Source Clock Signal	Source	Default Freq	Description
UART4	UART4_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	UART4 VBUS Clock
	UART4_FCLK (UART_CLK)	XTALCLK	External XTAL	25 MHz	UART4 Interface Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK: HSDIV0_CLKOUT0	400 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
		DPLL_PER_HSDIV0_CLKOUT0	PLL_PER_CLK:HSDIV0_CLKOUT0	160 MHz	
UART5	UART5_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	UART5 VBUS Clock
	UART5_FCLK (UART_CLK)	XTALCLK	External XTAL	25 MHz	UART5 Interface Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK: HSDIV0_CLKOUT0	400 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
		DPLL_PER_HSDIV0_CLKOUT0	PLL_PER_CLK:HSDIV0_CLKOUT0	160 MHz	

Table 13-77. *UART* Resets

This table describes the module reset signals.

Module Instance	Module Reset Input	Source Reset Signal	Source	Description
UART0	UART0_RST(VBUSP_RSTn)	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	UART0 Asynchronous Reset
UART1	UART1_RST(VBUSP_RSTn)	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	UART1 Asynchronous Reset
UART2	UART2_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	UART2 Asynchronous Reset

Table 13-77. *UART* Resets (continued)

This table describes the module reset signals.

Module Instance	Module Reset Input	Source Reset Signal	Source	Description
UART3	UART3_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	UART3 Asynchronous Reset
UART4	UART4_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	UART4 Asynchronous Reset
UART5	UART5_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	UART5 Asynchronous Reset

Table 13-78. *UART* Interrupt Requests

This table describes the module interrupt requests.

Module Instance	Module Interrupt Signal	Destination Interrupt Input	Destination	Type	Description
UART0	uart0_int_req	uart0_int_req	ALL R5FSS Cores ICSSM Core	Level	UART0 IP Status Information
UART1	uart1_int_req	uart1_int_req	ALL R5FSS Cores ICSSM Core		UART1 IP Status Information
UART2	uart2_int_req	uart2_int_req	ALL R5FSS Cores ICSSM Core		UART2 IP Status Information
UART3	uart3_int_req	uart4_int_req	ALL R5FSS Cores ICSSM Core		UART3 IP Status Information
UART4	uart4_int_req	uart4_int_req	ALL R5FSS Cores ICSSM Core		UART4 IP Status Information
UART5	uart5_int_req	uart5_int_req	ALL R5FSS Cores ICSSM Core		UART5 IP Status Information

Table 13-79. *UART* DMA Requests

This table describes the module DMA requests.

Module Instance	Module DMA Event	Destination DMA Event Input	Destination	Type	Description
UART0	UART0_DMA_0	UART0_dma_req[0]	EDMA Crossbar (DMA_XBAR)	Level	UART0 DMA Request
	UART0_DMA_1	UART0_dma_req[1]			
UART1	UART1_DMA_0	UART1_dma_req[0]	EDMA Crossbar (DMA_XBAR)	Level	UART1 DMA Request
	UART1_DMA_1	UART1_dma_req[1]			
UART2	UART2_DMA_0	UART2_dma_req[0]	EDMA Crossbar (DMA_XBAR)	Level	UART2 DMA Request
	UART2_DMA_1	UART2_dma_req[1]			
UART3	UART3_DMA_0	UART3_dma_req[0]	EDMA Crossbar (DMA_XBAR)	Level	UART3 DMA Request
	UART3_DMA_1	UART3_dma_req[1]			
UART4	UART4_DMA_0	UART4_dma_req[0]	EDMA Crossbar (DMA_XBAR)	Level	UART4 DMA Request
	UART4_DMA_1	UART4_dma_req[1]			
UART5	UART5_DMA_0	UART5_dma_req[0]	EDMA Crossbar (DMA_XBAR)	Level	UART5 DMA Request
	UART5_DMA_1	UART5_dma_req[1]			

Note

For more information on the interconnects, see the *System Interconnect* chapter.

For more information on power, reset, and clock management, see the corresponding sections within the *Device Configuration* chapter.

For more information on the device interrupt controllers, see the *Interrupt Controllers* chapter.

13.1.4.4 UART Functional Description

13.1.4.4.1 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.

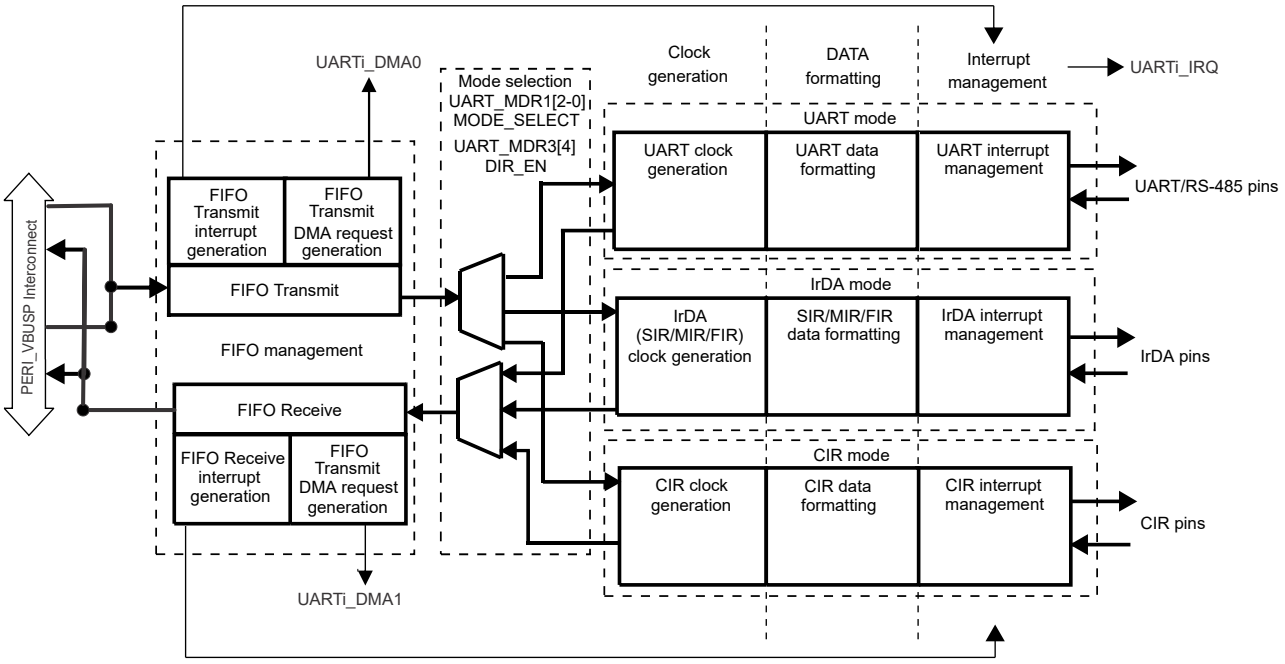


Figure 13-68. UART Functional Block Diagram

13.1.4.4.2 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.

13.1.4.4.3 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.

13.1.4.4.3.1 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.

13.1.4.4.4 UART Power Management

13.1.4.4.4.1 UART Mode Power Management

13.1.4.4.4.1.1 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 13.1.4.4.5.1.2, *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.

13.1.4.4.4.1.2 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*.

13.1.4.4.4.2 IrDA Mode Power Management

13.1.4.4.4.2.1 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.

The module wakes up when a change is detected on the RXD line or when data is written to the TX FIFO.

13.1.4.4.4.2.2 System Power Saving

System power saving for the IrDA mode has the same function as for the UART mode (see Section 13.1.4.4.4.1.2, *System Power Saving*).

13.1.4.4.4.3 CIR Mode Power Management

13.1.4.4.4.3.1 Module Power Saving

Module power saving for the CIR mode has the same function as for the IrDA mode (see Section 13.1.4.4.4.2.1, *Module Power Saving*).

13.1.4.4.4.3.2 System Power Saving

System power saving for the CIR mode has the same function as for the UART mode (see Section 13.1.4.4.4.1.2, *System Power Saving*).

13.1.4.4.4.4 Local Power Management

Table 13-80 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 13-80. UART Local Power-Management Features

Feature	Registers	Description
Clock autogating	N/A	Feature not available
Peripheral idle modes	N/A	Feature not available
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

13.1.4.4.5 UART Interrupt Requests

13.1.4.4.5.1 UART Mode Interrupt Management

13.1.4.4.5.1.1 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 13-81 summarizes the interrupt control functions.

Table 13-81. 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

Table 13-81. 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.

13.1.4.4.5.1.2 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.

13.1.4.4.5.2 IrDA Mode Interrupt Management

13.1.4.4.5.2.1 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 13-82). The interrupt line is activated when any interrupt is generated (there is no priority).

Table 13-82. 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.

Table 13-82. 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.

13.1.4.4.5.2.2 Wake-Up Interrupts

The wake-up interrupt for IrDA mode has the same function as that for UART mode (see Section 13.1.4.4.5.1.2, *Wake-Up Interrupt*).

13.1.4.4.5.3 CIR Mode Interrupt Management

13.1.4.4.5.3.1 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 13-83 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 13-83. CIR Mode Interrupts

IIR_CIR Bit Number	Interrupt Type	Interrupt Source	Interrupt Reset Method
0	RHR interrupt	DRDY (data ready) (FIFO disable) RX FIFO above trigger level (FIFO enable)	Read RHR until interrupt condition disapppears.
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	RX_STOP_IT	Receive stop interrupt (depending on value set in the BOF Length register (EBLR))	Read the UART_IIR_CIR register
3	RX overrun	Write to RHR when RX FIFO is full.	Read the RESUME register.
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

13.1.4.4.5.3.2 Wake-Up Interrupts

The wake-up interrupt for CIR mode has the same function as that for UART mode (see Section 13.1.4.4.5.1.2, *Wake-Up Interrupt*).

13.1.4.4.6 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 13-69 shows the FIFO management registers.

Note

Data in the UART_RHR register is not overwritten when an overflow occurs.

Note

The UART_SFLSR, UART_SFREGL, and UART_SFREGH status registers are used in IrDA mode only. For information about their use, see Section 13.1.4.4.8.3.3, *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 × UARTi_ICLK + 5 × UARTi_FCLK clock cycles. This delay is needed to finish the resetting of the corresponding FIFO and DMA control registers.

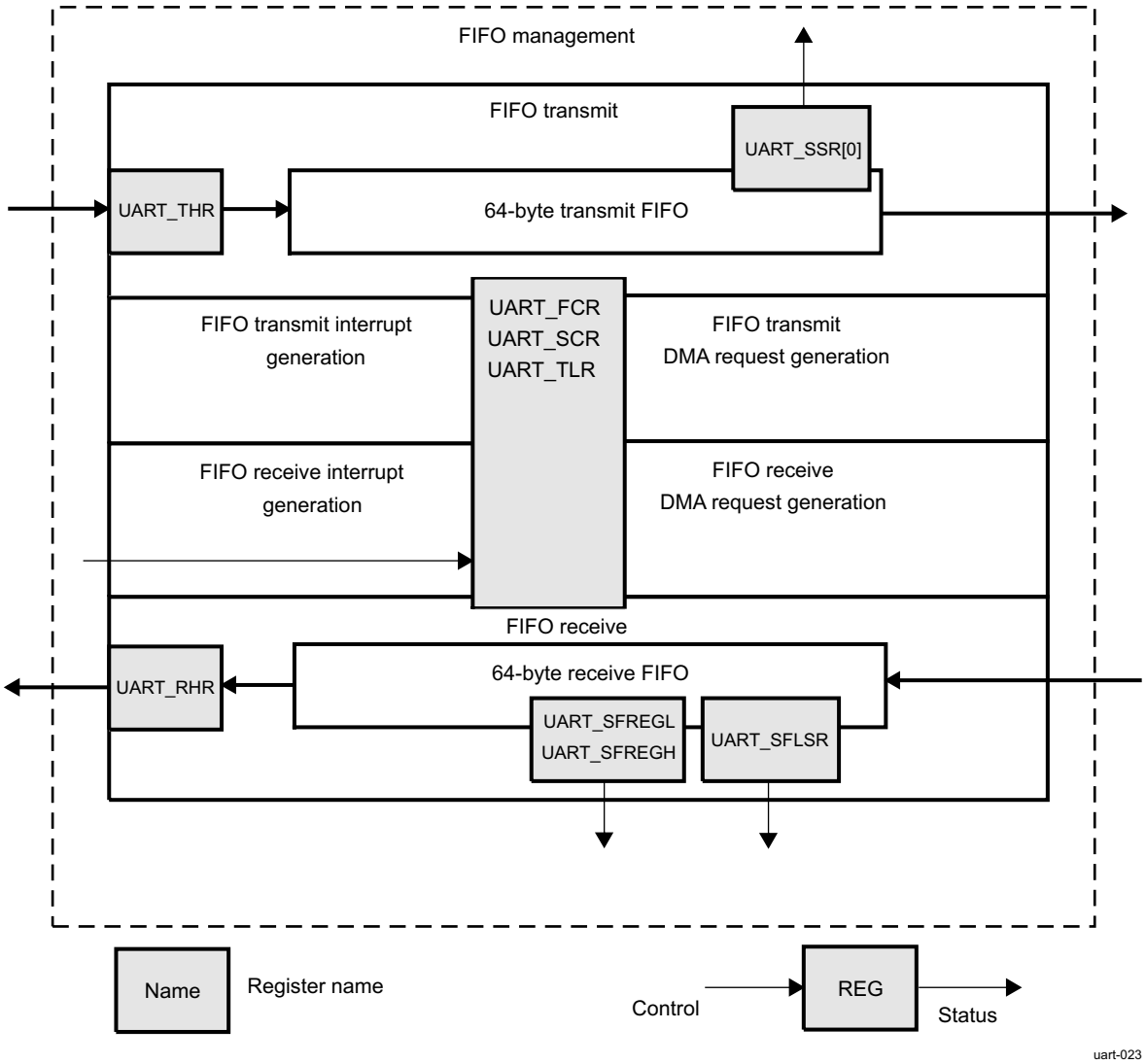


Figure 13-69. UART FIFO Management Registers

13.1.4.4.6.1 FIFO Trigger

13.1.4.4.6.1.1 Transmit FIFO Trigger

Table 13-84 lists the TX FIFO trigger level settings.

Table 13-84. 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)
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.

13.1.4.4.6.1.2 Receive FIFO Trigger

Table 13-85 lists the RX FIFO trigger-level settings.

Table 13-85. 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.

13.1.4.4.6.2 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 13-70 shows the generation of the RX FIFO interrupt request.

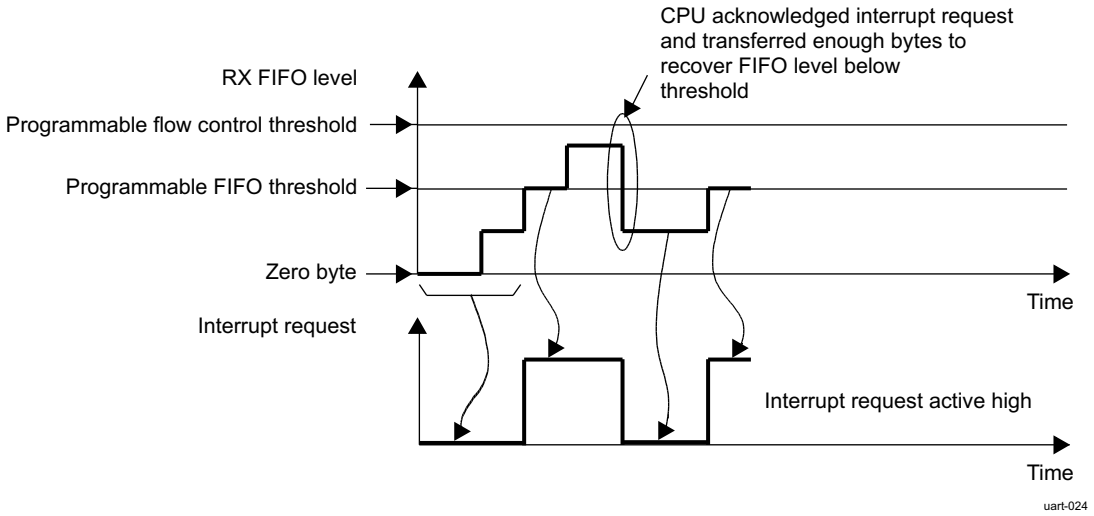


Figure 13-70. UART RX FIFO Interrupt Request Generation

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 13-71 shows the generation of the TX FIFO interrupt request.

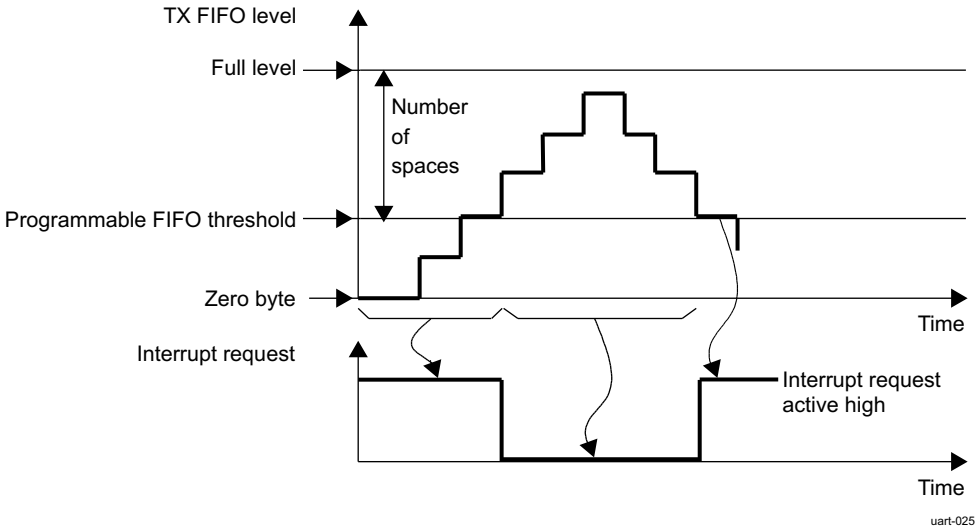


Figure 13-71. UART TX FIFO Interrupt Request Generation

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.

13.1.4.4.6.3 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.

13.1.4.4.6.4 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 5).

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.

13.1.4.4.6.4.1 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

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.

13.1.4.4.6.4.2 DMA Transfers (DMA Mode 1, 2, or 3)

Figure 13-72 through Figure 13-75 show the supported DMA operations.

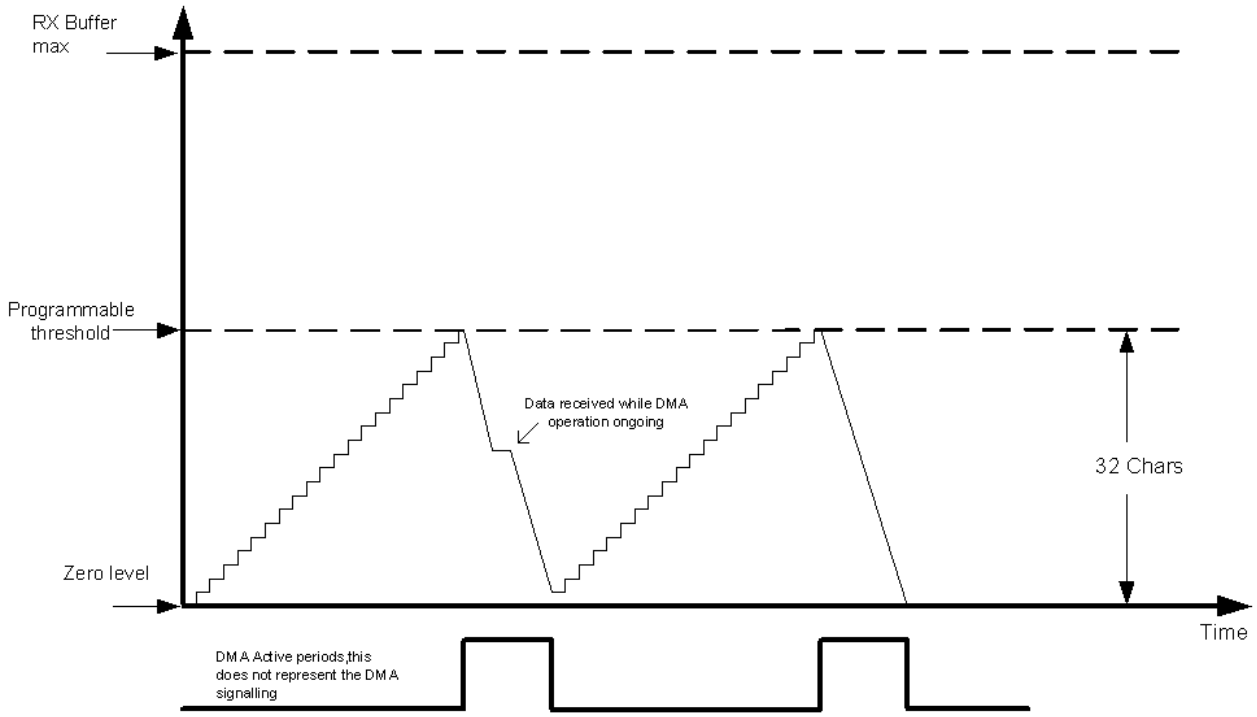


Figure 13-72. UART Receive FIFO DMA Request Generation (32 Characters)

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.

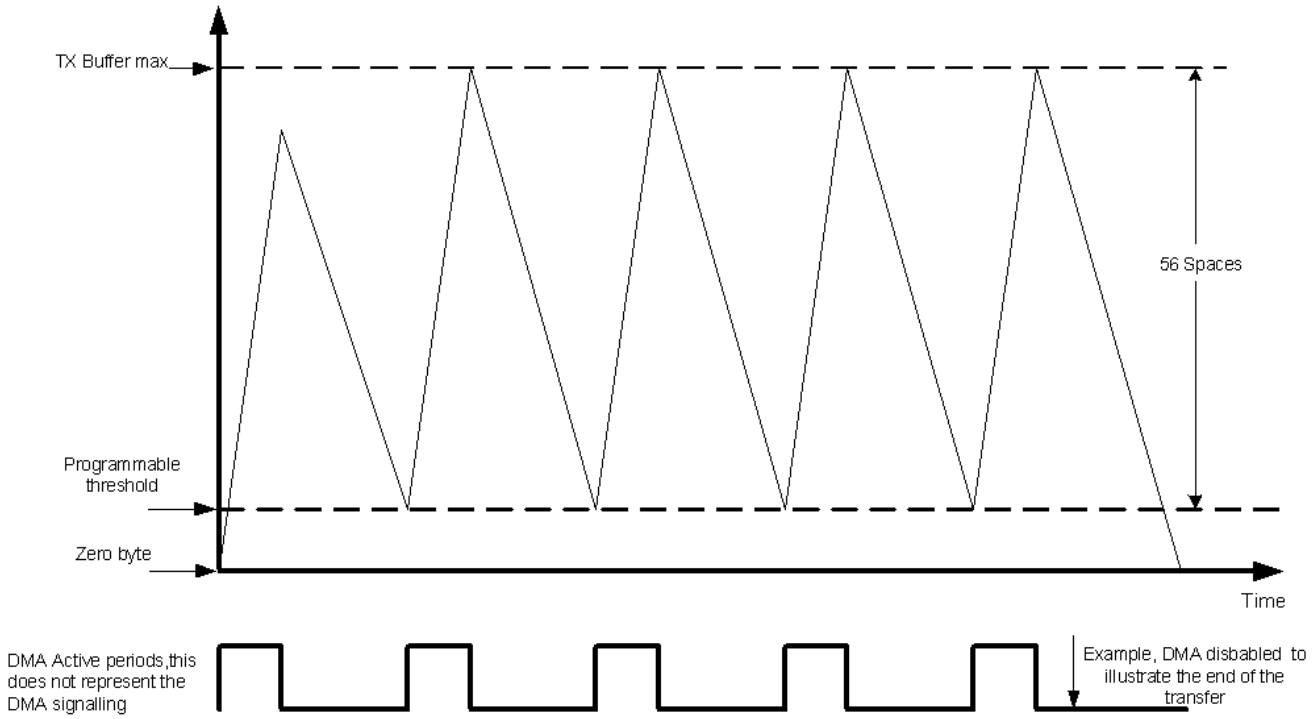


Figure 13-73. UART Transmit FIFO DMA Request Generation (56 Spaces)

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 13-73 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 13-74 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.

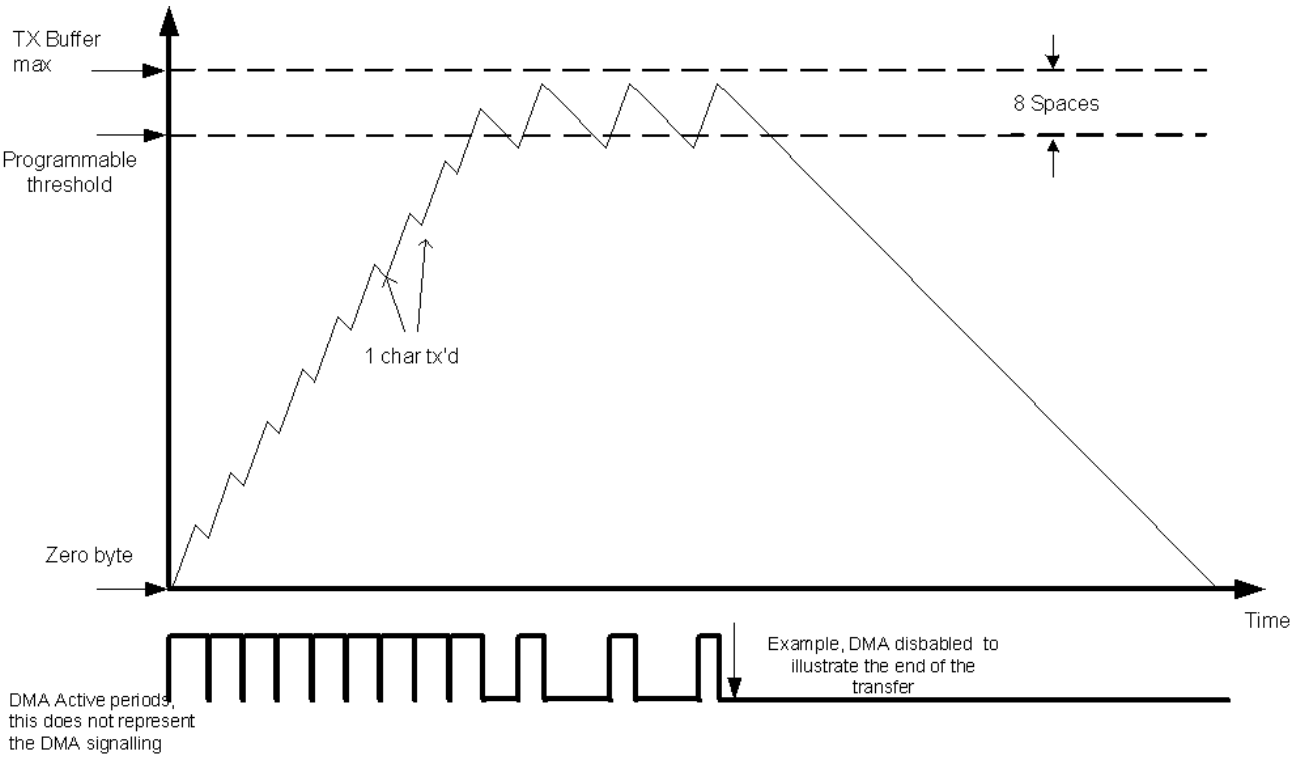


Figure 13-74. UART Transmit FIFO DMA Request Generation (8 Spaces)

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 13-75). 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.

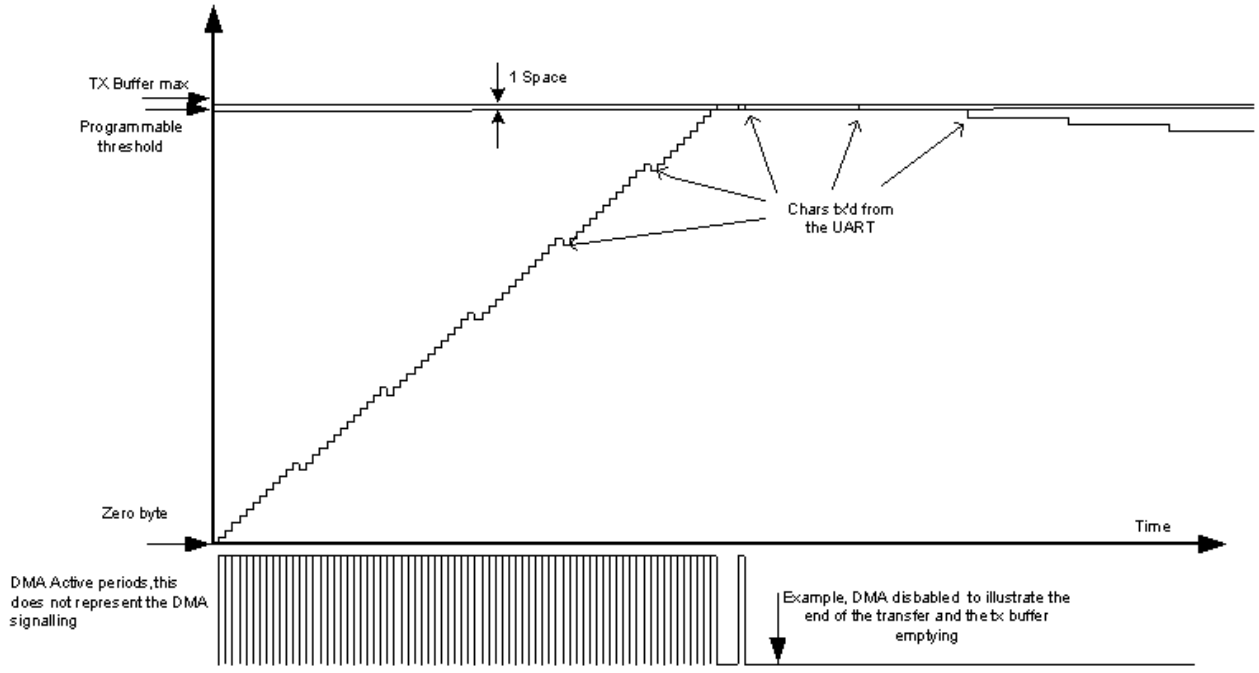


Figure 13-75. UART Transmit FIFO DMA Request Generation (1 Space)

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 13-76). 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.

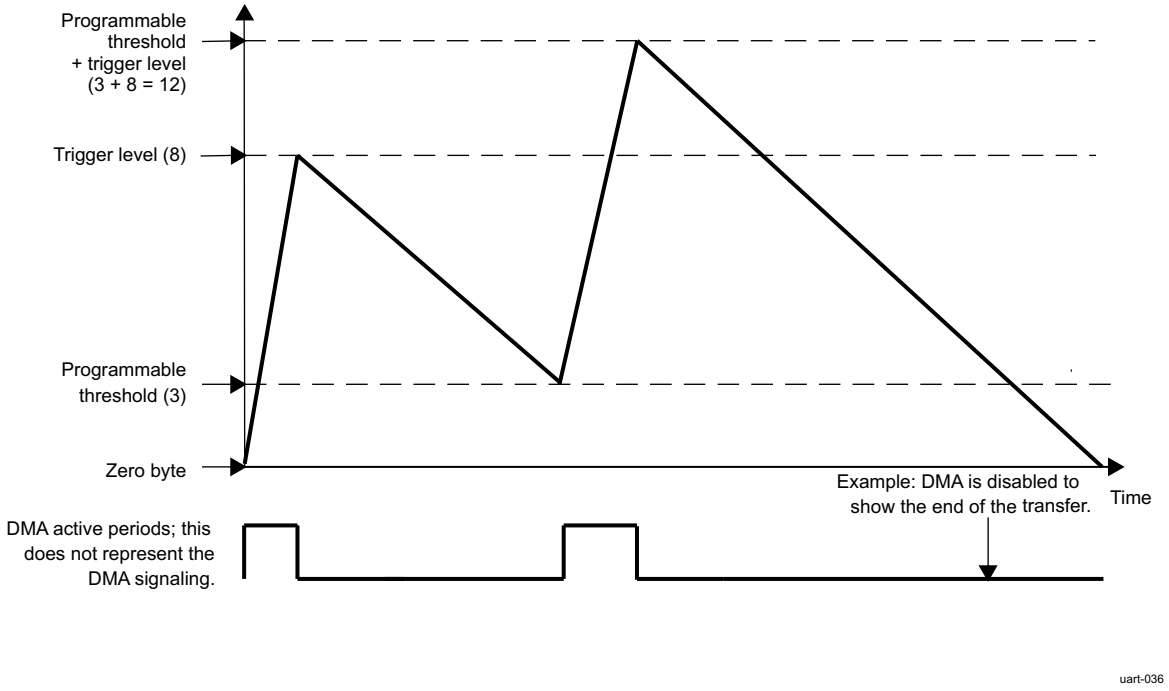


Figure 13-76. UART Transmit FIFO DMA Request Generation Using Direct TX DMA Threshold Programming. (Threshold = 3; Spaces = 8)

13.1.4.4.6.4.3 DMA Transmission

Figure 13-77 shows DMA transmission.

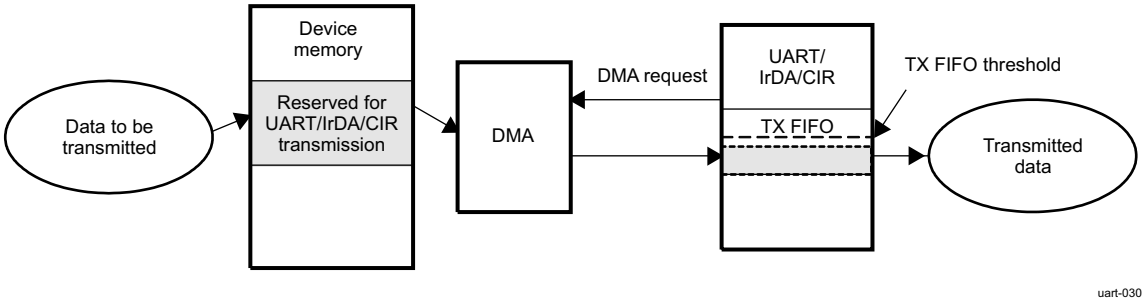


Figure 13-77. DMA Transmission

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.

13.1.4.4.6.4.4 DMA Reception

Figure 13-78 shows DMA reception.

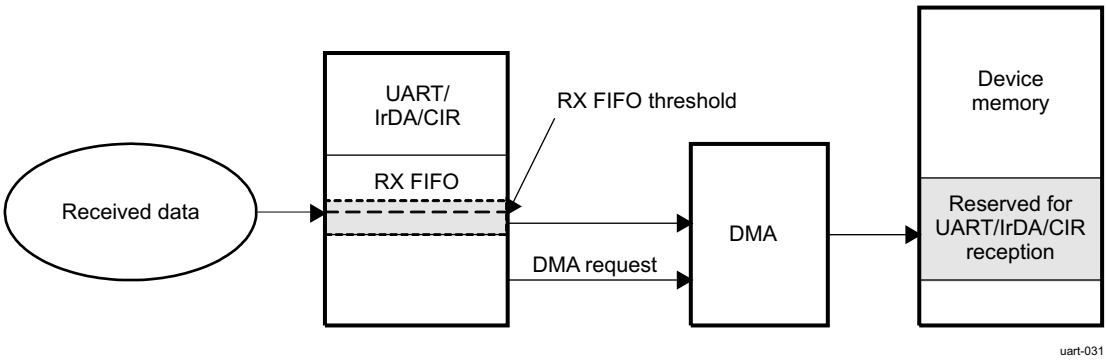


Figure 13-78. DMA Reception

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.

13.1.4.4.7 UART Mode Selection

13.1.4.4.7.1 Register Access Modes

13.1.4.4.7.1.1 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 13-86).

Table 13-86. 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

13.1.4.4.7.1.2 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 13-109, *UART Load FIFO Triggers Defined by the Concatenated Value*.

Table 13-87 through Table 13-89 summarize the register access submodes.

Table 13-87. UART Subconfiguration Mode A Summary

Mode	Condition
MSR_SPR	(UART_EFR[4] = 0x0 or UART_MCR[6] = 0x0)

Table 13-87. UART Subconfiguration Mode A Summary (continued)

Mode	Condition
TCR_TLR	UART_EFR[4] = 0x1 and UART_MCR[6] = 0x1

Table 13-88. 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 13-89. 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

13.1.4.4.7.1.3 Registers Available for the Register Access Modes

Table 13-90 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 13-90. 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

Table 13-90. 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

13.1.4.4.7.2 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 13-91).

Table 13-91. 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 13.1.4.4.7.1, *Register Access Modes*).

To select a RS-485 mode, set the UART_MDR3[4] DIR_EN bit field to 0x1.

13.1.4.4.7.2.1 Registers Available for the UART Function

Only the registers listed in Table 13-92 are used for the UART function.

Table 13-92. 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]

Table 13-92. 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.

13.1.4.4.7.2.2 Registers Available for the IrDA Function

Only the registers listed in Table 13-93 are used for the IrDA function.

Table 13-93. 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

Table 13-93. 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_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(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.

13.1.4.4.7.2.3 Registers Available for the CIR Function

Only the registers listed in Table 13-94 are used for the CIR function.

Table 13-94. 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)	–

Table 13-94. 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_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(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.

13.1.4.4.8 UART Protocol Formatting

13.1.4.4.8.1 UART Mode

13.1.4.4.8.1.1 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 13-79 shows the baud rate generator and associated controls.

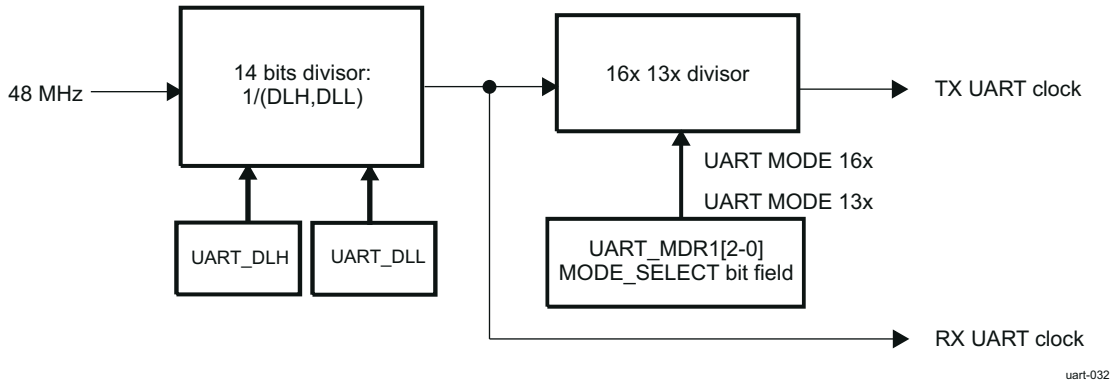


Figure 13-79. UART Baud Rate Generation

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.

13.1.4.4.8.1.2 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 13-95. 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

13.1.4.4.8.1.3 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 13-96 summarizes the operation of address matching based on the mentioned values.

Table 13-96. 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:

- 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.

13.1.4.4.8.1.4 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.

13.1.4.4.8.1.5 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.

13.1.4.4.8.1.5.1 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 13-97).

Table 13-97. 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

13.1.4.4.8.1.5.2 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.

13.1.4.4.8.1.5.3 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 13-98).

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 13-98. 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.

13.1.4.4.8.1.5.3.1 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.

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.

13.1.4.4.8.1.5.3.2 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.

13.1.4.4.8.1.5.4 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
  - 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)

13.1.4.4.8.1.5.5 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 13-81 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 13-81 for the UART mode interrupts).

13.1.4.4.8.1.5.6 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.

13.1.4.4.8.1.5.7 Time-Out and Break Conditions

13.1.4.4.8.1.5.7.1 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.

13.1.4.4.8.1.5.7.2 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.

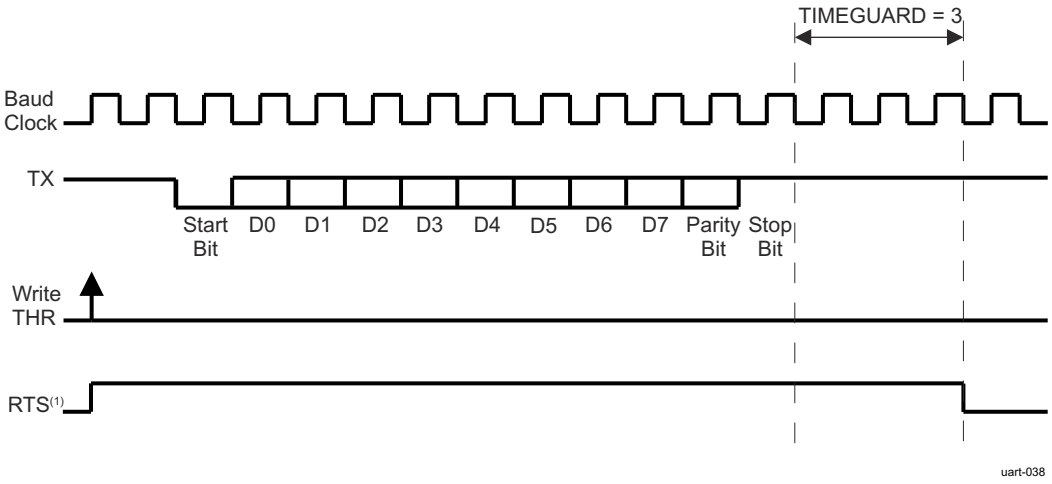
The time-out counter and break condition apply only to UART modem operation and not to IrDA/CIR mode operation.

13.1.4.4.8.2 RS-485 Mode

13.1.4.4.8.2.1 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 13-80 shows the direction control.



(1) Assumes DIR_POL = 1

Figure 13-80. RS-485 External Transceiver Direction Control

13.1.4.4.8.3 IrDA Mode

13.1.4.4.8.3.1 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 13-81 shows the baud rate generator and associated controls.

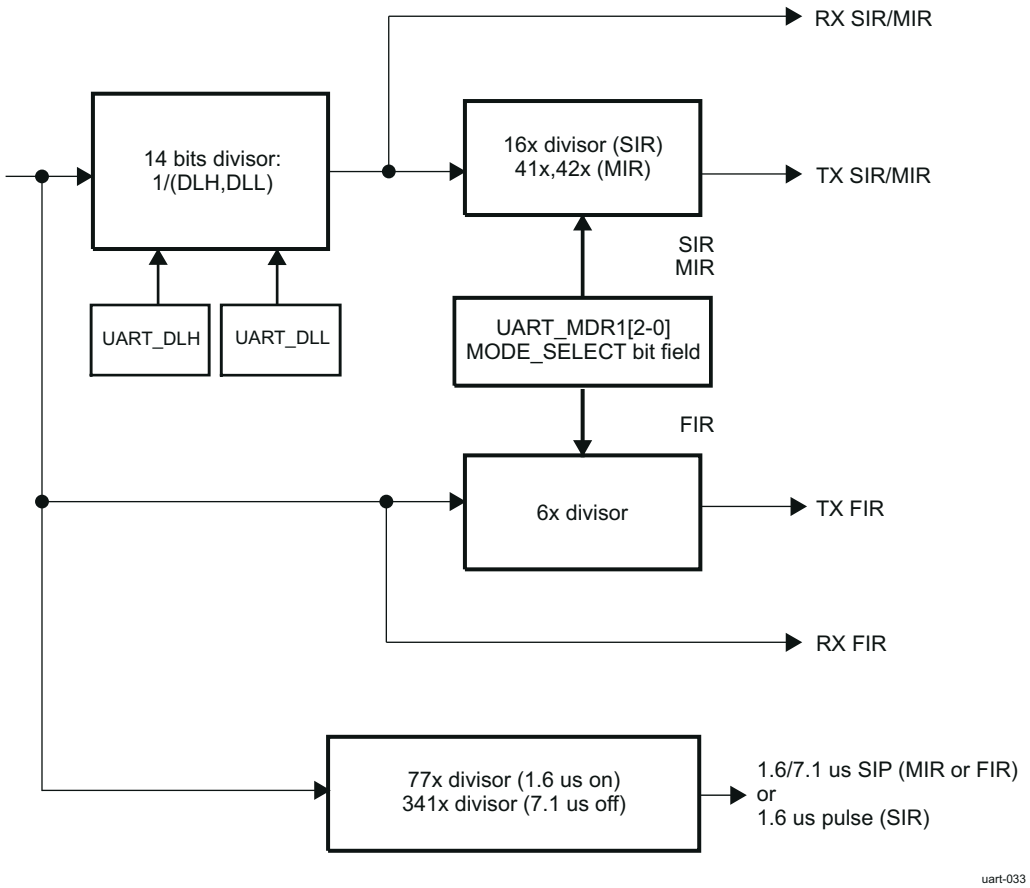


Figure 13-81. IrDA Baud Rate Generator

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.

13.1.4.4.8.3.2 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 13-99 lists the IrDA baud rate settings.

Table 13-99. 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

Table 13-99. IrDA Baud Rate Settings (continued)

Baud Rate	IR Mode	Baud Multiple	Encoding	DLH, DLL (Decimal)	Actual Baud Rate	Error (%)	Source Jitter (%)	Pulse Duration
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.

13.1.4.4.8.3.3 IrDA Data Formatting

The methods described in this section apply to all IrDA modes (SIR, MIR, and FIR).

13.1.4.4.8.3.3.1 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.

13.1.4.4.8.3.3.2 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.

13.1.4.4.8.3.3.3 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.

If address checking is disabled, all received frames write to the RX FIFO.

13.1.4.4.8.3.3.4 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.

13.1.4.4.8.3.3.5 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.

13.1.4.4.8.3.3.6 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 13-82). The STATUS FIFO must be read until empty (a maximum of eight reads is required).

13.1.4.4.8.3.3.7 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.

13.1.4.4.8.3.3.8 Overrun During Receive

Overrun during receive for the IrDA mode has the same function as that for the UART mode (see Section 13.1.4.4.8.1.5.6, *Overrun During Receive*).

13.1.4.4.8.3.3.9 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.

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.

13.1.4.4.8.3.4 SIR Mode Data Formatting

This section provides specific instructions for SIR mode programming.

13.1.4.4.8.3.4.1 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.

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.

13.1.4.4.8.3.4.2 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.

13.1.4.4.8.3.4.3 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.).

13.1.4.4.8.3.5 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.

13.1.4.4.8.4 CIR Mode

13.1.4.4.8.4.1 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 13-82.

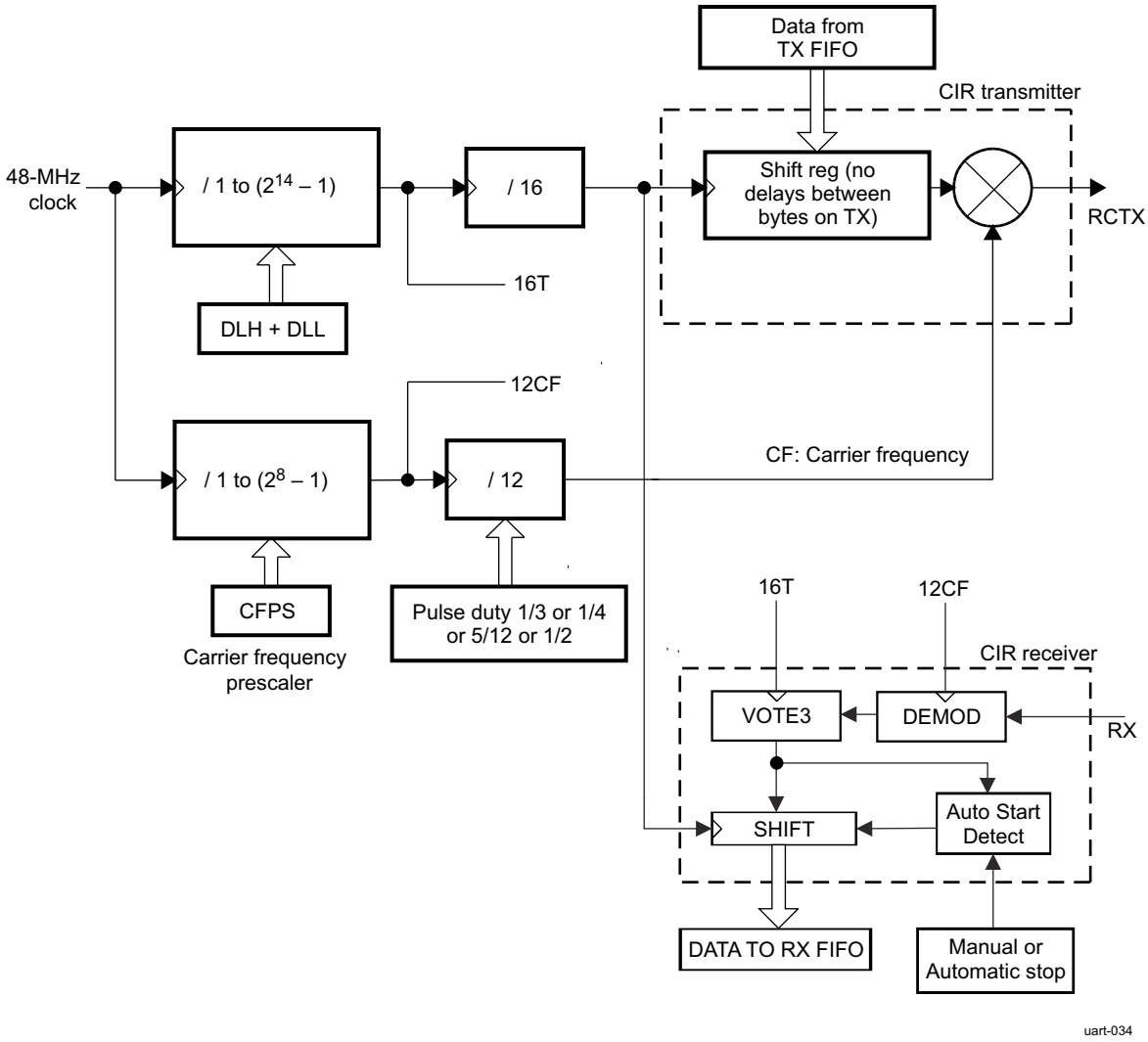


Figure 13-82. CIR Mode Block Components

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 13-100 shows the duty cycle.

Table 13-100. 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

13.1.4.4.8.4.2 CIR Data Formatting

The methods described in this section apply to all CIR modes.

13.1.4.4.8.4.2.1 IR RX Polarity Control

The IR RX polarity control for CIR mode has the same function as that for IrDA mode (see Section 13.1.4.4.8.3.3.1, *IR RX Polarity Control*).

13.1.4.4.8.4.2.2 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)

13.1.4.4.8.4.2.3 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.

13.1.4.5 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 13.1.4.4.7.1, *Register Access Modes*.

13.1.4.5.1 UART Global Initialization

13.1.4.5.1.1 Surrounding Modules Global Initialization

This section identifies the requirements for initializing the surrounding modules when the UART 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 UART.

For more information, see .

13.1.4.5.1.2 UART Module Global Initialization

The procedure in Table 13-101 can be used to initialize UART when performing software reset.

Table 13-101. UART Global Initialization

Step	Register/Bit Field/Programming Model	Value
Perform a software reset.	UART_SYSC[1] SOFTRESET	1
Wait until reset is finished.	UART_SYSS[0] RESETDONE	=1

13.1.4.5.2 UART Mode selection

Table 13-102 describes how to set different register access mode.

Table 13-102. 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

13.1.4.5.3 UART Submode selection

This section describes how to set different register access submode.

Table 13-103. UART Configure Register Access Submode TCR_TLR

Step	Register/Bit Field/Programming Model	Value
Configure the submode TCR_TLR		
Configure mode B	see Table 13-102	
Enable writing to register bits UART_MCR[7-5]	UART_EFR[4] ENHANCED_EN	1
Configure mode A	see Table 13-102	0x1
Set the submode TCR_TLR	UART_MCR[6] TCR_TLR	1

Table 13-104. 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 13-102	
Set the submode MSR_SPR	UART_EFR[4] ENHANCED_EN	0
Second option: configure the submode MSR_SPR		
Configure mode B	see Table 13-102	
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 13-105. UART Configure Register Access Submode XOFF

Step	Register/Bit Field/Programming Model	Value
Configure of the XOFF		
Configure B	see Table 13-102	
Set the submode XOFF	UART_EFR[4] ENHANCED_EN	0

13.1.4.5.4 UART Load FIFO trigger and DMA mode settings

13.1.4.5.4.1 DMA mode Settings

To enable and configure program the DMA mode, perform the following steps:

Table 13-106. 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 13.1.4.4.6.4	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 13.1.4.4.6.4	UART_SCR[2-1] DMA_MODE_2	-

13.1.4.5.4.2 FIFO Trigger Settings

In this section is described configuration and settings of FIFO trigger level, which enable DMA and interrupt generation.

Table 13-107. Load FIFO Triggers Defined by the FCR

Step	Register/Bit Field/Programming Model	Value
Configure register submode TCR_TLR	see Table 13-103	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 13-108. Load FIFO Triggers Defined by the TLR

Step	Register/Bit Field/Programming Model	Value
Configure register submode TCR_TLR	see Table 13-103	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 13-109. Load FIFO Triggers Defined by the Concatenated Value

Step	Register/Bit Field/Programming Model	Value
Configure register submode TCR_TLR	see Table 13-103	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-

Table 13-109. Load FIFO Triggers Defined by the Concatenated Value (continued)

Step	Register/Bit Field/Programming Model	Value
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-

13.1.4.5.5 UART Protocol, Baud rate and interrupt settings

13.1.4.5.5.1 Baud rate settings

Table 13-110. 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 13-102	
Enable access to UART_IER_UART[7-4]	UART_EFR[4] ENHANCED_EN	1
Switch register operational mode	see Table 13-102	
Disable sleep mode	UART_IER_UART[4] SLEEP_MODE	0
Switch to register configuration mode A or B	see Table 13-102	
Set the appropriate divisor value	UART_DLL[7-0] CLOCK_LSB UART_DLH[5-0] CLOCK_MSB	0x-

13.1.4.5.5.2 Interrupt settings

Table 13-111. UART Interrupt Settings

Step	Register/Bit Field/Programming Model	Value
Switch to register configuration mode B	see Table 13-102	0x7
Enable access to UART_IER_UART[7-4]	UART_EFR[4] ENHANCED_EN	1
Switch register operational mode	see Table 13-102	
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-

13.1.4.5.5.3 Protocol settings

Load the desired protocol formatting (parity, stop-bit, character length) and switch to register operational mode.

Table 13-112. UART Protocol Settings

Step	Register/Bit Field/Programming Model	Value
Load desired protocol formatting, see Section 13.1.4.4.8.1.5.1 , *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

13.1.4.5.5.4 UART/RS-485/IrDA(SIR/MIR/FIR)/CIR

Table 13-113. UART Mode Selection

Step	Register/Bit Field/Programming Model	Value
Load the desired UART/IrDA (SIR, MIR, FIR)/CIR modes, see Section 13.1.4.4.7.2, *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 13.1.4.4.7.2, *UART/RS-485/IrDA (SIR, MIR, FIR)/CIR Mode Selection*	UART_MDR3[4] DIR_EN	0x1

13.1.4.5.5.5 UART Multi-drop Parity Address Match Mode Configuration

Table 13-114. 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-
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

13.1.4.5.6 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.

13.1.4.5.6.1 Hardware Flow Control Configuration

Table 13-115. UART Hardware Flow Control Configuration

Step	Register/Bit Field/Programming Model	Value
Configure register submode TCR_TLR	see Table 13-103	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-

13.1.4.5.6.2 Software Flow Control Configuration

Table 13-116. UART Software Flow Control Configuration

Step	Register/Bit Field/Programming Model	Value
Set the register access submode XOFF	see Table 13-105	
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 13-103	
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-

13.1.4.5.7 IrDA Programming Model

13.1.4.5.7.1 SIR mode

13.1.4.5.7.1.1 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 13-117. 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	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
Enable the UART_RHR interrupt	UART_IER_IRDA[0] RHR_IT	1

13.1.4.5.7.1.2 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 13-118. 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

13.1.4.5.7.2 MIR mode

13.1.4.5.7.2.1 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 13-119. 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

Table 13-119. MIR Mode Receive Settings (continued)

Step	Register/Bit Field/Programming Model	Value
Enable the UART_RHR interrupt	UART_IER_IRDA[0] RHR_IT	1

13.1.4.5.7.2.2 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 13-120. 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
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

13.1.4.5.7.3 FIR mode

13.1.4.5.7.3.1 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 13-121. 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	0x0
	UART_DLH[7-0] CLOCK_MSB	
FIFO clear and enable	UART_FCR[2-0]	0x7
Set the FIFO trigger level	see Section 13.1.4.5.4, *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

13.1.4.5.7.3.2 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 13-122. 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

Table 13-122. FIR Mode Transmit Settings (continued)

Step	Register/Bit Field/Programming Model	Value
Enable access to change UART_FCR[0]	UART_DLL[7-0] CLOCK_LSB	0x0
	UART_DLH[5-0] CLOCK_MSB	
FIFO clear and enable	UART_FCR[2-0]	0x7
Set the FIFO trigger level	see Section 13.1.4.5.4, *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	0x4
	UART_TXFLH[7-0] TXFLH	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
SIP is sent at the end of transmission	UART_ACREG[3] SEND_SIP	1

13.2 High-speed Serial Interfaces

This section describes the high-speed serial interfaces in the device.

13.2.1 Gigabit Ethernet Switch (CPSW)

This chapter describes the Gigabit Ethernet Switch (CPSW) subsystem in the device.

13.2.1.1 CPSW0 Overview

The 3-port Gigabit Ethernet Switch (CPSW0) subsystem provides Ethernet packet communication for the device and can be configured as an Ethernet switch.

The device has one 3-port Gigabit Ethernet Switch subsystem named CPSW0 which supports two external Ethernet interfaces and one internal CPDMA host interface.

Figure 13-83 shows the CPSW0 module overview.

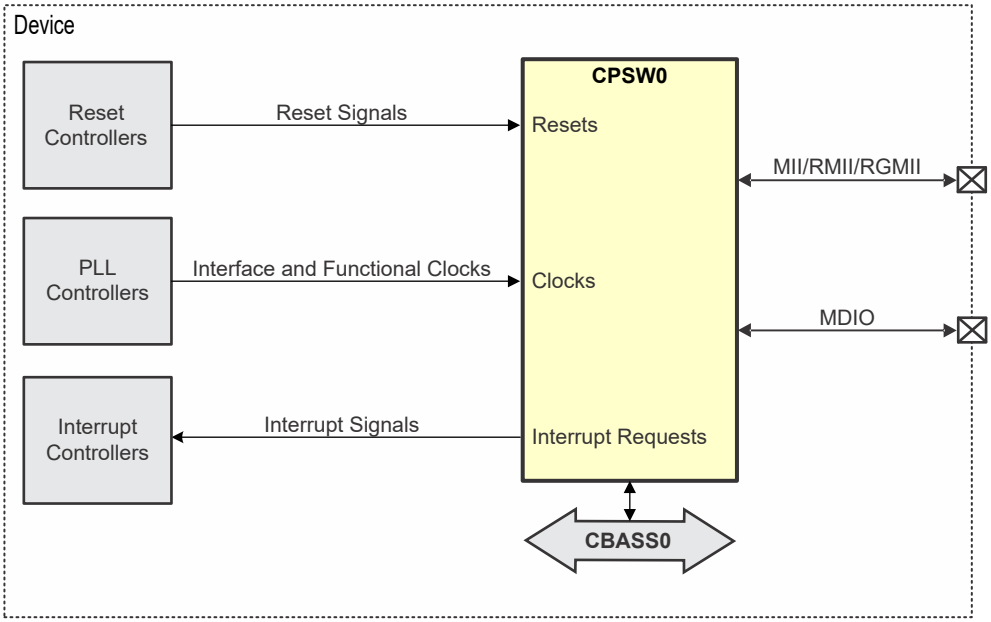


Figure 13-83. CPSW Module Overview

13.2.1.1.1 CPSW0 Features

The 3-port CPSW0 subsystem provides the following features:

- Two Ethernet ports (Port 1/Port 2) with selectable MII, RMII, and RGMII interfaces and a single internal Communications Port Programming Interface (CPPI) port (Port 0)
- Synchronous 10/100/1000 Mbit operation with Flexible logical FIFO-based packet buffer structure
  - Full duplex mode supported in 10/100/1000 Mbps modes
  - Half-duplex mode supported in 10/100 Mbps modes only
- 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
- One CPDMA CPPI 3.0 DMA Host Interface (Port 0)
- Emulation Mode, Digital loopback, and FIFO loopback modes supported
- RAM Error Detection and Correction (SECDED)
- 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)
  - Timestamp module capable of time stamping external timesync events like generating Pulse-Per-Second outputs
  - CPTS module that supports time stamping for IEEE1588 with support for 8 hardware push events and generation of compare output pulses
- DSCP Priority Mapping (IPv4 and IPv6)
- Energy Efficient Ethernet (EEE) support (802.3az)
- Non-Blocking switch fabric with Flow Control Support (802.3x) and Wire rate switching (802.1d)
- Time Sensitive Network (TSN) Support
  - IEEE 802.1Qbv Enhancements for Scheduled Traffic

- Address Lookup Engine (ALE)
  - 512 ALE table entries with configurable number of addresses plus VLANs
  - Wire rate lookup with spanning tree support
  - Host controlled time-based aging and/or auto-aging
  - L2 address lock and L2 filtering support
  - MAC authentication (802.1x) and address blocking
  - Receive/Destination-based Multicast and Broadcast rate limits
  - OUI (Vendor ID) host accept/deny feature and Source port locking
  - Configurable number of classifier/policers (32)
  - VLAN support
    - 802.1Q compliant
      - Auto add port VLAN for untagged frames on ingress
      - Auto VLAN removal on egress and auto pad to minimum frame size
- EtherStats and 802.3 Stats Remote Network Monitoring (RMON) statistics gathering (per port statistics)
- Support for Ethernet MAC transmit to MAC receive digital loopback mode

13.2.1.1.2 CPSW0 Not Supported Features

The following features are not supported by the CPSW0 switch:

- Maximum frame size of 9600 bytes
- GMII Mode
- SGMII Mode
- MACSEC
- Synchronous Ethernet
- Cut through
- Time Sensitive Network Support
  - IEEE P802.3br/D2.0 Interspersing Express Traffic

13.2.1.1.3 CPSW Terminology

AVB	Audio Video Bridging
AVBTP	Audio Video Bridging Transport Protocol
BCCA	Best Controller Clock Algorithm
CFI	Canonical Format Indicator
CPPI	Communications Port Programming Interface
CPSW	Common Platform Switch
DLR	Device Level Ring
DSCP	Differentiated Services Code Point
EEE	Energy Efficient Ethernet
EMAC	Ethernet Media Access Control
EOP	End of Packet
EOQ	End of Queue
IPG	Inter-Packet Gap
LPI	Low Power Indicator
MDIO	Management Data Input/Output
MOF	Middle of Frame
OUI	Organizationally Unique Identifier
PFC	Priority based Flow Control
PTP	Precision Time Protocol
RMON	Remote Monitoring
RTCP	RTP Control Protocol

RTP	Real-time Transport Protocol
SCR	Switched Central Resource
SRP	Stream Reservation Protocol
TOS	Type of Service
VLAN	Virtual Local Area Network

13.2.1.2 CPSW0 Environment

This section describes the CPSW0 external connections (environment).

13.2.1.2.1 CPSW0 MII Interface

Figure 13-84 shows a device with integrated EMAC and MDIO interfaced via a MII connection in a typical system. The EMAC module also includes a transmit error (MTXER) pin for Energy Efficient Ethernet operations.

The individual EMAC and MDIO signals for the MII interface are summarized in Table 13-123.

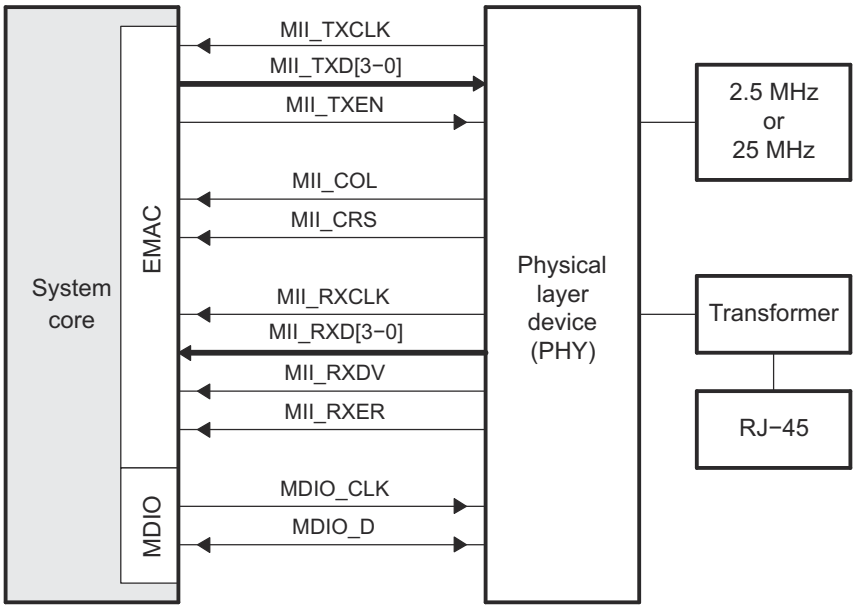


Figure 13-84. Ethernet Configuration—MII Connections

Table 13-123. EMAC and MDIO Signals for MII Interface

Signal	Type	Description
MII_TXCLK	I	Transmit clock (MII_TXCLK). The transmit clock is a continuous clock that provides the timing reference for transmit operations. The MII_TXD and MII_TXEN signals are tied to this clock. The clock is generated by the PHY and is 2.5 MHz at 10 Mbps operation and 25 MHz at 100 Mbps operation.
MII_TXD[3-0]	O	Transmit data (MII_TXD). The transmit data pins are a collection of 4 data signals comprising 4 bits of data. MTDX0 is the least-significant bit (LSB). The signals are synchronized by MII_TXCLK and valid when MII_TXEN is asserted or de-asserted.
MII_TXEN	O	Transmit enable (MII_TXEN). The transmit enable signal indicates that the MII_TXD pins are generating nibble data for use by the PHY. It is driven synchronously to MII_TXCLK.
MII_COL	I	Collision detected (MII_COL). In half-duplex operation, the MII_COL pin is asserted by the PHY when it detects a collision on the network. It remains asserted while the collision condition persists. This signal is not necessarily synchronous to MII_TXCLK nor MII_RXCLK. In full-duplex operation, the MII_COL pin is used for hardware transmit flow control. Asserting the MII_COL pin will stop packet transmissions; packets in the process of being transmitted when MII_COL is asserted will complete transmission. The MII_COL pin should be held low if hardware transmit flow control is not used.
MII_CRS	I	Carrier sense (MII_CRS). In half-duplex operation, the MII_CRS pin is asserted by the PHY when the network is not idle in either transmit or receive. The pin is deasserted when both transmit and receive are idle. This signal is not necessarily synchronous to MII_TXCLK nor MII_RXCLK. In full-duplex operation, the MII_CRS pin should be held low.
MII_RXCLK	I	Receive clock (MII_RXCLK). The receive clock is a continuous clock that provides the timing reference for receive operations. The MII_RXD, MII_RXDV, and MII_RXER signals are tied to this clock. The clock is generated by the PHY and is 2.5 MHz at 10 Mbps operation and 25 MHz at 100 Mbps operation.
MII_RXD[3-0]	I	Receive data (MII_RXD). The receive data pins are a collection of 4 data signals comprising 4 bits of data. MRDX0 is the least-significant bit (LSB). The signals are synchronized by MII_RXCLK and valid when MII_RXDV is asserted or de-asserted.
MII_RXDV	I	Receive data valid (MII_RXDV). The receive data valid signal indicates that the MII_RXD pins are generating nibble data for use by the EMAC. It is driven synchronously to MII_RXCLK.
MII_RXER	I	Receive error (MII_RXER). The receive error signal is asserted for one or more MII_RXCLK periods to indicate that an error was detected in the received frame. This is meaningful only during data reception when MII_RXDV is active.
MDIO_CLK	O	Management data clock (MDIO_CLK). The MDIO data clock is sourced by the MDIO module on the system. It is used to synchronize MDIO data access operations done on the MDIO pin. The frequency of this clock is controlled by the CLKDIV bits in the MDIO control register (CONTROL).
MDIO_D	I/O	Management data input output (MDIO_D). The MDIO data pin drives PHY management data into and out of the PHY by way of an access frame consisting of start of frame, read/write indication, PHY address, register address, and data bit cycles. The MDIO_D pin acts as an output for all but the data bit cycles at which time it is an input for read operations.

13.2.1.2.2 CPSW0 RMII Interface

Figure 13-85 shows a device with integrated EMAC and MDIO interfaced via a RMII connection in a typical system. The individual CPSW0 and MDIO signals for the RMII interface are summarized in Table 13-124.

The CPSW0 module integrated in the device supports internal and external clock sources in RMII mode. Figure 13-85 shows the internal clock source for RMIIn_MHZ_50_CLK clock. It is 50 MHz clock source that is provided on the CLKOUT0 device pin. This clock has to be routed on the PCB to the RMII_REF_CLK device pin and the external PHY, RMII clock input.

For more information, refer to either the IEEE 802.3 standard or ISO/IEC 8802-3:2000(E).

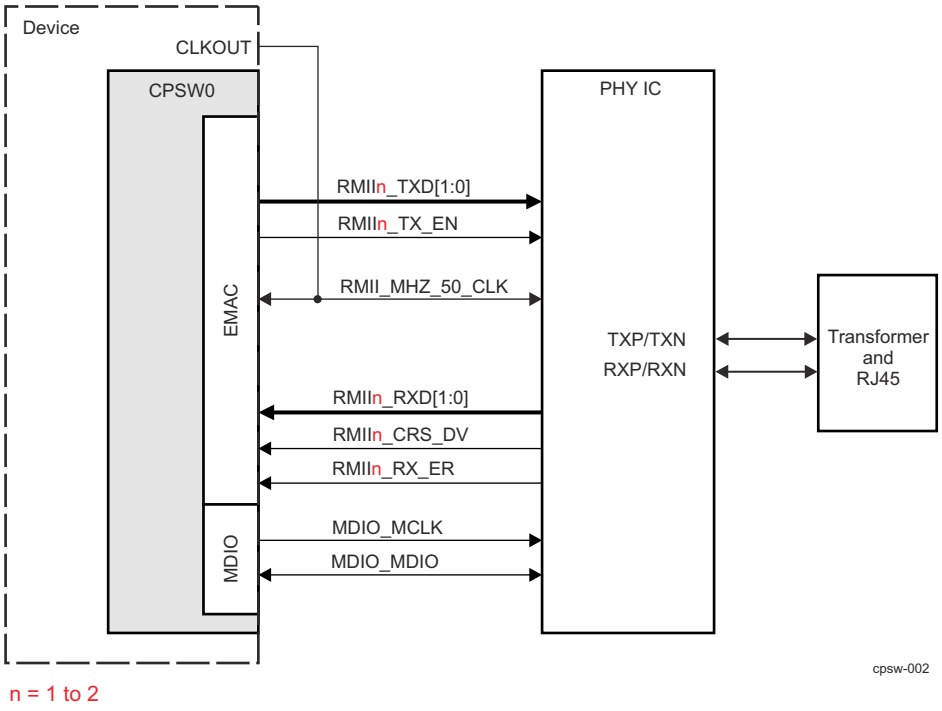


Figure 13-85. RMII Interface Typical Application (Internal Clock Source)

Figure 13-86 shows the external clock source for RMIIn_MHZ_50_CLK clock. In this case a 50 MHz clock is available on the PCB and it can be sourced from an oscillator or from the Ethernet PHY. This externally generated clock should be routed to both RMII_REF_CLK device pin and the external PHY, RMII clock input.

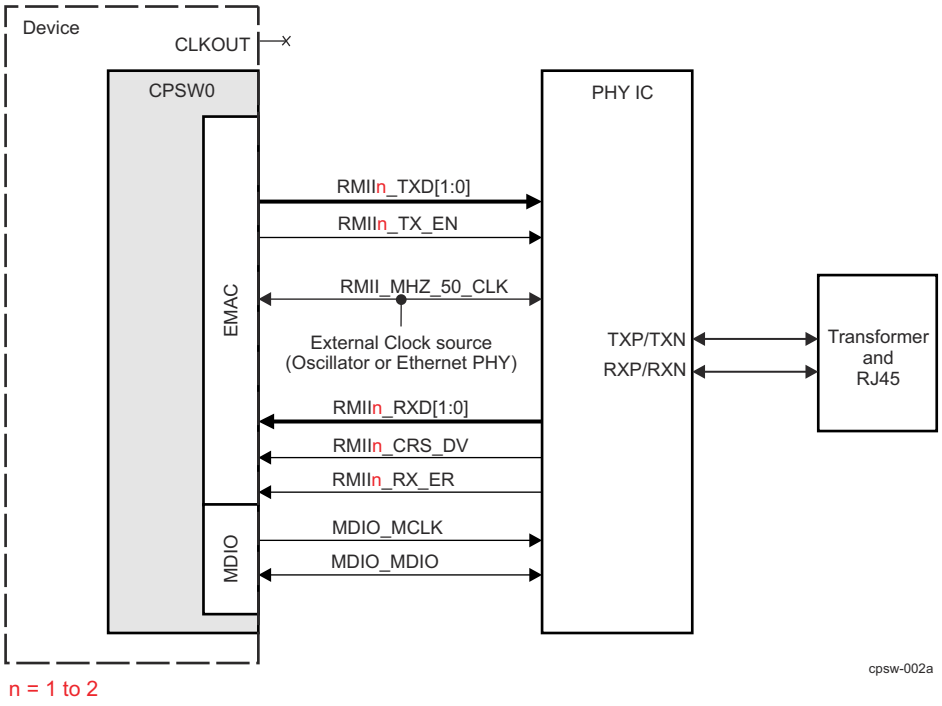


Figure 13-86. RMII Interface Typical Application (External Clock Source)

Table 13-124. RMII I/O Description

Signal[2]	Device Pin(s)	I/O[1]	Description
RMIIn_TXD[1:0]	RMIIn_TXD[1:0]	O	Transmit data. The transmit data pins are a collection of 2 bits of data. TXD0 is the least-significant bit (LSB). The signals are synchronized by RMII_MHZ_50_CLK and valid only when RMIIn_TX_EN is asserted.
RMIIn_TX_EN	RMIIn_TX_EN	O	RMII transmit enable. The transmit enable signal indicates that the RMIIn_TXD[1:0] pins are generating data for use by the PHY. RMIIn_TX_EN is synchronous to RMII_MHZ_50_CLK.
RMII_MHZ_50_CLK	RMII_REF_CLK	I	RMII 50MHz reference clock. The reference clock is used to synchronize all RMII signals. RMII_MHZ_50_CLK must be continuous and fixed at 50 MHz.
RMIIn_RXD[1:0]	RMIIn_RXD[1:0]	I	Receive data. The receive data pins are a collection of 2 bits of data. RXD0 is the least-significant bit (LSB). The signals are synchronized by RMII_MHZ_50_CLK and valid only when RMIIn_CRS_DV is asserted and RMIIn_RX_ER is de-asserted.
RMIIn_CRS_DV	RMIIn_CRS_DV	I	Carrier sense/receive data valid. Multiplexed signal between carrier sense and receive data valid.
RMIIn_RX_ER	RMIIn_RX_ER	I	Receive error. The receive error signal is asserted to indicate that an error was detected in the received frame.
MDIO_MCLK	MDIO0_MDC	O	Management data clock (MDIO_MCLK). The MDIO data clock is sourced by the MDIO module on the system. It is used to synchronize MDIO data access operations done on the MDIO0_MDIO data pin.
MDIO_MDIO	MDIO0_MDIO	I/O	MDIO data pin drives PHY management data into and out of the PHY by way of an access frame consisting of start of frame, read/write indication, PHY address, register address, and data bit cycles. The MDIO0_MDIO pin acts as an output for all but the data bit cycles at which time it is an input for read operations.

(1) I = Input; O = Output
(2) n 1 to 2

13.2.1.2.3 CPSW0 RGMII Interface

Figure 13-87 shows a device with integrated EMAC and MDIO interfaced via a RGMII connection in a typical system. The individual CPSW0 and MDIO signals for the RGMII interface are summarized in Table 13-125.

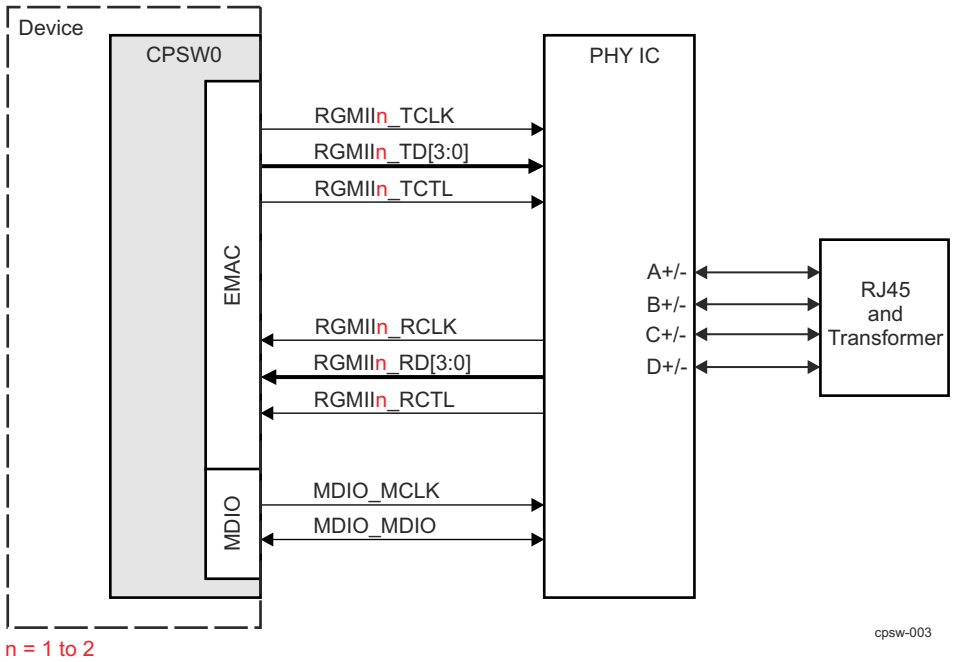


Figure 13-87. RGMII Interface Typical Application

Table 13-125. RGMII I/O Description

Signal[2]	Device Pin(s)	I/O[1]	Description
RGMIIn_TD[3:0]	RGMIIn_TD[3:0]	O	The transmit data pins are a collection of 4 bits of data. TD0 is the least-significant bit (LSB). The signals are valid only when RGMIIn_TCTL is asserted.
RGMIIn_TCTL	RGMIIn_TX_CTL	O	Transmit Control/enable. The transmit enable signal indicates that the TD pins are generating data for use by the PHY.
RGMIIn_TCLK	RGMIIn_TXC	O	The transmit reference clock. The clock is 2.5 MHz at 10 Mbps operation, 25 MHz at 100 Mbps operation, and 125 MHz at 1000 Mbps of operation.
RGMIIn_RD[3:0]	RGMIIn_RD[3:0]	I	The receive data pins are a collection of 4 bits of data. RD0 is the least-significant bit (LSB). The signals are valid only when RGMIIn_RX_CTL is asserted
RGMIIn_RCTL	RGMIIn_RX_CTL	I	The receive data valid/control signal indicates that the RD pins are nibble data for use by the EMAC.
RGMIIn_RCLK	RGMIIn_RXC	I	The receive clock is a continuous clock that provides the timing reference for receive operations. The clock is generated by the PHY and is 2.5 MHz at 10 Mbps operation, 25 MHz at 100 Mbps operation, 125 MHz at 1000 Mbps of operation.
MDIO_MCLK	MDIO0_MDC	O	Management data clock (MDIO_MCLK). The MDIO data clock is sourced by the MDIO module on the system. It is used to synchronize MDIO data access operations done on the MDIO0_MDIO pin.
MDIO_MDIO	MDIO0_MDIO	I/O	The MDIO0_MDIO pin drives PHY management data into and out of the PHY by way of an access frame consisting of start of frame, read/write indication, PHY address, register address, and data bit cycles. The MDIO0_MDIO pin acts as an output for all but the data bit cycles at which time it is an input for read operations.

(1) I = Input; O = Output
(2) n 1 to 2

Note

The Control Module registers assign the specific function to the device pads. For more information on Control Module settings, see Pad Configuration Registers in *Control Module (CTRL_MMR)* and the device-specific Datasheet.

13.2.1.3 CPSW Integration

There is 1x CPSW module integrated in the device. The diagram below provides a visual representation of the device integration details.

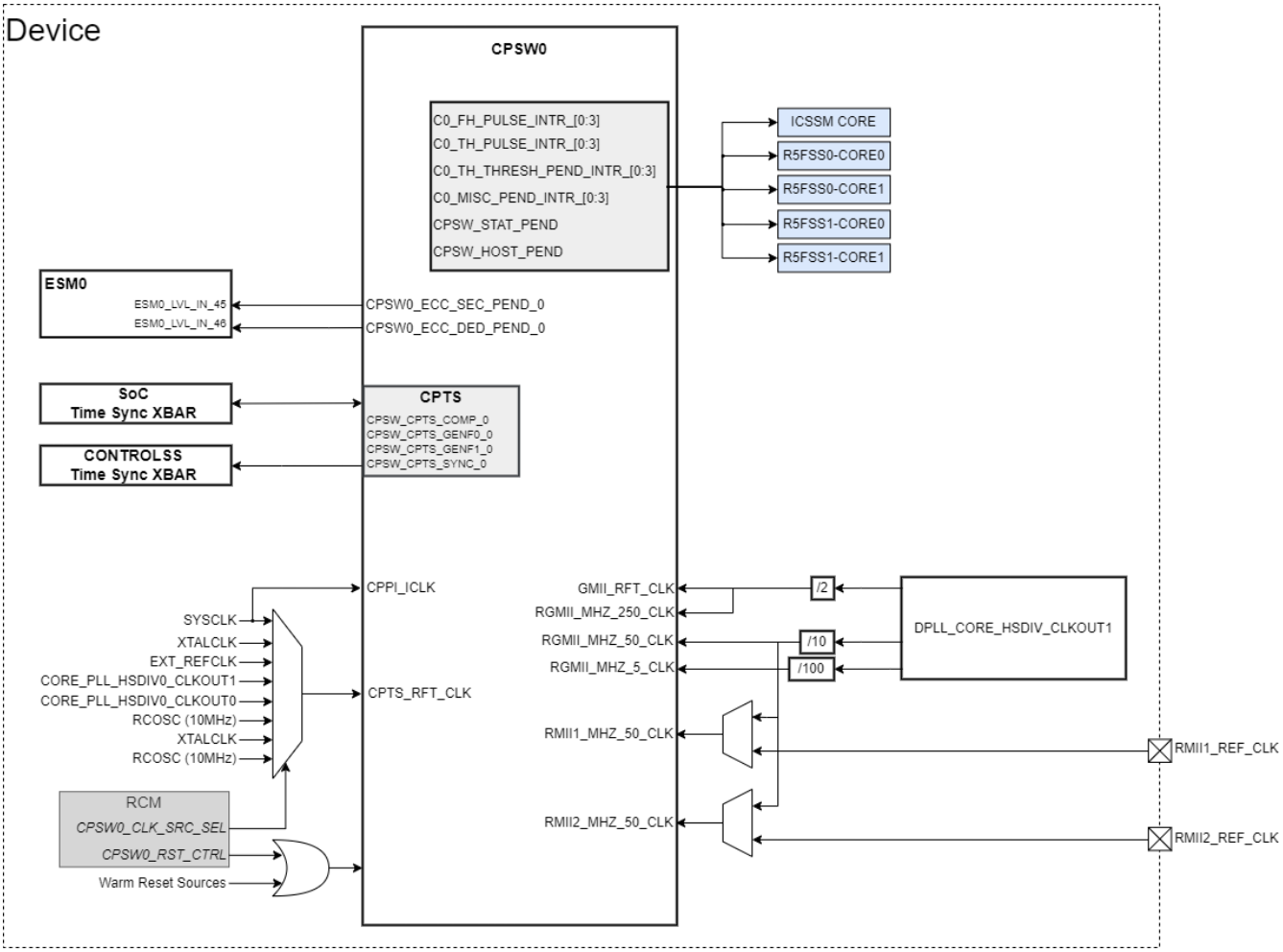


Figure 13-88. CPSW Integration Diagram

The tables below summarize the device integration details of CPSW0.

Table 13-126. *CPSW0* Device Integration

This table describes the module device integration details.

Module Instance	Device Allocation	SoC Interconnect
CPSW0	✓	INFRA0 VBUSP Interconnect

Table 13-127. *CPSW0* Clocks

This table describes the module clocking signals.

Module Instance	Module Clock Input	Source Clock Signal	Source	Default Freq	Description
CPSW0	CPPI_ICLK	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	CPSW0 Interface Clock
	CPTS_RFT_CLK	XTACLK	External XTAL	25 MHz	CPSW0 Interface Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	CPSW0 Interface Clock
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	CPSW0 Interface Clock
		DPLL_CORE_HSDIV0_CLKOUT1	PLL_CORE_CLK: HSDIV0_CLKOUT1	500 MHz	CPSW0 Interface Clock
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK: HSDIV0_CLKOUT0	400 MHz	CPSW0 Interface Clock
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	CPSW0 Interface Clock
		XTALCLK	External XTAL	25 MHz	CPSW0 Interface Clock
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	CPSW0 Interface Clock
	GMII_RFT_CLK	RGMII_250_CLK	RGMII 250 MHz Clock	250 MHz	CPSW0 Interface Clock
	RMII1_MHZ_50_CLK	RGMII_50_CLK	RGMII 50 MHz Clock	50 MHz	CPSW0 Interface Clock
		RMII1_REF_CLK	RMII1 Reference Clock	50 MHz[1]	CPSW0 Interface Clock
	RMII2_MHZ_50_CLK	RGMII_50_CLK	RGMII 50 MHz Clock	50 MHz	CPSW0 Interface Clock
		RMII2_REF_CLK	RMII2 Reference Clock	50 MHz[1]	CPSW0 Interface Clock
	RGMII_MHZ_50_CLK	RGMII_50_CLK	RGMII 50 MHz Clock	50 MHz	CPSW0 Interface Clock
	RGMII_MHZ_5_CLK	RGMII_5_CLK	RGMII 5 MHz Clock	5 MHz	CPSW0 Interface Clock
	RGMII_MHZ_250_CLK	RGMII_250_CLK	RGMII 250 MHz Clock	250 MHz	CPSW0 Interface Clock

Note

[1]The RMIIx_REF_CLK input pin can be drive by an external clock reference source. 50 MHz is required for proper operation.

Table 13-128. *CPSW0* Resets

This table describes the module reset signals.

Module Instance	Module Reset Input	Source Reset Signal	Source	Description
CPSW0	CPSW_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	CPSW0 Asynchronous Reset

Table 13-129. *CPSW0* Interrupt Requests

This table describes the module interrupt requests.

Module Instance	Module Interrupt Signal	Destination Interrupt Input	Destination	Type	Description
CPSW0	C0_FH_PULSE_INTR_[0:3]	C0_FH_PULSE_INTR	All R5FSS Cores ICSSM Core	Level	FHost (from host to Ethernet) paced pulse interrupt
	C0_TH_PULSE_INTR_[0:3]	C0_TH_PULSE_INTR	All R5FSS Cores ICSSM Core	Level	THost (from Ethernet to host) paced pulse interrupt
	C0_TH_THRESH_PULSE_INTR_[0:3]	C0_TH_THRESH_PULSE_INTR	All R5FSS Cores ICSSM Core	Level	THost (from Ethernet to host) non-paced pulse interrupt
	C0_MISC_PULSE_INTR_[0:3]	C0_MISC_PULSE_INTR	All R5FSS Cores ICSSM Core	Level	Miscellaneous non-paced pulse interrupt
	CPSW_STAT_PEND	STAT_PEND	All R5FSS Cores ICSSM Core	Level	Statistics level interrupt
	CPSW_HOST_PEND	HOST_PEND	All R5FSS Cores ICSSM Core	Level	CPDMA host error level interrupt
	CPSW_ECC_SEC_PULSE_INTR	ECC_SEC_PULSE_INTR	ESM	Level	ECC SEC pulse interrupt – output from CPSW ECC module.
	CPSW_ECC_DED_PULSE_INTR	ECC_DED_PULSE_INTR	ESM	Level	ECC DED pulse interrupt – output from CPSW ECC module.

Table 13-130. *CPSW0* Time Sync and Compare Event

This table describes the module capture event inputs.

Module Instance	Module Event	Destination Event Input	Destination	Type	Description
CPSW0	CPSW0_CPTS_COMP	SoC_TimeSyncXBAR[0:3]	CPSW0_CPTS_COMP_INTR	Level	CPSW0 Compare Event Interrupt
		C2K_TimeSyncXBAR[0:3]			
	CPSW0_CPTS_GENF0	SoC_TimeSyncXBAR[0:3]	CPSW0_CPTS_GENF0_INTR	Level	CPSW0 CPTS generator function event interrupt 0
		C2K_TimeSyncXBAR[0:3]			
	CPSW0_CPTS_GENF1	SoC_TimeSyncXBAR[0:3]	CPSW0_CPTS_GENF1_INTR	Level	CPSW0 CPTS generator function event interrupt 1
		C2K_TimeSyncXBAR[0:3]			
	CPSW0_CPTS_SYNC	SoC_TimeSyncXBAR[0:3]	CPSW0_CPTS_SYNC_INTR	Level	CPSW0 CPTS Sync Event Interrupt
		C2K_TimeSyncXBAR[0:3]			

Note

For more information on the interconnects, see the *System Interconnect* chapter.

For more information on power, reset, and clock management, see the corresponding sections within the *Device Configuration* chapter.

For more information on the device interrupt controllers, see the *Interrupt Controllers* chapter.

For pin information on RGMII_ID_MODE and RGMII_REFCLK_SEL, see Register information and the corresponding section within the *Device Configuration* chapter

13.2.1.4 CPSW0 Functional Description

The three-port switch Ethernet subsystem module (CPSW) is compliant to the IEEE Std 802.3 Specification. The CPPI CPDMA is compliant to the CPPI 3.0 and- CBA 3.1 specifications. The CPSW top level functional block diagram is shown in Figure 13-89.

13.2.1.4.1 Functional Block Diagram

The three-port Ethernet subsystem consists of:

- CPSW Peripheral Core
- One RGMIIn (where n = 1 to 2) interface module
- One RMIIn (where n = 1 to 2) interface module
- One MIIn (where n = 1 to 2) interface module
- One Host Port 0 CPPI 3.0 CPDMA
- CPSW subsystem control registers (REG)
- One MDIO interface module
- One Interrupt Controller module

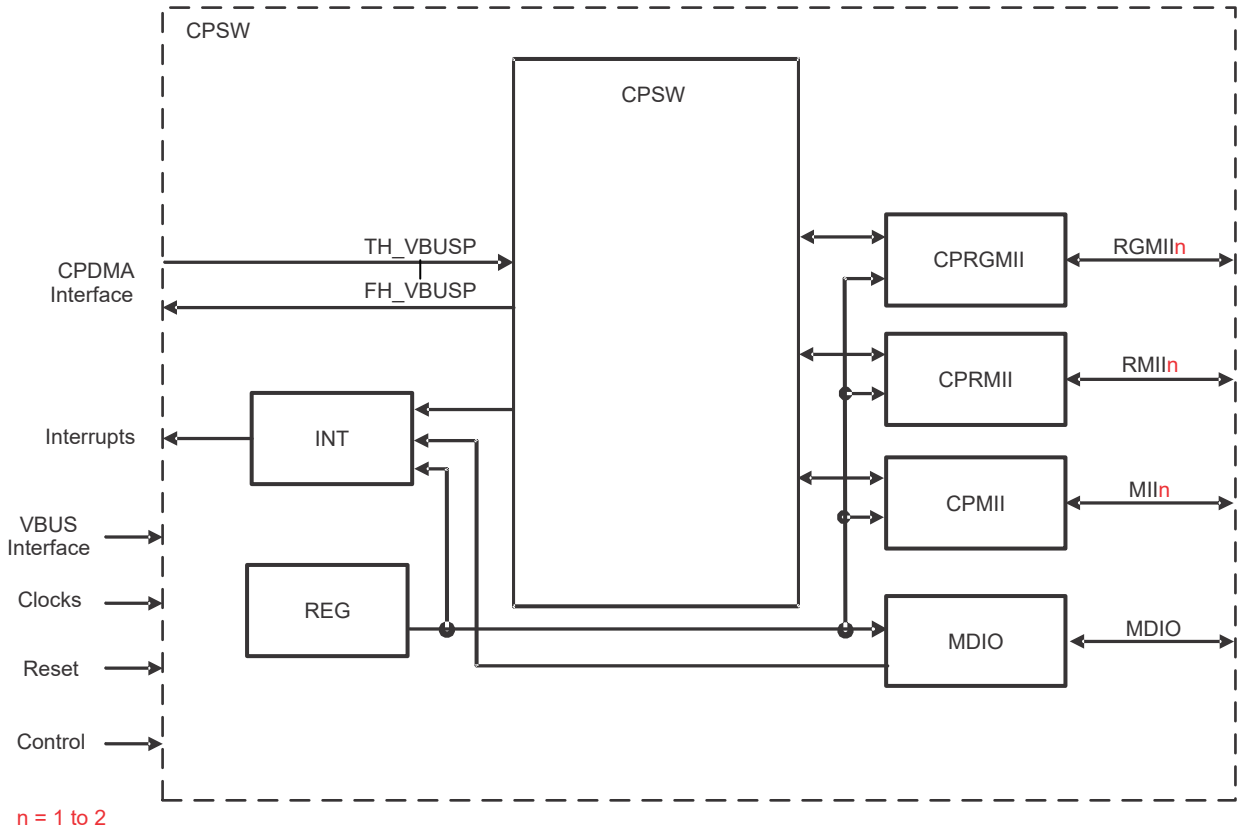


Figure 13-89. CPSW Functional Block Diagram

13.2.1.4.2 CPSW Ports

The Ethernet Subsystem has three ports. Port 0 is the Host port (internal to the Subsystem). Port 1 and 2 are the external ports connected to RGMII, RMII, MII interfaces as per the interface selected.

Naming conventions followed in this chapter:

- Port0 is referred to the CPDMA Host Port
- Port1 and 2 are referred to the interfaces RGMII/RMII/MII

13.2.1.4.2.1 Interface Mode Selection

The three-port switch (CPSW) Ethernet Subsystem has one 10/100/1000 Ethernet port with selectable RMII, RGMII, and MII interfaces.

The interface modes for all 2 Ethernet ports are selected by configuring the Ethernet interface mode selection bitfield (PORT_MODE_SEL) in the CTRLMMR_ENET1_CTRL and CTRLMMR_ENET2_CTRL registers.

See the device-specific Datasheet for configuring the pin mux mode as per the interface selected.

13.2.1.4.3 Clocking

13.2.1.4.3.1 Subsystem Clocking

CPSW clocking summary is shown in *CPSW Integration*.

13.2.1.4.3.2 Interface Clocking

Data is transmitted and received with respect to the reference clocks of the interface pins.

13.2.1.4.3.2.1 RGMII Interface Clocking

RGMII_RXC, RGMII_TXC frequencies are:

- 2.5 MHz at 10 Mbps
- 25 MHz at 100 Mbps
- 125 MHz at 1000 Mbps

Note

RGMII has ID_MODE for TX internal delay that is fixed and cannot be changed.

13.2.1.4.3.2.2 RMII Interface Clocking

RMII interface clock RMII_50MHZ_CLK frequency is:

- 50 MHz at 10 Mbps
- 50 MHz at 100 Mbps

For more details on RMII clocking, please see *CPSW0 Integration*

CTRLMMR_CLKOUT_CTRL[4]CLK_EN and CTRLMMR_CLKOUT_CTRL[0]CLK_SEL bits are used to enable and select the clock source for CLKOUT device pin.

13.2.1.4.3.2.3 MDIO Clocking

The MDIO clock is based on a divide-down of the interface (CPPI_ICLK) clock. The application software or driver must control the divide-down value.

See the CPSW_MDIO_CONTROL_REG register for configuring the Clock Divider ([15-0]CLKDIV) value.

13.2.1.4.4 Software IDLE

The submodule software idle register bits enable CPSW operation to be completely or partially suspended by software control. There are two CPSW submodules that contain software idle register bits. Each of the two submodules may be individually commanded to enter the idle state. The idle state is entered at packet boundaries, and no further packet operations will occur on an idled submodule until the idle command is removed. The CPSW software idle inhibits packages from starting to be unloaded from each port switch FIFO, but packets already in process are unaffected.

13.2.1.4.5 Interrupt Functionality

CPSW Ethernet Subsystem has six interrupt outputs:

- Cn_FH_PEND_INTR - FHost (from host to Ethernet) level interrupt
- Cn_TH_PEND_INTR - THost (from Ethernet to host) level interrupt
- Cn_TH_THRESH_PEND_INTR - THost (from Ethernet to host) non-paced level interrupt
- Cn_MISC_PEND_INTR - Miscellaneous level interrupt
  - ECC_SEC_PEND_INTR: ECC SEC level interrupt - from CPSW ECC module. This interrupt is also included in the C0_MISC_PEND_INTR if enabled or can be used separately if desired.
  - ECC_DED_PEND_INTR: ECC DED level interrupt - from CPSW ECC module. This interrupt is also included in the C0_MISC_PEND_INTR if enabled or can be used separately.
- STAT_PEND_INTR - Statistics level interrupt

- HOST_PEND_INTR - CPDMA host error level interrupt

Note

n = 0 to $num_cores-1

13.2.1.4.6 CPSW

The CPSW RMII/ RGMII interface is compliant to the IEEE Std 802.3 Specification.

The CPSW contains two Ethernet port interfaces (Ethernet port 1 and 2), one CPPI packet streaming interface host port (port 0), Common Platform Time Sync (CPTS), ALE Engine and Statistics (STATS). A top-level block diagram of the CPSW is shown in Figure 13-90.

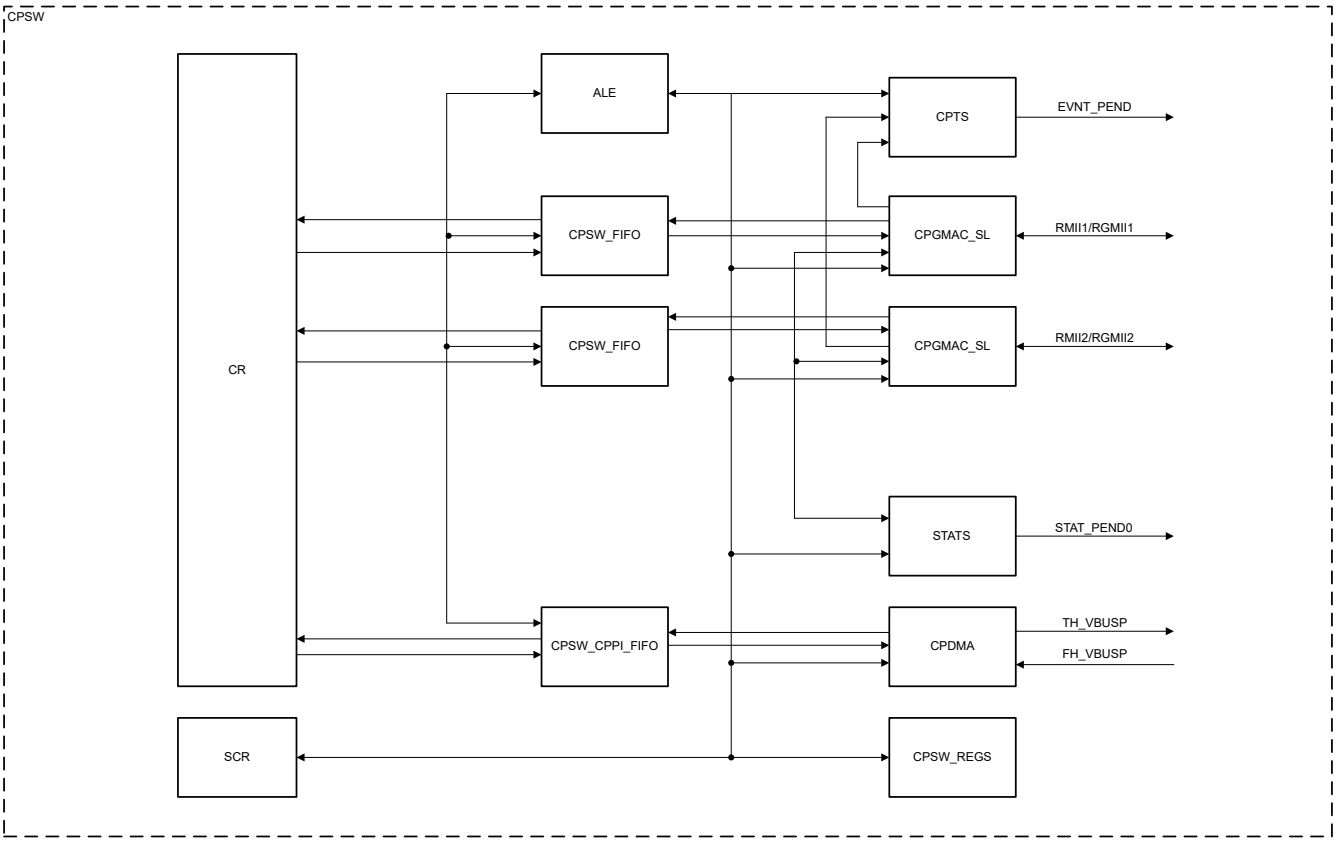


Figure 13-90. CPSW Block Diagram

13.2.1.4.6.1 Address Lookup Engine (ALE)

The Address Lookup Engine (ALE) is a sub-block of the CPSW Switch that processes all received packets and determines to which port(s) the packet should be forwarded. The ALE uses the incoming packet received port number, destination address, source address, length/type, and VLAN information to determine how the packet should be forwarded. The ALE outputs the port mask to the switch fabric that indicates the port(s) the packet should be forwarded to. The ALE is enabled when the ENABLE_ALE bit in the CPSW_ALE_CONTROL register is set. All packets are dropped when the ENABLE bit is cleared to 0.

13.2.1.4.6.1.1 Error Handling

In normal operation, the Ethernet port modules are configured to drop received packages that contain errors (runt, frag, oversize, jabber, crc, alignment, code etc.). However, when the CPSW_PN_MAC_CONTROL_REG_k configuration bit(s) RX_CEF_EN, RX_CSF_EN, or RX_CMF_EN are set, received Ethernet packets with errors are transferred to the host. When the ALE receives a packet that contains errors (due to a set header error bit), or a MAC control frame and does not receive an abort, the packet will be forwarded only to the host port (port 0). Packets with errors that are forwarded to the host have no VLAN untagging or drop due to rate limiting. No ALE learning occurs on packets with errors or mac control frames.

Learning is based on source address and lookup is based on destination address. Directed packets from the host are not learned, updated, or touched.

13.2.1.4.6.1.2 Bypass Operations

The ALE may be configured to operate in bypass mode by setting the ENABLE_BYPASS bit in the CPSW_ALE_CONTROL register. When in bypass mode, all Ethernet port received packets are forwarded only to the host port (port 0). In bypass mode, the ALE processes host port transmit packets the same as in normal mode. In general, packets would be directed by the host in bypass mode.

13.2.1.4.6.1.3 OUI Deny or Accept

The ALE may be configured to operate in OUI deny mode by setting the ENABLE_OUI_DENY bit in the CPSW_ALE_CONTROL register. When in OUI deny mode, a packet with a non-matching OUI source address will be dropped unless the destination address matches a supervisory table entry. When ENABLE_OUI_DENY bit is cleared, any packet source address matching an OUI address table entry will be dropped to the host unless the destination address matches with a supervisory address table entry. Broadcast packets will be dropped unless the broadcast address is entered into the table with the SUPER bit set. Unicast packets will be dropped unless the unicast address is in the table with BLOCK and SECURE both set (supervisory unicast packet).

13.2.1.4.6.1.4 Statistics Counting

The ALE sends per port statistics along with the frame routing to be counted in the CPSW statistics. There are multiple reasons that frames are dropped by the ALE. Each drop is counted in the CPSW statistics. For more information on ALE statistics refer to the *CPSW Network Statistics* section.

13.2.1.4.6.1.5 Automotive Security Features

The ALE has many automotive security features that most enterprise switches do not require.

- VLANs can be configured to not allow fragmented IPv4 frames.
- VLANs can be configured to only allow up to four different IPv4 Protocols or IPv6. Next Header values, for example a VLAN can be configured to only allow TCP traffic in both IPv4 and IPv6 packets.
- Drop invalid Source Addresses - drop Source Addresses with bit 40 set (Multicast/Broadcast indicator on Destination Addresses)
- IEEE802.3 Length Check, drop frames that the IEEE802.3 Length is not contained within the frame. (Ether Types 0-1500)
- Any Source Address can be secured to a port dropping any attempts from other ports to masquerade as a service.
- Any source or destination address can be blocked.
- Per Port or Per VLAN ingress checking, dropping traffic from non-member ports.
- Classification, Policing on L2 and L3 information.

13.2.1.4.6.1.6 CPSW Switching Solutions

The host port can operate in many different modes as well depending on the functionality of the host. It is important to understand the modes and configure them properly.

The ALE Table is designed to maximize the modes without compromise of the system functionality as well.

13.2.1.4.6.1.6.1 Basics of 3-port Switch Type

The 3-port switch has a host port that can operate in two fundamental modes. Bridge mode allows the host to extent the switched domain to another network like Wi-Fi or another multi-port switch. In this case the host must be able to see unknown unicast addresses so they can be broadcast to the other network. In Port mode the host need not see any unknown unicast traffic. The CPSW_ALE_CONTROL[8] EN_HOST_UNI_FLOOD bit determines the host mode for unknown unicast traffic. This bit should only be set if the user is bridging two or more networks together.

The 3-port switch can operate like a two-port switch using the ALE table just for the host info, the only adder is now other VLANs need to be supported for transit traffic between the external ports. This can easily be done using the default VLAN rules so no ALE table entries are used.

The 3-port switch can also operate in a fully authenticated environment where all network nodes are registered via the 802.1x based protocols. In this case the ALE table is filled with network node addresses that have been authenticated

13.2.1.4.6.1.7 VLAN Routing and OAM Operations

13.2.1.4.6.1.7.1 InterVLAN Routing

The CPSW module supports wire rate InterVLAN routing for a small number of routes, that is the host will setup an ALE classifier with an associated egress operation that will cause the CPSW to perform particular egress operations. The ALE can optionally check time to live validity as well.

The ALE uses the classifier along with an egress opcode, destination port mask and TTL check field to tell the CPSW how to manipulate the packet on the egress. The CPSW will use the opcode along with a per port operation table to process the packet. By setting up the CPSW egress operation table you can replace the DA, SA and VLAN along with optionally updating the time to live IP header field. This allows the CPSW to perform the routing function for a small set of routes without getting the local host/CPU involved.

The Egress opcode will only be use for a classifier match and the packet would normally be sent only to the host. That is the host would have routed the packet but the CPSW has been configured to do the work instead. In the event that the time-to-live check feature is enabled and the time-to-live is either 0 or 1, the packet will not get the egress opcode and instead be sent to the host as if the route is not setup. This allows the host to deal with invalid TTL fields.

13.2.1.4.6.1.7.2 OAM Operations

The ALE supports OAM loopback on ports so that a remote link can be tested. TA port placed in OAM loopback will echo packets received on a port back to the port with an egress opcode of 0xFF which will swap the SA and DA on egress in the CPSW.

Any supervisory packet will not be affected, so the spanning tree and other bridging functions are not affected.

Packets will only be echoed if the port is in OAM loopback mode, the received packet in not a supervisor packet, the port is in a forwarding state, the packet received DA!=SA, and there are no errors in the packet.

When a port is in OAM loopback the port will not egress traffic from other ports, no address for loop backed traffic will be learned if enabled. Any packet received on the OAM loopback port with an error will be processed as if the port is not in OAM. That is if the host has enabled copy errored frames the errored frames will be sent to the host instead.

13.2.1.4.6.1.8 Supervisory packets

Multicast supervisory packets are designated by the SUPER bit in the table entry. Unicast supervisory packets are indicated when BLOCK and SECURE are both set. Supervisory packets are not dropped due to rate limiting, OUI, or VLAN processing. The purpose of supervisory packets is to allow packets that would be otherwise blocked to be forwarded for special purposes.

13.2.1.4.6.1.9 Address Table Entry

The ALE table contains multiple table entry types. Each table entry represents a free entry, an address, a VLAN, an address/VLAN pair, or an OUI address. Software should ensure that there are no double address entries in the table. The double entry used would be indeterminate. Reserved table bits must be written with zeroes.

Source Address learning occurs for packets with a unicast, multicast or broadcast destination address and a unicast or multicast (including broadcast) source address. Multicast source addresses have the group bit (bit 40) cleared before ALE processing begins, changing the multicast source address to a unicast source address. A multicast address of all ones is the broadcast address which may be added to the table. A learned unicast source address is added to the table with the following control bits:

Table 13-131. Learned Address Control Bits

Bit(s)	Value
Ageable	1
Touch	1
BLOCK	0

Table 13-131. Learned Address Control Bits (continued)

Bit(s)	Value
SECURE	0

If a received packet has a source address that is equal to the destination address then the following occurs:

- The address is learned if the address is not found in the table.
- The address is updated if the address is found.
- The packet is dropped.

Table Entry Type

00 - Free Entry

01 - Address Entry : unicast or multicast determined by destination **address bit 40.**

10 - VLAN entry

11 - VLAN Address Entry : unicast or multicast determined by **address bit 40.**

13.2.1.4.6.1.9.1 Free Table Entry

Table 13-132. Free Table Entry Bit Values

70:62	61:60	59:0
Reserved	ENTRY_TYPE (00)	Reserved

Table Entry Type (ENTRY_TYPE)

00: Free entry

13.2.1.4.6.1.9.2 OUI Unicast Address Table Entry

Table 13-133. OUI Unicast Address Table Entry Bit Values

70:64	63:62	61:60	59:48	47:24	23:0
Reserved	UNICAST_TYPE (10)	ENTRY_TYPE (01)	Reserved	UNICAST_OUI	Reserved

Unicast Type (UNICAST_TYPE)

This field indicates the type of unicast address the table entry contains.

00 - Unicast address that is not ageable.

01 - Ageable unicast address that has not been touched.

10 - OUI address - lower 24-bits are don't cares (not ageable).

11 - Ageable unicast address that has been touched.

Table Entry Type (ENTRY_TYPE)

Address entry. Unicast or multicast determined by address bit 40.

01: Address entry. Unicast or multicast determined by address bit 40.

Packet Address (UNICAST_OUI)

For an OUI address, only the upper 24-bits of the address are used in the source or destination address lookup.

13.2.1.4.6.1.9.3 Unicast Address Table Entry (Bit 40 == 0)

Table 13-134. Unicast Address Table Entry Bit Values

70:69	68	67:66	65	64	63	62	61:60	59:48	47:0
RESERVED	TRUNK	PORT_NUMBER	BLOCK	SECURE	TOUCH	AGEABLE	ENTRY_TYPE (3h)	RESERVED	UNICAST_ADDRESS

Trunk Indicator (TRUNK)

0h = The port bits in the entry are the port number

1h = The port bits in the entry are the trunk number

Port Number (PORT_NUMBER)

This field indicates the port number (not port mask) that the packet with a unicast destination address may be forwarded to. Packets with unicast destination addresses are forwarded only to a single port (but not the receiving port).]

Block (BLOCK)

The block bit indicates that a packet with a matching source or destination address should be dropped (block the address).

0h = Address is not blocked.

1h = Drop a packet with a matching source or destination address (secure must be zero)

If block and secure are both set, then they no longer mean block and secure. When both are set, the block and secure bits indicate that the packet is a unicast supervisory (super) packet and they determine the unicast forward state test criteria. If both bits are set then the packet is forwarded if the receive port is in the Forwarding/Blocking/Learning state. If both bits are not set then the packet is forwarded if the receive port is in the Forwarding state.

Secure (SECURE)

This bit indicates that a packet with a matching source address should be dropped if the received port number is not equal to the table entry PORT_NUMBER.

0h = Received port number is a don't care.

1h = Drop the packet if the received port is not the secure port for the source address and do not update the address (block must be zero)

Touch Indicator (TOUCH)

Only valid when AGEABLE it a 1h.

0h = Ageable unicast address has not been touched

1h = Ageable unicast address that has been touched

Ageable (AGEABLE)

This bit indicates that the address is ageable.

0h = Unicast address that is not ageable

1h = Unicast address that is ageable

Table Entry Type (ENTRY_TYPE)

Address entry. Unicast or multicast determined by address bit 40.

01: Address entry. Unicast or multicast determined by address bit 40.

Packet Address (UNICAST_ADDRESS)

This is the 48-bit packet MAC address. All 48-bits are used in the lookup.

13.2.1.4.6.1.9.4 Multicast Address Table Entry (Bit 40==1)

Table 13-135. Multicast Address Table Entry Bit Values

70:69	68:66	65	64	63:62	61:60	59:48	47:0
RESERVED	PORT_MASK	SUPER	IGNMBITS	FWDSTLVL	ENTRY_TYPE (1h)	RESERVED	MULTICAST_ADDRESS

Port Mask(2:0) (PORT_MASK)

This 3-bit field is the port bit mask that is returned with a found multicast destination address. There may be multiple bits set indicating that the multicast packet may be forwarded to multiple ports (but not the receiving port).

Supervisory Packet (SUPER)

When set, this field indicates that the packet with a matching multicast destination address is a supervisory packet.

0: Non-supervisory packet

1: Supervisory packet

Ignore Multicast Bits (IGNMBITS)

Indication that the Multicast Address has ignored bits.

Forward State Level (FWDSTLVL)

Indicates the port state(s) required for the received port on a destination address lookup in order for the multicast packet to be forwarded to the transmit port(s).

A transmit port must be in the Forwarding state in order to forward the packet. If the transmit port_mask has multiple set bits then each forward decision is independent of the other transmit port(s) forward decision.

0h = Forwarding

1h = Blocking/Forwarding/Learning

2h = Forwarding/Learning

3h = Forwarding

The forward state test returns a true value if both the RX and TX ports are in the required state.

Table Entry Type (ENTRY_TYPE)

Address entry type. Unicast or multicast determined by address bit 40.

01: Address entry. Unicast or multicast determined by address bit 40.

Packet Address (MULTICAST_ADDRESS)

This is the 48-bit packet MAC address. For an OUI address, only the upper 24-bits of the address are used in the source or destination address lookup. Otherwise, all 48-bits are used in the lookup.

13.2.1.4.6.1.9.5 VLAN/Unicast Address Table Entry (Bit 40 == 0)

Table 13-136. Unicast Address Table Entry Bit Values

70:69	68	67:66	65	64	63	62	61:60	59:48	47:0
RESERVED	TRUNK	PORT_NUMBER	BLOCK	SECURE	TOUCH	AGEABLE	ENTRY_TYPE (3h)	VLAN_ID	UNICAST_ADDRESS

Trunk Indicator (TRUNK)

0h = The port bits in the entry are the port number

1h = The port bits in the entry are the trunk number

Port Number (PORT_NUMBER)

This field indicates the port number (not port mask) that the packet with a unicast destination address may be forwarded to. Packets with unicast destination addresses are forwarded only to a single port (but not the receiving port).]

Block (BLOCK)

The block bit indicates that a packet with a matching source or destination address should be dropped (block the address).

0h = Address is not blocked.

1h = Drop a packet with a matching source or destination address (secure must be zero)

If block and secure are both set, then they no longer mean block and secure. When both are set, the block and secure bits indicate that the packet is a unicast supervisory (super) packet and they determine the unicast forward state test criteria. If both bits are set then the packet is forwarded if the receive port is in the Forwarding/ Blocking/Learning state. If both bits are not set then the packet is forwarded if the receive port is in the Forwarding state.

Secure (SECURE)

This bit indicates that a packet with a matching source address should be dropped if the received port number is not equal to the table entry PORT_NUMBER.

0h = Received port number is a don't care.

1h = Drop the packet if the received port is not the secure port for the source address and do not update the address (block must be zero)

Touch Indicator (TOUCH)

Only valid when AGEABLE is a 1h.

0h = Ageable unicast address has not been touched

1h = Ageable unicast address that has been touched

Ageable (AGEABLE)

This bit indicates that the address is ageable.

0h = Unicast address that is not ageable

1h = Unicast address that is ageable

Table Entry Type (ENTRY_TYPE)

Address entry. Unicast or multicast determined by address bit 40.

3h: VLAN address entry. Unicast or multicast determined by address bit 40.

VLAN ID (VLAN_ID)

The unique identifier for VLAN identification. This is the 12-bit VLAN ID.

Packet Address (UNICAST_ADDRESS)

This is the 48-bit packet MAC address. All 48-bits are used in the lookup.

13.2.1.4.6.1.9.6 VLAN/Multicast Address Table Entry (Bit 40==1)

Table 13-137. VLAN/Multicast Address Table Entry Bit Values

70:69	68:66	65	64	63:62	61:60	59:48	47:0
RESERVED	PORT_MASK	SUPER	IGNMBITS	FWDSTLVL	ENTRY_TYPE (11)	VLAN_ID	MULTICAST_ADDRESS

Port Mask(2:0) (PORT_MASK)

This 3-bit field is the port bit mask that is returned with a found multicast destination address. There may be multiple bits set indicating that the multicast packet may be forwarded to multiple ports (but not the receiving port).

Supervisory Packet (SUPER)

When set, this field indicates that the packet with a matching multicast destination address is a supervisory packet.

0: Non-supervisory packet

1: Supervisory packet

Ignore Multicast Bits (IGNMBITS)

Indication that the Multicast Address has ignored bits.

Forward State Level (FWDSTLVL)

Indicates the port state(s) required for the received port on a destination address lookup in order for the multicast packet to be forwarded to the transmit port(s).

A transmit port must be in the Forwarding state in order to forward the packet. If the transmit port_mask has multiple set bits then each forward decision is independent of the other transmit port(s) forward decision.

0h = Forwarding

1h = Blocking/Forwarding/Learning

2h = Forwarding/Learning

3h = Forwarding

The forward state test returns a true value if both the RX and TX ports are in the required state.

Table Entry Type (ENTRY_TYPE)

Address entry type. Unicast or multicast determined by address bit 40.

11: VLAN address entry. Unicast or multicast determined by address bit 40.

VLAN ID (VLAN_ID)

The unique identifier for VLAN identification. This is the 12-bit VLAN ID.

Packet Address (MULTICAST_ADDRESS)

This is the 48-bit packet MAC address. For an OUI address, only the upper 24-bits of the address are used in the source or destination address lookup. Otherwise, all 48-bits are used in the lookup.

13.2.1.4.6.1.9.7 Inner VLAN Table Entry

Table 13-138. Inner VLAN Table Entry

70:69	68:66	65	64:62	61:60	59:48	47	46:39	38:36
RESERVED	NO_LEARN_MASK	VLAN_FORCE_INGRESS_CHECK	0h	ENTRY_TYPE (1h)	VLAN_ID	NOFRAG	RESERVED	REG_MCAST_FLOOD_INDEX

35:27	26:24	23	22:15	14:12	11:3	2:0
RESERVED	FORCE_UNTAGGED_EGRESS	LMTNXTHDR	RESERVED	UREGMSK	RESERVED	VLAN_MEMBER_LIST

No Learn Mask (NO_LEARN_MASK)

When a bit is set in this mask, a packet with an unknown source address received on the associated port will not be learned (i.e. When a VLAN packet is received and the source address is not in the table, the source address will not be added to the table).

VLAN Force Ingress Check (VLAN_FORCE_INGRESS_CHECK)

If the receive port is not a member of this VLAN then the packet is dropped. This is similar to the ly_REG_Py_VID_INGRESS_CHECK bit in the CPSW_ly_ALE_PORTCTL0_y registers except this check is for this VLAN only (not all VLANs).

Table Entry Type (ENTRY_TYPE)

2h: VLAN entry

VLAN ID (VLAN_ID)

The unique identifier for VLAN identification. This is the 12-bit VLAN ID.

(NOFRAG)

VLAN No IPv4 Fragmented frames Control - Causes IPv4 fragmented IP frames to be dropped.

Registered Multicast Flood Index (REG_MCAST_FLOOD_INDEX)

This field indicates which port(s) are the registered multicast flood mask.

Force Untagged Packet Egress (FORCE_UNTAGGED_EGRESS)

This field causes the packet VLAN tag to be removed on egress for the specified port(s) (except on port 0).

VLAN Limit Next Header Control (LMTNXTHDR)

This bit causes frames to be dropped if the Protocol/Nxt Header does not match the CPSW_ALE_NXT_HDR register values.

VLAN Unregister Multicast Mask (UREGMSK)

This field indicates which port(s) are the unregistered multicast flood mask.

VLAN Member List (VLAN_MEMBER_LIST)

This field indicates which port(s) are members of the associated VLAN. One bit per port.

13.2.1.4.6.1.9.8 Outer VLAN Table Entry

Table 13-139. Outer VLAN Table Entry

70:69	68:66	65	64:62	61:60	59:48	47	46:39	38:36
RESERVED	NO_LEARN_MASK	VLAN_FORCE_INGRESS_CHECK	2h	ENTRY_TYPE (2h)	VLAN_ID	NOFRAG	RESERVED	REG_MCAST_FLOOD_INDEX

35:27	26:24	23	22:15	14:12	11:3	2:0
RESERVED	FORCE_UNTAGGED_EGRESS	LMTNXTHDR	RESERVED	UREGMSK	RESERVED	VLAN_MEMBER_LIST

No Learn Mask (NO_LEARN_MASK)

When a bit is set in this mask, a packet with an unknown source address received on the associated port will not be learned (i.e. When a VLAN packet is received and the source address is not in the table, the source address will not be added to the table).

VLAN Force Ingress Check (VLAN_FORCE_INGRESS_CHECK)

If the receive port is not a member of this VLAN then the packet is dropped. This is similar to the Iy_REG_Py_VID_INGRESS_CHECK bit in the CPSW_Iy_ALE_PORTCTL0_y registers except this check is for this VLAN only (not all VLANs).

Table Entry Type (ENTRY_TYPE)

2h: VLAN entry

VLAN ID (VLAN_ID)

The unique identifier for VLAN identification. This is the 12-bit VLAN ID.

(NOFRAG)

VLAN No IPv4 Fragmented frames Control - Causes IPv4 fragmented IP frames to be dropped.

Registered Multicast Flood Index (REG_MCAST_FLOOD_INDEX)

Index into CPSW_ALE_MSK_MUX0 to CPSW_Ix_ALE_MSK_MUXx register array that is used to create the registered multicast flood mask.

Force Untagged Packet Egress (FORCE_UNTAGGED_EGRESS)

This field causes the packet VLAN tag to be removed on egress for the specified port(s) (except on port 0).

VLAN Limit Next Header Control (LMTNXTHDR)

This bit causes frames to be dropped if the Protocol/Nxt Header does not match the CPSW_ALE_NXT_HDR register values.

VLAN Unregister Multicast Mask (UREGMSK)

This field indicates which port(s) are the unregistered multicast flood mask.

VLAN Member List (VLAN_MEMBER_LIST)

This field indicates which port(s) are members of the associated VLAN. One bit per port.

13.2.1.4.6.1.9.9 EtherType Table Entry

Table 13-140. EtherType Table Entry

70:65	64:62	61:60	59:16	15:0
RESERVED	4h	ENTRY_TYPE (2h)	RESERVED	ETHERTYPE

Table Entry Type (ENTRY_TYPE)

2h: VLAN entry

Ether Type (ETHERTYPE)

16-bits Ether Type field.

13.2.1.4.6.1.9.10 IPv4 Table Entry

Table 13-141. IPv4 Table Entry

70	69:65	64:62	61:60	59:32	31:0
RESERVED	IGNMBITS	6h	ENTRY_TYPE (2h)	RESERVED	IPV4ADR

Ignore Multicast Bits (IGNMBITS)

Indication that the Multicast Address has ignored bits.

Table Entry Type (ENTRY_TYPE)

2h: VLAN entry

IPv4 Address (IPV4ADR)

32-bit IPv4 Address. Any ignored bits must be zero value in the table entry.

13.2.1.4.6.1.9.11 IPv6 Table Entry High

Table 13-142. IPv6 Table Entry High

70:64	63	62	61:60	59:0
IGNMBITS	RESERVED	(1h)	ENTRY_TYPE (2h)	IPV6ADR[127:68]

Ignore Multicast Bits (IGNMBITS)

Indication that the Multicast Address has ignored bits.

Table Entry Type (ENTRY_TYPE)

2h: VLAN entry

IPv6 Address - upper 64 bits (IPV6ADR[127:68])

This address is split into three fields in the IPv6 High and IPv6 Low table entry. Any ignored bits must be zero in the table entry(s).

13.2.1.4.6.1.9.12 IPv6 Table Entry Low

Table 13-143. IPv6 Table Entry Low

70:63	62	61:60	59:0
IPV6ADR[67:60]	1h	ENTRY_TYPE (2h)	IPV6ADR[59:0]

Table Entry Type (ENTRY_TYPE)

2h: VLAN entry

IPv6 Address - upper 8 bits (IPV6ADR[67:60])

IPv6 Address - upper 60 bits (IPV6ADR[59:0])

This address is plit into three fields in the IPv6 High and IPv6 Low table entry. Any ignored bits must be zero in the table entry(s).

Note: IPV6 table address entries operate differently than all other table entry types. IPV6 table entries have a high entry concatenated with a low entry. The high entry must have an entry_pointer value of the low entry_pointer plus 0x40. Bit six of the entry_pointer must be set for the high entry and must be zero for the low entry.

13.2.1.4.6.1.10 Multicast Address

Multicast addresses are addresses with bit 40 set. Only destination addresses can be Multicast addresses. The group bit (bit 40) of the source address is reserved in the IEEE standard.

A multicast address of all ones is the broadcast address which can be added to the lookup table if forwarding of broadcast packets need be modified.

13.2.1.4.6.1.10.1 Multicast Ranges

Added IgnMbits to indicate at least one bit of the multicast address is ignored. Up to 10 bits of the multicast address can be ignored to provide the ability to create multiple multicast address ranges.

if ((IgnMbits)&(MultiCastAddress[0] ==0x000)) { MultiCastAddress[0] is ignored in compare}

if ((IgnMbits)&(MultiCastAddress[1:0]==0x001)) { MultiCastAddress[1:0] is ignored in compare}

if ((IgnMbits)&(MultiCastAddress[2:0]==0x003)) { MultiCastAddress[2:0] is ignored in compare}

if ((IgnMbits)&(MultiCastAddress[3:0]==0x007)) { MultiCastAddress[3:0] is ignored in compare}

if ((IgnMbits)&(MultiCastAddress[4:0]==0x00F)) { MultiCastAddress[4:0] is ignored in compare}

if ((IgnMbits)&(MultiCastAddress[5:0]==0x01F)) { MultiCastAddress[5:0] is ignored in compare}

if ((IgnMbits)&(MultiCastAddress[6:0]==0x03F)) { MultiCastAddress[6:0] is ignored in compare}

if ((IgnMbits)&(MultiCastAddress[7:0]==0x07F)) { MultiCastAddress[7:0] is ignored in compare}

if ((IgnMbits)&(MultiCastAddress[8:0]==0x0FF)) { MultiCastAddress[8:0] is ignored in compare}

if ((IgnMbits)&(MultiCastAddress[9:0]==0x1FF)) { MultiCastAddress[9:0] is ignored in compare}

Below is 'C' code to modify ALE MultiCastAddress and IgnMbits when iNumOfBitsToIgnore is greater than zero. Where fGenMask(iOffset,iBitsToMask) creates a Mask for the value provided. For example fGenMast(0,5) will return 0x1F.

```
if(iNumOfBitsToIgnore)
{
int iIgnClrMsk,iIgnSetMsk;
iIgnClrMsk=fGenMask(0, iNumOfBitsToIgnore);
```

```
iIgnSetMsk=fGenMask(0, iNumOfBitsToIgnore -1);
MultiCastAddress &= ~iIgnClrMsk;
MultiCastAddress |= iIgnSetMsk;
IgnMbits = 1;
}
else
{
IgnMbits = 0;
}
```

Multicast Addresses or Ranges can overlap, in the event of an overlap; the higher ALE index will be used.

13.2.1.4.6.1.11 Aging and Auto Aging

The ALE supports software control or automatic aging of agable addresses.

Any time an agable address is seen as a source address entering from a port the source address entry will be marked as touched.

If the aging timer expires or the software sets the [29] AGE_OUT_NOW bit in the CPSW_ALE_CONTROL, the aging process will be started.

The aging process will read each ALE entry and for all entries that are an address with or without VLAN that is also agable, the touch check process will be done.

The touch check process will test the TOUCH bit and if clear, the entry will be marked as free, else the TOUCH bit will be cleared.

What this means is that if the aging interval was programmed as one second, any unused entry could stay in the ALE table for 1.000001 to 1.999999 seconds

13.2.1.4.6.1.12 ALE Policing and Classification

The ALE has a number of configurable classifier engines (policers) that can be used for classification. Classification is a subset of the policing function and uses a policer without the color marking or rate limiting functions. A policer is a hardware engine that is used for policing. The POLCNTDIV8 field in the CPSW_ALE_STATUS register indicates the number of policers available to be used for classification. Each policer can be enabled to match on one or more of any of the below packet fields for classification. All but Port and Priority are index references to the ALE table entries.

- Port Number
- Priority extracted from VLAN, mapped from DSCP if enabled, or Default Port Priority
- Organization Network Unique identifier - ONU
- Destination Address - DA
- Source Address - SA
- Outer VLANID -S-VLANID
- Inner VLANID -C-VLANID
- Ether Type
- IP Source Address - IPSA with full CIDR masking for IPv4 and IPv6
- IP Destination Address - IPSA with full CIDR masking for IPv4 and IPv6
- Support Host Thread/Flow ID mapping based on any packet classification above

13.2.1.4.6.1.12.1 ALE Policing

The policing function on each policer engine is implemented as dual-counter three-color marking engine as described in the IETF RFC2698. The first counter is the Committed Information Rate (CIR) counter and the second counter is the Peak Information Rate (PIR) counter. The policing function can use either or both counters. Based on the counter values the packet color is determined. The color is used to determine whether

the packet is dropped or forwarded. The ALE has a local feature that can drop packets regardless of queue state.

The policing rates are determined by the below equations:

CIR policing rate in Mbit/s = ((ALE frequency in Mhz) * CPSW_ALE_POLICECFG7[31-0] CIR_IDLE_INC_VAL) / 32768

PIR policing rate in Mbit/s = ((ALE frequency in Mhz) * CPSW_ALE_POLICECFG6[31-0] PIR_IDLE_INC_VAL) / 32768

Each policer has 10 different match operations (see Section 13.2.1.4.6.1.12). Since multiple policing entries can be hit on a single packet this provides the ability to create precise traffic stream control.

Packets are colored at ALE lookup time. Packets can be colored RED, YELLOW, or GREEN. If multiple policers are configured for a packet stream then the packet color is merged from all matching (hit) policers. If any policer is RED then the packet is marked RED. Else if any policer is YELLOW then the packet is marked YELLOW. Otherwise the packet is marked GREEN.

The Policing engine supports several modes such that packets that don't hit a policing/classifier match can be treated as RED, YELLOW, GREEN or policer 0 color. Using policer 0 allows for a system to regulate unregulated traffic.

13.2.1.4.6.1.12.2 Classifier to Host Thread Mapping

The ALE module allows Host Thread mapping based on any packet classification. That is the ALE can generate a thread ID used by the host based on ALE classifier matches.

When enabled the highest classifier match can map to a particular thread ID value.

The ALE also supports an optional default Thread ID value in the event that no classifier match.

Each Thread ID, including the default thread ID, has an enable functionality such that, if no classifier matches occur the default value is used, if the default is not enabled, the switch will use the 6-bit {port[2:0], switch_priority[2:0]} value instead. If multiple classifier matches occur, the highest matching entry with a thread enable bit set will be used.

Three registers are used for ALE classification thread mapping configuration (CPSW_ALE_THREADMAPDEF, CPSW_ALE_THREADMAPCTL and CPSW_ALE_THREADMAPVAL). The three thread mapping registers are used independently and are separate from the other ALE policing registers. The CPSW_ALE_THREADMAPCTL register allows the CPSW_ALE_THREADMAPVAL register contents to be written to the selected classifier. There is a CPSW_ALE_THREADMAPDEF register that is used for all classifiers. The thread mapping registers can be written or changed at any time but any packets that are already processed will not have their thread altered.

13.2.1.4.6.1.12.3 ALE Classification

When the policers are configured as classifiers, the color marking and policing functions of the policing/classifier engines are not used. One or multiple classifiers can be configured to match on a single packet. For example, a classifier can be enabled to match on priority while another classifier could match IP address.

13.2.1.4.6.1.13 Mirroring

The ALE supports three mirroring modes: destination port, source port and or table entry.

Destination port mirroring allows packets from any ingress port or trunk which ends up switching to a particular egress destination port or trunk to be mirrored to yet another egress destination port or trunk. For example any traffic from any port that is switched to port 'A' can be also mirrored to port 'B'. (MIRROR_DP=A, MIRROR_DEN=1h, MIRROR_TOP=B in the CPSW_ALE_CONTROL register).

Source port mirroring allows packets received on any enabled ingress source port or trunk to be switched to the mirror egress port as well as the actual egress destination ports. For example traffic received on ingress port 'A' can be switched to egress port 'B' as well as the intended egress destination port.(Iy_REG_Py_MIRROR_SP=1h in the CPSW_Iy_ALE_PORTCTL0_y register, MIRROR_SEN=1h, MIRROR_TOP=B in the CPSW_ALE_CONTROL register).

Table entry mirroring allows for any MAC Address, MAC Address with VLAN, ONU Address or VLAN entry that matches on ingress to be switched to the egress destination as well as the actual egress destination. For example all traffic for VLAN ID of 35 can be mirrored to port 'B'. That is any traffic switched on VLAN ID of 35 will be mirrored. ({VLAN ID of 35 in ALE Table entry index=C}, MIRROR_MIDX=C, MIRROR_MEN=1h, MIRROR_TOP=B)

In the event that mirrored packets are mirrored to or from a port that is also the mirror port the packet will not be duplicated or marked as a mirror packet since the packet has already been on the port as ingress or egress. The packet sent to the mirror port may have modified VLAN info based on the port and VLAN lookup table entries. The mirror port need not be a member of the VLAN ID it is mirroring, the ALE will forward traffic to the mirror port after ingress and egress filters are applied.

The switch may decide to drop any mirror traffic based on switch buffer thresholds as to prevent required traffic from becoming congested.

Port mirroring is controlled by register fields in CPSW_ALE_CONTROL, CPSW_ALE_CTRL2 and the port control registers.

- MIRROR_DP - The destination port that will have its traffic mirrored (CPSW_ALE_CONTROL register).
- MIRROR_TOP - The port to which mirrored traffic is sent (CPSW_ALE_CONTROL register).
- MIRROR_MEN - The enable for mirroring traffic that matches a supported lookup table entry (CPSW_ALE_CONTROL register).
- MIRROR_DEN - The Enable for destination port mirroring (CPSW_ALE_CONTROL register).
- MIRROR_SEN - The Enable for source port mirroring (CPSW_ALE_CONTROL register).
- MIRROR_MIDX - The index of a lookup table entry that will be mirrored (CPSW_ALE_CTRL2 register).
- Iy_REG_Py_MIRROR_SP - The enable for the Source port to be mirrored. Although multiple source ports can be mirrored concurrently, a mirror traffic bandwidth issue may occur on the mirror egress port (CPSW_Iy_ALE_PORTCTL0_y register).

13.2.1.4.6.1.14 Trunking

The ALE supports port trunking of any port in any of four trunk groups. That is, four trunk groups can be supported with up to eight ports in each trunk group. There are no port adjacency rules for trunk groups. When ports are a member of a trunk group, addresses added and used in the lookup table will refer to the trunk group rather than port as indicated in the lookup table entries. If ports are removed from a trunk group, the ALE will redistribute the traffic based on the crc polynomial of enabled fields and the remaining ports within the trunk group. A trunk group may contain only one port. Packet priority, DA, SA, C-VLAN ID, IPv4SA, IPv4DA, IPv6SA, and/or IPv6DA can be used in the hash to generate destination port within the trunk group. If all hash enables are disabled, the packet can be directed to a particular port within the trunk group which allows for testing paths etc. A host directed frame is directed to the directed port regardless of trunk group settings.

Trunking is controlled through fields in the CPSW_ALE_CTRL2 register and in each ALE CPSW_Iy_ALE_PORTCTL0_y register:

- TRK_EN_DST - Enable destination address hashing for trunk port calculation.
- TRK_EN_SRC - Enable source address hashing for trunk port calculation.
- TRK_EN_PRI - Enable priority hashing for trunk port calculation.
- TRK_EN_IVLAN - Enable inner C-VLAN ID hashing for trunk port calculation.
- TRK_EN_SIP - Enable source IP address hashing for trunk port calculation.
- TRK_EN_DIP - Enable destination IP address hashing for trunk port calculation.
- TRK_BASE - Hashing formula starting value and test port offset.
- Iy_REG_Py_TRUNKEN - Enable this port as a trunk group
- Iy_REG_Py_TRUNKNUM - Trunk group number defines this port as a member of a particular trunk group.

13.2.1.4.6.1.15 DSCP

The ALE can map DSCP field to priority prior to classification matching. When enabled the DSCP is mapped via 64 priority entries such that any DSCP value can be mapped to any of the eight priorities. When a packet is received without a VLAN priority this remapped priority can be used instead of the default Port VLAN priority field. See CPSW_P0_RX_DSCP_MAP_REG_y and CPSW_PN_RX_DSCP_MAP_REG_y registers in the Register Manual section for DSCP mapping.

13.2.1.4.6.1.16 Packet Forwarding Processes

There are three processes that an incoming received packet may go through to determine packet forwarding. The processes are *Ingress Filtering*, *VLAN Lookup*, and *Egress.*

Packet processing begins in the Ingress Filtering process. Each port has an associated packet forwarding state that can be one of four values (Disabled, Blocked, Learning, or Forwarding). The default state for all ports is Disabled. The host sets the packet forwarding state for each port.

In the packet ingress process (receive packet process), there is a forward state test for unicast destination addresses and a forward state test for multicast addresses. The multicast forward state test indicates the port states required for the receiving port in order for the multicast packet to be forwarded to the transmit port(s). A transmit port must be in the Forwarding state for the packet to be forwarded for transmission. The MCAST_FWD_STATE indicates the required port state for the receiving port as indicated in the preceding table. The unicast forward state test indicates the port state required for the receiving port in order to forward the unicast packet. The transmit port must be in the Forwarding state in order to forward the packet. The BLOCK and SECURE bits determine the unicast forward state test criteria. If both bits are set then the packet is forwarded if the receive port is in the Forwarding/Blocking/Learning state. If both bits are not set then the packet is forwarded if the receive port is in the Forwarding state. The transmit port must be in the Forwarding state regardless. The forward state test used in the ingress process is determined by the destination address packet type (multicast/unicast).

In general, packets received with errors are dropped by the address lookup engine without learning, updating, or touching the address. The error condition and the abort are indicated by the Ethernet port to the ALE. Packets with errors may be passed to the host (not aborted) by an ingress port, if the switch port setting has the RX_CMF_EN, RX_CEF_EN, or RX_CSF_EN bit(s) set in the CPSW_PN_MAC_CONTROL_REG register. Error packets that are passed to the host by the Ethernet port are considered to be bypass packets by the ALE and are sent only to the host. Error packets do not learn, update, or touch addresses regardless of whether they are aborted or sent to the host. Packets with long or short errors received by the host are dropped. Packets with errors received by the host are forwarded as normal.

The following control bits are in the CPSW_PN_MAC_CONTROL_REG register:

- [22] RX_CEF_EN - enables frames that are fragments, long, jabber, CRC, code, and alignment errors to be forwarded
- [23] RX_CSF_EN - enables short frames to be forwarded
- [24] RX_CMF_EN - enables MAC control frames to be forwarded.

13.2.1.4.6.1.16.1 Ingress Filtering Process

Condition and action
If ((ALE BYPASS) and (host port is not the receive port)) then use host portmask and go to Egress process
if (directed packet) then use directed port number and go to Egress process
If (Rx Iy_REG_Py_PORTSTATE is Disabled) then discard the packet
if ((error packet) and (host port is not the receive port)) then use host portmask and go to Egress process
if (((BLOCK) and (unicast source address found)) or ((BLOCK) and (unicast destination address found))) then discard the packet
if ((ENABLE_RATE_LIMIT) and (rate limit exceeded) and (not RATE_LIMIT_TX) then if (((Multicast/Broadcast destination address found) and (not SUPER)) or (Multicast/Broadcast destination address not found)) then discard the packet

if ((not forward state test valid) and (destination address found)) then discard the packet to any port not meeting the requirements • Unicast destination addresses use the unicast forward state test and multicast destination addresses use the multicast forward state test.
if ((destination address not found) and ((not transmit port forwarding) or (not receive port forwarding))) then discard the packet to any ports not meeting the above requirements
if (source address found) and (secure) and (not block) and (receive port number != port_number)) then discard the packet
if ((not super) and (drop_untagged) and ((non-tagged packet) or ((priority tagged) and not(en_vid0_mode))) then discard the packet

If (VLAN_Unaware)
CPSW_ALE_UVLAN_UNTAG = “000000”
CPSW_ALE_UVLAN_URCAST = “111111”
CPSW_ALE_UVLAN_URCAST = “111111”
UVLAN_MEMBER_LIST = “111111”
else if (VLAN not found)
CPSW_ALE_UVLAN_UNTAG = CPSW_ALE_UVLAN_UNTAG
CPSW_ALE_UVLAN_RMCAST = CPSW_ALE_UVLAN_RMCAST
CPSW_ALE_UVLAN_RMCAST = CPSW_ALE_UVLAN_RMCAST
CPSW_ALE_UVLAN_MEMBER = CPSW_ALE_UVLAN_MEMBER
else
CPSW_ALE_UVLAN_UNTAG = found CPSW_ALE_UVLAN_UNTAG
CPSW_ALE_UVLAN_URCAST = found CPSW_ALE_UVLAN_URCAST
CPSW_ALE_UVLAN_RMCAST = found CPSW_ALE_UVLAN_RMCAST
UVLAN_MEMBER_LIST = found UVLAN_MEMBER_LIST

if ((not SUPER) and (Iy_REG_Py_VID_INGRESS_CHECK) and (Rx port is not VLAN member)) then discard the packet
if ((ENABLE_AUTH_MODE) and (source address not found) and not(destination address found and (SUPER))) then discard the packet
if (destination address equals source address) then discard the packet
goto VLAN_Lookup process

13.2.1.4.6.1.16.2 VLAN Lookup Process

Condition and action
if ((unicast packet) and (destination address found with or without VLAN) and (not SUPER) then portmask is the logical "AND" of the PORT_NUMBER and UVLAN_MEMBER_LIST and goto Egress process
if ((unicast packet) and (destination address found with or without VLAN) and (SUPER)) then portmask is the PORT_NUMBER and goto Egress process

if ((Unicast packet) and (destination address not found)) then portmask is UVLAN_MEMBER_LIST less host port (if UNI_FLOOD_TO_HOST is not set) and goto Egress process
if ((Multicast packet) and (destination address found with or without VLAN) and (not SUPER)) then portmask is the logical “AND” of CPSW_ALE_UVLAN_URCAST and found destination address/VLAN portmask (PORT_MASK) and UVLAN_MEMBER_LIST and goto Egress process
if ((Multicast packet) and (destination address found with or without VLAN) and (SUPER)) then portmask is the PORT_MASK and goto Egress process
if ((Multicast packet) and (destination address not found)) then portmask is the logical “AND” of CPSW_ALE_UVLAN_URCAST and UVLAN_MEMBER_LIST then goto Egress process
if (Broadcast packet) then use found UVLAN_MEMBER_LIST and goto Egress process

Note

The UVLAN_MEMBER_LIST, UVLAN_UNREG_MCAST_FLOOD_MASK, UVLAN_REG_MCAST_FLOOD_MASK and UVLAN_FORCE_UNTAGGED_EGRESS are set in the Section 13.2.1.4.6.1.16.1 *Ingress Filtering Process*, based on VLAN_Unaware, Unknown_VLAN rules and VLAN table entries.

13.2.1.4.6.1.16.3 Egress Process

Condition and action
Clear Rx port from portmask (don’t send packet to Rx port).
Clear disabled ports from portmask.
if ((ENABLE_OUI_DENY) and (OUI source address not found) and (not ALE_BYPASS) and (not error packet) and (not ((destination address) and (SUPER)))) then Clear host port from portmask
if ((not ENABLE_OUI_DENY) and (OUI source address found) and (not ALE BYPASS) and (not error packet) and not ((destination address) and (SUPER)))) then Clear host port from portmask
if ((ENABLE_RATE_LIMIT) and (RATE_LIMIT_TX)) then if (not SUPER) and (rate limit exceeded on any tx port) then clear rate limited tx port from portmask If address not found then SUPER cannot be set.
If portmask is zero then discard packet
Send packet to portmask ports

13.2.1.4.6.1.16.4 Learning/Updating/Touching Processes

The learning, updating, and touching processes are applied to each receive packet that is not aborted. The processes are concurrent with the packet forwarding process. In addition to the following, a packet must be received without error in order to learn/update/touch an address.

13.2.1.4.6.1.16.4.1 Learning Process

The learning process is applied to each receive packet that is not aborted. The learning process is a concurrent process with the packet forwarding process.

Condition and action
If (directed) then do not learn, update, or set touched else continue
If (not (Learning or Forwarding) or (ENABLE_AUTH_MODE) or (packet error) or (Iy_REG_Py_NO_LEARN)) then do not learn address
if ((Non-tagged packet) and (Iy_REG_Py_DROP_UN_TAGGED)) then do not learn address
if ((VLAN_AWARE) and (VLAN not found) and (unknown UVLAN_MEMBER_LIST = “000”)) then do not learn address
if ((Iy_REG_Py_VID_INGRESS_CHECK) and (Rx port is not VLAN member) and (VLAN found)) then do not learn address
if ((source address found) and (receive port_number != PORT_NUMBER) and (SECURE or BLOCK)) then do not update address else continue
if ((source address found) and (receive port number != PORT_NUMBER)) then update address else continue
if ((source address not found) and (VLAN_AWARE) and not (LEARN_NO_VLANID)) then learn address with VLAN
if ((source address not found) and ((not VLAN_AWARE) or (VLAN_AWARE and LEARN_NO_VLANID))) then learn address without VLAN

13.2.1.4.6.1.16.4.2 Updating Process

Condition and action
if (directed) then do not update address
If (not(Learning or Forwarding) or (ENABLE_AUTH_MODE) or (packet error) or (Iy_REG_Py_NO_LEARN)) then do not update address
if ((Non-tagged packet) and (Iy_REG_Py_DROP_UN_TAGGED)) then do not update address
if ((VLAN_AWARE) and (VLAN not found) and (unknown UVLAN_MEMBER_LIST = “000”)) then do not update address
if ((Iy_REG_Py_VID_INGRESS_CHECK) and (Rx port is not VLAN member) and (VLAN found)) then do not update address
if ((source address found) and (receive port number != PORT_NUMBER) and (SECURE or BLOCK)) then do not update address
if ((source address found) and (receive port number != PORT_NUMBER)) then update address

13.2.1.4.6.1.16.4.3 Touching Process

if ((source address found) and (ageable) and (not touched)) then set touched

13.2.1.4.6.1.17 VLAN Aware Mode

The CPSW is in VLAN aware mode when the VLAN_AWARE bit is set in the CPSW_CONTROL_REG register.

In VLAN aware mode, transmitted packet data is changed depending on the packet type (PKT_TYPE), packet priority (PKT_PRI), and VLAN information.

The VLAN_LTYPE_SEL value is selected by the S_CN_SWITCH bit in the CPSW_CONTROL_REG register and is either the VLAN_LTYPE_INNER (8100h default) or VLAN_LTYPE_OUTER (88A8h default) value.

13.2.1.4.6.1.18 VLAN Unaware Mode

An egress port is operating in the VLAN unaware mode when the VLAN_AWARE bit in the CPSW_CONTROL_REG register is cleared to 0h. In VLAN unaware mode, transmit (egress) packets are not modified on egress.

13.2.1.4.6.2 Packet Priority Handling

There are three priorities that are used inside the CPSW: **packet priority**, **header packet priority**, and **switch priority**. The **packet priority** is the determined priority for the ingress packet. The **header packet priority** is used as the outgoing VLAN priority if the packet is egressing from the switch with a VLAN tag. The **switch priority** determines which of the 8 FIFO priority queues the packet uses during egress.

The VLAN_LTYPE_SEL value below is selected by the S_CN_SWITCH bit in the CPSW Control register and is either the VLAN_LTYPE_INNER (0x8100 default) or the VLAN_LTYPE_OUTER (0x88A8 default) value.

Packets are received on two types of ports (Ethernet and CPDMA host port). Received packets have a received packet priority (0 to 7, with 7 being the highest priority).

13.2.1.4.6.2.1 Ethernet Port Receive

The received packet priority for Ethernet receive packets is determined as follows:

1. If the first packet LTYPE = VLAN_LTYPE_SEL then the received packet priority is the packet priority (VLAN tagged and priority tagged packets).
2. Else if the first packet LTYPE = 0x0800 and byte 14 (following the LTYPE) is equal to 0x4X, and DSCP_IPV4_EN is set in CPSW_PN_CONTROL_REG, then the received packet priority is the 6-bit TOS field in byte 15 (upper 6 bits) mapped through the port's DSCP priority mapping registers (IPv4 packet).
3. Else if the first packet LTYPE = 0x86DD and the most significant nibble of byte 14 (following the LTYPE) is equal to 0x6, and DSCP_IPV6_EN is set in CPSW_PN_CONTROL_REG, then the received packet priority is the 6-bit priority (in the 6-bits following the upper nibble 0x6) mapped through the port's DSCP priority mapping registers (IPv6 packet).
4. Else the received packet priority is the source (ingress) port priority taken from the port's ENET_PN_PORT_VLAN register.

The packet priority is mapped through the receive ports associated packet-priority-to-header-packet-priority-mapping register (CPSW_PN_RX_PRI_MAP_REG) to obtain the header packet priority. The header packet priority is then used as the actual transmit packet priority if the VLAN information is to be sent on egress. The header packet priority is mapped at each destination FIFO through the CPSW_PN_TX_PRI_MAP_REG register (header priority to switch priority mapping register) to obtain the hardware switch priority (hardware queue 0 through 7).

13.2.1.4.6.2.2 CPDMA Port Receive

The received packet priority for CPDMA host port receive packets is determined as follows:

1. If the first packet LTYPE = VLAN_LTYPE_SEL then the received packet priority is the packet priority (VLAN tagged and priority tagged packets).
2. Else if the first packet LTYPE = 0x0800 and byte 14 (following the LTYPE) is equal to 0x4X, and DSCP_IPV4_EN is set in CPSW_P0_CONTROL_REG, then the received packet priority is the 6-bit TOS field in byte 15 (upper 6 bits) mapped through the port's DSCP priority mapping registers (IPv4 packet).
3. Else if the first packet LTYPE = 0x86DD and the most significant nibble of byte 14 (following the LTYPE) is equal to 0x6, and DSCP_IPV6_EN is set in CPSW_P0_CONTROL_REG, then the received packet priority is the 6-bit priority (in the 6-bits following the upper nibble 0x6) mapped through the port's DSCP priority mapping registers (IPv6 packet).

4. Else the received packet priority is the source (ingress) port priority taken from the port's P0_PORT_VLAN register.

The packet priority is mapped through the receive ports associated packet-priority-to-header-packet-priority-mapping register (CPSW_P0_RX_PRI_MAP_REG) to obtain the header packet priority. The header packet priority is then used as the actual transmit packet priority if the VLAN information is to be sent on egress.

For CPDMA host port receive packets, the destination port hardware switch priority is the below selected value remapped through CPSW_P0_RX_PRI_MAP_REG:

1. If the receive packet is priority or VLAN tagged:
   a. If CPSW_P0_RX_REMAP_VLAN_REG is clear then the destination hardware switch priority is the host receive channel number.
   b. If CPSW_P0_RX_REMAP_VLAN_REG is set then the destination hardware switch priority is the packet priority value. Port transmit remapping (ENET_PN_TX_PRI_MAP should remain the default value) is not compatible with this bit being set, but remapping can be configured on port 0 receive.
2. Else if the receive packet has the first packet LTYPE = 0x0800 and byte 14 (following the LTYPE) is equal to 0x4X, and DSCP_IPV4_EN is set in CPSW_P0_CONTROL_REG:
   a. If P0_RX_REMAP_DSCP_IPV4 is clear then the destination hardware switch priority is the host receive priority.
   b. If P0_RX_REMAP_DSCP_IPV4 is set then the destination hardware switch priority is the 6-bit TOS field in byte 15 (upper 6 bits) mapped through the port's DSCP priority mapping registers (IPV4 packet). Port transmit remapping (ENET_PN_TX_PRI_MAP should remain the default value) is not compatible with this bit being set, but remapping can be configured on port 0 receive.
3. Else if the receive packet has the first packet LTYPE = 0x86DD and the most significant nibble of byte 14 (following the LTYPE) is equal to 0x6, and DSCP_IPV6_EN is set in CPSW_P0_CONTROL_REG:
   a. If P0_RX_REMAP_DSCP_IPV6 is clear then the destination hardware switch priority is the host receive priority.
   b. If P0_RX_REMAP_DSCP_IPV6 is set then the destination hardware switch priority is the 6-bit priority (in the 6 bits following the upper nibble 0x6) mapped through the port's DSCP priority mapping registers (IPV6 packet). Port transmit remapping (ENET_PN_TX_PRI_MAP should remain the default value) is not compatible with this bit being set, but remapping can be configured on port 0 receive.
4. Else the receive packet is non-tagged and the destination hardware switch priority is the host receive channel number.

13.2.1.4.6.2.3 CPDMA Port Transmit

If the TH_CH_OVERRIDE bit in the CPDMA Control register is clear, then the CPDMA packet transmit channel number is the port 0 hardware **switch priority**. If TH_CH_OVERRIDE is set, then for packets with a classification match the transmit channel number is the lower three bits of the 6-bit address lookup engine classification match value (THREADVAL[2:0] in ALE register THREADMAPVAL). The FLOW value in the VLAN encapsulation word is all 6 bits of the THREADVAL for classifier matches regardless of the setting of TH_CH_OVERRIDE if the encapsulation word is transferred.

13.2.1.4.6.2.4 Priority Mapping and Transmit VLAN Priority

Figure 13-91 below, as well as the corresponding explanation that follows, explains each of the priorities, how they are determined, and how they are used. A number in parentheses in the figure indicates a process (Ethernet port ingress, host port egress, etc.). Each bullet in the text following the diagram explains one of the 5 processes pointed out in the figure.

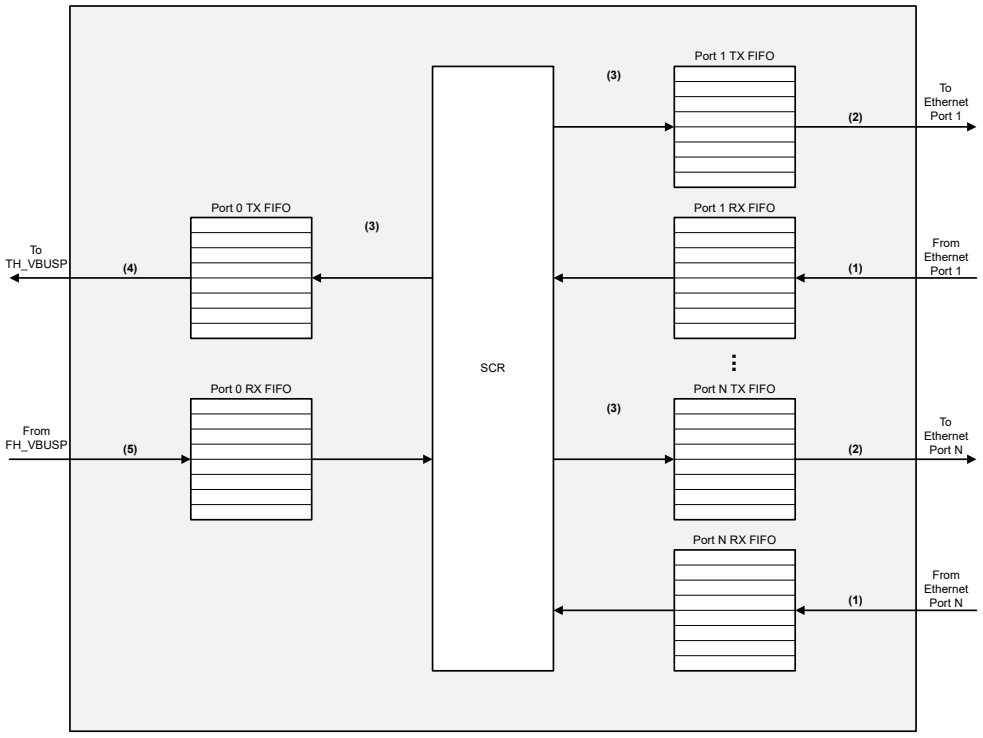


Figure 13-91. Gigabit Ethernet Switch Priority Mapping and Transmit VLAN Processing

From Figure 13-91 above:

- (1) is the ingress process that occurs at the external Ethernet ports
  - The incoming packet is assigned a **packet priority** based on either its VLAN priority, IPv4 or IPv6 DSCP value, or the ingress port's priority. This **packet priority** is then mapped to a **header packet priority** using the CPSW_PN_RX_PRI_MAP_REG register where N is the port where the packet entered the switch. This process is explained in further detail in Section 13.2.1.4.6.2.
- (2) is the egress process that occurs at the external Ethernet ports
  - If the switch is in VLAN Aware mode then the VLAN header may be added, replaced, or removed during the egress process. If the VLAN header is to be added or replaced, the VLAN priority will come from the **header packet priority** that was determined in process (1) or (5). Transmit VLAN processing is the same for both the host port and the external Ethernet ports and is described in Section 13.2.1.4.6.4.1.
- (3) is the process by which it is decided which priority TX queue to place the packet on in the Port N TX FIFO during egress
  - Each Port's TX FIFO has 8 queues that each correspond to a priority that is used when determining which packet will egress from the switch next at that port. The **header packet priority** (Ethernet port ingress, process (1)) or the receive packet channel (host port 0 ingress, process (5)) gets mapped through the CPSW_PN_RX_PRI_MAP_REG register (where N is the egress port number) to determine the **switch priority** of the packet. The **switch priority** determines which TX FIFO queue to place the packet in. The FIFO architecture is described in Section 13.2.1.4.6.9.5. The header packet priority to switch priority mapping is discussed in Section 13.2.1.4.6.2.
- (4) is the egress process that occurs at CPDMA Host Port 0
  - The egress process for CPDMA Host Port 0 is discussed in Section 13.2.1.4.6.2.3.
  - If the switch is in VLAN Aware mode then the VLAN header may be added, replaced, or removed during the egress process. If the VLAN header is to be added or replaced, the VLAN priority will come from the **header packet priority** that was determined in process (1). Transmit VLAN processing is the same for both the host port and the external Ethernet ports and is described in Section 13.2.1.4.6.4.1.
- (5) is the ingress process that occurs at CPDMA Host Port 0
  - The incoming packet is assigned a **packet priority** based on either its VLAN priority, IPv4 or IPv6 DSCP value, or the host port's priority. This packet priority is then mapped to a **header packet priority** using the CPSW_P0_RX_PRI_MAP_REG register.
  - The process to determine the destination hardware **switch priority** is discussed in Section 13.2.1.4.6.2.2.

13.2.1.4.6.3 CPPI Port Ingress

Packets received on the CPDMA host port have a received packet priority (0 to 7 with 7 being the highest priority).

The received packet priority is determined as follows:

1. If the first packet LTYPE = VLAN_LTYPE_SEL then the received packet priority is the packet priority (VLAN tagged and priority tagged packets).
2. Else if the first packet LTYPE = 0x0800 and byte 14 (following the LTYPE) is equal to 0x4X, and DSCP_IPV4_EN is set in CPSW_P0_CONTROL_REG or CPSW_PN_CONTROL_REG register, then the received packet priority is the 6-bit TOS field in byte 15 (upper 6 bits) mapped through the port's DSCP priority mapping registers (IPv4 packet).
3. Else if the first packet LTYPE = 0x86DD and the most significant nibble of byte 14 (following the LTYPE) is equal to 0x6, and DSCP_IPV6_EN is set in CPSW_P0_CONTROL_REG or CPSW_PN_CONTROL_REG register, then the received packet priority is the 6-bit priority (in the 6-bits following the upper nibble 0x6) mapped through the port's DSCP priority mapping registers (IPv6 packet).
4. Else the received packet priority is the source (ingress) port priority

For CPPI ingress packets, the destination port hardware switch priority is the below selected value remapped through CPSW_PN_RX_PRI_MAP_REG:

1. If the ingress packet is priority tagged or vlan tagged:
   - If RX_REMAP_VLAN in CPSW_P0_CONTROL_REG register is clear then the destination hardware switch priority is the CPPI receive channel number.

- If RX_REMAP_VLAN in CPSW_P0_CONTROL_REG register is set then the destination hardware switch priority is the packet priority value. Port transmit remapping (CPSW_PN_TX_PRI_MAP_REG should remain the default value) is not compatible with this bit being set, but remapping can be configured on port 0 receive.

2. Else if the ingress packet has the first packet LTYPE = 0x0800 and byte 14 (following the LTYPE) is equal to 0x4X, and DSCP_IPV4_EN is set in CPSW_P0_CONTROL_REG register:
   - If RX_REMAP_DSCP_V4 bit in CPSW_P0_CONTROL_REG register is clear then the destination hardware switch priority is the CPPI receive channel number.
   - If RX_REMAP_DSCP_V4 bit in CPSW_P0_CONTROL_REG register is set then the destination hardware switch priority is the 6-bit TOS field in byte 15 (upper 6-bits) mapped through the port's DSCP priority mapping registers (IPV4 packet). Port 1 transmit remapping (CPSW_PN_TX_PRI_MAP_REG should remain the default value) is not compatible with this bit being set, but remapping can be configured on port 0 receive.
3. Else if the ingress packet has the first packet LTYPE = 0x86DD and the most significant nibble of byte 14 (following the LTYPE) is equal to 0x6, and DSCP_IPV6_EN is set in P0_CONTROL_REG register:
   - If RX_REMAP_DSCP_V6 bit in CPSW_P0_CONTROL_REG register is clear then the destination hardware switch priority is the CPPI receive channel number.
   - If RX_REMAP_DSCP_V6 bit in CPSW_P0_CONTROL_REG register is set then the destination hardware switch priority is the 6-bit priority (in the 6-bits following the upper nibble 0x6) mapped through the port's DSCP priority mapping registers (IPV6 packet). Port 1 transmit remapping (CPSW_PN_TX_PRI_MAP_REG should remain the default value) is not compatible with this bit being set, but remapping can be configured on port 0 receive.
4. Else the ingress packet is non-tagged and the destination hardware switch priority is the CPPI receive channel number.

13.2.1.4.6.4 Packet CRC Handling

The P0_TX_CRC_REMOVE bit in the CPSW_CONTROL_REG register determines if host port egress packets have CRC included or not. If P0_TX_CRC_REMOVE is set to 1h then all packets that are transmitted from port 0 do not contain CRC. If P0_TX_CRC_REMOVE bit is cleared to 0h then all packets that are transmitted from port 0 contain CRC. The CRC type, if present, is determined by the CRC_TYPE bit in the CPSW_PN_MAC_CONTROL_REG register. If the CRC_TYPE bit is cleared to 0h then the CRC present in each packet after host port egress is Ethernet CRC. If the CRC_TYPE bit is set to 1h then the CRC present in each packet after host port egress is Castagnoli CRC.

Note

The CRC type present in the packet after host port egress is determined solely by the CRC_TYPE bit in the CPSW_PN_MAC_CONTROL_REG register regardless of the CRC type present in the packet during Ethernet port ingress.

13.2.1.4.6.4.1 Transmit VLAN Processing

Transmit packets are NOT modified during switch egress when the VLAN_AWARE bit in the CPSW_CONTROL_REG register is cleared to 0h. This means that the switch is not in VLAN-aware mode.

The next three sections cover transmit processing when the switch is in VLAN-aware mode for different packet types. The Gigagibit Ethernet switch is in VLAN-aware mode when the VLAN_AWARE bit is set in the CPSW_CONTROL_REG register. While in VLAN-aware mode, VLAN is added, removed, or replaced according to the type of packet as well as the CPSW_ALE_UVLAN_UNTAG[1-0] UVLAN_FORCE_UNTAGGED_EGRESS bit in the packet header as explained below.

13.2.1.4.6.4.1.1 Untagged Packets (No VLAN or Priority Tag Header)

Untagged packets are all packets that are not a VLAN packet or a priority tagged packet. According to the CPSW_ALE_UVLAN_UNTAG[1-0] UVLAN_FORCE_UNTAGGED_EGRESS bit in the packet header the packet may exit the switch with a VLAN tag inserted or the packet may leave the switch unchanged. The two cases are discussed below.

- Insert VLAN Case:

Untagged input packets have the header packet VLAN inserted when the CPSW_ALE_UVLAN_UNTAG[1-0] UVLAN_FORCE_UNTAGGED_EGRESS bit in the transmit packet header is de-asserted. For untagged packets, the VLAN EtherType = 0x8100 is inserted after the source address followed by the two byte header packet VLAN. The header packet VLAN is composed of the header packet priority along with the PORT_CFI and PORT_VID values from the CPSW_PN_PORT_VLAN_REG register (where N is the port that the untagged packet entered the switch) through. The packet length/type field is output four bytes later than it is input and is not removed or replaced.

- No Change Case:

  Untagged input packets are output unchanged when the CPSW_ALE_UVLAN_UNTAG[1-0] UVLAN_FORCE_UNTAGGED_EGRESS transmit packet header bit is asserted.

13.2.1.4.6.4.1.2 Priority Tagged Packets (VLAN VID == 0 && EN_VID0_MODE ==0h)

Priority tagged packets are packets that contain a VLAN header with VID = 0. According to the CPSW_ALE_UVLAN_UNTAG[1-0] UVLAN_FORCE_UNTAGGED_EGRESS bit in the packet header, priority tagged packets may exit the switch with their VLAN ID and priority replaced or they may have their priority tag completely removed. The two cases are discussed below.

Note

In order for a priority tagged packet to fall into this category the ENABLE_VID0_MODE bit in the CPSW_ALE_CONTROL register must also be set to 0h. If the ENABLE_VID0_MODE bit in the CPSW_ALE_CONTROL register is set to 1h, then packets with a VLAN VID of 0 will fall into the VLAN Tagged Packets category in Section 13.2.1.4.6.4.1.3 below.

- Replace Priority and VLAN ID Case:

  Priority tagged input packets have the packet VLAN ID and the packet priority replaced with the header packet VLAN ID and the header packet priority when the transmit packet header CPSW_FORCE_UNTAGGED_EGRESS_REG[1-0] MASK bit is de-asserted. The header packet VLAN ID comes from the PORT_VID bits in the CPSW_PN_PORT_VLAN_REG register (where N is the port where the packet entered the switch). The header packet priority is based on the packet priority to header packet priority mapping in the CPSW_PN_RX_PRI_MAP_REG register (where N is the port where the packet entered the switch).

- Remove VLAN Header Case:

  Priority tagged input packets have the 4-byte packet VLAN information removed when the transmit packet header CPSW_ALE_UVLAN_UNTAG[1-0] UVLAN_FORCE_UNTAGGED_EGRESS bit is asserted. The 0x8100 EtherType is removed as is the two byte packet VLAN. Input 64-67 byte priority tagged packets go out with the VLAN removed and padded to 64-bytes if the PASS_CRC input bit is asserted. The input CRC bytes are used as the pad data. Input 64-byte priority-tagged packets use all four input CRC bytes as pad, input 65-byte priority-tagged packets use three of the input CRC bytes as pad, and so on. No pad is performed if the PASS_CRC input bit is not asserted - input 64-67 byte (on the wire) priority-tagged packets go out as 60-63 byte packets. The output CRC is replaced with a generated CRC when the VLAN is removed.

13.2.1.4.6.4.1.3 VLAN Tagged Packets (VLAN VID != 0 || (EN_VID0_MODE ==1h && VLAN VID ==0))

VLAN tagged packets are packets that contain a VLAN header specifying the VLAN the packet belongs to (VID), the packet priority (PRI), and the drop eligibility indicator (CFI). According to the CPSW_ALE_UVLAN_UNTAG[1-0] UVLAN_FORCE_UNTAGGED_EGRESS bit in the packet header, VLAN tagged packets may exit the switch with their VLAN priority replaced or they may have their VLAN header completely removed. The two cases are discussed below.

- Replace Priority Case:

  VLAN tagged input packets are output with the packet priority replaced with the header packet priority when the transmit packet header CPSW_ALE_UVLAN_UNTAG[1-0] UVLAN_FORCE_UNTAGGED_EGRESS bit is deasserted.

- Remove VLAN Header Case:

  VLAN tagged input packets have the 4-byte packet VLAN information removed when the transmit packet header CPSW_ALE_UVLAN_UNTAG[1-0] UVLAN_FORCE_UNTAGGED_EGRESS bit is asserted. The VLAN_LTYPE_SEL length/type is removed as is the two byte packet VLAN. Input 64-67 byte VLAN tagged packets go out with the VLAN removed and padded to 64-bytes. The input CRC bytes are used as the pad data. Input 64-byte VLAN tagged packets use all four input CRC bytes as pad, input 65-byte priority tagged packets use three of the input CRC bytes as pad, and so on. The output CRC is replaced with a generated CRC when the VLAN is removed.

  Note

  VLAN tagged receive packets of 64 to 67 bytes will be padded to 64 bytes on egress (Ethernet and host port egress) if the VLAN is to be removed on egress.

13.2.1.4.6.4.2 Ethernet Port Ingress Packet CRC

All Ethernet ports check the ingress packet CRC in all modes/speeds. The receive port can check either Ethernet CRC or Castagnoli CRC as determined by the CRC_TYPE bit in the CPSW_PN_MAC_CONTROL_REG register.

13.2.1.4.6.4.3 Ethernet Port Egress Packet CRC

Ethernet ports transmit each egress packet with the CRC selected by the CRC_TYPE bit in the CPSW_PN_MAC_CONTROL_REG register, regardless of the type of CRC that the packet had on ingress to the switch. At the egress port after passing through the switch, the packet CRC is checked for correctness and if the CRC is correct then the packet is output with the generated selected output CRC. If the packet CRC is incorrect, due either to a bit flip in a memory or an error CRC passed in on host ingress, then the generated egress CRC type is used with at least a single byte of the CRC inverted to indicate the error. If the packet length including CRC is divisible by 4 then all 4 CRC bytes will be inverted on error. If there are three bytes remainder after dividing the packet length by 4 then three bytes will be inverted (and so on down to one byte remainder).

13.2.1.4.6.4.4 CPPI Port Ingress Packet CRC

CPPI port ingress packets can be passed in with or without a CRC. The ingress packet CRC type is indicated in the buffer descriptor word CRC_TYPE bit and can be Ethernet (or Castagnoli if $cppi_cast = 1). The packet CRC_TYPE can change from packet to packet if Castagnoli is supported ($cppi_cast = 1). The P0_RX_PASS_CRC_ERR bit in the CPSW Control register determines if ingress packets with CRC errors are passed or dropped. Passed packets with CRC errors will be transmitted on Ethernet egress with a CRC error.

13.2.1.4.6.4.5 CPPI Port Egress Packet CRC

The P0_TX_CRC_REMOVE bit in the CPSW_CONTROL_REG register determines if CPPI egress packet have a CRC included or not. If present, the CRC type for all packets is determined by the P0_TX_CRC_TYPE bit in the CPSW Control register. Egress packets not filtered on Ethernet ingress due to PN_RX_CEF_EN have the packet error CRC included (not replaced by the egress CRC type_) if the CRC is not removed on egress. The error is indicated in the buffer descriptor. CPPI egress packets that detected a CRC error on the internally generated Castagnoli CRC, due to a bit flip in logic or memory, will indicate the error with the drop bit set in the buffer descriptor.

13.2.1.4.6.5 FIFO Memory Control

Each of the two CPSW ports has an identical associated FIFO. Each FIFO contains a single logical receive queue and eight logical transmit queues (priority 0 through 7 with 7 the highest priority). Each FIFO memory contains 20,480 bytes (20k) total contained in a single memory instance. The FIFO memory is used for the associated port transmit and receive queues. The TX_MAX_BLKS field in the FIFOs associated CPSW_PN_MAX_BLKS_REG register determines the maximum number of 1k FIFO memory blocks to be allocated to the eight logical transmit queues (transmit total). The RX_MAX_BLKS field in the FIFO's associated CPSW_PN_MAX_BLKS_REG register determines the maximum number of 1k memory blocks to be allocated to the logical receive queue. The TX_MAX_BLKS value plus the RX_MAX_BLKS value must sum to 20 (the total number of blocks in the FIFO). If the sum were less than 20, then some memory blocks would be unused. The default is 16 (decimal) transmit blocks and four receive blocks. The FIFOs follow the naming convention of the Ethernet ports. Host Port is Port0 and External Ports is Port1.

Each transmit FIFO contains a total of twenty 1k blocks that can be allocated to any priority.

13.2.1.4.6.6 FIFO Transmit Queue Control

There are eight transmit queues in the Ethernet port transmit FIFO. Software has some flexibility in determining how packets are loaded into the queues and on how packet priorities are selected for transmission (how packets are removed and transmitted from queues).

13.2.1.4.6.6.1 CPPI Port Receive Rate Limiting

Port 0 receive operations can be configured to rate limit the packet data for each receive channel (priority). Receive has 8 priorities for QOS. There is a committed information rate (CPSW_P0_PRI_CIR_REG_y, where y = 0 to 7) and an excess information rate for each priority (CPSW_P0_PRI_EIR_REG_y, where y = 0 to 7). Rate limiting is enabled for a priority when the committed information rate for the priority is non-zero. The excess information rate for a priority is enabled when the excess information rate for the priority is non-zero. The committed information rate must be non-zero if the excess information rate is configured to be non-zero. That is, there must be a configured non-zero committed information rate for there to be a configured non-zero excess information rate. Bulk traffic on other non-rate limited priorities does not impact the committed information traffic on a priority. However, bulk traffic on other non-rate limited threads does impact the excess information rates. No bulk priority will be enabled to send unless there are CPSW_PN_PRI_CTL_REG[15-12] TX_HOST_BLKS_REM number of unused blocks remaining in each of the Ethernet port transmit FIFOs. The “blocks remaining check” ensures that bulk traffic from the host will not block rate-limited traffic from the host. Rate limited channels must be the highest priority channels. For example, if two rate limited channels are required then priorities 7 and 6 should be configured for committed information (and excess information if desired). When any channels are configured to be rate-limited, the priority type must be fixed for receive. Round-robin priority type is not allowed when rate-limiting is configured for any priority. The configured transfer rate includes the inter-packet gap (12 bytes) and the preamble (8 bytes). The rate in Mbits/second that each priority is configured to receive is controlled by the below equation. If the configured excess information rate is zero, then only the committed information rate is transferred:

Priority Transfer rate [Mbit/s] = ((((Frequency in MHZ) * CPSW_P0_PRI_CIR_REG_y) / 32768) + (((Frequency in MHZ) * CPSW_P0_PRI_EIR_REG_y) / 32768))

Where the *frequency* is the VBUSP_GCLK frequency (in MHz) and priority 0 to 7.

For example, 10Mbps on priority 7 would give the below:

10Mbps = ~ ((350 * 936) / 32768), at 350Mhz and CPSW_P0_PRI_CIR_REG_y[27-0] PRI_CIR value = 936 (no excess information rate)

13.2.1.4.6.6.2 Ethernet Port Transmit Rate Limiting

Ethernet port transmit operations can be configured to rate limit egress data for each egress priority. There is a committed information rate (CPSW_P0_PRI_CIR_REG_y, where y = 0 to 7) and an excess information rate for each priority (CPSW_P0_PRI_EIR_REG_y, where y = 0 to 7). Rate limiting is enabled for a priority when the committed information rate for the priority is non-zero. The excess information rate for a priority is enabled when the excess information rate for the priority is non-zero. The committed information rate must be non-zero if the excess information rate is configured to be non-zero. That is, there must be a configured non-zero committed information rate for there to be a configured non-zero excess information rate. Bulk traffic on other non-rate limited priorities does not impact the committed information traffic on a priority. However, bulk traffic on other non-rate limited priorities does impact the excess information rates. Rate limited channels must be the highest priority channels. For example, if two rate limited channels are required then priorities 7 and 6 should be configured for committed information (and excess information if desired). The configured transfer rate includes the inter-packet gap (12 bytes) and the preamble (8 bytes). The rate in Mbits/second that each priority is configured to send is controlled by the below equation. If the excess information rate is disabled then the committed information rate only is transferred:

Priority Transfer rate [Mbit/s] = ((((Frequency in MHZ) * CPSW_P0_PRI_CIR_REG_y) / 32768) + (((Frequency in MHZ) * CPSW_P0_PRI_EIR_REG_y) / 32768))

Where the *frequency* is the VBUSP_GCLK frequency (in MHz) and priority 0 to 7.

13.2.1.4.6.7 Enhanced Scheduled Traffic (EST – P802.1Qbv/D2.2)

13.2.1.4.6.7.1 Enhanced Scheduled Traffic Overview

- When enabled and configured, EST allows express queue traffic to be scheduled (placed) on the wire at specific repeatable time intervals.
- EST operates on a repeating time interval generated by the CPTS EST function generator. For example, a 125us repeating time interval can be configured.
- Each Ethernet port has 128 EST fetch commands maximum in the global EST fetch RAM.
- Each 22-bit fetch command consists of a 14-bit fetch count (14 MSB's) and an 8-bit priority fetch allow (8 LSB's) that will be applied for the fetch count time in wireside clocks.
- The configured port fetch commands are executed in sequence, beginning at port address zero each time through the time interval beginning at cycle start.
- EST allows non-scheduled express and prempt queue traffic to be cleared from the wire to ensure that the scheduled traffic is transmitted at the proper time (zero allow).
- EST can be used with or without premption. The CPSW_PN_IET_CONTROL_REG[23-16] MAC_PREMPT value determines whether the priority is enabled on the express or prempt queue. Whether a priority is on the express or prempt queue only effects the wire clear time from an EST operation perspective.
- Software should not move priorities to the prempt queue unless preemption is configured, enabled, and verified allowing preemption to occur.
- Express packet time stamp events can be enabled to assist software in configuring and timing EST operations.

13.2.1.4.6.7.2 Enhanced Scheduled Traffic Fetch RAM

- The EST fetch RAM is read/writable in the CPSW configuration address space.
- The Ethernet transmit port has 128 locations in the global EST fetch RAM.
  - Ethernet port 1 has EST fetch RAM addresses 0x000-0x07F.
- **One buffer operation:** When CPSW_PN_EST_CONTROL_REG[0] EST_ONEBUF is set to 1h, the 128 port locations operate as one buffer. The EST_BUFACT bit in CPSW_PN_FIFO_STATUS_REG register is the upper address bit of the port's fetch RAM address indicating whether operation is currently in the upper or lower 64 locations of the port's fetch RAM.
- **Two buffer operation:** When CPSW_PN_EST_CONTROL_REG[0] EST_ONEBUF is cleared there are two 64-location buffers with CPSW_PN_EST_CONTROL_REG[1] EST_BUFSEL selecting the buffer to be used. When the buffer is switched by changing the CPSW_PN_EST_CONTROL_REG[1] EST_BUFSEL value, the actual switch occurs on cycle start. The actual buffer being used is indicated by the EST_BUFACT bit in CPSW_PN_FIFO_STATUS_REG. Software should avoid writing the switched out buffer fetch RAM locations until it detects that the actual switch has occurred.
- The first address location in the port's fetch RAM space (location zero) is read at the beginning of each EST time interval (cycle start). Addresses are then read in ascending order for the duration of the interval. The port address zero is then read again at the beginning of the next cycle repeating the time interval packet operations.

13.2.1.4.6.7.3 Enhanced Scheduled Traffic Time Interval

- Each Ethernet port has an Enhanced Scheduled Traffic Function (ESTF) generator in the CPTS submodule.
- The EST function generator generates the EST time interval as a configured number of CPTS reference clocks (CPTS_RFT_CLK).
- The EST function generator rising edge is the cycle start time and the cycle repeats (cycle start occurs) after every time interval.
- The first fetch allowed value is at the port base address zero in the EST fetch RAM and is actually applied 16 wireside clocks after cycle start. The 16 clock cycle delay allows the first fetch value time to be fetched from the EST fetch RAM (prefetch time at cycle start).
- Each successive fetch allow is applied for the associated fetch count thereafter. The minimum non-zero fetch count is 16. The minimum value of 16 guarantees that the next fetch value has time to be fetched before the current fetch count is over. There are 64 maximum fetch values when CPSW_PN_EST_CONTROL_REG[0] EST_ONEBUF = 0h, and 128 maximum fetch values when CPSW_PN_EST_CONTROL_REG[0] EST_ONEBUF = 1h.
- The next cycle start then causes the fetch to once again start at the port address zero.

13.2.1.4.6.7.4 Enhanced Scheduled Traffic Fetch Values

- The 22-bit fetch value is made up of the 14-bit fetch count and the 8-bit fetch allow.
- The fetch time indicates the number of wireside clocks that the fetch allow will be active.
- The fetch count is in Ethernet wireside clocks which is bytes in Gigabit mode (CPSW_PN_MAC_CONTROL_REG[7] GIG = 1h) and nibbles in 10/100Mbps mode.
- When a fetch allow bit is set, the corresponding priority is enabled to begin packet transmission on an allowed priority subject to rate limiting. The actual packet transmission on the wire may carry over into the next fetch count and is the reason for the wire clear time in the fetch zero allow.
- When a fetch allow bit is cleared, the corresponding priority is not enabled to transmit for the fetch count time.
- A non-zero fetch allow value with a non-zero fetch count causes the fetch allow value to be applied for the fetch count number of wireside clocks.
- A zero fetch count causes the associated fetch allow to be held for the duration of the cycle (until the next cycle start).
- A zero fetch allow with a non-zero fetch count is intended to clear the wire for a scheduled (timed) express packet in the next fetch. A zero fetch allow indicates that no packet can be started for transmission for the associated fetch count. The associated fetch count must be sufficient to guarantee that the wire is cleared given that a packet on an allowed priority in the previous fetch could have been started on the previous clock and that there is hardware latency in the clear time. The timed packet should be sent on a priority that is enabled in the next fetch but disabled in the current zero allow fetch. The fetch allow previous to a zero allow should have only prempt priorities enabled or only express priorities enabled but not both.
- The number of clocks required to clear the wire varies depending Ethernet wire speed and on whether express or prempt priorities were allowed in the previous fetch command.

13.2.1.4.6.7.5 Enhanced Scheduled Traffic Packet Fill

Packet fill can be configured and enabled to occur in the fetch count time associated with a fetched zero allow that preceeds a timed express packet. The intention with fill is that a smaller packet on a non-timed priority might be able to be inserted on the wire during the wire clear time which would increase wire utilization. Fill must be configured to ensure that any fill packet does not conflict with the timed express packet allowed in the next fetch. Incorrect configuration might push out in time any express timed packet which indicates that the fill margin needs to be increased

Fill Configuration:

- The **est_fill_margin** value in **PN_EST_CONTROL_REG** should be written with a 0x100 value
- The **est_prempt_comp** value in **PN_EST_CONTROL_REG** should be written with a 0x12 value (if IET is to be configured and enabled). This value times eight is the number of wireside clocks required to clear prempt packets off the wire at the end of a zero allow
- The **est_fill_en** bit in **PN_EST_CONTROL_REG** should be set

13.2.1.4.6.7.6 Enhanced Scheduled Traffic Time Stamp

The EST can be configured to generate CPTS timestamp events for selected express traffic. The EST timestamp events use the CPTS host event type (CPSW_CPTS_EVENT_1_REG[23-20] EVENT_TYPE = 7 decimal. The EST timestamps will not override host sent timestamps for packets that were sent from the host with an enabled host timestamp.

- EST Events (host events) contain the below information:
  - Time Stamp of the selected express packet.
  - The event CPSW_CPTS_EVENT_1_REG[28-24] PORT_NUMBER indicates the transmit port number.
  - The event CPSW_CPTS_EVENT_1_REG[23-20] EVENT_TYPE is decimal 7 (host event).
  - The event CPSW_CPTS_EVENT_1_REG[23-20] MESSAGE_TYPE indicates the packet transmit hardware switch priority.
  - The event CPSW_CPTS_EVENT_1_REG[15-0] SEQUENCE_ID upper nibble indicates the packet receive port number.
  - The event CPSW_CPTS_EVENT_1_REG[15-0] SEQUENCE_ID lower byte indicates the sequence number of the express packet in numerical order. The first event is event one, the second is event two and so on. The sequence ID rolls over to zero after 0xFF (8-bits).

  - The event domain is the value from the CPSW_EST_TS_DOMAIN_REG[7-0] EST_TS_DOMAIN register.
- When CPSW_PN_EST_CONTROL_REG[2] EST_TS_EN is set, timestamp events will be generated on selected express traffic.
- When CPSW_PN_EST_CONTROL_REG[3] EST_TS_FIRST is also set, events will be generated only on the first express packet in each time interval. If CPSW_PN_EST_CONTROL_REG[4] EST_TS_ONEPRI is also set then the event will only be on the first CPSW_PN_EST_CONTROL_REG[7-5] EST_TS_PRI express packet in the time interval. If CPSW_PN_EST_CONTROL_REG[4] EST_TS_ONEPRI is clear then the event will be generated on the first express packet in the time interval on any priority.
- When CPSW_PN_EST_CONTROL_REG[3] EST_TS_FIRST is clear, events will be generated on every express packet. If CPSW_PN_EST_CONTROL_REG[4] EST_TS_ONEPRI is set then the event will be generated on every CPSW_PN_EST_CONTROL_REG[7-5] EST_TS_PRI express packet. If CPSW_PN_EST_CONTROL_REG[4] EST_TS_ONEPRI is clear then event will be generated on every express packet on any priority.

13.2.1.4.6.8 Audio Video Bridging

Audio Video Bridging is an ongoing project of IEEE 802.1 concerned with enabling low-latency streaming of time-sensitive audiovisual data over networks. Devices are designated as talkers (transmitters), bridges, or listeners (receivers). It is suggested that the maximum latency could be 2 ms over 7 hops for Class A devices and 20 ms over 7 hops for Class B devices. A hop is essentially a single local area network stage in the journey of a packet. Every time a bridge is encountered between one network section and another a hop is involved. One of the performance goals is that AVB streams will not use more than 75 percent of a link's bandwidth, leaving the remaining capacity for non-AVB streams.

The goal of developing AVB is simply--extend Ethernet's data-networking capabilities to the realm of reliable real-time audio/video networking.

An "Audio Video Bridging" network is one that implements a set of protocols being developed by the IEEE 802.1 Audio/Video Bridging Task Group. There are four primary differences between the proposed Audio Video Bridging architecture and existing 802 architectures (from now on the term "AVB" will be used instead of "Audio Video Bridging"):

1. Precise synchronization - IEEE 802.1AS: "*Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks.* "a.k.a. Precision Time Protocol (PTP).
2. Traffic shaping for media streams - IEEE 802.1Qav: "*Virtual Bridged Local Area Networks:* Forwarding and Queuing for Time-Sensitive Streams."
3. Admission controls - IEEE 802.1Qat: "*Virtual Bridged Local Area Networks - Amendment 9:* Stream Reservation Protocol (SRP)."
4. Identification of non-participating devices - IEEE 802.1BA: "Audio/Video Bridging (AVB) Systems"

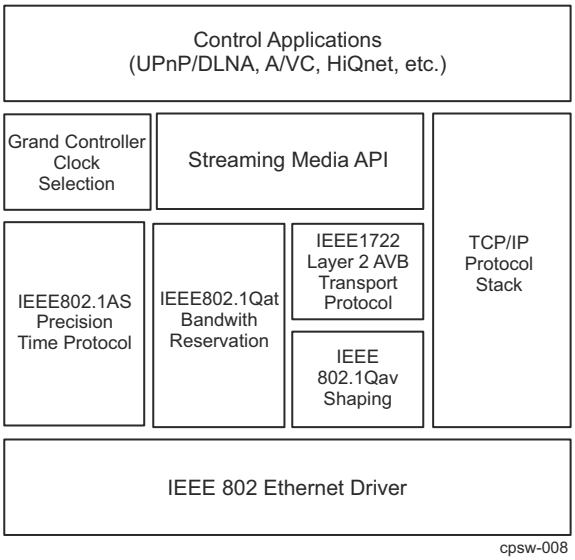


Figure 13-92. The Network Static with AVB

The following sections describe the media transport protocols that work within the AVB framework.

13.2.1.4.6.8.1 IEEE 802.1AS: Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks (Precision Time Protocol (PTP))

The protocol defined by 802.1AS automatically selects a device to be the controller clock, and then distributes this clock throughout the bridged LAN / IP subnet to all other network devices using link-specific transmit/receive time-stamping. However, we only use a two-step solution only on transmit. That is, we do not modify a packet with the timestamp on the way out. The timestamp packet is sent out and then a separate message with the timestamp is sent by the host afterward. Receive can be one or two step.

Note

The 802.1AS-distributed clock is not used as a media clock. Rather, the shared 802.1AS clock reference is used to regenerate the media clock at the listener/renderer. Such a reference removes the need to force the latency of the network to be constant, or compute long running averages in order to estimate the actual media rate of the transmitter in the presence of substantial network jitter. IEEE 802.1AS is based on the ratified IEEE 1588 standard.

Based on IEEE 1588:2002,A PTP devices exchange standard Ethernet messages that synchronize network nodes to a common time reference by defining clock controller selection and negotiation algorithms, link delay measurement and compensation, and clock rate matching and adjustment mechanisms.

Designed as a simplified profile of IEEE 1588, a primary difference between 1588 and IEEE 802.1AS is that PTP is a layer 2-in other words, a non-IP routable protocol. Like IEEE 1588, PTP defines an automatic method for negotiating the network clock controller, the Best Controller Clock Algorithm (BCCA). PTP nodes can be assigned one of eight priority levels, presumably based on clock quality. BMCA defines the underlying negotiation and signaling mechanism whose purpose is to identify the AVB LAN Grandcontroller. Once a Grandcontroller has been selected, synchronization automatically begins.

At the core of 802.1AS synchronization is time-stamping. In short, during PTP message ingress/egress from the 802.1AS-capable MAC, the PTP Ether type triggers the sampling of the value of a local real-time counter (RTC). Target nodes compare the value of their RTC against the PTP Grandcontroller and, by use of link delay measurement and compensation techniques, match their RTC value to the time of the AVB LAN PTP domain. After network time throughout the AVB LAN has converged, periodic SYNC and FOLLOW_UP messages provide the information that enables the PTP rate matching adjustment algorithms. The result is all PTP nodes are then synchronized to the same "Wall Clock" time. PTP assures 1-µs accuracy over seven network hops.

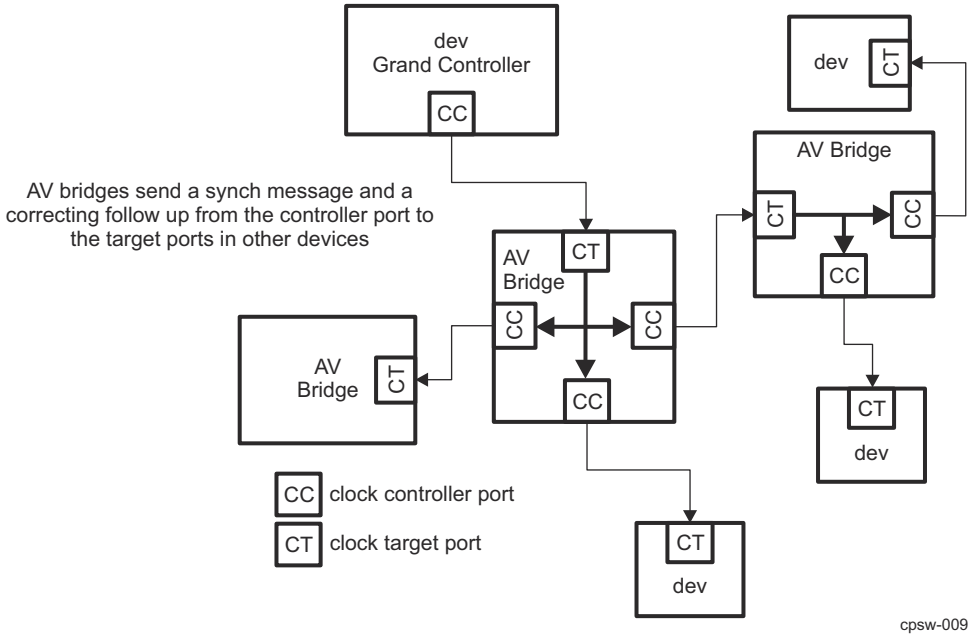


Figure 13-93. AVB Network & PTP Clock Entities

The media transport protocols that work within the AVB framework are:

13.2.1.4.6.8.1.1 IEEE 1722: "Layer 2 Transport Protocol for Time-Sensitive Streams"

AVBTP or 1722 sits above the IEEE 802.1 AVB plumbing and below the application layer. It acts as the conduit between an Ethernet MAC and a streaming application. AVBTP abstracts the underlying network transmission channel to enable the virtual connection of distributed audio and video CODECs over reliable Ethernet networks. A complete AVBTP Ethernet packet is shown in Figure 13-94 and illustrates how IEC 61883-6 AM824 uncompressed audio samples are encapsulated in an Ethernet frame.

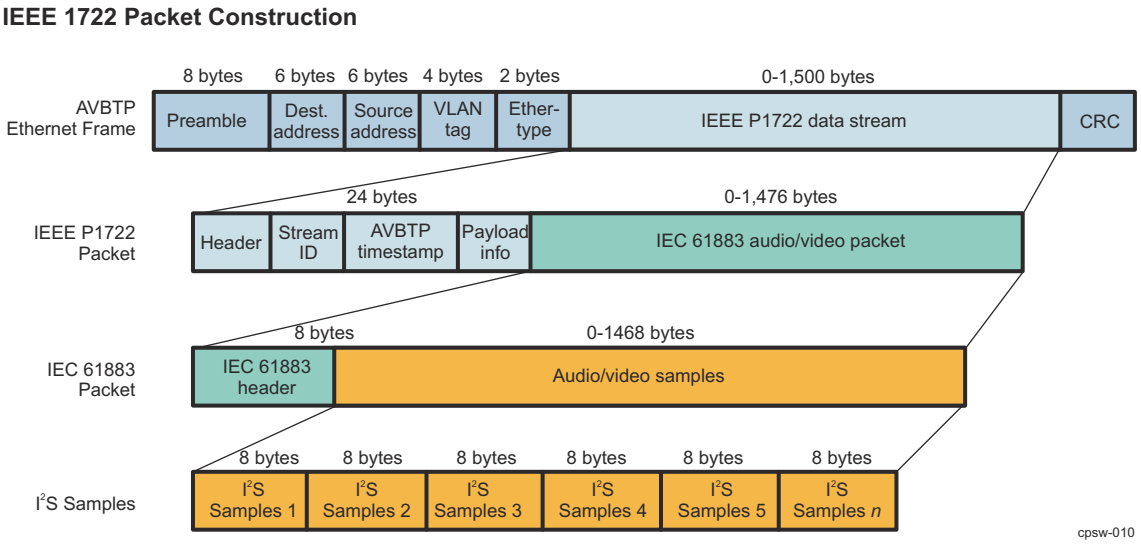


Figure 13-94. IEEE 1722 Packets

1722 or AVBTP Presentation Time and Synchronization:

Synchronization in an AVB network starts with the Precision Time Protocol but ends with synchronized media clocks. PTP is responsible for synchronizing all nodes in an AVB network to identical wall clock time; not for synchronizing media clocks. In other words, PTP does not actually transport synchronized media clocks but instead provides a low-level building block crucial for managing a distributed media synchronization system.

A crucial benefit of this approach is coexistence of multiple, independent media clock domains on an AVB network. Unrelated audio and video streams can simultaneously exist in the same LAN.

13.2.1.4.6.8.1.1.1 Cross-timestamping and Presentation Timestamps

AVBTP assumes that AVB node media clocks are clocked by free-running oscillators. It is also assumed that the node's internal concept of wall clock time has been synchronized to the PTP Grandcontroller. AVBTP media clock sources embed "AVBTP Presentation Timestamps" in AVBTP streaming packets. Figure 13-95 illustrates the relationship between PTP network time and AVBTP Presentation Timestamps.

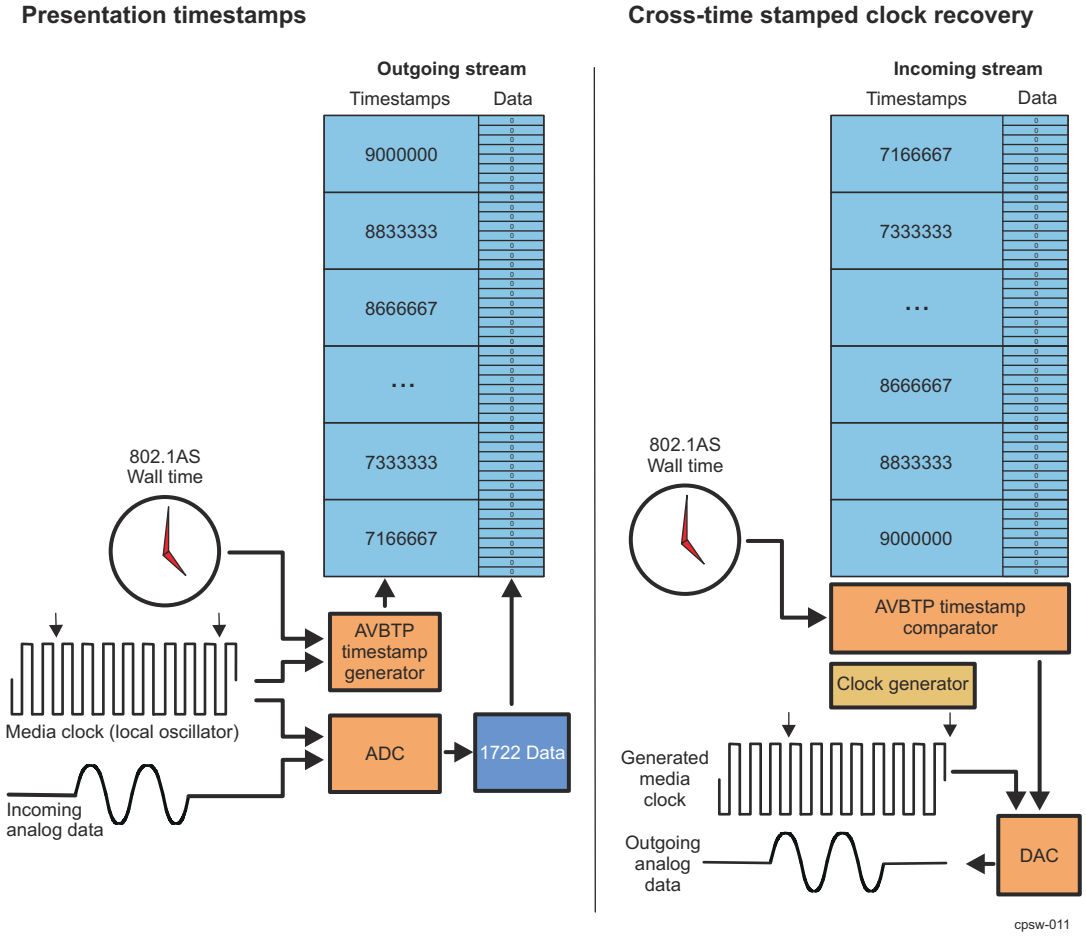


Figure 13-95. Cross Time Stamping and Presentation Timestamps

13.2.1.4.6.8.1.2 IEEE 1733: Extends RTCP for RTP Streaming over AVB-supported Networks

This standard specifies the protocol, data encapsulations, connection management and presentation time procedures used to ensure interoperability between audio and video based end stations that use standard networking services provided by all IEEE 802 networks meeting QoS requirements for time-sensitive applications by leveraging the Real-time Transport Protocol (RTP) family of protocols and IEEE 802.1 Audio/ Video Bridging (AVB) protocols.

13.2.1.4.6.8.2 IEEE 802.1Qav: "Virtual Bridged Local Area Networks: Forwarding and Queuing for Time-Sensitive Streams"

This standard allows bridges to provide guarantees for time-sensitive (that is, bounded latency and delivery variation), loss-sensitive real-time audio video (AV) data transmission (AV traffic). It specifies per priority ingress metering, priority regeneration, and timing-aware queue draining algorithms. This standard uses the timing derived from IEEE 802.1AS. Virtual Local Area Network (VLAN) tag encoded priority values are allocated, in aggregate, to segregate frames among controlled and non-controlled queues, allowing simultaneous support of both AV traffic and other bridged traffic over and between wired and wireless Local Area Networks (LANs).

Such a guarantee in bandwidth is provided by two functional entities:

- A registration protocol, which registers the service and its maximum network utilization with a device or switch (IEEE 802.1Qat: "Virtual Bridged Local Area Networks - Amendment 9: Stream Reservation Protocol (SRP)")
- A hardware bandwidth management service.
  - Receive policing
  - Transmit rate control.

End Station Behavior

In order for an end station to successfully participate in the transmission and reception of time-sensitive streams, it is necessary for their behavior to be compatible with the operation of the forwarding and queuing mechanisms employed in bridges.

The requirements for end stations that participate as "talkers" i.e., sources of time-sensitive streams are different from the requirements that apply to "listeners", the destination station(s) for the streams.

Talker Behavior

In order for Talker-originated data streams to make use of the credit-based shaper behavior in Bridges, it is a requirement for a Talker to use the priorities that the Bridges in the network recognize as being associated with SR classes exclusively for transmitting stream data.

It is also necessary for the Talker and the Bridges in the path to the Listener(s), to have a common view of the bandwidth required in order to transmit the Talker's streams, and for that bandwidth to be reserved along the path to the Listener(s). This latter requirement can be met by means of stream reservation mechanisms, such as defined in SRP, or by other management means.

End stations that are Talkers shall exhibit transmission behavior for frames that are part of "time-sensitive streams" that is consistent with the operation of the credit-based shaper algorithm, both in terms of the way they transmit frames that are part of an individual data stream, and in terms of the way they transmit stream data frames from a Port.

In effect, the queuing model for a Talker Port (and a Listener port), and for given priorities, can be considered to look like Figure 13-96.

Listener Behavior

The primary requirement for a listener station is that it is capable of buffering the amount of data that could be transmitted for a stream during a time period equivalent to the accumulated maximum jitter that could be experienced by stream data frames in transmission between Talker and Listener.

From the point of view of the specification of the forwarding and queuing requirements for time-sensitive streams, it is assumed that the listener will assess the buffering required for a stream as part of the stream bandwidth reservation mechanisms employed by the implementation.

The credit-based shaper's operation details are beyond the scope of this document.

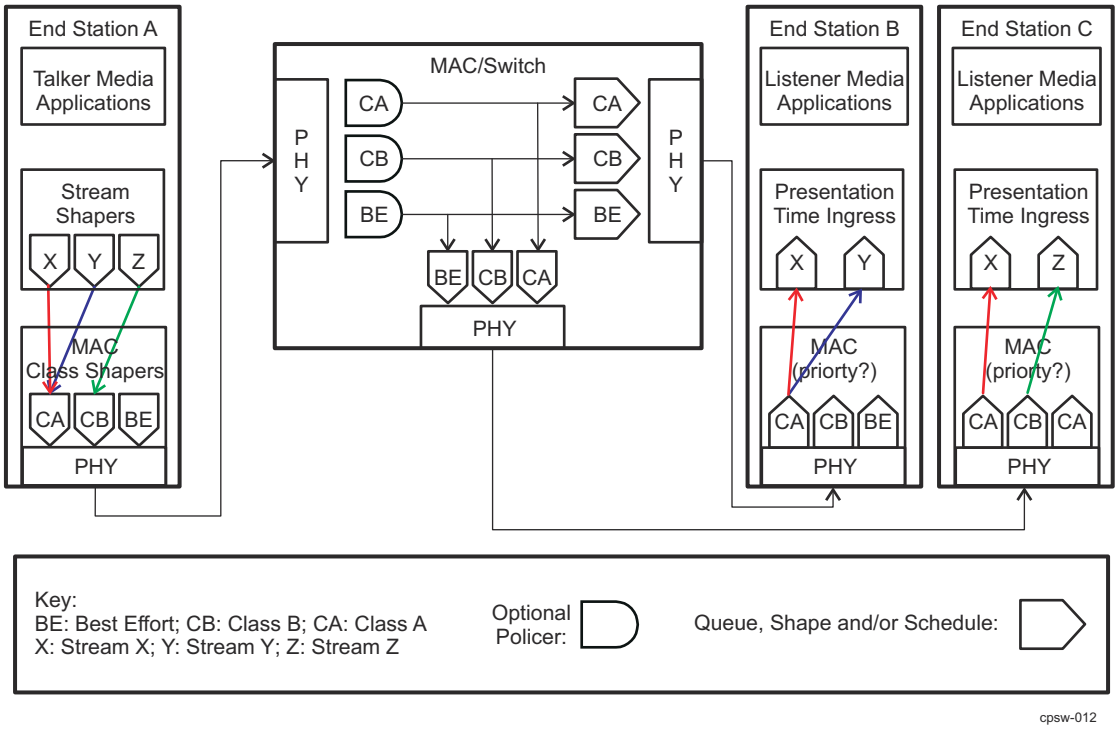


Figure 13-96. AV Stream Queuing/Policing

13.2.1.4.6.8.2.1 Configuring the Device for 802.1Qav Operation

There is no dedicated register-set to be configured for the time-sensitive stream handling. The list of functional features of CPSW that will have to be configured are:

- DESCRIPTORS and CHANNEL CONFIGURATIONS:
  - CPPI TX and RX descriptors
  - VLAN and Priority tags

Table 13-144. Example of TX Configuration

TX DMA CHANNEL	Packet Priority	Switch Queue Priority
7	7	3
6	5	2
5	3	1
4	1	0

Table 13-145. Example of RX Configuration

RX DMA CHANNEL	Packet Priority	Switch Queue Priority
0	7	0
0	5	0
0	3	0
0	1	0

- ALE Configuration:
  - ALE in VLAN-ware mode, Non-ALE in bypass mode.

13.2.1.4.6.9 Ethernet MAC Sliver

The Ethernet port peripheral is compliant to the IEEE Std 802.3 Specification. Half-duplex mode is supported in 10/100 Mbps mode, but not in 1000 Mbps (gigabit) mode.

Features:

- Synchronous 10/100/1000 Mbit operation
- RMII/RGMII Interface
- Hardware Error handling including CRC
- Full-Duplex Gigabit operation (half-duplex gigabit is not supported)
- EtherStats and 802.3Stats RMON statistics gathering support for external statistics collection module
- Transmit CRC generation selectable on a per channel basis
- Emulation Support
- VLAN Aware Mode Support
- Hardware flow control
- Programmable Inter Packet Gap (IPG).

13.2.1.4.6.9.1 Ethernet MAC Sliver Overview

13.2.1.4.6.9.1.1 CRC Insertion

The MAC generates and appends a 32-bit Ethernet CRC onto the transmitted data, if the transmit packet header PASS_CRC bit is 0h. For the Ethernet port generated CRC case, a CRC at the end of the input packet data is not allowed.

If the header word PASS_CRC bit is set, then the last four bytes of the TX data are transmitted as the frame CRC. The four CRC data bytes should be the last four bytes of the frame and should be included in the packet byte count value. The MAC performs no error checking on the outgoing CRC when the PASS_CRC bit is set.

13.2.1.4.6.9.1.2 MTXER

The MTXER signal is only used for EEE. If an underflow condition occurs on a transmitted frame, the frame CRC will be inverted to indicate the error to the network. Underflow is a hardware error.

13.2.1.4.6.9.1.3 Adaptive Performance Optimization (APO)

The Ethernet MAC port incorporates Adaptive Performance Optimization (APO) logic that may be enabled by setting the TX_PACE bit in the CPSW_PN_MAC_CONTROL_REG register. Transmission pacing to enhance performance is enabled when set. Adaptive performance pacing introduces delays into the normal transmission of frames, delaying transmission attempts between stations, reducing the probability of collisions occurring during heavy traffic (as indicated by frame deferrals and collisions) thereby increasing the chance of successful transmission.

When a frame is deferred, suffers a single collision, multiple collisions or excessive collisions, the pacing counter is loaded with an initial value of 31. When a frame is transmitted successfully (without experiencing a deferral, single collision, multiple collision or excessive collision) the pacing counter is decremented by one, down to zero.

With pacing enabled, a new frame is permitted to immediately (after one IPG) attempt transmission only if the pacing counter is zero. If the pacing counter is non zero, the frame is delayed by the pacing delay, a delay of approximately four inter-packet gap delays. APO only affects the IPG preceding the first attempt at transmitting a frame. It does not affect the back-off algorithm for re-transmitted frames.

13.2.1.4.6.9.1.4 Inter-Packet-Gap Enforcement

The measurement reference for the IPG of 96-bit times is changed depending on frame traffic conditions. If a frame is successfully transmitted without collision, and MCRS is de-asserted within approximately 48-bit times of MTXEN being de-asserted, then 96-bit times is measured from MTXEN. If the frame suffered a collision, or if MCRS is not de-asserted until more than approximately 48-bit times after MTXEN is de-asserted, then 96-bit times (approximately, but not less) is measured from MCRS.

The Ethernet port transmit inter-packet gap (IPG) may be shortened by eight bit times when short gap is enabled and triggered. Setting the [10] TX_SHORT_GAP_ENABLE bit in the CPSW_PN_MAC_CONTROL_REG register enables the gap to be shortened when triggered. The condition is triggered when the ports associated transmit packet FIFO has a user defined number of FIFO blocks used. The associated transmit FIFO blocks used value determines if the gap is shortened, and so on. The CPSW_GAP_THRESH_REG register value determines the short gap threshold. If the FIFO blocks used is greater than or equal to the GAP_THRESH value then short gap is triggered.

13.2.1.4.6.9.1.5 Back Off

The Gigabit Ethernet Mac Sliver implements the 802.3 binary exponential back-off algorithm.

13.2.1.4.6.9.1.6 Programmable Transmit Inter-Packet Gap

The transmit inter-packet gap (IPG) is programmable through the CPSW_PN_MAC_CONTROL_REG register. The default value is decimal 12. The transmit IPG may be increased to the maximum value of 1FFh. Increasing the IPG is not compatible with transmit pacing. The short gap feature will override the increased gap value, so the short gap feature may not be compatible with an increased IPG.

13.2.1.4.6.9.1.7 Speed, Duplex and Pause Frame Support Negotiation

The Ethernet port can operate in half duplex or full duplex in 10/100 Mbit modes, and can operate in full duplex only in 1000 Mbit mode. Pause frame support is included in 10/100/1000 Mbit modes as configured by the host.

13.2.1.4.6.9.2 RMII Interface

The CPRMII peripheral is compliant to the RMII specification document.

13.2.1.4.6.9.2.1 Features

- Source Synchronous 10/100 Mbit operation
- Full and Half Duplex support

13.2.1.4.6.9.2.2 RMII Receive (RX)

The CPRMII receive (RX) interface converts the input data from the external RMII PHY (or switch) into the required MII (CPGMAC) signals. The carrier sense and collision signals are determined from the RMII input data stream and transmit inputs as defined in the RMII specification.

An asserted RMII_RXER on any di-bit in the received packet will cause an MII_RXER assertion to the CPGMAC during the packet. In 10Mbps mode, the error is not required to be duplicated on 10 successive clocks. Any di-bit which has an asserted RMII_RXER during any of the 10 replications of the data will cause the error to be propagated.

Any received packet that ends with an improper nibble boundary aligned RMII_CRS_DV toggle will issue an MII_RXER during the packet to the CPGMAC. Also, a change in speed or duplex mode during packet operations will cause packet corruption.

The CPRMII can accept receive packets with shortened preambles, but 0x55 followed by a 0x5D is the shortest preamble that will be recognized (1 preamble byte with the start of frame byte). At least one byte of preamble with the start of frame indicator is required to begin a packet. An asserted RMII_CRS_DV without at least a single correct preamble byte followed by the start of frame indicator will be ignored.

13.2.1.4.6.9.2.3 RMII Transmit (TX)

The CPRMII transmit (TX) interface converts the CPGMAC MII input data into the RMII transmit format. The data is then output to the external RMII PHY.

The CPGMAC does not source the transmit error (MII_TXERR) signal. Any transmit frame from the CPGMAC with an error (underrun) will be indicated as an error by an error CRC. Transmit error is assumed to be de-asserted at all times and is not an input into the CPRMII module. Zeroes are output on RMII_TXD[1:0] for each clock that RMII_TXEN is de-asserted.

13.2.1.4.6.9.3 RGMII Interface

The CPRGMII peripheral is compliant to the RGMII specification document.

13.2.1.4.6.9.3.1 Features

- Supports 1000/100/10 Mbps speed
- Full and Half Duplex support (CPGMAC supports only Full duplex in Gigabit mode).
- MII mode support
- Energy Efficient Ethernet Support

13.2.1.4.6.9.3.2 RGMII Receive (RX)

The CPRGMII receive (RX) interface converts the source synchronous DDR input data from the external RGMII PHY into the required G/MII (CPGMAC) signals.

13.2.1.4.6.9.3.3 In-Band Mode of Operation

The CPRGMII is operating in the in-band mode of operation when the RGMII_RX_INBAND input is asserted. RGMII_RX_INPUT is asserted by configuring the CTL_EN bit to 1h of the CPSW_PN_MAC_CONTROL_REG register. The link status, duplexity, and speed are determined from the RGMII input data stream RXD[3:0] when RX_CTL is deasserted, as defined in the RGMII specification. The PHY might need to be configured beforehand to output in-band data. The in-band data is indicated as shown in Table 13-146.

Table 13-146. In-Band Data

RXD3	RXD[2:1]		RXD0
Duplex status:	Link Speed:	RXC_CLK Speed:	Link Status:
0h: half-duplex	0h: 10-Mbps mode	2.5 MHz	0h: Link is down
1h: full-duplex	1h: 100-Mbps mode	25 MHz	1h: Link is up
	2h: 1000-Mbps mode	125 MHz	
	3h: Reserved	Reserved	

13.2.1.4.6.9.3.4 Forced Mode of Operation

The CPRGMII is operating in the forced mode of operation when the RGMII_RX_INBAND input is deasserted by setting to 0h bit CTL_EN of the CPSW_PN_MAC_CONTROL_REG register. In the forced mode of operation, the in-band data is ignored if present. The link status is forced high, and the duplexity and speed are determined from the CPSW_PN_MAC_CONTROL_REG[0] FULLDUPLEX and CPSW_PN_MAC_CONTROL_REG[7] GIG bits. If bit [7] GIG = 1h, then CPRGMII is operating in Gigabit mode. If bit [7] GIG is cleared (0h), then CPRGMII is operating in 100 Mbps mode.

13.2.1.4.6.9.3.5 RGMII Transmit (TX)

The CPRGMII transmit (TX) interface converts the CPGMAC G/MII input data into the DDR RGMII format. The DDR data is then output to the external PHY.

The CPGMAC does not source the transmit error (TXERR) signal. Any transmit frame from the CPGMAC with an error (underrun) will be indicated as an error by an error CRC. Transmit error is assumed to be deasserted at all times and is not an input into the CPRGMII module.

In 10/100 Mbps mode, the TXD[7:0] data bus uses only the lower nibble. The CPRGMII will output the lower nibble twice in 10/100 Mbps mode to avoid unnecessary signal switching.

Packets will be precluded from transmission through the CPRGMII module for 4096 transmit clocks after the rising edge of RGMII_LINK. Packet transmission will begin on the first TX_CTL rising edge after the 4096 transmit clock count has expired.

13.2.1.4.6.9.4 Frame Classification

Received frames are proper (good) frames if they are between 64 and CPSW_P0_RX_MAXLEN_REG[13-0] RX_MAXLEN in length (inclusive) and contain no errors (code/align/CRC).

Received frames are long frames if their frame count exceeds the value in the CPSW_P0_RX_MAXLEN_REG/ CPSW_PN_RX_MAXLEN_REG register. The register reset (default) value is 1518 (decimal). Long received frames are either oversized or jabber frames. Long frames with no errors are oversized frames. Long frames with CRC, code, or alignment errors are jabber frames.

Received frames are short frames if their frame count is less than 64 bytes. Short frames that contain no errors are undersized frames. Short frames with CRC, code, or alignment errors are fragment frames. If RX_CSF_EN bit in CPSW_PN_MAC_CONTROL_REG is set to 1h, undersized frames from 33 to 63 bytes will be forwarded only to the host on a best effort basis (meaning that the ALE may or may not be able to keep up with the packet rate and the short packet may be dropped due to bandwidth limitations). If RX_CSF_EN and RX_CEF_EN in

CPSW_PN_MAC_CONTROL_REG are set, fragment frames from 33 to 63 bytes will also be forwarded only to the host on a best effort basis. Ethernet port received frames shorter than 33 bytes are dropped in all cases.

A received long packet will always contain RX_MAXLEN number of bytes transferred to memory (if CPSW_PN_MAC_CONTROL_REG[22]RX_CEF_EN = 1h). An example with RX_MAXLEN = 1518 is:

- If the frame length is 1518, then the packet is not a long packet and there will be 1518 bytes transferred to memory.
- If the frame length is 1519, there will be 1518 bytes transferred to memory. The last three bytes will be the first three CRC bytes.
- If the frame length is 1520, there will be 1518 bytes transferred to memory. The last two bytes will be the first two CRC bytes.
- If the frame length is 1521, there will be 1518 bytes transferred to memory. The last byte will be the first CRC byte.

If the frame length is 1522, there will be 1518 bytes transferred to memory. The last byte will be the last data byte.

13.2.1.4.6.9.5 Receive FIFO Architecture

This section describes the architecture of the Ethernet port's receive FIFOs. Internal to the Gigabit Ethernet switch, all Ethernet ports have an identical associated packet FIFO. Each transmit packet FIFO contains eight logical transmit queues (priority 0 through 7 with 7 the highest priority). Each transmit FIFO memory contains 81,920 bytes total organized as 2560 by 256-bit words. Each FIFO also contains a single memory for the receive queue. Each receive FIFO memory contains a total of 32768 bytes total organized as 1024 by 256-bit words.

13.2.1.4.6.10 Embedded Memories

Table 13-147. Embedded Memories

Memory Type Description	Number of Instances	
Single-port 3072-word × 64 RAM	1	(Combined FIFO RAM)
Single-port 128-word × 28-bit RAM	1	1 (EST)

13.2.1.4.6.11 Memory Error Detection and Correction

The CPSW error detection and correction logic uses the ECC Aggregator Module.

The ECC CPSW_ECC_VECTOR register is used to select which ECC RAM's status and control registers are currently being read or written as shown in Table 13-148. The CPSW FIFO RAMs implement ECC only on packet headers. The packet data is protected by Ethernet CRC. The ALE and EST RAMs have complete ECC as normal.

Table 13-148. ECC RAM to CPSW RAM Mapping

ECC RAM Number	CPSW RAM
0	ALE RAM
1	Port 0 FIFO RAM

13.2.1.4.6.11.1 Packet Header ECC

Only packet headers bits are protected by ECC in the FIFO RAMs. The ECC_ERR_CTRL1[31-0] ECC_ROW bit is not implemented. ECC_ERR_CTRL2 [15-0] ECC_BIT1 is implemented to determine which bit of the header is flipped for an SEC error when the ECC_CRC_MODE bit is cleared in the CPSW_CONTROL_REG register. The ECC status registers return the RAM row address flipped (ECC_ROW) along with the ECC_BIT1 value. Forcing double-bit errors in testing can cause indeterminate operation if multiple used packet header bits are flipped given that only single-bit errors are fixed by the ECC logic. Header bits 207 down to 200 are not currently used in the CPSW and may be used to test double bit errors without the possibility of requiring a reset for the switch to recover from the double bit error. No header bits are flipped when ECC_CRC_MODE is set to 1h. Either the RX_ECC_ERR_EN (enable receive ECC error operations) or the TX_ECC_ERR_EN (enable transmit ECC error operations) bits must be set in the CPSW_P0_CONTROL_REG register to test ECC header errors.

The header ECC code is stored in bits 255 down to 208. If any bit is flipped in the ECC code, the flipped bit will be corrected, but the index of the flipped bit will be reported as bit zero. This implies that when the aggregator reports that there is a SEC on bit 0, it can mean two things: either SEC on data bit 0 or SEC somewhere inside the ECC code. Any packet header with ECC error issues a pulse interrupt (ECC_PULSE_INTR) as does an ALE RAM ECC error.

13.2.1.4.6.11.2 Packet Protect CRC

Each packet received without error is passed through the CPSW memories with a generated Ethernet protect CRC. The protect CRC is checked on egress for correctness and is replaced. If the CRC is correct (no RAM bit errors), then the packet is output with the selected port CRC type. If a protect CRC error is detected on host egress then the MEMORY+PROTECT_ERROR buffer descriptor bit will be asserted so that the packet is dropped to the host. If a protect CRC error is detected on Ethernet egress then the egress CRC will be generated on the packet and at least one byte of the CRC will be inverted on output. CRC memory protect errors do not assert the ECC_PULSE_INTR signal. CRC memory protect errors are counted in the associated port statistics registers and issue an interrupt on STAT_PEND_INTR if any CRC memory protect error occurs (and the statistics for that port are enabled). When the ECC_CRC_MODE bit in the CPSW_CONTROL_REG register is set, the ECC_ERR_CTRL2 [15-0] ECC_BIT1 bit field will flip the associated column bit in any FIFO memory read operation, inducing a CRC protect error when the protect CRC is checked. No header bits are flipped when ECC_CRC_MODE is set. Either the RX_ECC_ERR_EN or the TX_ECC_ERR_EN bits must be set in the CPSW_P0_CONTROL_REG register to test packet CRC errors.

13.2.1.4.6.11.3 Aggregator RAM Control

The ECC logic for each FIFO RAM (receive and transmit) is divided into eight separate ECC encoders/decoders that encode/decode 26-bits of data each. Each of the 8 encoders (0 to 7) generates 6-bits of ECC code (48 code bits total), and each of the eight decoders (0 to 7) checks 6-bits of ECC code across the 26-bits of data (208 data bits total). The 48-bits of ECC code are passed through the RAM in the upper 48 unused bits in the header word. The header data bits and ECC code bits are shown in Table 13-149. The [15-0] ECC_BIT1 value returned on error is a 16-bit value that is the concatenation of 5 bits of zero, 3 bits of the encoder/decoder number (0 to 7), 3 bits of zero, and 5 bits of index into the indicated 26-bit encoder/decoder.

For example, an ECC_BIT1 value of 0x0308 is bit 8 of encoder/decoder 3, which is header bit 86 (that is, (26×3) + 8).

Table 13-149. ECC Submodule Header Data Bit to Encoder/Decoder Mapping

Header Data Bits	Encoder/Decoder
25:0	Encoder/Decoder 0 Data
51:26	Encoder/Decoder 1 Data
77:52	Encoder/Decoder 2 Data
103:78	Encoder/Decoder 3 Data
129:104	Encoder/Decoder 4 Data
155:130	Encoder/Decoder 5 Data
181:156	Encoder/Decoder 6 Data
207:182	Encoder/Decoder 7 Data
213:208	Encoder/Decoder 0 ECC
219:214	Encoder/Decoder 1 ECC
225:220	Encoder/Decoder 2 ECC
231:226	Encoder/Decoder 3 ECC
237:232	Encoder/Decoder 4 ECC
243:238	Encoder/Decoder 5 ECC
249:244	Encoder/Decoder 6 ECC
255:250	Encoder/Decoder 7 ECC

13.2.1.4.6.12 Ethernet Port Flow Control

The Ethernet port have flow control available for transmit and receive. Transmit flow control stops the Ethernet port from transmitting packets to the wire (switch egress) in response to a received pause frame. Transmit flow control does not depend on FIFO usage.

The Ethernet port have flow control available for receive operations (packet ingress). Ethernet port receive flow control is initiated when enabled and triggered. Packets received on an Ethernet port can be sent to the CPPI port. The destination port can trigger the receive Ethernet port flow control. An Ethernet destination port triggers another Ethernet receive flow control when the destination port is full.

When a packet is received on an Ethernet port interface with enabled flow control the below occurs:

- The packet will be sent to all ports that currently have room to take the entire packet.
- The packet will be retried until successful to all ports that indicate they don't have room for the packet.

The flow control trigger to the Ethernet port will be asserted until the packet has been sent, and there is room in the logical receive FIFO for packet runout from another flow control trigger (RX_BLK_CNT = 0h). Ethernet port receive flow control is disabled by default on reset. Ethernet port receive flow control requires that the RX_FLOW_EN bit in CPSW_PN_MAC_CONTROL_REG be set to 1h. When receive flow control is enabled on a port, the port's associated FIFO block allocation must be adjusted. The port RX allocation must increase from the default three blocks to accommodate the flow control runout. A corresponding decrease in the TX block allocation is required. If a sending port ignores a pause frame then packets may overrun on receive (and be dropped) but will not be dropped on transmit.

13.2.1.4.6.12.1 Ethernet Receive Flow Control

For every Ethernet port to be configured for fullduplex receive flow control, write a value of decimal 7 to the CPSW_PN_MAX_BLKS_REG[7-0] RX_MAX_BLKS bit field, and a value of decimal 13 to the CPSW_PN_MAX_BLKS_REG[15-8] TX_MAX_BLKS register. This re-allocation allows for flow control runout on the receive FIFO at the expense of FIFO memory on the Ethernet transmit side. 10/100Mbps half-duplex collision based receive flow control does not need this re-allocation. Receive flow control is enabled by the RX_FLOW_EN bit in the CPSW_PN_MAC_CONTROL_REG register.

13.2.1.4.6.12.1.1 Collision Based Receive Buffer Flow Control

Collision-based receive buffer flow control provides a means of preventing frame reception when the port is operating in half-duplex mode (FULLDUPLEX is cleared in CPSW_PN_MAC_CONTROL_REG). When receive

flow control is enabled and triggered, the port will generate collisions for received frames. The jam sequence transmitted will be the twelve byte sequence C3.C3.C3.C3.C3.C3.C3.C3.C3.C3.C3.C3 (hex). The jam sequence will begin no later than approximately as the source address starts to be received. Note that these forced collisions will not be limited to a maximum of 16 consecutive collisions, and are independent of the normal back-off algorithm. Receive flow control does not depend on the value of the incoming frame destination address. A collision will be generated for any incoming packet, regardless of the destination address.

13.2.1.4.6.12.1.2 IEEE 802.3X Based Receive Flow Control

IEEE 802.3x based receive flow control provides a means of preventing frame reception when the port is operating in full-duplex mode (FULLDUPLEX bit is set in the CPSW_PN_MAC_CONTROL_REG register). When receive flow control is enabled and triggered, the port will transmit a pause frame to request that the sending station stop transmitting for the period indicated within the transmitted pause frame.

The Ethernet port will transmit a pause frame to the reserved multicast address at the first available opportunity (immediately if currently idle, or following the completion of the frame currently being transmitted). The pause frame will contain the maximum possible value for the pause time (FFFFh). The MAC will count the receive pause frame time (decrements FF00h down to 0) and retransmit an outgoing pause frame if the count reaches zero. When the flow control request is removed, the MAC will transmit a pause frame with a zero pause time to cancel the pause request.

Note that transmitted pause frames are only a request to the other end station to stop transmitting. Frames that are received during the pause interval will be received normally (provided the RX FIFO is not full at which time the receive FIFO will overrun and CPSW_STAT0_RX_BOTTOM_OF_FIFO_DROP/ CPSW_STAT1_RX_BOTTOM_OF_FIFO_DROP[31-0] COUNT value will increment).

Pause frames will be transmitted if enabled and triggered regardless of whether or not the port is observing the pause time period from an incoming pause frame.

The Ethernet port will transmit pause frames as described below:

- The 48-bit reserved multicast destination address 01.80.C2.00.00.01.
- The 48-bit source address - from SL_SA[47-0] input.
- The 16-bit length/type field containing the value 88.08
- The 16-bit pause opcode equal to 00.01
- The 16-bit pause time value FF.FF. A pause-quantum is 512 bit-times. Pause frames sent to cancel a pause request will have a pause time value of 00.00.
- Zero padding to 64-byte data length (The Ethernet port will transmit only 64 byte pause frames).
- The 32-bit frame-check sequence (CRC word).

All quantities above are hexadecimal and are transmitted most-significant byte first. The least-significant bit is transferred first in each byte.

If CPSW_PN_MAC_CONTROL_REG[3] RX_FLOW_EN is cleared to 0h while the pause time is nonzero, then the pause time will be cleared to 0h and a 0 count pause frame will be sent.

13.2.1.4.6.12.2 Flow Control Trigger

Receive flow control is triggered (when enabled), when the number of words in the receive FIFO is greater than or equal to the value written in the CPSW_PN_RX_FLOW_THRESH_REG[8-0] COUNT bit field. The flow control packet runout is then contained in the remainder of the receive FIFO.

13.2.1.4.6.12.3 Ethernet Transmit Flow Control

Incoming pause frames are acted upon, when enabled, to prevent the Ethernet port from transmitting any further frames. Incoming pause frames are only acted upon when the [0] FULLDUPLEX and [4] TX_FLOW_EN bits in the CPSW_PN_MAC_CONTROL_REG register are set. Pause frames are not acted upon in half-duplex mode. Pause frame action will be taken if enabled, but normally the frame will be filtered and not transferred to memory. MAC control frames will be transferred to memory if the [24] RX_CMF_EN (RX Copy MAC Control Frames Enable) bit in the CPSW_PN_MAC_CONTROL_REG register is set. The [4] TX_FLOW_EN and [0] FULLDUPLEX bits effect whether or not MAC control frames are acted upon, but they have no effect upon whether or not MAC control frames are transferred to memory or filtered.

Pause frames are a subset of MAC Control Frames with an opcode field = 0001h. Incoming pause frames will only be acted upon by the port if:

- [4] TX_FLOW_EN is set in CPSW_PN_MAC_CONTROL_REG register, and
- the RX maximum frame length is 64 bytes inclusive (CPSW_PN_RX_MAXLEN_REG[13-0] RX_MAXLEN), and
- the frame contains no CRC error or align/code errors.

The pause time value from valid frames will be extracted from the two bytes following the opcode. The pause time will be loaded into the port's transmit pause timer and the transmit pause time period will begin.

If a valid pause frame is received during the transmit pause time period of a previous transmit pause frame then:

- if the destination address is not equal to the reserved multicast address or any enabled or disabled unicast address, then the transmit pause timer will immediately expire, or
- if the new pause time value is zero then the transmit pause timer will immediately expire, else the port transmit pause timer will immediately be set to the new pause frame pause time value. (Any remaining pause time from the previous pause frame will be discarded).

If [4] TX_FLOW_EN in CPSW_PN_MAC_CONTROL_REG register is cleared, then the pause-timer will immediately expire.

The port will not start the transmission of a new data frame any sooner than 512-bit times after a pause frame with a non-zero pause time has finished being received (MRXDV going inactive). No transmission will begin until the pause timer has expired (the port may transmit pause frames in order to initiate outgoing flow control). Any frame already in transmission when a pause frame is received will be completed and unaffected.

Incoming pause frames consist of the below:

- A 48-bit destination address equal to:
  - The reserved multicast destination address 01.80.C2.00.00.01, or the Ethernet port SL_SA [47:0] input.
- The 48-bit source address of the transmitting device.
- The 16-bit length/type field containing the value 88.08
- The 16-bit pause opcode equal to 00.01
- The 16-bit CPSW_PN_MAC_TX_PAUSETIMER_REG[15-0] TX_PAUSETIMER. A pause-quantum is 512 bit-times.
- Padding to 64-byte data length.
- The 32-bit frame-check sequence (CRC word).

All quantities above are hexadecimal and are transmitted most-significant byte first. The least-significant bit is transferred first in each byte.

The padding is required to make up the frame to a minimum of 64 Bytes. The standard allows pause frames longer than 64 Bytes to be discarded or interpreted as valid pause frames. The Ethernet port will recognize any pause frame between 64 Bytes and CPSW_PN_RX_MAXLEN_REG[13-0] RX_MAXLEN bytes in length.

13.2.1.4.6.13 Energy Efficient Ethernet Support (802.3az)

Energy Efficient Ethernet (EEE) allows the LPSC to turn off the module clock during inactive periods as determined by network and host traffic. The module can then be awakened by host queued transmit packet(s) or by a port's external Ethernet PHY. The module EEE clock stop interface is used by the external controller to control module EEE operations. EEE operations are configured as shown below:

1. The 12-bit EEE clock pre-scale value is written to the CPSW_EEE_PRESCALE_REG register. The pre-scaler is used to clock all EEE-related counters
2. The port Idle to LPI count values (CPSW_PN_IDLE2LPI_REG[23-0] COUNT) are written with the desired values
3. The port LPI to Wake count values (CPSW_PN_LPI2WAKE_REG[23-0] COUNT) are written with the desired values
4. The [0] EEE_EN bit is set in the switch CPSW_SS_CONTROL_REG register

EEE operation can begin after configuration. The host allows (through LPSC) the CPSW to enter a low power state by asserting the EEE_CLKSTOP_REQ signal. There are no requirements on host queues or traffic in order for the host to assert or de-assert EEE_CLKSTOP_REQ to the CPSW.

Each Ethernet port has a transmit and a receive LPI (low power indicate) state. The PHY indicates LPI by asserting MRXER with a MRXD[7:0] value of 0x01 while MRXDV is deasserted (inter-packet gap). The Ethernet transmit port indicates LPI after the CPSW_PN_IDLE2LPI_REG value has been counted (the transmit port has gone idle for the configured amount of time). If another packet is received for transmit during the count then the count is restarted. When the transmit port has been idle for the Idle to LPI time, the transmit port enters the LPI state and indicates LPI to the associated PHY. The LPI is indicated to the external PHY by an asserted MTXER with a MTXD[7:0] while MTXEN is deasserted (inter-packet gap). The CPPI (port 0) LPI state includes transmit and receive. The CPPI LPI state is entered when the CPPI transmit and receive have both been idle for the Idle to LPI time (CPSW_P0_IDLE2LPI_REG). The Idle to LPI time value for all ports must be large relative to the switch latency to ensure that the count is not able to complete between successive packets.

Note

External PHY signaling has the following conditions:

- RGMII is a DDR interface. TXEN and TXER are the sampled values of TX_CTL at the rising and the falling TXC edges, respectively. RXDV and RXER are the sampled values of RX_CTL at the rising and the falling RXC clock edges, respectively
- In RMII mode, EEE is not supported.

When all transmit and receive ports are in the LPI state (CPSW LPI state), the EEE_CLKSTOP_ACK signal is asserted, and the LPSC is allowed to stop the module clock. When EEE_CLKSTOP_ACK is asserted, the clock may be turned on and off as desired by the host. The host is allowed to restart the clock, perform target read/write operations to the CPSW memory address space, and then turn off the clock again while EEE_CLKSTOP_ACK is asserted.

The software can remove and disable from re-entering the CPSW LPI state by restarting the module clock and then de-asserting EEE_CLKSTOP_REQ. There must be at least one rising edge of the clock before EEE_CLKSTOP_REQ is de-asserted. The module EEE_CLKSTOP_ACK output signal will be deasserted on the clock after the de-assertion of EEE_CLKSTOP_REQ. The host may queue CPPI receive packets at any time without regard to the CPSW module LPI state. The Host must deassert EEE_CLKSTOP_REQ on wakeup for a minimum of two clock periods. If EEE_CLKSTOP_REQ is deasserted for less than 5 clock periods for a wakeup event from the host to a particular Ethernet port (or visa versa), then the wakeup event will not cause the other Ethernet port to awaken.

The external Ethernet PHY's can also wakeup the LPSC by removing the Ethernet receive LPI indication. If the CPSW module is in Idle state with EEE_CLKSTOP_ACK asserted and the receive LPI indication is removed, the EEE_CLKSTOP_WAKEUP signal will be asynchronously asserted. On wakeup, the LPSC restarts the clock and de-assert the EEE_CLKSTOP_REQ signal. The EEE_CLKSTOP_WAKEUP signal will be synchronously deasserted with EEE_CLKSTOP_ACK. Upon the deassertion of EEE_CLKSTOP_REQ, the Ethernet ports will count the CPSW_PN_LPI2WAKE_REG time for each port at which time the port is available for transmit.

13.2.1.4.6.14 Ethernet Switch Latency

When CPSW is configured as a store and forward switch, the switch latency is defined as the amount of time between the end of packet reception of the received packet to the start of the output packet transmit.

The store and forward latency is shown in Table 13-150:

Table 13-150. Switch Latency

Mode	Latency
Gig (1000)	880 ns
100	1.3 µs
10	6.5 µs

13.2.1.4.6.15 MAC Emulation Control

The emulation control input (EMUSUSP) and submodule emulation control registers allow CPSW operation to be completely or partially suspended. Each Ethernet port has associated emulation control registers (CPSW_EM_CONTROL_REG and CPSW_PN_MAC_EMCONTROL_REG). The submodule emulation control registers must be accessed to facilitate CPSW emulation control. The CPSW module enters the emulation suspend state if all three submodules are configured for emulation suspend and the emulation suspend input is asserted. A partial emulation suspend state is entered if one or two submodules is configured for emulation suspend and the emulation suspend input is asserted. Emulation suspend occurs at packet boundaries. The emulation control feature is implemented for compatibility with other peripherals.

Ethernet port Emulation Control

The emulation control input (TBEMUSUP) and register bits (SOFT and FREE bits in the CPSW_PN_MAC_EMCONTROL_REG register) allow Ethernet port operation to be suspended. When the emulation suspend state is entered, the Ethernet port will stop processing receive and transmit frames at the next frame boundary. Any frame currently in reception or transmission will be completed normally without suspension. For receive, frames that are detected by the Ethernet port after the suspend state is entered are ignored.

Table 13-151 shows the operations of the emulation control input and register bits.

Table 13-151. Emulation Control Input

EMUSUSP	SOFT	FREE	Description
0	X	X	Normal Operation
1	0	0	Normal Operation
1	1	0	Emulation Suspend
1	X	1	Normal Operation

13.2.1.4.6.16 MAC Command IDLE

The CMD_IDLE bit in the CPSW_PN_MAC_CONTROL_REG register allows MAC operation to be suspended. When the idle state is commanded, the MAC will stop processing receive and transmit frames at the next frame boundary. Any frame currently in reception or transmission will be completed normally without suspension. For transmission, any complete or partial frame in the TX cell FIFO will be transmitted. For receive, frames that are detected by the MAC after the suspend state is entered are ignored. No statistics will be kept for ignored frames. Commanded idle is similar in operation to emulation control and clock stop.

13.2.1.4.6.17 CPSW Network Statistics

The CPSW has a set of statistics that record events associated with frame traffic on selected switch ports. The statistics values are cleared to zero 38 clocks after the rising edge of CPSW0_RST. When one or more port enable (Pn_STAT_EN) bits in the CPSW_STAT_PORT_EN_REG register are set, all statistics registers are write to decrement. The value written will be subtracted from the register value with the result being stored in the register. If a value greater than the statistics value is written, then zero will be written to the register (writing 0xFFFF FFFF clears a statistics location). When all port enable bits are cleared to zero, all statistics registers are read/write (normal write direct, so writing 0x0000 0000 clears a statistics location). All write accesses must be 32-bit accesses.

The statistics interrupt (STAT_PEND0) will be issued if enabled when any statistics value is greater than or equal to 0x8000 0000. The statistics interrupt is removed by writing to decrement any statistics value greater than 0x8000 0000. The statistics are mapped into internal memory space and are 32-bits wide. All statistics rollover from 0xFFFF FFFF to 0x0000 0000.

Table 13-152 and Table 13-153 summarize network statistics.

13.2.1.4.6.17.1 Rx-only Statistics Descriptions

13.2.1.4.6.17.1.1 Good Rx Frames (Offset = 3A000h)

All ports

The total number of good frames received on the port. A good frame is defined to be:

- Any data or MAC control frame which matched a unicast, broadcast or multicast address, or matched due to promiscuous mode
- Had a length of 64 to RX_MAXLEN bytes inclusive
- Had no CRC error, alignment error or code error.

See the Section 13.2.1.4.6.17.1.6, *Rx Align/Code Errors* and Section 13.2.1.4.6.17.1.5, *Rx CRC errors* statistic descriptions for definitions of alignment, code and CRC errors.

Overruns have no effect upon this statistic.

13.2.1.4.6.17.1.2 Broadcast Rx Frames (Offset = 3A004h)

All ports

The total number of good broadcast frames received on the port. A good broadcast frame is defined to be:

- Any data or MAC control frame which was destined for address FF.FF.FF.FF.FF.FF
- Had a length of 64 to RX_MAXLEN bytes inclusive
- Had no CRC error, alignment error or code error.

See the Section 13.2.1.4.6.17.1.6, *Rx Align/Code Errors* and Section 13.2.1.4.6.17.1.5, *Rx CRC errors* statistic descriptions for definitions of alignment, code and CRC errors.

Overruns have no effect upon this statistic.

13.2.1.4.6.17.1.3 Multicast Rx Frames (Offset = 3A008h)

All ports

The total number of good multicast frames received on the port. A good multicast frame is defined to be:

- Any data or MAC control frame which was destined for any multicast address other than FF.FF.FF.FF.FF.FF
- Had a length of 64 to RX_MAXLEN bytes inclusive
- Had no CRC error, alignment error or code error

See the Section 13.2.1.4.6.17.1.6, *Rx Align/Code Errors* and Section 13.2.1.4.6.17.1.5, *Rx CRC errors* statistic descriptions for definitions of alignment, code and CRC errors.

Overruns have no effect upon this statistic.

13.2.1.4.6.17.1.4 Pause Rx Frames (Offset = 3A00Ch)

Ethernet port N where N = 1 to 2

The total number of IEEE 802.3X pause frames received by the port (whether acted upon or not). Such a frame:

- Contained any unicast, broadcast, or multicast address
- Contained the length/type field value 88.08 (hex) and the opcode 0x0001
- Was of length 64 to RX_MAXLEN bytes inclusive
- Had no CRC error, alignment error or code error
- Pause-frames had been enabled on the port (TX_FLOW_EN = 1h).

The port could have been in either half or full-duplex mode.

See the Section 13.2.1.4.6.17.1.6, *Rx Align/Code Errors* and Section 13.2.1.4.6.17.1.5, *Rx CRC errors* statistic descriptions for definitions of alignment, code and CRC errors.

Overruns have no effect upon this statistic.

13.2.1.4.6.17.1.5 Rx CRC Errors (Offset = 3A010h)

All ports

The total number of frames received on the port that experienced a CRC error. Such a frame:

- Was any data or MAC control frame which matched a unicast, broadcast or multicast address, or matched due to promiscuous mode
- Was of length 64 to RX_MAXLEN bytes inclusive
- Had no code/align error

- Had a CRC error

Overruns have no effect upon this statistic.

A CRC error is defined to be:

- A frame containing an even number of nibbles, and
- Failing the Frame Check Sequence test.

13.2.1.4.6.17.1.6 Rx Align/Code Errors (Offset = 3A014h)

Ethernet port N where N = 1 to 2

The total number of frames received on the port that experienced an alignment error or code error. Such a frame:

- Was any data or MAC control frame which matched a unicast, broadcast or multicast address, or matched due to promiscuous mode
- Was of length 64 to RX_MAXLEN bytes inclusive
- Had either an alignment error, or a code error.

Overruns have no effect upon this statistic.

An alignment error is defined to be:

- A frame containing an odd number of nibbles
- Failing the Frame Check Sequence test if the final nibble is ignored

A code error is defined to be a frame which has been discarded because the port's MRXER pin driven with a one for at least one bit-time's duration at any point during the frame's reception.

Note

RFC 1757 etherStatsCRCAlignErrors Ref. 1.5 can be calculated by summing Rx Align/Code Errors and Rx CRC errors.

13.2.1.4.6.17.1.7 Oversize Rx Frames (Offset = 3A018h)

All ports

The total number of oversized frames received on the port. An oversized frame is defined to be:

- Was any data or MAC control frame which matched a unicast, broadcast or multicast address, or matched due to promiscuous mode
- Was greater than RX_MAXLEN in bytes
- Had no CRC error, alignment error, or code error.

See the Section 13.2.1.4.6.17.1.6, *Rx Align/Code Errors* and Section 13.2.1.4.6.17.1.5, *Rx CRC errors* statistic descriptions for definitions of alignment, code and CRC errors.

Overruns have no effect upon this statistic.

13.2.1.4.6.17.1.8 Rx Jabbers (Offset = 3A01Ch)

All ports

The total number of jabber frames received on the port. A jabber frame:

- Was any data or MAC control frame which matched a unicast, broadcast or multicast address, or matched due to promiscuous mode
- Was greater than RX_MAXLEN in bytes
- Had a CRC error, alignment error or code error

See the Section 13.2.1.4.6.17.1.6, *Rx Align/Code Errors* and Section 13.2.1.4.6.17.1.5, *Rx CRC errors* statistic descriptions for definitions of alignment, code and CRC errors.

Overruns have no effect upon this statistic.

13.2.1.4.6.17.1.9 Undersize (Short) Rx Frames (Offset = 3A020h)

All ports

The total number of undersized frames received on the port. An undersized frame is defined to be:

- Was any data frame which matched a unicast, broadcast or multicast address, or matched due to promiscuous mode
- Was less than 64 bytes
- Had no CRC error, alignment error, or code error

See the Section 13.2.1.4.6.17.1.6, *Rx Align/Code Errors* and Section 13.2.1.4.6.17.1.5, *Rx CRC errors* statistic descriptions for definitions of alignment, code and CRC errors.

Overruns have no effect upon this statistic.

13.2.1.4.6.17.1.10 Rx Fragments (Offset = 3A024h)

Ethernet port N where N = 1 to 2

The total number of frame fragments received on the port. A frame fragment is defined to be:

- Any data frame (address matching does not matter)
- Less than 64 bytes long
- Having a CRC error, an alignment error, or a code error
- Not the result of a collision caused by half-duplex, collision-based flow control

See the Section 13.2.1.4.6.17.1.6, *Rx Align/Code Errors* and Section 13.2.1.4.6.17.1.5, *Rx CRC errors* statistic descriptions for definitions of alignment, code and CRC errors.

Overruns have no effect upon this statistic.

13.2.1.4.6.17.1.11 RX IPG Error (Offset = 3A05Ch)

The total number of 10G frames received on a port that had a correct preamble but did not have at least five bytes of IDLE preceding the frame. This does not indicate if the frame with the IPG error was kept or ignored.

13.2.1.4.6.17.1.12 ALE Drop (Offset = 3A028h)

All ports

The total number of frames received on a port such that the destination address was not equal to the source address and the packet was not destined to the port it was received on, but the frame was not forwarded to any port (the PORT_MASK was zero).

- Any data or MAC control frame which matched a unicast, broadcast or multicast address, or matched due to promiscuous mode
- Any length (including less than 64 bytes and greater than RX_MAXLEN bytes)
- had no CRC error, alignment error, or code error
- the destination address was not equal to the source address
- the packet was not destined for the port it was receive on
- had a zero PORT_MASK

13.2.1.4.6.17.1.13 ALE Overrun Drop (Offset = 3A02Ch)

All ports (non cut-thru mode)

The total number of frames received on a port that were dropped (zero PORT_MASK) due to exceeding the maximum ALE lookup rate (Port 0 should not have ALE Overrun Drops because the ingress rate is controlled to prevent it). This statistic should be zero and when non-zero indicates a system clock issue or indicates that short packets were sent with RX_CSF_EN at a rate that exceeded the maximum lookup rate.

- Any data or MAC control frame which matched a unicast, broadcast or multicast address, or matched due to promiscuous mode
- Any length (including less than 64 bytes and greater than RX_MAXLEN bytes)
- The port has no receive priorities enabled for cut-thru, and
- the maximum ALE lookup rate was exceeded so the lookup was aborted and the packet was dropped.

13.2.1.4.6.17.1.14 Rx Octets (Offset = 3A030h)

All ports

The total number of bytes in all good frames received on the port. A good frame is defined to be:

- Any data or MAC control frame which matched a unicast, broadcast or multicast address, or matched due to promiscuous mode
- Of length 64 to RX_MAXLEN bytes inclusive
- Had no CRC error, alignment error or code error

See the Section 13.2.1.4.6.17.1.6, *Rx Align/Code Errors* and Section 13.2.1.4.6.17.1.5, *Rx CRC errors* statistic descriptions for definitions of alignment, code and CRC errors.

Overruns have no effect upon this statistic.

13.2.1.4.6.17.1.15 Rx Bottom of FIFO Drop (Offset = 3A084h)

Ethernet port N where N = 1 to 2

The total number of frames received on a port that overran the port's receive FIFO and were dropped (bottom of receive FIFO). Port 0 (CPPI receive port) should not drop packets on receive because port 0 receive flow control should be enabled. The Ethernet ports will only drop packets in the receive FIFO when receive flow control is enabled and the sending port ignores sent pause frame and then overruns the receive FIFO. An overrun frame is defined to be:

- Any data or MAC control frame which matched a unicast, broadcast or multicast address, or matched due to promiscuous mode
- Any length (including less than 64 bytes and greater than RX_MAXLEN bytes)
- Was dropped on port 0 due to a lack of memory space in the receive FIFO.

Note

This statistic should be zero if proper flow control is being followed.

Host port 0

This statistic also counts frames dropped on port 0 that were 17 to 33 bytes (only for port 0). For Ethernet ports, the drop count for frames shorter than 33 bytes is included in the undersized or fragment count. Port 0 simply gives an indication that a packet was dropped. No other statistics are counted for frames shorter than 33 bytes.

13.2.1.4.6.17.1.16 Portmask Drop (Offset = 3A088h)

All ports

The total number of frames received on a port that were dropped by the ALE (the ALE did not forward the packet to any port). Port mask drop frame is defined to be:

- Any data or MAC control frame
- Any length greater than 32 bytes
- Was dropped by the ALE due to PORT_MASK=0 (was not sent to any destination port)
- The frame could have been dropped due to error or other counted reason, so it could be counted elsewhere also.

Note

This statistic does not count in the overall total as it includes every packet received greater than 32 bytes that had a zero PORT_MASK.

13.2.1.4.6.17.1.17 Rx Top of FIFO Drop (Offset = 3A08Ch)

All ports

The total number of frames received on a port that had a start-of-frame (SOF) overrun on any destination port egress (when attempting to load the packet from the top of the ingress port receive FIFO into any other port's

transmit FIFO). If a multicast/broadcast packet is dropped by multiple destination ports then this statistic will increment by the number of ports that dropped the packet. Rx Top Of FIFO Drop is defined to be:

- Any data or MAC control frame which matched a unicast, broadcast or multicast address, or matched due to promiscuous mode
- Any length (including less than 64 bytes and greater than RX_MAXLEN bytes)
- had no CRC error, alignment error or code error
- had a SOF of frame overrun on another port egress.

13.2.1.4.6.17.1.18 ALE Rate Limit Drop (Offset = 3A090h)

All ports

The total number of frames received on a port that were dropped (zero PORT_MASK) due to receive rate limiting on this port or due to transmit rate limiting on any destination port (not sent to all expected destination ports if transmit rate limiting).

- Any data or MAC control frame which matched a unicast, broadcast or multicast address, or matched due to promiscuous mode
- Any length (including less than 64 bytes and greater than RX_MAXLEN bytes)
- had no CRC error, alignment error, or code error
- the receive rate was exceeded and the packet was dropped, or the transmit rate was exceeded to any destination port and the packet was dropped to one or more expected destination ports (indicates that the destinations were reduced due to rate limiting).

13.2.1.4.6.17.1.19 ALE VLAN Ingress Check Drop (Offset = 3A094h)

All ports

The total number of frames received on a port that were dropped (zero PORT_MASK) due to VLAN ingress check failure.

- Any data or MAC control frame which matched a unicast, broadcast or multicast address, or matched due to promiscuous mode
- Any length (including less than 64 bytes and greater than RX_MAXLEN bytes)
- had no CRC error, alignment error, or code error
- the VLAN ID ingress check failed (the receive port was not in the group)
- The address lookup did not return a match with the SUPER bit set.

13.2.1.4.6.17.1.19.1 ALE DA=SA Drop (Offset = 3A098h)

All ports

The total number of frames received on a port that were dropped (zero PORT_MASK) due to destination address equal to source address.

- Any data or MAC control frame which matched a unicast, broadcast or multicast address, or matched due to promiscuous mode
- Any length (including less than 64 bytes and greater than RX_MAXLEN bytes)
- had no CRC error, alignment error, or code error
- the destination address was equal to the source address
- the source address was not an entry in the table.

13.2.1.4.6.17.1.19.2 Block Address Drop (Offset = 3A09Ch)

The total number of frames received on a port that were dropped (zero PORT_MASK) due to the destination or source address being blocked.

- was any data or MAC control frame which matched a unicast, broadcast or multicast address, or matched due to promiscuous mode, and
- Any length (including less than 64 bytes and greater than RX_MAXLEN bytes)
- had no CRC error, alignment error, or code error, and
- the source or destination address matched a table entry with the block bit set.

13.2.1.4.6.17.1.19.3 ALE Secure Drop (Offset = 3A0A0h)

The total number of frames received on a port that were dropped (zero port_mask) due to a secure violation (the source address is owned by a different receive port).

- was any data or MAC control frame which matched a unicast, broadcast or multicast address, or matched due to promiscuous mode, and
- Any length (including less than 64 bytes and greater than RX_MAXLEN bytes)
- had no CRC error, alignment error, or code error, and
- the source address is an entry in the table with the SECURE bit set and a port number for a different receive port.

13.2.1.4.6.17.1.19.4 ALE Authentication Drop (Offset = 3A0A4h)

The total number of frames received on a port that were dropped (zero port_mask) due to authentication failure.

- was any data or MAC control frame which matched a unicast, broadcast or multicast address, or matched due to promiscuous mode, and
- Any length (including less than 64 bytes and greater than RX_MAXLEN bytes)
- had no CRC error, alignment error, or code error, and
- CPSW_ALE_CONTROL[1] ENABLE_AUTH_MODE is set to 1h, and
- the source address is not equal to the destination address, and
- the source address is not a table entry, and
- the destination address is not a table entry with the SUPER bit set.

13.2.1.4.6.17.1.19.5 ALE Unknown Unicast (Offset = 3A0A8h)

All ports

The total number of frames received on a port that had a unicast destination address with an unknown source address.

- was any data frame with a unicast destination address
- the source address was not a table entry
- was of length 64 to RX_MAXLEN bytes inclusive
- had no CRC error, alignment error, or code error

Note: The ALE Unknown Unicast Bytecount statistic is the number of bytes contained in the ALE Unknown Unicast frames.

13.2.1.4.6.17.1.19.6 ALE Unknown Unicast Bytecount (Offset = 3A0ACh)

The total number of bytes received on a port that had a unicast destination address with an unknown source address.

13.2.1.4.6.17.1.19.7 ALE Unknown Multicast (Offset = 3A0B0h)

The total number of frames received on a port that had a multicast destination address with an unknown source address. The frame is defined to be:

- was any data frame with a multicast destination address
- the source address was not a table entry, and
- was of length 64 to RX_MAXLEN bytes inclusive
- had no CRC error, alignment error or code error

Note: The ALE Unknown Multicast Bytecount statistic is the number of bytes contained in the ALE Unknown Multicast frames.

13.2.1.4.6.17.1.19.8 ALE Unknown Multicast Bytecount (Offset = 3A0B4h)

The total number of bytes received on a port that had a multicast destination address with an unknown source address.

13.2.1.4.6.17.1.19.9 ALE Unknown Broadcast (Offset = 3A0B8h)

The total number of frames received on a port that had a broadcast destination address with an unknown source address. The frame is defined to be:

- was any data frame with a broadcast destination address
- the source address was not a table entry, and
- was of length 64 to RX_MAXLEN bytes inclusive
- had no CRC error, alignment error or code error

Note: The ALE Unknown Broadcast Bytecount statistic is the number of bytes contained in the ALE Unknown Broadcast frames.

13.2.1.4.6.17.1.19.10 ALE Unknown Broadcast Bytecount (Offset = 3A0BCh)

The total number of bytes received on a port that had a broadcast destination address with an unknown source address.

13.2.1.4.6.17.1.19.11 ALE Policer Match (Offset = 3A0C0h)

All ports

The total number of frames received on a port that matched a policer. The frame is defined to be:

- was any data frame
- matched a condition on a policer,
- was of length 64 to RX_MAXLEN bytes inclusive
- had no CRC error, alignment error, or code error

13.2.1.4.6.17.1.19.12 ALE Policer Match Red (Offset = 3A0C4h)

The total number of frames received on a port that had matched a policer and the condition was red. The frame is defined to be:

- was any data frame
- matched the condition red on a policer,
- was of length 64 to RX_MAXLEN bytes inclusive
- had no CRC error, alignment error, or code error

13.2.1.4.6.17.1.19.13 ALE Policer Match Yellow (Offset = 3A0C8h)

The total number of frames received on a port that had matched a policer and the condition was yellow. The frame is defined to be:

- was any data frame
- matched the condition yellow on a policer,
- was of length 64 to RX_MAXLEN bytes inclusive
- had no CRC error, alignment error, or code error

13.2.1.4.6.17.2 Rx Cut Thru with No Delay

The total number of frames received on the port and subsequently sent cut-thru to all internal destination port FIFOs with no delay. Such a frame:

- was any data or MAC control frame destined for any unicast, broadcast or multicast address, and
- was any size greater than or equal to 128 bytes, and
- was sent cut-thru by the receive port to all destination ports without delay.

13.2.1.4.6.17.3 Rx Cut Thru with Delay

The total number of frames received on the port and subsequently sent cut-thru to all destination ports with some amount of delay. Such a frame:

- was any data or MAC control frame destined for any unicast, broadcast or multicast address, and
- was any size greater than or equal to 128 bytes, and
- was sent cut-thru by the receive port to all destination ports with some amount of delay.

13.2.1.4.6.17.4 Rx Cut Thru Store-and-Forward

The total number of frames received on the port were attempted to be sent cut-thru to all destination ports but were held off due to configuration or traffic conditions. Such a frame:

- was any data or MAC control frame destined for any unicast, broadcast or multicast address, and
- was any size greater than or equal to 128 bytes, and
- was attempted to be sent cut-thru by the receive port to all destination ports but was sent store-and-forward.

13.2.1.4.6.17.5 Tx-only Statistics Descriptions

The maximum and minimum transmit frame size is software controllable.

Transmit overruns have no effect on TX statistics. They are counted separately.

13.2.1.4.6.17.5.1 Good Tx Frames (Offset = 3A034h)

All ports

The total number of good frames transmitted on the port. A good frame is defined to be:

- Any data or MAC control frame which matched a unicast, broadcast or multicast address
- Any length
- Had no late or excessive collisions, no carrier loss and no underrun

13.2.1.4.6.17.5.2 Broadcast Tx Frames (Offset = 3A038h)

All ports

The total number of good broadcast frames transmitted on the port. A good broadcast frame is defined to be:

- Any data or MAC control frame which was destined for only address FF.FF.FF.FF.FF.FF
- Any length
- Had no late or excessive collisions, no carrier loss and no underrun

13.2.1.4.6.17.5.3 Multicast Tx Frames (Offset = 3A03Ch)

All ports

The total number of good multicast frames transmitted on the port. A good multicast frame is defined to be:

- Any data or MAC control frame which was destined for any multicast address other than FF.FF.FF.FF.FF.FF
- Any length
- Had no late or excessive collisions, no carrier loss and no underrun

13.2.1.4.6.17.5.4 Pause Tx Frames (Offset = 3A040h)

Ethernet port N where N = 1 to 2

This statistic indicates the number of IEEE 802.3X pause frames transmitted by the port.

Pause frames cannot contain a CRC error because they are created in the transmitting MAC, so these error conditions have no effect upon the statistic. Pause frames sent by software will not be included in this count.

Since pause frames are only transmitted in full duplex, carrier loss and collisions have no effect upon this statistic.

Transmitted pause frames are always 64-byte multicast frames so will appear in the *Tx Multicast Frames* and *64octet Frames* statistics.

13.2.1.4.6.17.5.5 Deferred Tx Frames (Offset = 3A044h)

Ethernet port N where N = 1 to 2

The total number of frames transmitted on the port that first experienced deferment. Such a frame:

- Was any data or MAC control frame destined for any unicast, broadcast or multicast address
- Was any size
- Had no carrier loss and no underrun
- Experienced no collisions before being successfully transmitted

- Found the medium busy when transmission was first attempted, so had to wait.

CRC errors have no effect upon this statistic.

13.2.1.4.6.17.5.6 Collisions (Offset = 3A048h)

Ethernet port N where N = 1 to 2

This statistic records the total number of times that the port experienced a collision. Collisions occur under two circumstances.

1. When a transmit data or MAC control frame:
   - Was destined for any unicast, broadcast or multicast address
   - Was any size
   - Had no carrier loss and no underrun
   - Experienced a collision. A jam sequence is sent for every non-late collision, so this statistic will increment on each occasion if a frame experiences multiple collisions (and increments on late collisions)

   CRC errors have no effect upon this statistic.
2. When the port is in half-duplex mode, flow control is active, and a frame reception begins.

13.2.1.4.6.17.5.7 Single Collision Tx Frames (Offset = 3A04Ch)

Ethernet port N where N = 1 to 2

The total number of frames transmitted on the port that experienced exactly one collision. Such a frame:

- Was any data or MAC control frame destined for any unicast, broadcast or multicast address
- Was any size
- Had no carrier loss and no underrun
- Experienced one collision before successful transmission. The collision was not late.

CRC errors have no effect upon this statistic.

13.2.1.4.6.17.5.8 Multiple Collision Tx Frames (Offset = 3A050h)

Ethernet port N where N = 1 to 2

The total number of frames transmitted on the port that experienced multiple collisions. Such a frame:

- Was any data or MAC control frame destined for any unicast, broadcast or multicast address
- Was any size
- Had no carrier loss and no underrun
- Experienced 2 to 15 collisions before being successfully transmitted. None of the collisions were late.

CRC errors have no effect upon this statistic.

13.2.1.4.6.17.5.9 Excessive Collisions (Offset = 3A054h)

Ethernet port N where N = 1 to 2

The total number of frames for which transmission was abandoned due to excessive collisions. Such a frame:

- Was any data or MAC control frame destined for any unicast, broadcast or multicast address
- Was any size
- Had no carrier loss and no underrun
- Experienced 16 collisions before abandoning all attempts at transmitting the frame. None of the collisions were late.

CRC errors have no effect upon this statistic.

13.2.1.4.6.17.5.10 Late Collisions (Offset = 3A058h)

Ethernet port N where N = 1 to 2

The total number of frames on the port for which transmission was abandoned because they experienced a late collision. Such a frame:

- Was any data or MAC control frame destined for any unicast, broadcast or multicast address

- Was any size
- Experienced a collision later than 512 bit-times into the transmission. There may have been up to 15 previous (non-late) collisions which had previously required the transmission to be re-attempted. The *Late Collisions* statistic dominates over the single-, multiple- , and excessive- collision statistics. If a late collision occurs, the frame will not be counted in any of these other three statistics.

CRC errors, carrier loss, and underrun have no effect upon this statistic.

13.2.1.4.6.17.5.11 Carrier Sense Errors (Offset = 3A060h)

Ethernet port N where N = 1 to 2

The total number of frames on the port that experienced carrier loss. Such a frame:

- Was any data or MAC control frame destined for any unicast, broadcast or multicast address
- Was any size
- The carrier sense condition was lost or never asserted when transmitting the frame (the frame is not retransmitted). This is a transmit only statistic. Carrier Sense is a don't care for received frames. Transmit frames with carrier sense errors are sent until completion and are not aborted.

CRC errors and underrun have no effect upon this statistic.

13.2.1.4.6.17.5.12 Tx Octets (Offset = 3A064h)

All ports

The total number of bytes in all good frames transmitted on the port. A good frame is defined to be:

- Any data or MAC control frame which was destined for any unicast, broadcast or multicast address
- Was any size
- Had no late or excessive collisions, no carrier loss and no underrun.

13.2.1.4.6.17.5.13 Transmit Priority 0-7 (Offset = 3A180h to 3A1A8h)

The total number of frames transmitted on the port from transmit FIFO priority 0-7. Collision retries do not affect this statistic. Pause frames do not affect this statistic.

- Any frame transmitted from priority 0-7, and
- Was less than or equal to CPSW_TX_PRI0_MAXLEN_REG to CPSW_TX_PRI7_MAXLEN_REG
- Collision retries are not counted in this statistic.
- Pause frames are not counted in this statistic.
- Carrier sense errors do not affect this statistic.

Note: The Transmit Priority 0-7 Bytecount statistic is the number of bytes contained in the frames of the Transmit Priority 0-7 statistic.

13.2.1.4.6.17.5.14 Transmit Priority 0-7 Drop (Offset = 3A1C0h to 3A1E8)

The total number of transmit frames on the port that overran the transmit FIFO priority 0-7 and were dropped. This count includes frames dropped due to CPSW_TX_PRI0_MAXLEN_REG to CPSW_TX_PRI7_MAXLEN_REG.

- Any frame destined to be transmitted from priority 0-7, and
- Was any size, and
- Was dropped due to priority 0-7 FIFO overrun (Start of packet overrun).
- Was dropped due to frame size larger than CPSW_TX_PRI0_MAXLEN_REG to CPSW_TX_PRI7_MAXLEN_REG.

Note: The Transmit Priority 0-7 Drop Bytecount statistic is the number of bytes contained in the frames of the Transmit Priority 0-7 Drop statistic.

13.2.1.4.6.17.5.15 Tx Memory Protect Errors (Offset = 3A17Ch)

All ports

The total number of transmit frames on the port that had a memory protect CRC error on egress:

- Any frame destined to be transmitted,

- Was any size
- Had a memory protect CRC error on egress.

Note

1. Frames to the host with memory protect errors are indicated to be dropped with a set receive buffer descriptor **drop** bit. Ethernet frames will have at least one byte of the generated port type CRC inverted on egress.
2. This statistic is 8-bits wide only and will not rollover but will limit at 0xFF.
3. A non-zero value in this statistic will issue a STAT_PEND0 interrupt for the associated port.

13.2.1.4.6.17.5.16 Tx CRC Errors

The total number of frames transmitted on the port with a CRC error. Such a frame:

- was any data frame destined for any unicast, broadcast or multicast address, and
- was any size, and
- was sent cut-thru by the receive port and was received with a CRC error, or
- was a transmitted frame that also had a memory protect error (counted also in CPSW_STATN_TX_MEMORY_PROTECT_ERROR_k).

Note

For port0 this statistics location (CPSW_STAT0_TXDROP) counts the number of drops to the host due to a memory protect error, or due to a cut-thru packet having an error on reception but was forwarded with the error. The Ethernet port receive error could be long, CRC, jabber, or code. For Ethernet ports, the receive error could be the same but the packet is transmitted with at least one byte of the actual outgoing packet CRC inverted to indicate the error.

Note

A nonzero CPSW_STATN_TXDEFERREDFRAMES_k value should be subtracted from the CPSW_STATN_TXGOODFRAMES_k value to obtain the actual good frames value (the good frames statistic also increments on a transmitted CRC error).

13.2.1.4.6.17.5.17 Tx Cut Thru

The total number of cut-thru frames transmitted on the port with or without delay. Such a frame:

- was any data frame destined for any unicast, broadcast or multicast address, and
- was any size greater than or equal to 128 bytes, and
- was sent cut-thru by the receive port.
- was transmitted cut-thru with or without delay.

13.2.1.4.6.17.5.18 Tx Cut Thru Store-and-Forward

The total number of cut-thru frames transmitted store and forward. Such a frame:

- was any data frame destined for any unicast, broadcast or multicast address, and
- was any size greater than or equal to 128 bytes, and
- was sent cut-thru by the receive port.
- was transmitted store-and-forward due to traffic congestion.

13.2.1.4.6.17.6 Rx- and Tx (Shared) Statistics Descriptions

13.2.1.4.6.17.6.1 Rx + Tx 64 Octet Frames (Offset = 3A068h)

All ports

The total number of 64-byte frames received and transmitted on the port. Such a frame is defined to be:

- Any data or MAC control frame which was destined for any unicast, broadcast or multicast address
- Did not experience late collisions, excessive collisions, or carrier sense error

- Was exactly 64 bytes long. (If the frame was being transmitted and experienced carrier loss that resulted in a frame of this size being transmitted, then the frame will be recorded in this statistic).

CRC errors, code/align errors and overruns do not affect the recording of frames in this statistic.

13.2.1.4.6.17.6.2 Rx + Tx 65–127 Octet Frames (Offset = 3A06Ch)

All ports

The total number of frames of size 65 to 127 bytes received and transmitted on the port. Such a frame is defined to be:

- Any data or MAC control frame which was destined for any unicast, broadcast or multicast address
- Did not experience late collisions, excessive collisions, or carrier sense error
- Was 65 to 127 bytes long

CRC errors, code/align errors, underruns and overruns do not affect the recording of frames in this statistic.

13.2.1.4.6.17.6.3 Rx + Tx 128–255 Octet Frames (Offset = 3A070h)

All ports

The total number of frames of size 128 to 255 bytes received and transmitted on the port. Such a frame is defined to be:

- Any data or MAC control frame which was destined for any unicast, broadcast or multicast address
- Did not experience late collisions, excessive collisions, or carrier sense error
- Was 128 to 255 bytes long

CRC errors, code/align errors, underruns and overruns do not affect the recording of frames in this statistic.

13.2.1.4.6.17.6.4 Rx + Tx 256–511 Octet Frames (Offset = 3A074h)

All ports

The total number of frames of size 256 to 511 bytes received and transmitted on the port. Such a frame is defined to be:

- Any data or MAC control frame which was destined for any unicast, broadcast or multicast address
- Did not experience late collisions, excessive collisions, or carrier sense error
- Was 256 to 511 bytes long

CRC errors, code/align errors, underruns and overruns do not affect the recording of frames in this statistic.

13.2.1.4.6.17.6.5 Rx + Tx 512–1023 Octet Frames (Offset = 3A078h)

All ports

The total number of frames of size 512 to 1023 bytes received and transmitted on the port. Such a frame is defined to be:

- Any data or MAC control frame which was destined for any unicast, broadcast or multicast address
- Did not experience late collisions, excessive collisions, or carrier sense error
- Was 512 to 1023 bytes long

CRC errors, code/align errors, underruns and overruns do not affect the recording of frames in this statistic.

13.2.1.4.6.17.6.6 Rx + Tx 1024_Up Octet Frames (Offset = 3A07Ch)

All ports

The total number of frames of size 1024 to RX_MAXLEN bytes for receive or 1024 up for transmit on the port. Such a frame is defined to be:

- Any data or MAC control frame which was destined for any unicast, broadcast or multicast address
- Did not experience late collisions, excessive collisions, or carrier sense error
- Was 1024 to RX_MAXLEN bytes long on receive, or any size on transmit

CRC errors, code/align errors, underruns and overruns do not affect the recording of frames in this statistic.

13.2.1.4.6.17.6.7 Net Octets (Offset = 3A080h)

All ports

The total number of bytes of frame data received and transmitted on the port. Each frame counted:

- was any data or MAC control frame destined for any unicast, broadcast or multicast address (address match does not matter)
- Any length (including less than 64 bytes and greater than RX_MAXLEN bytes)

Also counted in this statistic is:

- Every byte transmitted before a carrier-loss was experienced
- Every byte transmitted before each collision was experienced, (that is, multiple retries are counted each time)
- Every byte received if the port is in half-duplex mode until a jam sequence was transmitted to initiate flow control. (The jam sequence was not counted to prevent double-counting)

Error conditions such as alignment errors, CRC errors, code errors, overruns and underruns do not affect the recording of bytes by this statistic.

The objective of this statistic is to give a reasonable indication of Ethernet utilization.

13.2.1.4.6.17.7

Table 13-152. Rx Statistics Summary

Rx Statistic	Frame/ Oct	Rx/ Rx+ Tx	Frame Type					Frame Size (bytes)								Event				
			MAC control		Data[5]															
			Pause frame	Non-pause[4]	Multi cast	Broad cast	Uni cast	<64	64	65-127	128-255	256-511	512-1023	1024-rx_ maxlen	>rx_ maxlen	Flow Coll.[8]	CRC Error	Align/ Code	Overrun	Addr. Disc.
Good Rx Frames	F	Rx	(y|[1]	y|	y|	y|	y)	n	(y|	y|	y|	y|	y|	y)	n	-[2]	n	n	-	n
Broadcast Rx Frames	F	Rx	(%|[6]	%|	n	y)	n	n	(y|	y|	y|	y|	y|	y)	n	-	n	n	-	n
Multicast Rx Frames	F	Rx	(%|	%|	y)	n	n	n	(y|	y|	y|	y|	y|	y)	n	-	n	n	-	n
Pause Rx Frames	F	Rx	y	n	n	n	n	n	(y|	y|	y|	y|	y|	y)	n	-	n	n	-	-
Rx CRC Errors	F	Rx	(y|	y|	y|	y|	y)	n	(y|	y|	y|	y|	y|	y)	n	-	y	n	-	n
Rx Align/Code Errors	F	Rx	(y|	y|	y|	y|	y)	n	(y|	y|	y|	y|	y|	y)	n	-	-	y	-	n
Oversized Rx Frames	F	Rx	(y|	y|	y|	y|	y)	n	n	n	n	n	n	n	y	-	n	n	-	n
Rx Jabbers	F	Rx	(y|	y|	y|	y|	y)	n	n	n	n	n	n	n	y	-	(y|	y)	-	n
Undersized Rx Frames	F	Rx	n	n	(y|	y|	y)	y	n	n	n	n	n	n	n	-	n	n	-	n
Rx Fragments	F	Rx	n	n	(y|	y|	y)	y^[7]	n	n	n	n	n	n	n	-	(y|	y)	-	-
Rx Overruns[9]	F	Rx	(y|	y|	y|	y|	y)	(y|	y|	y|	y|	y|	y|	y|	y)	-	-	-	y	n
64octet Frames	F	Rx+ Tx[3]	(y|	y|	y|	y|	y)	n	y	n	n	n	n	n	n	-	-	-	-	n
65-127octet Frames	F	Rx+ Tx	(y|	y|	y|	y|	y)	n	n	y	n	n	n	n	n	-	-	-	-	n
128-255octet Frames	F	Rx+ Tx	(y|	y|	y|	y|	y)	n	n	n	y	n	n	n	n	-	-	-	-	n
256-511octet Frames	F	Rx+ Tx	(y|	y|	y|	y|	y)	n	n	n	n	y	n	n	n	-	-	-	-	n

Table 13-152. Rx Statistics Summary (continued)

Rx Statistic	Frame/ Oct	Rx/ Rx+ Tx	Frame Type					Frame Size (bytes)								Event				
			MAC control		Data[5]															
			Pause frame	Non-pause[4]	Multicast	Broadcast	Unicast	<64	64	65-127	128-255	256-511	512-1023	1024-rx_maxlen	>rx_maxlen	Flow Coll.[8]	CRC Error	Align/ Code	Overrun	Addr. Disc.
512-1023octet Frames	F	Rx+ Tx	(y|	y|	y|	y|	y)	n	n	n	n	n	y	n	n	-	-	-	-	n
1024-UPoctet Frames	F	Rx+ Tx	(y|	y|	y|	y|	y)	n	n	n	n	n	n	y	n	-	-	-	-	n
Rx Octets	O	Rx	(y|	y|	y|	y|	y)	n	(y|	y|	y|	y|	y|	y)	n	-	n	n	-	n
Net Octets	O	Rx+ Tx	(y|	y|	y|	y|	y)	(y|	y|	y|	y|	y|	y|	y|	y|	y)	-	-	-	-

(1) "AND" is assumed horizontally across the table between all conditions which form the statistic (marked y or n) except where (y|y), meaning "OR" is indicated. Parentheses are significant.
(2) "-" indicates conditions which are ignored in the formations of the statistic.
(3) Statistics marked "Rx+Tx" are formed by summing the Rx and Tx statistics, each of which is formed independently.
(4) The non-pause column refers to all MAC control frames (for example, frames with length/type=88.08) with opcodes other than 0x0001. The pauseframe column refers to MAC frames with the opcode=0x0001.
(5) The multicast, broadcast and unicast columns in the table refer to non-MAC Control/non-pause frames (i.e. data frames).
(6) "%" If either a MAC control frame or pause frame has a multicast or broadcast destination address then the appropriate statistics will be updated.
(7) "y^" Frame fragments are not counted if less than 8 bytes.
(8) Flow coll. are half-duplex collisions forced by the MAC to achieve flow-control. A collision will be forced during the first 8 bytes so should not show in frame fragments. Some of the '-'s in this column might in reality be 'n's.
(9) The rx_overruns stat is for RX_MOF_OVERRUNS and RX_SOF_OVERRUNS added together.

Table 13-153. Tx Statistics Summary

Tx Statistic[9]	Frame/ Oct	Tx/ Rx +Tx	Frame Type					Frame Size (bytes)								Event								
			MAC control[4]		Data												Collision Type								
			Pause-MAC	Any-CPU	Multicast	Broadcast	Unicast	64	65-127	128-255	256-511	512-1023	1024-1535	>1535	CRC Error	Flow[8]	1	2-15	16	Late	No Carrier	Queued	Deferred	Underrun
Good Tx Frames	F	Tx	(y| (1)	y|	y|	y|	y)	(y|	y|	y|	y|	y|	y|	y)	-[2]	-	-	-	n	n	n	-	-	n
Broadcast Tx Frames	F	Tx	n	(%| (5)	n	y)	n	(y|	y|	y|	y|	y|	y|	y)	-	-	-	-	n	n	n	-	-	n
Multicast Tx Frames	F	Tx	(y|	%|	y)	n	n	(y|	y|	y|	y|	y|	y|	y)	-	-	-	-	n	n	n	-	-	n
Pause Tx Frames	F	Tx	y	n	n	n	n	y	n	n	n	n	n	n	-	-	-	-	-	-	-	-	-	-
Collisions	F	Tx	n	(y|	y|	y|	y)	(y|	y|	y|	y|	y|	y|	y)	-	(+ (6)	+	+	+	+)	n	-	-	-
Single Collision Tx Frames	F	Tx	n	(y|	y|	y|	y)	(y|	y|	y|	y|	y|	y|	y)	-	-	-	y	n	n	n	n	-	-	-
Multiple Collision Tx Frames	F	Tx	n	(y|	y|	y|	y)	(y|	y|	y|	y|	y|	y|	y)	-	-	-	n	y	n	n	n	-	-	-
Excessive Collisions	F	Tx	n	(y|	y|	y|	y)	(y|	y|	y|	y|	y|	y|	y)	-	-	-	n	n	y	n	n	-	-	-
Late Collisions	F	Tx	n	(y|	y|	y|	y)	n	(y|	y|	y|	y|	y|	y)	-	-	-	-	-	-	y	-	-	-	-
Deferred Tx Frames	F	Tx	n	(y|	y|	y|	y)	(y|	y|	y|	y|	y|	y|	y)	-	-	-	n	n	n	n	n	-	y	n

Table 13-153. Tx Statistics Summary (continued)

Tx Statistic[9]	Frame/ Oct	Tx/ Rx +Tx	Frame Type					Frame Size (bytes)							Event									
			MAC control [4]	Data												Collision Type								
			Pause-MAC	Any-CPU	Multicast	Broadcast	Unicast	64	65-127	128-255	256-511	512-1023	1024-1535	>1535	CRC Error	Flow[8]	1	2-15	16	Late	No Carrier	Queued	Deferred	Underrun
Carrier Sense Errors	F	Tx	(y|	y|	y|	y|	y)	(y|	y|	y|	y|	y|	y|	y)	-	-	-	-	-	-	y	-	-	-
64octet Frames	F	Rx+Tx [3]	(y|	y|	y|	y|	y)	y	n	n	n	n	n	n	-	-	-	-	n	n	n	-	-	-
65-127octet Frames	F	Rx+Tx	(y|	y|	y|	y|	y)	n	y	n	n	n	n	n	-	-	-	-	n	n	n	-	-	-
128-255octet Frames	F	Rx+Tx	(y|	y|	y|	y|	y)	n	n	y	n	n	n	n	-	-	-	-	n	n	n	-	-	-
256-511octet Frames	F	Rx+Tx	(y|	y|	y|	y|	y)	n	n	n	y	n	n	n	-	-	-	-	n	n	n	-	-	-
512-1023octet Frames	F	Rx+Tx	(y|	y|	y|	y|	y)	n	n	n	n	y	n	n	-	-	-	-	n	n	n	-	-	-
1024-UPoctet Frames	F	Rx+Tx	(y|	y|	y|	y|	y)	n	n	n	n	n	y	y	-	-	-	-	n	n	n	-	-	-
Tx Octets	O	Tx	(y|	y|	y|	y|	y)	(y|	y|	y|	y|	y|	y|	y)	-	-	-	-	n	n	n	-	-	n
Net Octets	O	Rx+Tx	(y|	y|	y|	y|	y)	(y|	y|	y|	y|	y|	y|	y)	-	-	$[7]	$	$	$	$	-	-	-

(1) "AND" is assumed horizontally across the table between all conditions which form the statistic (marked y or n) except where (y|y), meaning “OR” is indicated. Parentheses are significant.
(2) “-“ indicates conditions which are ignored in the formations of the statistic.
(3) Statistics marked “Rx+Tx” are formed by summing the Rx and Tx statistics, each of which is formed independently.
(4) Pause (MAC) frames are issued in the MAC as perfect (no CRC error) 64 byte frames in full duplex only, so they cannot collide.
(5) “%” If a CPU sourced MAC control frame has a multicast or broadcast destination address then the appropriate statistics will be updated.
(6) “+” indicates collisions which are “summed” (i.e. every collision is counted in the Collisions statistic). Jam sequences used for halfduplex flow control are also counted.
(7) “$” Every byte written on the wire during each retry attempt is also counted in addition to frames which experience no collisions or carrier loss.
(8) The flow collision type is for half-duplex collisions forced by the MAC to achieve flow control. Some of the ‘-‘s in this column might in reality be ‘n’s. To prevent double-counting, Net Octets are unaffected by the jam sequence – the ‘received’ bytes, however, are counted. (See Table 13-152.)
(9) When the transmit Tx FIFO is drained due to the MAC being disabled or link being lost, then the frames being purged will not appear in the Tx statistics.

13.2.1.4.7 Common Platform Time Sync (CPTS)

The Common Platform Time Sync (CPTS) module is used to facilitate host control of time sync operations. It enables compliance with the IEEE 1588-2008 standard for a precision clock synchronization protocol.

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 up to 8 hardware timestamp push inputs
- Supports timestamp counter compare output (CPTS_COMP)
- Supports timestamp counter bit output (CPTS_SYNC)
- Supports a configurable number of timestamp Generator bit outputs (CPTS_GENFn).
- Supports Ethernet Enhanced Scheduled Traffic Operations (CPTS_ESTFn).

- 32-bit and 64-bit timestamp modes with PPM and nudge adjustment.

13.2.1.4.7.1 CPTS Architecture

Figure 13-97 shows the architecture of the CPTS module inside the CPSW Ethernet Subsystem. Time stamp values for every packet transmitted or received on external port of the CPSW are recorded. At the same time, each packet is decoded to determine if it is a valid time sync event. If so, an event is loaded into the Event FIFO for processing containing the recorded time stamp value when the packet was transmitted or received.

In addition, both hardware (HWn_TS_PUSH) and software (TS_PUSH) can be used to read the current time stamp value though the Event FIFO. The reference clock used for the time stamp (CPTS_RFT_CLK) can be derived from several sources.

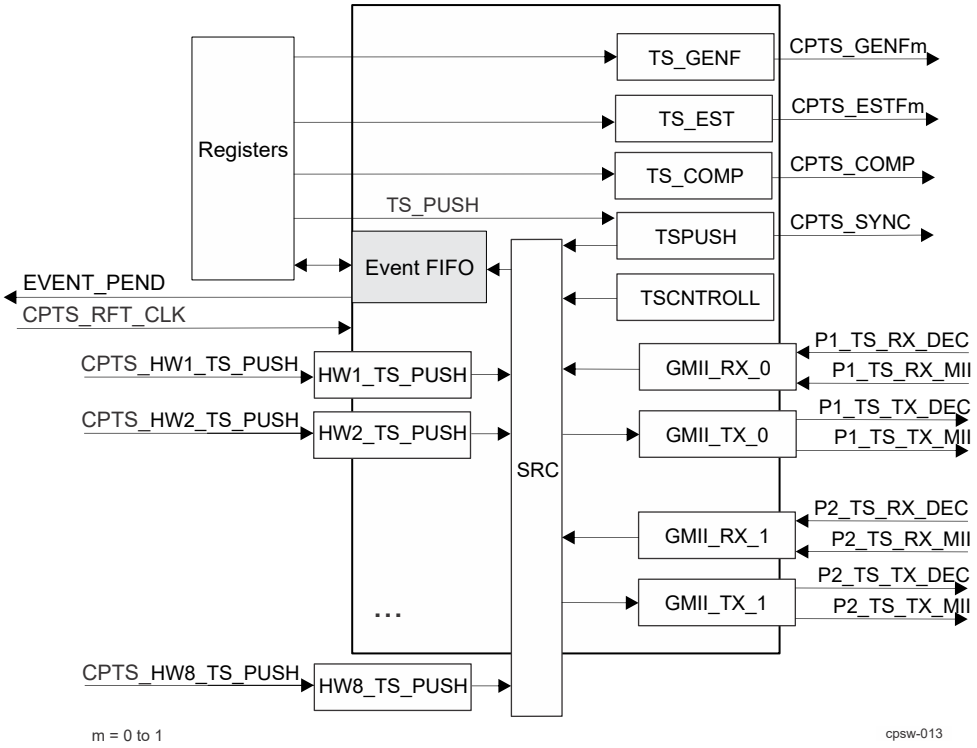


Figure 13-97. CPTS Block Diagram

Note

See *CPSW0 CPTS Integration* for CPTS integration in the device CPSW0 module.

13.2.1.4.7.2 CPTS Initialization

The CPTS module should be configured as follows:

1. Reset the CPTS module.
2. Write the CPTS_CLKSEL value in the CTRLMMR_CPTS_CLKSEL register with the desired reference clock selection. This value is allowed to be written only when the CPTS_EN bit in the CPSW_CPTS_CONTROL_REG register is cleared to zero.
3. Set the CPTS_EN bit in the CPSW_CPTS_CONTROL_REG register.
4. If using interrupts and not polling, enable the interrupt by setting the TS_PEND_EN bit in the CPSW_CPTS_INT_ENABLE_REG register.

13.2.1.4.7.3 32-bit Time Stamp Value

The time stamp value is a 32-bit value that increments on each CPTS_RFT_CLK rising edge when CPTS_EN bit is set to 1h. When CPTS_EN bit is cleared to 0h, the time stamp value is reset to 0h.

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 (CPSW_CPTS_TS_LOAD_VAL_REG / CPSW_CPTS_TS_LOAD_HIGH_VAL_REG and CPSW_CPTS_TS_LOAD_EN_REG registers).

13.2.1.4.7.4 64-bit Time Stamp Value

The time stamp value is a 64-bit value that increments on each CPTS_RFT_CLK rising edge when CPTS_EN bit is set to 1h. When CPTS_EN bit is cleared to 0h, the time stamp value is reset to 0h.

64-bit mode is selected when CPSW_CPTS_CONTROL_REG[5] MODE bit set to 1h.

For test purposes, the time stamp value can be written via the time stamp load function (CPSW_CPTS_TS_LOAD_EN_REG, CPSW_CPTS_TS_LOAD_VAL_REG, and CPSW_CPTS_TS_LOAD_HIGH_VAL_REG registers). The CPSW_CPTS_TS_ADD_VAL_REG feature is included to allow 1ns timestamp operations with an CPTS_RFT_CLK rate less than 1Ghz. Table 13-154 shows the CPTS_RFT_CLK and CPSW_CPTS_TS_ADD_VAL_REG values for 1ns operations.

Table 13-154. ADD_VAL feature

CPTS_RFT_CLK (MHz)	CPSW_CPTS_TS_ADD_VAL_REG[2-0] ADD_VAL
1 GHz	0
500 MHz	1
333.33 MHz	2
250 MHz	3
200 MHz	4
166.66 MHz	5
142.85714 MHz	6
125 MHz	7

13.2.1.4.7.5 64-Bit Timestamp Nudge

The 64-bit TIME_STAMP value can be adjusted by writing the CPSW_CPTS_TS_NUDGE_VAL_REG[7-0] TS_NUDGE_VAL bit field value which is a two's complement value. A value of FFh will subtract 1 clock cycle from the next incremented 64-bit time stamp value (CPSW_CPTS_EVENT_0_REG[31-0] TIME_STAMP and CPSW_CPTS_EVENT_3_REG[31-0] TIME_STAMP value). A nudge value of 1h will add 1 clock cycle to the next incremented TIME_STAMP[63-0] value. For example, if the current TIME_STAMP value is F06h, and CPSW_CPTS_TS_ADD_VAL_REG[2-0] ADD_VAL = 3h, the next incremented timestamp value would be F0Ah without a nudge and F0Ah +/- [7-0] TS_NUDGE_VAL with a nudge. The [7-0] TS_NUDGE_VAL value is cleared to zero when the nudge has occurred.

13.2.1.4.7.6 64-bit Timestamp PPM

The 64-bit TIME_STAMP can be adjusted by parts per million or by parts per hour. Writing a non-zero value to the CPSW_CPTS_TS_PPM_LOW_VAL_REG[31-0] TS_PPM_LOW_VAL (Time stamp PPM Low value) and CPSW_CPTS_TS_PPM_HIGH_VAL_REG[9-0] TS_PPM_HIGH_VAL (Time stamp PPM High value) enables PPM operations. The adjustment is up or down depending on the [7] TS_PPM_DIR bit in the CPSW_CPTS_CONTROL_REG register. The TIME_STAMP value is increased by the PPM value when [7] TS_PPM_DIR bit is cleared. The TIME_STAMP value is decreased by the PPM value when [7] TS_PPM_DIR bit is set.

Parts Per Million example:
To adjust for 100 parts per million the configured value for TS_PPM[41-0] (through CPSW_CPTS_TS_PPM_LOW_VAL_REG[31-0] TS_PPM_LOW_VAL and CPSW_CPTS_TS_PPM_HIGH_VAL_REG[9-0] TS_PPM_HIGH_VAL) is:
1,000,000/100 = 10,000(decimal)

Parts Per Hour example:
To adjust for 1 part per hour at 1 GHz CPTS_RFT_CLK the configured value for TS_PPM[41-0] (through CPSW_CPTS_TS_PPM_LOW_VAL_REG[31-0] TS_PPM_LOW_VAL and CPSW_CPTS_TS_PPM_HIGH_VAL_REG[9-0] TS_PPM_HIGH_VAL) is:
(1,000,0000,000Hz/1pph) * (3600 seconds/hour) = 34630B8A000 (hex)

13.2.1.4.7.7 Event FIFO

All time sync events are pushed onto the Event FIFO. There are 32 locations in the event FIFO with no overrun indication supported. Software must service the event FIFO in a timely manner to prevent FIFO overrun.

13.2.1.4.7.8 Timestamp Compare Output

CPTS features one Time Stamp Compare (CPTS_COMP) output. The CPTS_COMP function is a software oriented feature that is intended to be replaced going forward by the hardware oriented GENF function. CPTS_COMP is not compatible with timestamp PPM or a non-zero CPSW_CPTS_TS_ADD_VAL_REG[2-0] ADD_VAL value.

13.2.1.4.7.8.1 Non-Toggle Mode: 32-bit

The CPTS_COMP output is asserted for CPSW_CPTS_TS_COMP_LEN_REG[31-0] TS_COMP_LENGTH periods when the CPSW_CPTS_EVENT_0_REG[31-0] TIME_STAMP value (lowe 32-bits) compares with the CPSW_CPTS_TS_COMP_VAL_REG[31-0] TS_COMP_VAL and the length value is non-zero. The CPTS_COMP rising edge occurs three CPTS_RFT_CLK clock periods after the values compare. A timestamp compare event is pushed into the event FIFO when CPTS_COMP is asserted. The polarity of the CPTS_COMP output is determined by the CPSW_CPTS_CONTROL_REG[2] TS_COMP_POLARITY bit. The output is asserted low when the polarity bit is 0h.

13.2.1.4.7.8.2 Non-Toggle Mode: 64-bit

64-bit mode operation is identical to 32-bit mode except that all 64-bits of the TIME_STAMP are used (CPSW_CPTS_EVENT_0_REG and CPSW_CPTS_EVENT_3_REG). In 32-bit mode only the lower 32-bits (CPSW_CPTS_EVENT_0_REG) are used.

13.2.1.4.7.8.3 Toggle Mode: 32-bit

The CPTS_COMP output is asserted (CPSW_CPTS_TS_COMP_LEN_REG[31-0] TS_COMP_LENGTH) for CPTS_RFT_CLK clock periods when the TIME_STAMP[31:0] value compares with the CPSW_CPTS_TS_COMP_VAL_REG and the length value is non-zero. The CPTS_COMP toggles thereafter on CPSW_CPTS_TS_COMP_VAL_REG[31-0] TS_COMP_LENGTH for CPTS_RFT_CLK periods. The length high or low can be adjusted by writing the CPSW_CPTS_TS_COMP_NUDGE_REG[7-0] NUDGE bit field value which is a two's complement value. A value of FFh will subtract one CPTS_RFT_CLK period from the CPSW_CPTS_TS_COMP_VAL_REG[31-0] TS_COMP_LENGTH value. A value of 0x01h will add one CPTS_RFT_CLK period to the CPSW_CPTS_TS_COMP_LEN_REG[31-0] TS_COMP_LENGTH value. Only a single high or low time is adjusted (nudged) and the CPSW_CPTS_TS_COMP_NUDGE_REG[7-0] NUDGE value is cleared to zero when the nudge has occurred. The CPTS_COMP output is asserted low when the CPSW_CPTS_CONTROL_REG[2] TS_COMP_POLARITY bit is 0h. No compare events and no CPTS_EVNT interrupts are generated in toggle mode. The CPSW_CPTS_CONTROL_REG[6] TS_COMP_TOG bit must be set for toggle mode (value 1h). Note this bit must be set before writing a non-zero value to CPSW_CPTS_TS_COMP_VAL_REG register.

13.2.1.4.7.8.4 Toggle Mode: 64-bit

64-bit mode operation is identical to 32-bit mode except that all 64-bits of the TIMESTAMP are used (CPSW_CPTS_EVENT_0_REG and CPSW_CPTS_EVENT_3_REG). In 32-bit mode only the lower 32-bits (CPSW_CPTS_EVENT_0_REG) are used.

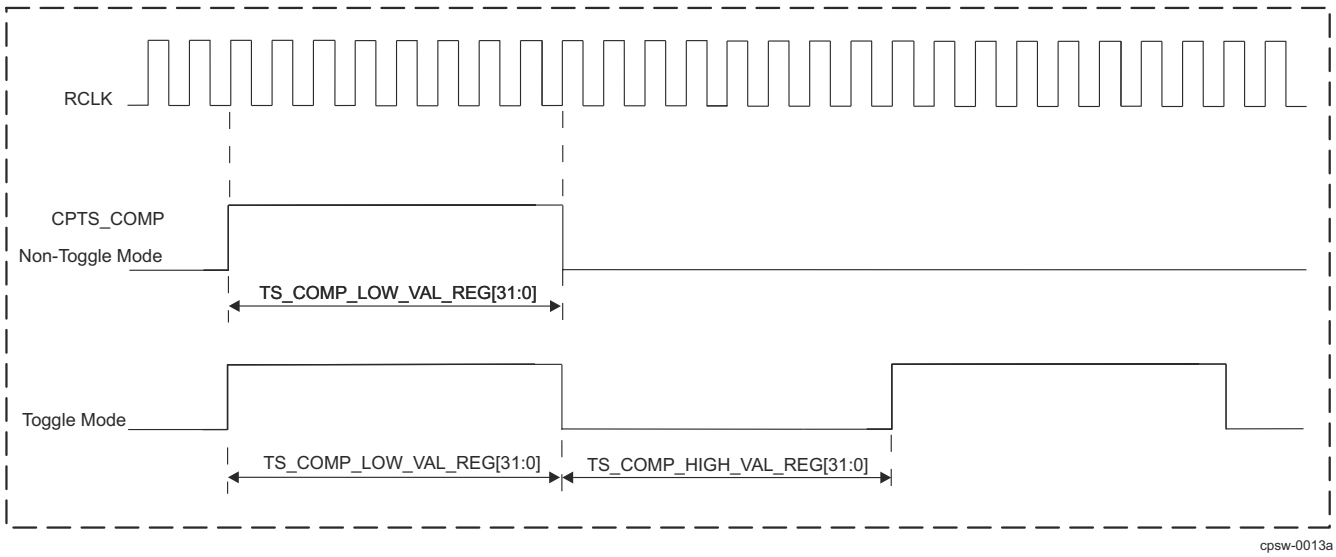


Figure 13-98. CPTS_COMP Output in Toggle and Non-Toggle Mode

13.2.1.4.7.9 Timestamp Sync Output

The CPTS_SYNC output is a selected bit of the [31-0]TIME_STAMP counter value. One of bits 17-31 can be selected in CPSW_CPTS_CONTROL_REG[31-28] TS_SYNC_SEL. The CPTS_SYNC output is disabled when CPSW_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 CPTS_SYNC output. A rising edge will occur on the CPTS_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 CPTS_SYNC operation. The CPTS_SYNC output is two CPTS_RFT_CLK periods after the actual count value.

13.2.1.4.7.10 Timestamp GENFn Output

The CPTS_GENFn outputs have a programmable cycle (frequency) with a PPM feature and software nudge feature. The CPTS_GENFn output cycle is CPSW_GENF0_LENGTH_REG_L[31-0] CPTS_RFT_CLK periods (which is different than CPTS_COMP operation). Figure 13-99 represents the CPTS_GENFn output signal.

The CPTS_GENFn output cycle is CPSW_GENF0_LENGTH_REG_L[31-0] CPTS_RFT_CLK periods beginning when the 64-bit TIME_STAMP value compares with the 64-bit GENFn_COMP value (CPSW_GENF0_COMP_LOW_REG_L and CPSW_GENF0_COMP_HIGH_REG_L registers) and the length value is non-zero. The CPTS_GENFn output cycle repeats thereafter every CPSW_GENF0_LENGTH_REG_L[31-0] CPTS_RFT_CLK 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 CPSW_CPTS_CONTROL_REG[2] TS_COMP_POLARITY bit is low. No compare events and no CPTS_EVNT interrupts are generated.

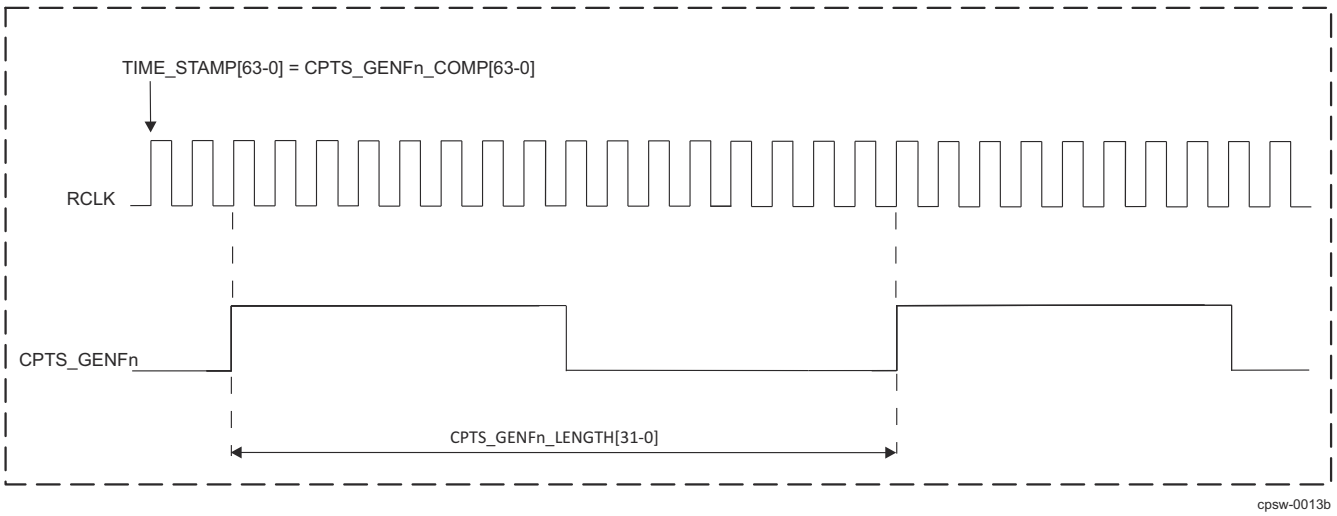


Figure 13-99. CPTS_GENFn Output Signal Diagram

13.2.1.4.7.10.1 GENFn Nudge

The cycle length can be adjusted by writing the CPSW_CPTS_TS_COMP_NUDGE_REG[7-0] NUDGE register value which is a two's complement value. A value of FFh will subtract 1 CPTS_RFT_CLK from the CPSW_GENF0_LENGTH_REG_L[31-0] value. A value of 1h will add 1 CPTS_RFT_CLK to the CPSW_GENF0_LENGTH_REG_L[31-0] value. The CPSW_CPTS_TS_COMP_NUDGE_REG[7-0] NUDGE value is cleared to zero when the nudge has occurred.

13.2.1.4.7.10.2 GENFn PPM

The CPTS_GENFn output cycle can be adjusted by parts per million or by parts per hour. Writing a non-zero value to CPSW_GENF0_PPM_LOW_REG_L/ CPSW_GENF0_PPM_HIGH_REG_L enables PPM operations. The PPM counter continually loads and decrements to zero and then loads again. A single CPTS_RFT_CLK adjustment is made when the PPM counter decrements to zero. The adjustment is up or down depending on the CPSW_GENF0_TS_GENF_CONTROL_REG[0] PPM_DIR bit. When PPM_DIR bit is set a single CPTS_RFT_CLK 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 bit is cleared a single CPTS_RFT_CLK time is added to the generate function counter which has the effect of decreasing the generate function frequency by the PPM amount.

Parts Per Million example:
To adjust for 100 parts per million the configured value for GENF_PPM[41-0] (through CPSW_GENF0_PPM_LOW_REG_L and CPSW_GENF0_PPM_HIGH_REG_L) is:
1,000,000/100 = 10,000(decimal)

Parts Per Hour example:
To adjust for 1 part per hour at 1 GHz CPTS_RFT_CLK the configured value for GENF_PPM[41-0] (through CPSW_GENF0_PPM_LOW_REG_L and CPSW_GENF0_PPM_HIGH_REG_L) is:
(1,000,0000,000Hz/1pph) * (3600 seconds/hour) = 34630B8A000 (hex)

13.2.1.4.7.11 Timestamp ESTFn

Each Ethernet port has a dedicated ESTFn generator which operates identically to the GENFn function.

13.2.1.4.7.12 Time Sync Events

Time Sync events are 96-bit values that are pushed onto the event FIFO and read by software in 32-bit reads. Four 32-bit registers, CPSW_CPTS_EVENT_0_REG through CPSW_CPTS_EVENT_3_REG hold the data of a time sync event. There are eight types of sync events:

- 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
- Ethernet Receive Event
- Ethernet Transmit Event
- Time Stamp Compare Event
- Host Transmit Event

13.2.1.4.7.12.1 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 [0]TS_PUSH bit of the CPSW_CPTS_TS_PUSH_REG register. The time stamp value is returned in the event, along with a time stamp push event code. The upper 32-bits (CPSW_CPTS_EVENT_3_REG register) of the timestamp are zero in 32-bit mode.

13.2.1.4.7.12.2 Time Stamp Counter Rollover Event (32-bit mode only)

The CPTS module contains a 32-bit time stamp value (CPSW_CPTS_EVENT_0_REG). 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. This event occurs only in 32-bit mode.

13.2.1.4.7.12.3 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 (CPSW_CPTS_EVENT_0_REG[31-0] TIME_STAMP) has incremented from 0x7FFF FFFF to 0x8000 0000. The half-rollover event is included to enable software to correct a misaligned event condition. This event occurs only in 32-bit mode.

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 13-100 shows a misalignment condition. This misaligned event condition arises because an Ethernet event time stamp occurs at the beginning of a packet and time passes before the packet is determined to be a valid synchronization packet. The misaligned event condition occurs if the rollover occurs in the middle, after the packet time stamp has been taken, but before the packet has been determined to be a valid time sync packet.

Host software must detect and correct for misaligned event conditions. For every event time stamp 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. 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.

When a rollover occurs, software increments the software time stamp upper value. The misaligned case indicates to software that the misaligned event time stamp has a valid upper value that is pre-increment, so one must be subtracted from the upper value to allow software to calculate the correct time for the misaligned event.

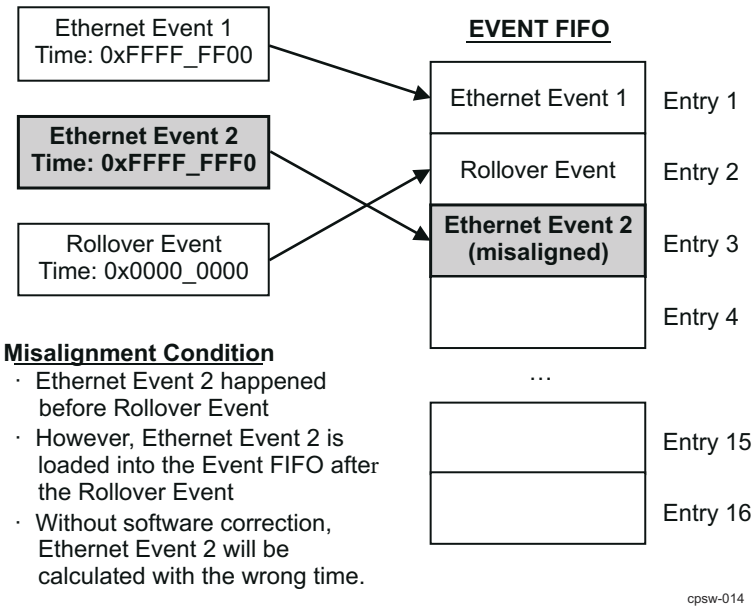


Figure 13-100. Event FIFO Misalignment Condition

13.2.1.4.7.12.4 Hardware Time Stamp Push Event

There are four hardware time stamp inputs (CPTS_HW[1:4]_TS_PUSH events) 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 *CPSW0 CPTS Integration*. The event is loaded into the event FIFO on the rising edge of the timer, and the PORT_NUMBER field in the CPSW_CPTS_EVENT_1_REG register indicates the hardware push input that caused the event (encoded).

The hardware time stamp inputs are asynchronous and are low frequency signals. The CPTS logic synchronizes and performs a rising edge detect on the incoming asynchronous input.

Each hardware time stamp input must be asserted for at least 10 periods of the selected CPTS_RFT_CLK clock. Each input can be enabled or disabled by setting the respective bits in the CPSW_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.

13.2.1.4.7.12.5 Ethernet Port Events

Packets transmitted or received on each Ethernet port can generate Ethernet Transmit Events or Ethernet Receive Events, respectively. The CPTS hardware will decode each packet to determine if it is a valid CPTS time sync event.

According to the IEEE 802.3 Ethernet standard, each Ethernet frame contains a 2-octet EtherType field to indicate which protocol is encapsulated in the PayLoad field, as shown in Figure 13-101. For standard time sync packets, this will contain the EtherType for the Precision Time Protocol (IEEE 1588), which is defined as 0x88F7. The CPTS hardware will compare this field to the TS_LTYPE1 field in the CPSW_PN_TS_SEQ_LTYPE_REG or the TS_LTYPE2 field in CPSW_PN_TS_CTL_LTYPE2_REG register (depending on which enable bit was set) , which should also be programmed to 88F7h.

When a virtual LAN is used, an additional 4-octet 802.1Q tag is inserted in the Ethernet frame before the EtherType field, as shown in Figure 13-101. To indicate to the CPTS hardware that a virtual LAN is in use, the TS_TX_VLAN_LTYPE1_EN (or TS_TX_VLAN_LTYPE2_EN) enable bit must be set in the CPSW_PN_TS_CTL_REG register. The EtherType for the 802.1Q tag is defined as 0x8100, and the CPTS hardware will compare this value to the TS_VLAN_LTYPE1 (or TS_VLAN_LTYPE2 depending on which enable

bit was set) field in the CPSW_PN_TS_VLAN_LTYPE_REG register, which should also be programmed to 0x8100.

When two stacked VLANs are used, two additional 4-octet 801.Q tags are inserted in the Ethernet frame before the EtherType field, as shown in Figure 13-101. In this case, both TS_VLAN_LTYPE1 and TS_VLAN_LTYPE2 must be enabled. The outer tag must match the value of the TS_VLAN_LTYPE1 field, and the inner tag must match the value of the TS_VLAN_LTYPE2 field.

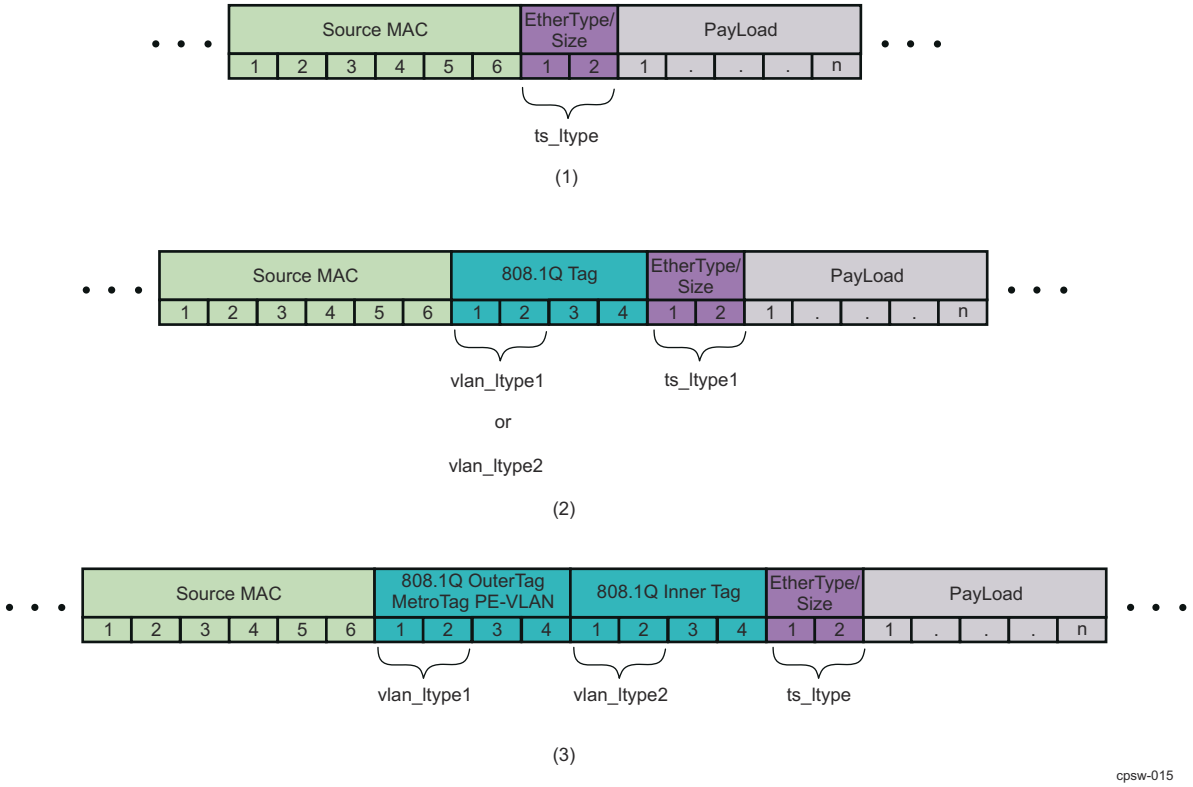


Figure 13-101. Partial Ethernet-II Frames Showing Register Mapping of EtherTypes for a Simple Frame (1), a Single 1Q Tag Added (2), and Two 1Q Tags Added (3)

13.2.1.4.7.12.5.1 Ethernet Port Receive Event

This section describes Ethernet port receive events. Ethernet port generates time synchronization events for valid received time sync packets. For every packet received on the Ethernet port, a timestamp will be captured by the receive module inside the CPTS for the corresponding port. The time stamp will be captured by the receive module regardless of whether or not the packet is a time synchronization packet to make sure that the time stamp is captured as soon as possible. The packet is sampled on both the rising and falling edges of the CPTS_RFT_CLK, and the time stamp will be captured once the start of frame delimiter for the receive packet is detected.

After the time stamp has been captured, the receive interface will begin parsing the packet to determine if it is a valid Ethernet time synchronization packet. The CPSW decoder determines if the packet is a valid Ethernet receive time synchronization event. The receive interface for the port will use the following criteria to determine if the packet is a valid Annex D, Annex E, or Annex F time synchronization Ethernet receive event:

Annex D (IPv4)

1. Receive annex D time sync is enabled (TS_RX_ANNEX_D_EN is set in the CPSW_PN_TS_CTL_REG register).
2. One of the sequences below is true.
   a. The first packet LTYPE matches 0x0800

   b. The first packet LTYPE matches TS_VLAN_LTYPE1 in the CPSW_PN_TS_VLAN_LTYPE_REG register and TS_RX_VLAN_LTYPE1_EN is set in the CPSW_PN_TS_CTL_REG register and the second packet LTYPE matches 0x0800
   c. The first packet LTYPE matches TS_VLAN_LTYPE2 in the CPSW_PN_TS_VLAN_LTYPE_REG register and TS_RX_VLAN_LTYPE2_EN is set in the CPSW_PN_TS_CTL_REG register and the second packet LTYPE matches 0x0800
   d. The first packet LTYPE matches TS_VLAN_LTYPE1 in the CPSW_PN_TS_VLAN_LTYPE_REG register and TS_RX_VLAN_LTYPE1_EN is set in the CPSW_PN_TS_CTL_REG register and the second packet LTYPE matches TS_VLAN_LTYPE2 in the CPSW_PN_TS_VLAN_LTYPE_REG register and TS_RX_VLAN_LTYPE2_EN is set in the CPSW_PN_TS_CTL_REG register and the third packet LTYPE matches 0x0800
3. Byte 14 (the byte after the LTYPE) contains 0x45 (IPv4).

Note

The byte numbering assumes that there are no VLANs. The byte number is intended to show the relative order of the bytes.

4. Byte 20 contains 0bXXX00000 (5 lower bits zero) and Byte 21 contains 0x00 (fragment offset zero)
5. Byte 22 contains 0x01 (HOP Limit = 1) if the TS_TTL_NONZERO bit in the switch CPSW_PN_TS_CTL_LTYPE2_REG register is cleared to 0h, or byte 22 contains any value if CPSW_PN_TS_CTL_LTYPE2_REG is set to 1h. Byte 22 is the TTL/HOP field.
6. Byte 23 contains 0x11 (Next Header UDP Fixed).
7. The TS_UNI_EN bit in the CPSW_PN_TS_CTL_LTYPE2_REG register is cleared to 0h and Bytes 30 through 33 contain:
   a. Decimal 224.0.1.129 and the TS_129 bit in the CPSW_PN_TS_CTL_LTYPE2_REG register is set, or
   b. Decimal 224.0.1.130 and the TS_130 bit in the CPSW_PN_TS_CTL_LTYPE2_REG register is set, or
   c. Decimal 224.0.1.131 and the TS_131 bit in the CPSW_PN_TS_CTL_LTYPE2_REG register is set, or
   d. Decimal 224.0.1.132 and the TS_132 bit in the CPSW_PN_TS_CTL_LTYPE2_REG register is set, or
   e. Decimal 224.0.0.107 and the TS_107 bit in the CPSW_PN_TS_CTL_LTYPE2_REG register is set

   -OR-

   The TS_UNI_EN bit in the CPSW_PN_TS_CTL_LTYPE2_REG register is set and Bytes 30 through 33 contain any values.
8. Bytes 36 and 37 contain:
   a. Decimal 0x01 and 0x3F respectively and the TS_319 bit in the CPSW_PN_TS_CTL_LTYPE2_REG register is set -OR-
   b. Decimal 0x01 and 0x40 respectively and the TS_320 bit in the CPSW_PN_TS_CTL_LTYPE2_REG register is set.
9. The PTP message begins in byte 42.
10. The packet message type is enabled in the TS_MSG_TYPE_EN field in the CPSW_PN_TS_CTL_REG register.
11. The packet was received without error (not long/short/mac_ctl/CRC/code/align).

Annex E (IPv6)

1. Receive annex E time sync is enabled (TS_RX_ANNEX_E_EN bit is set in the switch CPSW_PN_TS_CTL_REG register).
2. One of the sequences below is true.
   a. The first packet LTYPE matches 0x86dd.
   b. The first packet LTYPE matches TS_VLAN_LTYPE1 in the CPSW_PN_TS_VLAN_LTYPE_REG register and TS_RX_VLAN_LTYPE1_EN is set in the CPSW_PN_TS_CTL_REG register and the second packet LTYPE matches 0x86dd
   c. The first packet LTYPE matches TS_VLAN_LTYPE2 in the CPSW_PN_TS_VLAN_LTYPE_REG register and TS_RX_VLAN_LTYPE2_EN is set in the CPSW_PN_TS_CTL_REG register and the second packet LTYPE matches 0x86dd
   d. The first packet LTYPE matches TS_VLAN_LTYPE1 in the CPSW_PN_TS_VLAN_LTYPE_REG register and TS_RX_VLAN_LTYPE1_EN is set in the CPSW_PN_TS_CTL_REG register and the second

packet LTYPE matches TS_VLAN_LTYPE2 in the CPSW_PN_TS_VLAN_LTYPE_REG register and TS_RX_VLAN_LTYPE2_EN is set in the CPSW_PN_TS_CTL_REG register and the third packet LTYPE matches 0x86dd

3. Byte 14 (the byte after the LTYPE) contains 0x6X (IPv6).
4. Byte 20 contains 0x11 (UDP Fixed Next Header).
5. Byte 21 contains 0x01 (Hop Limit = 1) if the TS_TTL_NONZERO bit in the switch CPSW_PN_TS_CTL_LTYPE2_REG register is cleared to 0h, or byte 21 contains any value if TS_TTL_NONZERO is set to 1h. Byte 21 is the TTL/HOP field.
6. The TS_UNI_EN bit in the CPSW_PN_TS_CTL_LTYPE2_REG register is cleared to 0 and Bytes 38 through 53 contain:
   a. FF0M:0:0:0:0:0:0:0181 and the TS_129 bit in the CPSW_PN_TS_CTL_LTYPE2_REG register is set, or
   b. FF0M:0:0:0:0:0:0:0182 and the TS_130 bit in the CPSW_PN_TS_CTL_LTYPE2_REG register is set, or
   c. FF0M:0:0:0:0:0:0:0183 and the TS_131 bit in the CPSW_PN_TS_CTL_LTYPE2_REG register is set, or
   d. FF0M:0:0:0:0:0:0:0184 and the TS_132 bit in the CPSW_PN_TS_CTL_LTYPE2_REG register is set, or
   e. FF0M:0:0:0:0:0:0:006B and the TS_107 bit in the CPSW_PN_TS_CTL_LTYPE2_REG register is set

Note

All values above are 16-bit hex numbers where M is enabled in the TS_MCAST_TYPE_EN field in the CPSW_PN_TS_CTL2_REG register.

-OR-

The TS_UNI_EN bit in the CPSW_PN_TS_CTL_LTYPE2_REG register is set to 1h and Bytes 38 through 53 contain any value.

7. Bytes 56 and 57 contain (UDP Header in bytes 54 through 61):
   a. Decimal 0x01 and 0x3F respectively and the TS_319 bit in the CPSW_PN_TS_CTL2_REG register is set, or
   b. Decimal 0x01 and 0x40 respectively and the TS_320 bit in the CPSW_PN_TS_CTL2_REG register is set.
8. The PTP message begins in byte 62.
9. The packet message type is enabled in the MSG_TYPE_EN field in the CPSW_PN_TS_CTL2_REG register.
10. The packet was received without error (not long/short/mac_ctl/CRC/code/align).

Annex F (IEEE 802.3)

1. Receive Annex F time sync is enabled (TS_RX_ANNEX_F_EN is set in the switch CPSW_PN_TS_CTL_REG register).
2. One of the sequences below is true:
   a. The first packet LTYPE matches TS_LTYPE1 in the CPSW_PN_TS_SEQ_LTYPE_REG/ CPSW_PN_TS_SEQ_LTYPE_REG register. LTYPE 1 should be used when only one time sync LTYPE is to be enabled.
   b. The first packet LTYPE matches TS_LTYPE2 in the CPSW_PN_TS_CTL_LTYPE2_REG register and LTYPE2_EN is set in the CPSW_PN_TS_CTL_REG register.
   c. The first packet LTYPE matches TS_VLAN_LTYPE1 in the CPSW_PN_TS_VLAN_LTYPE_REG register and TS_RX_VLAN_LTYPE1_EN is set in the CPSW_PN_TS_CTL_REG register and the second packet LTYPE matches TS_LTYPE1 in the CPSW_PN_TS_SEQ_LTYPE_REG register
   d. The first packet LTYPE matches TS_VLAN_LTYPE1 in the CPSW_PN_TS_VLAN_LTYPE_REG register and TS_RX_VLAN_LTYPE1_EN is set in the CPSW_PN_TS_CTL_REG register and the second packet LTYPE matches TS_LTYPE2 in the CPSW_PN_TS_CTL_LTYPE2_REG register and TS_LTYPE2_EN is set in the CPSW_PN_TS_CTL_REG register.
   e. The first packet LTYPE matches TS_VLAN_LTYPE2 in the CPSW_PN_TS_VLAN_LTYPE_REG register and TS_RX_VLAN_LTYPE2_EN is set in the CPSW_PN_TS_CTL_REG register and the second packet LTYPE matches TS_LTYPE1 in the CPSW_PN_TS_SEQ_LTYPE_REG register.
   f. The first packet LTYPE matches TS_VLAN_LTYPE2 in the CPSW_PN_TS_VLAN_LTYPE_REG register and TS_RX_VLAN_LTYPE2_EN is set in the CPSW_PN_TS_CTL_REG register and the second packet

LTYPE matches TS_LTYPE2 in the CPSW_PN_TS_CTL_LTYPE2_REG register and TS_LTYPE2_EN is set in the CPSW_PN_TS_CTL_REG register.

g. The first packet LTYPE matches TS_VLAN_LTYPE1 in the CPSW_PN_TS_VLAN_LTYPE_REG register and TS_RX_VLAN_LTYPE1_EN is set in the CPSW_PN_TS_CTL_REG register and the second packet LTYPE matches TS_VLAN_LTYPE2 in the CPSW_PN_TS_VLAN_LTYPE_REG register and TS_RX_VLAN_LTYPE2_EN is set in the CPSW_PN_TS_CTL_REG register and the third packet LTYPE matches TS_LTYPE1 in the CPSW_PN_TS_SEQ_LTYPE_REG register.

h. The first packet LTYPE matches TS_VLAN_LTYPE1 in the CPSW_PN_TS_VLAN_LTYPE_REG register and TS_RX_VLAN_LTYPE1_EN is set in the CPSW_PN_TS_CTL_REG register and the second packet LTYPE matches TS_VLAN_LTYPE2 in the CPSW_PN_TS_VLAN_LTYPE_REG register and TS_RX_VLAN_LTYPE2_EN is set in the CPSW_PN_TS_CTL_REG register and the third packet LTYPE matches TS_LTYPE2 in the CPSW_PN_TS_CTL_LTYPE2_REG register and TS_LTYPE2_EN is set in the CPSW_PN_TS_CTL_REG register

3. The PTP message begins in the byte after the LTYPE.
4. The packet message type is enabled in the TS_MSG_TYPE_EN field in the CPSW_PN_TS_CTL_REG register.
5. The packet was received without error (not long/short/mac_ctl/CRC/code/align).

If all of the criteria described above are met for either Annex D, Annex E, or Annex F, and the packet is determined to be a valid time synchronization packet, then the RX interface will push an Ethernet receive event into the event FIFO.

13.2.1.4.7.12.5.2 Ethernet Port Transmit Event

This section describes Ethernet port transmit events. For every packet transmitted on the Ethernet ports, the port transmit interface will begin parsing the packet to determine if it is a valid Ethernet time synchronization packet. The CPTS transmit interface for the port will use to the following criteria to determine if the packet is a valid time synchronization Ethernet transmit event. The CPSW decoder determines if the packet is a valid ethernet receive time synchronization event. To be a valid Ethernet transmit time synchronization event, the conditions listed below must be true for either Annex D, Annex E, or Annex F:

Annex D (IPv4)

1. Transmit time sync is enabled (TS_TX_ANNEX_D_EN is set in the CPSW_PN_TS_CTL_REG register).
2. One of the sequences below is true.
   a. The first packet LTYPE matches 0x0800
   b. The first packet LTYPE matches TS_VLAN_LTYPE1 in the CPSW_PN_TS_VLAN_LTYPE_REG register and TS_TX_VLAN_LTYPE1_EN is set in the CPSW_PN_TS_CTL_REG register and the second packet LTYPE matches 0x0800
   c. The first packet LTYPE matches TS_VLAN_LTYPE2 in the CPSW_PN_TS_VLAN_LTYPE_REG register and TS_TX_VLAN_LTYPE2_EN is set in the CPSW_PN_TS_CTL_REG register and the second packet LTYPE matches 0x0800
   d. The first packet LTYPE matches TS_VLAN_LTYPE1 in the CPSW_PN_TS_VLAN_LTYPE_REG register and TS_TX_VLAN_LTYPE1_EN is set in the CPSW_PN_TS_CTL_REG register and the second packet LTYPE matches TS_VLAN_LTYPE2 in the CPSW_PN_TS_VLAN_LTYPE_REG register and TS_TX_VLAN_LTYPE2_EN is set in the CPSW_PN_TS_CTL_REG register and the third packet LTYPE matches 0x0800
3. Byte 14 (the byte after the LTYPE) contains 0x45 (IPv4).

Note

The byte numbering assumes that there are no VLANs. The byte number is intended to show the relative order of the bytes.

4. Byte 20 contains 0bXXX00000 (5 lower bits zero) and Byte 21 contains 0x00 (fragment offset zero)
5. Byte 22 contains 0x01 (HOP Limit = 1) if the TS_TTL_NONZERO bit in the switch CPSW_PN_TS_CTL_LTYPE2_REG register is cleared to 0h, or byte 22 contains any value if TS_TTL_NONZERO is set to 1h. Byte 22 is the TTL/HOP field.
6. Byte 23 contains 0x11 (Next Header UDP Fixed).

7. The TS_UNI_EN bit in the CPSW_PN_TS_CTL_LTYPE2_REG register is cleared to 0h and Bytes 30 through 33 contain:
   a. Decimal 224.0.1.129 and the TS_129 bit in the CPSW_PN_TS_CTL_LTYPE2_REG register is set, or
   b. Decimal 224.0.1.130 and the TS_130 bit in the CPSW_PN_TS_CTL_LTYPE2_REG register is set, or
   c. Decimal 224.0.1.131 and the TS_131 bit in the CPSW_PN_TS_CTL_LTYPE2_REG register is set, or
   d. Decimal 224.0.1.132 and the TS_132 bit in the CPSW_PN_TS_CTL_LTYPE2_REG register is set, or
   e. Decimal 224.0.0.107 and the TS_107 bit in the CPSW_PN_TS_CTL_LTYPE2_REG register is set, or
   f. The TS_UNI_EN bit in the CPSW_PN_TS_CTL_LTYPE2_REG register is set and Bytes 30 through 33 contain any values.
8. Bytes 36 and 37 contain:
   a. Decimal 0x01 and 0x3F respectively and the TS_319 bit in the CPSW_PN_TS_CTL_LTYPE2_REG register is set, or
   b. Decimal 0x01 and 0x40 respectively and the TS_320 bit in the CPSW_PN_TS_CTL_LTYPE2_REG register is set.
9. The PTP message begins in byte 42.
10. The packet message type is enabled in the TS_MSG_TYPE_EN field in the CPSW_PN_TS_CTL_REG register.
11. The packet was sent by host port 0.

Annex E (IPv6)

1. Transmit annex E time sync is enabled (TS_TX_ANNEX_E_EN bit is set in the switch CPSW_PN_TS_CTL_REG register).
2. One of the sequences below is true.
   a. The first packet LTYPE matches 0x86dd.
   b. The first packet LTYPE matches TS_VLAN_LTYPE1 in the CPSW_PN_TS_VLAN_LTYPE_REG register and TS_TX_VLAN_LTYPE1_EN is set in the CPSW_PN_TS_CTL_REG register and the second packet LTYPE matches 0x86dd
   c. The first packet LTYPE matches TS_VLAN_LTYPE2 in the CPSW_PN_TS_VLAN_LTYPE_REG register and TS_TX_VLAN_LTYPE2_EN is set in the CPSW_PN_TS_CTL_REG register and the second packet LTYPE matches 0x86dd
   d. The first packet LTYPE matches TS_VLAN_LTYPE1 in the CPSW_PN_TS_VLAN_LTYPE_REG register and TS_TX_VLAN_LTYPE1_EN is set in the CPSW_PN_TS_CTL_REG register and the second packet LTYPE matches TS_VLAN_LTYPE2 in the CPSW_PN_TS_VLAN_LTYPE_REG register and TS_TX_VLAN_LTYPE2_EN is set in the CPSW_PN_TS_CTL_REG register and the third packet LTYPE matches 0x86dd
3. Byte 14 (the byte after the LTYPE) contains 0x6X (IPv6).
4. Byte 20 contains 0x11 (UDP Fixed Next Header).
5. Byte 21 contains 0x01 (Hop Limit = 1) if the TS_TTL_NONZERO bit in the switch CPSW_PN_TS_CTL_LTYPE2_REG register is cleared to 0h, or byte 21 contains any value if TS_TTL_NONZERO is set to 1h. Byte 21 is the TTL/HOP field..
6. The TS_UNI_EN bit in the CPSW_PN_TS_CTL_LTYPE2_REG register is cleared to 0 and Bytes 38 through 53 contain:
   a. FF0M:0:0:0:0:0:0:0181 and the TS_129 bit in the CPSW_PN_TS_CTL_LTYPE2_REG register is set, or
   b. FF0M:0:0:0:0:0:0:0182 and the TS_130 bit in the CPSW_PN_TS_CTL_LTYPE2_REG register is set, or
   c. FF0M:0:0:0:0:0:0:0183 and the TS_131 bit in the CPSW_PN_TS_CTL_LTYPE2_REG register is set, or
   d. FF0M:0:0:0:0:0:0:0184 and the TS_132 bit in the CPSW_PN_TS_CTL_LTYPE2_REG register is set, or
   e. FF0M:0:0:0:0:0:0:006B and the TS_107 bit in the CPSW_PN_TS_CTL_LTYPE2_REG register is set

Note

All values above are 16-bit hex numbers where M is enabled in the TS_MCAST_TYPE_EN field in the CPSW_PN_TS_CTL2_REG register.

-OR-

The TS_UNI_EN bit in the CPSW_PN_TS_CTL_LTYPE2_REG register is set to 1h and Bytes 38 through 53 contain any value.

7. Bytes 56 and 57 contain (UDP Header in bytes 54 through 61):
   a. Decimal 0x01 and 0x3F respectively and the TS_319 bit in the CPSW_PN_TS_CTL_LTYPE2_REG register is set, or
   b. Decimal 0x01 and 0x40 respectively and the TS_320 bit in the CPSW_PN_TS_CTL_LTYPE2_REG register is set.
8. The PTP message begins in byte 62.
9. The packet message type is enabled in the TS_MSG_TYPE_EN field in the CPSW_PN_TS_CTL_REG register.
10. The packet was sent by host port 0.

Annex F (IEEE 802.3)

1. Transmit time sync is enabled (TS_TX_ANNEX_F_EN is set in the switch CPSW_PN_TS_CTL_REG register).
2. One of the sequences below is true:
   a. The first packet LTYPE matches TS_LTYPE1 in the CPSW_PN_TS_SEQ_LTYPE_REG register. LTYPE 1 should be used when only one time sync LTYPE is to be enabled.
   b. The first packet LTYPE matches TS_LTYPE2 in the CPSW_PN_TS_CTL_LTYPE2_REG register and TS_LTYPE2_EN is set in the CPSW_PN_TS_CTL_REG register.
   c. The first packet LTYPE matches TS_VLAN_LTYPE1 in the CPSW_PN_TS_VLAN_LTYPE_REG register and TS_TX_VLAN_LTYPE1_EN is set in the CPSW_PN_TS_CTL_REG register and the second packet LTYPE matches TS_LTYPE2 in the CPSW_PN_TS_CTL_LTYPE2_REG register.
   d. The first packet LTYPE matches TS_VLAN_LTYPE2 in the CPSW_PN_TS_VLAN_LTYPE_REG register and TS_TX_VLAN_LTYPE2_EN is set in the CPSW_PN_TS_CTL_REG register and the second packet LTYPE matches TS_LTYPE1 in the CPSW_PN_TS_SEQ_LTYPE_REG register.
   e. The first packet LTYPE matches TS_VLAN_LTYPE2 in the CPSW_PN_TS_VLAN_LTYPE_REG register and TS_TX_VLAN_LTYPE2_EN is set in the CPSW_PN_TS_CTL_REG register and the second packet LTYPE matches TS_LTYPE2 in the CPSW_PN_TS_CTL_LTYPE2_REG register.
   f. The first packet LTYPE matches TS_VLAN_LTYPE1 in the CPSW_PN_TS_VLAN_LTYPE_REG register and TS_TX_VLAN_LTYPE1_EN is set in the CPSW_PN_TS_CTL_REG register and the second packet LTYPE matches TS_LTYPE2 in the CPSW_PN_TS_CTL_LTYPE2_REG register and TS_LTYPE2_EN is set in the CPSW_PN_TS_CTL_REG register.
   g. The first packet LTYPE matches TS_VLAN_LTYPE1 in the CPSW_PN_TS_VLAN_LTYPE_REG register and TS_TX_VLAN_LTYPE1_EN is set in the CPSW_PN_TS_CTL_REG register and the second packet LTYPE matches TS_VLAN_LTYPE2 in the CPSW_PN_TS_VLAN_LTYPE_REG register and TS_TX_VLAN_LTYPE2_EN is set in the CPSW_PN_TS_CTL_REG register and the third packet LTYPE matches TS_LTYPE2 in the CPSW_PN_TS_CTL_LTYPE2_REG register.
   h. The first packet LTYPE matches TS_VLAN_LTYPE1 in the CPSW_PN_TS_VLAN_LTYPE_REG register and TS_TX_VLAN_LTYPE1_EN is set in the CPSW_PN_TS_CTL_REG register and the second packet LTYPE matches TS_VLAN_LTYPE2 in the CPSW_PN_TS_VLAN_LTYPE_REG register and TS_TX_VLAN_LTYPE2_EN is set in the CPSW_PN_TS_CTL_REG register and the third packet LTYPE matches TS_LTYPE2 in the CPSW_PN_TS_CTL_LTYPE2_REG register and TS_LTYPE2_EN is set in the CPSW_PN_TS_CTL_REG register
3. The packet message type is enabled in the TS_MSG_TYPE_EN field in the CPSW_PN_TS_CTL_REG register.
4. The packet was sent by host port 0.

If all of the criteria described above are met, and the packet is determined to be a valid time synchronization packet, then the time stamp for the transmit event will not be generated until the start of frame delimiter of the packet is actually transmitted. The start of frame delimiter will be sampled on every rising and falling edge of the CPTS_RFT_CLK. Once the packet is transmitted, then the TX interface will push an Ethernet transmit event into the event FIFO.

Table 13-155. Values of Message Type Field

Message Type	Value (hex)
Sync	0
Delay_Req	1

Table 13-155. Values of Message Type Field (continued)

Message Type	Value (hex)
Pdelay_Req	2
Pdelay_Resp	3
Reserved	4:7
Follow_Up	8
Delay_Resp	9
Pdelay_Resp_Follow_Up	A
Announce	B
Signaling	C
Management	D
Reserved	E:F

Once a transmitted or received packet is determined to be a valid time sync packet, the Ethernet Transmit Event or Ethernet Receive Event is loaded onto the Event FIFO.

The CPSW_CPTS_EVENT_1_REG register contains the Message Type and Sequence ID values from the original time sync packet. The CPSW_CPTS_EVENT_0_REG (and CPSW_CPTS_EVENT_3_REG) register contains the time stamp value when the packet arrived at the corresponding port.

Table 13-156. Values of Message Type Field

Message Type	Value (hex)
Sync	0
Delay_Req	1
Pdelay_Req	2
Pdelay_Resp	3
Reserved	4:7
Follow_Up	8
Delay_Resp	9
Pdelay_Resp_Follow_Up	A
Announce	B
Signaling	C
Management	D
Reserved	E:F

Once a transmitted or received packet is determined to be a valid time sync packet, the Ethernet Transmit Event or Ethernet Receive Event is loaded onto the Event FIFO.

The CPSW_CPTS_EVENT_1_REG register contains the Message Type and Sequence ID values from the original time sync packet. The CPSW_CPTS_EVENT_0_REG (and CPSW_CPTS_EVENT_3_REG) register contains the time stamp value when the packet arrived at the corresponding port.

13.2.1.4.7.12.5.3

Table 13-157. Values of Message Type Field

Message Type	Value (hex)
Sync	0
Delay_Req	1
Pdelay_Req	2
Pdelay_Resp	3
Reserved	4:7
Follow_Up	8

Table 13-157. Values of Message Type Field (continued)

Message Type	Value (hex)
Delay_Resp	9
Pdelay_Resp_Follow_Up	A
Announce	B
Signaling	C
Management	D
Reserved	E:F

Once a transmitted or received packet is determined to be a valid time sync packet, the Ethernet Transmit Event or Ethernet Receive Event is loaded onto the Event FIFO.

The CPSW_CPTS_EVENT_1_REG register contains the Message Type and Sequence ID values from the original time sync packet. The CPSW_CPTS_EVENT_0_REG (and CPSW_CPTS_EVENT_3_REG) register contains the time stamp value when the packet arrived at the corresponding port.

13.2.1.4.7.13 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.

13.2.1.4.7.13.1 32-Bit Mode

The CPTS_COMP output is also asserted when the event is generated. The event is generated when the 32-bit time stamp value (CPSW_CPTS_EVENT_0_REG) compares with the CPSW_CPTS_TS_COMP_VAL_REG register and the CPSW_CPTS_TS_COMP_LEN_REG value is non-zero. The CPSW_CPTS_TS_COMP_LEN_REG value should be written by software after the CPSW_CPTS_TS_COMP_VAL_REG register is written and should be zero when the comparison value is written.

13.2.1.4.7.13.2 64-Bit Mode

The CPTS_COMP output is also asserted when the event is generated. The event is generated when the 64-bit time stamp value (CPSW_CPTS_EVENT_0_REG and CPSW_CPTS_EVENT_3_REG) compares with the CPSW_CPTS_TS_COMP_VAL_REG and CPSW_CPTS_TS_COMP_HIGH_VAL_REG registers and the CPSW_CPTS_TS_COMP_LEN_REG value is non-zero. The CPSW_CPTS_TS_COMP_LEN_REG value should be written by software after the CPSW_CPTS_TS_COMP_VAL_REG register is written and should be zero when the comparison value is written.

13.2.1.4.7.14 Host Transmit Event

The host can send a packet to be transmitted on an Ethernet port that will generate a time synchronization event. The host sets the TSTAMP_EN bit and sends the DOMAIN, MESSAGE_TYPE, and SEQUENCE_ID in the additional control information that resides in the protocol specific section of the descriptor that is transmitted to the CPSW. An event is then generated and placed on the event FIFO once the packet is transmitted. Host events allow the user to timestamp exactly when a software generated packet exits the device.

13.2.1.4.7.15 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 CPSW_CPTS_INT_ENABLE_REG register.
2. Upon interrupt, read the CPSW_CPTS_EVENT_0_REG through CPSW_CPTS_EVENT_3_REG registers values.

3. Set the CPSW_CPTS_EVENT_POP_REG[0] EVENT_POP bit to 1h to pop the previously read value off of the event FIFO.
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 CPSW_CPTS_INT_ENABLE_REG
2. Upon interrupt, enter the CPTS service routine.
3. Read the CPSW_CPTS_EVENT_0_REG through CPSW_CPTS_EVENT_3_REG registers values.
4. Set the CPSW_CPTS_EVENT_POP_REG[0] EVENT_POP bit to 1h to pop the previously read value off of the event FIFO.
5. Wait for an amount of time greater than four CPTS_RFT_CLK periods plus four CPPI_ICLK periods.
6. Read the TS_PEND_RAW bit in the CPSW_CPTS_INTSTAT_RAW_REG register to determine if another valid event is in the event FIFO. If bit TS_PEND_RAW is asserted, go to step 3. If bit TS_PEND_RAW is not asserted proceed with step 7.
7. 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 CPSW_CPTS_INTSTAT_RAW_REG register to determine if a valid event is on the event FIFO.

13.2.1.4.8 CPDMA Host Interface

The CPDMA submodule is a CPPI 3.0 and CBA 3.1 compliant packet DMA transfer controller. The CPPI interface is port 0.

13.2.1.4.8.1 Functional Operation

Host software sends and receives network frames via the CPDMA CPPI 3.0 compliant host interface. The host interface includes module registers and host memory data structures. The host memory data structures are buffer descriptors and data buffers. Buffer descriptors are data structures that contain information about a single data buffer. Buffer descriptors may be linked together to describe frames of queues of frames for transmission of data from the host to Ethernet and free buffer queues available for packet data from Ethernet to the host.

After reset, initialization, and configuration the host may initiate CPDMA host interface operations. Receive DMA operations are initiated by host writes to the appropriate receive channel head descriptor pointer. The receive DMA controller then fetches the first packet in the packet chain from memory in accordance with CPPI 3.0 protocol and proceeds with packet operations. The DMA controller fetches the packet data in 64-byte (maximum) bursts.

Host CPDMA transmit operation are initiated by host writes to the appropriate transmit channel head descriptor pointer after host initialization and configuration. The transmit DMA controller writes Ethernet received packet data to external host memory in accordance with CPPI 3.0 protocol.

13.2.1.4.8.2 Transmit CPDMA Interface

The transmit CPDMA (Ethernet to host) is an eight channel CPPI 3.0 compliant interface. Each priority/channel has a single queue for frame reception.

13.2.1.4.8.2.1 Transmit CPDMA Host Configuration

To configure the CPDMA for transmit operations the host must do the following:

1. Initialize the TX_HDP registers to 0.
2. Enable the desired transmit interrupts in the CPDMA_TX_INTMASK_SET register.
3. Write the thost_buffer_offset register value.
4. Setup the transmit channel(s) buffer descriptors in host memory as defined in CPPI 3.0.
5. Enable the CPDMA controller by setting the thost_en bit in the CPDMA_TX_CONTROL register.

13.2.1.4.8.2.2 Transmit CPDMA Buffer Descriptors

A transmit buffer descriptor is a contiguous block of four 32-bit data words aligned on a 32-bit word boundary.

Figure 13-102. TX Buffer Descriptor Format (Word 1)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16

Figure 13-102. TX Buffer Descriptor Format (Word 1) (continued)

NEXT_DESCRIPTOR_POINTER															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NEXT_DESCRIPTOR_POINTER															

Bit	Field	Description
31-0	NEXT_DESCRIPTOR_POINTER	The 32-bit word aligned memory address of the next buffer descriptor in the RX queue. This is the mechanism used to reference the next buffer descriptor from the current buffer descriptor. If the value of this pointer is zero, then the current buffer is the last buffer in the queue. Set by the host.

Figure 13-103. TX Buffer Descriptor Format (Word 2)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BUFFER_POINTER															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BUFFER_POINTER															

Bit	Field	Description
31-0	BUFFER_POINTER	The byte aligned memory address of the buffer associated with the buffer descriptor. Set by the host.

Figure 13-104. TX Buffer Descriptor Format (Word 3)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RESERVED				BUFFER_OFFSET											
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED				BUFFER_LENGTH											

Bit	Field	Description
31-28	RESERVED	Reserved
27-16	BUFFER_OFFSET	Indicates how many unused bytes are at the start of the buffer. The buffer offset is reduced to 12-bits. A value of 0x0000 indicates that there are no unused bytes at the start of the buffer and that valid data begins on the first byte of the buffer. A value of 0x000F indicates that the first 15 bytes of the buffer are to be ignored by the port and that valid buffer data starts on byte 16 of the buffer. The port writes BUFFER_OFFSET with the value from the CPDMA_TH_BUFFER_OFFSET_REG register value. The host initializes the BUFFER_OFFSET to zero for free buffers. The BUFFER_LENGTH must be greater that the CPDMA_TH_BUFFER_OFFSET_REG register value. The buffer offset is valid only on SOP.
15-12	RESERVED	Reserved
11-0	BUFFER_LENGTH	Indicates how many valid data bytes are in the buffer. The buffer length is reduced to 12-bits. Unused or protocol specific bytes at the beginning of the buffer are not counted in the Buffer Length field. The host initializes the BUFFER_LENGTH, but the port may overwrite the host initiated value with the actual buffer length value on SOP and/or EOP buffer descriptors. SOP buffer length values will overwritten if the packet size is less than the size of the buffer or if the offset is nonzero. EOP buffer length values will be overwritten if the entire buffer is not filled up with data. The BUFFER_LENGTH must be greater than zero.

Figure 13-105. TX Buffer Descriptor Format (Word 4)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SOP	EOP	OWNERSHIP	EOQ	TEARDOWN_COMPLETE	PASSED_CRC	LONG	SHORT	MAC_CTL	OVERRUN	PKT_ERR		VLAN_ENCAP	FROM_PORT		

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TS_ENCAP	MEMORY_PROTECT_ERROR	CRC_TYPE	CHKSUM_ENCAP	PACKET_LENGTH											

Bit	Field	Description
31	SOP	Start of Packet- Indicates that the descriptor buffer is the first buffer in the packet. The port sets the SOP bit. 0h - Not start of packet buffer 1h - Start of packet buffer
30	EOP	End of Packet- Indicates that the descriptor buffer is the last buffer in the packet. The port sets the EOP bit. 0h - Not end of packet buffer 1h - End of packet buffer
29	OWNERSHIP	Ownership- Indicates ownership of the packet and is valid only on SOP. This bit must be set by the host and is cleared by the port when the packet has been transferred (and the TH_OWNERSHIP bit is clear). The host uses this bit to reclaim buffers. If the TH_OWNERSHIP bit is set then the port does not clear this bit which can reduce the host workload in some applications. 0h - The packet is owned by the host 1h - The packet is owned by the port
28	EOQ	End of Queue- Set by the port to indicated that the RX queue empty condition exists. This bit is valid only on EOP. The port determines the end of queue condition by a zero NEXT_DESCRIPTOR_POINTER. 0h - The RX queue has more buffers available for reception. 1h - The Descriptor buffer is the last buffer in the last packet in the queue.
27	TEARDOWN_COMPLETE	Teardown Complete- Set by the port to indicate that the host commanded teardown process is complete, and the channel buffers may be reclaimed by the host. The bit is valid only on SOP. 0h - The port has not completed the teardown process 1h - The port has completed the commanded teardown process.
26	PASSED_CRC	Set by the port to indicate that the CRC was passed with the data. The PACKET_LENGTH includes the CRC bytes. The PASSED_CRC bit is valid only on SOP. The P0_TX_CRC_REMOVE bit in the CPDMA_CONTROL register determines if CPPI 3.0 transmit packets have a CRC included or not. The CRC type if present is determined by the P0_TX_CRC_TYPE bit in the CPDMA_Control register.
25	LONG	Jabber Frame - Indicates that the frame is a jabber frame and was not discarded because RX_CEF_EN was set in the ingress port ETH_MAC_0_PN_MAC_CONTROL_REG register. Valid only on SOP.
24	SHORT	Fragment Frame - Indicates that the frame is a fragment and was not discarded because RX_CEF_EN was set in the ingress port ETH_MAC_0_PN_MAC_CONTROL_REGregister. Valid only on SOP.

Bit	Field	Description
23	MAC_CTL	Control Frame - Indicates that the frame is a MAC control frame and was not discarded because the RX_CMF_EN was set in the ingress port ETH_MAC_0_PN_MAC_CONTROL_REG register. Valid only on SOP.
22	OVERRUN	Overrun - Set by the port to indicate that the frame reception was aborted due to transmit buffer overrun. This bit is valid only on SOP. 0h - no overrun occurred on the packet 1h - The packet was aborted due to overrun
21-20	PKT_ERROR	Packet Contained Error on Ethernet Ingress. This field is valid on SOP. 00h - no error 01h - CRC error on ingress 10h - Code error on ingress 11h - align error on ingress
19	VLAN_ECAP	VLAN Encapsulated Packet- Indicates when set that the packet data contains a 32-bit VLAN header word that is included in the packet byte count. This field is set by the port to be the value of the CPDMA_CONTROL_REG register TH_VLAN_ENCAP bit. If both VLAN_ENCAP and TS_ENCAP are set then the VLAN is first. This encapsulated word also contains the ALE classification FLOW (threadval). This bit is valid on SOP.
18-16	FROM_PORT	Indicates the Ethernet ingress port number. This field is valid only on SOP.
15	TS_ENCAP	Timestamp Encapsulated Packer - Indicates when set that the packet data contains a 64-bit timestamp (two 32-bit words with the lower 32-bit word first) that is included in the packet byte count. This field is set by the port to be the value of the CPDMA_CONTROL_REG register TH_TS_ENCAP bit. If both VLAN_ENCAP and TS_ENCAP are set then the VLAN is first. This bit is valid on SOP.
14	MEMORY_PROTECT_ERROR	An error was detected in the packet Castignoli protect CRC. The packet should be dropped by the host.
13	CRC_TYPE	The packet CRC type. 0h: Ethernet CRC 1h: Castagnoli CRC
12	CHKSUM_ENCAP	Checksum Encapsulated Packet - Indicates when set that the packet data contains 4-bytes of transmit checksum information at the end of the packet (last 4 bytes). The packet length includes the checksum bytes. This bit will be set for every packet to the Host when P0_TX_CHKSUM_EN is set. This bit is valid on SOP
11-0	PACKET_LENGTH	Specifies the number of bytes in the entire packet. Offset bytes are not included. The sum of the BUFFER_LENGTH fields should equal the PACKET_LENGTH. Valid only on SOP.

13.2.1.4.8.2.3 Transmit Channel Teardown

The host commands a transmit channel teardown by writing the channel number to the CPDMA_TX_TEARDOWN register. When a teardown command is issued to an enabled transmit channel the following will occur:

- Any frame currently in transmission will complete normally
- The TEARDOWN_COMPLETE bit will be set in the next transmit buffer descriptor (if there is one).
- The channel head descriptor pointer will be cleared to 0.
- An interrupt will be issued to inform the host of the channel teardown.
- The software should acknowledge a teardown interrupt with a FFFF FFFCh acknowledge value

Channel teardown may be commanded on any channel at any time. The host is informed of the teardown completion by the set TEARDOWN_COMPLETE buffer descriptor bit. The port does not clear any channel enables due to a teardown command. A teardown command to an inactive channel issues an interrupt that software should acknowledge with a FFFF FFFCh acknowledge value (note that there is no buffer descriptor

in this case). Software may read the interrupt acknowledge location to determine if the interrupt was due to a commanded teardown. The read value will be FFFF FFFCh if the interrupt was due to a teardown command.

13.2.1.4.8.3 Receive CPDMA Interface

The receive CPDMA is an eight channel CPPI 3.0 compliant interface. Each channel has a single queue for frame reception. Priority between the eight queues may either be fixed or round robin as selected by FH_PTYPE in the CPDMA_Control register. If the priority type is fixed, then channel 7 has the highest priority and channel 0 has the lowest priority. Round robin priority proceeds from channel 0 to channel 7. Packet Data transfers occur on the TX_VBUSP interface in 64-byte maximum burst transfers. Any packet can be designated by the host to generate a host timesync event on Ethernet egress by setting the HOST_EVENT bit in the packet buffer descriptor.

13.2.1.4.8.3.1 Receive CPDMA Host Configuration

To configure the RX CPDMA for receive operations the software must perform the following:

1. Initialize the RX_HDP registers to 0.
2. Enable the desired receive interrupts in the CPDMA_RX_INTMASK_SET register.
3. Setup the transmit channel(s) buffer descriptors in host memory as required by CPPI 3.0
4. Configure and enable the receive operation as desired in the CPDMA_RX_CONTROL register.
5. Write the appropriate RX_HDP registers with the appropriate values to start packet operations.

13.2.1.4.8.3.2 Receive DMA Host Configuration

A receive buffer descriptor is a contiguous block of four 32-bit data words aligned on a 32-bit word boundary.

Figure 13-106. RX Buffer Descriptor Format (Word 1)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
NEXT_DESCRIPTOR_POINTER															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NEXT_DESCRIPTOR_POINTER															

Bit	Field	Description
31-0	NEXT_DESCRIPTOR_POINTER	The 32-bit word aligned memory address of the next buffer descriptor in the RX queue. This is the mechanism used to reference the next buffer descriptor from the current buffer descriptor. If the value of this pointer is zero, then the current buffer is the last buffer in the queue. Set by the host.

Figure 13-107. RX Buffer Descriptor Format (Word 2)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BUFFER_POINTER															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BUFFER_POINTER															

Bit	Field	Description
31-0	BUFFER_POINTER	The byte aligned memory address of the buffer associated with the buffer descriptor. Set by the host.

Figure 13-108. RX Buffer Descriptor Format (Word 3)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BUFFER_OFFSET															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 13-108. RX Buffer Descriptor Format (Word 3) (continued)

BUFFER_LENGTH

Bit	Field	Description
32-16	BUFFER_OFFSET	Indicates how many unused bytes are at the start of the buffer. The buffer offset is reduced to 12-bits. A value of 0x0000 indicates that there are no unused bytes at the start of the buffer and that valid data begins on the first byte of the buffer. A value of 0x000F indicates that the first 15 bytes of the buffer are to be ignored by the port and that valid buffer data starts on byte 16 of the buffer. The port writes BUFFER_OFFSET with the value from the THost_BUFFER_OFFSET register value. The host initializes the BUFFER_OFFSET to zero for free buffers. The BUFFER_LENGTH must be greater that the THost_BUFFER_OFFSET register value. The buffer offset is valid only on SOP.
15-0	BUFFER_LENGTH	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. The host sets the BUFFER_LENGTH. The BUFFER_LENGTH must be greater than zero.

Figure 13-109. RX Buffer Descriptor Format (Word 4)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SOP	EOP	OWNERSHIP	EOQ	TEARDOWN_COMPLETE	PASS_CRC	CRC_TYPE	RESERVED				TO_PORT_EN	TO_PORT			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HOST_EVENT	CHKSUM_ENCAP	RESERVED		PACKET_LENGTH											

Bit	Field	Description
31	SOP	Start of Packet- Indicates that the descriptor buffer is the first buffer in the packet. The port sets the SOP bit. 0h - Not start of packet buffer 1h - Start of packet buffer
30	EOP	End of Packet- Indicates that the descriptor buffer is the last buffer in the packet. The port sets the EOP bit. 0h - Not end of packet buffer 1h - End of packet buffer
29	OWNERSHIP	Ownership- Indicates ownership of the packet and is valid only on SOP. This bit must be set by the host and is cleared by the port when the packet has been transferred and the TH_OWNERSHIP bit is zero. The host uses this bit to reclaim buffers. If the TH_OWNERSHIP bit is set then the port does not clear this bit which can reduce the host workload in some applications. 0h - The packet is owned by the host 1h - The packet is owned by the port
28	EOQ	End of Queue- Set by the port to indicated that the RX queue empty condition exists. This bit is valid only on EOP. The port determines the end of queue condition by a zero NEXT_DESCRIPTOR_POINTER on an EOP buffer. 0h - The RX queue has more buffers available for reception. 1h - The Descriptor buffer is the last buffer in the last packet in the queue.

Bit	Field	Description
27	TEARDOWN_COMPLETE	Teardown Complete- Set by the port to indicate that the host commanded teardown process is complete, and the channel buffers may be reclaimed by the host. The bit is valid only on SOP. 0h - The port has not completed the teardown process 1h - The port has completed the commanded teardown process.
26	PASS_CRC	Pass CRC - Valid only on SOP 0h - A CRC is not included with the packet data. The Ethernet port(s) will generate the CRC on Ethernet egress. A CRC (or placeholder) at the end of the data is allowed, but not required, and the BUFFER_COUNT and PACKET_LENGTH fields should not include the CRC bytes if they are present. 1h - A CRC is included with the host packet data. The PACKET_LENGTH and BUFFER_COUNT fields should include the four CRC bytes. The host SUPPLIED CRC should be in the last four bytes of the data.
25	CRC_TYPE	The packet CRC type. 0h: Ethernet CRC 1h: Castagnoli CRC
24-21	RESERVED	Reserved
20	TO_PORT_EN	To Port Enable- Indicates when set that the packet is a directed packet to be sent to the TO_PORT field port number. This field is set by the host. The packet is sent to one port only (index not mask). This bit is valid on SOP. 0h - Not a directed packet 1h - Directed packet
19-16	TO_PORT	To Port - Port number to send the directed packet to. This field is set by the host. The field is valid on SOP. Directed packets go to the directed port, but an ALE lookup is performed to determine untagged egress in VLAN_AWARE more. 1h - Send the packet to port 1 if TO_PORT_EN is asserted. 2h - Send the packet to port 2 if TO_PORT_EN is asserted.
15	HOST_EVENT	Host Timesync Event - Generate a host timesync event on Ethernet egress. The upper 28-bits of the packet SOP buffer descriptor address are the domain[7:0], message_type[3:0], and sequence_id[15:0] in that order. 0h - The packet will not generate a host event on Ethernet egress. 1h - The packet will generate a host event on Ethernet egress.
14	CHKSUM_ENCAP	Checksum Encapsulated Packet - Indicates when set that the packet data contains 4-bytes of transmit checksum information at the end of the packet (last 4 bytes). The packet length includes the checksum bytes.
13-12	RESERVED	Reserved
11-0	PACKET_LENGTH	Specifies the number of bytes in the entire packet. Offset bytes are not included. The sum of the BUFFER_LENGTH fields should equal the PACKET_LENGTH. Valid only on SOP. The packet length must be greater than zero. The packet data will be truncated to the packet length if the packet length is shorter than the sum of the packet buffer descriptor buffer lengths. A host error occurs if the packet length is greater than the sum of the packet buffer descriptor buffer lengths.

13.2.1.4.8.3.3 Receive Channel Teardown

The host commands a receive channel teardown by writing the channel number to the CPDMA_RX_TEARDOWN register. When a teardown command is issued to an enabled receive channel the following will occur:

- Any current frame in reception will complete normally.
- The TEARDOWN_COMPLETE bit will be set in the next buffer descriptor in the chain (if there is one).
- The channel head descriptor pointer will be cleared to 0.

- A receive interrupt for the channel will be issued to the host.
- The software should acknowledge a teardown interrupt with a FFFF FFFCh Acknowledge value.

Channel teardown may be commanded on any channel at any time. The host is informed of the teardown completion by a set teardown complete buffer descriptor bit. The port does not clear any channel enables due to a teardown command. A teardown command to an inactive channel issues an interrupt that software should acknowledge with a FFFF FFFCh acknowledge value (note that there is no buffer descriptor in this case). Software may read the interrupt acknowledge location to determine if the interrupt was due to a commanded teardown. The read value will be FFFF FFFCh if the interrupt was due to a teardown command.

13.2.1.4.8.3.4 Receive CPDMA Hardware Controlled Packet Transmission

When configured with hardware packet transmission the receive interface can be enabled to transfer packets due to rising edges on a channel's corresponding RX_HW_TRIG[7:0] input. Each channel has a corresponding independent internal sent_cnt[15:0] counter. To enable hardware controlled packet transmission for a channel, software sets the channel's corresponding bit in the rx_hw_trig_en[7:0] field in the CPDMA_RX_Control2 register. Hardware packet transmission then operates as described below:

1. The channel send_cnt[15:0] is cleared to zero when the channel HDP is zero (IDLE).
2. Software writes the channel HDP to begin the packet chain operation.
3. An asserted RX_HW_TRIG[7:0] input increments the associated channel sent_cnt[15:0] when the channel's HDP is non-zero.
4. A single packet is transferred when send_cnt is greater than 0 and then the send_cnt is decremented.
5. Go to IDLE (#1) on EOQ (which also zeroes the HDP), otherwise continue with packet transmission (#4).

Note

- Each channel has an associate send_cnt[15:0]. the send_cnt[15:0] register will not overflow or underflow.
- The RX_HW_TRIG[7:0] inputs are asynchronous. They are synchronized and rising edge detected by the CPTS_RFTCLK. The pulse must be asserted high long enough for the high to be seen by the synchronizer, and asserted low long enough for the low to be seen by the synchronizer.
- A rising edge on the RX_HW_TRIG bit increments the count regardless of the status of any previous packet transfer when the head descriptor pointer is nonzero.

13.2.1.4.8.4 VLAN Aware Mode

The CPSW is in VLAn Aware mode when the CPSW Control register vlan_aware bit is set. In VLAN aware mode port 0 transmit packets may or may not be VLAN encapsulated depending on the CPSW Control register TX_VLAN_ENCAP bit. The header packet VLAN is generated as described in later sections of this specification. VLAN encapsulated packets are specified by a set VLAN_ENCAP bit in the packet buffer descriptor. The VLAN encapsulation header is included in the packet length and has the below format:

Figure 13-110. 32-bit VLAN Header Encapsulation Word Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
HDR_PKT_PRIORITY			HDR_PKT_CFI	HDR_PKT_VID											
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FLOW						PKT_TYPE		RESERVED							

Bit	Field	Description
31-29	HDR_PKT_PRIORITY	Header Packet VLAN priority (7 is highest priority)
28	HDR_PKT_CFI	Header Packet VLAN CFI bit
27-16	HDR_PKT_VID	Header Packet VLAN ID

Bit	Field	Description
15-10	FLOW	FLOW - A nonzero value indicates that the ALE matched a classifier with the FLOW (threadval)
9-8	PKT_TYPE	Packet Type - Indicates whether the packet is a VLAN tagged, priority tagged, or non-tagged packet. 00h - VLAN tagged packet 01h - Reserved 10h - priority tagged packet 11h - non-tagged packet
7-0	RESERVED	Reserved

13.2.1.4.8.5 VLAN Unaware Mode

The CPSW is in VLAN unaware mode when the CPSW Control register vlan_aware bit is cleared. Port 0 transmit packets (egress) may or may not be VLAn encapsulated depending on the CPSW Control register TX_VLAN_ENCAP bit.

13.2.1.4.8.6 CPDMA Big Endian Mode

When the CPSW_BIG_ENDIAN input is asserted, the CPDMA assumes that the packet data is contained in memory in big endian format. When the CPDMA_BIG_ENDIAN input is deasserted, the CPDMA assumes that packet data is contained in memory in little endian format. The CPDMA_BIG_ENDIAN input causes big endian packet data to go out on the wire in the same order that the little endian packet data goes out on the wire when the input is not asserted (byte 0 first). The CPDMA_BIG_ENDIAN input has no effect on buffer descriptor data reads or writes because buffer descriptor data is a 32-bit quantity (unlike packet data which is an 8-bit quantity).

Table 13-158. Little Endian

High Add			Low Add
Byte 3	Byte 2	Byte 1	Byte 0
Byte 7	Byte 6	Byte 5	Byte 4
Byte 11	Byte 10	Byte 9	Byte 8
			...

Table 13-159. Big Endian

High Add			Low Add
Byte 0	Byte 1	Byte 2	Byte 3
Byte 4	Byte 5	Byte 6	Byte 7
Byte 8	Byte 9	Byte 10	Byte 11
			...

13.2.1.4.8.7 CPDMA Command IDLE

The cmd_idle bit in the CPDMA_Control register allows CPDMA operation to be suspended. When the idle state is commanded, the CPDMA will stop processing transmit and receive frames at the next frame boundary. Any frame currently in reception or transmission will be completed normally without suspension. For receive, and frame in process will be completed. For transmit, frames that are detected by the CPDMA after the suspend state is entered are ignored. No statistics will be kept for ignored frames. Commanded idle is similar in operation to emulation control and clock stop.

13.2.1.4.8.8 CPDMA CPPI 3.0 Interface Bandwidth

The HOST CPPI 3.0 Receive and Transmit interfaces are capable of supporting linerate on the Ethernet ports provided that the clock frequency is sufficient, and provided that the Host master VBUSP read/write latency is low.

13.2.1.4.9 Cut-Thru

An Ethernet port received packet can be sent cut-thru to all destination ports when the following is true:

- The CPSW_CONTROL_REG[19] CUT_THRU_ENABLE bit is set, and
- The receive port is full duplex, and
- The Ethernet receive port speed is non-zero, and
- The packet was received on an express priority, and
- The Ethernet receive port has the remapped received packet priority enabled via the CPSW_PN_CUT_THRU_REG_k[15-8] RX_PRI_CUT_THRU_EN field, and
- All Ethernet destination ports have the remapped packet priority enabled via the CPSW_PN_CUT_THRU_REG_k[7-0] TX_PRI_CUT_THRU_EN field, and
- All Ethernet destination ports are full duplex, and
- All Ethernet destination ports are express priorities, and
- All Ethernet destination ports have a non-zero CPSW_PN_SPEED_REG_k[3-0] SPEED, and
- All Ethernet destination ports are equal to or slower in speed than the Ethernet receive port, and
- The host port is not in the destination port list unless the CPSW_P0_CONTROL_REG[19] CUT_MODE_ETH bit is set.

Whether or not a packet actually egresses cut-thru (with or without delay) on an Ethernet destination port depends on traffic congestion on that port. Cut-thru is compatible with Enhanced Scheduled Traffic (EST).

13.2.1.4.9.1 Host Port Cut-Thru Operations

Packets received (CPSW ingress) on the host port are sent store-and-forward to all Ethernet destination ports. Setting the CPSW_P0_CONTROL_REG[19] CUT_MODE_ETH bit enables Ethernet received packets to be sent cut-thru to a destination port mask that includes the host port. The packet can egress cut-thru on Ethernet ports, but the packet will not actually egress cut-thru on the host port. Setting CPSW_P0_CONTROL_REG[19] CUT_MODE_ETH can cause head of line blocking on the host priority if multiple Ethernet ports are sending to the same host port priority. The head of line blocking issue is worsened if there are differing Ethernet port speeds. Clearing the CPSW_P0_CONTROL_REG[19] CUT_MODE_ETH bit causes packets with the host port in the destination mask to egress store-and-forward on all egress ports.

13.2.1.4.9.2 Cut-Thru Error Packets

Any received packets with errors sent cut-thru from an Ethernet receive port to any Ethernet transmit port(s) will egress with at least one byte of the generated outgoing packet CRC inverted to indicate the error. This is due to the fact that cut-thru operations begin before the end of packet when the receive port determines that the packet had an error (long/code/align/CRC).

Any packet also sent cut-thru with the host included in the destination port mask will be dropped to the host (TXST_PKT_DROP) with the packet error indicated on TXST_PKT_ERR[3:0].

Any long/Jabber/Code/Align/CRC errored Ethernet received packet that is not actually sent cut-thru by the receive port will dropped unless CPSW_PN_MAC_CONTROL_REG_k[22] RX_CEF_EN is set (which if set will cause the packet to be sent to the host port with the error indicated on TXST_PKT_ERR[3:0]. If a long/Jabber/Code/Align/CRC errored Ethernet received packet is sent cut-thru by the received port then a set CPSW_PN_MAC_CONTROL_REG_k[22] RX_CEF_EN will not cause the errored packet to be sent to the host port since the packet has already been transferred to the destination ports.

The following cut-thru notes are valid:

- Any Ethernet received packet and that is sent cut-thru to any destination port will not be dropped at any destination port due to priority maximum lengths (CPSW_TX_PRIx_MAXLEN).
- Any Ethernet received packet that is decoded as a timesync packet will be sent store-and-forward to all destination ports.
- Cut-Thru is full duplex only.
- Cut-Thru is not compatible with any form of flow control.
- Cut-Thru should not be enabled for any receive port that is in ALE bypass mode.

13.2.1.4.10 CPPI Checksum Offload

The CPPI host port can be enabled to perform checksum offload on host port packet ingress and egress. UDP (User Datagram Protocol) and TCP (Transmission Control Protocol) over IPV4 and IPV6 are supported. For the purposes of checksum description, the first packet byte (the first byte of the destination address) is byte 1 (not byte 0). That is, a 64 byte packet goes from byte 1 to byte 64. For all packet types, the S_CN_SWITCH bit in the CPSW_CONTROL_REG register must be set for the Outer VLAN L type to be supported.

13.2.1.4.10.1 CPPI Transmit Checksum Offload

IPV4 and IPV6 UDP and TCP packets that are received on any Ethernet port and destined for port 0 egress are checked for correct checksum as described below. The EOP Transmit buffer descriptor bit CHKSUM_ENCAP indicates whether or not the transmit checksum information is included with the egress packet or not. If the checksum information is included in the packet, the PACKET_LENGTH includes the four checksum information bytes. The byte counts below are shown for packets with no VLAN's. The byte counts vary with one or two packet VLAN's. Packets received on an Ethernet port with errors are not checked for a correct checksum if they are passed to the host (no checksum information with the error packet).

13.2.1.4.10.1.1 IPV4 UDP

- Byte 15 Upper Nibble = 4 for IPV4
- Byte 15 Lower Nibble = IHL - Nibble with number of 32-bit words in IPV4 header (5 to 15 supported).
- Bytes 20-21 = fragment[15-0] – Bit 13 is the MF bit and bits [12-0] are the Fragment offset. A packet is a fragment if the MF bit is set or if the fragment offset is non-zero. The first packet fragment has MF=1 with a zero offset. Middle fragments have MF=1 with a nonzero offset. The last packet fragment has MF=0 with a nonzero offset. Non-fragmented packets have MF=0 and a zero offset. A count is output for packet fragments but no errors are reported. First fragments have the UDP header included in the count. Middle and last fragments have only data included in the count (there is no UDP header).
- Byte 24 = 0x11 for UDP protocol.
- Received packet UDP checksum of zero means that there is no IPV4 checksum sent with the packet so no error will be issued.
- Received packet UDP checksum of 0xFFFF means that the checksum was calculated to be 0xFFFF or 0x0000 but was sent in the transmitted packet as 0xFFFF by the sending originating entity.

13.2.1.4.10.1.2 IPV4 TCP

- Byte 15 Upper Nibble = 4 for IPV4
- Byte 15 Lower Nibble = IHL - Nibble with number of 32-bit words in IPV4 header (5 to 15 supported).
- Bytes 20-21 = fragment[15-0] – Bit 13 is the MF bit and bits [12-0] are the Fragment offset. A packet is a fragment if the MF bit is set or if the fragment offset is non-zero. The first packet fragment has MF=1 with a zero offset. Middle fragments have MF=1 with a nonzero offset. The last packet fragment has MF=0 with a nonzero offset. Non-fragmented packets have MF=0 and a zero offset. A count is output for packet fragments but no errors are reported. First fragments have the UDP header included in the count. Middle and last fragments have only data included in the count (there is no TCP header).
- Byte 24 = 0x06 for TCP protocol.

13.2.1.4.10.1.3 IPV6 UDP

- Byte 15 upper nibble = 6 for IPV6.
- Byte 21 = 0x11 for UDP protocol as next header.
- Fragment extension headers are supported. First fragments have a fragment extension header (byte 21 = 0x2C) followed by a UDP header (byte 55 = 0x11). Middle and last fragments have a fragment extension header followed by data only (no UDP header). The first packet fragment has MF=1 with a zero offset. Middle fragments have MF=1 with a nonzero offset. The last packet fragment has MF=0 with a nonzero offset. Non-fragmented packets do not have a fragment extension header. A count is output for packet fragments but no errors are reported.
- Received packet UDP checksum of zero means that there is no IPV6 checksum sent with the packet so no error will be issued.
- Received packet UDP checksum of 0xFFFF means that the checksum was calculated to be 0xFFFF or 0x0000 but was sent in the transmitted packet as 0xFFFF by the sending originating entity.

13.2.1.4.10.1.4 IPV6 TCP

- Byte 15 upper nibble = 6 for IPV6.
- Byte 21 = 0x06 for TCP protocol as next header.
- Fragment extension headers are supported. First fragments have a fragment extension header (byte 21 = 0x2C) followed by a UDP header (byte 55 = 0x06). Middle and last fragments have a fragment extension header followed by data only (no TCP header). The first packet fragment has MF=1 with a zero offset. Middle fragments have MF=1 with a nonzero offset. The last packet fragment has MF=0 with a nonzero offset. Non-fragmented packets do not have a fragment extension header. A count is output for packet fragments but no errors are reported.

13.2.1.4.10.1.5 Transmit Checksum Encapsulation Word

The 4-byte checksum encapsulation word is included as the last 4-bytes of the transmit packet data when EOP buffer descriptor CHKSUM_ENCAP is set. The PACKET_LENGTH includes the four encapsulation bytes.

Figure 13-111. Transmit Checksum Encapsulation Word Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RESERVED											IPV4_VALID	IPV6_VALID	TCP_UDP_N	FRAGMENT	CHKSUM_ERROR
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CHECKSUM_ADD															

Bit	Field	Description
31-21	RESERVED	Reserved
20	IPV4_VALID	An IPV4 TCP or UDP packet was detected
19	IPV6_VALID	An IPV6 TCP or UDP Packet was detected
18	TCP_UDP_N	TCP or UDP packet - Valid only when either the IPV4_VALID or IPV6_VALID bits are set 0h - Indicates UDP packet was detected 1h - Indicates TCP packet was detected
17	FRAGMENT	Indicates that an IP fragment was detected. Valid only when either the IPV4_VALID or IPV6_VALID bits are set.
16	CHKSUM_ERROR	Checksum Error detected. Valid only when either the IPV4_VALID or IPV6_VALID bits are set.
15-0	CHECKSUM_ADD	Checksum Add Value - this is the value that was summed during the checksum computation. This value is 0xFFFF for IPV4/6 UDP/TCP packets with no checksum error.

13.2.1.4.10.2 CPPI Receive Checksum Offload

Packets sent from host port 0 (switch ingress) to any Ethernet port can have a checksum calculated and inserted into the Ethernet egress packet. The RX_CHECKSUM_EN bit in the CPSW_P0_CONTROL_REG register must be set for receive checksum operation to be enabled. When bit RX_CHECKSUM_EN is enabled and when the CHKSUM_ENCAP SOP receive buffer descriptor is set, the first four packet bytes contain the checksum information which determines how the checksum is calculated. The CHECKSUM_RESULT field determines where the checksum is inserted in the egress packet. The checksum result location is adjusted by the egress port if a VLAN is to be inserted or removed on Ethernet port egress.

13.2.1.4.10.2.1 Receive Checksum Encapsulation Word

The 4-byte checksum encapsulation word is included as the first 4-bytes of the receive packet data when SOP buffer descriptor CHKSUM_ENCAP is set. The PACKET_LENGTH includes the four encapsulation bytes.

Figure 13-112. Receive Checksum Encapsulation Word Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CHECKSUM_RESULT								CHECKSUM_START_BYTE							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CHKSUM_INV	RESERVED	CHECKSUM_BYTECOUNT													

Bit	Field	Description
31-24	CHECKSUM_RESULT	Checksum Result Byte Location. This is the packet byte number where the checksum result will be placed in the egress packet. The first packet byte which is the first byte of the destination address is byte 1 (not byte zero).
23-16	CHECKSUM_START_BYTE	Checksum Start Byte. This is the packet byte number to start the checksum calculation on. The first packet byte is byte 1.
15	CHKSUM_INV	Checksum Invert Zero. When set, a zero checksum value will be inverted and sent as 0xFFFF.
14	RESERVED	Reserved
13-0	CHECKSUM_BYTECOUNT	Checksum Byte Count. This is the number of bytes to calculate the checksum on. The outgoing Ethernet packet will have a checksum inserted when this value is non-zero.

13.2.1.4.11 Egress Packet Operations

Each CPSW egress port (Ethernet and Host) is capable of performing egress packet processing operations (CPSW_ALE_EGRESSOP). IntraVLAN processing either adds, removes, or replaces VLAN information or does nothing. InterVLAN routing allows hardware routing between a limited number of VLANs - thereby allowing high-bandwidth or other routing operations to be offloaded from software to the CPSW (hardware). IntraVLAN processing and InterVLAN routing operations are mutually exclusive. In addition, the packet source and destination addresses can be swapped on egress to facilitate OAM or generic testing operations.

13.2.1.4.12 MII Management Interface (MDIO)

The MII Management interface module implements the 802.3 serial management interface to interrogate and control external Ethernet PHY using a two-wire bus.

13.2.1.4.12.1 MDIO Frame Formats

Table 13-160 shows the address, Table 13-161 shows the read format and Table 13-162 shows the write format of the supported Clause 45 MII Management interface frames. Post-increment accesses are not supported.

Table 13-160. MDIO Clause 45 Address Frame Format

Pre-amble	Start Delimiter	Operation Code	PHY Address	MMD Number	Turnaround	Data
FFFF FFFFh	00	00	AAAAA	RRRRR	10	AAAA.AAAA.AAAA.AAAA

Table 13-161. MDIO Clause 45 Read Frame Format

Pre-amble	Start Delimiter	Operation Code	PHY Address	MMD Number	Turnaround	Data
FFFF FFFFh	00	11	AAAAA	RRRRR	Z0	DDDD.DDDD.DDDD.DDDD

Table 13-162. MDIO Clause 45 Write Frame Format

Pre-amble	Start Delimiter	Operation Code	PHY Address	MMD Number	Turnaround	Data
FFFF FFFFh	00	01	AAAAA	RRRRR	10	DDDD.DDDD.DDDD.DDDD

The default or idle state of the two wire serial interface is a logic one. All tri-state drivers should be disabled and the PHY's pull-up resistor will pull the MDIO line to a logic 1. Prior to initiating any other transaction, the station management entity shall send a preamble sequence of 32 contiguous logic 1 bits on the MDIO line with 32 corresponding cycles on MDCLK to provide the PHY with a pattern that it can use to establish synchronization. A PHY shall observe a sequence of 32 contiguous logic one bits on MDIO with 32 corresponding MDCLK cycles before it responds to any other transaction. The MDIO CPSW_MDIO_USER_ADDR0_REG register must be written before a read or write operation is performed to set the address used in the operation. Each read or write operation has a preceeding address frame.

Preamble

The start of a frame is indicated by a preamble, which consists of a sequence of 32 contiguous bits all of which are a 1. This sequence provides the PHY a pattern to use to establish synchronization. The preamble is required in clause 45 operation.

Start Delimiter

The preamble is followed by the start delimiter which is indicated by a 00 pattern.

Operation Code

The operation code for an address transaction is 00. The operation code for a read is 11, while the operation code for a write is a 01.

PHY Address

The PHY address is 5 bits allowing 32 unique values. The first bit transmitted is the MSB of the PHY address.

MMD Number

The MMD number is the 5 bits allowing 32 unique values. The first bit transmitted is the MSB.

Turnaround

An idle bit time during which no device actively drives the MDIO signal shall be inserted between the register address field and the data field of a read frame in order to avoid contention. During a read frame, the PHY shall drive a zero bit onto MDIO for the first bit time following the idle bit and preceding the Data field. During a write frame, this field shall consist of a one bit followed by a zero bit.

Address

The address field is 16 bits on address operations. The first bit transmitted is the MSB of the address word. Each read/write operation initiated has an automatic address operation initiated first that uses the MDIO CPSW_MDIO_USER_ADDR0_REG/ CPSW_MDIO_USER_ADDR1_REG register values as the 16-bit address.

Data

The Data field is 16 bits on read and write operations. The first bit transmitted and received is the MSB of the data word.

13.2.1.4.12.2 MDIO Functional Description

The MII Management I/F will remain idle until enabled by setting the ENABLE bit in the CPSW_MDIO_CONTROL_REG register. The MII Management I/F will then continuously poll the link status from within the Generic Status Register of all possible 32 PHY addresses in turn recording the results in the MDIO CPSW_MDIO_LINK_REG register. Individual PHY's can be enabled or disabled for polling the associated bit in the CPSW_MDIO_POLL_EN_REG register. The CPSW_MDIO_LINK_REG and CPSW_MDIO_ALIVE_REG register bit values are updated on the poll of each PHY. The LINKSEL bit in the CPSW_MDIO_USER_PHY_SEL_REG_k register determines the status input that is used. A change in the link status of the two PHYs being monitored will set the appropriate bit in the MDIO CPSW_MDIO_LINK_INT_RAW_REG register and the MDIO CPSW_MDIO_LINK_INT_MASKED_REG register, if enabled by the LINKINT_ENABLE bit in the CPSW_MDIO_USER_PHY_SEL_REG_k register.

The MDIO CPSW_MDIO_ALIVE_REG register is updated by the MII Management I/F module if the PHY acknowledged the read of the generic status register. In addition, any PHY register read transactions initiated by the host also cause the MDIO CPSW_MDIO_ALIVE_REG register to be updated.

At any time, the host can define a transaction for the MII Management interface module to undertake using the DATA, PHYADR, REGADR, and WRITE fields in a CPSW_MDIO_USER_ACCESS_REG_k register. When the host sets the GO bit in this register, the MII Management interface module will begin the transaction without any further intervention from the host. Upon completion, the MII Management interface will clear the GO bit and set the USERINTRAW field in the CPSW_MDIO_USER_INT_RAW_REG register corresponding to the CPSW_MDIO_USER_ACCESS_REG_k register being used. The corresponding bit in the CPSW_MDIO_USER_INT_MASKED_REG register may also be set depending on the mask setting in the MDIO CPSW_MDIO_USER_INT_MASK_SET_REG and CPSW_MDIO_USER_INT_MASK_CLEAR_REG registers. A round-robin arbitration scheme is used to schedule transactions that may be queued by the host in different CPSW_MDIO_USER_ACCESS_REG_k registers. The host should check the status of the GO bit in the MDIO CPSW_MDIO_USER_ACCESS_REG_k register before initiating a new transaction to ensure that the previous transaction has completed. The host can use the ACK bit in the MDIO CPSW_MDIO_USER_ACCESS_REG_k register to determine the status of a read transaction.

It is necessary for software to use the MII Management interface module to setup the auto-negotiation parameters of each PHY attached to a MAC port, retrieve the negotiation results, and setup the CPSW_PN_MAC_CONTROL_REG register in the corresponding MAC.

13.2.1.5 CPSW0 Programming Guide

13.2.1.5.1 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_ENET1_CTRL and CTRLMMR_ENET2_CTRL[2-0] PORT_MODE_SEL fields.
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_k and CPSW_PN_SA_H_REG_k)
7. Configure the CPSW statistic port enable register CPSW_STAT_PORT_EN_REG

8. Configure the ALE (Section 13.2.1.4.6.1, *Address Lookup Engine*)
9. Configure the MDIO (Section 13.2.1.5.5.1, *Initializing the MDIO Module*)
10. Configure Ethernet port, as per the desired mode of operations

13.2.1.5.2 Transmit Operation

After reset, the host must write zeroes to all TX DMA State head descriptor pointers. The TX port may then be enabled. To initiate packet transmission the host constructs transmit queues in memory (one or more packets for transmission) and then writes the appropriate TX DMA state head descriptor pointers. For each buffer added to a transmit queue, the host must initialize the TX buffer descriptor values as follows:

1. Write the Next Descriptor Pointer with the 32-bit aligned address of the next descriptor in the queue (zero if last descriptor)
2. Write the Buffer Pointer with the byte aligned address of the buffer data
3. Write the Buffer Length with the number of bytes in the buffer
4. Write the Buffer Offset with the number of bytes in the offset to the data (nonzero with SOP only)
5. Set the SOP, EOP, and Ownership bits as appropriate
6. Clear the End Of Queue bit

The port begins TX packet transmission on a given channel when the host writes the channel's TX queue head descriptor pointer with the address of the first buffer descriptor in the queue (nonzero value). Each channel may have one or more queues, so each channel may have one or more head descriptor pointers. The first buffer descriptor for each TX packet must have the Start of Packet (SOP) bit and the Ownership bit set to one by the host. The last buffer descriptor for each TX packet must have the End of Packet (EOP) bit set to one by the host. The port will transmit packets until all queued packets have been transmitted and the queue(s) are empty. When each packet transmission is complete, the port will clear the Ownership bit in the packet's SOP buffer descriptor and issue an interrupt to the host by writing the packet's last buffer descriptor address to the queue's TX DMA State Completion Pointer. The interrupt is generated by the write, regardless of the value written. When the last packet in a queue has been transmitted, the port sets the End Of Queue bit in the EOP buffer descriptor, clears the Ownership bit in the SOP Descriptor, zeroes the appropriate DMA state head descriptor pointer, and then issues a TX interrupt to the host by writing to the queue's associated TX completion pointer (address of the last buffer descriptor processed by the port). The port issues a maskable level interrupt (which may then be routed through external interrupt control logic to the host).

On interrupt from the port, the host processes the buffer queue, detecting transmitted packets by the status of the Ownership bit in the SOP buffer descriptor. If the Ownership bit is cleared to zero, then the packet has been transmitted and the host may reclaim the buffers associated with the packet. The host continues queue processing until the end of the queue or until a SOP buffer descriptor is read that contains a set Ownership bit indicating that the packet transmission is not complete. The host determines that all packets in the queue have been transmitted when the last packet in the queue has a cleared Ownership bit in the SOP buffer descriptor, the End of Queue bit is set in the last packet EOP buffer descriptor, and the Next Descriptor Pointer of the last packet EOP buffer descriptor is zero. The host acknowledges an interrupt by writing the address of the last buffer descriptor to the queue's associated TX Completion Pointer in the TX DMA State. If the host written buffer address value is different from the buffer address written by the port, then the level interrupt remains asserted. If the host written buffer address value is equal to the port written value, then the level interrupt is de-asserted. The port write to the completion pointer actually stores the value in the state register (RAM). The host written value is actually not written to the register location. The host written value is compared to the register contents (which was written by the port) and if the two values are equal, the interrupt is removed, otherwise the interrupt remains asserted. The host may process multiple packets previous to acknowledging an interrupt, or the host may acknowledge interrupts for every packet.

A mis-queued packet condition may occur when the host adds a packet to a queue for transmission as the port finishes transmitting the previous last packet in the queue. The mis-queued packet is detected by the host when queue processing detects a cleared Ownership bit in the SOP buffer descriptor, a set End of Queue bit in the EOP buffer descriptor, and a nonzero Next Descriptor Pointer in the EOP buffer descriptor. A mis-queued packet means that the port read the last EOP buffer descriptor before the host added the new last packet to the queue, so the port determined queue empty just before the last packet was added. The host corrects the mis-queued packet condition by initiating a new packet transfer for the mis-queued packet by writing the mis-queued packet's SOP buffer descriptor address to the appropriate DMA State TX Queue head Descriptor Pointer.

The host may add packets to the tail end of an active TX queue at any time by writing the Next Descriptor Pointer to the current last descriptor in the queue. If a TX queue is empty (inactive), the host may initiate packet transmission at any time by writing the appropriate TX DMA State head descriptor pointer. The host software should always check for and reinitiate transmission for mis-queued packets during queue processing on interrupt from the port. In order to preclude software underrun, the host should avoid adding buffers to an active queue for any TX packet that is not complete and ready for transmission.

The port determines that a packet is the last packet in the queue by detecting the End of Packet bit set with a zero Next Descriptor Pointer in the packet buffer descriptor. If the End of Packet bit is set and the Next Descriptor Pointer is nonzero, then the queue still contains one or more packets to be transmitted. If the EOP bit is set with a zero Next Descriptor Pointer, then the port will set the EOQ bit in the packet's EOP buffer descriptor and then zero the appropriate head descriptor pointer previous to interrupting the port (by writing the completion pointer) when the packet transmission is complete.

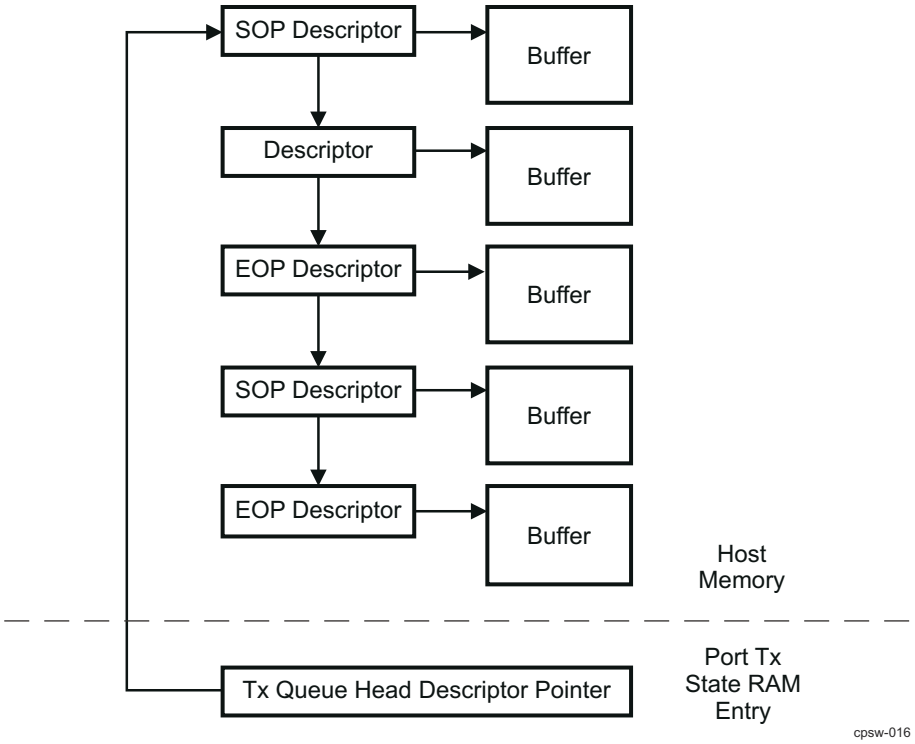


Figure 13-113. TX Queue Head Descriptor

13.2.1.5.3 Receive Operation

After reset, the host must write zeroes to all RX DMA State head descriptor pointers. The RX port may then be enabled. To initiate packet reception, the host constructs receive queues in memory and then writes the appropriate RX DMA state head descriptor pointer. For each RX buffer descriptor added to the queue, the host must initialize the RX buffer descriptor values as follows:

1. Write the Next Descriptor Pointer with the 32-bit aligned address of the next descriptor in the queue (zero if last descriptor)
2. Write the Buffer Pointer with the byte aligned address of the buffer data
3. Clear the Offset field
4. Write the Buffer Length with the number of bytes in the buffer
5. Clear the SOP, EOP, and EOQ bits
6. Set the Ownership bit

The host enables packet reception on a given channel by writing the address of the first buffer descriptor in the queue (nonzero value) to the channel's head descriptor pointer in the channel's RX DMA state. When packet

reception begins on a given channel, the port fills each RX buffer with data in order starting with the first buffer and proceeding through the RX queue. If the Buffer Offset in the RX DMA State is nonzero, then the port will begin writing data after the offset number of bytes in the SOP buffer. The port performs the following operations at the end of each packet reception:

1. Overwrite the buffer length in the packet's EOP buffer descriptor with the number of bytes actually received in the packet's last buffer. The host initialized value is the buffer size. The overwritten value will be less than or equal to the host initialized value.
2. Set the EOP bit in the packet's EOP buffer descriptor.
3. Set the EOQ bit in the packet's EOP buffer descriptor if the current packet is the last packet in the queue.
4. Overwrite the packet's SOP buffer descriptor Buffer Offset with the RX DMA state value (the host initialized the buffer descriptor Buffer Offset value to zero). All non SOP buffer descriptors must have a zero Buffer Offset initialized by the host.
5. Overwrite the packet's SOP buffer descriptor buffer length with the number of valid data bytes in the buffer. If the buffer is filled up, the buffer length will be the buffer size minus buffer offset.
6. Set the SOP bit in the packet's SOP buffer descriptor.
7. Write the SOP buffer descriptor Packet Length field.
8. Clear the Ownership bit in the packet's SOP buffer descriptor.
9. Issue an RX host interrupt by writing the address of the packet's last buffer descriptor to the queue's RX DMA State Completion Pointer. The interrupt is generated by the write to the RX DMA State Completion Pointer address location, regardless of the value written.

On interrupt the host processes the RX buffer queue detecting received packets by the status of the Ownership bit in each packet's SOP buffer descriptor. If the Ownership bit is cleared then the packet has been completely received and is available to be processed by the host. The host may continue RX queue processing until the end of the queue or until a buffer descriptor is read that contains a set Ownership bit indicating that the next packet's reception is not complete. The host determines that the RX queue is empty when the last packet in the queue has a cleared Ownership bit in the SOP buffer descriptor, a set End of Queue bit in the EOP buffer descriptor, and the Next Descriptor Pointer in the EOP buffer descriptor is zero.

A mis-queued buffer may occur when the host adds buffers to a queue as the port finishes the reception of the previous last packet in the queue. The mis-queued buffer is detected by the host when queue processing detects a cleared Ownership bit in the SOP buffer descriptor, a set End of Queue bit in the EOP buffer descriptor, and a nonzero Next Descriptor Pointer in the EOP buffer descriptor. A mis-queued buffer means that the port read the last EOP buffer descriptor before the host added buffer descriptor(s) to the queue, so the port determined queue empty just before the host added more buffer descriptor(s). In the transmit case, the packet transmission is delayed by the time required for the host to determine the condition and reinitiate the transaction, but the packet is not actually lost. In the receive case, receive overrun condition may occur in the mis-queued buffer case. If a new packet reception is begun during the time that the port has determined the end of queue condition, then the received packet will overrun (start of packet overrun). If the mis-queued buffer occurs during the middle of a packet reception then middle of packet overrun may occur. If the mis-queued buffer occurs after the last packet has completed, and is corrected before the next packet reception begins, then overrun will not occur. The host acts on the mis-queued buffer condition by writing the added buffer descriptor address to the appropriate RX DMA State Head Descriptor Pointer.

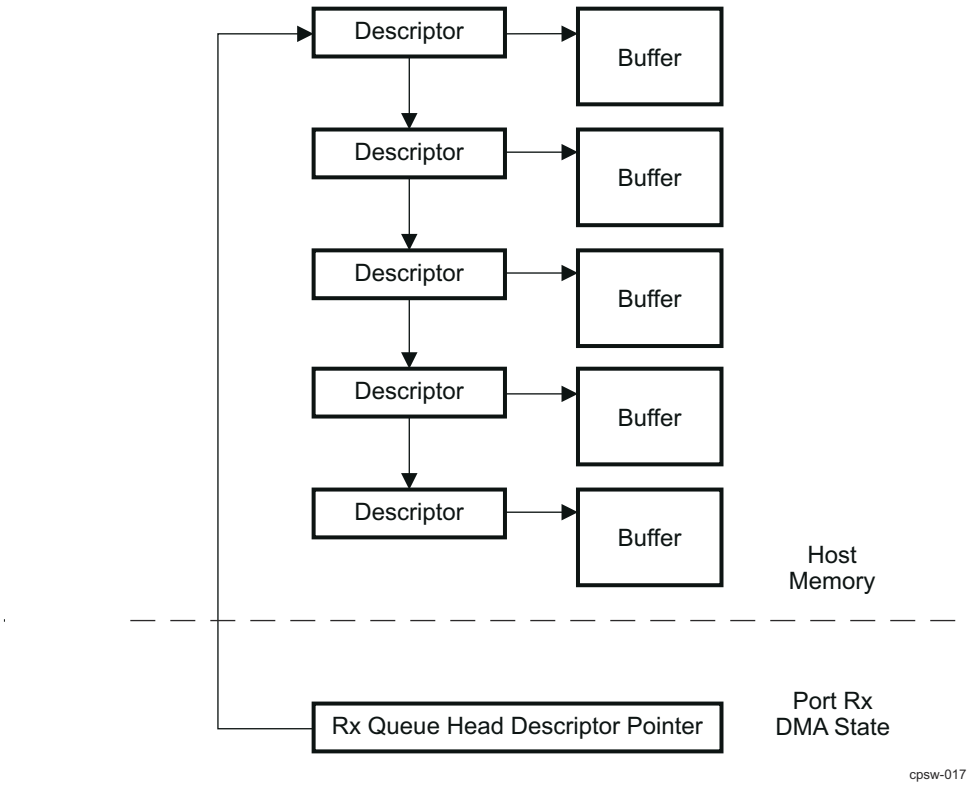


Figure 13-114. RX Queue Head Descriptor

13.2.1.5.4 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.

13.2.1.5.5 MDIO Software Interface

13.2.1.5.5.1 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).

13.2.1.5.5.2 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.

13.2.1.5.5.3 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.

13.3 Memory Interfaces

This section describes the memory interfaces in the device.

13.3.1 Multimedia Card (MMC)

This chapter describes the MMC of the device.

13.3.1.1 Introduction

13.3.1.1.1 MMCSD Features

The general features of the MMCSD host controller IP are:

- Built-in 1024-byte buffer for read or write
- Two DMA channels, one interrupt line
- Clock support
  - 96-MHz functional clock source input
  - up to 384Mbit/sec (48MByte/sec) in MMC mode 8-bit data transfer
  - up to 192Mbit/sec (24MByte/sec) in High-Speed SD mode 4-bit data transfer
  - up to 24Mbit/sec (3MByte/sec) in Default SD mode 1-bit data transfer
- Support for SDA 3.0 Part A2 programming model
- Serial link supports full compliance with:
  - MMC command/response sets as defined in the MMC standard specification v4.3.
  - SD command/response sets as defined in the SD Physical Layer specification v2.00
  - SDIO command/response sets and interrupt/read-wait suspend-resume operations as defined in the SD part E1 specification v 2.00
  - SD Host Controller Standard Specification sets as defined in the SD card specification Part A2 v2.00

13.3.1.1.2 Unsupported MMCSD Features

The MMCSD module features not supported in this device are shown in Table 13-163.

Table 13-163. Unsupported MMCSD Features

Feature	Reason
MMC Out-of-band interrupts	MMC_OBI input tied low
Master DMA operation	Disabled through synthesis parameter
Card Supply Control (MMCSD(1-2))	Signal not pinned out
Dual Data Rate (DDR) mode	Timing not supported

13.3.1.2 Integration

This device contains one instance of the Multimedia Card (MMC), Secure Digital (SD), and Secure Digital I/O (SDIO) high speed interface module (MMCSD). The controller provides an interface to an MMC, SD memory card or SDIO card.

The application interface is responsible for managing transaction semantics; the MMC/SDIO host controller deals with MMC/SDIO protocol at transmission level, packing data, adding CRC, start/end bit and checking for syntactical correctness. Figure 13-115 through Figure 13-117 below show examples of systems using the MMCSD controller.

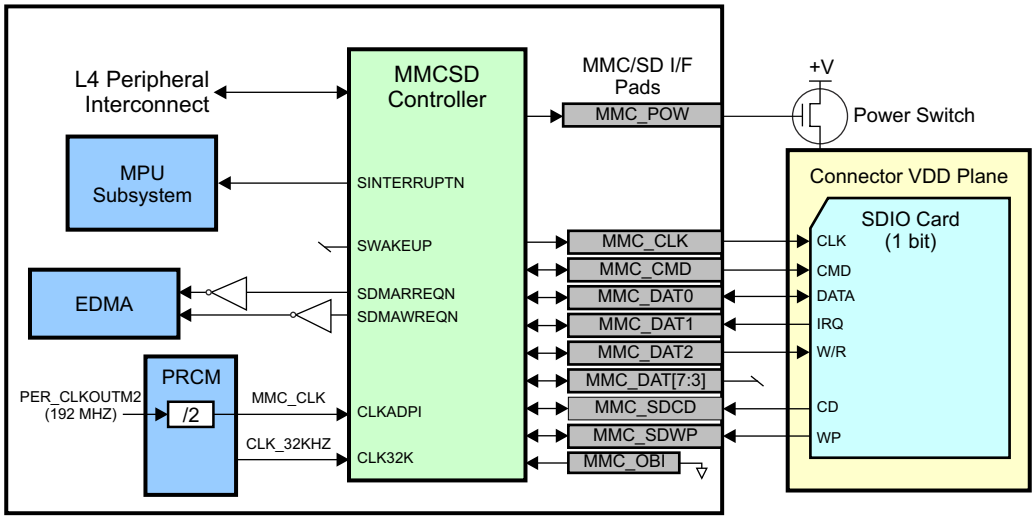


Figure 13-115. MMCSD Module SDIO Application

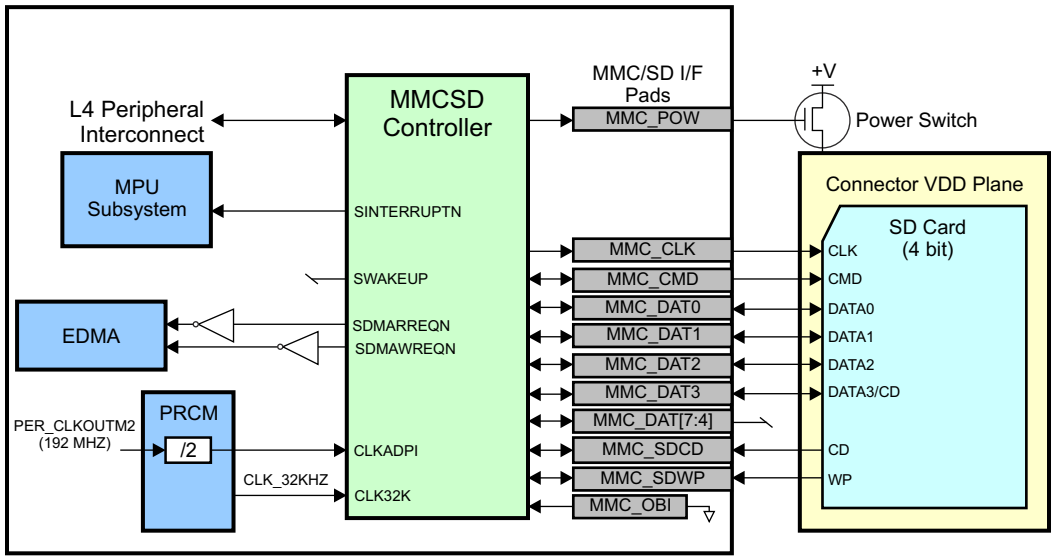


Figure 13-116. MMCSD (4-bit) Card Application

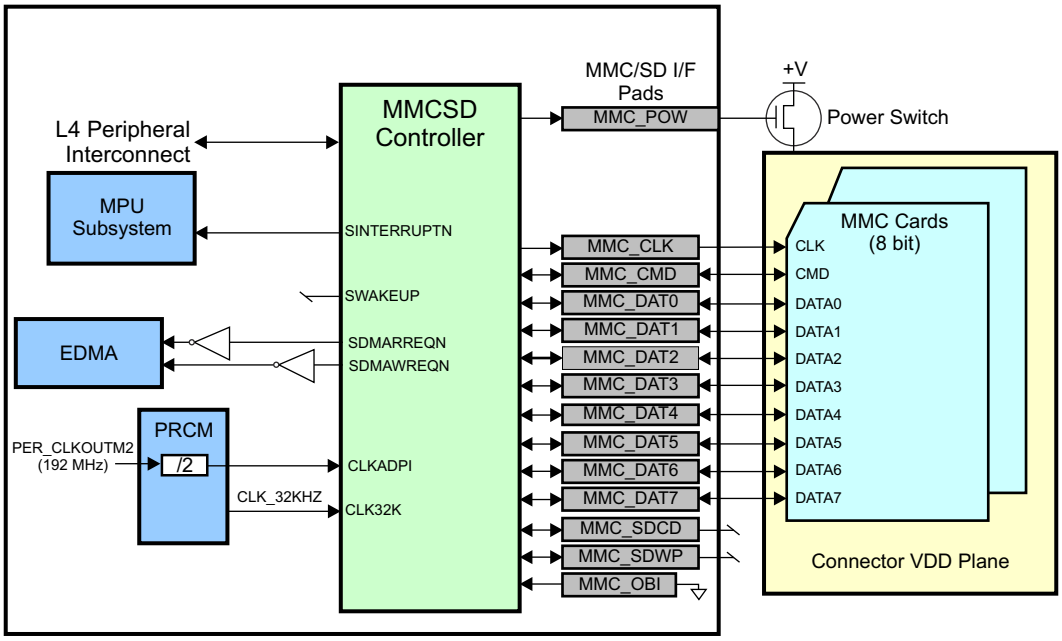


Figure 13-117. MMCSD Module MMC Application

13.3.1.2.1 MMCSD Integration

There is 1x MMCSD integrated in the device. The diagram below provides a visual representation of the device integration details.

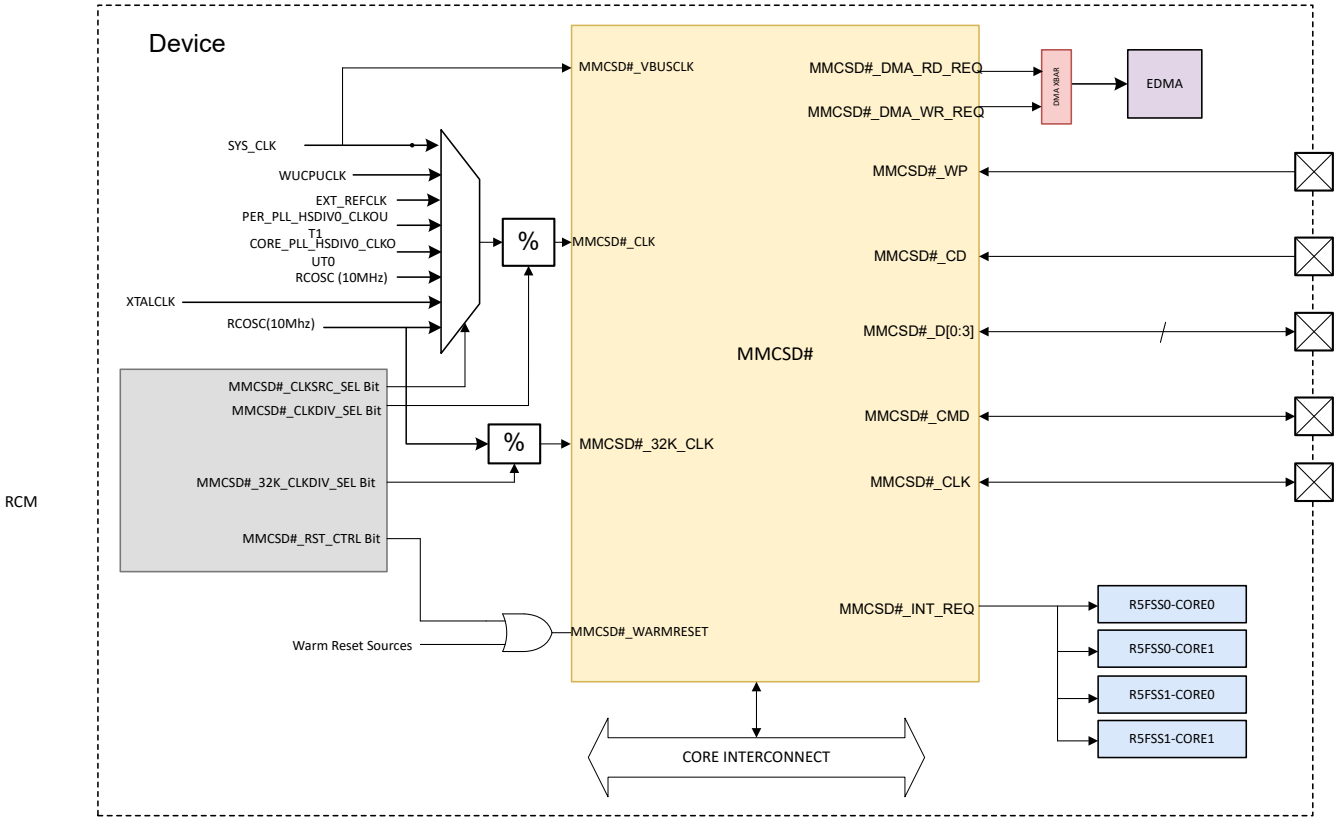


Figure 13-118. MMCSD Integration

The tables below summarize the device integration details of the MMC/SD module.

Table 13-164. *MMCSD* Device Integration

This table describes the module integration details.

MMCSD Instance	Device Allocation	SoC Interconnect
MMCSD0	✓	CORE VBUSM Interconnect

Table 13-165. *MMCSD* Clocks

This table describes the module clocking signals.

MMCSD Instance	MMCSD Clock Input	Source Clock Signal	Source	Default Freq	Description
MMCSD0	MMCSD0_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	MMC/SD Interface Clock
	MMCSD0_32K_CLK	MMCSD0_32K_CLK	XTALCLK	32 KHz	MMC/SD Debounce Clock
	MMCSD0_FCLK (MMCSD_CLK)	XTALCLK	External XTAL	25 MHz	MMC/SD Interface Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK: HSDIV0_CLKOUT0	400 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	

Table 13-166. *MMCSD* Resets

This table describes the module reset signals.

MMCSD Instance	MMCSD Reset Input	Source Reset Signal	Source	Description
MMCSD0	MMCSD0_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	MMCSD0 Asynchronous Reset

Table 13-167. *MMCSD* Interrupt Requests

This table describes the module interrupt requests.

MMCSD Instance	MMCSD Interrupt Signal	Destination Interrupt Input	Destination	Type	Description
MMCSD0	MMCSD0_INT_req_0	MMCSD0_INT_req_0	ALL R5FSS Cores	Level	MMC/SD Interrupt

Table 13-168. *MMCSD* DMA Requests

This table describes the module DMA requests.

MMCSD Instance	MMCSD DMA Event	Destination DMA Event Input	Destination	Type	Description
MMCSD0	MMCSD0_DMA_RD_REQ	MMCSD0_DMA_RD_REQ	EDMA Crossbar (DMA_XBAR)	Level	MMC/SD DMA Read Request
	MMCSD0_DMA_WR_REQ	MMCSD0_DMA_WR_REQ			MMC/SD DMA Write Request

Note

For more information on the interconnects, see the *System Interconnect* chapter.

For more information on power, reset, and clock management, see the corresponding sections within the *Device Configuration* chapter.

For more information on the device interrupt controllers, see the *Interrupt Controllers* chapter.

13.3.1.2.2 MMCSD Connectivity Attributes

The general connectivity attributes for the three MMCSD modules are shown in Table 13-169.

Table 13-169. MMCSD Connectivity Attributes

Attributes	Type
Power Domain	Peripheral Domain
Clock Domain	PD_PER_L4LS_GCLK (OCP) PD_PER_MMC_FCLK (Func) CLK_32KHZ (Debounce)
Reset Signals	PER_DOM_RST_N
Idle/Wakeup Signals	Smart Idle
Interrupt Requests	1 interrupt per instance to MPU Subsystem (MMCSDxINT)
DMA Requests	2 DMA requests per instance to EDMA (SDTXEVTx, SDRXEVTx) (Active low, need to be inverted in glue logic)
Physical Address	L4 Peripheral target port

13.3.1.2.3 MMCSD Clock and Reset Management

The MMCSD controller has separate bus interface and functional clocks. The debounce clock is created by dividing the 48-MHz (24 MHz @ OPP50) clock in the PRCM by two and then dividing the resulting 24-MHz (12 MHz @ OPP50) clock by a fixed 732.4219 (366.2109 @ OPP50) in the Control Module to get a 32-kHz clock. This clock is fed back into the PRCM for clock gating. (See the CTRL_CLK32KDIVRATIO register in *Control Module* for more details).

Table 13-170. MMCSD Clock Signals

Clock Signal	Max Freq	Reference / Source	Comments
CLK Interface clock	100 MHz	CORE_CLKOUTM4 / 2	pd_per_l4ls_gclk from PRCM
CLKADPI Functional clock	96 MHz	PER_CLKOUTM2 / 2	pd_per_mmc_fclk from PRCM
CLK32 Input de-bounce clock	32.768 KHz	CLK_24 / 732.4219	clk_32KHz from PRCM

Note

Maximum MMC_CLK signal frequency is 48 MHz.

13.3.1.2.4 MMCSD Pin List

The MMCSD interface pins are summarized in Table 13-171.

Table 13-171. MMCSD Pin List

Pin	Type	Description
MMC*x*_CLK	I/O(1)	MMC/SD serial clock output
MMC*x*_CMD	I/O	MMC/SD command signal
MMC*x*_DAT0	I/O	MMC/SD data signal
MMC*x*_DAT1	I/O	MMC/SD data signal, SDIO interrupt input
MMC*x*_DAT2	I/O	MMC/SD data signal, SDIO read wait output
MMC*x*_DAT[7:3]	I/O	MMC/SD data signals
MMC*x*_POW	O	MMC/SD power supply control (MMCSD0 only)
MMC*x*_SDCD	I	SD card detect (from connector)
MMC*x*_SDWP	I	SD write protect (from connector)
MMC*x*_OBI	I	MMC out of band interrupt

(1) These signals are also used as inputs to re-time or sync data. The associated CONF_<module>_pin_RXACTIVE bit for these signals must be set to 1 to enable the inputs back to the module. It is also recommended to place a 33-ohm resistor in series (close to the processor) on each of these signals to avoid signal reflections.

The direction of the data lines depends on the selected data transfer mode as summarized in Table 13-172.

Table 13-172. DAT Line Direction for Data Transfer Modes

	MMC/SD 1-bit mode	MMC/SD 4-bit mode	MMC/SD 8-bit mode	SDIO 1-bit mode	SDIO 4-bit mode
DAT[0]	I/O	I/O	I/O	I/O	I/O
DAT[1]	I(1)	I/O	I/O	I(2)	I/O or I(2)
DAT[2]	I(1)	I/O	I/O	I/O(3)	I/O or O(3)
DAT[3]	I(1)	I/O	I/O	I(1)	I/O
DAT[4]	I(1)	I(1)	I/O	I(1)	I(1)
DAT[5]	I(1)	I(1)	I/O	I(1)	I(1)
DAT[6]	I(1)	I(1)	I/O	I(1)	I(1)

Table 13-172. DAT Line Direction for Data Transfer Modes (continued)

	MMC/SD 1-bit mode	MMC/SD 4-bit mode	MMC/SD 8-bit mode	SDIO 1-bit mode	SDIO 4-bit mode
DAT[7]	I(1)	I(1)	I/O	I(1)	I(1)

(1) Hi-Z state to avoid bus conflict.
(2) To support incoming interrupt from the SDIO card.
(3) To support read wait to the SDIO card. By default it is Input, Output only in read wait period.

The direction of the MMCSD data buffers are controlled by ADPDATDIROQ signals. ADPDATDIROQ[i] = 1 sets the corresponding DAT signal(s) in read position (input) and ADPDATDIROQ[i] = 0 sets the corresponding DAT signal(s) in write position (output). Additionally, the ADPDATDIRLS signals are provided (with opposite polarity) to control the direction of external level shifters. The value of these control signals for the various data modes are summarized in Table 13-173.

Table 13-173. ADPDATDIROQ and ADPDATDIRLS Signal States

	MMC/SD 1-bit mode	MMC/SD 4-bit mode	MMC/SD 8-bit mode	SDIO 1-bit mode	SDIO 4-bit mode
DAT[0]	ADPDATDIRLS[0] = 0 / 1 ADPDATDIROQ[0] = 1 / 0	ADPDATDIRLS[0] = 0 / 1 ADPDATDIROQ[0] = 1 / 0	ADPDATDIRLS[0] = 0 / 1 ADPDATDIROQ[0] = 1 / 0	ADPDATDIRLS[0] = 0 / 1	ADPDATDIRLS[0] = 0 / 1 ADPDATDIROQ[0] = 1 / 0
DAT[2]	ADPDATDIRLS[2] = 0 ADPDATDIROQ[2] = 1	ADPDATDIRLS[2] = 0 / 1 ADPDATDIROQ[2] = 1 / 0	ADPDATDIRLS[2] = 0 / 1 ADPDATDIROQ[2] = 1 / 0	ADPDATDIRLS[2] = 0 / 1 ADPDATDIROQ[2] = 1 / 0	ADPDATDIRLS[2] = 0 / 1 ADPDATDIROQ[2] = 1 / 0
DAT[1]	ADPDATDIRLS[1] = 0 ADPDATDIROQ[1] = 1	ADPDATDIRLS[1] = 0 / 1 ADPDATDIROQ[1] = 1 / 0	ADPDATDIRLS[1] = 0 / 1 ADPDATDIROQ[1] = 1 / 0	ADPDATDIRLS[1] = 0 ADPDATDIROQ[1] = 1	ADPDATDIRLS[1] = 0 / 1 ADPDATDIROQ[1] = 1 / 0
DAT[3]					
DAT[4]	ADPDATDIRLS[3] = 0 ADPDATDIROQ[3] = 1	ADPDATDIRLS[3] = 0 ADPDATDIROQ[3] = 1	ADPDATDIRLS[3] = 0 / 1 ADPDATDIROQ[3] = 1 / 0	ADPDATDIRLS[3] = 0 ADPDATDIROQ[3] = 1	ADPDATDIRLS[3] = 0 ADPDATDIROQ[3] = 1
DAT[5]					
DAT[6]					
DAT[7]					

ADPDATIRLSx = 0 for input and 1 for output — these signals are not pinned out on this device.

ADPDATIROQx = 1 for output and 1 for input.

Grayed cells indicate that the data line is not used in the selected transfer mode.

13.3.1.3 Functional Description

One MMC/SD/SDIO host controller can support one MMC memory card, one SD card, or one SDIO card.

Other combinations (for example, two SD cards, one MMC card, and one SD card) are not supported through a single controller.

13.3.1.3.1 MMC/SD/SDIO Functional Modes

13.3.1.3.1.1 MMC/SD/SDIO Connected to an MMC, an SD Card, or an SDIO Card

Figure 13-119 shows the MMC/SD/SDIO1 and MMC/SD/SDIO2 host controllers connected to an MMC, an SD, or an SDIO card and its related external connections.

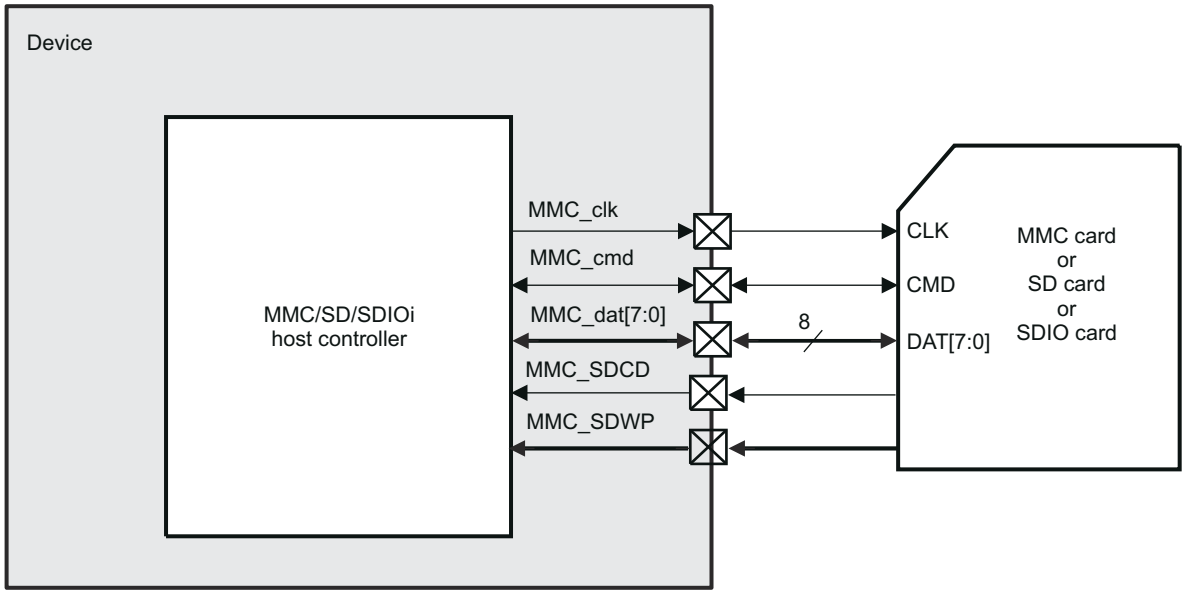


Figure 13-119. MMC/SD1/2 Connectivity to an MMC/SD Card

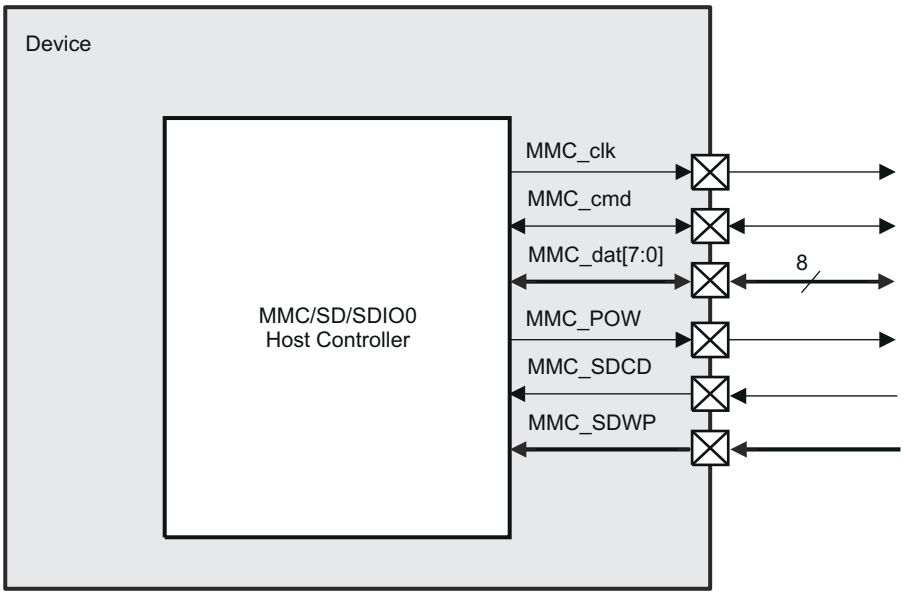


Figure 13-120. MMC/SD0 Connectivity to an MMC/SD Card

Figure 13-120 shows the MMC/SD/SDIO0 host controller connected to an MMC, SD, or SDIO card and its related external connections. Note that MMC/SD/SDIO0 uses the same signals as MMC/SD/SDIO1 and 2 but adds MMC_POW.

The following MMC/SD/SDIO controller pins are used

- **MMC_CMD** This pin is used for two-way communication between the connected card and the MMC/SD/SDIO controller. The MMC/SD/SDIO controller transmits commands to the card and the memory card drives responses to the commands on this pin.
- **MMC_DAT7-0** These pins are connected based on the type of card being used. Table 13-174 outlines which pins are required based on the mode. The number of DAT pins (the data bus width) is set by the Data Transfer Width (DTW) bit in the MMC control register (SD_HCTL). For more information see MMCSD Registers in the Register Addendum.
- **MMC_CLK** This pin provides the clock to the memory card from the MMC/SD controller.
- **MMC_POW** This output pin is used for the MMC/SD card on/off power supply control. The output being high denotes the power-on condition.
- **MMC_SDCD** This input pin serves as the MMC/SD/SDIO card detect. This signal is received from a mechanical switch on the slot.
- **MMC_SDWP** This input pin is used for the SD/SDIO card's write protect. This signal is received from a mechanical protect switch on the slot (system dependant). Applicable only for SD and SDIO cards that have a mechanical sliding tablet on the side of the card.

  Note: The MMC_CLK pin functions as an output but must be configured as an I/O to internally loopback the clock to time the inputs.

Table 13-174 provides a summary of these pins.

Table 13-174. MMC/SD/SDIO Controller Pins and Descriptions

Pin	Type	1-Bit Mode	4-Bit Mode	8-Bit Mode	Reset Value
MMC_CLK(1)	O	Clock Line	Clock Line	Clock Line	High impedance
MMC_CMD	I/O	Command Line	Command Line	Command Line	High impedance
MMC_DAT0	I/O	Data Line 0	Data Line 0	Data Line 0	0
MMC_DAT1	I/O	(not used)	Data Line 1	Data Line 1	0
MMC_DAT2	I/O	(not used)	Data Line 2	Data Line 2	0
MMC_DAT3	I/O	(not used)	Data Line 3	Data Line 3	0
MMC_DAT4	I/O	(not used)	(not used)	Data Line 4	0
MMC_DAT5	I/O	(not used)	(not used)	Data Line 5	0
MMC_DAT6	I/O	(not used)	(not used)	Data Line 6	0
MMC_DAT7	I/O	(not used)	(not used)	Data Line 7	0

(1) The MMC_CLK pin functions as an output but must be configured as an I/O to internally loopback the clock to time the inputs.

13.3.1.3.1.2 Protocol and Data Format

The bus protocol between the MMC/SD/SDIO host controller and the card is message-based. Each message is represented by one of the following parts:

Command: A command starts an operation. The command is transferred serially from the MMC/SD/SDIO host controller to the card on the mmc_cmd line.

Response: A response is an answer to a command. The response is sent from the card to the MMC/SD/SDIO host controller. It is transferred serially on the mmc_cmd line.

Data: Data are transferred from the MMC/SD/SDIO host controller to the card or from a card to the MMC/SD/SDIO host controller using the DATA lines.

Busy: The mmc_dat0 signal is maintained low by the card as far as it is programming the data received.

CRC status: CRC result is sent by the card through the mmc_dat0 line when executing a write transfer. When a transmission error occurs on any of the active data lines, the card sends a negative CRC status on mmc_dat0. When a successful transmission occurs over all active data lines, the card sends a positive CRC status on mmc_dat0 and starts the data programming procedure.

13.3.1.3.1.2.1 Protocol

There are two types of data transfer:

- Sequential operation
- Block-oriented operation

There are specific commands for each type of operation (sequential or block-oriented). See the *Multimedia Card System Specification*, the *SD Memory Card Specifications*, and the *SDIO Card Specification, Part E1* for details about commands and programming sequences supported by the MMC, SD, and SDIO cards.

CAUTION

Stream commands are supported only by MMC cards.

Figure 13-121 and Figure 13-122 show how sequential operations are defined. Sequential operation is only for 1-bit transfer and initiates a continuous data stream. The transfer terminates when a stop command follows on the mmc_cmd line.

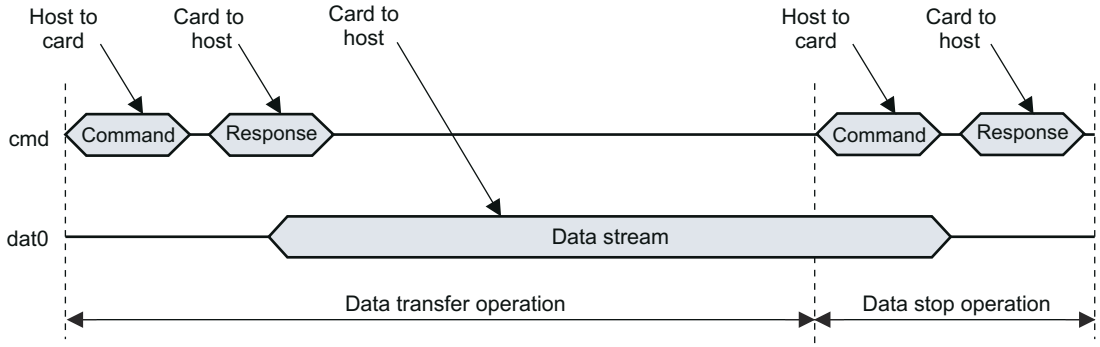


Figure 13-121. Sequential Read Operation (MMC Cards Only)

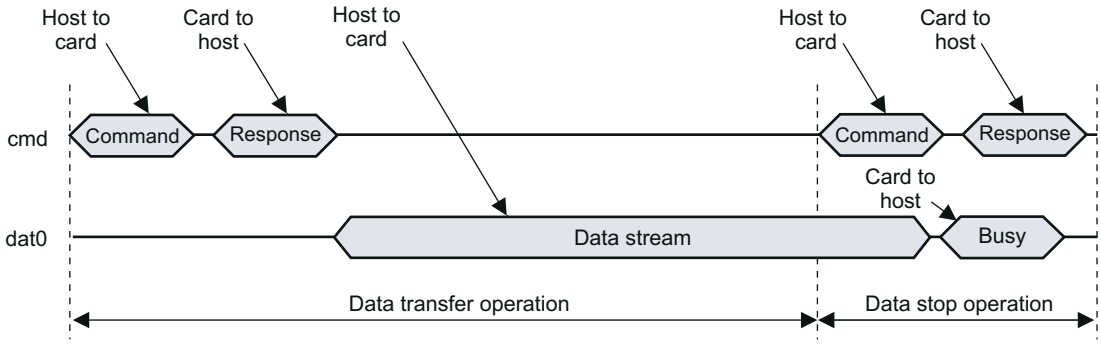


Figure 13-122. Sequential Write Operation (MMC Cards Only)

Figure 13-123 and Figure 13-124 show how multiple block-oriented operations are defined. A multiple block-oriented operation sends a data block plus CRC bits. The transfer terminates when a stop command follows on the mmc_cmd line. These operations are available for all kinds of cards.

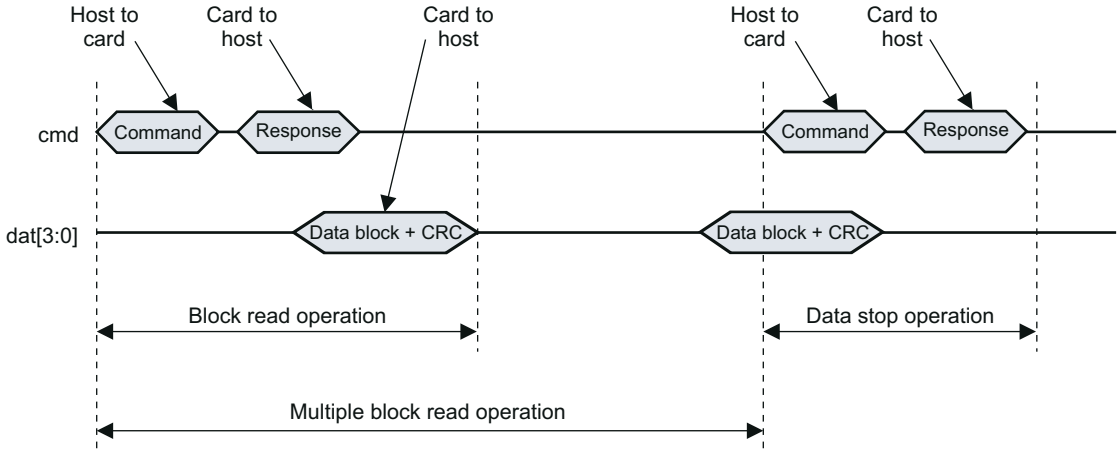


Figure 13-123. Multiple Block Read Operation (MMC Cards Only)

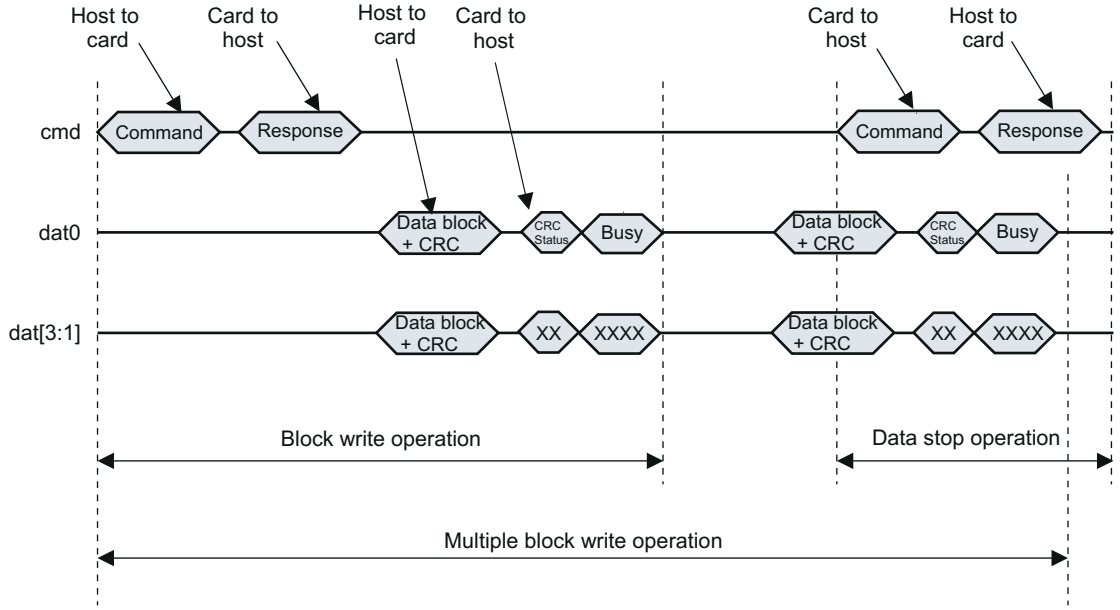


Figure 13-124. Multiple Block Write Operation (MMC Cards Only)

Note

1. The card busy signal is not always generated by the card; the previous examples show a particular case.
2. It is the software's responsibility to do a software reset after a data timeout to ensure that mmc_clk is stopped. The software reset is done by setting bit 26 in the SD_SYSCTL register to 1.
3. For multiblock transfer, and especially for MMC cards, you can abort a transfer without using a stop command. Use a CMD23 before a data transfer to define the number of blocks that will be transferred, then the transfer stops automatically after the last block (provided the MMC card supports this feature).

13.3.1.3.1.2.2 Data Format

Coding Scheme for Command Token

Command packets always start with 0 and end with 1. The second bit is a transmitter bit1 for a host command. The content is the command index (coded by 6 bits) and an argument (for example, an address), coded by 32 bits. The content is protected by 7-bit CRC checksum (see Figure 13-125).

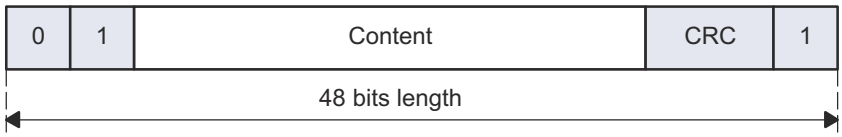


Figure 13-125. Command Token Format

Coding Scheme for Response Token

Response packets always start with 0 and end with a 1. The second bit is a transmitter bit0 for a card response. The content is different for each type of response (R1, R2, R3, R4, R5, and R6) and the content is protected by 7-bit CRC checksum. Depending on the type of commands sent to the card, the SD_CMD register must be configured differently to avoid false CRC or index errors to be flagged on command response (see Table 13-175). For more details about response types, see the *Multimedia Card System Specification*, the *SD Memory Card Specification*, or the *SDIO Card Specification*.

Table 13-175. Response Type Summary

Response Type SD_CMD[17:16] RSP_TYPE(1)	Index Check Enable SD_CMD[20] CICE	CRC Check Enable SD_CMD[19] CCCE	Name of Response Type
00	0	0	No Response
01	0	1	R2
10	0	0	R3 (R4 for SD cards)
10	1	1	R1, R6, R5 (R7 for SD cards)
11	1	1	R1b, R5b

(1) The MMC/SD/SDIO host controller assumes that both clocks may be switched off, whatever the value set in the SD_SYSCONFIG[9:8] CLOCKACTIVITY bit.

Figure 13-126 and Figure 13-127 depict the 48-bit and 136-bit response packets.

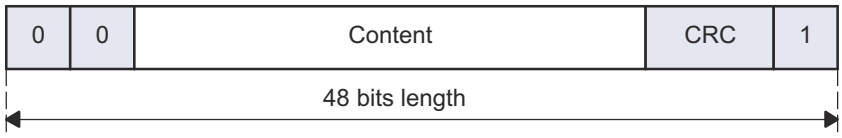


Figure 13-126. 48-Bit Response Packet (R1, R3, R4, R5, R6)

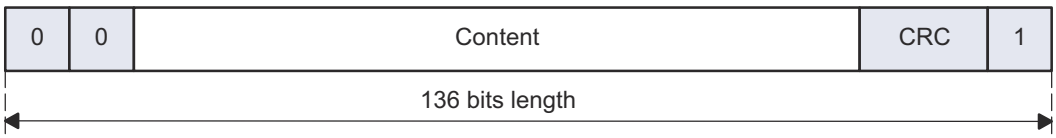


Figure 13-127. 136-Bit Response Packet (R2)

Coding Scheme for Data Token

Data tokens always start with 0 and end with 1 (see Figure 13-128, Figure 13-129, Figure 13-130, and Figure 13-131).

Figure 13-128. Data Packet for Sequential Transfer (1-Bit)

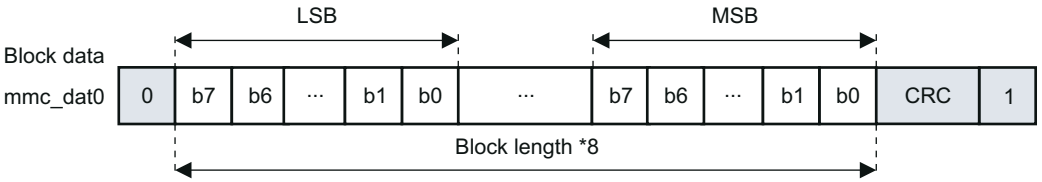


Figure 13-129. Data Packet for Block Transfer (1-Bit)

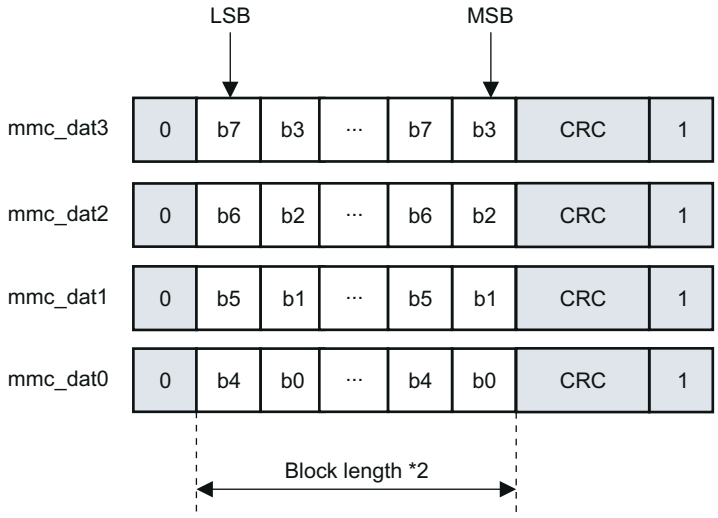


Figure 13-130. Data Packet for Block Transfer (4-Bit)

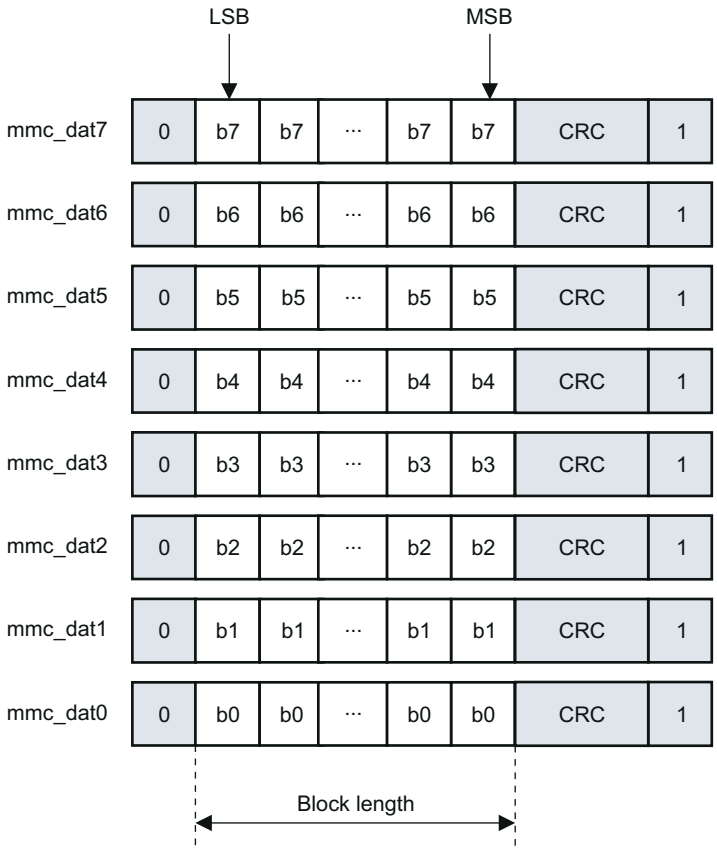


Figure 13-131. Data Packet for Block Transfer (8-Bit)

13.3.1.3.2 Resets

13.3.1.3.2.1 Hardware Reset

The module is reinitialized by the hardware.

The SD_SYSSTS[0] RESETDONE bit can be monitored by the software to check if the module is ready-to-use after a hardware reset.

This hardware reset signal has a global reset action on the module. All configuration registers and all state machines are reset in all clock domains.

This hardware reset signal has a global reset action on the module. All configuration registers and all state-machines are reset in all clock domains.

13.3.1.3.2.2 Software Reset

The module is reinitialized by software through the SD_SYSCONFIG[1] SOFTRESET bit. This bit has the same action on the module logic as the hardware signal except for:

- Debounce logic
- SD_PSTATE, SD_CAPA, and SD_CUR_CAPA registers (see corresponding register descriptions)

The SOFTRESET bit is active high. The bit is automatically reinitialized to 0 by the hardware. The SD_SYSCTL[24] SRA bit has the same action as the SOFTRESET bit on the design.

The SD_SYSSTS[0] RESETDONE bit can be monitored by the software to check if the module is ready-to-use after a software reset.

Moreover, two partial software reset bits are provided:

- SD_SYSCTL[26] SRD bit
- SD_SYSCTL[25] SRC bit

These two reset bits are useful to reinitialize data or command processes respectively in case of line conflict. When set to 1, a reset process is automatically released when the reset completes:

- The SD_SYSCTL[26] SRD bit resets all finite state-machines and status management that handle data transfers on both the interface and functional side.
- The SD_SYSCTL[25] SRC bit resets all finite state-machines and status management that handle command transfers on both the interface and functional side.

Note

If **any** of the clock inputs are not present for the MMC/SD/SDIO peripheral, the software reset will not complete.

13.3.1.3.3 Power Management

The MMC/SD/SDIO host controller can enter into different modes and save power:

- Normal mode
- Idle mode

The two modes are mutually exclusive (the module can be in normal mode or in idle mode). The MMC/SD/SDIO host controller is compliant with the PRCM module handshake protocol. When the MMC/SD/SDIO power domain is off, the only way to wake up the power domain and different MMC/SD/SDIO clocks is to monitor the mmc_dat1 input pin state via a different GPIO line for each MMC/SD/SDIO interface.

13.3.1.3.3.1 Normal Mode

The autogating of interface and functional clocks occurs when the following conditions are met:

- The SD_SYSCONFIG[0] AUTOIDLE bit is set to 1.
- There is no transaction on the MMC interface.

The autogating of interface and functional clocks stops when the following conditions are met:

- A register access occurs through the L3 (or L4) interconnect.
- A wake-up event occurs (an interrupt from a SDIO card).
- A transaction on the MMC/SD/SDIO interface starts.

Then the MMC/SD/SDIO host controller enters in low-power state even if SD_SYSCONFIG[0] AUTOIDLE is cleared to 0. The functional clock is internally switched off and only interconnect read and write accesses are allowed.

13.3.1.3.3.2 Idle Mode

The clocks provided to MMC/SD/SDIO are switched off upon a PRCM module request. They are switched back upon module request. The MMC/SD/SDIO host controller complies with the PRCM module handshaking protocol:

- Idle request from the system power manager
- Idle acknowledgment from the MMC/SD/SDIO host controller

The idle acknowledgment varies according to the SD_SYSCONFIG[4:3] SIDLEMODE bit field:

- 0: Force-idle mode. The MMC/SD/SDIO host controller acknowledges the system power manager request unconditionally.
- 1h: No-idle mode. The MMC/SD/SDIO host controller ignores the system power manager request and behaves normally as if the request was not asserted.
- 2h: Smart-idle mode. The MMC/SD/SDIO host controller acknowledges the system power manager request according to its internal state.
- 3h: Reserved.

During the smart-idle mode period, the MMC/SD/SDIO host controller acknowledges that the OCP and Functional clocks may be switched off based on the values set in the SD_SYSCONFIG[9:8] CLOCKACTIVITY field.

13.3.1.3.3.3 Transition from Normal Mode to Smart-Idle Mode

Smart-idle mode is enabled when the SD_SYSCONFIG[4:3] SIDLEMODE bit field is set to 2h or 3h. The MMC/SD/SDIO host controller goes into idle mode when the PRCM issues an idle request, according to its internal activity. The MMC/SD/SDIO host controller acknowledges the idle request from the PRCM after ensuring the following:

- The current multi/single-block transfer is completed.
- Any interrupt or DMA request is asserted.
- There is no card interrupt on the SD_dat1 signal.

As long as the MMC/SD/SDIO controller does not acknowledge the idle request, if an event occurs, the MMC/SD/SDIO host controller can still generate an interrupt or a DMA request. In this case, the module ignores the idle request from the PRCM.

As soon as the MMC/SD/SDIO controller acknowledges the idle request from the PRCM:

- If Smart-Idle mode the module does not assert any new interrupt or DMA request

13.3.1.3.3.4 Transition from Smart-Idle Mode to Normal Mode

The MMC/SD/SDIO host controller detects the end of the idle period when the PRCM deasserts the idle request. For the wake-up event, there is a corresponding interrupt status in the SD_STAT register. The MMC/SD/SDIO host controller operates the conversion between wake-up and interrupt (or DMA request) upon exit from smart-idle mode if the associated enable bit is set in the SD_ISE register.

Interrupts and wake-up events have independent enable/disable controls, accessible through the SD_HCTL and SD_ISE registers. The overall consistency must be ensured by software.

The interrupt status register SD_STAT is updated with the event that caused the wake-up in the CIRQ bit when the SD_IE[8] CIRQ_ENABLE associated bit is enabled. Then, the wake-up event at the origin of the transition from smart-idle mode to normal mode is converted into its corresponding interrupt or DMA request. (The SD_STAT register is updated and the status of the interrupt signal changes.)

When the idle request from the PRCM is deasserted, the module switches back to normal mode. The module is fully operational.

13.3.1.3.3.5 Force-Idle Mode

Force-idle mode is enabled when the SD_SYSCONFIG[4:3] SIDLEMODE bit field is cleared to 0. Force-idle mode is an idle mode where the MMC/SD/SDIO host controller responds unconditionally to the idle request from the PRCM. Moreover, in this mode, the MMC/SD/SDIO host controller unconditionally deasserts interrupts and DMA request lines are asserted.

The transition from normal mode to force-idle mode does not affect the bits of the SD_STAT register. In force-idle mode, the interrupt and DMA request lines are deasserted. Interface Clock (OCP) and functional clock (CLKADPI) can be switched off.

CAUTION

In Force-idle mode, an idle request from the PRCM during a command or a data transfer can lead to an unexpected and unpredictable result.
When the module is idle, any access to the module generates an error as long as the OCP clock is alive.

The module exits the force-idle mode when the PRCM deasserts the idle request. Then the module switches back to normal mode. The module is fully operational. Interrupt and DMA request lines are optionally asserted one clock cycle later.

13.3.1.3.3.6 Local Power Management

Table 13-176 describes power-management features available for the MMC/SD/SDIO modules.

Table 13-176. Local Power Management Features

Feature	Registers	Description
Clock Auto Gating	SD_SYSCONFIG AUTOIDLE bit	This bit allows a local power optimization inside module, by gating the OCP clock upon the interface activity or gating the CLKADPI clock upon the internal activity.
Target Idle Modes	SD_SYSCONFIG SIDLEMODE bit	Force-idle, No-idle, and Smart-idle modes
Clock Activity	SD_SYSCONFIG CLOCKACTIVITY bit	Please see Table 13-177 for configuration details.
Global Wake-Up Enable	SD_SYSCONFIG ENAWAKEUP bit	This bit enables the wake-up feature at module level.
Wake-Up Sources Enable	SD_HCTL register	This register holds one active high enable bit per event source able to generate wake-up signal.

Table 13-177. Clock Activity Settings

	Clock State When Module is in IDLE State			
CLOCKACTIVITY Values	OCP Clock	CLKADPI	Features Available when Module is in IDLE State	Wake-Up Events
00	OFF	OFF	None	Card Interrupt
10	OFF	ON	None	
01	ON	OFF	None	
11	ON	ON	All	

CAUTION

The PRCM module has no hardware means of reading CLOCKACTIVITY settings. Thus, software must ensure consistent programming between the CLOCKACTIVITY and MMC clock PRCM control bits.

13.3.1.3.4 Interrupt Requests

Several internal module events can generate an interrupt. Each interrupt has a status bit, an interrupt enable bit, and a signal status enable:

- The status of each type of interrupt is automatically updated in the SD_STAT register; it indicates which service is required.
- The interrupt status enable bits of the SD_IE register enable/disable the automatic update of the SD_STAT register on an event-by-event basis.
- The interrupt signal enable bits of the SD_ISE register enable/disable the transmission of an interrupt request on the interrupt line MMC_IRQ (from the MMC/SD/SDIO host controller to the MPU subsystem interrupt controller) on an event-by-event basis.

If an interrupt status is disabled in the SD_IE register, then the corresponding interrupt request is not transmitted, and the value of the corresponding interrupt signal enable in the SD_ISE register is ignored.

When an interrupt event occurs, the corresponding status bit is automatically set to 1 (the MMC/SD/SDIO host controller updates the status bit) in the SD_STAT register. If later a mask is applied on the interrupt in the SD_ISE register, the interrupt request is deactivated.

When the interrupt source has not been serviced, if the interrupt status is cleared in the SD_STAT register and the corresponding mask is removed from the SD_ISE register, the interrupt status is not asserted again in the SD_STAT register and the MMC/SD/SDIOi host controller does not transmit an interrupt request.

CAUTION

If the buffer write ready interrupt (BWR) or the buffer read ready only interrupt (BRR) are not serviced and are cleared in the SD_STAT register, and the corresponding mask is removed, then the MMC/SD/SDIOi host controller will wait for the service of the interrupt without updating the status SD_STAT or transmitting an interrupt request.

Table 13-178 lists the event flags, and their mask, that can cause module interrupts.

Table 13-178. Events

Event Flag	Event Mask	Map To	Description
SD_STAT[29] BADA	SD_IE[29] BADA_ENABLE	MMC_IRQ	Bad Access to Data space. This bit is set automatically to indicate a bad access to buffer when not allowed. This bit is set during a read access to the data register (SD_DATA) while buffer reads are not allowed (SD_PSTATE[11] BRE=0). This bit is set during a write access to the data register (SD_DATA) while buffer writes are not allowed (SD_STATE[10] BWE=0)
SD_STAT[28] CERR	SD_IE[28] CERR_ENABLE	MMC_IRQ	Card Error. This bit is set automatically when there is at least one error in a response of type R1, R1b, R6, R5 or R5b. Only bits referenced as type E(error) in status field in the response can set a card status error. An error bit in the response is flagged only if corresponding bit in card status response errors SD_CSRE is set. There is not card detection for auto CMD12 command.
SD_STAT[25] ADMAE	SD_IE[25] ADMAE_ENABLE	MMC_IRQ	ADMA error. This bit is set when the host controller detects errors during ADMA based data transfer. The stat of the ADMA at an error occurrence is saved in the ADMA Error Status Register. In addition, the host controller generates this interrupt when it detects invalid descriptor data (Valid=0) at the ST_FDS state.
SD_STAT[24] ACE	SD_IE[24] ACE_ENABLE	MMC_IRQ	Auto CMD12 error. This bit is set automatically when one of the bits in Auto CMD12 Error status register has changed from 0 to 1
SD_STAT[22] DEB	SD_IE[22] DEB_ENABLE	MMC_IRQ	Data End Bit error. This bit is set automatically when detecting a 0 at the end bit position of read data on DAT line or at the end position of the CRC status in write mode.
SD_STAT[21] DCRC	SD_IE[21] DCRC_ENABLE	MMC_IRQ	Data CRC error. This bit is set automatically when there is a CRC16 error in the data phase response following a block read command or if there is a 3-bit CRC status different of a position "010" token during a block write command.

Table 13-178. Events (continued)

Event Flag	Event Mask	Map To	Description
SD_STAT[20] DTO	SD_IE[20] DTO_ENABLE	MMC_IRQ	Data Timeout error. This bit is set automatically according to the following conditions: A) busy timeout for R1b, R5b response. B) busy timeout after write CRC status. C) write CRC status timeout, or D) read data timeout.
SD_STAT[19] CIE	SD_IE[19] CIE_ENABLE	MMC_IRQ	Command Index Error. This bit is set automatically when response index differs from corresponding command index previously emitted. The check is enabled through SD_CMD[20] CICE bit.
SD_STAT[18] CEB	SD_IE[18] CEB_ENABLE	MMC_IRQ	Command End Bit error. This bit is set automatically when detecting a 0 at the end bit position of a command response.
SD_STAT[17] CCRC	SD_IE[17] CCRC_ENABLE	MMC_IRQ	Command CRC error. This bit is set automatically when there is a CRC7 error in the command response. CRC check is enabled through the SD_CMD[19] CCCE bit.
SD_STAT[16] CTO	SD_IE[16] CTO_ENABLE	MMC_IRQ	Command Timeout error. This bit is set automatically when no response is received within 64 clock cycles from the end bit of the command. For commands the reply within 5 clock cycles, the timeout is still detected at 64 clock cycles.
SD_STAT[15] ERRI	SD_IE[15] ERRI_ENABLE	MMC_IRQ	Error Interrupt. If any of the bits in the Error Interrupt Status register (SD_STAT[24:15]) are set, the this bit is set to 1.
SD_STAT[10] BSR	SD_IE[10] BSR_ENABLE	MMC_IRQ	Boot Status Received interrupt. This bit is set automatically when SD_CON[18] BOOT_CF0 is set to 1 or 2h and boot status is received on the dat0 line. This interrupt is only used for MMC cards.
SD_STAT[8] CIRQ	SD_IE[8] CIRQ_ENABLE	MMC_IRQ	Card Interrupt. This bit is only used for SD, SDIO, and CE-ATA cards. In 1-bit mode, interrupt source is asynchronous (can be a source of asynchronous wake-up). In 4-bit mode, interrupt source is sampled during the interrupt cycle. In CE-ATA mode, interrupt source is detected when the card drive CMD line to zero during one cycle after data transmission end.
SD_STAT[5] BRR	SD_IE[5] BRR_ENABLE	MMC_IRQ	Buffer Read ready. This bit is set automatically during a read operation to the card when one block specified by SD_BLK[10:0] BLEN is completely written in the buffer. It indicates that the memory card has filled out the buffer and the local host needs to empty the buffer by reading it.
SD_STAT[4] BWR	SD_IE[4] BWR_ENABLE	MMC_IRQ	Buffer Write ready. This bit is automatically set during a write operation to the card when the host can write a complete block as specified by SD_BLK[10:0] BLEN. It indicates that the memory card has emptied one block from the bugger and the local host is able to write one block of data into the buffer.
SD_STAT[3] DMA	SD_IE[3] DMA_ENABLE	MMC_IRQ	DMA interrupt. This status is set when an interrupt is required in the ADMA instruction and after the data transfer is complete.
SD_STAT[2] BGE	SD_IE[2] BGE_ENABLE	MMC_IRQ	Block Gap event. When a stop at block gap is requested (SD_HCTL[16] SBGR), this bit is automatically set when transaction is stopped at the block gap during a read or write operation.
SD_STAT[1] TC	SD_IE[1] TC_ENABLE	MMC_IRQ	Transfer completed. This bit is always set when a read/write transfer is completed or between two blocks when the transfer is stopped due to a stop at block gap requested (SD_HCTL[16 SBGR). In read mode this bit is automatically set on completion of a read transfer (SD_PSTATE[9] RTA). In write mode, this bit is automatically set on completion of the DAT line use (SD_PSTATE[2] DLA).
SD_STAT[0] CC	SD_IE[0] CC_ENABLE	MMC_IRQ	Command complete. This bit is set when a 1-to-0 transition occurs in the register command inhibit (SD_PSTATE[0] CMDI). If the command is a type for which no response is expected, then the command complete interrupt is generated at the end of the command. A command timeout error (SD_STAT[16] CTO) has higher priority than command complete (SD_STAT[0] CC). If a response is expected but none is received, the a Command Timeout error is detected and signaled instead of the Command Complete interrupt.

13.3.1.3.4.1 Interrupt-Driven Operation

An interrupt enable bit must be set in the SD_IE register to enable the module internal source of interrupt.

When an interrupt event occurs, the single interrupt line is asserted and the LH must:

- Read the SD_STAT register to identify which event occurred.
- Write 1 into the corresponding bit of the SD_STAT register to clear the interrupt status and release the interrupt line (if a read is done after this write, this would return 0).

Note

In the SD_STAT register, Card Interrupt (CIRQ) and Error Interrupt (ERRI) bits cannot be cleared.

The SD_STAT[8] CIRQ status bit must be masked by disabling the SD_IE[8] CIRQ_ENABLE bit (cleared to 0), then the interrupt routine must clear SDIO interrupt source in SDIO card common control register (CCCR).

The SD_STAT[15] ERRI bit is automatically cleared when all status bits in SD_STAT[31:16] are cleared.

13.3.1.3.4.2 Polling

When the interrupt capability of an event is disabled in the SD_ISE register, the interrupt line is not asserted:

- Software can poll the status bit in the SD_STAT register to detect when the corresponding event occurs.
- Writing 1 into the corresponding bit of the SD_STAT register clears the interrupt status and does not affect the interrupt line state.

Note

Please see the note in Section 13.3.1.3.4.1 concerning CIRQ and ERRI bits clearing.

13.3.1.3.5 DMA Modes

The device supports DMA responder mode only. In this case, the controller is a responder on DMA transaction managed by two separated requests (SDMAWREQN and SDMARREQN)

13.3.1.3.5.1 DMA Responder Mode Operations

The MMC/SD/SDIO controller can be interfaced with a DMA controller. At system level, the advantage is to discharge the local host (LH) of the data transfers. The module does not support wide DMA access (above 1024 bytes) for SD cards as specified in the *SD Card Specification* and *SD Host Controller Standard Specification*.

The DMA request is issued if the following conditions are met:

- The SD_CMD[0] DE bit is set to 1 to trigger the initial DMA request (the write must be done when running the data transfer command).
- A command was emitted on the SD_cmd line.
- There is enough space in the buffer of the MMC/SD/SDIO controller to write an entire block (BLEN writes).

13.3.1.3.5.1.1 DMA Receive Mode

In a DMA block read operation (single or multiple), the request signal SDMARREQN is asserted to its active level when a complete block is written in the buffer. The block size transfer is specified in the SD_BLK[10:0] BLEN field.

The SDMARREQN signal is deasserted to its inactive level when the sDMA has read one single word from the buffer. Only one request is sent per block; the DMA controller can make a 1-shot read access or several DMA bursts, in which case the DMA controller must manage the number of burst accesses, according to block size BLEN field.

New DMA requests are internally masked if the sDMA has not read exactly BLEN bytes and a new complete block is not ready. As DMA accesses are in 32-bit, then the number of sDMA read is Integer(BLEN/4)+1.

The receive buffer never overflows. In multiple block transfers for block size above 512 bytes, when the buffer gets full, the MMC_CLK clock signal (provided to the card) is momentarily stopped until the sDMA or the MPU performs a read access, which reads a complete block in the buffer.

Figure 13-132 provides a summary:

- DMA transfer size = BLEN buffer size in one shot or by burst
- One DMA request per block

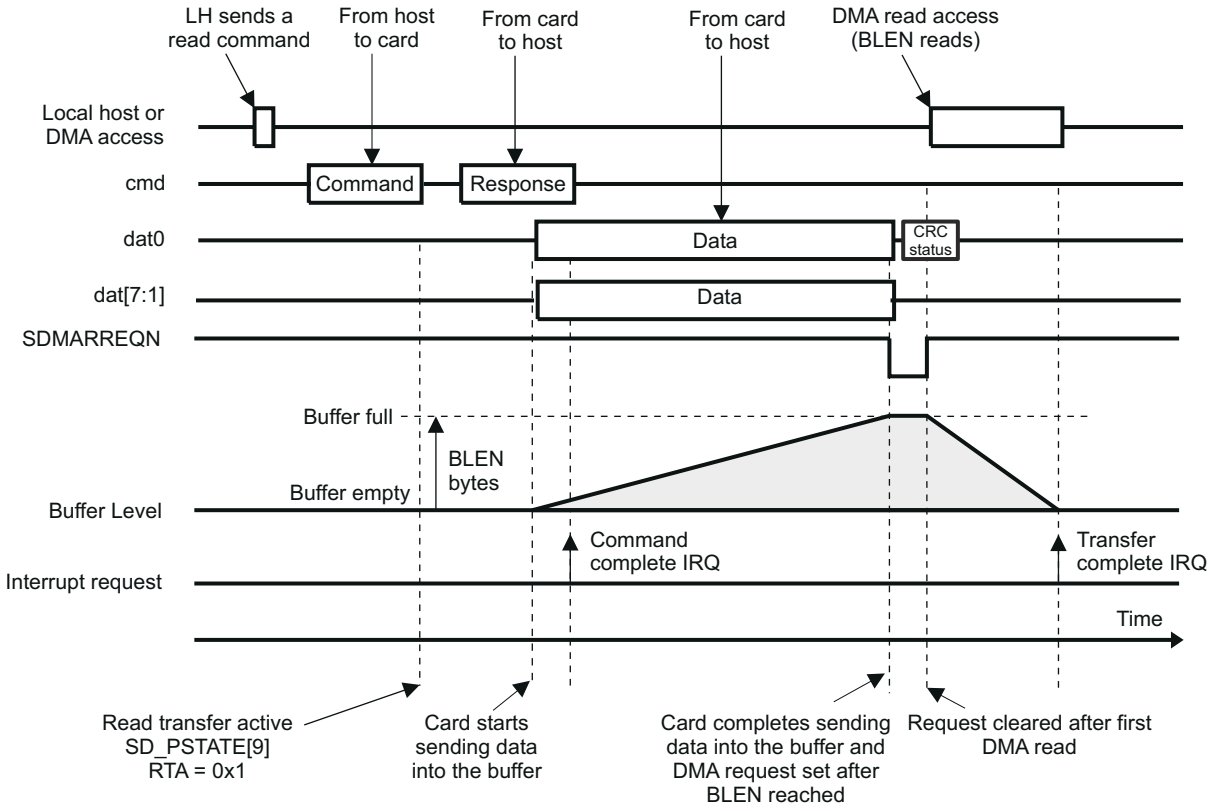


Figure 13-132. DMA Receive Mode

13.3.1.3.5.1.2 DMA Transmit Mode

In a DMA block write operation (single or multiple), the request signal SDMAWREQN is asserted to its active level when a complete block is to be written to the buffer. The block size transfer is specified in the SD_BLK[10:0] BLEN field.

The SDMAWREQN signal is deasserted to its inactive level when the sDMA has written one single word to the buffer.

Only one request is sent per block; the DMA controller can make a 1-shot write access or multiple write DMA bursts, in which case the DMA controller must manage the number of burst accesses, according to block size BLEN field.

New DMA requests are internally masked if the sDMA has not written exactly BLEN bytes (as DMA accesses are in 32-bit, then the number of sDMA read is Integer(BLEN/4)+1) and if there is not enough memory space to write a complete block in the buffer.

Figure 13-133 provides a summary:

- DMA transfer size = BLEN buffer size in one shot or by burst
- One DMA request per block

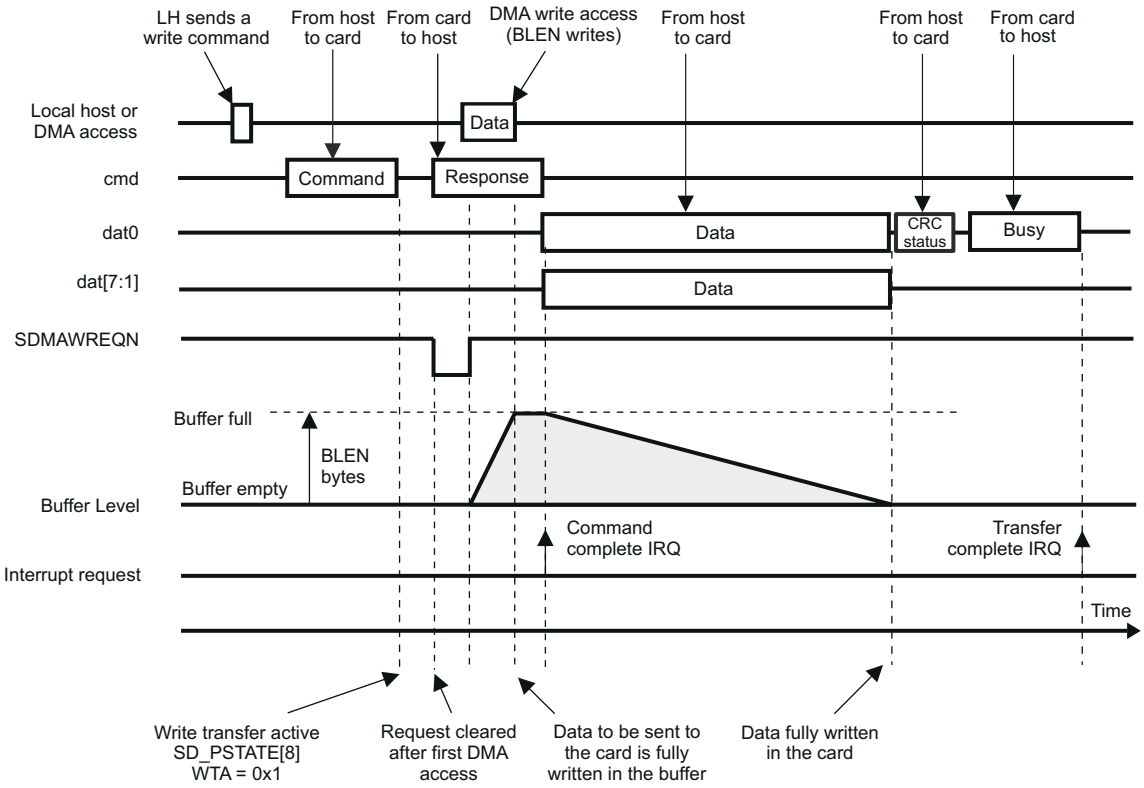


Figure 13-133. DMA Transmit Mode

13.3.1.3.6 Mode Selection

The MMC/SD/SDIO host controller can be use in two modes, MMC and SD/SDIO modes. It has been designed to be the most transparent with the type of card. The type of the card connected is differentiated by the software initialization procedure.

Software identifies the type of card connected during software initialization. For each given card type, there are corresponding commands. Some commands are not supported by all cards. See the *Multimedia Card System Specification*, the *SD Memory Card Specifications*, and the *SDIO Card Specification, Part E1* for more details.

The purpose of the module is to transfer commands and data, to whatever card is connected, respecting the protocol of the connected card. Writes and reads to the card must respect the appropriate protocol of that card.

13.3.1.3.7 Buffer Management

13.3.1.3.7.1 Data Buffer

The MMC/SD/SDIO host controller uses a data buffer. This buffer transfers data from one data bus (Interconnect) to another data bus (SD, SDIO, or MMC card bus) and vice versa.

The buffer is the heart of the interface and ensures the transfer between the two interfaces (L4 and the card). To enhance performance, the data buffer is completed by a prefetch register and a post-write buffer that are not accessible by the host controller.

The read access time of the prefetch register is faster than the one of the data buffer. The prefetch register allows data to be read from the data buffer at an increased speed by preloading data into the prefetch register.

The entry point of the data buffer, the prefetch buffer, and the post-write buffer is the 32-bit register SD_DATA. A write access to the SD_DATA register followed by a read access from the SD_DATA register corresponds to a write access to the post-write buffer followed by a read access to the prefetch buffer. As a consequence, it is normal that the data of the write access to the SD_DATA register and the data of the read access to the SD_DATA register are different.

The number of 32-bit accesses to the SD_DATA register that are needed to read (or write) a data block with a size of SD_BLK[10:0] BLEN, and equals the rounded up result of BLEN divided by 4. The maximum block size supported by the host controller is hard-coded in the register SD_CAPA[17:16] MBL field and cannot be changed.

A read access to the SD_DATA register is allowed only when the buffer read enable status is set to 1 (SD_PSTATE[11] BRE); otherwise, a bad access (SD_STAT[29] BADA) is signaled.

A write access to the SD_DATA register is allowed only when the buffer write enable status is set to 1 (SD_PSTATE[10] BWE); otherwise, a bad access (SD_STAT[29] BADA) is signaled and the data is not written.

The data buffer has two modes of operation to store and read of the first and second portions of the data buffer:

- When the size of the data block to transfer is less than or equal to MEM_SIZE/2 (in double buffering), two data transfers can occur from one data bus to the other data bus and vice versa at the same time. The MMC/SD/SDIO controller uses the two portions of the data buffer in a ping-pong manner so that storing and reading of the first and second portions of the data buffer are automatically interchanged from time to time so that data may be read from one portion (for instance, through a DMA read access on the interconnect bus) while data (for instance, from the card) is being stored into the other portion and vice versa. When BLEN is less than or equal to 200h (that is, less or equal to 512Bytes), each of the two portions of the buffer that can be used have a size of BLEN (that is, 32-bits x BLEN div by 4). Not more than this total size of 2 times 32-bits × BLEN div by 4 can be used.
- When the size of the data block to transfer is larger than MEM_SIZE/2, only one data transfer can occur from one data bus to the other data bus at a time. The MMC/SD/SDIO host controller uses the entire data buffer as a single portion. In this mode, a bad access (SD_STAT[29] BADA) is signaled when two data transfers occur from one data bus to the other data bus and vice versa at the same time.

CAUTION

The SD_CMD[4] DDIR bit must be configured before a transfer to indicate the direction of the transfer.

Figure 13-134 shows the buffer management for writing and Figure 13-135 shows the buffer management for reading.

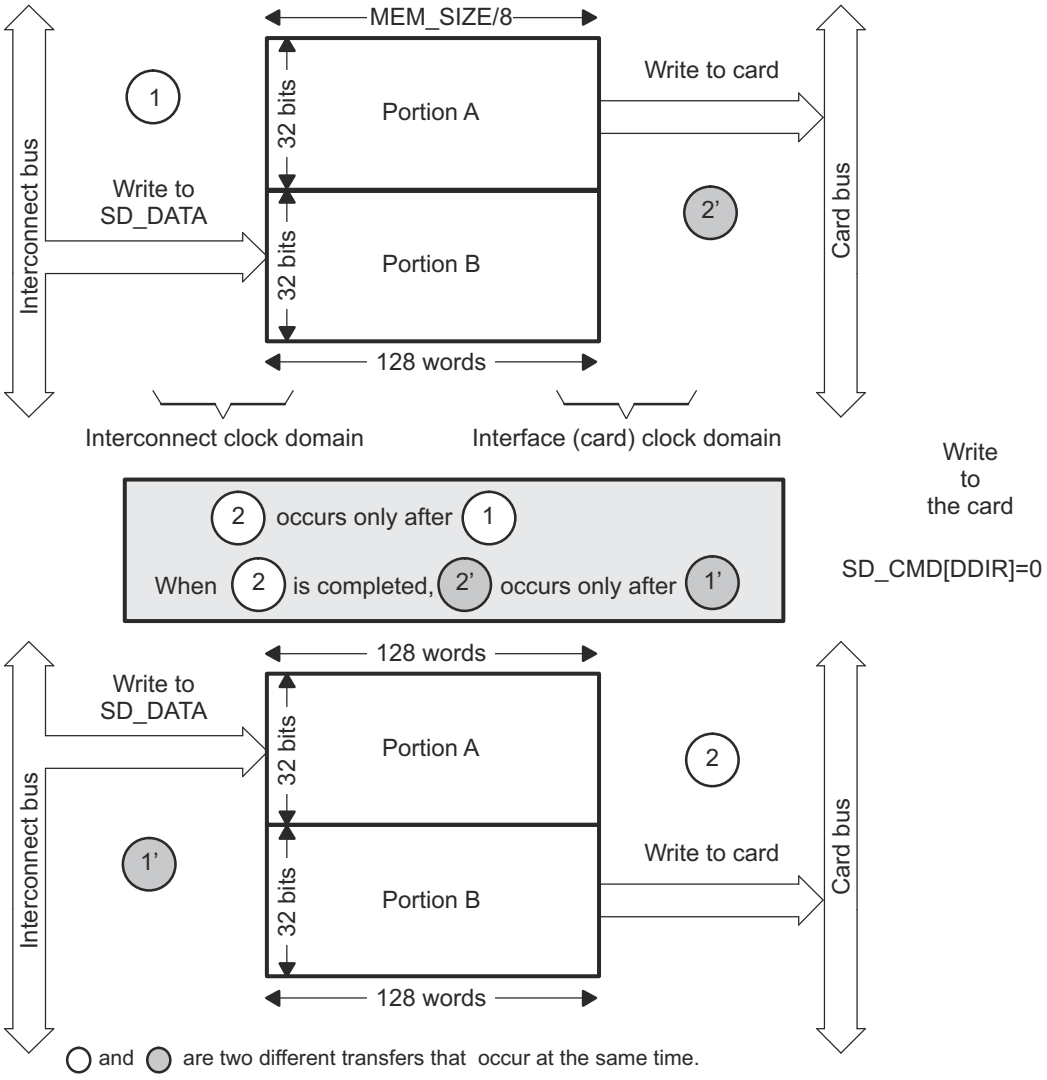


Figure 13-134. Buffer Management for a Write

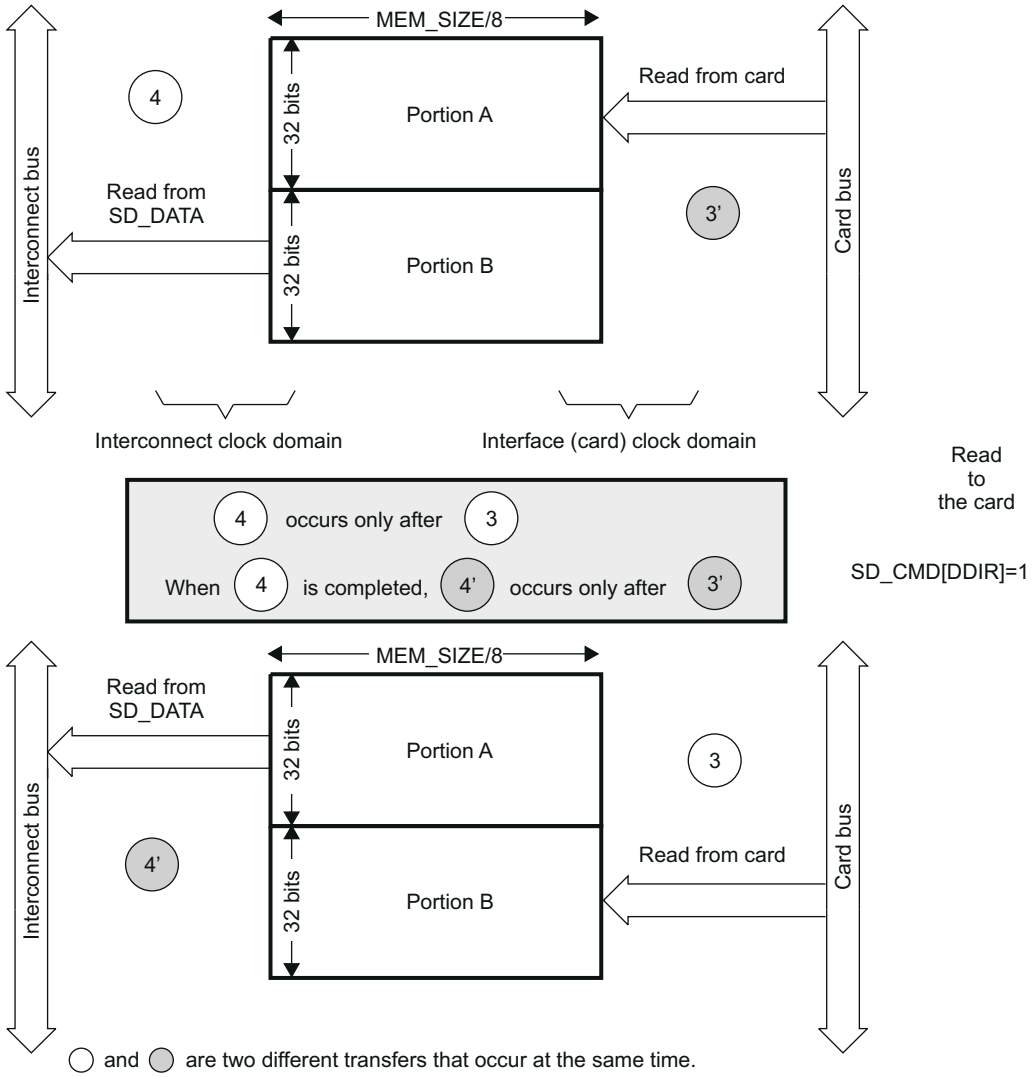


Figure 13-135. Buffer Management for a Read

13.3.1.3.7.1.1 Memory Size, Block Length, and Buffer Management Relationship

The maximum block length and buffer management that can be targeted by system depend on memory depth setting.

Table 13-179. Memory Size, BLEN, and Buffer Relationship

Memory Size([5:2] MEMSIZE in bytes)	512	1024	2048	4096
Maximum block length supported	512	1024	2048	2048
Double-buffering for maximum block length	N/A	BLEN <= 512	BLEN <= 1024	BLEN <= 2048
Single-buffering for block length	BLEN<=512	512 < BLEN <= 1024	1024 < BLEN <= 2048	N/A

13.3.1.3.7.1.2 Data Buffer Status

The data buffer status is defined in the following interrupt status register and status register:

- Interrupt status registers (see SD_STAT):
  - SD_STAT[29] BADA Bad access to data space
  - SD_STAT[5] BRR Buffer read ready
  - SD_STAT[4] BWR Buffer write ready
- Status registers (see SD_PSTATE):
  - SD_PSTATE[11] BRE Buffer read enable
  - SD_PSTATE[10] BWE Buffer write enable

13.3.1.3.8 Transfer Process

The process of a transfer is dependent on the type of command. It can be with or without a response, with or without data.

13.3.1.3.8.1 Different Types of Commands

Different types of commands are specific to MMC, SD, or SDIO cards. See the *Multimedia Card System Specification*, the *SD Memory Card Specifications*, the *SDIO Card Specification, Part E1*, or the *SD Card Specification, Part A2, SD Host Controller Standard Specification* for more details.

13.3.1.3.8.2 Different Types of Responses

Different types of responses are specific to MMC, SD, or SDIO cards. See the *Multimedia Card System Specification*, the *SD Memory Card Specifications*, the *SDIO Card Specification, Part E1*, or the *SD Card Specification, Part A2, SD Host Controller Standard Specification* for more details.

Table 13-180 shows how the MMC, SD, and SDIO responses are stored in the SD_RSPxx registers.

Table 13-180. MMC, SD, SDIO Responses in the SD_RSPxx Registers

Kind of Response	Response Field	Response Register
R1, R1b (normal response), R3, R4, R5, R5b, R6, R7	RESP[39:8](1)	SD_RSP10[31:0]
R1b (Auto CMD12 response)	RESP[39:8](1)	SD_RSP76[31:0]
R2	RESP[127:0](1)	SD_RSP76[31:0] SD_RSP54[31:0] SD_RSP32[31:0] SD_RSP10[31:0]

(1) RESP refers to the command response format described in the specifications mentioned above.

When the host controller modifies part of the SD_RSPxx registers, it preserves the unmodified bits.

The host controller stores the Auto CMD12 response in the SD_RSP76[31:0] register because the Host Controller may have a multiple block data DAT line transfer executing concurrently with a command. This allows the host controller to avoid overwriting the Auto CMD12 response with the command response stored in SD_RSP10 register and vice versa.

13.3.1.3.9 Transfer or Command Status and Error Reporting

Flags in the MMC/SD/SDIO host controller show status of communication with the card:

- A timeout (of a command, a data, or a response)
- A CRC

Error conditions generate interrupts. See Table 13-181 and register description for more details.

Table 13-181. CC and TC Values Upon Error Detected

Error hold in the SD_STAT Register		CC	TC	Comments
29	BADA			No dependency with CC or TC. BADA is related to the register accesses. Its assertion is not dependent of the ongoing transfer.
28	CERR	1		CC is set upon CERR.
22	DEB		1	TC is set upon DEB.
21	DCRC		1	TC is set upon DCRC.
20	DTO			DTO and TC are mutually exclusive. DCRC and DEB cannot occur with DTO.
19	CIE	1		CC is set upon CIE.
18	CEB	1		CC is set upon CEB.
17	CCRC	1		CC can be set upon CCRC - See CTO comment
16	CTO			CTO and CC are mutually exclusive. CIE, CEB and CERR cannot occur with CTO. CTO can occur at the same time as CCRC it indicates a command abort due to a contention on CMD line. In this case no CC appears.

SD_STAT[21] DCRC event can be asserted in the following conditions:

- Busy timeout for R1b, R5b response type
- Busy timeout after write CRC status
- Write CRC status timeout
- Read data timeout
- Boot acknowledge timeout

13.3.1.3.9.1 Busy Timeout for R1b, R5b Response Type

Figure 13-136 shows DCRD event condition asserted when there is a busy timeout for R1b or R5b responses.

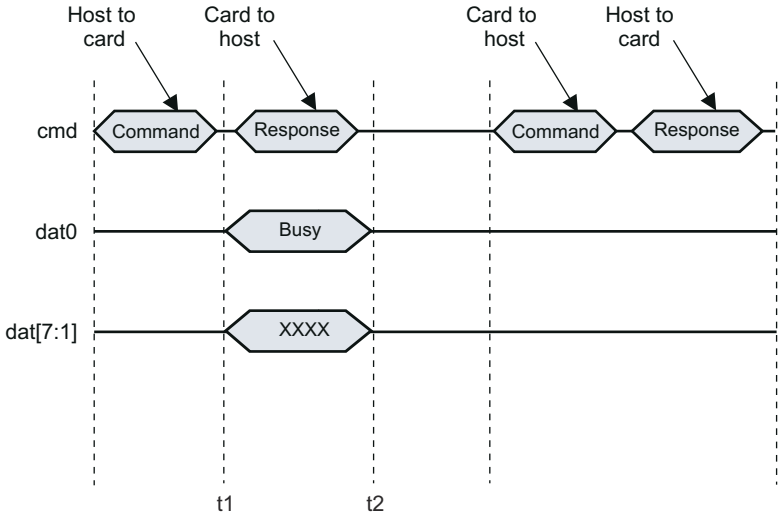


Figure 13-136. Busy Timeout for R1b, R5b Responses

t1 - Data timeout counter is loaded and starts after R1b, R5b response type.

t2 - Data timeout counter stops and if it is 0, SD_STAT[21] DCRC is generated.

13.3.1.3.9.2 Busy Timeout After Write CRC Status

Figure 13-137 shows DCRC event condition asserted when there is busy timeout after write CRC status.

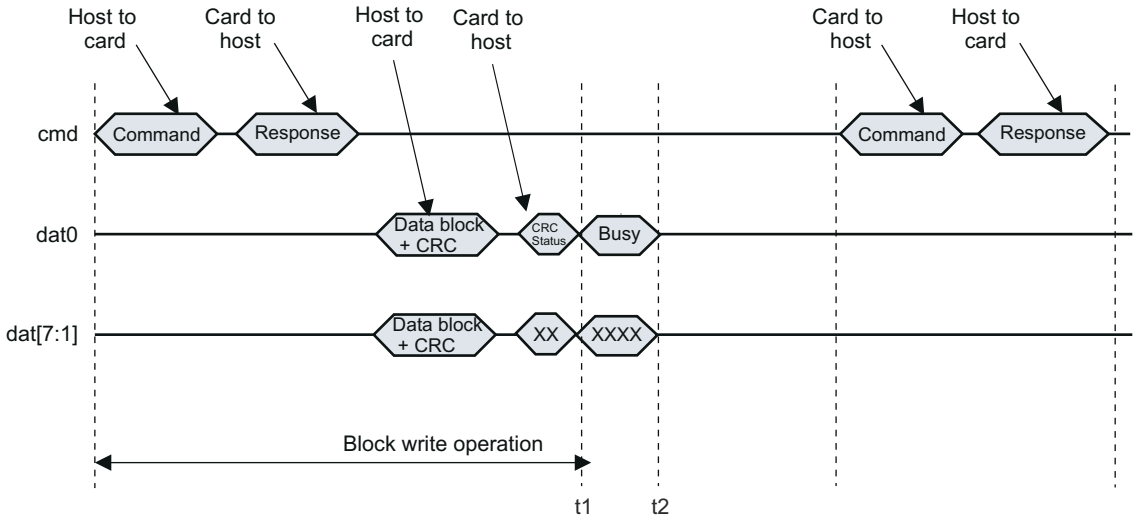


Figure 13-137. Busy Timeout After Write CRC Status

t1 - Data timeout counter is loaded and starts after CRC status.

t2 - Data timeout counter stops and if it is 0, SD_STAT[21] DCRC is generated.

13.3.1.3.9.3 Write CRC Status Timeout

Figure 13-138 shows DCRC event condition asserted when there is write CRC status timeout.

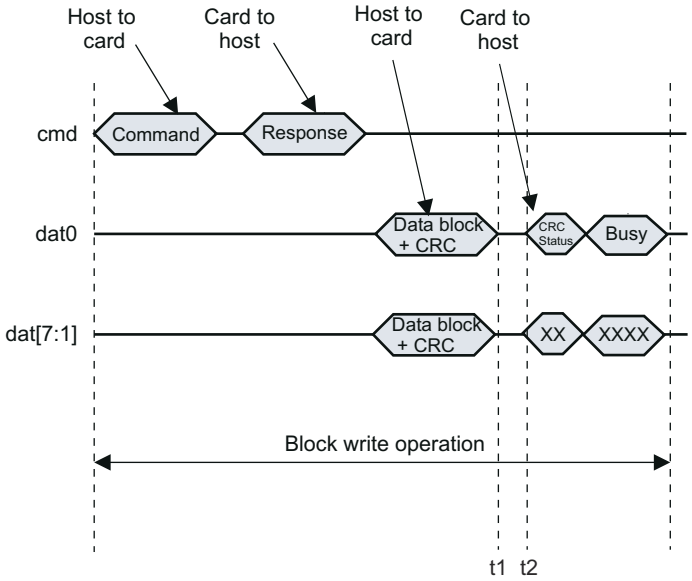


Figure 13-138. Write CRC Status Timeout

t1 - Data timeout counter is loaded and starts after Data block + CRC.

t2 - Data timeout counter stops and if it is 0, SD_STAT[21] DCRC is generated.

13.3.1.3.9.4 Read Data Timeout

Figure 13-139 shows DCRC event condition asserted when there is read data timeout.

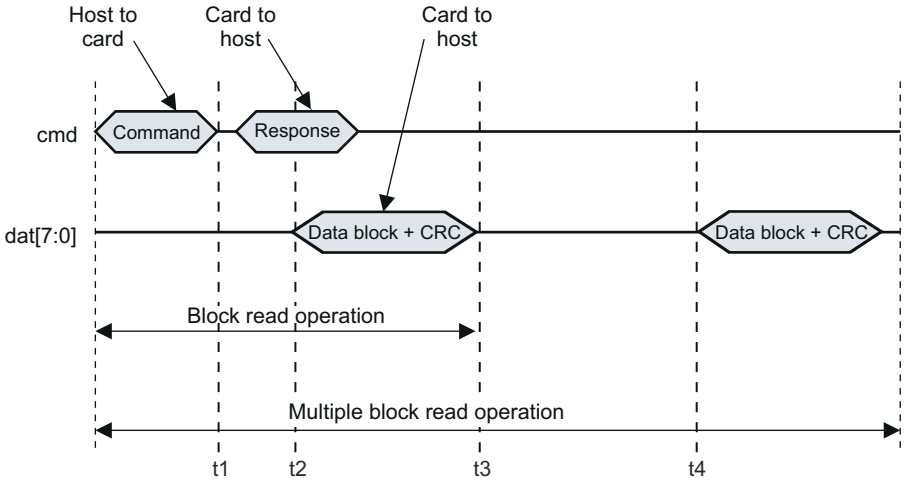


Figure 13-139. Read Data Timeout

t1 - Data timeout counter is loaded and starts after Command transmission.

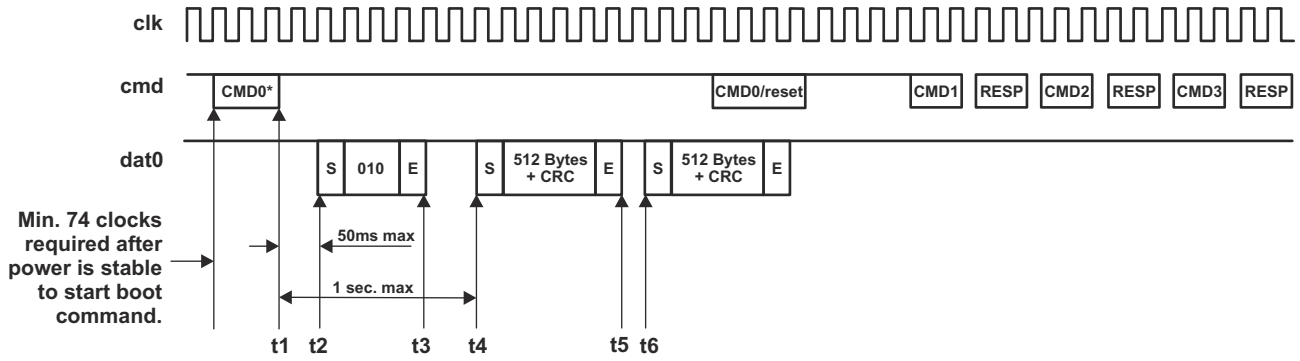
t2 - Data timeout counter stops and if it is 0, SD_STAT[21] DCRC is generated.

t3 - Data timeout counter is loaded and starts after Data block + CRC transmission.

t4 - Data timeout counter stops and if it is 0, SD_STAT[21] DCRC is generated.

13.3.1.3.9.5 Boot Acknowledge Timeout

Figure 13-140 shows DCRC event condition asserted when there is boot acknowledge timeout and CMD0 is used.



* Refer to MMC Specification for correct argument.

Figure 13-140. Boot Acknowledge Timeout When Using CMD0

t1 - Data timeout counter is loaded and starts after CMD0.

t2 - Data timeout counter stops and if it is 0, SD_STAT[21] DCRC is generated.

t3 - Data timeout counter is loaded and starts.

t4 - Data timeout counter stops and if it is 0, SD_STAT[21] DCRC is generated.

t5 - Data timeout counter is loaded and starts after Data + CRC transmission.

t6 - Data timeout counter stops and if it is 0, SD_STAT[21] DCRC is generated.

Figure 13-141 shows DCRC event condition asserted when there is boot acknowledge timeout and CMD line is held low.

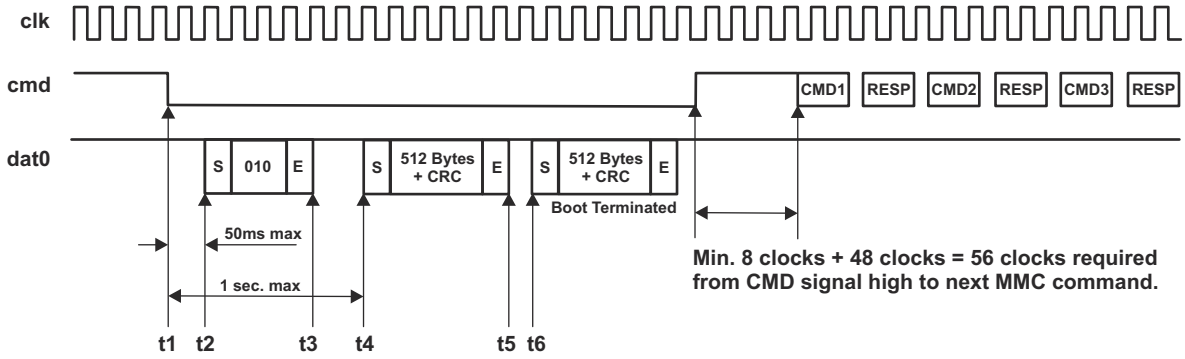


Figure 13-141. Boot Acknowledge Timeout When CMD Held Low

t1 - Data timeout counter is loaded and starts after cmd line is tied to 0.

t2 - Data timeout counter stops and if it is 0, SD_STAT[21] DCRC is generated.

t3 - Data timeout counter is loaded and starts.

t4 - Data timeout counter stops and if it is 0, SD_STAT[21] DCRC is generated.

t5 - Data timeout counter is loaded and starts after Data + CRC transmission.

t6 - Data timeout counter stops and if it is 0, SD_STAT[21] DCRC is generated.

13.3.1.3.10 Auto Command 12 Timings

With the UHS definition of SD cards with higher frequency for MMC clocks up to 208, SD standard imposes a specific timing for Auto CMD12 "end bit" arrival.

13.3.1.3.10.1 Auto Command 12 Timings During Write Transfer

A margin named Ncrc in range of 2 to 8 cycles has been defined for SDR50 and SDR104 card components for write data transfers, as auto command 12 'end bit' shall arrive after the CRC status "end bit".

Figure 13-142 shows auto CMD12 timings during write transfer.

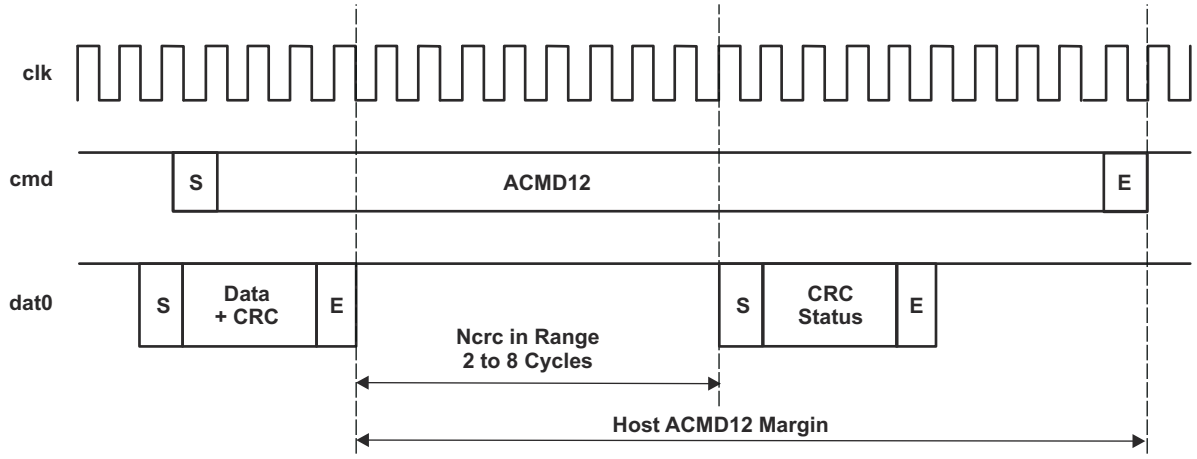


Figure 13-142. Auto CMD12 Timing During Write Transfer

The Host controller has a margin of 18 clock cycles to make sure that auto CMD12 'end bit' arrives after the CRC status. This margin does not depend on MMC bus configuration, DDR or standard transfer, 1,4 or 8 bus width.

13.3.1.3.10.2 Auto Command 12 Timings During Read Transfer

With UHS (Ultra High Speed) cards, the gap timing between 2 successive cards has been extended to 4 cycles instead of 2. This provides more flexibility for Host Auto CMD12 arrival in order to receive the last complete and reliable block. The SD controller only follows the 'Left Border Case' defined by SD UHS specification.

Figure 13-143 shows ACMD12 timings during read transfer.

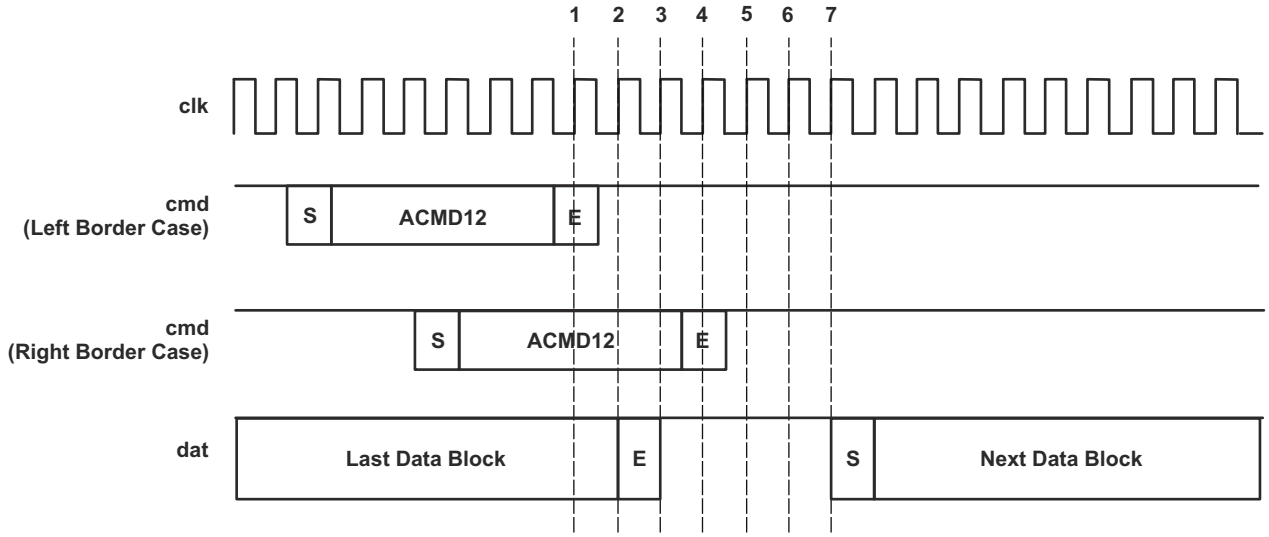


Figure 13-143. Auto Command 12 Timings During Read Transfer

The Auto CMD12 arrival sent by the Host controller is not sensitive to the MMC bus configuration whether it is DDR or standard transfer and whether it is a 1, 4 or 8 bit bus width transfer.

13.3.1.3.11 Transfer Stop

Whenever a transfer is initiated, the transmission may be willed to stop whereas it is still not finished. Several cases can be faced depending on the transfer type:

- Multiple blocks oriented transfers (for which transfer length is known)
- Continuous stream transfers (which have an infinite length)

Note

Since the MMC/SD/SDIO controller manages transfers based on a block granularity, the buffer will accept a block only if there is enough space to completely store it. Consequently, if a block is pending in the buffer, no command will be sent to the card because the card clock will be shut off by the controller.

The MMC/SD/SDIO controller includes two features which make a transfer stop more convenient and easier to manage:

- Auto CMD12 (for MMC and SD only).

  This feature is enabled by setting the SD_CMD[2] ACEN bit to 1 (this setting is relevant for a MMC/SD transfer with a known number of blocks to transfer). When the Auto CMD12 feature is enabled, the MMC/SD/SDIO controller will automatically issue a CMD12 command when the expected number of blocks has been exchanged.
- Stop at block gap

  This feature is enabled by setting the SD_HCTL[16] SBGR bit to 1. When enabled, this capability holds the transfer on until the end of a block boundary. If a stop transmission is needed, software can use this pause to send a CMD12 to the card.

Table 13-182 shows the common ways to stop a transfer, indicating command to send and features to enable.

Table 13-182. MMC/SD/SDIO Controller Transfer Stop Command Summary

		WRITE Transfer		READ Transfer	
		MMC/SD	SDIO	MMC/SD	SDIO
Single block		Transfer ends automatically Wait TC	Transfer ends automatically Wait TC	Transfer ends automatically Wait TC	Transfer ends automatically Wait TC
Multi blocks (finite or infinite)	Before the programmed block boundary	Send CMD12 Wait TC	Send CMD52 Wait TC	Send CMD12 Wait TC	Send CMD52 Wait TC
	Stop at the end of the transfer (finite transfer only)	Auto CMD12 active Transfer ends automatically Wait TC	Set SD_HCTL[16] SBGR bit to 1. Send CMD52 Wait TC	Auto CMD12 active Transfer ends automatically Wait TC	**If READ_WAIT supported** Stop at block gap Wait TC
					If READ_WAIT not supported Send CMD52 Wait TC

Note

The MMC/SD/SDIO controller will send the stop command to the card on a block boundary, regardless the moment the command was written to the controller registers.

13.3.1.3.12 Output Signals Generation

The MMC/SD/SDIO output signals can be driven on either falling edge or rising edge depending on the SD_HCTL[2] HSPE bit. This feature allows to reach better timing performance, and thus to increase data transfer frequency.

13.3.1.3.12.1 Generation on Falling Edge of MMC Clock

The controller is by default in this mode to maximize hold timings. In this case, SD_HCTL[2] HSPE bit is cleared to 0.

Figure 13-144 shows the output signals of the module when generating from the falling edge of the MMC clock.

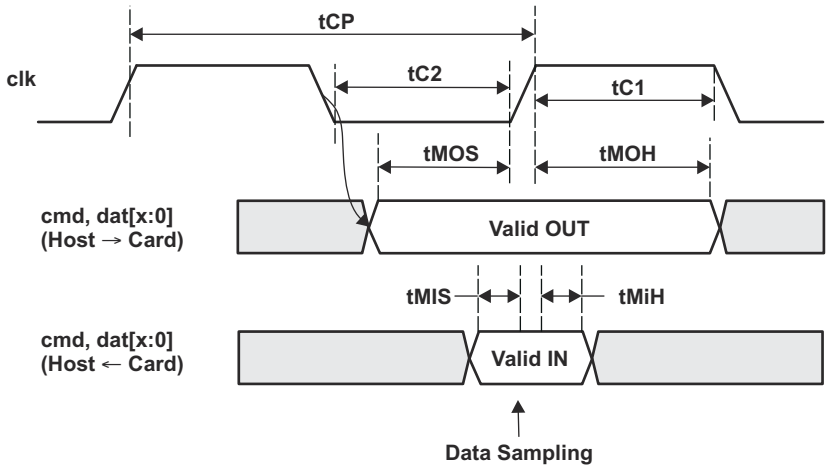


Figure 13-144. Output Driven on Falling Edge

13.3.1.3.12.2 Generation on Rising Edge of MMC Clock

This mode increases setup timings and allows reaching higher bus frequency. This feature is activated by setting SD_HCTL[2] HSPE bit to 1. The controller shall be set in this mode to support SDR transfers.

Note

Do not use this feature in Dual Data Rate mode (when SD_CON[19] DDR is set to 1).

Figure 13-145 shows the output signals of the module when generating from the rising edge of the MMC clock.

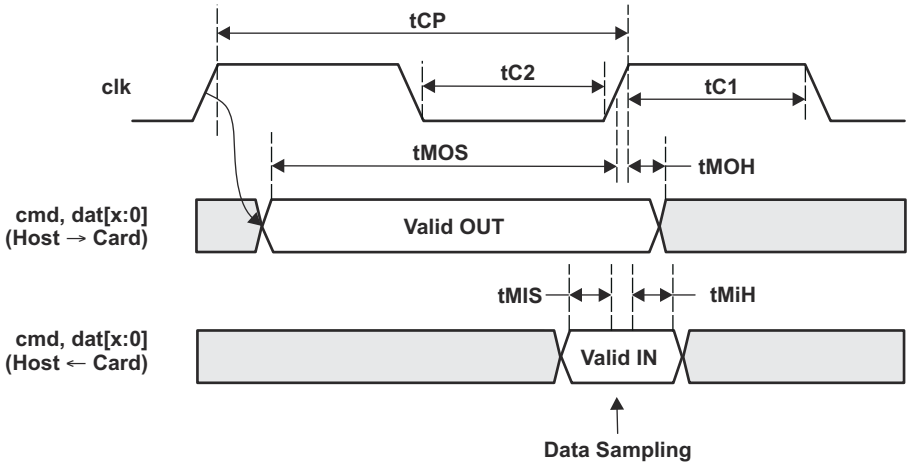


Figure 13-145. Output Driven on Rising Edge

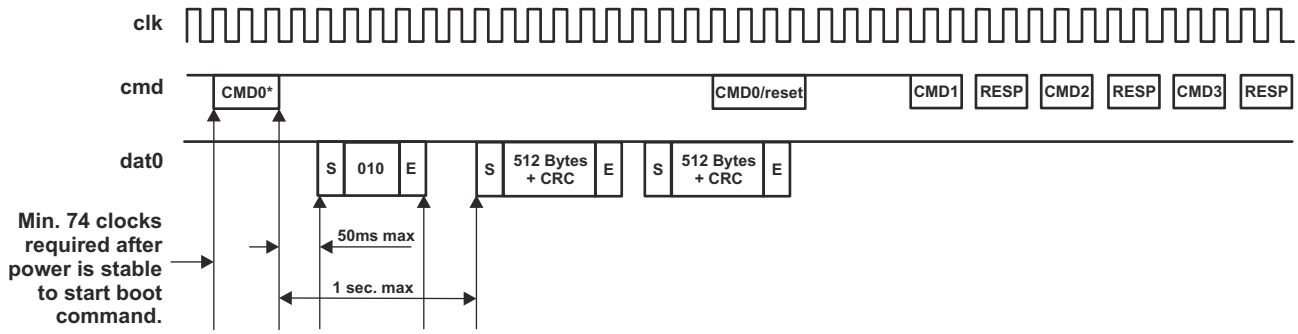
13.3.1.3.13 Card Boot Mode Management

Boot Operation Mode allows the MMC/SD/SDIO host controller to read boot data from the connected slave (MMC device) by keeping CMD line low after power-on (or sending CMD0 with specific argument) before issuing CMD1. The data can be read from either boot area or user area, depending on register setting. Power-on boot defines a way for the boot-code to be accessed by the MMC/SD/SDIO host controller without an upper-level software driver, speeding the time it takes for a controller to access the boot code.

The two possible ways to issue a boot command (either issuing a CMD0 or driving the CMD line to 0 during the whole boot phase) are described in the following sections.

13.3.1.3.13.1 Boot Mode Using CMD0

Figure 13-146 shows the timing diagram of a boot sequence using CMD0.



* Refer to MMC Specification for correct argument.

Figure 13-146. Boot Mode With CMD0

- Configure:
  - SD_CON[BOOT_CF0] to 0
  - SD_CON[BOOT_ACK] (if an acknowledge will be received) to 0x1
  - SD_BLK with the correct block length and number of block
  - SD_SYSCTL[DTO] for timeout

  If transfer is done in DDR mode also set SD_CON[DDR] to 1.
- Write register SD_ARG with correct argument (see MMC Specification).
- Write in SD_CMD register to start CMD0 transfer with these bit fields set:
  - INDX set to 0x00
  - DP set to '1'
  - DDIR set to '1'
  - MSBS set to '1'
  - BCE set to '1'
- If boot status is not received within the timing defined, the SD_STAT[DTO] will be generated. Otherwise the SD_STAT[BSR] is arisen.
- After the transfer is complete, the controller will generate the SD_STAT[TC], and then the system can emit another CMD0 (SD_CON[BOOT_ACK] previously cleared to 0x0) to exit the card from boot state.
- If the system wants to abort the boot sequence it must issue a CMD0 with SD_CMD[CMD_TYPE] set to 0x3 (SD_CON[BOOT_ACK] previously cleared to 0x0) during the transfer to abort transfer and enable card to exit from boot state.

13.3.1.3.13.2 Boot Mode With CMD Held Low

Figure 13-147 shows the timing diagram of a boot sequence with CMD line tied to 0.

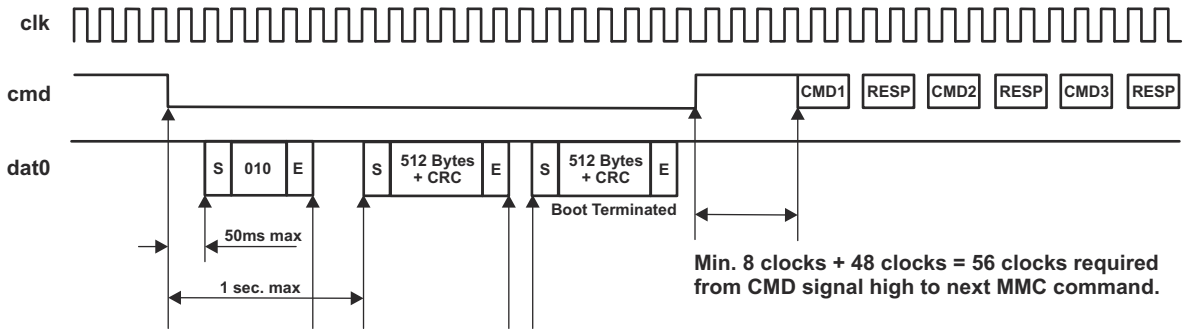


Figure 13-147. Boot Mode With CMD Line Tied to 0

- Configure:
  - SD_CON[BOOT_CF0] and SD_CON[BOOT_ACK] (if an acknowledge will be received) to 0x1
  - SD_BLK with correct block length and number of block
  - SD_SYSCTL[DTO] for timeout

  If transfer is done in DDR mode also set SD_CON[DDR] to 1.
- Write in SD_CMD register to start boot sequence with:
  - DP set to '1'
  - DDIR set to '1'
  - MSBS set to '1'
  - BCE set to '1'

  This leads the controller to force CMD line to '0'.
- If the boot status is not received within the timing defined, the SD_STAT[DTO] will be generated. Otherwise the SD_STAT[BSR] is arisen.
- After the transfer is complete, the controller will generate the SD_STAT[TC], and then the system must clear SD_CON[BOOT_CF0] to 0x0 to release the CMD line and enable the card to exit from boot state.
- If the system wants to abort the boot sequence it must clear SD_CON[BOOT_CF0] to 0x0 during transfer to enable the card to exit from boot state.

13.3.1.3.14 CE-ATA Command Completion Disable Management

The MMC/SD/SDIO controller supports CE-ATA features, in particular the detection of command completion token. When a command that requires a command completion signal (SD_CON[12] CEATA and SD_CMD[2] ACEN set to 1) is launched, the host system is no longer allowed to emit a new command in parallel of data transfer unless it is a command completion disable token.

The settings to emit a command completion disable token follow:

- SD_CON[12] CEATA is set to 1.
- SD_CON[2] HR set to 1.
- Clear the SD_ARG register.
- Write into SD_CMD register with value 0000 0000h.

When a command completion disable token was emitted (that is, SD_STAT[0] CC received), the host system is again allowed to emit another type of command (for example a transfer abort command CMD12 to abort transfer).

A critical case can be met when command completion signal disable (CCSD) is emitted during the last data block transfer, the sequence on command line could be sent very close to command completion signal (CCS) token sent by the card.

Three cases can be met:

- CCS is receive just before CCSD is emitted:

  An interrupt CIRQ is generated with CCS detection, CCSD is transmitted to card then an interrupt CC is generated when CCSD ends. In this case, card consider the CCSD sequence.
- CCS is not generated or generated during the CCSD transfer:

  The CCS bit cannot be detected (conflict is not possible as they drive the same level on command line, then no CIRQ interrupt is generated; besides CC interrupt is generated when CCSD ends).
- CCS is generated without CCSD token required:

  Only the interrupt CIRQ is generated when CCS is detected.

13.3.1.3.15 Test Registers

Test registers are available to be compliant with SD Host controller specification. This feature is useful to generate interrupts manually for driver debugging. The Force Event register (SD_FE) is used to control the Error Interrupt Status and Auto CMD12 Error Status. The System Test register (SD_SYSTEST) is used to control the signals that connect to I/O pins when the module is configured in system test (SD_CON[4] MODE = 1) mode for boundary connectivity verification.

13.3.1.3.16 MMC/SD/SDIO Hardware Status Features

Table 13-183 summarizes the MMC/SD/SDIO hardware status features.

Table 13-183. MMC/SD/SDIO Hardware Status Features

Feature	Type	Register/Bit Field/Observability Control	Description
Interrupt flags		See Section 13.3.1.3.4.	
CMD line signal level	Status	[24] CLEV	Indicates the level of the cmd line
DAT lines signal level	Status	[23:20] DLEV	Indicates the level of the data lines
Buffer read enable	Status	[11] BRE	Readable data exists in the buffer.
Buffer write enable	Status	[10] BWE	Indicates whether there is enough space in the buffer to write BLEN bytes of data
Read transfer active	Status	[9] RTA	This status is used for detecting completion of a read transfer.
Write transfer active	Status	[8] WTA	This status indicates a write transfer active.
Data line active	Status	[2] DLA	Indicates whether the data lines are active
Command Inhibit (data lines)	Status	[1] DATI	Indicates whether issuing of command using data lines is allowed
Command inhibit (CMD line)	Status	[0] CMDI	Indicates whether issuing of command using CMD line is allowed

13.3.1.4 Low-Level Programming Models

13.3.1.4.1 Surrounding Modules Global Initialization

This section identifies the requirements of initializing the surrounding modules when the module has to be used for the first time after a device reset. This initialization of surrounding modules is based on the integration and environment of the MMC/SD/SDIO modules.

Table 13-184. Global Init for Surrounding Modules

Surrounding Modules	Comments
Power, Reset, and Clocking	Module interface and functional clocks must be enabled. For more information on power, reset, and clock management, see the corresponding sections within the *Device Configuration* chapter.
Control Module	Module-specific pad muxing and configuration must be set in the control module. For more information, see the *Control Module* section within the *Device Configuration* chapter.
(optional) MPU INTC	MPU INTC configuration must be done to enable the interrupts from the SD module. See , *Interrupts*.
(optional) EDMA	DMA configuration must be done to enable the module DMA channel requests. See , *EDMA*.
(optional) Interconnect	For more information about the interconnect configuration, see *Interconnects*.

Note

The MPU interrupt controller and the EDMA configurations are necessary, if the interrupt and DMA based communication modes are used.

13.3.1.4.2 MMC/SD/SDIO Controller Initialization Flow

The next sections outline the four steps to initialize the MMC/SD/SDIO controller:

- Initialize Clocks
- Software reset of the controller
- Set module's hardware capabilities
- Set module's Idle and Wake-Up modes

13.3.1.4.2.1 Enable OCP and CLKADPI Clocks

Prior to any SD register access one must enable the SD OCP clock and CLKADPI clock in PRCM module registers. For more information, see , *Power, Reset, and Clock Management*.

13.3.1.4.2.2 SD Soft Reset Flow

Figure 13-148 shows the soft reset process of MMC/SD/SDIO controller.

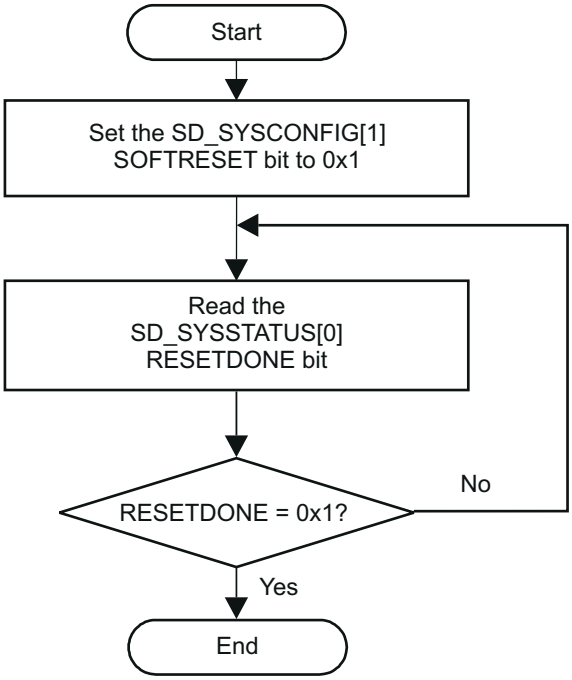


Figure 13-148. MMC/SD/SDIO Controller Software Reset Flow

13.3.1.4.2.3 Set SD Default Capabilities

Software must read capabilities (in boot ROM for instance) and is allowed to set (write) SD_CAPA[26:24] and SD_CUR_CAPA[23:0] registers before the MMC/SD/SDIO host driver is started.

13.3.1.4.2.4 Wake-Up Configuration

Table 13-185 details SD controller wake-up configuration.

Table 13-185. MMC/SD/SDIO Controller Wake-Up Configuration

Step	Access Type	Register/Bit Field/Programming Model
Configure wake-up bit (if necessary).	W	SD_SYSCONFIG[2] ENAWAKEUP
Enable wake-up events on SD card interrupt (if necessary).	W	SD_HCTL[24] IWE
SDIO Card onlyEnable card interrupt (if necessary).	W	SD_IE[8] CIRQENABLE

13.3.1.4.2.5 MMC Host and Bus Configuration

Figure 13-149 details the MMC bus configuration process.

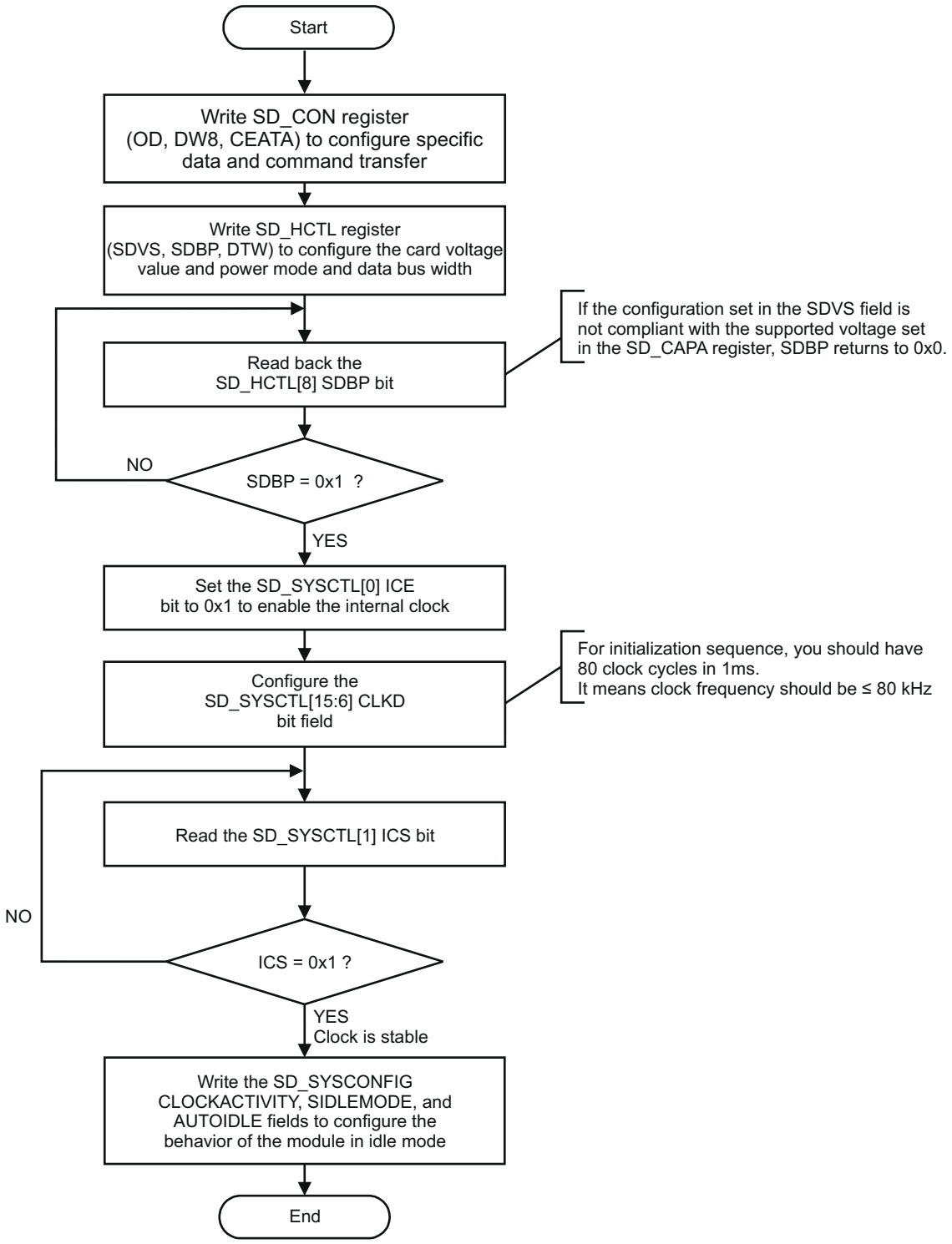


Figure 13-149. MMC/SD/SDIO Controller Bus Configuration Flow

13.3.1.4.3 Operational Modes Configuration

13.3.1.4.3.1 Basic Operations for MMC/SD/SDIO Host Controller

The MMC/SD/SDIO controller performs data transfers: data to card (referred to as write transfers) and data from card (referred to as read transfers).

The host controller requires transfers to run on a block-by-block basis, rather than on a DMA burst size basis. A single DMA request (or block request interrupt) is signaled for each block. Pipelining is supported as long as the block size is less than one half of the memory buffer size.

13.3.1.4.3.2 Card Detection, Identification, and Selection

Figure 13-150 and Figure 13-151 show the card identification and selection process.

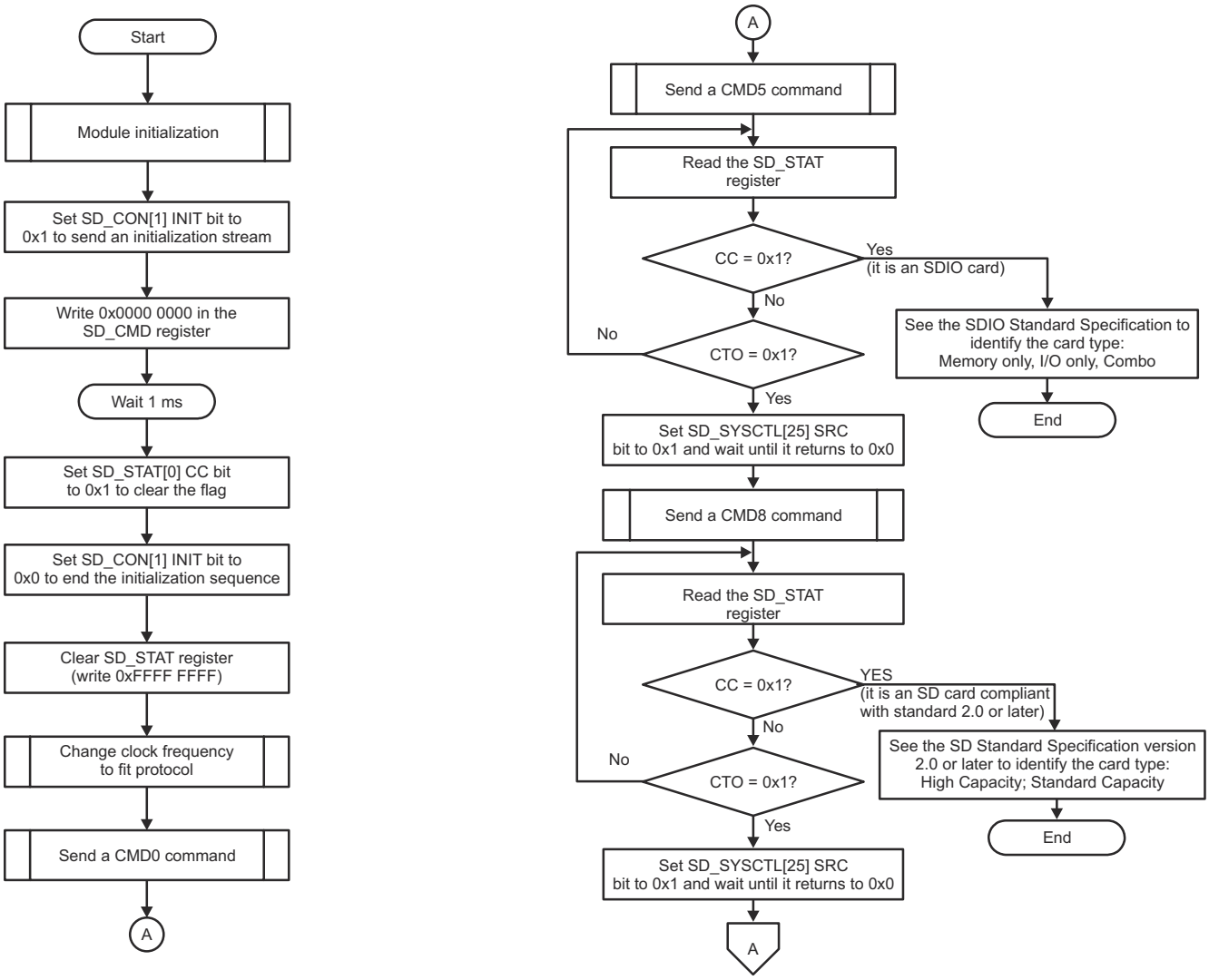


Figure 13-150. MMC/SD/SDIO Controller Card Identification and Selection - Part 1

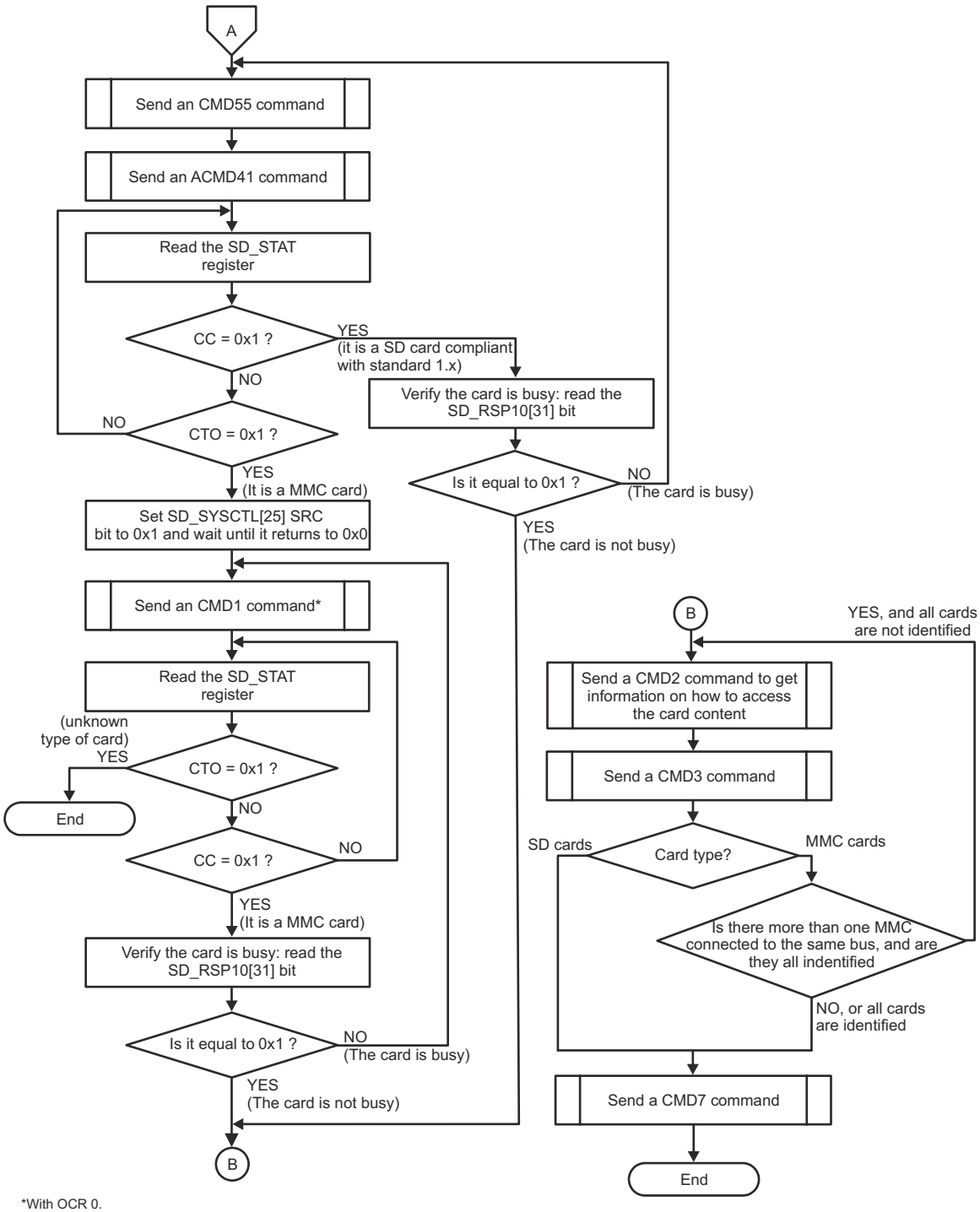


Figure 13-151. MMC/SD/SDIO Controller Card Identification and Selection - Part 2

13.3.2 OptiFlash Submodules

13.3.2.1 Regional L2-Cache (RL2)

13.3.2.1.1 Overview RL2

The Remote L2 (RL2) is responsible for caching 1 to 16MB of target space using system memory for the actual cache data storage (remote cache data storage memory). This significantly reduces the size of the L2 cache module while allowing flexible configuration for target applications.

The RL2 does not modify any request for the CPU that is within a RL2 caching range. This allows the critical word first (CWF) to be processed by the downstream module for best performance. A downstream CBA3-to-CBA4 bridge is suggested if it's required for a downstream CBA4 fabric be able to split any request that is a wrapping burst that is not aligned to the burst size boundary and targeting system SRAM.

Since the RL2 is expected to cache data from a slow peripheral like a Flash device, the request is aligned to the burst boundary and the RL2 restores the critical word first returned data to the CPU. RL2 supports split burst returns, all requested burst must be returned as full bus words (8 bytes). Split burst returns less than 8 bytes disable the RL2 or post an error.

The Remote L2 (RL2) is an 8 Way set associative read allocate LRU cache. The RL2 allocation is based on full cache line reads. The *SET* index is based on the programmed operating size.

The RL2 uses 32-byte cache lines which matches the R5 CPU cache line so that best performance is achieved. That is, the RL2 does not read more data than the CPU requested. Once a cache line is in the RL2, the CPU can read any quanta of data from that cache line.

Since the RL2 cache data is in system memory, the system must ensure the RL2 is protected from other accesses to prevent cache corruption. The RL2 passes the secure, priv and privid from the originating initiator for any operations that *DO NOT* go to the translated remote cache data storage memory. Any transaction to the translated remote cache data storage memory contains the secure, priv and privid from the RL2 Privilege InterFace (PIF).

Since the RL2 filters an address range to be cached, the RL2 passes through any address not within that range or crosses a cache line boundary, the SOC must ensure that the placement of the RL2 does not create a loop in the fabric. The proper use of the RL2 is that the CPUs are upstream to the RL2, and the flash, and remote cache data storage memory is downstream to the RL2.

The RL2 can be used to cache peripherals like flash, PCIe or Hyperlink, so the SOC need to ensure that these targets are downstream of the RL2.

The RL2 does not change the size or 32-byte offset of any transaction going through the module, that is, the byte count or offset (caddress[4:0]) of any command are left as is. This allows the cbytecnt, wcnt, rcnt, rbytecnt and ralign[4:0] to be passed through unmodified. Since the remote cache data storage memory addresses are aligned to 2K boundaries, and WAYs are aligned to 32 boundaries, the offset alignment within a cache access to the remote cache data storage memory is always the same as the requested offset.

The RL2 support a Dual Mode which allows two cache lines to share a single WAY, offering double the remote cache data storage memory while only using the same number of total cache line entries supported. Since Dual Mode shares the same bits within a WAY, the cacheable range is reduced to support the management for the two sub cache lines.

13.3.2.1.1.1 Supported Features

RL2 Features

- UP to 4096 L2 cache lines supported
- 32 byte cache line size supported
- configurable cache size up to 4096 entries
  - Up to 1MB of cacheable target space for 8KB of cache
  - Up to 2MB of cacheable target space for 16KB of cache
  - Up to 4MB of cacheable target space for 32KB of cache
  - Up to 8MB of cacheable target space for 64KB of cache
  - Up to 16MB of cacheable target space for 128KB of cache
  - Up to 8MB of cacheable target space for 256KB of cache (Dual Mode)
- 8-Way set associative LRU Cache
- ECC protected Tag LRU RAM

13.3.2.1.2 Environment

This section describes the OptiFlash module's connectivity and environment

13.3.2.1.2.1 Connectivity Examples

The Remote L2 OptiFlash module (RL2_OF) can be connected to multiple CPU or per CPU.

In the below figure, an example is shown where a single RL2_OF is used to RAT, FLC, and cache Flash code for multiple CPU

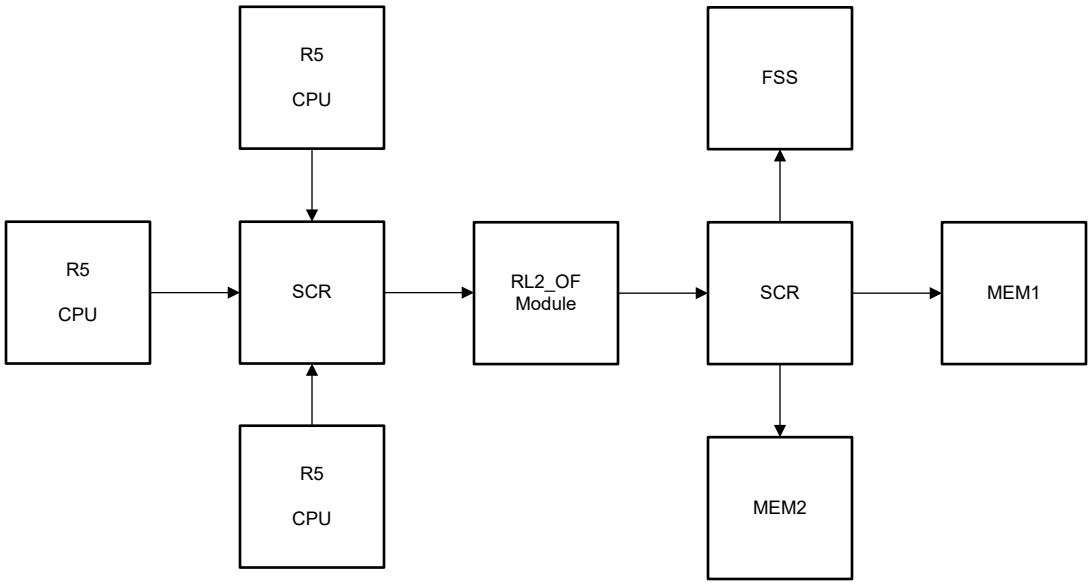


Figure 13-152. Connectivity of the RL2_OF Module - Multiple CPU

In the below example connectivity of a single RL2_OF for each CPU to RAT, FLC or cache Flash code. The benefit is that each CPU has a full use of a RL2_OF. However, if CPUs are executing code in common areas, would result in reading the same data for each CPU sharing code.

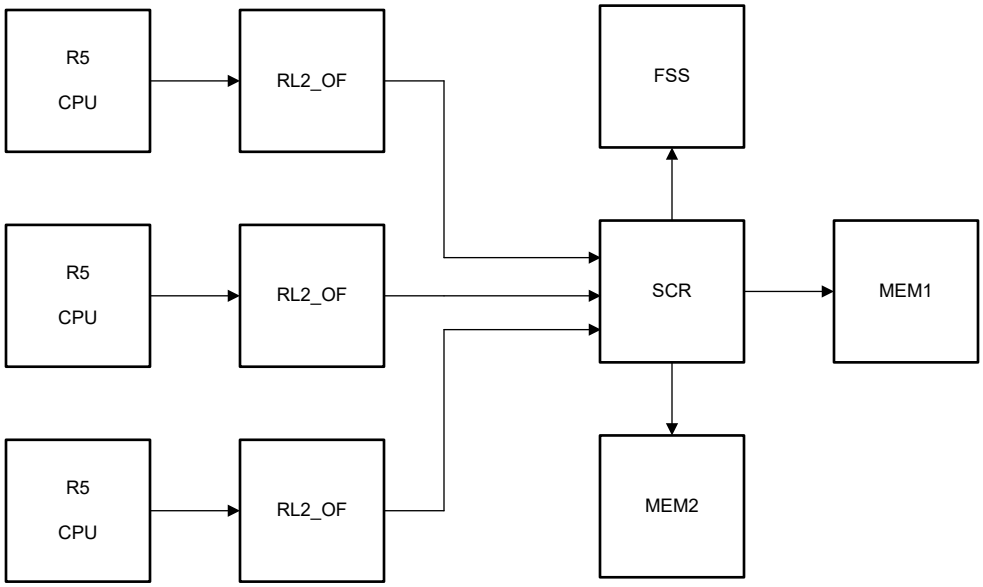


Figure 13-153. Connectivity of the RL2 Module - Per CPU

13.3.2.1.3 Functional Description

13.3.2.1.3.1 Block Diagram

Below is the Block Diagram for the OptiFlash module

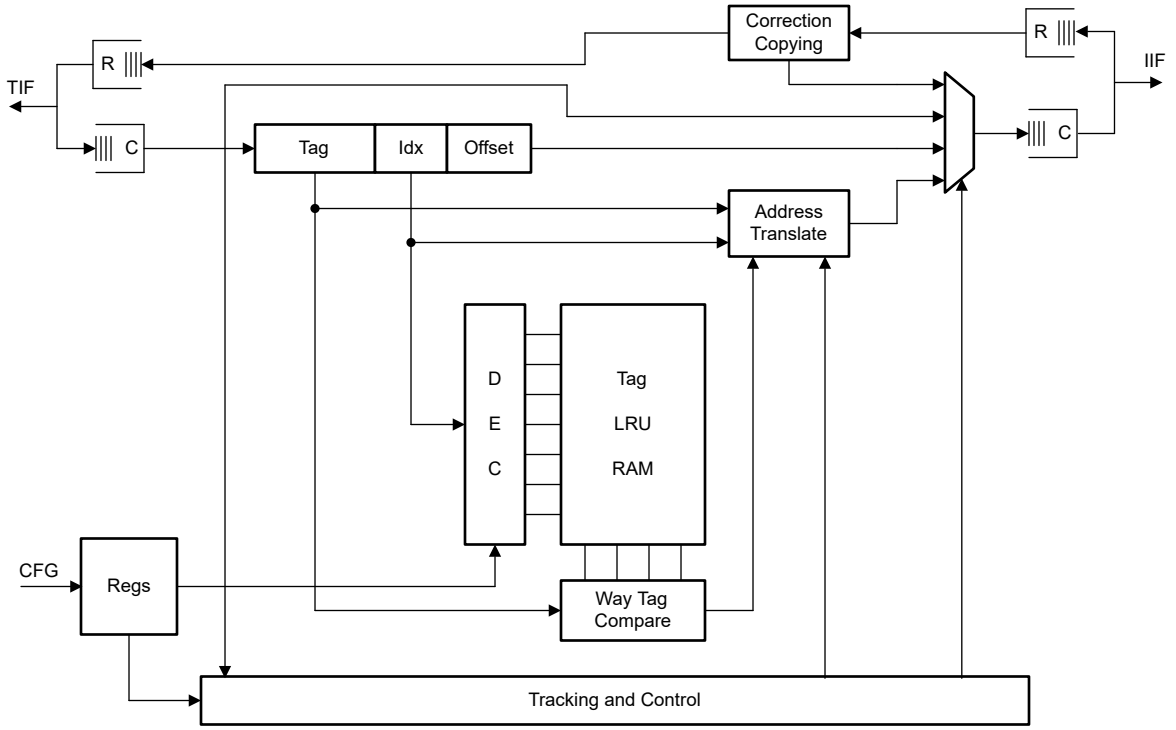


Figure 13-154. RL2 OptiFlash Block Diagram

13.3.2.1.3.1.1 C and R FIFO Blocks

The C and R FIFO blocks deal with the bus protocol to make sure the bus is retimed at the interface of the module

13.3.2.1.3.1.2 Correction and Copying Block

The Correction and Copying block are responsible for fixing the bus read alignment, ID back to meet the CBA requirements and also copies the data to be written to the remote cache data storage memory for RL2 cache data storage accesses.

13.3.2.1.3.1.3 Address Translation Block

The Address Translation block is responsible for remapping the CBA address for the RAT, FLC and RL2

For the RAT, the Address Translation Block it is purely an address translation.

For the FLC, the address is translated to the FLC remote SRAM when that address is within the FLC range and the data has previously been copied. Any access to the FLC remote SRAM uses the PIF security contest.

For RL2 of the request to the address of the remote cache data storage memory when a cache line hit is detected. The remote cache data storage memory address is referenced by *SET/WAY* so that a reduce *SET* selection can occur.

13.3.2.1.3.1.4 Tag LRU RAM Block

The tag LRU RAM within the RL2 holds the LRU states, *WAY* valid bits and the address tag for each *WAY* by the number of *SETs* for each cache line stored in the remote cache data storage memory

For each *WAY,* there are 3 bits for the LRU, a data valid bit, an address valid bit, and the 10 tag bits for address compare. In Dual mode, the two MS bits of the tag are used for the sub line valid bits. The format of the TAG RAM WAY in Dual Mode is 3 bits for the LRU, two data valid bits hi and lo, two address valid bits hi and lo, and the 8 tag bits

Based on the number of *SETs* in play, the tag is compared to the portion of the address just above the *SET* bits. The *SET* bits are normally the bits just above the address bits within the 32-byte burst. That is the *SET* bits are

typically the address bits [n:5] where the address bits [4:0] are the byte address within the 32 byte burst. The width of the *SET* bits is controlled by the *rl2_regs_ctrl.size* register

Whatever the specified *rl2_regs_ctrl.size* register is programmed to; the tag gets compared to the bits just above the masked *SET* bits. This allows more memory to be cacheable when the *rl2_regs_ctrl.size* register specifies a greater amount of enabled cache

13.3.2.1.3.1.5 DEC Block

The DEC block decodes the incoming address for the *SET* selection, when using smaller L2 size configurations, the DEC block maskes out the bits so that a reduces *SET* that is used.

13.3.2.1.3.1.6 Regs Block

The Regs block stores the address ranges for L2 cache and well as the address ranges for remote cache data storage memory.

13.3.2.1.3.1.7 Way Tag Compare Block

The Way Tag Compare block determines which *'WAY'* the address was found within so that the address can be corrected accordingly.

13.3.2.1.3.1.8 Mux Block

The Mux block control which command is forwarded to the initiator interface.

13.3.2.1.3.1.9 Tracking and Control Block

The Tracking and Control block controls which addresses are Translated by the RAT, copied by the FLC or cached in the RL2 and manipulates the bus to save the data while returning the data to the original initiator.

13.3.2.1.3.2 Privilege Passing and Manipulation

The FLC passes the originating *secure, priv,* and *privid* from the original initiator on any command that is passed without address translation. All accesses to the FLC remote SRAM or accesses that copy the FLC range from the Flash carry the Privilege InterFace (PIF) *secure, priv,* and *privid* values. This allows the FLC remote data memory firewall to be able to determine who is the source of the operation and protect the FLC remote data memory from other initiators.

Note

Since FLC copy accesses to the Flash also use the PIF security context, the system must allow the PIF security context to access the FLC range in the Flash.

The RL2 passes the originating *secure, priv,* and *privid* from the original initiator on any command that is not modified due to L2 hit detection. When the remote cache data storage memory is written due to a read allocate or read due to an L2 hit; the Privilege InterFace (PIF) *secure, priv,* and *privid* values are used. This allows the remote cache data storage memory firewall to be able to determine who is the source of the operation and protect the cache data storage from other initiators.

13.3.2.1.3.3 Error Handling and Flushing

For all commands that are passed through, the return status is also passed through

For any read within cacheable range that is not currently cached, that meets RL2 allocation policy, and returns a read error, the following occurs:

- The error is passed through to the initiator
- The remote caceh data storage memory is written with the errant data.
- The cache line is marked invalid
- If not in Dual Mode or only a single cache line is allocated within the WAY, the WAY LRU is reverted and marked as the oldest WAY for future allocations

Any write to the cacheable range as defined by *L2_LO*i>=CachedRange<=*L2_HI* while the RL2 cache is enabled and operating, logically disables the RL2 cache and set the *wr_hit* error bit in the Interrupt Raw Status Register.

It is expected that the remote cache data storage memory area is protected using the ~PFI interface ~priv, *privid,* and *secure* parameters.

Any write to the remote cache data storage memory due to a read allocation that returns a write error, logically disables the RL2 cache and set the *wr_err* error bit in the Interrupt Raw Status Register.

13.3.2.1.3.4 Tightly Coupled RL2 Tag Memory

The RL2 has Memory just outside the module for tag information that is *SETs* deep and *WAYs* times *WAY* info wide. This memory start auto initialization when a write to the *rl2_regs_ctrl.size* field occurs that changes the current size, or the rl2_regs_ctrl.enable bit is changed from a '0' to a '1' and all pending operations have completed. The RL2 performs any correctable ECC writeback in the event ECC is enables and a single error correct occurs. The memory is not used until the *rl2_regs_ctrl.size* field is written and the *enable bit is set.*

The tag memory auto initialization preloads the LRU info into each way of all the defined sets.

13.3.2.1.3.5 Remote Cache Data Storage Memory

The remote cache data storage memory stores the cache data managed by the RL2 module. This storage holds a number of 64 byte blocks as specified by the *rem*_len* registers. The remote cache data storage memory stores the 8 *WAYs* consecutively for each *SET* address. In Dual Mode each *WAY* has two 32 byte cache lines (64 bytes) versus other *l2_ctrl.size* values which only hold a single 32 byte cache line *WAY.* In dual Mode the *WAY* has a high and low sub cache line, the high is stored at the higher address and the low is stored at the *WAY* base.

Table 13-186. Remote Cache Data Storage Example

WAY N... WAY 0 - SET 0
WAY N... WAY 0 - SET 1
...
WAY N... WAY 0 - SET (M-1)
WAY N... WAY 0 - SET M

13.3.2.1.3.6 RL2 Cache Allocation

The RL2 does not modify the length of any transaction that the RL2 processes send to the slow memory it is intending to cache. As described earlier the RL2 allocation is based on full 32 byte read burst within the range of the specified target area. Once cached, any size or wrapping burst request to the same line can occur and can read the cached data. In the event that a allocated cache line returns an error on the read from the cacheable target, the allocation will restore the LRU aging such that the next allocated *WAY* is invalidated due to the target response error.

13.3.2.1.3.7 Dual Mode

The Dual Mode is an extension of the RL2 that supports two times that max remote cache data storage memory while keeping the tag RAM the same size. That is each way instead of representing a single cache line, represents two consecutive cache lines. This allows the remote cache data storage memory to be twice as large using the same tag RAM. When the *l2_ctrl.size* is set to 5 the dual mode is selected. In this mode there are the same number of cache line entries (4096), but the remote cache data storage memory is doubled. Each *WAY* now supports 64 bytes of cache data in two halves, a hi and lo sub cache line. This allows twice as much memory while keeping the same number of cache entries. When in this mode, the tag field is reduced by two bits to make room for the extra valid bits for the two sub lines held. This reduction reduces the cacheable range the RL2 can cache.

13.3.2.1.3.8 Emulation Debug

When the *EMUDBG* signal is set for a RAT transaction the region address translation occurs.

When the *EMUDBG* signal is set for a any transaction, the FLC pauses the copy function if copying until an non *EMUDBG* transaction is seen. This simulate what happens if user is single stepping the processor. The *dbgsusp* port also pauses the FLC without requiring a transaction. The FLC resumes after *dbgsusp* port goes back low.

When the *EMUDBG* signal is set for a RL2 transaction, the cache state is maintained. That is, *EMUDBG* request won't allocate a cache line in the case of a 'miss', or update the LRU state in the case of a 'hit'. But if the address is within the cache, the *EMUDBG* request gets the data from the remote cache data storage memory.

13.3.2.1.3.9 ASIL-D/SIL-3 Safety

The Automotive Safety Integrity Level (ASIL) or Safety Integrity Level (SIL) rating is determined by how OptiFlash is used in the SoC.

ASIL-D/SIL-3 can be obtained by duplicating the OptiFlash, supplying the same inputs to both copies, and comparing all the outputs for consistency (including RAM controls). If any output between the two copies do not match, an error has occurred. It is expected that both copies see the same external TAG memory contents. The RAM controls should be checked for consistency in this configuration.

There are cases that may require the safety version of the OptiFlash to be delayed by one cycle, so delaying all inputs to the safety copy and creating a delayed version of the outputs of the primary OptiFlash before comparing outputs.

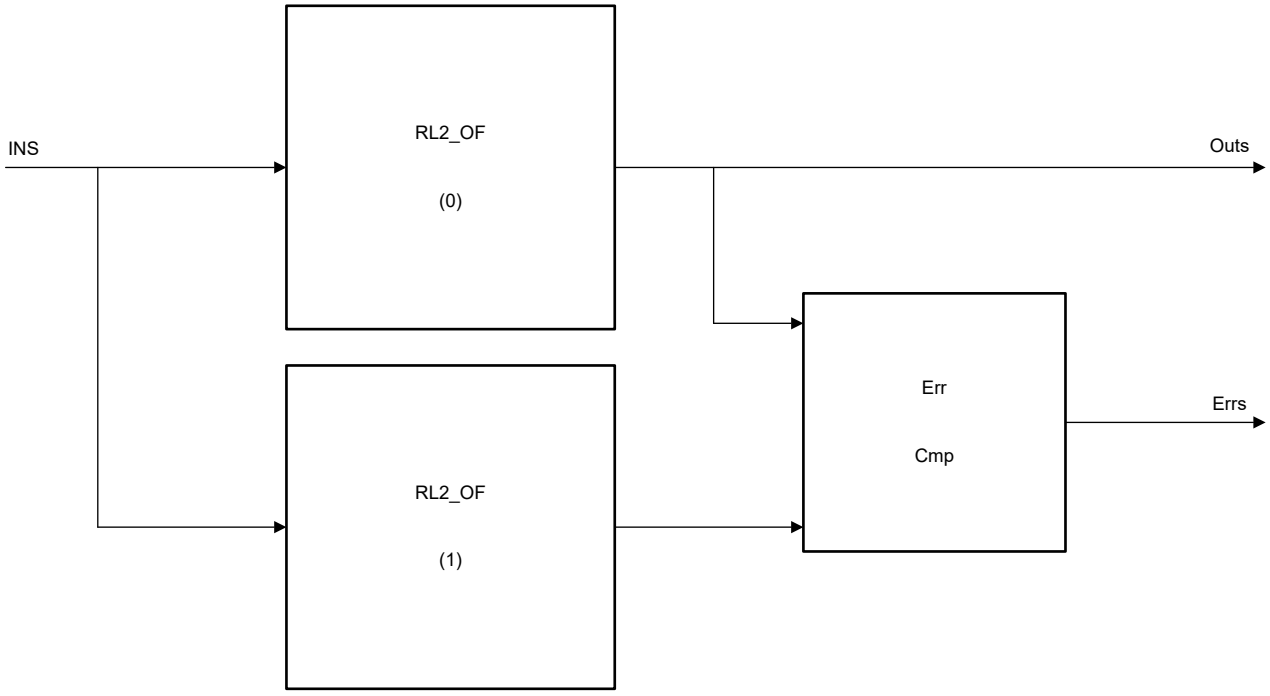


Figure 13-155. ASIL-D/SIL-3 Safety Implementation

13.3.2.1.3.10 Latencies

OptiFlash was built with the ability for commands passing through to have 0 latency.

Commands that use the TAG memory have a 3-cycle delay to accommodate ram latencies and any ECC error correction from the RAM.

Commands are stalled when a write to RAM is done either through cache miss return or FLC copy is active.

13.3.2.1.3.11 Critical Word First (CWF)

Critical Word First is a CPU cache concept which states that although the CPU wishes to fetch a particular cache line, the CPU prefers to see an offset into the cache line prior to seeing the beginning of the cache line. This is also known as wrapping burst, where a burst request to a cache line is not aligned to the start of the cache line. For example, if the CPU jumps to the last word of a cache line not currently in the L1, the CPU requests a cache line burst, but the starting address of the burst is the last bus aligned word address. Say the cache line size is 32 bytes, and the bus width is 64 bits wide. If the CPU jumps to an address of 0x1c offset into a particular non-allocated cache line, the CPU requests address 0x18 in a wrapping burst of 32 bytes, resulting in

offset addresses 0x18, 0x00, 0x08, 0x10 being the burst order requested. This is known as critical word first or wrapping burst request. That is the offset 0x18 is the critical word the CPU want to see in the cache line.

13.3.2.2 Fast Local Copy (FLC)

13.3.2.2.1 Overview FLC

The FLC allows a region of the slow memory like Flash to be dynamically copied to a section of system SRAM and at the same time run the code that was specified to be copied. The FLC redirects the request to the internal SRAM on the fly if the data has been transferred to the SRAM. Otherwise the request is sent to the slow device. This allow the SW to specify a piece of code to be resident in the system but executed out of the flash using internal system SRAM. The FLC ranges can start and end on any 4K Byte boundary. Care must be taken as to not create overlapping ranges. The minimum size of a FLC region is 4K Bytes.

The FLC does not modify any request for the CPU that is within a FLC range. This allows the critical word first (CWF) to be processed by the downstream module for best performance. A downstream CBA3-to-CBA4 bridge is suggested if it's required for a downstream CBA4 fabric be able to split any request that is a wrapping burst that is not aligned to the burst size boundary and targeting system SRAM.

Since the FLC is expected to copy data from a slow peripheral like a Flash device, the request is aligned to the burst boundary and the FLC restores the critical word first returned data to the CPU. FLC does not support split burst returns, all requested burst must be returned as a minimum of 8 byte burst return.

All FLC request are 32-byte or less, either wrapping burst or linear, but never cross a 32-byte aligned address. Any FLC request that crosses a 32-byte boundary is returned to the original address.

Since the copy of FLC range is independent from RAT ranges, the FLC copy function can be used to transfer the Flash data to the RAT target address.However, the FLC range access the target if it's already copied. So once copied, the FLC range must be disabled.

13.3.2.2.1.1 Supported Features

FLC Features

- Supports 4 ranges that can be copied to a remote internal SRAM
- Ranges can start and finsih on any 4KB boundary

13.3.2.3 Region based Address Translation (RAT)

13.3.2.3.1 Overview RAT

The RAT - Region based Address Translation allows segments of the memory map to be relocated to a different address in the system. The RAT only translates the address of a request and does not do anything to the request size or data. The RAT can translate region sizes of 4KB to 4G. The minimum size of a RAT region is 4KB, this eliminates any transaction splitting artifacts. Any enabled RAT region won't be translated or processed by (FLC) or (RL2) ranges. That is, RAT region address translation takes priority over FLC and RL2 address translation. Overlapping RAT ranges are not supported.

13.3.2.3.1.1 Supported Features

RAT Features

- Supports 4 regions to be translated
  - Each region can be configured for 4K to 4G bytes

13.3.3 Flash Subsystem (FSS)

This section describes the Flash Subsystem (FSS) in the device.

13.3.3.1 FSS Overview

The Flash Subsystem (FSS) provides access to external Flash devices via Octal Serial Peripheral Interface (OSPI).

The FSS includes one OSPI. For more information, see *Octal Serial Peripheral Interface (OSPI)*.

Table 13-187 shows FSS allocation across device domains.

Table 13-187. FSS Allocation Across Device Domains

Instance	Domain	
	MCU	MAIN
FSS0	-	✓

Figure 13-156 shows the FSS overview.

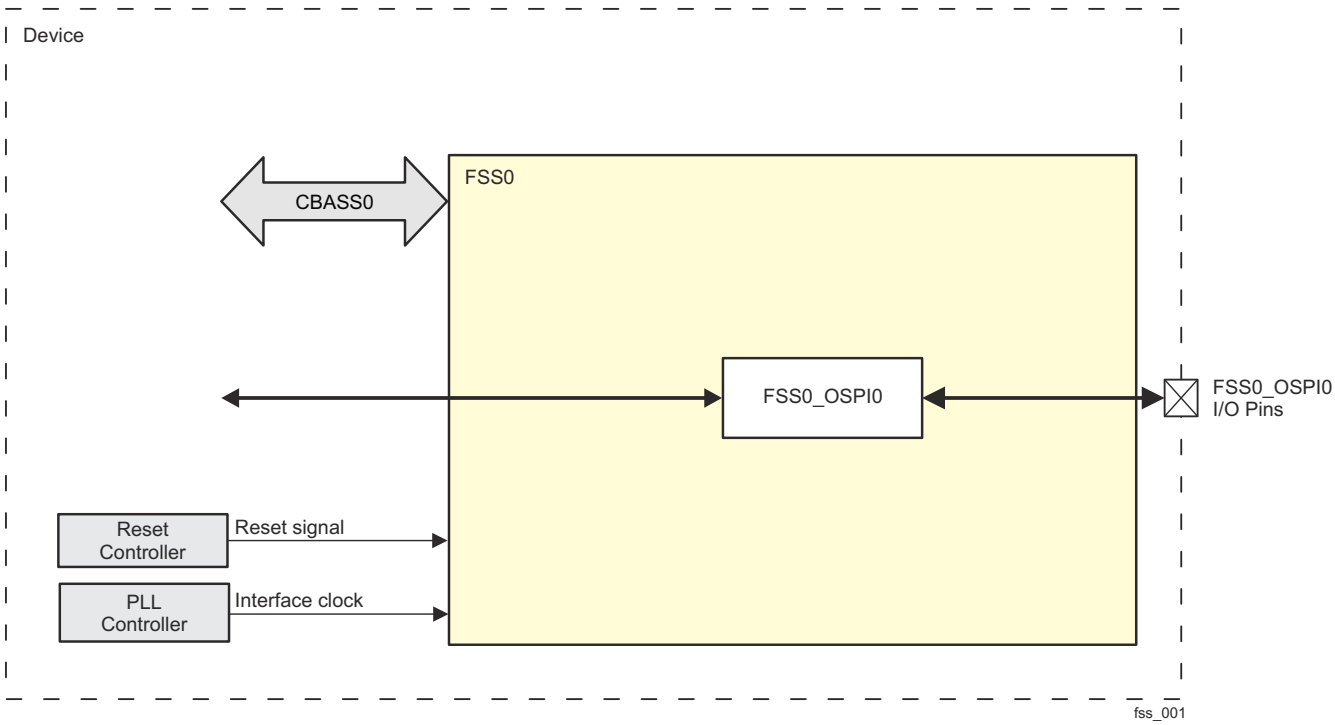


Figure 13-156. FSS Overview

13.3.3.1.1 FSS Features

For more information, see Section 13.3.3.6.1.1, *OSPI Features*.

13.3.3.1.2 FSS Not Supported Features

For more information, see Section 13.3.3.6.1.2, *OSPI Not Supported Features*.

13.3.3.2 FSS Environment

The FSS is hereinafter also referred to as FSS0.

13.3.3.2.1 FSS Typical Application

Figure 13-157 shows typical FSS application.

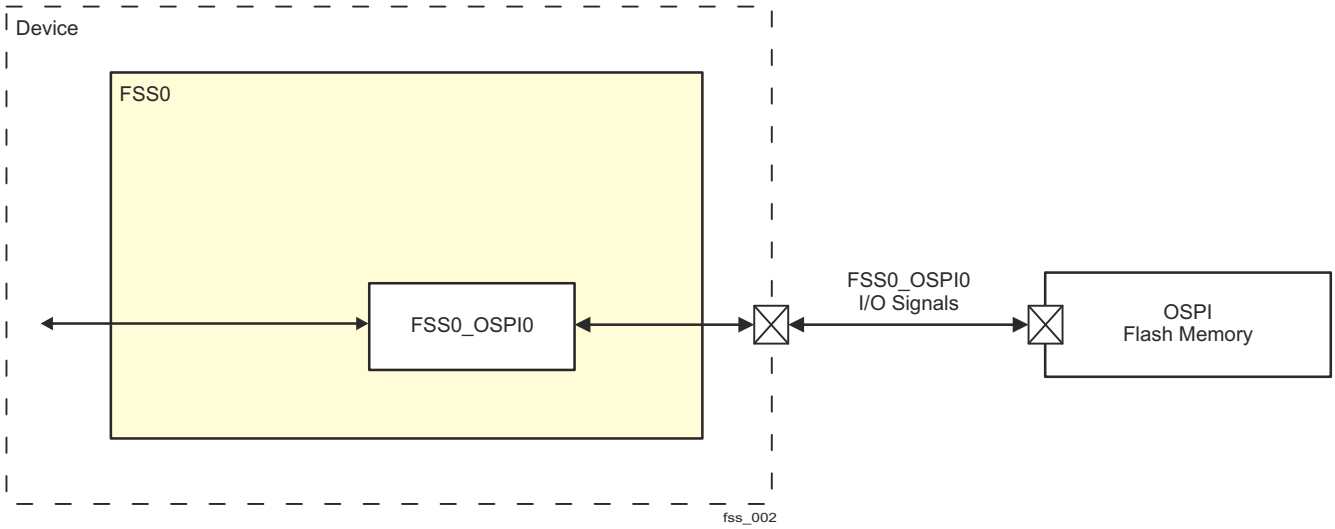


Figure 13-157. FSS Typical Application

Table 13-188 describes the FSS I/O signals.

Table 13-188. FSS I/O Signals

FSS0 Interface	I/O Signals
FSS0_OSPI0	For more information about OSPI I/O signals, see Table 13-194.

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.

13.3.3.3 FSS Integration

This section describes the FSS integration in the device, including information about clocks, resets, and hardware requests.

13.3.3.3.1 FSS Integration in MAIN Domain

There is one FSS integrated in the device MAIN domain - FSS0. Figure 13-158 shows the integration of FSS0.

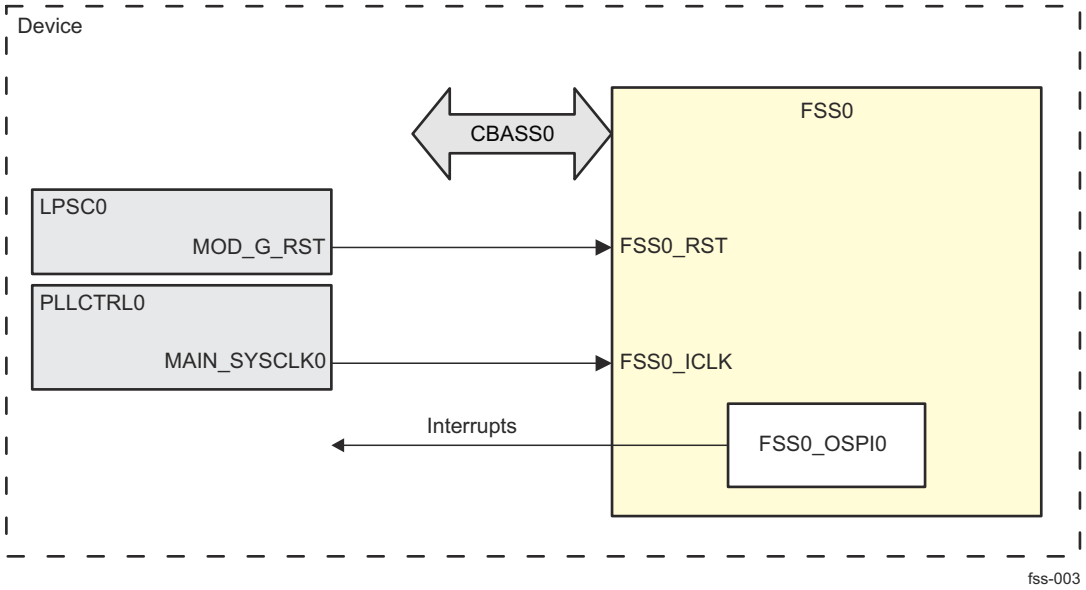


Figure 13-158. FSS0 Integration

Table 13-189 through Table 13-192 summarize the integration of FSS0 in the device MAIN domain.

Table 13-189. FSS0 Integration Attributes

Module Instance	Power Sleep Controller	Power Domain	Module Domain	Interconnect
FSS0	PSC0	PD0	LPSC0	CBASS0

Table 13-190. FSS0 Clocks

Module Instance	Module Clock Input	Source Clock Signal	Source	Description
FSS0	FSS0_ICLK	MAIN_SYSCLK0	PLLCTRL0	FSS0 interface clock

Table 13-191. FSS0 Resets

Module Instance	Module Reset Input	Source Reset Signal	Source	Description
FSS0	FSS0_RST	MOD_G_RST	LPSC0	FSS0 system reset

Table 13-192. FSS0 Hardware Requests

Module Instance	Module Interrupt Signal	Destination Interrupt Input	Destination	Description	Type
For more information, see Table 4-43.					

Note

For more information on the OSPI Integration, see Section 4.18.

Note

For more information on the interconnects, see Section 3, *System Interconnect*.

For more information on the power, reset and clock management, see the corresponding sections within Section 6, *Device Configuration*.

For more information on the device interrupt controllers, see Section 10.2, *Interrupt Controllers*.

13.3.3.4 FSS Functional Description

13.3.3.4.1 FSS Block Diagram

Figure 13-159 shows the FSS block diagram.

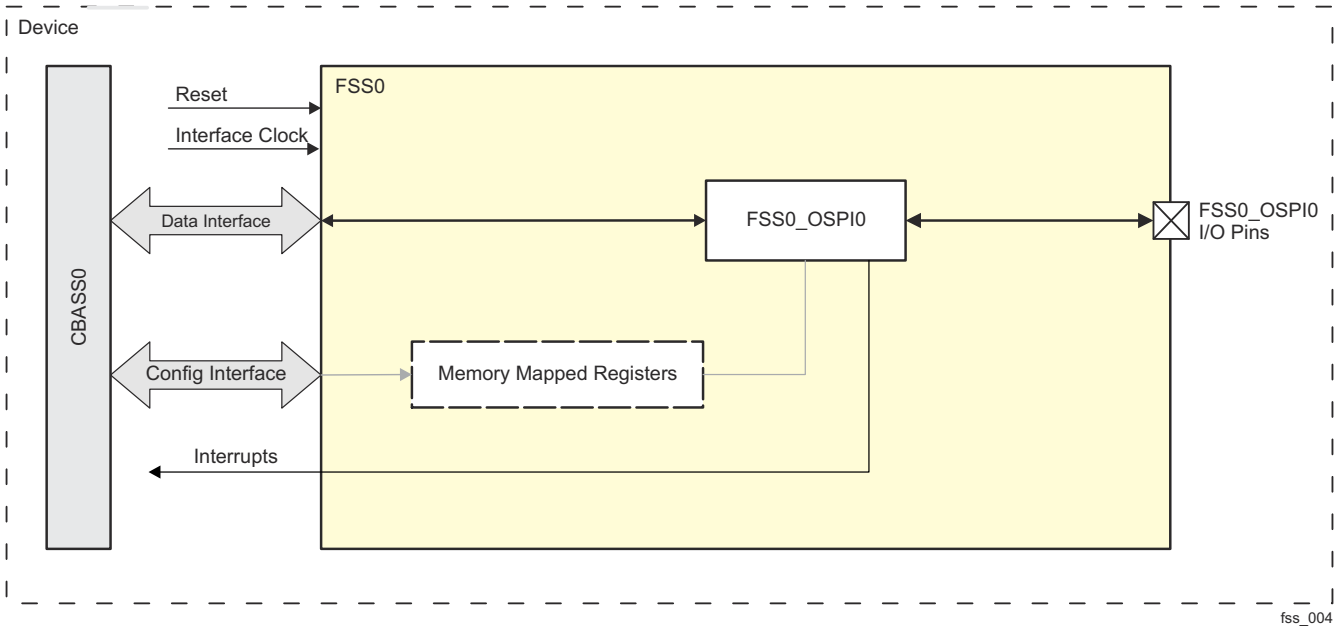


Figure 13-159. FSS Block Diagram

FSS Blocks:

- CBASS0: The CBASS0 interconnect allows FSS to communicate with the device modules and subsystems.
- Data Interface: It is 64-bit data/32-bit address multi issue data interface with coherent in-band bypass. It provides accessibility to the FSS0_OSPI0.
- Config Interface: It is used for configuration of the memory mapped registers within the FSS.
- Interface Clock and Reset:
  - For more information, see *FSS0 Clocks and Resets*.
  - For more information, see *FSS0_OSPI Clocks and Resets*.
- Interrupts: For more information, see *FSS0_OSPI Hardware Requests*.
- Memory Mapped Registers: This block includes the FSS registers. The configuration of these registers defines which FSS features are used. For more information, see *FSS Registers*.
- FSS0_OSPI0: For more information about OSPI, please see *Octal Serial Peripheral Interface (OSPI)*.
- FSS0_OSPI0 I/O Pins: For more information, see OSPI I/O Signals.

13.3.3.4.2 FSS Regions

13.3.3.4.2.1 FSS Regions Boot Size Configuration

The boot size for FSS defaults to 64 MB but can be configured to be 128 MB. Selection of boot block which will be used is also configurable. For more information see CTRLMMR_FSS_CTRL register in *Control Module (CTRL_MMR)*.

13.3.3.4.3 FSS Memory Regions

Table 13-193 shows the FSS memory regions.

Table 13-193. FSS Memory Regions

Address Range	Size	Description
0x400000000 - 0x4FFFFFFFF	4 GB	External Memory Space (Region 0)
0x060000000 - 0x067FFFFFF	128 MB	Boot Space (Region 1)
0x500000000 - 0x5FFFFFFFF	4 GB	External Memory Space (Region 3)

13.3.3.5 FSS Programming Guide

13.3.3.5.1 FSS Initialization Sequence

Initialization steps:

- Configure the main boot parameters for FSS (see Section 13.3.3.4.2.1):
  - Select the boot block to be used.
  - Select the size of the boot block to be used.
- Enable FSS in PSC.
- Configure the FSS0_OSPI0.
- Enable the FSS0_OSPI0 in PSC. For more information about OSPI configuration, please see Section 13.3.3.6, *Octal Serial Peripheral Interface (OSPI)*.

13.3.3.5.2 FSS Power Up/Down Sequence

There are two PSC controls for the FSS: for FSS0 itself and for FSS0_OSPI0. The CPU enables the FSS0_OSPI0 before using the FSS. Software should ensure the FSS0_OSPI0 is enabled prior to FSS transactions.

Normal Power Down Sequence:

- Block any new transaction.
- Power down the FSS0_OSPI0.
- In the event when the FSS0_OSPI0 is powered down, the FSS can then be powered down.

13.3.3.6 Octal Serial Peripheral Interface (OSPI)

This section describes the Octal Serial Peripheral Interface (OSPI) module for the device.

13.3.3.6.1 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.

shows the OSPI allocation across device domains.

Figure 13-160 shows the OSPI module overview.

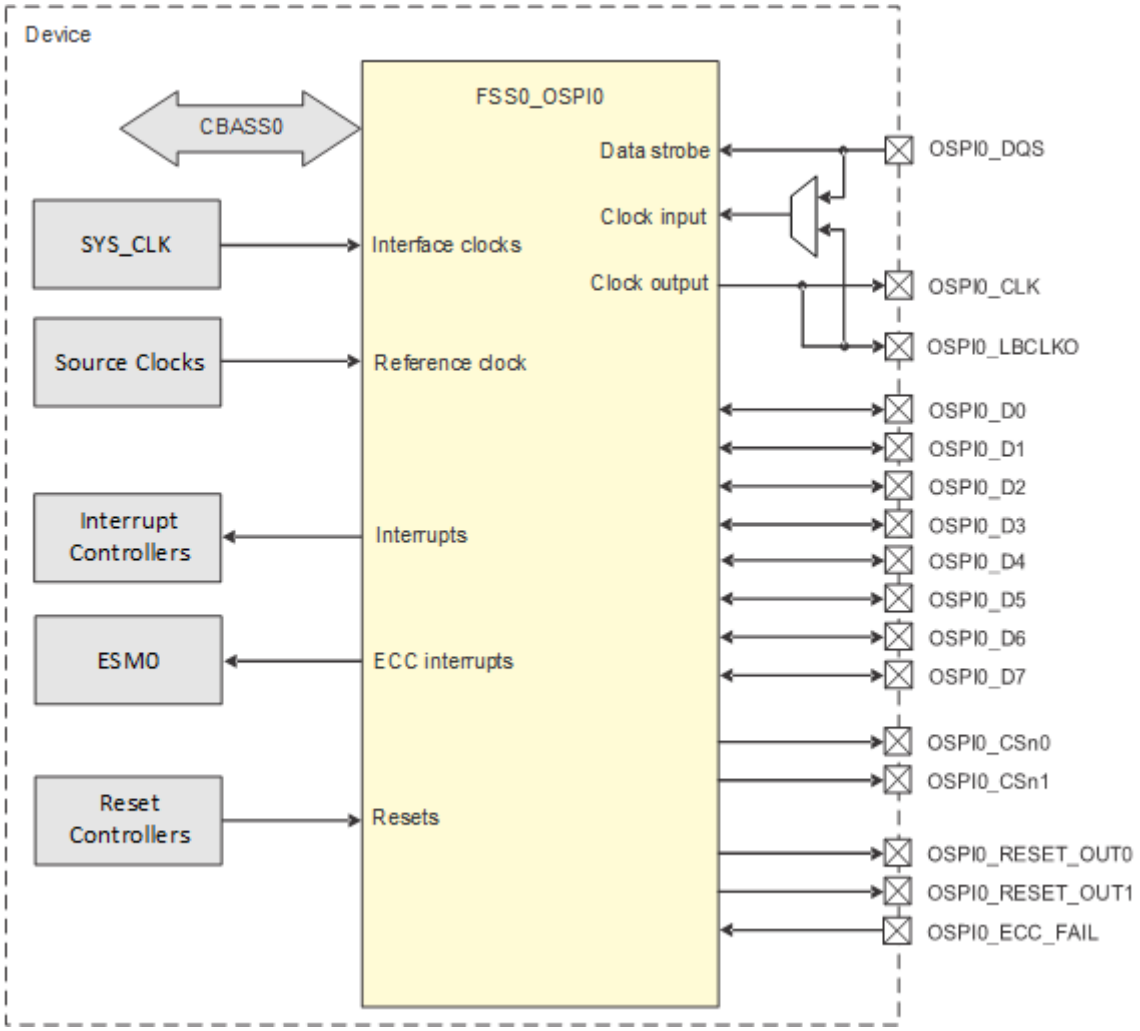


Figure 13-160. OSPI Overview

13.3.3.6.1.1 OSPI Features

The OSPI module has the following features:

- Support for single, dual, quad (QSPI mode) or octal I/O instructions.
- Supports 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 (AHB) 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
- Programmable data decoder, enables continuous addressing mode for each of the connected devices and auto-detection of boundaries between devices.
- 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.
- 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)

13.3.3.6.1.2 OSPI Not Supported Features

The following features are not supported on this family of devices:

- DMA not supported in INDAC mode
- Pulse events not used.
- In Octal-SPI and Quad-SPI mode, Mode 1, 2, and 3 are not supported.

13.3.3.6.2 OSPI Environment

The FSS0_OSPI0 module is hereinafter referred to as OSPI module.

This section describes the OSPI external connections (environment).

The OSPI module is primarily intended for fast booting from Octal- and Quad-SPI flash memories. Figure 13-161 shows a typical connection of the OSPI module to an external Octal-SPI flash memory.

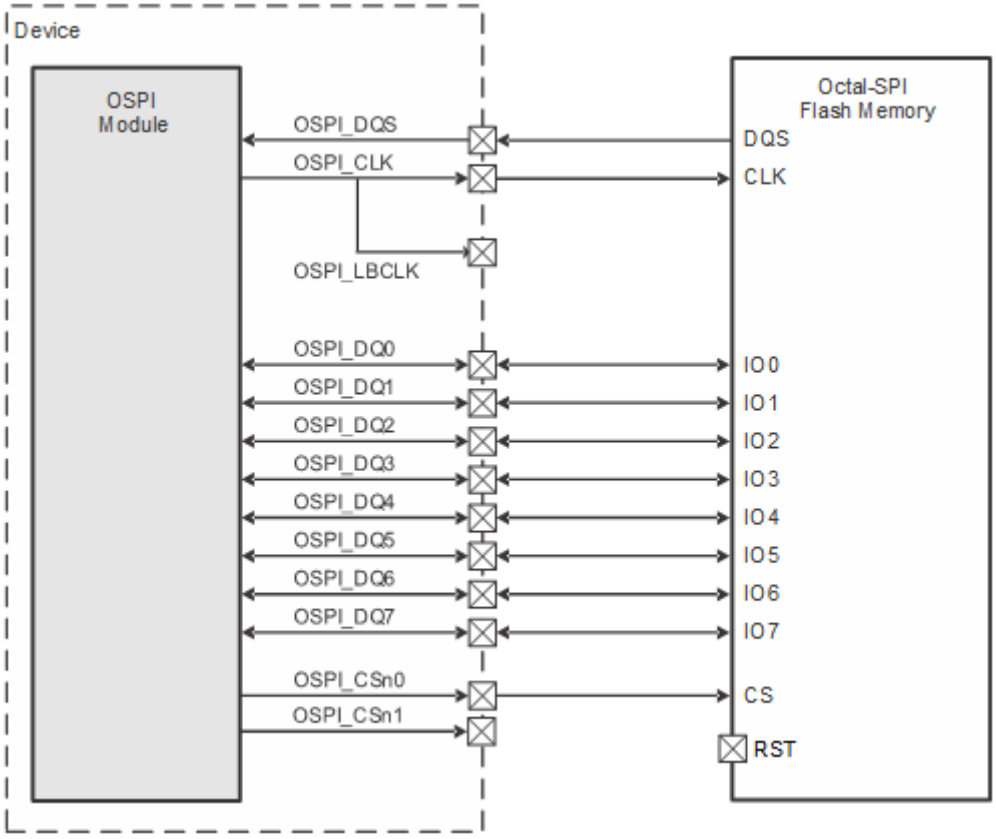


Figure 13-161. OSPI Connected to an External Octal-SPI Flash Memory

Table 13-194 lists and describes the FSS0_OSPI I/O signals.

Table 13-194. OSPI I/O Signals

Module Pin	Device Level Signal	I/O[1]	Description	Module Pin Reset Value[2]
			FSS0_OSPI0	
DQ0	OSPI0_D0	IO	FSS0_OSPI0 data input/output 0	HiZ
DQ1	OSPI0_D1	IO	FSS0_OSPI0 data input/output 1	HiZ
DQ2	OSPI0_D2	IO	FSS0_OSPI0 data input/output 2	HiZ
DQ3	OSPI0_D3	IO	FSS0_OSPI0 data input/output 3	HiZ
DQ4	OSPI0_D4	IO	FSS0_OSPI0 data input/output 4	HiZ
DQ5	OSPI0_D5	IO	FSS0_OSPI0 data input/output 5	HiZ
DQ6	OSPI0_D6	IO	FSS0_OSPI0 data input/output 6	HiZ
DQ7	OSPI0_D7	IO	FSS0_OSPI0 data input/output 7	HiZ
N_SS_OUT0	OSPI0_CSn0	O	FSS0_OSPI0 external flash device chip select 0	0x1
N_SS_OUT1	OSPI0_CSn1	O	FSS0_OSPI0 external flash device chip select 1	0x1
OCLK	OSPI0_CLK	O	FSS0_OSPI0 clock output for the external flash device	0x0
	OSPI0_LBCLKO	O	FSS0_OSPI0 external loopback output	0x0
DQS	OSPI0_DQS	I[3]	FSS0_OSPI0 data strobe / external loopback input	Don't care

Table 13-194. OSPI I/O Signals (continued)

Module Pin	Device Level Signal	I/O[1]	Description	Module Pin Reset Value[2]
RESET_OUT0	OSPI0_RESET_OUT0	O	FSS0_OSPI0 reset output 0 for the external flash device. Pin is active low.	0x1
RESET_OUT1	OSPI0_RESET_OUT1	O	FSS0_OSPI0 reset output 1 for the external flash device. Pin is active low.	0x1
ECC_FAIL	OSPI0_ECC_FAIL	I	FSS0_OSPI0 ECC status from the external flash device	0x1

(1) I = Input; O = Output
(2) HiZ = High Impedance
(3) 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 MSS_OSPI_CONFIG[6:4] MSS_OSPI_CONFIG_ICLK_SEL bits in *MSS_CTRL*.

Table 13-195 describes the OSPI I/O connectivity to external SPI devices.

Table 13-195. OSPI I/O Connectivity to External SPI Devices

Module Pin	I/O[1]	Description			
		4-pin[1] SPI - Single Read/Write (SIO) (DATA_XFER_TYPE_EXT_MODE_FLD=0x0)	4-pin[1] SPI - Dual Read/ Write (DATA_XFER_TYPE_EXT_MODE_FLD=0x1)	6-pin[1] SPI - Quad Read/ Write (DATA_XFER_TYPE_EXT_MODE_FLD=0x2)	11-pin[1] SPI - Octal Read/ Write (DATA_XFER_TYPE_EXT_MODE_FLD=0x3)
DQ0	IO	Used as SPI data output	Used as SPI data input 0 Used as SPI data output 0	Used as SPI data input 0 Used as SPI data output 0	Used as SPI data input 0 Used as SPI data output 0
DQ1	IO	Used as SPI data input	Used as SPI data input 1 Used as SPI data output 1	Used as SPI data input 1 Used as SPI data output 1	Used as SPI data input 1 Used as SPI data output 1
DQ2	IO	Not used	Not used	Used as SPI data input 2 Used as SPI data output 2	Used as SPI data input 2 Used as SPI data output 2
DQ3	IO	Not used	Not used	Used as SPI data input 3 Used as SPI data output 3	Used as SPI data input 3 Used as SPI data output 3
DQ4	IO	Not used	Not used	Not used	Used as SPI data input 4 Used as SPI data output 4
DQ5	IO	Not used	Not used	Not used	Used as SPI data input 5 Used as SPI data output 5
DQ6	IO	Not used	Not used	Not used	Used as SPI data input 6 Used as SPI data output 6
DQ7	IO	Not used	Not used	Not used	Used as SPI data input 7 Used as SPI data output 7
DQS	I	Not used	Not used	Not used	Data strobe or loopback clock input
OCLK	O	Output clock or loopback clock output. For more information, see Table 13-194.			
N_SS_OUT0	O	External SPI device chip-select 0			
N_SS_OUT1	O	External SPI device chip-select 1			
RESET_OUT0	O	External SPI device reset 0. Pin is active low.			
RESET_OUT1	O	External SPI device reset 1. Pin is active low.			
ECC_FAIL	I	External SPI device ECC failure indication			

(1) This is the pin count at the external SPI flash memory side.

Note

For OSPI0_CLK, OSPI0_LBCLKO, and OSPI0_DQS signals to work properly, the RXACTIVE bit of the appropriate registers should be set to 0x1 because of retiming purposes.

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.

13.3.3.6.3 OSPI Integration

This section describes module integration in the device, including information about clocks, resets, and hardware requests.

13.3.3.6.3.1 OSPI Integration

There is 1x OSPI module integrated in the device. The diagram below provides a visual representation of the device integration details.

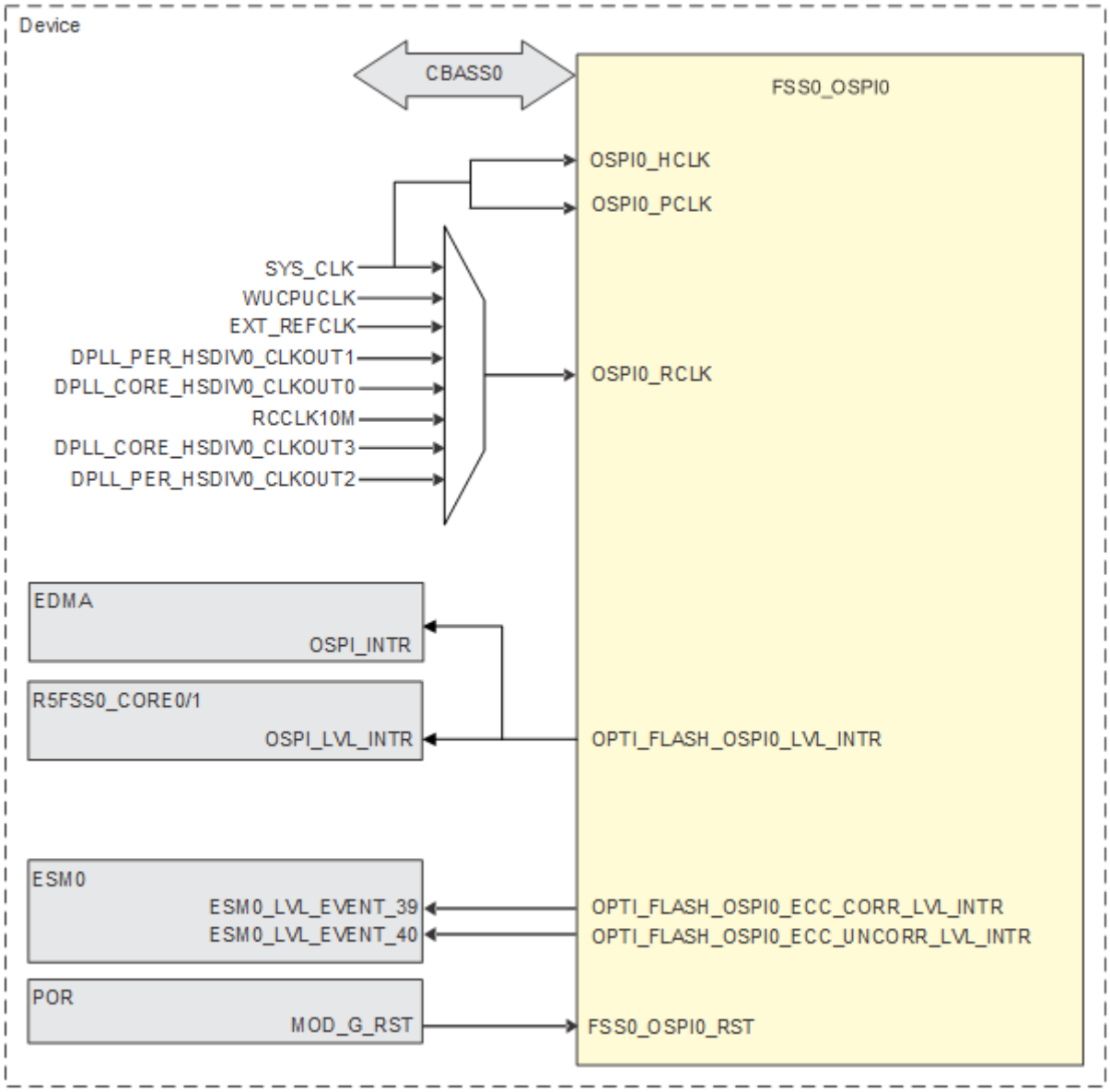


Figure 13-162. OSPI Integration Diagram

The tables below summarize the device integration details of OSPI.

Table 13-196. *OSPI* Device Integration

This table describes the module device integration details.

Module Instance	Device Allocation	SoC Interconnect
OSPI0	✓	CORE VBUSM Interconnect

Table 13-197. FSS0_OSPI Clocks

Module Instance	Module Clock Input	Source Clock Signal	Source	Description
FSS0_OSPI0	OSPI0_HCLK	SYS_CLK	SYS_CLK	FSS0_OSPI0 data transfer clock
	OSPI0_PCLK	SYS_CLK	SYS_CLK	FSS0_OSPI0 configuration clock

Table 13-197. FSS0_OSPI Clocks (continued)

Module Instance	Module Clock Input	Source Clock Signal	Source	Description
	OSPI0_RCLK	OSPI_CLK	WUCPUCLK	FSS0_OSPI0 Reference clock. Mux controlled by *MSS_RCM:OSPI0_CLK_SRC_SEL*
			EXT_REFCLK	
			SYS_CLK	
			DPLL_PER_HSDIV0_CLKOUT1	
			DPLL_CORE_HSDIV0_CLKOUT0	
			RCCLK10M	
			DPLL_CORE_HSDIV0_CLKOUT3	
			DPLL_PER_HSDIV0_CLKOUT2	

Table 13-198. FSS0_OSPI Resets

Module Instance	Module Reset Input	Source Reset Signal	Source	Description
FSS0_OSPI0	FSS0_OSPI0_RST	MOD_G_RST	POR	FSS0_OSPI0 reset

Table 13-199. FSS0_OSPI Interrupt Requests

Module Instance	Module Interrupt Signal	Destination Interrupt Input	Destination	Description	Type
FSS0_OSPI0	OPTI_FLASH_OSPI0_LVL_INTR	OSPI0_LVL_INTR	All R5FSS Cores ICSSM Core	FSS0_OSPI0 interrupt	Level
	OPTI_FLASH_OSPI0_ECC_CORR_LVL_INTR	ESM0_LVL_EVENT_39	OPTI_FLASH	FSS0_OSPI0 ECC Aggregator correctable error interrupt	Level
	OPTI_FLASH_OSPI0_ECC_UNCORR_LVL_INTR	ESM0_LVL_EVENT_40	OPTI_FLASH	FSS0_OSPI0 ECC Aggregator uncorrectable error interrupt	Level

Table 13-200. *OSPI* DMA Requests

This table describes the module DMA requests.

Module Instance	Module DMA Event	Destination DMA Event Input	Destination	Type	Description
OSPI0	OPTI_FLASH	OSPI_INTR	OPTI_FLASH_OSPI0_LVL_INTR	Pulse	OSPI0 DMA Event Request

Note

For more information on the interconnects, see the *System Interconnect* chapter.

For more information on power, reset, and clock management, see the corresponding sections within the *Device Configuration* chapter.

For more information on the device interrupt controllers, see the *Interrupt Controllers* chapter.

13.3.3.6.4 OSPI Functional Description

13.3.3.6.4.1 OSPI Block Diagram

Figure 13-163 shows the OSPI module block diagram.

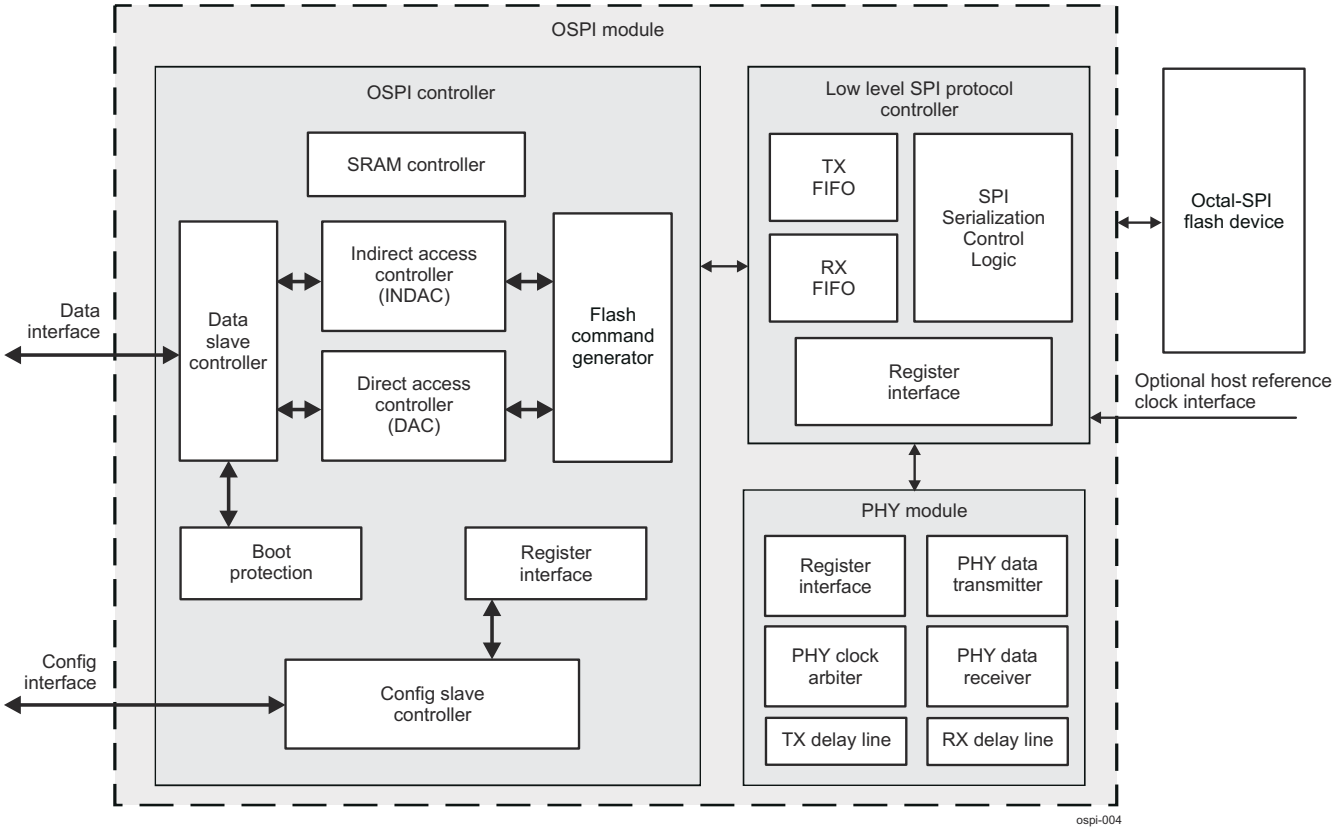


Figure 13-163. OSPI Block Diagram

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.

13.3.3.6.4.1.1 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.

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.

13.3.3.6.4.1.2 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 13.3.3.6.4.11, *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.

13.3.3.6.4.1.3 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.

13.3.3.6.4.2 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 13-201 gives a brief description of these modes.

Table 13-201. 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

Table 13-201. 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 Section 13.3.3.6.4.2.1, *Read Data Capture*).

13.3.3.6.4.2.1 Read Data Capture

Figure 13-164 shows the Read Data Capture Logic in the OSPI module.

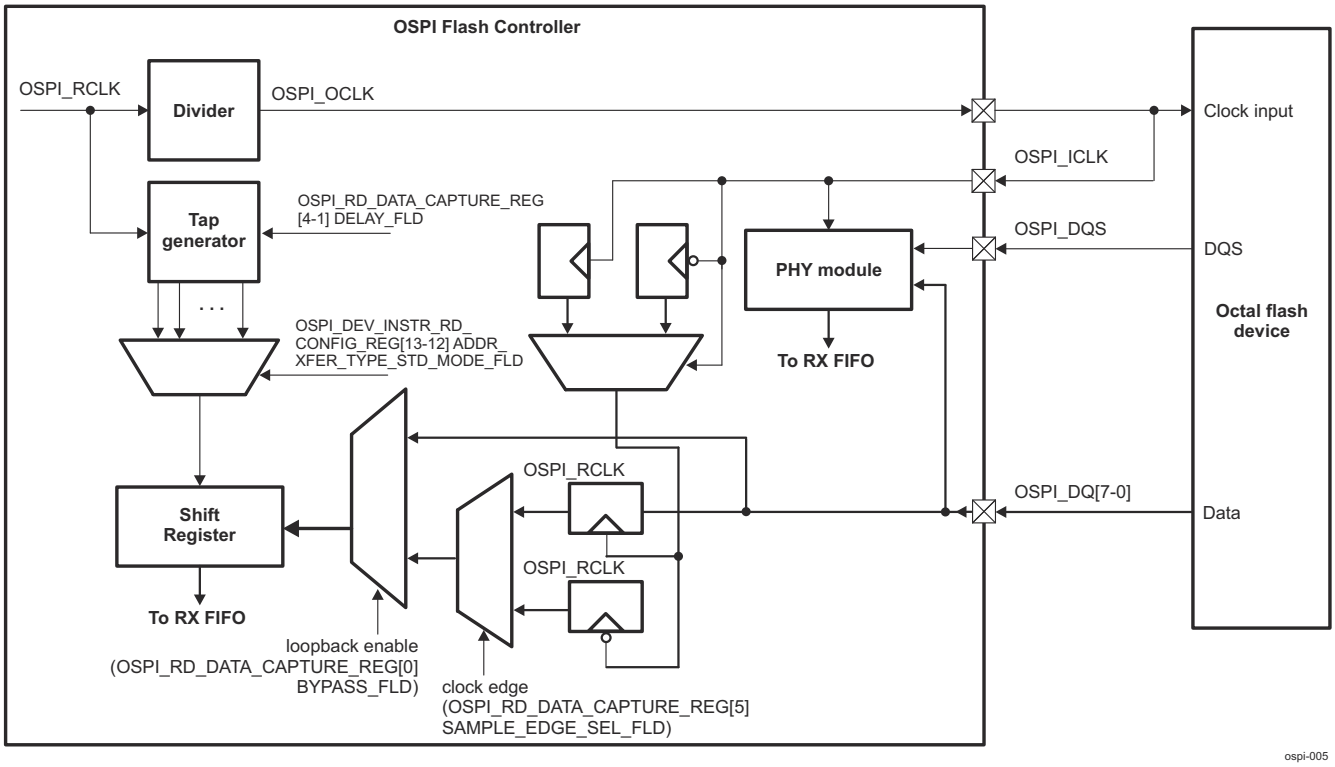


Figure 13-164. Read Data Capture Logic

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

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.

13.3.3.6.4.2.1.1 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.

13.3.3.6.4.2.1.2 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.

13.3.3.6.4.2.1.3 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 13.3.3.6.4.16.2, *Read Data Capturing by the PHY Module*.

13.3.3.6.4.2.1.4 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.

13.3.3.6.4.3 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.

OSPI_PCLK and OSPI_HCLK share the same clock stop request/acknowledge and clock enable/acknowledge interface.

13.3.3.6.4.4 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).

13.3.3.6.4.5 Flash Reset

OSPI provides Flash reset out ports. These ports are active low and controlled thru OSPI_CONFIG_REG register.

13.3.3.6.4.6 OSPI Memory Regions

OSPI Memory Map shows the OSPI memory map in domain.

Note

For more information about the memory space, see *FSS Memory Regions*.

13.3.3.6.4.7 OSPI Interrupt Requests

The OSPI module generates three interrupts. The ECC interrupts (FSS0_OSPI_0_OSPI_ECC_CORR_LVL_INTR_0 and FSS0_OSPI_0_OSPI_ECC_UNCORR_LVL_INTR_0) are generated by the OSPI ECC aggregator.

The other interrupt (FSS0_OSPI_0_OSPI_LVL_INTR_0) is generated by the OSPI module.

Table 13-202 lists the event flags and the corresponding mask bits of the sources which can cause interrupts.

Table 13-202. 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	

Table 13-202. 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	

13.3.3.6.4.8 OSPI Data Interface

13.3.3.6.4.8.1 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.

13.3.3.6.4.8.2 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.

13.3.3.6.4.8.3 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 13.3.3.6.4.10, *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.

13.3.3.6.4.9 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).

13.3.3.6.4.10 OSPI Indirect Access Controller (INDAC)

13.3.3.6.4.10.1 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 13.3.3.6.4.10.3, *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.

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.

13.3.3.6.4.10.1.1 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.

13.3.3.6.4.10.2 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 13.3.3.6.4.10.3, *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.

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.

13.3.3.6.4.10.2.1 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.

13.3.3.6.4.10.3 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.

13.3.3.6.4.10.4 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.

13.3.3.6.4.10.5 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.

- 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 13-203 shows priority allocated to these sources.

Table 13-203. 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.

13.3.3.6.4.11 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

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 16 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 16, 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).

13.3.3.6.4.11.1 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 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.

13.3.3.6.4.12 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

13.3.3.6.4.13 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.

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 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 13-204. 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

Table 13-204. Flash SPI Modes (continued)

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

13.3.3.6.4.14 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 13-205 shows how software should configure the OSPI module for selected specific READ and WRITE instruction supported by the abovementioned device.

Table 13-205. READ and WRITE Instruction Configuration

READ							
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

Table 13-205. READ and WRITE Instruction Configuration (continued)

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

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

Table 13-205. READ and WRITE Instruction Configuration (continued)

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.

13.3.3.6.4.15 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 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.

13.3.3.6.4.16 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 13.3.3.6.4.16.1, *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 13.3.3.6.5.2, *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 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 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 13.3.3.6.4.16.1, *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.

13.3.3.6.4.16.1 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.

13.3.3.6.4.16.2 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.

13.3.3.6.5 OSPI Programming Guide

13.3.3.6.5.1 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.

13.3.3.6.5.2 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 13.3.3.6.4.16.1, *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

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.

13.3.3.6.5.3 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.

13.3.3.6.5.4 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

to return valid data after many dummy cycles, the TX FIFO could become full before the controller sends all of dummy data.

13.3.3.6.5.5 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.

13.3.3.6.5.6 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.

13.3.3.6.5.7 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.

13.3.3.7 Firmware Upgrade Over the Air (FOTA)

This section describes the Firmware Upgrade Over the Air (FOTA) module for the device.

13.3.3.7.1 FOTA Overview

For applications like ADAS, Automotive Gateway, and Industry Automation, Firmware Over the Air (FOTA) updates are required to address bug fixes and security vulnerabilities. In order to meet the system cost, a single flash solution is required, which would require to pause the application execution until the update (firmware download) is completed. To reduce overall down-time of the system during a firmware update, FOTA allows the system to update new firmware/software image during concurrent system operation - reading from external flash (XIP).

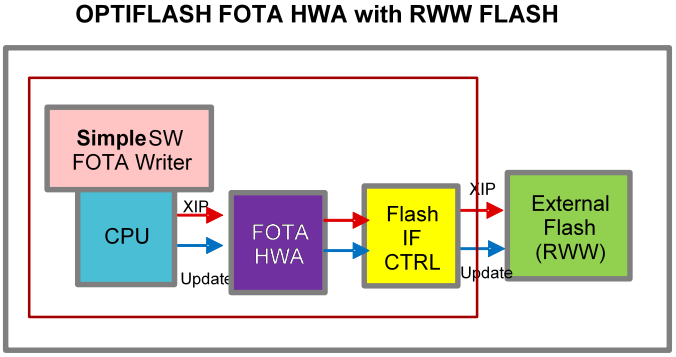


Figure 13-165. FOTA Block Diagram

Typical FOTA solutions address this problem by performing read while write in software. However, without any hardware support, it becomes complex as it requires complex synchronization across threads/CPUs, increasing the XIP downtime. With the OptiFlash FOTA Hardware Accelerator IP, as shown in Figure 13-165, it is possible to further reduce the XIP downtime and be able to perform concurrent XIP read(s) while FOTA update happens in the background, with zero software overheads on the MCU. Primarily, this is useful when using Read While Write (RWW) capable flash memory with dual/multiple banks, which allows reads while write/erase is in progress (which can take >1ms to complete) in a different bank.

13.3.3.7.2 FOTA Features

13.3.3.8 On the Fly Encryption and Authentication (OTFA)

This section describes the On the Fly Encryption and Authentication (OTFA) module for the device.

13.3.3.8.1 OTFA Overview

OptiFlash provides a Security Engine with in-line encryption/decryption/Authentication (AES/GCM) on flash data to enable secure external flash use. It supports on-the-fly address translation to provide SW transparent view to account for additional storage of MAC (Message Authentication Code). The MAC size is programmable.

13.3.3.8.2 OTFA Features

13.3.3.9 Error Correction Code (ECC) and Safety

This section describes the Error Correction Code (ECC) module for the device.

13.3.3.9.1 ECC Overview

OptiFlash provides a Safety Engine with in-line ECC insertion during write and checking/correction during read to provide safety and reliability with external flash. It supports on-the-fly address translation to provide SW transparent view to account for additional storage of ECC data bytes.

13.3.3.9.2 ECC Features

13.4 Industrial and Control Interfaces

This section describes the industrial and control interfaces in the device.

13.4.1 Modular Controller Area Network (MCAN)

This section describes the Modular Controller Area Network (MCAN) modules in the device.

13.4.1.1 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.

The device supports 8 MCAN modules:

They connect to the physical layer of the CAN network through external (for the device) transceivers. Each MCAN module supports flexible bit rates and is compliant to ISO 11898-1:2015.

13.4.1.1.1 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

13.4.1.1.2 MCAN Not Supported Features

- Host Bus Firewall
- GPIO Mode
- Clock Calibration
- External (IO) Loopback Mode
- Debug DMA (see Section 13.4.1.4.7.4)
- TX DMA Channels

13.4.1.2 MCAN Environment

This section describes the MCAN external connections (environment).

CAN network physical layer consists of two-wire differential bus, usually twisted pair, and provides high level of interference immunity. External CAN transceiver IC is needed to access a CAN bus by the MCAN.

Figure 13-166 shows the MCAN typical application.

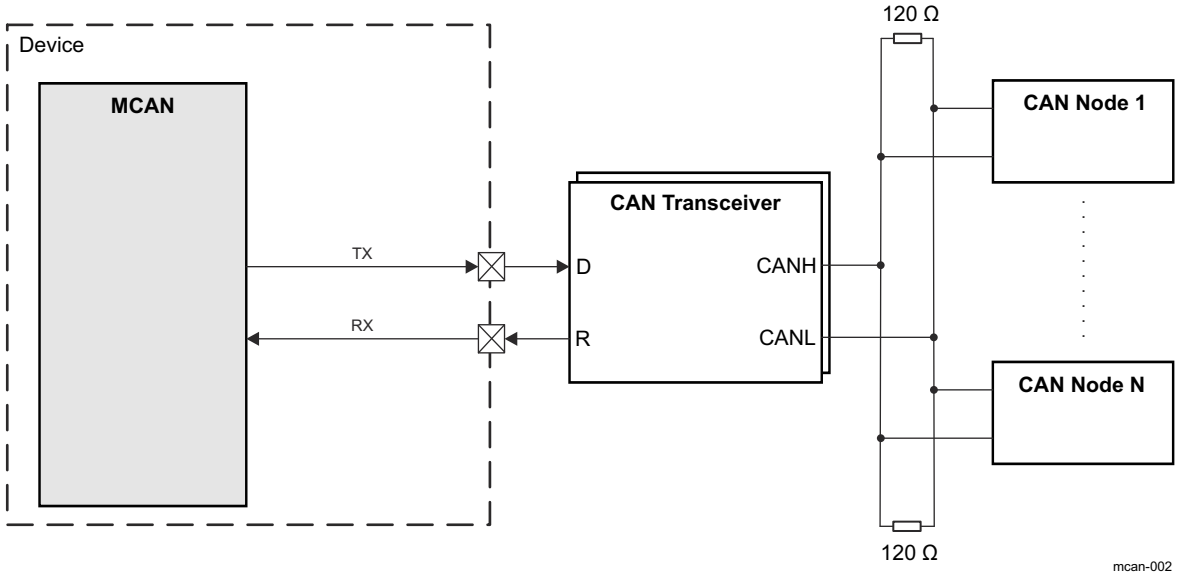


Figure 13-166. MCAN Typical Application

Table 13-206 describes the MCAN I/O signals.

Table 13-206. MCAN I/O Signals

Module Pin	Device Level Signal	I/O[1]	Description	Module Pin Reset Value[1]
RX	MCAN0_RX	I	Serial data input from external CAN transceiver	HiZ
TX	MCAN0_TX	O	Serial data output to external CAN transceiver	1
RX	MCAN1_RX	I	Serial data input from external CAN transceiver	HiZ
TX	MCAN1_TX	O	Serial data output to external CAN transceiver	1
RX	MCAN2_RX	I	Serial data input from external CAN transceiver	HiZ
TX	MCAN2_TX	O	Serial data output to external CAN transceiver	1
RX	MCAN3_RX	I	Serial data input from external CAN transceiver	HiZ
TX	MCAN3_TX	O	Serial data output to external CAN transceiver	1
RX	MCAN4_RX	I	Serial data input from external CAN transceiver	HiZ
TX	MCAN4_TX	O	Serial data output to external CAN transceiver	1
RX	MCAN5_RX	I	Serial data input from external CAN transceiver	HiZ
TX	MCAN5_TX	O	Serial data output to external CAN transceiver	1
RX	MCAN6_RX	I	Serial data input from external CAN transceiver	HiZ
TX	MCAN6_TX	O	Serial data output to external CAN transceiver	1
RX	MCAN7_RX	I	Serial data input from external CAN transceiver	HiZ
TX	MCAN7_TX	O	Serial data output to external CAN transceiver	1

(1) I = Input; O = Output; HiZ = High Impedance

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.

13.4.1.2.1 CAN Network Basics

- CAN bus is a 2-wire differential bus using Non-Return-to-Zero (NRZ) encoding and has two states:
  - Recessive state (logical 1)
  - Dominant state (logical 0)
- The network is multicontroller. When two or more nodes (ECUs) attempt to transmit at the same time, a non-destructive arbitration technique guarantees messages are sent in order of priority and no messages are lost.
- The message transmission is multicast. Data messages transmitted are identifier based, not address based.
- Content of message is labeled by the identifier that is unique throughout the network (for example: rpm, temperature, position, pressure, and so forth).
- All nodes on network receive the message and each performs an acceptance test on the identifier. If message is relevant, it is processed, otherwise it is ignored.
- The unique identifier also determines the priority of the message (the lower the numerical value of the identifier, the higher the priority is).
- Data is transmitted and received using message frames, consisting of the following basic fields:
  - Arbitration field
  - Control field
  - Data field (up to 8 bytes for Classical CAN and up to 64 bytes for CAN FD)
  - CRC field
  - ACK field

For more information, see *ISO 11898-1:2015: CAN data link layer and physical signaling*.

13.4.1.3 MCAN Integration

There are 8x MCAN modules integrated in the device. The diagram below provides a visual representation of the device integration details.

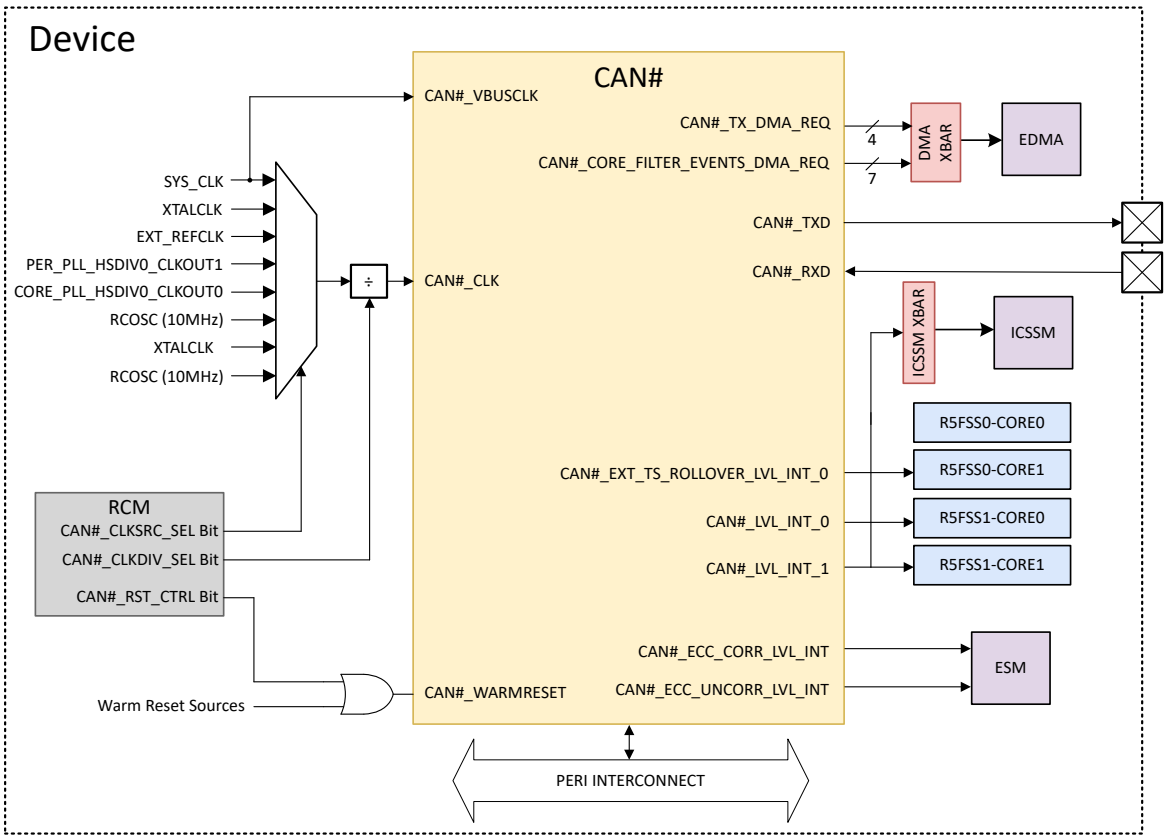


Figure 13-167. MCAN Integration Diagram

The tables below summarize the device integration details of MCAN# (where # = 0 to 7).

Table 13-207. *MCAN* Device Integration

This table describes the module device integration details.

Module Instance	Device Allocation	SoC Interconnect
MCAN0	✓	Peripheral VBUSP Interconnect
MCAN1	✓	Peripheral VBUSP Interconnect
MCAN2	✓	Peripheral VBUSP Interconnect
MCAN3	✓	Peripheral VBUSP Interconnect
MCAN4	✓	Peripheral VBUSP Interconnect
MCAN5	✓	Peripheral VBUSP Interconnect
MCAN6	✓	Peripheral VBUSP Interconnect
MCAN7	✓	Peripheral VBUSP Interconnect

Table 13-208. *MCAN* Clocks

This table describes the module clocking signals.

Module Instance	Module Clock Input	Source Clock Signal	Source	Default Freq	Description
MCAN0	MCAN0_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK:HSDIV0_CLKOUT0	200 MHz	MCAN0 Interface Clock
	MCAN0_FCLK (CAN_CLK)	XTALCLK	External Crystal (XTAL)	25 MHz	MCAN0 Functional Clock
		EXT_REFCLK	External Reference Clock(EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK:HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK:HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK:HSDIV0_CLKOUT1	500 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator(RCCLK10M)	10 MHz	
		XTALCLK	External Crystal (XTAL)	25 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator(RCCLK10M)	10 MHz	
MCAN1	MCAN1_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK:HSDIV0_CLKOUT0	200 MHz	MCAN1 Interface Clock
	MCAN1_FCLK (CAN_CLK)	XTALCLK	External Crystal (XTAL)	25 MHz	MCAN1 Functional Clock
		EXT_REFCLK	External Reference Clock(EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK:HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK:HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK:HSDIV0_CLKOUT1	500 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator(RCCLK10M)	10 MHz	
		XTALCLK	External Crystal (XTAL)	25 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator(RCCLK10M)	10 MHz	

Table 13-208. *MCAN* Clocks (continued)

This table describes the module clocking signals.

Module Instance	Module Clock Input	Source Clock Signal	Source	Default Freq	Description
MCAN2	MCAN2_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK:HSDIV0_CLKOUT0	200 MHz	MCAN2 Interface Clock
	MCAN2_FCLK (CAN_CLK)	XTALCLK	External Crystal (XTAL)	25 MHz	MCAN2 Functional Clock
		EXT_REFCLK	External Reference Clock(EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK:HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK:HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK:HSDIV0_CLKOUT1	500 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator(RCCLK10M)	10 MHz	
		XTALCLK	External Crystal (XTAL)	25 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator(RCCLK10M)	10 MHz	
MCAN3	MCAN3_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK:HSDIV0_CLKOUT0	200 MHz	MCAN3 Interface Clock
	MCAN3_FCLK (CAN_CLK)	XTALCLK	External Crystal (XTAL)	25 MHz	MCAN3 Functional Clock
		EXT_REFCLK	External Reference Clock(EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK:HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK:HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK:HSDIV0_CLKOUT1	500 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator(RCCLK10M)	10 MHz	
		XTALCLK	External Crystal (XTAL)	25 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator(RCCLK10M)	10 MHz	
MCAN4	MCAN4_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK:HSDIV0_CLKOUT0	200 MHz	MCAN4 Interface Clock
	MCAN4_FCLK (CAN_CLK)	XTALCLK	External Crystal (XTAL)	25 MHz	MCAN4 Functional Clock
		EXT_REFCLK	External Reference Clock(EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK:HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK:HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK:HSDIV0_CLKOUT1	500 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator(RCCLK10M)	10 MHz	
		XTALCLK	External Crystal (XTAL)	25 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator(RCCLK10M)	10 MHz	

Table 13-208. *MCAN* Clocks (continued)

This table describes the module clocking signals.

Module Instance	Module Clock Input	Source Clock Signal	Source	Default Freq	Description
MCAN5	MCAN5_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK:HSDIV0_CLKOUT0	200 MHz	MCAN5 Interface Clock
	MCAN5_FCLK (CAN_CLK)	XTALCLK	External Crystal (XTAL)	25 MHz	MCAN5 Functional Clock
		EXT_REFCLK	External Reference Clock(EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK:HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK:HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK:HSDIV0_CLKOUT1	500 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator(RCCLK10M)	10 MHz	
		XTALCLK	External Crystal (XTAL)	25 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator(RCCLK10M)	10 MHz	
MCAN6	MCAN6_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK:HSDIV0_CLKOUT0	200 MHz	MCAN6 Interface Clock
	MCAN6_FCLK (CAN_CLK)	XTALCLK	External Crystal (XTAL)	25 MHz	MCAN6 Functional Clock
		EXT_REFCLK	External Reference Clock(EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK:HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK:HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK:HSDIV0_CLKOUT1	500 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator(RCCLK10M)	10 MHz	
		XTALCLK	External Crystal (XTAL)	25 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator(RCCLK10M)	10 MHz	
MCAN7	MCAN7_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK:HSDIV0_CLKOUT0	200 MHz	MCAN7 Interface Clock
	MCAN7_FCLK (CAN_CLK)	XTALCLK	External Crystal (XTAL)	25 MHz	MCAN7 Functional Clock
		EXT_REFCLK	External Reference Clock(EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK:HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK:HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK:HSDIV0_CLKOUT1	500 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator(RCCLK10M)	10 MHz	
		XTALCLK	External Crystal (XTAL)	25 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator(RCCLK10M)	10 MHz	

Table 13-209. *MCAN* Resets

This table describes the module reset signals.

Module Instance	Module Reset Input	Source Reset Signal	Source	Description
MCAN0	MCAN0_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	Asynchronous MCAN0 Module Reset
MCAN1	MCAN1_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	Asynchronous MCAN1 Module Reset
MCAN2	MCAN2_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	Asynchronous MCAN2 Module Reset
MCAN3	MCAN3_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	Asynchronous MCAN3 Module Reset
MCAN4	MCAN4_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	Asynchronous MCAN4 Module Reset
MCAN5	MCAN5_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	Asynchronous MCAN5 Module Reset
MCAN6	MCAN6_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	Asynchronous MCAN6 Module Reset
MCAN7	MCAN7_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	Asynchronous MCAN7 Module Reset

Table 13-210. *MCAN* Interrupt Requests

This table describes the module interrupt requests.

Module Instance	Module Interrupt Signal	Destination Interrupt Input	Destination	Type	Description
MCAN0	MCAN0_INT_0	R5FSS0_CORE0_INTR_IN_27	R5FSS0-0	Level	MCAN0 Line 0 Interrupt Request
		R5FSS0_CORE1_INTR_IN_27	R5FSS0-1		
		R5FSS1_CORE0_INTR_IN_27	R5FSS1-0		
		R5FSS1_CORE1_INTR_IN_27	R5FSS1-1		
		PRU_ICSS0_INTR_IN_40	PRU_ICSS		
	MCAN0_INT_1	R5FSS0_CORE0_INTR_IN_28	R5FSS0-0	Level	MCAN0 Line 1 Interrupt Request
		R5FSS0_CORE1_INTR_IN_28	R5FSS0-1		
		R5FSS1_CORE0_INTR_IN_28	R5FSS1-0		
		R5FSS1_CORE1_INTR_IN_28	R5FSS1-1		
		PRU_ICSS0_INTR_IN_41	PRU_ICSS		
	MCAN0_EXT_TS_ROLLOVER_INT_0	R5FSS0_CORE0_INTR_IN_26	R5FSS0-0	Level	MCAN0 External TimeStamp Counter Rollover Interrupt
		R5FSS0_CORE1_INTR_IN_26	R5FSS0-1		
		R5FSS1_CORE0_INTR_IN_26	R5FSS1-0		
		R5FSS1_CORE1_INTR_IN_26	R5FSS1-1		
		PRU_ICSS0_INTR_IN_39	PRU_ICSS0		
	MCAN0_ECC_CORR_LVL_INT_0	ESM0_LVL_IN_2	ESM0	Level	MCAN0 ECC Correctable Error Interrupt
	MCAN0_ECC_UNCORR_LVL_INT_0	ESM0_LVL_IN_3	ESM0	Level	MCAN0 ECC Uncorrectable Error Interrupt

Table 13-210. *MCAN* Interrupt Requests (continued)

This table describes the module interrupt requests.

Module Instance	Module Interrupt Signal	Destination Interrupt Input	Destination	Type	Description
MCAN1	MCAN1_INT_0	R5FSS0_CORE0_INTR_IN_30	R5FSS0-0	Level	MCAN1 Line 0 Interrupt Request
		R5FSS0_CORE1_INTR_IN_30	R5FSS0-1		
		R5FSS1_CORE0_INTR_IN_30	R5FSS1-0		
		R5FSS1_CORE1_INTR_IN_30	R5FSS1-1		
		PRU_ICSS0_INTR_IN_43	PRU_ICSS		
	MCAN1_INT_1	R5FSS0_CORE0_INTR_IN_31	R5FSS0-0	Level	MCAN1 Line 1 Interrupt Request
		R5FSS0_CORE1_INTR_IN_31	R5FSS0-1		
		R5FSS1_CORE0_INTR_IN_31	R5FSS1-0		
		R5FSS1_CORE1_INTR_IN_31	R5FSS1-1		
		PRU_ICSS0_INTR_IN_44	PRU_ICSS		
	MCAN1_EXT_TS_ROLLOVER_INT_0	R5FSS0_CORE0_INTR_IN_29	R5FSS0-0	Level	MCAN1 External TimeStamp Counter Rollover Interrupt
		R5FSS0_CORE1_INTR_IN_29	R5FSS0-1		
		R5FSS1_CORE0_INTR_IN_29	R5FSS1-0		
		R5FSS1_CORE1_INTR_IN_29	R5FSS1-1		
		PRU_ICSS0_INTR_IN_42	PRU_ICSS		
	MCAN1_ECC_CORR_LVL_INT_0	ESM0_LVL_IN_4	ESM0	Level	MCAN1 ECC Correctable Error Interrupt
	MCAN1_ECC_UNCORR_LVL_INT_0	ESM0_LVL_IN_5	ESM0	Level	MCAN1 ECC Uncorrectable Error Interrupt
MCAN2	MCAN2_INT_0	R5FSS0_CORE1_INTR_IN_33	R5FSS0-0	Level	MCAN1 Line 0 Interrupt Request
		R5FSS0_CORE1_INTR_IN_33	R5FSS0-1		
		R5FSS1_CORE0_INTR_IN_33	R5FSS1-0		
		R5FSS1_CORE1_INTR_IN_33	R5FSS1-1		
		PRU_ICSS0_INTR_IN_46	PRU_ICSS		
	MCAN2_INT_1	R5FSS0_CORE0_INTR_IN_34	R5FSS0-0	Level	MCAN2 Line 1 Interrupt Request
		R5FSS0_CORE1_INTR_IN_34	R5FSS0-1		
		R5FSS1_CORE0_INTR_IN_34	R5FSS1-0		
		R5FSS1_CORE1_INTR_IN_34	R5FSS1-1		
		PRU_ICSS0_INTR_IN_47	PRU_ICSS		
	MCAN2_EXT_TS_ROLLOVER_INT_0	R5FSS0_CORE0_INTR_IN_32	R5FSS0-0	Level	MCAN2 External TimeStamp Counter Rollover Interrupt
		R5FSS0_CORE1_INTR_IN_32	R5FSS0-1		
		R5FSS1_CORE0_INTR_IN_32	R5FSS1-0		
		R5FSS1_CORE1_INTR_IN_32	R5FSS1-1		
		PRU_ICSS0_INTR_IN_45	PRU_ICSS		
	MCAN2_ECC_CORR_LVL_INT_0	ESM0_LVL_IN_6	ESM0	Level	MCAN2 ECC Correctable Error Interrupt
	MCAN2_ECC_UNCORR_LVL_INT_0	ESM0_LVL_IN_7	ESM0	Level	MCAN2 ECC Uncorrectable Error Interrupt

Table 13-210. *MCAN* Interrupt Requests (continued)

This table describes the module interrupt requests.

Module Instance	Module Interrupt Signal	Destination Interrupt Input	Destination	Type	Description
MCAN3	MCAN3_INT_0	R5FSS0_CORE0_INTR_IN_36	R5FSS0-0	Level	MCAN3 Line 0 Interrupt Request
		R5FSS0_CORE1_INTR_IN_36	R5FSS0-1		
		R5FSS1_CORE0_INTR_IN_36	R5FSS1-0		
		R5FSS1_CORE1_INTR_IN_36	R5FSS1-1		
		PRU_ICSS0_INTR_IN_49	PRU_ICSS		
	MCAN3_INT_1	R5FSS0_CORE0_INTR_IN_37	R5FSS0-0	Level	MCAN3 Line 1 Interrupt Request
		R5FSS0_CORE1_INTR_IN_37	R5FSS0-1		
		R5FSS1_CORE0_INTR_IN_37	R5FSS1-0		
		R5FSS1_CORE1_INTR_IN_37	R5FSS1-1		
		PRU_ICSS0_INTR_IN_50	PRU_ICSS		
	MCAN3_EXT_TS_ROLLOVER_INT_0	R5FSS0_CORE0_INTR_IN_35	R5FSS0-0	Level	MCAN3 External TimeStamp Counter Rollover Interrupt
		R5FSS0_CORE1_INTR_IN_35	R5FSS0-1		
		R5FSS1_CORE0_INTR_IN_35	R5FSS1-0		
		R5FSS1_CORE1_INTR_IN_35	R5FSS1-1		
		PRU_ICSS0_INTR_IN_48	PRU_ICSS		
	MCAN3_ECC_CORR_LVL_INT_0	ESM0_LVL_IN_8	ESM0	Level	MCAN3 ECC Correctable Error Interrupt
	MCAN3_ECC_UNCORR_LVL_INT_0	ESM0_LVL_IN_9	ESM0	Level	MCAN3 ECC Uncorrectable Error Interrupt
MCAN4	MCAN4_INT_0	R5FSS0_CORE0_INTR_IN_198	R5FSS0-0	Level	MCAN4 Line 0 Interrupt Request
		R5FSS0_CORE1_INTR_IN_198	R5FSS0-1		
		R5FSS1_CORE0_INTR_IN_198	R5FSS1-0		
		R5FSS1_CORE1_INTR_IN_198	R5FSS1-1		
		PRU_ICSS0_INTR_IN_60	PRU_ICSS		
	MCAN4_INT_1	R5FSS0_CORE0_INTR_IN_199	R5FSS0-0	Level	MCAN4 Line 1 Interrupt Request
		R5FSS0_CORE1_INTR_IN_199	R5FSS0-1		
		R5FSS1_CORE0_INTR_IN_199	R5FSS1-0		
		R5FSS1_CORE1_INTR_IN_199	R5FSS1-1		
		PRU_ICSS0_INTR_IN_61	PRU_ICSS		
	MCAN4_EXT_TS_ROLLOVER_INT_0	R5FSS0_CORE0_INTR_IN_197	R5FSS0-0	Level	MCAN4 External TimeStamp Counter Rollover Interrupt
		R5FSS0_CORE1_INTR_IN_197	R5FSS0-1		
		R5FSS1_CORE0_INTR_IN_197	R5FSS1-0		
		R5FSS1_CORE1_INTR_IN_197	R5FSS1-1		
		PRU_ICSS0_INTR_IN_59	PRU_ICSS		
	MCAN4_ECC_CORR_LVL_INT_0	ESM0_LVL_IN_39	ESM0	Level	MCAN4 ECC Correctable Error Interrupt
	MCAN4_ECC_UNCORR_LVL_INT_0	ESM0_LVL_IN_40	ESM0	Level	MCAN4 ECC Uncorrectable Error Interrupt

Table 13-210. *MCAN* Interrupt Requests (continued)

This table describes the module interrupt requests.

Module Instance	Module Interrupt Signal	Destination Interrupt Input	Destination	Type	Description
MCAN5	MCAN5_INT_0	R5FSS0_CORE0_INTR_IN_201	R5FSS0-0	Level	MCAN5 Line 0 Interrupt Request
		R5FSS0_CORE1_INTR_IN_201	R5FSS0-1		
		R5FSS1_CORE0_INTR_IN_201	R5FSS1-0		
		R5FSS1_CORE1_INTR_IN_201	R5FSS1-1		
		PRU_ICSS0_INTR_IN_63	PRU_ICSS		
	MCAN5_INT_1	R5FSS0_CORE0_INTR_IN_202	R5FSS0-0	Level	MCAN5 Line 1 Interrupt Request
		R5FSS0_CORE1_INTR_IN_202	R5FSS0-1		
		R5FSS1_CORE0_INTR_IN_202	R5FSS1-0		
		R5FSS1_CORE1_INTR_IN_202	R5FSS1-1		
		PRU_ICSS0_INTR_IN_64	PRU_ICSS		
	MCAN5_EXT_TS_ROLLOVER_INT_0	R5FSS0_CORE0_INTR_IN_200	R5FSS0-0	Level	MCAN5 External TimeStamp Counter Rollover Interrupt
		R5FSS0_CORE1_INTR_IN_200	R5FSS0-1		
		R5FSS1_CORE0_INTR_IN_200	R5FSS1-0		
		R5FSS1_CORE1_INTR_IN_200	R5FSS1-1		
		PRU_ICSS0_INTR_IN_62	PRU_ICSS		
	MCAN5_ECC_CORR_LVL_INT_0	ESM0_LVL_IN_41	ESM0	Level	MCAN5 ECC Correctable Error Interrupt
	MCAN5_ECC_UNCORR_LVL_INT_0	ESM0_LVL_IN_42	ESM0	Level	MCAN5 ECC Uncorrectable Error Interrupt
MCAN6	MCAN6_INT_0	R5FSS0_CORE0_INTR_IN_204	R5FSS0-0	Level	MCAN6 Line 0 Interrupt Request
		R5FSS0_CORE1_INTR_IN_204	R5FSS0-1		
		R5FSS1_CORE0_INTR_IN_204	R5FSS1-0		
		R5FSS1_CORE1_INTR_IN_204	R5FSS1-1		
		PRU_ICSS0_INTR_IN_66	PRU_ICSS		
	MCAN6_INT_1	R5FSS0_CORE0_INTR_IN_205	R5FSS0-0	Level	MCAN6 Line 1 Interrupt Request
		R5FSS0_CORE1_INTR_IN_205	R5FSS0-1		
		R5FSS1_CORE0_INTR_IN_205	R5FSS1-0		
		R5FSS1_CORE1_INTR_IN_205	R5FSS1-1		
		PRU_ICSS0_INTR_IN_67	PRU_ICSS		
	MCAN6_EXT_TS_ROLLOVER_INT_0	R5FSS0_CORE0_INTR_IN_203	R5FSS0-0	Level	MCAN6 External TimeStamp Counter Rollover Interrupt
		R5FSS0_CORE1_INTR_IN_203	R5FSS0-1		
		R5FSS1_CORE0_INTR_IN_203	R5FSS1-0		
		R5FSS1_CORE1_INTR_IN_203	R5FSS1-1		
		PRU_ICSS0_INTR_IN_65	PRU_ICSS		
	MCAN6_ECC_CORR_LVL_INT_0	ESM0_LVL_IN_43	ESM0	Level	MCAN6 ECC Correctable Error Interrupt
	MCAN6_ECC_UNCORR_LVL_INT_0	ESM0_LVL_IN_44	ESM0	Level	MCAN6 ECC Uncorrectable Error Interrupt

Table 13-210. *MCAN* Interrupt Requests (continued)

This table describes the module interrupt requests.

Module Instance	Module Interrupt Signal	Destination Interrupt Input	Destination	Type	Description
MCAN7	MCAN7_INT_0	R5FSS0_CORE0_INTR_IN_207	R5FSS0-0	Level	MCAN7 Line 0 Interrupt Request
		R5FSS0_CORE1_INTR_IN_207	R5FSS0-1		
		R5FSS1_CORE0_INTR_IN_207	R5FSS1-0		
		R5FSS1_CORE1_INTR_IN_207	R5FSS1-1		
		PRU_ICSS0_INTR_IN_69	PRU_ICSS		
	MCAN7_INT_1	R5FSS0_CORE0_INTR_IN_208	R5FSS0-0	Level	MCAN7 Line 1 Interrupt Request
		R5FSS0_CORE1_INTR_IN_208	R5FSS0-1		
		R5FSS1_CORE0_INTR_IN_208	R5FSS1-0		
		R5FSS1_CORE1_INTR_IN_208	R5FSS1-1		
		PRU_ICSS0_INTR_IN_70	PRU_ICSS		
	MCAN7_EXT_TS_ROLLOVER_INT_0	R5FSS0_CORE0_INTR_IN_206	R5FSS0-0	Level	MCAN7 External TimeStamp Counter Rollover Interrupt
		R5FSS0_CORE1_INTR_IN_206	R5FSS0-1		
		R5FSS1_CORE0_INTR_IN_206	R5FSS1-0		
		R5FSS1_CORE1_INTR_IN_206	R5FSS1-1		
		PRU_ICSS0_INTR_IN_68	PRU_ICSS		
	MCAN7_ECC_CORR_LVL_INT_0	ESM0_LVL_IN_45	ESM0	Level	MCAN7 ECC Correctable Error Interrupt
	MCAN7_ECC_UNCORR_LVL_INT_0	ESM0_LVL_IN_46	ESM0	Level	MCAN7 ECC Uncorrectable Error Interrupt

Table 13-211. *MCAN* DMA Requests

This table describes the module DMA requests.

Module Instance	Module DMA Event	Destination DMA Event Input	Destination	Type	Description
MCAN0	MCAN0_FE_INTR_0	EDMA_XBAR_147	EDMA	Pulse	MCAN0 Receive Filter Event 0 DMA Request
	MCAN0_FE_INTR_1	EDMA_XBAR_148	EDMA	Pulse	MCAN0 Receive Filter Event 1 DMA Request
	MCAN0_FE_INTR_2	EDMA_XBAR_149	EDMA	Pulse	MCAN0 Receive Filter Event 2 DMA Request
	MCAN0_FE_INTR_3	EDMA_XBAR_150	EDMA	Pulse	MCAN0 Receive Filter Event 3 DMA Request
	MCAN0_FE_INTR_4	EDMA_XBAR_151	EDMA	Pulse	MCAN0 Receive Filter Event 4 DMA Request
	MCAN0_FE_INTR_5	EDMA_XBAR_152	EDMA	Pulse	MCAN0 Receive Filter Event 5 DMA Request
	MCAN0_FE_INTR_6	EDMA_XBAR_153	EDMA	Pulse	MCAN0 Receive Filter Event 6 DMA Request
	MCAN0_TXDMA_0	EDMA_XBAR_74	EDMA	Pulse	MCAN0 Transmit Core DMA Request 0
	MCAN0_TXDMA_1	EDMA_XBAR_75	EDMA	Pulse	MCAN0 Transmit Core DMA Request 1
	MCAN0_TXDMA_2	EDMA_XBAR_76	EDMA	Pulse	MCAN0 Transmit Core DMA Request 2
	MCAN0_TXDMA_3	EDMA_XBAR_77	EDMA	Pulse	MCAN0 Transmit Core DMA Request 3
MCAN1	MCAN1_FE_INTR_0	EDMA_XBAR_154	EDMA	Pulse	MCAN1 Receive Filter Event 0 DMA Request
	MCAN1_FE_INTR_1	EDMA_XBAR_155	EDMA	Pulse	MCAN1 Receive Filter Event 1 DMA Request
	MCAN1_FE_INTR_2	EDMA_XBAR_156	EDMA	Pulse	MCAN1 Receive Filter Event 2 DMA Request
	MCAN1_FE_INTR_3	EDMA_XBAR_157	EDMA	Pulse	MCAN1 Receive Filter Event 3 DMA Request
	MCAN1_FE_INTR_4	EDMA_XBAR_158	EDMA	Pulse	MCAN1 Receive Filter Event 4 DMA Request
	MCAN1_FE_INTR_5	EDMA_XBAR_159	EDMA	Pulse	MCAN1 Receive Filter Event 5 DMA Request
	MCAN1_FE_INTR_6	EDMA_XBAR_160	EDMA	Pulse	MCAN1 Receive Filter Event 6 DMA Request
	MCAN1_TXDMA_0	EDMA_XBAR_78	EDMA	Pulse	MCAN1 Transmit Core DMA Request 0
	MCAN1_TXDMA_1	EDMA_XBAR_79	EDMA	Pulse	MCAN1 Transmit Core DMA Request 1
	MCAN1_TXDMA_2	EDMA_XBAR_80	EDMA	Pulse	MCAN1 Transmit Core DMA Request 2
	MCAN1_TXDMA_3	EDMA_XBAR_81	EDMA	Pulse	MCAN1 Transmit Core DMA Request 3

Table 13-211. *MCAN* DMA Requests (continued)

This table describes the module DMA requests.

Module Instance	Module DMA Event	Destination DMA Event Input	Destination	Type	Description
MCAN2	MCAN2_FE_INTR_0	EDMA_XBAR_161	EDMA	Pulse	MCAN2 Receive Filter Event 0 DMA Request
	MCAN2_FE_INTR_1	EDMA_XBAR_162	EDMA	Pulse	MCAN2 Receive Filter Event 1 DMA Request
	MCAN2_FE_INTR_2	EDMA_XBAR_163	EDMA	Pulse	MCAN2 Receive Filter Event 2 DMA Request
	MCAN2_FE_INTR_3	EDMA_XBAR_164	EDMA	Pulse	MCAN2 Receive Filter Event 3 DMA Request
	MCAN2_FE_INTR_4	EDMA_XBAR_165	EDMA	Pulse	MCAN2 Receive Core Filter Event 4 DMA Request
	MCAN2_FE_INTR_5	EDMA_XBAR_166	EDMA	Pulse	MCAN2 Receive Filter Event 5 DMA Request
	MCAN2_FE_INTR_6	EDMA_XBAR_167	EDMA	Pulse	MCAN2 Receiver Filter Event 6 DMA Request
	MCAN2_TXDMA_0	EDMA_XBAR_82	EDMA	Pulse	MCAN2 Transmit Core DMA Request 0
	MCAN2_TXDMA_1	EDMA_XBAR_83	EDMA	Pulse	MCAN2 Transmit Core DMA Request 1
	MCAN2_TXDMA_2	EDMA_XBAR_84	EDMA	Pulse	MCAN2 Transmit Core DMA Request 2
	MCAN2_TXDMA_3	EDMA_XBAR_85	EDMA	Pulse	MCAN2 Transmit Core DMA Request 3
MCAN3	MCAN3_FE_INTR_0	EDMA_XBAR_168	EDMA	Pulse	MCAN3 Receive Filter Event 0 DMA Request
	MCAN3_FE_INTR_1	EDMA_XBAR_169	EDMA	Pulse	MCAN3 Receive Filter Event 1 DMA Request
	MCAN3_FE_INTR_2	EDMA_XBAR_170	EDMA	Pulse	MCAN3 Receive Filter Event 2 DMA Request
	MCAN3_FE_INTR_3	EDMA_XBAR_171	EDMA	Pulse	MCAN3 Receive Filter Event 3 DMA Request
	MCAN3_FE_INTR_4	EDMA_XBAR_172	EDMA	Pulse	MCAN3 Receive Filter Event 4 DMA Request
	MCAN3_FE_INTR_5	EDMA_XBAR_173	EDMA	Pulse	MCAN3 Receive Filter Event 5 DMA Request
	MCAN3_FE_INTR_6	EDMA_XBAR_174	EDMA	Pulse	MCAN3 Receive Filter Event 6 DMA Request
	MCAN3_TXDMA_0	EDMA_XBAR_86	EDMA	Pulse	MCAN3 Transmit Core DMA Request 0
	MCAN3_TXDMA_1	EDMA_XBAR_87	EDMA	Pulse	MCAN3 Transmit Core DMA Request 1
	MCAN3_TXDMA_2	EDMA_XBAR_88	EDMA	Pulse	MCAN3 Transmit Core DMA Request 2
	MCAN3_TXDMA_3	EDMA_XBAR_89	EDMA	Pulse	MCAN3 Transmit Core DMA Request 3

Table 13-211. *MCAN* DMA Requests (continued)

This table describes the module DMA requests.

Module Instance	Module DMA Event	Destination DMA Event Input	Destination	Type	Description
MCAN4	MCAN4_FE_INTR_0	EDMA_XBAR_192	EDMA	Pulse	MCAN4 Receive Filter Event 0 DMA Request
	MCAN4_FE_INTR_1	EDMA_XBAR_193	EDMA	Pulse	MCAN4 Receive Filter Event 1 DMA Request
	MCAN4_FE_INTR_2	EDMA_XBAR_194	EDMA	Pulse	MCAN4 Receive Filter Event 2 DMA Request
	MCAN4_FE_INTR_3	EDMA_XBAR_195	EDMA	Pulse	MCAN4 Receive Filter Event 3 DMA Request
	MCAN4_FE_INTR_4	EDMA_XBAR_196	EDMA	Pulse	MCAN4 Receive Filter Event 4 DMA Request
	MCAN4_FE_INTR_5	EDMA_XBAR_197	EDMA	Pulse	MCAN4 Receive Filter Event 5 DMA Request
	MCAN4_FE_INTR_6	EDMA_XBAR_198	EDMA	Pulse	MCAN4 Receive Filter Event 6 DMA Request
	MCAN4_TXDMA_0	EDMA_XBAR_176	EDMA	Pulse	MCAN4 Transmit Core DMA Request 0
	MCAN4_TXDMA_1	EDMA_XBAR_177	EDMA	Pulse	MCAN4 Transmit Core DMA Request 1
	MCAN4_TXDMA_2	EDMA_XBAR_178	EDMA	Pulse	MCAN4 Transmit Core DMA Request 2
	MCAN4_TXDMA_3	EDMA_XBAR_179	EDMA	Pulse	MCAN4 Transmit Core DMA Request 3
MCAN5	MCAN5_FE_INTR_0	EDMA_XBAR_199	EDMA	Pulse	MCAN5 Receive Filter Event 0 DMA Request
	MCAN5_FE_INTR_1	EDMA_XBAR_200	EDMA	Pulse	MCAN5 Receive Filter Event 1 DMA Request
	MCAN5_FE_INTR_2	EDMA_XBAR_201	EDMA	Pulse	MCAN5 Receive Filter Event 2 DMA Request
	MCAN5_FE_INTR_3	EDMA_XBAR_202	EDMA	Pulse	MCAN5 Receive Filter Event 3 DMA Request
	MCAN5_FE_INTR_4	EDMA_XBAR_203	EDMA	Pulse	MCAN5 Receive Filter Event 4 DMA Request
	MCAN5_FE_INTR_5	EDMA_XBAR_204	EDMA	Pulse	MCAN5 Receive Filter Event 5 DMA Request
	MCAN5_FE_INTR_6	EDMA_XBAR_205	EDMA	Pulse	MCAN5 Receive Filter Event 6 DMA Request
	MCAN5_TXDMA_0	EDMA_XBAR_180	EDMA	Pulse	MCAN5 Transmit Core DMA Request 0
	MCAN5_TXDMA_1	EDMA_XBAR_181	EDMA	Pulse	MCAN5 Transmit Core DMA Request 1
	MCAN5_TXDMA_2	EDMA_XBAR_182	EDMA	Pulse	MCAN5 Transmit Core DMA Request 2
	MCAN5_TXDMA_3	EDMA_XBAR_183	EDMA	Pulse	MCAN5 Transmit Core DMA Request 3

Table 13-211. *MCAN* DMA Requests (continued)

This table describes the module DMA requests.

Module Instance	Module DMA Event	Destination DMA Event Input	Destination	Type	Description
MCAN6	MCAN6_FE_INTR_0	EDMA_XBAR_206	EDMA	Pulse	MCAN6 Receive Filter Event 0 DMA Request
	MCAN6_FE_INTR_1	EDMA_XBAR_207	EDMA	Pulse	MCAN6 Receive Filter Event 1 DMA Request
	MCAN6_FE_INTR_2	EDMA_XBAR_208	EDMA	Pulse	MCAN6 Receive Filter Event 2 DMA Request
	MCAN6_FE_INTR_3	EDMA_XBAR_209	EDMA	Pulse	MCAN6 Receive Filter Event 3 DMA Request
	MCAN6_FE_INTR_4	EDMA_XBAR_210	EDMA	Pulse	MCAN6 Receive Filter Event 4 DMA Request
	MCAN6_FE_INTR_5	EDMA_XBAR_211	EDMA	Pulse	MCAN6 Receive Filter Event 5 DMA Request
	MCAN6_FE_INTR_6	EDMA_XBAR_212	EDMA	Pulse	MCAN6 Receive Filter Event 6 DMA Request
	MCAN6_TXDMA_0	EDMA_XBAR_184	EDMA	Pulse	MCAN6 Transmit Core DMA Request 0
	MCAN6_TXDMA_1	EDMA_XBAR_185	EDMA	Pulse	MCAN6 Transmit Core DMA Request 1
	MCAN6_TXDMA_2	EDMA_XBAR_186	EDMA	Pulse	MCAN6 Transmit Core DMA Request 2
	MCAN6_TXDMA_3	EDMA_XBAR_187	EDMA	Pulse	MCAN6 Transmit Core DMA Request 3

Table 13-211. *MCAN* DMA Requests (continued)

This table describes the module DMA requests.

Module Instance	Module DMA Event	Destination DMA Event Input	Destination	Type	Description
MCAN7	MCAN7_FE_INTR_0	EDMA_XBAR_213	EDMA	Pulse	MCAN7 Receive Filter Event 0 DMA Request
	MCAN7_FE_INTR_1	EDMA_XBAR_214	EDMA	Pulse	MCAN7 Receive Filter Event 1 DMA Request
	MCAN7_FE_INTR_2	EDMA_XBAR_215	EDMA	Pulse	MCAN7 Receive Filter Event 2 DMA Request
	MCAN7_FE_INTR_3	EDMA_XBAR_216	EDMA	Pulse	MCAN7 Receive Filter Event 3 DMA Request
	MCAN7_FE_INTR_4	EDMA_XBAR_217	EDMA	Pulse	MCAN7 Receive Filter Event 4 DMA Request
	MCAN7_FE_INTR_5	EDMA_XBAR_218	EDMA	Pulse	MCAN7 Receive Filter Event 5 DMA Request
	MCAN7_FE_INTR_6	EDMA_XBAR_219	EDMA	Pulse	MCAN7 Receive Filter Event 6 DMA Request
	MCAN7_TXDMA_0	EDMA_XBAR_188	EDMA	Pulse	MCAN7 Transmit Core DMA Request 0
	MCAN7_TXDMA_1	EDMA_XBAR_189	EDMA	Pulse	MCAN7 Transmit Core DMA Request 1
	MCAN7_TXDMA_2	EDMA_XBAR_190	EDMA	Pulse	MCAN7 Transmit Core DMA Request 2
	MCAN7_TXDMA_3	EDMA_XBAR_191	EDMA	Pulse	MCAN7 Transmit Core DMA Request 3

Note

For more information on the interconnects, see the *System Interconnect* chapter.

For more information on power, reset, and clock management, see the corresponding sections within the *Device Configuration* chapter.

For more information on the device interrupt controllers, see the *Interrupt Controllers* chapter.

13.4.1.4 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 13-168 shows the MCAN module block diagram.

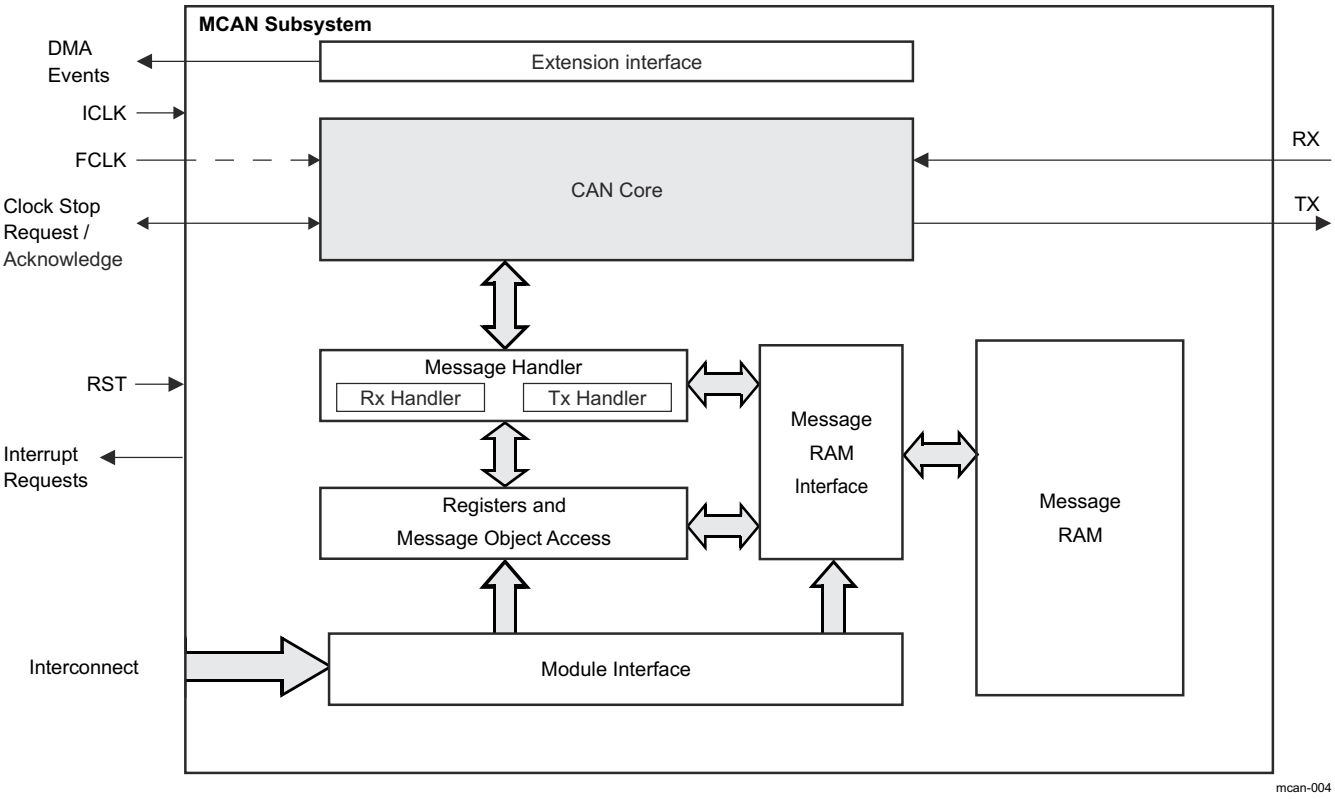


Figure 13-168. MCAN Block Diagram

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 Section 13.4.1.4.10, *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

- **Clocking**: two clocks are provided to the MCAN module: the peripheral synchronous clock (interface clock ICLK) and the peripheral asynchronous clock (functional clock - FCLK).
- **Extension Interface**: this interface is used for DMA requests signaling (see Section 13.4.1.4.2.2).

13.4.1.4.1 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 Section 6.4, *Clocking* and the device-specific Datasheet.

13.4.1.4.2 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 13.4.1.4.3.8.2, *Suspend Mode*).

13.4.1.4.2.1 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)

- External Timestamp Unserviced Interrupts Counter Register (MCANSS_EXT_TS_UNSERVICED_INTR_CNTR)

13.4.1.4.2.2 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).

13.4.1.4.3 Operating Modes

13.4.1.4.3.1 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.

Table 13-212 shows the steps to configure the MCAN module.

Table 13-212. 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

13.4.1.4.3.2 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.

13.4.1.4.3.3 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 13-169, 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.

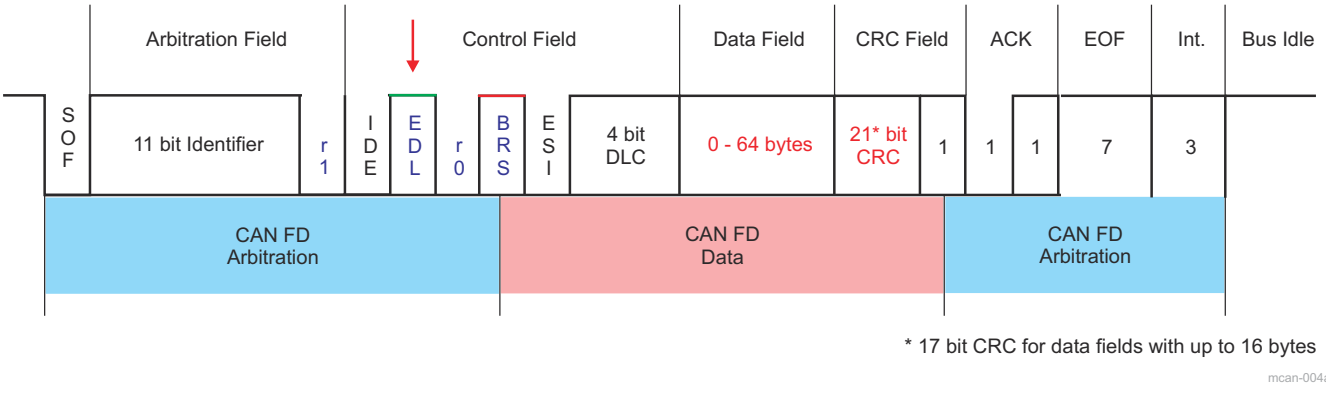


Figure 13-169. CAN FD Frame

Note

Figure 13-169 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 13-213).

Table 13-213. 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

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 13-170) 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.

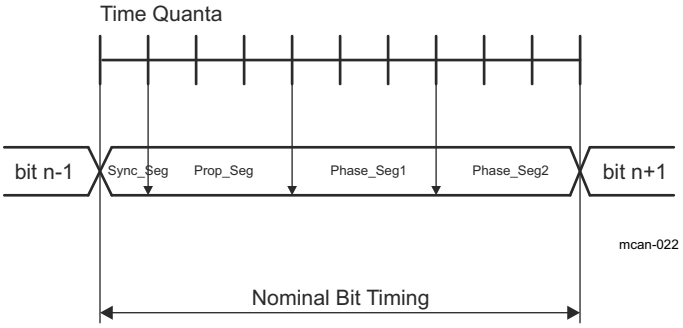


Figure 13-170. CAN Bit Timing

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.

The calculation of the parameters required for CAN bit timing configuration is dependent on a few fundamental equations. The bit rate (bits per second) calculation is based on the speed of the CAN clock, the bit rate pre-scalar value (BRP), and the time segments used to define the sampling point for the bit. The sampling point is the point of time at which the bus level is read and interpreted as the value at that respective time. The typical value of the sampling point is between 75-90%. Time segment 1 (TSEG1) is the time before the sampling point and is defined as the Prog_Seg time plus the Phase_Seg1 time per the CAN Bit Timing diagram. Time segment 2 (TSEG2) is the time after the sampling point which makes it equal to Phase_Seg2. When necessary, the Sync_Seg period of the nominal bit timing interval should be reflected in the equations by adding '1'.

$$TSEG1 = \text{Prop\_Seg} + \text{Phase\_Seg1} \tag{26}$$

$$TSEG2 = \text{Phase\_Seg2} \tag{27}$$

$$Sampling\ Point\ (\%) = \frac{1 + \text{TSEG1}}{1 + \text{TSEG1} + \text{TSEG2}} \tag{28}$$

$$Bit\ rate\left(bits\ per\ second\right) = \frac{\frac{\text{CAN clock speed in Hz}}{BRP}}{1 + \text{TSEG1} + \text{TSEG2}} \tag{29}$$

13.4.1.4.3.4 Transmitter Delay Compensation

13.4.1.4.3.4.1 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 13-171). 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.

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.

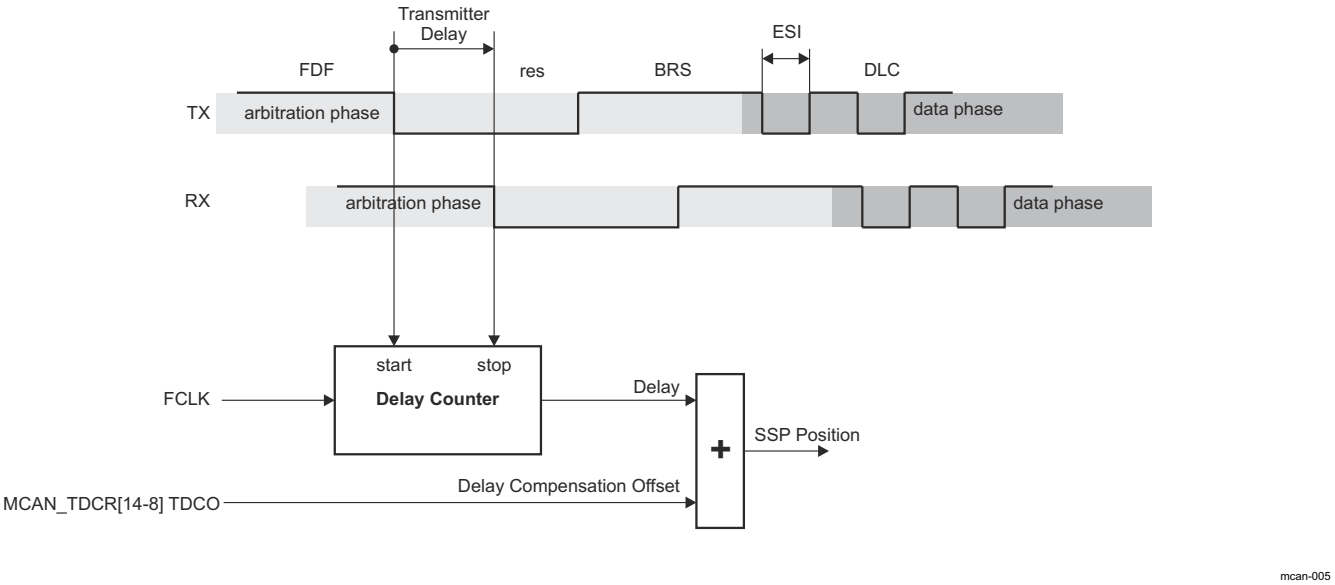


Figure 13-171. Transmitter Delay Measurement

13.4.1.4.3.4.2 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 13-171). 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.

13.4.1.4.3.5 Restricted Operation Mode

In Restricted Operation Mode the CAN node is able to receive data and remote frames and acknowledge 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 receive 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.

The Restricted Operation Mode is automatically entered when the Tx Handler does not 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

Restricted Operation Mode must not be combined with Internal Loopback Mode.

13.4.1.4.3.6 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 13-172 shows the connection of the TX and RX signals to the MCAN module in Bus Monitoring Mode.

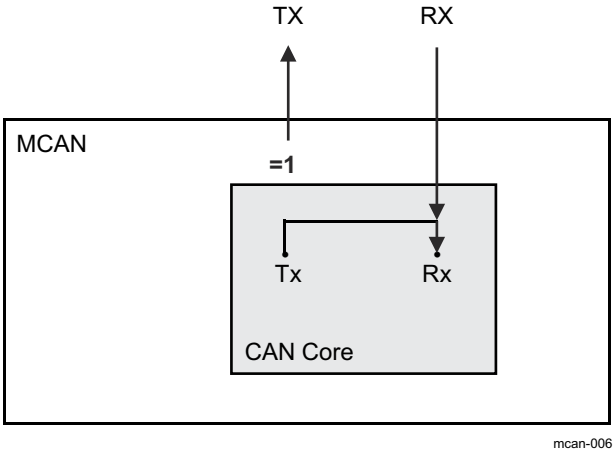


Figure 13-172. Connection of Signals in Bus Monitoring Mode

13.4.1.4.3.7 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).

13.4.1.4.3.7.1 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
- 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).

13.4.1.4.3.8 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).

13.4.1.4.3.8.1 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 13.4.1.4.3.8.3, *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.

13.4.1.4.3.8.2 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 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 13.4.1.4.3.8.3, *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

13.4.1.4.3.8.3 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. It takes a few FCLK clock cycles for before the write to clear function effects the MCAN_CCCR[0] INIT bit. During clock stop the FCLK clock is turned off and is re-enabled when clock stop request is removed. It takes a few FCLK clock cycles for the FCLK clock to be re-enabled followed by a few more for the synchronization of the MCAN_CCCR[0] INIT bit to take effect. After completion of these steps the MCAN core resumes fully active 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

13.4.1.4.3.9 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.

13.4.1.4.3.9.1 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 13-173 shows the connection of the TX and RX pins to the MCAN module in case of Internal Loopback Mode.

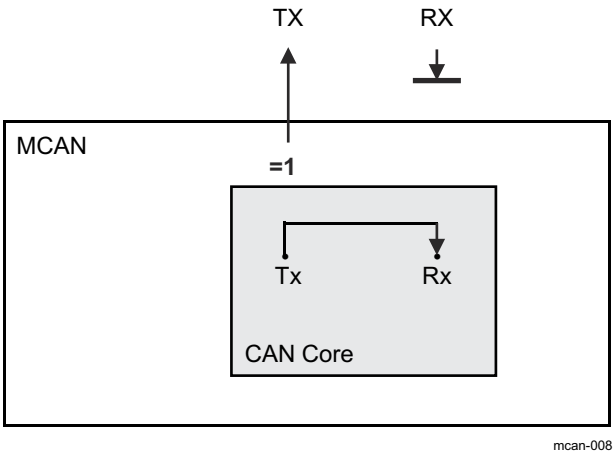


Figure 13-173. Internal Loopback Mode

13.4.1.4.4 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 13.4.1.4.10, *Message RAM*.

13.4.1.4.4.1 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 13-174). 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.

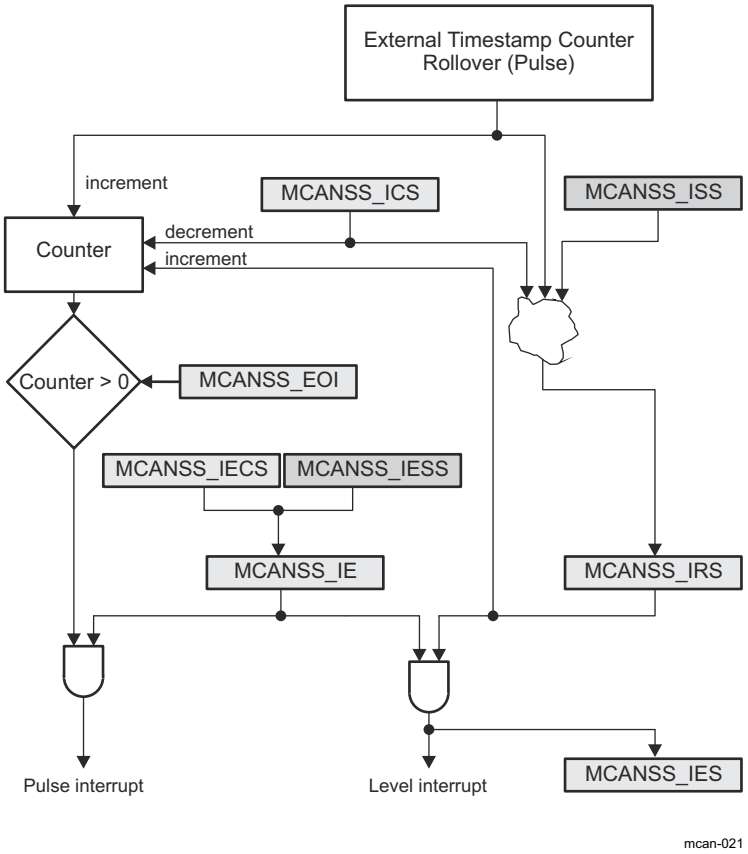


Figure 13-174. External Timestamp Counter Interrupt

13.4.1.4.5 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.

13.4.1.4.6 ECC Support

The Message Memory is wrapped in an ECC wrapper providing SECDED parity functionality. The ECC wrapper is controlled by an ECC Aggregator.

13.4.1.4.6.1 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.

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.

13.4.1.4.7 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

13.4.1.4.7.1 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 13.4.1.4.10, *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 13.4.1.4.2.1, *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:
  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 13.4.1.4.7.2.2 have to be considered.

13.4.1.4.7.1.1 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 13.4.1.4.10.5, *Standard Message ID Filter Element* and Section 13.4.1.4.10.6, *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.

13.4.1.4.7.1.2 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 13.4.1.4.10.5, *Standard Message ID Filter Element* and Section 13.4.1.4.10.6, *Extended Message ID Filter Element*.

13.4.1.4.7.1.3 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.

13.4.1.4.7.1.4 Standard Message ID Filtering

The standard Message ID (11-bit ID) filtering flow is shown in Figure 13-175. Section 13.4.1.4.10.5, *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

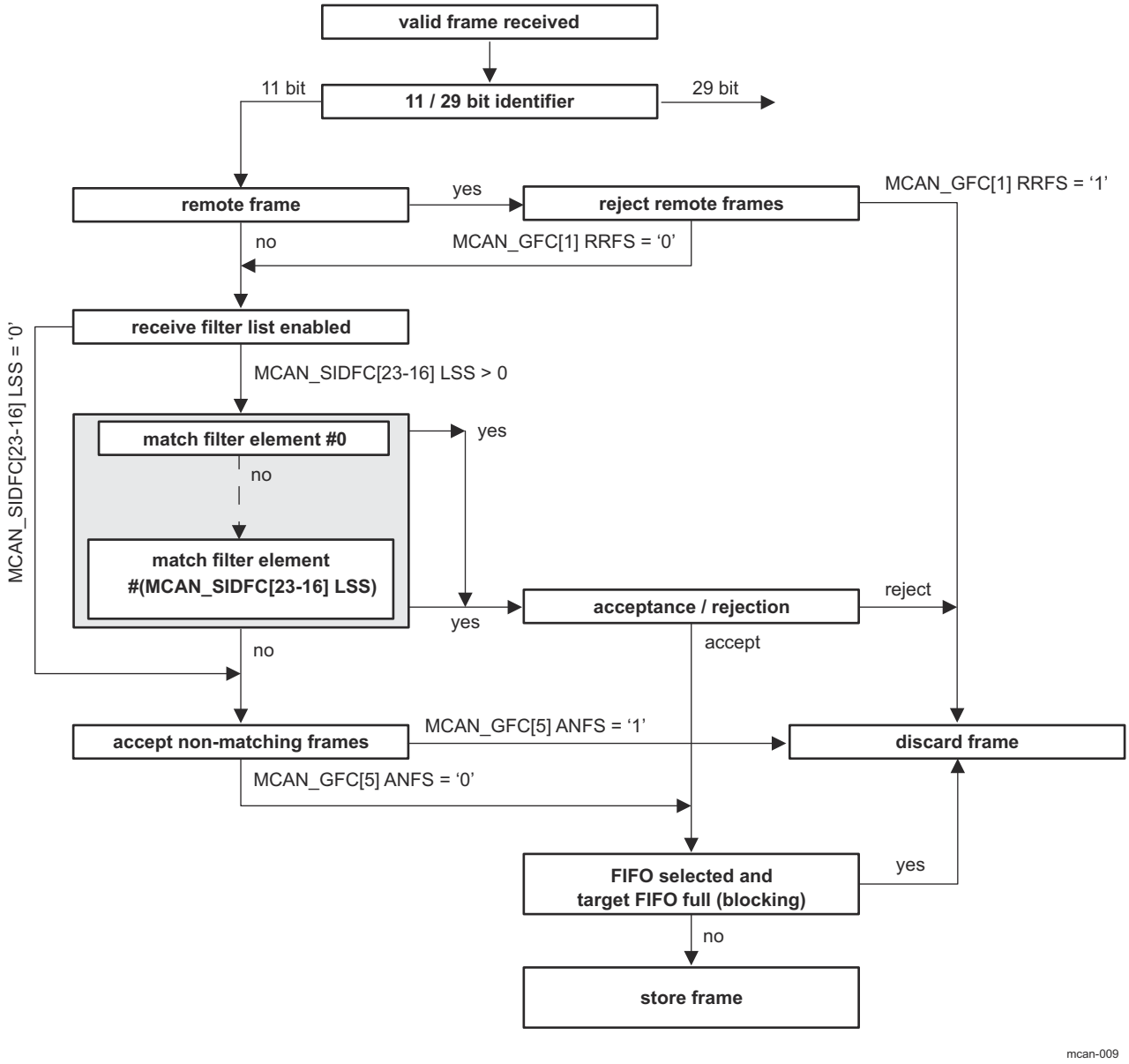


Figure 13-175. Standard Message ID Filter Path

13.4.1.4.7.1.5 Extended Message ID Filtering

The extended Message ID (29-bit ID) filtering flow is shown in Figure 13-176. Section 13.4.1.4.10.6, *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).

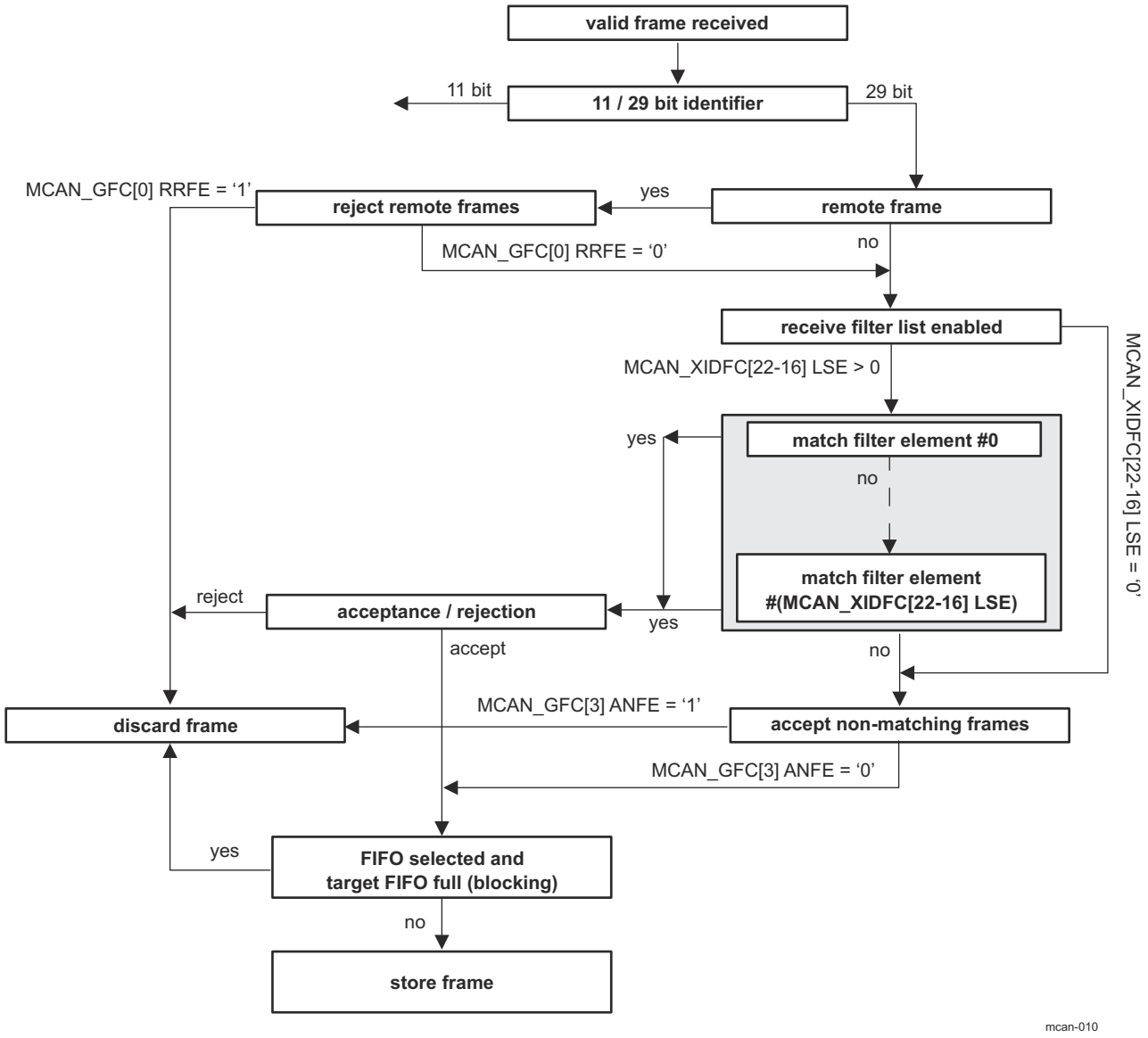


Figure 13-176. Extended Message ID Filter Path

13.4.1.4.7.2 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 13.4.1.4.7.1, *Acceptance Filtering*. Section 13.4.1.4.10.2, *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 13-177 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).

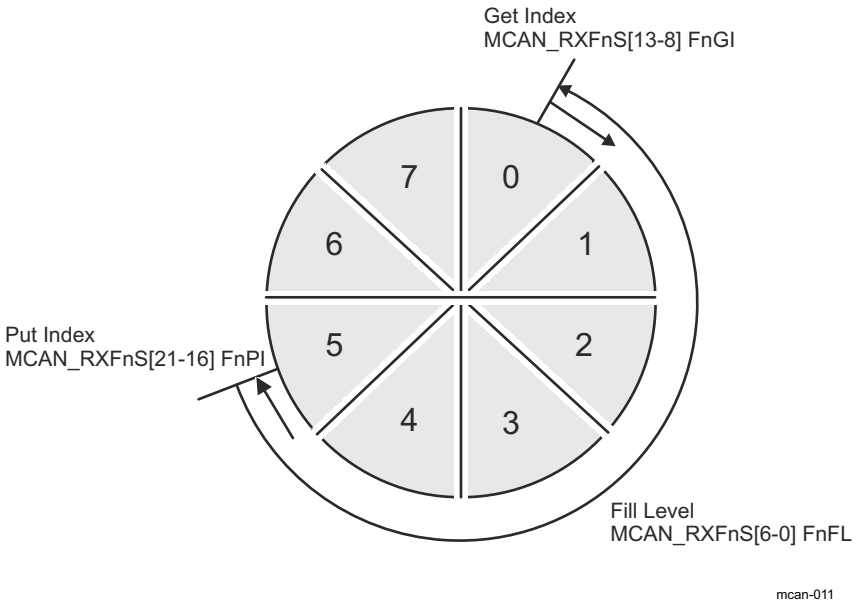


Figure 13-177. Rx FIFO Status

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 13-214 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 13-214. 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

13.4.1.4.7.2.1 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.

13.4.1.4.7.2.2 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, 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 13-178 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.

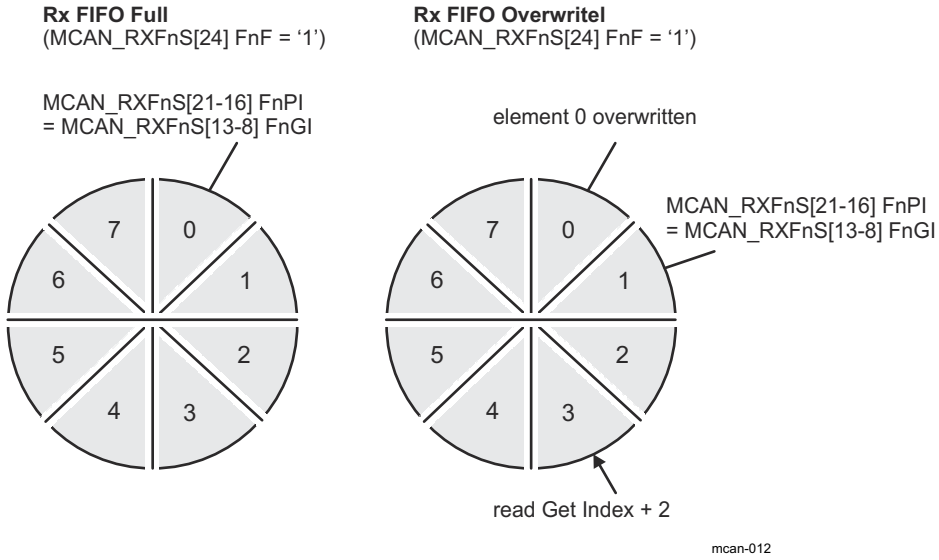


Figure 13-178. Rx FIFO Overflow Handling

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).

13.4.1.4.7.3 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 13.4.1.4.10.5, *Standard Message ID Filter Element* and Section 13.4.1.4.10.6, *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 13-215 shows Example Filter Configuration for Rx Buffers.

Table 13-215. 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

Table 13-215. Example Filter Configuration for Rx Buffers (continued)

Filter Element	SFID1[10-0] EFID1[28-0]	SFID2[10-9] EFID2[10-9]	SFID2[5-0] EFID2[5-0]
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 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.

13.4.1.4.7.3.1 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

13.4.1.4.7.4 Debug on CAN Support

Debug DMA is not supported feature. Debug messages can be traced through the RX FIFO (see Section 13.4.1.4.7.2).

13.4.1.4.8 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 13.4.1.4.10.3 describes the Tx Buffer Element. Table 13-216 shows the possible configurations for message transmission.

Table 13-216. 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.

13.4.1.4.8.1 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 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.

13.4.1.4.8.2 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 13-217 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 13-217. 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

13.4.1.4.8.3 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 13-217). The start address of the next available (free) Tx FIFO 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.

13.4.1.4.8.4 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 13-217). 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.

13.4.1.4.8.5 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 13-179 shows Mixed Dedicated Tx Buffers/Tx FIFO example.

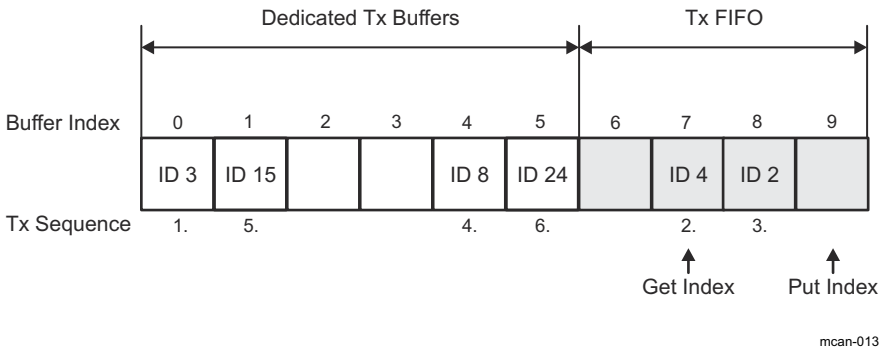


Figure 13-179. Mixed Dedicated Tx Buffers/Tx FIFO (example)

13.4.1.4.8.6 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

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 13-180 shows Mixed Dedicated Tx Buffers/Tx Queue example.

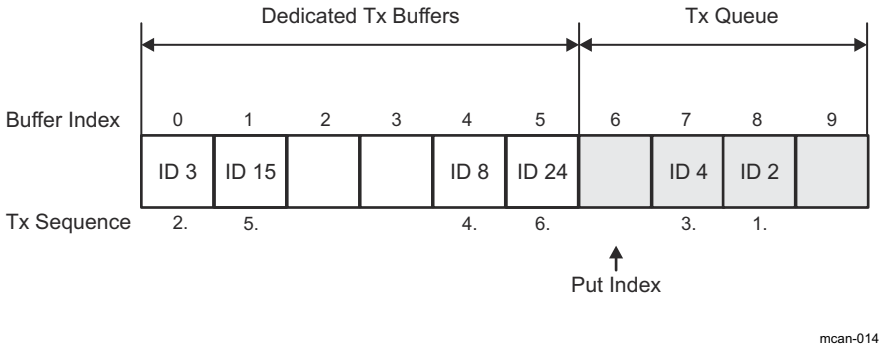


Figure 13-180. Mixed Dedicated Tx Buffers/Tx Queue (example)

13.4.1.4.8.7 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.

13.4.1.4.8.8 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 13.4.1.4.10.4 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 MCAN_TXEFS[12-8] EFGI field has to be added to the Tx Event FIFO start address MCAN_TXEFC[15-2] EFSA field.

13.4.1.4.9 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.

13.4.1.4.10 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

13.4.1.4.10.1 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 following table presents the Message RAM Address Range for all device MCAN instances.

Table 13-218. Message RAM Address Range

Module Instance	Region Name	Address Range		Size
		Start	End	
MCAN0	MCAN0_MSGMEM_RAM	5260 8204h	5261 0203h	32 KB
MCAN1	MCAN1_MSGMEM_RAM	5261 8204h	5262 0203h	32 KB
MCAN2	MCAN2_MSGMEM_RAM	5262 8204h	5263 0203h	32 KB
MCAN3	MCAN3_MSGMEM_RAM	5263 8204h	5264 0203h	32 KB
MCAN4	MCAN4_MSGMEM_RAM	5264 8204h	5265 0203h	32 KB
MCAN5	MCAN5_MSGMEM_RAM	5265 8204h	5266 0203h	32 KB
MCAN6	MCAN6_MSGMEM_RAM	5266 8204h	5267 0203h	32 KB
MCAN7	MCAN7_MSGMEM_RAM	5267 8204h	5268 0203h	32 KB

The Message RAM is capable to include each of the sections listed in Figure 13-181. It is not necessary to configure each of the sections (a section in the Message RAM can be 0) and there is no restriction in regards to the sequence of the sections. For parity checking or ECC, the respective number of bits must to be added to each word.

The MCAN module addresses 32-bit words when addressing the Message RAM. The start addresses are configurable and 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

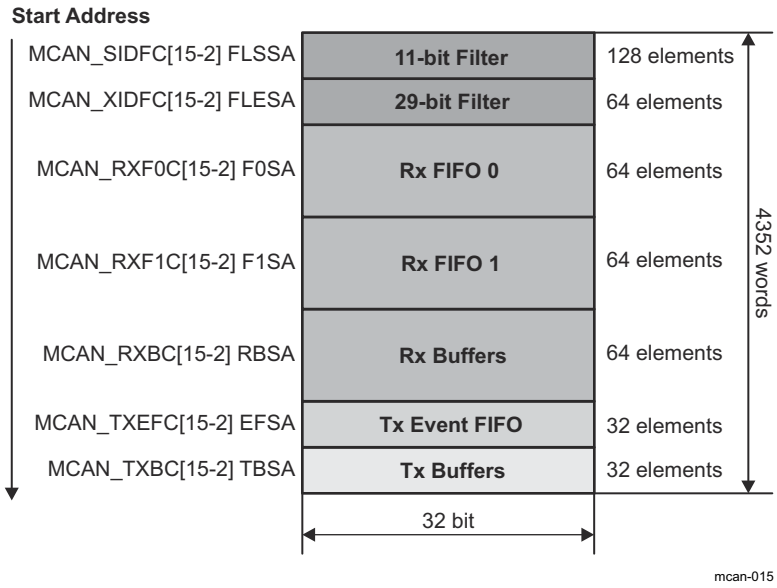


Figure 13-181. Message RAM Configuration

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 to prevent falsification or loss of data.

13.4.1.4.10.2 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 13-182 shows Rx Buffer/Rx FIFO element structure.

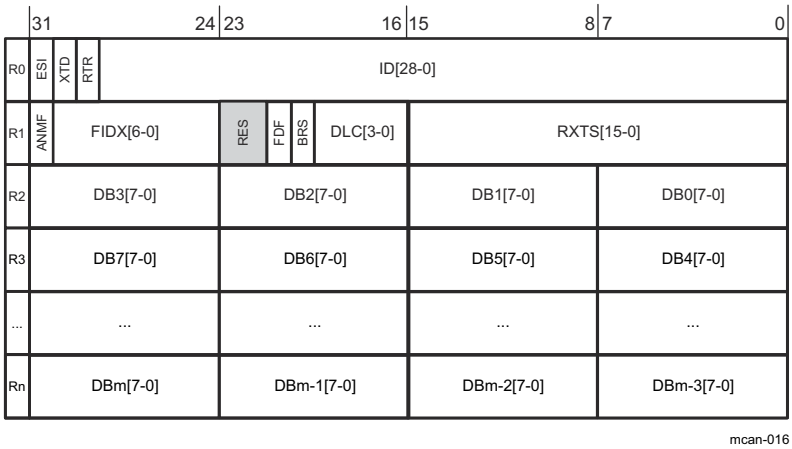


Figure 13-182. Rx Buffer/Rx FIFO Element Structure

Table 13-219 shows Rx Buffer/Rx FIFO element field descriptions.

Table 13-219. 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].
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.

Table 13-219. Rx Buffer/Rx FIFO Element Field Descriptions (continued)

Word	Bits	Field Name	Description
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.

13.4.1.4.10.3 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 13-183 shows Tx Buffer element structure.

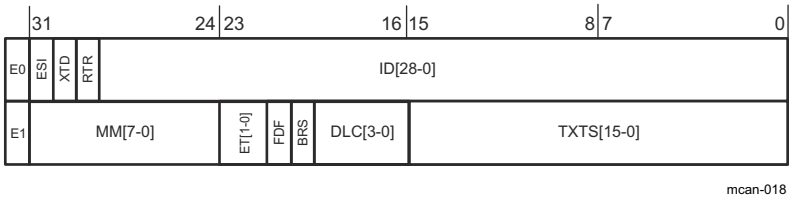


Figure 13-183. Tx Buffer Element Structure

Table 13-220 shows Tx Buffer element field descriptions.

Table 13-220. 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].

Table 13-220. 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 13-221).
	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.

13.4.1.4.10.4 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 13-184 shows Tx Event FIFO element structure.

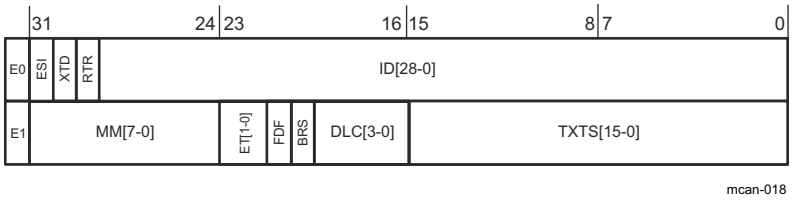


Figure 13-184. Tx Event FIFO Element Structure

Table 13-221 shows Tx Event FIFO element field descriptions.

Table 13-221. 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].

Table 13-221. 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 13-220).
	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.

13.4.1.4.10.5 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 13-185 shows Standard Message ID Filter element structure.

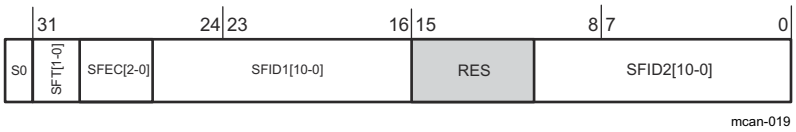


Figure 13-185. Standard Message ID Filter Element Structure

Table 13-222 shows Standard Message ID Filter element field descriptions.

Table 13-222. 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.

13.4.1.4.10.6 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 13-186 shows Extended Message ID Filter element structure.

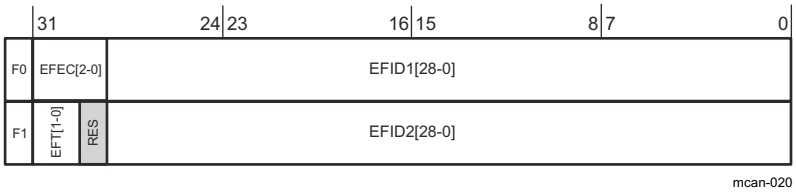


Figure 13-186. Extended Message ID Filter Element Structure

Table 13-223 shows Extended Message ID Filter element field descriptions.

Table 13-223. 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 13.4.1.4.7.1.5, *Extended Message ID Filtering*) is used.

Table 13-223. 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.

13.4.1.5 MCAN Programming Guide

Driver Information

Driver features are available at the MCAN driver page

Software API Information

The MCAN driver provides an API to configure the MCAN module. Full documentation is located on APIs for MCAN

Example Usage

The below links shows an example on how to use MCAN

- MCAN:
  - MCAN External Read/Write
  - MCAN Loopback (Interrupt-based)
  - MCAN Loopback (Polling-based)

13.4.2 Local Interconnect Network (LIN)

This chapter describes the local interconnect network (LIN) module. Since this module can also operate like a conventional serial communications interface (SCI) port, this module is referred to as the SCI/LIN module in this document. In SCI compatibility mode, this module is functionally compatible to the standalone SCI module. However, since the SCI/LIN module uses a different register/bit structure, code written for this module cannot be directly ported to the standalone SCI module and conversely.

This module can be configured to operate in either SCI (UART) or LIN mode.

13.4.2.1 Introduction

The SCI/LIN is compliant to the LIN 2.1 protocol specified in the *LIN Specification Package*.The SCI/LIN module can be programmed to work either as an SCI or as a LIN. The SCI hardware features are augmented to achieve LIN compatibility.

The SCI module is a universal asynchronous receiver-transmitter (UART) that implements the standard non-return to zero format.

The LIN standard is based on the SCI (UART) serial data link format. The communication concept is single-commander/multiple-responder with a message identification for multicast transmission between any network nodes.

Throughout the chapter, compatibility mode refers to SCI mode functionality of the SCI/LIN module. Section 13.4.2.4 explains about the SCI functionality and Section 13.4.2.5 explains about the LIN functionality. Though the registers are common for LIN and SCI, the register descriptions has notes to identify the register and bit usage in different modes.

13.4.2.1.1 SCI Features

The following are the features of the SCI module:

- Standard universal asynchronous receiver-transmitter (UART) communication
- Supports full- or half-duplex operation
- Standard non-return to zero (NRZ) format
- Double-buffered receive and transmit functions in compatibility mode
- Supports two individually enabled interrupt lines: level 0 and level 1
- Configurable frame format of 3 to 13 bits per character based on the following:
  - Data word length programmable from one to eight bits
  - Additional address bit in address-bit mode
  - Parity programmable for zero or one parity bit, odd or even parity
  - Stop programmable for one or two stop bits
- Asynchronous communication mode
- Two multiprocessor communication formats allow communication between more than two devices
- Sleep mode is available to free CPU resources during multiprocessor communication and then wake up to receive an incoming message
- The 24-bit programmable baud rate supports 2^{24} different baud rates provide high accuracy baud rate selection
- At 100MHz peripheral clock, 3.125Mbits/s is the maximum baud rate achievable
- Capability to use direct memory access (DMA) for transmit and receive data
- Five error flags and seven status flags provide detailed information regarding SCI events
- Two external pins: LINRX and LINTX
- Multibuffered receive and transmit units

Note

The SCI/LIN module is functionally compatible with the C2000™ SCI modules, but not directly software compatible due to different register control structures.

The SCI/LIN module does not support UART hardware flow control. This feature can be implemented in software using a general-purpose I/O pin.

The SCI/LIN module does not support isosynchronous mode as there is no SCICLK pin.

13.4.2.1.2 LIN Features

The following are the features of the LIN module:

- Compatibility with LIN 1.3 , 2.0, and 2.1 protocols
- Configurable baud rate up to 20 kbps
- Two external pins: LINRX and LINTX.
- Multibuffered receive and transmit units
- Identification masks for message filtering
- Automatic commander header generation
  - Programmable synchronization break field
  - Synchronization field
  - Identifier field
- Responder Automatic Synchronization
  - Synchronization break detection
  - Optional baud rate update
  - Synchronization validation
- 2^{31} programmable transmission rates with 7 fractional bits
- Wakeup on LINRX dominant level from transceiver
- Automatic wakeup support
  - Wakeup signal generation
  - Expiration times on wakeup signals
- Automatic bus idle detection
- Error detection
  - Bit error
  - Bus error
  - No-response error
  - Checksum error
  - Synchronization field error
  - Parity error
- Capability to use Direct Memory Access (DMA) for transmit and receive data.
- 2 interrupt lines with priority encoding for:
  - Receive
  - Transmit
  - ID, error, and status
- Support for LIN 2.0 checksum
- Enhanced synchronizer finite state machine (FSM) support for frame processing
- Enhanced handling of extended frames
- Enhanced baud rate generator
- Update wakeup/go to sleep

13.4.2.1.3 Block Diagram

The SCI/LIN module contains the core SCI block with added sub-blocks to support LIN protocol.

The three major components of the SCI Module are:

- **Transmitter** (TX) contains two major registers to perform the double-buffering:
  - The transmitter data buffer register (SCITD) contains data loaded by the CPU to be transferred to the shift register for transmission.
  - The transmitter shift register (SCITXSHF) loads data from the data buffer (SCITD) and shifts data onto the LINTX pin, one bit at a time.
- **Baud Clock Generator**
  - A programmable baud generator produces a baud clock scaled from the input clock VCLK
  - LIN VCLK is based on the SYSCLK frequency. VCLK input from SYSCLK and can be divided by 1, 2, or 4 using the CLK_CFG_REGS PERCLKDIVSEL.LINxCLKDIV field for each LIN module individually. By default, VCLK input is SYSCLK divided by 2
- **Receiver** (RX) contains two major registers to perform the double-buffering:
  - The receiver shift register (SCIRXSHF) shifts data in from the LINRX pin one bit at a time and transfers completed data into the receive data buffer.
  - The receiver data buffer register (SCIRD) contains received data transferred from the receiver shift register

The SCI receiver and transmitter are double-buffered, and each has a separate enable and interrupt bits. The receiver and transmitter can each be operated independently or simultaneously in full duplex mode.

To maintain data integrity, the SCI checks the data the SCI receives for breaks, parity, overrun, and framing errors. The bit rate (baud) is programmable to over 16 million different rates through a 24-bit baud-select register. Figure 13-187 shows the detailed SCI block diagram.

The SCI/LIN module is based on the standalone SCI with the addition of an error detector (parity calculator, checksum calculator, and bit monitor), a mask filter, a synchronizer, and a multibuffered receiver and transmitter. The SCI interface and the baud generator are modified as part of the hardware enhancements for LIN compatibility. Figure 13-188 shows the SCI/LIN block diagram.

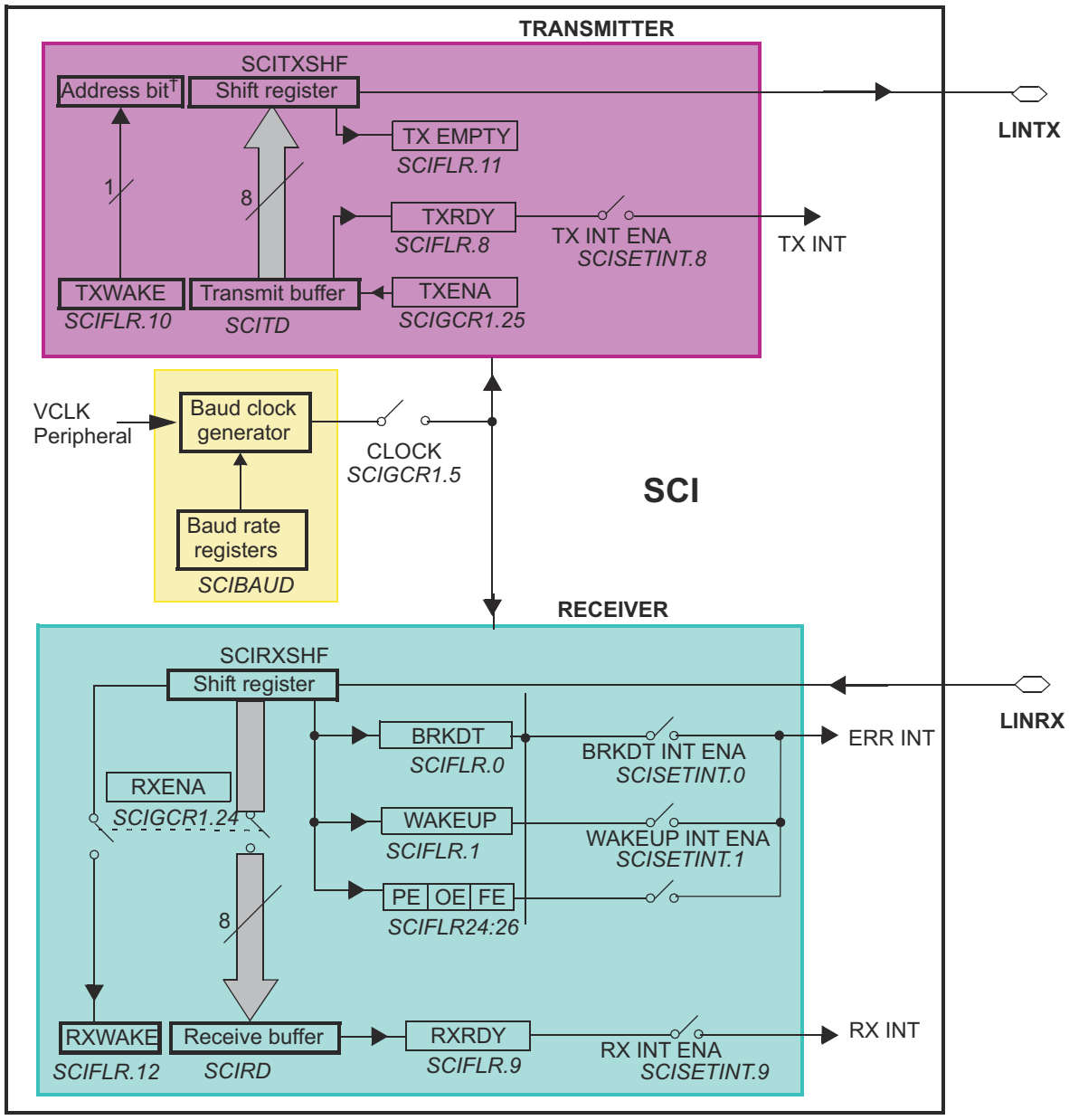


Figure 13-187. SCI Block Diagram

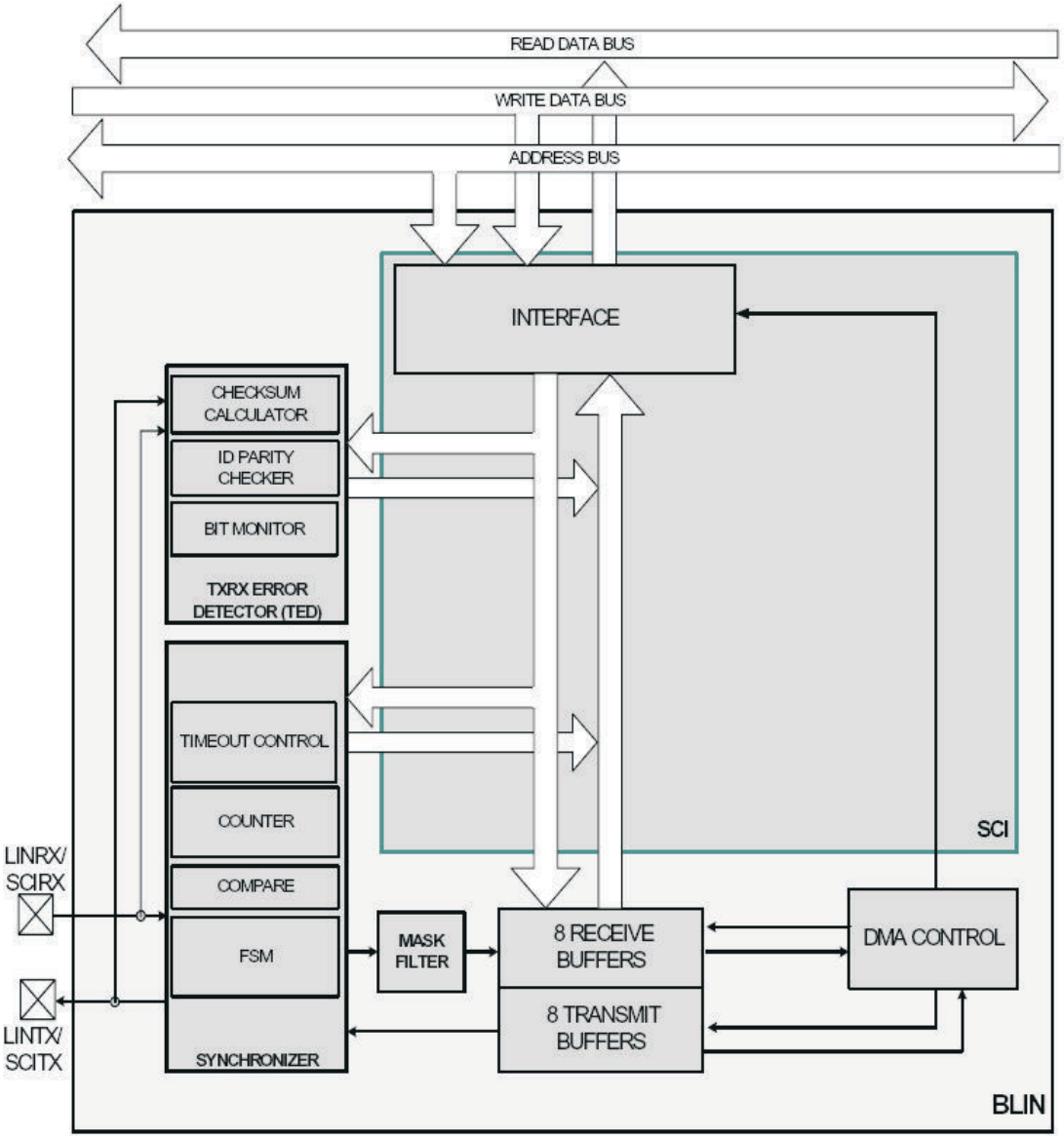


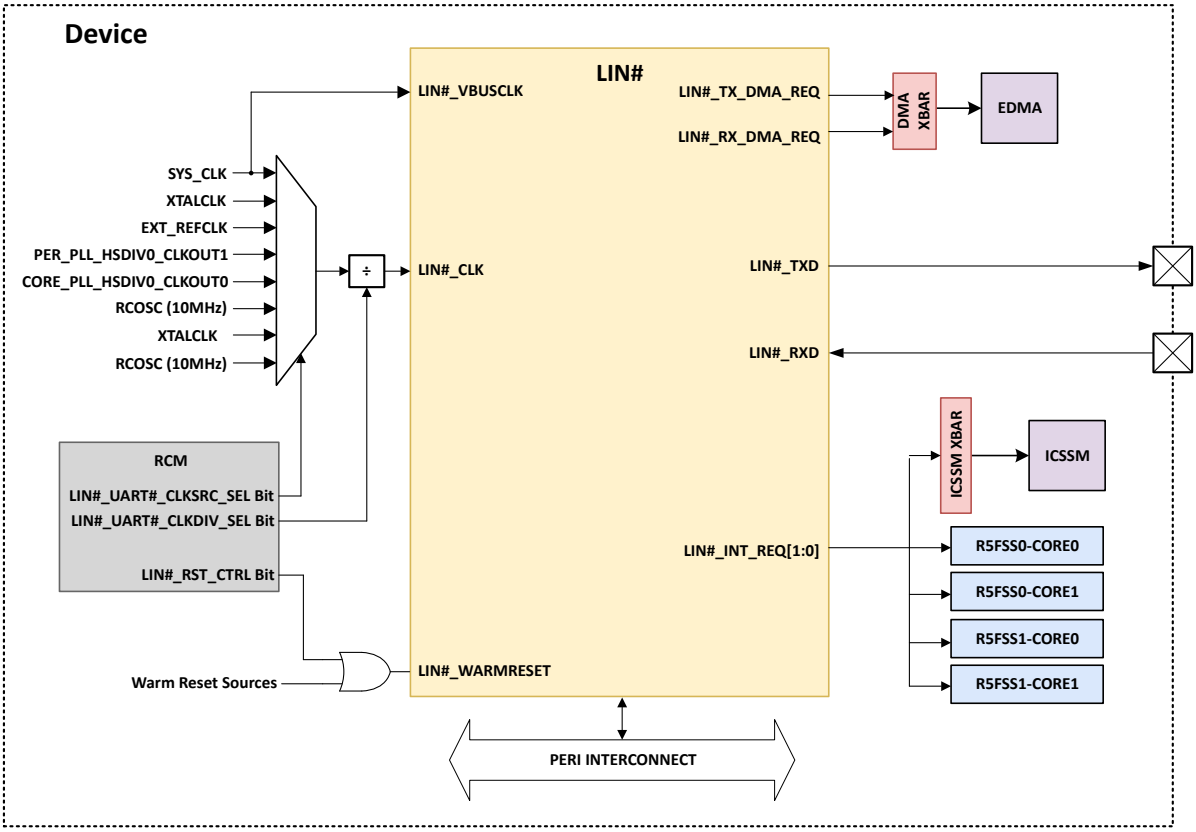
Figure 13-188. SCI/LIN Block Diagram

13.4.2.2 LIN Integration

There are 5x LIN modules integrated in the device. The diagram below provides a visual representation of the device integration details.

= 0 to 4

Figure 13-189. LIN Integration



The tables below summarize the device integration details of LIN# (where # = 5).

Table 13-224. *LIN* Device Integration

This table describes the LIN device integration details.

LIN Instance	Device Allocation	SoC Interconnect
LIN0	✓	Peripheral VBUSP Interconnect
LIN1	✓	Peripheral VBUSP Interconnect
LIN2	✓	Peripheral VBUSP Interconnect
LIN3	✓	Peripheral VBUSP Interconnect
LIN4	✓	Peripheral VBUSP Interconnect

Table 13-225. *LIN* Clocks

This table describes the LIN clocking signals.

LIN Instance	LIN Clock Input	Source Clock Signal	Source	Default Freq	Description
LIN0	LIN0_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	LIN0 Interface Clock (LIN0_CLK should be running for register access)
	LIN0_FCLK (LIN_CLK)	XTALCLK	External XTAL	25 MHz	LIN0 Functional Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK: HSDIV0_CLKOUT0	400 MHz	
		RCCLK10M	Internal 10MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
		DPLL_PER_HSDIV0_CLKOUT0	PLL_PER_CLK:HSDIV0_CLKOUT0	160 MHz	
LIN1	LIN1_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	LIN1 Interface Clock (LIN1_CLK should be running for register access)
	LIN1_FCLK (LIN_CLK)	XTALCLK	External XTAL	25 MHz	LIN1 Functional Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK: HSDIV0_CLKOUT0	400 MHz	
		RCCLK10M	Internal 10MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
		DPLL_PER_HSDIV0_CLKOUT0	PLL_PER_CLK:HSDIV0_CLKOUT0	160 MHz	

Table 13-225. *LIN* Clocks (continued)

This table describes the LIN clocking signals.

LIN Instance	LIN Clock Input	Source Clock Signal	Source	Default Freq	Description
LIN2	LIN2_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	LIN2 Interface Clock (LIN2_CLK should be running for register access)
	LIN2_FCLK (LIN_CLK)	XTALCLK	External XTAL	25 MHz	LIN2 Functional Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK: HSDIV0_CLKOUT0	400 MHz	
		RCCLK10M	Internal 10MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
		DPLL_PER_HSDIV0_CLKOUT0	PLL_PER_CLK:HSDIV0_CLKOUT0	160 MHz	
LIN3	LIN3_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	LIN3 Interface Clock (LIN3_CLK should be running for register access)
	LIN3_FCLK (LIN_CLK)	XTALCLK	External XTAL	25 MHz	LIN3 Functional Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK: HSDIV0_CLKOUT0	400 MHz	
		RCCLK10M	Internal 10MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
		DPLL_PER_HSDIV0_CLKOUT0	PLL_PER_CLK:HSDIV0_CLKOUT0	160 MHz	

Table 13-225. *LIN* Clocks (continued)

This table describes the LIN clocking signals.

LIN Instance	LIN Clock Input	Source Clock Signal	Source	Default Freq	Description
LIN4	LIN4_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	LIN4 Interface Clock (LIN4_CLK should be running for register access)
	LIN4_FCLK (LIN_CLK)	XTALCLK	External XTAL	25 MHz	LIN4 Functional Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT0	PLL_CORE_CLK: HSDIV0_CLKOUT0	400 MHz	
		RCCLK10M	Internal 10MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
		DPLL_PER_HSDIV0_CLKOUT0	PLL_PER_CLK:HSDIV0_CLKOUT0	160 MHz	

Table 13-226. *LIN* Resets

This table describes the LIN reset signals.

LIN Instance	LIN Reset Input	Source Reset Signal	Source	Description
LIN0	LIN0_RST (VBUSP_RSTn)	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	LIN0 Asynchronous Reset
LIN1	LIN1_RST (VBUSP_RSTn)	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	LIN1 Asynchronous Reset
LIN2	LIN2_RST (VBUSP_RSTn)	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	LIN2 Asynchronous Reset
LIN3	LIN3_RST (VBUSP_RSTn)	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	LIN3 Asynchronous Reset
LIN4	LIN4_RST (VBUSP_RSTn)	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	LIN4 Asynchronous Reset

Table 13-227. *LIN* Interrupt Requests

This table describes the LIN interrupt requests.

LIN Instance	LIN Interrupt Signal	Destination Interrupt Input	Destination	Type	Description
LIN0	LIN0_INT_req_0	LIN0_INT_req_0	ALL R5FSS Cores, ICSSMXBAR	Pulse	LIN0 Event Interrupts
	LIN0_INT_req_1	LIN0_INT_req_1			
LIN1	LIN1_INT_req_0	LIN1_INT_req_0	ALL R5FSS Cores, ICSSMXBAR	Pulse	LIN1 Event Interrupts
	LIN1_INT_req_1	LIN1_INT_req_1			
LIN2	LIN2_INT_req_0	LIN2_INT_req_0	ALL R5FSS Cores, ICSSMXBAR	Pulse	LIN2 Event Interrupts
	LIN2_INT_req_1	LIN2_INT_req_1			
LIN3	LIN3_INT_req_0	LIN3_INT_req_0	ALL R5FSS Cores, ICSSMXBAR	Pulse	LIN3 Event Interrupts
	LIN3_INT_req_1	LIN3_INT_req_1			

Table 13-227. *LIN* Interrupt Requests (continued)

This table describes the LIN interrupt requests.

LIN Instance	LIN Interrupt Signal	Destination Interrupt Input	Destination	Type	Description
LIN4	LIN4_INT_req_0	LIN4_INT_req_0	ALL R5FSS Cores, ICSSMXBAR	Pulse	LIN4 Event Interrupts
	LIN4_INT_req_1	LIN4_INT_req_1			

Table 13-228. *LIN* DMA Requests

This table describes the LIN DMA requests.

LIN Instance	LIN DMA Event	Destination DMA Event Input	Destination	Type	Description
LIN0	LIN0_TX_DMA_REQ	LIN0_tx_dma_req	EDMA Crossbar (DMA_XBAR)	Pulse	LIN0 TX DMA Request
	LIN0_RX_DMA_REQ	LIN0_rx_dma_req			LIN0 RX DMA Request
LIN1	LIN1_TX_DMA_REQ	LIN1_tx_dma_req	EDMA Crossbar (DMA_XBAR)	Pulse	LIN1 TX DMA Request
	LIN1_RX_DMA_REQ	LIN1_rx_dma_req			LIN1 RX DMA Request
LIN2	LIN2_TX_DMA_REQ	LIN2_tx_dma_req	EDMA Crossbar (DMA_XBAR)	Pulse	LIN2 TX DMA Request
	LIN2_RX_DMA_REQ	LIN2_rx_dma_req			LIN2 RX DMA Request
LIN3	LIN3_TX_DMA_REQ	LIN3_tx_dma_req	EDMA Crossbar (DMA_XBAR)	Pulse	LIN3 TX DMA Request
	LIN3_RX_DMA_REQ	LIN3_rx_dma_req			LIN3 RX DMA Request
LIN4	LIN4_TX_DMA_REQ	LIN4_tx_dma_req	EDMA Crossbar (DMA_XBAR)	Pulse	LIN4 TX DMA Request
	LIN4_RX_DMA_REQ	LIN4_rx_dma_req			LIN4 RX DMA Request

Note

For more information on the interconnects, see the *System Interconnect* chapter.

For more information on power, reset, and clock management, see the corresponding sections within the *Device Configuration* chapter.

For more information on the device interrupt controllers, see the *Interrupt Controllers* chapter.

13.4.2.3 LIN Integration

This section describes modules integration in the device, including information about clocks, resets, and hardware requests.

13.4.2.4 Serial Communications Interface Module

13.4.2.4.1 SCI Communication Formats

The SCI module can be configured to meet the requirements of many applications. Because communication formats vary depending on the specific application, many attributes of the SCI/LIN are user configurable. The configuration options are:

- SCI Frame format
- SCI Timing modes
- SCI Baud rate
- SCI Multiprocessor modes

13.4.2.4.1.1 SCI Frame Formats

The SCI uses a programmable frame format. All frames consist of the following:

- One start bit
- One to eight data bits
- Zero or one address bit
- Zero or one parity bit
- One or two stop bits

The frame format for both the transmitter and receiver is programmable through the bits in the SCIGCR1 register. Both receive and transmit data is in nonreturn to zero (NRZ) format, which means that the transmit and receive lines are at logic high when idle. Each frame transmission begins with a start bit, in which the transmitter pulls the SCI line low (logic low). Following the start bit, the frame data is sent and received least significant bit first (LSB).

An address bit is present in each frame if the SCI is configured to be in address-bit mode but is not present in any frame if the SCI is configured for idle-line mode. The format of frames with and without the address bit is illustrated in Figure 13-190.

A parity bit is present in every frame when the PARITY ENA bit is set. The value of the parity bit depends on the number of one bits in the frame and whether odd or even parity has been selected using the PARITY ENA bit. Both examples in Figure 13-190 have parity enabled.

All frames include one stop bit, which is always a high level. This high level at the end of each frame is used to indicate the end of a frame to make sure synchronization between communicating devices. Two stop bits are transmitted, if the STOP bit in SCIGCR1 register is set. The examples shown in Figure 13-190 use one stop bit per frame.

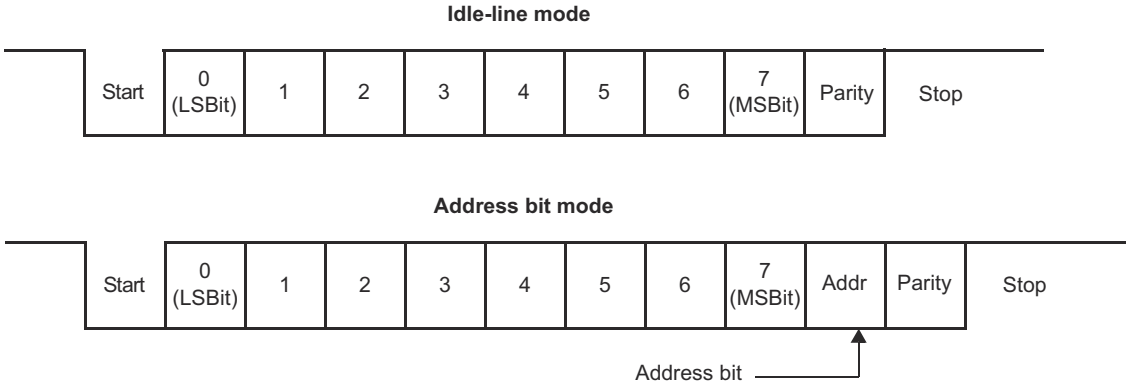


Figure 13-190. Typical SCI Data Frame Formats

13.4.2.4.1.2 SCI Asynchronous Timing Mode

The SCI can be configured to use the asynchronous timing mode using TIMING MODE bit in SCIGCR1 register.

The asynchronous timing mode uses only the receive and transmit data lines to interface with devices using the standard universal asynchronous receiver- transmitter (UART) protocol.

In the asynchronous timing mode, each bit in a frame has a duration of 16 SCI baud clock periods. Each bit therefore consists of 16 samples (one for each clock period). When the SCI is using asynchronous mode, the baud rates of all communicating devices must match as closely as possible. Receive errors result from devices communicating at different baud rates.

With the receiver in the asynchronous timing mode, the SCI detects a valid start bit if the first four samples after a falling edge on the LINRX pin are of logic level 0. As soon as a falling edge is detected on LINRX, the SCI assumes that a frame is being received and synchronizes to the bus.

To prevent interpreting noise as Start bit SCI expects LINRX line to be low for at least four contiguous SCI baud clock periods to detect a valid start bit. The bus is considered idle if this condition is not met. When a valid start bit is detected, the SCI determines the value of each bit by sampling the LINRX line value during the seventh, eighth, and ninth SCI baud clock periods. A majority vote of these three samples is used to determine the value stored in the SCI receiver shift register. By sampling in the middle of the bit, the SCI reduces errors caused by propagation delays and rise and fall times and data line noises. Figure 13-191 illustrates how the receiver samples a start bit and a data bit in asynchronous timing mode.

The transmitter transmits each bit for a duration of 16 SCI baud clock periods. During the first clock period for a bit, the transmitter shifts the value of that bit onto the LINTX pin. The transmitter then holds the current bit value on LINTX for 16 SCI baud clock periods.

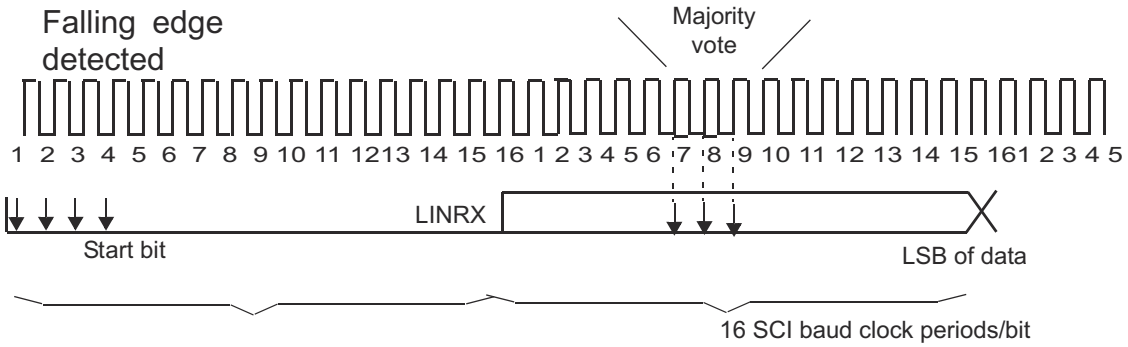


Figure 13-191. Asynchronous Communication Bit Timing

13.4.2.4.1.3 SCI Baud Rate

The SCI/LIN has an internally generated serial clock determined by the peripheral VCLK and the prescalers P and M in this register. The SCI uses the 24-bit integer prescaler P value in the BRS register to select the required baud rates. The additional 4-bit fractional divider M refines the baud rate selection.

In asynchronous timing mode, the SCI generates a baud clock according to the following formula:

$$SCICLK\ Frequency = \frac{VCLK\ Frequency}{P + 1 + \frac{M}{16}}$$

$$Asynchronous\ baud\ value = \frac{SCICLK\ Frequency}{16}$$

For P = 0,

$$Asynchronous\ baud\ value = \frac{VCLK\ Frequency}{32}$$

13.4.2.4.1.4 SCI Multiprocessor Communication Modes

In some applications, the SCI can be connected to more than one serial communication device. In such a multiprocessor configuration, several frames of data can be sent to all connected devices or to an individual device. In the case of data sent to an individual device, the receiving devices must determine when the devices are being addressed. When a message is not intended for them, the devices can ignore the following data. When only two devices make up the SCI network, addressing is not needed, so multiprocessor communication schemes are not required.

SCI supports two multiprocessor communication modes which can be selected using COMM MODE bit:

- Idle-Line Mode
- Address Bit Mode

When the SCI is not used in a multiprocessor environment, software can consider all frames as data frames. In this case, the only distinction between the idle-line and address-bit modes is the presence of an extra bit (the address bit) in each frame sent with the address-bit protocol.

The SCI allows full-duplex communication where data can be sent and received using the transmit and receive pins simultaneously. However, the protocol used by the SCI assumes that only one device transmits data on the same bus line at any one time. No arbitration is done by the SCI.

13.4.2.4.1.4.1 Idle-Line Multiprocessor Modes

In idle-line multiprocessor mode, a frame that is preceded by an idle period (10 or more idle bits) is an address frame. A frame that is preceded by fewer than 10 idle bits is a data frame. Figure 13-192 illustrates the format of several blocks and frames with idle-line mode.

There are two ways to transmit an address frame using idle-line mode:

Method 1: In software, deliberately leave an idle period between the transmission of the last data frame of the previous block and the address frame of the new block.

Method 2: Configure the SCI to automatically send an idle period between the last data frame of the previous block and the address frame of the new block.

Although Method 1 is only accomplished by a delay loop in software, Method 2 can be implemented by using the transmit buffer and the TXWAKE bit in the following manner:

Step 1: Write a 1 to the TXWAKE bit.

Step 2: Write a dummy data value to the SCITD register. This triggers the SCI to begin the idle period as soon as the transmitter shift register is empty.

Step 3: Wait for the SCI to clear the TXWAKE flag.

Step 4: Write the address value to SCITD.

As indicated by Step 3, software can wait for the SCI to clear the TXWAKE bit. However, the SCI clears the TXWAKE bit at the same time the SCI sets TXRDY (that is, transfers data from SCITD into SCITXSHF). Therefore, if the TX INT ENA bit is set, the transfer of data from SCITD to SCITXSHF causes an interrupt to be generated at the same time that the SCI clears the TXWAKE bit. If this interrupt method is used, software is not required to poll the TXWAKE bit waiting for the SCI to clear the bit.

When idle-line multiprocessor communications are used, software must make sure that the idle time exceeds 10 bit periods before addresses (using one of the methods mentioned above), and software must also make sure that data frames are written to the transmitter quickly enough to be sent without a delay of 10 bit periods between frames. Failure to comply with these conditions results in data interpretation errors by other devices receiving the transmission.

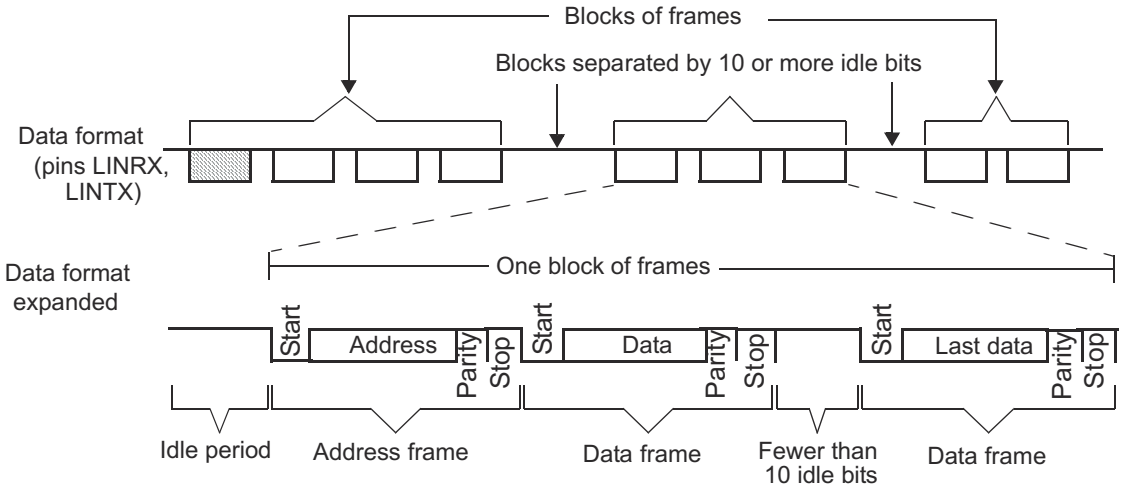


Figure 13-192. Idle-Line Multiprocessor Communication Format

13.4.2.4.1.4.2 Address-Bit Multiprocessor Mode

In the address-bit protocol, each frame has an extra bit immediately following the data field called an address bit. A frame with the address bit set to 1 is an address frame; a frame with the address bit set to 0 is a data frame. The idle period timing is irrelevant in this mode. Figure 13-193 illustrates the format of several blocks and frames with the address-bit mode.

When address-bit mode is used, the value of the TXWAKE bit is the value sent as the address bit. To send an address frame, software must set the TXWAKE bit. This bit is cleared as the contents of the SCITD are shifted from the TXWAKE register so that all frames sent are data except when the TXWAKE bit is written as a 1.

No dummy write to SCITD is required before an address frame is sent in address-bit mode. The first byte written to SCITD after the TXWAKE bit is written to 1 is transmitted with the address bit set when address-bit mode is used.

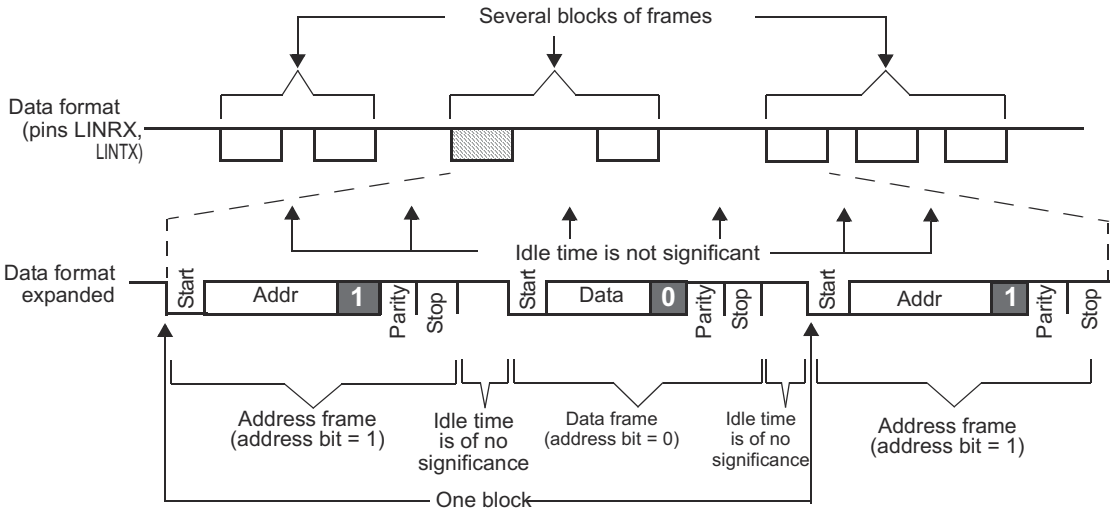


Figure 13-193. Address-Bit Multiprocessor Communication Format

13.4.2.4.1.5 SCI Multibuffered Mode

To reduce CPU load when receiving or transmitting data in interrupt mode or DMA mode, the SCI/LIN module has eight separate receive and transmit buffers. Multibuffered mode is enabled by setting the MBUF MODE bit.

The multibuffer 3-bit counter counts the data bytes transferred from the SCIRXSHF register to the RDy receive buffers and TDy transmit buffers register to SCITXSHF register. The 3-bit compare register contains the number of data bytes expected to be received or transmitted. the LENGTH value in SCIFORMAT register indicates the expected length and is used to load the 3-bit compare register.

A receive interrupt (RX interrupt; see the SCIINTVECT0 and SCIINTVECT1 registers), and a receive ready RXRDY flag set in SCIFLR register, as well as a DMA request (RXDMA) can occur after receiving a response if there are no response receive errors for the frame (such as, there is, frame error, and overrun error).

A transmit interrupt (TX interrupt), and a transmit ready flag (TXRDY flag in SCIFLR register), and a DMA request (TXDMA) can occur after transmitting a response.

Figure 13-194 and Figure 13-195 show the receive and transmit multibuffer functional block diagram, respectively.

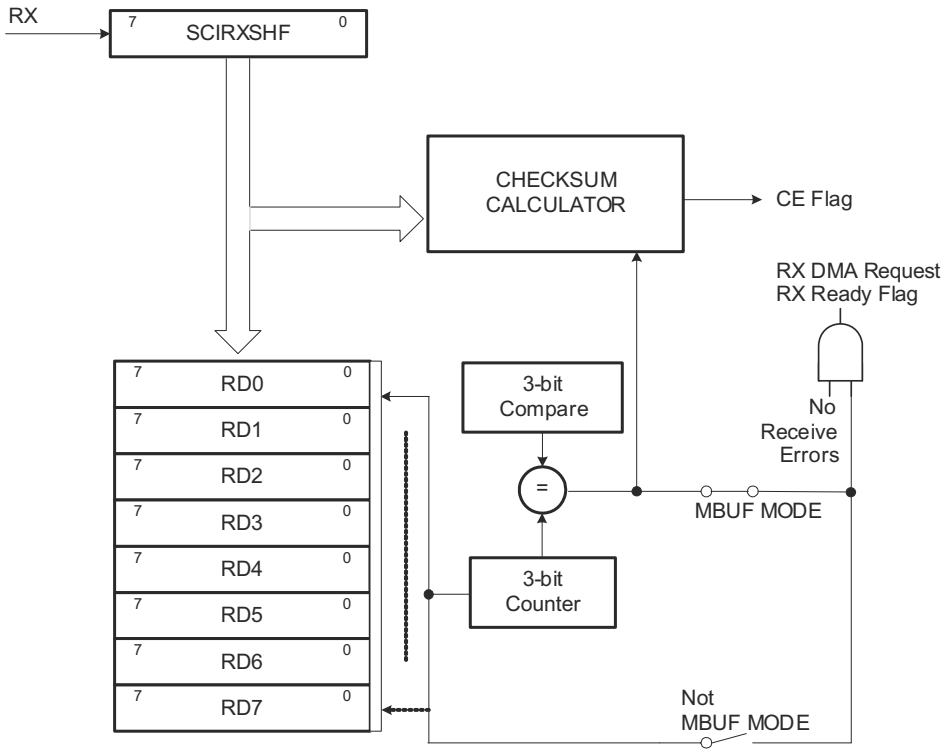


Figure 13-194. Receive Buffers

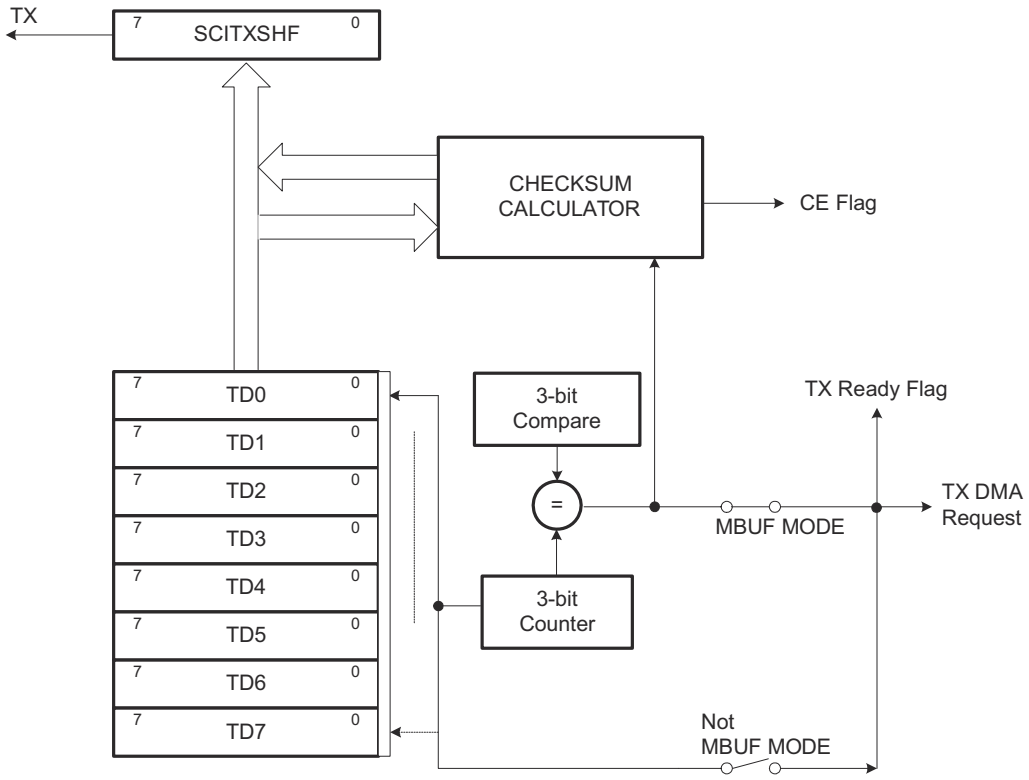


Figure 13-195. Transmit Buffers

13.4.2.4.2 SCI Interrupts

The SCI/LIN module has two interrupt lines, level 0 and level 1, to the vectored interrupt manager (VIM) module (see Figure 13-196). Two offset registers SCIINTVECT0 and SCIINTVECT1 determine which flag triggered the interrupt according to the respective priority encoders. Each interrupt condition has a bit to enable/disable the interrupt in the SCISETINT and SCICLRINT registers, respectively.

Each interrupt also has a bit that can be set as interrupt level 0(INT0) or as interrupt level 1(INT1). By default, interrupts are in interrupt level 0. SCISETINTLVL sets a given interrupt to level1. SCICLEARINTLVL resets a given interrupt level to the default level 0.

The interrupt vector registers SCIINTVECT0 and SCIINTVECT1 return the vector of the pending interrupt line INT0 or INT1. If more than one interrupt is pending, the interrupt vector register holds the highest priority interrupt.

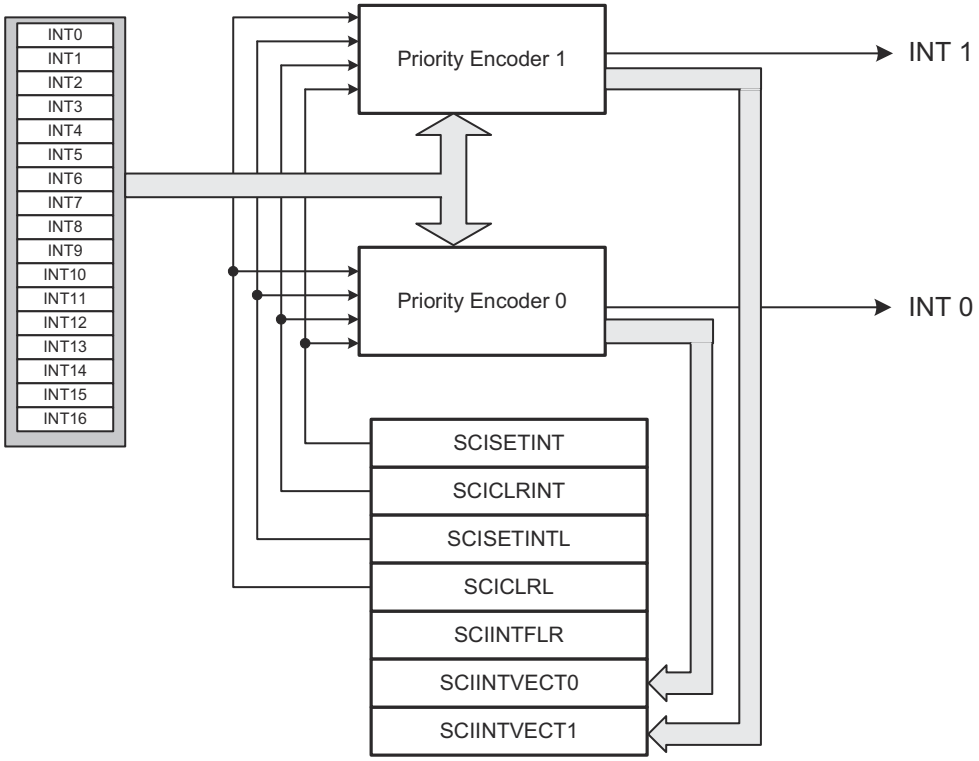


Figure 13-196. General Interrupt Scheme

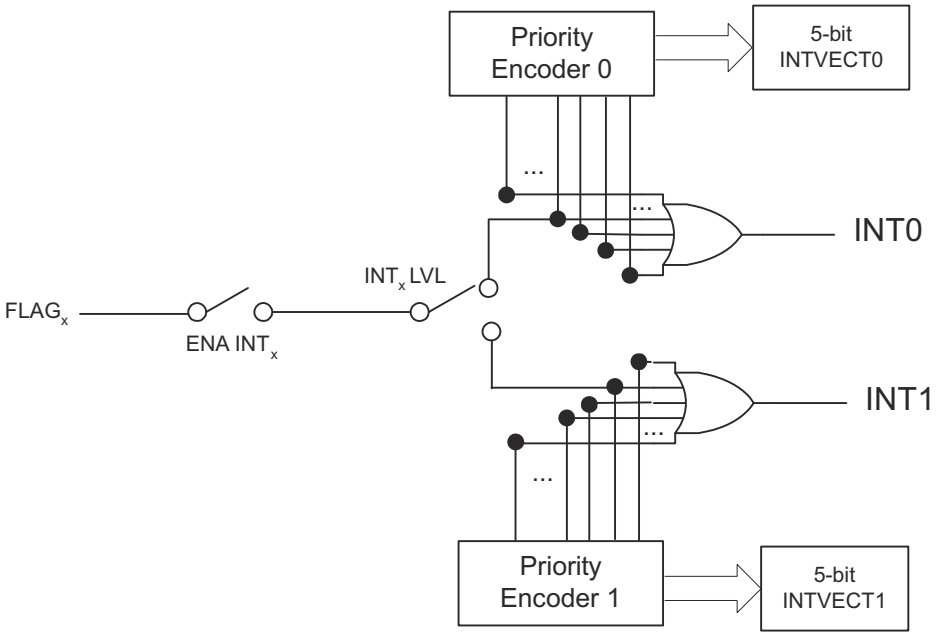


Figure 13-197. Interrupt Generation for Given Flags

13.4.2.4.2.1 Transmit Interrupt

To use transmit interrupt functionality, SET TX INT bit must be enabled and SET TX DMA bit must be cleared. The transmit ready (TXRDY) flag is set when the SCI transfers the contents of SCITD to the shift register, SCITXSHF. The TXRDY flag indicates that SCITD is ready to be loaded with more data. In addition, the SCI sets the TX EMPTY bit if both the SCITD and SCITXSHF registers are empty. If the SET TX INT bit is set, then a transmit interrupt is generated when the TXRDY flag goes high. Transmit Interrupt is not generated immediately after setting the SET TX INT bit unlike transmit DMA request. Transmit Interrupt is generated only after the first transfer from SCITD to SCITXSHF, that is first data has to be written to SCITD before any interrupt gets generated. To transmit further data, data can be written to SCITD in the transmit Interrupt service routine.

Writing data to the SCITD register clears the TXRDY bit. When this data has been moved to the SCITXSHF register, the TXRDY bit is set again. The interrupt request can be suspended by setting the CLR TX INT bit; however, when the SET TX INT bit is again set to 1, the TXRDY interrupt is asserted again. The transmit interrupt request can be eliminated until the next series of values is written to SCITD, by disabling the transmitter using the TXENA bit, by a software reset SWnRST, or by a device hardware reset.

13.4.2.4.2.2 Receive Interrupt

The receive ready (RXRDY) flag is set when the SCI transfers newly received data from SCIRXSHF to SCIRD. The RXRDY flag therefore indicates that the SCI has new data to be read. Receive interrupts are enabled by the SET RX INT bit. If the SET RX INT is set when the SCI sets the RXRDY flag, then a receive interrupt is generated. The received data can be read in the Interrupt Service routine.

On a device with both SCI and a DMA controller, SET RX DMA must be cleared to select interrupt functionality.

13.4.2.4.2.3 WakeUp Interrupt

SCI sets the WAKEUP flag if bus activity on the RX line either prevents power-down mode from being entered, or RX line activity causes an exit from power-down mode. If enabled (SET WAKEUP INT), wakeup interrupt is triggered once WAKEUP flag is set.

13.4.2.4.2.4 Error Interrupts

The following error detections are supported with an interrupt by the SCI module:

- Parity errors (PE)
- Frame errors (FE)
- Break Detect errors (BRKDT)
- Overrun errors (OE)
- Bit errors (BE)

There are 16 interrupt sources in the SCI/LIN module. In SCI mode, 8 interrupts are supported, as listed in Table 13-229.

If all of these errors (PE, FE, BRKDT, OE, BE) are flagged, an interrupt for the flagged errors is generated if enabled. A message is valid for both the transmitter and the receiver, if there is no error detected until the end of the frame. Each of these flags is located in the receiver status (SCIFLR) register (Table 13-230 and Table 13-231).

Table 13-229. SCI/LIN Interrupts

Offset(1)	Interrupt	Applicable to SCI	Applicable to LIN
0	No interrupt	-	-
1	Wakeup	Yes	Yes
2	Inconsistent-sync-field error (ISFE)	No	Yes
3	Parity error (PE)	Yes	Yes
4	ID	No	Yes
5	Physical bus error (PBE)	No	Yes
6	Frame error (FE)	Yes	Yes
7	Break detect (BRKDT)	Yes	No
8	Checksum error (CE)	No	Yes
9	Overrun error (OE)	Yes	Yes
10	Bit error (BE)	Yes	Yes
11	Receive	Yes	Yes
12	Transmit	Yes	Yes
13	No-response error (NRE)	No	Yes
14	Timeout after wakeup signal (150ms)	No	Yes
15	Timeout after three wakeup signals (1.5s)	No	Yes
16	Timeout (Bus Idle, 4s)	No	Yes

(1) Offset 1 is the highest priority. Offset 16 is the lowest priority.

Table 13-230. SCI Receiver Status Flags

SCI Flag	Register	Bit	Value After Reset(1)
CE	SCIFLR	29	0
ISFE	SCIFLR	28	0
NRE	SCIFLR	27	0
FE	SCIFLR	26	0
OE	SCIFLR	25	0
PE	SCIFLR	24	0
RXWAKE	SCIFLR	12	0
RXRDY	SCIFLR	9	0
BUSY	SCIFLR	3	0
IDLE	SCIFLR	2	1
WAKEUP	SCIFLR	1	0
BRKDT	SCIFLR	0	0

(1) The flags are frozen with the reset value while SWnRST = 0.

Table 13-231. SCI Transmitter Status Flags

SCI Flag	Register	Bit	Value After Reset(1)
BE	SCIFLR	31	0
PBE	SCIFLR	30	0
TXWAKE	SCIFLR	10	0
TXEMPTY	SCIFLR	11	1
TXRDY	SCIFLR	8	1

(1) The flags are frozen with the reset value while SWnRST = 0.

13.4.2.4.3 SCI Configurations

Before the SCI sends or receives data, the SCI registers can be properly configured. Upon power-up or a system-level reset, each bit in the SCI registers is set to a default state. The registers are writable only after the RESET bit in the SCIGCR0 register is set to 1. Of particular importance is the SWnRST bit in the SCIGCR1 register. The SWnRST is an active-low bit initialized to 0 and keeps the SCI in a reset state until the bit is programmed to 1. Therefore, all SCI configuration can be completed before a 1 is written to the SWnRST bit.

The following list details the configuration steps that software can perform prior to the transmission or reception of data. As long as the SWnRST bit is cleared to 0 the entire time that the SCI is being configured, the order in which the registers are programmed is not important.

- Enable SCI by setting the RESET bit to 1.
- Clear the SWnRST bit to 0 before SCI is configured.
- Select the desired frame format by programming the SCIGCR1 register.
- Set both the RX FUNC and TX FUNC bits in SCIPIO0 to 1 to configure the LINRX and LINTX pins for SCI functionality.
- Select the baud rate to be used for communication by programming the BRS register.
- Set the CLOCK bit in SCIGCR1 to 1 to select the internal clock.
- Set the CONT bit in SCIGCR1 to 1 to make SCI not halt for an emulation breakpoint until the current reception or transmission is complete (this bit is used only in an emulation environment).
- Set the LOOP BACK bit in SCIGCR1 to 1 to connect the transmitter to the receiver internally (this feature is used to perform a self-test).
- Set the RXENA bit in SCIGCR1 to 1, if data is to be received.
- Set the TXENA bit in SCIGCR1 to 1, if data is to be transmitted.
- Set the SWnRST bit to 1 after SCI is configured.
- Perform receiving or transmitting data (see Section 13.4.2.4.3.1 or Section 13.4.2.4.3.2).

13.4.2.4.3.1 Receiving Data

The SCI receiver is enabled to receive messages, if both the RX FUNC bit and the RXENA bit are set to 1. If the RX FUNC bit is not set, the LINRX pin functions as a general-purpose I/O pin rather than as an SCI function pin.

SCI module can receive data in one of the following modes:
- Single-Buffer (Normal) Mode
- Multibuffer Mode

After a valid idle period is detected, data is automatically received as the data arrives on the LINRX pin.

13.4.2.4.3.1.1 Receiving Data in Single-Buffer Mode

Single-buffer mode is selected when the MBUF MODE bit in SCIGCR1 is cleared to 0. In this mode, SCI sets the RXRDY bit when the SCI transfers newly received data from SCIRXSHF to SCIRD. The SCI clears the RXRDY bit after the new data in SCIRD has been read. Also, as data is transferred from SCIRXSHF to SCIRD, the SCI sets the FE, OE, or PE flags if any of these error conditions were detected in the received data. These error conditions are supported with configurable interrupt capability. The wakeup and break-detect status bits are also set if one of these errors occurs, but the bits do not necessarily occur at the same time that new data is being loaded into SCIRD.

You can receive data by:

1. Polling Receive Ready Flag
2. Receive Interrupt
3. DMA

In polling method, software can poll for the RXRDY bit and read the data from the SCIRD register once the RXRDY bit is set high. The CPU is unnecessarily overloaded by selecting the polling method. To avoid this, you can use either the interrupt or DMA method. To use the interrupt method, the SET RX INT bit is set. To use the DMA method, the SET RX DMA bit is set. Either an interrupt or a DMA request is generated the moment the RXRDY bit is set.

13.4.2.4.3.1.2 Receiving Data in Multibuffer Mode

Multibuffer mode is selected when the MBUFMODE bit in SCIGCR1 is set to 1. In this mode, SCI sets the RXRDY bit after receiving the programmed number of data in the receive buffer, the complete frame. The error condition detection logic is similar to the single-buffer mode, except that this logic monitors for the complete frame. Like single-buffer mode, use the polling, DMA, or interrupt method to read the data. The SCI clears the RXRDY bit after the new data in SCIRD has been read.

13.4.2.4.3.2 Transmitting Data

The SCI transmitter is enabled if both the TX FUNC bit and the TXENA bit are set to 1. If the TX FUNC bit is not set, the LINTX pin functions as a general-purpose I/O pin rather than as an SCI function pin. Any value written to the SCITD before TXENA is set to 1 is not transmitted. Both of these control bits allow for the SCI transmitter to be held inactive independently of the receiver.

SCI module can transmit data in one of the following modes:

- Single-Buffer (Normal) Mode
- Multibuffered or Buffered SCI Mode

13.4.2.4.3.2.1 Transmitting Data in Single-Buffer Mode

Single-buffer mode is selected when the MBUF MODE bit in SCIGCR1 is cleared to 0. In this mode, SCI waits for data to be written to SCITD, transfers the data to SCITXSHF, and transmits the data. The TXRDY and TX EMPTY bits indicate the status of the transmit buffers. That is, when the transmitter is ready for data to be written to SCITD, the TXRDY bit is set. Additionally, if both SCITD and SCITXSHF are empty, then the TX EMPTY bit is also set.

You can transmit data by:

1. Polling Transmit Ready Flag
2. Transmit Interrupt
3. DMA

In polling method, software can poll for the TXRDY bit to go high before writing the data to the SCITD register. The CPU is unnecessarily overloaded by selecting the polling method. To avoid this, you can use the interrupt or DMA method. To use the interrupt method, the SET TX INT bit is set. To use the DMA method, the SET TX DMA bit is set. Either an interrupt or a DMA request is generated the moment the TXRDY bit is set. When the SCI has completed transmission of all pending frames, the SCITXSHF register and SCITD are empty, the TXRDY bit is set, and an interrupt/DMA request is generated, if enabled. Because all data has been transmitted, the interrupt/DMA request must be halted. This can either be done by disabling the transmit interrupt (CLR TX INT) / DMA request (CLR TX DMA bit), or by disabling the transmitter (clear TXENA bit).

Note

The TXRDY flag cannot be cleared by reading the corresponding interrupt offset in the SCIINTVECT0 or SCIINTVECT1 register.

13.4.2.4.3.2.2 Transmitting Data in Multibuffer Mode

Multibuffer mode is selected when the MBUF MODE bit in SCIGCR1 is set to 1. Like single-buffer mode, you can use the polling, DMA, or interrupt method to write the data to be transmitted. The transmitted data has to be written to the SCITD registers. SCI waits for data to be written to the SCITD register and transfers the programmed number of bytes to SCITXSHF to transmit one by one automatically.

13.4.2.4.4 SCI Low-Power Mode

The SCI/LIN can be put in either local or global low-power mode. Global low-power mode is asserted by the system and is not controlled by the SCI/LIN. During global low-power mode, all clocks to the SCI/LIN are turned off so the module is completely inactive.

Local low-power mode is asserted by setting the POWERDOWN bit; setting this bit stops the clocks to the SCI/LIN internal logic and the module registers. Setting the POWERDOWN bit causes the SCI to enter local low-power mode and clearing the POWERDOWN bit causes SCI/LIN to exit from local low-power mode. All the registers are accessible during local power-down mode as any register access enables the clock to SCI for that particular access alone.

The wakeup interrupt is used to allow the SCI to exit low-power mode automatically when a low level is detected on the LINRX pin and also this clears the POWERDOWN bit. If wakeup interrupt is disabled, then the SCI/LIN immediately enters low-power mode whenever it is requested and also any activity on the LINRX pin does not cause the SCI to exit low-power mode.

Note

Enabling Local Low-Power Mode During Receive and Transmit

If the wakeup interrupt is enabled and low-power mode is requested while the receiver is receiving data, then the SCI immediately generates a wakeup interrupt to clear the powerdown bit and prevents the SCI from entering low-power mode and thus completes the current reception. Otherwise, if the wakeup interrupt is disabled, then the SCI completes the current reception and then enters the low-power mode.

13.4.2.4.4.1 Sleep Mode for Multiprocessor Communication

When the SCI receives data and transfers that data from SCIRXSHF to SCIRD, the RXRDY bit is set and if RX INT ENA is set, the SCI also generates an interrupt. The interrupt triggers the CPU to read the newly received frame before another one is received. In multiprocessor communication modes, this default behavior can be enhanced to provide selective indication of new data. When SCI receives an address frame that does not match the address, the device can ignore the data following this non-matching address until the next address frame by using sleep mode. Sleep mode can be used with both idle-line and address-bit multiprocessor modes.

If sleep mode is enabled by the SLEEP bit, then the SCI transfers data from SCIRXSHF to SCIRD only for address frames. Therefore, in sleep mode, all data frames are assembled in the SCIRXSHF register without being shifted into the SCIRD and without initiating a receive interrupt or DMA request. Upon reception of an address frame, the contents of the SCIRXSHF are moved into SCIRD, and the software must read SCIRD and determine if the SCI is being addressed by comparing the received address against the address previously set in the software and stored somewhere in memory (the SCI does not have hardware available for address comparison). If the SCI is being addressed, the software must clear the SLEEP bit so that the SCI loads SCIRD with the data of the data frames that follow the address frame.

When the SCI has been addressed and sleep mode has been disabled (in software) to allow the receipt of data, the SCI can check the RXWAKE bit (SCIFLR.12) to determine when the next address has been received. The bit is set to 1, if the current value in SCIRD is an address; the bit is set to 0, if SCIRD contains data. If the RXWAKE bit is set, then software can check the address in SCIRD against the address. If SCIRD is still being addressed, then sleep mode can remain disabled; otherwise, the SLEEP bit can be set again.

Following is a sequence of events typical of sleep mode operation:

- The SCI is configured and both sleep mode and receive actions are enabled.
- An address frame is received and a receive interrupt is generated.
- Software compares the received address frame against that set by software and determines that the SCI is not being addressed, so the value of the SLEEP bit is not changed.
- Several data frames are shifted into SCIRXSHF, but no data is moved to SCIRD and no receive interrupts are generated.
- A new address frame is received and a receive interrupt is generated.
- Software compares the received address frame against that set by software and determines that the SCI is being addressed and clears the SLEEP bit.
- Data shifted into SCIRXSHF is transferred to SCIRD, and a receive interrupt is generated after each data frame is received.
- In each interrupt routine, software checks RXWAKE to determine if the current frame is an address frame.
- Another address frame is received, RXWAKE is set, software determines that the SCI is not being addressed and sets the SLEEP bit back to 1. No receive interrupts are generated for the data frames following this address frame.

By ignoring data frames that are not intended for the device, fewer interrupts are generated. Otherwise, these interrupts require CPU intervention to read data that is of no significance to this specific device. Using sleep mode can help free some CPU resources.

Except for the RXRDY flag, the SCI continues to update the receiver status flags (see Table 13-230) while sleep mode is active. In this way, if an error occurs on the receive line, an application can immediately respond to the error and take the appropriate corrective action.

Because the RXRDY bit is not updated for data frames when sleep mode is enabled, the SCI can enable sleep mode and use a polling algorithm if desired. In this case, when RXRDY is set, software knows that a new address has been received. If the SCI is not being addressed, then the software can not change the value of the SLEEP bit and can continue to poll RXRDY.

13.4.2.5 Local Interconnect Network Module

13.4.2.5.1 LIN Communication Formats

The SCI/LIN module can be used in LIN mode or SCI mode. The enhancements for baud generation, DMA controls, and additional receive/transmit buffers necessary for LIN mode operation are also part of the enhanced buffered SCI module. LIN mode is selected by enabling LIN MODE bit in SCIGCR1 register.

Note

The SCI/LIN is built around the SCI platform and uses a similar sampling scheme: 16 samples for each bit with majority vote on samples 8, 9, and 10. For the START bit, the first three samples are used.

The SCI/LIN control registers are located at the SCI/LIN base address.

13.4.2.5.1.1 LIN Standards

For compatibility with LIN2.0 standard the following additional features are implemented over LIN1.3:

1. Support for LIN 2.0 checksum
2. Enhanced synchronizer FSM support for frame processing
3. Enhanced handling of extended frames
4. Enhanced baud rate generator
5. Update wakeup/go to sleep

The LIN module covers the CPU performance-consuming features, defined in the *LIN Specification Package* Revision 1.3 and 2.0 by hardware. The Commander Mode of LIN module is compatible with LIN 2.1 standard.

13.4.2.5.1.2 Message Frame

The LIN protocol defines a message frame format, shown in Figure 13-198. Each frame includes one commander header, one response, one in-frame response space, and inter-byte spaces. In-frame-response and inter-byte spaces can be 0.

There is no arbitration in the definition of the LIN protocol; therefore, multiple responder nodes responding to a header can be detected as an error.

The LIN bus is a single-channel wired-AND bus. The bus has a binary level: either dominant for a value of 0 or recessive for a value of 1.

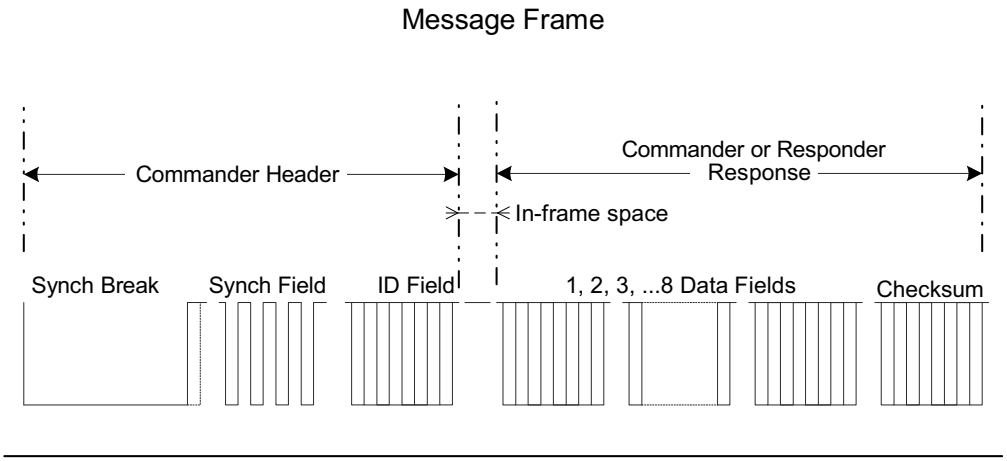


Figure 13-198. LIN Protocol Message Frame Format: Commander Header and Responder Peripheral Response

13.4.2.5.1.2.1 Message Header

The header of a message is initiated by a commander (see Figure 13-199) and consists of a three field-sequence:

- The synchronization break field signaling the beginning of a message
- The synchronization field conveying bit rate information of the LIN bus
- The identification field denoting the content of a message

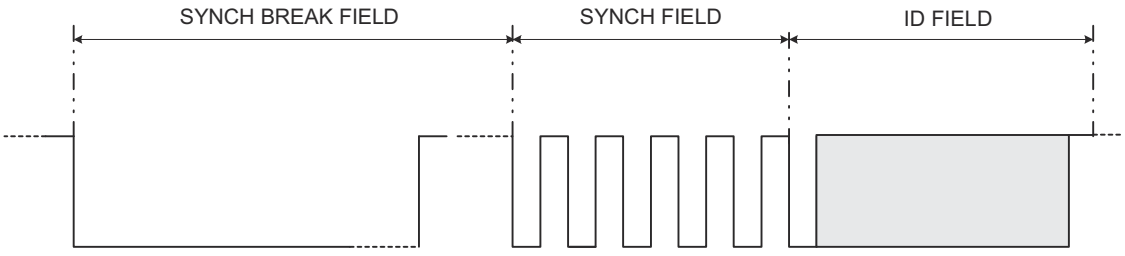


Figure 13-199. Header 3 Fields: Synch Break, Synch, and ID

13.4.2.5.1.2.2 Response

The format of the response is as illustrated in Figure 13-200. There are two types of fields in a response: data and checksum. The data field consists of exactly one data byte, one start bit, and one stop bit, for a total of 10 bits. The LSB is transmitted first. The checksum field consists of one checksum byte, one start bit and one stop bit. The checksum byte is the inverted modulo-256 sum over all data bytes in the data fields of the response.

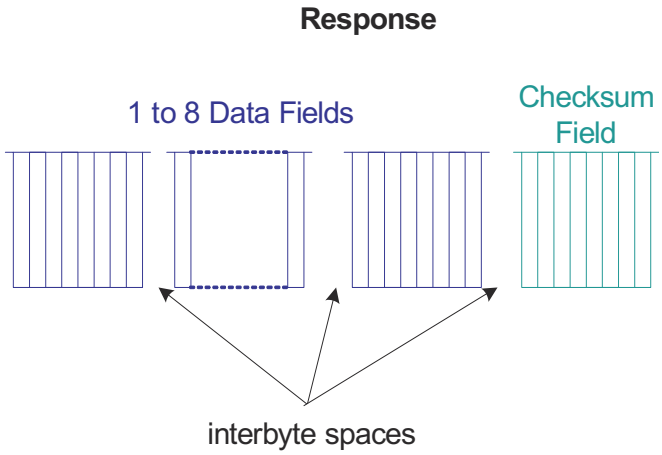


Figure 13-200. Response Format of LIN Message Frame

The format of the response is a stream of N data fields and one checksum field. Typically N is from 1 to 8, with the exception of the extended command frames (Section 13.4.2.5.1.6). The length N of the response is indicated either with the optional length control bits of the ID Field (this is used in standards earlier than LIN 1.x); see Table 13-232, or by LENGTH value in SCIFORMAT[18:16] register; see Table 13-233. The SCI/LIN module supports response lengths from 1 to 8 bytes in compliance with LIN 2.0.

Table 13-232. Response Length Info Using IDBYTE Field Bits [5:4] for LIN Standards Earlier than v1.3

ID5	ID4	Number of Data Bytes
0	0	2
0	1	2
1	0	4
1	1	8

Table 13-233. Response Length with SCIFORMAT[18:16] Programming

SCIFORMAT[18:16]	Number of Bytes
000	1
001	2
010	3
011	4
100	5
101	6
110	7
111	8

13.4.2.5.1.3 Synchronizer

The synchronizer has three major functions in the messaging between commander and responder nodes. It generates the commander header data stream, it synchronizes to the LIN bus for responding, and it locally detects timeouts. A bit rate is programmed using the prescalers in the BRSR register to match the indicated LIN_speed value in the LIN description file.

The LIN synchronizer performs the following functions: commander header signal generation, responder detection and synchronization to message header with optional baud rate adjustment, response transmission timing and timeout control.

The LIN synchronizer is capable of detecting an incoming break and initializing communication at all times.

13.4.2.5.1.4 Baud Rate

The transmission baud rate of any node is configured by the CPU at the beginning; this defines the bit time T_{bit}. The bit time is derived from the fields P and M in the baud rate selection register (BRSR). There is an additional 3-bit fractional divider value, field U in the BRSR register, which further fine-tunes the data-field baud rate.

The ranges for the prescaler values in the BRSR register are:

P = 0, 1, 2, 3, . . . , 2^{24} - 1

M = 0, 1, 2, . . . , 15

U = 0, 1, 2, 3, 4, 5, 6, 7

The P, M, and U values in the BRSR register are user programmable. The P and M dividers can be used for both SCI mode and LIN mode to select a baud rate. The U value is an additional 3-bit value determining that "**a**T_{VCLK}" (with **a** = 0, 1) is added to each T_{bit} as explained in Section 13.4.2.5.1.4.2. If the ADAPT bit is set and the LIN peripheral is in adaptive baud rate mode, then all these divider values are automatically obtained during header reception when the synchronization field is measured.

The LIN protocol defines baud rate boundaries as:

$1kHz \le F_{LINCLK} \le 20kHz$

All transmitted bits are shifted in and out at T_{bit} periods.

13.4.2.5.1.4.1 Fractional Divider

The M field of the BRSR register modifies the integer prescaler P for fine tuning of the baud rate. The M value adds in increments of 1/16 of the P value.

The bit time, T_{bit} is expressed in terms of the VCLK period T_{VCLK} as follows:

For all P other than 0, and all M,

$$T_{bit} = 16\left(P + 1 + \frac{M}{16}\right)T_{VCLK}$$

For P= 0 : $T_{bit} = 32T_{VCLK}$

Therefore, the LINCLK frequency is given by:

$$F_{LINCLK} = \frac{F_{VCLK}}{16(P + 1 + \frac{M}{16})} \quad \text{For all P other than zero}$$

$$F_{LINCLK} = \frac{F_{VCLK}}{32} \quad \text{For } P = 0$$

13.4.2.5.1.4.2 Superfractional Divider

The superfractional divider scheme applies to the following modes:

- LIN commander mode (sync field + identifier field + response field + checksum field)
- LIN responder mode (response field + checksum field)

13.4.2.5.1.4.2.1 Superfractional Divider In LIN Mode

Building on the 4-bit fractional divider M (BRSR[27:24], the superfractional divider uses an additional 3-bit modulating value, illustrated in Table 13-234. The sync field (0x55), the identifier field, and the response field can all be seen as 8-bit data bytes flanked by a start bit and a stop bit. The bits with a 1 in the table have an additional VCLK period added to the T_{bit}. In LIN commander mode, bit modulation applies to sync field + identifier field + response field. In LIN responder mode, bit modulation applies to identifier field + response field.

Table 13-234. Superfractional Bit Modulation for LIN Commander Mode and Responder Mode

BRSR[30:28]	Start Bit	D[0]	D[1]	D[2]	D[3]	D[4]	D[5]	D[6]	D[7]	Stop Bit
0h	0	0	0	0	0	0	0	0	0	0
1h	1	0	0	0	0	0	0	0	1	0
2h	1	0	0	0	1	0	0	0	1	0
3h	1	0	1	0	1	0	0	0	1	0
4h	1	0	1	0	1	0	1	0	1	0
5h	1	1	1	0	1	0	1	0	1	1
6h	1	1	1	0	1	1	1	0	1	1
7h	1	1	1	1	1	1	1	0	1	1

The baud rate varies over a LIN data field to average according to the BRSR[30:28] value by a *d* fraction of the peripheral internal clock: 0<d<1.

The instantaneous bit time is expressed in terms of T_{VCLK} as follows:

For all P other than 0, and all M and d (0 or 1),

$$T^{i}bit = \left[16\left(P + 1 + \frac{M}{16}\right) + d\right]T_{VCLK}$$

For P = 0, $T_{bit} = 32T_{VCLK}$

The averaged bit time is expressed in terms of T_{VCLK} as follows:

For all P other than 0, and all M and d (0 < d <1),

$$T^{a}bit = \left[16\left(P + 1 + \frac{M}{16}\right) + d\right]T_{VCLK}$$

For P = 0, $T_{bit} = 32T_{VCLK}$

13.4.2.5.1.5 Header Generation

Automatic generation of the LIN protocol header data stream is supported without CPU interaction. The CPU or the DMA triggers a message header generation and the LIN state machine handles the generation. A commander node initiates header generation on the CPU or DMA writes to the IDBYTE in the LINID register. The header is always sent by the commander to initiate a LIN communication and consists of three fields: synchronization break field, synchronization field, and identification field, as seen in Figure 13-201.

Note

The LIN protocol uses the parity bits in the identifier. The control length bits are optional to the LIN protocol.

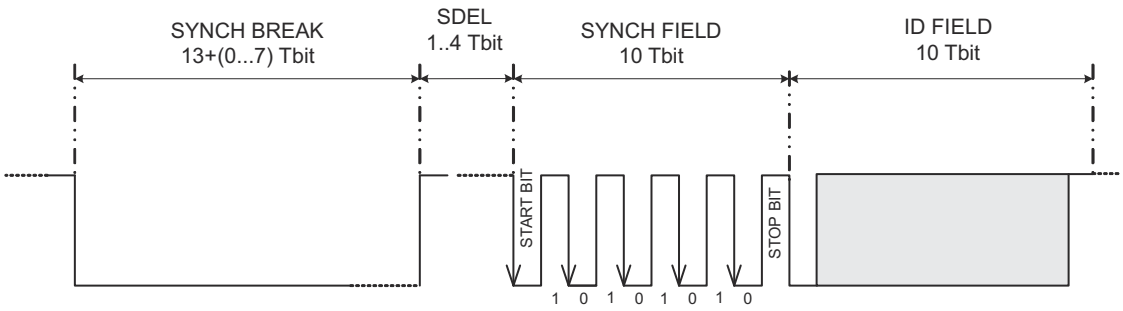


Figure 13-201. Message Header in Terms of T_{bit}

- The break field consists of two components:
  - The synchronization break (SYNCH BREAK) consists of a minimum of 13 (dominant) low bits to a maximum of 20 dominant bits. The sync break length can be extended from the minimum with the 3-bit SBREAK value in the LINCOMP register.
  - The synchronization break delimiter (SDEL) consists of a minimum of 1 (recessive) high bit to a maximum of 4 recessive bits. The delimiter marks the end of the synchronization break field. The sync break delimiter length depends on the 2-bit SDEL value in the LINCOMP register.
- The synchronization field (SYNCH FIELD) consists of one start bit, byte 0x55, and a stop bit. SYNCH FIELD is used to convey T_{bit} information and resynchronize LIN bus nodes.
- The identifier field ID byte can use 6 bits as an identifier, with optional length control and two optional bits as parity of the identifier. The identifier parity is used and checked if the PARITY ENA bit is set. If length control bits are not used, then there can be a total of 64 identifiers plus parity. If neither length control or parity are used there can be up to 256 identifiers. See Figure 13-202 for an illustration of the ID field.

Note

Optional Control Length Bits

The control length bits only apply to LIN standards prior to LIN 1.3. IDBYTE field conveys response length information if compliant to standards earlier than LIN1.3. The SCIFORMAT register stores the length of the response for later versions of the LIN protocol.

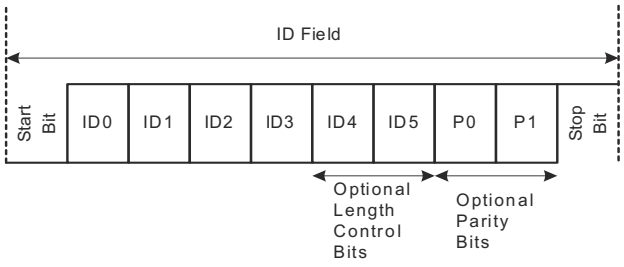


Figure 13-202. ID Field

Note

If the LIN module, configured as a responder in multibuffer mode, is in the process of transmitting data while a new header comes in, the module can end up responding with the data from the previous interrupted response (not the data corresponding to the new ID). To avoid this scenario, the following procedure can be used:

1. Check for the Bit Error (BE) during the response transmission. If the BE flag is set, this indicates that a collision has happened on the LIN bus (here because of the new Synch Break).
2. In the Bit Error ISR, configure the TD0 and TD1 registers with the next set of data to be transmitted on a TX Match for the incoming ID. Before writing to TD0/TD1 make sure that there was not already an update because of a Bit Error; otherwise, TD0/TD1 can be written twice for one ID.
3. Once the complete ID is received, based on the match, the newly configured data is transmitted by the node.

13.4.2.5.1.5.1 Event Triggered Frame Handling

The LIN 2.0 protocol uses event-triggered frames that can occasionally cause collisions. Event-triggered frames are handled in software.

If no responder answers to an event triggered frame header, the commander node sets the NRE flag, and a NRE interrupt occurs if enabled. If a collision occurs, a frame error and checksum error can arise before the NRE error. Those errors are flagged and the appropriate interrupts occur, if enabled.

Frame errors and checksum errors depend on the behavior and synchronization of the responding responders. If the responders are totally synchronized and stop transmission once the collision occurred, it is possible that only the NRE error is flagged despite the occurrence of a collision. To detect if there has been a reception of one byte before the NRE error is flagged, the BUS BUSY flag can be used as an indicator.

The BUS BUSY flag is set on the reception of the first bit of the header and remains set until the header reception is complete, and again is set on the reception of the first bit of the response. In the case of a collision, the flag is cleared in the same cycle as the NRE flag is set.

Software can implement the following sequence:

- Once the reception of the header is done (poll for RXID flag), wait for the BUS BUSY flag to get set or the NRE flag to get set.
- If the BUS BUSY flag is not set before the NRE flag, then a true no response is the case (no data has been transmitted onto the bus).
- If the BUS BUSY flag gets set, then wait for the NRE flag to get set or for successful reception. If the NRE flag is set, then a collision has occurred on the bus.

Even in the case of a collision, the received (corrupted) data is accessible in the RX buffers; registers LINRD0 and LINRD1.

13.4.2.5.1.5.2 Header Reception and Adaptive Baud Rate

A responder node baud rate can optionally be adjusted to the detected bit rate as an option to the LIN module. The adaptive baud rate option is enabled by setting the ADAPT bit. During header reception, a responder measures the baud rate during detection of the synch field. If ADAPT bit is set, then the measured baud rate is compared to the responder node programmed baud rate and adjusted to the LIN bus baud rate if necessary.

The responder node adjusts to any measured baud rate that is within ±10% of the programmed baud rate. For example, if the expected baud rate is programmed at 20kbps, the responder node detects any baud rate between 18kbps and 22kbps and adjusts accordingly. The MBRSR register prescaler is determined by the following formula:

$$\text{MBR} = \frac{\text{F}_{\text{VCLK}}}{1.1 \times \text{F}_{\text{LINCLK}}}$$

The LIN synchronizer determines two measurements: BRK_count and BAUD_count (Figure 13-203). These values are always calculated during the Header reception for synch field validation (Figure 13-204).

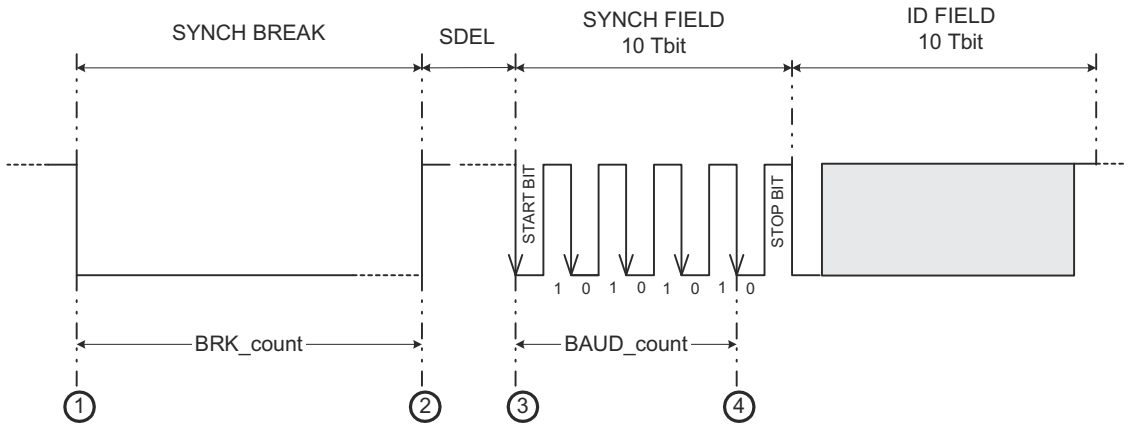


Figure 13-203. Measurements for Synchronization

By measuring the values BRK_count and BAUD_count, a valid sync break sequence can be detected as described in Figure 13-204. The four numbered events in Figure 13-203 signal the start/stop of the synchronizer counter. The synchronizer counter uses VCLK as the time base.

The synchronizer counter is used to measure the sync break relative to the detecting node T_{bit}. For a responder node receiving the sync break, a threshold of 11 T_{bit} is used as required by the LIN protocol. For detection of the dominant data stream of the sync break, the synchronizer counter is started on a falling edge and stopped on a rising edge of the LINRX. On detection of the sync break delimiter, the synchronizer counter value is saved and then reset.

On detection of five consecutive falling edges, the BAUD_count is measured. Bit timing calculation and consistency to required accuracy is implemented following the recommendations of LIN revision 2.0. A responder node can calculate a single T_{bit} time by division of BAUD_count by 8. In addition, for consistency between the detected edges the following is evaluated:

BAUD_count + BAUD_count » 2 + BAUD_count » 3 ≤ BRK_count

The BAUD_count value is shifted 3 times to the right and rounded using the first insignificant bit to obtain a T_{bit} unit. If the ADAPT bit is set, then the detected baud rate is compared to the programmed baud rate.

During the header reception processing as illustrated in Figure 13-204, if the measured BRK_count value is less than 11 T_{bit}, the sync break is not valid according to the protocol for a fixed rate. If the ADAPT bit is set, then the MBRS register is used for measuring BRK_count and BAUD_count values and automatically adjusts to any allowed LIN bus rate (refer to *LIN Specification Package 2.0*).

Note

In adaptive mode, the MBRS divider can be set to allow a maximum baud rate that is not more than 10% above the expected operating baud rate in the LIN network. Otherwise, a 0x00 data byte can mistakenly be detected as a sync break.

The break-threshold relative to the responder node is 11 T_{bit}. The break is 13 T_{bit} as specified in LIN v1.3.

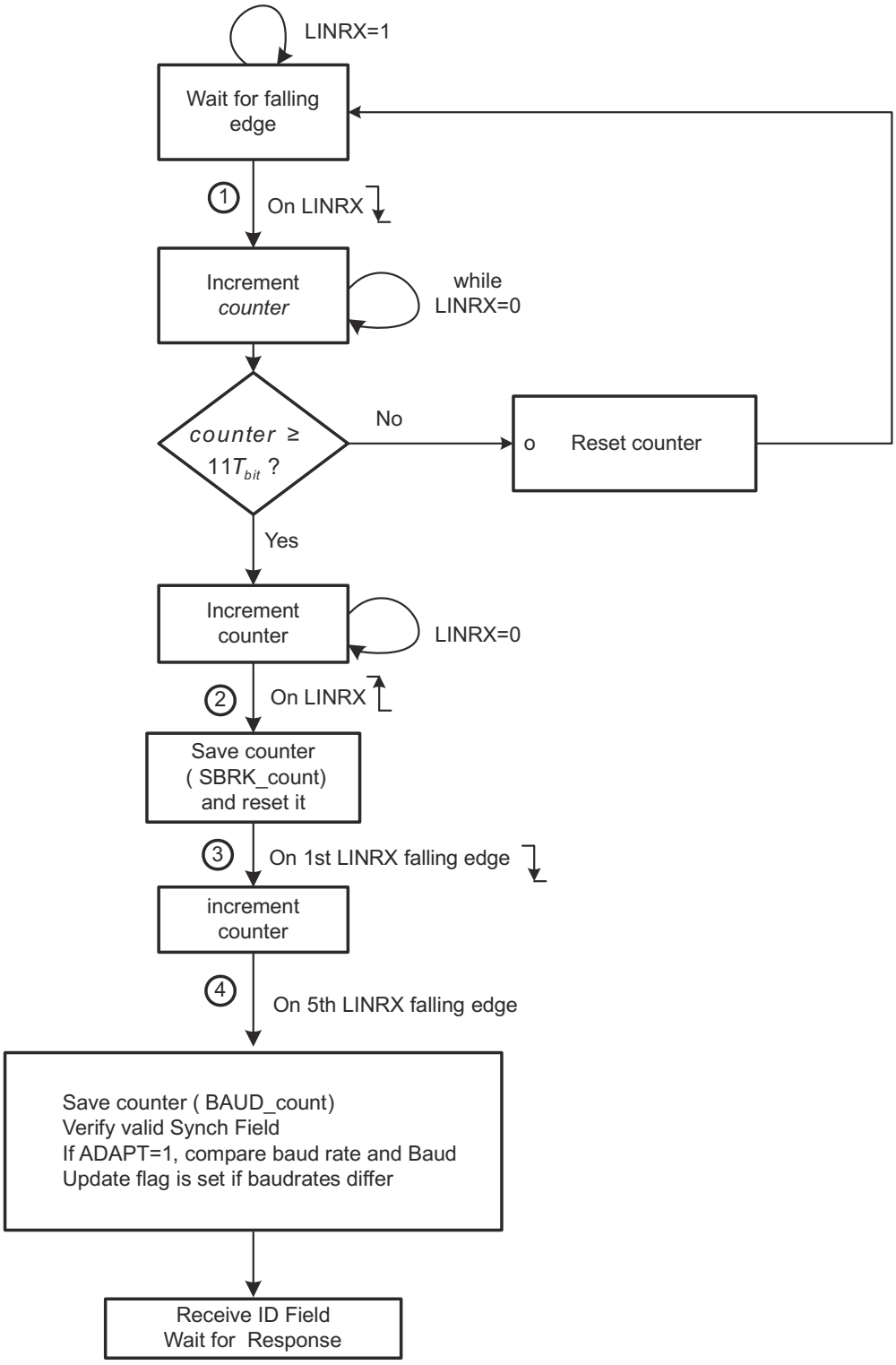


Figure 13-204. Synchronization Validation Process and Baud Rate Adjustment

If the synch field is not detected within the given tolerances, the inconsistent-sync-field-error (ISFE) flag is set. An ISFE interrupt is generated, if enabled by the respective bit in the SCISETINT register. The ID byte can be received after the synch field validation was successful. Any time a valid break (larger than 11 T_{bit}) is detected, the receiver state machine can reset to reception of this new frame. This reset condition is only valid during response state, not if an additional synch break occurs during header reception.

Note

When an inconsistent synch field (ISFE) error occurs, suggested action for the application is to reset the SWnRST bit and set the SWnRST bit to make sure that the internal state machines are back to the normal states.

13.4.2.5.1.6 Extended Frames Handling

The LIN protocol 2.0 and prior includes two extended frames with identifiers 62 (user-defined) and 63 (reserved extended). The response data length of the user-defined frame (ID 62, or 0x3E) is unlimited. The length for this identifier is set at network configuration time to be shared with the LIN bus nodes.

Extended frame communication is triggered on reception of a header with identifier 0x3E; see Figure 13-205. Once the extended frame communication is triggered, unlike normal frames, this communication needs to be stopped before issuing another header. To stop the extended frame communication the STOP EXT FRAME bit must be set.

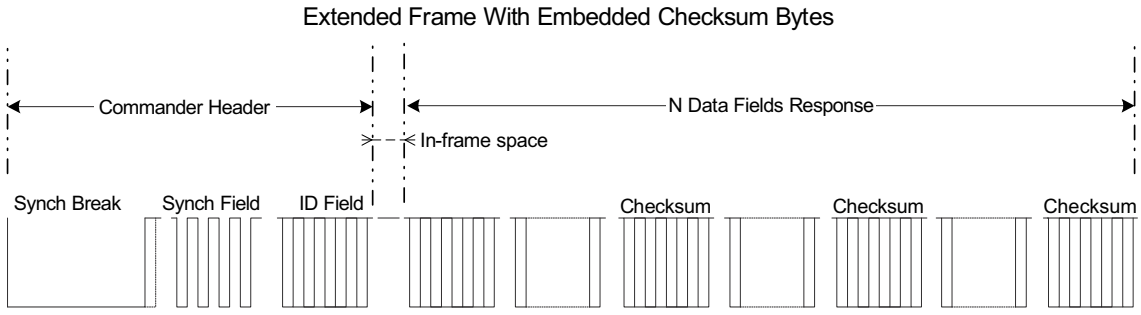


Figure 13-205. Optional Embedded Checksum in Response for Extended Frames

An ID interrupt is generated (if enabled and there is a match) on reception of ID 62 (0x3E). This interrupt allows the CPU using a software counter to keep track of the bytes that are being sent out and decides when to calculate and insert a checksum byte (recommended at periodic rates). To handle this procedure, SC bit is used. A write to the send checksum bit SC initiates an automatic send of the checksum byte. The last data field can always be a checksum in compliance with the LIN protocol.

The periodicity of the checksum insertion, defined at network configuration time, is used by the receiving node to evaluate the checksum of the ongoing message, and has the benefit of enhanced reliability.

For the sending node, the checksum is automatically embedded each time the send checksum bit SC is set. For the receiving node, the checksum is compared each time the compare checksum bit CC is set; see Figure 13-206.

Note

The LIN 2.0 enhanced checksum does not apply to the reserved identifiers. The reserved identifiers always use the classic checksum.

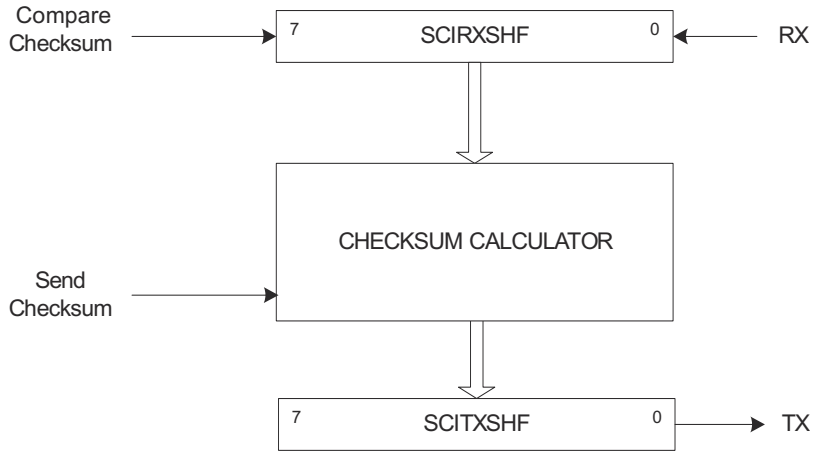


Figure 13-206. Checksum Compare and Send for Extended Frames

13.4.2.5.1.7 Timeout Control

Any LIN node listening to the bus and expecting a response initiated from a commander node can flag a no-response error timeout event. The LIN protocol defines four types of timeout events, which are all handled by the hardware of the LIN module. The four LIN protocol events are:

- No-response timeout error
- Bus idle detection
- Timeout after wakeup signal
- Timeout after three wakeup signals

13.4.2.5.1.7.1 No-Response Error (NRE)

The no-response error occurs when any node expecting a response waits for T_{FRAME_MAX} time and the message frame is not fully completed within the maximum length allowed, T_{FRAME_MAX}. After this time, a no-response error (NRE) is flagged in the NRE bit of the SCIFLR register. An interrupt is triggered, if enabled.

As specified in the LIN 1.3 standard, the minimum time to transmit a frame is:

$$T_{FRAME\_MIN} = T_{HEADER\_MIN} + T_{DATA\_FIELD} + T_{CHECKSUM\_FIELD} = 44 + 10N$$

where N = number of data fields.

And the maximum time frame is given by:

$$T_{FRAME\_MAX} = T_{FRAME\_MIN} * 1.4 = (44 + 10N) * 1.4$$

The timeout value T_{FRAME_MAX} is derived from the *N* number of data fields value, see Table 13-235. The *N* value is either embedded in the header ID field for messages or is part of the description file. In the latter case, the 3-bit CHAR value in SCIFORMAT register indicates the value for *N*.

Note

The length coding of the ID field does not apply to two extended frame identifiers, ID fields of 0x3E (62) and 0x3F (63). In these cases, the ID field can be followed by an arbitrary number of data byte fields. Also, the LIN 2.0 protocol specification mentions that ID field 0x3F (63) cannot be used. For these two cases, the NRE is not handled by the LIN hardware.

Table 13-235. Timeout Values in T_{bit} Units

N	T_{DATA_FIELD}	T_{FRAME_MIN}	T_{FRAME_MAX}
1	10	54	76
2	20	64	90
3	30	74	104
4	40	84	118
5	50	94	132
6	60	104	146
7	70	114	160
8	80	124	174

13.4.2.5.1.7.2 Bus Idle Detection

The second type of timeout can occur when a node detects an inactive LIN bus: no transitions between recessive and dominant values are detected on the bus. This happens after a minimum of 4 seconds (this is 80,000 F_{LINCLK} cycles with the fastest bus rate of 20kbps). If a node detects no activity in the bus as the TIMEOUT bit is set, assume that the LIN bus is in sleep mode. Application software can use the Timeout flag to determine when the LIN bus is inactive and put the LIN into sleep mode by writing the POWERDOWN bit.

Note

After the timeout was flagged, a SWnRESET must be asserted before entering Low-Power Mode. This is required to reset the receiver in case that an incomplete frame is on the bus before the idle period.

13.4.2.5.1.7.3 Timeout After Wakeup Signal and Timeout After Three Wakeup Signals

The third and fourth types of timeout are related to the wakeup signal. A node initiating a wakeup must expect a header from the commander within a defined amount of time: timeout after wakeup signal. See Section 13.4.2.6.3 for more details.

13.4.2.5.1.8 TXRX Error Detector (TED)

The following sources of error are detected by the TXRX error detector logic (TED). The TED logic consists of a bit monitor, an ID parity checker, and a checksum error. The following errors are detected:

- Bit errors (BE)
- Physical bus errors (PBE)
- Identifier parity errors (PE)
- Checksum errors (CE)

All of these errors (BE, PBE, PE, CE) are flagged. An interrupt for the flagged errors is generated if enabled. A message is valid for both the transmitter and the receiver if there is no error detected until the end of the frame.

13.4.2.5.1.8.1 Bit Errors

A bit error (BE) is detected at the bit time when the bit value that is monitored is different from the bit value that is sent. A bit error is indicated by the BE flag in SCIFLR. After signaling a BE, the transmission is aborted no later than the next byte. The bit monitor makes sure that the transmitted bit in LINTX is the correct value on the LIN bus by reading back on the LINRX pin as shown in Figure 13-207.

Note

If a bit occurs due to receiving a header during a responder response, NRE/TIMEOUT flag is not set for the new frame.

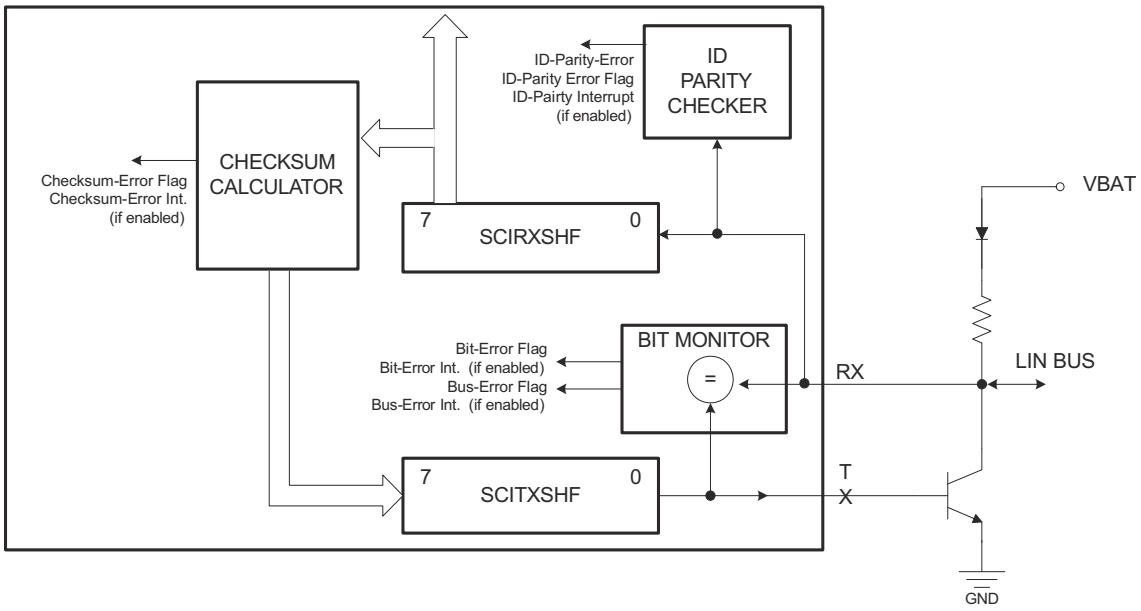


Figure 13-207. TXRX Error Detector

13.4.2.5.1.8.2 Physical Bus Errors

A Physical Bus Error (PBE) has to be detected by a commander, if no valid message can be generated on the bus (bus shorted to GND or VBAT). The bit monitor detects a PBE during the header transmission, if no Synch Break can be generated (for example, because of a bus shortage to VBAT) or if no Synch Break delimiter can be generated (for example, because of a bus shortage to GND). Once the Sync Break Delimiter was validated, all other deviations between the monitored and the sent bit value are flagged as Bit Errors (BE) for this frame.

13.4.2.5.1.8.3 ID Parity Errors

If parity is enabled, an ID parity error (PE) is detected if any of the two parity bits of the sent ID byte are not equal to the calculated parity on the receiver node. The two parity bits are generated using the following mixed parity algorithm:

$$P0 = ID0 \oplus ID1 \oplus ID2 \oplus ID4 \ (even\ Parity)$$
$$P1 = ID1 \oplus ID3 \oplus ID4 \oplus ID5 \ (odd\ Parity)$$

If an ID-parity error is detected, the ID-parity error is flagged, and the received ID is not valid. See Section 13.4.2.5.1.9 for details.

13.4.2.5.1.8.4 Checksum Errors

A checksum error (CE) is detected and flagged at the receiving end, if the calculated modulo-256 sum over all received data bytes (including the ID byte if the enhanced checksum type) plus the checksum byte does not result in 0xFF. The modulo-256 sum is calculated over each byte by adding with carry, where the carry bit of each addition is added to the LSB of the resulting sum.

For the transmitting node, the checksum byte sent at the end of a message is the inverted sum of all the data bytes (see Figure 13-208) for classic checksum implementation. The checksum byte is the inverted sum of the identifier byte and all the data bytes (see Figure 13-209) for the LIN 2.0 compliant enhanced checksum implementation. The classic checksum implementation can always be used for reserved identifiers 60 to 63; therefore, the CTYPE bit is overridden in this case. For signal-carrying-frame identifiers (0 to 59) the type of checksum used depends on the CTYPE bit.

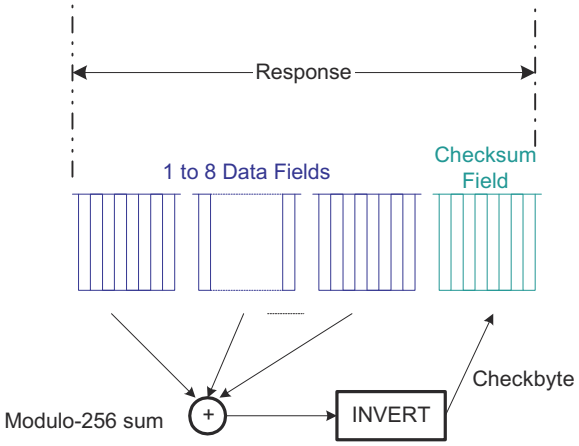


Figure 13-208. Classic Checksum Generation at Transmitting Node

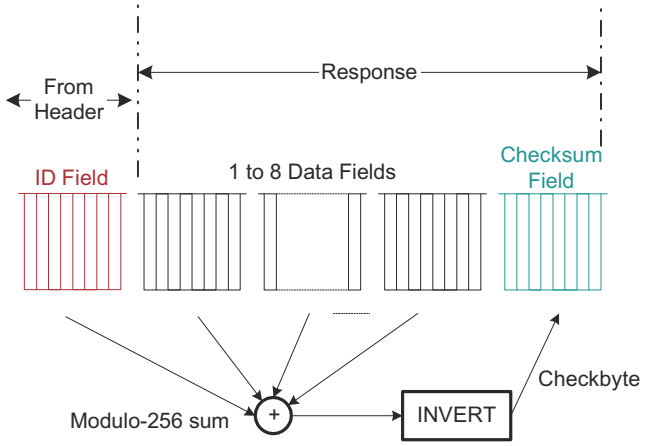


Figure 13-209. LIN 2.0-Compliant Checksum Generation at Transmitting Node

13.4.2.5.1.9 Message Filtering and Validation

Message filtering uses the entire identifier to determine which nodes participate in a response, either receiving or transmitting a response. Therefore, two acceptance masks are used as shown in Figure 13-210. During header reception, all nodes filter the ID-Field (ID-Field is the part of the header explained in Figure 13-202) to determine whether the nodes transmit a response or receive a response for the current message. There are two masks for message ID filtering: one to accept a response reception, the other to initiate a response transmission. See Figure 13-210. All nodes compare the received ID to the identifier stored in the ID-Responder Task BYTE of the LINID register and use the RX ID MASK and the TX ID MASK fields in the LINMASK register to filter the bits of the identifier that can not be compared.

If there is an RX match with no parity error and the RXENA bit is set, there is an ID RX flag and an interrupt is triggered if enabled. If there is a TX match with no parity error and the TXENA bit is set, there is an ID TX flag and an interrupt is triggered if enabled in the SCISETINT register.

The masked bits become "don't cares" for the comparison. To build a mask for a set of identifiers, an XOR function can be used.

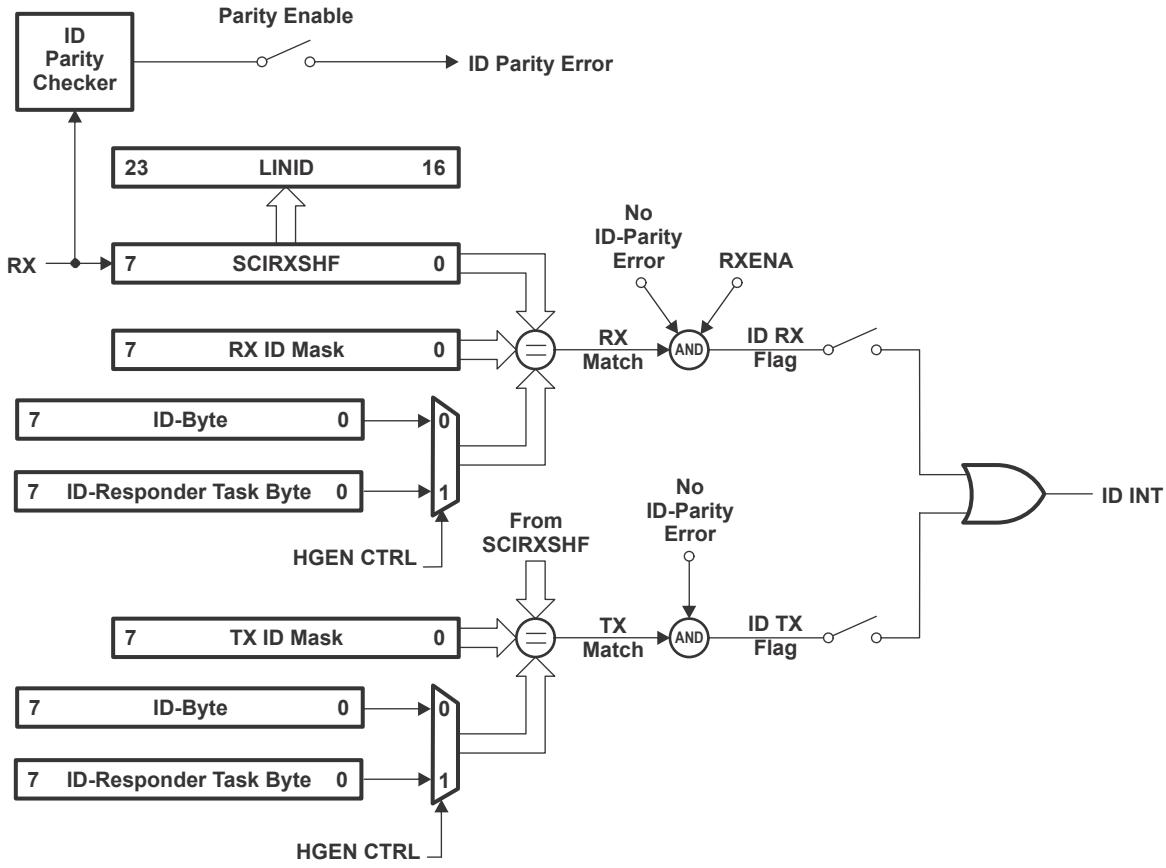


Figure 13-210. ID Reception, Filtering, and Validation

For example, to build a mask to accept IDs 0x26 and 0x25 using LINID[7:0] = 0x20; that is, compare 5 most-significant bits (MSBs) and filter 3 least-significant bits (LSBs), the acceptance mask can be:

$$(0x26 + 0x25) \oplus 0x20 = 0x07$$

A mask of all zeros compares all bits of the received identifier in the shift register with the ID-BYTE in LINID[7:0]. If HGEN CTRL is set to 1, a mask of 0xFF always causes a match. A mask of all 1s filters all bits of the received identifier, and thus there is an ID match regardless of the content of the ID-Responder Task BYTE field in the LINID register.

Note

When the HGEN CTRL bit = 0, the LIN nodes compare the received ID to the ID-BYTE field in the LINID register, and use the RX ID MASK and the TX ID MASK in the LINMASK register to filter the bits of the identifier that can not be compared.

If there is an RX match with no parity error and the RXENA bit is set, there is an ID RX flag and an interrupt is triggered if enabled. A mask of all 0s compares all bits of the received identifier in the shift register with the ID-BYTE field in LINID[7:0]. A mask of all 1s filters all bits of the received identifier and there is no match.

If HGEN CTLR = 1:

- Received ID is compared with the ID-Responder Task byte, using the RXID mask and the TXID mask.
- A mask of all 1s always result in a match.
- A mask of all 0s means all the bits must be the same to result in a match.
- If a mask has some bits that are 1s, then those bits are not used for the filtering criterion.

If HGEN CTRL = 0:

- Received ID is compared with the ID byte, using the RXID mask and the TXID mask.
- A mask of all 1s results in no match.
- A mask of all 0s means all the bits must be the same to result in a match.
- If a mask has some bits that are 1s, then those bits are not used for the filtering criterion.

During header reception, the received identifier is copied to the Received ID field LINID[23:16]. If there is no parity error and there is either a TX match or an RX match, then the corresponding TX or RX ID flag is set. If the ID interrupt is enabled, then an ID interrupt is generated.

After the ID interrupt is generated, the CPU can read the Received ID field LINID[23:16] and determine what response to load into the transmit buffers.

Note

When byte 0 is written to TD0 (LINTD0[31:24]), the response transmission is automatically generated.

In multibuffer mode, the TXRDY flag is set when all the response data bytes and checksum byte are copied to the shift register SCITXSHF. In non-multibuffer mode, the TXRDY flag is set each time a byte is copied to the SCITXSHF register, and also for the last byte of the frame after the checksum byte is copied to the SCITXSHF register.

In multibuffer mode, the TXEMPTY flag is set when both the transmit buffers TDy and the SCITXSHF shift register are emptied and the checksum has been sent. In non-multibuffer mode, TXEMPTY is set each time TD0 and SCITXSHF are emptied, except for the last byte of the frame where the checksum byte must also be transmitted.

If parity is enabled, all responder receiving nodes validate the identifier using all eight bits of the received ID byte. The SCI/LIN flags a corrupted identifier if an ID-parity error is detected.

13.4.2.5.1.10 Receive Buffers

To reduce CPU load when receiving a LIN N-byte (with N = 1–8) response in interrupt mode or DMA mode, the SCI/LIN module has eight receive buffers. These buffers can store an entire LIN response in the RDy receive buffers. Figure 13-194 illustrates the receive buffers.

The checksum byte following the data bytes is validated by the internal checksum calculator. The checksum error (CE) flag indicates a checksum error and a CE interrupt is generated if enabled in the SCISETINT register.

The multibuffer 3-bit counter counts the data bytes transferred from the SCIRXSHF register to the RDy receive buffers if multibuffer mode is enabled, or to RD0 if multibuffer mode is disabled. The 3-bit compare register contains the number of data bytes expected to be received. In cases where the ID BYTE field does not convey message length (see *Note: Optional Control Length Bits* in Section 13.4.2.5.1.5), the LENGTH value, indicates the expected length and is used to load the 3-bit compare register. Whether the length control field or the LENGTH value is used is selectable with the COMM MODE bit.

A receive interrupt, and a receive ready RXRDY flag , and a DMA request (RXDMA) can occur after receiving a response, if there are no response receive errors for the frame (such as, there is no checksum error, frame error, and overrun error). The checksum byte is compared before acknowledging a reception. A DMA request can be generated for each received byte or for the entire response depending on whether the multibuffer mode is enabled or not (MBUF MODE bit).

Note

In multibuffer mode following are the scenarios associated with clearing the RXRDY flag bit:

1. The RXRDY flag cannot be cleared by reading the corresponding interrupt offset in the SCIINTVECT0/1 register.
2. For LENGTH less than or equal to 4, Read to RD0 register clears the RXRDY flag.
3. For LENGTH greater than 4, Read to RD1 register clears the RXRDY flag.

13.4.2.5.1.11 Transmit Buffers

To reduce the CPU load when transmitting a LIN N-byte (with N = 1–8) response in interrupt mode or DMA mode, the SCI/LIN module has 8 transmit buffers, TD0–TD7 in LINTD0 and LINTD1. With these transmit buffers, an entire LIN response field can be preloaded in the TXy transmit buffers. Optionally, a DMA transfer can be done on a byte-per-byte basis when multibuffer mode is not enabled (MBUF MODE bit). Figure 13-195 illustrates the transmit buffers.

The multibuffer 3-bit counter counts the data bytes transferred from the TDy transmit buffers register if multibuffer mode is enabled, or from TD0 to SCITXSHF if multibuffer mode is disabled. The 3-bit compare register contains the number of data bytes expected to be transmitted. If the ID field is not used to convey message length (see *Note: Optional Control Length Bits* in Section 13.4.2.5.1.5), the LENGTH value indicates the expected length and is used instead to load the 3-bit compare register. Whether the length control field or the LENGTH value is used is selectable with the COMM MODE bit.

A transmit interrupt (TX interrupt) and a transmit ready flag (TXRDY flag), as well as a DMA request (TXDMA) can occur after transmitting a response. A DMA request can be generated for each transmitted byte or for the entire response depending on whether multibuffer mode is enabled or not (MBUF MODE bit).

The checksum byte is automatically generated by the checksum calculator and sent after the data-fields transmission is finished. The multibuffer 3-bit counter counts the data bytes transferred from the TDy buffers into the SCITXSHF register.

Note

The transmit interrupt request can be eliminated until the next series of data is written into the transmit buffers LINTD0 and LINTD1, by disabling the corresponding interrupt using the SCICLRINT register or by disabling the transmitter using the TXENA bit.

13.4.2.5.2 LIN Interrupts

LIN and SCI modes have a common interrupt block, as explained in Section 13.4.2.4.2. There are 16 interrupt sources in the SCI/LIN module, with 8 of them being LIN mode only, as seen in Table 13-229.

A LIN message frame indicating the timing and sequence of the LIN interrupts that can occur is shown in Figure 13-211.

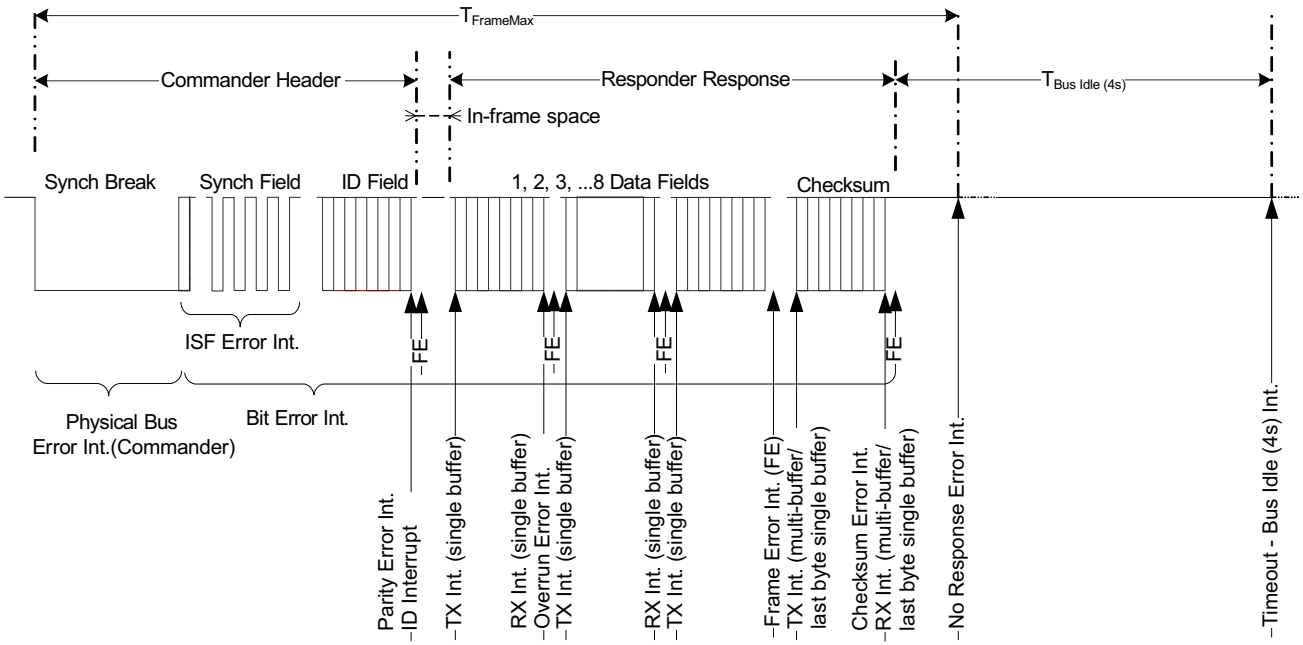


Figure 13-211. LIN Message Frame Showing LIN Interrupt Timing and Sequence

13.4.2.5.3 Servicing LIN Interrupts

When servicing an interrupt, clear the corresponding flag in the flag register (SCIFLR) before clearing the global interrupt (LIN_GLB_INT_CLR). The ISR can follow the guidelines below. This prevents any spurious or duplicate interrupt from occurring.

- Clear the LIN interrupt flag in the SCIFLR register.
- Read the LIN interrupt status register to make sure the flag is cleared.
- Clear the global interrupt flag bit in LIN_GLB_INT_CLR.

Note

The transmit interrupt is generated before the LIN transmitter is ready to accept new data. Inside of the LIN transmit ISR, the software can wait until the buffer is completely empty before loading the next data. This can be done by polling for the Bus Busy Flag (SCIFLR.BUSY) to be 0.

13.4.2.5.4 LIN Configurations

The following list details the configuration steps that software can perform prior to the transmission or reception of data in LIN mode. As long as the SWnRST bit in the SCIGCR1 register is cleared to 0 the entire time that the LIN is being configured, the order in which the registers are programmed is not important.

- Enable LIN by setting RESET bit.
- Clear SWnRST to 0 before configuring the LIN.
- Enable the LINRX and LINTX pins by setting the RX FUNC and TX FUNC bits.
- Select LIN mode by programming LIN MODE bit.
- Select commander or responder mode by programming the CLOCK bit.
- Select the desired frame format (checksum, parity, length control) by programming SCIGCR1.
- Select multibuffer mode by programming MBUF MODE bit.

- Select the baud rate to be used for communication by programming BRSR.
- Set the maximum baud rate to be used for communication by programming MBRSR.
- Set the CONT bit to make LIN not halt for an emulation breakpoint until the LIN current reception or transmission is complete (this bit is used only in an emulation environment).
- Set LOOP BACK bit to connect the transmitter to the receiver internally if needed (this feature is used to perform a self-test).
- Select the receiver enable RXENA bit, if data is to be received.
- Select the transmit enable TXENA bit, if data is to be transmitted.
- Select the RX ID MASK and the TX ID MASK fields in the LINMASK register.
- Set SWnRST to 1 after the LIN is configured.
- Perform Receive or Transmit data (see Section 13.4.2.5.1.9, Section 13.4.2.5.4.1, and Section 13.4.2.5.4.2).

Note

If TXENA is set and the SWnRST is released, the LIN only generates a new DMA request. The LIN hardware does not generate a new transmit interrupt request. If using interrupts, the first transmission must be started by software by writing the data to transmit and followed by writing to LIN TX to initiate the transmission.

13.4.2.5.4.1 Receiving Data

The LIN receiver is enabled to receive messages if both the RX FUNC bit and the RXENA bit are set to 1. If the RX FUNC bit is not set, the LINRX pin functions as a general-purpose I/O pin rather than as a LIN function pin.

The ID RX FLAG is set after a valid LIN ID is received with RX Match. An ID interrupt is generated, if enabled.

13.4.2.5.4.1.1 Receiving Data in Single-Buffer Mode

Single-buffer mode is selected when the MBUF MODE bit is cleared to 0. In this mode, LIN sets the RXRDY bit when the LIN transfers newly received data from SCIRXSHF to RD0. The SCI clears the RXRDY bit after the new data in RD0 has been read. Also, as data is transferred from SCIRXSHF to RD0, the LIN sets the FE, OE, or PE flags if any of these error conditions were detected in the received data. These error conditions are supported with configurable interrupt capability.

You can receive data by:

1. Polling Receive Ready Flag
2. Receive Interrupt
3. DMA

In polling method, software can poll for the RXRDY bit and read the data from RD0 byte of the LINRD0 register once the RXRDY bit is set high. The CPU is unnecessarily overloaded by selecting the polling method. To avoid this, you can use the interrupt or DMA method. To use the interrupt method, the SET RX INT bit is set. To use the DMA method, the SET RX DMA bit must be set. Either an interrupt or a DMA request is generated the moment the RXRDY bit is set. If the checksum scheme is enabled by setting the Compare Checksum (CC) bit to 1, the checksum is compared on the byte that is currently being received, which is expected to be the checksum byte. The CC bit is cleared once the checksum is received. A CE is immediately flagged, if there is a checksum error.

13.4.2.5.4.1.2 Receiving Data in Multibuffer Mode

Multibuffer mode is selected when the MBUF MODE bit is set to 1. In this mode, LIN sets the RXRDY bit after receiving the programmed number of data in the receive buffer and the checksum field, the complete frame. The error condition detection logic is similar to the single-buffer mode, except that this logic monitors for the complete frame. Like single-buffer mode, you can use the polling, DMA, or interrupt method to read the data. The received data has to be read from the LINRD0 and LINRD1 registers, based on the number of bytes. For a LENGTH less than or equal to 4, a read from the LINRD0 register clears the RXRDY flag. For a LENGTH greater than 4, a read from the LINRD1 register clears the RXRDY flag. If the checksum scheme is enabled by setting the Compare Checksum (CC) bit to 1 during the reception of the data, then the byte that is received after the reception of the programmed number of data bytes indicated by the LENGTH field is treated as a checksum byte. The CC bit is cleared once the checksum is received and compared.

13.4.2.5.4.2 Transmitting Data

The LIN transmitter is enabled if both the TX FUNC bit and the TXENA bit are set to 1. If the TX FUNC bit is not set, the LINTX pin functions as a general-purpose I/O pin rather than as a LIN function pin. Any value written to the TD0 before the TXENA bit is set to 1 is not transmitted. Both of these control bits allow for the LIN transmitter to be held inactive independently of the receiver.

The ID TX flag is set after a valid LIN ID is received with TX Match. An ID interrupt is generated, if enabled.

13.4.2.5.4.2.1 Transmitting Data in Single-Buffer Mode

Single-buffer mode is selected when the MBUF MODE bit is cleared to 0. In this mode, LIN waits for data to be written to TD0, transfers the data to SCITXSHF, and transmits the data. The TXRDY and TX EMPTY bits indicate the status of the transmit buffers. That is, when the transmitter is ready for data to be written to TD0, the TXRDY bit is set. Additionally, if both TD0 and SCITXSHF are empty, then the TX EMPTY bit is also set.

You can transmit data by:

1. Polling Transmit Ready Flag
2. Transmit Interrupt
3. DMA

In polling method, software can poll for the TXRDY bit to go high before writing the data to the TD0. The CPU is unnecessarily overloaded by selecting the polling method. To avoid this, you can use the interrupt or DMA method. To use the interrupt method, the SET TX INT bit is set. To use the DMA method, the SET TX DMA bit is set. Either an interrupt or a DMA request is generated the moment the TXRDY bit is set. When the LIN has completed transmission of all pending frames, the SCITXSHF register and the TD0 are empty, the TXRDY bit is set, and an interrupt/DMA request is generated, if enabled. Because all data has been transmitted, the interrupt/DMA request can be halted. This can either be done by disabling the transmit interrupt (CLR TX INT) / DMA request (CLR TX DMA bit) or by disabling the transmitter (clear TXENA bit). If the checksum scheme is enabled by setting the Send Checksum (SC) bit to 1, the checksum byte is sent after the current byte transmission. The SC bit is cleared after the checksum byte has been transmitted.

Note

The TXRDY flag cannot be cleared by reading the corresponding interrupt offset in the SCIINTVECT0 or SCIINTVECT1 register.

13.4.2.5.4.2.2 Transmitting Data in Multibuffer Mode

Multibuffer mode is selected when the MBUF MODE bit is set to 1. Like single-buffer mode, you can use the polling, DMA, or interrupt method to write the data to be transmitted. The transmitted data has to be written to the LINTD0 and LINTD1 registers, based on the number of bytes. LIN waits for data to be written to Byte 0 (TD0) of the LINTD0 register and transfers the programmed number of bytes to SCITXSHF to transmit one by one automatically. If the checksum scheme is enabled by setting the Send Checksum (SC) bit to 1, the checksum is sent after transmission of the last byte of the programmed number of data bytes, indicated by the LENGTH field. The SC bit is cleared after the checksum byte has been transmitted.

13.4.2.6 Low-Power Mode

The SCI/LIN module can be put in either local or global low-power mode. Global low-power mode is asserted by the system and is not controlled by the SCI/LIN module. During global low-power mode, all clocks to the SCI/LIN are turned off so the module is completely inactive. If global low-power mode is requested while the receiver is receiving data, then the SCI/LIN completes the current reception and then enters the low-power mode, that is, module enters low-power mode only when Busy bit (SCIFLR.3) is cleared.

The LIN module can enter low-power mode either when there was no activity on the LINRX pin for more than 4 seconds (this can be either a constant recessive or dominant level) or when a Sleep Command frame was received. Once the Timeout flag (SCIFLR.4) was set or once a Sleep Command was received, the POWERDOWN bit (SCIGCR2.0) must be set by the application software to make the module enter local low-power mode. A wakeup signal terminates the sleep mode of the LIN bus.

Note

Enabling Local Low-Power Mode During Receive and Transmit

If the wakeup interrupt is enabled and low-power mode is requested while the receiver is receiving data, then the SCI/LIN immediately generates a wakeup interrupt to clear the power-down bit. Thus, the SCI/LIN is prevented from entering low-power mode and completes the current reception. Otherwise, if the wakeup interrupt is disabled, the SCI/LIN completes the current reception and then enters the low-power mode.

13.4.2.6.1 Entering Sleep Mode

In LIN protocol, a sleep command is used to broadcast the sleep mode to all nodes. The sleep command consists of a diagnostic commander request frame with identifier 0x3C (60), with the first data field as 0x00. There must be no activity in the bus once all nodes receive the sleep command: the bus is in sleep mode.

Local low-power mode is asserted by setting the POWERDOWN bit; setting this bit stops the clocks to the SCI/LIN internal logic and registers. Clearing the POWERDOWN bit causes SCI/LIN to exit from local low-power mode. All the registers are accessible during local power-down mode. If a register is accessed in low-power mode, this access results in enabling the clock to the module for that particular access alone.

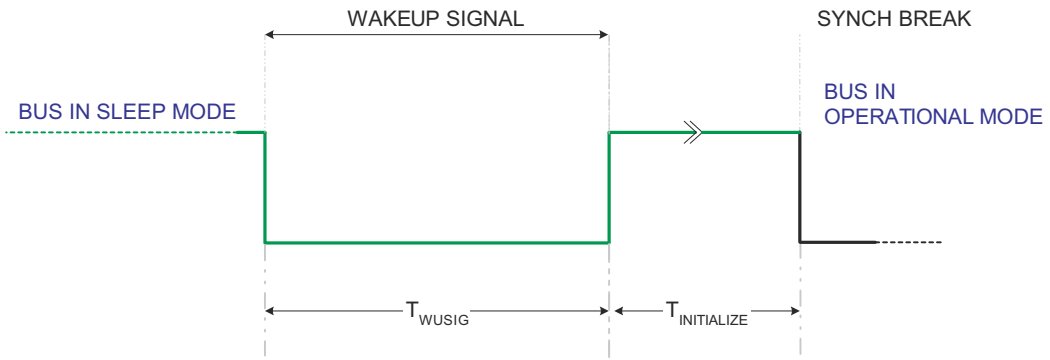
13.4.2.6.2 Wakeup

The wakeup interrupt is used to allow the SCI/LIN module to automatically exit a low-power mode. A SCI/LIN wakeup is triggered when a low level is detected on the receive RX pin, and this clears the POWERDOWN bit.

Note

If the wakeup interrupt is disabled, then the SCI/LIN enters low-power mode whenever the SCI/LIN is requested to do so, but a low level on the receive RX pin does not cause the SCI/LIN to exit low-power mode.

In LIN mode, any node can terminate sleep mode by sending a wakeup signal, see Figure 13-212. A responder node that detects the bus in sleep mode, and with a wakeup request pending, sends a wakeup signal. The wakeup signal is a dominant value on the LIN bus for T_{WUSIG}; this is at least 5 T_{bits} for the LIN bus baud rates. The wakeup signal is generated by sending a 0xF0 byte containing 5 dominant T_{bits} and 5 recessive T_{bits}.



0.25ms ≤ T_{WUSIG} ≤ 5ms

Figure 13-212. Wakeup Signal Generation

Assuming a bus with no noise or loading effects, a write of 0xF0 to TD0 loads the transmitter to meet the wakeup signal timing requirement for T_{WUSIG}. Then, setting the GENWU bit transmits the preloaded value in TD0 for a wakeup signal transmission.

Note

The GENWU bit can be set/reset only when SWnRST is set to 1 and the node is in power-down mode. The bit is cleared on a valid synch break detection. A commander sending a wakeup request, exits power-down mode upon reception of the wakeup pulse. The bit is cleared on a SWnRST. This can be used to stop a commander from sending further wakeup requests.

The TI TPIC1021 LIN transceiver, upon receiving a wakeup signal, translates it to the microcontroller for wakeup with a dominant level on the RX pin, or a signal to the voltage regulator. While the POWERDOWN bit is set, if the LIN module detects a recessive-to-dominant edge (falling edge) on the RX pin, the LIN module generates a wakeup interrupt if enabled in the SCISETINT register.

According to LIN protocol 2.0, the TI TPIC1021 LIN transceiver detecting a dominant level on the bus longer than 150ms detects it as a wakeup request. The LIN responder is ready to listen to the bus in less than 100ms ($T_{INITIALIZE}$<100ms) after a dominant-to-recessive edge (end-of-wakeup signal).

13.4.2.6.3 Wakeup Timeouts

The LIN protocol defines the following timeouts for a wakeup sequence. After a wakeup signal has been sent to the bus, all nodes wait for the commander to send a header. If no synch field is detected before 150ms (3,000 cycles at 20kHz) after a wakeup signal is transmitted, a new wakeup is sent by the same node that requested the first wakeup. This sequence is not repeated more than two times. After three attempts to wake up the LIN bus, wakeup signal generation is suspended for a 1.5s (30,000 cycles at 20kHz) period after three breaks.

Note

To achieve compatibility to LIN1.3 timeout conditions, the MBRS register must be set to make sure that the LIN 2.0 (real-time-based) timings meet the LIN 1.3 bit time base. A node triggering the wakeup can set the MBRS register accordingly to meet the targeted time as 128 Tbits × programmed prescaler.

The LIN handles the wakeup expiration times defined by the LIN protocol with a hardware implementation.

13.4.2.7 Emulation Mode

In emulation mode, the CONT bit determines how the SCI/LIN operates when the program is suspended. The SCI/LIN counters are affected by this bit during debug mode. when set, the counters are not stopped and when cleared, the counters are stopped debug mode.

Any reads in emulation mode to a SCI/LIN register do not have any effect on the flags in the SCIFLR register.

Note

When emulation mode is entered during the Frame transmission or reception of the frame and CONT bit is not set, Communication is not expected to be successful. The suggested usage is to set CONT bit during emulation mode for successful communication.

13.4.2.8 LIN Programming Guide

Driver Information

Driver features are available at the LIN driver page.

Software API Information

The LIN driver provides an API to configure the LIN module. Full documentation is located on APIs for LIN.

Example Usage

The below links shows an example on how to use LIN.

- LIN:
  - LIN Internal Loopback (Interrupt-based)
  - LIN/SCI Internal Loopback (Interrupt-based)
  - LIN/SCI DMA Loopback
  - LIN Internal Loopback (Polling-based)
  - LIN External Commander

13.5 Timer Modules

This section describes the timer modules in the device.

13.5.1 Real Time Interrupts/Windowed Watchdog Timer (RTI/WWDT)

This section describes the Real Time Interrupt/Windowed Watchdog Timer (RTI/WWDT) module implemented in the device.

13.5.1.1 RTI/WWDT Overview

The Real Time Interrupt module of the RTI/WWDT module provides general timer functionality for operating systems and for benchmarking code. The module incorporates several counters, which define the timebases needed for operating system-based scheduling requirements.

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.

There are eight RTI modules in the device and four WWDT modules in the device. Table 13-236 shows the RTI allocation across device domains.

Note

Eight instances are configured in RTI-only mode to function as general purpose timers. Another four instances are configured in WWDT-only mode to function as watchdog timers.

Table 13-236. RTI Device Instance Table

Instance	Device	
RTI0		✓
RTI1		✓
RTI2		✓
RTI3		✓
RTI4		✓
RTI5		✓
RTI6		✓

Table 13-236. RTI Device Instance Table (continued)

Instance	Device	
RTI7		✓

This diagram shows the module overview.

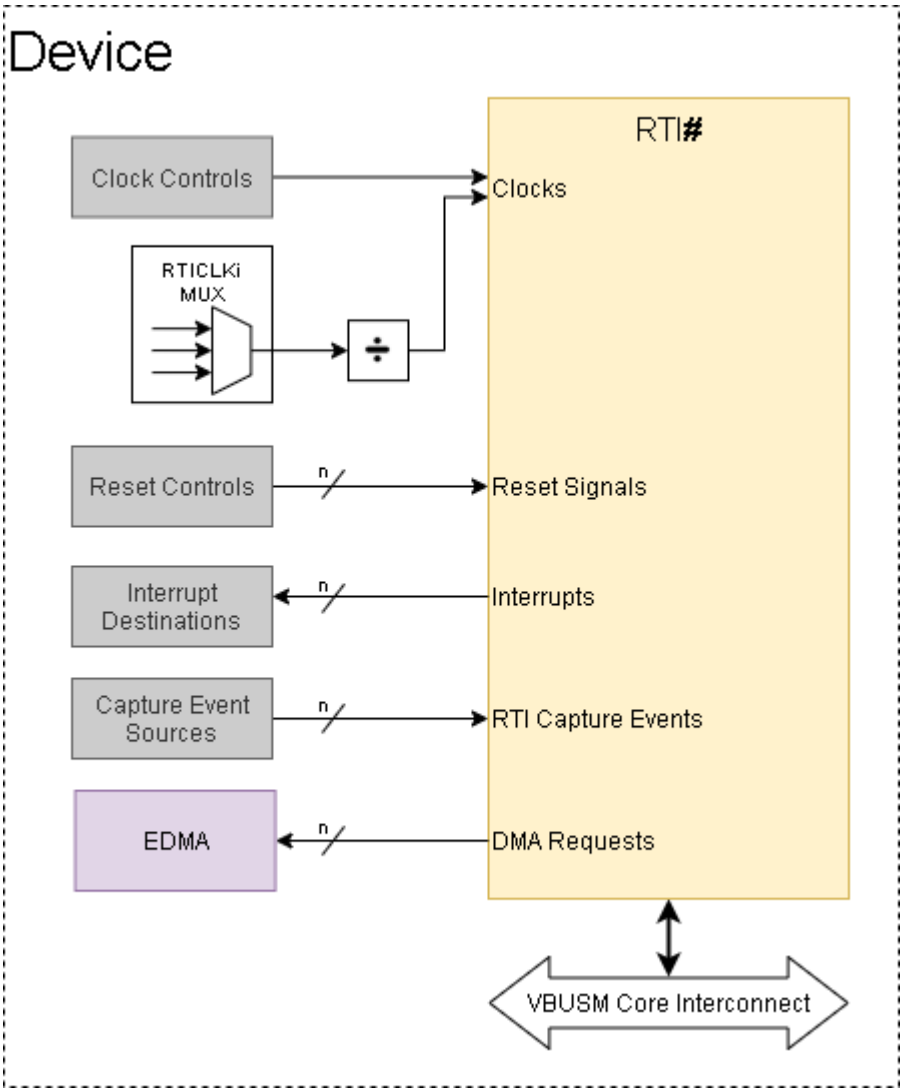


Figure 13-213. RTI Overview

= 0, 1, 2, 3
= 0, 1, 2, 3

Table 13-237 shows the WWDT allocation across device domains.

Table 13-237. WWDT Device Instance Table

Instance	Device
WWDT0	✓
WWDT1	✓
WWDT2	✓
WWDT3	✓

The WWDT instances are intended to function as a digital windowed watchdog for the CPU core that they are associated with:

- WWDT0 is dedicated to the first R5F CPU core (R5FSS0_CORE0)
- WWDT1 is dedicated to the second R5F CPU core (R5FSS0_CORE1)
- WWDT2 is dedicated to the third R5F CPU core (R5FSS1_CORE0)
- WWDT3 is dedicated to the fourth R5F CPU core (R5FSS1_CORE1)

All WWDT instances that are provisioned for a particular CPU core should not be used by any other CPU cores.

This diagram shows the module overview.

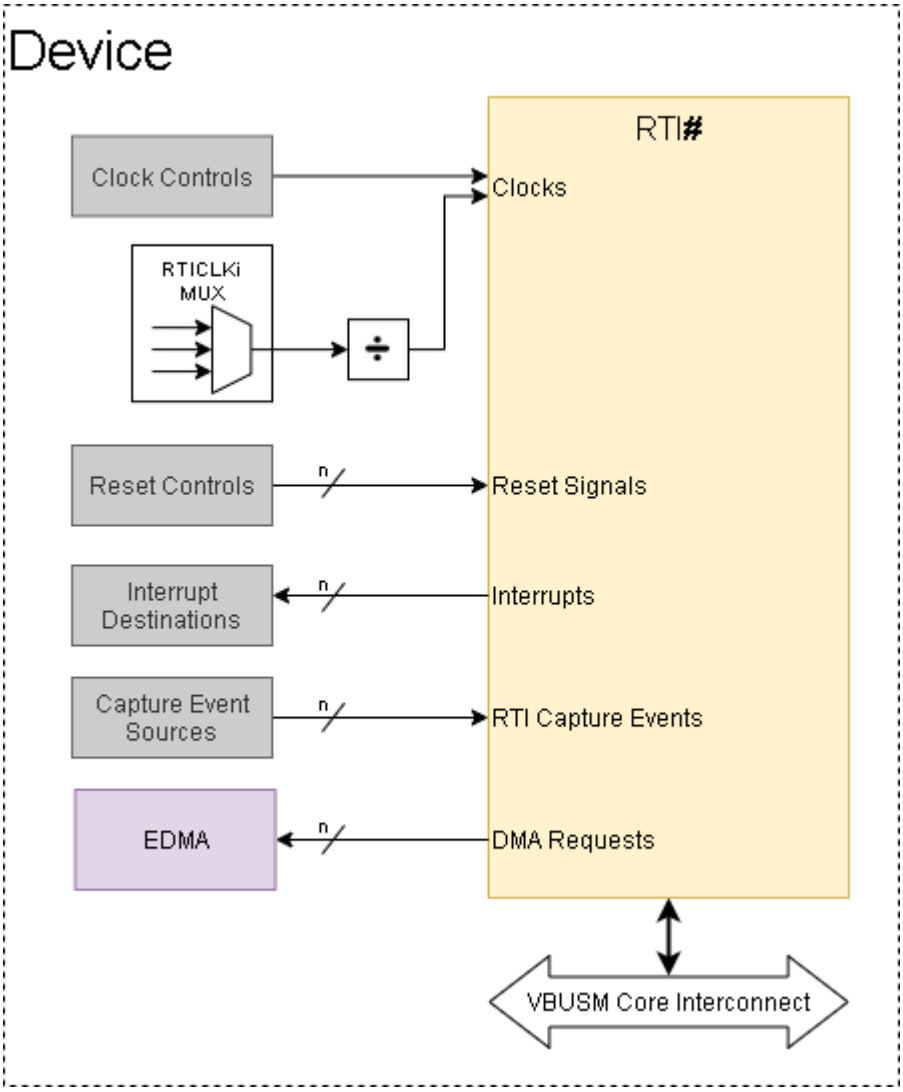


Figure 13-214. WWDT Overview

13.5.1.1.1 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 the internal prescale counter.
- 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.

13.5.1.1.2 RTI Unsupported Features

The RTI modules do not support the following features:

- External clock supervising circuit to switch to internal prescale counter 0, if external clock source fails to increment in a predefined window.
- DMA requests and events.
- Automatic update of all compare registers on compare match to generate periodic interrupts.
- Capture events to capture timestamps through recording of timer status.
- Two time-stamp (capture) functions for system or peripheral interrupts, one for each counter block.
- Analog Watchdog via external RC Network to prevent for runaway code.

13.5.1.2 RTI Integration

There are 8x RTI modules integrated in the device. The diagram and tables below show the device integration details.

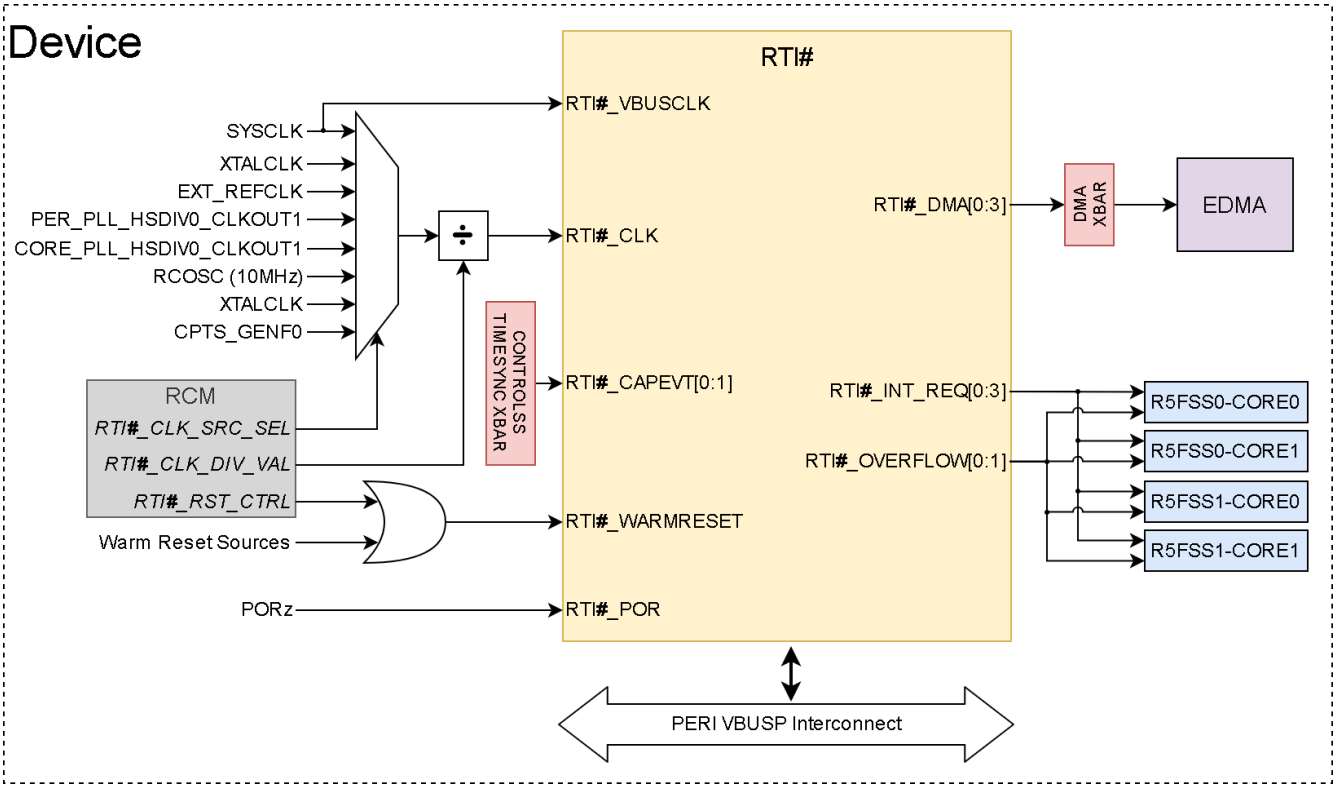


Figure 13-215. RTI Integration

The tables below summarize the integration of RTI# (where # = 0, 1, 2, 3, 4, 5, 6, 7) in the device.

Each RTI# instance is supplied by dedicated RTICLK# mux.

Table 13-238. *RTI* Device Integration

This table describes the module device integration details.

Module Instance	Device Allocation	SoC Interconnect
RTI0	✓	VBUSP CORE Interconnect
RTI1	✓	VBUSP CORE Interconnect
RTI2	✓	VBUSP CORE Interconnect
RTI3	✓	VBUSP CORE Interconnect
RTI4	✓	VBUSP CORE Interconnect
RTI5	✓	VBUSP CORE Interconnect
RTI6	✓	VBUSP CORE Interconnect
RTI7	✓	VBUSP CORE Interconnect

Table 13-239. *RTI* Clocks

This table table describes the module clocking signals.

Module Instance	Module Clock Input	Source Clock Signal	Source	Default Freq	Description
RTI0	RTI0_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	RTI0 VBUSP Interface Clock
	RTI0_FCLK (RTI_CLK)	XTALCLK	External XTAL	25 MHz	RTI0 Functional Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT1	PLL_CORE_CLK: HSDIV0_CLKOUT1	500 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
		CTPS_GENF0	CPSW3G CPTS GENF0 Clock	50 MHz	
RTI1	RTI1_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	RTI1 VBUSP Interface Clock
	RTI1_FCLK (RTI_CLK)	XTALCLK	External XTAL	25 MHz	RTI1 Functional Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT1	PLL_CORE_CLK: HSDIV0_CLKOUT1	500 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
		CTPS_GENF0	CPSW3G CPTS GENF0 Clock	50 MHz	

Table 13-239. *RTI* Clocks (continued)

This table table describes the module clocking signals.

Module Instance	Module Clock Input	Source Clock Signal	Source	Default Freq	Description
RTI2	RTI2_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	RTI2 VBUSP Interface Clock
	RTI2_FCLK (RTI_CLK)	XTALCLK	External XTAL	25 MHz	RTI2 Functional Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT1	PLL_CORE_CLK: HSDIV0_CLKOUT1	500 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
		CTPS_GENF0	CPSW3G CPTS GENF0 Clock	50 MHz	
RTI3	RTI3_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	RTI3 VBUSP Interface Clock
	RTI3_FCLK (RTI_CLK)	XTALCLK	External XTAL	25 MHz	RTI3 Functional Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT1	PLL_CORE_CLK: HSDIV0_CLKOUT1	500 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
		CTPS_GENF0	CPSW3G CPTS GENF0 Clock	50 MHz	

Table 13-239. *RTI* Clocks (continued)

This table table describes the module clocking signals.

Module Instance	Module Clock Input	Source Clock Signal	Source	Default Freq	Description
RTI4	RTI4_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	RTI4 VBUSP Interface Clock
	RTI4_FCLK (RTI_CLK)	XTALCLK	External XTAL	25 MHz	RTI4 Functional Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT1	PLL_CORE_CLK: HSDIV0_CLKOUT1	500 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
RTI5	RTI5_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	RTI5 VBUSP Interface Clock
	RTI5_FCLK (RTI_CLK)	XTALCLK	External XTAL	25 MHz	RTI5 Functional Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT1	PLL_CORE_CLK: HSDIV0_CLKOUT1	500 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	
RTI6	RTI6_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	RTI6 VBUSP Interface Clock
	RTI6_FCLK (RTI_CLK)	XTALCLK	External XTAL	25 MHz	RTI6 Functional Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT1	PLL_CORE_CLK: HSDIV0_CLKOUT1	500 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	

Table 13-239. *RTI* Clocks (continued)

This table table describes the module clocking signals.

Module Instance	Module Clock Input	Source Clock Signal	Source	Default Freq	Description
RTI7	RTI7_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	RTI7 VBUSP Interface Clock
	RTI7_FCLK (RTI_CLK)	XTALCLK	External XTAL	25 MHz	RTI7 Functional Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT1	PLL_CORE_CLK: HSDIV0_CLKOUT1	500 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External XTAL	25 MHz	

Table 13-240. *RTI* Resets

This table describes the module reset signals.

Module Instance	Module Reset Input	Source Reset Signal	Source	Description
RTI0	RTI0_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	RTI0 Asynchronous Reset
	RTI0_POR_RST	POR Reset (MOD_POR_RST)	Device Power-On Reset	RTI0 Power-On Reset
RTI1	RTI1_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	RTI1 Asynchronous Reset
	RTI1_POR_RST	POR Reset (MOD_POR_RST)	Device Power-On Reset	RTI1 Power-On Reset
RTI2	RTI2_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	RTI2 Asynchronous Reset
	RTI2_POR_RST	POR Reset (MOD_POR_RST)	Device Power-On Reset	RTI2 Power-On Reset
RTI3	RTI3_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	RTI3 Asynchronous Reset
	RTI3_POR_RST	POR Reset (MOD_POR_RST)	Device Power-On Reset	RTI3 Power-On Reset
RTI4	RTI4_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	RTI4 Asynchronous Reset
	RTI4_POR_RST	POR Reset (MOD_POR_RST)	Device Power-On Reset	RTI4 Power-On Reset
RTI5	RTI5_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	RTI5 Asynchronous Reset
	RTI5_POR_RST	POR Reset (MOD_POR_RST)	Device Power-On Reset	RTI5 Power-On Reset
RTI6	RTI6_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	RTI6 Asynchronous Reset
	RTI6_POR_RST	POR Reset (MOD_POR_RST)	Device Power-On Reset	RTI6 Power-On Reset

Table 13-240. *RTI* Resets (continued)

This table describes the module reset signals.

Module Instance	Module Reset Input	Source Reset Signal	Source	Description
RTI7	RTI7_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	RTI7 Asynchronous Reset
	RTI7_POR_RST	POR Reset (MOD_POR_RST)	Device Power-On Reset	RTI7 Power-On Reset

Table 13-241. *RTI* Interrupt Requests

This table describes the module interrupt requests.

Module Instance	Module Interrupt Signal	Destination Interrupt Input	Destination	Type	Description
RTI0	RTI0_INT_REQ_0	RTI0_INT_REQ_0	ALL R5FSS Cores	Pulse	RTI0 Status Event Interrupt
	RTI0_INT_REQ_1	RTI0_INT_REQ_1			
	RTI0_INT_REQ_2	RTI0_INT_REQ_2			
	RTI0_INT_REQ_3	RTI0_INT_REQ_3			
	RTI0_OVL_REQ_0	RTI0_OVERFLOW_LEVEL_0			RTI0 Counter Overflow Event Interrupt
	RTI0_OVL_REQ_1	RTI0_OVERFLOW_LEVEL_1			
RTI1	RTI1_INT_REQ_0	RTI1_INT_REQ_0	ALL R5FSS Cores	Pulse	RTI1 Status Event Interrupt
	RTI1_INT_REQ_1	RTI1_INT_REQ_1			
	RTI1_INT_REQ_2	RTI1_INT_REQ_2			
	RTI1_INT_REQ_3	RTI1_INT_REQ_3			
	RTI1_OVL_REQ_0	RTI1_OVERFLOW_LEVEL_0			RTI1 Counter Overflow Event Interrupt
	RTI1_OVL_REQ_1	RTI1_OVERFLOW_LEVEL_1			
RTI2	RTI2_INT_REQ_0	RTI2_INT_REQ_0	ALL R5FSS Cores	Pulse	RTI2 Status Event Interrupt
	RTI2_INT_REQ_1	RTI2_INT_REQ_1			
	RTI2_INT_REQ_2	RTI2_INT_REQ_2			
	RTI2_INT_REQ_3	RTI2_INT_REQ_3			
	RTI2_OVL_REQ_0	RTI2_OVERFLOW_LEVEL_0			RTI2 Counter Overflow Event Interrupt
	RTI2_OVL_REQ_1	RTI2_OVERFLOW_LEVEL_1			
RTI3	RTI3_INT_REQ_0	RTI3_INT_REQ_0	ALL R5FSS Cores	Pulse	RTI3 Status Event Interrupt
	RTI3_INT_REQ_1	RTI3_INT_REQ_1			
	RTI3_INT_REQ_2	RTI3_INT_REQ_2			
	RTI3_INT_REQ_3	RTI3_INT_REQ_3			
	RTI3_OVL_REQ_0	RTI3_OVERFLOW_LEVEL_0			RTI3 Counter Overflow Event Interrupt
	RTI3_OVL_REQ_1	RTI3_OVERFLOW_LEVEL_1			
RTI4	RTI4_INT_REQ_0	RTI4_INT_REQ_0	ALL R5FSS Cores	Pulse	RTI4 Status Event Interrupt
	RTI4_INT_REQ_1	RTI4_INT_REQ_1			
	RTI4_INT_REQ_2	RTI4_INT_REQ_2			
	RTI4_INT_REQ_3	RTI4_INT_REQ_3			
	RTI4_OVL_REQ_0	RTI4_OVERFLOW_LEVEL_0			RTI4 Counter Overflow Event Interrupt
	RTI4_OVL_REQ_1	RTI4_OVERFLOW_LEVEL_1			

Table 13-241. *RTI* Interrupt Requests (continued)

This table describes the module interrupt requests.

Module Instance	Module Interrupt Signal	Destination Interrupt Input	Destination	Type	Description
RTI5	RTI5_INT_REQ_0	RTI5_INT_REQ_0	ALL R5FSS Cores	Pulse	RTI5 Status Event Interrupt
	RTI5_INT_REQ_1	RTI5_INT_REQ_1			
	RTI5_INT_REQ_2	RTI5_INT_REQ_2			
	RTI5_INT_REQ_3	RTI5_INT_REQ_3			
	RTI5_OVL_REQ_0	RTI5_OVERFLOW_LEVEL_0			RTI5 Counter Overflow Event Interrupt
	RTI5_OVL_REQ_1	RTI5_OVERFLOW_LEVEL_1			
RTI6	RTI6_INT_REQ_0	RTI6_INT_REQ_0	ALL R5FSS Cores	Pulse	RTI6 Status Event Interrupt
	RTI6_INT_REQ_1	RTI6_INT_REQ_1			
	RTI6_INT_REQ_2	RTI6_INT_REQ_2			
	RTI6_INT_REQ_3	RTI6_INT_REQ_3			
	RTI6_OVL_REQ_0	RTI6_OVERFLOW_LEVEL_0			RTI6 Counter Overflow Event Interrupt
	RTI6_OVL_REQ_1	RTI6_OVERFLOW_LEVEL_1			
RTI7	RTI7_INT_REQ_0	RTI7_INT_REQ_0	ALL R5FSS Cores	Pulse	RTI7 Status Event Interrupt
	RTI7_INT_REQ_1	RTI7_INT_REQ_1			
	RTI7_INT_REQ_2	RTI7_INT_REQ_2			
	RTI7_INT_REQ_3	RTI7_INT_REQ_3			
	RTI7_OVL_REQ_0	RTI7_OVERFLOW_LEVEL_0			RTI7 Counter Overflow Event Interrupt
	RTI7_OVL_REQ_1	RTI7_OVERFLOW_LEVEL_1			

Table 13-242. *RTI* DMA Requests

This table describes the module DMA requests.

Module Instance	Module DMA Event	Destination DMA Event Input	Destination	Type	Description
RTI0	RTI0_DMA_0	RTI0_DMA_REQ_0	EDMA Crossbar (EDMA_XBAR)	Pulse	RTI0 DMA Request
	RTI0_DMA_1	RTI0_DMA_REQ_1			
	RTI0_DMA_2	RTI0_DMA_REQ_2			
	RTI0_DMA_3	RTI0_DMA_REQ_3			
RTI1	RTI1_DMA_0	RTI1_DMA_REQ_0			RTI1 DMA Request
	RTI1_DMA_1	RTI1_DMA_REQ_1			
	RTI1_DMA_2	RTI1_DMA_REQ_2			
	RTI1_DMA_3	RTI1_DMA_REQ_3			
RTI2	RTI2_DMA_0	RTI2_DMA_REQ_0			RTI2 DMA Request
	RTI2_DMA_1	RTI2_DMA_REQ_1			
	RTI2_DMA_2	RTI2_DMA_REQ_2			
	RTI2_DMA_3	RTI2_DMA_REQ_3			
RTI3	RTI3_DMA_0	RTI3_DMA_REQ_0			RTI3 DMA Request
	RTI3_DMA_1	RTI3_DMA_REQ_1			
	RTI3_DMA_2	RTI3_DMA_REQ_2			
	RTI3_DMA_3	RTI3_DMA_REQ_3			
RTI4	RTI4_DMA_0	RTI4_DMA_REQ_0			RTI4 DMA Request
	RTI4_DMA_1	RTI4_DMA_REQ_1			
	RTI4_DMA_2	RTI4_DMA_REQ_2			
	RTI4_DMA_3	RTI4_DMA_REQ_3			
RTI5	RTI5_DMA_0	RTI5_DMA_REQ_0			RTI5 DMA Request
	RTI5_DMA_1	RTI5_DMA_REQ_1			
	RTI5_DMA_2	RTI5_DMA_REQ_2			
	RTI5_DMA_3	RTI5_DMA_REQ_3			
RTI6	RTI6_DMA_0	RTI6_DMA_REQ_0			RTI6 DMA Request
	RTI6_DMA_1	RTI6_DMA_REQ_1			
	RTI6_DMA_2	RTI6_DMA_REQ_2			
	RTI6_DMA_3	RTI6_DMA_REQ_3			
RTI7	RTI7_DMA_0	RTI7_DMA_REQ_0			RTI7 DMA Request
	RTI7_DMA_1	RTI7_DMA_REQ_1			
	RTI7_DMA_2	RTI7_DMA_REQ_2			
	RTI7_DMA_3	RTI7_DMA_REQ_3			

Table 13-243. *RTI* Capture Events

This table describes the module capture events.

Module Instance	Module Capture Event Input	Capture Event Source Signal	Source	Type	Description
RTI0	RTI0_CAPEVT_0	SoC_TIMESYNC_XBAROUT_2	SoC Time Sync Crossbar (TIMESYNC_XBAR)	Pulse	RTI0 Counter Capture Input Event
	RTI0_CAPEVT_1	SoC_TIMESYNC_XBAROUT_3			
RTI1	RTI1_CAPEVT_0	SoC_TIMESYNC_XBAROUT_4			RTI1 Counter Capture Input Event
	RTI1_CAPEVT_1	SoC_TIMESYNC_XBAROUT_5			
RTI2	RTI2_CAPEVT_0	SoC_TIMESYNC_XBAROUT_6			RTI2 Counter Capture Input Event
	RTI2_CAPEVT_1	SoC_TIMESYNC_XBAROUT_7			
RTI3	RTI3_CAPEVT_0	SoC_TIMESYNC_XBAROUT_8			RTI3 Counter Capture Input Event
	RTI3_CAPEVT_1	SoC_TIMESYNC_XBAROUT_9			
RT4	RTI3_CAPEVT_0	SoC_TIMESYNC_XBAROUT_12			RTI4 Counter Capture Input Event
	RTI3_CAPEVT_1	SoC_TIMESYNC_XBAROUT_13			
RTI5	RTI3_CAPEVT_0	SoC_TIMESYNC_XBAROUT_14			RTI5 Counter Capture Input Event
	RTI3_CAPEVT_1	SoC_TIMESYNC_XBAROUT_15			
RTI6	RTI3_CAPEVT_0	SoC_TIMESYNC_XBAROUT_16			RTI6 Counter Capture Input Event
	RTI3_CAPEVT_1	SoC_TIMESYNC_XBAROUT_17			
RTI7	RTI3_CAPEVT_0	SoC_TIMESYNC_XBAROUT_18			RTI7 Counter Capture Input Event
	RTI3_CAPEVT_1	SoC_TIMESYNC_XBAROUT_19			

Note

For more information on the interconnects, see the *System Interconnect* chapter.

For more information on the power, reset and clock management, see the corresponding sections within the *Device Configuration* chapter.

For more information on the device interrupt controllers, see the *Interrupt Controllers* chapter.

13.5.1.3 WWDT Integration

There are 4x WWDT modules integrated in the device. The diagram and tables below show the device integration details.

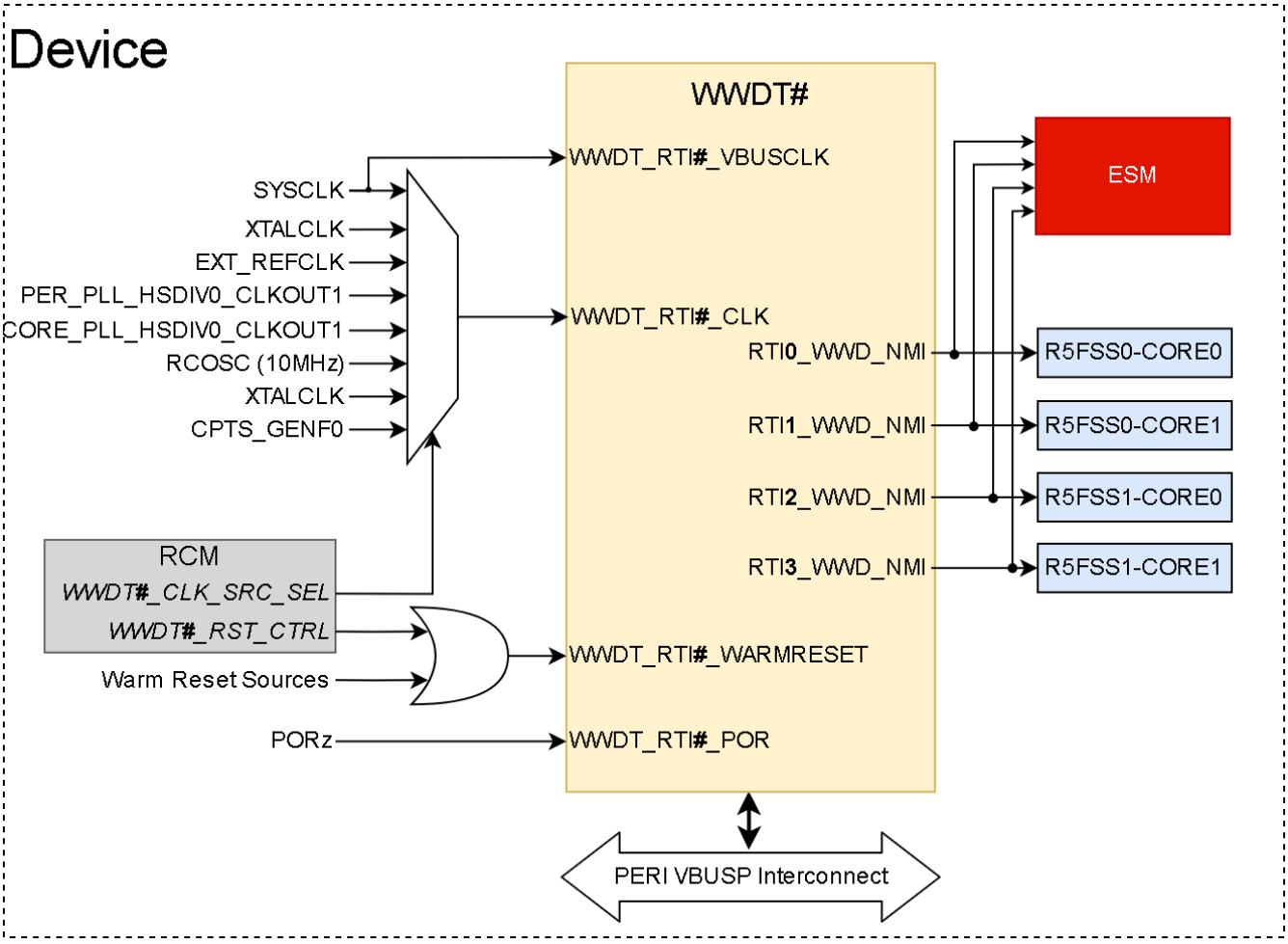


Figure 13-216. WWDT Integration

The tables below summarize the integration of WWDT# (where # = 0, 1, 2, 3) in the device.

Each WWDT# instance is supplied by dedicated WWDTCLK# mux.

Table 13-244. *WWDT* Device Integration

This table describes the module device integration details.

Module Instance	Device Allocation	SoC Interconnect
WWDT0	✓	VBUSP CORE Interconnect
WWDT1	✓	VBUSP CORE Interconnect
WWDT2	✓	VBUSP CORE Interconnect
WWDT3	✓	VBUSP CORE Interconnect

Table 13-245. *WWDT* Clocks

This table describes the module clocking signals.

Module Instance	Module Clock Input	Source Clock Signal	Source	Default Freq	Description
WWDT0	WWDT0_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	WWDT0 VBUSP Interface Clock
	WWDT0_FCLK (WWDT_CLK)	XTALCLK	External Crystal (XTAL)	25 MHz	WWDT0 Functional Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT1	PLL_CORE_CLK: HSDIV0_CLKOUT1	500 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External Crystal (XTAL)	25 MHz	
		CTPS_GENF0	CPSW3G CPTS GENF0 Clock	50 MHz	
WWDT1	WWDT1_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	WWDT1 VBUSP Interface Clock
	WWDT1_FCLK (WWDT_CLK)	XTALCLK	External Crystal (XTAL)	25 MHz	WWDT1 Functional Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT1	PLL_CORE_CLK: HSDIV0_CLKOUT1	500 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External Crystal (XTAL)	25 MHz	
		CTPS_GENF0	CPSW3G CPTS GENF0 Clock	50 MHz	

Table 13-245. *WWDT* Clocks (continued)

This table describes the module clocking signals.

Module Instance	Module Clock Input	Source Clock Signal	Source	Default Freq	Description
WWDT2	WWDT2_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	WWDT2 VBUSP Interface Clock
	WWDT2_FCLK (WWDT_CLK)	XTALCLK	External Crystal (XTAL)	25 MHz	WWDT2 Functional Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT1	PLL_CORE_CLK: HSDIV0_CLKOUT1	500 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External Crystal (XTAL)	25 MHz	
		CTPS_GENF0	CPSW3G CPTS GENF0 Clock	50 MHz	
WWDT3	WWDT3_ICLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	WWDT3 VBUSP Interface Clock
	WWDT3_FCLK (WWDT_CLK)	XTALCLK	External Crystal (XTAL)	25 MHz	WWDT3 Functional Clock
		EXT_REFCLK	External Reference Clock (EXT_REFCLK)	100 MHz	
		SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	
		DPLL_PER_HSDIV0_CLKOUT1	PLL_PER_CLK: HSDIV0_CLKOUT1	192 MHz	
		DPLL_CORE_HSDIV0_CLKOUT1	PLL_CORE_CLK: HSDIV0_CLKOUT1	500 MHz	
		RCCLK10M	Internal 10 MHz RC Oscillator (RCCLK10M)	10 MHz	
		XTALCLK	External Crystal (XTAL)	25 MHz	
		CTPS_GENF0	CPSW3G CPTS GENF0 Clock	50 MHz	

Table 13-246. *WWDT* Resets

This table describes the module reset signals.

Module Instance	Module Reset Input	Source Reset Signal	Source	Description
WWDT0	WWDT0_RST	Warm Reset (MOD_G_RST)	RCM Reset Control Register + Warm Reset Sources	WWDT0 Asynchronous Reset
	WWDT0_POR_RST	POR Reset (MOD_POR_RST)	Device Power-On Reset	WWDT0 Power-On Reset

Table 13-246. *WWDT* Resets (continued)

This table describes the module reset signals.

Module Instance	Module Reset Input	Source Reset Signal	Source	Description
WWDT1	WWDT1_RST	Warm Reset (MOD_G_RST)	RCM Reset Control Register + Warm Reset Sources	WWDT1 Asynchronous Reset
	WWDT1_POR_RST	POR Reset (MOD_POR_RST)	Device Power-On Reset	WWDT1 Power-On Reset
WWDT2	WWDT2_RST	Warm Reset (MOD_G_RST)	RCM Reset Control Register + Warm Reset Sources	WWDT2 Asynchronous Reset
	WWDT2_POR_RST	POR Reset (MOD_POR_RST)	Device Power-On Reset	WWDT2 Power-On Reset
WWDT3	WWDT3_RST	Warm Reset (MOD_G_RST)	RCM Reset Control Register + Warm Reset Sources	WWDT3 Asynchronous Reset
	WWDT3_POR_RST	POR Reset (MOD_POR_RST)	Device Power-On Reset	WWDT3 Power-On Reset

Table 13-247. *WWDT* Interrupt Requests

This table describes the module interrupt requests.

Module Instance	Module Interrupt Signal	Destination Interrupt Input	Destination	Type	Description
WWDT0	WWDT0_NMI_REQ	ESM0_PLS_IN_0	ESM0	Pulse	WWDT0 Window Watchdog Violation Non-Maskable Interrupt (NMI) Event
		R5FSS0_0_VIM_128	R5FSS0_CORE0		
WWDT1	WWDT1_NMI_REQ	ESM0_PLS_IN_1	ESM0	Pulse	WWDT1 Non-Maskable Interrupt (NMI) Event
		R5FSS0_1_VIM_128	R5FSS0_CORE1		
WWDT2	WWDT2_NMI_REQ	ESM0_PLS_IN_2	ESM0	Pulse	WWDT2 Non-Maskable Interrupt (NMI) Event
		R5FSS1_0_VIM_128	R5FSS1_CORE0		
WWDT3	WWDT3_NMI_REQ	ESM0_PLS_IN_3	ESM0	Pulse	WWDT3 Non-Maskable Interrupt (NMI) Event
		R5FSS1_1_VIM_128	R5FSS1_CORE1		

Note

For more information on the interconnects, see the *System Interconnect* chapter.

For more information on the power, reset and clock management, see the corresponding sections within the *Device Configuration* chapter.

For more information on the device interrupt controllers, see the *Interrupt Controllers* chapter.

13.5.1.4 RTI Functional Description

The RTI# and WWDT# (where # = 0, 1, 2, 3) modules are hereinafter referred to as RTI, RTI_WWDT, or RTI/WWDT.

13.5.1.4.1 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

The following section only applies to the WWDT defined modules.

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.

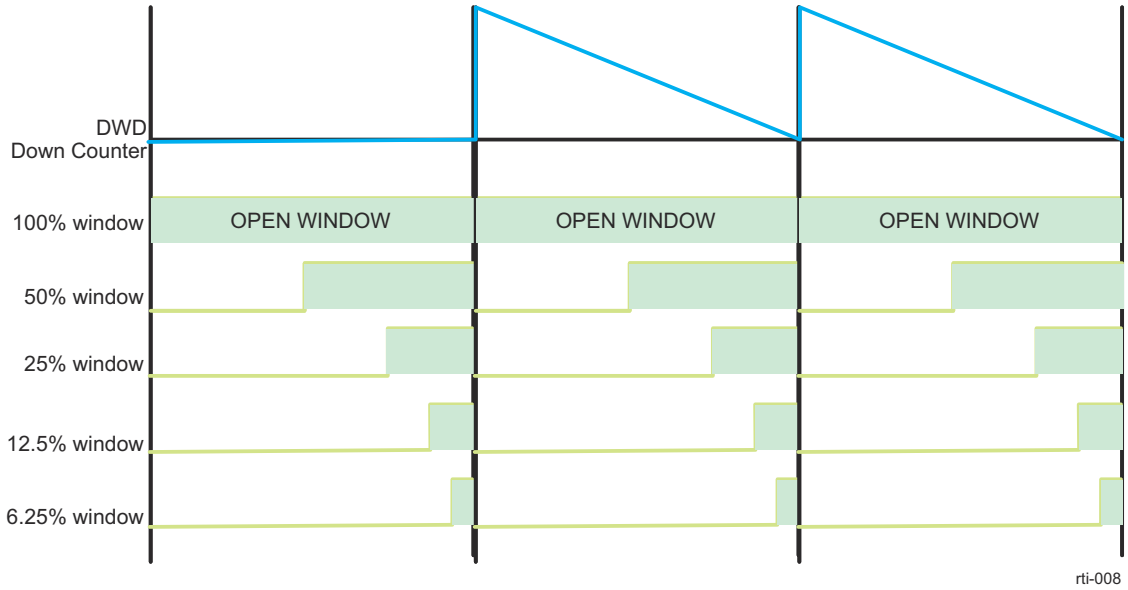


Figure 13-217. RTI Digital Windowed Watchdog Timing Example

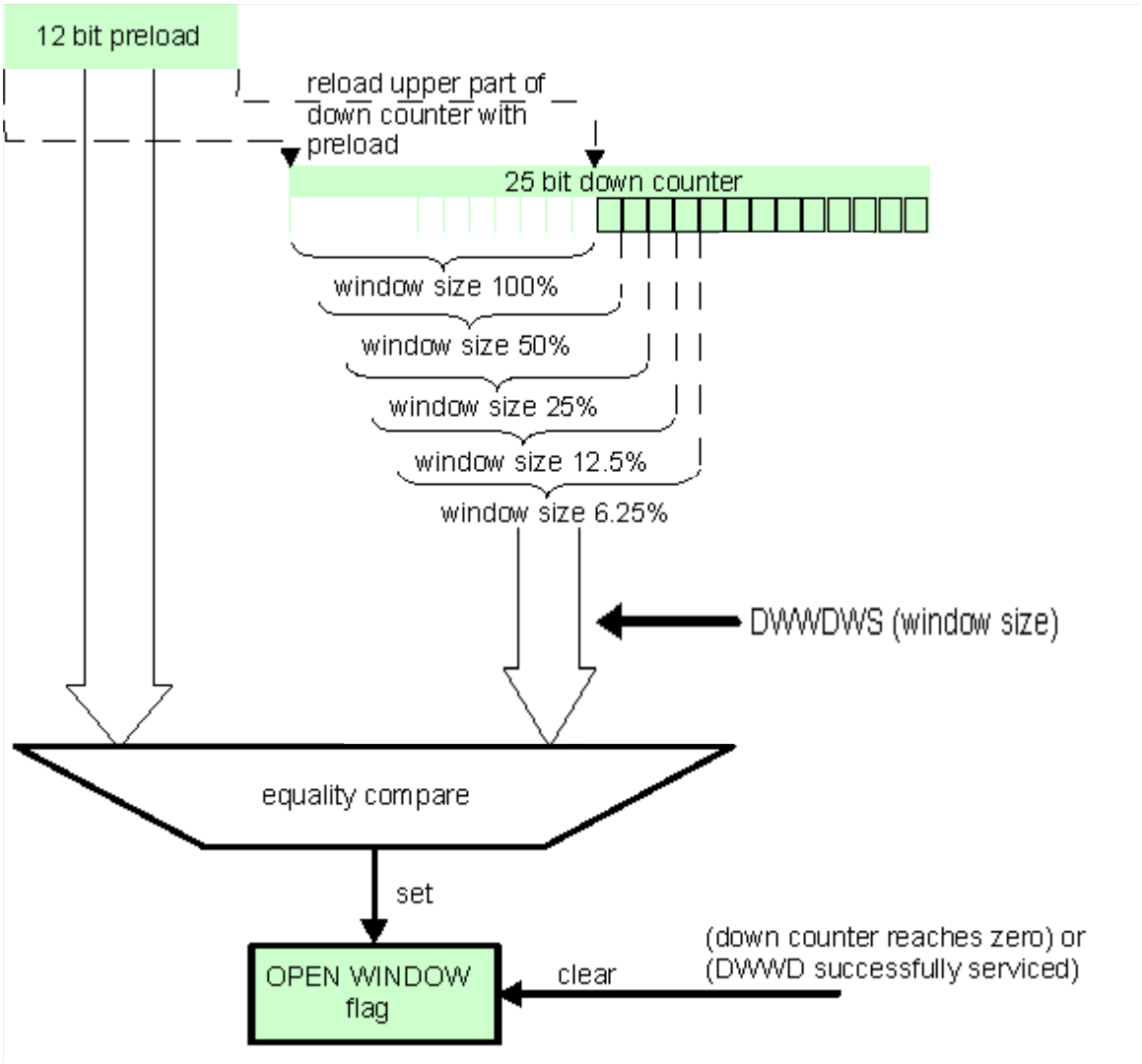


Figure 13-218. RTI Digital Windowed Watchdog Operation Block Diagram

13.5.1.4.1.1 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.

13.5.1.4.2 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*.

Note

The following section only applies to the WWDT defined modules.

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 13-219 shows the digital watchdog functional block.

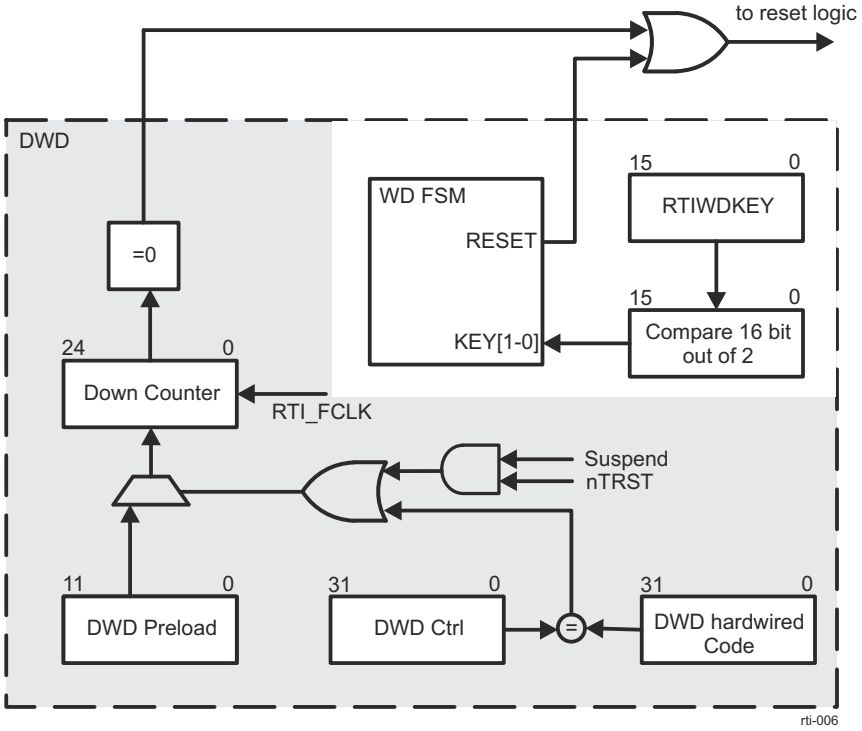


Figure 13-219. RTI Digital Watchdog Functional Block Diagram

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.

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.

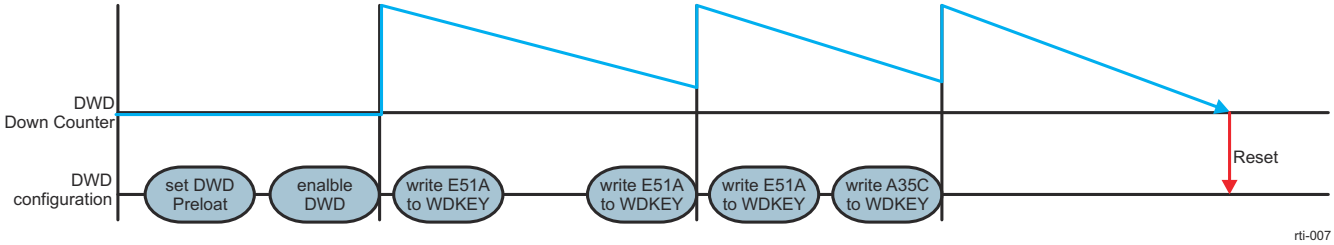


Figure 13-220. RTI Digital Watchdog Operation

The expiration time of the DWD Down Counter can be determined with following equation:

$$t_{exp} = (RTI\_DWDPRLD + 1) \times 2^{13} / RTI\_FCLK \quad (30)$$

where RTI_DWDPRLD = 0...4095

13.5.1.4.3 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 13-221 shows the RTI module counter blocks. The RTI module supports two counter blocks.

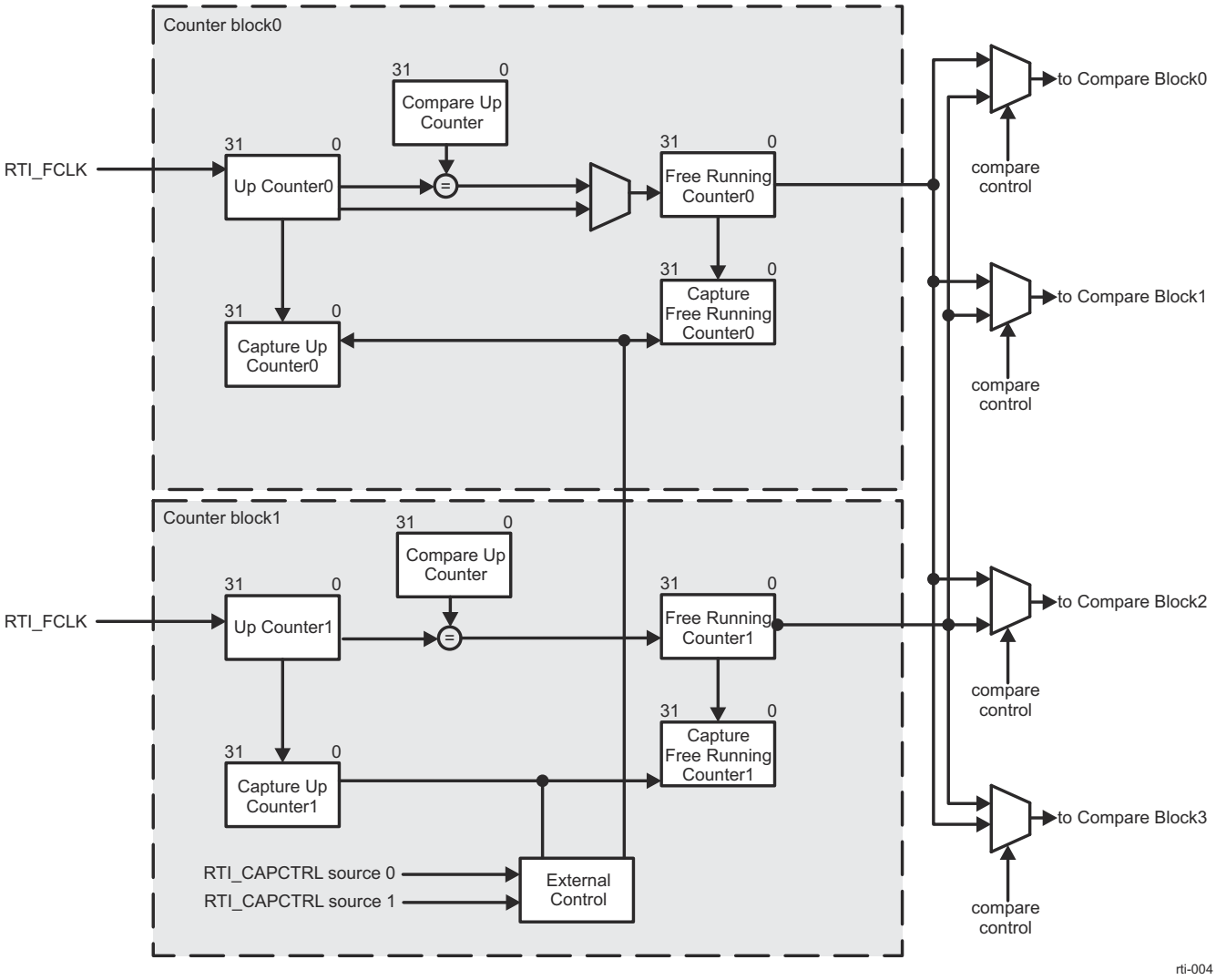


Figure 13-221. RTI Counters Block Diagram

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 values have to be determined, read the Free Running Counter first. This makes sure that at the time when the counter register is read, the Up Counter value has been stored into the counter register. The second read is then performed 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 event sources come from the interrupt manager, enabling the device to generate a capture event when a peripheral module generates an interrupt. The peripheral which generates an RTI capture 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 must be in the same order as the counter register reads. So, the CAFRCx must be read first, and then the CAUCx registers are read after the CAFRCx value has been determined. While CAFRCx is read, the CAUCx value is loaded into a shadow register to maintain data consistency, in case a capture event

happens during the two reads. If the application fails to read the two registers before a second capture event happens, the previous data is overwritten.

Figure 13-222 shows the block diagram for one compare block. The RTI module supports four compare blocks.

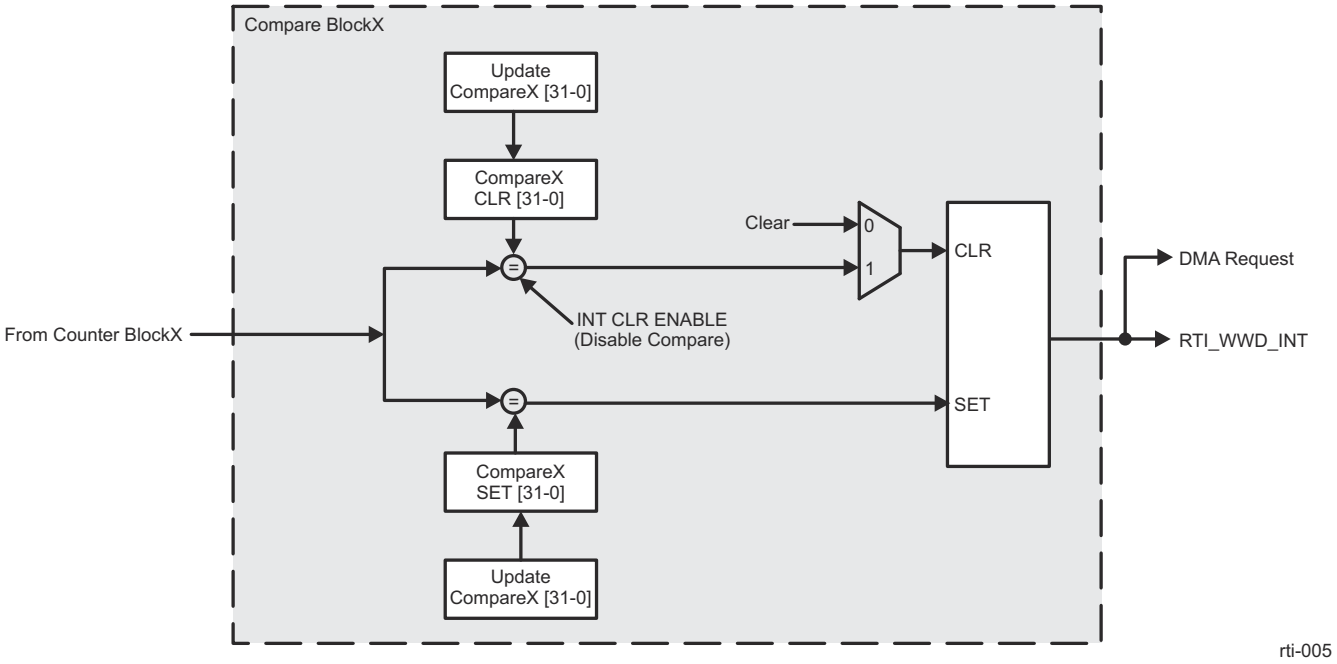


Figure 13-222. RTI Compare Block Diagram

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.

13.5.1.5 RTI/WWDT Programming Guide

Driver Information

Driver features are available at the RTI driver page and WWDT driver page.

Software API Information

The RTI/WWDT driver provides an API to configure the RTI/WWDT module. Full documentation is located on APIs for RTI and APIs for WWDT.

Example Usage

The below links shows an example on how to use RTI/WWDT.

- RTI:
  - RTI LED Blink
- WWDT:
  - Watchdog Reset Mode
  - Watchdog Interrupt Mode

13.6 Internal Diagnostics Modules

This section describes the internal diagnostics modules in the device.

13.6.1 Dual Clock Comparator (DCC)

This section describes the Dual Clock Comparator (DCC) modules in the device.

13.6.1.1 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.

The device has four instances of DCC modules.

13.6.1.1.1 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 generation
- Error signal generation when one of the clocks is out of specification
- Clock frequency measurement
- Ability to continue the check in continuous mode despite the error. This is programmable capability.
- Ability to register up-to 4 readings of the error for all associated counts in FIFO.
- Synchronized handoffs between counting clock domains, processing clock domain or reporting clock domains (bus clock).

13.6.1.1.2 DCC Not Supported Features

The DCC does not support the following features:

- Debug suspend functionality depreciated . Software will have to disable DCCs if it starts messing with clocks during debug.

13.6.1.2 DCC Integration

This section describes the DCC integration in the device, including information about clocks, resets, and hardware requests.

13.6.1.2.1 DCC Integration

There are 4x DCC modules integrated in the device. The diagram below provides a visual representation of the device integration details.

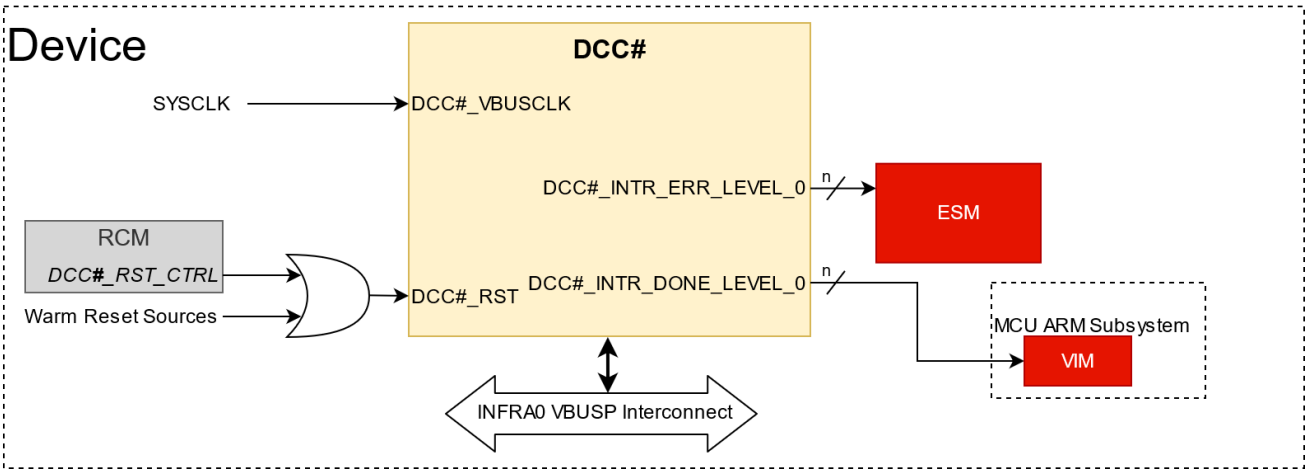


Figure 13-223. DCC Integration Diagram

The tables below summarize the device integration details of DCC.

Table 13-248. DCC Device Integration

This table describes the module device integration details.

Module Instance	Device Allocation	SoC Interconnect
DCC0	✓	INFRA0 VBUSP Interconnect
DCC1	✓	INFRA0 VBUSP Interconnect
DCC2	✓	INFRA0 VBUSP Interconnect
DCC3	✓	INFRA0 VBUSP Interconnect

Table 13-249. DCC Clocks

This table describes the module clocking signals.

Module Instance	Module Clock Input	Source Clock Signal	Source	Default Freq	Description
DCC0	DCC0_CLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	DCC0 Interface Clock
DCC1	DCC1_CLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	DCC1 Interface Clock
DCC2	DCC2_CLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	DCC2 Interface Clock
DCC3	DCC3_CLK (VBUSP_CLK)	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	DCC3 Interface Clock

Table 13-250. DCC Resets

This table describes the module reset signals.

Module Instance	Module Reset Input	Source Reset Signal	Source	Description
DCC0	DCC0_RST	Warm Reset (SYNC_RST_N)	RCM + Warm Reset Sources	Synchronous Assertion Reset, Active Low

Table 13-250. DCC Resets (continued)

This table describes the module reset signals.

Module Instance	Module Reset Input	Source Reset Signal	Source	Description
DCC1	DCC1_RST	Warm Reset (SYNC_RST_N)	RCM + Warm Reset Sources	Synchronous Assertion Reset, Active Low
DCC2	DCC2_RST	Warm Reset (SYNC_RST_N)	RCM + Warm Reset Sources	Synchronous Assertion Reset, Active Low
DCC3	DCC3_RST	Warm Reset (SYNC_RST_N)	RCM + Warm Reset Sources	Synchronous Assertion Reset, Active Low

Table 13-251. DCC Interrupt Requests

This table describes the module interrupt requests.

Module Instance	Module Interrupt Signal	Destination Interrupt Input	Destination	Type	Description
DCC0	DCC0_DONE	DCC0_DONE	ALL R5FSS Cores	Level	DCC0 Done Interrupt
	DCC0_ERROR	DCC0_ERROR	ESM	Level	DCC0 Error Interrupt
DCC1	DCC1_DONE	DCC1_DONE	ALL R5FSS Cores	Level	DCC1 Done Interrupt
	DCC1_ERROR	DCC1_ERROR	ESM	Level	DCC1 Error Interrupt
DCC2	DCC2_DONE	DCC2_DONE	ALL R5FSS Cores	Level	DCC2 Done Interrupt
	DCC2_ERROR	DCC2_ERROR	ESM	Level	DCC2 Error Interrupt
DCC3	DCC3_DONE	DCC3_DONE	ALL R5FSS Cores	Level	DCC3 Done Interrupt
	DCC3_ERROR	DCC3_ERROR	ESM	Level	DCC3 Error Interrupt

Note

For more information on the interconnects, see the *System Interconnect* chapter.

For more information on power, reset, and clock management, see the corresponding sections within the *Device Configuration* chapter.

For more information on the device interrupt controllers, see the *Interrupt Controllers* chapter.

13.6.1.3 DCC Functional Description

Figure 13-224. DCC Functional Block Diagram

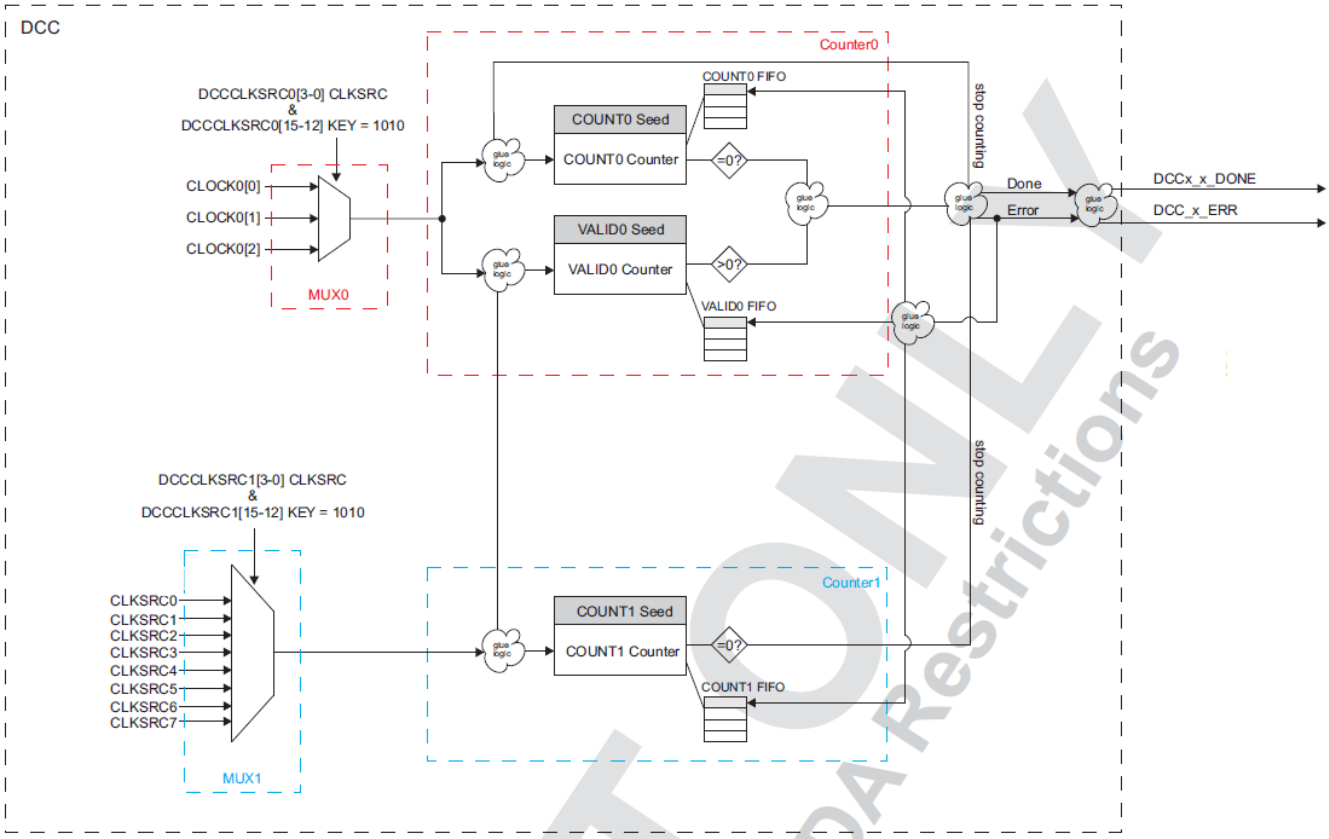


Figure 13-225. DCC Functional Block Diagram

13.6.1.3.1 DCC Counter Operation

DCC has two sets of counters with each set having programmable clock selection.

- The first set has two counters COUNT0 and VALID0. These operate from Clock0 (reference clock).
- The second set has one counter COUNT1 that is operated from Clock1 (measured clock). The selection of input clock from list of different counters increases the utility of DCC for debug and test across different clock sources available on the device.

COUNT0 and COUNT1 are configured based on the ratio between the frequencies of Clock0 and Clock1 (Clock1 frequency * COUNT0 = Clock0 frequency * COUNT1). Further, the tunable counter VALID0 on the Clock0 (reference clock) defines the window of margin for COUNT1 to end after COUNT0. This COUNT1 needs to complete within valid window for operation where clock relationship is as expected.

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.

Any of these errors causes the counters to stop counting by default. An application may then read out the counter values to help determine what caused the error. It would take multiple clocks (2-3 in each clock domain i.e. source and VBUSP_CLK) to stop the counters due to the cross-clock domain synchronizations. Counters can also be configured in a mode to reload and continue down-counting despite error so successive error event is not missed. Error is reported as exception and application is expected to read the counter values for determining quantum and direction of error.

Note

Reads of the counter value may not be exact since the read operation is synchronized to the VBUSP_CLK

Reloads or restarts occur under the following 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).
- In continuous mode without any error.
- In continuous mode with error, given CONT_ON_ERR is set, upon which counters restart counting as soon as the error is hit and the error counts are archived.

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 to be captured to the VBUSP_CLK domain. There is 1 VBUSP_CLK period uncertainty on either side of the fixed width counting window (VALID0) in generating the error signal since the counters work in a different clock domain. This should be accounted for, when setting the count value for VALID0.

Operating the DCC with '0' in the COUNTSEED1 or COUNTSEED0 or VALIDSEED0 register will result in undefined operation

Figure 13-226 through Figure 13-230 shows examples of counters relationship and error generation.

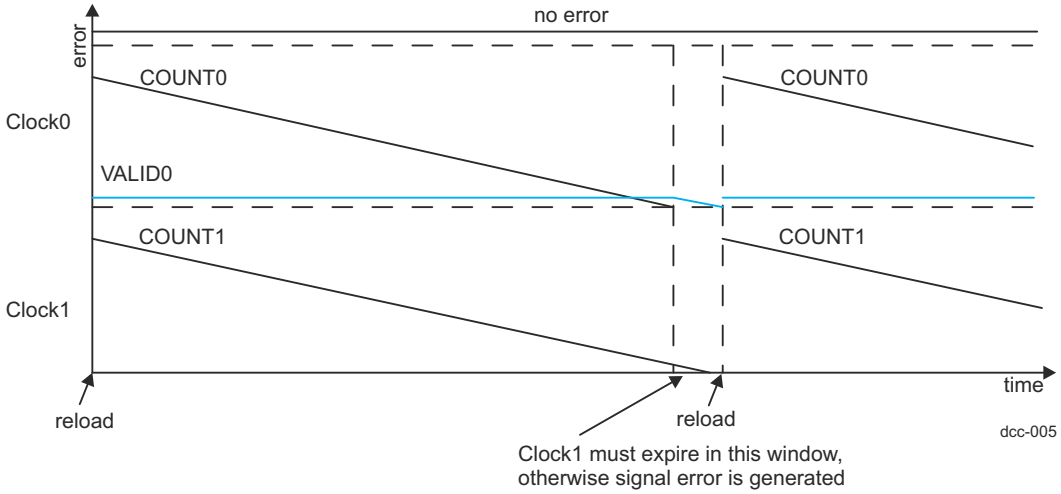


Figure 13-226. DCC Clock0 and Clock1 With no Error

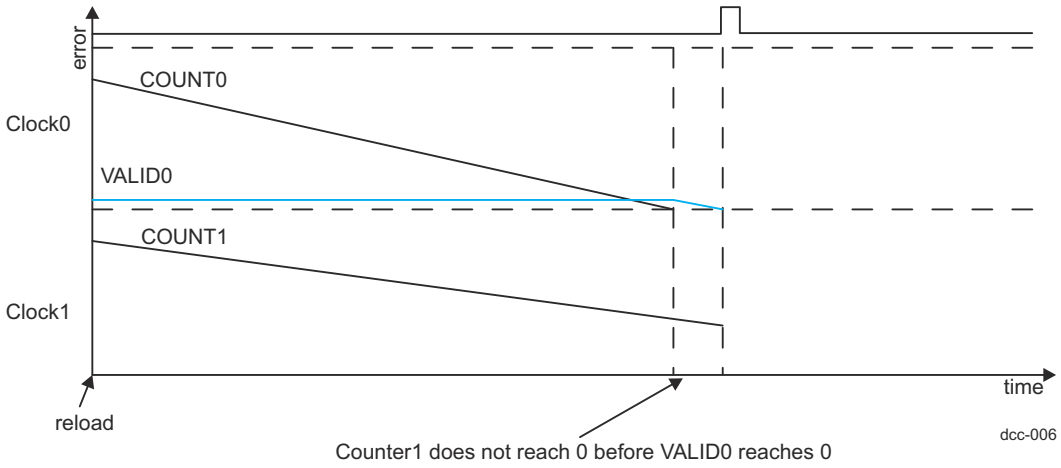


Figure 13-227. DCC Clock1 slower than Clock0 results in an error and stops counting

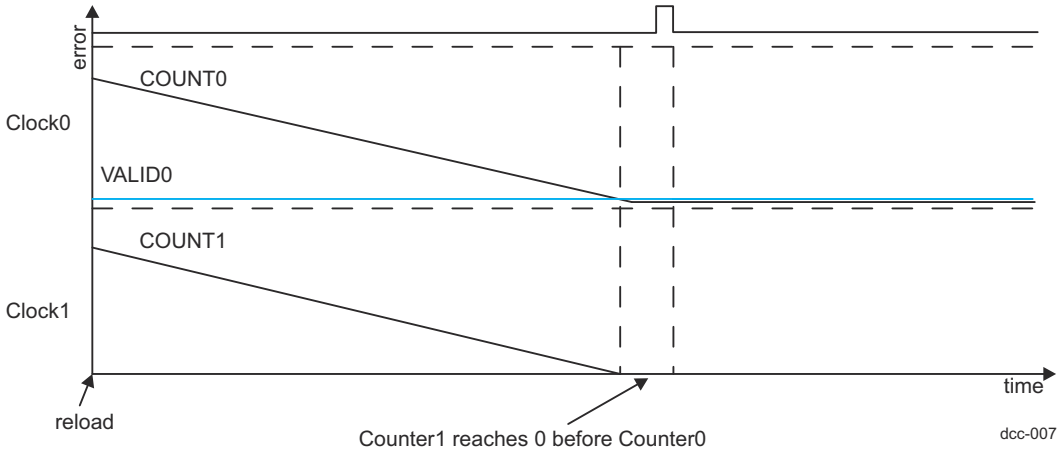


Figure 13-228. DCC Clock1 faster than Clock0 results in an error and stops counting

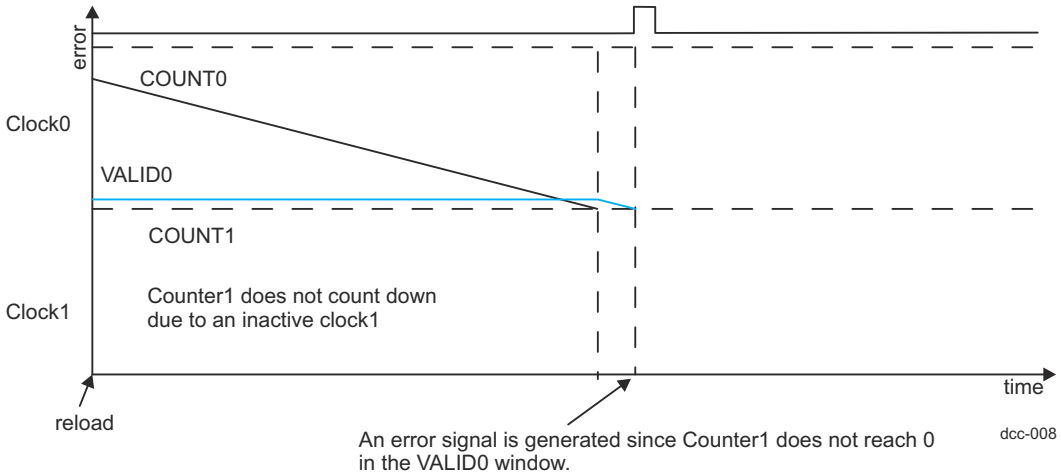


Figure 13-229. DCC Clock1 not present results in an error and stops counting

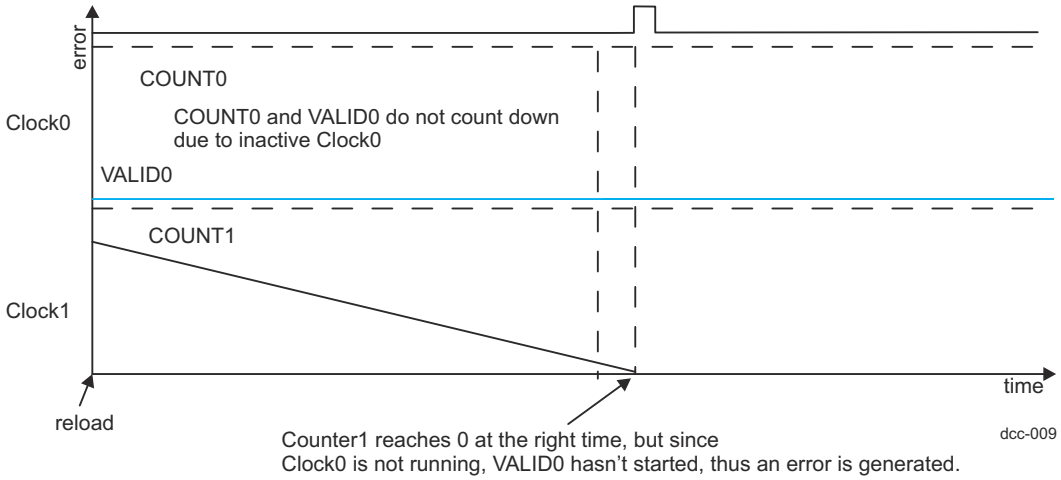


Figure 13-230. DCC Clock0 not present results in an error and stops counting

13.6.1.3.2 DCC Clock Sources

DCC0 - DCC1 Input Source Clock Mapping and DCC2 - DCC3 Input Source Clock Mapping summarizes the DCC input source clock options for the device.

Table 13-252. DCC0 - DCC1 Input Source Clock Mapping

		MAIN_DCC0												MAIN_DCC1											
		Input0				Input1								Input0				Input1							
		MUX0				MUX1								MUX0				MUX1							
DCC_CLKSRC0 / DCC_CLKSRC1 value:		0	1	2	3	0	1	2	3	4	5	6	7	0	1	2	3	0	1	2	3	4	5	6	7
Clock Source:	**Input:**	CLK0				CLK1								CLK0				CLK1							
XTALCLK	Crystal Clock	✓							✓					✓											
RCCLK10M	Internal 10 MHz RC Oscilator. Always on			✓												✓					✓				
EXT_REFCLK	External Ref Clock		✓							✓					✓										
RCCLK32K	32 KHz RC Clock				✓						✓						✓								
PLL_CORE_CLKOUT (PLL_CORE)																									
DPLL_CORE_HSDIV0_CLKOUT0	Root clock for Processor SS and Interconnect (Not Mapped to DCC - covered by SYS_CLK below)																								
DPLL_CORE_HSDIV0_CLKOUT1	CPSW/ICSS RGMII/GMII Clock																		✓						
PLL_PER_CLKOUT (PLL_PER)																									
DPLL_PER_HSDIV0_CLKOUT0	UART 5 Mbps Clocking																	✓							
DPLL_PER_HSDIV0_CLKOUT1	Peripheral Clocking																			✓					
Other IP Clocks																									
R5FSS0_CLK	R5F Cluster 0 Clock						✓																		
R5SFS1_CLK	R5F Cluster 1 Clock							✓																	
SYS_CLK	Interconnect System Clock					✓																			
WDT0_CLK	Watch Dog Timer																								
WDT1_CLK	Watch Dog Timer																								
WDT2_CLK	Watch Dog Timer																								
WDT3_CLK	Watch Dog Timer																								
MCAN0_CLK	MCAN Clock																								
MCAN1_CLK	MCAN Clock																								

Table 13-252. DCC0 - DCC1 Input Source Clock Mapping (continued)

		MAIN_DCC0												MAIN_DCC1											
		Input0				Input1								Input0				Input1							
		MUX0				MUX1								MUX0				MUX1							
DCC_CLKSRC0 / DCC_CLKSRC1 value:		0	1	2	3	0	1	2	3	4	5	6	7	0	1	2	3	0	1	2	3	4	5	6	7
Clock Source:	Input:	CLK0				CLK1								CLK0				CLK1							
TEMPSENSE_32K_CLK	32 KHz Clock (divided down from XTALCLK)																								
RMII1_REFCLK	IO Reference Clock Input																								
RMII2_REFCLK	IO Reference Clock Input																								
RGMII1_RXC	IO Receive Clock Input																								
RGMII2_RXC	IO Receive Clock Input																								
MII1_RXCLK	IO Receive Clock Input																								
MII2_RXCLK	IO Receive Clock Input																								
PR0_MII0_RXCLK	IO Receive Clock Input																								
PR0_MII1_RXCLK	IO Receive Clock Input																								
FSI0_RX_CLK	IO Receive Clock Input																					✓			
FSI1_RX_CLK	IO Receive Clock Input																						✓		
FSI2_RX_CLK	IO Receive Clock Input																							✓	
FSI3_RX_CLK	IO Receive Clock Input																								✓

Table 13-253. DCC2 - DCC3 Input Source Clock Mapping

		MAIN_DCC2												MAIN_DCC3											
		Input0				Input1								Input0				Input1							
		MUX0				MUX1								MUX0				MUX1							
DCC_CLKSRC0 / DCC_CLKSRC1 value:		[0, 3-F]	1	2	0	1	2	3	4	5	6	7		[0, 3-F]	1	2	0	1	2	3	4	5	6	7	
Clock Source:	Input:	CLK0			CLK1									CLK0			CLK1								
XTALCLK	Crystal Clock	✓												✓											
RCCLK10M	Internal 10 MHz RC Oscilator. Always on			✓												✓									
EXT_REFCLK	External Ref Clock		✓												✓										
RCCLK32K	32 KHz RC Clock																								
PLL_CORE_CLKOUT (PLL_CORE)																									
DPLL_CORE_HSDIV0_CLKOUT0	Root clock for Processor SS and Interconnect (Not Mapped to DCC - covered by SYS_CLK below)																								
DPLL_CORE_HSDIV0_CLKOUT1	CPSW/ICSS RGMII/ GMII Clock																								
PLL_PER_CLKOUT (PLL_PER)																									
DPLL_PER_HSDIV0_CLKOUT0	UART 5 Mbps Clocking																								
DPLL_PER_HSDIV0_CLKOUT1	Peripheral Clocking																								
Other IP Clocks																									
R5SS0_CLK	R5 Cluster 0 Clock																								
R5SS1_CLK	R5 Cluster 1 Clock																								
SYS_CLK	Interconnect System Clock				✓																				
WDT0_CLK	Watch Dog Timer					✓																			

Table 13-253. DCC2 - DCC3 Input Source Clock Mapping (continued)

		MAIN_DCC2											MAIN_DCC3									
		Input0			Input1								Input0			Input1						
		MUX0			MUX1								MUX0			MUX1						
DCC_CLKSRC0 / DCC_CLKSRC1 value:		[0, 3-F]	1	2	0	1	2	3	4	5	6	7	[0, 3-F]	1	2	0	1	2	3	4	5	6	7
Clock Source:	**Input:**	CLK0			CLK1								CLK0			CLK1							
WDT1_CLK	Watch Dog Timer						✓																
WDT2_CLK	Watch Dog Timer							✓															
WDT3_CLK	Watch Dog Timer								✓														
MCAN0_CLK	MCAN Clock									✓													
MCAN1_CLK	MCAN Clock										✓												
TEMPSENSE_32K_CLK	32 KHz Clock (divided down from XTALCLK)											✓											
RMII1_REFCLK	IO Reference Clock Input															✓							
RMII2_REFCLK	IO Reference Clock Input																✓						
RGMII1_RXC	IO Receive Clock Input																	✓					
RGMII2_RXC	IO Receive Clock Input																		✓				
MII1_RXCLK	IO Receive Clock Input																			✓			
MII2_RXCLK	IO Receive Clock Input																				✓		
PR0_MII0_RXCLK	IO Receive Clock Input																					✓	
PR0_MII1_RXCLK	IO Receive Clock Input																						✓
FSI0_RX_CLK	IO Receive Clock Input																						
FSI1_RX_CLK	IO Receive Clock Input																						
FSI2_RX_CLK	IO Receive Clock Input																						
FSI3_RX_CLK	IO Receive Clock Input																						

Note

Refer to the Application Note DCC computation tool to obtain the register value configurations of desired clock sources to be compared

13.6.1.3.3 DCC Mode of Operation

13.6.1.3.3.1 DCC Single-Shot Mode

The DCC may 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
- COUNT1 reaches 0

At the end of one sequence in single-shot mode, the DCC will de-assert the enable value (DCCENA), disabling further counting. At the end of one sequence in single-shot mode, if it is no error that stops counting, then the done status bit is set and a done interrupt is driven. Application must clear the done bit before restarting counts. At the end of a sequence in single-shot mode, if there is an error, then the error status bit will be set. Application must clear the error status bit before starting the next sequence.

13.6.1.3.3.2 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.

13.6.1.3.3.2.1 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_GCTRL2DCCGCTRL2 DCCGCTRL2[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.

13.6.1.3.3.2.2 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 (DCCERRCNT) for CPU to clear when new trace of number of errors is required to be maintained.

13.6.1.3.4 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.

13.6.1.3.4.1 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.

13.6.1.3.4.2 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_GCTRL2DCCGCTRL2 DCCGCTRL2 [11-7] 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.

13.6.1.3.4.3 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 DCCSTATUS2. DCC_STATUS2DCCSTATUS2

13.6.1.3.5 DCC Count Read Registers

DCC has provision to read the counts during operation. This is performed using DCCCNT0, DCCVALID0DCC_CNT0DCCCNT0DCC_VALID0DCCVALID0, and DCC_CNT1DCCCNT1 DCCCNT1 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_GCTRL2DCCGCTRL2 DCCGCTRL2[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 until full.

Note

- Input0_clk and input1_clk are two asynchronous clock domains. In a system, VBUS clock may also be asynchronous relative to both Input0_clk and Input1_clk. The module must be able to generate an error when either Input0_clk or Input1_clk is not present. VBUS clock should not sample or affect the clock counting logic in any way.
- In general, the reference clock should be hooked up to Input0_clk and measured clock should be connected to Input1_clk. The default clock source i.e. '0' should be assigned to connect with device native clock such as internal oscillator reference on both.
- The error interrupt signal is independent of the error flag bit. If the interrupt is masked, the error flag is still set when an error occurs. The error flag stays set until it is cleared, regardless of the status of the interrupt.
- The done flag in the DCCSTAT register would be set when the single shot mode completes without error and is independent of the DONEENA bits in the DCCGCTRL register. The done level interrupt would be set only if it is enabled by the DONEENA bits.
- Upon debug access, the FIFO pointers do not advance, hence not impacting the functional behavior.

13.6.1.3.6 Limp Mode Generation

Error on MAIN_DCC0 instance can trigger Limp-mode for added safety. This feature can be enabled by setting the MMR bits in TOP_RCM.LIMP_MODE_EN.DCC0_ERROR_EN

Note

More details on LIMP_MODE can be found in the Clocking section of Device Configuration chapter of the TRM

13.6.2 ECC Aggregator

This section describes the common ECC aggregator functionality.

13.6.2.1 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, ECC wrapper are considered as single entity and are hereinafter referred to as ECC aggregator unless otherwise explicitly specified.

Table 13-254 lists the device modules and subsystems which have ECC aggregator.

Table 13-254. Device Modules and Subsystems with ECC Aggregator

This table lists the device modules and subsystems which have an ECC aggregator

Module Instance	ECC Aggregator Support	RAM ID Number
SoC/Interconnect	✓	Not Applicable
R5FSS0-0	✓	See *R5FSS ECC Support*
R5FSS0-1	✓	See *R5FSS ECC Support*
R5FSS1-0	✓	See *R5FSS ECC Support*
R5FSS1-1	✓	See *R5FSS ECC Support*
ICSSM	✓	See *PRU_ICSSM RAM Index Allocation*
MCAN0	✓	1
MCAN1	✓	1
MCAN2	✓	1
MCAN3	✓	1
CPSW	✓	See *Memory Error Detection and Correction*

13.6.2.1.1 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
- 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 is ECC RAM
- 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

13.6.2.2 ECC Aggregator Integration

This section describes an ECC aggregator integration in the device, including information about clocks, resets, and hardware requests.

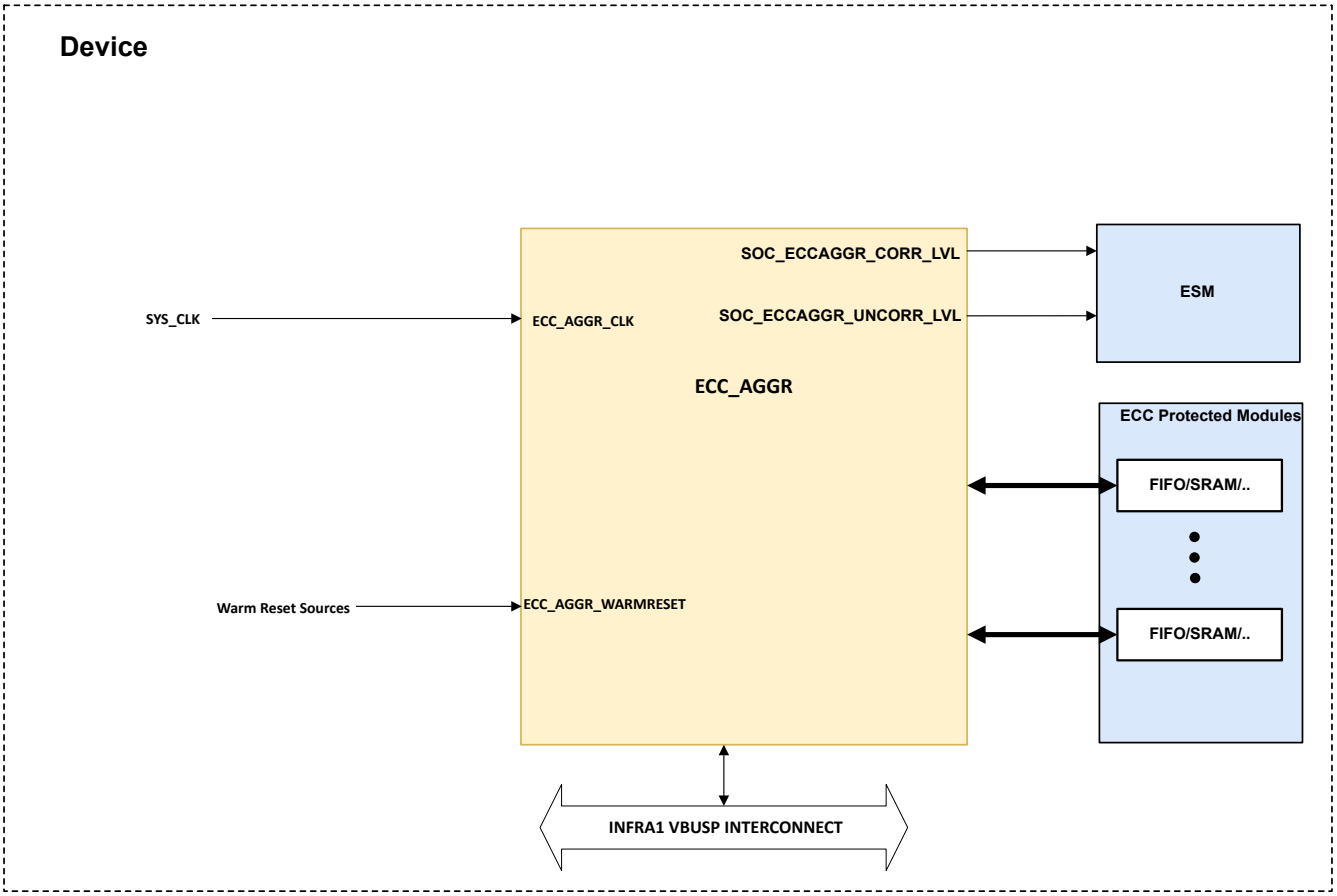
Note

For a list of the device modules and subsystems which have ECC aggregator, see Device Modules and Subsystems with ECC Aggregator.

13.6.2.2.1 ECC Aggregator Integration

There is 1x ECC Aggregator integrated in the device. The diagram below provides a visual representation of the device integration details.

Figure 13-231. ECC Aggregator Integration



The tables below summarize the device integration details of ECC Aggregator.

Table 13-255. *ECC Aggregator*Device Integration

This table describes the ECC Aggregator device integration details.

ECC Aggregator Instance	Device Allocation	SoC Interconnect
ECC Aggregator0	✓	INFRA1 VBUSP Interconnect

Table 13-256. *ECC Aggregator*Clocks

This table describes the ECC Aggregator clocking signals.

ECC Aggregator Instance	ECC Aggregator Clock Input	Source Clock Signal	Source	Default Freq	Description
ECC Aggregator0	ECC_AGGR_CLK	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	ECC Aggregator Interface Clock

Table 13-257. *ECC Aggregator* Resets

This table describes the ECC Aggregator reset signals.

ECC Aggregator Instance	ECC Aggregator Reset Input	Source Reset Signal	Source	Description
ECC Aggregator 0	ECC_AGGR_WARMRESET(VBUSP_RSTn)	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	ECC Aggregator0 Asynchronous Reset

Table 13-258. *ECC Aggregator* Event Requests

This table describes the ECC Aggregator interrupt requests.

ECC Aggregator Instance	ECC Aggregator Interrupt Signal	Destination Interrupt Input	Destination	Type	Description
ECC Aggregator0	SOC_ECCAGGR_UNCORR_LVL_0	SOC_ECCAGGR_UNCORR_LVL_0	ESM	Level	ECC Aggregator0 uncorrectable error event
	SOC_ECCAGGR_CORR_LVL_0	SOC_ECCAGGR_CORR_LVL_0			ECC Aggregator0 correctable error event

Table 13-259. *Device modules with ECC Aggregator*

This table describes the ECC Aggregator interrupt requests.

ECC Aggregator	ECC Aggregator Module instances
ECC Aggregator0	L2OCRAM_BANK0
	L2OCRAM_BANK1
	L2OCRAM_BANK2
	L2OCRAM_BANK3
	MBOX_SRAM
	TPTC00
	TPTC01

Note

For more information on the interconnects, see the *System Interconnect* chapter.

For more information on power, reset, and clock management, see the corresponding sections within the *Device Configuration* chapter.

For more information on the device interrupt controllers, see the *Interrupt Controllers* chapter.

13.6.2.3 ECC Aggregator Functional Description

This section describes the architecture and functional details of the ECC aggregator.

13.6.2.3.1 ECC Aggregator Block Diagram

Figure 13-232 shows the ECC aggregator block diagram.

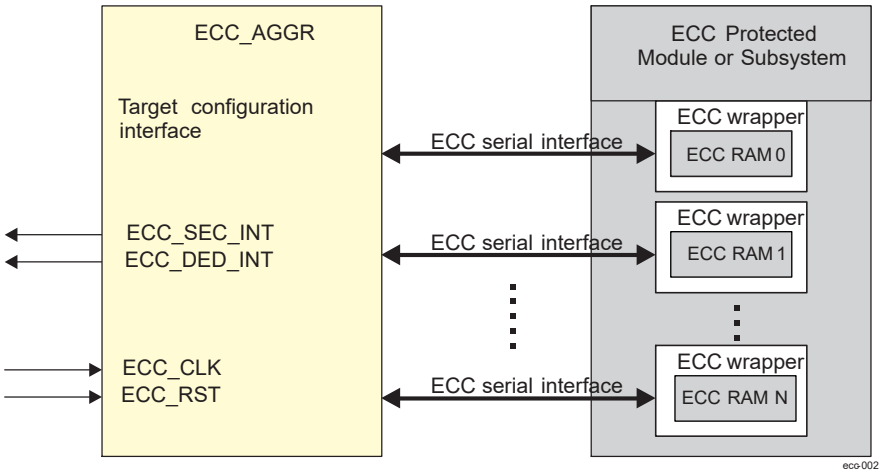


Figure 13-232. ECC Aggregator Block Diagram

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 target 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.

13.6.2.3.2 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 0x24 for the ECC RAM endpoint. 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 13.6.2.3.3. These registers have also different functionality for types of ECC endpoints.
- **Interrupt registers**. They include interrupt status, interrupt enable, interrupt disable, and EOI registers. For more information, see Section 13.6.2.3.5.

13.6.2.3.3 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 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 register to get its value.

13.6.2.3.4 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.

13.6.2.3.5 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 register:
  - ECC_SEC_ENABLE_SET_REG0 for the correctable interrupt
  - ECC_DED_ENABLE_SET_REG0 for the non-correctable interrupt
- On receiving an interrupt, software checks which ECC endpoint has caused the error by reading the following interrupt status register:
  - ECC_SEC_STATUS_REG0 for the correctable interrupt
  - ECC_DED_STATUS_REG0 for the non-correctable interrupt
- Software performs serial read operations as described in Section 13.6.2.3.3 to read the following status registers that contain details about the error:
  - The endpoint is ECC RAM:
    - ECC_ERR_STAT1
    - ECC_ERR_STAT2
    - ECC_ERR_STAT3
- 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

13.6.2.3.6 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.

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. The ECC_ERR_CTRL1 and ECC_ERR_CTRL2 registers should be written first to setup the injection. Then, either the ECC_CTRL[3] FORCE_SEC or the ECC_CTRL[4] FORCE_DED 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_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_CTRL register to set the FORCE_SEC or FORCE_DED again. Reading 0x0 from either the FORCE_SEC or the FORCE_DED 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.

13.6.3 Error Signaling Module (ESM)

This section describes the Error Signaling Module (ESM) in the device.

13.6.3.1 ESM Overview

The Error Signaling Module (ESM) aggregates safety-related events from throughout the SoC into one location. It can signal both low and high priority interrupts to a processor to deal with a safety event and/or manipulate an I/O pin to signal an external system or controller that act upon error to take appropriate action with the SoC Ex: Reset or set the system in a safe, known state. This module does not specify any methods of intervention, but only the facilitates alerting internal CPUs and external monitor(s) of an existing error event.

ESM Overview shows ESM allocation across the device.

Figure 13-233 shows the ESM modules overview.

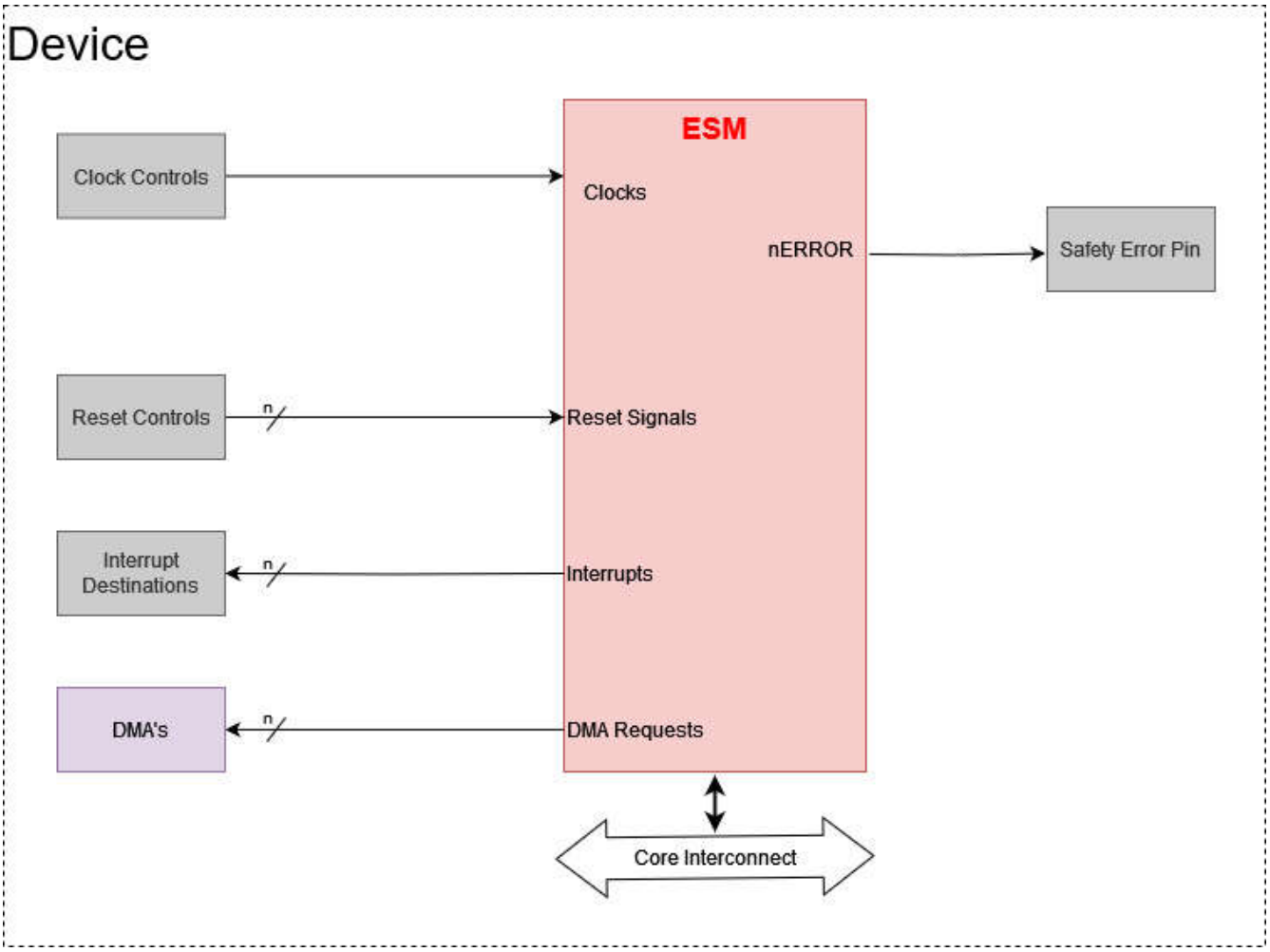


Figure 13-233. ESM Overview

13.6.3.2 ESM Features

Each ESM module implements the following features:

- Up to 96 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 to the external world
  - Support of level or PWM modes

- Configurable timebase for error signal
- Error forcing capability for Diagnostic testing
- Redundant logic to detect pulse events

13.6.3.3 ESM Integration

The #none# provides a visual representation of the device integration details.

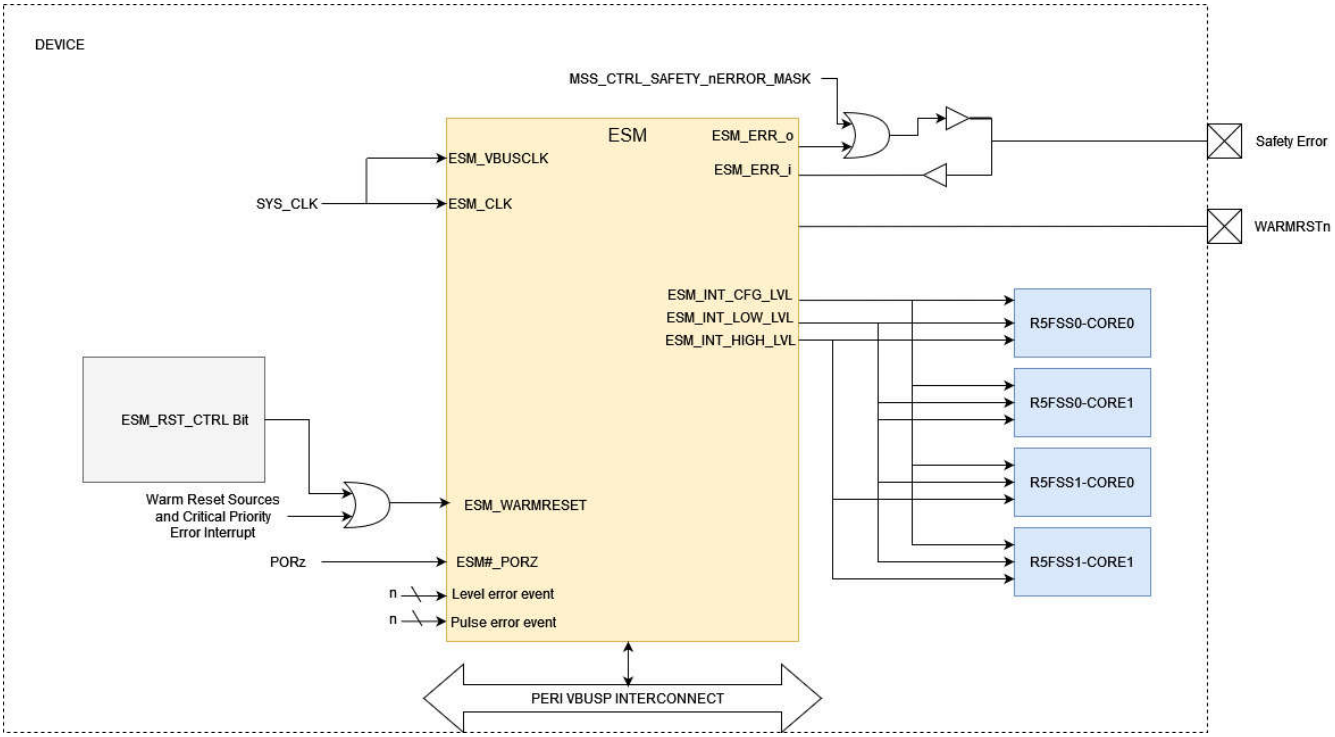


Figure 13-234. ESM integration Diagram

The tables below summarize the device integration details of ESM.

Table 13-260. *ESM* Device Integration

This table describes the ESM device integration details.

ESM Instance	Device Allocation	SoC Interconnect
ESM	✓	INFRA0 VBUSP Interconnect

Table 13-261. *ESM* Clock Integration

This table describes the ESM clocking signals.

ESM Instance	ESM Clock Input	Source Clock Signal	Source	Default Freq	Description
ESM	ESM_VBUSCLK	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	ESM VBUSP Interface Clock
	ESM_CLK				ESM Functional Clock

Table 13-262. *ESM* Resets

This table describes the ESM reset signals.

ESM Instance	ESM Reset Input	Source Reset Signal	Source	Description
ESM	ESM_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	ESM Asynchronous Reset
	ESM_POR_RST	POR Reset (MOD_POR_RST)	Device Power-On Reset	ESM Power-On Reset

Table 13-263. *ESM* Interrupt Requests

This table describes the ESM interrupt requests.

ESM Instance	ESM Interrupt Signal	Destination Interrupt Input	Destination	Type	Description
ESM	ESM_INT_CFG_LVL_0	ESM_INT_CFG_LVL	ALL R5FSS Cores	Level	ESM Configuration Error Interrupt
	ESM_INT_LOW_LVL_0	ESM_INT_LOW_LVL			ESM Low Priority Interrupt
	ESM_INT_HIGH_LVL_0	ESM_INT_HIGH_LVL			ESM High Priority Interrupt

Note

For more information on the interconnects, see the *System Interconnect* chapter.

For more information on power, reset, and clock management, see the corresponding sections within the *Device Configuration* chapter.

For more information on the device interrupt controllers, see the *Interrupt Controllers* chapter.

13.6.3.4 ESM Functional Description

13.6.3.4.1 ESM Functional Operation

The Error Signaling Module (ESM) centralizes fault reports. The module 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 also selects whether the interrupt that is generated is high priority or low priority.

By reporting the faults in a central location, the system can determine what caused the fault and what action can be taken. In general, the faults can be split into two categories:

- Correctable faults
- Non-correctable faults

The ESM reports errors in two ways:

- An interrupt to a processor inside the device. This enables the device to analyze and try to recover from an error.
- An external ERROR pin in the device. This enables 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 remains 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 goes inactive (high). If signal does not go inactive in that time, then an external agent must intervene, as an unrecoverable error can occur. In PWM mode, the error causes the output pin to maintain the value for a minimum period of time. After that period of time, if the error has been cleared by an internal processor, the pin continues the PWM pattern. If the signal does not go inactive in that time, then an external agent must intervene, as an unrecoverable error can occur.

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 recovery fails, then the external system is still alerted. If recovery succeeds, then the ERROR pin assertion can be removed 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 monitors of an existing error event.

ESM Block Diagram shows the ESM module block diagram.

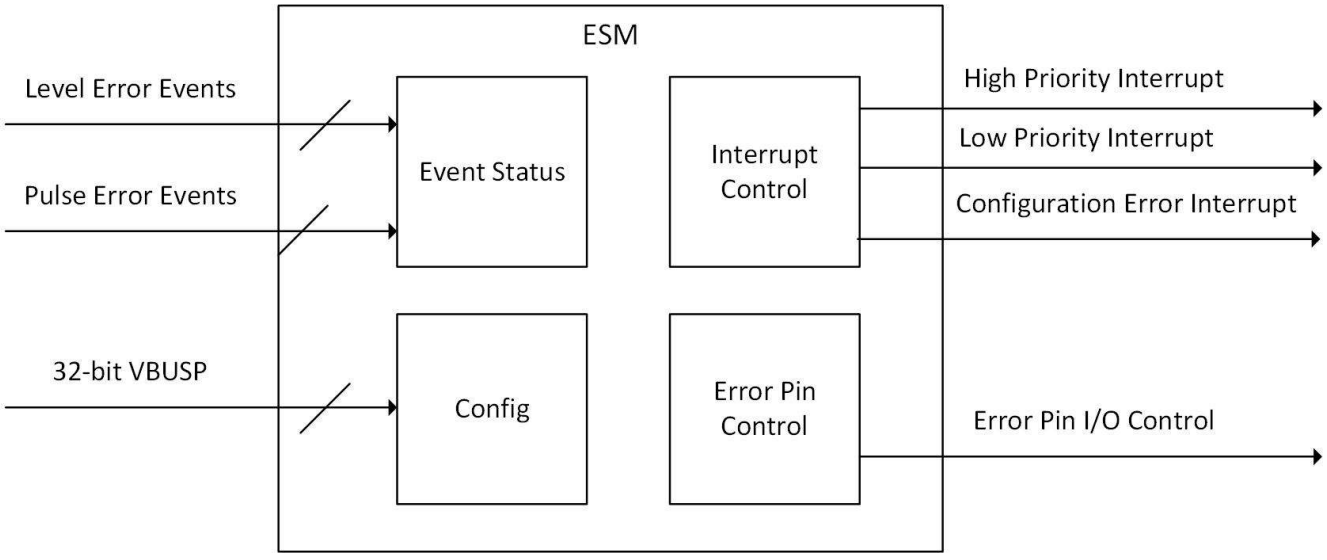


Figure 13-235. ESM Block Diagram

13.6.3.4.2 Error Event Inputs

The ESM can have up to 96 error event inputs, configurable by groups of 32. Error event inputs can be either level (default) or pulse. This device is configured with 2 level group events and 1 pulse group events.

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 circuit. 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 circuit. Once an edge has been detected on any of the three inputs, the raw status is set. Subsequent pulses are likely to come concurrently or quickly enough that software will not have reacted yet. This circuit is intentionally biased against false negatives and towards false positives. An SEU that causes an event where none actually occurred will just cause software to be called in to action. Software will observe that there is no real error and clear the false status.

13.6.3.4.3 Error Interrupt Outputs

Error Interrupt Outputs are provided so that a processor in the SoC 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 (Error Group N Interrupt Enabled Set Register (Base Address + 0x400 + N*0x20 + 0x08)). Additionally, each error event input can be programmed to influence either the Low Priority (Default) interrupt or the High Priority interrupt (Error Group N Interrupt Priority Register (Base Address + 0x400 + N*0x20 + 0x10)). 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.

13.6.3.4.4 ESM Error Pin Output

The Error Pin Output 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 (Error Group N Error Pin Influence Set Register (Base Address + 0x400 + N*0x20 + 0x14)). The ESM does not actually incorporate an I/O, this must be done at the SoC level. The Error Pin Output is active low or PWM based on the Error Pin Control register pwm_en field. This pwm_en field should only be modified when the ESM is disabled, based on the Global Enable register.

During Power-On-Reset, the Error Pin is active (asserted low). It is expected that the SoC drives this via a weak internal pull-down. The I/O is under the control of the SoC. When POR is removed from the ESM, it will be driving the Error Pin so the SoC can hand over control to the ESM. The customer may also add an external pull-down that is only active when the SoC is in reset.

During a Warm Reset the state of the Error Pin is unchanged (i.e. the Error Pin logic is only reset by a Power-On-Reset). The SoC should leave the I/O active during a warm reset.

The I/O input from the cell should be looped back to the err_i input. In this way, the status of the error I/O can be directly observed from the I/O buffer loopback path, instead of just from the internal state to the ESM.

The isolation value for the err_o output of ESM is active (0).

Figure 13-236 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 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 Section 13.6.3.4.9) for how to clear level and pulse pending events). The second is the CLEAR event meant to de-assert the Error Pin.

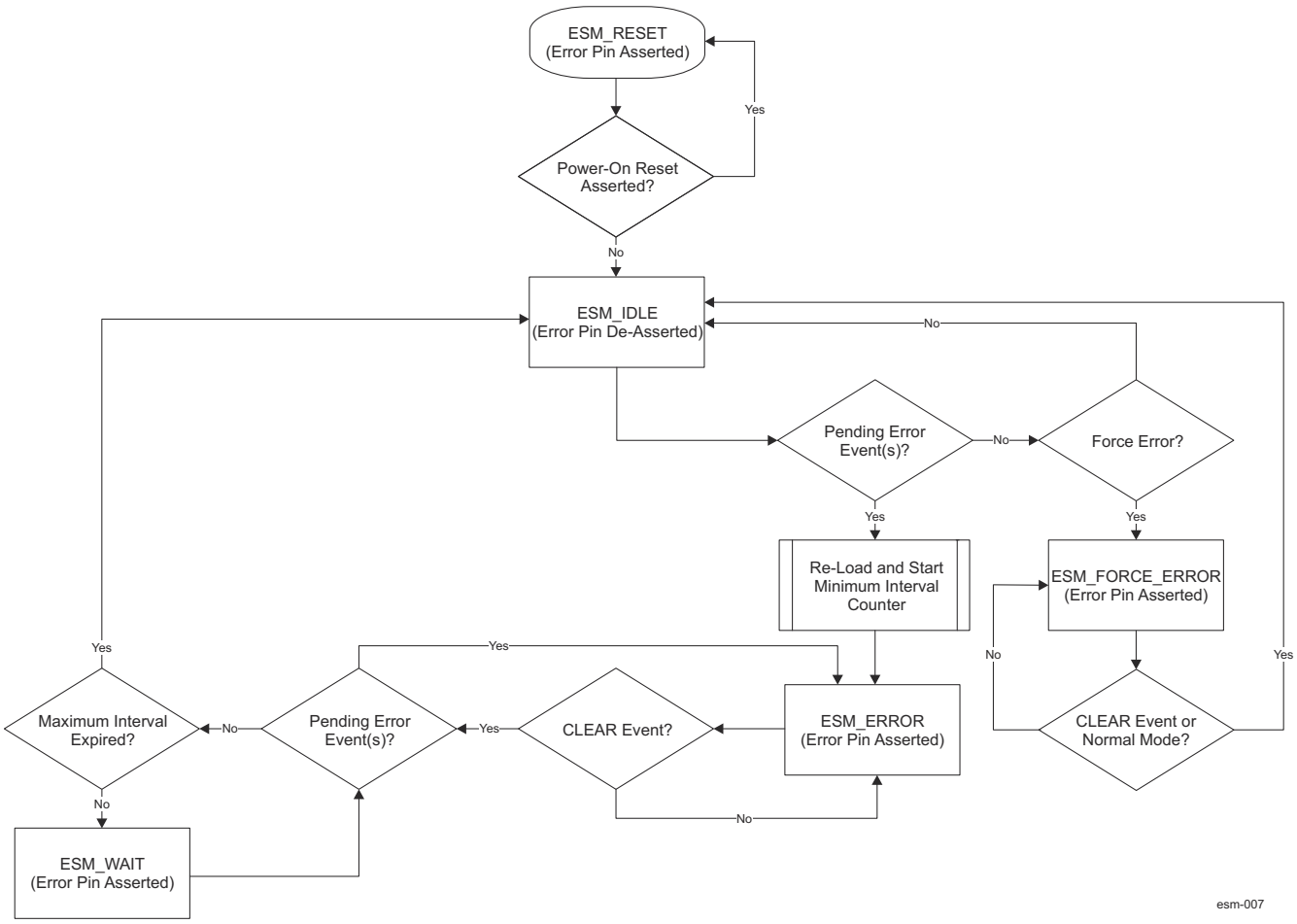


Figure 13-236. ESM Error Pin State Flowchart

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 Error Pin Counter Pre-Load Register (Base Address + 0x4C)). 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 13.6.3.4.9)
3. A CLEAR must be written to the Error Pin Control Register (Base Address + 0x40)

Note

Step 3 should happen after step 2, but either (or both) of these steps may happen before or after step 1.

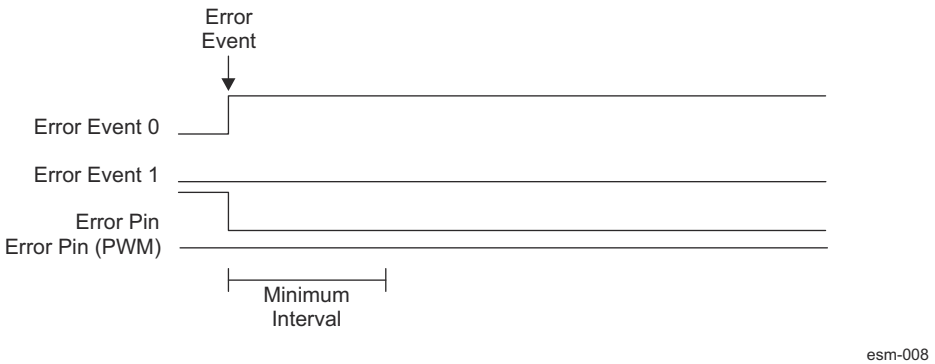


Figure 13-237. ESM Error Pin Assertion

If, during the minimum time, CLEAR is written to the error key, then the error pin will de-assert after the minimum interval.

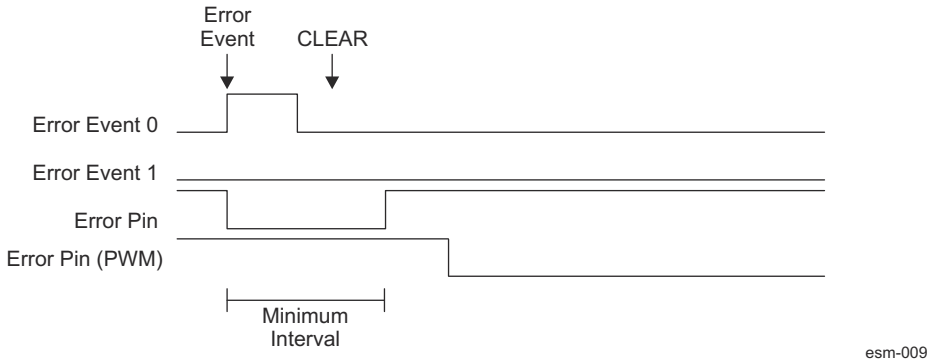


Figure 13-238. ESM Error Pin Assertion with CLEAR during Minimum Interval

If CLEAR is not written till after the minimum interval, the error pin will de-assert when CLEAR is written. This is regardless of whether the error event itself is removed before or after the minimum interval, as shown by the dotted line in Figure 13-239

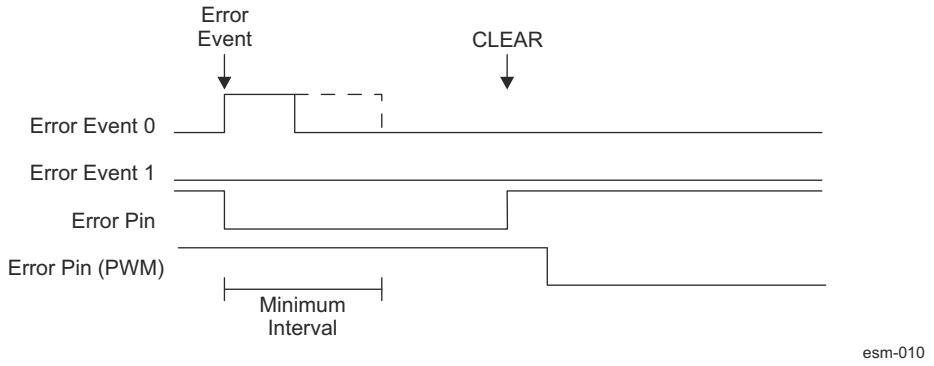


Figure 13-239. ESM Error Pin Asserting with CLEAR after Minimum Interval

When in the ESM_ERROR state and a CLEAR event happens, 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 interval counter.

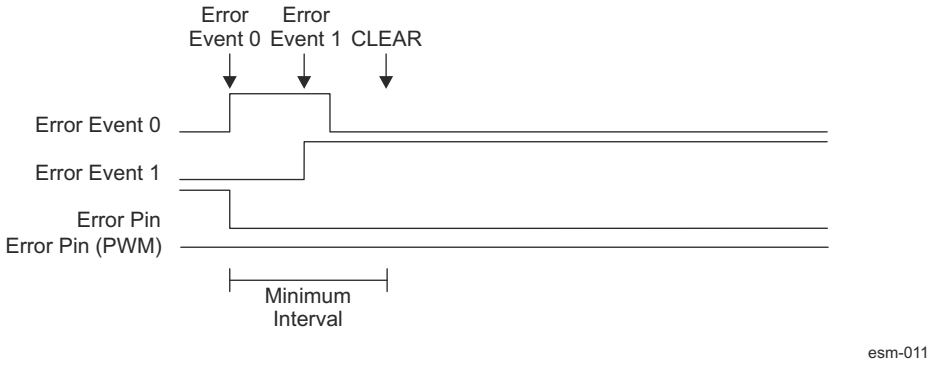


Figure 13-240. ESM Error Pin Asserting with Interval Reset by Additional Error Event(s)

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 13-241), 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 interval

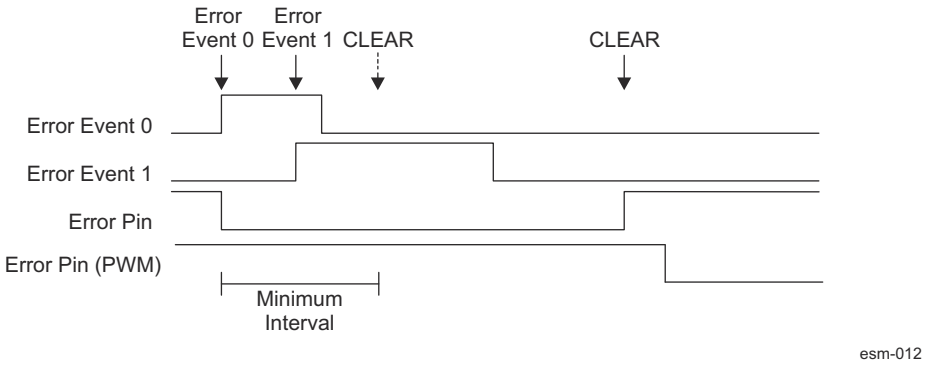


Figure 13-241. ESM Error Pin Asserting with Single CLEAR for Multiple Events

If all error events are cleared and the ESM is in the ESM_WAIT state, waiting for the minimum interval to expire, and a new error interrupt event occurs, the ESM will go back to the ESM_ERROR state. The minimum interval will not reset, but a new CLEAR event will be required.

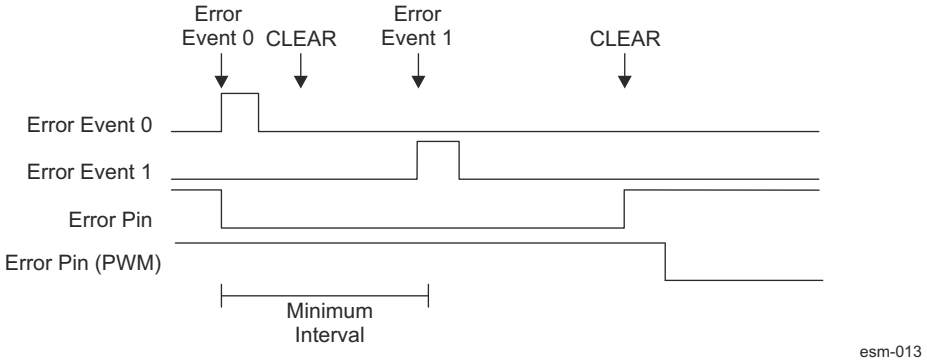


Figure 13-242. ESM Error Pin Asserting with New Error During Minimum Time Interval

13.6.3.4.5 Error Pin Behavior During Reset

Table 13-264 shows some common scenarios of how the error pin status and the values of two associated registers, Error Pin Control Register (Base Address + 0x40) and Error Pin Status Register (Base Address + 0x44), will correspond.

Table 13-264. 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.

13.6.3.4.6 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.

13.6.3.4.7 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 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 Error Pin Counter Pre-Load Register (Base Address + 0x4C) and the clock frequency of the ESM. Software and SoC integration must calculate the value accordingly. The Minimum Time Interval should be set according to the needs of the application.

13.6.3.4.8 Safety Protection for MMRs

The configuration MMRs for each Error Group N are backed 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 below:

- Config Error Interrupt Raw Status/Set Register (Base Address + 0x10)
- Config Error Interrupt Enabled Status/Clear Register (Base Address + 0x14)
- Config Error Interrupt Enabled Set Register (Base Address + 0x18)
- Config Error Interrupt Enabled Clear Register (Base Address + 0x1C)
- Error Group N Event Raw Status/Set Register (Base Address + 0x400 + N*0x20 + 0x00)
- Error Group N Interrupt Enabled Status/Clear Register (Base Address + 0x400 + N*0x20 + 0x04)
- Error Group N Interrupt Enabled Set Register (Base Address + 0x400 + N*0x20 + 0x08)
- Error Group N Interrupt Enabled Clear Register (Base Address + 0x400 + N*0x20 + 0x0C)
- Error Group N Interrupt Priority Register (Base Address + 0x400 + N*0x20 + 0x10)
- Error Group N Error Pin Influence Set Register (Base Address + 0x400 + N*0x20 + 0x14)
- Error Group N Error Pin Influence Clear Register (Base Address + 0x400 + N*0x20 + 0x18)

The Error Pin Control Register (Base Address + 0x40) contains a multi-bit field. The Error Pin Counter Pre-Load Register (Base Address + 0x4C) should also be read and checked periodically by software. 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 13-322 lists the KEY values and their respective meaning.

Table 13-265. 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.

13.6.3.4.9 ESM Interrupts

The ESM module generates three output interrupts to the device interrupt controllers:

- Configuration error interrupt (see Section 13.6.3.4.10.1)

- High priority error interrupt (see Section 13.6.3.4.10.2)
- Low priority error interrupt (see Section 13.6.3.4.10.3)

The error interrupt outputs are provided so that a processor in the device can be signaled to intervene whenan error event occurs. Each error event input can be enabled, via software, to cause an error interrupt to occur (via the Error Group N Interrupt Enabled Set Register). Additionally, each error event input can be programmed to influence either the low priority (default) interrupt or the high priority interrupt (via the Error Group N Interrupt Priority Register). 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.

13.6.3.4.10 Programming Guide

13.6.3.4.10.1 Configuration Error Interrupt

The Configuration Error Interrupt indicates that there is an inconsistency in the configuration of one (or more) Error Group N MMRs. In such inconsistencies, the internal copies of any of the MMRs caused by a SER associated with Error Group N, the corresponding raw status will be set in the Config Error Interrupt Raw Status/Set Register (Base Address + 0x10). If the corresponding bit is enabled, a Configuration Error Interrupt will be triggered.

The Configuration Error Interrupt is not enabled by default and it should be enabled by the processor by writing:

1. Write the respective Error group bit which needs to be monitored for Configuration Error
   a. Config Error Interrupt Enabled Set Register (Base Address + 0x18)

   Note

   Software should ensure that no status is set in RAW status register of Config Error before enabling the interrupt in SET register. The RAW status register should be read and cleared before enabling the ESM interrupt to avoid triggering of false interrupt to CPU
2. Enable the Configuration Error Interrupt in VIM for CPU which is configuring the ESM

Note

Refer R5SS Interrupt Map for Interrupt number

When a Configuration Error Interrupt is received, the acting processor should follow these steps:

1. Read the Config Error Interrupt Enabled Status/Clear Register (Base Address + 0x14) to determine which Group as a configuration error
2. Write the correct values to the following registers

   Note

   Software should maintain a copy of the correct values to ensure that they can be re-programmed. Software may just try to read back the values, but they cannot be guaranteed to be correct if there was a 2-bit error

   a. Error Group N Interrupt Enabled Set Register (Base Address + 0x400 + N*0x20 + 0x08)
   b. Error Group N Interrupt Enabled Clear Register (Base Address + 0x400 + N*0x20 + 0x0C)
   c. Error Group N Interrupt Priority Register (Base Address + 0x400 + N*0x20 + 0x10)
   d. Error Group N Error Pin Influence Set Register (Base Address + 0x400 + N*0x20 + 0x14)
   e. Error Group N Error Pin Influence Clear Register (Base Address + 0x400 + N*0x20 + 0x18)
3. Service any pending interrupts via the steps in the following sections
   a. The raw status of any pending interrupts may be inconsistent. Servicing the interrupt will return it to consistency (Error Group N Event Raw Status/Set Register (Base Address + 0x400 + N*0x20 + 0x00))
4. Write a 1 to the appropriate bits in the Config Error Interrupt Enabled Status/Clear Register (Base Address + 0x14)

a. This will clear the raw status
b. If the error event is still asserted (or re-asserted) the raw status will be set back to 1
c. If there are no additional errors, the level interrupt will go low

5. Write the EOI vector to the EOI Interrupt Register (Base Address + 0x30)
   a. If there are additional Configuration Error enabled error events pending, then a new pulse will be generated and the level interrupt will remain asserted
6. If there are no additional Low Priority enabled error events pending, there will be no new pulse

13.6.3.4.10.2 Low Priority Error Interrupt

Events mapped to the low priority error interrupt 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. It is enabled by programming,

1. Write the correct value of the register by setting the event bit which needs to be monitored
   a. Error Group N Interrupt Enabled Set Register (Base Address + 0x400 + N*0x20 + 0x08)

Note

Software should ensure that no status is set in RAW status register before enabling the interrupt in SET register. The RAW status register should be read and cleared before enabling the ESM interrupt to avoid triggering of false interrupt to CPU

2. By default, the priority of all events is set to low. Software should read register and ensure the same or program it by writing 0x0 into Error Group N Interrupt Priority Register (Base Address + 0x400 + N*0x20 + 0x10) to map the events to low priority error interrupt
3. Enable the ESM Low Priority Error Interrupt in VIM for CPU which is monitoring the safety in system

Note

Refer R5SS Interrupt Map for Interrupt number to be configured in VIM

When a low priority error interrupt is received, the acting processor must perform the following steps:

1. Read the Low Interrupt Status Register (Base Address + 0x20)
   a. If both low_level_prio and low_pulse prio are equal to 0xFFFF, then END (Interrupt is no longer asserted)
   b. If either low_level_pend or low_pulse_pend (or both) are not equal to 0xFFFF, software has two options for determining what event to service
      i. First Option: Record the value of value in low_pulse_prio and/or low_level_prio. Determine which is higher priority. This is the Global Event Number of the highest priority Low Priority Error Event
      ii. Second option:
         1. Read the Low Priority Interrupt Status Register (Base Address + 0x28) to determine which Event Group(s) have pending Low Priority Interrupts
         2. Read the desired Error Group N Interrupt Enabled Status/Clear Register (Base Address + 0x400 + N*0x20 + 0x04)
         3. Identify which Low Priority Interrupt to service
2. Determine based on the ESM Interrupt Mapping of the SoC for the source of the Interrupt
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 peripheral that triggered the error
      iii. Resetting the device
      iv. Communicating outside the SoC for outside intervention
   b. The rest of these steps assume that the Error has been handled and the system wants to clear the error event
   c. The rest of the handling depends on whether the event is a pulse or level event

Level Event

1. Clear the Error Event at the Source
2. Write a 0x1 to the appropriate bit in the Error Group N Interrupt Enabled Status/Clear Register (Base Address + 0x400 + N*0x20 + 0x04)
   a. This will clear the raw status
   b. If the error event is still asserted (or re-asserted) the raw status will be set back to 1
   c. 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 EOI vector to the EOI Interrupt Register (Base Address + 0x30)
   a. If there are additional Low Priority enabled error events pending, then a new pulse will be generated
   b. If there are no additional Low Priority enabled error events pending, there will be no new pulse
4. Write a CLEAR to the Error Pin Control Register (Base Address + 0x40)
   a. This step is optional if the event is not enabled to influence the Error Pin (Error Group N Error Pin Influence Set Register (Base Address + 0x400 + N*0x20 + 0x14)), but may be done regardless as an extra CLEAR is not harmful

Pulse Event

1. Write a 0x1 to the appropriate bit in the Error Group N Interrupt Enabled Status/Clear Register (Base Address + 0x400 + N*0x20 + 0x04)
   a. This will clear the raw status
   b. This will de-assert the level interrupt
2. Write the EOI vector to the EOI Interrupt Register (Base Address + 0x30)
   a. If there are additional Low Priority enabled error events pending, then a new pulse will be generated and the level interrupt will remain asserted
   b. If there are no additional Low Priority enabled error events pending, there will be no new pulse
3. Clear the Error Event at the source
   a. The source may generate a new pulse which will show up as a new Error Event at the ESM
4. Write a CLEAR to the Error Pin Control Register (Base Address + 0x40)
   a. This step is optional if the event is not enabled to influence the Error Pin (Error Group N Error Pin Influence Set Register (Base Address + 0x400 + N*0x20 + 0x14)), but may be done regardless as an extra CLEAR is not harmful

13.6.3.4.10.3 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.

Any error event can be mapped to the high priority error interrupt. It is enabled by programming,

1. Write the correct value of the register by setting the event bit which needs to be monitored
   a. Error Group N Interrupt Enabled Set Register (Base Address + 0x400 + N*0x20 + 0x08)

Note

Software should ensure that no status is set in RAW status register before enabling the interrupt in SET register. The RAW status register should be read and cleared before enabling the ESM interrupt to avoid triggering of false interrupt to CPU

2. Set the respective field to 0x1 in Error Group N Interrupt Priority Register (Base Address + 0x400 + N*0x20 + 0x10) to map the events to high priority error interrupt
3. Enable the ESM High Priority Error Interrupt in VIM for CPU which is monitoring the safety in system

Note

Refer R5SS Interrupt Map for Interrupt number to be configured in VIM

When a High Priority Error Interrupt is received, the acting processor should follow these steps:

1. Read the High Interrupt Status Register (Base Address + 0x24)
   a. If both high_level_prio and high_pulse prio are equal to 0xFFFF, then END (Interrupt is no longer asserted)
   b. If either high_level_pend or high_pulse_pend (or both) are not equal to 0xFFFF, software has two options for determining what event to service
      i. First Option: Record the value of value in high_pulse_prio and/or high_level_prio. Determine which is higher priority. This is the Global Event Number of the highest priority High Priority Error Event
      ii. Second option:
         1. Read the High Priority Interrupt Status Register (Base Address + 0x2C) to determine which Event Group(s) have pending High Priority Interrupts
         2. Read the desired Error Group N Interrupt Enabled Status/Clear Register (Base Address + 0x400 + N*0x20 + 0x04)
         3. Identify which High Priority Interrupt to service
2. Determine based on the ESM Interrupt Mapping of the SoC for the source of the Interrupt
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 peripheral that triggered the error
      iii. Resetting the device
      iv. Communicating outside the SoC for outside intervention
   b. The rest of these steps assume that the Error has been handled and the system wants to clear the error event
   c. The rest of the handling depends on whether the event is a pulse or level event

Level Event

1. Clear the Error Event at the Source
2. Write a 0x1 to the appropriate bit in the Error Group N Interrupt Enabled Status/Clear Register (Base Address + 0x400 + N*0x20 + 0x04)
   a. This will clear the raw status
   b. If the error event is still asserted (or re-asserted) the raw status will be set back to 1
   c. 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 EOI vector to the EOI Interrupt Register (Base Address + 0x30)
   a. If there are additional High Priority enabled error events pending, then a new pulse will be generated
   b. If there are no additional High Priority enabled error events pending, there will be no new pulse
4. Write a CLEAR to the Error Pin Control Register (Base Address + 0x40)
   a. This step is optional if the event is not enabled to influence the Error Pin (Error Group N Error Pin Influence Set Register (Base Address + 0x400 + N*0x20 + 0x14)), but may be done regardless as an extra CLEAR is not harmful

Pulse Event

1. Write a 0x1 to the appropriate bit in the Error Group N Interrupt Enabled Status/Clear Register (Base Address + 0x400 + N*0x20 + 0x04)
   a. This will clear the raw status
   b. This will de-assert the level interrupt
2. Write the EOI vector to the EOI Interrupt Register (Base Address + 0x30)
   a. If there are additional High Priority enabled error events pending, then a new pulse will be generated and the level interrupt will remain asserted
   b. If there are no additional High Priority enabled error events pending, there will be no new pulse
3. Clear the Error Event at the source
   a. The source may generate a new pulse which will show up as a new Error Event at the ESM
4. Write a CLEAR to the Error Pin Control Register (Base Address + 0x40)
   a. This step is optional if the event is not enabled to influence the Error Pin (Error Group N Error Pin Influence Set Register (Base Address + 0x400 + N*0x20 + 0x14)), but may be done regardless as an extra CLEAR is not harmful

13.6.4 Memory Cyclic Redundancy Check (MCRC) Controller

This chapter describes the Memory Cyclic Redundancy Check (MCRC) controller in the device.

13.6.4.1 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.

13.6.4.1.1 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

13.6.4.2 MCRC Integration

There is 1x MCRC integrated in the device. The diagram below provides a visual representation of the device integration details.

p

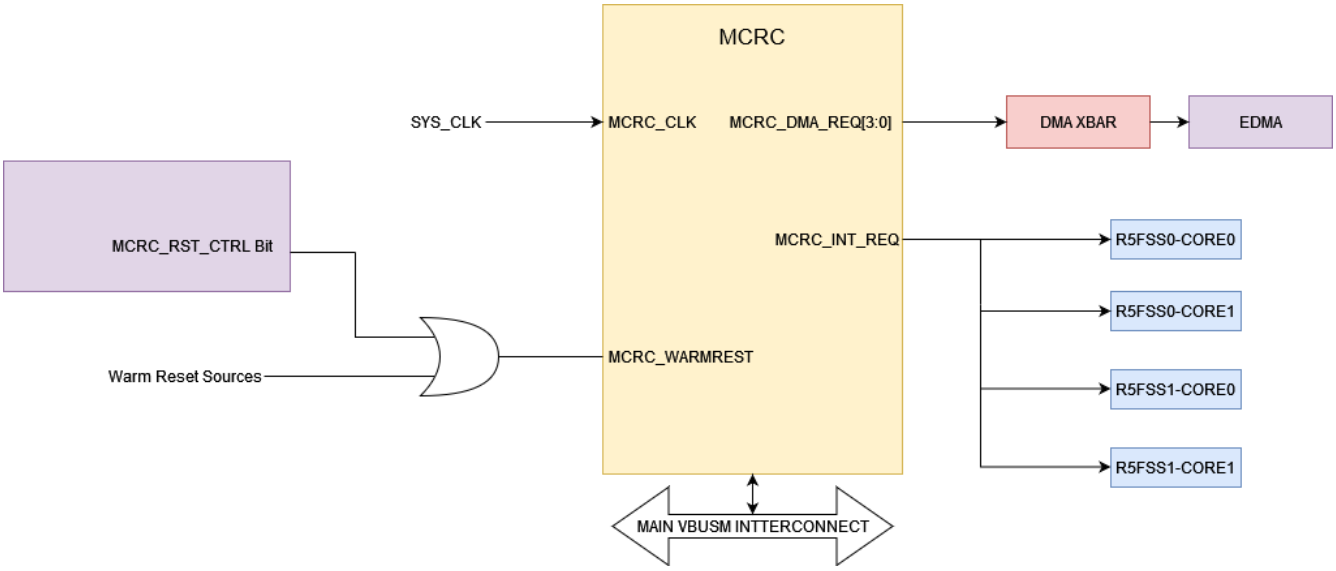


Figure 13-243. MCRC Integration

The tables below summarize the device integration details of MCRC# (where # = 1).

Table 13-266. *MCRC* Device Integration

This table describes the MCRC device integration details.

MCRC Instance	Device Allocation	SoC Interconnect
MCRC0	✓	CORE VBUSM Interconnect

Table 13-267. *MCRC* Clocks

This table describes the MCRC clocking signals.

MCRC Instance	MCRC Clock Input	Source Clock Signal	Source	Default Freq	Description
MCRC0	MCRC_CLK	SYS_CLK	PLL_CORE_CLK: HSDIV0_CLKOUT0	200 MHz	MCRC0 Interface Clock

Table 13-268. *MCRC* Resets

This table describes the MCRC reset signals.

MCRC Instance	MCRC Reset Input	Source Reset Signal	Source	Description
MCRC0	MCRC0_RST	Warm Reset (MOD_G_RST)	RCM + Warm Reset Sources	MCRC0 Asynchronous Reset

Table 13-269. *MCRC* Interrupt Requests

This table describes the MCRC interrupt requests.

MCRC Instance	MCRC Interrupt Signal	Destination Interrupt Input	Destination	Type	Description
MCRC0	MCRC0_INT_req	MCRC0_INT_req	ALL R5FSS Cores	Level	MCRC0 Event Interrupt

Table 13-270. *MCRC* DMA Requests

This table describes the MCRC DMA requests.

MCRC Instance	MCRC DMA Event	Destination DMA Event Input	Destination	Type	Description
MCRC0	MCRC0_DMA_0	MCRC0_dma_req[0]	EDMA Crossbar (DMA_XBAR)	Pulse	MCRC0 DMA Request
	MCRC0_DMA_1	MCRC0_dma_req[1]			
	MCRC0_DMA_2	MCRC0_dma_req[2]			
	MCRC0_DMA_3	MCRC0_dma_req[3]			

Note

For more information on the interconnects, see the *System Interconnect* chapter.

For more information on power, reset, and clock management, see the corresponding sections within the *Device Configuration* chapter.

For more information on the device interrupt controllers, see the *Interrupt Controllers* chapter.

13.6.4.3 MCRC Functional Description

13.6.4.3.1 MCRC Block Diagram

Figure 13-244 shows the MCRC internal blocks.

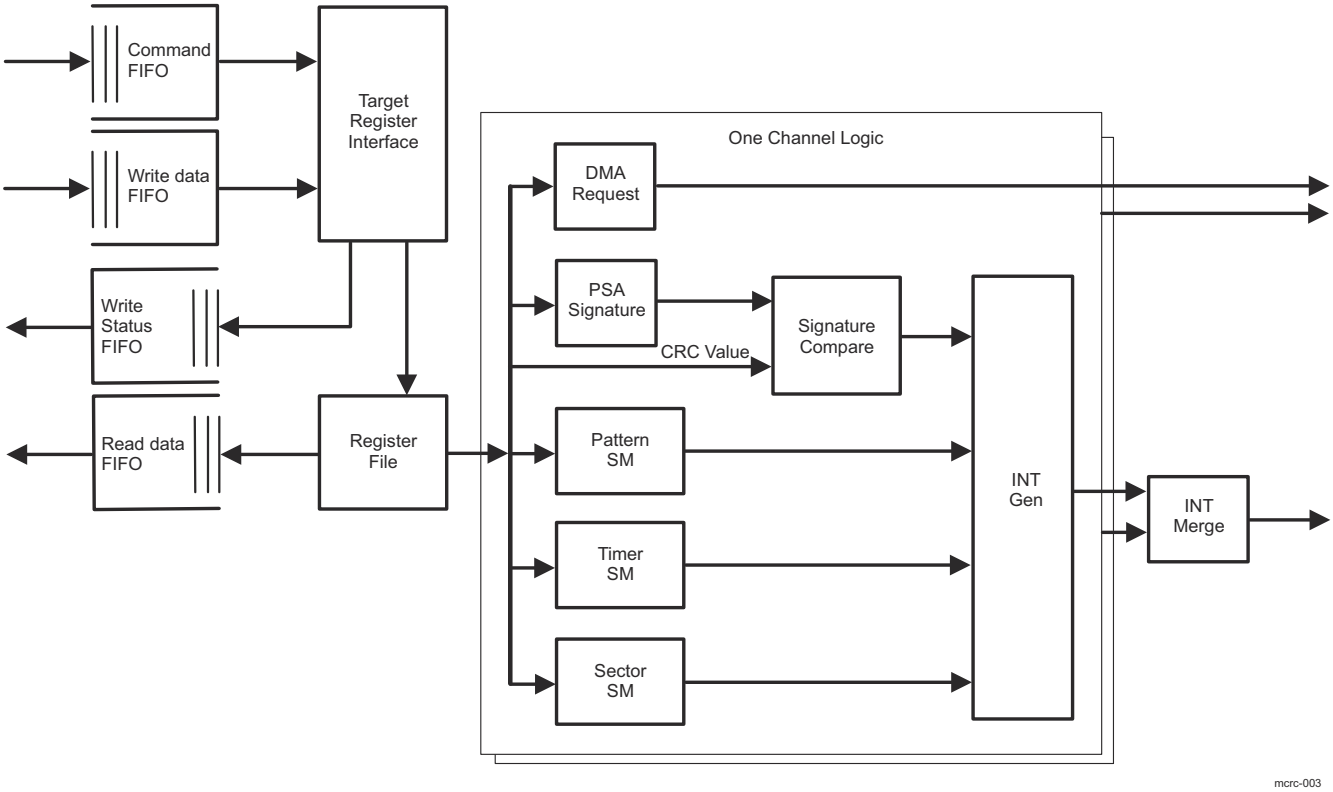


Figure 13-244. MCRC Block Diagram

- **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.

13.6.4.3.2 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.

13.6.4.3.3 MCRC Modes of Operation

MCRC Controller can operate in AUTO, Semi-CPU, and Full-CPU modes.

13.6.4.3.3.1 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.

13.6.4.3.3.2 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.

13.6.4.3.3.3 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.

13.6.4.3.4 PSA Signature Register

The 64-bit PSA Signature Register (MCRC_PSA_SIGREGL1-4 and MCRC_PSA_SIGREGH1-4) is based on the primitive polynomial found in (EQ 1) to produce the maximum length LFSR (Linear Feedback Shift Register).

$$f(x) = x^{64} + x^4 + x^3 + x + 1 \quad (31)$$

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 are compressed

13.6.4.3.5 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 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.

13.6.4.3.6 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.

13.6.4.3.7 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.

13.6.4.3.8 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.

13.6.4.3.8.1 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 13-245, a timer can be used to trigger a DMA request to 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 13-245, 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.

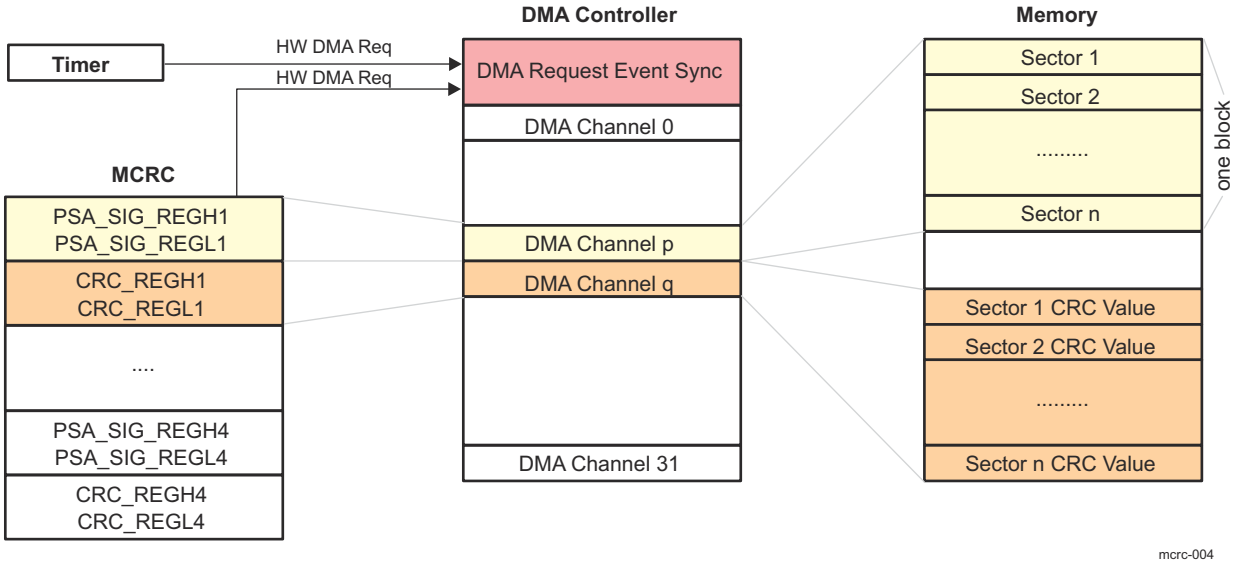


Figure 13-245. AUTO Mode Using Hardware Timer Trigger

13.6.4.3.8.2 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.

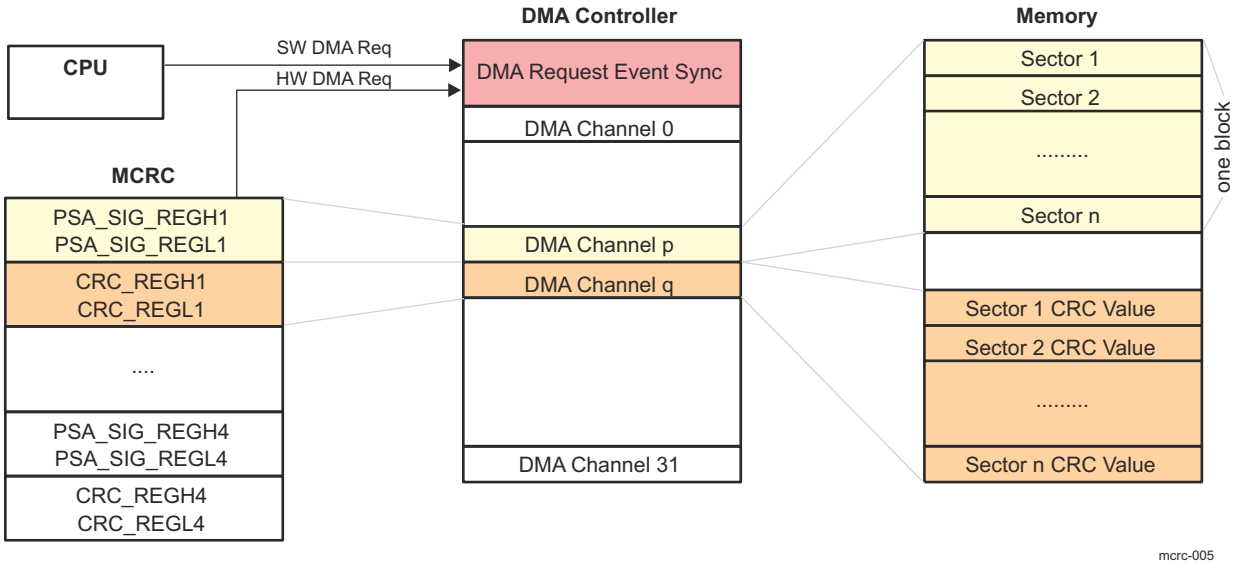


Figure 13-246. AUTO Mode With Software CPU Trigger

13.6.4.3.8.3 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 13-247 illustrates the DMA setup using semi-CPU mode with hardware timer trigger.

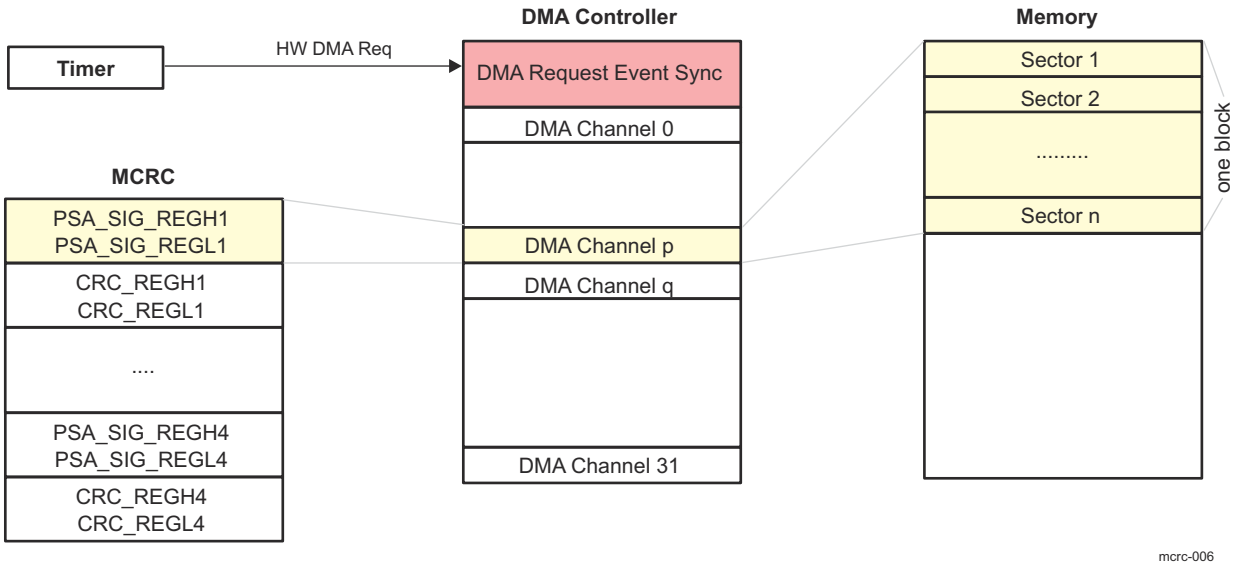


Figure 13-247. Semi-CPU Mode With Hardware Timer Trigger

Table 13-271. 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

13.6.4.3.9 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.

13.6.4.3.10 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.

$$\text{CRC Pattern Count} \times \text{CRC Sector Count} = \text{DMA Element Count} \times \text{DMA Frame Count} \quad (32)$$

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.

13.6.4.3.11 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 13-272. 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

13.6.4.3.11.1 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.

13.6.4.3.11.2 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.

13.6.4.3.11.3 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.

13.6.4.3.11.4 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

13.6.4.3.11.5 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 13-273 shows the offset interrupt vector address of each interrupt in an ascending order of priority.

Table 13-273. Interrupt Offset Mapping

Interrupt Condition	Offset Value
Phantom	0x0
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

13.6.4.3.11.6 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.

13.6.4.3.12 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.

13.6.4.3.13 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. .

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.

13.6.4.4 MCRC Programming Examples

13.6.4.4.1 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.

13.6.4.4.1.1 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.

13.6.4.4.1.2 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.

13.6.4.4.1.3 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.

13.6.4.4.2 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.

13.6.4.4.2.1 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.

- 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.

13.6.4.4.2.2 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.

13.6.4.4.3 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.

13.6.4.4.3.1 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.

13.6.4.4.3.2 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.

13.6.4.4.3.3 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

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.

13.6.4.4.4 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.

13.6.4.4.4.1 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.

13.6.5 Self-Test Controller (STC)

13.6.5.1 STC Overview

The enhanced Self-Test Controller (STC) is used to test logic cores based on the On-Product Multiple Input Signature Register (OPMISR) scan compression architecture.

Software-based self-test programs for the cores are available, but offer less test coverage. Due to the complexity of the soft cores, the coverage required can be difficult to achieve and will result in a larger program size.

For these complex cores, on-chip logic BIST support for the self-test is preferred.

The main features of the STC include:

- Implements the OPMISR controller, along with the on-chip self-test controller for the synthesizable module logic, which enables high test coverage.
- The self-test controller facilitates complete isolation of the logical segment under the test from the rest of the system during the self-test run. Configure critical control signals in the initiator and target ports of the logical segment under the test to a safe state.
- The self-tested CPU core initiator bus transaction signals are configured to be in idle mode during the self-test run.
- Time-out counter for the self-test run as a fail-safe feature.
- Can capture power reduction using dead cycles before and after the capture pulse.
- Coverage improvements technique – ROM inverse access mode. In this, the patterns are read in a reverse order from ROM and applied to the UUT. Pattern randomization due to this approach results in coverage improvement, without an increase in the number of patterns. Corresponding INV_MISR is also stored in the ROM.

A self test segment corresponds to a portion of discreet safety-critical logic which can be tested in isolation from the rest of the system by the self test controller and OPMISR logic.

13.6.5.1.1 Unsupported Features

- Section 13.6.5.3.7.1 – Launch-on-last-shift. TR_T =1
- Section 13.6.5.3.7.2 – Transition delay fault model. FT =1
- Section 13.6.5.3.7.6 – Low-power scan mode. MSS_STC.STCGCR1.LP_SCAN_MODE = 1

- Section 13.6.5.3.7.7 and Section 13.6.5.3.7.8 – Coverage improvement techniques – MSS_STC.STCGCR1.ROM_ACCESS_INV =1 Mode
- Interval-based testing
- MSS_STC.STC_CLKDIV clock division features

13.6.5.1.2 STC Memory Map

Table 13-274. STC Memory Map for AM263Px

Name	Start Address	Frame Address (Hex) End	Size	Description
R5FSS0_STC	TBD	TBD	284 Bytes	R5FSS0_STC module configuration registers
R5FSS1_STC	TBD	TBD	284 Bytes	R5FSS1_STC module configuration registers

13.6.5.1.3 OPMISR Concept

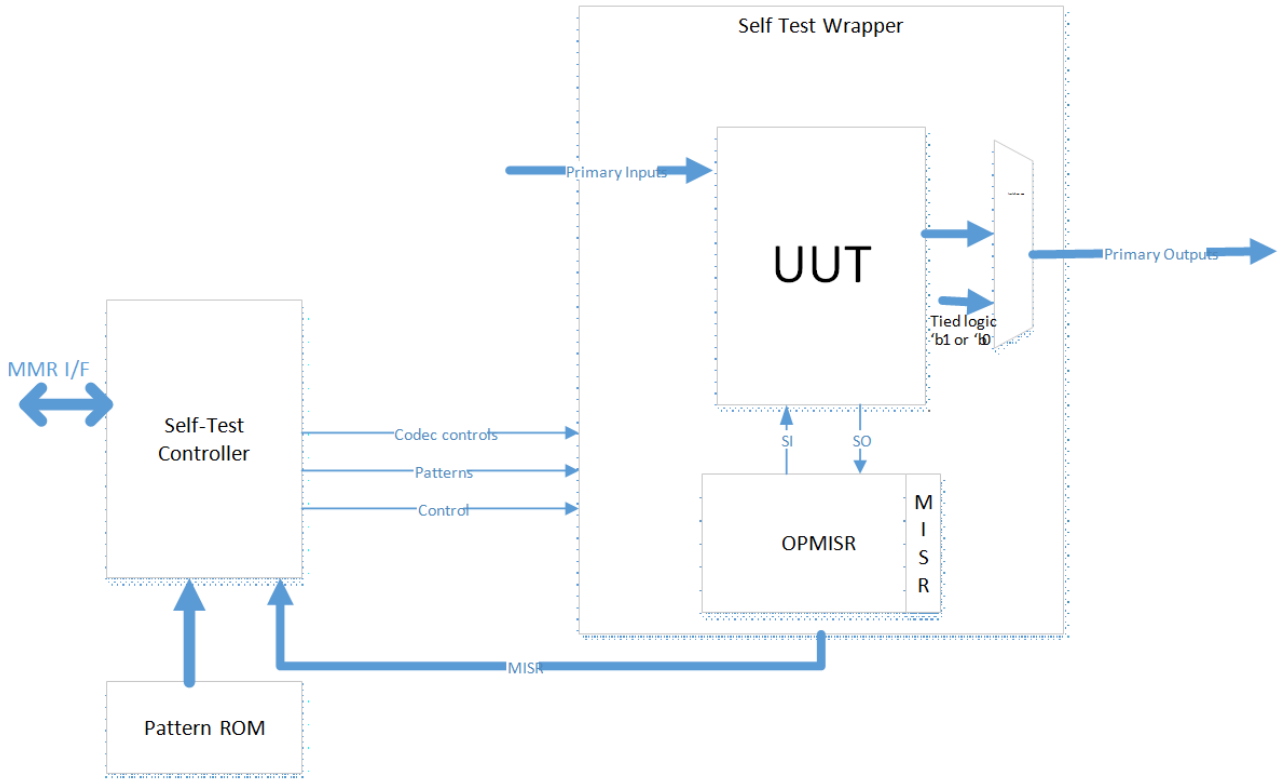


Figure 13-248. OPMISR Conceptual Diagram

STC enables fetching deterministic ATPG vectors from STC ROM and applies them to the UUT using XoR decompressor. BIST is implemented on the application-critical R5 and HSM cores.

The self-test controller uses the existing compression scan chains and applies the patterns from ROM. The scan chains are further unloaded into MISR during shift out operation.At the end of self test, the on chip MISR signature is compared with golden signature stored in pattern ROM.

The On-Product Multiple-Input Signature Register (OPMISR) is a methodology which moves the test pattern generation on-chip. Logic BIST is implemented on functional partitions (BIST'ed COREs) that are speed-critical and have high gate count.

The MISR test structure modifies the typical fullscan scan chain such that each scan data input internally drives many chains. These chains feed to the inserted MISR structure. The chain's values are captured into the MISR during shift, generating a resulting signature that can be shifted out.

A given Unit Under Test (UUT) is scan-inserted, and the scan chains are hooked to the OPMISR logic. A self-test wrapper is created around the UUT and the OPMISR logic. The inputs to the UUR driving the D pin of flops are overridden with a controllable flop inside the UUT. The outputs of the UUT are isolated by an isolation control signal during the STC operation. These features ensure that the core and UUT are isolated from the rest of the system during the self-test.

13.6.5.2 Block Diagram

The STC module is composed of following blocks:

- ROM interface
- FSM and sequence control
- Register file
- STC bypass / ATE interface
- Peripheral bus interface (VBUSP interface)

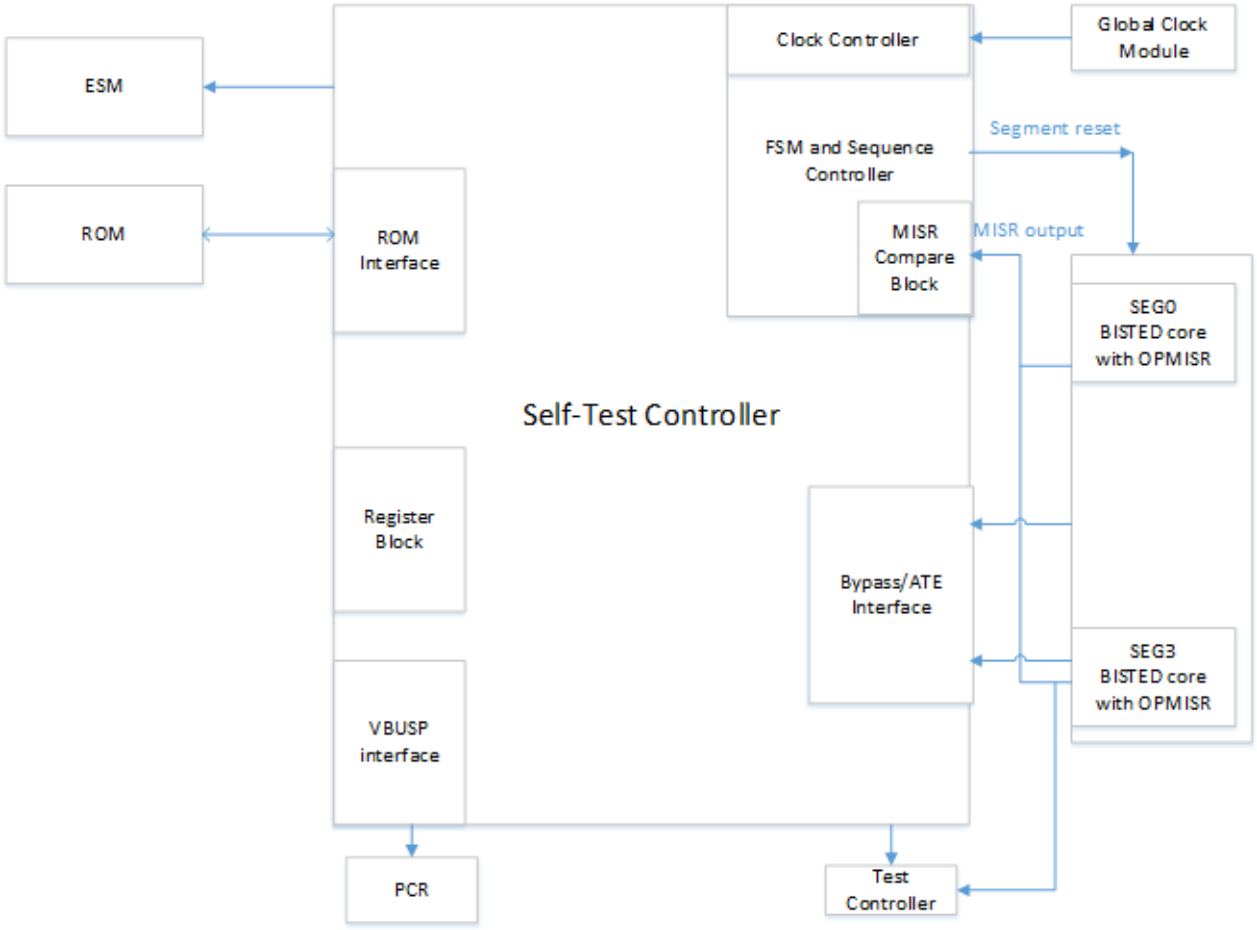


Figure 13-249. Block Diagram for STC With Multiple Segments

13.6.5.3 Module Description

13.6.5.3.1 ROM Interface

This block handles the ROM address and control signal generation to read the self-test microcode from the ROM. The test microcode, patterns, and golden signature value for each interval is stored in ROM.

Detailed information of the ROM microcode is available at ROM.

13.6.5.3.2 FSM and Sequence Control

This block generates the signals and data to OPMISR controller based on the test type and scan chain depth. The sequence of operation per interval is defined in Section 13.6.5.3.5.

13.6.5.3.2.1 Clock Control

The CLOCK CNTRL sub-block handles the clock selection and clock generation for ROM, OPMISR controller, and BIST'ed CORE clocks.

13.6.5.3.2.2 MISR Compare Block

At the end of the each self-test interval, an 896-bit MISR value from the OPMISR controller is shifted into NSTC. This is compared with the MISR_GOLDEN value, which is copied into a buffered register before the start of the interval. The result is updated into the status registers.

13.6.5.3.3 Register Block

This block implements the user-programmable control registers that determine when to start a self test, at what clock frequency the scan test should be performed, which segment to be selected for the test, how many pattern intervals to be completed before stopping, and so forth.

The register block also captures various status information of the self test for the user.

13.6.5.3.4 VBUSP Interface

The control and the status registers of the STC module can be accessed through the VBUSP interface. During application programming, configuration registers are programmed through the peripheral interface, to enable and run the self-test controller.

13.6.5.3.5 STC Flow

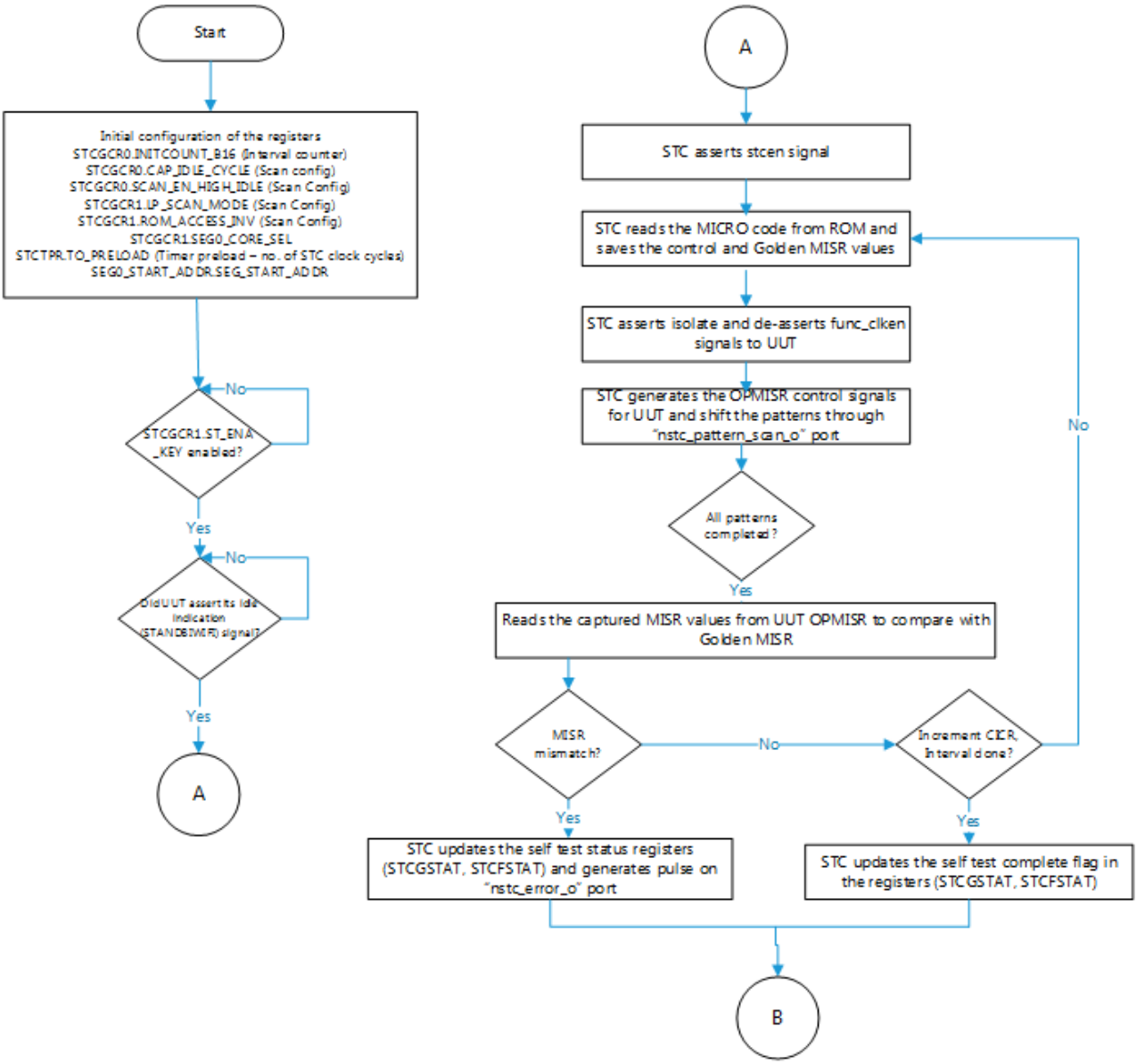


Figure 13-250. STC Flow (1 of 2)

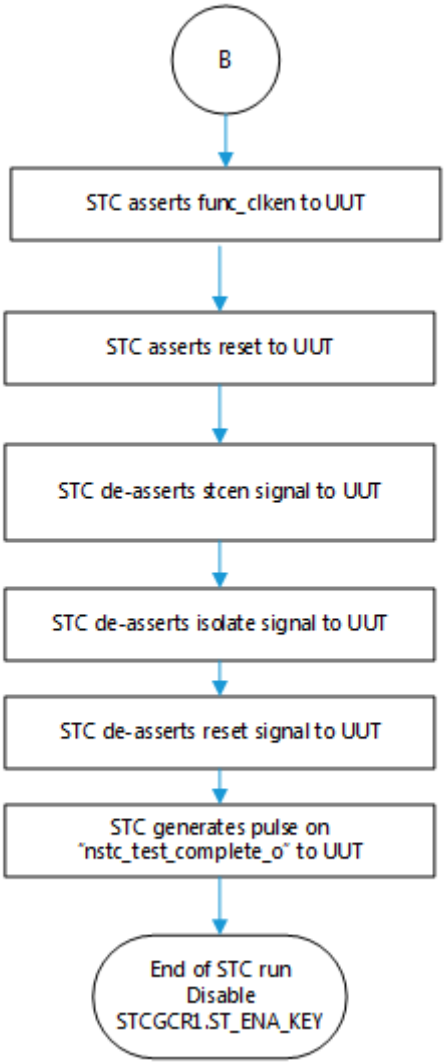


Figure 13-251. STC Flow (2 of 2)

13.6.5.3.6 Programming Sequence

The following sequence describes the step-by-step guide to trigger a logic Self-Test operation the device cores.

Table 13-275. STC - Programming Sequence (Default Mode)

Step No.	Steps	Register/Bit Field/Programming (For R5SS0/R5SS1/HSM)	Value
1	Configure the number of intervals to be run	STC.STCGCR0.INTCOUNT_B16	0x1
2	Configure both cores for Logic Self-Test.	STC.STCGCR1.SEG0_CORE_SEL	0x1
3	Scan mode configuration. Fixed Configuration – Only this configuration value is supported.	STC.STCGCR1.LP_SCAN_MODE	0x0
4	Scan mode configuration. Fixed Configuration – Only this configuration value is supported.	STC.STCGCR1.CODEC_SPREAD_MODE	0x1
5	Scan mode configuration. Fixed Configuration – Only this configuration value is supported.	STC.STCGCR0.CAP_IDLE_CYCLE	0x3

Table 13-275. STC - Programming Sequence (Default Mode) (continued)

6	Scan mode configuration. Fixed Configuration – Only this configuration value is supported.	STC.STCGCR0.SCANEN_HIGH_CAP_IDLE_CYCLE	0x3
7	Program the Timer Register for max run time	STC.STCTPR	0x18E2E
8	Program the Clock Divider Register – Maximum frequency of STC – 200MHz	STC.STC_CLKDIV. CLKDIV0	0x1
9	Program the STC ROM start address	STC.SEG0_START_ADDR.SEG_START_ADDR	0x0
10	Configure the pointer for STC ROM start address	STC.STCGCR0.RS_CNT_B1	0x1
11	Configure this register to disable STC diagnostic check	STC.STCSCSCR.FAULT_INS_B1	0x0
12	Disable the key for STC diagnostic check	STC.STCSCSCR. SELF_CHECK_KEY_B4	0x0
13	Kick off the test	STC.STCGCR1.ST_ENA_B4	0xA
14	Wait for standby – WFI signal from UUT (idle)		
15	Wait for Test done Interrupt or ESM error (Test done interrupt for R5SS0 is routed to R5SS1 and vice versa)		
16	Read the status register to check the STC test completion.	STC.STCGSTAT.TEST_DONE	0x1(READ)

Table 13-276. STC - Programming Sequence (WFI Override Mode)

Step No.	Steps	Register/Bit Field/Programming (For R5SS0/R5SS1/HSM)	Value
1	Configure the number of intervals to be run	STC.STCGCR0.INTCOUNT_B16	0x1
2	Configure both/single core(s) for Logic Self-Test.	STC.STCGCR1.SEG0_CORE_SEL	0x1
3	Scan mode configuration. Fixed Configuration – Only this configuration value is supported.	STC.STCGCR1.LP_SCAN_MODE	0x0
4	Scan mode configuration. Fixed Configuration – Only this configuration value is supported.	STC.STCGCR1.CODEC_SPREAD_MODE	0x1
5	Scan mode configuration. Fixed Configuration – Only this configuration value is supported.	STC.STCGCR0.CAP_IDLE_DELAY_CYCLE	0x3
6	Scan mode configuration. Fixed Configuration – Only this configuration value is supported.	STC.STCGCR0.SCANEN_HIGH_CAP_IDLE_DELAY_CYCLE	0x3
7	Program the timer register for max run time	STC.STCTPR	0x18E2E
8	Program the Clock Divider Register – Maximum frequency of STC – 200MHz	STC.STC_CLKDIV. CLKDIV0	0x1
9	Program the STC ROM start address	STC.SEG0_START_ADD.SEG_START_ADDR	0x0
10	Configure the pointer for STC ROM start address	STC.STCGCR0.RS_CNT_B1	0x1
11	Configure this register to disable STC diagnostic check	STC.STCSCSCR.FAULT_INS_B1	0x0
12	Disable the key for STC diagnostic check	STC.STCSCSCR.SELF_CHECK_KEY_B4	0x0
13	Kick off the test	STC.STCGCR1.ST_ENA_B4	0xA
14	Provide override WFI signal to STC indicating processor idle state	MSS_CTRL.R5SS*_FORCE_WFI.CR5_WFI_OVERIDE	0x7
15	Wait for Test done Interrupt or ESM error (Test done interrupt for R5SS0 is routed to R5SS1 and vice versa)		

Table 13-276. STC - Programming Sequence (WFI Override Mode) (continued)

16	Read the status register to check the STC test completion.	STC.STCGSTAT.TEST_DONE	0x1(READ)
17	Read the register to check the failure status of the STC test.	STC.STCGSTAT.TEST_FAIL	(READ) 0x0 - No failure

13.6.5.3.7 ROM Organization

Table 13-277. ROM Organization for 2 Intervals

COMMENTS	55:40	41:32	31:16	15:8	7:4	3	2	1	0
INTERVAL 0									
CFG for interval 0, when rom_access_inversion =0	Reserved	pattern_count[9:0]	Reserved	Reserved	Reserved	Seg_ID[1]	Seg_ID[0]	FT	TR_T
MISR for interval 0, when rom_access_inversion =0	MISR_GOLDEN[895:840]								
	MISR_GOLDEN[839:784]								
	MISR_GOLDEN[783:728]								
	MISR_GOLDEN[727:672]								
	MISR_GOLDEN[671:616]								
	MISR_GOLDEN[615:560]								
	MISR_GOLDEN[559:504]								
	MISR_GOLDEN[503:448]								
	MISR_GOLDEN[447:392]								
	MISR_GOLDEN[391:336]								
	MISR_GOLDEN[335:280]								
	MISR_GOLDEN[279:224]								
	MISR_GOLDEN[223:168]								
	MISR_GOLDEN[167:112]								
	MISR_GOLDEN[111:56]								
	MISR_GOLDEN[55:0]								
LP_MISR for interval 0, when rom_access_inversion =0	LP_MISR_GOLDEN[895:840]								
	LP_MISR_GOLDEN[839:784]								
	LP_MISR_GOLDEN[783:728]								
	LP_MISR_GOLDEN[727:672]								
	LP_MISR_GOLDEN[671:616]								
	LP_MISR_GOLDEN[615:560]								
	LP_MISR_GOLDEN[559:504]								
	LP_MISR_GOLDEN[503:448]								
	LP_MISR_GOLDEN[447:392]								
	LP_MISR_GOLDEN[391:336]								
	LP_MISR_GOLDEN[335:280]								
	LP_MISR_GOLDEN[279:224]								
	LP_MISR_GOLDEN[223:168]								
	LP_MISR_GOLDEN[167:112]								
	LP_MISR_GOLDEN[111:56]								
	LP_MISR_GOLDEN[55:0]								
Patterns for interval 0	P1_SD8[6:0]	P1_SD7[6:0]	P1_SD6[6:0]	….	….	…	P1_SD1[6:0]		
		…	…	…	…	…	…	P1_SD9[6:0]	
		…	…	…	…	…	…	…	…

Table 13-277. ROM Organization for 2 Intervals (continued)

COMMENTS	55:40	41:32	31:16	15:8	7:4	3	2	1	0
LP_MISR for interval 0, when rom_access_inversion =1	LP_INV_MISR_GOLDEN[55:0]								
	LP_INV_MISR_GOLDEN[111:56]								
	LP_INV_MISR_GOLDEN[167:112]								
	LP_INV_MISR_GOLDEN[223:168]								
	LP_INV_MISR_GOLDEN[279:224]								
	LP_INV_MISR_GOLDEN[335:280]								
	LP_INV_MISR_GOLDEN[391:336]								
	LP_INV_MISR_GOLDEN[447:392]								
	LP_INV_MISR_GOLDEN[503:448]								
	LP_INV_MISR_GOLDEN[559:504]								
	LP_INV_MISR_GOLDEN[615:560]								
	LP_INV_MISR_GOLDEN[671:616]								
	LP_INV_MISR_GOLDEN[727:672]								
	LP_INV_MISR_GOLDEN[783:728]								
	LP_INV_MISR_GOLDEN[839:784]								
	LP_INV_MISR_GOLDEN[895:840]								
MISR for interval 0, when rom_access_inversion =1	INV_MISR_GOLDEN[55:0]								
	INV_MISR_GOLDEN[111:56]								
	INV_MISR_GOLDEN[167:112]								
	INV_MISR_GOLDEN[223:168]								
	INV_MISR_GOLDEN[279:224]								
	INV_MISR_GOLDEN[335:280]								
	INV_MISR_GOLDEN[391:336]								
	INV_MISR_GOLDEN[447:392]								
	INV_MISR_GOLDEN[503:448]								
	INV_MISR_GOLDEN[559:504]								
	INV_MISR_GOLDEN[615:560]								
	INV_MISR_GOLDEN[671:616]								
	INV_MISR_GOLDEN[727:672]								
	INV_MISR_GOLDEN[783:728]								
	INV_MISR_GOLDEN[839:784]								
	INV_MISR_GOLDEN[895:840]								
CFG for interval 0, when rom_access_inversion =1 (same as when_rom_access_inversion =0)	Reserved	pattern_count[9:0]	Reserved	Reserved	Reserved	Seg_ID[1]	Seg_ID[0]	FT	TR_T
INTERVAL 1									
CFG for interval 1, when rom_access_inversion =0	Reserved	pattern_count[9:0]	Reserved	Reserved	Reserved	Seg_ID[1]	Seg_ID[0]	FT	TR_T

Table 13-277. ROM Organization for 2 Intervals (continued)

COMMENTS	55:40	41:32	31:16	15:8	7:4	3	2	1	0
MISR for interval 1, when rom_access_inversion =0	MISR_GOLDEN[895:840]								
	MISR_GOLDEN[839:784]								
	MISR_GOLDEN[783:728]								
	MISR_GOLDEN[727:672]								
	MISR_GOLDEN[671:616]								
	MISR_GOLDEN[615:560]								
	MISR_GOLDEN[559:504]								
	MISR_GOLDEN[503:448]								
	MISR_GOLDEN[447:392]								
	MISR_GOLDEN[391:336]								
	MISR_GOLDEN[335:280]								
	MISR_GOLDEN[279:224]								
	MISR_GOLDEN[223:168]								
	MISR_GOLDEN[167:112]								
	MISR_GOLDEN[111:56]								
	MISR_GOLDEN[55:0]								
LP_MISR for interval 1, when rom_access_inversion =0	LP_MISR_GOLDEN[895:840]								
	LP_MISR_GOLDEN[839:784]								
	LP_MISR_GOLDEN[783:728]								
	LP_MISR_GOLDEN[727:672]								
	LP_MISR_GOLDEN[671:616]								
	LP_MISR_GOLDEN[615:560]								
	LP_MISR_GOLDEN[559:504]								
	LP_MISR_GOLDEN[503:448]								
	LP_MISR_GOLDEN[447:392]								
	LP_MISR_GOLDEN[391:336]								
	LP_MISR_GOLDEN[335:280]								
	LP_MISR_GOLDEN[279:224]								
	LP_MISR_GOLDEN[223:168]								
	LP_MISR_GOLDEN[167:112]								
	LP_MISR_GOLDEN[111:56]								
	LP_MISR_GOLDEN[55:0]								
Patterns for interval 1	P1_SD8[6:0]	P1_SD7[6:0]	P1_SD6[6:0]	….	….	…	P1_SD1[6:0]		
		…	…	…	…	…	…	P1_SD9[6:0]	
		…	…	…	…	…	…	…	…

Table 13-277. ROM Organization for 2 Intervals (continued)

COMMENTS	55:40	41:32	31:16	15:8	7:4	3	2	1	0
LP_MISR for interval 1, when rom_access_inversion =1	LP_INV_MISR_GOLDEN[55:0]								
	LP_INV_MISR_GOLDEN[111:56]								
	LP_INV_MISR_GOLDEN[167:112]								
	LP_INV_MISR_GOLDEN[223:168]								
	LP_INV_MISR_GOLDEN[279:224]								
	LP_INV_MISR_GOLDEN[335:280]								
	LP_INV_MISR_GOLDEN[391:336]								
	LP_INV_MISR_GOLDEN[447:392]								
	LP_INV_MISR_GOLDEN[503:448]								
	LP_INV_MISR_GOLDEN[559:504]								
	LP_INV_MISR_GOLDEN[615:560]								
	LP_INV_MISR_GOLDEN[671:616]								
	LP_INV_MISR_GOLDEN[727:672]								
	LP_INV_MISR_GOLDEN[783:728]								
	LP_INV_MISR_GOLDEN[839:784]								
	LP_INV_MISR_GOLDEN[895:840]								
MISR for interval 1, when rom_access_inversion =1	INV_MISR_GOLDEN[55:0]								
	INV_MISR_GOLDEN[111:56]								
	INV_MISR_GOLDEN[167:112]								
	INV_MISR_GOLDEN[223:168]								
	INV_MISR_GOLDEN[279:224]								
	INV_MISR_GOLDEN[335:280]								
	INV_MISR_GOLDEN[391:336]								
	INV_MISR_GOLDEN[447:392]								
	INV_MISR_GOLDEN[503:448]								
	INV_MISR_GOLDEN[559:504]								
	INV_MISR_GOLDEN[615:560]								
	INV_MISR_GOLDEN[671:616]								
	INV_MISR_GOLDEN[727:672]								
	INV_MISR_GOLDEN[783:728]								
	INV_MISR_GOLDEN[839:784]								
	INV_MISR_GOLDEN[895:840]								
CFG for interval 1, when rom_access_inversion =1 (same as when_rom_access_inversion =0)	Reserved	pattern_count[9:0]	Reserved	Reserved	Reserved	Seg_ID[1]	Seg_ID[0]	FT	TR_T

The ROM contains the data to be processed by STC for the self-test run. This includes the control fields such as Segment ID, Pattern Count, and Golden MISR value for the STC, and the pattern scan data for the OPMISR controller.

The ROM space is divided into chunks, with each chunk containing the data corresponding to one OPMISR interval. The size required for an interval varies depending on the number patterns packed into the interval and the length of internal scan chains required.

Because each interval requires 64 rows of ROM for storing control and Golden MISR values, minimizing the number of intervals by packing more patterns into each interval provides the best ROM size. This works best if the self-test must be run only as a part of the boot-up sequence. However, if the self-test is performed during

application IDLE time, the number of patterns that can be packed into each interval will be dictated by the IDLE time available for the self-test, because an interval is the smallest granularity of a self-test run.

Details of the ROM image micro-code fields are given in the following sections.

13.6.5.3.7.1 TR_T: Transition Delay Methodology Type

This specifies the transition delay methodology for the current transition delay interval.

0	Launch-on-System-Clock

13.6.5.3.7.2 FT: Fault Model for the BIST Run

This specifies the fault model for the current interval of the test.

0	Stuck-at

13.6.5.3.7.3 SEG_ID[1:0]

This indicates which logical segment is selected for the associated interval during the self-test run.

SEG_SEL[1:0]	Segment Under Test
00	Segment 0
01	Segment 1
10	Segment 2
11	Segment 3

13.6.5.3.7.4 Pattern Count (patt_count[9:0])

This specifies the number of scan data patterns within a self-test interval. The pattern counts can vary from a minimum of 2 to a maximum of 1024.

patt_count[9:0]	Patterns per Interval
00_0000_0000	Not a valid interval [defaults to 2 patterns per interval]
00_0000_0001	2 patterns per interval
00_0000_0010	3 patterns per interval
...	...
11_1111_1110	1023 patterns per interval
11_1111_1111	1024 patterns per interval

13.6.5.3.7.5 MISR_GOLDEN[895:0]: Golden Signature Data Bits

This part of ROM contains the golden signature data of the current interval. This value is used to compare with the actual MISR value, when ST_GCR1.ROM_ACCESS_INV=0 and ST_GCR1.LP_SCAN_MODE=0, to generate the pass/fail information of the interval.

13.6.5.3.7.6 LP_MISR_GOLDEN[895:0]: Low Power Mode Golden Signature Data Bits

This part of ROM contains the LP golden signature data of the current interval. This value is used to compare with the actual MISR value, when STCGCR1.ROM_ACCESS_INV=0 and STCGCR1.LP_SCAN_MODE=1, to generate the pass/fail information of the interval.

13.6.5.3.7.7 INV_MISR_GOLDEN[895:0]: Inverse Mode Golden Signature Data Bits

This part of ROM contains the inverse mode golden signature data of the current interval. This value is used to compare with the actual MISR value, when STCGCR1.ROM_ACCESS_INV=1 and STCGCR1.LP_SCAN_MODE=0, to generate the pass/fail information of the interval.

13.6.5.3.7.8 LP_INV_MISR_GOLDEN[895:0]: Low Power Inverse Mode Golden Signature Data Bits

This part of ROM contains the low-power inverse mode golden signature data of the current interval. This value is used to compare with the actual MISR value, when STCGCR1.ROM_ACCESS_INV=1 and STCGCR1.LP_SCAN_MODE=1, to generate the pass/fail information of the interval.

13.6.5.3.7.9 Pn_SDm[7:0] (n - no. of patterns, m - scan chain length): OP-MISR Scan Data

This part of the ROM contains the scan data corresponding to each pattern. Each interval can have n number of scan patterns, as defined in the patt_count field. The number of 7bits of scan data in a pattern is equal to the length of the scan chain formed inside the UUT.

14 On-Chip Debug

The debug subsystem contains the OneMCU DEBUGSS at its core and enables JTAG interface access to device components. The debug subsystem is designed to provide the following debug features:

- JTAG debug access to debug resources, mapped through an ARM SWJ-DP and TI ICEPickM scan module
- System memory access without halting the processor
- ETM-based trace for ARM R5F
- Cross trigger to halt and restart cores and peripherals based on events such as watchdog, timers, DMA, and time-stamp events
- Capability to read the device ID data

14.1 On-Chip Debug

This chapter describes the on-chip debug support, including details on the various capabilities and features available through the SoC debug framework.

14.1.1 On-Chip Debug Overview

This chapter describes the properties and capabilities of the various features available through the On-Chip Debug framework deployed on this device - also known as Debug SS.

The On-Chip Debug framework enables various Debug and Trace use-cases, including:

- JTAG Tooling – Access to on-chip debug resources is supported by an IEEE 1149.1 (JTAG) compliant interface that is supported by an Arm® CoreSight™ DAP JTAG-DP.
- Self-Hosted Tooling – code running on programmable cores within the device is able to use on-chip debug resources to enable embedded tooling solutions.
- PCB-level interconnect testing – IEEE 1149.1 and IEEE 1149.6 compliant Boundary Scan supports product level integration testing
- Stop Mode debugging – Debug of embedded processors is supported using various mechanisms that can halt the pipeline of a CPU. Breakpoints (Software and Hardware), Watchpoints, Cross-Triggering, and On-demand (e.g. user requested) halt request mechanisms may be supported based on the capabilities of a given processor.
- Debug-aware Peripherals – Peripheral awareness of processor execution state allows safe suspension of peripheral operation. Supported by select peripherals.
- Synchronized Debug – Wide deployment of Cross-Triggering allows multiple processors and/or debug elements to be grouped together to process various actions based on a common event occurrence.
- Processor Trace – Support for the generation of a trace stream with the encoding of processor state that may include some combination of program flow, timing details (execution and stall), and memory references (address and/or data) with the goal of facilitating processor state reconstruction for debug purposes.
- Software Messaging Trace – Support for software messaging trace where embedded code running within the device can be instrumented to use memory writes to send important debug information to a trace stream.
- Trace Correlation (through timestamping) – Support for the correlation of different trace streams is enabled through the use of a common global timestamp that is distributed to supported trace sources.
- Trace data movement – Trace data movement on chip is supported using standard Arm ATB trace infrastructure components. Concurrent use of the trace bus by multiple trace sources is supported, with each trace source identifiable through a unique ID. An Arm® CoreSight™ Trace Router supports sending a trace stream off-chip (TPIU), to dedicated memory on-chip (ETB), or broadcasted to both (TPIU + ETB).
- The trace buffer used for trace data movement is ARM CSETB 32KB. Refer to ARM CSETB TRM for more details.
- On-Chip trace collection via dedicated buffer – An on-chip trace buffer is supported by logic that implements capturing trace data until either the memory fills (stop-on-full, system-bridge) or continuously until a request to stop is received (circular buffer). Interleaving of multiple trace streams is made possible through the use of a standardized encoding that embeds trace data along with the corresponding trace source ID.
- Trace export over TPIU – Trace data is exported over device LVCMOS pins using a standard protocol that embeds trace data along with the corresponding trace source ID.

14.1.2 On-Chip Debug Features

The On-Chip Debug framework provides a comprehensive hardware platform for a rich debug and development experience. The On-Chip Debug framework in AM263Px supports these features:

- An IEEE 1149.1 (JTAG and Boundary Scan) compliant device interface to provide debug access to debug resources through ARM SWJ-DP module.
- System Memory access without halting the processors
- Trace port device interface
- ETM based program flow trace for ARM R5F
- Software instrumentation trace using STM
- Breakpoint-based debug
- Cross-trigger to halt and restart various R5F cores and M4 CPU based on SOC internal and external events such as timers and other peripheral interrupts

- Trace capture on-chip via dedicated buffer
- Arm® CoreSight™ compliant debug components deployed to streamline 3rd party tooling support

14.1.3 On-Chip Debug Functional Description

14.1.3.1 On-Chip Debug Block Diagram

The Debug subsystem is responsible for supporting the debug features of AM263Px device.

An overview of the interconnectivity of the debug ports and trace ports are shown in Figure 14-1.

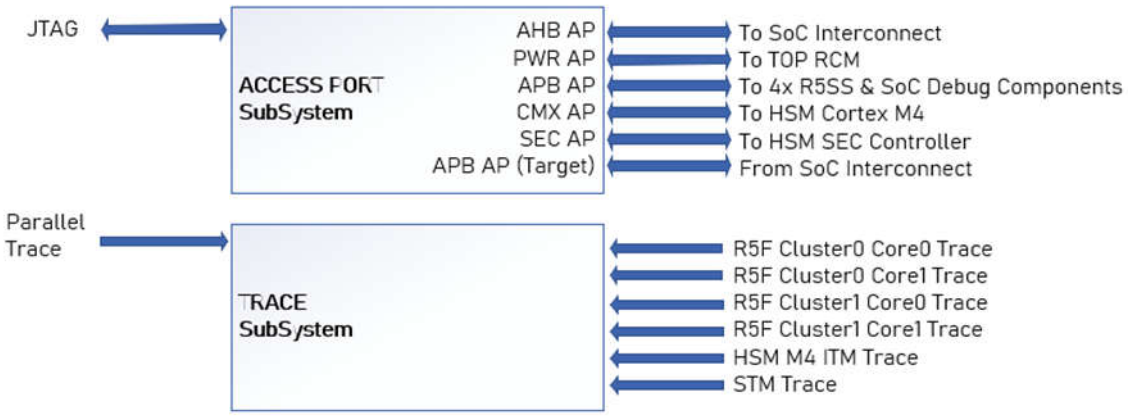


Figure 14-1. Debug SS Overview

A logical partitioning of the On-Chip Debug features deployed on this device is illustrated in Figure 14-2.

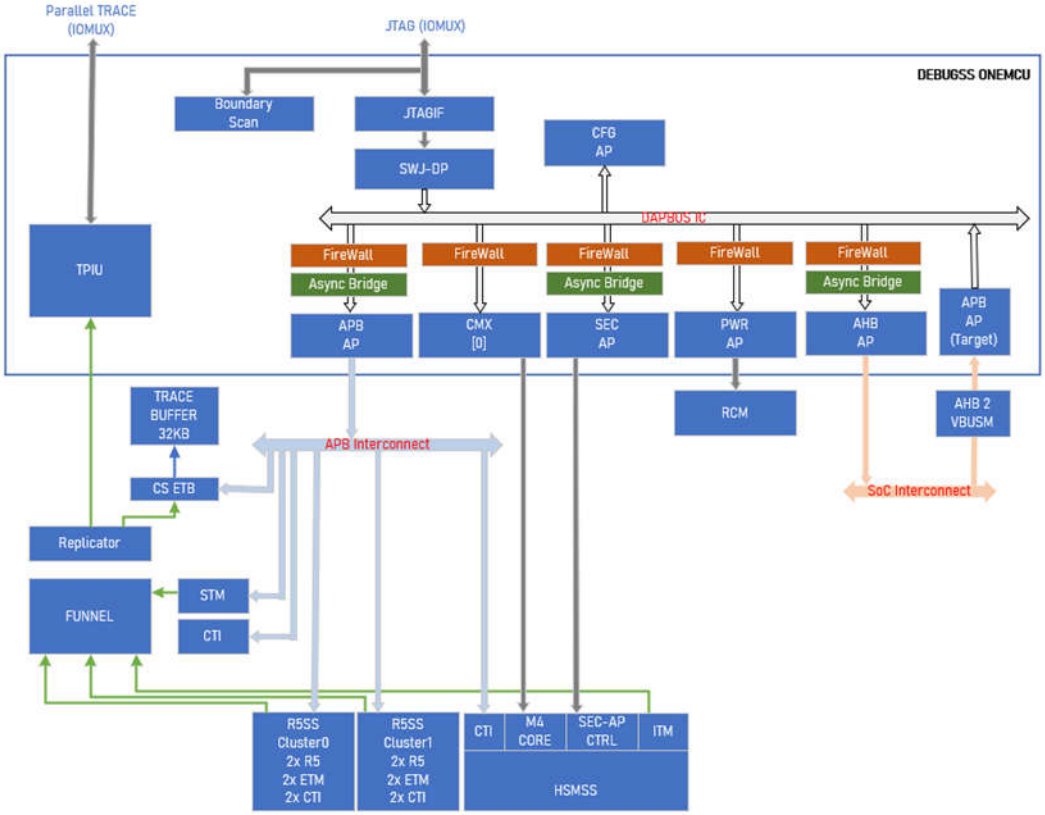


Figure 14-2. On Chip Debug Block Diagram

14.1.3.2 Device Interfaces

On-Chip Debug features are supported through two device interfaces.

JTAG: IEEE 1149.1 compliant interface that provides access to Boundary Scan and acts as the primary interface for off-chip access to On-Chip debug resources (see Section 14.1.3.2.1).

Trace Port: Arm TPIU compliant Trace Port interface is used to facilitate export of trace (see Section 14.1.3.2.2).

Texas Instruments supports a variety of eXtended Development System (XDS) JTAG controllers with various debug capabilities beyond only JTAG support. The following document is a good reference for guidelines: Emulation and Trace Headers. More information can also be found here:XDS Target Connection Guide.

14.1.3.2.1 JTAG Interface

Table 14-1. JTAG Interface Signals

Signal Name	I/O Type	Description
TCK	I	*Test Clock*. Controls the timing of the test interface independently from any system clocks. TCK is pulsed by the equipment controlling the test and not by the tested device.
TMS	I	*Test Mode Select*. Controls the transitions of the test interface state machine
TDI	I	*Test Data Input*. Supplies the data to the JTAG registers
TDO	O/Z	*Test Data Output*. Used to serially output the data from the JTAG registers to the equipment controlling the test.

14.1.3.2.2 Trace Port Interface

Table 14-2. Trace Port Signals

Signal Name	I/O Type	Description
TRC_CLK	O	Trace Clock
TRC_CTL	O	Trace Control
TRC_DATA[15:0]	O	Trace Data

Note

The Trace Port interface signals are associated with device pins that are multiplexed with other signal functions.

14.1.3.3 Debug and Boundary Scan Access and Control

On-Chip debug resources are made available through two mechanisms:

- JTAG access via DAP and related APs
- JTAG access via Boundary Scan TAP

14.1.3.3.1 DAP

Off-chip debug tools are able to access On-Chip debug resources via the JTAG interface.

A CoreSight™ Compliant DAP architecture provides access via a DP and a collection of APs:

- SWJ-DP: Arm® CoreSight™ compliant SWJ Debug Port provides support for a JTAG interface with a 4-bit IR Note: Even though an SWJ-DP is implemented on-chip, only JTAG is supported. This device does not support SWD.
- APB-AP: Arm® CoreSight™ APB Access Port provides access to the Debug-APB address space which is the primary configuration space for On-Chip Debug resources.
- AHB-AP: Arm® CoreSight™ AHB Access Port provides access to the SoC address space, allowing visibility and control over system resources.

- Config-AP: TI Configuration AP supports access to SoC debug management registers
- Power-AP: TI Configuration AP supports reset management

14.1.3.3.1.1 Debug Subsystem Address Map

The memory map view for DAP AHB is the same as SOC memory map view seen by Cortex R5F CPU (except for dedicated R5F Core memories and peripherals).

Table 14-3 shows the APB AP address map for AM263Px.

Table 14-3. APB AP Memory Map

APB PORT	Block Name	Start Address	End Address	Size	Register Details
APB INTERNAL PORT0	Debugss ROM Table	0x0000 0000	0x0000 0FFF	4KB	ROM LUT
APB INTERNAL PORT0	Debugss CTI	0x0000 1000	0x0000 1FFF	4KB	CTI register summary
APB INTERNAL PORT0	Debugss TPIU	0x0000 2000	0x0000 2FFF	4KB	TPIU register summary
APB EXTERNAL PORT 0	Ext Port0 ROM TABLE	0x0001 0000	0x0001 0FFF	4KB	ROM LUT
APB EXTERNAL PORT 0	ATB REPLICATOR	0x0001 1000	0x0001 1FFF	4KB	ATB register summary
APB EXTERNAL PORT 0	CSETB	0x0001 2000	0x0001 2FFF	4KB	ETB register summary
APB EXTERNAL PORT 0	STM	0x0001 3000	0x0001 3FFF	4KB	
APB EXTERNAL PORT 0	STM-CTI	0x0001 4000	0x0001 4FFF	4KB	CTI register summary
APB EXTERNAL PORT 0	HSM CM4 CTI	0x0001 5000	0x0001 5FFF	4KB	CTI register summary
APB EXTERNAL PORT 1	R5SS0 ROM Table	0x0002 0000	0x0002 0FFF	4KB	ROM LUT
APB EXTERNAL PORT 1	R5SS0 CPU0	0x0003 0000	0x0003 0FFF	4KB	R5 Core Debug Register Summary
APB EXTERNAL PORT 1	R5SS0 CPU1	0x0003 2000	0x0003 2FFF	4KB	R5 Core Debug Register Summary
APB EXTERNAL PORT 1	R5SS0 CPU0 CTI	0x0003 8000	0x0003 8FFF	4KB	CTI register summary
APB EXTERNAL PORT 1	R5SS0 CPU1 CTI	0x0003 9000	0x0003 9FFF	4KB	CTI register summary
APB EXTERNAL PORT 1	R5SS0 CPU0 ETM	0x0003 C000	0x0003 CFFF	4KB	ETM register summary
APB EXTERNAL PORT 1	R5SS0 CPU1 ETM	0x0003 D000	0x0003 DFFF	4KB	ETM register summary
APB EXTERNAL PORT 2	R5SS1 ROM Table	0x0004 0000	0x0004 0FFF	4KB	ROM LUT
APB EXTERNAL PORT 2	R5SS1 CPU0	0x0005 0000	0x0005 0FFF	4KB	R5 Core Debug Register Summary
APB EXTERNAL PORT 2	R5SS1 CPU1	0x0005 2000	0x00052FFF	4KB	R5 Core Debug Register Summary
APB EXTERNAL PORT 2	R5SS1 CPU0 CTI	0x0005 8000	0x00058FFF	4KB	CTI register summary
APB EXTERNAL PORT 2	R5SS1 CPU1 CTI	0x0005 9000	0x00059FFF	4KB	CTI register summary
APB EXTERNAL PORT 2	R5SS1 CPU0 ETM	0x0005 C000	0x0005CFFF	4KB	ETM register summary
APB EXTERNAL PORT 2	R5SS1 CPU1 ETM	0x0005 D000	0x0005DFFF	4KB	ETM register summary

CTI register summary

Seehere for more details on the below registers:

Name	Offset	Type	Reset	Description
CTICONTROL	0x000	RW	0x00000000	CTI Control Register

Name	Offset	Type	Reset	Description
CTIINTACK	0x010	WO	0x00000000	CTI Interrupt Acknowledge Register
CTIAPPSET	0x014	RW	0x00000000	CTI Application Trigger Set Register
CTIAPPCLEAR	0x018	WO	0x00000000	CTI Application Trigger Clear Register
CTIAPPPULSE	0x01C	WO	0x00000000	CTI Application Pulse Register
CTIINEN0	0x020	RW	0x00000000	CTI Trigger 0 to Channel Enable Register
CTIINEN1	0x024	RW	0x00000000	CTI Trigger 1 to Channel Enable Register
CTIINEN2	0x028	RW	0x00000000	CTI Trigger 2 to Channel Enable Register
CTIINEN3	0x02C	RW	0x00000000	CTI Trigger 3 to Channel Enable Register
CTIINEN4	0x030	RW	0x00000000	CTI Trigger 4 to Channel Enable Register
CTIINEN5	0x034	RW	0x00000000	CTI Trigger 5 to Channel Enable Register
CTIINEN6	0x038	RW	0x00000000	CTI Trigger 6 to Channel Enable Register
CTIINEN7	0x03C	RW	0x00000000	CTI Trigger 7 to Channel Enable Register
CTIOUTEN0	0x0A0	RW	0x00000000	CTI Channel to Trigger 0 Enable Register
CTIOUTEN1	0x0A4	RW	0x00000000	CTI Channel to Trigger 1 Enable Register
CTIOUTEN2	0x0A8	RW	0x00000000	CTI Channel to Trigger 2 Enable Register
CTIOUTEN3	0x0AC	RW	0x00000000	CTI Channel to Trigger 3 Enable Register
CTIOUTEN4	0x0B0	RW	0x00000000	CTI Channel to Trigger 4 Enable Register
CTIOUTEN5	0x0B4	RW	0x00000000	CTI Channel to Trigger 5 Enable Register
CTIOUTEN6	0x0B8	RW	0x00000000	CTI Channel to Trigger 6 Enable Register
CTIOUTEN7	0x0BC	RW	0x00000000	CTI Channel to Trigger 7 Enable Register
CTITRIGINSTATUS	0x130	RO	0x00000000	CTI Trigger In Status Register
CTITRIGOUTSTATUS	0x134	RO	0x00000000	CTI Trigger Out Status Register
CTICHINSTATUS	0x138	RO	0x00000000	CTI Channel In Status Register
CTICHOUTSTATUS	0x13C	RO	0x00000000	CTI Channel Out Status Register
CTIGATE	0x140	RW	0x0000000F	Enable CTI Channel Gate Register
ASICCTL	0x144	RW	0x00000000	External Multiplexer Control Register
ITCHINACK	0xEDC	WO	0x00000000	ITCHINACK Register
ITTRIGINACK	0xEE0	WO	0x00000000	ITTRIGINACK Register
ITCHOUT	0xEE4	WO	0x00000000	ITCHOUT Register

Name	Offset	Type	Reset	Description
ITTRIGOUT	0xEE8	WO	0x00000000	ITTRIGOUT Register
ITCHOUTACK	0xEEC	RO	0x00000000	ITCHOUTACK Register
ITTRIGOUTACK	0xEF0	RO	0x00000000	ITTRIGOUTACK Register
ITCHIN	0xEF4	RO	0x00000000	ITCHIN Register
ITTRIGIN	0xEF8	RO	0x00000000	ITTRIGIN Register
ITCTRL	0xF00	RW	0x00000000	Integration Mode Control Register
CLAIMSET	0xFA0	RW	0x0000000F	Claim Tag Set Register
CLAIMCLR	0xFA4	RW	0x00000000	Claim Tag Clear Register
LAR	0xFB0	WO	0x00000000	Lock Access Register
LSR	0xFB4	RO	0x00000003	Lock Status Register
AUTHSTATUS	0xFB8	RO	0x00000005	Authentication Status Register
DEVID	0xFC8	RO	0x00040800	Device Configuration Register
DEVTYPE	0xFCC	RO	0x00000014	Device Type Identifier Register
PIDR4	0xFD0	RO	0x00000004	Peripheral ID4 Register
PIDR5	0xFD4	RO	0x00000000	Peripheral ID5 Registers
PIDR6	0xFD8	RO	0x00000000	Peripheral ID6 Registers
PIDR7	0xFDC	RO	0x00000000	Peripheral ID7 Registers
PIDR0	0xFE0	RO	0x00000006	Peripheral ID0 Register
PIDR1	0xFE4	RO	0x000000B9	Peripheral ID1 Register
PIDR2	0xFE8	RO	0x0000003B	Peripheral ID2 Register
PIDR3	0xFEC	RO	0x00000000	Peripheral ID3 Register
CIDR0	0xFF0	RO	0x0000000D	Component ID0 Register
CIDR1	0xFF4	RO	0x00000090	Component ID1 Register
CIDR2	0xFF8	RO	0x00000005	Component ID2 Register
CIDR3	0xFFC	RO	0x000000B1	Component ID3 Register

ETB Register Summary

See here for more details on the below registers:

Name	Offset	Type	Reset	Description
RDP	0x004	RO	0x00000000	ETB RAM Depth Register
STS	0x00C	RO	0x00000008	ETB Status Register
RRD	0x010	RO	0x00000000	ETB RAM Read Data Register
RRP	0x014	RW	0x00000000	ETB RAM Read Pointer Register
RWP	0x018	RW	0x00000000	ETB RAM Write Pointer Register
TRG	0x01C	RW	0x00000000	ETB Trigger Counter Register
CTL	0x020	RW	0x00000000	ETB Control Register
RWD	0x024	WO	0x00000000	ETB RAM Write Data Register
FFSR	0x300	RO	0x00000002	ETB Formatter and Flush Status Register
FFCR	0x304	RW	0x00000000	ETB Formatter and Flush Control Register
ITMISCOP0	0xEE0	WO	0x00000000	Integration Test Miscellaneous Output Register 0
ITTRFLINACK	0xEE4	WO	0x00000000	Integration Test Trigger In and Flush In Acknowledge Register
ITTRFLIN	0xEE8	RO	0x00000000	Integration Test Trigger In and Flush In Register
ITATBDATA0	0xEEC	RO	0x00000000	Integration Test ATB Data Register 0
ITATBCTR2	0xEF0	WO	0x00000000	Integration Test ATB Control Register 2
ITATBCTR1	0xEF4	RO	0x00000000	Integration Test ATB Control Register 1
ITATBCTR0	0xEF8	RO	0x00000000	Integration Test ATB Control Register 0
ITCTRL	0xF00	RW	0x00000000	Integration Mode Control Register
CLAIMSET	0xFA0	RW	0x0000000F	Claim Tag Set Register
CLAIMCLR	0xFA4	RW	0x00000000	Claim Tag Clear Register
LAR	0xFB0	WO	0x00000000	Lock Access Register
LSR	0xFB4	RO	0x00000003	Lock Status Register
AUTHSTATUS	0xFB8	RO	0x00000000	Authentication Status Register
DEVID	0xFC8	RO	0x00000000	Device Configuration Register
DEVTYPE	0xFCC	RO	0x00000021	Device Type Identifier Register
PIDR4	0xFD0	RO	0x00000004	Peripheral ID4 Register
PIDR5	0xFD4	RO	0x00000000	Peripheral ID5 Registers
PIDR6	0xFD8	RO	0x00000000	Peripheral ID6 Registers
PIDR7	0xFDC	RO	0x00000000	Peripheral ID7 Registers

Name	Offset	Type	Reset	Description
PIDR0	0xFE0	RO	0x00000007	Peripheral ID0 Register
PIDR1	0xFE4	RO	0x000000B9	Peripheral ID1 Register
PIDR2	0xFE8	RO	0x0000003B	Peripheral ID2 Register
PIDR3	0xFEC	RO	0x00000000	Peripheral ID3 Register
CIDR0	0xFF0	RO	0x0000000D	Component ID0 Register
CIDR1	0xFF4	RO	0x00000090	Component ID1 Register
CIDR2	0xFF8	RO	0x00000005	Component ID2 Register
CIDR3	0xFFC	RO	0x000000B1	Component ID3 Register

ATB Register Summary

See here for more details on the below registers:

Name	Offset	Type	Reset	Description
Ctrl_Reg	0x000	RW	0x00000300	Funnel Control Register
Priority_Ctrl_Reg	0x004	RW	0x00000000	Priority Control Register
ITATBDATA0	0xEEC	RW	0x00000000	Integration Test ATB Data0 Register
ITATBCTR2	0xEF0	RW	0x00000000	Integration Test ATB Control 2 Register
ITATBCTR1	0xEF4	RW	0x00000000	Integration Test ATB Control 1 Register
ITATBCTR0	0xEF8	RW	0x00000000	Integration Test ATB Control 0 Register
ITCTRL	0xF00	RW	0x00000000	Integration Mode Control Register
CLAIMSET	0xFA0	RW	0x0000000F	Claim Tag Set Register
CLAIMCLR	0xFA4	RW	0x00000000	Claim Tag Clear Register
LOCKACCESS	0xFB0	WO	0x00000000	Lock Access Register
LOCKSTATUS	0xFB4	RO	0x00000003	Lock Status Register
AUTHSTATUS	0xFB8	RO	0x00000000	Authentication Status Register
DEVID	0xFC8	RO	0x00000038	Device Configuration Register
DEVTYPE	0xFCC	RO	0x00000012	Device Type Identifier Register
PIDR4	0xFD0	RO	0x00000004	Peripheral ID4 Register
PIDR5	0xFD4	RO	0x00000000	Peripheral ID5 Registers
PIDR6	0xFD8	RO	0x00000000	Peripheral ID6 Registers

Name	Offset	Type	Reset	Description
PIDR7	0xFDC	RO	0x00000000	Peripheral ID7 Registers
PIDR0	0xFE0	RO	0x00000008	Peripheral ID0 Register
PIDR1	0xFE4	RO	0x000000B9	Peripheral ID1 Register
PIDR2	0xFE8	RO	0x0000002B	Peripheral ID2 Register
PIDR3	0xFEC	RO	0x00000000	Peripheral ID3 Register
CIDR0	0xFF0	RO	0x0000000D	Component ID0 Register
CIDR1	0xFF4	RO	0x00000090	Component ID1 Register
CIDR2	0xFF8	RO	0x00000005	Component ID2 Register
CIDR3	0xFFC	RO	0x000000B1	Component ID3 Register

STM Register Summary

See here for more details on the below registers:

Name	Offset	Type	Reset	Description
STMDMASTARTR	0xC04	WO	-	ARM STM Specification
STMDMASTOPR	0xC08	WO	-	ARM STM Specification
STMDMASTATR	0xC0C	RO	-	ARM STM Specification
STMDMACTLR	0xC10	RW	0x00000000	DMA Control Register
STMDMAIDR	0xCFC	RO	0x00000002	ARM STM Specification
STMHEER	0xD00	RW	-	ARM STM Specification
STMHETER	0xD20	RW	-	ARM STM Specification
STMHEBSR	0xD60	RW	0x00000000	ARM STM Specification
STMHEMCR	0xD64	RW	0x00000000	ARM STM Specification
STMHEEXTMUXR	0xD68	RW	0x00000000	Hardware Event External Multiplex Control Register
STMHEMASTR	0xDF4	RO	0x00000080	Hardware Event Initiator Number Register
STMHEFEAT1R	0xDF8	RO	0x30200035	Hardware Event Features 1 Register
STMHEIDR	0xDFC	RO	0x00000011	Hardware Event ID Register
STMSPER	0xE00	RW	0x00000000	ARM STM Specification
STMSPTER	0xE20	RW	0x00000000	ARM STM Specification
STMSPSCR	0xE60	RW	0x00000000	ARM STM Specification

Name	Offset	Type	Reset	Description
STMSPMSCR	0xE64	RW	0x00000000	ARM STM Specification
STMSPOVERRIDER	0xE68	RW	0x00000000	ARM STM Specification
STMSPMOVERRIDER	0xE6C	RW	0x00000000	ARM STM Specification
STMSPTRIGCSR	0xE70	RW	0x00000000	ARM STM Specification
STMTCSR	0xE80	RW	-	Trace Control and Status Register
STMTSSTIMR	0xE84	WO	-	ARM STM Specification
STMTSFREQR	0xE8C	RW	0x00000000	ARM STM Specification
STMSYNCR	0xE90	RW	0x00000000	ARM STM Specification
STMAUXCR	0xE94	RW	0x00000000	Auxiliary Control Register
STMFEAT1R	0xEA0	RO	0x006587D1	STM Features 1 Register
STMFEAT2R	0xEA4	RO	0x000114F2	STM Features 2 Register
STMFEAT3R	0xEA8	RO	0x0000007F	STM Features 3 Register
STMITTRIGGER	0xEE8	WO	-	Integration Test for Cross-trigger Outputs Register
STMITATBDATA0	0xEEC	WO	-	Integration Mode ATB Data 0 Register
STMITATBCTR2	0xEF0	RO	-	Integration Mode ATB Control 2 Register
STMITATBID	0xEF4	WO	-	Integration Mode ATB Identification Register
STMITATBCTR0	0xEF8	WO	-	Integration Mode ATB Control 0 Register
STMITCTRL	0xF00	RW	0x00000000	Integration Mode Control Register
STMCLAIMSET	0xFA0	RW	0x0000000F	ARM STM Specification
STMCLAIMCLR	0xFA4	RW	0x00000000	ARM STM Specification
STMLAR	0xFB0	WO	-	Lock Access Register
STMLSR	0xFB4	RO	-	Lock Status Register
STMAUTHSTATUS	0xFB8	RO	0x000000AA	Authentication Status Register
STMDEVARCH	0xFBC	RO	0x47710A63	Device Architecture Register
STMDEVID	0xFC8	RO	0x00010000	Device Configuration Register
STMDEVTYPE	0xFCC	RO	0x00000063	Device Type Identifier Register
STMPIDR0	0xFE0	RO	0x00000063	Peripheral ID0 Register
STMPIDR1	0xFE4	RO	0x000000B9	Peripheral ID1 Register
STMPIDR2	0xFE8	RO	0x0000000B	Peripheral ID2 Register

Name	Offset	Type	Reset	Description
STMPIDR3	0xFEC	RO	0x00000000	Peripheral ID3 Register
STMPIDR4	0xFD0	RO	0x00000004	Peripheral ID4 Register
STMCIDR0	0xFF0	RO	0x0000000D	Component ID0 Register
STMCIDR1	0xFF4	RO	0x00000090	Component ID1 Register
STMCIDR2	0xFF8	RO	0x00000005	Component ID2 Register
STMCIDR3	0xFFC	RO	0x000000B1	Component ID3 Register

ETM Register Summary

See here for more details on the below registers:

Name	Offset	Type	Reset	Description
ETMCR	0x000	RW	0x00000441	Main Control Register
ETMCCR	0x004	RO	0x8D014024	Configuration Code Register
ETMTRIGGER	0x008	RW	-	Trigger Event Register in the ARM ETM Specification
ETMASICCTLR	0x00C	RW	0x00000000	ASIC Control Register
ETMSR	0x010	RW	-	ETM Status Register in the ARM ETM Specification
ETMSCR	0x014	RO	0x00020C0C	System Configuration Register in the ARM ETM Specification
ETMTSSCR	0x018	RW	-	TraceEnable Start/Stop Control Register in the ARM ETM Specification
ETMTECR2	0x01C	RW	-	TraceEnable Control 2 Register in the ARM ETM Specification
ETMTEEVR	0x020	RW	-	TraceEnable Event Register in the ARM ETM Specification
ETMTECR1	0x024	RW	-	TraceEnable Control 1 Register in the ARM ETM Specification
ETMFFLR[e]	0x02C	RW	-	FIFOFULL Level Register in the ARM ETM Specification
ETMVDEVR	0x030	RW	-	ViewData Event Register in the ARM ETM Specification
ETMVDCR1	0x034	RW	-	ViewData Control 1 Register in the ARM ETM Specification
ETMVDCR3	0x03C	RW	-	ViewData Control 3 Register in the ARM ETM Specification
ETMACVR1-8	0x40 - 0x58	RW	-	Address Comparator Value Registers in the ARM ETM Specification
ETMACTR1-8	0x80 - 0x98	RW	-	Address Comparator Access Type Registers in the ARM ETM Specification
ETMDCVR1[f]	0x0C0	RW	-	Data Comparator Value Registers in the ARM ETM Specification

Name	Offset	Type	Reset	Description
ETMDCVR3[f]	0x0D0	RW	-	Data Comparator Value Registers in the ARM ETM Specification
ETMDCMR1[f]	0x100	RW	-	Data Comparator Mask Registers in the ARM ETM Specification
ETMDCMR3[f]	0x110	RW	-	Data Comparator Mask Registers in the ARM ETM Specification
ETMCNTRLDVR1-2	0x140, 0x144	RW	-	Counter Reload Value Registers in the ARM ETM Specification
ETMCNTENR1-2	0x150, 0x154	RW	-	Counter Enable Registers in the ARM ETM Specification
ETMCNTRLDEVR1-2	0x160, 0x164	RW	-	Counter Reload Event Registers in the ARM ETM Specification
ETMCNTVR1-2	0x170, 0x174	RW	-	Counter Value Registers in the ARM ETM Specification
ETMSQEVR	0x180 - 0x194	RW	-	Sequencer State Transition Event Registers in the ARM ETM Specification
ETMSQR	0x19C	RW	-	Current Sequencer State Register in the ARM ETM Specification
ETMEXTOUTEVR1-2	0x1A0, 0x1A4	RW	-	External Output Event Registers in the ARM ETM Specification
ETMCIDCVR	0x1B0	RW	-	Context ID Comparator Value Registers in the ARM ETM Specification
ETMCIDCMR	0x1BC	RW	-	Context ID Comparator Mask Register in the ARM ETM Specification
ETMSYNCFR	0x1E0	RW	0x00000400	Synchronization Frequency Register in the ARM ETM Specification
ETMIDR	0x1E4	RO	0x4104F23x	ID Register
ETMCCER	0x1E8	RO	0x000009BA	Configuration Code Extension Register
ETMEXTINSELR	0x1EC	RW	-	Extended External Input Selection Register
ETMTRACEIDR	0x200	RW	0x00000000	CoreSight Trace ID Register in the ARM ETM Specification
ETMPDSR	0x314	RO	-	Power-Down Status Register
ITETMIF	0xED8	RO [h]	-	Processor-ETM Interface Register
ITMISCOUT	0xEDC	WO	-	Miscellaneous Outputs Register
ITMISCIN	0xEE0	RO [h]	-	Miscellaneous Inputs Register
ITTRIGGERACK	0xEE4	RO [h]	-	Trigger Acknowledge Register
ITTRIGGERREQ	0xEE8	WO	-	Trigger Request Register
ITATBDATA0	0xEEC	WO	-	ATB Data Register 0
ITATBCTR2	0xEF0	RO [h]	-	ATB Control Register 2
ITATBCTR1	0xEF4	WO	-	ATB Control Register 1

Name	Offset	Type	Reset	Description
ITATBCTR0	0xEF8	WO	-	ATB Control Register 0
ETMITCTRL	0xF00	RW	0x00000000	Integration Mode Control Register in the ARM ETM Specification
ETMCLAIMSET	0xFA0	RW	0x000000FF	Claim Tag Set Register in the ARM ETM Specification
ETMCLAIMCLR	0xFA4	RW	0x00000000	Claim Tag Clear Register in the ARM ETM Specification
ETMLAR	0xFB0	WO	-	Lock Access Register in the ARM ETM Specification
ETMLSR	0xFB4	RO	0x00000003	Lock Status Register in the ARM ETM Specification
ETMAUTHSTATUS	0xFB8	RO	-	Authentication Status Register in the ARM ETM Specification
ETMDEVID	0xFC8	RO	0x00000000	CoreSight Device Configuration Register in the ARM ETM Specification
ETMDEVTYPE	0xFCC	RO	0x00000013	CoreSight Device Type Register in the ARM ETM Specification
ETMPIDR0-7	0xFD0 - 0xFEC	RO	-	Peripheral Identification Registers
ETMCIDR0-3	0xFF0 - 0xFFC	RO	-	ETM Component Identification Registers

TPIU Register Summary

See here for more details on the below registers:

Name	Offset	Type	Reset	Description
TPIU_SPORTSZ	0x000	RO	0x00000001	Supported Port Size Register
TPIU_CPORTSZ	0x004	RW	0x00000001	Current Port Size Register
TPIU_STRIGM	0x100	RO	0x0000011F	Supported Trigger Modes Register
TPIU_TRIGCNT	0x104	RW	0x00000000	Trigger Counter Value Register
TPIU_TRIGMUL	0x108	RW	0x00000000	Trigger Multiplier Register
TPIU_STSTPTRN	0x200	RO	0x0003000F	Supported Test Patterns/Modes Register
TPIU_CTSTPTRN	0x204	RW	0x00000000	Current Test Pattern/Modes Register
TPIU_TPRCNTR	0x208	RW	0x00000000	TPIU Test Pattern Repeat Counter Register
TPIU_FFSTS	0x300	RO	0x00000000	Formatter and Flush Status Register
TPIU_FFCTRL	0x304	RW	0x00000000	Formatter and Flush Control Register
TPIU_FSCNTR	0x308	RW	0x00000040	Formatter Synchronization Counter Register
TPIU_EXCTLIN	0x400	RO	0x00000000	TPIU EXCTL Port Register - In
TPIU_EXCTLOUT	0x404	RW	0x00000000	TPIU EXCTL Port Register - Out

Name	Offset	Type	Reset	Description
TPIU_ITTRFLINACK	0xEE4	WO	0x00000000	Integration Test Trigger In and Flush In Acknowledge Register
TPIU_ITTRFLIN	0xEE8	RO	0x00000000	Integration Test Trigger In and Flush In Register
TPIU_ITATBDATA0	0xEEC	RO	0x00000000	Integration Test ATB Data Register 0
TPIU_ITATBCTR2	0xEF0	WO	0x00000000	Integration Test ATB Control Register 2
TPIU_ITATBCTR1	0xEF4	RO	0x00000000	Integration Test ATB Control Register 1
TPIU_ITATBCTR0	0xEF8	RO	0x00000000	Integration Test ATB Control Register 0
TPIU_ITCTRL	0xF00	RW	0x00000000	Integration Mode Control Register
TPIU_CLAIMSET	0xFA0	RW	0x0000000F	Claim Tag Set Register
TPIU_CLAIMCLR	0xFA4	RW	0x00000000	Claim Tag Clear Register
TPIU_LAR	0xFB0	WO	0x00000000	Lock Access Register
TPIU_LSR	0xFB4	RO	0x00000003	Lock Status Register
TPIU_AUTHSTATUS	0xFB8	RO	0x00000000	Authentication Status Register
TPIU_DEVID	0xFC8	RO	0x000000A0	Device Configuration Register
TPIU_DEVTYPE	0xFCC	RO	0x00000011	Device Type Identifier Register
TPIU_PIDR4	0xFD0	RO	0x00000004	Peripheral ID4 Register
TPIU_PIDR5	0xFD4	RO	0x00000000	Peripheral ID5 Register
TPIU_PIDR6	0xFD8	RO	0x00000000	Peripheral ID6 Register
TPIU_PIDR7	0xFDC	RO	0x00000000	Peripheral ID7 Register
TPIU_PIDR0	0xFE0	RO	0x00000012	Peripheral ID0 Register
TPIU_PIDR1	0xFE4	RO	0x000000B9	Peripheral ID1 Register
TPIU_PIDR2	0xFE8	RO	0x0000004B	Peripheral ID2 Register
TPIU_PIDR3	0xFEC	RO	0x00000000	Peripheral ID3 Register
TPIU_CIDR0	0xFF0	RO	0x0000000D	Component ID0 Register
TPIU_CIDR1	0xFF4	RO	0x00000090	Component ID1 Register
TPIU_CIDR2	0xFF8	RO	0x00000005	Component ID2 Register
TPIU_CIDR3	0xFFC	RO	0x000000B1	Component ID3 Register

R5 Core Debug Register Summary

See here for more details on the below registers:

Mnemonic	Register number	Offset	Access	Description
DIDR	c0	0x000	R	CP14 c0, Debug ID Register
-	c1-c5	0x004-0x014	R	RAZ (Reads as zero)
WFAR	c6	0x18	RW	Watchpoint Fault Address Register
VCR	c7	0x01C	RW	Vector Catch Register
-	c8	0x020	R	RAZ (Reads as zero)
ECR	c9	0x024	RW	Not implemented. Reads as zero.
DSCCR	c10	0x028	RW	Debug State Cache Control Register
-	c11	0x02C	R	RAZ (Reads as zero)
-	c12-c31	0x030-0x07C	R	RAZ (Reads as zero)
DTRRX	c32	0x080	RW	Data Transfer Register
ITR	c33	0x084	W	Instruction Transfer Register
DSCR	c34	0x088	RW	CP14 c1, Debug Status and Control Register
DTRTX	c35	0x08C	RW	Data Transfer Register
DRCR	c36	0x090	W	Debug Run Control Register
-	c37-c63	0x094-0x0FC	R	RAZ (Reads as zero)
BVR	c64-c71	0x100-0x11C	RW	Breakpoint Value Registers
-	c72-c79	0x120-0x13C	R	RAZ (Reads as zero)
BCR	c80-c87	0x140-0x15C	RW	Breakpoint Control Registers
-	c88-c95	0x160-0x17C	R	RAZ (Reads as zero)
WVR	c96-c103	0x180-0x19C	RW	Watchpoint Value Registers
-	c104-c111	0x1A0-0x1BC	R	RAZ (Reads as zero)
WCR	c112-c119	0x1C0-0x1DC	RW	Watchpoint Control Registers
-	c120-c127	0x1E0-0x1FC	R	RAZ (Reads as zero)
-	c128-c191	0x200-0x2FC	R	RAZ (Reads as zero)
OSLAR	c192	0x300	R	Not implemented. Reads as zero.
OSLSR	c193	0x304	R	Operating System Lock Status Register
OSSRR	c194	0x308	R	Not implemented. Reads as zero.
-	c195	0x30C	R	RAZ (Reads as zero)
PRCR	c196	0x310	RW	Device Power-down and Reset Control Register

Mnemonic	Register number	Offset	Access	Description
PRSR	c197	0x314	R	Device Power-down and Reset Status Register
-	c198-c511	0x318-0x7FC	R	RAZ (Reads as zero)
-	c512-575	0x800-0x8FC	R	RAZ (Reads as zero)
-	c576-c831	0x900-0xCFC	R	RAZ (Reads as zero)
-	c832-c895	0xD00-0xDFC	R	Processor ID Registers
-	c896-c927	0xE00-0xE7C	R	RAZ (Reads as zero)
-	c928-c959	0xE80-0xEFC	-	Integration Test Registers
-	c960-c1023	0xF00-0xFFC	-	Management Registers

14.1.3.3.2 Boundary Scan

This device supports boundary scan using an IEEE 1149.1 compliant JTAG TAP. IEEE 1149.1 and 1149.6 Boundary Scan support are defined in device-specific BSDL files that can be found in the respective device's product folder on ti.com.

14.1.3.4 Reset Management

Reset isolation: critical configuration and trace datapaths and logic are not sensitive to warm reset.

Configuration independence: debug configuration occurs over a debug-only interconnect, separate from SoC traffic to ensure debug logic remains available even during deadlock scenarios.

Power-AP: a CoreSight™ compliant Access Port (AP) developed by TI that provides a standard interface for debug tooling to access status and control over power, reset, and clocking for the system. Power-AP can control the reset of the system through the following registers:

- SYSTEMRESET_SPREC: Asserts system reset. A pulse to level signal is needed
- WIRREG_SPREC: Extends the system reset in RCM
- NRESET_SPREC: Reads the status of system reset
- GLOBALRELEASEWIR_SPREC: Writing 1 releases the WIRREQ
- PWRAP_SPREC: Writing 1 to bit 0 initiates system reset. Bit 8 toggles from 0 to 1 to 0, and users need to wait for the sequence to finish

These registers are not memory mapped, so the registers can only be accessed by typing register name in expression window of DAP connection.

14.1.3.5 Debug Cross Triggering

This device supports an Arm® CoreSight™ compliant four-channel programmable on-chip cross triggering network.

Conceptually, each channel of cross triggering can be viewed as mapping of a user-defined set of events to a user-defined set of actions, where the occurrence of any event in the set-of-events results in the generation of the set-of-actions.

The cross triggering network of AM263Px is shown in Figure 14-3.

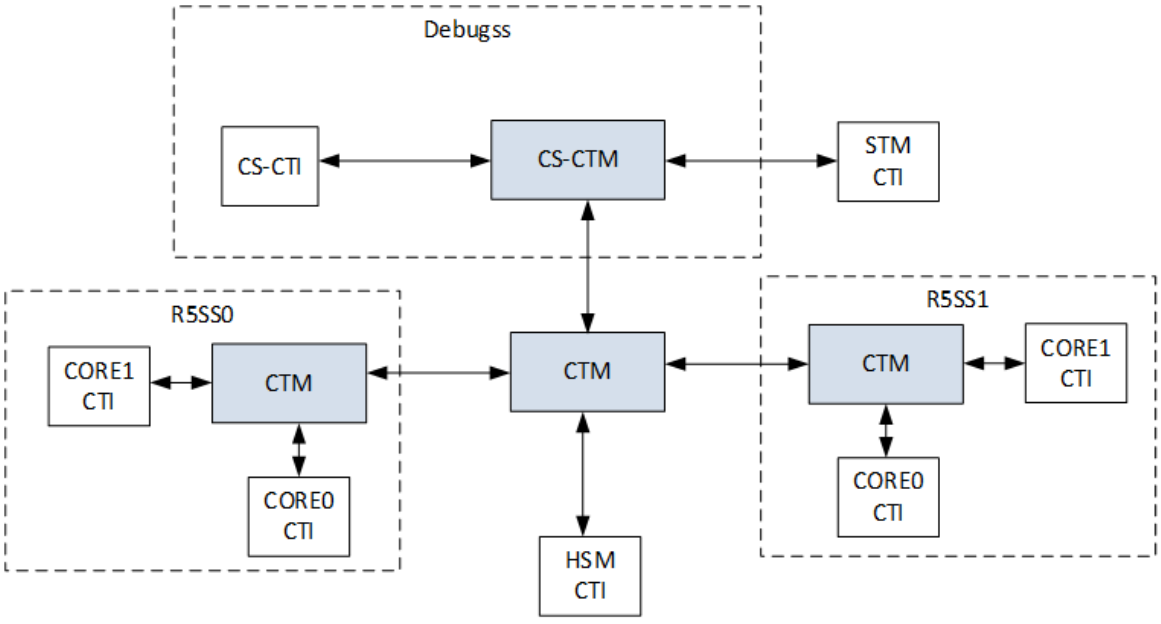


Figure 14-3. Cross Trigger Network

14.1.3.5.1 R5F CTI Trigger Connections

Table 14-4. R5F CTI Trigger Input Connections (One per R5F Core)

Trigger Input Bit	Source	Comments
[7]	Muxed VIM Interrupt sources	A mux selects which of the VIM interrupt event is routed as Trigger MSS_CTRL. R5SS*_CTI_TRIG_SEL.TRIG*[7:0] bits control the mux.
[6]	ETM: ETMTRIGGER	ETM managed Trigger. Generated internal to Cortex R5 Susbsystem
[5]	CORE:COMMTX	Communications channel transmit. Generated internal to Cortex R5 Susbsystem
[4]	CORE:COMMRX	Communications channel receive. Generated internal to Cortex R5 Susbsystem
[3]	ETM:ETMEXTOUT[1]	ETM managed External Output Event 1. Generated internal to Cortex R5 Susbsystem
[2]	ETM:ETMEXTOUT[0]	ETM managed External Output Event 0. Generated internal to Cortex R5 Susbsystem
[1]	CORE: PMUIRQ	Interrupt request from performance monitoring unit. Generated internal to Cortex R5 Subsystem
[0]	CORE: DBGTRIGGER	CPU is entering the debug state (halted). Generated internal to Cortex R5 Susbsystem

Table 14-5. R5F CTI Trigger Output Connections (One per R5F Core)

Trigger Output Bit	Destination	Comments
[7]	CORE:DBGRESTART	External restart request
[6]	Not Used	
[5]	Not Used	
[4]	Not Used	
[3]	VIM :CTI Interrupt	VIM Interrupt
[2]	ETM:EXTIN[1]	ETM External Input 1
[1]	ETM:EXTIN[0]	ETM External Input 0
[0]	CORE: EDBGRQ	External debug request

14.1.3.5.2 Cortex M4 CTI Trigger Connections

Table 14-6. M4 CTI Trigger Input Connections

Trigger Input Bit	Source Signal	Comments
[7]	Reserved	
[6]	DWT:ETMTRIGGER[2]	DWT generated Trigger 2
[5]	DWT:ETMTRIGGER[1]	DWT generated Trigger 1
[4]	DWT:ETMTRIGGER[0]	DWT generated Trigger 0
[3]	Reserved	
[2]	Reserved	
[1]	Reserved	
[0]	CORE:HALTED	CPU Has Halted

Table 14-7. M4 CTI Trigger Output Connections

Trigger Output Bit	Destination	Comments
[7]	CORE:DBGRESTART	External restart request

Table 14-7. M4 CTI Trigger Output Connections (continued)

Trigger Output Bit	Destination	Comments
[6]	Not Used	
[5]	Not Used	
[4]	Not Used	
[3]	NVIC:CTI_IRQ0	NVIC Interrupt. Refer to Processor Interrupt Map for more details
[2]	NVIC:CTI_IRQ1	NVIC Interrupt. Refer to Processor Interrupt Map for more details
[1]	Not Used	-
[0]	CORE:EDBGRQ	External Debug Request.

14.1.3.5.3 STM CTI Trigger Connections

Table 14-8. STM CTI Trigger Input Connections

Trigger Input Bit	Source Signal	Comments
[7]	Reserved	
[6]	Reserved	
[5]	Reserved	
[4]	Reserved	
[3]	Reserved	
[2]	STM:ASYNCOUT	Alignment synchronization output. The STM asserts this signal for one clock cycle when an ASYNC-VERSION-FREQ sequence is output on the ATB interface, and the **ASYNCOUT** signal can be used for cross-triggering.
[1]	STM:TRIGOUTSW	The STM asserts this signal for one clock cycle when a trigger event is generated on writes to a TRIG location in the extended stimulus port registers
[0]	STM:TRIGOUTSPTE	The STM asserts this signal for one clock cycle when a trigger event is detected on a match using the STMSPTER.

14.1.3.5.4 DEBUGSS CS-CTI Trigger Connections

Table 14-9. DEBUGSS CS-CTI Trigger Input Connections

Trigger Input Bit	Source	Comments
[7]	Muxed VIM3 Interrupt Inputs	Select any one of the 256 VIM3 interrupt. Configure by writing to MSS_CTRL.DBGSS_CTI_TRIG_SEL.TRIG3
[6]	Muxed VIM2 Interrupt Inputs	Select any one of the 256 VIM2 interrupt. Configure by writing to MSS_CTRL.DBGSS_CTI_TRIG_SEL.TRIG2
[5]	Muxed VIM1 Interrupt Inputs	Select any one of the 256 VIM1 interrupt. Configure by writing to MSS_CTRL.DBGSS_CTI_TRIG_SEL.TRIG1
[4]	Muxed VIM0 Interrupt Inputs	Select any one of the 256 VIM0 interrupt. Configure by writing to MSS_CTRL.DBGSS_CTI_TRIG_SEL.TRIG0
[3]	Reserved	Reserved
[2]	Reserved	Reserved
[1]	Reserved	Reserved
[0]	PWR-AP:SYNCRUNOUT	Debugss Power AP

Table 14-10. DEBUGSS CS-CTI Trigger Output Connections

Trigger Output Bit	Destination	Comments
[7]	Not Used	Not Used
[6]	Not Used	Not Used
[5]	Not Used	Not Used
[4]	CS-ETB: TRIGIN	Embedded Trace Buffer (ETB)

Table 14-10. DEBUGSS CS-CTI Trigger Output Connections (continued)

Trigger Output Bit	Destination	Comments
[3]	Not Used	Not Used
[2]	Not Used	Not Used
[1]	TPIU:FLUSHIN	DEBUGSS TPIU FLUSH
[0]	TPIU: TRIGIN	DEBUGSS TPIU TRIGGER

14.1.3.6 SOC Debug and Trace

This device includes debug capabilities deployed at the system level, including:

- Software messaging trace
- Debug-aware peripherals
- Global timestamping for trace

More details for each of these capability areas can be found in the corresponding sections below.

14.1.3.6.1 Software Messaging Trace

Software messaging trace is supported on this device by an MIPI STP-V2 compliant Arm® CoreSight™ STM with supporting logic that maps initiator IDs to specific STP Major Source IDs. The following device initiators support software messaging.

Table 14-11. STM Apperture Assignment

16 MB Address Apperture of STM	Master
0	R5SS0_CORE0_AXI_W
1	R5SS0_CORE1_AXI_W
2	R5SS1_CORE0_AXI_W
3	R5SS1_CORE1_AXI_W
4	HSM
5	ICSSM PRU0
6	ICSSM PRU1

14.1.3.6.2 Debug Aware Peripherals

Select peripherals support a debug feature that allows them to react to the debug state of a controlling processor. For instance, a timer peripheral that is allocated to a particular processor could be configured to stop counting when the associated processor is in the halted state. This device includes programmable support for shared peripherals that allows the developer to select the processor whose debug state a given peripheral should receive.

The Halt enable control register corresponding to the peripheral can be programmed to select which R5F CPU when halted will suspend the peripheral.

Table 14-12. Suspend Peripherals

Peripherals	Halt Enable Control Register
MCAN* [0-7]	MSS_CTRL: MCAN*_HALTEN
LIN* [0-4]	MSS_CTRL: LIN*_HALTEN
I2C* [0-3]	MSS_CTRL: I2C*_HALTEN
RTI* [0-7]	MSS_CTRL: RTI*_HALTEN
CPSW	MSS_CTRL: CPSW_HALTEN
MCRC0	MSS_CTRL: MCRC0_HALTEN
EPWM*[0-31]	CONTROLSS_CTRL: EPWM*_HALTEN
CMPSSA* [0-9]	CONTROLSS_CTRL: CMPSSA*_HALTEN
CMPSSB* [0-9]	CONTROLSS_CTRL: CMPSSB*_HALTEN
ECAP*[0-9]	CONTROLSS_CTRL: ECAP*_HALTEN
EQEP*[0-2]	CONTROLSS_CTRL: EQEP*_HALTEN

14.1.3.7 Trace Infrastructure

Trace traffic originates from a trace source, is distributed across the device using Arm® CoreSight™ compliant trace infrastructure components, and reaches one of two possible trace sinks. The trace infrastructure is shown in Figure 14-4.

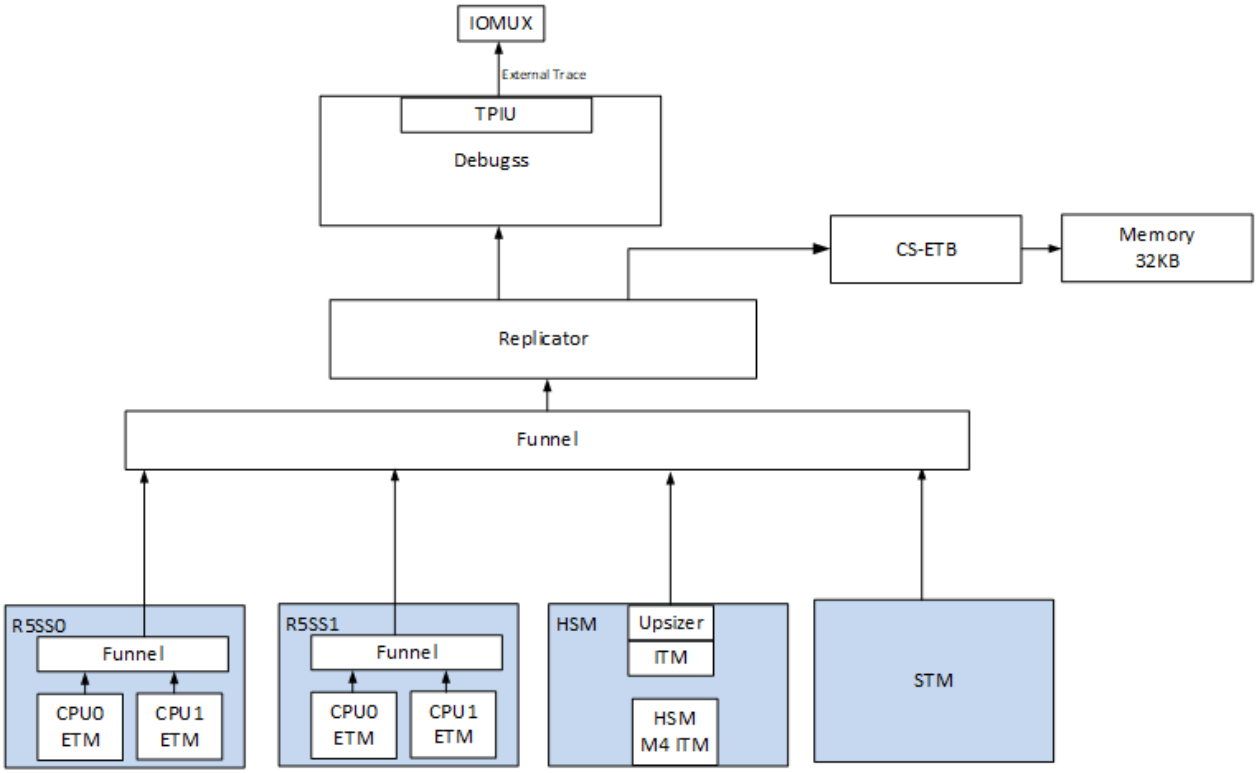


Figure 14-4. Trace Infrastructure

14.1.3.7.1 Trace Sources

The following trace sources are present in this device:

- STM
- HSM M4 ITM
- R5FSS0 Core0 ETM
- R5FSS0 Core1 ETM
- R5FSS1 Core 0 ETM
- R5FSS1 Core 1 ETM

14.1.3.7.2 Trace Distribution

Trace distribution is accomplished using standard Arm® CoreSight™ compliant trace infrastructure components:

CoreSight™ Trace Funnels (CSTF): Non-programmable CSTFs are used at points of interleaving where multiple trace sources converge and form a single stream of trace traffic. This device includes one instance of a CSTF that is deployed immediately before the TPIU.

CoreSight™ Trace Replicator (CSREP): A programmable CSREP is used as a routing device, that can be used to forward trace traffic, based on its ID, to one, both, or none of the device trace sinks. This device includes one instance of a CSTF that is deployed immediately before the TPIU to route the trace traffic to either CS-ETB or TPIU.

14.1.3.7.3 Trace Sinks

Two trace sinks are supported on this device:

Arm® CoreSight™ TPIU: TPIU supports export of trace off-chip via LVCMOS device pins (See 1.3.2.2) for capture by an external receiver.

CS-ETB Trace Buffer with 34KB of storage: CS-ETB can be setup to capture trace data until the internal buffer fills system bridge mode supports interrupt and event notification capabilities that support integration with device level CPUs and/or DMAs to support.

Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from Revision A (April 2024) to Revision B (May 2024)	Page
• Removed EARLY PRELIMINARY INFORMATION note	6

Changes from September 1, 2023 to April 1, 2024 (from Revision * (September 2023) to Revision A (April 2024))	Page
• [Overview] Removed mentions of TSN	8
• [Module Allocation and Instances] Update Resolver footnote to include ZCZ-F package	11
• Specified per CPU for Instruction and Data Cache	12
• Added that CTRLMMR is used to configure initialization values for the TCM	12
• Added definition for DRBG and noted acronyms for True / Pseudo Random Number Generation	14
• [Real-time Control Subsystem (CONTROLSS)] Elaborated that certain ADC / DAC features apply to all	15
• [Real-time Control Subsystem (CONTROLSS)] Added line about DAC VREF support	15
• [Real-time Control Subsystem (CONTROLSS)] Clarified how each instance of the CMPSS supports window comparison	15
• Added descriptions for TPCC and TPTC, and the feature list from the EDMA chapter	15
• Renamed CPSW3G to CPSW for consistency across document	17
• Updated feature list to align with the feature list from CPSW chapter	17
• Removed mention of two features that were incorrectly placed in this section	19
• Rewrite of System Interconnect Chapter	34
• Updating top-level system interconnect block diagram and overview description change	35
• Updated block diagram with minor corrections	37
• Updated rows to filter between AM263x/AM263Px	37
• Updated block diagram to show R5SS0_CONFIG_SLV and R5SS1_CONFIG_SLV	41
• Slight change in description	43
• Updated block diagram to remove STC and CCMR	44
• Slight update in description	44
• Initial content for AM263P	46
• Reorganized MPU Functional Description to cover the material in 3 sub sections instead of 8 sub sections	56
• Updated to use inclusive terminology	56
• Updated to use inclusive terminology	59
• Spelling fix	59
• Updated to use inclusive terminology	60
• Updated MPU Memory regions table with corrected end addresses	64
• Updated MPU Memory regions table default values for HSSE devices	64
• Removed bits 03:00	67
• RTI: Updated interrupt table to reflect that RTI interrupts are pulse type and not level type	139
• Added Content of Initialization Chapter	163
• New figure and explanation added on the overview of Initialization	164
• Few sentence formation changes to convey in a simpler way	165
• Added UART fallback support and mentioned DDR for 8D in OSPI Bootmodes	166
• Expanded meminit as memory initialization	166
• Image and description update, sub-topics deleted and new ones are added	168

- changed topic name....250
- Changed the description....250
- changed the Intro content for warm reset....250
- Re-worked section to better describe WARMRSTn HW Pin functionality....251
- Content has been expanded to better highlight all functionality....251
- Description modified....253
- Minor content edits....253
- Expanded content to better elaborate on all available reset functionality....253
- Minor text updates and included a link to the Power chapter....254
- Table added listing effect of each reset....254
- Changed starting sentence....254
- Added this section....255
- Added this section....255
- removed mention of Direct mode. Created event mappin table. Gave detailed explanation on PHASELOCK signal operation....258
- Changed SPI frequency from 40 to 50....258
- Changed SPI frequency from 50 to 48....258
- added new first paragraph that describes PLL's general usage and purpose. Reorganized section: moved formula to end of section and table to middle. Updated figure....260
- Section renamed to "PLL Module". First paragraph was redacted to better reflect PLL operation section was reorganized and removed irrelevatn information for simplicity....260
- Removed note at end fo the topic, since same infomrmation is shown in PLL Hookup section....260
- Changed instances of ADPLLLJ to PLL. Updated image. Added more detail to DIVx description. Removed mention of Bypass mode....260
- Removed paragraph describing that the CLKOUT1 and 4 can also be called M4-M7 outputs as tis information is not relevant for the customer. Clarified note and split info in two notes....260
- Added link to CTRL MMR Section. New sentence in initial paragraph....261
- Remove inline notes that mention that checking crytal present status is optional if ROM already checks for this. Deleted Step 6 relating to configuration of N2 divider since this parameter is not used in our design. New step 6 defines setting the SELFREQDCO value, previously undefined. Changed nomenclature of SEL_FREQ to SELFREQDCO across topic. Added TOPRCM to referenced registers....262
- Added TOP_RCM. prefix to all applicable referenced registers....263
- Added link to IP Clock Configurations section. Specified TOP_RCM. MMR region on MMR from item 3....263
- Added links to Root Clock and Core PLL configuration sections....263
- Added link to IP Clock Configurations section. Specified TOP_RCM. MMR region on all MMRs....264
- Specified TOP_RCM. prefix for registers....264
- Specified TOP_RCM. prefix for registers....264
- Previously only used as a root topic. added content....267
- Removed a row from R5SS_CORE_CLK:SYSCLK Achievable Ratio table that was not valid....267
- Added more information about sysclk and GCM on initial paragraphs....267
- created sentence at the end cross referencing the Clock Selection table....268
- Made instance generic ("x" numbering). Specified the MMR region MSS_RCM....283
- Made instance generic ("x" numbering). Specified the MMR region MSS_RCM....283
- updated name of CLKD to DCLK_DIV to align with name of the bitfield. Specified the MMR region MSS_RCM....283
- Made instance generic ("x" numbering). Specified the MMR region MSS_RCM....283
- renamed i2c high and low dividers to match MMR names. Made instance generic ("x" numbering). Specified the MMR region MSS_RCM....284
- Made instance generic ("x" numbering). Specified the MMR region MSS_RCM....285
- Made instance generic ("x" numbering). Specified the MMR region MSS_RCM....285
- Made instance generic ("x" numbering). Specified the MMR region MSS_RCM....286
- Made instance generic ("x" numbering). Specified the MMR region MSS_RCM. Updated Clock source selection value to 0x222....286
- modified MSS_RCM.MMCx_CLK_STATUS.CLKINUSE from 0x10 to 0x04 for FREQ=50MHz operation....286
- Specified the MMR region MSS_RCM....286

- Reformatting usage summary to align with new figure and removing information on unsupported modes... 547
- Reformatting usage summary to align with new figure and removing information on unsupported modes... 548
- Reformatting to reference CONTOLSS registers... 549
- Added a DAC Programming Guide to provide additional api and driver information... 549
- Added links and filtering for AM263Px... 549
- EPWM: Removed HRPWM Examples Using Optimized Assembly Code... 550
- Added OTTO-HRPWM to ePWM feature list ... 551
- Added list of Type 5 ePWM features in Introduction section of ePWM chapter... 551
- Change italics into working links... 554
- Fixed spacing issue in image that made signal a bit hard to read ... 554
- Updated Multiple ePWM Modules and Submodules and Signal Connections for an EWPM ... 554
- Added in details of Type 5 ePWM features including images, new XCMP sections, and Deadband info ... 554
- Added useful links about ePWM... 558
- Change ePWM integration image... 560
- Added ePWM Time-Base Submoldule image ... 562
- ePWM: Updated procedure for enabling ePWM clocks... 566
- Removed reference to PCLKCRx and align clk taxonomy... 566
- ePWM: Fixed italics to be links ... 567
- Removed duplicate ePWM SYNC Selection table... 569
- Aligned EPWM_CLKSYNC naming... 570
- Removed CAPENT and CAPIN images from *Edge Detection Within a Programmable TBTCR Range* section because images are repeated in chapter... 573
- Defined REGx in ePWM Global Load... 574
- Updated image of ePWM Counter-Compare Submodule... 576
- Added image of EPWM Counter-Compare Submodule... 576
- Updated image of EPWM Action-Qualifier (AQ) Submodule... 582
- Added a detail about shadow register in shadow mode ... 586
- ePWM: Removed reference of CAD in up-count mode... 588
- Updated image of EPWM Dead-Band Generator... 595
- Added image of EPWM Dead-Band Generator... 595
- Moved DBRED and DBFED info to *Simultaneous Writes to DBRED and DBFED Registers Between ePWM Modules (Type 5 EPWM)* ... 597
- Updated MINDB block diagram... 602
- Added subsections about EPWM MINDB... 602
- Updated image of EPWM PWM Chopper Submodule... 606
- Added image of EPWM PWM Chopper Submodule... 606
- Corrected Info on TZ4 in ePWM... 610
- Added image of EPWM Trip-Zone Submodule... 610
- Added image about Trip Zone TRIPOUT Selection... 611
- Update block diagram in ePWM Diode Emulation... 617
- Added section about EPWM Diode Emulation Submodule... 617
- Updated image of EPWM Event-Trigger Submodule... 623
- Added image of EPWM Event-Trigger Submodule... 623
- Added Event-Trigger Submodule Showing Event Inputs and Prescaled Outputs image... 624
- Removed mention of PIPE/PIE to interrupt controller for future devices and keeping it general... 624
- Added image of EPWM Digital Compare Submodule... 627
- Added Event Filtering section to ePWM... 636
- Updated *CAPIN and CAPGATE Source Selection*. Added threshold logic to *Counter Capture Logic* ... 639
- Combined *MIN and MAX Threshold Detection Logic*and *Counter Capture Logic* image into one image, so image removed from *MIN and MAX Detection Circuit* ... 640
- Reworte ePWM MIN-MAX Event Logic... 641
- EPWM: Removed self check diagnostics feature from HRPWM... 649
- EPWM: Updated inputs in HRPWM's Trip Zone diagram ... 651
- EPWM: Added HRCAL to HRPWM source clock paragraph... 654
- Added ePWM Source Clock... 654

- EPWM: Removed swapping feature from HRPWM ... 654
- Added paragraph about linking CMPBHR to CMPAHR ... 654
- EPWM: Replaced software information with link to EPWM Programming Guide ... 657
- EPWM: Aligned clock names to EPWM_SYNC ... 664
- EPWM: Replace reference to a specific file release to the EPWM Programming Guide ... 666
- Adjusted the simplified module image to reflect how SyncIn and SyncOut are internally routed ... 674
- Adjusted the sync images to reflect how SyncIn and SyncOut are internally routed ... 675
- Adjusted the block diagram image to reflect how SyncIn and SyncOut are internally routed ... 676
- Adjusted the block diagram image to reflect how SyncIn and SyncOut are internally routed ... 678
- Adjusted the block diagram image to reflect how SyncIn and SyncOut are internally routed ... 680
- Adjusted the block diagram image to reflect how SyncIn and SyncOut are internally routed ... 682
- Adjusted the block diagram image to reflect how SyncIn and SyncOut are internally routed ... 684
- Adjusted the block diagram image to reflect how SyncIn and SyncOut are internally routed ... 685
- Adjusted the block diagram image to reflect how SyncIn and SyncOut are internally routed ... 688
- removed 'syncout' and 'syncin' text from block diagram image ... 690
- removed 'syncout' and 'syncin' text from 'control of two resonant converter stages' fig ... 691
- Added EPWM Programming Guide ... 692
- Removed section Capture and APWM Operating Mode, moving the information into relevant sections ... 693
- Moved from Configuring Device Pins for the eCAP ... 695
- Updated the block diagram image and added a footnote ... 699
- Added Input Capture Signal Selection section ... 701
- Added Modulo 4 Counter section ... 701
- Added details originally in a different section to streamline information ... 702
- Added ... 702
- Added image for Active Low Mode ... 702
- Revised and re-named image for Active High Mode ... 702
- Added Error Events section ... 713
- Added Disabling the Signal Monitoring Unit section ... 713
- Added Shadow Control section ... 714
- Added Trip Signal section ... 714
- Added eCAP Programming Guide section ... 719
- Incomplete line removed ... 720
- Removed reference to GPxQSELn and GPyPUD and made them IOMUX ... 723
- Corrected reference to GPIO chapter ... 723
- EQEP Integration Diagram figure updates ... 724
- Updated Functional Block Diagram of the eQEP Peripheral to improve picture quality ... 728
- Added eQEP Programming Guide to show API and driver information ... 746
- FSI: Updated FSI Block Diagram ... 748
- FSI: Added details on clock gating and software reset ... 750
- FSI: Clarified instructions on configuring GPIO for FSI ... 752
- FSI: Renamed RX_INT1_CTRL and RX_INT2_CTRL registers as RX_INT1_CTRL_ALT1_ and RX_INT2_CTRL_ALT1_ ... 753
- FSI: Rename RX_EVT_STS and RX_EVT_CLR registers as RX_EVT_STS_ALT1_ and RX_EVT_CLR_ALT1_ ... 754
- FSI: Renamed TX_OPER_CTRL_LO register as TX_OPER_CTRL_LO_ALT2_ ... 754
- FSI: Renamed TX_PING_CTRL, TX_OPER_CTRL_HI and TX_OPER_CTRL_LO registers as TX_PING_CTRL_ALT1_, TX_OPER_CTRL_HI_ALT1_ and TX_OPER_CTRL_LO_ALT2_ ... 754
- FSI: Renamed TX_OPER_CTRL_LO register as TX_OPER_CTRL_LO_ALT2_ ... 758
- FSI: TX_OPER_CTRL_LO register as TX_OPER_CTRL_LO_ALT2_ ... 759
- FSI: Renamed TX_OPER_CTRL_LO register as TX_OPER_CTRL_LO_ALT2_ ... 760
- FSI: Renamed TX_PING_CTRL register as TX_PING_CTRL_ALT1_ ... 760
- FSI: Renamed TX_PING_CTRL register as TX_PING_CTRL_ALT1_ ... 761
- FSI: Renamed TX_OPER_CTRL_LO register as TX_OPER_CTRL_LO_ALT2_ ... 761
- FSI: Renamed TX_OPER_CTRL_LO register as TX_OPER_CTRL_LO_ALT2_ ... 763

- Added detail to main spinlock operations....838
- Removed unnecessary text in diagram....838
- Corrected the register name to MSS_VIM_PRIFIQ instead of MSS_VIM_FIQVEC....847
- Added SOC_TIMESYNC_XBAR1 into the table and modified the superscript 1 explanation, removed "input" from "input interrupt type"....849
- changed SOC_MAILBOX source module to MSS_CTRL....850
- Formatted the table PRU-ICSS-XBAR-INTRTR0 in-intr Hardware Requests to appear correctly....850
- Diagram change for bus representation Eg, x4 EDMA Trigger [7:4] to x4 EDMA Trigger and EDMA-XBAR-INTRTR0 Output Hardware Requests table formatted to appear correctly....853
- GPIO-XBAR-INTRTR0 in-intr Hardware Requests table reformatted to display correctly....862
- Modifed OSPI to QSPI....872
- Modifed OSPI to QSPI....878
- Modifed OSPI to QSPI....884
- Modifed OSPI to QSPI....890
- [EDMA - Third Party Transfer Controller] updated interconnect naming to L3_VBUSM. Fixed typos (distant->destination register). Updated read/write data bus to 64 bits in TPTC Block Diagram and in note below diagram....912
- [Types of EDMA Controller Transfers] changed 3rd dimension count definition space naming from register to PaRAM memory....914
- [Parameter RAM (PaRAM)] fixed typos and added note that channel remap is done in boot flow....916
- [Channel Options Parameter] added references to RAs for AM26x devices....919
- [Channel Source Address (SRC)] updated addressing mode to FIFO for SAM....919
- [Channel Destination Address (DST)] updated addressing mode to FIFO for DAM....919
- [Count for 1st Dimension (ACNT)] updated ACNT valid values to range of 1 to 65535/....919
- [Active Memory Protection] removed Example Access Denied register table, Example Access Allowed table since they are also in the RA....946
- [Block Move Example] changed 'greater than 64K bytes' to 'greater than or equal to'....957
- [Setting Up an EDMA Transfer] edited note under step 2 to reference step 1-d-ii instead of 1-b-ii....962
- [EDMA Debug Checklist] filtered McASP example in Table 11-23 for AM273x only....964
- Updated to use inclusive terminology....966
- Added note aboutr available GPIO on AM263....980
- Corrected the number of GPIO modules to 4 for AM263x....981
- Updated Integration diageram to reflect proper enumeration of [143:0]....982
- Added GPIO XBAR comment for CPU interrupt routing on AM263x....986
- Added comment about AM263x GPIO requiring GPIO XBAR for DMA events....987
- [I2C] Formatting and grammar fixes....996
- [I2C] Adjusted module counts....999
- [I2C Block Diagram] Added list of blocks and description of primary blocks....1008
- I2C Clocking: Added table for clocking calculations....1009
- I2C Clocking: Updated register names and equations to reflect correct register naming....1009
- I2C Interrupt Requests: Updated flag and register names....1010
- [I2C] Added general statement about the I2Cn register numbering scheme....1011
- [I2C] Revised section to reflect correct register names and steps....1011
- [I2C] Revised section to reflect correct register names and steps....1011
- [I2C] New section added....1011
- [I2C] Revised section to reflect correct register names and steps....1011
- [I2C] Revised section to reflect correct register names and steps....1011
- [I2C] Revised section to reflect correct register names and steps....1011
- [I2C] Revised section to reflect correct register names and steps....1012
- [I2C] Revised interrupt sequence to reflect ICIVR register bits....1012
- Gigabit Ethernet Switch (CPSW): Changed all references from "CPSW3G" and "CPSW_3G" to "CPSW". 1133
- Gigabit Ethernet Switch (CPSW): Updated Host port information across CPSW Chapter....1133
- Added Programming Guide for MCAN....1399
- DMA-based Transfer/Receive details added....1400
- Superfractional divider details added....1400

- LIN 2.0 and LIN2.1 included in supported specifications list.... 1400
- Added second paragraph and formula in Section 13.4.2.5.1.5.2 1432
- Added Programming Guide for LIN.... 1450
- Added links and filtering for AM263Px.... 1450
- RTI: Updated interrupt table to reflect that RTI interrupts are pulse type and not level type.... 1456
- (RTI Digital Windowed Watchdog): Fixed error in RTI Digital Windowed Watchdog Operation Block Diagram.... 1469
- (RTI Digital Watchdog): Added note that this feature is only available for the WWDT defined modules.... 1472
- Added Programming Guide for RTI/WWDT.... 1476
- Removed sections DCC Suspend mode behaviour and low power mode which are not applicable. DCC Control and count hand off across domains is explained in the DCC Counter operation section and hence removed.... 1480
- Removed reference to app note "Continuous monitor of the PLL frequency with the DCC" and added reference to DCC Computation tool for AM263" instead.... 1480
- Renamed input0_clk as Clock0 and input1_clk as Clock1 for consistency.... 1480
- Added reference to DCC application note.... 1483
- DCC clock sources re-ordered.... 1483
- DCC clocking table layout updated.... 1483
- Added register name for count on error.... 1486
- Added DCC error count register name.... 1486
- Fixed typo and removed note on SYSCLK monitoring.... 1486
- Moved note on debug mode behavior of FIFO here.... 1486
- Added DCCGCTRL2 register name.... 1486
- Updated the ESM inttegaration diagram and updated the register names.... 1498
- Updated the Chapter organization.... 1499
- Updated the ESM Block Diagram.... 1500
- Added Error Event Inputs chapter.... 1501
- Added new chapter-Error Interrupt Outputs.... 1501
- Updated the whole data and register names.... 1501
- Added a new chapter- Error pin behaviour during Reset.... 1504
- Register name changed.... 1505
- Added the programming guide section.... 1506
- Updated ESM Configuration Error Interrupt section.... 1506
- Updated the procedure flow and register names.... 1506
- Added few steps to the process.... 1507
- Added few steps to the process.... 1508
- STC General Description: Removed unsupported features (Interval Testing).... 1525

Trademarks

TI E2E™ is a trademark of Texas Instruments.
PROFINET™ is a trademark of PROFIBUS Nutzerorganisation e.V.
EtherNet/IP™ is a trademark of ODVA, Inc.
PROFIBUS® is a registered trademark of PROFIBUS Nutzerorganisation e.V.
EtherCAT® is a registered trademark of Beckhoff Automation GmbH.
ARM® and Cortex® are registered trademarks of ARM Limited.
Arm® is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.
All trademarks are the property of their respective owners.

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