TMS320C6424/21
Digital Signal Processor (DSP)
Silicon Revisions 1.3, 1.2, 1.1, and 1.0

Silicon Errata

Literature Number: SPRZ252D
March 2007–Revised August 2011

TMS320C642x DSP **Silicon Revisions 1.3, 1.2, 1.1, and 1.0**

1 Introduction

This document describes the known exceptions to the functional specifications for the TMS320C642x digital signal processor (DSP) devices (TMS320C6424 and TMS320C6421). For more detailed information on these devices, see the device-specific data manual:

- *TMS320C6424 Digital Signal Processor* Data Manual (literature number SPRS347)
- *TMS320C6421 Digital Signal Processor* Data Manual (literature number SPRS346)

Throughout this document, unless otherwise specified, TMS320C642x and C642x refer to the TMS320C6424 and TMS320C6421 devices. For additional information, see the latest version of the *TMS320C642x DSP Peripherals Overview* Reference Guide (literature number SPRUEM3).

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.,). 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 Revision Identification

The device revision can be determined by the symbols marked on the top of the package. Figure 1 and Figure 2 provide examples of the TMS320C642x and TMX320C642x device markings.

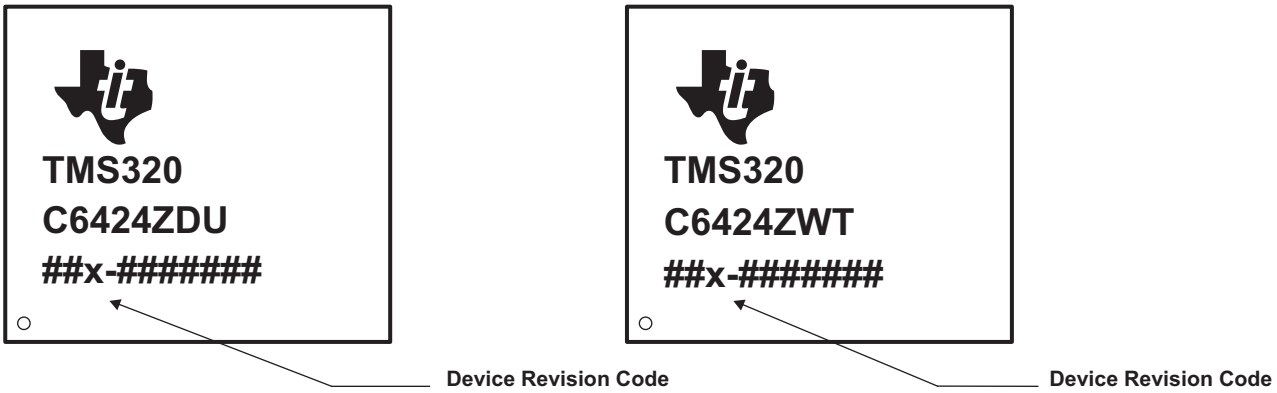

Figure 1. Example, Device Revision Codes for TMS320C642x (ZDU and ZWT Packages)

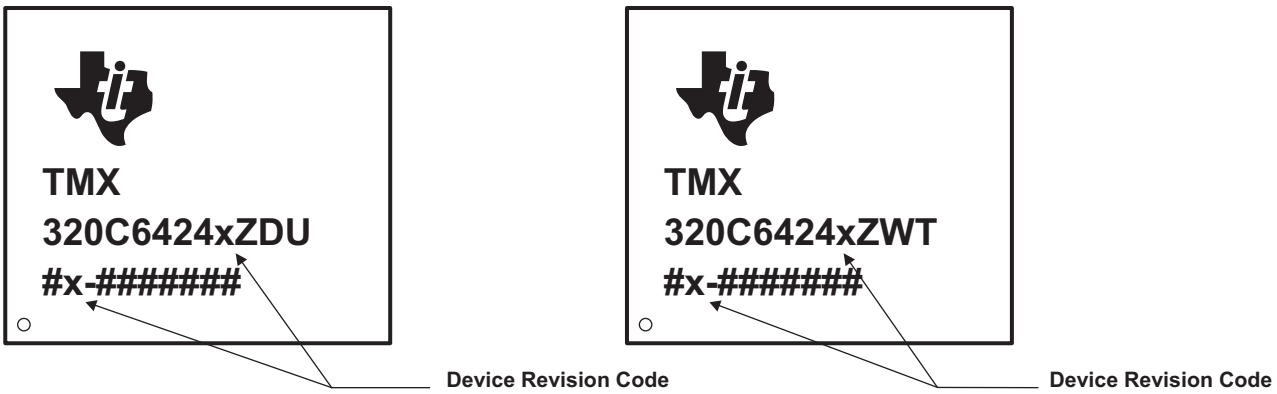

Figure 2. Example, Device Revision Codes for TMX320C642x (ZDU and ZWT Packages)

NOTES:

(A) Qualified devices are marked with the letters "TMS" at the beginning of the device name, while non-qualified devices are marked with the letters "TMX" or "TMP" at the beginning of the device name.

(B) "#" denotes an alphanumeric character. "x" denotes an alpha character only.

(C) On some "TMX" devices, the device revision code is only shown in the line that contains the part number (i.e., "320C6424xZWT").

For "TMX" initial devices, the device number *C6424* is shown on the package for all C642x devices (C6424 and C6421).

Silicon revision is identified by a code marked on the package. The code is of the format ##x-####### or #x-#######, where "x" denotes the silicon revision. On TMS devices, if x is “(blank)”, then the silicon is revision 1.3. On TMX devices, if x is"(blank)", then the silicon is revision 1.0; if x is “A”, then the silicon is revision 1.1, etc. Table 1 and Table 2 list the information associated with each silicon revision on TMS and TMX devices, respectively.

Table 1. TMS320C642x Device Revision Codes

DEVICE REVISION CODE (x)	SILICON REVISION	COMMENTS
(blank)	1.3	

Table 2. TMX320C642x Device Revision Codes

DEVICE REVISION CODE (x)	SILICON REVISION	COMMENTS
C	1.3	
B	1.2	
A	1.1	
(blank)	1.0	

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 ROM code revision can be read from address location 0010 1A00h. Table 3 shows the ROM code revision for each revision of the device.

Table 3. Silicon Revision Variables

SILICON REVISION	CPU REVISION	C64x+ MEGAMODULE REVISION	ROM CODE REVISION
1.3	1.0 (CPU ID = 10h, REVISION_ID = 00h)	Revision 2 (MM_REVID[REVISION] = 2h)	0001 0400h
1.2	1.0 (CPU ID = 10h, REVISION_ID = 00h)	Revision 2 (MM_REVID[REVISION] = 2h)	0001 0300h
1.1	1.0 (CPU ID = 10h, REVISION_ID = 00h)	Revision 2 (MM_REVID[REVISION] = 2h)	0001 0200h
1.0	1.0 (CPU ID = 10h, REVISION_ID = 00h)	Revision 2 (MM_REVID[REVISION] = 2h)	27B2 A120h

2 Silicon Revision 1.3 Usage Notes and Known Design Exceptions to Functional Specifications

This section describes the usage notes and advisories that apply to silicon revision 1.3 of the TMS320C6424 and TMS320C6421 devices.

Note: The peripherals supported on the various C642x devices are different. The user should only refer to usage notes and advisories pertaining to features supported on the specific device. For example, usage notes and advisories pertaining to PCI ***do not*** apply to the C6421 device. For a complete list of the supported features of the TMS320C6424 and TMS320C6421 devices, see the device-specific data manuals.

2.1 Usage Notes for Silicon Revision 1.3

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 usage 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 Bus Priority Inversion can Affect DDR2 Memory Controller Throughput

On all C642x silicon revisions, under certain conditions low-priority modules can occupy the bus and prevent high-priority modules from getting the required DDR2 throughput. The DDR2 memory controller arbitration policy gives preference to accesses to open banks, regardless of the requesting modules' priorities. Normally this policy is not an issue; however, it can cause a problem if a low-priority module performs extremely fast, contiguous accesses. This condition can effectively lock out other modules (even with higher priorities) trying to access the DDR2 memory controller for a long period.

For more details on master peripheral priorities, see the *Bandwidth Management* subsection of the *C64x+ DSP Megamodule* Reference Guide (literature number SPRU871).

The DDR2 memory controller Peripheral Bus Burst Priority Register (PBBPR address 0x2000 0020) contains a user-programmable field to indicate the maximum number of 32-byte DDR2 burst transfers that can go through before the DDR2 memory controller raises the priority of the oldest request in the queue. At reset, this value defaults to 0xFF, meaning that the priority raise feature is disabled—commands in the queue can be held off indefinitely due to continued accesses to an open bank by another peripheral.

It is recommended that the value of the DDR2 memory controller Peripheral Bus Burst Priority Register (PBBPR address 0x2000 0020) be re-configured to limit the length of time that a peripheral can be held off due to the policy giving preference to the open bank. There is a performance trade-off between fast, low-priority peripherals and time-critical, high-priority peripherals when determining a value for this parameter. A hex value of 0x20 should provide a good DSP (cache enabled) performance and still allow good utilization by the other modules.

2.1.2 Serial Port (McASP, McBSP) Transfers Should be Buffered in Internal Memory

On all C642x silicon revisions, the Multichannel Buffered Serial Port (McBSP) and the Multichannel Audio Serial Port (McASP) transfers are recommended to originate and complete from on-chip buffers-the DSP RAM. This is due to the fact that there is no tolerance for audio data dropouts that may occur due to the delays in DDR2 memory controller accesses from other masters and from unavoidable DDR2 memory controller refresh cycles; even if the Q0/TC0 is dedicated to transfers from off-chip memories.

On-chip buffers might be used to ensure immunity from DDR2 memory controller latencies. Once completed, the data can be shuttled between the internal buffer and the DDR2 memory by using EDMA Q1/TC1.

Due to Advisory 1.3.11, McASP and/or McBSP buffers should be placed in L1D and SDMA/IDMA accesses to L2 should be elimintaed whenever possible. An EDMA transfer from L2 to/from a McBSP or McASP register could be stalled if an MDMA access (for example a cache access to DDR) is stalled. For more details, see Advisory 1.3.11, *DSP SDMA/IDMA: Unexpected Stalling When L2 Memory Configured as Non-cache (RAM)*.

2.1.3 DDR2 Memory Controller VTP I/O Calibration Must be Performed Following Device Power-Up and Device Reset

On all C642x silicon revisions, the DDR2 memory controller is able to control the output impedance of the I/O. This feature allows the DDR2 memory controller to tune the output impedance of the I/O to match that of the printed circuit board (PCB). Control of the output impedance of the I/O is an important feature because impedance matching reduces reflections, creating a cleaner signal propagation. Calibrating the output impedance of the I/O also reduces the power consumption of the DDR2 memory controller. The calibration is performed with respect to Voltage, Temperature, and Process (VTP). The VTP information obtained from the calibration is used to control the output impedance of the I/O.

VTP I/O calibration ***must*** be performed following device power-up and device reset. If the DDR2 memory controller is reset via the Power and Sleep Controller (PSC), and the VTP input clock is disabled, accesses to the DDR2 memory controller will not complete. To re-enable accesses to the DDR2 memory controller, enable the VTP input clock, then perform the VTP calibration sequence again. The VTP calibration is part of the DDR2 memory controller initialization sequence. For more information on the VTP calibration and the proper DDR2 memory controller initialization sequence, see the *TMS320C642x DSP DDR2 Memory Controller* User's Guide (literature number SPRUEM4).

(Internal Reference Number: 6)

2.1.4 Apply External Pullup Resistors on $\overline{\text{EM\_CS2}}$, $\overline{\text{EM\_CS3}}$, $\overline{\text{EM\_CS4}}$, and $\overline{\text{EM\_CS5}}$ Pins for EMIFA

On all C642x silicon revisions, the EMIFA $\overline{\text{EM\_CS2}}$, $\overline{\text{EM\_CS3}}$, $\overline{\text{EM\_CS4}}$, and $\overline{\text{EM\_CS5}}$ pins feature an internal pulldown resistor. If the $\overline{\text{EM\_CS3}}$, $\overline{\text{EM\_CS4}}$, and $\overline{\text{EM\_CS5}}$ pins are to be connected and used as EMIFA chip select signals, an external pullup resistor should be applied to these pins to ensure the EMIF chip select signals default to an inactive (high) state immediately following reset. An external pullup resistor should also be applied to the $\overline{\text{EM\_CS2}}$ pin if AEM[2:0] pins are set to a value other than zero, and this pin is used as EMIFA chip select.

2.1.5 PCI Reset ($\overline{\text{PRST}}$) and Chip-Level Global Reset Must be Asserted Together (C6424)

On all C6424 silicon revisions, the PCI module has both an internal and external reset source (see Figure 3). The external reset is asserted through *either* the PCI Reset pin ($\overline{\text{PRST}}$) *or* the Power-On Reset pin ($\overline{\text{POR}}$). The internal reset is controlled by the Local Power and Sleep Controller (LPSC) of the PCI module. The PSC asserts the internal reset of the PCI module after a device-level global reset or when the DSP code programs the PSC to do so. A device-level global reset is generated through the Power-On-Reset pin ($\overline{\text{POR}}$) or the Warm Reset pin ($\overline{\text{RESET}}$); however, there are other methods to generate a device-level global reset. For these other methods, see Table 4.

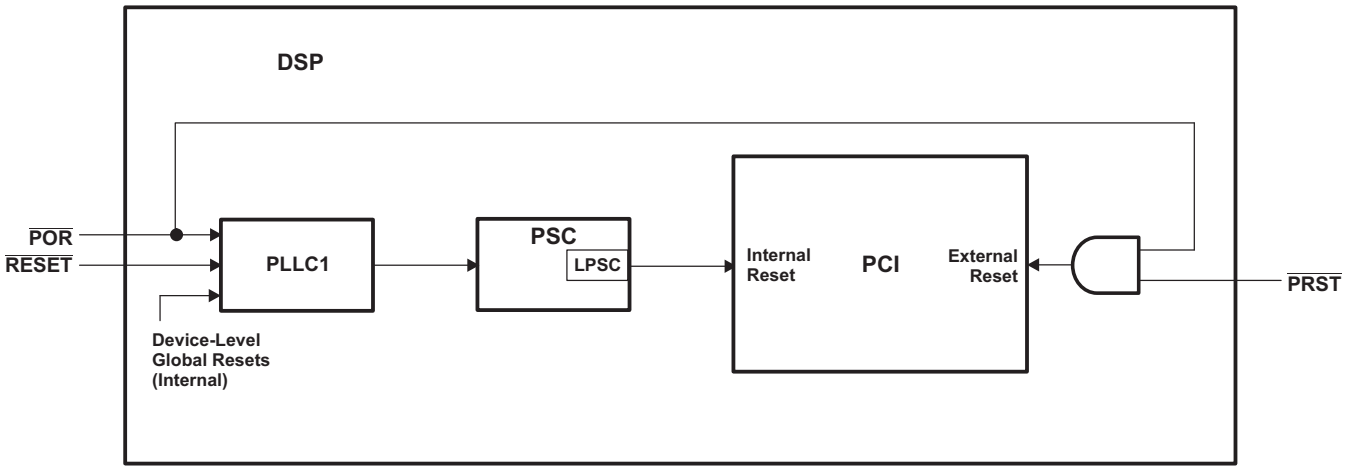

Figure 3. Internal and External Reset Inputs of the PCI Module

Systems should be designed such that a device-level global reset *always* generates the PCI Reset ($\overline{\text{PRST}}$) and vice-versa. 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.

Figure 4 and Figure 5 show example circuits which ensure both the device-level global reset and the PCI Reset ($\overline{\text{PRST}}$) are asserted together.

Note: 1) Asserting a Power-On-Reset $\overline{\text{POR}}$ also asserts a PCI *External Reset* [internally] (see Figure 3) 2) The device boot and configuration pins are latched upon either $\overline{\text{POR}}$ or $\overline{\text{RESET}}$ deassertion (high). In Figure 4 and Figure 5 where PCI Reset $\overline{\text{PRST}}$ and Warm Reset ($\overline{\text{RESET}}$) are tied together, a PCI reset will cause the device boot and configuration pins to be latched; therefore, the device boot and configuration pins must be at valid levels to ensure the device is taken out of reset in the desired modes.

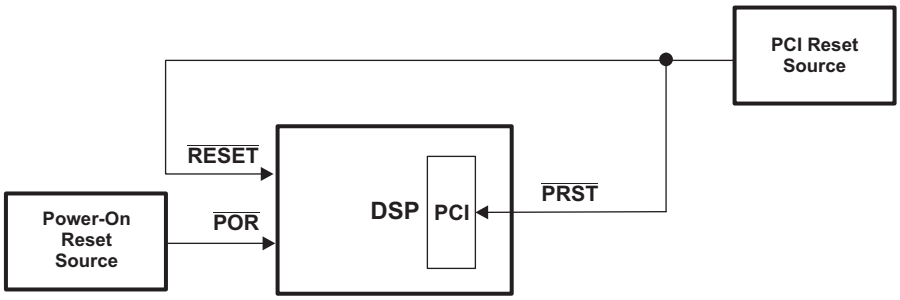

Figure 4. PCI Reset ($\overline{\text{PRST}}$) and Warm Reset ($\overline{\text{RESET}}$) Tied Together from Single External Source

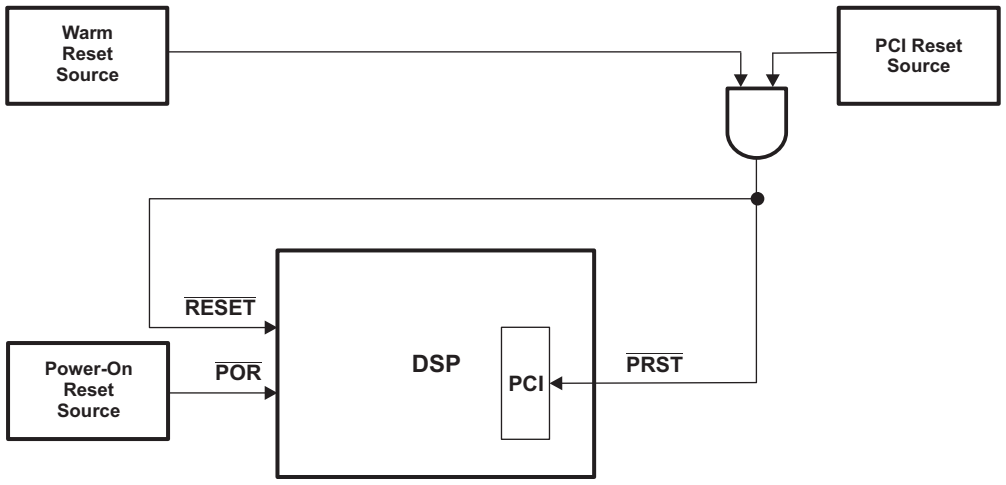

Figure 5. Independent External Power-On Reset (POR), Warm Reset (RESET), and PCI Reset (PRST) Sources

In Figure 4 and Figure 5 a device-level global reset resets the DSP and the PCI module only; other devices on the PCI bus *will not* be reset. To avoid interrupting any ongoing PCI transactions to/from the DSP, the DSP PCI should be "disconnected" from the PCI system.

Note: Resetting the DSP PCI in the middle of a master or slave transaction can cause a PCI bus fault at the system level.

The DSP PCI System Disconnection software 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.

1. Ensure the PCI bus is not parked on the DSP PCI pins since the DSP will place its PCI output pins in a high-impedance (Hi-Z) state whenever it is reset.
2. Stop all PCI transactions started within the DSP. This includes any ongoing EDMA transactions to/from the PCI memory space.
3. Disable the PCI slave and master. This can be done by the DSP code through writing to the back-end registers of the PCI or by the external host through configuration/memory accesses to the PCI registers.
   (a) Clear the base enable bits (BASEx_EN, where x = 5–0) in the PCI Slave Control Register (PCISLVCNTL).
   (b) Clear the bus master bit (BUS_MS) and memory access bit (MEM_SP) in the PCI Command/Status Register (PCICSR).
4. Assert the DSP device-level global reset and PCI Reset (PRST).
5. Once DSP comes out of reset (RESETOUT pin high), re-configure the PCI via either DSP code or an external host.
6. PCI transactions to/from the DSP can now be restarted.

Once the DSP PCI System Disconnection software sequence above has been executed, the DSP does not acknowledge accesses from external devices until the PCI has been re-configured (via either DSP software or an external host). Devices trying to access the DSP before the PCI is reconfigured could generate a master abort condition; this must be handled by the PCI system.

Generally, a device-level global reset is generated through the Power-On Reset pin (POR) or the Warm Reset pin (RESET); however, there are other methods to generate a device-level global reset. See Table 4 below for these methods and use conditions for different reset sources when using the PCI on the C642x device.

Table 4. Device-Level Global Reset Sources When Using the PCI Module

RESET	USE CONDITIONS
Power-On reset through the $\overline{\text{POR}}$ pin • Entire chip is reset during a power-on reset. • The external reset of the PCI will be asserted internally and the PSC will assert the internal PCI reset until programmed otherwise.	• The DSP PCI System Disconnection software sequence should be followed to "disconnect" the DSP from the PCI bus before Power-On Reset ($\overline{\text{POR}}$) is asserted.
Warm reset generated through the $\overline{\text{RESET}}$ pin • Entire chip minus on-chip emulation logic is reset. • The PSC asserts the internal PCI reset until programmed otherwise.	• The $\overline{\text{RESET}}$ pin must be tied to the $\overline{\text{PRST}}$ pin such that the external PCI reset is also asserted. • The DSP PCI System Disconnection software sequence should be followed to "disconnect" the DSP from the PCI bus before Warm Reset ($\overline{\text{RESET}}$) is asserted.
Emulation generates chip-level Reset • Entire chip minus on-chip emulation logic is reset. • The PSC asserts the internal PCI reset until programmed otherwise.	• The PSC should be programmed to only deassert the internal PCI reset after the external PCI reset ($\overline{\text{PRST}}$) has been asserted. • The DSP PCI System Disconnection software sequence should be followed to "disconnect" the DSP from the PCI bus before chip-level reset is asserted.
PSC Reset • DSP code configures PSC of PCI to assert the internal PCI reset.	• The PCI PSC should not be used to place the PCI in reset after it has taken the PCI out of reset.
Watchdog timer chip reset after timeout • Entire chip minus on-chip emulation logic is reset. • PSC asserts internal PCI reset until programmed otherwise.	• Since the external PCI reset is not asserted, the Watchdog timer should not be used to reset the chip when PCI is being used. However, the Watchdog timer could however be used to toggle a GPIO pin such that an external device acknowledges that the entire DSP needs to be reset.

2.1.6 ROM Bootloader Disables Cache After Boot

On all C642x silicon revisions, the ROM bootloader disables ***all*** cache for L2 RAM and L1 RAM (both L1 Data (L1D) and L1 Program (L1P)) during the boot process. If cache is enabled during the boot process via Application Image Script (AIS) commands, please be aware that the bootloader code ***will*** disable cache ***again*** after the application code is fully loaded and prior to the branch to application start; therefore, the application code ***must*** explicitly enable cache, if cache use is required. The application cannot assume cache is in the default power-on state, especially if cache was enabled during boot. The bootloader ***does not*** restore cache registers to their power-on defaults; it simply disables the cache upon exit.

For more information on the bootloader, refer to the *Using the TMS320C642x Bootloader* Application Report (literature number SPRAAK5). For more information on programming the C64x+ cache, refer to the *C64x+ DSP Megamodule* Reference Guide (literature number SPRU871).

2.1.7 C64x+ L1 Cache: L1PCFG Register Definition

On all C642x silicon revisions, the L1P Configuration Register (L1PCFG) is device-specific and varies from what is shown in the *TMS320C64x+ DSP Megamodule* Reference Guide (literature number SPRU871).

For more detailed information on the L1PCFG register and its bit field descriptions, see the device-specific data manuals: TMS320C6424 Data Manual (literature number SPRS347) and TMS320C6421 Data Manual (literature number SPRS346).

Note: On all C6421 silicon revisions, the default setting L1PCFG.L1PMODE = 7h is invalid because the C6421 device only supports 16-KB L1P. During the device initialization sequence after reset, the user must modify the L1PMODE setting to a valid setting (0, 1h, 2h, or 3h). For more details on device initialization sequence after reset, see the *TMS320C6421 Digital Signal Processor Data Manual* (literature number SPRS346), Section 3.8, *Device Initialization Sequence After Reset*.

2.1.8 C642x PCI Cannot Burst More Than 64 Bytes When Used in Master Mode (C6424)

On all C6424 silicon revisions, the PCI on the C642x can operate as a PCI master and slave. As a slave, the C642x PCI responds to accesses initiated by an off-chip PCI master. As a master, the C642x PCI, itself, initiates transfers on the PCI bus. Usually, for memory read and write transfers, another C642x master such as the EDMA is configured to move data to/from the C642x PCI.

As a PCI master, the C642x PCI is only capable of bursting a maximum of up to 64 bytes. In other words, for memory transfers larger than 64 bytes, the C642x 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 C642x PCI. Note that the C642x 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 C642x DDR2 memory may stall due to other master accesses or because of a scheduled DDR2 memory refresh. In this case the C642x PCI will generate a target retry until the DDR2 memory controller is ready to service the PCI request.

Because of this limitation, the C642x 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 C642x PCI whenever possible.

2.1.9 Race Condition Between Data and Register Accesses When Writing to NAND Memory

On all C642x silicon revisions, a possible race condition exists between data and register accesses when writing to NAND memory. The C64x+ Megamodule has three ports: the Configuration Registers (CFG) port, Master DMA (MDMA) port, and Slave DMA (SDMA) port (see the *TMS320C64x+ Megamodule* Reference Guide (SPRU871). Generally, C64x+ Megamodule internal module (i.e., CPU and IDMA) accesses to chip or peripheral registers are carried out through the CFG port. Accesses to other DSP resources such as external memory are carried out through the MDMA port. On C642x devices, CPU or IDMA accesses to memory ranges between 0x01C0 0000 and 0x01FF FFFF will generate a CFG port command.

Accesses to the C642x asynchronous EMIF (EMIFA) memory-mapped registers (MMRs) and EMIFA data space follow different architectural paths. The EMIFA MMRs are accessed through the C64x+ Megamodule CFG port. The EMIFA data space is accessed through the MDMA port. Therefore, it is possible for commands to these resources to complete at different times, even if initiated by the same master (e.g., CPU, IDMA) in a particular sequence. For example, the CPU could issue several write commands to the EMIFA data space followed by read commands to the EMIFA MMRs. However, the EMIFA MMR read commands can complete before the EMIFA data space write commands.

Software can “flush” all pending data write commands issued via the MDMA port by performing a dummy data read from an unused chip select space. Similarly, software can “flush” all pending register writes issued via the CFG port by performing a dummy register read from the EMIFA Revision Code and Status Register (RCSR).

This race condition is especially critical when using the EMIFA NAND Flash ECC Registers (NANDF1ECC - NANDF4ECC). Software will use these registers to determine the error correction code (ECC) of data written to or read from NAND memory. The following sequence should be followed when writing to NAND memory:

1. Send command sequence to NAND memory.
2. Perform a dummy data read from an unused chip select space to ensure all write commands from step 1 have been executed.
3. Clear any pre-existing ECC values by reading the corresponding NANDF*n*ECC register.
4. In the NAND Flash Control Register (NANDFCR), set ECC Start Bit corresponding to the chip select space connected to the NAND memory.
5. Perform a dummy register read from EMIFA Revision Code and Status Register (RCSR) to ensure the write command from step 4 has been executed.
6. Perform all data reads from NAND memory.
7. Perform a dummy data read from an unused chip select space to ensure all write commands from step 6 have been executed.
8. Read the NANDF*n*ECC registers to determine the ECC value.

Similarly, the following sequence should be followed when reading from NAND memory:

1. Send command sequence to NAND memory.
2. Perform a dummy data read from an unused chip select space to ensure all write commands from step 1 have been executed.
3. Clear any pre-existing ECC values by reading the corresponding NANDF*n*ECC register.
4. In the NAND Flash Control Register (NANDFCR), set ECC Start Bit corresponding to the chip select space connected to the NAND memory.
5. Perform a dummy register read from EMIFA Revision Code and Status Register (RCSR) to ensure the write command from step 4 has been executed.
6. Perform all data reads from NAND memory.
7. Read the NANDF*n*ECC registers to determine the ECC value.

2.1.10 Data Corruption Possible When Multiple Masters Write to PCI Data Space or EDMA3 Transfer and Channel Controller Registers (C6424)

On all C6424 silicon revisions, concurrent writes to the PCI data space from multiple masters (e.g., GEM MDMA, EDMA3 transfer controller) can cause data corruption. Only ***one*** master should initiate write commands to the PCI data space to avoid possible data corruption. The same restriction applies when writing to the EDMA3 channel controller (CC) and transfer controller (TC) registers. The PCI data space is located at byte address 0x3000 0000h-0x3FFF FFFF. The EDMA3 CC registers are located at byte address 0x01C0 0000-0x01C0 03FF, and the EDMA3 TC0, TC1, and TC2 registers are located at 0x01C1 0000-0x01C1 0BFF.

If more than one master is required to write to the PCI data space, the EDMA3 CC registers, or the EDMA3 TC registers, the system software should ensure ***only one*** master writes to these resources at the same time. When accessing these resources, the master should ensure its previous write commands to a resource have completed before another master initiates a new write command to the same resource.

Note: When a master issues a dummy read from a particular resource (e.g., PCI data space), all pending writes to that resource/location from that master are completed before the read command is started. Therefore, concurrent writes can be avoided by issuing dummy reads. For example, if the CPU writes to the PCI data space via the GEM MDMA port and then needs to initiate an EDMA3 transfer to the PCI data space using EDMA3 TC0, then the CPU should "flush" all its previous writes to the PCI data space by issuing a dummy read from the PCI data space before starting the EDMA3 TC0 transfers.

2.1.11 L1D Memory Size Increased (C6421 Only)

On C6421 silicon revision 1.3, the size of the L1D memory has been increased from 16KB to 48KB. The additional memory will be supported starting on silicon revision 1.3 devices. Table 5 shows the starting and ending addresses for the new L1D memory. For more information, see the *TMS320C6421 Fixed-Point Digital Signal Processor* data manual (literature number SPRS346) .

Table 5. C6421 L1D Memory Address Ranges

START ADDRESS	END ADDRESS	SIZE (Bytes)	C64x+ MEMORY MAP	EDMA PERIPHERAL MEMORY MAP
0x00F0 4000	0x00F0 BFFF	32K	Reserved	Reserved
0x00F0 C000	0x00F0 FFFF	16K	L1D RAM	
0x00F1 0000	0x00F1 7FFF	32K	L1D RAM/Cache	
0x10E1 0000	0x10F0 BFFF	1M-16K	Reserved	Reserved
0x10F0 C000	0x10F0 FFFF	16K	L1D RAM	L1D RAM
0x10F1 0000	0x10F1 7FFF	32K	L1D RAM/Cache	L1D RAM/Cache

2.1.12 DSP Module and Local Resets Currently Not Supported

As described in the *TMS320DM643x DMP DSP Subsystem* Reference Guide (literature number SPRU978), with access to the power and sleep controller (PSC) registers, an external host (for example, PCI, HPI) can assert and de-assert DSP local reset and DSP module reset. The intent of DSP module reset is for the external host to completely reset the DSP. The intent of DSP local reset is to allow the host to hold the CPU in reset while it is loading code into the DSP internal memory.

However, the effects of these resets have not been fully validated. Therefore, these resets are not supported and should not be used at this time. Instead, the PORz or RESETz pins should be used to reset the entire DSP.

2.1.13 McBSP: Maximum Frame Frequency May Not Be Possible When Word Length is 8-Bits

As described in the *TMS320DM643x DMP Multichannel Buffered Serial Port (McBSP) Interface User's Guide* (literature number SPRU943), the maximum frame frequency (defined as the bit-clock frequency divided by the number of bit clocks between frame sync signals) may not be possible when the word length is 8-bits, depending on the clock divide value CLKGDV. The CPU or EDMA may not be able to service serial port requests as frequently as every 8 bit clock. This situation can be resolved by allowing additional space between words or choosing a slower bit clock (larger than CLKGDV value).

2.1.14 Correctly Set up Low-Level System Initialization (e.g., EDMA, PLLs)

Failure to properly perform low-level system configurations (either during boot or early application startup), may result in failure of the boot or failure of the booted application to run. The user should confirm that all device-level configurations are correct as part of any debug procedures related to an application's failure to start correctly.

On all DM643x silicon revisions, from a system software perspective, care must be taken to correctly configure the various aspects of the chip during or after boot.

For example:

- External memory interfaces must be configured with correct timings for data and code to be read and written reliably.
- PLL settings must be correct for the device's specified operating range.
- Erroneously enabled EDMA events can trigger illegal data transfers that result in system failure.
  - System resources, such as EDMA usage, must be properly configured to avoid such inadvertent data transfers.
- Pinmux and other external pin configurations must be done correctly.
- Peripherals not used, should not be enabled.

2.2 Silicon Revision 1.3 Known Design Exceptions to Functional Specifications

Table 6. Silicon Revision 1.3 Advisory List

Advisory 1.3.4 *DSP Subsystem: Back-to-Back SPLOOPs with Interrupts can Cause Incorrect Operation on C64x+*

Revision(s) Affected 1.3 and earlier

Details Back-to-back software pipeline loops (SPLOOP) with interrupts can cause incorrect operation on C64x+. 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) C6000 compiler release v6.0.6 and above detects this problem. If there are fewer than 2 execute packets between the SPKERNEL & 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 Reference Number: 4)

Advisory 1.3.5 *DSP Subsystem: C64x+ Incorrectly Generates False Exceptions for Multiple Writes*

Revision(s) Affected 1.3 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 which generate the incorrect exception:

```
 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
```

(Internal Reference Number: 5)

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.

Example of sequence with incorrectly suppressed exception:

```
 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.

Advisory 1.3.11 *DSP SDMA/IDMA: Unexpected Stalling and Potential Deadlock Condition When L2 Memory Configured as Non-cache (RAM)*

Revision(s) Affected 1.3 and earlier

Details

Note: This advisory is not applicable if DSP L2 memory is configured as 100% cache ***or*** L2 RAM *is not* accessed by IDMA or SDMA during run-time.

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, VLYNQ remote 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, including EDMA transfer controllers, UHPI, PCI, EMAC, and VLYNQ, to access Level 1 Data (L1D), Level 1 Program (L1P), and L2 RAM DSP memories. The DSP Internal DMA (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 L2 cache both implementing write-back data caches– it 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 (dashed-dotted line in Figure 6) and SDMA/IDMA paths (dashed lines in Figure 6) operate independently with minimal interference. Normally MDMA accesses may stall for extended periods of time due to expected system level delays (e.g., bandwidth limitations, DDR2 memory refreshes). However, when using L2 as RAM, SDMA and 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. An IDMA stall cannot produce a deadlock condition.

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 6 is provided for illustrative purposes and is incomplete for simplification. The IDMA/SDMA (dashed-lines) path could also go to L1D/L1P memories, and IDMA can go to DSP CFG peripherals. MDMA transactions can originate also from L1P or L1D through the L2 controller or directly from DSP.

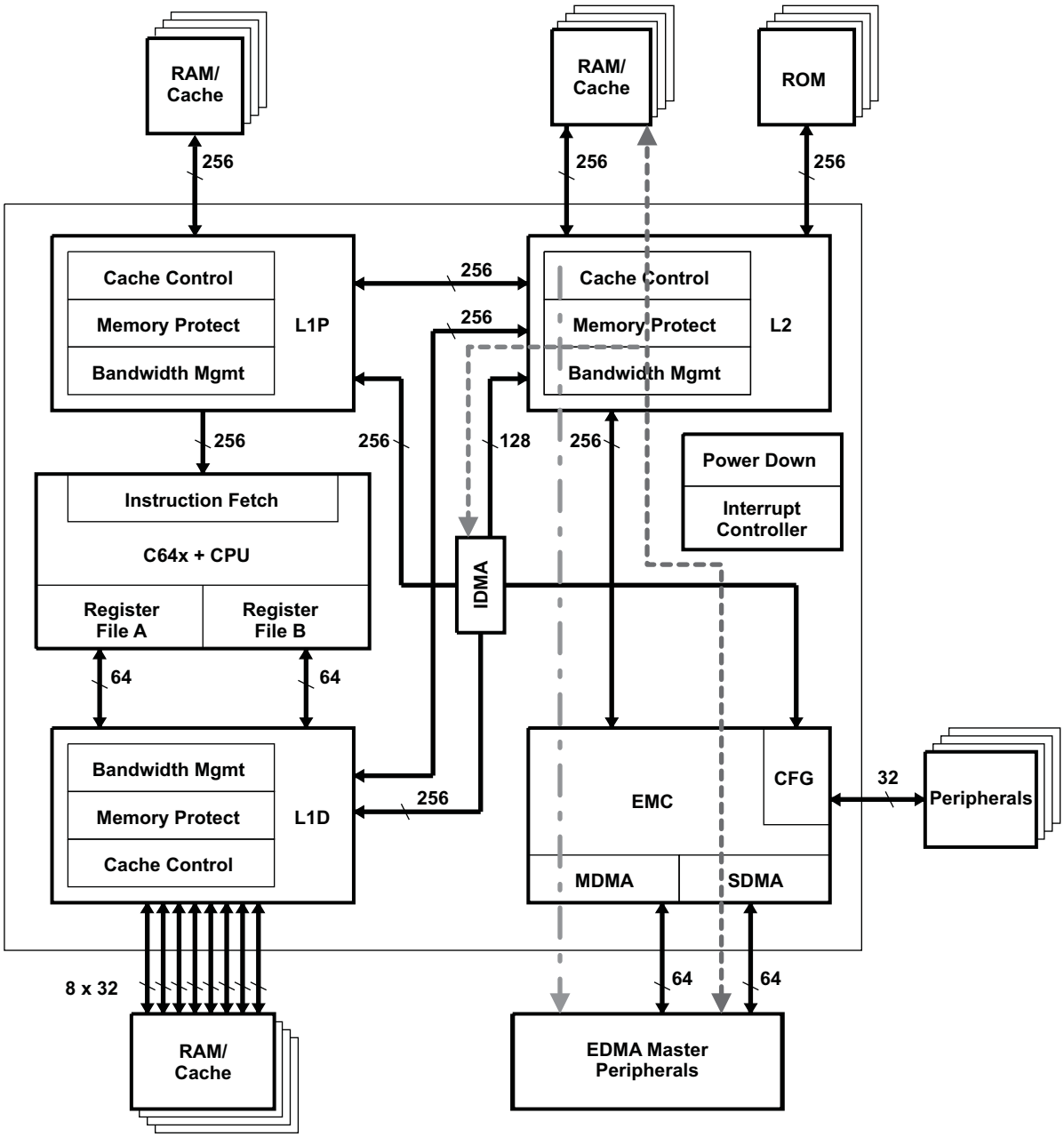

Figure 6. IDMA, SDMA, MDMA Paths

NOTES:

1. The dashed lines in Figure 6 represent SDMA/IDMA paths.
2. The dashed-dotted line in Figure 6 represents the MDMA path.

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, VLYNQ remote). The DSP buffers up to 4 non-cacheable writes. When this buffer fills, SDMA/IDMA is blocked until the buffer is no longer full. Therefore, bursts of non-cacheable writes longer than three writes can stall SDMA/IDMA traffic.
2. Various combinations of L1 and L2 cache activity:
   (a) L1D read miss generating victim traffic to L2 (Cache or SRAM) or external memory. The SDMA/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.
   (b) L1D read request missing L2 (going external) while another L1D request is pending. The SDMA/IDMA may be stalled until the external memory access is complete.
   (c) L2 victim traffic to external memory during any pending L1D request. SDMA/IDMA may be stalled until external memory access and the pending L1D request is complete.

The duration of the IDMA/SDMA stalls depends on the quantity/characteristics of the L1/L2 cache and 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 and how long stalling will occur.

IDMA/SDMA 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 procedure 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 or McASP 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 McASP, HPI block transfer to/from L2). If there are no SDMA transfers going into 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:
   (a) Temporarily transfer all the L2 SDMA transfers to L1D SRAM.
   (b) 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. Still, there should be ***no*** L2 SDMA transfers.
   (c) If 'a' and 'b' are not possible, move the transfer with the real-time deadline to 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 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 EMAC CPPI RAM. Some throughput degradation is expected when the buffers are moved to EMAC CPPI RAM.

   Note: EMAC CPPI RAM memory is only word-addressable. Therefore, EDMA transfers to/from EMAC CPPI RAM must have SRCBIDX/DSTBIDX = 4.

If real-time deadlines are still missed after implementing any of the options in Step 3, then IDMA/SDMA 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 IDMA/SDMA 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 SCR1 either directly or indirectly through a bridge. For DM643x devices the masters that fall into this category are the EDMA transfer controllers (EDMA TCs), the PCI, and the masters connected to Bridge 2 as shown in the system interconnect block diagram in the device data manual (EMAC, VLYNQ, and HPI). If the following sequence of events occurs, then a deadlock situation might arise.

1. One of the following three accesses occur:
   (a) An EDMA TC issues a write command to the DSP's SDMA followed by a subsequent write command to DDR2 (or EMIFA).
   (b) Any master connected to Bridge 2 issues a write command to the DSP's SDMA and the same or different master connected to Bridge 2 issues a subsequent write command to DDR2 (or EMIFA).
   (c) The PCI issues a write command to the DSP's SDMA followed by a subsequent write command to DDR2 (or EMIFA).
2. The DSP's SDMA asserts itself not ready and is unable to accept more write data, and a cache line writeback is initiated from DSP memory to DDR2 memory (or another slave memory, e.g. EMIFA).

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 potential for a deadlock condition using one or both of the following methods:

1. Configure entire L2 RAM as 100% cache (e.g., move all data buffers to L1D/P, EMAC CPPI memory, or external memory).

   Note: Some throughput degradation is expected when the buffers are moved to EMAC CPPI RAM. CPPI memory is only word-addressable. Therefore, EDMA transfers to/from EMAC CPPI RAM must have SRCBIDX/DSTBIDX = 4.
2. Eliminate all IDMA/SDMA access to L2 RAM during any time IDMA/SDMA stalling would have an impact (e.g., could preload data/code through IDMA/SDMA during system initialization/re-configuration).

Method 2

Perform any of the following to reduce the IDMA/SDMA stalling system impact:

1. Improve system tolerance on DMA side (IDMA/SDMA/MDMA):
   (a) Understand and minimize latency-critical SDMA/IDMA accesses to L2 or L1P/D.
   (b) 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).

(c) Reduce DSP MDMA latency by:

(i) Increase priority of 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 which dictate higher priority than DSP.

(ii) Lower PR_OLD_COUNT 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.

(iii) Do not perform EMIFA access using EMIFA WAIT handshaking during DSP run-time. Devices using WAIT potentially insert excessive latency to external memory accesses.

2. Minimize offending scenarios on DSP/Caching side:

(a) If DSP performing non-cacheable writes is causing the issue, insert non-cacheable reads every few writes to allow write buffer to drain.

(b) 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.

(c) Use explicit cache commands to trigger cache write-backs during appropriate times (L1D Writeback All, L2 Writeback All). ***Do not*** use these commands when real-time deadlines must be met.

(d) Restructure program data and data flow to minimize the offending cache activity.

(i) Define the read-only data as "const." The const C keyword tells the compiler that the array will not be written to. 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.

(ii) 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.

(iii) Avoid directly accessing data in slow memories (e.g., flash); copy at initialization time to faster memories.

(e) Modify troublesome code.

(i) 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 likelihood of IDMA/SDMA stalling.

(ii) Re-block the loops. In some cases, restructuring loops can increase reuse in the cache, and reduce the total traffic to external memory.

(iii) 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 the following to eliminate the potential for a deadlock condition:

1. Force each EDMA TC to perform writes to either DSP memory space or DDR2 (or EMIFA) memory space, but not to both.

2. For EMAC, VLYNQ, HPI, and PCI, do one of the following:
   (a) Force the completion of pending 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 that in the case of DDR2 and EMIFA, a read command only forces the completion of write commands within a 2KB-aligned window.
   (b) Force each master to perform writes to either DSP memory space or DDR2 (or EMIFA) memory space, but not to both.

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

3 Silicon Revision 1.2 Usage Notes and Known Design Exceptions to Functional Specifications

This section describes the usage notes and advisories that apply to silicon revision 1.2 of the TMS320C6424 and TMS320C6421 devices.

3.1 Usage Notes for Silicon Revision 1.2

Silicon revision 1.2 applicable usage notes have been found on a later silicon revision. For more details, see Section 2.1, *Usage Notes for Silicon Revision 1.3*.

3.2 Silicon Revision 1.2 Known Design Exceptions to Functional Specifications

Some silicon revision 1.2 applicable advisories have been found on a later silicon revision. For more details, see Section 2.2, *Silicon Revision 1.3 Known Design Exceptions to Functional Specifications*.

Table 7. Silicon Revision 1.2 Advisory List

Advisory 1.2.10 *Bootloader: NAND Flash Boot Not Supported*

Revision(s) Affected 1.2 and 1.1

Details The current bootloader implementation for NAND Flash boot is incorrect. The following issues prevent support of boot from NAND Flash devices:

- The bootloader code reads from the incorrect chip select (CS) ECC register (NANDF*n*ECC), preventing ECC checks during boot.
- The bootloader improperly polls for NAND ready causing a premature, incorrect data read from the NAND Flash.
- An address insertion error for read of NAND Flash devices prevents boot support for small block devices larger than 256MB and large block devices greater than 1GB. Boot-supported devices have the following device identification codes: 33h, 35h, 43h, 45h, 53h, 55h, 73h, E3h, E5h, E6h, A1h, B1h, C1h, F1h.

Workaround(s) A two-stage boot process can be used to work around this issue. During the first stage of the boot process, a secondary bootloader is loaded via any of the other primary boot methods supported on the device (e.g., I2C boot, SPI boot). In the second stage of the boot process, the secondary bootloader would load the application code from a NAND Flash device.

Source for a secondary bootloader that loads an application from a NAND Flash device is provided in the following link: www.ti.com/litv/zip/sprc460. This web link also contains documentation and source code for placing the secondary bootloader code on either an I2C or SPI EEPROM. The application code can still be written to the NAND Flash device as described in the *Using the TMS320C642x Bootloader* Application Report (literature number SPRAAK5).

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

This section describes the usage notes and advisories that apply to silicon revision 1.1 of the TMS320C6424 and TMS320C6421 devices.

4.1 Usage Notes for Silicon Revision 1.1

Silicon revision 1.1 applicable usage notes have been found on a later silicon revision. For more details, see Section 2.1, *Usage Notes for Silicon Revision 1.3*.

4.2 Silicon Revision 1.1 Known Design Exceptions to Functional Specifications

Some silicon revision 1.1 applicable advisories have been found on a later silicon revision. For more details, see Section 2.2, *Silicon Revision 1.3 Known Design Exceptions to Functional Specifications*.

Table 8. Silicon Revision 1.1 Advisory List

Advisory 1.1.6 *PCI: Internal Pullup/Pulldown on PCI Pins are Incorrectly Enabled in PCI Mode (C6424)*

Revision(s) Affected 1.1 and earlier

Details

For C642x devices that support PCI, when PCI ***is*** selected on C642x (PCIEN = 1), the internal pullup/pulldown (IPU/IPD) on the PCI capable pins are incorrectly enabled by default, causing up to 190 µA leakage current on these pins instead of the 10 µA leakage current permitted by the *PCI Local Bus Specification Revision 2.3* published by the PCI Special Interest Group (http://www.pcisig.com).

The IPU/IPD on these PCI capable pins can be disabled by setting bit 1 in PINMUX1 register (location 0x01C4 0004) to 1. For all boot modes where the DSP defaults to executing from the internal ROM (i.e., all boot modes except "EMIFA Direct Boot in PLL Bypass Mode" with BOOTMODE[3:0] = 0100, FASTBOOT = 0), the bootloader code sets the PINMUX1 bit 1 to disable the IPU/IPD on the PCI pins if PCIEN = 1. This bootloader programming to disable the IPU/IPD occurs within 3500 MXI/CLKIN cycles after device-level global reset is deasserted ($\overline{\text{POR}}$, $\overline{\text{RESET}}$, Max Reset, or System Reset).

In summary, for boot modes that default to executing from the internal ROM:

- PCI pins IPU/IPD are incorrectly enabled up to 3500 MXI/CLKIN cycles after device-level global reset is deasserted
- PCI pins IPU/IPD are disabled 3500 MXI/CLKIN cycles after device-level global reset is deasserted

For EMIFA Direct Boot in PLL Bypass Mode (BOOTMODE[3:0] = 0100, FASTBOOT = 0)

- PCI pins IPU/IPD are incorrectly enabled. If desired, user software must set PINMUX1 bit 1 to disable these IPU/IPD while the PCI is still kept in SwRstDisable mode by the Power and Sleep Controller (PSC).

Refer to the device-specific data manual for the list of PCI pins, and whether internal pullup (IPU) or internal pulldown (IPD) is featured on each of those pins.

Workaround(s)

Method 1:
The system designer should take into account that the IPU/IPD on the PCI pins are enabled by default with up to 190 µA leakage current. The user can apply stronger external pullup resistors to compensate for the IPD on the C642x PCI pins. When selecting the external pullup, the user must also take into account that the bootloader code disables the IPU/IPD within 3500 MXI/CLKIN cycles after device-level global reset is deasserted.

Method 2:
An alternative workaround is to have the system assert PCI reset ($\overline{\text{PRST}}$) until the IPU/IPD on the PCI pins are disabled by the bootloader code.

(Internal Reference Number: 8)

Advisory 1.1.7 *Chip: Internal Pullup/Pulldown on Some Device Pins are Incorrectly Disabled*

Revision(s) Affected 1.1 and earlier

Details On C6421, the internal pullup/pulldown (IPU/IPD) on the following pins are incorrectly disabled (see Table 9).

On C6424, when PCI ***is not*** selected (PCIEN = 0), the internal pullup/pulldown (IPU/IPD) on the following pins are incorrectly disabled (see Table 9) .

Table 9. Device Pins with Internal Pullup/Pulldown Incorrectly Disabled

C6424 PINS	C6421 PINS
AD26	RSV17
AD28	RSV18
AD30	RSV19
EM_A[13]/AD25/GP[51]	EM_A[13]/GP[51]
EM_A[14]/AD27/GP[50]	EM_A[14]/GP[50]
EM_A[15]/AD29/GP[49]	EM_A[15]/GP[49]
EM_A[16]/$\overline{\text{PGNT}}$/GP[48]	EM_A[16]/GP[48]
EM_A[17]/AD31/GP[47]	EM_A[17]/GP[47]
EM_A[18]/$\overline{\text{PRST}}$/GP[46]	EM_A[18]/GP[46]
EM_A[19]/$\overline{\text{PREQ}}$/GP[45]	EM_A[19]/GP[45]
EM_A[20]/$\overline{\text{PINTA}}$/GP[44]	EM_A[20]/GP[44]
AD0/GP[0]	GP[0]
AD1/GP[1]	GP[1]
AD2/GP[2]	GP[2]
AD4/GP[3]	GP[3]
HHWIL/MRXDV/AD13/GP[74]	HHWIL/MRXDV/GP[74]
HCNTL1/MTXEN/AD11/GP[75]	HCNTL1/MTXEN/GP[75]
HCNTL0/MRXER/AD10/GP[76]	HCNTL0/MRXER/GP[76]
HR/$\overline{\text{W}}$/MRXCLK/AD8/GP[77]	HR/$\overline{\text{W}}$/MRXCLK/GP[77]
$\overline{\text{HDS2}}$/MRXD0/AD9/GP[78]	$\overline{\text{HDS2}}$/MRXD0/GP[78]
$\overline{\text{HDS1}}$/MRXD1/AD7/GP[79]	$\overline{\text{HDS1}}$/MRXD1/GP[79]
$\overline{\text{HRDY}}$/MRXD2/$\overline{\text{PCBE0}}$/GP[80]	$\overline{\text{HRDY}}$/MRXD2/GP[80]
$\overline{\text{HCS}}$/MDCLK/AD5/GP[81]	$\overline{\text{HCS}}$/MDCLK/GP[81]
$\overline{\text{HINT}}$/MRXD3/AD6/GP[82]	$\overline{\text{HINT}}$/MRXD3/GP[82]
$\overline{\text{HAS}}$/MDIO/AD3/GP[83]	$\overline{\text{HAS}}$/MDIO/GP[83]

Workaround(s)

If a pin from this list is floating (e.g., if the pin is not constantly being driven to a valid logic level by an external device), the recommendation is to implement an external pullup/pulldown resistor for that pin. When choosing the value for the external pullup/pulldown resistor, keep in mind that on future silicon revisions, this advisory will be corrected (i.e. the IPU/IPD on the pins listed in Table 9 will be correctly enabled). Therefore, choose an external pullup/pulldown resistor with a resistance such that the voltage at that pin is at a valid logic level (within the V_{IL} and V_{IH} range specified in the device-specific data manual) regardless if the IPU/IPD is present.

The user may minimize the number of external pullup/pulldown resistors by tying together pins that have no function (and default to high-impedance) in the system, and applying a single external pulldown resistor to those pins. For example, the RSV17, RSV18, RSV19 pins have no function on the C6421 device and default to high-impedance (Hi-Z); therefore, they can be tied together through a single external pulldown resistor.

(Internal Reference Number: 8)

Advisory 1.1.8 *Emulation: High Speed RTDX (HSRTDX) Does Not Work*

Revision(s) Affected 1.1 and earlier

Details On C642x silicon revision 1.1 and earlier, High Speed RTDX (HSRTDX) ***does not*** work.

Workaround(s) ***Do not*** use HSRTDX on C642x silicon revision 1.1 and earlier.

(Internal Reference Number: 10)

5 Silicon Revision 1.0 Usage Notes and Known Design Exceptions to Functional Specifications

This section describes the usage notes and advisories that apply to silicon revision 1.0 of the TMS320C6424 and TMS320C6421 devices.

5.1 Usage Notes for Silicon Revision 1.0

Silicon revision 1.0 applicable usage notes have been found on a later silicon revision. For more details, see Section 2.1, *Usage Notes for Silicon Revision 1.3*.

5.2 Silicon Revision 1.0 Known Design Exceptions to Functional Specifications

Some silicon revision 1.0 applicable advisories have been found on a later silicon revision. For more details, see Section 2.2, *Silicon Revision 1.3 Known Design Exceptions to Functional Specifications*.

Table 10. Silicon Revision 1.0 Advisory List

Advisory 1.0.1 *Boot: Only EMIFA Direct ROM Boot Supported*

Revision(s) Affected 1.0 with initial ROM code

Details The initial ROM code release on the silicon revision 1.0 is non-functional. As a result, any boot mode that defaults to the internal ROM is non-functional. Initial ROM code can be identified by reading 0x27B2 A120 from ROM code version address location 0x0010 1A00. The table below indicates which bootmode is functional or non-functional.

Table 11. TMS320C642x DSP Boot Modes

BOOTMODE[3:0]	FASTBOOT	BOOTMODE DESCRIPTION	FUNCTIONAL ?
0000	0 or 1	No Boot (Emulation Boot)	NO
0001	1	Reserved (if PCIEN = 0) or PCI Boot (if PCIEN = 1) without Auto Initialization	NO
0010	0 or 1 1	HPI Boot (if PCIEN=0) or PCI Boot (if PCIEN = 1) with Auto Initialization	NO
0011	-	Reserved	N/A
0100	0	EMIFA ROM Direct Boot	YES
0100	1	EMIFA ROM Fast Boot	NO
0101	0 or 1	I2C Boot	NO
0110	0 or 1	SPI Boot (McBSP0 peripheral)	NO
0111	0 or 1	NAND Flash	NO
1000	0 or 1	UART Boot without Hardware Flow Control [UART0]	NO
1001 - 1101	-	Reserved	N/A
1110	0 or 1	UART Boot with Hardware Flow Control [UART0]	NO
1111	-	Reserved	N/A

Workaround(s) Use EMIFA ROM Direct Boot (BOOTMODE[3:0] = 0100, FASTBOOT = 0), as this is the only functional bootmode. Secondary bootloader can be implemented in the EMIFA ROM.

In addition, emulation is functional. Alternatively, the user can boot up the device in any of the valid bootmodes above (even if it's non-functional due to the ROM code bug), and then download code into the C642x through the emulator.

(Internal Reference Number: 7)

Revision History

This silicon errata revision history highlights the technical changes made to the *SPRZ252C* revision to make it an *SPRZ252D* revision.

Scope: Applicable updates relating to the *TMS320C6424 and TMS320C6421* devices have been incorporated.

C642x **Revision History**

SEE	ADDITIONS/CHANGES/DELETIONS
Section 2.1	Added the following Usage Note for Silicon Revision 1.3: • Section 2.1.14 – Correctly Set up Low-Level System Initialization (e.g., EDMA, PLLs)

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

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**	
Audio	www.ti.com/audio	Communications and Telecom	www.ti.com/communications
Amplifiers	amplifier.ti.com	Computers and Peripherals	www.ti.com/computers
Data Converters	dataconverter.ti.com	Consumer Electronics	www.ti.com/consumer-apps
DLP® Products	www.dlp.com	Energy and Lighting	www.ti.com/energy
DSP	dsp.ti.com	Industrial	www.ti.com/industrial
Clocks and Timers	www.ti.com/clocks	Medical	www.ti.com/medical
Interface	interface.ti.com	Security	www.ti.com/security
Logic	logic.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Power Mgmt	power.ti.com	Transportation and Automotive	www.ti.com/automotive
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com	Wireless	www.ti.com/wireless-apps
RF/IF and ZigBee® Solutions	www.ti.com/lprf		
	TI E2E Community Home Page		e2e.ti.com

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