TMS320DM6446
Digital Media System-on-Chip
Silicon Revisions 2.3, 2.1, 1.3, 1.2, and 1.1

Silicon Errata

Literature Number: SPRZ241N
December 2005–Revised July 2010

List of Figures

List of Tables

TMS320DM6446 DMSoC Silicon Revisions 2.3, 2.1, 1.3, 1.2, and 1.1

1 Introduction

This document describes the known exceptions to the functional specifications for the TMS320DM6446 Digital Media System-on-Chip (DMSoC). [See the *TMS320DM6446 Digital Media System-on-Chip* data manual (literature number SPRS283F or later).] Throughout this document, TMS320DM644x and DM644x refer to the TMS320DM6446 device.

For additional information, see the latest version of the *TMS320DM644x DMSoC Peripherals Overview Reference Guide* (literature number SPRUE19).

The advisory numbers in this document are not sequential. Some advisory numbers have been moved to the next revision and others have been removed and documented in the user's guide. When items are moved or deleted, the remaining numbers remain the same and 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., **TMS**320DM6446**A**ZWT**A**). 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:

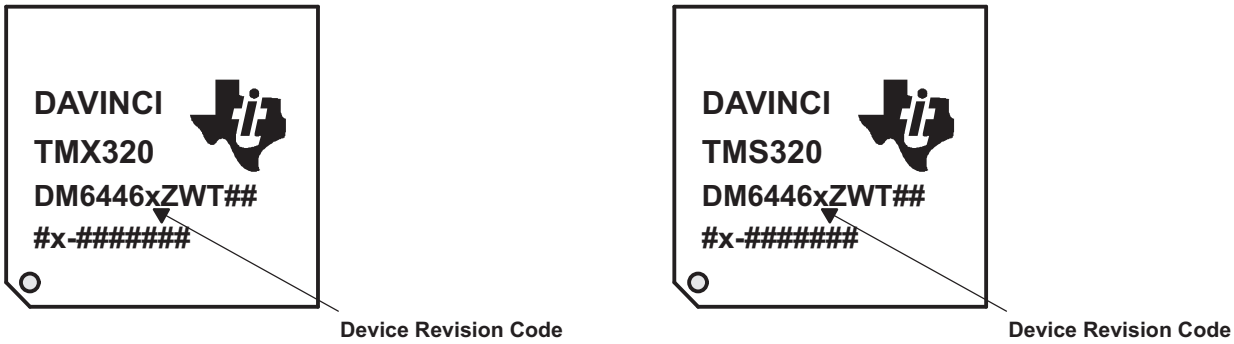
"Developmental product is intended for internal evaluation purposes."

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 device revision code marked on the top of the package. The location of the device revision code for the ZWT package is shown in Figure 1. Figure 1 shows some examples of the types of DM6446 package symbolization.



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 an alpha character only.

Figure 1. Example, Device Revision Codes for TMX320DM6446 (ZWT) and TMS320DM6446 (ZWT)

Silicon revision is identified by a code on the chip. If x is "blank," then the silicon is revision 1.1 for TMX devices or revision 1.3 for TMS devices; if x is "A", then the silicon is revision 1.2 for TMX devices or revision 2.1 for TMS devices; if x is "B", then the silicon is revision 1.3 for TMX devices or revision 2.3 for TMS devices.Table 1 lists the silicon revisions associated with each device revision code for the DM6446 device.

Table 1. Device Revision Codes

Device Revision Code (x)	Silicon Revision	Comments
B	2.3	TMS320DM6446**B**ZWT, TMS320DM6446**B**ZWT**8**, TMS320DM6446**B**ZWT**A**
A	2.1	TMS320DM6446**A**ZWT, TMS320DM6446**A**ZWT**A**
(blank)	1.3	TMS320DM6446ZWT
B	1.3	TMX320DM6446**B**ZWT
A	1.2	TMX320DM6446**A**ZWT
(blank)	1.1	TMX320DM6446ZWT

2 Silicon Revision 2.3 Usage Notes and Known Design Exceptions to Functional Specifications

NOTE: In Silicon Revision 2.3, the SPI boot mode is added as a backup to the NAND boot mode in the ROM bootloader (RBL). Additional updates have also been made to the RBL. For more information, see the *TMS320DM644x DMSoC ARM Subsystem Reference Guide* (literature number SPRUE14) and the *TMS320DM644x ROM Migration Guide* (literature number SPRAB80).

2.1 Usage Notes for Silicon Revision 2.3

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 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 EDMA Transfer Request (TR) Dequeue Priority Limitation

On DM6446 Silicon Revision 2.3 and earlier, if there are multiple events in both Q0 and Q1, then the transfer requests associated with events in Q0 will get submitted to TC0 prior to any transfer requests associated with events in Q1 getting submitted to TC1 — even if TC0 is busy processing earlier transfer requests and TC1 is idling. This can cause delays in submission of requests on Q1. Therefore, it is recommended to reserve the higher priority Q0/TC0 for submission of urgent, small, real-time sensitive transfers (such as audio data transferred to and from ASP) and allocate Q1/TC1 for longer, non-real-time sensitive transfers (such as block memory DDR2 to internal memory and vice versa).

2.1.2 Bus Priority Inversion Can Affect DDR2 Throughput

On DM6446 Silicon Revision 2.3 and earlier, under certain conditions low priority modules can occupy the bus and prevent high priority modules like the VPSS 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. This is not normally an issue, but 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 example, when the ARM is executing the STM (Store Multiple) instruction with the D-cache enabled, the ARM is able to achieve a high-peak transfer rate to cache and the DDR2 burst writes from the cache can stall the OSD (On-Screen Display) subsystem from DDR2 reads to the point that the display can be unusable. This can occur even though the VPSS, by default, has the highest priority. For more details on master peripheral priorities, see the "Bandwidth Management" subsection of the *TMS320DM644x DMSoC ARM Subsystem Reference Guide* (literature number SPRUE14).

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 255 (0xFF), meaning that this feature is disabled and a request can remain in the queue indefinitely.

It is recommended that the value of the DDR2 memory controller Peripheral Bus Burst Priority Register (PBBPR address 0x2000 0020) be reduced 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 ARM (cache enabled) performance and still allow good utilization by the VPSS or other modules.

2.1.3 Audio Serial Port (ASP) Transfers Should be Buffered in Internal Memory

On DM6446 Silicon Revision 2.3 and earlier, Audio Serial Port (ASP) transfers may need to originate and complete from on-chip buffers, either in ARM Internal RAM (TCM) or 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 accesses from other masters and from unavoidable DDR2 refresh cycles; even if the Q0/TC0 is dedicated to transfers from off-chip memories. On-chip buffers might be needed to ensure immunity from DDR2 latencies. DDR2 latencies are system-dependent, varying between applications, and are impacted by the amount and type of data traffic to DDR2 memories. Once completed, the data can be shuttled between the internal buffer and the DDR2 memory by using EDMA Q1/TC1.

For silicon revision 1.3 and earlier, if using on-chip buffers for ASP transfers, also see the following two advisories: *1.3.24 ARM: Concurrent Access to ARM Internal Memory May Fail*, and *2.3.33 DSP SDMA/IDMA: Unexpected Stalling When Memories DSP Level 2 Memory Ports used as RAM* .

2.1.4 DDR2 VTP I/O Calibration Must be Performed Following Device Power-up and Device Reset

On DM6446 Silicon Revision 2.3 and earlier, the DDR2 memory controller is able to control the impedance of the output I/O. This feature allows the DDR2 memory controller to tune the output impedance of the I/O to match that of the PCB board. 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 *TMS320DM644x DMSoC DDR2 Memory Controller User's Guide* (literature number SPRUE22).

2.1.5 ASP: Initialization Procedure When External Device is Frame-Sync Master

On DM6446 Silicon Revision 2.3 and earlier, if the ASP transmitter expects a frame sync from an external device, care must be taken to ensure that the proper action is employed. After the transmitter comes out of reset (XRST = 1), it waits for a frame sync from the external device. If the first frame sync arrives very shortly after the transmitter is enabled, the CPU or EDMA controller may not have a chance to service the ASP data transmit register (DXR). In this case, the transmitter shifts out the default data in the transmit shift register (XSR) instead of the desired value, which has not yet arrived in the DXR. This causes problems in some applications such that the first data element in the frame is invalid. The data stream appears element-shifted (the first data word may appear in the second channel instead of the first).

To ensure proper operation when the external device is the frame master, you must make sure that the DXR is already serviced with the first word when a frame sync occurs. To do so, you can keep the transmitter in reset until the first frame sync is detected. The software is set up such that it will only take the transmitter out of reset (XRST = 1) promptly after detecting the first frame sync. This ensures that the transmitter does not begin data transfers at the data pin during the first frame-sync period. This also provides almost an entire frame period for the DM6446 device to service the DXR with the first word before the second frame sync occurs. The transmitter only begins data transfers upon receiving the second frame sync. At this point, the DXR is already serviced with the first word.

The ASP transmitter and receiver on the DM6446 device are capable of generating an interrupt upon the detection of frame synchronization. However, on the DM6446 device, the receiver and/or transmitter must be out of reset to enable this feature. Therefore, instead of directly using the ASP interrupt to detect the first frame sync, on the DM6446 device you can use the GPIO peripheral. This can be achieved by connecting the frame-sync signal to a GPIO pin. The software can either poll the GPIO pin to detect the first frame sync or program the GPIO peripheral to generate an interrupt to the CPU upon detecting the first frame-sync edge. For more information on the GPIO peripheral, see the *TMS320DM644x DMSoC General-Purpose Input/Output (GPIO) User's Guide* (SPRUE25). For details on the initialization sequence when the external device is the frame-sync master, see the *TMS320DM644x DMSoC Audio Serial Port (ASP) User's Guide* (SPRUE29).

2.1.6 SPI Master Mode: CSHOLD Bit Must be Initialized Twice After Reset

On DM6446 Silicon Revision 2.3 and earlier, in addition to the procedure described in Advisory 2.3.32 (SPI Master Mode: Extra Step Required to Use CSHOLD), the SPIDAT1.CSHOLD bit must be initialized twice with the same value after reset and before the first SPI transfer. This is required to clear an internal pipeline stage in the CSHOLD logic.

2.1.7 ATA Postwrite and Pre-Fetch Do Not Provide Benefits

On DM6446 Silicon Revision 2.3 and earlier, when the Host Controller (DM6446) is performing a Read/Write transaction onto an HDD or CF Devices via a PIO Transaction, then the CPU (ARM) is responsible for Reading and Writing the data by directly accessing the Data Register Port. Since the interface speed is slower than the ARM CPU, it is required that an Internal Wait State be generated by the Host Controller to slow down the CPU (ARM). However, this wait time is much larger than a PIO Cycle time increasing the cycle time indirectly.

This problem is a common problem with many CPUs including ARM. In order to alleviate this problem, the ATA module designer has incorporated a feature that allows the CPU to write directly to the DMA FIFO, as opposed to the Data Register, in order to perform a Burst Access and avoiding the back to back wait state issue. This feature is enabled via activating the Postwrite/Prefetch capability.

Even though this feature is activated, it is observed that the CPU is taking about the same time to offload and upload data from the FIFO when compared with accessing the Data Register, i.e., data path through the FIFO did not give us any additional leverage.

2.2 Silicon Revision 2.3 Known Design Exceptions to Functional Specifications

Table 2. Silicon Revision 2.3 Advisory List

Advisory 2.3.3 *VPFE: CCDC DC-Subtract Function Does Not Clip Luma to Zero for YUV Modes*

Revision(s) Affected: 2.3 and earlier

Details: The CCD Controller (CCDC) in the VPFE subsystem includes an optional Luma DC-subtract function for YUV processing. This subtract operation does not clip Luma to zero.

Note: The Optical Black Clamp/DC-subtract function for the *Raw Data* modes ***does*** properly clip to zero for R/G/B and Ye/Cy/G/Mg color spaces.

Workaround(s): Do not use the DC-subtract function for YUV modes.

Advisory 2.3.4 *VPFE: CCDC Register Write Shadowing Does Not Work*

Revision(s) Affected: 2.3 and earlier

Details: Register write shadowing in the CCDC does not work correctly (CCDCFG.VDLC = 0). Due to this bug, the CCDC PCR.ENABLE and the CCDCFG.YCINSWP fields are always shadowed, regardless of the setting of CCDCFG.VDLC. The SDR_ADDR register will only work correctly when CCDCFG.VDLC = 1, otherwise a delayed write to the SDR_ADDR occurs, creating an unexpected delay in outputting the image to SDRAM. Other registers, listed in the subsection *Programming the CCDC* under "Registering Accessibility During Frame Processing" of the *TMS320DM644x DMSoC Video Processing Front End (VPFE) User's Guide* (literature number SPRUE38), are affected such that shadowing does not occur on the next frame as specified.

Workaround(s): Set CCDCFG.VDLC = 1, such that all registers, except those noted above, are Busy-Writable, which forces register writes to take effect immediately. If changes to the other registers are required, the CCDC should be disabled, and the current frame should be allowed to complete. Register changes can then be made, and the CCDC can then be re-enabled.

Advisory 2.3.5 *VPFE: Pixel Misalignment on CCDC to Preview Engine Path*

Revision(s) Affected: 2.3 and earlier

Details: There can be a timing glitch between the asynchronous VPFE input pixel clock and the internal VPFE clock that can cause pixel misalignment on the CCDC to PREV path. This misalignment is observed as causing a "pink" effect on the processed image (since the color pattern is misaligned). Once this first frame is adversely affected, the PREV hardware does not reset itself properly for subsequent frames, so the output will keep looking pink for the duration of the Preview mode. A similar issue on the end pixel can occur as well.

Workaround(s): Define the Preview Engine processing frame to start no earlier than the second pixel on a line and to end no later than the second-to-last pixel on a line. This requires the CCDC to be configured to transfer more pixels per line than is needed by the Preview Engine.

Advisory 2.3.11 *VPBE: RGB666 Pin Mux Option Does Not Work*

Revision(s) Affected: 2.3 and earlier

Details: The intention of the RGB666 pin mux option (PINMUX0 register at 0x01C4 0000) is to allow the users to configure **PWM2/B2/GPIO47** and **PWM1/R2/GPIO46** pins as B2 and R2 functions, respectively. However, due to a hardware limitation, setting the RGB666 bit (PINMUX0.22) ***does not*** enable the B2 and R2 functions.

Workaround(s): Enable the RGB888 pin mux option (PINMUX0.23 = 1).

When the RGB888 pin mux option is enabled, the **B2** and **R2** pin functions are available; however, by enabling the RGB888 pin mux option, the six additional pins lose their GPIO pin function capability (see Table 3).

Table 3. RBG888/RBG666 Pin Mux Options

PINMUX0		PIN FUNCTIONS							
RGB888 (Bit 23)	**RGB666 (Bit 22)**	**PWM2/ B2/ GPIO47**	**PWM1/ R2/ GPIO46**	**R1/ GPIO38**	**B1/ GPIO6**	**G1/ GPIO5**	**C_FIELD/ R0/ GPIO4**	**LCD_FIELD/ B0/ GPIO3**	**G0/ GPIO2**
0	0	GPIO47	GPIO46	GPIO38	GPIO6	GPIO5	GPIO4	GPIO3	GPIO2
0	**1**	**B2**	**R2**	GPIO38	GPIO6	GPIO5	GPIO4	GPIO3	GPIO2
1	x	**B2**	**R2**	*R1*	*B1*	*G1*	*R0*	*B0*	*G0*

Advisory 2.3.12 *VPBE: Restriction on Horizontal Width for RGB888 Video Windows*

Revision(s) Affected: 2.3 and earlier

Details: When a video window is configured for RGB888 data-input mode, certain horizontal width configurations result in corrupted video windows. The problem occurs at different widths, depending on enabling horizontal zoom (1x, 2x, and 4x) and/or 9/8 horizontal expansion modes.

Workaround(s): To work around this problem, the following constraints must be met for each case:

Case 1a: Video Window 0 (in RGB888 mode)

When 1x, 2x, 4x zoom, or 9/8 expansion is enabled, VIDWIN0**XL** must **NOT** be inside the ranges defined below:

- Z* (179 + 256N – 3) pixels < VIDWIN0**XL** < Z*(179 + 256N + 3) pixels
- Z*(261.5 + 256N – 5.5) pixels < VIDWIN0**XL** < Z*(261.5 + 256N + 5.5) pixels

for all integer values of N where:

Z = 1, 2, 4, or 9/8

XL register is the video window's horizontal display width in pixels (window width).

EXAMPLE:

When 1x zoom (Z = 1) is enabled on video window 0, the prohibited VIDWIN0**XL** ranges are the following:

1. (179 + 256N – 3) < VIDWIN0**XL** < (179 + 256N + 3) for all integer values of N
2. (261.5 + 256N – 5.5) < VIDWIN0**XL** < (261.5 + 256N + 5.5) for all integer values of N

For example,

- VIDWIN0**XL** = 436 will not work because this value falls inside the first range defined above when N = 1 (432 < VIDWIN0**XL** < 438).
- VIDWIN0**XL** = 1026 will not work because this value falls inside the second range defined above when N = 3 (1024 < VIDWIN0**XL** < 1030).
- VIDWIN0**XL** = 1024 will work because this value does not fall inside either ranges defined above for any N.

Case 1b: Video Window 0 (in RGB888 mode)

When (2x)*(9/8) or (4x)*(9/8) is enabled, VIDWIN0**XL** must **NOT** be inside the ranges defined below:

- Z* (179 + 256N – 3) pixels < VIDWIN0**XL** < Z*(179 + 256N + 3) pixels
- Z*(261.5 + 256N – 5.5) pixels < VIDWIN0**XL** < Z*(261.5 + 256N + 5.5) pixels
- VIDWIN0**XL** = (Z/2)*(1 + 72N)

for all integer values of N where:

Z = 2 when (2x)*(9/8) is enabled and

Z = 4 when (4x)*(9/8) is enabled

XL register is the video window's horizontal display width in pixels (window width).

Case 2a: Video Window 1 (in RGB888 mode)

When 1x, 2x, 4x zoom or 9/8 expansion is enabled, VIDWIN1**XL** must **NOT** be inside the ranges defined below:

- Z*(91.5 + 256N – 5.5) pixels < VIDWIN1**XL** < Z*(91.5 + 256N + 5.5) pixels
- Z*(173.5 + 256N – 3.5) pixels < VIDWIN1**XL** < Z*(173.5 + 256N + 3.5) pixels

for all integer values of N where:

Z = 1, 2, 4, or 9/8

XL register is the video window's horizontal display width in pixels (window width).

Case 2b: Video Window 1 (in RGB888 mode)

When (2x)*(9/8) or (4x)*(9/8) is enabled, VIDWIN1**XL** must **NOT** be inside the ranges defined below:

- Z*(91.5 + 256N – 5.5) pixels < VIDWIN1**XL** < Z*(91.5 + 256N + 5.5) pixels
- Z*(173.5 + 256N – 3.5) pixels < VIDWIN1**XL** < Z*(173.5 + 256N + 3.5) pixels
- VIDWIN1**XL** = (Z/2)*(1 + 72N)

for all integer values of N where:

Z = 2 when (2x)*(9/8) is enabled and

Z = 4 when (4x)*(9/8) is enabled

XL register is the video window's horizontal display width in pixels (window width).

Advisory 2.3.13 *USB: Extraneous USB Interrupts Generated*

Revision(s) Affected: 2.3 and earlier

Details: When the DM6446 device is in high-speed (HS) USB peripheral mode and suspended, an extraneous SUSPEND interrupt can be generated if the host issues a RESET. Although an extraneous SUSPEND interrupt is generated, the reset interrupt still arrives as expected at the end-of-reset sequence.

Also, when the DM6446 device is in peripheral mode (full-speed [FS] or high-speed [HS]) and being reset to HS mode (RESET with chirping), an extraneous RESET interrupt can be generated during the reset sequence.

Workaround(s): To work around the problem, the following constraints must be met:

1. Software should ignore SUSPEND interrupts when already in a "suspended" state.
2. Software *must* service every USB RESET interrupt. The interrupt flags *must* be cleared before doing any register reads or setups so that any following USB interrupts are not missed.

Advisory 2.3.18 *SPI: Receive Overrun Interrupt and Bit Error Can be Lost*

Revision(s) Affected: 2.3 and earlier

Details: Receive Overrun Interrupt (RXOVINT) and Bit Error interrupt (BITERRINT) can be lost if: Reading of the SPIFLG register coincides with the setting of these interrupt flag bits. Reading of the upper 16 bits of SPIBUF register coincides with the setting of these interrupt bits.

Workaround(s): Use the interrupt instead of the polling method to check the status of these interrupts.

Access only the lower 16 bits of the SPIBUF register to read received data.

If the polling method must be used, group the error interrupts into one Level (i.e., Level0) and the RX complete interrupt into the other Level (i.e., Level1). Use the SPIINTVECT0 and SPIINTVECT1 registers to find out the interrupt status first and then only read the SPIFLG register to decode the source of the error interrupts.

Advisory 2.3.19 *SPI: RXINTFLG Bit in SPIFLG Register May Not Get Cleared*

Revision(s) Affected: 2.3 and earlier

Details: The RXINTFLG bit in the SPIFLG register may not get cleared by reading the SPIBUF register when the read coincides with the setting of the RXINTFLG bit due to new data arrival.

Workaround(s): When the above condition occurs, the system is at the verge of receive overrun. Therefore, either optimize the SPIBUF servicing routine to avoid receive overrun or use the EDMA3 to avoid the race condition from occurring.

Advisory 2.3.20 *SPI: A Write to SPIFLG Receiver Overrun Bit Does Not Clear the Flag*

Revision(s) Affected: 2.3 and earlier

Details: A write to the SPIFLG receiver overrun (SPIFLG.OVRNINTFLG) bit does not clear the flag if the write coincides with the setting of the receive interrupt flag (SPIFLG.RXINTFLG).

Workaround(s): Write to the SPIFLG.OVRNINTFLG bit, then read back the value of the flag. If the flag did not clear, then write to clear the flag again.

Advisory 2.3.21 *SPI: The Receive Overrun Interrupt Flag is Not Set*

Revision(s) Affected: 2.3 and earlier

Details: The Receive Overrun Interrupt flag is not set if the overrun condition that sets the interrupt flag coincides with a read of the upper bytes (bits 31:24) of the SPIBUF register.

Workaround(s): None

Advisory 2.3.27 *USB: Some Electrical Parameters Violate USB Specification*

Revision(s) Affected: 2.3 and earlier

Details: Some electrical characteristics violate the USB 2.0 specification; see Table 4.

Workaround(s): Consider this violation and design your system accordingly.

Table 4. USB Electrical Characteristics in Violation

		USB SPECIFICATION		DM6446 DATA MANUAL		UNIT
		MIN	MAX	MIN	MAX	
V_{HSDSC} (1)	USB high-speed disconnect detection threshold (differential signal amplitude)	525	625	525	The lesser of: 1. VOD_DIS(2) - 75 2. 710	mV

(1) V_{HSDSC} violates the USB 2.0 specification.
(2) VOD_DIS = High-speed differential output voltage during a disconnected state.

Advisory 2.3.28 *DSP Subsystem: Back-to-Back SPLOOPs With Interrupts Can Cause Incorrect Operation on C64x+*

Revision(s) Affected: 2.3 and earlier

Details: Back-to-back software pipeline loops (SPLOOPs) 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): 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 2.3.29 *DSP Subsystem: C64x+ Incorrectly Generates False Exceptions for Multiple Writes*

Revision(s) Affected: 2.3 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 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 2.3.30 *SPI: SPIINTVECT and SPIFLG Registers are Cleared When Read in Debug Mode*

Revision(s) Affected: 2.3 and earlier

Details: Both the INTVECT and SPIFLG registers are cleared when refreshing the memory window in debug mode with CCS. These registers should be cleared only by regular CPU reads, not during debug/suspend mode.

Workaround(s): None

Advisory 2.3.31 *SPI: SPI Master Receives Extra Bit When SPICLK Polarity Changes*

Revision(s) Affected: 2.3 and earlier

Details: If the polarity of the SPICLK pin is changed and the change aligns with the receive edge for the new buffer, then it will be considered as a real SPICLK edge and the receive shift register shifts the data.

Workaround(s): Pre-select the SPIFMTx register by byte writing to just the DFSEL field in the SPIDAT1 register before actually writing to the SPIDAT1 field of the SPIDAT1 register. This additional step needs to be done only when there is going to be an SPICLK polarity change for the new buffer.

Advisory 2.3.32 *SPI Master Mode: Extra Step Required to Use CSHOLD*

Revision(s) Affected: 2.3 and earlier

Details: The SPI module chip-select hold (CSHOLD) feature allows the device to instruct the SPI to keep the chip-select pin asserted between transfers. This feature applies in master mode and is enabled by writing a '1' to SPIDAT1.CSHOLD (bit 28).

When data is written to the SPIDAT1 register with the CSHOLD bit set to '1', the master is supposed to keep the $\overline{\text{SPI\_ENx}}$ pin asserted after the transfer completes. When data is written to the SPIDAT1 register with CSHOLD set to '0', the master is supposed to de-assert the $\overline{\text{SPI\_ENx}}$ pin after the transfer completes.

For example, assume that the device needs to send two 16-bit words (0x1234 and 0x5678) to an SPI slave that requires its chip select to remain asserted between the transfers. This is a common requirement when communicating with SPI memory devices.

According to the SPI specification, the following sequence should produce the expected result as illustrated in Figure 2:

- Write 0x10001234 to SPIDAT1 for transmission of 0X1234 (CSHOLD = 1)
- Write 0x00005678 to SPIDAT1 for transmission of 0x5678 (CSHOLD = 0)

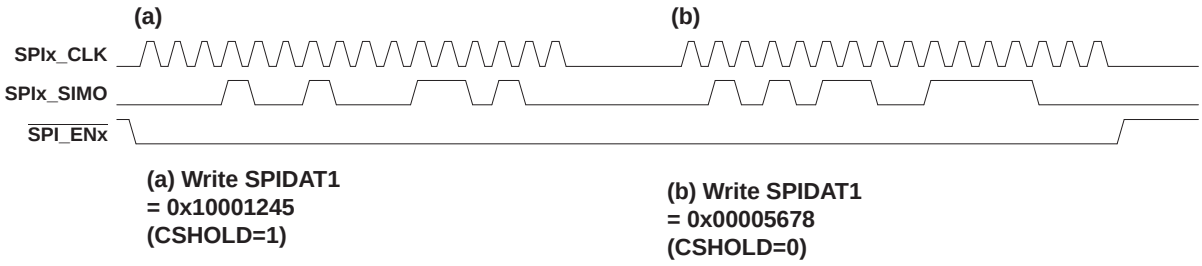


Figure 2. Expected CSHOLD Behavior

Instead, what actually occurs is that $\overline{\text{SPI\_ENx}}$ is momentarily de-asserted at the beginning of the second write, as illustrated in Figure 3.

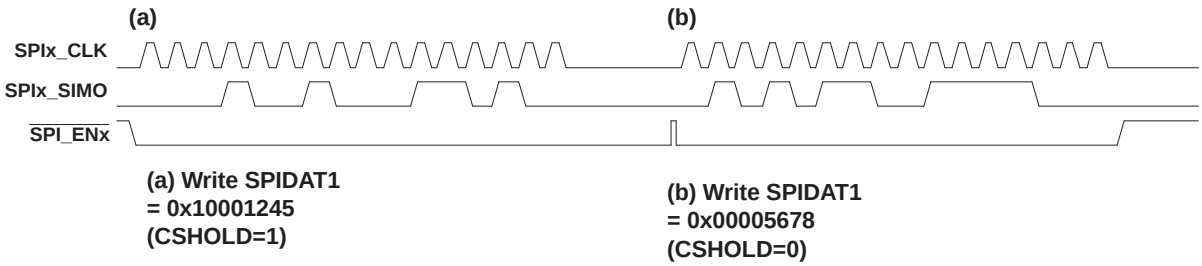


Figure 3. Actual CSHOLD Behavior–32-Bit Writes to SPIDAT1

Both Figure 2 and Figure 3 assume that SPIDAT1 is written using a single 32-bit write instruction. If SPIDAT1 is instead written using an 8-bit or 16-bit instruction to write to the CSHOLD field, followed by a 16-bit write to the transmit shift register field of SPIDAT1, then what actually occurs is illustrated in Figure 4. This is the same case illustrated in Figure 3 except that the de-assertion of $\overline{\text{SPI\_ENx}}$ lasts for the duration between writing a '0' to the CSHOLD field and writing new data to the transmit shift register.

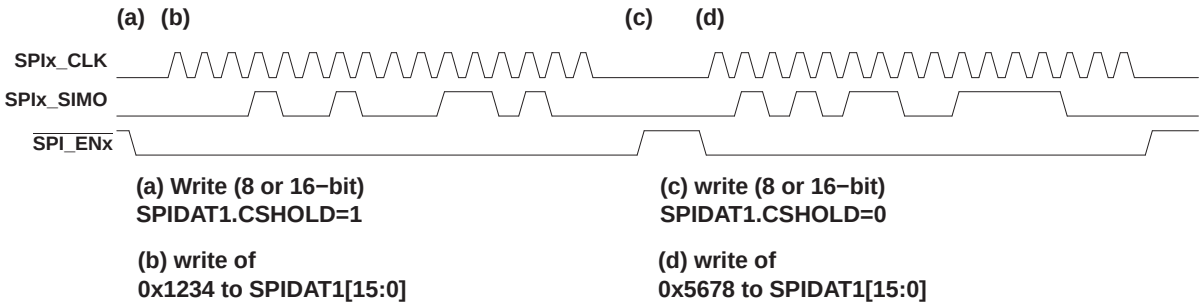


Figure 4. Actual CSHOLD Behavior−Halfword Writes to SPIDAT1

Workaround(s): For each word in the sequence of words during which $\overline{\text{SPI\_ENx}}$ should be held low, write to the SPIDAT1 register with the CSHOLD bit set to '1'. Follow this by a write to only the CSHOLD field of SPIDAT1, setting CSHOLD = 0 to de-assert $\overline{\text{SPI\_ENx}}$. See Figure 5 for an illustration.

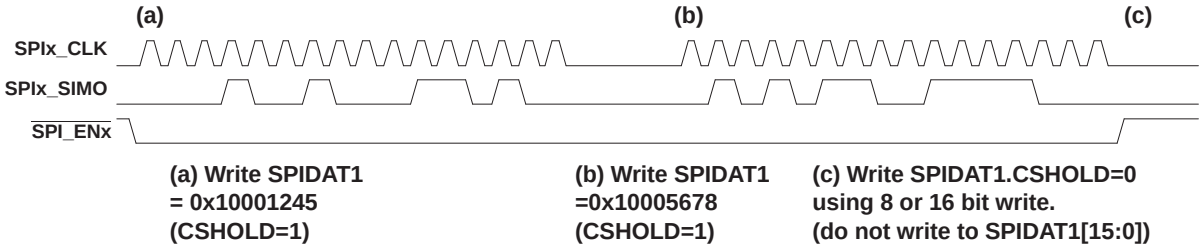


Figure 5. Workaround Assuming 32-Bit Writes to SPIDAT1 Followed by a Write Only to CSHOLD

Alternatively, only write to the SPIDAT1 CSHOLD field before and after the transfer to toggle the $\overline{\text{SPI\_ENx}}$ pin. During the transfer, write only to the data field of SPIDAT1[15:0] using 16-bit (halfword) write commands. For an illustration, see Figure 6.

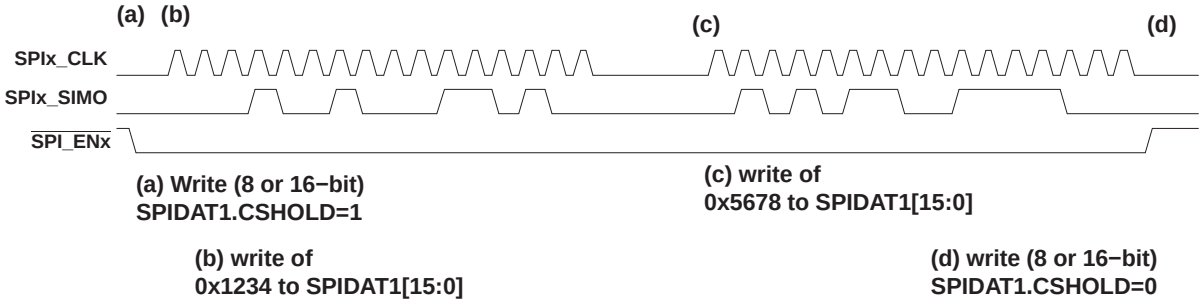


Figure 6. Workaround Assuming Halfword Writes to SPIDAT1

Advisory 2.3.33 *DSP SDMA/IDMA: Unexpected Stalling When Memories DSP Level 2 Memory Ports Used as RAM*

Revision(s) Affected: 2.3 and earlier

Details:

Note: This advisory is not applicable if DSP L2 memory is configured as 100% cache, ***or*** if L2 RAM and VICP RAM *are 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 RAM/Cache, VICP RAM) 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, VPSS, UHPI, USB, ATA, EMAC, and VLYNQ, to access Level 1 Data (L1D), Level 1 Program (L1P), and L2 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+ DSP Megamodule Reference Guide* (literature number SPRU871).

Ideally, the MDMA (dashed-dotted line in Figure 7) and SDMA/IDMA paths (dashed lines in Figure 7) 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/VICP RAM 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 SOC. 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 Level 2 memory ports (L2 RAM and VICP RAM). Unexpected SDMA/IDMA stalls to L1 happen *only* when they are pipelined behind Level 2 memory port accesses. Additionally, the deadlock scenario will be avoided if there are no SDMA accesses to the Level 2 memory ports.

Figure 7 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.

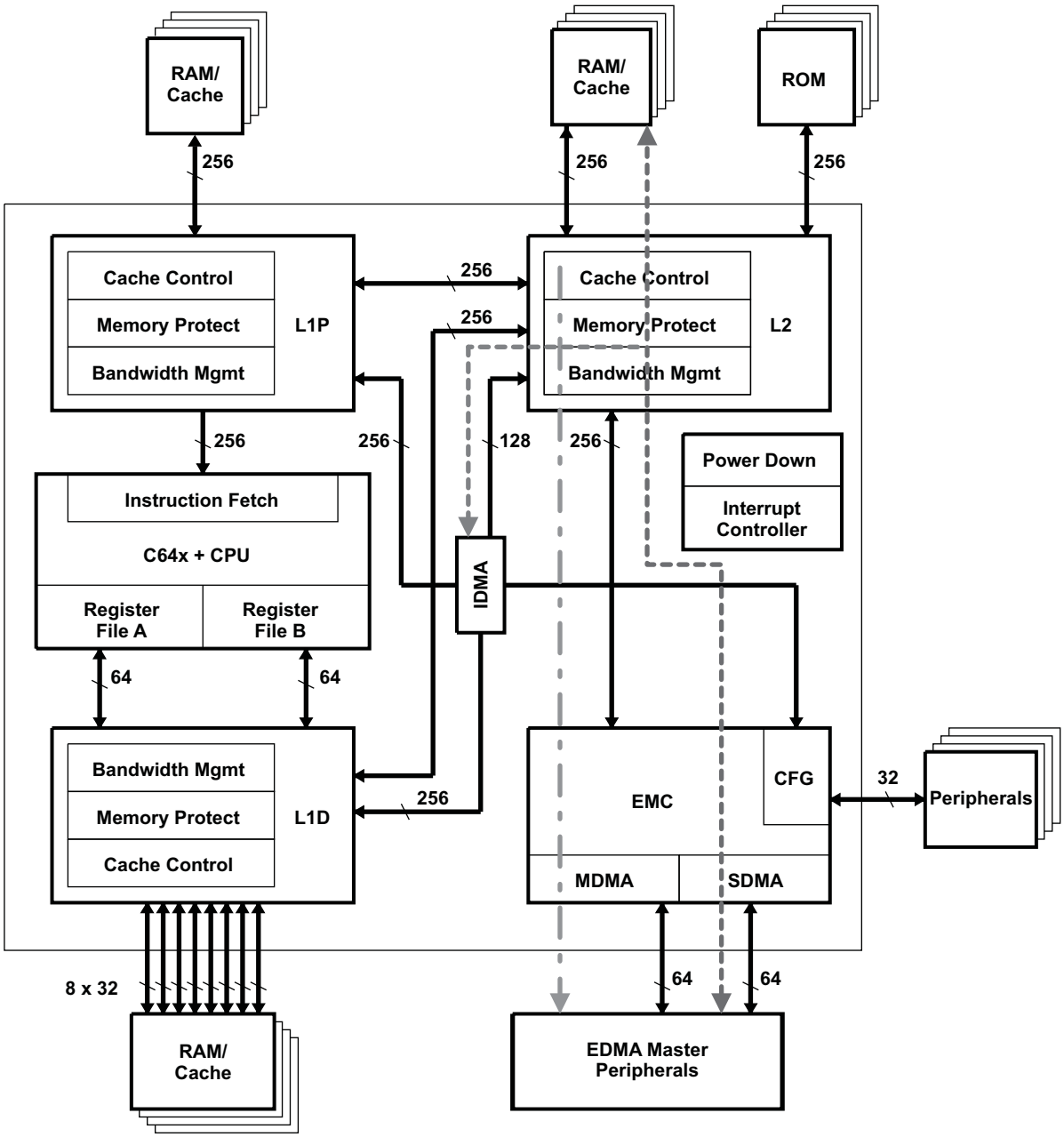


Figure 7. IDMA, SDMA, MDMA Paths

NOTES:

1. The dashed lines in Figure 7 represent SDMA/IDMA paths.
2. The dashed-dotted line in Figure 7 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. L2 read is also missed and goes to external memory via MDMA. The SDMA/IDMA may be stalled until both the read miss and victim are complete (includes an external memory access).
   (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 ASP samples and low peripheral throughput.

1. Determine if the transfer missing the real-time deadline is accessing VICP RAM, 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 Level 2 memory ports (L2 RAM or VICP RAM) [e.g., EDMA transfer to/from L2 from/to an ASP, HPI block transfer to/from L2, EDMA transfer to/from VICP RAM]. 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 or ARM RAM.
   (b) If not all L2 SDMA transfers can be moved to L1D memory or ARM RAM memory, temporarily direct some of the transfers to DDR memory and keep the rest in L1D memory/ARM RAM 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.

Note: The above steps 2 and 3 are applicable only to the SDMA accesses to L2 RAM. Accesses to VICP RAM are typically driven by an application-specific requirement and cannot be redirected to some other memory. Additionally, for memory to memory transfers involving VICP, the issue would translate more into unexpected longer stalls/delays, and thereby performance degradation, not necessarily missing real-time deadlines. For accesses to VICP RAM via the SDMA interface, if unexpected performance degradation is seen, use the guidelines under Method 2 in the Workaround(s) section.

As previously mentioned, a possible deadlock scenario is introduced in the presence of the SDMA stalls just described. This scenario occurs when multiple masters are connected to the Data SCR indirectly through a bridge. For DM644x device the masters that fall into this category are the ARM data port (ARM-D) and the masters connected to Bridge 2 as shown in the Interconnect diagram in the data sheet (UHPI/ EMAC/ VLYNQ/ USB/ ATA). If the following sequence of events occurs then a deadlock situation might arise.

1. One of the following two accesses occur:
   (a) ARM-D issues a write 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 to the DSP's SDMA followed by a subsequent write command to DDR2 (or EMIFA) from the same or different master connected to Bridge 2.
2. If at this time the DSP's SDMA asserts itself not ready and is unable to accept more write data, and if a cache line writeback from the DSP to DDR2 (or 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 traffic 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

For applications involving L2 RAM only, entirely eliminate IDMA/SDMA stalling and potential for a deadlock condition using one or both of the following:

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

   Note: Some throughput degradation is expected when the buffers are moved to ARM RAM or EMAC CPPI RAM. Additionally, 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

For applications involving IDMA/SDMA accesses to L2 RAM and/or VICP RAM, 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 Level 2 memories 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 VPSS, 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 the ARM-D to perform writes to either the DSP memory space or the DDR2/EMIFA memory space but not to both.
2. For UHPI, EMAC, VLYNQ, USB, and ATA, do one of the following:

(a) Force the completion of pending write commands to either the DSP memory space or the 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.

(b) Force each master to perform writes to either the DSP memory space or the DDR2 (or EMIFA) memory space, but not to both. This means that as a group these masters must only perform writes to either DSP memory or DDR2/EMIFA memory. For example, if EMAC writes to DSP memory and USB writes to DDR2 memory, the potential for the deadlock is still present.

Advisory 2.3.34 *VPBE: Restriction When 6/5 Vertical Expansion Filter is Enabled*

Revision(s) Affected: 2.3 and earlier

Details: The video windows are corrupted when the 6/5 vertical Expansion Filter is enabled under the following configuration: both video window 0 and video window 1 are enabled, 6/5 expansion is enabled, expansion filter is enabled, and the windows are at specific vertical coordinates.

Additionally, the video windows are corrupted when 6/5 expansion is enabled, expansion filter is enabled, and 4x or 2x vertical zoom is enabled for video window 0.

Workaround(s): To work around this problem, when the 6/5 vertical Expansion Filter is enabled, the following constraints ***must be*** met:

- (VIDWIN1YP + VIDWIN1YL - VIDWIN0YP) + 1 = Z*(6N)
  Where: Z = 1, 2, or 4 is the vertical zoom factor for video window 0 or 1, N is an integer, and YP register is a window's vertical position (upper-left pixel offset from base pixel).
- Additionally, when 6/5 vertical Expansion Filter is enabled, 4x or 2x vertical zoom cannot be enabled in video window 0.

Advisory 2.3.36 *DSP: Internal Clock Misalignment*

Revision(s) Affected: 2.3 and earlier

Details: If using ARM Boot DSP Mode, there is a possibility that the DSP's internal clock dividers will become out of phase with respect to the rest of chip-level clocks. This can cause violations in the timing requirements for some critical paths.

One known symptom of this advisory is that applications that make use of the Video/Imaging Co-Processor (VICP), might exhibit random failures in the form of application stalls, video artifacts, or VICP memory/data corruption.

The likelihood of experiencing this issue varies from one DM644x device to the next. Depending on the DM644x device, you may see different behavior. For some DM644x devices, you may not see any issues when operating at room temperature and nominal core voltage but, other devices might experience the issue under the same nominal conditions. Factors such as high case temperature, high frequency, and lower core voltage can cause this issue to become more prominent.

This advisory is ***not*** encountered when using DSP Self-Boot Mode.

Workaround(s): This issue can be avoided by following the PSC initialization sequence below. The following sequence should be implemented by all applications, ***including those not*** using VICP.

1. Apply power to the power pins (CV_{DDDSP}) of the DSP Subsystem power domain. In this step, the ARM coordinates with an external device (e.g., a microcontroller) to supply power to the power pins. For information on the power pins of the DSP power domain, see the device-specific data manual.
2. Wait for the GOSTAT[1] bit in the PTSTAT register to clear to "0".
3. Set the NEXT bit in the PDCTL1 register to "1" to prepare the DSP power domain for an "on" transition.
4. Set the NEXT bits [bits 2:0] in the MDCTL39 register to "0h" to prepare the DSP module for a SwRstDisable transition.
5. Set the GO[1] bit in the PTCMD register to "1" to initiate the state transitions.
6. Wait for the EPC bit in the EPCPR register to change to "1", indicating that the power has been applied.
7. Set the DSPPWRON bit in the CHP_SHRTSW system control module register to "1" to short the power rails of the *Always On* and *DSP* power domains.
8. Set the EPCGOOD bit in the PDCTL1 register to "1", indicating that power has been applied. The PSC proceeds with the transition after software sets the bit to "1".
9. Wait for the GOSTAT[1] bit in the PTSTAT register to clear to "0".
10. Set the NEXT bits in the MDCTL39 register to "3h" to prepare the DSP module for an enable transition.
11. Set the GO [1] bit in the PTCMD register to "1" to initiate the state transitions.
12. Wait for the GOSTAT [1] bit in the PTSTAT register to clear to "0".

Advisory 2.3.37 *VPFE: Preview Engine Hangs When the Video Port is Enabled in CCDC*

Revision(s) Affected: 2.3 and earlier

Details: For RGB Bayer pattern input data, the CCDC can be setup in such a way that it can output raw data to DDR2 while concurrently sending the same data via its video port to the Preview Engine, H3A and Histogram modules. The Preview Engine and Histogram modules can alternatively get input data from DDR2 while the H3A module can only get input data from the CCDC via the video port. In other words, the following con-current data path should work.

CCDC → H3A

CCDC → DDR2 → Preview Engine → DDR2

The problem is that these two data paths can not work con-currently. Once the video port is enabled in the CCDC by setting FMTCFG.VPEN = 1, the Preview Engine hangs after a while, i.e., the second data path breaks when the first is enabled. When the Preview engine hangs, its PCR.BUSY bit stays at 1 and the interrupt is never triggered. A VPSS reset is needed to bring the Preview Engine back to normal functional mode.

The Preview Engine is configured in one shot mode and its data source is from memory.

Workaround(s): To work around this issue implement one of the following recommendations:

1. Disable the CCDC video port(FMTCFG.VPEN = 0) while using the Preview Engine in one shot mode getting data from memory.
2. When H3A is used, configure the Preview Engine to get input from video port.

Advisory 2.3.40 *VPBE: VENC Default Luma Interpolation Filter Does Not Clip to Zero*

Revision(s) Affected: 2.3 and earlier

Details: The Video Encoder (VENC) in the VPBE subsystem includes an optional 2x interpolation function for the luma signal. The default filter used for this interpolation (VMISC.YUPF = 0), *does not* clip the luma to zero.

Workaround(s): ***Do not*** use the default setting for luma 2x interpolation filter (VMISC.YUPF = **0**). ***Instead***, use the alternate setting for luma 2x interpolation filter (VMISC.YUPF = **1**).

Advisory 2.3.41 *VLYNQ/ASP: VLYNQ and Audio Transfers Induce Noise Into the Audio Stream*

Revision(s) Affected: 2.3 and earlier

Details: VLYNQ and ASP share the same SCR7 internal bus. VLYNQ accesses tend to be high latency. When the DM644x CPU or EDMA3 initiates read/write accesses to a remote VLYNQ device, it can create blocking conditions and stalls for concurrent accesses made by the CPU or EDMA3 to the ASP. Thus, a VLYNQ access while the ASP is transferring data will generate enough internal bus stalls to cause the ASP to miss its sampling window/real-time deadlines. The missed sampling data induces noise into the audio stream.

Workaround(s): No specific workaround can be documented besides the general advice of preventing concurrent transfers on both of the peripherals.

Through internal testing, we have found the following use cases may help:

- Use only “Remotely Initiated Transactions” - i.e., the remote device pushes/pulls data where necessary, rather than via a “Host Initiated Access”
- Eliminate host initiated VLYNQ write stalls - by ensuring the write FIFO is empty before issuing a *new* write transaction
- Ensure host initiated VLYNQ slave reads complete as fast as possible to minimize stall time
- Operate VLYNQ at its' max frequency and max bus-width (4 RX/TX pins)

Advisory 2.3.42 *VPBE (S/W): Linux Legacy Video Driver Causes Jittering on Video Windows*

Revision(s) Affected: 2.3 and 2.1

Details:

A workaround to "*Advisory 1.3.8, VPBE: OSD Field Signal is Inverted for All but Video Window 0*" has been included in the VPBE driver of DaVinci Linux Support Package (LSP) Version 1.30 (Beta) and earlier. The VPBE driver disables the VIDWIN0 and inverts the field signals for all other window planes to bypass the issue as described in the advisory.

For Linux-based designs using the DM644x Linux Support Package found in the DaVinci DVSDK (see Table 5), the user can remove the temporary restriction on VIDWIN0 by patching the kernel.

Table 5 lists the software packages the LSPs are under and the corresponding patches needed to remove the temporary restriction on VIDWIN0. **Note:** Patches are currently supported on LSPs: 1.10, 1.20, and 1.30 (Beta) *only*.

Workaround(s):

Download the appropriate patch from the TI DVEVM update web site: www.ti.com/dvevmupdates. Simply Log in and select the patch needed. Table 5 lists the patch file names for each particular DVDSK and associated LSP.

If the driver source has been modified (on LSP 1.10), the patch *may not* apply cleanly (i.e., the command returns with errors).

In this case, the user can:

(a) Try to debug the patch using the "*Helpful Tips If Patch Does Not Apply Cleanly*" section below

(b) Use the steps provided in the "*Instructions to Manually Remove the VIDWIN0 Restriction*" section below

Helpful Tips If Patch Does Not Apply Cleanly

These tips address only the LSP 1.10 patch which uses the FBDev driver. If the user has modified the "davincifb.c" file, please perform a "dry-run" of the patch application before applying the patch to verify the patch application will apply cleanly.

```
patch -p1 --dry-run < patch_file.patch
```

where patch_file.patch is, insert patch file name from Table 5.

Any messages about offsets, can be ignored — for example:

```
Hunk #1 succeeded at 51 (offset 3 lines)
```

If the patch has failed hunks, the user will need to either: apply the patch and fix the rejects *or* view the patch and manually apply the changes.

In the case where the user has made extensive modifications to the FBDev driver, and wishes to remove the VIDWIN0 restriction manaully, *please* use the patch content to make these changes.

Instructions to Manually Remove the VIDWIN0 Restriction

The following modifications can be used to specifically remove the workaround to the LSP 1.10 FBDev Frame Buffer driver as described in Advisory 1.3.8, *VPBE: OSD Field Signal is Inverted for All but Video Window 0*:

Note: These steps *do not* cover the full patch as provided using the patch method. The full patch also adds methods to detect the CPU ID to determine which workaround is needed.

1. The global field inversion bit is set in the OSD_MODE register by writing 0x200. This setting is *not* needed for PG 2.3 and PG 2.1 silicon.
   Search for the following piece of code which sets the global field inversion bit in the OSD_MODE reigster:

   ```
   dispc_reg_out(OSD_MODE, 0x200);
   ```

   This line should be removed for PG 2.3 and PG 2.1 silicon.

2. In the *set_win_enable* function, currently video window 0 (VIDWIN0) is being disabled by the following piece of code:

```
dispc_reg_merge(OSD_VIDWINMD, 0, OSD_VIDWINMD_ACT0);
```

The second parameter in the above call has to be changed to "1", so that VIDWIN0 is enabled.

Table 5. DaVinci DVSDK Software Packages, LSPs, and Patches

S/W PACKAGE	LSP	PATCH FILE NAME
DVSDK 1.20	1.10	ti_davinci_mv_1_10_dm644x_pg_2_1_vid0_revert_field_inversion_fix_046 https://www-a.ti.com/downloads/sds_support/targetcontent/psp/mv_lsp_1_10/index.html
DVSDK 1.30	1.20	lsp_1_20_dm644x_pg_2_1_vid0_revert_field_inversion_fix_001 https://www-a.ti.com/downloads/sds_support/targetcontent/psp/mv_lsp_1_20/index.html
DVSDK 1.40	1.30 (Beta)	lsp_1_30_beta_dm644x_pg_2_1_vid0_revert_field_inversion_fix_001 https://www-a.ti.com/downloads/sds_support/targetcontent/psp/mv_lsp_1_30/index.html

Advisory 2.3.43 *USB (Device Mode): Calculated CRC Value Does Not Match Host CRC Value*

Revision(s) Affected: 2.3 and earlier

Details: The USB Controller can occasionally calculate a bad CRC for a received data packet. This error is rare and only occurs when ***ALL*** of the following conditions are met:

- USB Controller is in Device Mode of Operation and is receiving data
- Received data packet has a good CRC value of 0x7FF2
- A timing violation caused by a synchronization error (race condition)

The timing synchronization error is caused by a race condition between two control signals in the PHY Clock and System Clock domains. When these two synchronized control signals are crossing a clock boundary and the received data packet has a good CRC value of 0x7FF2, a race condition may occur causing one of the control signals to be latched a few pico-seconds ahead of the other control signal.

The issue has been observed on both Bulk (Non-Isochronous) and Isochronous transfers and may potentially exist on Control and Interrupt transfers since the data paths for all these transfers are the same or are very similar.

When the problem occurs in Non-Isochronous transfer types, the data that was "in-flight" to the USB Controller's FIFO from the Host is discarded by the USB Controller. Due to the error condition, the USB Controller also refrains from sending an ACK packet to the Host, as mandated by the USB transfer protocol. This forces the Host to re-transmit the data packet, anticipating an error in data transmission. The problem is usually corrected when the Host re-transmits the data packet.

When this problem occurs in Isochronous transfer mode for either High- or Full-speed, the USB Controller flags the device application S/W that a CRC error existed but retains the received data within the FIFO as well as captures the received data packet size value minus one byte from the actual data size. Since the magnitude of the actual timing violated due to the synchronization problem is only in pico-seconds, the entire data sent from the Host is routed into the USB device receive FIFO (i.e., even though the received data counter is one byte less, the full data packet is available for the USB driver).

Workaround(s): **Case 1a: Non-Isochronous Transfers (High-Speed):** For non-Isochronous transfers operating in High-Speed mode, the Host and Device H/W perform the necessary re-transmission; thererfore, the issue should be transparent to the Host driver. The issue will also be transparent to the USB device driver since the H/W flushes the received data and forces the Host driver to re-transmit by not sending an ACK packet. For this reason, no interrupt is generated by the H/W to signify an error condition to the device-side application S/W.

Although quite rare, when both the Host and Device are operating in High-Speed mode and all the three consecutive transmissions did not occur without an error, the Host will use a PING packet at a later time to check if the endpoint is ready for accepting data. Upon the Host receiving an ACK packet in response to the PING packet, the Host re-initiates the previously failed transmission again. This process continues until the transfer takes place without error. For this reason, the Non-Isochronous High-Speed transfer is immune to this issue except for a throughput reduction for the time it takes for the re-transmission.

Case 1b: Non-Isochronous Transfers (Full-Speed): For non-Isochronous transfers operating in Full-Speed mode, it is recommended that the Host driver be constructed in such a way that it invokes the transfer multiple times prior to forcing a reset to the USB device. When the transfer is repeated, it is expected for the transfer to complete "error-free".

If the Host driver is not "set up" to invoke multiple failed transfers then, the Host driver will reset the USB driver, re-enumerate, and continue from where it left off.

Case 2: Isochronous Transfers (High- and Full-Speed): For Isochronous Transfers operating in either High- or Full-Speed modes, upon receiving a CRC error, the USB controller flags the device application S/W that a CRC error existed but will retain the received data within the FIFO as well as capture the received data packet size value minus one byte from the actual data packet size. Since the magnitude of the actual timing violated due to the synchronization problem is only in pico-seconds, the entire data sent from the Host is routed into the USB device receive FIFO (i.e., even though the received data counter is one byte less, the full data packet is available for the USB driver) and the USB driver should ignore the received CRC error and read one more additional byte from the receive FIFO. This one-byte counter difference is transparent to the Host H/W and S/W.

Due to the rare occurrence of this issue and its very minimal impact on applications, there are no plans to correct this issue in future silicon revisions.

Advisory 2.3.44 *DMA Access to L2 SRAM May Stall When the C64x+ CPU Command Priority is Lower Than or Equal to the DMA Command Priority*

Revision(s) Affected: 2.3 and earlier

Details:

Note: DMA refers to all non-CPU requests. This includes Internal Direct Memory Access (IDMA) requests and all other system DMA master requests via the Slave Direct Memory Access (SDMA) port.

The C64x+ Megamodule uses a bandwidth management (BWM) system to arbitrate between the DMA and CPU requests issued to L2 RAM. For more information on the BWM feature, see the *TMS320C64x+ DSP Megamodule Reference Guide* (literature number SPRU871). The BWM arbitration grants L2 bandwidth based on programmable priorities and contention-cycle-counters. The contention-cycle-counters count the number of cycles for which the associated L2 requests are blocked by higher-priority requests. When the contention-cycle-counter reaches a programmed threshold (MAXWAIT), the associated L2 request is granted a slice of L2 bandwidth. This prevents indefinite blocking of lower-priority requests when faced with the continuous presence of higher-priority requests.

Ideally, the BWM arbitration will grant equal L2 bandwidth between equal priority DMA and CPU requests. Instead, when requests arrive at the BWM such that the CPU priority is *lower than or equal to* the DMA priority, the bandwidth is always granted in favor of the CPU over the DMA. In the case of successive CPU requests, it is possible for the CPU to block all DMA requests until the CPU traffic subsides. Figure 8 shows a high-level diagram of the arbitration scheme used for L2 RAM requests.

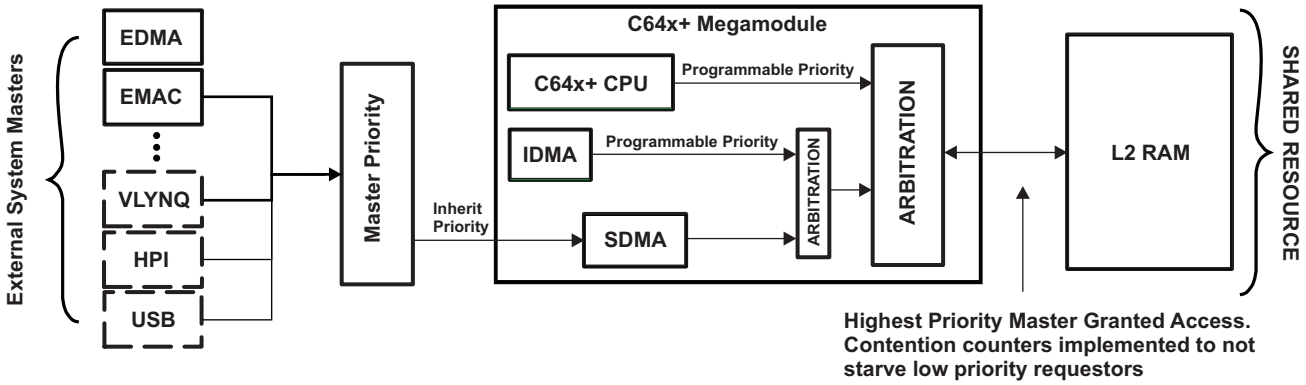


Figure 8. Priority Arbitration Scheme for L2 RAM

When the SDMA has finished sending all of its commands to the L2 controller, the C64x+ Megamodule drops the effective transfer priority to seven. The L2 Controller uses this effective priority to arbitrate the SDMA command with the CPU.

This happens regardles of what the actual SDMA priority is. This means that if the CPU Priority is equal to seven, then it can also trigger the issue highlighted by this advisory.

Workaround:

Configure the DMA and CPU requests to different priority levels such that the CPU priority level is higher than the DMA priority level. Priority seven should not be used for the CPU. There is no penalty for setting the IDMA and SDMA priorities equal to each other.

The following table highlights which priority combinations are affected and what combinations are valid:

Table 6. Allowable CPU and SDMA Priorities

CPU PRIORITY	SDMA PRIORITIES ALLOWED
0	Not allowed; Affected by Advisory Issue
1	0
2	0-1
3	0-2
4	0-3
5	0-4
6	0-5
7	Not allowed; Affected by Advisory Issue

Pseudo code provided below highlights how the various requestor priorities can be configured:

CPU request priority is programmed within the CPUARBU register:

```
/** Pseudo code only **/

        Uint32 *CPUARBU;

        CPUARBU = ( Uint32 * ) ( 0x01841000 );

        /* Set priority different from IDMA/SDMA */
        *CPUARBU = [CPU_PRIORITY];
```

IDMA request priority is programmed within the IDMA1_COUNT register

```
/** Pseudo code only **/

        Uint32 *IDMA1_SRC, *IDMA1_DST;
        Uint32 *IDMA1_CNT;

        IDMA1_SRC = ( Uint32 * ) ( 0x01820108 );
        IDMA1_DST = ( Uint32 * ) ( 0x0182010C );
        IDMA1_CNT = ( Uint32 * ) ( 0x01820110 );

        *IDMA1_SRC = sourceAddress;
        *IDMA1_DST = destinationAddress;

        /* Set IDMA priority different from CPU */
        *IDMA_CNT = ( [IDMA_PRI] << [IDMA_PRI_SHIFT] ) | buffSize ;
```

SDMA request priority is inherited from the MSTPRIn registers

```
/** Pseudo code only **/


        Uint32 *MSTPRI0, *MSTPRI1;

        MSTPRI0 = ( Uint32 * ) ( 0x01C4003C );
        MSTPRI1 = ( Uint32 * ) ( 0x01C40040 );

        /* Set SDMA master priorities different from CPU */
        *MSTPRI0 = [MAST_PRI] << [MAST_SHIFT];
        *MSTPRI1 = [MAST_PRI] << [MAST_SHIFT];
```

Advisory 2.3.45 *VPBE: Video Window 0 Corruption in RGB Mode When Video Window 1 Enabled*

Revision(s) Affected: 2.3 and earlier

Details: When Video Window 0 is in RGB mode and Video Window 1 is also enabled, certain locations of Video Window 1 cause Video Window 0 to become corrupted.

Workaround: To avoid video corruption, when Video Window 0 is in RGB mode and Video Window 1 is also enabled, the following equation must be satisfied:

- (VPBE_VIDWIN1XP - VPBE_VIDWIN0XP) MODULO 256 NOT between 81-96 or 161-176

Advisory 2.3.46 *VPBE: Video Window Buffer Location Limitation in Multibuffer Application*

Revision(s) Affected: 2.3 and earlier

Details: When the source buffer locations are being updated in real-time for Video Window 0 and Video Window 1, there might be video corruption due to the swapping of the buffers. This might happen in both "ping-pong" mode and "manual" mode where VIDWIN0ADR and VIDWIN1ADR values are updated manually.

Workaround: To avoid video corruption, when multiple buffers are used, all buffers must be allocated on the same 128-MB half of the DDR2 memory (i.e., all buffers must be allocated between address ranges 0x8000 0000 and 0x87FF FFFF ***or*** 0x8800 0000 and 0x8FFF FFFF).

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

3.1 Usage Notes for Silicon Revision 2.1

Silicon Revision 2.1 applicable usage note(s) have been found on a later silicon revision; for more detail, see Section 2.1, *Usage Notes for Silicon Revision 2.3*, of this document.

3.2 Silicon Revision 2.1 Known Design Exceptions to Functional Specifications

All known design exceptions to functional specifications for silicon revision 2.1 still apply and have been moved up to Section 2.2, *Silicon Revision 2.3 Known Design Exceptions to Functional Specifications*, of this document.

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

4.1 Usage Notes for Silicon Revision 1.3

Silicon Revision 1.3 applicable usage note(s) have been found on a later silicon revision; for more detail, see Section 2.1, *Usage Notes for Silicon Revision 2.3*, of this document.

4.2 Silicon Revision 1.3 Known Design Exceptions to Functional Specifications

All other known design exceptions to functional specifications for silicon revision 1.3 still apply and have been moved up to Section 2.2 (*Silicon Revision 2.3 Known Design Exceptions to Functional Specifications*) of this document.

Table 7. Silicon Revision 1.3 Advisory List

Advisory 1.3.1 *VPBE: Limited OSD Window Addressing Range*

Revision(s) Affected: 1.3 and earlier

Details: The OSD window addressing is limited to 128 MB of DDR2 memory. After the first 128 MB of DDR2 boundary is exceeded, the addressing will wrap around.

Workaround: Place OSD window content in the first 128 MB block of DDR2 memory.

Advisory 1.3.2 *EMU: Loss of DSP/ARM/ARM ETM Cross Triggering Capability Halts Emulation*

Revision(s) Affected: 1.3 and earlier

Details: Cross triggering capability between DSP and ARM and on ARM ETM (Embedded Trace) halts due to buffer full or buffer overflow do not function properly.

Standard debug capability using the internal ARM breakpoint and control logic is still functional. Cache benchmarking, profiling, and adaptive clocking logic are still functional.

Workaround: None

Advisory 1.3.6 *DSP Shutdown: DSP Clock Stop Sequence Does Not Work*

Revision(s) Affected: 1.3 and earlier

Details: The DSP clock stop sequence does not work as documented in the *DSP Module Clock Off* section of the *TMS320DM644x DMSoC ARM Subsystem Reference Guide* (literature number SPRUE14).

The documented sequence works *only* when the emulator is connected.

Workaround: The software ***must*** be structured such that no peripheral is allowed to access the DSP resources before shutting down the DSP clock. The DSP ***must*** check for completion of all its master peripheral initiated requests (i.e., IDMA, MDMA, EDMA, cache operations, etc.). The ARM ***must*** check for the completion of all its master peripheral initiated transactions to the DSP resources.

Use the following DSP clock stop sequence:

1. The ARM stops all masters from accessing the DSP and DSP memory.
2. The ARM polls all masters for write-completion status (or wait N number of cycles if the transfer completion status is not implemented).
3. The DSP ***must*** have the Power-Down Controller (PDC) interrupt enabled and the PDC Interrupt Service Routine (ISR) set up before the ARM initiates the DSP clock shutdown procedure.

   The ARM issues the DSP clock stop command (PSC DISABLE Command) to the DSP Power domain by writing a 0x02 value in the NEXT bit field (bits 2:0) of the DSP Local Power Sleep Controller (LPSC) Module Control (MDCTL) register. This generates a DSP PDC interrupt.

   (a) Write a 0x01 value in the GO[1:0] bit field (bits 1:0) of the DSP Power Domain Transition Command (PTCMD) register to start the transition sequence for DSP LPSC.
   (b) Check (poll for 0) the GOSTAT[1:0] bit field (bits 1:0) in the DSP Power Domain Transition Status (PTSTAT) register for power transition sequence completion.
   (c) Check value (poll for 0x02) in the STATE bit field (bits 5:0) of the DSP LPSC Module Status (MDSTAT) register indicating the DSP clock stop sequence completion.
4. Complete the following tasks within the DSP PDC Interrupt Service Routine (ISR):

   (a) Check for completion of all DSP master requests (The DSP polls transfer completion statuses of all Master peripherals)
   (b) Enable one of the ARM2DSP interrupts — ARM2DSP0, ARM2DSP1, ARM2DSP2, ARM2DSP3 or the NMI interrupt that will be used to wake up the DSP during the DSP clock-on sequence.
   (c) Write 0x0001 5555 value to the PDCCMD register in the DSP Power-Down Controller (PDC).
   (d) Set FORCE bit (bit 0) in the PSC Global Control Register (GBLCTL) at 0x01C4 1000 address.
   (e) Execute the IDLE instruction.
5. The ARM clears the FORCE bit in the PSC GBLCTL register after the PSC indicates the DSP clock shutdown is complete.

NOTE: Power ***must not*** be removed from the CV_{DDDSP} pins.

Advisory 1.3.7 *VLYNQ: VLYNQ Clock Buffer has Limited Drive Strength*

Revision(s) Affected: 1.3 and earlier

Details: In VLYNQ master mode (DM644x sourcing the VLYNQ clock), the limited drive strength of the buffer results in poor signal integrity at high speeds (> 50 MHz).

Workaround: Use one of the following options:

- Use the VLYNQ in slave mode, so the clock is sourced by the downstream device.
- For high speed designs that source the VLYNQ clock, use only a point-to-point connection on this signal. Do not use the multiplexed $\overline{\text{EM\_CS5}}$/GPIO8/VLYNQ_CLOCK functionality.
  In addition, keep the trace length for this signal limited to 4 inches or less.

Advisory 1.3.8 *VPBE: OSD Field Signal is Inverted for All but Video Window 0*

Revision(s) Affected: 1.3 and earlier

Details: When using interlaced video out (e.g., standard NTSC/PAL output), the field signals within the OSD module are inverted for all windows except VIDWIN0. The order in which the field data is fetched is swapped for the other windows (VIDWIN1, OSDWIN0, and OSDWIN1). The result is a poor display of the input window data in frame mode (contains the full vertical resolution of the display) compared to those in field mode (contains data for a single field, which is fetched twice to generate the output image).

Workaround: Choose one of the following three options:

- Use VIDWIN0 and limit the use of VIDWIN1 and OSDWIN1/2 to field mode data (½ vertical resolution data).
- Swap the field order for all windows by setting the OSD mode register field signal inversion bit [OSD.MODE.FSINV = 1], use VIDWIN1 and OSDWIN1/2, and limit use of VIDWIN0 to field mode data (½ vertical resolution data).
- Swap the DDR2 memory start address for VIDWIN0 using the ping-pong buffer as described below.

  In the following explanation, vid0_addr is the start address for image data in DDR2 memory and vid0_buflen is the line offset in bytes.

  Configure the following parameters at initialization time:

  ```
  OSD.MODE.FSINV = 1;
  OSD.VIDWIN0ADR = vid0_addr - vid0_buflen;
  OSD.VIDWIN0OFST.V0LO = vid0_buflen / 32;
  OSD.PPVWIN0ADR = vid0_addr + vid0_buflen;
  OSD.VIDWINMD.VFF0 = 1;
  OSD.MISCCTL.PPRV = 1;
  ```

  In the Video Encoder (VENC) interrupt service routine, swap the address of VIDWIN0 as follows:

  ```
  OSD.MISCCTL.PPSW = VENC.VSTAT.FIDST;
  ```

  The VENC reads lines 0, 2, 4, etc. from the DDR2 memory while displaying the *odd field* and reads lines 1, 3, 5, etc. from the DDR2 memory while displaying the *even field*. To work around this problem, the OSD.VIDWIN0ADR register is set one line *before* the start of data, the OSD.PPVWIN0ADR register is set one line *after* the start of data, and the address of VIDWIN0 is swapped depending on the field ID in the VENC (end of frame) interrupt service routine. When the field ID is even, the OSD.VIDWIN0ADR is selected; when the field ID is odd, the OSD.PPVWIN0ADR is selected; thus, the even and odd lines are read out of DDR2 memory in proper order.

Advisory 1.3.9 *VPBE: Video Window 0 is Shifted When Overlaid by Video Window 1*

Revision(s) Affected: 1.3 and earlier

Details: Position of Video Window 0 (VIDWIN0) is shifted when overlaid by certain sizes of Video Window 1 (VIDWIN1) at certain locations.

Workaround: To work around this problem, the following constraints must be met:

Case 1:

VIDWIN0 is YUV input-data format

VIDWIN1 is YUV or RGB888 input-data format

- VIDWIN1**XP** > VIDWIN0**XP**
- VIDWIN1**XP** - VIDWIN0**XP** = 16N
- VIDWIN1**XL** = 16N + 8

  Where:

  N is an integer

  XP register is the video window's horizontal display start position (upper-left pixel offset from base pixel)

  XL register is the video window's horizontal display width in pixels (window width)

Case 2:

VIDWIN0 is RGB888 input-data format

VIDWIN1 is YUV input-data format

- VIDWIN1**XP** > VIDWIN0**XP**
- VIDWIN1**XL** + VIDWIN1**XP** - VIDWIN0**XP** = 32N pixels

  Where:

  N is an integer

  XP register is the video window's horizontal display start position (upper-left pixel offset from base pixel)

  XL register is the video window's horizontal display width in pixels (window width)

Advisory 1.3.10 *VPBE: RGB888 Video Windows Corrupted When Overlaid by OSD Windows*

Revision(s) Affected: 1.3 and earlier

Details: The video windows are corrupted when configured in RGB888 data-input mode and *overlaid* at *specific locations* by a *specific size* OSD window in a bitmap or RGB565 data-input format.

The problem occurs for both Video Windows 0 and 1 and occurs for *all* OSD data-input formats (i.e., 8-bit, 4-bit, 2-bit, 1-bit, and RGB565).

Workaround: To work around this problem, the following constraints must be met:

OSDWINn**XL** + OSDWINn**XP** - VIDWINn**XP** ≠ 80 + 256M ± 16 pixels

Where:

M is an integer

n = 0 or 1 (video- or OSD-window number)

XP register is the video window's horizontal display start position (upper-left pixel offset from base pixel)

XL register is the video window's horizontal display width in pixels (window width)

Advisory 1.3.14 *Device Configuration: MSTPRI0 Register Does Not Work. MSTPRI1 Register Affects All Fixed-Priority Masters.*

Revision(s) Affected: 1.3 and earlier

Details: The DM6446 device has two bus master priority control registers (MSTPRI0 and MSTPRI1) for system-level performance tuning. These registers are located at address 0x01C4 003C and 0x01C4 0040, respectively.

Due to a hardware connection problem, the MSTPRI0 register has no effect on the ARM_DMA, ARM_CFG, C64x+_CFG, and VICP bus priority. On the other hand, the MSTPRI1 register affects *all* fixed-priority masters according to Figure 9, *MSTPRI1 Register*.

31							24
Reserved							
R-0000 0000							
23					19	18	16
Reserved						VLYNQP/VICPP	
R-01000						R/W-100	
15	14			12	11	10	8
Rsvd	ATAP				Rsvd	USBP/C64x+_CFGP	
R-0	R/W-100				R-0	R/W-100	
7	6			4	3	2	0
Rsvd	ARM_CFGP				Rsvd	EMACP/ARM_DMAP	
R-0	R/W-100				R-0	R/W-100	

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

Figure 9. MSTPRI1 Register

The default bus master priority values are also changed (see Table 8).

Table 8. Bus Master Priority Defaults

Master	Default Priority
VPSS	0
TC0	0
TC1	0
ARM (DMA)	4
ARM (CFG)	4
C64x+ (DMA)	7
C64x+ (CFG)	4
EMAC	4
USB	4
ATA/CF	4
VLYNQ	4
VICP	4

Workaround: None

Advisory 1.3.15 *Clock Domain: Peripherals in the Fixed-Clock Domain (MXI) may Lose Register Contents if Clock is Turned Off*

Revision(s) Affected: 1.3 and earlier

Details: The UART, I2C, PWM, Timer, and WD Timer are the fixed-clock domain peripherals that run at the rate of input clock (MXI). If for some reason (i.e., power savings) the clocks to these peripherals are shut down via their respective Power and Sleep Controller (PSC), their register content may be lost.

Workaround: Use one of the following:

- ***Do not*** shut down the clocks to these peripherals (UART, I2C, PWM, and Timers) once programmed and enabled.
- If clocks to these peripherals (UART, I2C, PWM, and Timers) are stopped, restart the clock, then reprogram all the peripheral registers.

Advisory 1.3.16 *L1D Cache: C64x+ L1D Cache May Lose Data or Hang DMA Operations Under Certain Conditions*

Revision(s) Affected: 1.3 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 the EMAC 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: 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 the locations of parallel predicated loads that may fail. The script is available at the same update advisor location as the Code Gen patch.

Advisory 1.3.17 *DSP Subsystem: C64x+ CPU STORE Instruction to Any MMRs or EMIFA While IDMA Channel 0 Transfer is in Progress Can Hang C64x+ CPU*

Revision(s) Affected: 1.3 and earlier

Details: The C64x+ DSP uses IDMA Channel 0 to transfer contents between the DSP internal memory and the resources on the CFG address spaces. The C64x+ DSP accessible resources on the CFG bus are:

- Asynchronous EMIF
- Timer
- ASP
- EDMA
- PSC
- VICP

The C64x+ core can potentially hang if any CPU STORE instruction to the CFG bus takes place while previously initiated IDMA Channel 0 transfers are in progress.

Workaround: The user should analyze the application for potential places in which IDMA Channel 0 is used with potential places in which STWs to CFG address space are performed. Extra attention should be paid to portions of the application that are multi-threaded. For example, if the main-line application code performs IDMA Channel 0 transactions and an Interrupt Service routine performs CPU STORE instructions to CFG address space, the application is at risk to encounter this bug, depending on the timing of interrupt occurrences.

For those systems using DSP/BIOS 5.30, the HWI interrupt dispatcher will ensure that this cannot occur. In this version of DSP/BIOS, the default behavior for the dispatcher checks for IDMA0 completion before actually calling the ISR, meaning that if software modules do not violate the constraints described above, the combination of the individual modules in a multi-tasking environment will also work. See DSP/BIOS 5.30 documentation for more details at https://www-a.ti.com/downloads/sds_support/targetcontent/bios/index.html.

For systems that do not use DSP/BIOS, the system integrator will need to ensure that the following steps are taken:

1. Modify the interrupt service routine (or any routine(s) that perform CPU STORE instruction to CFG address space) to ensure that IDMA transfers are not in progress (poll on IDMA0_STAT) prior to performing CPU STORE instruction to CFG address space.

   Note: This has negative impact of increasing interrupt service routine latency in the event that an IDMA is in progress when the interrupt service routine begins. However, this can be limited to only the interrupt service routines which perform CPU STORE instructions to CFG address space.
2. Modify the IDMA submission code to:
   - Disable the specific interrupts prior to performing IDMA
   - Perform IDMA transaction
   - Poll for IDMA completion
   - Re-enable interrupts

   Note: This has the negative impact of Increasing interrupt service routing latency, potentially eliminating a key advantage of IDMA, where the CPU can perform useful computation while IDMA is in progress.

Advisory 1.3.22 *PSC: PTSTAT Register Does Not Clear After Warm/Maximum Reset*

Revision(s) Affected: 1.3 and earlier

Details: Following either a warm reset or a maximum reset, the GOSTAT bit field in the PTSTAT register will not clear to 0. Code executing state transitions at either the power domain level or the module level will hang when it polls for the GOSTAT bit to clear to 0. Section 7.4 in the *TMS320DM644x DMSoC ARM Subsystem Reference Guide* (literature number SPRUE14) describes how to properly execute state transitions at the power domain level as well as at the module level.

Workaround: Following a warm reset or a maximum reset, clear the reserved location at address 0x01C4 1A20 prior to executing any state transitions at the power domain level or at the module level. This means that prior to enabling any power domains or changing the state of any of the peripherals modules (such as transitioning from disable to enable) the reserved location at address 0x01C4 1A20 must be cleared to 0.

Advisory 1.3.23 *EMU: Poor Quality of the RTCK and TDO Emulation Signals May Cause Loss of Synchronization*

Revision(s) Affected: 1.3 and earlier

Details: Poor quality of the RTCK and TDO emulation signals, due to weak emulation output buffers, may slow down the interface between the device and the debugger hardware (debug controller pod) and software (Code Composer Studio). In some extreme cases, the debugger hardware and the software may experience loss of synchronization with the device.

Figure 10 illustrates the typical hardware interface between the TMS320DM644x DSP and the current TI or third-party debug controller pod. This combination of debug controller pod and DSP may not work reliably for two reasons:

- The RTCK and TDO buffers have approximately 100 Ω output impedance and rise and fall time of 5 ns and 4 ns respectively to minimize EMI and switching noise.
- The emulation pod is designed for a low impedance high drive output buffer with less than 3-ns rise/fall time. The debug controller pod expects the chip to drive the AC termination inside the emulation pod shown in Figure 10.

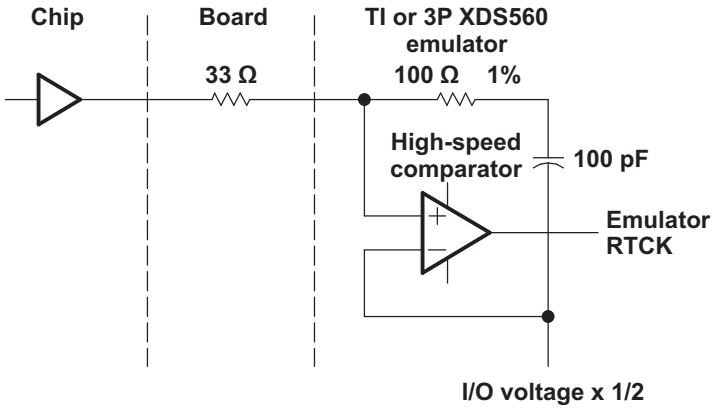


Figure 10. Typical Emulation Interface Circuit

The 100 Ω chip buffer impedance, combined with 33 Ω series termination on the target board creates a voltage divider at the debug controller pod terminal with a RC time constant on the order of 10 ns. The slow 10 ns rise time causes the debug controller pod input to oscillate as the signal passes through the switching threshold. In addition, at faster debug controller pod operating frequencies (max TCK = 34 MHz), the RTCK and TDO signals are not able to swing to the full range (0 V - 1.8 V), instead these signals swing between 0.4 V and 1.45 V.

Workaround: Four solutions are recommended, based on the selected debug controller pod and DSP target combination.

To determine which version of the TI pod is used to examine the label on the bottom of the enclosure. The revision code is the least significant digit, i.e., "ABCD EFG HJx" where "x" is the revision code. For third party debug controller pods, contact the vendor to determine which version you have.

For the purposes of this advisory, "existing" refers to Revision "B" or older debug controller pods. A "new debug controller pod" refers to any TI XDS560 revision "D" or later product, an ZDS560DT, or a comparable third-party debug emulation pod.

1. Existing Target Boards/Existing Debug controller pods:

   When using an existing debug controller pod (i.e., XDS560 Revision B or compatible third-party debug controller pod) the use of an external buffer board (such as TMDSADP1414 or TMDSADP1420) is required. Figure 11 illustrates this method.

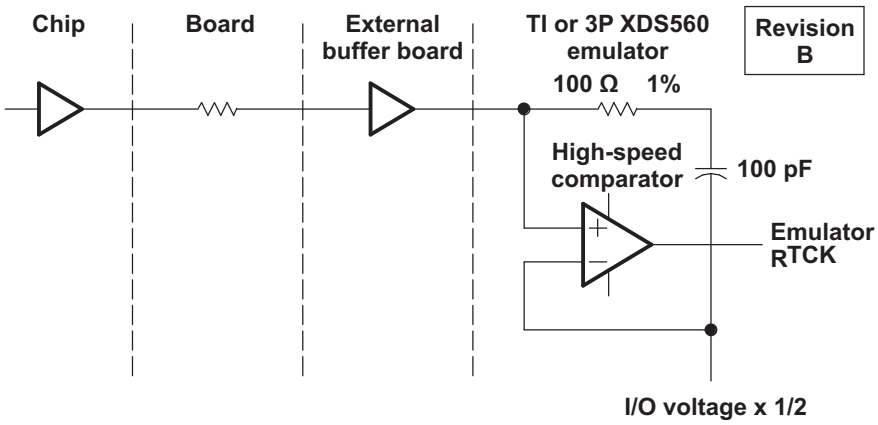


Figure 11. Emulation Interface Circuit with Existing Fielded Debug Controller Pods

2. New Target Board Designs/Existing Debug controller pods:
   (a) When designing a new target board to be used with an existing debug controller pod, it is recommended that a buffer is added to the RTCK net and the series resistor is changed. See Figure 12.
   (b) The series termination resistor utilized should be a 10 Ω, (or 22 Ω depending on net lengths), instead of the original 33-Ω value.
   (c) The recommended buffer for the RTCK line is a TI SN74AUC1G17, Single Schmitt-Trigger Buffer. The buffer characteristics for RTCK is shown in Table 9.

Table 9. RTCK Buffer Characteristics

Signal	Drive	Rise	Tpd	Fall
RTCK	8 mA	1.5 nS	1.1 nS	1.5 nS

   (d) The Tpd delay for the new buffer is rated at a nominal at 1.1 nS. The delay induced by this buffer in the RTCK line will increase the TDI setup time (Tsu) time by approximately 1.1 nS and decrease the hold time (Th) by 1.1 nS relative to RTCK.
   (e) The delay induced by this buffer in the RTCK line will decrease the TDI setup time (Tsu) by approximately 1.1 nS and increase the hold time (Th) by 1.1 nS relative to RTCK.

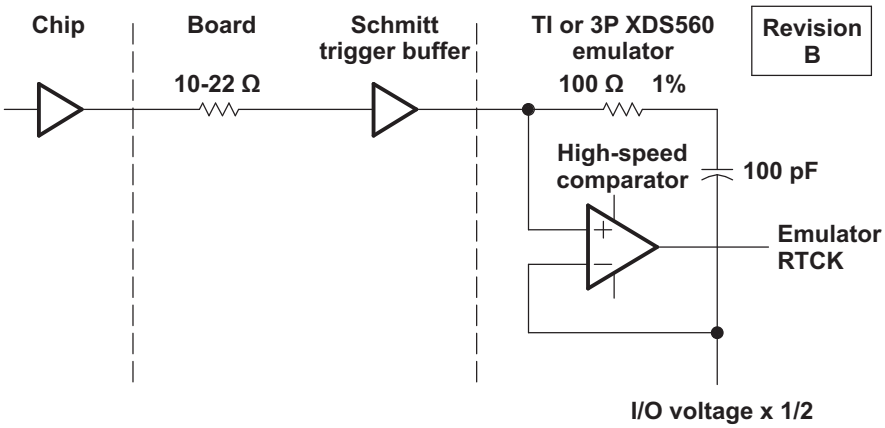


Figure 12. New Target Design with Existing Fielded Debug Controller Pods

3. Existing Target Boards/New Debug controller pods:
   When using an existing target board without any added buffering and a new debug controller pod such as TI XDS560 Revision D, XDS560T, or comparable third-party debug controller pod, the use of an external buffer board or on board buffer is not required.
4. New Target Boards/New Debug controller pods:
   (a) When designing a new target board to be used with a new debug controller pod, it is not necessary to add an additional buffer in the RTCK net on the target board. Inserting a buffer, such as the TI SN74AUC1G17, Single Schmitt-Trigger Buffer (as shown in Figure 12) will not impact performance nor the timing significantly.
   (b) The series termination resistor used should be a 10 Ω, or 22 Ω depending on net lengths, instead of the original 33 Ω value.
   (c) Figure 13 illustrates the recommended configuration for a new target and debug controller pod.

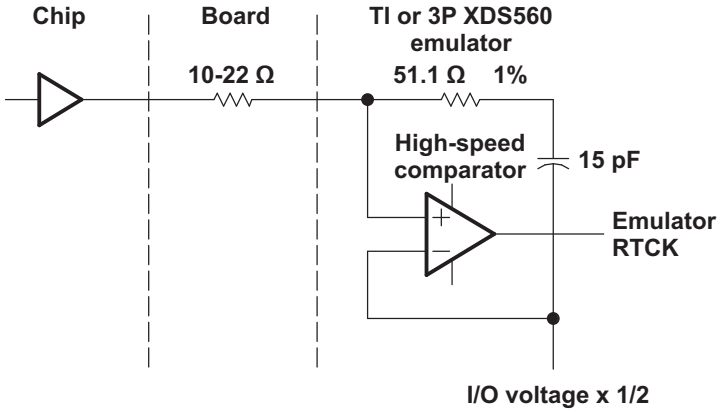


Figure 13. New Target Design with New Debug Controller Pods

Advisory 1.3.24 *ARM: Concurrent Access to ARM Internal Memory May Fail*

Revision(s) Affected: 1.3 and earlier

Details: ARM internal memory consists of two physical memories: RAM0 and RAM1. The ARM processor can access these memories over two separate busses: ITCM bus (RAM0: 0x0000 - 0x1FFF; RAM1: 0x2000 - 0x3FFF) and DTCM bus (RAM0: 0x8000 - 0x9FFF; RAM1: 0xA000 - 0xBFFF). The EDMA3.0 and other bus masters (including DSP, USB, ATA, HPI, and EMAC) are able to access RAM0 and RAM1 via the DTCM address range.

Under certain conditions, access errors may occur when the ARM, the EDMA, and other bus masters attempt to access ARM internal memory at the same time. An access error means that data is not written or read properly. Access errors may occur under the following conditions:

- When the ARM accesses RAM0 via the DTCM bus (0x8000 - 0x9FFF) and another master concurrently accesses RAM1 (0xA000 - 0xBFFF).
- When an ARM ITCM, an ARM DTCM, and a bus master (including EDMA3.0, DSP, USB, ATA, HPI, and EMAC) access are attempted to the same physical memory (RAM0 or RAM1), concurrently.

Workaround: Avoid conditions that cause access errors. Design your software so that accesses to ARM internal memory are made according to the *Bug Summary* shown in Table 10. In Table 10, P (pass) means that no bug will occur under the corresponding conditions.

Table 10. Bug Summary(1) (2)

	3 Active Accesses								2 Active Accesses												1 Active Access						
ARM ITCM access	R0	R0	R0	R0	R1	R1	R1	R1	R0	R0	R1	R1	R0	R0	R1	R1	n	n	n	n	R0	n	n	R1	n	n	n
ARM DTCM access	R0	R0	R1	R1	R0	R0	R1	R1	R0	R1	R0	R1	n	n	n	n	R0	R0	R1	R1	n	R0	n	n	R1	n	n
DMA access	R0	R1	R0	R1	R0	R1	R0	R1	n	n	n	n	R0	R1	R0	R1	R0	R1	R0	R1	n	n	R0	n	n	R1	n
Pass/Fail	F	F	P	P	P	F	P	F	P	P	P	P	P	P	P	P	P	F	P	P	P	P	P	P	P	P	P

(1) "3 active accesses" means all three (ITCM, DTCM, and DMA) accesses at the same time, "2 active accesses" means any two (ITCM, DTCM, or DMA) accesses at the same time, and "1 active access" means only one (ITCM, DTCM, or DMA) access at a time.

(2) R0 = RAM0, R1 = RAM1, n = access to neither RAM0 or RAM1, F = Fail, P = Pass

Advisory 1.3.25 *VPBE: AC Timings Differ From Data Manual Specifications*

Revision(s) Affected: 1.3 and earlier

Details: The timing parameters in Table 11 differ from those specified in the *TMS320DM6446 Digital Media System-on-Chip* data manual (literature number SPRS283D or later). Table 11 lists the differences between the data manual values and the actual values on silicon revision 1.3 and earlier.

Workaround(s): During PCB board design and layout, the AC timings specified in Table 11 should be considered when designing interfaces to the VPBE peripheral.

Table 11. Switching Characteristics Over Recommended Operating Conditions for VPBE Control and Data Output With Respect to VCLK

NO.	PARAMETER		MIN	MAX	UNIT
23	$t_{d(VCLKL-VCTLV)}$	Delay time, VCLK negative edge to VCTL valid		5.5	ns
	$t_{d(VCLKH-VCTLV)}$	Delay time, VCLK positive edge to VCTL valid		4.4	ns
24	$t_{d(VCLKL-VCTLIV)}$	Delay time, VCLK negative edge to VCTL invalid	0.9		ns
	$t_{d(VCLKH-VCTLIV)}$	Delay time, VCLK positive edge to VCTL invalid	-0.1		ns
25	$t_{d(VCLKL-DATAV)}$	Delay time, VCLK negative edge to DATA valid		5.5	ns
	$t_{d(VCLKH-DATAV)}$	Delay time, VCLK positive edge to DATA valid		4.4	ns
26	$t_{d(VCLKL-DATAIV)}$	Delay time, VCLK negative edge to DATA invalid	0.9		ns
	$t_{d(VCLKH-DATAIV)}$	Delay time, VCLK positive edge to DATA invalid	-0.1		ns

Advisory 1.3.26 *USB: Electrostatic Discharge (ESD) Sensitivity Classification*

Revision(s) Affected: 1.3 and earlier

Details: Based on JESD22-A114D, *Electrostatic Discharge (ESD) Sensitivity Testing Human Body Model (HBM)*, test results show that the TMS320DM6446 device's electrostatic discharge (ESD) sensitivity classification is Class 1B due to the two USB pins: USB_V_{DDA3P3} and USB_ID. All other pins meet the Texas Instruments design goal ESD testing classification of Class 2.

JESD22-C101C, *Field-Induced Charged-Device Model Test Method for Electrostatic-Discharge-Withstand Thresholds of Microelectronic Components*, testing was also conducted and results demonstrated that the TMS320DM6446 device's charged-device model (CDM) sensitivity classification is Class III (500 to 1000 V). These results are consistent with the Texas Instruments CDM design goal.

Workaround: Ensure that proper ESD-sensitivity device handling procedures are followed.

This advisory will be corrected or improved to meet Texas Instruments design goals in a future silicon revision.

Advisory 1.3.35 *VPBE: Video Corruption on VIDWIN1 Caused by Synchronization Issue*

Revision(s) Affected: 1.3 and earlier

Details: There is a clock synchronization issue for VIDWIN1. The synchronization issue which occurs only in the VIDWIN1 control logic causes temporary corruption of data that is observed as an occasional flicker on the display. This behavior is not seen on the VIDWIN0 or on any of the two OSD windows when VIDWIN1 is disabled. This issue is independent of whether analog or digital display output is used and impacts both progressive and interlaced modes. For interlaced systems, the issue is less frequent but not completely eliminated when using the internally sourced 27-MHz clock for the VPBE clock.

Workaround(s): To prevent this exception, use VIDWIN0; ***do not*** use VIDWIN1.

Advisory 1.3.38 *EMIFA: AC Timings Differ From Data Manual Specifications*

Revision(s) Affected: 1.3 and earlier

Details: The timing parameters in Table 12 differ from those specified in the *TMS320DM6446 Digital Media System-on-Chip* data manual (literature number SPRS283F or later or later). Table 12 lists only the differences between the data manual values and the actual values on silicon revision 1.3 and earlier.

Workaround(s): During PCB board design and layout, the AC timings specified in Table 12 should be considered when designing interfaces to the EMIFA peripheral.

Table 12. Switching Characteristics Over Recommended Operating Conditions for Asynchronous Memory Cycles for EMIFA Module(1) (2) (see Figure 6-21 and Figure 6-22)

NO.	PARAMETER		-594		UNIT
			MIN	MAX	
4	$t_{su(EMCSL-EMOEL)}$	Output setup time, $\overline{EM\_CS}$[5:2] low to $\overline{EM\_OE}$ low (SS = 0)		(RS + 1) * E + **1.2**	ns
		Output setup time, $\overline{EM\_CS}$[5:2] low to $\overline{EM\_OE}$ low (SS = 1)			ns
5	$t_{h(EMOEH-EMCSH)}$	Output hold time, $\overline{EM\_OE}$ high to $\overline{EM\_CS}$[5:2] high (SS = 0)	(RH + 1) * E - **1.3**		ns
		Output hold time, $\overline{EM\_OE}$ high to $\overline{EM\_CS}$[5:2] high (SS = 1)	**-1.4**		
7	$t_{h(EMOEH-EMBAIV)}$	Output hold time, $\overline{EM\_OE}$ high to $\overline{EM\_BA}$[1:0] invalid	(RH + 1) * E - **2.1**		ns
9	$t_{h(EMOEH-EMAIV)}$	Output hold time, $\overline{EM\_OE}$ high to $\overline{EM\_A}$[21:0] invalid	(RH + 1) * E - **2.4**		ns
17	$t_{h(EMWEH-EMCSH)}$	Output hold time, $\overline{EM\_WE}$ high to $\overline{EM\_CS}$[5:2] high (SS = 0)	(WH + 1) * E - **1.4**		ns
		Output hold time, $\overline{EM\_WE}$ high to $\overline{EM\_CS}$[5:2] high (SS = 1)	**-1.4**		
21	$t_{h(EMWEH-EMBAIV)}$	Output hold time, $\overline{EM\_WE}$ high to $\overline{EM\_BA}$[1:0] invalid	(WH + 1) * E - **2.2**		ns
23	$t_{h(EMWEH-EMAIV)}$	Output hold time, $\overline{EM\_WE}$ high to $\overline{EM\_A}$[21:0] invalid	(WH + 1) * E - **2.5**		ns

(1) RS = Read setup, RST = Read STrobe, RH = Read Hold, WS = Write Setup, WST = Write STrobe, WH = Write Hold, TA = Turn Around, EW = Extend Wait mode, SS = Select Strobe mode. These parameters are programmed via the Asynchronous Bank and Asynchronous Wait Cycle Configuration Registers and support the following range of values: TA[3:0], RS[15:0], RST[63:0], RH[7:0], WS[15:0], WST[63:0], WH[7:0], and EW[255:0]. For more information, see the *TMS320DM644x DMSoC Asynchronous External Memory Interface (EMIF) Reference Guide* (literature number SPRUE20).

(2) E = SYSCLK5 period in ns for EMIFA. For example, when running the DSP CPU at 594 MHz, use E = 10.1 ns.

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

5.1 Usage Notes for Silicon Revision 1.2

Silicon Revision 1.2 applicable usage note(s) have been found on a later silicon revision; for more detail, see Section 2.1, *Usage Notes for Silicon Revision 2.3*, of this document.

5.2 Silicon Revision 1.2 Known Design Exceptions to Functional Specifications

All known design exceptions to functional specifications for silicon revision 1.2 still apply and have been moved up to Section 2.2, *Silicon Revision 2.3 Known Design Exceptions to Functional Specifications* and to Section 4.2, *Silicon Revision 1.3 Known Design Exceptions to Functional Specifications*, of this document.

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

6.1 Usage Notes for Silicon Revision 1.1

Silicon Revision 1.1 applicable usage note(s) have been found on a later silicon revision; for more detail, see Section 2.1, *Usage Notes for Silicon Revision 2.3*, of this document.

6.2 Silicon Revision 1.1 Known Design Exceptions to Functional Specifications

All other known design exceptions to functional specifications for silicon revision 1.1 still apply and have been moved up to Section 2.2 (*Silicon Revision 2.3 Known Design Exceptions to Functional Specifications*) and to Section 4.2 (*Silicon Revision 1.3 Known Design Exceptions to Functional Specifications*) of this document.

Table 13. Silicon Revision 1.1 Advisory List

Advisory 1.1.6 *DDR2: Multiple Master Access to the DDR2 at the Same Time may Cause Master to Stop*

Revision(s) Affected: 1.1

Details: If multiple masters (CPUs or master peripherals) are accessing the DDR2 Memory Controller simultaneously and at least one of the masters is performing 64-byte burst transfers to the DDR2 Memory Controller, one of the masters may stop, requiring a power-up reset to recover.

Workaround: Reliable operation is achieved when the DDR2 Memory Controller VCLK and MCLK are set to a 1:1 ratio. The lockup is sensitive to non 1:1 frequency ratios between the clock used by the DDR2 Memory Controller (VCLK) and the clock used by the DDR2 PHY interface to the external bus (MCLK).

For more detailed information on the clocks to the DDR2 Memory Controller, see the *TMS320DM644x DMSoC DDR2 Memory Controller User's Guide* (literature number SPRUE22).

The recommended clock configurations are:

- DSP CPU clock at 486 MHz
  ARM CPU clock at 243 MHz
  VCLK at 162 MHz
  MCLK at 162 MHz
- DSP CPU clock at 567 MHz
  ARM CPU clock at 283.5 MHz
  VCLK at 189 MHz
  MCLK at 189 MHz

Note: A DDR2 clock rate of 189 MHz is only supported on silicon revision 1.1 as a workaround for this advisory. For silicon revisions 1.2 and later, see the device-specific data manual for the DDR maximum clock rate supported.

Revision History

This silicon errata revision history highlights the technical changes made to the SPRZ241M revision to make it an SPRZ241N revision.

DM6446 Revision History

SEE	ADDITIONS/MODIFICATIONS/DELETIONS
Global	Added Silicon Revision 2.3 information/data. **NOTE:** In Silicon Revision 2.3, the SPI boot mode is added as a backup to the NAND boot mode in the ROM bootloader (RBL). Additional updates have also been made to the RBL. For more information, see the *TMS320DM644x DMSoC ARM Subsystem Reference Guide* (literature number SPRUE14) and the *TMS320DM644x ROM Migration Guide* (literature number SPRAB80).
Section 2.1	Usage Notes for Silicon Revision 2.3: • Moved applicable Silicon Revision 2.1 Usage Notes to this section
Section 2.2	Silicon Revision 2.3 Known Design Exceptions to Functional Specifications: • Moved applicable Silicon Revision 2.1 Advisories to this section

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

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