TMS320C6455/54
Digital Signal Processor
Silicon Revisions 2.0, 1.1

Silicon Errata

Literature Number: SPRZ234I
December 2005–Revised May 2008

Contents

List of Figures

List of Tables

TMS320C6455/54 Digital Signal Processor Silicon Revisions 2.0, 1.1

1 Introduction

This document describes the known exceptions to the functional specifications for the TMS320C6455/54 digital signal processors. [See the *TMS320C6455 Fixed-Point Digital Signal Processor* data manual (literature number SPRS276) and the *TMS320C6454 Fixed-Point Digital Signal Processor* data manual (literature number SPRS311).]

For additional information, see the latest version of the *TMS320C6000 DSP Peripherals Overview Reference Guide* (literature number SPRU190).

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 sheet), 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., TMS320C6455ZTZ). 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:

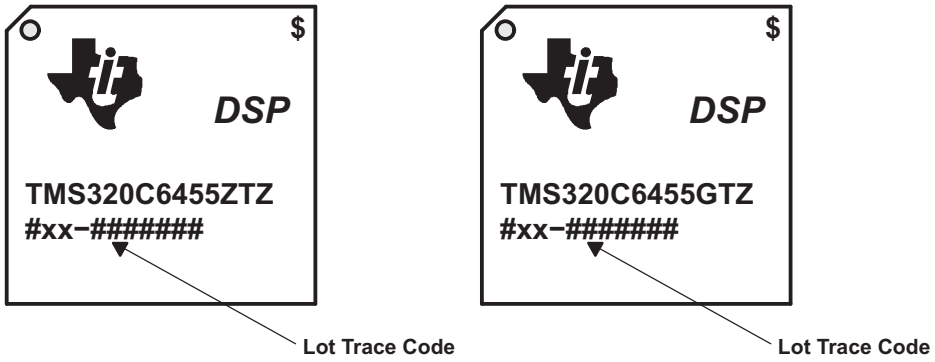
"Developmental product is intended for internal evaluation purposes."

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 examples of C6455 device package symbolization. The package symbolization for the C6454 device is similar, the only difference is the part number. 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 ZTZ and GTZ 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 TMS320C6455/54 (ZTZ and GTZ 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. In addition, orderable part numbers reflect the silicon revision. For example, silicon revision 2.0 has a new orderable part number, TMS320C6455**B**ZTZ, while orderable part number TMS320C6455ZTZ indicates silicon revision 1.1. Table 1 lists the silicon revisions associated with each lot trace code for the C6455/54 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 *TMS320C6455 Fixed-Point Digital Signal Processor* data manual (literature number SPRS276) and the *TMS320C6454 Fixed-Point Digital Signal Processor* data manual (literature number SPRS311).

Table 1. Lot Trace Codes

LOT TRACE CODE (xx)	SILICON REVISION	COMMENTS
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
7	720 MHz	Commercial

Table 3. Silicon Revision Variables

SILICON REVISION	CPU REVISION	C64X+ MEGAMODULE REVISION	JTAG ID REGISTER VALUE
2.0	1.0 (REVISION_ID = 0h)	Rev. 1 (MM_REVID[REVISION] = 1h)	0x1008 A02Fh (VARIANT = 1000b)
1.1	1.0 (REVISION_ID = 0h)	Rev. 0 (MM_REVID[REVISION] = 0h)	0x0008 A02Fh (VARIANT = 0000b)

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

This section describes the usage notes and advisories that apply to revision 2.0 of the C6455 and C6454 devices.

Note: The peripherals supported on the C6454 device are different from those supported on the C6455 device. Usage notes and advisories pertaining to SRIO, VCP2, TCP2, and UTOPIA do not apply to the C6454 device. For a complete list of the supported features of the C6454 and C6455 devices, see the device-specific data manuals.

2.1 Usage Notes for Silicon Revision 2.0

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 sheet), 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

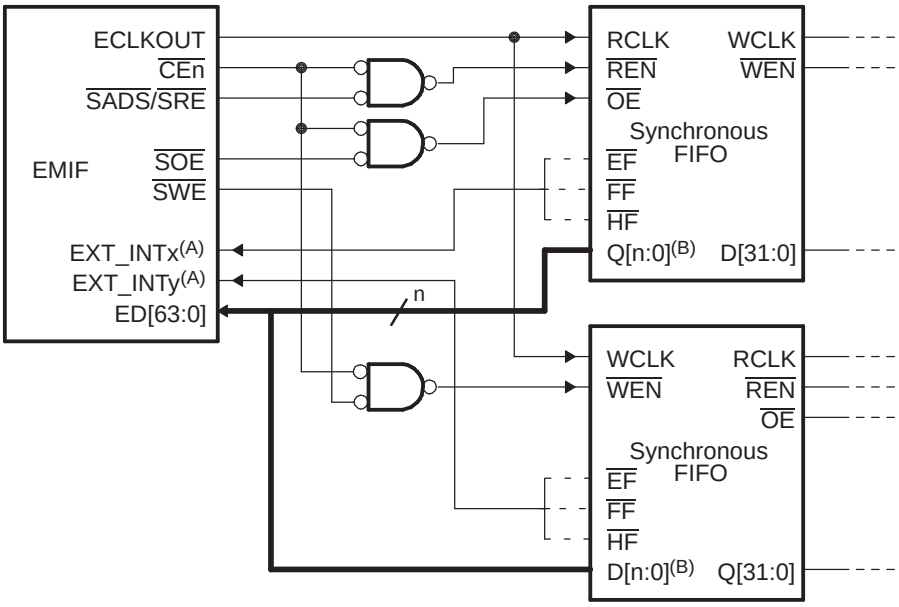
The chip enable pin of the DDR2 memory controller, $\overline{\text{DCE0}}$, is used to enable the DDR2 SDRAM memory device during external memory accesses. Currently, TI documentation shows that the $\overline{\text{DCE0}}$ pin is goes low to enable the DDR2 SDRAM memory device during external memory accesses and then goes high at the end of the access. However, on the TMS320C6455/54 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 ($\overline{CE}$[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 C645x devices may be configured as external interrupt sources to the CPU. For more details, see the *TMS320C645x DSP General-Purpose Input/Output (GPIO) User's Guide* (literature number SPRU724).

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 *TMS320C645x DSP Enhanced DMA (EDMA3) Controller User's Guide* (literature number SPRU966), 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 (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 C6455/54.

Attempts to perform reads from the EMIFA with FIFO mode (SAM == FIFO) on C6455/54 revision 1.1 will result in reading invalid data.

In order to avoid the issues described above with reads/writes to EMIFA when EDMA is configured for FIFO mode, it is required 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.

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

Two approaches are presented below for interfacing with FIFOs using the EMIFA. Each approach describes the necessary EMIFA connection to the FIFO and the configuration of the EDMA.

In general, these approaches are numbered in order of preference. 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 $\times$ 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.

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 Can Be Used to Lower Case Temperature and Power Consumption

The latest power data for the TMS320C6455/54 devices 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 TMS320C6455/54 devices, see the *TMS320C6455/54 Power Consumption Summary* (literature number SPRAAE8) 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 C6455/54 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 C6455/54 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 or UTOPIA

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

Note that the PCI peripheral consists 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 C6455/54 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. C6455/54 devices will not be damaged by tying the RSV32 and RSV34 pins to V_{SS}. New C6455/54 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 C6455/54 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 512 MB in 32-bit Mode

The maximum addressable memory has been increased from 256 MB to 512 MB in 32-bit mode and from 128 MB to 256 MB in 16-bit mode. Revision G of the *TMS320C6455 Fixed-Point Digital Signal Processor* data manual (literature number SPRS276) will be updated to show this change. Revision B of the *TMS320C645x DSP DDR2 Memory Controller User's Guide* (literature number SPRU970) 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.

2.2 Silicon Revision 2.0 Known Design Exceptions to Functional Specifications

Table 4. Silicon Revision 2.0 Advisory List

Advisory 2.0.1 *VCP2: Specific Parameter Combinations Generate Incorrect Results*

Revision(s) Affected: 2.0 and earlier

Details: Using the following parameter combinations with VCP2 and a constraint length of K = 8 will generate incorrect results on the C6455 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 2.0.2 *VCP2 and TCP2: Emulator Access to TCP2 and VCP2 Registers Through EDMA Bus Return Incorrect Results*

Revision(s) Affected: 2.0 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 2.0.3 *EMAC: RMII Interface Cannot be Used in Half-Duplex Mode*

Revision(s) Affected: All

Details: The RMII Ethernet MAC interface on C6455/54 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 C6455/54 given that the system negotiates to full-duplex mode immediately and a reset to the C6455/54 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, but will be added for future revisions.

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

Revision(s) Affected: 2.0 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 2.0.5 *EMAC: Signal Transitions on RMRXER are Ignored for Least Significant Di-Bit*

Revision(s) Affected: 2.0 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 2.0.6 *EMAC: RMCRSDV Not Being Passed Asynchronously to the EMAC*

Revision(s) Affected: 2.0 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 2.0.7 *EMU: Emulation Prone to Failure Under Certain Situations*

Revision(s) Affected: 2.0 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.

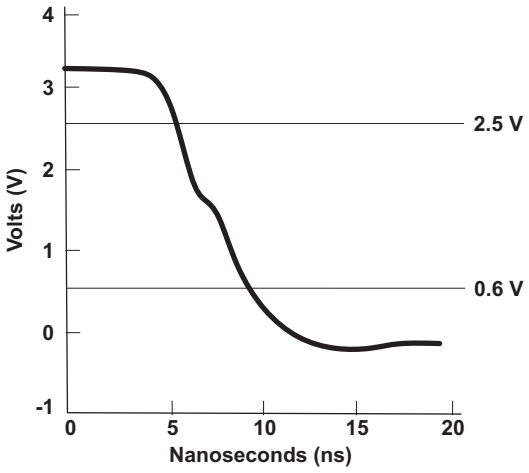


Figure 3. Bad TCK Transition

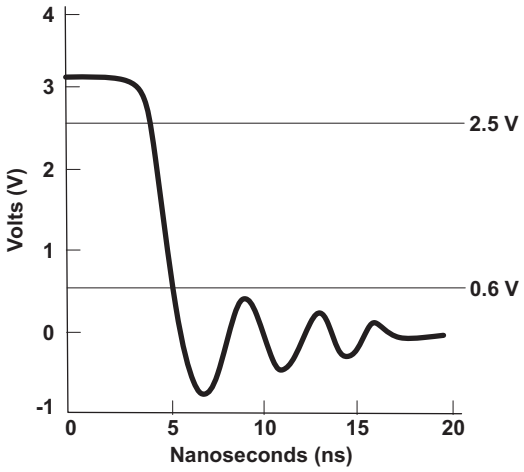


Figure 4. Good TCK Transition

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 2.0.8 *McBSP: Emulation Access to McBSP Registers May Cause Sample Loss*

Revision(s) Affected: 2.0 and earlier

Details: The McBSP registers on the C6455/54 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 2.0.9 *SRIO: Using NREAD to Read Invalid Memory Space Causes a Timeout and Halts the Port that Processed the NREAD Request*

Revision(s) Affected: 2.0 and earlier

Details: On C6455 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 future silicon revisions, the behavior of the device will be changed such that the SRIO module will ignore the DMA bus error message and send a valid response packet with a payload of garbage data. The outbound portion of the SRIO port will also not be halted. Therefore, if a master node uses NREAD to access invalid or reserved memory space, the master node will receive a packet with a garbage payload. If the master node is another DSP, the DSP will receive the packet and store it into receive memory; the completion code of LSUn_REG6 will be set to 000b (transaction complete, no errors).

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

Advisory 2.0.10 *L1D Cache: C64x+ L1D Cache May Lose Data or Hang DMA Operations Under Certain Conditions*

Revision(s) Affected: 2.0 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 2.0.11 *McBSP: Transfers Less than 32 Bits are Ignored in Some Cases When Device is Configured for Big-Endian Mode*

Revision(s) Affected: 2.0 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 C6455/54 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 2.0.12 *PCI: PCI Reset and Chip Reset Must Always Be Asserted Together*

Revision(s) Affected: 1.1 and 2.0

Details: The PCI module in the C6455/54 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, $\overline{\text{PRST}}$.

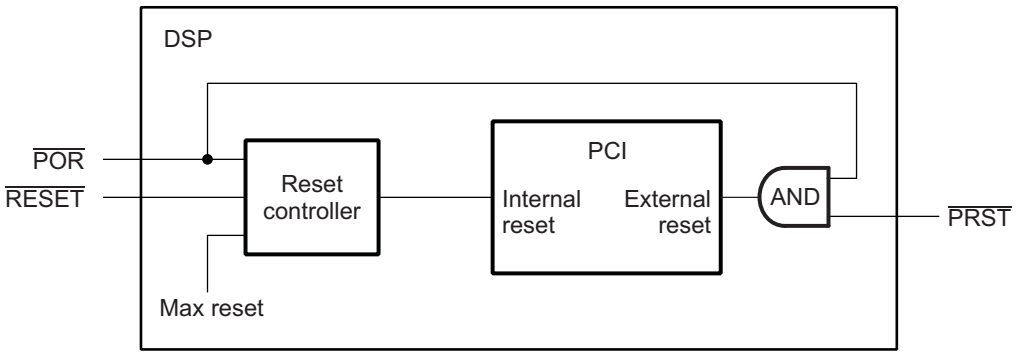


Figure 5. Internal Reset Asserted By Reset Controller

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 2.0.13 for issues relating to max reset. Also, see Advisory 2.0.14 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.

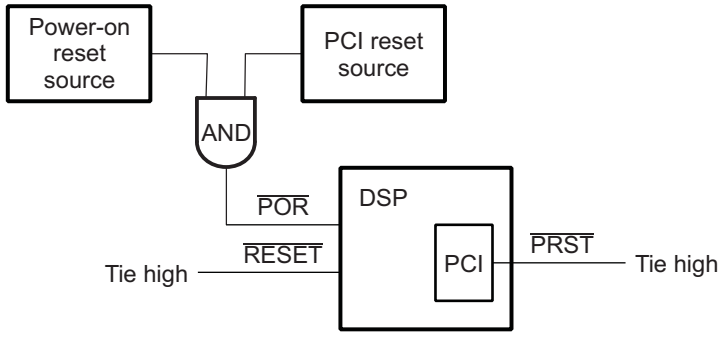


Figure 6. Example With $\overline{\text{RESET}}$ and $\overline{\text{PRST}}$ Pins Tied High

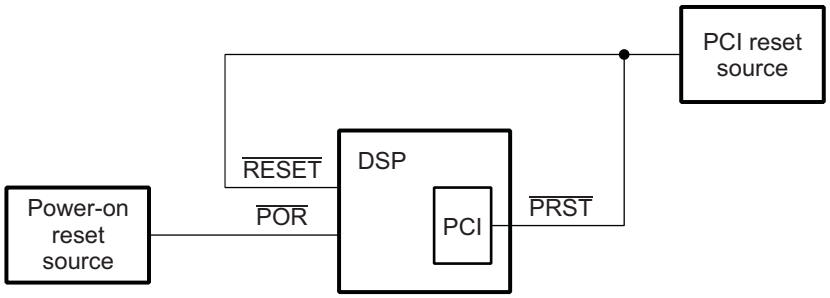


Figure 7. Example With PRST and RESET Pins Tied Together

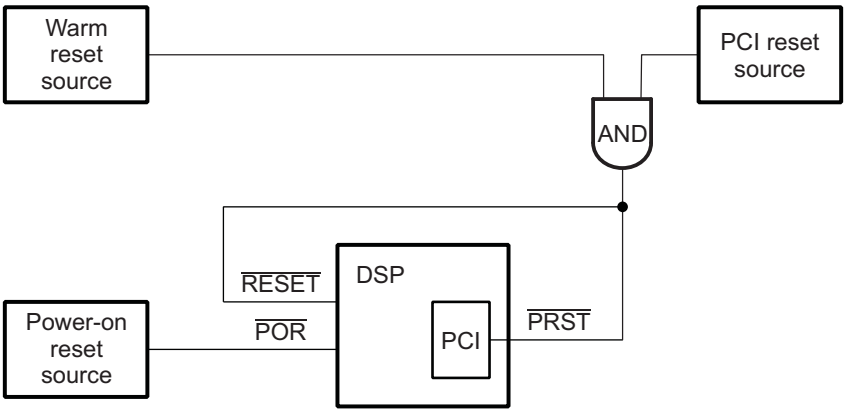


Figure 8. Example With Independent Power-On Reset, Warm Reset, and PCI Reset Sources

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 (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 2.0.13 *PCI: SRIO Max Reset Should Not Be Used When PCI is Used*

Revision(s) Affected: 1.1 and 2.0

Details: As described in Advisory 2.0.12, 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 2.0.14 *PCI: Device State Control Registers Should Not Be Used to Disable the PCI Once it is Enabled*

Revision(s) Affected: 1.1 and 2.0

Details: As described in Advisory 2.0.12, 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 2.0.15 *Chip: Writing to Certain Peripheral Memory-Mapped Registers Will Modify Value of PRI_ALLOC Register*

Revision(s) Affected: 1.1 and 2.0

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 bug 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 2.0.16 *SRIO: Performance Issues Identified Prohibiting Full Utilization of Pin Bandwidth*

Revision(s) Affected: 1.1 and 2.0

Details: On the C6455 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 5 illustrates the approximate internal utilization that can be achieved based on a 10-Gbps bus.

Table 5. 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 6 and Table 7 represent simulation data incorporating both the receive and transmit performance issues. The same performance levels were observed when measuring actual silicon.

Table 6. 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 7. 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 2.0.17 *CPU: Back-to-Back SPLOOPs With Interrupts Can Cause Incorrect Operation on C64x+ CPU*

Revision(s) Affected 2.0 and earlier

Details Back-to-back software pipeline loops (SPLOOPs) with interrupts can cause incorrect operation on the C64x+ CPU. This bug 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 bug, 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 2.0.18 *CPU: C64x+ CPU Incorrectly Generates False Exceptions for Multiple Writes*

Revision(s) Affected 2.0 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 2.0.19 *SRIO: Packet Forwarding Cannot Be Used With NREAD Response Packets Greater Than 16 Bytes*

Revision(s) Affected: 2.0 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.

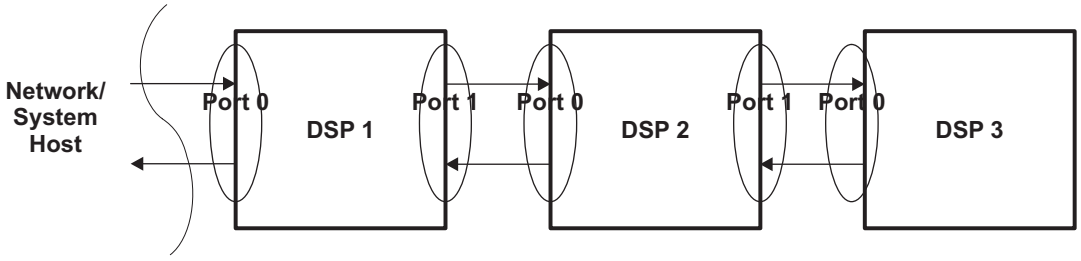


Figure 9. Daisy-Chain Example

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

Revision(s) Affected: 2.0 and earlier

Details:

As documented in the *TMS320C645x DSP Software-Programmable Phase-Locked Loop (PLL) Controller User's Guide* (literature number SPRUE56), 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 C6455/54 devices, 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 C6455/54 devices include two PLL controllers. Only divider D4 and divider D5 of PLL controller 1 and divider D1 of PLL controller 2 are programmable on these devices. 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 *TMS320C645x DSP Software-Programmable Phase-Locked Loop (PLL) Controller User's Guide* (literature number SPRUE56) 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 2.0.21 *DSP SDMA/IDMA: Unexpected Stalling of SDMA/IDMA Accesses to L2 SRAM and Potential Deadlock Condition*

Revision(s) Affected 2.0 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 or when requested to do so by the application. 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 10) and SDMA/IDMA paths (the orange lines in Figure 10) 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 10 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.

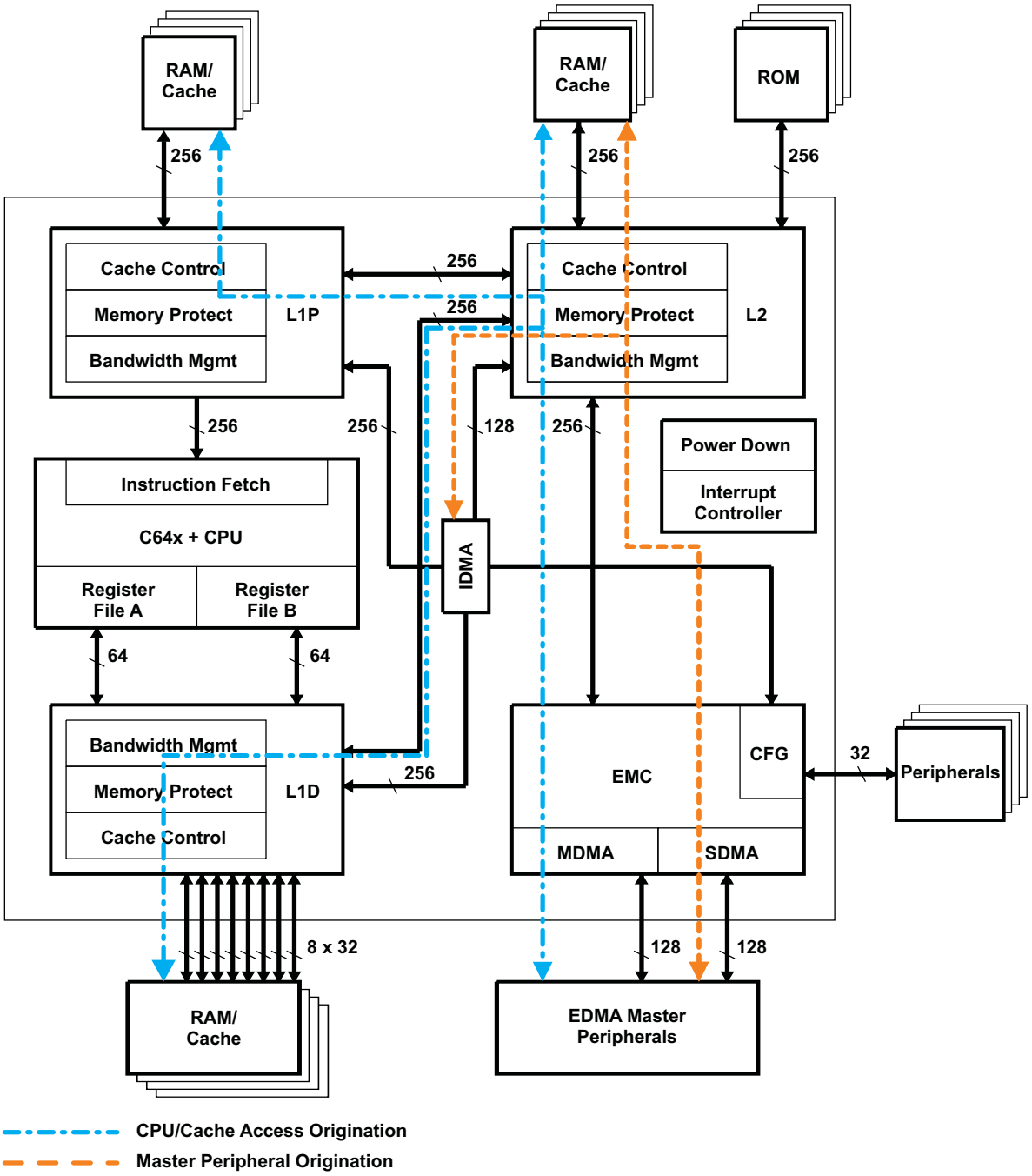


Figure 10. IDMA, SDMA, and MDMA Paths

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:
   - L1D read miss generating victim traffic to L2 (cache or SRAM) or external memory. The SDMA/IDMA 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.
   - 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.
   - 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 *TMS320C6455 Fixed-Point Digital Signal Processor* data manual (literature number SPRS276)]. For C645x devices, the masters that are affected are the EMAC, HPI, PCI, 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, or PCI 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 non-cacheable reads 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.

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

- Force EMAC, HPI, PCI, 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, 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, and PCI 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.

Advisory 2.0.22 *Potential SerDes Clocking Issue*

Revision(s) Affected: 2.0 and earlier

Details: A bug 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: The TCK pin should be externally pulled down with a 1-kΩ resistor.

3 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 C6455 and C6454 devices.

Note: The peripherals supported on the C6454 device are different from those supported on the C6455 device. Usage notes and advisories pertaining to SRIO, VCP2, TCP2, and UTOPIA do not apply to the C6454 device. For a complete list of the supported features of the C6454 and C6455 devices, see the device-specific data manuals.

3.1 Usage Notes for Silicon Revision 1.1

Some silicon revision 1.1 applicable usage notes have been found on a later silicon revision; for more detail, see the *Usage Notes for Silicon Revision 2.0* section of this document.

3.1.1 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 will also be reflected in the *TMS320C6455 Fixed-Point Digital Signal Processor* data manual (literature number SPRS276 revision B and later).

3.2 Silicon Revision 1.1 Known Design Exceptions to Functional Specifications

Table 8. Silicon Revision 1.1 Advisory List

Advisory 1.1.0 *Reset: RESETSTAT Pin Not Active When POR Pin is Active*

Revision(s) Affected: 1.1

Details: The RESETSTAT pin is used by the DSP to signal the end of a reset sequence, including power-on reset. For more details on the C6455/54 reset sequences, see the *TMS320C6455 Fixed-Point Digital Signal Processor* data manual (literature number SPRS276) and the *TMS320C6454 Fixed-Point Digital Signal Processor* data manual (literature number SPRS311).

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

Workaround: After the POR pin is brought high, wait 15,000 CLKIN1 cycles before polling the 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 C6455 device, the Master Enable bit does not control whether out-going requests can be sent by the DSP.

Workaround: 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 *TMS320C6455 Fixed-Point Digital Signal Processor* data manual (literature number SPRS276 revision B and later).

On the C6455/54 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 TMS320C6455/54 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 C6455/54 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 *TMS320C6455 Fixed-Point Digital Signal Processor* data manual (literature number SPRS276).

On the C6455/54 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: 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 C6455/54 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: 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 C6455/54 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 C6455/54 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: 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: 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 C6455/54 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 C6455/54 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 9 shows which modules are accessible by each TC.

Table 9. TC Connection Matrix

	TCP2	VCP2	McBSPs	UTOPIA2	Configuration Crossbar	PCI	DDR2 Memory Controller	EMIFA	C64x+ Megamodule
TC0	Y	Y	N	N	N	N	Y	Y	Y
TC1	N	N	Y	Y	Y	Y	Y	Y	Y
TC2	N	N	N	N	N	Y	Y	Y	Y
TC3	N	N	N	N	N	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 11 and Table 10) 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	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
R-0															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	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 11. QUETCMAP Register (02A0 0280h)

Table 10. 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

Appendix A Revision History

This silicon errata revision history highlights the technical changes made to the SPRZ234H revision to make it an SPRZ234I revision.

Scope: Applicable updates relating to the C6455/54 devices have been incorporated.

Table A-1. C6455/54 Revision History

SEE	ADDITIONS/MODIFICATIONS/DELETIONS
Section 2.2	Silicon Revision 2.0 Known Design Exceptions to Functional Specifications: Added new advisory: Advisory 2.0.22 - Potential SerDes Clocking Issue

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Clocks and Timers	www.ti.com/clocks	Digital Control	www.ti.com/digitalcontrol
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
RFID	www.ti-rfid.com	Telephony	www.ti.com/telephony
RF/IF and ZigBee® Solutions	www.ti.com/lprf	Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265