TMS320TCI6482 Digital Signal Processor Silicon Revisions 3.1, 2.1, 2.0, 1.1

Silicon Errata

TEXAS INSTRUMENTS

Literature Number: SPRZ235R

October 2005–Revised January 2012

Contents

List of Figures

List of Tables

Texas Instruments

Silicon Errata

SPRZ235R – October 2005 – Revised January 2012

TMS320TCI6482 Digital Signal Processor Silicon Revisions 3.1, 2.1, 2.0, 1.1

1 Introduction

This document describes the known exceptions to the functional specifications for the TMS320TCI6482 digital signal processor; see the *TMS320TCI6482 Communications Infrastructure Digital Signal Processor* data manual (literature number SPRS246).

The advisory numbers in the document are not sequential. Some advisory numbers have been moved to the next revision. When items are moved, the remaining advisory numbers are not resequenced.

This document also contains "Usage Notes." Usage Notes highlight and describe particular situations where the device's behavior may not match presumed or documented behavior. This may include behaviors that affect device performance or functional correctness. These notes will be incorporated into future documentation updates for the device (such as the device-specific data manual), and the behaviors they describe will not be altered in future silicon revisions.

1.1 Device and Development-Support Tool Nomenclature

To designate the stages in the product development cycle, TI assigns prefixes to the part numbers of all DSP devices and support tools. Each DSP commercial family member has one of three prefixes: TMX, TMP, or TMS (e.g., TMS320TCI6482CTZ). Texas Instruments recommends two of three possible prefix designators for its support tools: TMDX and TMDS. These prefixes represent evolutionary stages of product development from engineering prototypes (TMX/TMDX) through fully qualified production devices/tools (TMS/TMDS).

Device development evolutionary flow:

TMX	Experimental device that is not necessarily representative of the final device's electrical specifications
TMP	Final silicon die that conforms to the device's electrical specifications but has not completed quality and reliability verification
TMS	Fully qualified production device

Support tool development evolutionary flow:

TMDX	Development-support product that has not yet completed Texas Instruments internal qualification testing
TMDS	Fully qualified development-support product

TMX and TMP devices and TMDX development-support tools are shipped against the following disclaimer:

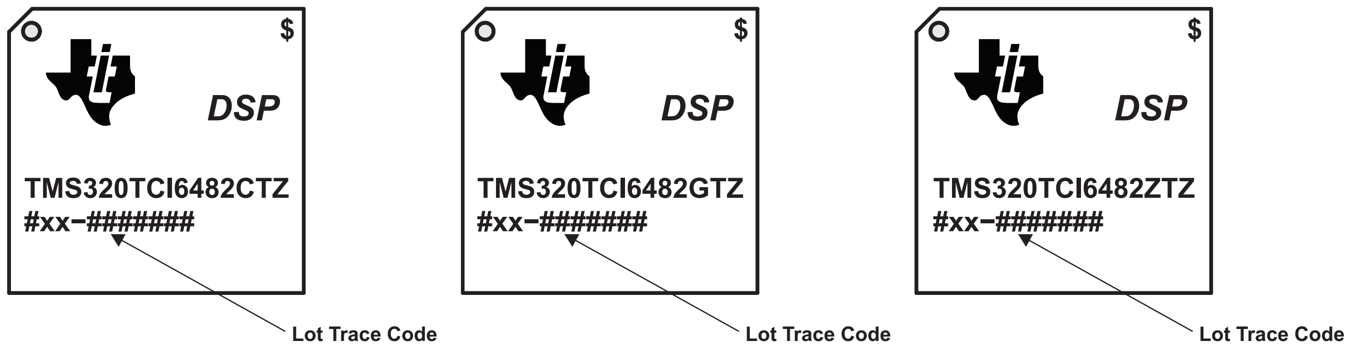
"Developmental product is intended for internal evaluation purposes."

TMS devices and TMDS development-support tools have been characterized fully, and the quality and reliability of the device have been demonstrated fully. TI's standard warranty applies.

Predictions show that prototype devices (TMX or TMP) have a greater failure rate than the standard production devices. Texas Instruments recommends that these devices not be used in any production system because their expected end-use failure rate still is undefined. Only qualified production devices are to be used.

1.2 Package Symbolization and Revision Identification

Figure 1 shows an example of TCI6482 package symbolization. The device revision can be determined by the lot trace code marked on the top of the package. The location of the lot trace code for the CTZ, GTZ, and ZTZ packages is shown in Figure 1.



A Qualified devices are marked with the letters "TMS" at the beginning of the device name, while nonqualified devices are marked with the letters "TMX" or "TMP" at the beginning of the device name.

B "#" denotes an alphanumeric character. "x" denotes a numeric character only.

C "$" denotes the speed and temperature grade of the device. For more information, see Table 2. **Note that this symbol is included on all TMS devices. Some TMX devices may not have this symbolization.**

Figure 1. Lot Trace Code Examples for TMS320TCI6482 (CTZ/GTZ/ZTZ Packages)

Silicon revision correlates to the lot trace code marked on the package. This code is of the format #xx-#######. If xx is "11", then the silicon is revision 1.1; if xx is "20", then the silicon is revision 2.0, etc. Table 1 lists the silicon revisions associated with each lot trace code for the TCI6482 devices. Each package also contains symbolization at the top right corner that denotes the speed and temperature grade of the device, see Figure 1 and Table 2. **Note that this symbol is included on all TMS devices. Some TMX devices may not have this symbolization.**

Each silicon revision uses a specific revision of the CPU and the C64x+ Megamodule. The CPU revision ID identifies the silicon revision of the CPU. Table 3 lists the CPU and C64x+ Megamodule revision associated with each silicon revision. The CPU revision can be read from the REVISION_ID field of the CPU Control Status Register (CSR). The C64x+ Megamodule revision can be read from the REVISION field of the Megamodule Revision ID register (MM_REVID) located at address 0181 2000h.

The VARIANT field of the JTAG ID Register (located at 02A8 008h) changes between silicon revisions. Table 2 lists the contents of the JTAG ID Register for each revision of the device. More details on the JTAG ID Register can be found in the *TMS320TCI6482 Communications Infrastructure Digital Signal Processor* data manual (literature number SPRS246).

Table 1. Lot Trace Codes

LOT TRACE CODE (xx)	SILICON REVISION	COMMENTS
31	3.1	Silicon revision 3.1
21	2.1	Silicon revision 2.1
20	2.0	Silicon revision 2.0
11	1.1	Initial silicon revision

Table 2. Speed and Temperature Grade Symbolization

SPEED/TEMPERATURE GRADE SYMBOLIZATION	SPEED GRADE	TEMPERATURE GRADE
1GHZ	1 GHz	Commercial
A1GHZ	1 GHz	Extended
<blank>	850 MHz	Commercial

Table 3. Silicon Revision Variables

SILICON REVISION	CPU REVISION	C64X+ MEGAMODULE REVISION	JTAG ID REGISTER VALUE
3.1	1.0 (REVISION_ID = 0h)	Rev.1 (MM_REVID[REVISION] = 5h)	0x4008 A02Fh (VARIANT = 0100)
2.1	1.0 (REVISION_ID = 0h)	Rev. 1 (MM_REVID[REVISION] = 1h)	0x2008 A02Fh (VARIANT = 0010b)
2.0	1.0 (REVISION_ID = 0h)	Rev. 1 (MM_REVID[REVISION] = 1h)	0x1008 A02Fh (VARIANT = 0001b)
1.1	1.0 (REVISION_ID = 0h)	Rev. 0 (MM_REVID[REVISION] = 0h)	0x0008 A02Fh (VARIANT = 0000b)

2 Silicon Revision 3.1 Usage Notes and Known Design Exceptions to Functional Specifications

This section describes the usage notes and advisories that apply to revision 3.1 of the TCI6482 device.

2.1 Usage Notes for Silicon Revision 3.1

Usage Notes highlight and describe particular situations where the device's behavior may not match presumed or documented behavior. This may include behaviors that affect device performance or functional correctness. These notes will be incorporated into future documentation updates for the device (such as the device-specific data manual), and the behaviors they describe will not be altered in future silicon revisions.

2.1.1 DDR2 Memory Controller: Chip Enable Pin Remains Low, Always Active

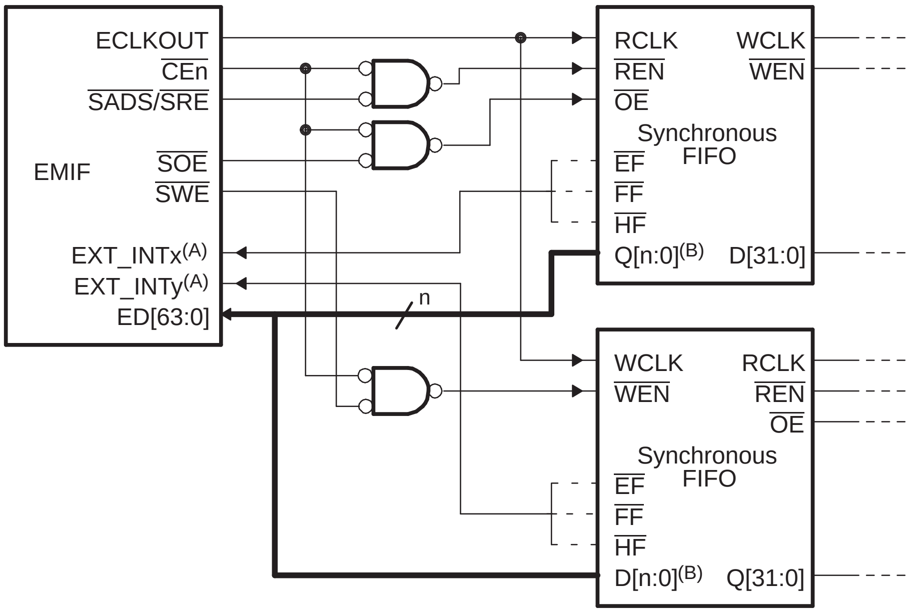
The chip enable pin of the DDR2 memory controller, $\overline{\text{DCE0}}$, is used to enable the DDR2 SDRAM memory device during external memory accesses. Currently, TI documentation shows that the $\overline{\text{DCE0}}$ pin is goes low to enable the DDR2 SDRAM memory device during external memory accesses and then goes high at the end of the access. However, on the TMS320TCI6482 device, the $\overline{\text{DCE0}}$ pin stays low throughout the operation of the DDR2 memory controller; it never goes high. Note that this behavior does not affect the ability of the DDR2 memory controller to access DDR2 SDRAM memory devices.

2.1.2 PLL: Hosts Should Not Access the DSP While PLL Registers are Being Configured

The PLL1 controller and PLL2 controller registers can only be programmed through the CPU and the emulator. To configure the PLL controllers, hosts like the HPI and PCI would have to load a small program that does this. However, hosts should hold off accesses to the DSP while the PLL controllers are being configured. Therefore, a mechanism must be in place such that the DSP can let the host know when the PLL configuration has been completed.

2.1.3 EMIFA: Chip Enable Pin Must Be Used to Interface With Devices Connected to EMIFA

The state of the EMIFA control pins is not defined while the chip enable pins ($\overline{\text{CE[5:2]}}$) are high. Furthermore, some control pins may become active while the chip enable pins are driven high. To avoid erroneously activating devices connected to EMIFA, the chip enable pins should be used to select/deselect these devices. If a device being used does not have a chip enable input, then the control pins going to that device should be qualified with a chip enable pin. An example of this is shown in Figure 2.



A GPIO pins on TCI648x devices may be configured as external interrupt sources to the CPU. For more details, see the *TMS320TCI648x DSP General-Purpose Input/Output (GPIO) User's Guide* (literature number SPRU725).

Figure 2. Read and Write Synchronous FIFO Interface With Glue Block Diagram

2.1.4 EMIFA: EDMA FIFO Addressing Mode Should Not Be Used When Reading from EMIFA

As documented in the *TMS320TCI648x DSP Enhanced DMA (EDMA3) Controller User's Guide* (literature number SPRU727), the EDMA includes two types of addressing modes: increment mode and FIFO mode (constant addressing mode). One of these modes must be selected for the source address mode and the destination address mode (SAM and DAM in the channel options parameter (OPT)).

Even though the EDMA supports FIFO mode configurations for SAM and DAM, the EMIFA does not fully support constant addressing mode. Attempts to perform writes to the EMIFA with FIFO mode (constant addressing mode) (DAM == FIFO) will result in the destination address incrementing within a Modulo-64-byte address range; i.e., the data will be written to address <Dst> to <Dst>+63 bytes, repeatedly, until the transfer count programmed for the EDMA transfer is exhausted. This behavior will not be modified on future versions of the TCI6482 device.

Attempts to perform reads from the EMIFA with FIFO mode (SAM == FIFO) on TCI6482 revision 1.1 will result in reading invalid data. An enhancement will be added to TCI6482 revision 2.0 such that reads from the EMIFA in constant addressing mode (SAM == FIFO) will behave much like writes; i.e., reads will be issued to address <SRC> to <SRC> + 63 bytes, repeatedly, until the transfer count programmed for the EDMA transfer is exhausted.

In order to avoid the issues described above with reads/writes to EMIFA when EDMA is configured for FIFO mode, irrespective of device revision, it is suggested that the EDMA always be programmed with SAM and DAM in increment mode. The ACNT and BCNT values, along with proper indexing, can be used to mimic FIFO addressing mode. This is addressed in the first and second recommendations/use cases below. For legacy reasons, it may be necessary to use FIFO mode. This is addressed in the third recommended use case below.

No special requirements exist for addresses accessed by the EDMA in increment mode or for single-word accesses by the CPU or DMA.

2.1.4.1 Recommended Implementations for Read/Writes to FIFO Connected to EMIFA

NOTE: In the case of ACNT < 64 bytes even though FIFO mode (fixed-mode addressing) is selected by EDMA configuration, the EMIFA will execute the transfer as if it were an increment address transfer. Therefore, the recommended implementation for read/write to FIFO should be increment mode instead of FIFO mode. The examples below provide suggestions for read/writes to FIFO using increment mode.

The following scenarios highlight various options for interfacing with a FIFO connected to the EMIFA and programming the EDMA to access the FIFO.

In general, the use cases are numbered in order of priority. The system designer should attempt to design a system/board/ASIC memory map such that a relatively large memory range is devoted to a FIFO. In this way, the EDMA can be programmed in increment mode for a given DMA transfer, and the transfer from start to finish will reside in the memory range dedicated to a given FIFO.

Approach 1: Address space dedicated to the FIFO is greater than or equal to the largest expected EDMA transfer. No performance hit since EDMA ACNT is not artificially constrained.

- FIFO should be aligned on a 64-byte boundary in EMIFA address space.

  System memory map and glue logic should be implemented to use EMIFA MS-address bits to decode FIFO address and select FIFO.
- EDMA transfer starting address should match the FIFO's base address.
- Use INCR transfer SRC/DST address mode with ACNT = transfer size.
- Use SBIDX or DBIDX of 0 such that the next EDMA transfer will also begin at the base address of the FIFO (assuming BCNT and/or CCNT are greater than 1).

For example, if the largest possible DMA transfer to/from a FIFO interfaced to EMIFA is 1024 bytes, then a memory range of at least 1024 bytes should be devoted to the FIFO in the EMIFA's memory map. The system glue logic should use the chip enable signals and logical address bits 10 and above if multiple FIFOs reside in the chip enable space. The EDMA transfer can be set with ACNT = 1024 bytes, BCNT = X, CCNT = Y, and an EDMA synchronization type of A-synchronized, The index for the FIFO side of the transfer (either SRC or DST) should be set to 0 such that the same address is used for the next DMA trigger.

Approach 2: If the amount of space dedicated to the FIFO is less than the largest expected EDMA, then ACNT and BCNT value with appropriate indexing can be used to control access to the FIFO. This will result in a potential performance impact depending on the size of ACNT.

- FIFO should be aligned on 64-byte boundary in EMIFA address space.

  System memory map and glue logic should be implemented to use EMIFA MS-address bits to decode FIFO address and select FIFO.
- EDMA transfer starting address should match the FIFO's base address.
- The EDMA size must be broken into
  - ACNT × BCNT = transfer size.
  - ACNT must be less than or equal to the address space dedicated to the FIFO.
  - If the EDMA transfer size is not a multiple of ACNT, then two EDMA channels must be used. Completion of the first channel can chain to the second channel, where the second channel is used to transfer the remaining data.
- Use SBIDX or DBIDX of 0 such that the next EDMA transfer will also begin at the base address of the FIFO (assuming BCNT and/or CCNT are greater than 1).

For example, if the desired DMA transfer size to/from a FIFO interfaced to EMIFA is 1024 bytes, but the memory range dedicated to the FIFO is only 64 bytes, then the EDMA transfer must be broken into a 2-D transfer, with ACNT = 64 bytes, BCNT = 16, CCNT = X, and an EDMA synchronization type of AB-synchronized. The index for the FIFO side of the transfer (either SRC or DST) should be set to 0 such that the same address is used for the next DMA trigger.

With the same example, if the desired DMA transfer size is 1028-bytes, an additional channel with ACNT = 4 bytes must be used. Completion of the first channel needs to chain trigger the second channel.

Approach 3: For legacy purposes, FIFO mode can be used with the restrictions below. This is a compromise between Approach 1 and Approach 2. Approach 3 allows the programmer to use an unrestricted ACNT value while the hardware forces the addresses accessed to reside within a 64-byte boundary, whereas Approach 2 uses a software mechanism to restrict addresses to the desired address ranges. If a FIFO requires an address range smaller than 64 bytes, then FIFO mode cannot be used; Approach 2 must be used instead.

TCI6482 revision 1.1

Writes to EMIFA in FIFO mode are allowed but addresses will increment within a 64-byte boundary.

- FIFO should be aligned on 64-byte boundary in EMIFA address space.

  System memory map and glue logic should be implemented to use EMIFA MS-address bits to decode FIFO address and select FIFO.
- EDMA transfer starting address should match the FIFO's base address.
- Use FIFO transfer DST address mode with ACNT = transfer size.
- Use DBIDX of 0 such that the next EDMA transfer will also begin at the base address of the FIFO (assuming BCNT and/or CCNT are greater than 1).

For example, if a transfer is greater than or equal to 64 bytes, then a memory range of at least 64 bytes should be devoted to the FIFO in the EMIFA's memory map. The system glue logic should use the chip enable signal and logical address bits 6 and above for address decoding if multiple FIFOs or other memory types reside in the chip enable space. The EDMA transfer can be set with ACNT = X-bytes, BCNT = Y, CCNT = Z, and an EDMA synchronization type of A-synchronized. The DBIDX should be set to 0 such that the same address is used for the next DMA trigger.

If a transfer is X bytes, where X is less than 64 bytes, then a memory range of at least X bytes should be devoted to the FIFO in the EMIFA's memory map. The system glue logic should use the chip enable signal and the appropriate MS-bits of the address for address decoding if multiple FIFOs or other memory types reside in the chip enable space. The EDMA transfer can be set with ACNT = X-bytes, BCNT = Y, CCNT = Z, and an EDMA synchronization type of A-synchronized. The DBIDX should be set to 0 such that the same address is used for the next DMA trigger.

Reads from EMIFA in FIFO mode will return invalid data and therefore must not be used. Use increment mode for SAM instead of FIFO mode. See Approaches 1 and 2 above.

TCI6482 revision 2.0

Reads and writes from EMIFA in FIFO mode will both behave like writes in revision 1.1, as described above (increment within 64-byte boundary).

- FIFO should be aligned on 64-byte boundary in EMIFA address space.

  System memory map and glue logic should be implemented to use EMIFA MS-address bits to decode FIFO address and select FIFO.
- EDMA transfer starting address should match the FIFO's base address.
- Use FIFO transfer address mode for the FIFO side of the transfer (either SRC or DST) with ACNT = transfer size in bytes.
- Use SBIDX or DBIDX of 0 such that the next EDMA transfer will also begin at the base address of the FIFO (assuming BCNT and/or CCNT are greater than 1).

For example, if a transfer is greater than or equal to 64 bytes, then a memory range of at least 64 bytes should be devoted to the FIFO in the EMIFA's memory map. The system glue logic should use the chip enable signal and logical address bits 6 and above for address decoding if multiple FIFOs or other memory types reside in the chip enable space. The EDMA transfer can be set with ACNT = X-bytes, BCNT = Y, CCNT = Z, and an EDMA synchronization type of A-synchronized. The SBIDX or DBIDX should be set to 0 such that the same address is used for the next DMA trigger.

If a transfer is X bytes, where X is less than 64 bytes, then a memory range of at least X bytes should be devoted to the FIFO in the EMIFA's memory map. The system glue logic should use the chip enable signal and the appropriate MS-bits of the address for address decoding if multiple FIFOs or other memory types reside in the chip enable space. The EDMA transfer can be set with ACNT = X-bytes, BCNT = Y, CCNT = Z, and an EDMA synchronization type of A-synchronized. The SBIDX or DBIDX should be set to 0 such that the same address is used for the next DMA trigger.

2.1.5 HPI: Certain HPIC Register Bits Will Reset to Default Value Only With Power-On Reset

The following bits of the Host Port Interface Control register (HPIC) will only reset to their default values with a power-on reset ($\overline{\text{POR}}$ pin). Other resets, like warm reset ($\overline{\text{RESET}}$ pin) and emulation reset, will not affect these bits.

- HWOB (bit 0)
- HRDY (bit 3)
- HWOBSTAT (bit 8)
- DUALHPIA (bit 9)
- HPIARWSEL (bit 11)

2.1.6 DDR2 Memory Controller and EMIFA: PRIO_RAISE Bits Should Be Changed From Default Following Reset

The reordering and scheduling rules used by EMIFA and DDR2 Memory Controller may lead to command starvation, which is the prevention of certain commands from being processed. Command starvation can result when a continuous stream of high-priority read commands blocks a low-priority write command.

To avoid this condition, EMIFA and DDR2 Memory Controller momentarily raise the priority of the oldest command in the command FIFO after a set number of transfers have been made. The PRIO_RAISE field in the Burst Priority Register (BPRIO) sets the number of the transfers that must be made before the priority of the oldest command is raised.

By default, this feature of EMIFA and DDR2 Memory Controller is disabled. This means commands can stay in the command FIFO indefinitely. Therefore, to enable this feature with the highest level of allowable memory transfers, the PRIO_RAISE bits should be set to FEh immediately following reset. These bits can be left as FEh unless advanced bandwidth/prioritization control is required. It is suggested that prioritization be set at the system level to avoid placing high-bandwidth masters on the highest priority levels.

2.1.7 Device: Heatsink/Airflow Recommended to Lower Case Temperature

It is strongly recommended that users complete system-level thermal analysis to account for details of heatsink requirements, airflow, and other factors in order to achieve the case temperature specification of 90°C. The latest power data for the TMS320TCI6482 device indicates that static power is a significant contributor to overall power. Since static power varies with case temperature and voltage, a lower case temperature can greatly impact the overall power consumption. Therefore, the use of a heatsink to lower the case temperature is an effective way to lower power consumption. For further details on the power consumption of the TMS320TCI6482 device, see the *TMS320TCI6482 Power Consumption Summary* (literature number SPRAAF1) application report.

2.1.8 McBSP: Receiver and/or Transmitter Must Out of Reset to Enable Frame-Sync Detection

The McBSP transmitter and receiver on the TCI6482 device are capable of generating an interrupt upon the detection of frame synchronization. The *TMS320C6000 DSP Multichannel Buffered Serial Port (McBSP) Reference Guide* (literature number SPRU580) states that this feature will operate even while the associated portion of the serial port is in reset. However, on the TCI6482 device, the receiver and/or transmitter must be out of reset to enable this feature.

Note that frame synchronization can be detected while the receiver or transmitter are in reset by using the GPIO mode of the frame-sync pin (FSR or FSX). In this configuration, the CPU can monitor the status of the frame-sync pin and switch to the non-GPIO mode when a transition on the frame-sync pin is detected. For more information on the GPIO mode and frame sync detection feature of the McBSP, see the *TMS320C6000 DSP Multichannel Buffered Serial Port (McBSP) Reference Guide* (literature number SPRU580).

2.1.9 McBSP: Performance Degradation Can Be Seen When Using PCI, UTOPIA, or VLYNQ

The McBSP Data Receive Register (DRR) and Data Transmit Register (DXR) can be accessed through two separate busses: the configuration bus and the data bus. Both the CPU and the EDMA can access these busses and, in most cases, the highest performance method is achieved by using the data bus.

However, as shown in the device-specific data manual (see section 4, *System Interconnect*), the McBSP data bus shares a bridge to the data switched central resource with the PCI, UTOPIA, and VLYNQ peripherals. Performance degradations can be observed if any of these peripherals are used and the McBSP DRR and DXR are accessed through the data bus.

Therefore, when the PCI, UTOPIA, and VLYNQ peripherals are used, it is recommended the configuration bus is used to access the McBSP DRR and DXR.

Note that the PCI and VLYNQ peripherals consist of an independent master and slave. The above performance degradation is only an issue when the peripheral is used to initiate transactions on the external bus.

2.1.10 Boundary Scan: Warnings Relating to the RSV32 and RSV34 Pins May Be Observed When Using Boundary Scan

Previously, the device-specific data manual required that the RSV32 and RSV34 be tied to V_{SS} for proper device operation. This is an incorrect configuration for these pins. This configuration may case boundary-scan tools to generate warnings relating to these pins; these warnings are not critical and can be ignored.

The current device-specific data manual has been corrected such that the requirement is now to tie the RSV32 and RSV34 pins to the 1.8-V I/O supply (DV_{DD18}) via a 1-kΩ resistor. This configuration will prevent any boundary-scan warnings relating to these pins.

Existing TCI6482 designs need not be modified to meet the new requirement. The boundary-scan warnings relating to these two pins are not critical and can be ignored. TCI6482 devices will not be damaged by tying the RSV32 and RSV34 pins to V_{SS}. New TCI6482 designs must tie the RSV32 and RSV34 pins to the 1.8-V I/O supply via a pull-up resistor to avoid these boundary-scan warnings.

2.1.11 PCI: DSP PCI Cannot Burst More Than 64 Bytes When Used in Master Mode

The PCI on the TCI6482 can operate as a PCI master and slave. As a slave, the DSP PCI responds to accesses initiated by an off-chip PCI master. As a master, the DSP PCI, itself, initiates transfers on the PCI bus. Usually, for memory read and write transfers, another DSP master such as the EDMA is configured to move data to/from the DSP PCI.

As a PCI master, the DSP PCI is only capable of bursting a maximum of up to 64 bytes. In other words, for memory transfers larger than 64 bytes, the DSP PCI will initiate a transfer, transfer 64 bytes, stop the transfer, and then repeat. As a PCI slave, external PCI masters can burst an infinite amount of data to the DSP PCI. Note that the DSP PCI may insert wait states or generate a target retry if it cannot meet the latency requirement set forth by the PCI system. For example, a PCI access to DSP DDR2 memory may stall due to other master accesses or because of a scheduled DDR2 memory refresh. In this case the DSP PCI will generate a target retry until the DDR2 memory controller is ready to service the PCI request.

Because of this limitation, the DSP PCI throughput will be lower in master mode than in slave mode. To avoid low throughput performance, external PCI masters should be used to move data to/from the DSP PCI whenever possible.

2.1.12 DDR2 Memory Controller: Maximum Addressable Memory Increased to 512MB in 32-bit Mode

The maximum addressable memory has been increased from 256MB to 512MB in 32-bit mode and from 128MB to 256MB in 16-bit mode. Revision G and later of the *TMS320TCI6482 Communications Infrastructure Digital Signal Processor* data manual (literature number SPRS246) has been updated to show this change. Revision B and late of the *TMS320TCI648x DSP DDR2 Memory Controller User's Guide* (literature number SPRU894) has been updated to reflect this change.

2.1.13 EMAC: Gigabit Mode Cannot Be Used With CPU Running at Speeds Lower Than 750 MHz

The EMAC internal bus frequency must be greater than or equal to the I/O bus frequency. The EMAC internal bus is clocked by SYSCLK3 of the PLL1 controller, which has a frequency equal to the CPU frequency divided by 6. The I/O bus frequency of the EMAC is determined by the bit rate being used: 1.25 MHz for 10 Mbps, 12.5 MHz for 100 Mbps, and 125 MHz for 1000 Mbps. This restriction applies whether RGMII or GMII mode is being used.

Note that if the CPU speed is less than 750 MHz, the gigabit mode of the EMAC (1000 Mbps) cannot be used since the SYSCLK3 frequency will be less than 125 MHz.

2.1.14 DDR2 EMIF: Delay Before CKE Goes High With Different Combinations of REFRESH_RATE and DDR Clock

The SDRAM refresh control register (SDRFC) contains a count value that is used for two purposes. At power up, it is used to control the delay before CKE goes high. Later, it is used to control the time between refreshes. The DDR2 JEDEC specification requires a 200 μs delay before CKE goes high during initialization. The calculation of the delay before CKE goes high involves the following: The default value for REFRESH_RATE is 0x1388 at POR.

If the DDR clock period is set at 3.75 ns, the delay would be 16 * (0x1388) / 266.666 = 300 μs. Users have to make sure that any time when the DDR2 is enabled, the delay before CKE goes high with different combinations of REFRESH_RATE and DDR clock is always longer than 200 μs. Table 4 lists a few typical calculated values (200-μs delay).

Table 4. 200-μs Delay Calculated Values

CLOCK PERIOD	REFRESH_RATE
3.75 ns	0xD04
5 ns	0x9C4
8 ns	0x61A

During normal operation, the DDR memories require a refresh cycle at an average interval of 7.8125 μs (MAX). The calculation of RERESH_RATE involves the following:

REFRESH_RATE = DDR2CLKOUT frequency × memory refresh period

If the DDR clock period is set at 3.75 ns, the RERESH_RATE would be:

REFRESH_RATE = 266.666 MHz × 7.8125 μs = 2082 = 0x822.

Table 5 lists a few typical calculated values (an average interval of 7.8125 μs).

Table 5. 7.8125-μs Interval Calculated Values

CLOCK PERIOD	REFRESH_RATE
3.75 ns	0x822
5 ns	0x61A
8 ns	0x3D0

If the DDR2 needs to be put into self-refresh mode or power-down mode, users need to write a new value to the REFRESH_RATE field of the SDRFC register to guarantee the 200-μs delay of CKE during power-up or self-refresh mode exit.

2.1.15 Manual Cache Coherence Operation

When an L1DWB, L1DWBINV, L2DWB, or L2DWBINV command is executed, and the writeback is complete, the C64x+ Megamodule sends a single 128-bit message with the address of the last word for that block operation. On the TCI6482 device, TI did not hook up this signal and, therefore, this looks like any other write command.

Because CPU-to-CPU transfers are not allowed in the connectivity of the SCR, the address is treated as an invalid address and the command is immediately terminated at the null-endpoint within the SCR and goes nowhere. There should be no effect at all to the system by this behavior.

2.1.16 AEA3 Must be Tied High with a 1-kΩ Resisitor if Power is Applied to the SRIO Supply Pins

The AEA3 pin must be pulled up at device reset using a 1-kΩ resistor if power is applied to the SRIO supply pins. Failure to do so may occur in L2 memory errors and Die-ID not being read correclty. If the SRIO peripheral is not used and the SRIO supply pins are connected to VSS, the AEA3 pin must be pulled down to VSS using a 1-kΩ resistor.

2.2 Silicon Revision 3.1 Known Design Exceptions to Functional Specifications

Table 6. Silicon Revision 3.1 Advisory List

Title **Page**

Advisory 3.1.1 *VCP2: Specific Parameter Combinations Generate Incorrect Results*

Revision(s) Affected: 3.1 and earlier

Details: Using the following parameter combinations with VCP2 and a constraint length of K = 8 will generate incorrect results on the TCI6482 device.

TRACE-BACK MODE	K	C	R	F
Convergent	8	98	119	2557
Convergent	8	105	63	120

Note this advisory applies to convergent trace-back mode with K = 8 and these particular combinations of R, C and F. These may be used in some VoIP, HD radio, and radio network applications, but are not used in any 3-G wireless infrastructure standards.

Workaround(s): There is no workaround for this advisory.

Advisory 3.1.2 *VCP2 and TCP2: Emulator Access to TCP2 and VCP2 Registers Through EDMA Bus Return Incorrect Results*

Revision(s) Affected: 3.1 and earlier

Details: Some VCP2 and TCP2 registers are accessed either through the EDMA bus or through the configuration bus. Register access through the EDMA bus is only supported for 64-bit accesses.

The emulation software (Code Composer Studio) accesses memory mapped registers (MMRs) via 32-bit accesses. Therefore, reading VCP2 and TCP2 registers through the EDMA bus results in the *odd word* accesses being returned as 0s.

Workaround(s): There is no hardware workaround. In order to view these registers for debug purposes, a software workaround must be implemented.

One possible limited intrusive method would be to use an interrupt service routine (ISR) to access the registers and store them in a global structure. The ISR would not be executed during normal operations, but can be executed when desired through the use of a GEL script. Debugging would proceed as normal, except when it is desired to view the TCP2 and/or VCP2 registers.

A GEL script can be written to generate an interrupt and, hence, execute the ISR. A breakpoint should be placed at the end of the ISR. After the ISR has completed execution, the data can be viewed in the global structure.

The ISR needs to access the registers with 64-bit accesses. This can be achieved with LDDWs to access two registers at a time or via DMA accesses. IDMA channel 0 is a good option for this, as it is fast and easy to set up. The setup of the ISR and mapping of interrupts is left up to the developer. The code segments shown in Example 1 through Example 3 are provided as examples to show how this workaround can be implemented.

Example 1. Interrupt Service Routine Setup

```
interrupt TCP2_regDump_ISR(){
      /* Code to copy DMA Based MMRs to Global Struct */
}
```

Example 2. Interrupt Initialization

```
void DoInterruptsInitialization()
{
    CSL_IntcParam vectId1;
    CSL_IntcGlobalEnableState state;
/* Setup the global Interrupt */
    context.numEvtEntries = 1;
    context.eventhandlerRecord = Record;
    CSL_intcInit(&context);
   /* Enable NMIs  */
    CSL_intcGlobalNmiEnable();
   /* Enable Global Interrupts  */
    CSL_intcGlobalEnable(&state);
/* VectorID for the TCP2 Register Dump Event  */
    vectId1 = CSL_INTC_VECTID_4;
/* Opening a handle for the Global EDMA Event */
    hIntcTcp2        = CSL_intcOpen(    &intcTcp2,
                                    CSL_INTC_EVENTID_##, // Event Driven By GEL
      &vectId1,
                                    NULL);
//Hook the ISRs
    CSL_intcHookIsr(vectId1,& TCP2_regDump_ISR);
    // Clear the Interrupt
    CSL_intcHwControl(hIntcTcp2, CSL_INTC_CMD_EVTCLEAR, NULL);
//Enable the Event & the interrupt
    CSL_intcHwControl(hIntcTcp2, CSL_INTC_CMD_EVTENABLE, NULL);
    ipr[0] = 0xFFFFFFFF;
    ipr[1] = 0xFFFFFFFF;
}
```

Example 3. GEL Script

```
/************************************************/
/* GEL FILE                                                                            */
/************************************************/
#define EVENTSET0             0x01800020  // Event Set Register 0
#define EVENTSET1             0x01800024  // Event Set Register 1
#define EVENTSET2             0x01800028  // Event Set Register 2
#define EVENTSET3             0x0180002C  // Event Set Register 3
StartUp()
{
   /* Initialization portion of GEL File */
}
/*-------------------------------------------------------------*/
/* TCP2 VCP2 Register Access                                   */
/*-------------------------------------------------------------*/
menuitem "TCP2VCP2RegAccess";
hotmenu TCP2RegAccess()
{
*(int *)EVENTSET0 = 0xXXXXXXXX;  // Only need to set one of these
*(int *)EVENTSET1 = 0xXXXXXXXX;  // corresponding to the event
*(int *)EVENTSET2 = 0xXXXXXXXX;  // used to trigger the ISR
*(int *)EVENTSET3 = 0xXXXXXXXX;
         GEL_Go(TCP2_regDump_ISR); /* Runs until TCP2_regDump_ISR function is called */
}
hotmenu VCP2RegAccess()
{
*(int *)EVENTSET0 = 0xXXXXXXXX;  // Only need to set one of these
*(int *)EVENTSET1 = 0xXXXXXXXX;  // corresponding to the event
*(int *)EVENTSET2 = 0xXXXXXXXX;  // used to trigger the ISR
*(int *)EVENTSET3 = 0xXXXXXXXX;
         GEL_Go(VCP2_regDump_ISR); /* Runs until VCP2_regDump_ISR function is called */
}
```

Advisory 3.1.3 *EMAC: RMII Interface Cannot be Used in Half-Duplex Mode*

Revision(s) Affected: 3.1 and earlier

Details: The RMII Ethernet MAC interface on TCI6482 cannot be used during normal operation in half-duplex mode. Only full-duplex mode is supported.

It is anticipated that some RMII devices connecting to this device will initialize by default to half-duplex mode. This will still be allowed on TCI6482 given that the system negotiates to full-duplex mode immediately and a reset to the TCI6482 RMII is issued via the EMAC configuration register (EMACCFG). Note that this register and the ability to reset the RMII individually does not exist in silicon revision 1.1.

Workaround(s): No workaround exists for support of RMII in half-duplex mode.

If half-duplex mode must be used, MII should be considered for connection to 10/100 Mbps devices and RGMII should be considered for 1 Gbps devices.

Advisory 3.1.4 *EMAC: RMCRSDV Signal is Asserted from the PHY Asynchronously and Can Cause Undefined Behavior Internal to the RMII Module*

Revision(s) Affected: 3.1 and earlier

Details: The RMCRSDV input pin, used in the Ethernet MAC RMII interface, is susceptible to metastability due to the fact that it is not properly synchronized inside the device. Any activating transition on the RMCRSDV input may cause undefined behavior during EMAC RMII reception operation. This failure can result in lost frames.

Workaround(s): The best way to eliminate the issue is to create a synchronized gating signal externally (based on the RMII reference clock) then AND that signal with the RMCRSDV input. Care must be taken as RMCRSDV also toggles at the end of frames.

Advisory 3.1.5 *EMAC: Signal Transitions on RMRXER are Ignored for Least Significant Di-Bit*

Revision(s) Affected: 3.1 and earlier

Details: For RMII operation, if a pulse on RMRXER is driven during the least significant di-bit time, the pulse is ignored. This can result in corrupt receive frames that are not seen as erroneous data. This can cause errant frames to be seen as normal frames with corrupt data.

Workaround(s): This issue can be resolved in hardware by ORing a delayed version of RMRXER (previous di-bit) with the current RMRXER. Note that AC timing must still be met.

Advisory 3.1.6 *EMAC: RMCRSDV Not Being Passed Asynchronously to the EMAC*

Revision(s) Affected: 3.1 and earlier

Details: During RMII operation, RMCRSDV should be received asynchronously to the RMREFCLK and passed to the EMAC as CRS to minimize latency. However, this signal is being clocked internally which can cause a slight performance impact.

Workaround(s): There is currently no workaround for this issue.

Advisory 3.1.7 *EMU: Emulation Prone to Failure Under Certain Situations*

Revision(s) Affected: 3.1 and earlier

Details: Under certain conditions, the emulation hardware may corrupt the emulation control state machine or may cause it to lose synchronization with the emulator software. When emulation commands fail as a result of the problem, Code Composer Studio Integrated Development Environment (IDE) may be unable to start or it may report errors when interacting with the C64x+ DSP (for example, when halting the CPU, reaching a breakpoint, etc.).

This phenomenon is observed when an erroneous clock edge is generated from the TCK signal inside the C64x+ DSP. This can be caused by several factors, acting independently or cumulatively:

- TCK transition times (as measured between 2.5 V and 0.6 V) in excess of 3 ns.
- Operating the C64x DSP in a socket, which can aggravate noise or glitches on the TCK input.
- Simultaneous switching EMU pins during trace can affect the TCK signal.
- Poor signal integrity on the TCK line from reflections or other layout issues.

A TCK edge that can cause this problem might look similar to the one shown in Figure 3. A TCK edge that does not cause the problem will look similar to the one shown in Figure 4. The key difference between the two figures is that Figure 4 has a clean and sharp transition whereas Figure 3 has a "knee" in the transition zone. Problematic TCK signals may not have a knee that is as pronounced as the one in Figure 3. Due to the TCK signal amplification inside the chip, any perturbation of the signal can create erroneous clock edges.

As a result of the faster edge transition, there is increased ringing in Figure 4. As long as the ringing does not cross logic input thresholds (0.6 V for falling edges, and 2.5 V for rising edges), this ringing is acceptable.

When examining a TCK signal for this issue, either in board simulation or on an actual board, it is very important to probe the TCK line as close to the DSP input pin as possible. In simulation, it should not be difficult to probe right at the DSP input. For most physical boards, this means using the via for the TCK pad on the back side of the board. Similarly, ground for the probe should come from one of the nearby ground pad vias to minimize EMI noise picked up by the probe.

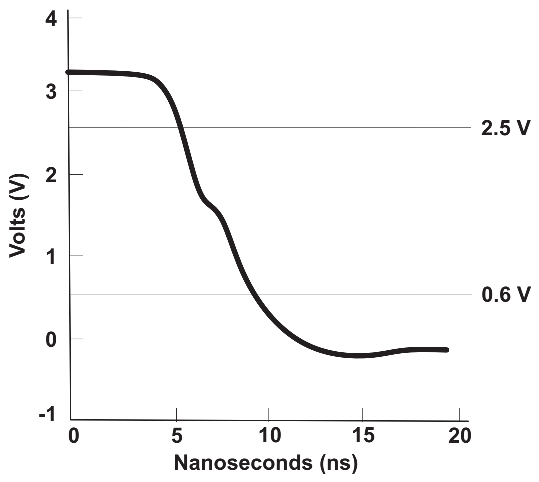


Figure 3. Bad TCK Transition

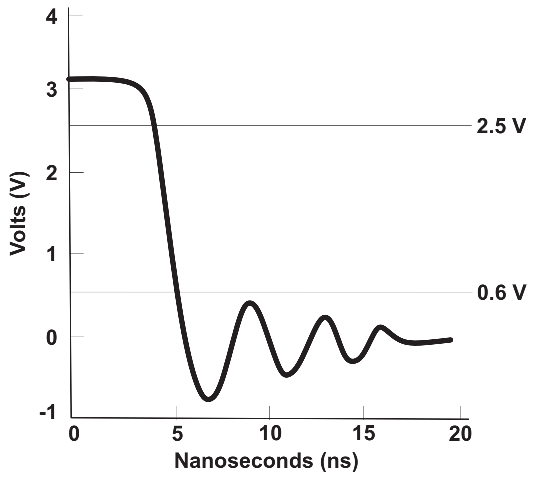


Figure 4. Good TCK Transition

Workaround(s): As the problem may be caused by one or more of the above factors, one or more of the steps outlined below may be necessary to fix it:

- Avoid using a socket.
- Ensure that the board design achieves rise times and fall times of less than 3 ns with clean, monotonic edges for the TCK signal.
- For designs where TCK is supplied by the emulation pod, use an external buffer equal, or similar to, TI's CDCV304 on the TCK signal.

Advisory 3.1.8 *McBSP: Emulation Access to McBSP Registers May Cause Sample Loss*

Revision(s) Affected: 3.1 and earlier

Details: The McBSP registers on the TCI6482 DSP can be accessed through two separate busses: a configuration bus and a data bus. McBSP registers are accessed through the data bus at byte address 0x3000 0000 and through the configuration bus at byte address 0x02A0 0000.

The McBSP allows emulation access only through the configuration bus since this does not affect the values of the RRDY and XRDY bits in the serial port control register (SPCR). The data bus is intended for functional accesses using the EDMA engine and not for emulation accesses. Furthermore, competing accesses through the configuration and data bus may affect the operation of the McBSP.

Two potential issues could be encountered while debugging an application that uses the EDMA to service the McBSP:

- An emulation access to the data receive register (DDR) through data bus may clear the RRDY bit.
- An EDMA access to the DRR or the data transmit register (DXR) through the data bus may be lost if an emulation access is pending through the configuration bus to those same registers.

The likelihood of encountering these issues is very low; however, they should be noted when debugging applications which use the EDMA to service the McBSP.

Workaround(s): To avoid these issues, do one of the following:

1. In Code Composer Studio, do not open any memory, watch, or register windows on the McBSP registers using the configuration bus address at 0x02A0 0000 while the EDMA is running.
2. Use the data bus address at offset 0x3000 0000 to read the McBSP registers, except for the data receive register (DDR), through Code Composer Studio.
3. Clear the FREE bit in the serial port control register (SPCR).

Advisory 3.1.9 *SRIO: Using NREAD to Read Invalid Memory Space Causes a Timeout and Halts the Port that Processed the NREAD Request*

Revision(s) Affected: 3.1 and earlier

Details: On TCI6482 devices, reading an invalid or reserved memory space generates a bus error message from the DMA bus.

On silicon revision 1.1, when a master node uses the NREAD command to access an invalid or reserved memory space on the DSP, the SRIO port will send a corrupted response packet instead of an error response. The outbound portion of the SRIO port will also be halted. If the master node is another DSP, the completion code of the LSUn Control Regiser 6 (LSUn_REG6) will be set to 001b (transaction timeout occurred on non-posted transaction).

On silicon revisions 2.0, 2.1, and 3.1, the behavior of the device has been changed such that the SRIO module ignores the DMA bus error message and sends a valid response packet with a payload of garbage data. The outbound portion of the SRIO port also is not halted. Therefore, if a master node uses NREAD to access invalid or reserved memory space, the master node receives a packet with a garbage payload. If the master node is another DSP, the DSP receives the packet and stores it into receive memory; the completion code of LSUn_REG6 is set to 000b (transaction complete, no errors).

Workaround(s): Always make sure to use NREAD requests to access valid memory space.

Advisory 3.1.10 *L1D Cache: C64x+ L1D Cache May Lose Data or Hang DMA Operations Under Certain Conditions*

Revision(s) Affected: 3.1 and earlier

Details: Under certain conditions, parallel loads with predication to the same cache line may cause victims to be dropped and/or the DMA to hang.

All of the following conditions ***must*** be true in order for this problem to occur:

1. Two LD instructions in parallel.
2. Both are LDs to the same cache line (upper 26 address bits are the same).
3. The LD using T1 is predicated and the predicate is *false*.
4. The LD using T2 is either not predicated, or is predicated and the predicate is *true*.
5. The cache line is absent from the cache.
6. The two other lines in the same L1D set are valid.
7. The LRU cache line in the set is dirty.

Results:

- L1D informs L2 to expect a victim for the affected set.
- L2 stalls DMAs with addresses that correspond to that set.

NOTE: DMA includes accesses from IDMA, EDMA, and any external masters, such as PCI or other CPUs.

- L1D processes the true-predicated request correctly.
- L1D does not send the indicated victim.

Impact: If the load instruction reads a cacheable location:

- The updated data in the LRU line gets dropped.
- DMA accesses whose addresses match the affected set hang.

If the load instruction reads a non-cacheable location:

- L1D retains the updated data from the LRU line.
- DMA reads may see stale data if the LRU line's address is in L2 memory.

Workaround(s): Use Code Gen patch 6.0.3 (available on update advisor) to recompile your source code and avoid this issue. Libraries supplied by TI will be re-released using the 6.0.3 compiler patch. Customer-generated libraries from TI's third-party supplier may also need to be recompiled.

For existing object code and libraries, an available Perl script can determine locations of parallel predicated loads that may fail. The script is available at the same update advisor location as the Code Gen patch.

Advisory 3.1.11 *McBSP: Transfers Less than 32 Bits are Ignored in Some Cases When Device is Configured for Big-Endian Mode*

Revision(s) Affected: 3.1 and earlier

Details: The McBSP is capable of transmitting and receiving various word lengths; e.g., 32, 24, 16, and 8 bits. Usually, the EDMA is used to service the McBSP with the necessary number of bytes on each transmit and/or receive event, but the CPU can alos be used. Furthermore, the CPU and EDMA can read/write data to the McBSP's Data Transmit Register (DXR) and Data Receive Register (DRR) through two separate buses: a configuration bus and an EDMA bus.

On the TCI6482 device, when the device is configured in big-endian mode, the McBSP will ignore any transfer that is less than 32 bits, irrespective of the bus being used. Therefore, the EDMA and CPU must be configured to always transfer 32 bits when accessing the McBSP registers.

Note that none of these issues apply when the device is configured in little-endian mode.

Workaround(s): If the device is being used in big-endian mode, program the EDMA and CPU to use 32-bit transfers.

Note that, regardless of the number of bytes written or read by the CPU and EDMA, the McBSP word length can be programmed to be smaller than 32 bits. Extra bits written to the McBSP DXR and DRR are ignored.

Advisory 3.1.12 *PCI: PCI Reset and Chip Reset Must Always Be Asserted Together*

Revision(s) Affected: 3.1 and earlier

Details: The PCI module in the TCI6482 device has an internal and an external reset source. The internal reset is asserted by the reset controller during power-on, warm, and max reset, see Figure 5. The internal reset is also asserted when the device state control registers are used to disable the PCI module. The external reset is asserted through the PCI reset pin, $\overline{\text{PRST}}$.

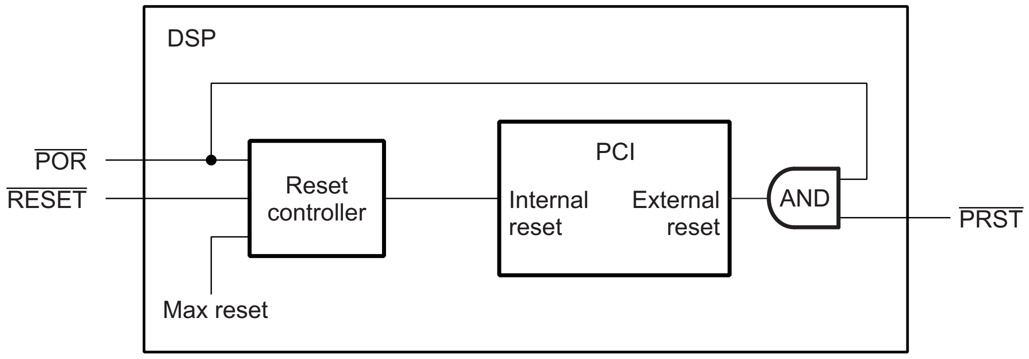


Figure 5. Internal Reset Asserted By Reset Controller

The PCI module can lock up or go into a bad state if its internal and external resets are asserted independently, meaning one is asserted and the other is not. Power-on and warm reset are controlled differently from system to system and, therefore, the effect of this issue is system specific.

Systems that assert the warm reset pin ($\overline{\text{RESET}}$) or the power-on reset pin ($\overline{\text{POR}}$) and the PCI reset from the same source and at the same time are not affected by this issue. Generating these resets from the same source at the same time ensures that both the internal and external resets of the PCI module are asserted together.

Systems that assert the chip warm reset pin ($\overline{\text{RESET}}$) or the power-on reset pin ($\overline{\text{POR}}$) and the PCI reset pin ($\overline{\text{PRST}}$) from independent sources at different times will be affected by this issue.

See Advisory 3.1.13 for issues relating to max reset. Also, see Advisory 3.1.14 for issues relating to the device state control registers.

Workaround(s): Systems should be designed such that a power-on or warm reset always generates the PCI reset and vice versa. Example circuits that ensure both the chip resets and the PCI reset are asserted together are shown in Figure 6, Figure 7, and Figure 8. Note that asserting power-on reset also asserts PCI reset internally. Also, since a PCI reset will cause a chip reset, the configuration pins of the device will also be latched on a PCI reset. Therefore, these pins must be at valid levels to ensure the device is taken out of reset in the desired mode.

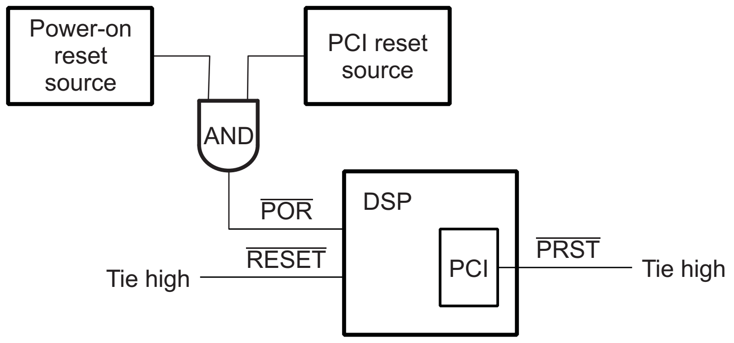


Figure 6. Example With $\overline{\text{RESET}}$ and $\overline{\text{PRST}}$ Pins Tied High

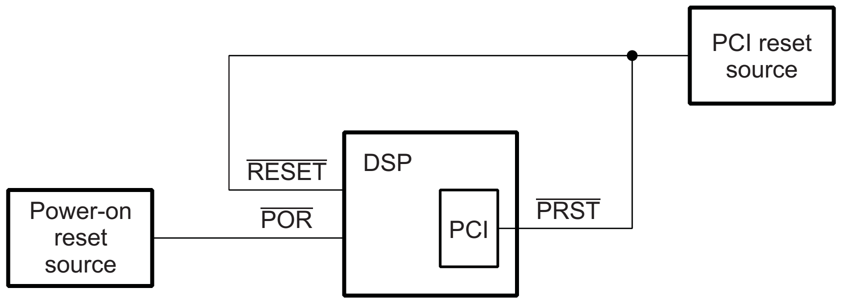


Figure 7. Example With $\overline{\text{PRST}}$ and $\overline{\text{RESET}}$ Pins Tied Together

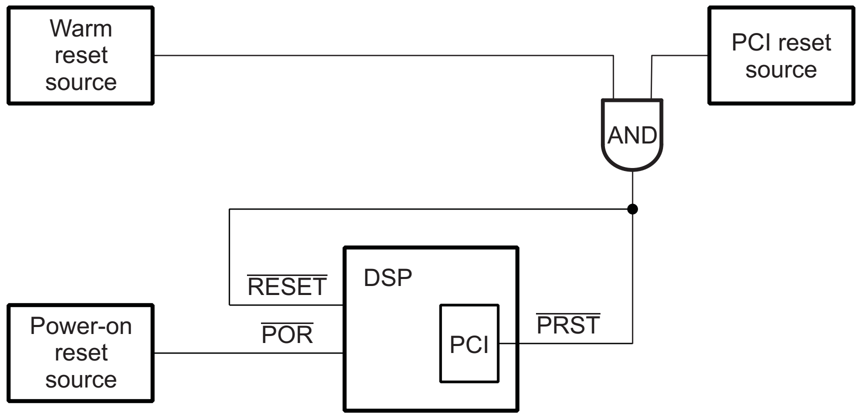


Figure 8. Example With Independent Power-On Reset, Warm Reset, and PCI Reset Sources

Note that in the above circuits a power-on or warm reset will reset the DSP and its PCI only; other devices on the PCI bus will not be reset. To avoid interrupting an ongoing PCI transaction to/from the DSP, the DSP PCI should be *disconnected* from the PCI system. The sequence below ensures that the DSP PCI is *disconnected* from the PCI system. This sequence can be executed by the DSP itself or by an external host. Note that resetting the DSP PCI in the middle of a master or slave transaction can cause a PCI bus fault at the system level.

1. Ensure that the PCI bus is not parked on the DSP PCI pins since the DSP will place its PCI output pins in a high-impedance state whenever it is reset.
2. Stop all PCI transactions started within the DSP. This includes any ongoing EDMA transactions to/from PCI memory space.
3. Disable the PCI slave and master. DSP code can do this through the back-end registers of the PCI or an external host can do this through configuration/memory

   accesses to the PCI registers.
   (a) Clear the base enable bits ($\overline{\text{BASE}}$_EN) of the Slave Control Register (PCISLVCNTL).
   (b) Clear the bus master bit (BUS_MS) and memory access bit (MEM_SP) of the Command/Status Register (PCICSR).
4. Assert the DSP and PCI reset.
5. Once DSP comes out of reset, re-configure the PCI (DSP code or an external host can do this).
6. Restart PCI transactions to/from the DSP.

Also, once the sequence above has been executed, the DSP will not acknowledge accesses from external devices until the PCI has been reconfigured (either via DSP software or an external host). Devices trying to access the DSP before the PCI is reconfigured will likely generate a master abort condition; this must be comprehended by the PCI system.

Advisory 3.1.13 *PCI: SRIO Max Reset Should Not Be Used When PCI is Used*

Revision(s) Affected:	3.1 and earlier
Details:	As described in Advisory 3.1.12, the PCI module can lock up or go into a bad state if its internal and external resets are asserted independently, meaning one is asserted and the other is not. Systems that use SRIO to generate a max reset to the DSP and also use PCI are affected by this issue. This is because a max reset will assert the internal reset of the PCI, but it will not affect its external reset. Therefore, the max reset should not be used when the PCI is used.
Workaround(s):	There is no workaround for this issue.

Advisory 3.1.14 *PCI: Device State Control Registers Should Not Be Used to Disable the PCI Once it is Enabled*

Revision(s) Affected:	3.1 and earlier
Details:	As described in Advisory 3.1.12, the PCI module can lock up or go into a bad state if its internal and external resets are asserted independently, meaning one is asserted and the other is not. The device state control registers should not be used to disable the PCI module once it has been enabled. Doing so will assert the internal reset of the PCI without affecting its external reset. To disable the PCI, reset the entire device, as well as the PCI, through its external reset pin.
Workaround(s):	There is no workaround for this issue.

Advisory 3.1.15 *Chip: Writing to Certain Peripheral Memory-Mapped Registers Will Modify Value of PRI_ALLOC Register*

Revision(s) Affected: 3.1 and earlier

Details: As described in the device-specific data manual, the PRI_ALLOC register controls the system priority of peripherals like the HPI and PCI. This register is normally configured once during device initialization by DSP software.

Writing to some peripheral memory-mapped registers (MMRs) affects the value of the PRI_ALLOC register, essentially clearing out the previously programmed value.

Writes to the PRI_ALLOC register do not affect any of these peripheral MMRs and also reads from these MMRs do not return the value of the PRI_ALLOC register.

Writing to the registers below will affect the value of the PRI_ALLOC register:

- PCI Command/Status Register (PCICSR), address 02C0 0004h, read/write register.
- MCBSP1 Data Transmit Register (DXR), address 0290 0004h, read/write register.
- HPI Power and Emulation Management Register (PWREMU_MGMT), address 0288 0004h, read/write register.
- EMAC Transmit Control Register (TXCONTROL), address 02C8 0004h, read/write register.

Workaround(s): The PRI_ALLOC register needs to be updated whenever these registers are modified. There are several ways to do this depending on which peripheral MMR is being used.

PCI Command/Status Register and EMAC Transmit Control Register (TXCONTROL)

Configure PCI and EMAC registers prior to PRI_ALLOC writes.

MCBSP1 Data Transmit Register (DXR)

The Data Transmit Register (DXR) can be accessed through two separate buses: a configuration bus and a data bus. Both paths can be used by the CPU and the EDMA. The impact of this issue depends on the use of the PCI and McBSP peripherals.

- System not using PCI:

  Use the EDMA path to update the Data Transmit register to work around this issue.
- System using PCI:

  If the CPU or EMDA accesses the DXR through the DMA bus the there are no issues.

  If the EDMA accesses the DXR through the configuration bus then the EDMA channel servicing the McBSP must trigger a second channel after writing to the McBSP DXR. The second channel must be set up to update the PRI_ALLOC register. Chaining with intermediate transfer completion can be used to implement this workaround.

  If the CPU accesses the DXR through the configuration bus the value of the PRI_ALLOC register must be saved prior to the CPU write. This value must be restored after the CPU write.

HPI Power and Emulation Management Register (PWREMU_MGMT)

The PRI_ALLOC register need to be updated whenever these registers are modified.

Advisory 3.1.16 *SRIO: Performance Issues Identified Prohibiting Full Utilization of Pin Bandwidth*

Revision(s) Affected: 3.1 and earlier

Details: On the TCI6482 devices, there may be performance degradation depending on mode of operation and packet size used.

During 4x operation at 3.125 and 2.5 Gbaud, there are issues that limit the transmit performance by 25%, or higher, depending on packet size. This is caused by two issues within the physical layer of the Serial RapidIO peripheral. The first issue is due to a stuttering buffer used for crossing a clock boundary. This causes extra clock cycles between packets when moving data within the peripheral. The second issue is that an EOP control symbol and 3 idle sequences are sent after every transmitted packet, creating an interpacket gap on the link itself. Both issues impact overhead for sending a packet and have a larger impact on smaller packet traffic. For 4x operation at 1.25 Gbaud and 1x operation at any rate, these issues do not present themselves.

There is also a receive physical layer performance issue that affects both 4x and 1x operation. This is an issue where clock cycles are inserted between packets being moved within the peripheral. The degree of impact on the receive throughput depends on data rate, number of ports, and packet size. Receive physical layer RETRYs are issued if the buffering is depleted. Hardware at the transmitter is responsible for resending packets.

Table 7 illustrates the approximate internal utilization that can be achieved based on a 10-Gbps bus.

Table 7. Receive Internal Bus Utilization

PAYLOAD BYTES	APPROXIMATE BUS UTILIZATION
8	45%
16	45%
32	50%
64	63%
128	75%
256	85%

For example, if the peripheral is set up in 4x mode running at 3.125 Gbaud, then the performance will be degraded to approximately these levels. If the peripheral is setup in 1x mode, using 2 ports at 3.125 Gbaud (providing a theoretical maximum data rate of 5 Gbps), the performance degradation is negligible and only seen at the very small packet sizes.

Table 8 and Table 9 represent simulation data incorporating both the receive and transmit performance issues. The same performance levels were observed when measuring actual silicon.

Table 8. 1 Port 4x mode Using 4 LSUs, NWRITE packets, 3.125 Gbaud

PAYLOAD BYTES	MEASURED RESULTS WITH ISSUE	EXPECTED RESULTS WITHOUT ISSUE
8	2.9 Gbps	3.5 Gbps
16	3.2 Gbps	4.5 Gbps
32	4.1 Gbps	6.3 Gbps
64	5.2 Gbps	10 Gbps
128	6.3 Gbps	10 Gbps
256	7.2 Gbps	10 Gbps

Table 9. 1 Port 1x mode Using 1 LSU, NWRITE packets, 3.125 Gbaud

PAYLOAD BYTES	MEASURED RESULTS WITH ISSUE	EXPECTED RESULTS WITHOUT ISSUE
8	1.7 Gbps	1.7 Gbps
16	2.2 Gbps	2.2 Gbps
32	2.31 Gbps	2.5 Gbps
64	2.38 Gbps	2.5 Gbps
128	2.43 Gbps	2.5 Gbps
256	2.46 Gbps	2.5 Gbps

Workaround(s): There are no workaround options available.

Advisory 3.1.17 *CPU: Back-to-Back SPLOOPs With Interrupts Can Cause Incorrect Operation on C64x+ CPU*

Revision(s) Affected: 3.1 and earlier

Details: Back-to-back software pipeline loops (SPLOOPs) with interrupts can cause incorrect operation on the C64x+ CPU. This issue occurs when the first SPLOOP is interrupted and there are less than 2 execute packets between the SPKERNEL of the first SPLOOP block (SPKERNEL instruction marks the end of the first SPLOOP block) and the SPLOOP instruction of the second SPLOOP block (SPLOOP instruction marks the beginning of the second SPLOOP block). The first SPLOOP block terminates abruptly (i.e., without completing the loop, even though the termination condition is false). The failure mechanism can be seen as a hang or by the first SPLOOP block draining for the interrupt and starting the second SPLOOP block without taking the interrupt or returning to complete the first SPLOOP block.

Workaround(s): The C6000 compiler release v6.0.6 and above detects this problem. If there are fewer than 2 execute packets between the SPKERNEL and SPLOOP instructions, the compiler will add the appropriate number of NOP instructions following the SPKERNEL instruction.

For example,

```
          ...
          SPKERNEL     0, 0
          NOP          1       ; SDSCM00012367 HW bug workaround
          MVK     .L1  0x1,A0
[ A0]     SPLOOPW      3       ;12
          NOP          1
          ...
```

The assembler will detect sequences that could potentially trigger this issue, and issue a remark. For example,

```
"neg_test.asm", REMARK at line 21 [R5001] SDSCM00012367 potentially
     triggered by this execute packet sequence. SLOOP must be at
     least 2 EPs away from previous SPKERNEL for safe interrupt
     behavior.
```

Note: The assembler tool, asm6x.exe, can be used to determine if a previous version of the compiler generated code that could potentially be affected by this silicon issue. The assembler can also be used on assembly source code to see if the source could be affected by this issue. Replace the old version of asm6x.exe with the 6.0.6 asm6x.exe in your current build setup and recompile or reassemble.

Internal Tracking Number: 4

Advisory 3.1.18 *CPU: C64x+ CPU Incorrectly Generates False Exceptions for Multiple Writes*

Revision(s) Affected: 3.1 and earlier

Details: The C64x+ CPU may generate an incorrect resource conflict exception when taking an interrupt. This only affects applications that run with exceptions enabled. Applications enable exceptions by writing 1 to the GEE bit in the Task State Register (TSR). Applications that do not enable exceptions are not affected by this errata.

The CPU generates this incorrect exception in the following scenario:

1. The CPU begins draining the pipeline as part of an interrupt context switch. During this time, the CPU annuls instructions in the pipeline that have not yet reached the E1 pipeline phase while it drains the pipeline.
2. The first annulled execute packet (resident in the DC pipeline stage at the time draining begins) writes to one or more predicate registers. Because it is annulled, the writes do not occur.
3. The second annulled execute packet (resident in the DP pipeline stage at the time draining begins) has a predicated single cycle instruction that uses a predicate written by the execute packet described in item 2. Because it is annulled, the write does not occur.
4. The value held in the predicate register would cause the instruction in the second annulled execute packet to write to some register in the same cycle as another instruction if it were not annulled. The conflicting writes would not happen if the first execute packet had not been annulled.

The exception is not a valid exception. If the CPU executed instructions described in items 2 and 3 above, rather than annulling them while draining the pipeline for an interrupt, the execute packet in item 2 would set the predicate(s) such that the writes in the subsequent execute packet do not conflict.

Examples of sequences that generate the incorrect exception are:

```
            ZERO A0
            ZERO B0
---------------------> interrupt occurs
            MVK 1, A0  ;(1st annulled EPKT)
  [!A0]     MVK 2, A1  ;(2nd annulled EPKT) \_ Appears both MVKs write A1,
||[!B0]     MVK 3, A1  ;(2nd annulled EPKT) /  triggers invalid exception.

...

            ZERO A0
  [!A0]     LDW *A4, A5
            NOP
            NOP
--------------------> interrupt occurs
            MVK 1, A0  ;(1st annulled EPKT)
  [!A0]     MVK 2, A5  ;(2nd annulled EPKT) LDW writes A5 this cycle

...

            ZERO A0
  [!A0]     DOTP2 A3, A4, A5
            NOP
-------------------> interrupt occurs
            MVK 1, A0 ;(1st annulled EPKT)
  [!A0]     MVK 2, A5 ;(2nd annulled EPKT) DOTP2 writes A5 this cycle
```

Workaround(s): The CPU only recognizes the incorrect exception while it drains the pipeline for an interrupt. As a result, the CPU begins exception processing upon reaching the interrupt handler. The NRP (NMI Return Pointer Register) and NTSR (NMI Task State Register) will reflect the state of the machine upon arriving at the interrupt handler.

Therefore, to identify the incorrect resource conflict exception in software, verify the following conditions at the beginning of the exception handler prior to normal exception processing:

1. Exception occurred during an interrupt context switch.
   (a) In NTSR, verify that INT=1, SPLX=0, IB=0, CXM=00.
   (b) Verify that NRP points to an interrupt service fetch packet. That is, (NRP & 0xFFFF FE1F) == (ISTP & 0xFFFF FE1F).
2. The exception is a resource conflict exception. In IERR, verify that RCX == 1 and all other IERR bits == 0.
3. The exception is an internal exception. In EFR, verify that IXF == 1 and all other EFR bits == 0.

Upon matching the above conditions, suppress the exception as follows:

1. Clear EFR.IXF by writing 2 to ECR.
2. Resume the interrupt handler by branching to NRP.

The above workaround identifies and suppresses all cases of the incorrect resource conflict exception. It resumes normal program execution when the incorrect exception occurs, and has minimal impact on the execution time of program code. The interrupted code sequence runs as expected when the interrupt handler returns.

The workaround also suppresses a particular valid exception case that is indistinguishable from the incorrect case. Specifically, the code will suppress the exception generated by two instructions with different delay slots (e.g., LDW and DOTP2) writing to the same register in the same cycle, where the conflicting writes occur during the interrupt context switch.

An example of a sequence with incorrectly suppressed exception is:

```
      LDW *A0, A1
      DOTP2 A3, A2, A1
      NOP
-----------------> interrupt occurs
      NOP
      NOP             ; Both LDW and DOTP2 write to A1 this cycle
```

The workaround will not suppress these valid resource conflict exceptions if the multiple writes occur outside an interrupt context switch. That is, the workaround will not suppress the exception generated by the code above when it executes without an interfering interrupt.

For more details, see the following sections in the *TMS320C64x/C64x+ DSP CPU and Instruction Set Reference Guide* (literature number SPRU732).

- *Interrupt Service Table Pointer Register (ISTP)* describes the ISTP control register.
- *Nonmaskable Interrupt (NMI) Return Pointer Register (NRP)* describes the NRP control register.
- *TMS320C64x+ DSP Control Register File Extensions* describes the ECR, EFR, IERR, TSR and NTSR control registers.
- *Pipeline* describes the overall operation of the C64x+ pipeline, including the behavior of the E1, DC and DP pipeline phases.
- *Actions Taken During Nonreset Interrupt Processing* describes the operation of the C64x+ pipeline during interrupt processing, including how it annuls instructions.
- *C64x+ CPU Exceptions* describes exception processing.

Internal Tracking Number: 5

Advisory 3.1.19 *SRIO: Packet Forwarding Cannot Be Used With NREAD Response Packets Greater Than 16 Bytes*

Revision(s) Affected: 3.1 and earlier

Details: Packet forwarding uses programmable look-up tables to direct incoming packets to an outbound port when the packets do not belong to the local device. Packet forwarding is carried out at the logical layer of the serial RapidIO (SRIO) without the interaction of the CPU. The SRIO logical layer copies incoming packets from an inbound buffer to an outbound buffer. When used for packet forwarding, it forwards all types of packets, including response, maintenance, DOORBELL, and message packets.

The current SRIO design fails to correctly copy response packets with a payload greater than 16 bytes from the inbound to the outbound buffer. The first 16 bytes are correctly copied, but the remainder of the payload is discarded. This issue affects only NREAD response packets since their payloads can be up to 256 bytes. Packet types with small responses, such as NWRITE_R, maintenance, message, and DOORBELL packets are not affected by this issue.

Workaround 1: Use a data *push* model, where each device in the daisy chain only submits write requests. Using this approach will avoid the issue and provide the lowest latency solution.

Workaround 2: Two options exist if NREAD response packets cannot be avoided; for example, when reading core dump information from an unresponsive processor which is unable to initiate traffic by itself. The first option is to use software to segment read requests into 16-byte NREADs. Note that this option will work functionally, but may take too much time.

The second option is illustrated in Figure 9.

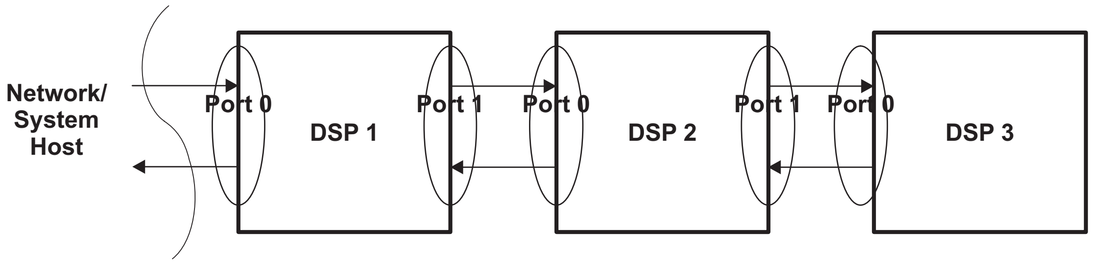


Figure 9. Daisy-Chain Example

In this example, assume that DSP3 is down and the system host wants to do a large NREAD of DSP3 to examine the core dump. The issue discussed above prohibits the NREAD from completing correctly because, as the response packets from DSP3 are sent back, they are corrupted by DSP2 and DSP1 packet forwarding. Instead, the system host needs to request that the adjacent DSP (DSP2) generates the NREAD request to DSP3. The NREAD responses are sent to DSP2 and temporarily stored in memory. Then, DSP2 can generate NWRITE/NWRITE_R/SWRITE packets to the system host with the needed payload. These packets are correctly forwarded by DSP1 to the system host since they are request packets and not responses.

Advisory 3.1.20 *PLL Controller: GOSTAT Bit of PLL Controller Does Not Reflect GO Operation Status*

Revision(s) Affected: 3.1 and earlier

Details: As documented in the *TMS320TCI648x DSP Software-Programmable Phase-Locked Loop (PLL) Controller User's Guide* (literature number SPRU806), after software starts a GO operation to change the clock divider ratios, it must poll the GOSTAT bit in the PLL controller status register (PLLSTAT) to determine the GO operation has completed before continuing since internal clocks may be stopped.

On current revisions of the TCI6482 device , the GOSTAT bit does not reflect the status of the GO operation. Software should not rely on the GOSTAT bit of the PLL controller status register (PLLSTAT) to determine the status of the GO operation.

The TCI6482 device includes two PLL controllers. Only divider D4 and divider D5 of PLL controller 1 and divider D1 of PLL controller 2 are programmable on this device . Software must not access any resource in the clock domain of these dividers N number of SYSCLKREF cycles after changing the divider clock frequency through a GO operation. The number N can be calculated using the formula given in the workaround below.

NOTE: Existing software may not need to be modified if the resources, such as memory-mapped registers, of the modules being affected by the clock change are not being accessed within N cycles after setting the GOSET bit.

Workaround(s): A modified programming sequence from that given in the *TMS320TCI648x DSP Software-Programmable Phase-Locked Loop (PLL) Controller User's Guide* (literature number SPRU806) can be followed to ensure the GO operation has completed.

Modified Divider Programming Sequence

1. Check for the GOSTAT bit in PLLSTAT to clear to 0 to indicate that no GO operation is currently in progress. (Step included for forward compatibility only.)
2. Program the RATIO field in PLLDIV*n* with the desired divide factors. If the RATIO field changed, the PLL controller will flag the change in the corresponding bit of DCHANGE.
3. Set the GOSET bit in PLLCMD to 1 to initiate the GO operation to change the divide values. During this transition, any SYSCLK being changed will be paused momentarily.
4. Wait for N number of SYSCLKREF clock cycles to ensure divider changes have completed. The number N can be calculated using the following formula:

   (2 x Least Common Multiple (LCM) of all of the old SYSCLK divide values) + 50 cycles overhead
5. Wait for the GOSTAT bit in PLLSTAT to clear to 0. (Step included for forward compatibility only.)

Example on Calculating Number of Clock Cycles N for PLL Controller 1

Settings before divider change:

- PLLDIV4.RATIO = 3 (divide-by-8)
- PLLDIV5.RATIO = 3 (divide-by-4)

New divider settings:

- PLLDIV4.RATIO = 4 (divide-by-10)
- PLLDIV5.RATIO = 3 (divide-by-4)

The least common multiple between the old divider values of /4 and /8 is /8. Therefore, the number of cycles N is:

N = (2 x 8) + 50 overhead = 66 SYSCLKREF source clock cycles

If PLL controller 1 is in PLL mode (PLLCTL.PLLEN = 1), the SYSREFCLK source clock is the PLL1 output clock. If PLL controller 1 is in PLL bypass mode (PLLCTL.PLLEN = 0), the SYSREFCLK source clock is the device clock source CLKIN1.

Example on Calculating Number of Clock Cycles N for PLL Controller 2

Settings before divider change:

- PLLDIV1RATIO = 1(divide-by-2)

New divider settings:

- PLLDIV1RATIO = 4 (divide-by-5)

Since there is only one configurable clock, the least common multiple will always be the older divider value, in this case that is /2. Therefore, the number of cycles N is:

N = (2 x 2) + 50 overhead = 54 SYSCLKREF source clock cycles

The PLL controller 2 is always in PLL mode (PLLCTL.PLLEN = 1), hence the SYSREFCLK frequency is always equal to CLKIN2 x 10.

Advisory 3.1.21 *Potential SerDes Clocking Issue*

Revision(s) Affected:	3.1 and earlier
Details:	An issue has been found in the SerDes interfaces that causes a SerDes clocking problem in normal functional operation. This problem will not occur when external pull-down is applied on the TCK pin (JTAG controller clock). SerDes are used in the Serial RapidIO interface (SRIO). The TCK pin (JTAG controller clock) is internally assigned to an internal signal that is used by the SerDes macro. For the SerDes macro to get proper clocking in the normal functional operation, it needs the internal signal to be held low. However, there is an internal pull-up on the TCK, creating problems for SerDes operation. This problem exists on all SerDes interfaces.
Workaround(s):	The TCK pin should be externally pulled down with a 1-kΩ resistor.

Advisory 3.1.22 *EMIFA: Occurrence of Read Data Corruption for Synchronous Interface Due to Impedance Mismatch at AECLKOUT*

Revision(s) Affected: 3.1 and earlier

Details: The EMIFA AECLKOUT is used to synchronize external devices and synchronous read data latch. For this reason, the EMIFA timing characteristics and requirements are more dependent upon the specific loading topologies of the targeted system. It is important to properly terminate the targeted system to ensure the following:

- Minimize reflections at the DSP and external device.
- Over/undershoot within the device-specific data manual specification.
- The AECLKOUT is monotonic at the DSP and the external device between V_{IL}/V_{IH} for the rising/falling edge with respect to weak/strong buffers.

If the above conditions are not within specification — as defined in the *TMS320TCI6482 Communications Infrastructure Digital Signal Processor* data manual (literature number SPRS246) — then read data corruption can occur. If the signal quality is not achievable with the proper termination, then external buffering may be required. This ensures that the AECLKOUT is monotonic at the DSP and at the external device between V_{IL}/V_{IH}. In all cases, techniques for high-speed digital design should be applied to the EMIFA interface. Since the the AECLKOUT is looped back for read data latch, it is very important to terminate the AECLKOUT at the DSP to ensure clock signal integrity. There is an inherent "shelf" near the switch point at the driver, which causes the transition to be non-monotonic at the DSP. The impact of this issue depends on where the AECLKOUT is non-monotonic at the DSP. The AECLKOUT topology must ensure that the AECLKOUT signal at the DSP and the load is monotonic between V_{IL}/V_{IH}.

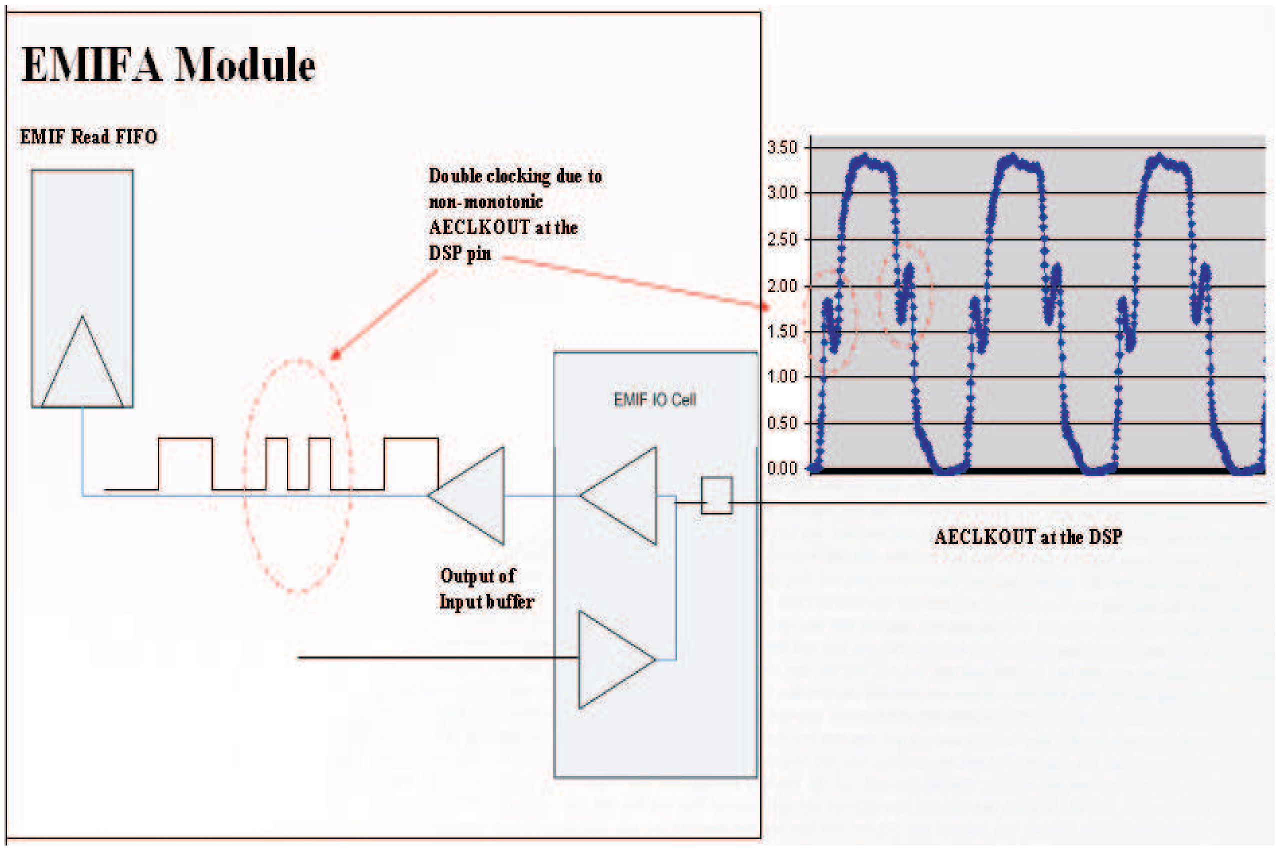


Figure 10. AECLKOUT at the DSP

Workaround(s): The workaround provides details on how to select the correct series termination values at the driver and loads to minimize reflections and over/undershoot. The workaround must also ensure that the AECLKOUT signal is always monotonic between V_{IL}/V_{IH} at the driver and the loads for weak and strong conditions.

Termination at the DSP

In order to select the appropriate series termination resistor at the DSP, the following information is provided:

- DSP impedance = 25 Ω
- Example AECLKOUT board topology

In this example we assume trace impedance = 50 Ω (typical impedance range is 50-75 Ω). In this example, use the following formulas to calculate the driver switch voltage, V_{OUT}, with respect to the chosen series termination resistor at the DSP. Note that the switch point is the voltage where the far-end reflection causes the AECLKOUT to be non-monotonic at the DSP. The goal here is to ensure that the switch-point voltage is always above V_{IH} and below V_{IL} for weak and strong buffers.

- $Driver_R$ = output driver impedance
- $Trace_R$ = board trace impedance

$$V_{OUT} \not\approx \{ Driver_R / (Driver_R + Trace_R) \} * 3.3\ V \tag{1}$$

- $Switch_V$ = driver switch-point voltage

$$Switch_V \not\approx 3.3\ V - V_{out} \tag{2}$$

To optimize for a specific switch point voltage

1. Solve Equation 2 for V_{OUT}.
2. Solve Equation 1 for $Trace_R$ by plugging in the new value for V_{OUT}.

Note: The higher the series termination resistor at the DSP, the lower the rise time is at the load.

Termination at the Load

Select the appropriate series termination resistor needed to match the impedance at the load. As an example, assume the trace impedance at the stub = 50 Ω and the trace lengths are symmetrical. If your trace lengths are not symmetrical you have to calculate the appropriate values. As the signal travels down the transmission line, it comes to a point of divergence where AECLKOUT sees two 50-Ω pathways for a total of 25 Ω or $Z_o/2$. This creates a three-way split of the signal energy:

- 1/3 returns to the source
- 1/3 goes toward load1
- 1/3 goes toward load2

To minimize reflections, terminate each load with series termination resistance = $Z_o/2$ = 100 Ω. This reduces the energy that returns to the source, thus reducing reflections at the DSP. This also allows the full signal to propagate toward the loads. After establishing the proper termination values, confirm the AC timings at the load and source for setup, hold, and rise time according to the device-specific data manual specifications.

Advisory 3.1.23 *DDR2 EMIF Buffers Not Totally Compensated by Default*

Revision(s) Affected: 3.1 and earlier

Details: The output buffers on the DDR2 EMIF contain dynamic impedance compensation circuitry to maintain a constant output impedance across temperature, voltage, and silicon process variation. This impedance compensation circuitry must configure each individual output buffer. The output buffer compensation for each DDR2 output buffer is not complete until both a 1-to-0 and 0-to-1 transition has occurred on that output.

Until this compensation occurs, the output drive strength is probably less than ideal. The DDR2 EMIF cycles that occur before dynamic compensation is complete may fail. Since the mode register (MR) write cycles are the first cycles initiated by the DDR2 EMIF after a reset, these cycles are at risk. Similarly, after EMIF configuration, the first writes to DDR2 memory are also at risk.

This issue has not been seen in the field. It has only been observed on test fixtures at TI. Therefore, it appears to have a very low probability of affecting existing designs. The recommended topologies that only have one or two loads per DDR2 EMIF without VTT termination appear to be resilient to this condition. The possibility of failure can only be eliminated if the software workaround described below is implemented.

Some system start-up sequences improve the probability of robust operation, such as:

- Incomplete compensation may cause mode register (MR) writes to fail. This could result in DDR2 performance lower than expected or complete failure. The risk of this occurring is reduced through multiple MR write operations since compensation of most address and control output buffers and both clock output buffers are completed before the final MR write. Most DDR2 EMIF configuration sequences, including the one implemented in the CSL, result in multiple MR write operations. MR write cycles are also designed to complete on the slowest possible DDR2 memory devices. This is another reason these cycles have a high probability of success.
- Incomplete compensation may cause initial DDR2 memory writes to fail. This could cause the DSP to execute incorrectly if these initial writes are code or critical data. Many system implementations use a secondary bootloader to load the full binary image. The secondary bootloader is then discarded after boot completion. Therefore, any invalid writes would occur in the bootloader and the full application code is loaded after the DDR2 EMIF output buffers have been activated many times. (This is not a full guarantee of complete compensation but the probability is high that all of the output buffers are fully compensated by the time the secondary bootloader is written.)
- A better guarantee that this compensation has no latent impact is validation of the full binary image through some type of code checksum at the end of the boot process. If the code is verified in this way, the system is guaranteed to be robust.

Workaround(s): This workaround has to be executed every time the DDR2 EMIF is initialized. Since it should occur before valid mode register writes can be completed, the EMIF configuration has to be repeated after the output buffers are fully compensated. The sequence of steps listed below completes the dynamic compensation for all of the DDR2 EMIF output buffers. The CSL API function CSL_ddr2HwSetup is called during the normal bring-up process to trigger MR writes. Therefore, the workaround code can be put before the API function call.

Sample code:

```
/* Define the following variables. */
Uint32     tempData0, tempData1;
/* Define the following pointers to compensate the address buffers. */
Uint32     *pDdr2Data_temp0 = (Uint32 *) 0xEAAAAAA8;
Uint32     *pDdr2Data_temp1 = (Uint32 *) 0xE5555554;
/* Use 0xF5555554 on systems using 512MB of memory. */

/**************************************************************
```

```
The following code needs to be executed at the beginning of every DDR2 EMIF
initialization.
*************************************************************/

/* Enable self-refresh mode and set an appropriate REFRESH_RATE to guarantee
200us delay before CKE goes high.  REFRESH_RATE value has to be calculated based
on DDR2 clock being used. */
hDdr2->regs->SDRFC = 0x80001388;

/* Disable self-refresh mode and set an appropriate REFRESH_RATE to have a
correct refresh cycle.  REFRESH_RATE value has to be calculated based on DDR2
clock being used. */
hDdr2->regs->SDRFC = 0x00000753;

/*Write and read the first location with a 0xAAAAAAAA pattern.*/
tempData0 = 0xAAAAAAAA;
*pDdr2Data_temp0 = tempData0;     /* DDR2 memory write */
tempData1 = *pDdr2Data_temp0;     /* DDR2 memory read */

/* Perform two more writes with a 0x55555555 and 0xAAAAAAAA pattern to complete
the compensation cycle. */
tempData0 = 0x55555555;
*pDdr2Data_temp1 = tempData0;     /* DDR2 memory write */
tempData0 = 0xAAAAAAAA;
*pDdr2Data_temp0=tempData0;       /* DDR2 memory write */
```

Advisory 3.1.24 *SRIO Port 0 Reset Affects Other Ports*

Revision(s) Affected: 3.1 and earlier

Details: The SerDes for SRIO should allow the reset of individual 1X ports without affecting the state of the other operational ports. There are dedicated MMR bits to reset 1X ports, which are the BLKn_EN (n=5..8) at offsets 0x60, 0x68, 0x70, and 0x78. However, the BLK5_EN that controls reset for port 0 also resets all other ports. Therefore, it is impossible to reset port 0 without affecting all other ports.

Workaround(s): There is no workaround for this advisory.

Advisory 3.1.25 *SRIO OUTBOUND_ACKID Field Not Read Correctly*

Revision(s) Affected: 3.1 and earlier

Details: The OUTBOUND_ACKID field of the RIO_SP(n)_ACKID_STAT register should be updated by hardware each time a packet is sent out. The value should reflect the ACKID value to be used on the next transmit packet. This field is being updated by the hardware as expected. The field can also be written by the software and these writes also succeed. However, a hardware error prevents this field from being read. The OUTBOUND_ACKID always reads as zero. This problem does not cause any impact to link operation.

Workaround(s): There is no workaround for this advisory.

Advisory 3.1.26 *PCI AC Timings Differ From Specifications*

Revision(s) Affected: 3.1 and earlier

Details: The device's PCI signals' slew rate differs from that specified in the *PCI Local Bus Specification* revision 2.3, as shown in Table 10.

Table 10. PCI Slew Rate

PARAMETER	PCI SPECIFICATION VALUE	ACTUAL VALUE
Tf, minimum	1 V/ns	0.65 V/ns

Although the actual minimum slew rate is below the specified minimum, overall PCI timings are within the bounds permitted by the PCI specification. As shown in Table 11, actual values for the valid signal delay (T_{val}) are well below the permitted maximum. This allows additional time for system propagation delay to compensate for the reduced slew rate. Figure 11 shows the buffer slew and timing performance versus the specification performance in the 66-MHz environment. The 33-MHz environment has even more timing margin than the 66-MHz environment, making 66 MHz the worst case.

Table 11. Valid Signal Delay

PARAMETER	PCI SPECIFICATION VALUE	ACTUAL VALUE
Tval, maximum (33 MHz)	11 ns	6.8 ns
Tval, maximum (66 MHz)	6 ns	4.6 ns

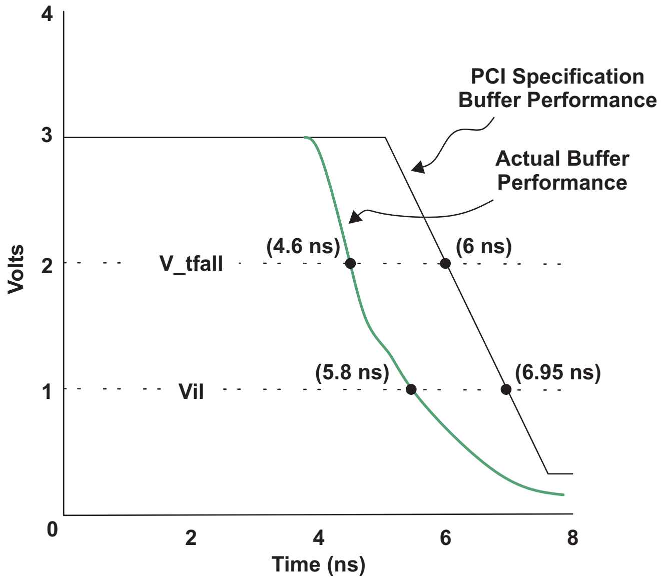


Figure 11. 66-MHz Buffer Slew and Timing Performance vs Specification Performance

Table 12 shows the system timing budget for 66-MHz PCI with values from the specification and values in a system where the device is driving with reduced slew rate.

Table 12. 66-MHz PCI System Timing

PARAMETER	PCI SPECIFICATION VALUE	ALTERNATE COMPATIBLE TIMING
Tval	6 ns	4.6 ns
Tprop	5 ns	6.4 ns
Tskew	1 ns	1 ns
Tsu	3 ns	3 ns
Total Cycle Time	15 ns (66 MHz)	15 ns (66 MHz)

Since Tskew and Tsu are identical in the two systems, no changes to PCI-acceptable layouts or other components on the bus are required. The device can function normally in PCI systems despite the change in timing.

Advisory 3.1.27 *DMA Access to L2 SRAM May Stall When the DMA and the CPU Command Priority is Equal*

Revision(s) Affected: 3.1 and earlier

Details: The L2 memory controller in the C64x+ Megamodule has programmable bandwidth management features that are used to control bandwidth allocation for all requestors. There are two parameters to control this, command priority and arbitration counter MAXWAIT values.

Each requestor has a command priority and the requestor with the higher priority wins. However, there are also counters associated with each requestor that track the number of cycles each requestor loses arbitration. When this counter reaches a threshold (MAXWAIT), which is programmed by the user (or default value), the losing requestor gets an arbitration slot and wins for that cycle.

There are four such requestors: CPU, DMA (SDMA and IDMA), user cache coherency operation, and global cache coherence. Global-coherence operations are highest priority, while user-coherence operations are lowest priority. However, there is active arbitration done for the CPU and the DMA (SDMA/IDMA) commands. The priority for DMA commands comes from an external master as part of the SDMA command or a programmable register, IDMA1_COUNT, in the C64x+ Megamodule for IDMA commands. The priority for CPU accesses to L2 is in a programmable register, CPUARBU, in the C64x+ Megamodule. For the default priority values, see Table 13.

Table 13. TCI6482 Default Master Priorities

MASTER	DEFAULT MASTER PRIORITIES (0 = Highest priority, 7 = Lowest priority)	PRIORITY CONTROL
EDMA3TCx	0	QUEPRI.PRIQx (EDMA3 register)
SRIO (Data Access)	0	PER_SET_CNTL.CBA_TRANS_PRI (SRIO register)
SRIO (Descriptor Access)	0	PRI_ALLOC.SRIO_CPPI
EMAC	1	PRI_ALLOC.EMAC
HPI	2	PRI_ALLOC.HOST
PCI	2	PRI_ALLOC.HOST
VYLNQ	4	PRI_ALLOC.VLYNQ
C64x+ Megamodule (MDMA port)	7	MDMAARBE.PRI (C64x+ Megamodule register)
C64x+ Megamodule (CPU Arbitration control to L2)	1	CPUARBU (C64x+ Megamodule register)
C64x+ Megamodule (IDMA channel 1)	0	IDMA1_COUNT (C64x+ Megamodule register)

The L2 memory controller is supposed to give equal bandwidth to the DMA and the CPU, by alternating between the two for arbitration. Instead, the L2 memory controller gives larger bandwidth allocation to the CPU accesses when the DMA and the CPU priorities are same. The CPU commands keep winning arbitration over the DMA as long as there are no other internal conditions (stalls, etc.) that force the DMA to win arbitration. This typically happens when CPU accesses keep the L2 memory controller busy every cycle, hence, the DMAs stall until the stream of CPU accesses completes. For example, if a continuous stream of L1D write misses to L2 keep the L2 memory controller busy every cycle, the DMAs stall for the entire duration of the write miss stream.

NOTE: When the SDMA has finished sending all of its commands to the L2 controller the C64x+ Megamodule drops the effective transfer priority down to 7 if no further commands are in the pipeline. This condition happens when there is a single word access, a burst of less than 32B with no other SDMA commands pending, or for only the last 64B of a burst that is greater than 64B with no other SDMA commands pending. This effective priority level is what the L2 controller uses to arbitrate these SDMA commands with the CPU, irrespective of what the actual programmed priority value is of the master peripheral. This means that if the CPU is programmed to priority 7, via the CPUARB register, this issue will be triggered. Therefore, priority 7 is not a valid priority level for CPU. If for any reason this *demoted* transfer is still pending upon initiation of another transfer, it will automatically inherit the priority of that new transfer and be pushed through such that it does not stall the new transfer.

Workaround(s): Set the CPU and the DMA commands to L2 on different priorities. As noted above, Priority 7 is not a valid priority for the CPU.

Advisory 3.1.29 *Potential McBSP Transmit Frame Corruption When XDATDLY = 0 and CLKX/FSX is Input Pin Driven By External Clock*

Revision(s) Affected: 3.1 and earlier

Details: There is a potential McBSP transmit problem for the first frame when both the following conditions are met:

1. XDATDLY = 0 mode is used.
2. Either CLKX or FSX, or both, are used as input pins and are driven by external clocks.

The problem is due to the McBSP timing issue on the internal state machine causing the second element copy while the first element transfer is still in progress. In other words, there is a spurious transfer from DXR to XSR after the first bit of the first element is transmitted. Due to this, the first element is partially overwritten by the second element. For example in case of 16-bit elements, the first element transmitted consists of bit 15 of the first element and bit 14 to bit 0 of the second element. The second element actually transmitted is the third element, etc. Since the second element of the frame overwrites the first element, all the following elements are also shifted forward one element position in the frame.

The problem is seen only on the first frame transmitted and not every frame after that. Also note that the problem is only on the transmit side, there is no problem on the receive side with the RDATDLY = 0 setting.

Overview of XDATDLY

The start of a frame is defined by the first clock cycle in which frame synchronization is active. The beginning of actual data transmission with respect to the start of the frame can be delayed, if required. XDATDLY specifies the data delay for transmission. The range of programmable data delay is zero to two bit clocks (XDATDLY = 00b to10b). The XDATDLY field is set using the transmit control register (XCR).

Workaround(s): XDATDLY = 0 mode will be no longer supported when either CLKX or FSX or both are used as input pins and are driven by external clocks. Change XDATDLY from 0 to either 1 or 2. This setting change for the McBSP transmitter requires matching change on whatever external receiver is connected.

Advisory 3.1.31 *SPLOOP CPU Cross-Path Stall*

Revision(s) Affected: 3.1 and earlier

Details: If the following three rules are met, a stall is seen when an SPKERNEL instruction is executed.

1. **Cross-path instruction rule:** An instruction reading a register via the cross path in the first cycle after SPKERNEL instruction.
2. **Data dependence rule:** An instruction in the SPLOOP body that writes to the above cross-path read register. This instruction can be anywhere in the SPLOOP body.
3. **Functional unit rule:** No instruction in parallel with the SPKERNEL instruction that uses the same functional unit as the cross-path read instruction mentioned in rule 1 above.

This results in a one CPU cycle stall for each iteration of the loop. The following are three examples of code that are affected by this issue:

Example 1

```
SPLOOP 1
MV .S1 A0, A1 ;stalls every iteration due to MV after loop
SPKERNEL
MV .S2X A1, B2
```

Example 2

```
PLOOP 14
MV .S1 A0, A1 ;stalls every iteration due to MV after loop
NOP 9
NOP 9
NOP 9
NOP 9
SPKERNEL
MV .S2X A1, B2
```

Example 3

```
SMV .S1 A0, A1 ;stalls every iteration due to MV after loop
SPKERNEL
||NEG .L2 B3, B4 ;Qualifies for rule 3, functional unit rule
MV .S2X A1, B2
```

The following three examples are not affected by this issue:

Example 1

```
;No stalls: No cross path in instruction after SPKERNEL
SPLOOP 1
MV .S1 A0, A1
SPKERNEL
MV .S1 A1, A2
```

Example 2

```
;No stalls: A1 not written to in loop body
SPLOOP 1
MV .S1 A0, A2
SPKERNEL
MV .S2X A1, B2
```

Example 3

```
;No stalls: Instruction in parallel with SPKERNEL prevents bug since
;it's in the same unit as the instruction that uses the cross-path.
SPLOOP 1
MV .S1 A0, A1
SPKERNEL
||NEG .S2 B3, B4 ;masks the bug
MV .S2X A1, B2
```

Workaround(s): The way SPLOOP code is scheduled is controlled by the compiler. Therefore, there are no direct workarounds for non-assembly source code. There are new revisions of the latest compilers that ensure that these three conditions are never met. The following compiler releases include the fix:

- 6.0.25 or later
- 6.1.15 or later
- 7.0.2 or later
- 7.1.0B2 or later
- 7.2.0A or later.

Advisory 3.1.32 *DMA Corruption of L1D$ Allocation*

Revision(s) Affected: 3.1

Details: Under a specific set of circumstances, a snoop-write updates unintended data being allocated into L1D$ from external, cacheable memory. This can lead directly to program misbehavior. If that line is then modified by CPU accesses, a subsequent victim writeback from L1D could commit this corrupted line to lower levels of memory. The key requirements for this issue are:

- Two clean lines in L1D$.
  - This means that a CPU has read two L2 or external, cacheable addresses and has not modified them.
- One more allocated line in L1D$ that can be clean or dirty.
  - Dirty means that a CPU has read and written to any L2 or external, cacheable address.
- Two more parallel CPU reads (occurring in the same CPU cycle).
  - One of the reads must create an L2$ hit (implying an external, cacheable address) and must be a set match to one of the clean lines already in L1D$.
  - The other can be from an L2 SRAM address or an external, cacheable address and must be a set match to the L1D$ cache line mentioned above as clean or dirty.
- Two DMA writes to buffers in L2 SRAM that are a set match to the two clean lines in L1D$.

NOTE:

1. For information on L1D cache coherence protocol, see section 3.3.6, *Cache Coherence Protocol*, in the *C64x+ DSP Megamodule Reference Guide* (SPRU871).
2. The DMA in the following description refers to all non-CPU requestors. This includes IDMA, EDMA, and any other master in the system.

Under a specific set of circumstances listed below, a snoop-write results in data corruption of L1D$. The issue occurs when there is a DMA to L2 for one of the allocated (clean) lines that is also in the process of being replaced by an allocation from external, cacheable memory (implying there was a set match between the two); this is along with another allocation and a DMA to the other allocated (clean) line. L2 sends the DMA requests as snoop-writes to the L1D cache. When the error occurs, the line the second snoop-write was destined for has already been replaced by the allocation from external, cacheable memory. The logic to kill the snoop-write did not get sensitized and the snoop-write ends up corrupting the line that was allocated. Subsequent writes to the corrupted line cause this to get committed to lower levels of memory.

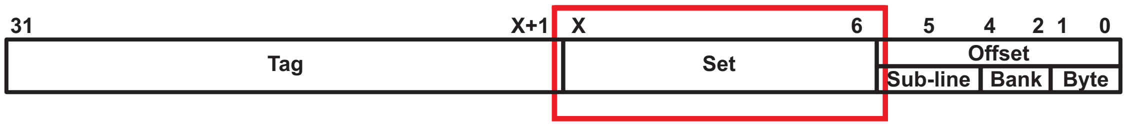
The prerequisite before the window where the issue occurs is:

- The CPU reads two L2 locations that are not a set match to each other and have not been modified since then (CPU/DMA has not written to it). For a description on how to determine if you have a set match or not, see below.
  - These are now two separate 64B cache lines allocated and clean in L1D (referred to here as Cache Lines B and E).
- The CPU reads another L2 location that is not a set match to Cache Lines B and E. It does not matter whether this particular cache line is modified or not before the issue window arrives.
  - Because of this, another 64B cache line is allocated in L1D as clean or dirty (referred to here as Cache Line A).
  - Note that both ways for this particular set must be occupied. It may require more

than one read to this particular cache set.

How to determine if two addresses are a set match:

Determining if two addresses are a set match can be done by comparing certain bits of two addresses. The mapping of an address to a location in L1D cache is shown in Figure 12.



The value X is determined by how large the L1D cache is in the particular application (see Table 14).

Figure 12. L1D Cache Address Mapping

Table 14. Value of X for L1D Cache

AMOUNT OF L1D CACHE	X BIT POSITION
0KB	N/A
4KB	10
8KB	11
16KB	12
32KB	13

If you use the default configuration, 32KB, as an example, bits [13:6] are a set match if they are identical in two different addresses. Some examples of set matches are shown below:

- 0x0080 2A80 00000000100000000010101010000000
- 0x8000 2A80 10000000100000000010101010000000
- 0x0080 2A8A 00000000100000000010101010001010

The following steps must all occur in a very tight window to see the issue:

1. The DMA writes to Cache Line E. This means that it is not necessarily the same exact address, but within the same 64B cache line.
   - As a result, a snoop- write request is generated.
2. The DMA writes to Cache Line B. This means that it is not necessarily the same exact address, but within the same 64B cache line.
   - As a result, a snoop-write request is generated but not immediately issued as it is blocked by the snoop-write issued in the previous Step 1.
   - Once the snoop-write from Step 1 is complete, this snoop-write is processed.
3. The CPU reads from any address in external, cacheable memory that is a set match to Cache Line B. This must also create an L2$ hit (referred to here as Cache Line D).
   - This results in a cache miss from the CPU and sends a read request to L2 cache for the line.
   - Assuming this was also mapped to the same way as Cache Line B, this results in a replacement of Cache Line B since it was clean in L1D$.
   - Note that there is no method to determine what particular way is used, so it is not possible to tell whether this replacement would actually happen for a particular operation. This is why only a set match is mentioned here.
4. In parallel (the same CPU cycle) with Step 3, the CPU reads from any address in L2 SRAM that is a set match to Cache Line A, mentioned in prerequisite Step 2 (referred to here as Cache Line C).
   - This results in a cache miss from the CPU and sends a read request to L2 SRAM for the line.

- Assuming this was also mapped to the same way as Cache Line A, this results in a replacement of Cache Line A if it was clean in L1D$. If Cache Line A was dirty, an eviction would occur before the allocation completed.
- Note that there is no method to determine what particular way is used, so it is not possible to tell whether this replacement would actually happen for a particular operation. This is why only a set match is mentioned here.

The results of the above cause the following:

(A) The snoop-write to Cache Line E, from Step 1 above, is now in process and blocking the snoop-write to Cache Line B from Step 2.

(B) While Step A is going on, Cache Line A has either now been evicted and/or replaced by Cache Line C from Step 4 above and Cache Line B (the intended target of the delayed snoop-write) is now replaced with Cache Line D from Step 3 above.

(C) Once the first snoop-write from operation C1 completes, the second (delayed) snoop-write mentioned in Step A to Cache Line B should be killed since Cache Line B was replaced in the operation in Step B. Instead, it is not killed and the line cached (which is now actually Cache Line D) is now updated incorrectly.

As a result, the following is true:

1. Cache Line D now holds data that was corrupted by the operation in Step C above (as a result of Step 2 above).
   - A subsequent read of this data returns a corrupted value.
   - Subsequent writes to this cache line also cause the corrupted values to be committed to lower levels of memory.

Figure 13 shows the sequence of events.

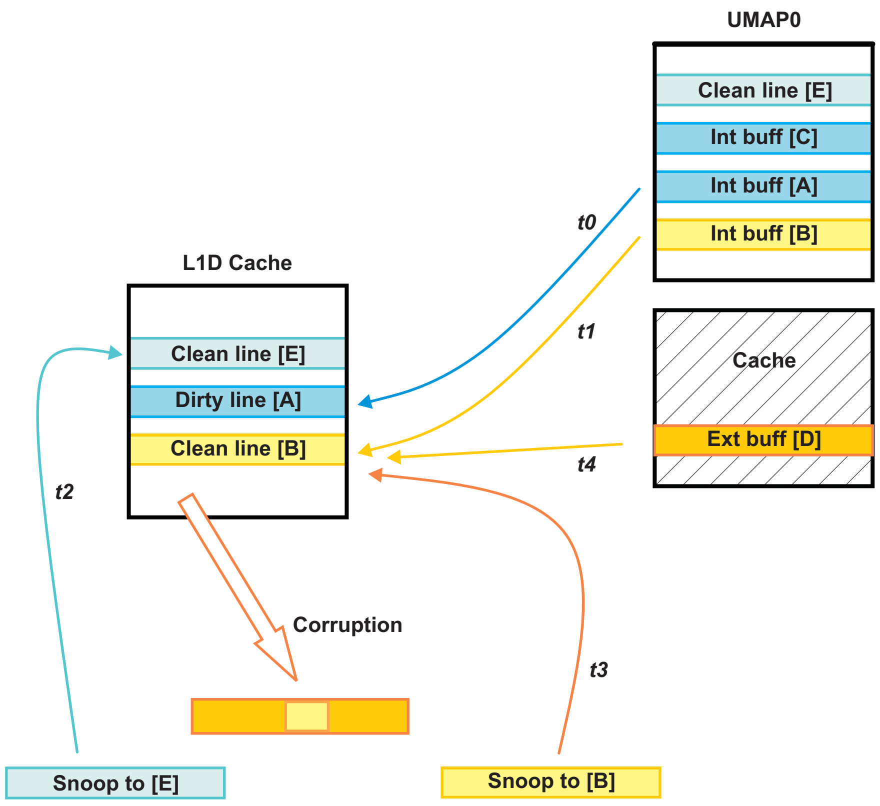


Figure 13. Sequence of Events

Workaround(s): A compiler flag (--c64p_dma_l1d_workaround) has been added to the latest Code Generation Tools to resolve this potential issue. This flag can be utilized for all code in the system or used on particular files/functions that may be susceptible to the conditions listed in this advisory.

Advisory 3.1.33 *Error Detection and Correction Incorrectly Reporting Error*

Revision(s) Affected: 3.1 and earlier

Details: The C64x+ Megamodule L2 Memory Controller provides support for error detection and correction (EDC). The primary purpose of this is to protect code and largely static data held in L2 memory. Because the likelihood of a bit error on a given bit is proportional to the time since it was last written, and program images are rarely written, the focus of EDC is on those portions of L2 that are written rarely but must be correct when read.

The EDC implements a distance-3 "detect 2, correct 1" Hamming code. The L2 controller always performs a full Hamming code check on 256-bit reads, regardless of whether the fetch is from L1D controller, L1P controller, IDMA, or DMA. There is a parity value associated with every 256 bits (32B) of L2 memory and a valid bit to qualify each parity value. EDC uses parity RAM to store this parity information. Parity is calculated and made valid in the parity RAM for following operations:

- 256 bits IDMA write
- 256 bits DMA writes through SDMA
- L2 cache allocate (both read and write allocate, except for the line to which the write allocate writes).

Parity is made invalid in the parity RAM for the following operations:

- DMA writes through SDMA ***or*** IDMA writes for less than 256 bits.
- All L1D writes to L2, either cache or SRAM.
- L1D writes that cause an L2 write allocate on the line that gets written (part of the L2 cache line).
- All L1D victims.

EDC configuration registers are available to enable EDC individually for each of the L2 memory pages. Status registers are also available to report the address that shows the EDC error as well as the type of the error, whether it is 1-bit error or multiple-bit error. It also indicates whether it is corrected or not.

Problem Symptoms:

EDC is reporting EDC error (parity error) even when there is no error present in L2 memory. The error is random and the status register reports either 1-bit or multiple-bit error. It is also not consistent that after some defined iterations EDC reports an error. The EDC error can occur at any time and at any location in the memory. The error is a false positive; i.e., there is actually no error present in the memory, but EDC reports an error. There are two dedicated events (event 116, corrected bit error, and event 117, uncorrected bit error) going from EDC to the megamodule INTC. If interrupt is enabled and configured for those events, then the CPU reports an EDC interrupt.

Problem Prerequisites:

The following two operations must happen in parallel for this error to occur:

- L2 block coherence operation (WB and WBInv Only)
- L1D victim generation.

When there is an L2 block coherence operation going on (it could be either L2_WB or L2_WBInv) and before that operation is complete, if the CPU does the operations that generates the L1D victims, then it is possible that the L1D victim operation will mark the parity valid bit to be 1, which is incorrect behavior. This can easily occur when there are interrupts happening during the L2 Block WriteBack (L2_WB) or L2 WriteBackInvalidate (L2_WBInv) operation. The error does not occur during block invalidate operation. As mentioned above, it is a random occurrence that the L1D victim could validate the parity and generate the EDC interrupt.

Correct Behavior:

- L2 coherence operation in progress

  and

- L1D victim generated
- L1D victims are not EDC protected and, so, the parity valid bit should get reset to 0 and junk should be written to parity RAM.

Incorrect Behavior:

- L2 coherence operation in progress

  and

- L1D victim generated
- L1D victims are not EDC protected but the parity valid bit is marked valid with no parity calculated and junk written to parity RAM.
- Any subsequent reads to this cache line cause the L2 EDC error. EDC protection is performed as per junk parity data on that cache sub-line (256 bits) and it can corrupt the data in that cache sub-line.

Workaround(s):

Workaround 1:

Disable interrupts during L2 block coherence operations. If there are large block coherence operations and disabling the interrupt during those coherence operations is not feasible, then divide the big coherence operation into multiple, small coherence operations and protect each of them against allowing interrupts during two coherence operations.

Workaround 2:

Allow interrupts, but put the L1D cache in freeze mode before starting L2 block coherence operation so that L1D victims are not generated during the L2 block coherence operation. Un-freeze the L1D cache as soon as the L2 block coherence operation is complete.

Advisory 3.1.34 *SRIO May Fail to Send Interrupt for Completed TX or RX Message*

Revision(s) Affected: 3.1 and earlier

Details: The interrupt clearing/setting mechanism for the RXU/TXU gives priority to clearing the interrupt rather than setting it. The sequence of the peripheral for handling buffer descriptors of a completed message is to: write the buffer descriptor info, set the ICSR interrupt bit, and, finally, write the completion pointer (CP). As software processes the buffer descriptors during an ISR, it ends the process by writing the CP register to indicate to the peripheral what was the last buffer descriptor processed. This clears the interrupt, if both peripheral and software are at the same point; i.e., the interrupt is not cleared and will fire again once the pacing register has completed its countdown.

Due to the implementation of the interrupt clearing/setting, where priority is given to clearing the interrupt, if software writes the CP (which the peripheral compares to it's CP and matches) causing the interrupt to be cleared on the same internal clock cycle as the peripheral trying to set the interrupt bit for the next buffer descriptor, the interrupt bit is cleared and the interrupt for that next packet is lost. Note that no data is actually lost, the interrupt simply does not occur. Once an additional message is processed and the descriptor is completed, the interrupt is fired as normal and all descriptors can be processed at that point. Although not guaranteed, it is possible for this missed interrupt condition to occur with every ISR that attempts to write the TX or RX CP. However, since the missed interrupt descriptor can be processed during the next interrupt ISR, the only concern is added latency. For systems with a steady flow of messages, this added latency is usually insignificant, but it is evident on scenarios where it occurs on the last buffer descriptor in a group of messages since nothing is behind it to cause another interrupt. For example, if the RX queue received 10 messages and the tenth interrupt is lost, and no other messages were ever routed to that same RX queue, it will never fire another interrupt.

Workaround: Change the ISR as shown in the following steps and in Figure 14. Every time an interrupt is received:

1. Determine that the interrupt is related to CPPI. If not, call another handler.
2. Fetch the next descriptor (software maintains a current pointer, SW_Pointer).
3. Check the ownership bit for this next descriptor:
   (a) If it is not owned by software, go to Step 6.
   (b) If it is owned by software, then check the “CC” code and perform the remaining packet processing.
   (c) If EOQ is reached, write the completion pointer and go to Step 8.
   (d) Otherwise, continue with Step 4.
4. Move the SW_Pointer to point to this next descriptor.
5. Go back to Step 3.
6. Write the completion pointer based on the current SW_Pointer value.
7. Check the ownership bit for the next descriptor again:
   (a) If it is owned by software, go to Step 3b.
   (b) Otherwise, continue with Step 8.
8. Write the interrupt pacing register to enable the next interrupt.

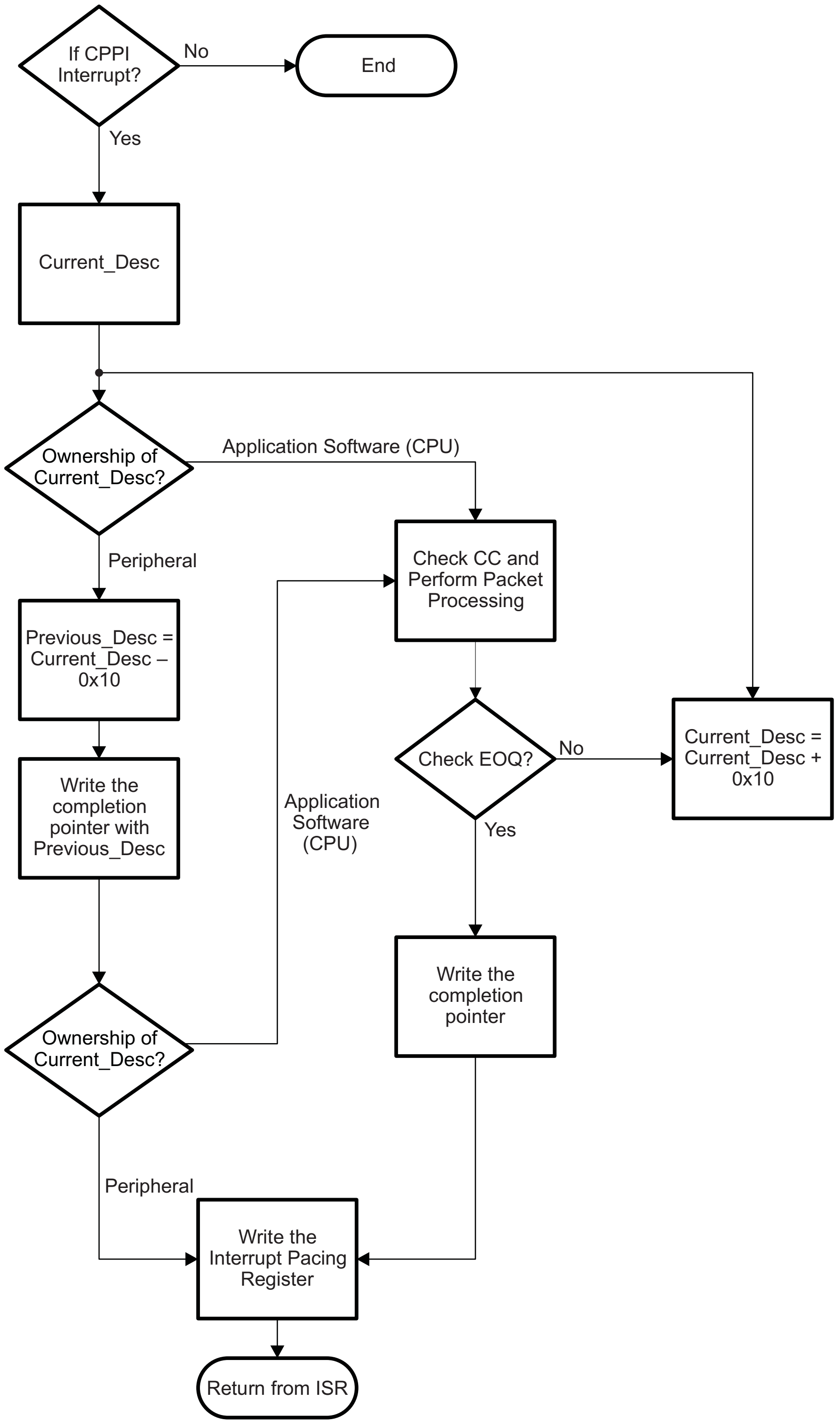

If CPPI Interrupt?
No
End
Yes
Current_Desc
Ownership of Current_Desc?
Application Software (CPU)
Check CC and Perform Packet Processing
Peripheral
Previous_Desc = Current_Desc – 0x10
Check EOQ?
No
Current_Desc = Current_Desc + 0x10
Write the completion pointer with Previous_Desc
Application Software (CPU)
Yes
Write the completion pointer
Ownership of Current_Desc?
Peripheral
Write the Interrupt Pacing Register
Return from ISR

Figure 14. ISR Workaround Flowchart

3 Silicon Revision 2.1 Usage Notes and Known Design Exceptions to Functional Specifications

This section describes the usage notes and advisories that apply to revision 2.1 of the TCI6482 device.

NOTE: No functional design changes were made from silicon revision 2.0 to silicon revision 2.1. All usage notes and advisories for TCI6482 silicon revision 2.0 also apply to silicon revision 2.1.

3.1 Usage Notes for Silicon Revision 2.1

Silicon revision 2.1 applicable usage notes have been found on later silicon revisions; for more detail, see Section 2.1, *Usage Notes for Silicon Revision 3.1*.

3.2 Silicon Revision 2.1 Known Design Exceptions to Functional Specifications

Table 15. Silicon Revision 2.1 Advisory List

Advisory 2.1.21 *DSP SDMA/IDMA: Unexpected Stalling of SDMA/IDMA Access to L2 SRAM*

Revision(s) Affected: 2.1 and earlier

Details:

NOTE: If DSP L2 memory is used only as cache OR if L2 RAM is **not** accessed by IDMA or via the SDMA interface during run-time, then this exception does not apply.

The C64x+ megamodule has a Master Direct Memory Access (MDMA) bus interface and a Slave Direct Memory Access (SDMA) bus interface. The MDMA interface provides DSP access to resources outside the C64x+ megamodule (i.e., DDR2, EMIFA, and PCI memory). The MDMA interface is typically used for CPU/cache accesses to memory beyond the level 2 (L2) memory level. These accesses include cache line allocates, write-backs, and non-cacheable loads and stores to/from system memories. The SDMA interface allows other master peripherals in the system to access level 1 data (L1D), level 1 program (L1P), and L2 RAM DSP memories. The masters allowed accesses to these memories are EDMA transfer controllers, HPI, PCI, EMAC, and SRIO. The DSP Internal Direct Memory Access (IDMA) is a C64x+ megamodule DMA engine used to move data between internal DSP memories (L1, L2) and/or the DSP peripheral configuration bus. The IDMA engine shares resources with the SDMA interface.

The C64x+ megamodule has an L1D cache and an L2 caches, both of which implement write-back data caches. The C64x+ megamodule holds updated values for external memory as long as possible. It writes these updated values, called *victims*, to external memory when it needs to make room for new data, when requested to do so by the application, or when a load is performed from a non-cacheable memory for which there is a set match in the cache (i.e., the non-cacheable line would replace a dirty line if cached). The L1D sends its victims to L2. The caching architecture has pipelining, meaning multiple requests could be pending between L1, L2, and MDMA. For more details on the C64x+ megamodule and its MDMA and SDMA ports, see the *TMS320C64x+ Megamodule Reference Guide* (literature number SPRU871).

Ideally, the MDMA (the blue lines in Figure 15) and SDMA/IDMA paths (the orange lines in Figure 15) operate independently with minimal interference. Normally, MDMA accesses may stall for extended periods of time (clock cycles) due to expected system level delays (e.g., bandwidth limitations, DDR2 memory refreshes). However, when using L2 as RAM, SDMA and/or IDMA accesses to L2/L1 may experience unexpected stalling in addition to the normal stalls seen by the MDMA interface. For latency-sensitive traffic, the SDMA stall can result in missing real-time deadlines. In a more severe case, the SDMA stall can produce a deadlock condition in the device.

NOTE: SDMA/IDMA accesses to L1P/D will not experience an unexpected stall if there are no SDMA/IDMA accesses to L2. Unexpected SDMA/IDMA stalls to L1 happen only when they are pipelined behind L2 accesses. Additionally, the deadlock scenario will be avoided if there are no SDMA accesses to L2.

Figure 15 is a simplified view for illustrative purposes only. The IDMA/SDMA path (orange lines) can also go to L1D/L1P memories and IDMA can go to the DSP CFG peripherals. MDMA transactions (blue lines) can also originate from L1P or L1D through the L2 controller or directly from the DSP.

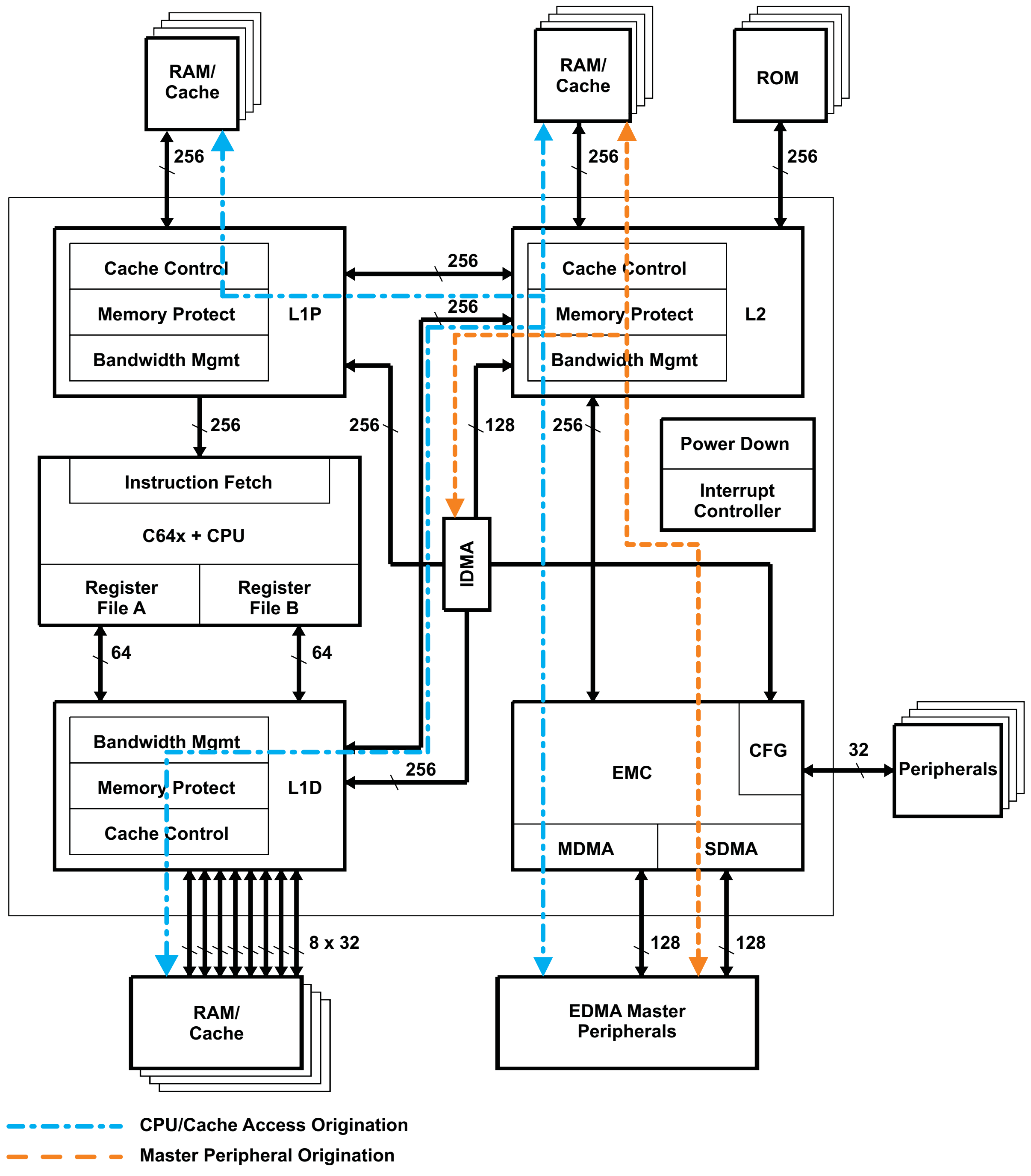


Figure 15. IDMA, SDMA, and MDMA Paths

SDMA/IDMA stalls may occur during the following scenarios. Each of these scenarios describes expected normal DSP functionality, but the SDMA/IDMA access potentially exhibits additional unexpected stalling.

1. Bursts of writes to non-cacheable MDMA space (i.e., DDR2, EMIFA, and PCI memory). The DSP buffers up to 4 non-cacheable writes. When this buffer fills, SDMA/IDMA is blocked until the buffer is no longer full. Therefore, bursts of non-cacheable writes longer than three writes can stall SDMA/IDMA traffic.
2. Various combinations of L1 and L2 cache activity:
   (a) L1D read miss generating victim traffic to L2 (cache or SRAM) or external memory. The SDMA/MDMA may be stalled while servicing the read miss and the victim. If the read miss also misses L2 cache, the SDMA/IDMA may be stalled until data is fetched from external memory to service the read miss. If the read access is to non-cacheable memory there will still potentially be an L1D victim generated even though the read data will not replace the line in the L1D cache.
   (b) L1D read request missing L2 (going external) while another L1D request is pending. The SDMA/IDMA may be stalled until the external memory access is complete.
   (c) L2 victim traffic to external memory during any pending L1D request. The SDMA/IDMA may be stalled until external memory access and the pending L1D request are complete.

The duration of the SDMA/IDMA stalls depends on the quantity/characteristics of the L1/L2 cache and the MDMA traffic in the system. In cases 2a, 2b, and 2c, stalling may or may not occur depending on the state of the cache request pipelines and the traffic target locations. These stalling mechanisms may also interact in various ways, causing longer stalls. Therefore, it is difficult to predict if stalling will occur and for how long.

SDMA/IDMA stalling and any system impact is most likely in systems with excessive context switching, L1/L2 cache miss/victim traffic, and heavily loaded EMIF.

Use the following steps to determine if SDMA/IDMA stalling is the cause of real-time deadline misses for existing applications. Situations where real-time deadlines may be missed include loss of McBSP samples and low peripheral throughput.

1. Determine if the transfer missing the real-time deadline is accessing L2 or L1D memory. If not, then SDMA/IDMA stalling is not the source of the real-time deadline miss.
2. Identify all SDMA transfers to/from L2 memory (e.g., EDMA transfer to/from L2 from/to a McBSP or HPI block transfer to/from L2). If there are no SDMA transfers going to L2, then SDMA/IDMA stalling is not the source of the problem.
3. Redirect all SDMA transfers to L2 memory to other memories using one of the following methods:
   - Temporarily transfer all the L2 SDMA transfers to L1D SRAM.
   - If not all L2 SDMA transfers can be moved to L1D memory, temporarily direct some of the transfers to DDR memory and keep the rest in L1D memory. There should be ***no*** L2 SDMA transfers.
   - If neither of the above approaches are possible, move the transfer with the real-time deadline to the EMAC CPPI RAM. If the EMAC CPPI RAM is not big enough, a two-step mechanism can be used to page a small working buffer defined in the EMAC CPPI RAM into a bigger buffer in L2 SRAM. The EDMA module can be setup to automate this double buffering scheme without CPU intervention for moving data from the EMAC CPPI RAM. Some throughput degradation is expected when the buffers are moved to the EMAC CPPI RAM.

   Note: Note that EMAC CPPI RAM memory is word-addressable only and, therefore, must be accessed using an EDMA index of 4 bytes.

If real-time deadlines are still missed after implementing any of the options in Step 3, then SDMA/IDMA stalling is likely ***not*** the cause of the problem. If real-time deadline misses are solved using any of the options in Step 3, then SDMA/IDMA stalling ***is*** likely the source of the problem.

As previously mentioned, a possible deadlock scenario is introduced in the presence of the SDMA stalls just described. This scenario occurs for certain masters connected to the data SCR indirectly through a bridge [see the System Interconnect section of the *TMS320TCI6482 Communications Infrastructure Digital Signal Processor* data manual (literature number SPRS246)]. For TCI648x devices, the masters that are affected are the EMAC, HPI, PCI, VLYNQ, and SRIO . If the following sequence of events occurs, then a deadlock situation might arise:

1. One of the following two accesses occur:
   (a) SRIO issues a write command to the DSP's SDMA port followed by a subsequent write command to DDR2 (or EMIFA).
   (b) EMAC, HPI, PCI, or VLYNQ issues a write command to the DSP's SDMA port and the same or different master in this group issues a subsequent write command to DDR2 (or EMIFA).
2. If, at this time, the DSP's SDMA asserts itself not ready and unable to accept more write data and a cache line writeback from the DSP to DDR2 (or another slave, e.g., EMIFA) occurs.

In the above scenario it is possible for data phases from the write command issued to DDR2 (or EMIFA) to be stuck behind the data phases for the write to the DSP's SDMA in the SCR.

Therefore, if the DSP issues victim traffic to the same slave (DDR2 or EMIFA), then data associated with the victim traffic (#2) intended for DDR2 (or EMIFA) will be stuck behind write commands issued for #1. However, due to the MDMA/SDMA blocking issue, the SDMA traffic for #1 will be waiting for the MDMA traffic for #2 to finish, manifesting itself into a deadlock situation.

Workaround(s): **Method 1**

Entirely eliminate IDMA/SDMA stalling and the potential for a deadlock condition by removing all SDMA/IDMA accesses to L2 SRAM. For example, EMAC descriptors and EMAC payload cannot reside in L2. Master peripherals like the EDMA, PCI, HPI, and SRIO cannot access L2. There are no issues with the CPU itself accessing code/data in L2. This issue only pertains to SDMA/IDMA accesses to L2.

Method 2

Issues such as dropped McBSP samples can be worked around by moving latency-sensitive buffers outside the C64x+ megamodule. For example, rather than placing buffers for the McBSP into L1/L2, those buffers can instead be placed in other memory, such as the EMAC CPPI RAM.

Note: Note that EMAC CPPI RAM memory is word-addressable only and, therefore, must be accessed using an EDMA index of 4 bytes.

Method 3

To reduce the SDMA/IDMA stalling system impact, perform any of the following:

1. Improve system tolerance on DMA side (SDMA/IDMA/MDMA):
   - Understand and minimize latency-critical SDMA/IDMA accesses to L2 or L1P/D.
   - Directly reduce critical real-time deadlines, if possible, at peripheral/IO level (e.g., increase word size and/or reduce bit rates on serial ports).
   - To reduce DSP MDMA latency:
     – Increase the priority of the DSP access to DDR2/EMIFA such that MDMA latency of MDMA accesses causing stalls is minimized.

       Note: Other masters, such as HPI, may have real-time deadlines that dictate higher priority than the DSP.

     – Lower the PRIO_RAISE field setting in the DDR2 memory controller's burst priority register. Values ranging between 0x10 and 0x20 should give decent performance and minimize latency; lower values may cause excessive SDRAM row thrashing.

     – Do not perform EMIFA access using EMIFA ARDY handshaking during DSP run time. (Devices using ARDY potentially insert excessive latency to external memory accesses.)
2. Minimize offending scenarios on DSP/caching side:
   - If the DSP performing non-cacheable writes is causing the issue, insert protected non-cacheable reads (as shown in the last list item below) every few writes to allow the write buffer to empty.
   - Avoid caching from slow memories such as asynchronous memory. Instead, page the data via the EDMA from the off-chip async memory to L2 SRAM or SDRAM space before accessing the data from the DSP.

     Note: Paging cannot occur while real-time deadlines must be met.
   - Use explicit cache commands to trigger cache writebacks during appropriate times (L1D Writeback All, L2 Writeback All). ***Do not*** use these commands when real-time deadlines must be met.
   - Restructure program data and data flow to minimize the offending cache activity.
     – Define the read-only data as *const*. The const C keyword tells the compiler not to write to the array. By default, such arrays are allocated to the .const section as opposed to BSS. With a suitable linker command file, the developer can link the .const section off chip, while linking .bss on chip. Because programs initialize .bss at run time, this reduces the program's initialization time and total memory image.

     – Explicitly allocate lookup tables and writeable buffers to their own sections. The #pragma DATA_SECTION (label, *section*) directive tells the compiler to place a particular variable in the specified COFF section. The developer can

  explicitly control the layout of the program with this directive and an appropriate linker command file.
  - Avoid directly accessing data in slow memories (e.g., flash); copy at initialization time to faster memories.
- Modify troublesome code.
  - Rewrite using DMAs to minimize data cache writebacks. If the code accesses a large quantity of data externally, consider using DMAs to bring in the data, using double buffering and related techniques. This will minimize cache write-back traffic and the likelihood of SDMA/IDMA stalling.
  - Re-block the loops. In some cases, restructuring loops can increase reuse in the cache and reduce the total traffic to external memory.
  - Throttle the loops. If restructuring the code is impractical, then it is reasonable to slow it down. This reduces the likelihood that consecutive SDMA/IDMA blocks stack up in the cache request pipelines, resulting in a long stall.
- Protect non-cacheable reads from generating an SDMA stall by freezing the L1D cache during the non-cacheable read access(es). The following example code contains a function that protects non-cacheable reads, avoids blocking during the reads, and, therefore, avoids the deadlock state.

```
;; ======================================================================= ;;
;;  Long Distance Load Word                                                ;;
;;                                                                         ;;
;;  int long_dist_load_word(volatile int *addr)                            ;;
;;                                                                         ;;
;;  This function reads a single word from a remote location with the L1D  ;;
;;  cache frozen. This prevents L1D from sending victims in response to    ;;
;;  these reads, thus preventing the L1D victim lock from engaging for the ;;
;;  corresponding L1D set.                                                 ;;
;;                                                                         ;;
;;  The code below does the following:                                     ;;
;;                                                                         ;;
;;      1. Disable interrupts                                              ;;
;;      2. Freeze L1D                                                      ;;
;;      3. Load the requested word                                         ;;
;;      4. Unfreeze L1D                                                    ;;
;;      5. Restore interrupts                                              ;;
;;                                                                         ;;
;;  Interrupts are disabled while the cache is frozen to prevent affecting ;;
;;  the performance of interrupt handlers. Disabling interrupts during     ;;
;;  the long distance load does not greatly impact interrupt latency,      ;;
;;  because the CPU already cannot service interrupts when it's stalled by ;;
;;  the cache. This function adds a small amount of overhead (~20 cycles)  ;;
;;  to that operation.                                                     ;;
;;                                                                         ;;
;; ======================================================================= ;;

        .asg    0x01840044,     L1DCC           ; L1D Cache Control
        .global _long_dist_load_word
        .text
        .asmfunc
; int long_dist_load_word(volatile int *addr)
_long_dist_load_word:
        MVKL    L1DCC,          B4
        MVKH    L1DCC,          B4
||      DINT                                    ; Disable interrupts
||      MVK     1,              B5
        STW     B5,             *B4             ; \_ Freeze cache
        LDW     *B4,            B5              ; /
        NOP     4
        SHR     B5,     16,     B5              ; POPER -> OPER
||      LDW     *A4,            A4              ; read value remotely
        NOP     4
        STW     B5,             *B4             ; \_ Restore cache
        RET     B3
||      LDW     *B4,            B5              ; /
        NOP     4
        RINT                                    ; Restore interrupts
        .endasmfunc

;; ===================================================================== ;;
;;  End of file: ldld.asm ;;
;; ===================================================================== ;;
```

Perform one of the following to eliminate the potential for a deadlock condition:

- Force EMAC, HPI, PCI, VLYNQ, and SRIO to perform writes to either DSP memory space or DDR2 (or EMIFA) memory space, but not to both.
- Force the completion of pending EMAC, HPI, PCI, VLYNQ, and SRIO write commands to either DSP memory space or DDR2/EMIFA memory space before initiating writes to a different destination. Pending write commands from a particular master are forced to complete when the same master initiates a read from the same destination memory.

NOTE: In the case of EMAC, HPI, PCI, and VLYNQ as a group, all of these masters must only perform writes to either DSP memory or DDR2/EMIFA memory, but not both. For example, if EMAC writes to DSP memory and PCI writes to DDR2 memory, the potential for the deadlock condition is still present.

This issue has been fixed on silicon revision 3.1.

Advisory 2.1.27 *L2 Victim Traffic Due To L2 Block Writeback During Any Pending CPU Request*

Revision(s) Affected: 2.1 and earlier

Details: This advisory is an update to Advisory 2.1.21 in this document. Advisory 2.1.21 lists the following four blocking conditions to trigger an SDMA/IDMA stall:

1. Bursts of writes to non-cacheable locations.
2. L1D read miss generating victim traffic to L2 (cache or SRAM) or external memory.
3. L1D read request missing L2 (going external) while another L1D request is pending.
4. L2 victim traffic to external memory during any pending L1D request.

NOTE: Items 1, 2, 3, and 4 shown in the list above and in Table 16 below are actually labeled as 1, 2a, 2b, and 2c in Advisory 2.1.21.

This advisory covers one more blocking condition:

5. L2 victim traffic due to L2 block writeback during any pending CPU request.

For silicon revisions 1.1, 2.0, and 2.1 that contain the original SDMA/IDMA blocking errata, this is a fifth way to encounter the issue in addition to the previously communicated four errata conditions in Advisory 2.1.21.

No additional deadlock risk potential is created by the addition of the new condition to silicon revisions 1.1, 2.0, and 2.1 that currently contain the SDMA/IDMA blocking conditions 1-4. This means that this issue can lead to a deadlock in the same manner that the other four conditions can. On silicon revision 2.0, without the original stall conditions 1-4, this creates a deadlock condition that is identical to the previous revisions.

Table 16. Stall Conditions on Silicon Revisions

SILICON REVISIONS	STALL CONDITIONS					
	1	2	3	4	5	6
1.1	YES	YES	YES	YES	YES	NO
2.0	YES	YES	YES	YES	YES	NO
2.1	YES	YES	YES	YES	YES	NO

Under certain conditions, L2 victim traffic due to a block writeback can block SDMA/IDMA accesses to UMAP0 during CPU requests.

There are four transactions that must happen to cause an SDMA/IDMA to stall because of this condition:

1. L1D/L1P needs to create an L2$ hit. This happens as a result of one of the following:
   - An L1D victim (through L1D writeback or writeback-invalidate).
   - An L1D read+victim (through L1D read miss resulting in a writeback).
   - An L1D write miss (write-through to an uncached line).
   - An L1D read miss.
   - An L1P fetch miss.
2. A user-initiated L2 block writeback must occur involving the same cache set as the L1D/L1P cache accesses in the previous bullet.
3. An SDMA access to UMAP0.
4. The CPU also accesses the same cache set as the L1D/L1P cache accesses and the L2 block writeback as described in the first two bullets. This happens as a result of a CPU LDx/STx instruction that causes one of the following:
   - An L1D victim (through L1D writeback or writeback-invalidate).
   - An L1D write miss (write-through to an uncached line).
   - An L1D read miss.

- An L1P fetch miss.

As a result of the four items above, any further SDMAs to UMAP0 are blocked. Note that three of these items ***must*** involve the same L2$ set in order to see the issue and, thus, is not as likely as the other conditions listed in the original errata. The stall persists until the operations above are complete.

Workaround(s): **Workaround 1: Leave in previous SDMA/IDMA stall workarounds**

For silicon revisions 1.1, 2.0, and 2.1 that were already affected with the other four conditions of the SDMA/IDMA stall issue from Advisory 2.1.21, there is no additional workaround needed. If all of the deadlock avoidance steps listed in Advisory 2.1.21 have been followed, there is no risk for a deadlock because of this issue. Methods to reduce stalling due to this issue are also already covered in Advisory 2.1.21.

For silicon revision 2.0 that fixed the initial four conditions of SDMA/IDMA stall issue, the deadlock avoidance steps that are already listed in Advisory 2.1.21 for previous revisions of silicon should be followed to ensure that there is no chance of a deadlock. The workarounds to avoid stalls are also the same as communicated in previous revisions of the device with the issue.

Workaround 2: Do not use L2$

Systems that do not use L2$ are not affected by this issue.

This issue has been fixed on silicon revision 3.1.

Advisory 2.1.28 *Serial RapidIO Internal Digital Loopback is Not Always Stable*

Revision(s) Affected: 2.1 and earlier

Details: A digital loopback control function provides testability features with the ability to loop a port's transmit data back to the receive side. Digital loopback is controlled through bit 25 of the RIO_PER_SET_CNTL register. This single bit control affects every 1X port, or all lanes of a 4X port, depending on the supported mode of the device. This loopback is done in the digital logic domain and is before the SerDes. An issue was discovered where ports that are in digital loopback exhibit sporadic errors and are unreliable. In these instances, the ports are unable to maintain Port_ok status and may encounter multiple various error stopped states.

In digital loopback, the normal physical layer RX FIFO is bypassed altogether for data. The data is actually handed from TX to RX via a separate path. This handoff is being performed correctly, however, the RX FIFO sideband signals that indicate under/over run conditions are erroneously being evaluated by the digital logic, instead of being ignored. This means that the RX state machine continues acting upon the under/over run signals that can be affected by external signals or even noise coming in on the device pins. For example, if the SerDes device pins are connected to a link partner's active transmitter, the port is not able to remain initialized in loopback since the under/over run signals are following the link traffic. Unreliable digital loopback has also been observed without an active transmitting device attached.

Workaround: Avoid using the digital loopback mode. TX-to-RX loopback is also supported within the SerDes macros themselves. This internal SerDes loopback mode incorporates the complete RapidIO data path (including the RX FIFO) and eliminates the above mentioned issue. SerDes loopback is very stable and can be enabled with the following bits in the RapidIO SerDes registers:

RIO_SERDES_CFG1_CNTL[7:6] = 0b10
RIO_SERDES_CFGRXn_CNTL[1] = 0b1
RIO_SERDES_CFGTXn_CNTL[1] = 0b1

Note that loopback needs to be individually enabled for each port, or each lane of a 4X port, by setting bit 1 of the appropriate RIO_SERDES_CFGRXn_CNTL and RIO_SERDES_CFGTXn_CNTL register.

4 Silicon Revision 2.0 Usage Notes and Known Design Exceptions to Functional Specifications

NOTE: All usage notes and advisories for TCI6482 silicon revision 2.0 also apply to silicon revision 2.1. For details, see Section 3, *Revision 2.1 Usage Notes and Known Design Exceptions to Functional Specifications*

5 Silicon Revision 1.1 Usage Notes and Known Design Exceptions to Functional Specifications

This section describes the usage notes and advisories that apply to revision 1.1 of the TCI6482 device.

5.1 Usage Notes for Silicon Revision 1.1

Some silicon revision 1.1 applicable usage notes have been found on later silicon revisions; for more detail, see Section 2.1, *Usage Notes for Silicon Revision 3.1*.

5.1.1 EMAC: RMII Reference Clock Will Be Changed to Input on Silicon Revision 2.0 and Later

Due to Advisory 1.1.5, *EMAC: RMII Cannot Be Used to Talk to a Switch*, the RMII reference clock (RMREFCLK) will be changed from an output pin to an input pin on silicon revision 2.0 and later. This change has also been reflected in the *TMS320TCI6482 Communications Infrastructure Digital Signal Processor* data manual (literature number SPRS246).

5.2 Silicon Revision 1.1 Known Design Exceptions to Functional Specifications

Table 17. Silicon Revision 1.1 Advisory List

Advisory 1.1.0 — *Reset: $\overline{\text{RESETSTAT}}$ Pin Not Active When $\overline{\text{POR}}$ Pin is Active*

Revision(s) Affected: 1.1

Details: The $\overline{\text{RESETSTAT}}$ pin is used by the DSP to signal the end of a reset sequence, including power-on reset. For more details on the TCI6482 reset sequences, see the *TMS320TCI6482 Communications Infrastructure Digital Signal Processor* data manual (literature number SPRS246) .

During device power-up, the $\overline{\text{RESETSTAT}}$ pin will be in a high-impedance state; therefore, the $\overline{\text{RESETSTAT}}$ pin should **not** be polled to determine when the power-on reset sequence completes.

Workaround(s): After the $\overline{\text{POR}}$ pin is brought high, wait 15,000 CLKIN1 cycles before polling the $\overline{\text{RESETSTAT}}$ pin to determine if the DSP power-on reset sequence is complete.

Internal Tracking Number: 2

Advisory 1.1.1 — *Serial RapidIO: Master Enable Bit Does Not Gate Out-Going Requests*

Revision(s) Affected: 1.1

Details: The Master Enable bit in the RapidIO Port General Control CSR Register (RIO_SP_GEN_CTL - address 02D0 113C) should control whether a device is allowed to issue requests into the system.

On the TCI6482 device, the Master Enable bit does not control whether out-going requests can be sent by the DSP.

Workaround(s): The CPU **must** poll the Master Enable bit until it becomes "1" before enabling out-going requests such as Direct I/O, Maintenance, Doorbell, or Message Passing.

Internal Tracking Number: 22

Advisory 1.1.2 *EMIFA: Internal Clock Source Cannot Be Divided by a Number Greater Than Eight*

Revision(s) Affected: 1.1

Details: The PLL1 controller SYSCLK4 signal is used as the EMIFA internal clock. The frequency of SYSCLK4 is controlled by divider D4. The RATIO field of the PLL1 Controller Divider 4 Register (PLLDIV4) is used to specify the divide-by value of divider D4. For more information on the PLL1 controller, see the PLL1 Controller section of the *TMS320TCI6482 Communications Infrastructure Digital Signal Processor* data manual (literature number SPRS246 revision B and later) .

On the TCI6482 device, the RATIO field can only be set to a value of 000b (/1) through 111b (/8). This means SYSREFCLK (used as CPU clock) cannot be divided by a value greater than eight. When the device is configured such that EMIFA uses an internal clock, care must be taken so that the maximum clock speed of EMIFA is not violated.

Workaround(s):

1. **Do not** use SYSCLK4 to clock EMIFA. Instead, provide an external clock source for EMIFA through the AECLKIN pin by pulling the AEA15 pin low during reset.
2. Slow down the frequency of SYSREFCLK so that SYSCLK4 ***does not*** run above the maximum clock speed of EMIFA. Note: slowing down SYSREFCLK directly affects the CPU clock speed.

Internal Tracking Number: 7

Advisory 1.1.3 *Reset: Internal Pullup and Pulldown Resistors Disabled During $\overline{\text{POR}}$ Low Pulse*

Revision(s) Affected: 1.1

Details: A number of pins, including configuration pins, on the TMS320TCI6482 device include internal pullup/pulldown (IPU/IPD) resistors. These IPU/IPD resistors define the state of the pins while neither the DSP nor an external force is driving them.

During device power-up, the TCI6482 device requires that the $\overline{\text{POR}}$ pin be low. Once the device's power supplies are stable, the $\overline{\text{POR}}$ pin can be brought high to bring the device out of reset. When the $\overline{\text{POR}}$ pin transitions from low to high, the device configuration pins are latched. For more details on IPU/IPD resistors, power-up, and reset sequences, see the *TMS320TCI6482 Communications Infrastructure Digital Signal Processor* data manual (literature number SPRS246) .

On the TCI6482 device, the IPU/IPD resistors are disabled whenever the $\overline{\text{POR}}$ pin is low. As most pins will be in a high-impedance state until the $\overline{\text{POR}}$ pin goes high, the IPU/IPD resistors should not be relied upon to set the state of the device pins while $\overline{\text{POR}}$ is low; this is especially important for device configuration pins. Also, since a large number of pins are floating, i.e., in a high-impedance state, the DV_{DD33} supply draws a large amount of current (up to 500 mA).

Workaround(s): To ensure the configuration of the device following power-on reset, attach external pullup and pulldown resistors to the device configuration pins.

Note: if the end application requires that other DSP pins have a valid state during power-up, then an **external** pullup or pulldown resistor **must** be used.

Internal Tracking Number: 1

Advisory 1.1.4 *EMAC: RGMII Cannot be Used to Communicate With Devices that Do Not Use In-Band Signaling*

Revision(s) Affected: 1.1

Details: The Reduced Gigabit Media Independent Interface (RGMII) industry standard (version 2.0) has a list of in-band signaling features that are required and some that are optional. The optional features can be used to set the link rate, duplex mode, and also to carry the link status. The EMAC on the TCI6482 DSP always uses these optional features, making them a requirement. This means that the EMAC cannot communicate with PHYs or switches that do not support these optional features.

Workaround(s): None.

Internal Tracking Number: 12

Advisory 1.1.5 *EMAC: RMII Cannot be Used to Talk to a Switch*

Revision(s) Affected: 1.1

Details: The RMII reference clock output, RMREFCLK, on the TCI6482 DSP may be used to talk to a PHY, but it is highly likely that this will not work with a switch. In a system with multiple TCI6482 DSPs, each DSP would have a RMREFCLK output and there is a slim chance these clocks will be aligned. Furthermore, a switch would only have a single reference clock input.

Note that the clock used by the PHY or switch cannot be fed into CLKIN2 such that all devices use a common clock. This is because the CLKIN2 input is passed through the DSP's PLL2, which changes the alignment of the clock with respect to the original clock.

Workaround(s): There is no workaround for this advisory; however, on silicon revision 2.0 and later, the RMREFCLK pin will be changed to an input such that this issue can be avoided.

Internal Tracking Number: 13

Advisory 1.1.6 *Bootloader: Serial RapidIO Configurations 1, 2, and 3 Cannot be Used for Booting*

Revision(s) Affected: 1.1

Details: Serial RapidIO boot is selected when BOOTMODE[3:0] = 1000b through 1011b. BOOTMODE[1:0] selects one of four boot configurations of the Serial RapidIO peripheral; i.e., "00b" refers to Serial RapidIO Boot Configuration 0, "01b" refers to Serial RapidIO Boot Configuration 1, etc.

However, the SerDes 1, 2, and 3 configuration registers are not set up properly for Serial RapidIO Boot Configurations 1, 2, and 3. Therefore, Serial RapidIO Configurations 1, 2, and 3 cannot be used for booting. Only Serial RapidIO Boot Configuration 0 can be used for Serial RapidIO boot; this configuration utilizes SerDes 0 in 1X mode.

Workaround(s): Use Serial RapidIO Configuration 0 to boot a second-level bootloader. The second-level bootloader can be used to correctly set up the SerDes 1, 2, and 3 and force re-initialization of the link to come up in 4X mode.

Internal Tracking Number: 8

Advisory 1.1.7 *EDMA: Lower Priority Queue Does Not Get Serviced When Higher Priority Queue has Pending Event*

Revision(s) Affected: 1.1

Details: The TCI6482 DSP EDMA channel controller contains four event queues (Q0, Q1, Q2, and Q3). By default, each queue maps directly to a corresponding transfer controller (TC), i.e., Q0 maps to TC0, Q1 maps to TC1, etc.

If events are pending in more than one queue and the corresponding TCs are "available," then the lowest numbered queue is processed first; i.e., if all four queues have an event pending and all the corresponding TCs are available, Q0 will be processed first.

If Q0 and Q1 both have pending events and TC0 is busy (meaning not ready to accept a new TR) and TC1 is available then it should be possible for Q1 to be serviced resulting in a transfer request (TR) submission to TC1. This maximizes overall performance by making sure the TCs are kept busy as much as possible with TRs.

The issue on the TCI6482 DSP (Rev 1.1) is that a lower priority queue is not serviced as long as a higher priority queue has pending events even if the TC associated with the higher priority queue is busy. If Q0 and Q1 both have pending events, TC0 is busy and TC1 is available, a TR will not be submitted to TC1. Q1 is not processed as long as Q0 has events pending.

This results in the low priority TCs going unused (thereby delaying the submission of TRs on these queues) until the events in all queues that are higher priority are submitted to their corresponding TCs.

Workaround(s):

1. The destination queue (and hence, the TC used) for an event can be selected through EDMA registers. When selecting the destination queue for an event, care must be taken to ensure that traffic on higher priority queues is kept to a minimum (both in event frequency and in transfer duration) such that lower priority queues can be serviced.

   When selecting a queue for an event, keep in mind that each TC has access to only specific DSP modules. Table 18 shows which modules are accessible by each TC.

Table 18. TC Connection Matrix

	TCP2	VCP2	McBSPs	UTOPIA2	Configuration Crossbar	VLYNQ	PCI	DDR2 Memory Controller	EMIFA	C64X+ Megamodule
TC0	Y	Y	N	N	N	N	N	Y	Y	Y
TC1	N	N	Y	Y	Y	Y	Y	Y	Y	Y
TC2	N	N	N	N	N	Y	Y	Y	Y	Y
TC3	N	N	N	N	N	Y	Y	Y	Y	Y

2. By default, each queue maps directly to a corresponding transfer controller (TC); i.e., Q0 maps to TC0, Q1 maps to TC1, etc. If there is a real-time requirement for servicing a peripheral which cannot be accessed via TC0, then Q0 can be mapped to the TC through which the peripheral can be accessed. Queue to TC mapping can be changed via software through the QUETCMAP register (see Figure 16 and Table 19) as long as only one queue maps to one TC.

   For example, abiding by the default settings, McBSP events need to be submitted on Q1 since TC1 is the only TC with connectivity to this peripheral. McBSP transfers might be at risk if there are events pending on Q0. In order to reduce this risk, the default mapping of Q0 can be changed such that it maps to TC1. In this manner, an event on Q0 results in a TR submission to TC1 thus minimizing the risk of a real-time miss due to this issue.

Opting for this workaround has other implications on the system. For example, if Q0 is mapped to TC1 then another queue must be mapped to TC0. If Q1 is mapped to TC0, the VCP2 and TCP2 events must be submitted on Q1. Whether this is acceptable would need to be determined by the user.

31–16
Reserved
R-0

15	14–12	11	10–8	7	6–4	3	2–0
Rsvd	TCNUMQ3	Rsvd	TCNUMQ2	Rsvd	TCNUMQ1	Rsvd	TCNUMQ0
R-0	R/W-3	R-0	R/W-2	R-0	R/W-1	R-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Figure 16. QUETCMAP Register (02A0 0280h)

Table 19. QUETCMAP Register Field Descriptions

BIT	FIELD	VALUE	DESCRIPTION
31:15	Reserved		
15:0	TCNUMQ*n*		TC Number for Queue *n* Defines the TC number to which Event Queue *n* TRs are submitted.

Internal Tracking Number: 15

Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.