TMS320DM355
Digital Media System-on-Chip (DMSoC)
Silicon Revision 1.1, 1.3, and 1.4

Silicon Errata

Literature Number: SPRZ264E
September 2007–Revised July 2010

List of Figures

List of Tables

TMS320DM355 Digital Media System-on-Chip (DMSoC) **Silicon Revision 1.1, 1.3, and 1.4**

1 Introduction

This document describes the known exceptions to the functional specifications for the TMS320DM355 Digital Media System-on-Chip (DMSoC). [See *TMS320DM355 Digital Media System-on-Chip Data Manual* (literature number SPRS463).] Throughout this document, TMS320DM35x and DM35x refer to the TMS320DM355 device.

For additional information, see the latest version of the *TMS320DM35x DMSoC Peripherals Overview Reference Guide* (literature number SPRUFC8).

The advisory numbers in this document may not be sequential. Some advisory numbers may be moved to the next revision and others may 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., ***TMS320DM355***). 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

Figure 1 provide examples of the ***TMS320DM355*** and device markings. The device revision can be determined by the symbols marked on the top of the package. Some prototype devices may have markings different from those illustrated.

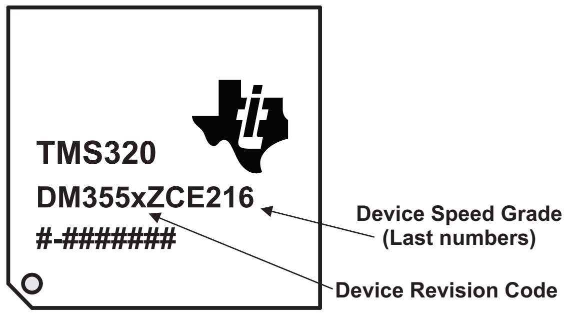

Figure 1. Example, Device Revision Codes for TMS320DM355 (ZCE)

NOTES:

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

Silicon revision is identified by a code on the chip as shown in Figure 1. If x is "blank", then the silicon is revision 1.1. Table 1 lists the silicon revisions associated with each device revision code for the DM355 device.

Table 1. Device Silicon Revisions[(1)]

Device Revision Code (x)	SILICON REVISION	COMMENTS
(blank)	Indicates Revision 1.1	TMX320DM355ZCE216, TMX320DM355ZCE270, TMS320DM355ZCE135, TMS320DM355ZCE216, TMS320DM355ZCE270, TMS320DM355ZCEA216, TMS320DM355ZCEA135
C	Indicates Revision 1.3	TMS320DM355CZCE135, TMS320DM355CZCE216, TMS320DM355CZCE270, TMS320DM355CZCEA13, TMS320DM355CZCEA21
D	Indicates Revision 1.4	TMS320DM355DZCE135, TMS320DM355DZCE216, TMS320DM355DZCE270, TMS320DM355DZCE27J, TMS320DM355DZCEA13, TMS320DM355DZCEA21

(1) Silicon revision 1.2 is not offered for TMS320DM355.

2 Silicon Revision 1.4 Usage Notes and Known Design Exceptions to Functional Specifications

2.1 Usage Notes for Silicon Revision 1.4

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 ROM Bootloader (RBL) Functionality

The only difference between silicon revision 1.1 and silicon revisions 1.3 and later is the replacement of the ROM Boot Loader (RBL) with a different version with different functional behavior. There are no other functional changes to the device.

The Rom Boot Loader (RBL) is firmware which is stored in ROM on the DM355. It is responsible for handling the boot process. When the boot process is initiated, it will sense the state of the BOOTSEL[1:0] pins and (based on that state) loads a User Boot Loader (UBL) from external media and branches to the entry point of the UBL. On silicon revisions 1.3 and later, the ROM Boot Loader (RBL) supports three boot modes: NAND Boot, SD/MMC Boot, and UART Boot. On silicon revision 1.4, SPI boot mode has been added which is executed in NAND boot mode. If SPI fails (or there is no EEPROM), NAND boot mode is executed.

Silicon revisions 1.3 and later support the following changes to NAND boot functionality:

- Support for 4K and 8K page size devices has been added
- The NAND device ID table has been extended
- Additional devices not in the Device ID table are supported through fetching characteristics from registers stored on the NAND
- The assumed layout of the data in the boot NAND has changed
- The UART boot functionality has changed
- A bug in the ECC error correction has been fixed

Silicon revision 1.4 have added the following changes to NAND boot functionality:

- Support for 16K page size devices has been added
- ONFI and Samsung 4th byte support
- NAND parameters/geometry information from the SPI EEPROM
- The RBL Code 4bit ECC Mode 4 error detection limitation has been fixed

Support for 4K, 8K, and 16K NAND Devices Added

The Rom Boot Loader (RBL) used in silicon revision 1.1 supported NAND page sizes of 512 bytes and 2048 bytes. The RBL used in silicon revision 1.3 support page sizes of 512 bytes, 2048 bytes, 4096 bytes, and 8192 bytes. Silicon revision 1.4 also support 16384 bytes page sizes.

Note: At the time of documentation for this device, 8192-byte and 16384-byte devices were not available for testing. The code does contain support for these devices; however, it has not yet been tested.

The Rom Boot Loader (RBL) used in silicon 1.1 and later versions all use magic numbers to identify NAND layout and boot options. A magic number of the form 0xA1AC EDxx was used in revision 1.1. Using a magic number in this form for revisions 1.3 and later will place the DM355 in compatibility mode. This will force the device to operate exactly as if it were silicon revision 1.1. You cannot use the larger NAND devices when operating in compatibility mode. Using a magic number of the form 0xA1BC EDxx will place the DM355 in standard mode. Standard mode for revisions 1.3 and later allows for the use of the larger NAND devices. For more information on specific uses of magic numbers, see the *TMS320DM355 Digital Media System-on-Chip Data Manual* (literature number SPRS463).

NAND Device ID Table Updated

The RBL contains an internal table containing a list of known NAND devices. This list has been updated on Silicon Revisions 1.3 and later. Table 2 and Table 3 show the devices contained in the tables.

Table 2. NAND Devices in NAND Device ID Table (Silicon Revision 1.1)

Device ID	Pages per Block	Bytes pr Page	Block Shift value for address	Number of address cycles
0xE3	16	512+16	12	3
0xE5	16	512+16	12	3
0xE6	16	512+16	12	3
0x39	16	512+16	13	3
0x6B	16	512+16	13	3
0x73	32	512+16	13	3
0x33	32	512+16	13	3
0x75	32	512+16	13	3
0x35	32	512+16	13	3
0x43	32	512+16	13	4
0x45	32	512+16	13	4
0x53	32	512+16	13	4
0x55	32	512+16	13	4
0x76	32	512+16	13	4
0x36	32	512+16	13	4
0x79	32	512+16	13	4
0x71	32	512+16	13	4
0x46	32	512+16	13	4
0x56	32	512+16	13	4
0x74	32	512+16	13	4
0xF1	64	2048+64	22	4
0xA1	64	2048+64	22	4
0xAA	64	2048+64	22	5
0xDA	64	2048+64	22	5
0xAC	64	2048+64	22	5
0xDC	64	2048+64	22	5
0xB1	64	2048+64	22	5
0xC1	64	2048+64	22	5

Table 3. NAND Devices in NAND Device ID Table (Silicon Revisions 1.3 and Later)

Device ID	Pages per Block	Bytes pr Page	Block Shift value for address	Number of address cycles
0xE3	16	512+16	12	3
0xE5	16	512+16	12	3
0xE6	16	512+16	12	3
0x6B	16	512+16	13	3
0x73	32	512+16	13	3
0x33	32	512+16	13	3
0x75	32	512+16	13	3
0x35	32	512+16	13	3
0x43	32	512+16	13	4
0x45	32	512+16	13	4
0x53	32	512+16	13	4
0x55	32	512+16	13	4
0x76	32	512+16	13	4

Table 3. NAND Devices in NAND Device ID Table (Silicon Revisions 1.3 and Later) (continued)

Device ID	Pages per Block	Bytes pr Page	Block Shift value for address	Number of address cycles
0x36	32	512+16	13	4
0x79	32	512+16	13	4
0x71	32	512+16	13	4
0x46	32	512+16	13	4
0x56	32	512+16	13	4
0x74	32	512+16	13	4
0xF1	64	2048+64	22	4
0xA1	64	2048+64	22	4
0xAA	64	2048+64	22	5
0xDA	64	2048+64	22	5
0xAC	64	2048+64	22	5
0xDC	64	2048+64	22	5
0xB1	64	2048+64	22	5
0xC1	64	2048+64	22	5

If the NAND device is not found in this table, then the RBL will read the fourth byte of the NAND ID (stored on the actual device) and attempt to decode this to obtain the necessary parameters.

For the purpose of determining NAND block size and page size the information from the fourth byte is considered as follows:

- Bits 5 and 4 determine the block size
  - Bits 5,4 = 00: 64KB
  - Bits 5,4 = 01: 128KB
  - Bits 5,4 = 10: 256KB
  - Bits 5,4 = 11: 512 KB
- Bits 1 and 0 determine the page size
  - Bits 1,0 = 00: 1KB
  - Bits 1,0 = 01: 2KB
  - Bits 1,0 = 10: 4KB
  - Bits 1,0 = 11: 8KB

In silicon revision 1.4, the latest Samsung (manufacturer ID: 0xEC) 4th ID definition has been added which is as follows:

- Bits 5 and 4 determine the block size
  - Bits 5,4 = 00: 128KB
  - Bits 5,4 = 01: 256KB
  - Bits 5,4 = 10: 512KB
  - Bits 5,4 = 11: 1024KB
- Bits 1 and 0 determine the page size
  - Bits 1,0 = 00: 2KB
  - Bits 1,0 = 01: 3KB
  - Bits 1,0 = 10: 4KB
  - Bits 1,0 = 11: Reserved

NAND Layout Changes on Silicon Revisions 1.3 and Later

The position of data on the NAND device assumed by the Rom Boot Loader (RBL) has changed. Table 4 shows the layout for Silicon Revision 1.1, and Table 5 shows the layout for Silicon Revisions 1.3 and later.

Table 4. NAND Layout Used by Silicon Revision 1.1

512-Byte Page Size	2048-Byte Page Size
512 bytes Data	512 bytes Data
16 bytes ECC Data	16 bytes ECC Data
	512 bytes Data
	16 bytes ECC Data
	512 bytes Data
	16 bytes ECC Data
	512 bytes Data
	16 bytes ECC Data

Table 5. NAND Layout Used by Silicon Revisions 1.3 and Later(1)

512-Byte Page Size	2048-Byte Page Size	4096-Byte Page Size	8192-Byte Page Size
512 bytes Data	2048 bytes Data	4096 bytes Data	8192 bytes Data
16 bytes ECC Data	64 bytes ECC Data	128 bytes ECC Data	256 bytes ECC Data

(1) Silicon Revisions 1.3 and later assume that all the data on a NAND page is in a single block with the ECC and bad block data contained in the spare bytes area.

UART Boot Functionality in Silicon Revision 1.1

The boot sequence used in silicon revision 1.1 is given below:

1. The RBL will repeatedly send a “BOOTME” string until it receives a correct response from the host.
2. The host will send
   - “ ACK\0”
   - UBL Checksum
   - UBL size in bytes
   - UBL physical start address
   - "0000”
3. The DM355 will send “ BEGIN”
4. The Host will send the CRC-32 lookup table
5. The DM355 will verify the checksum for the lookup table and send “DONE” if the checksum is correct. If the checksum is bad, the DM355 will send “CORRUPT” and branch back to step 1.
6. The Host will send the UBL
7. The DM355 will verify the checksum for the UBL and send DONE if the checksum is correct. If the checksum is bad, the DM355 will send “CORRUPT” and branch back to step 1.
8. The DM355 will branch to the UBL start address.

UART Boot Functionality in Silicon Revisions 1.3 and Later

There are two functional changes to the UART boot mode:

- The RBL tested for a terminating '\0' in the " ACK" string sent in step 2a. The RBL only tests for 7 characters in silicon revisions 1.3 and later, so the terminating '\0' is ignored.
- If the “0000” string in step 2e is replaced by a “0001” string, then the DM355 will not check the checksum for the CRC-32 lookup table until the UBL has been sent. After the CRC-32 lookup table and the UBL have been sent, the DM355 will check both and send two “DONE” strings. The flow in this case will be:
  1. The RBL will repeatedly send a “BOOTME” string until it receives a correct response from the host.
  2. The host will send
     - “ ACK\0”
     - UBL Checksum
     - UBL size in bytes
     - UBL physical start address

- "0001"
- The Host will send the CRC-32 lookup table
- The Host will send the UBL
- The DM355 will verify the checksum for the lookup table and send “DONE” if the checksum is correct. If the checksum is bad, the DM355 will send “CORRUPT” and branch back to step 1.
- The DM355 will verify the checksum for the UBL and send DONE if the checksum is correct. If the checksum is bad, the DM355 will send "CORRUPT" and branch back to step 1.
- The DM355 will branch to the UBL start address

ROM Version ID

The ROM ID is stored at address 0x00009FFC and is four bytes long. To identify the silicon revision, examine the values at this address and compare with Table 6.

Table 6. ROM Version IDs

ROM Version	ID Stored at 0x00009FFC
Silicon Revision 1.1	0x10040101
Silicon Revision 1.3	0x10040103
Silicon Revision 1.4	0x10040104

ECC Bug fixed

The Rom Boot Loader (RBL) used in silicon revision 1.1 contained a bug which prevented the ECC hardware from detecting and correcting bit stream errors while the UBL was read from the NAND device. This bug has been fixed in silicon revision 1.4.

2.1.2 Possible Emulator Crash If TCK Frequency Is Greater than MXI Frequency

If the frequency of TCK is greater than the frequency of MXI, there is a chance the emulator will crash when changing the PLL1 clock frequency. This can happen while stepping through code that configures the PLL1 controller or while using a GEL function that configures the PLL1 controller.

Additionally, the act of applying reset could cause the clock on the TCK pin to stop being generated, thereby crashing the emulator. This condition happens if the TCK frequency is 2x or greater than the MXI frequency during reset.

To avoid both of these issues, the MXI clock frequency must be greater than or equal to the TCK clock frequency.

2.1.3 Incorrect Pin Descriptions in Original DM355 Data Sheet

The pin definitions and descriptions given in the original version of the DM355 data sheet (SPRS463) were incorrect in several places. These issues have been corrected in revision A of the *TMS320DM355 Digital Media System-on-Chip (DMSoc) Data Sheet* (SPRS463). Customers are advised to use the updated pin information in revision A of the DM355 data sheet and disregard the pin information given in the original version of the data sheet.

2.1.4 DM355 EVM $V_{SS\_USB\_REF}$ Pin Not Connected As Specified

On some versions of the DM355 EVM, the $V_{SS\_USB\_REF}$ pin is not connected as specified in the data sheet. This ***does not*** create any USB functional issues; however, for USB compliance, DM355 designs ***must*** follow the pin connection specified in the USB Reference Resistor Routing figure in the ***TMS320DM355 Digital Media System-on-Chip (DMSoC)*** data sheet (literature number SPRS463).

2.1.5 SD/SDIO card: How to Read M bytes (M=1, 2, 3) from SD or SDIO card

Direction: Read from SD or SDIO

Data size: 32*N+M byte (where M=1, 2, 3 and N=0, 1, 2, 3,...) FIFO size is 32 bytes (ACCWD (FIFOCTL [4:3] = 0), FIFO trigger level is 256-bits (FIFOLEV (MMCFIFOCTL [2]) =1)

Reading from SD or SDIO when FIFO Trigger Level FIFOLEV is 256 bits causes the DRRDY(MMCST0[10]) bit not to be set for the last M bytes of the above data size equation and therefore the DMA read event and CPU interrupt for RRDY(MMCST0[10]) are not getting asserted, but the data in FIFO is correct.

There are two possible methods to work around this limitation.

1. Include SD status checking and/or SD interrupt enable for TRNDNE (MMCST0 [12]) or DATDNE (MMCST0 [0])
2. Use 128-bit FIFO trigger level (i.e. FIFOLEV (MMCFIFOCTL [2]) =0), since this is only a problem when FIFOLEV is 256 bits.

Note:

1. To ensure the start of the MMCSD transfer correctly, the DMATRIG bit (MMCCMD [16]) should always be set to ‘1’ when writing the command to MMCCMD registers.
2. For SDIO read and write function, the MMCBLEN and MMCNBLK registers have to be set to equal the total byte count and total block count configured by the SDIO command (CMD52 or CMD53). For example, to read 3 bytes using CMD53, the software has to set MMCBLEN=3 and MMCNBLK=1.
3. The FIFOFUL and FIFOEMP status bits are set based on access size(ACCWD):
   - If the number of bytes stored in the FIFO is smaller than access size, FIFOEMP is 1. Otherwise, FIFOEMP is 0.
   - If the size of the remaining empty spaces in FIFO is smaller than access size, FIFOFUL is 1. Otherwise, FIFOFUL is 0.

   For example, when the access size (ACCWD) is 4 bytes and the FIFO level (FIFOLEV=1) is 256-bits or 32 bytes, and if the number of bytes in FIFO is

   - 0 to 3: then, FIFOFUL=0, FIFOEMP=1
   - 4 to 28: then, FIFOFUL=0, FIFOEMP=0
   - 29 to 32: then, FIFOFUL=1, FIFOEMP=0

2.1.6 SD/SDIO card: How to Handle SDIO interrupt

SDIO interrupt may be missed since SDIO interrupt processing is only done based on the IOINT (SDIOIST [0]) status.

SDIO Interrupt Detecting: SDIO interrupt is a level interrupt on SDIO protocol, but the interrupt generation logic detects the edge of DAT1 signal inside the host controller.

SDIO Interrupt Masking: The SDIO Interrupt may be enabled or disabled by the SDIO stack at any time.

SDIO Interrupt Status Clearing: SDIO interrupt status may not be cleared in an atomic sequence since a pending interrupt has to be cleared not only on the host controller but also on the SDIO module.

To guarantee that the SDIO interrupt is not missed, the host controller has to check the IOINT (SDIOIST[0]) status register as well as sample the DAT1 signal by checking the DAT1(SDIOST0[1-0]) status register at a certain condition.

The following is a suggested sequence for properly handling the SDIO interrupt:

1. SDIO stack informs the SDIO host controller to enable or unmask the SDIO interrupt.
2. SDIO host controller enables or unmasks the SDIO interrupt. Before enabling the SDIO interrupt, the SDIO host controller software has to first sample the DAT1 signal (INTPRD==1 && DAT1==1) to make sure that the pending interrupt is reported to the SDIO stack and then to the SDIO client/function driver.

   Note: SDIO interrupt is enabled at the request of the client/function driver of the SDIO stack. SDIO interrupt needs to be enabled both on the host controller and the SDIO card which may not happen in an atomic way. By the time the host controller enables the interrupt, the SDIO interrupt may be already pending. The SDIO controller can not detect interrupt pending before the interrupt is enabled on the controller.

3. When the SDIO interrupt is detected by the SDIO controller, the SDIO ISR has to process the SDIO interrupts in the following sequence:
   - Check and clear SDIO interrupt status IOINT(SDIOIST[0]) immediately.
   - Mask the SDIO interrupt on the SDIO host controller by setting IOINTEN (SDIOIEN[0]) =0.
   - Notify the Interrupt event first to the SDIO stack and then to the SDIO client/function driver.

   Note: The clearing of the SDIO interrupt on the SDIO card and SDIO controller does not happen in an atomic way. To mask the SDIO interrupt, it has to be ensured that the interrupt is not mistakenly detected by the host controller again. In addition, the client/function driver of the SDIO stack can therefore control the occurrence of interrupt and also the readiness to process the interrupt.
4. The SDIO stack client/function driver clears the SDIO interrupt status on the SDIO card.

2.1.7 Peripherals: Electrostatic Discharge (ESD) Sensitivity Classification

JESD22-A114D, Electrostatic Discharge (ESD) Sensitivity Testing Human Body Model (HBM), test results indicate that the TMS320DM355 device's electrostatic discharge (ESD) sensitivity classification is Class 0 due to 4 reserved pins (BGA ID: J1, K1, L1, M1). All other pins meet the Texas Instruments design goal ESD testing classification of Class 2. No workaround is required. Standard ESD-sensitivity device handling procedures provide sufficient protection.

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 TMS320DM355 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.

2.1.8 ASP: Transfers Should be Buffered in Internal Memory

Audio Serial Port (ASP) transfers may need to originate and complete from on-chip buffers in ARM Internal RAM (TCM). This is due to the fact that there is no tolerance for audio data dropouts that may occur due to the delays in DDR2/mDDR accesses from other masters and from unavoidable DDR2/mDDR 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/mDDR latencies. DDR2/mDDR latencies are system-dependent, varying between applications, and are impacted by the amount and type of data traffic to DDR2/mDDR memories. Once completed, the data can be shuttled between the internal buffer and the DDR2/mDDR memory by using EDMA Q1/TC1.

If using on-chip buffers for ASP transfers, also see the following advisories:

- 1.1.2 Concurrent Access to ARM Internal Memory May Result in Access Errors

2.1.9 GIO0 Low Setting During Device Boot May Cause Boot to Fail

If the GIO0 pin is low during device boot from a NAND device, the DM355 will not follow the normal boot process. The DM355 ROM Boot Loader (RBL) will attempt to read the User Boot Loader (UBL) header from a different NAND page range and will look for a different number in that header than that used in the normal boot process, which may cause the boot to fail.

To prevent this issue from occurring, ensure that the GIO0 pin is held high during NAND boot. Details about this issue will be incorporated into the next revision of the *TMS320DM355 Digital Media System-on-Chip (DMSoC)* Data Manual (literature number SPRS463).

2.2 Silicon Revision 1.4 Known Design Exceptions to Functional Specifications

Table 7. Silicon Revision 1.4 Advisory List

Title **Page**

Advisory 1.4.1 *H3A data may get corrupted*

Revision(s) Affected 1.4 and earlier.

Details This problem affects H3A auto white balance, auto exposure, and auto focus when the image sensor data is large (i.e. greater than approximately 10 mega pixels) and the data path is from imager to CCDC to H3A .

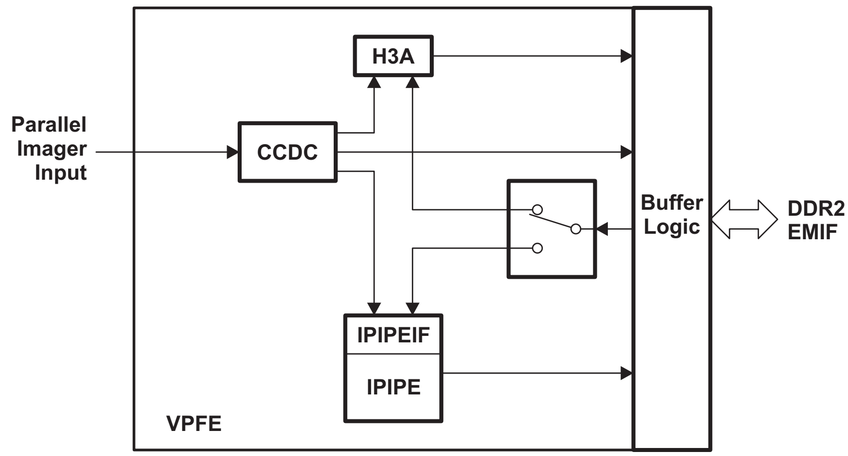

Figure 2. Video Processing Front End (VPFE) Block Diagram and Data Flows

When the H3A module receives more than 4096 pixel clock cycles between consecutive HD rising edges, the H3A internal line buffer may become corrupt. The internal line buffer address is reset to zero at HD rising edge, is incremented every pixel clock cycle, and wraps back to address zero after 4096 pixel clocks. H3A calculations may not be correct when the line buffer is corrupt.

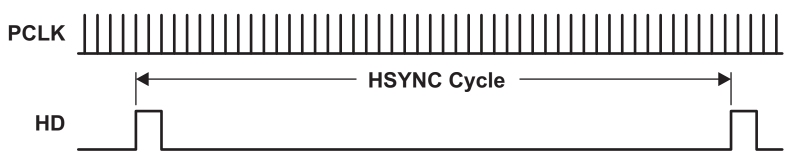

Figure 3. HSYNC Cycle Period

Workaround(s) To workaround this problem, use any of the four workarounds described below.

1. Constrain the valid AEW/AF data area. The valid data in the internal line buffer will not become corrupt under certain constraints. In particular, the problem will not occur when you adhere to the follow constraints (see figures):
   - B1 + V < 4096 and
   - B2 <= 4096 – (B1+V) + B1

   Where
   - B1 = Data area from HD rising edge to start of valid AEW/AF data.
   - V = Valid AEW/AF data area. This is the data area for AEW/AF calculations. This area is specified using registers in the H3A module. Note that the registers are different for AEW and for AF. For AEW, bits WINSH specify the horizontal start position of the AEW windows, bits WINW specify the horizontal width of the windows, and bits WINHC specify the total number of windows. The total number of pixels in the valid AEW data area is equal to V=WINW*WINHC. For AF, bits IIRSH specify the horizontal start position of the IIR filter, bits PAXSH specify the horizontal start position of the AF paxels, bits PAXW specify the horizontal width of the paxels, and bits PAXHC specify the total number of paxels. The total number of pixels in the valid AF data area is equal to V=[(PAXSH - IIRSH) + PAXW*PAXHC]. See the VPFE PRG for complete bit descriptions.
   - B2 = Data area from end of valid AEW/AF data to next HD rising edge.

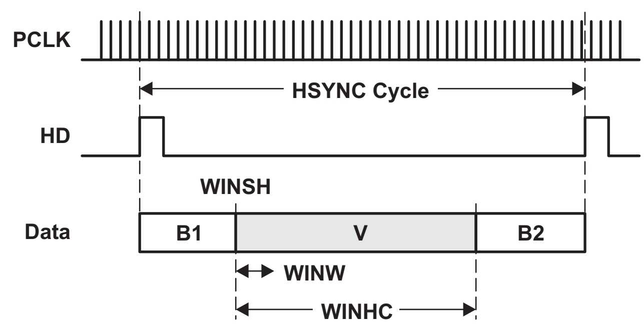

Figure 4. Valid Window Width and Horizontal Window Count Data Area

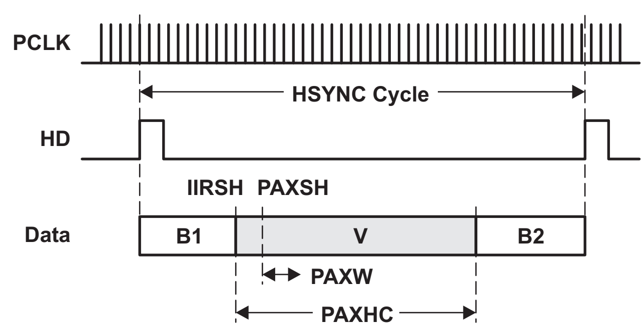

Figure 5. Paxel Width and Horizontal Paxel Count

2. Use the imager's movie readout mode.
   In the imager's movie readout mode the problem does not occur, because the number of horizontal pixels and blacking are is less than 4096 pixels and the H3A module receives less than 4096 pixel clock cycles between consecutive HD rising edges.
   Note that the number of pixel clock cycles between consecutive HD rising edges that the H3A module receives is determined by the external or internal timing generator and also, if used, by the reformatter inside the CCDC. The reformatter is used when imagers have special readout patterns. For example, if the readout pattern contains

two lines of actual image data per HSYNC, then the reformatter will convert the data from one to two lines, internally. The number of pixel clock cycles between consecutive HD rising edges that the H3A modules receives corresponds to the timing at the output of the reformatter.

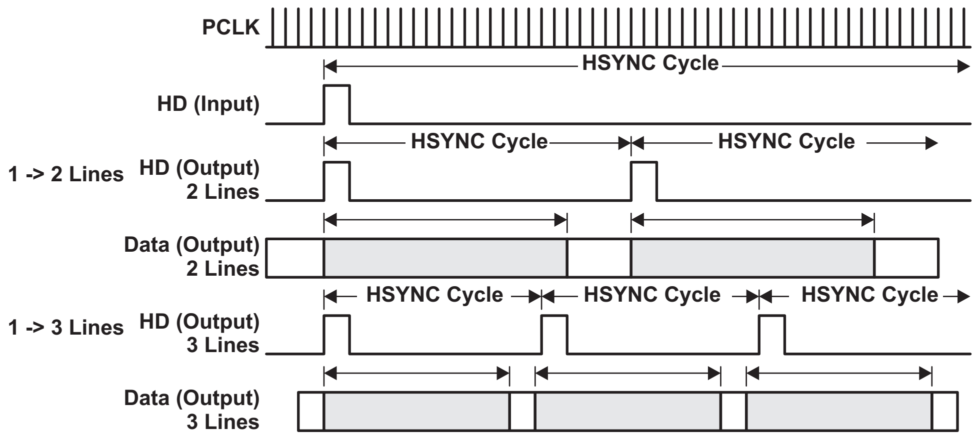

Figure 6. Readout Patterns Comparing Timing at the Output of the Reformatter

3. Use the alternative data path for H3A.
   There are two possible data paths for H3A:
   (a) Imager->CCDC->H3A
   (b) Imager->CCDC->DDR->H3A
   This problem is only applicable when the path is Imager->CCDC->H3A, so you may avoid this problem completely by using the alternative path Imager->CCDC->DDR->H3A. Note that the alternative path requires raw image data to be store in DDR memory prior to H3A processing.

Advisory 1.4.2 *Concurrent Access to ARM Internal Memory May Result in Access Errors*

Revisions Effected 1.4 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 and DTCM bus. The EDMA and USB module are DMA bus masters that can access these memories over the DMA bus, via special bus arbiters. See the *TMS320DM355 ARM Subsystem* Reference Guide, (literature number SPRUFB3). Under certain conditions, access errors may occur when the ARM and these DMA bus masters attempt to access ARM internal memory at the same time. An access error means that data is not written or read properly. The conditions depend on whether ARM internal memory access is configured for 0 or 1 wait states and are further described below. Use bits AIM_WAIST in the MISC register in the System Control Module to configure the wait states [see *TMS320DM355 ARM Subsystem* Reference Guide, (literature number SPRUFB3)].

For both the 0 and 1 wait state configurations, access errors may occur under the following conditions (see Table 8 and Table 9):

- Access errors may occur in situations where an ARM DTCM access to RAM0 and a DMA bus access to RAM1 are attempted at the same time.
- Access errors may occur in situations where an ARM ITCM, an ARM DTCM, and a DMA bus access are attempted to the same physical memory (RAM0 or RAM1) at the same time.

For only the 1 wait state configuration, access errors may occur under the following conditions:

- Access errors may occur in situations where an ARM ITCM and a DMA bus access are attempted to the same physical memory (RAM0 or RAM1) at the same time.
- Access errors may occur in situations where an ARM DTCM and a DMA bus access are attempted to the same physical memory (RAM0 or RAM1) at the same time.

Workaround(s)

Avoid the conditions that cause accesses errors. Design your software so that accesses to ARM internal memory are according to the Bug Summary Table 8 and Table 9). In the tables, P means that no bug will occur under the corresponding conditions.

Table 8. Bug Summary for the 0 Wait State Configuration

	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

Table 9. Bug Summary for the 1 Wait State Configuration

	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	F	F	F	F	P	F	P	P	P	P	F	P	P	F	F	F	P	F	P	P	P	P	P	P	P

Advisory 1.4.3 — *SPI: Receive Overrun Interrupt and Bit Error Can be Lost*

Revision(s) Affected: 1.4 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: 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 1.4.4 — *SPI: RXINTFLG Bit in SPIFLG Register May Not Get Cleared*

Revision(s) Affected: 1.4 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: 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 1.4.5 — *SPI: A Write to SPIFLG Receiver Overrun Bit Does Not Clear the Flag*

Revision(s) Affected: 1.4 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: 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 1.4.6 — *SPI: SPIINTVECT and SPIFLG Registers are Cleared When Read in Debug Mode*

Revision(s) Affected 1.4 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 1.4.7 — *SPI: SPI Master Receives Extra Bit When SPICLK Polarity Changes*

Revision(s) Affected 1.4 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 1.4.8 *SPI Master Mode: Extra Step Required to Use CSHOLD*

Revision(s) Affected 1.4 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 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 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 7:

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

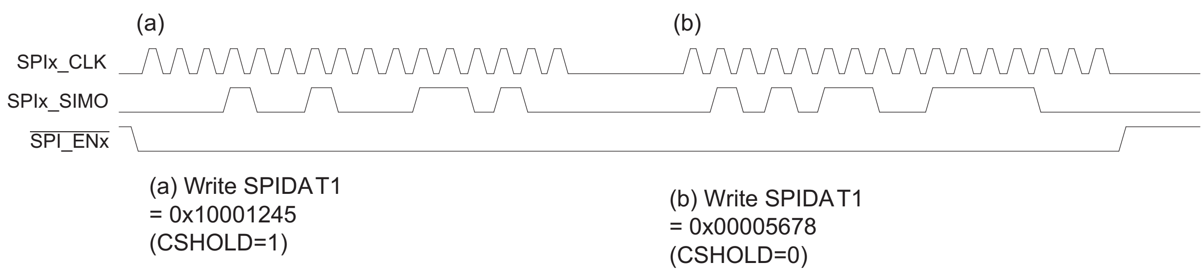

Figure 7. Expected CSHOLD Behavior

Instead, what actually occurs is that SPI_ENx is momentarily de-asserted at the beginning of the second write, as illustrated in Figure 8.

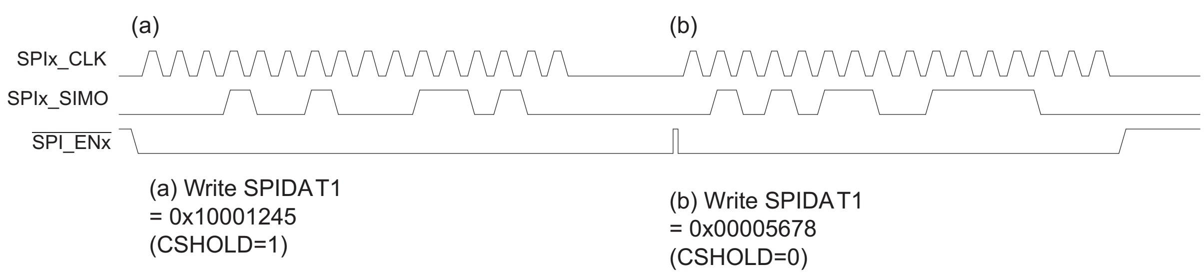

Figure 8. Actual CSHOLD Behavior–32-Bit Writes to SPIDAT1

Both Figure 7 and Figure 8 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 9. This is the same case illustrated in Figure 8 except that the de-assertion of SPI_ENx lasts for the duration between writing a '0' to the CSHOLD field and writing new data to the transmit shift register.

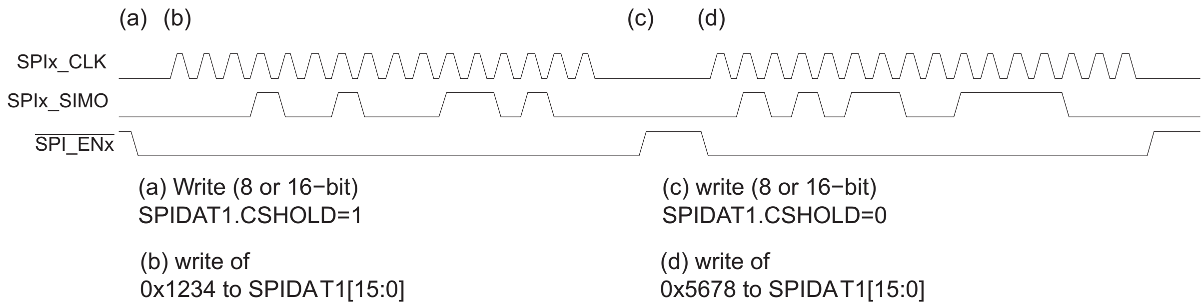

Figure 9. Actual CSHOLD Behavior–Halfword Writes to SPIDAT1

Workaround(s)

For each word in the sequence of words during which 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 SPI_ENx. See Figure 10 for an illustration.

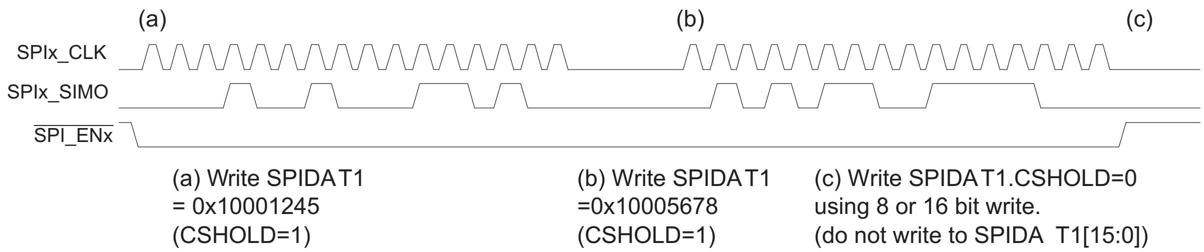

Figure 10. Workaround Assuming 32-Bit Writes to SPIDAT1 Followed by a Write Only to CSHOLD

Alternatively, only write to the SPIDAT1 CSHOLD field before and after the transfer to toggle the 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 11.

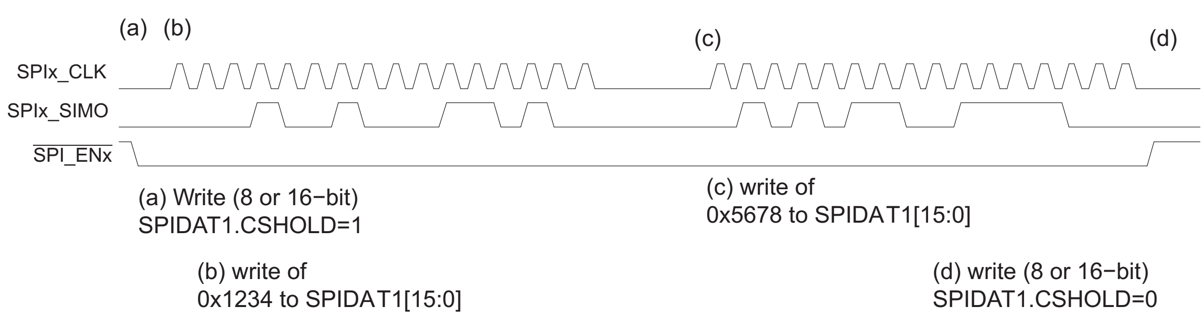

Figure 11. Workaround Assuming Halfword Writes to SPIDAT1

Advisory 1.4.9 *USB: Some Electrical Parameters Violate USB Specification*

Revision(s) Affected 1.4 and earlier.

Details Some electrical characteristics violate the USB 2.0 specification; see Table 10.

Table 10. USB Electrical Characteristics in Violation

		USB SPECIFICATION		DM355 SPECIFICATION		UNIT
		MIN	MAX	MIN	MAX	
V_{HSDSC}	USB high-speed disconnect detection threshold (differential signal amplitude)	525	625	525	675	mV
V_{BUS}	USB external charge pump input	4.75	5.25	4.85	5.25	V

Workaround(s) Consider these violations and design your system accordingly.

Advisory 1.4.10 *VPBE: VENC Default Luma Interpolation Filter Does Not Clip to Zero*

Revision(s) Affected 1.4 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 luma 2x interpolation filter (VMISC.YUPF = 0). Instead, use the alternate luma 2x interpolation filter (VMISC.YUPF = 1).

Advisory 1.4.11 *USB (Device Mode): Calculated CRC Value Does Not Match Host CRC Value*

Revision(s) Affected: 1.4 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 1.4.12 *DEVICE_ID System Register DEVREV Bits [31:28] Show Incorrect Device Silicon Revision*

Revision(s) Affected 1.4 and earlier.

Details Ideally, the DEVREV Bits [31:28] of this DEVICE_ID register (located at address 0x01c40028) should reflect the device silicon revision as listed below.

DEVREV Bits [31:28]	Silicon Revision
0001	Silicon Revision 1.1
0011	Silicon Revision 1.3
0100	Silicon Revision 1.4

However, on Silicon Revision 1.4 and earlier, the DEVICE_ID register contains the value of 0x0B73E02F, incorrectly reflecting the DEVREV Bits [31:28] as being ‘0000’.

Workaround(s) Silicon revisions 1.4 and earlier only include ROM fixes; hence a workaround to identify the device silicon revision is to read an internal memory location in ARM ROM (at address 0x00009FFC) which holds the ROM Version ID. The association between the device silicon revisions and the ROM version ID is listed below.

Silicon Revision	ROM Version ID
Silicon Revision 1.1	0x10040101
Silicon Revision 1.3	0x10040103
Silicon Revision 1.4	0x10040104

Advisory 1.4.13 *Change in ROM Boot Loader (RBL) NAND Boot Device ID Table*

Revision(s) Affected 1.4 and 1.3

Details The following NAND device ID table entry described in Table 11 has been removed from the lookup table (in the RBL for NAND bootmode) in Revision 1.3 and 1.4 (as compared to Rev 1.1).

Table 11. NAND Device ID

Device ID	Number of pages per block	Bytes per page (including extra data)	Block shift value (For address)	No. of address cycles
0x39	16	512 + 16	12	3

The table entry provides pre-defined NAND parameters indexed by device ID (in this case 0x39). These parameters are used for the RBL to communicate with the NAND device.

As a result of this missing table entry, RBL will read the fourth byte of the NAND ID (stored on the actual device) and attempt to decode this to obtain the necessary parameters. In silicon revision 1.4 a SPI EEPROM can be used if needed to store the NAND parameters to be read by the RBL.

Workaround(s) A workaround for this issue would be to use a SPI EEPROM (supported with Rev 1.4) to store the NAND parameters.

Another recommendation would be to use a different ROM boot mode, such as SPI (as part of NAND bootmode in Rev 1.4), MMC/SD, AEMIF (OneNAND) or UART) to load a secondary boot loader that can correctly access the NAND and load the remaining system software from the NAND device.

This limitation has no impact on any of the other boot modes: MMC/SD boot mode, AEMIF/OneNAND boot mode, UART boot mode.

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

3.1 Usage Notes for Silicon Revision 1.3

Silicon revision 1.3 applicable usage notes have been found on a later silicon revision. For more details, see Section 2, *Usage Notes for Silicon Revision 1.4*.

3.2 Silicon Revision 1.3 Known Design Exceptions to Functional Specifications

Some silicon revision 1.3 applicable advisories have been found on a later silicon revision. For more details, see Table 7, *Silicon Revision 1.4 Known Design Exceptions to Functional Specifications*.

Table 12. Silicon Revision 1.3 Advisory List

Advisory 1.3.1 *Bootloader: RBL Code 4bit ECC Mode Limitation*

Revision(s) Affected 1.3 (not present on silicon revision 1.4)

Details There is an issue with ROM NAND 4bit ECC mode.

The NAND code in the ROM should read 512-byte data chunks from NAND, and the AEMIF ECC hardware has the capability to correct up to 4 bit errors per 512 bytes of data. However, when 5 or more ECC errors are detected for a certain chunk of 512 bytes of data, existing TI software components (UBL, U-Boot and kernel) and the RBL will not read the NANDERRADD1/NANDERRADD2 registers. As a result, the 4BITECC_ADD_CALC_START bit in NANDFCR register is not cleared.

If there are any ECC errors in the ensuing chunks of 512 bytes of data, no ECC corrections will take place. The software workaround for this is that when the NAND driver detects 5 bit errors, the driver will perform a dummy read of the NANDERRVAL1 register or NANDERRADD1 register. This clears the 4BITECC_ADD_CALC_START (bit 13 of NANDFCR register). Doing this ensures ECC error correction takes place even after encountering 5-bit errors for the ensuing chunks of 512 bytes of data. There is no workaround for this issue in the RBL. This is expected to be rectified when TI comes up with new revisions of the DM355.

In DM355, TI uses a 2 stage boot loader; the RBL loads a UBL into IRAM which then loads the U-Boot to DDR.

This bug exists in the RBL. This means that if the NAND driver in the RBL encounters 5-bit errors, ECC correction will stop. The UBL, which is the first software component in the bootup sequence, will do a dummy read of the NANDERRADD1 register after setting up the AEMIF as shown below (#1 of FIXES IN TI SOFTWARE COMPONENTS) . If this is not done, UBL will not be able to correct any errors due to the RBL bug. This will occur if the RBL encounters 5-bit errors.

A dummy read of the NANDERRADD1 register will take place even after the NAND driver encounters no bit errors although experiments performed did show that such a dummy read was not required.

The issue is more likely to be seen in NAND devices that require 4-bit ECC.

Even though there might be ECC errors in the UBL image, it is possible to load a UBL with errors in the UBL image. Random boot failures can occur as a result, or if boot appears to succeed, it is also possible to see system stability issues due to possible corrupted system configuration values (e.g., PLL multiplier, PSC domains).

This is a limitation in the RBL. This is not a limitation in the ECC hardware that is a feature of the Asynchronous EMIF (AEMIF) peripheral. Therefore any software outside of the RBL, such as the UBL and NAND driver in the U-Boot and kernel, can use the ECC hardware to implement NAND error correction and detection.

This limitation has no impact on any of the other boot modes: MMC/SD boot mode, AEMIF (OneNAND) boot mode, UART boot mode.

FIXES IN TI SOFTWARE COMPONENTS

1. **Mitigation of RBL BUG in UBL**
   The UBL code is modified to include the sequence of code mentioned below. The following is done in the “device.c” file that is part of the UBL projects for DM355.
   ```
   // AEMIF Setup
   if (status == E_PASS) status |= DEVICE_EMIFInit();

             temp = AEMIF->NANDERRADD1
   ```
   This ensures that 4BITECC_ADD_CALC_START is “low” thereby ensuring that the UBL can perform ECC correction if it encounters bit errors.
2. **Software Fix in NAND writer and UBL**
   The NAND writer and UBL both use the same NAND driver and are built from the

same source. The NAND driver for both the NAND writer and UBL has been modified according to the above explanation. Code excerpt from the NAND driver is shown below. The required sequence has been marked in bold.

```
if ((corrState == 1) || (corrState > 3))
{
  temp = AEMIF->NANDERRADD1;
  return E_FAIL;
}
else if (corrState == 0)
{
  temp = AEMIF->NANDERRADD1;
  return E_PASS;
}
```

3. **Software Fix in U-Boot and kernel**
   TI makes sure that the NAND driver code in both the U-Boot and kernel are exactly the same. The following is an excerpt from the kernel NAND driver that is part of LSP 2.10 and later versions.

```
If (iserror == ECC_STATE_NO_ERR) {
          val = __raw_readl(info->emifregs + NANDERRVAL1);
          return 0;
}
else if (iserror == ECC_STATE_TOO_MANY_ERRS) {
          val = __raw_readl(info->emifregs + NANDERRVAL1);
          printk(KERN_ERR "%s Too many errors to be corrected!\n", __func__);
          return -1;
}
```

In the kernel and U-boot the NANDERRVAL1 register is read, but in the UBL and NAND writer the NANDERRADD1 register is read. Doing a dummy read of any of these registers has exactly the same effect of clearing the 4BITECC_ADD_CALC_START in the NANDFCR register.

NOTE: All TI Software releases for DM355 SOC's (including the PSP 3.01 releases) have been updated with the above software fixes. TI has patches for DM355 SOCs in addition to the official LSP 2.10 and later releases.

Workaround(s)

Other Workaround(s) to mitigate RBL issue:

- A workaround for this issue would be to use a different ROM boot mode, such as MMC/SD, AEMIF (OneNAND or UART) to load a secondary boot loader that can correctly access the NAND and load the remaining system software from the NAND device.
- Another recommendation would be to replace the current 4-bit ECC NAND device with a NAND device that uses less than 4-bit ECC. By employing the 4-bit ECC in the ROM we automatically cover any ECC requirements less than or equal to 4-bits (including 1-bit).

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

4.1 Usage Notes for Silicon Revision 1.1

Silicon revision 1.1 applicable usage notes have been found on a later silicon revision. For more details, see Section 2, Usage Notes for Silicon Revision 1.4.

4.1.1 NAND Layout Assumed by RBL for Big Block NAND Does Not Match NAND Manufacturers' Recommendations

Typically NAND manufactures place Bad Block information in the spare bytes area. The RBL assumes a data layout as below:

512 bytes Data
16 bytes ECC Data
512 bytes Data
16 bytes ECC Data
512 bytes Data
16 bytes ECC Data
512 bytes Data
16 bytes ECC Data

This layout can cause real data to be placed in the spare area, which would erase meta-data that is placed there by the NAND manufacturer. Typically, NAND programmers assume consecutive data followed by the meta-data in the spare areas as below:

2048 bytes Data
64 bytes ECC Data

This does not affect small block NAND (512 bytes/page). Any device with page sizes larger than 512 bytes will be affected.

4.2 Silicon Revision 1.1 Known Design Exceptions to Functional Specifications

Some silicon revision 1.1 applicable advisories have been found on a later silicon revision. For more details, see Table 7, Silicon Revision 1.4 Known Design Exceptions to Functional Specifications.

Table 13. Silicon Revision 1.1 Advisory List

Advisory 1.1.1 *RBL Code ECC Limitation*

Revision(s) Affected 1.1 (not present on silicon revision 1.3 and 1.4)

Details During NAND boot, the ROM Bootloader (RBL) does not implement error correction and detection (ECC). In particular, during NAND boot, the RBL will neither detect nor correct bit errors while loading the user boot loader (UBL).

Once the UBL is loaded, the UBL can implement error correction and detection. For more information on the NAND boot process, see the “NAND Boot Mode” section in the *TMS320DM355 Digital Media System-on-Chip ARM Subsystem Reference Guide* (literature number SPRUFB3).

This is a limitation in the RBL. This is *not* a limitation in the ECC hardware that is a feature of the Asynchronous EMIF(AEMIF) peripheral. Therefore any software outside of the RBL, such as the UBL and NAND driver, can use the ECC hardware to implement NAND error correction and detection.

Workaround(s) The impact of this problem is limited by the incidence of read errors of the NAND device. In very rare situations, a bit error could occur in the UBL transfer. The impact can be limited by choosing a NAND device with lower incidence of read errors. SLC NAND is recommended. SLC NAND typically has a lower incidence of read errors compared to MLC NAND. The impact can also be limited by minimizing the size of the UBL. **Note**: the maximum UBL size is 30Kbytes.

This limitation has no impact on any of the other boot modes: MMC/SD boot mode, AEMIF/OneNAND boot mode, UART boot mode. This limitation also has no impact on managed NAND devices.

Appendix A Revision History

This data sheet revision history highlights the technical changes made to the SPRZ264D device-specific data sheet to make it an **E** revision.

Scope: Added 1.3 and 1.4 Advisories and Usage Notes.

Table 14. Revision History

ADDS/CHANGES/DELETES
Updated Figure 1.
Deleted 1st note in Section 1.2.
Added Silicon Revision 1.3 and 1.4 to Table 1.
Added Silicon Revision 1.3 and 1.4 Advisories and Usage Notes.
Removed ZWK Package and change to ZCE Package Section.
Moved Section 4.1.1 from Silicon Revision 1.4 to Silicon revision 1.1.

IMPORTANT NOTICE

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