*TMS320C5505/C5504*
*Fixed-Point Digital Signal Processor*
**Silicon Revision 2.0**

# Silicon Errata

Literature Number: SPRZ310D
January 2010–Revised July 2015

# Contents

# List of Figures



# List of Tables

# TMS320C550x Silicon Revision 2.0

## 1 Introduction

This document describes the known exceptions to the functional specifications for the TMS320C550x devices (i.e., TMS320C5505, and TMS320C5504). For more detailed information on these devices, see the device-specific data manual:

- *TMS320C5504 Fixed-Point Digital Signal Processor* data manual (SPRS659)
- *TMS320C5505 Fixed-Point Digital Signal Processor* data manual (SPRS660)

Throughout this document, unless otherwise specified, TMS320C550x and C550x, refer to the TMS320C5505, and TMS320C5504 devices.

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.

### 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.,TMS320C5505AZCHA12). 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.

TI device nomenclature also includes a suffix with the device family name. This suffix indicates the package type (for example, ZCH), the temperature range (for example, "Blank" is the commercial temperature range), and the device speed range in megahertz (for example, "10" is the default 100 MHz device).

## 1.2 Revision Identification

Figure 1 provides an example(s) of the TMS320C550x device markings. The device revision can be determined by the symbols marked on the top of the package.

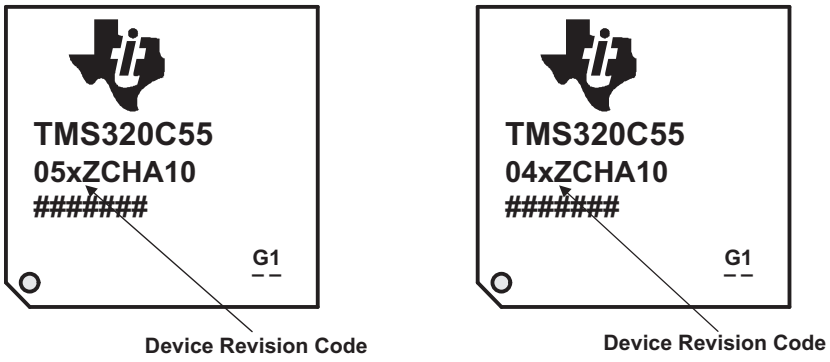


**Figure 1. Example and Device Revision Codes**

Silicon revision is identified by a device revision code marked on the package. The code on the package is of the format C5505x or C5504x, where "x" denotes the silicon revision . If x is "A" in the device part number, it represents Silicon Revision 2.0 TMS devices. Table 1 lists the information associated with each silicon revision.

**Table 1. C5505 and C5504 Device Revision Codes**

| DEVICE PART NUMBER DEVICE REVISION CODE (x) | SILICON REVISION | PART NUMBERS/COMMENTS |
|---|---|---|
| A | 2.0 | **TMS**320C5505AZCH, **TMS**320C5504AZCH, **TMX**320C5505AZCH, **TMX**320C5504AZCH |

Through software, the user can read bits 15-12 of the I/O space Die ID Register 3 (DIEIDR3) [1C43h].

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

This section describes the usage notes and advisories that apply to silicon revision 2.0 of the TMS320C5505 and TMS320C5504 devices.

## 2.1 Usage Notes for Silicon Revision 2.0

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

### 2.1.1 Master Clock Gating With WAKEUP, $\overline{\text{INT0}}$, or $\overline{\text{INT1}}$ Asserted

On silicon revision 2.0, the C550x DSP can disable the Master Clock by setting bit 15 of the PCGCR register (0x1C02). Once the master clock is disabled, it can only be re-enabled by one of the following events:

- Hardware reset being asserted ($\overline{\text{RESET}}$ = low)
- An enabled RTC alarm or periodic interrupt occurring
- The $\overline{\text{INT0}}$ or $\overline{\text{INT1}}$ pins being asserted (low) (level-sensitive)
- The WAKEUP pin being asserted (high) (level-sensitive)

When the master clock is disabled, there are no clocks for edge detection and therefore the $\overline{\text{INT0}}$, $\overline{\text{INT1}}$, and WAKEUP pins are level-sensitive. This means that a low on either the $\overline{\text{INT0}}$ or $\overline{\text{INT1}}$ or a high on the WAKEUP pin will force bit 15 of the PCGCR register to "0", enabling the master clock. Attempting to write a "1" to bit 15 of the PCGCR register while $\overline{\text{INT0}}$, $\overline{\text{INT1}}$, or WAKEUP are asserted will be unsuccessful since re-enabling the clocks has a higher priority than disabling them.

When the WAKEUP pin is configured as an output-pin, the WAKEUP pin only functions as a GPO and no longer functions as a WAKEUP pin to re-enable the master clocks. When the WAKEUP pin is configured as an input-pin, the WAKEUP pin's state must be low to disable the master clocks.

### 2.1.2 Serial Boot Modes Only Support 16-bit Address Mode

On silicon revision 2.0, the C550x DSP supports the following boot modes: NOR Flash, NAND Flash, SPI EEPROM, and I2C EEPROM. The SPI EEPROM boot supports both 16-bit and 24-bit address modes, while the I2C EEPROM boot mode *only* supports 16-bit address mode.

### 2.1.3 Reserved Bits in the RTC Oscillator Register (RTCOSC) [0x192C]

For proper C550x device operation on silicon revision 2.0, the "RESERVED" bits in the RTCOSC register (0x192C) should always be set to "zero".

### 2.1.4 Two 1149.1 JTAG Tap Controllers for JTAG Pins ($\overline{\text{TRST}}$, TCK, TMS, TDI, TDO)

The silicon revision 2.0 C550x devices have two internal 1149.1 JTAG Tap controllers but only one set of corresponding JTAG pins ($\overline{\text{TRST}}$, TCK, TMS, TDI, TDO). One TAP controller supports emulation and the other supports JTAG 1149.1 Boundary Scan. Only one of the two TAPs is internally connected to the pins at a time and it is the latched state of the EMU0 pin that determines which TAP is connected. The EMU0 pin is latched on the rising edge of $\overline{\text{TRST}}$ and from that time forward the selected tap is connected to the pins. If the latched state of EMU0 is "0", the boundary scan tap is selected and customers may perform boundary scan testing. If the latched state of EMU0 is "1", the DSP's emulation tap is selected and customers may perform emulation with TI's Code Composer Studio™ IDE Emulation Debugger.

**Note:** Because of the C550x device's internal (and recommended external) pullup on the EMU0 pin and the fact that the emulation pods (e.g., XDS560) do not drive the EMU0 pin while $\overline{\text{TRST}}$ is driven low-to-high, the emulation tap will normally be the one selected. However, customers who wish to do boundary scan testing will need to have an external pulldown (2 kΩ is recommended), with sufficient strength to overcome the internal pullup, so that the C550x boundary scan tap is connected to the JTAG pins.

### 2.1.5 Bootloader Disables Peripheral Clocks

After hardware reset on silicon revision 2.0 C550x devices, the DSP boots via the bootloader code in ROM. During the boot process, the bootloader queries each peripheral to determine if it can boot from the peripheral. At that time, the required peripheral's clock will be enabled for the query and then disabled when the bootloader is finished with the peripheral. By the time the bootloader releases control to the user code, all peripheral clocks will be off and all domains in the ICR, except the CPU domain, will be idled. After the boot process is complete, the user is responsible for enabling and programming the required clock configuration for the DSP.

For example on the C5505 device, the bootloader disables both the MPORT and FFTHWA. To enable the MPORT and FFT HWA, write 0x000E to the ICR registers and issue an "idle" command.

*Assembly Code Example:*

```
*port(#0x0001) = #(0x000E)
idle
```

*C Code Example:*

```
*(ioport volatile unsigned *)0x0001 = 0x000E;
asm("idle");  // must add at least one blank before idle in " ".
```

For example on the C5504 device, the bootloader disables the MPORT. To enable the MPORT, write 0x0002E to the ICR registers and issue an "idle" command.

*Assembly Code Example:*

```
*port(#0x0001) = #(0x0002E)
idle
```

*C Code Example:*

```
*(ioport volatile unsigned *)0x0001 = 0x0002E;
asm("idle");  // must add at least one blank before idle in " ".
```

### 2.1.6 SPI Booting Does Not Support 24-Bit Encrypted Reauthoring and Bound to Device

On silicon revision 2.0, C550x devices do not support 24-bit SPI encrypted reauthoring and bound to device boot mode. The user must use 16-bit SPI in order to use encrypted reauthoring and bound to device boot mode. See Table 2 for the supported boot modes for 16- and 24-bit SPI.

**Table 2. Supported Boot Modes for SPI**

| Boot Mode | Unencrypted | Encrypted | Encrypted and Bound to Device | Encrypted Reauthoring and Bound to Device |
|---|---|---|---|---|
| 16-Bit SPI | Supported | Supported | Supported | Supported |
| 24-Bit SPI | Supported | Supported | Supported | Not Supported |

### 2.1.7 SPI Enters a Reset State When Writing to the SPICDR Register

On silicon revision 2.0, the SPI enters a reset state when writing to the SPICDR register. When software writes to the SPICDR register at 0x3000, the software also evaluates and applies the content of the SPICCR register at 0x3001. Thus, if the RST bit (bit 14) of SPICCR is set to a "1" inadvertently, a SPI reset will occur.

Before writing to the SPICDR register, the user must read the SPICCR register and mask out the RST bit with the rest of the bits preserved and write back to SPICCR. Next, write to the SPICDR register.

Following is an example sequence:

- Read SPICCR.
- Mask with 0xBFFF.
- Write back to SPICCR.
- Write to SPICDR.

*C Code Example:*

```
#define SPICDR     *(volatile ioport Uint16*)     (0x3000)
      #define SPICCR     *(volatile ioport Uint16*)     (0x3001)
      Uint16 temp;
      temp = SPICCR;
      SPICCR = temp & 0xBFFF;
      SPICDR = 0x18;
```

### 2.1.8 SPI Clock State Is Changed When Writing to the SPICDR Register

On silicon revision 2.0, software writes to the SPICCR register (0x3001) when writing to the SPICDR register (0x3000). As a result, the CLKEN bit (bit 15) of SPICCR is set when writing to SPICDR.

The user is responsible for reading the SPICCR register, writing the appropriate value of CLKEN, and preserving the remaining bits back to SPICCR **before** writing to the SPICDR register.

### 2.1.9 HWAFFT Data and Scratch Pointers Copied as 16-Bit Instead of 23-Bit

On silicon revision 2.0, the HWAFFT uses two data buffers, data and scratch, to pass data to the FFT coprocessor, to store intermediate results, and to store the FFT output. The hwafft_Npts routines available in ROM use pointers to access the data and scratch buffers. These pointers are copied as 16-bit addresses instead of 23-bit addresses when copying from AR0 to AR3 and from AR1 to AR2. The upper bits are not copied which can lead to incorrect FFT results and potential corruption of data incorrectly addressed by the hwafft_Npts routine.

The user is responsible for following one of the following workarounds:

- Execute hwafft_Npts from RAM using the updated hwafft.asm with bug fixes. The hwafft.asm file included with SPRABB6.zip incorporates bug fixes for this errata. The HWAFFT routines stored in ROM cannot be updated and do not incorporate these bug fixes. For reference, hwafft_rom.asm is included in SPRABB6.zip and contains the HWAFFT routines exactly as they exist in the ROM of the affected revisions.
- Execute hwafft_Npts from RAM or from ROM while passing duplicate pointers to the scratch and data buffers to correctly initialize the upper bits of AR0, AR1, AR2, and AR3. The data and scratch buffers can be located in word addresses greater than 0x10000, but the buffers must not cross 16-bit address boundaries (that is, address bits 22–16 must not change). Additionally, if hwafft_512pts is used, data and scratch must not reside at the beginning of a page boundary (that is, word address 0x10000 or 0x20000). This workaround initializes the most significant bits of internal registers AR2 and AR3 such that when the 16-bit address is copied from AR0 and AR1, and provided the assumptions above are satisfied, the full 23-bit address remains correct throughout HWAFFT execution.

  In the user program wherever hwafft_Npts is called, replace:

```
out_sel = hwafft_Npts(data, scratch, fft_flag, scale_flag);
```

  With:

```
out_sel = hwafft_Npts(data, scratch, scratch, data, fft_flag, scale_flag);
```

  In hwafft.h, replace:

```
Uint16 hwafft_Npts(
    Int32 *data,
    Int32 *scratch,
    Uint16 fft_flag,
    Uint16 scale_flag
);
```

  With:

```
Uint16 hwafft_Npts(
    Int32 *data,
    Int32 *scratch,
    Int32 *duplicate_scratch,
    Int32 * duplicate_data,
    Uint16 fft_flag,
    Uint16 scale_flag
);
```

### 2.1.10 USB Oscillator Consumes Power When Core Voltage Is Removed

On silicon revision 2.0, the USB oscillator is disabled by writing a '1' to the USBOSCDIS bit of the USBSCR register. Whenever voltage to the core is removed (for example, during RTC-only mode), the value written to the USBOSCDIS register is lost. If the USB I/O supplies are supplied, the USB oscillator reactivates and consumes power.

To disable the USB oscillator when core voltage is removed, the user must power down all USB I/O supplies (USB_$V_{DDOSC}$, USB_$V_{DDA3P3}$, and USB_$V_{DDPLL}$).

## 2.2 Silicon Revision 2.0 Known Design Exceptions to Functional Specifications

**Table 3. Silicon Revision 2.0 Advisory List**

## Advisory 2.0.1 *USB: Endianess Incompatibility*

**Revision(s) Affected** 2.0

**Details**

The C550x CPU is a word addressable and big endian architecture. The CPU interfaces to the rest of the system through several ports: MPORT, XPORT, DPORT, and IPORT. The DMA transfers data from peripherals to on-chip memory through the MPORT in 32-bit packets. The CPU accesses the peripherals through the XPORT in 8- or 16-bit packets. The CPU accesses external memory in 8-bit (configured through a system register setting), 16-bit (through CPU single word access), or 32-bit (through CPU double-word access). External data accesses occur through the DPORT and the EMIF. The CPU fetches code from external memory through the IPORT in the EMIF in 32-bit packets.

Some C550x peripherals (e.g., EMIF, USB, LCD, and MMC/SD) have sensitivity to the data endianess. EMIF: Endianess is hardcoded to big endian. LCD: Endianess is controlled through software. Default is little endian. MMC/SD: Endianess is controlled through software. Default is big endian. USB: DMA transfer to/from USB buffer is big endian. CPU XPORT access to USB buffer is little endian. USB endianess is not software controllable. The two endianesses of the USB could result in inter byte swap.

Figure 2 shows data paths that could create byte/word swaps.

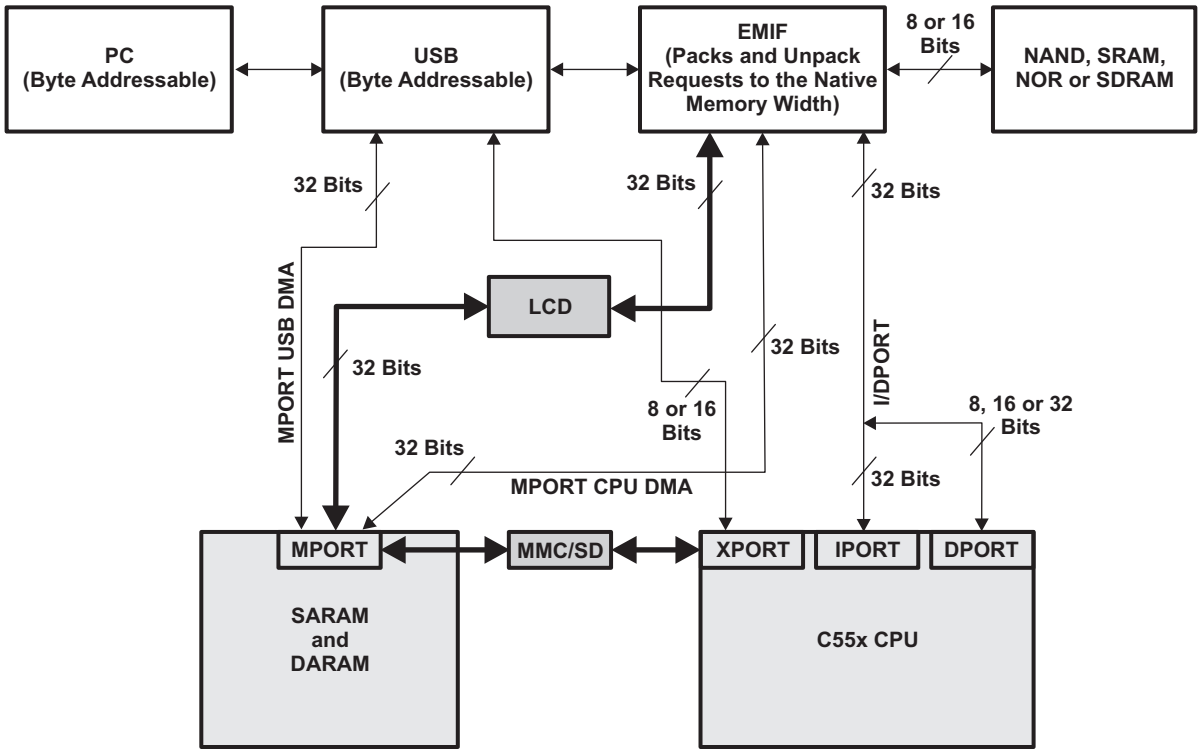


**Figure 2. CPU, USB, EMIF LCD, and MMC/SD Data Paths**

**Example Case:** Data transfer between the USB and CPU (see Figure 3 and Figure 4).

Data can be read from the USB by the USB DMA or CPU. The USB DMA accesses on-chip memory through the MPORT and only performs a 32-bit read while the CPU can perform a 16-bit read via the XPORT. When USB DMA transfers data from the USB buffers to on-chip memory, the data transfer is 32-bits and is handled in big endian fashion, so no data swap occurs (see Figure 3). However, when the CPU, through the XPORT, accesses the USB buffers, the data is accessed 16-bits at a time in little endian fashion resulting in an inter byte swap (see the Figure 4).

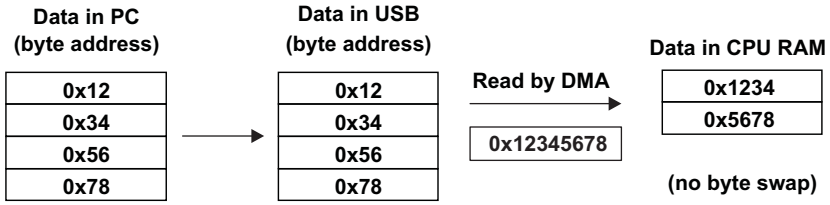


**Figure 3. USB DMA Read**

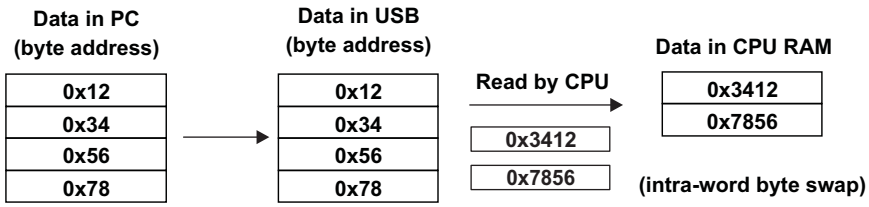


**Figure 4. CPU Read From USB**

**Workaround(s)**

To correct this issue, software is required to fix the intra-word byte swap on the data prior to processing on the C550x CPU.

## Advisory 2.0.4 *I2S: I2S Internal Data Delay*

**Revision(s) Affected** 2.0

**Details**

The I2S module has an internal delay for the data transmit/receive path that varies depending on the settings of the Pack bit and Word Length in the I2S Control Register and the FSDIV in the Sample Rate Generator Register.

Table 4 shows the transmit path internal data delays. Feedback path refers to an I2S transmit pin that is externally connected to the I2S receive pin.

**Table 4. Transmit Path Internal Data Delays**

| PACK | DATA DELAY |
|---|---|
| 1 | The first five transmit frames will be zero data. On the sixth transmit frame, the data written to the transmit register will be shifted out on the DX pin. |
| 0 | The first three transmit frames will be zero data. On the fourth transmit frame, the data written to the transmit register will be shifted out on the DX pin. |

Table 5 shows the receive path internal data delays:

**Table 5. Receive Path Internal Data Delays**

| WORD LENGTH | DATA DELAY |
|---|---|
| 8-bit | The RX data registers will contain zero data for the first two frames. |
| 10, 12, 14, and 16-bit | The RX data registers will contain zero data for the first three frames. |
| 18, 20, 24, and 32-bit | The RX data registers will contain zero data for the first two frames. |

Table 6 shows the feedback path internal data delays:

**Table 6. Feedback Path Internal Data Delays**

| PACK | WORD LENGTH | FSDIV BITS I2SSRATE.[5:3] | DATA DELAY |
|---|---|---|---|
| 1 | 8-bit | 000 | Data received on the 7th sample |
| | 8-bit | 001 – 101 | Data received on the 6th sample |
| | 10-, 12-, 14-, and 16-bit | 001 | Data received on the 5th sample |
| | 10-, 12-, 14-, and 16-bit | 010 – 101 | Data received on the 4th sample |
| 0 | 8-, 10-, 12-, 14-, 16-, 18-, 20-, 24-, and 32-bit | 000 – 101 | Data received on the 3rd interrupt |

**Workaround(s)** Zero data should be Ignored.

## Advisory 2.0.5 *USB: USB Queue Manager Reads Only 16-bit Address of USB Descriptors*

**Revision(s) Affected** 2.0

**Details** The C550x has 23-bit address space but the USB Queue Manager (QMGR) is a 32-bit register that holds the address of the USB descriptors. A descriptor itself is a structure with information about the addresses of the source/destination data buffers and their sizes. The address of a particular descriptor is written to the QMGR register for a particular DMA endpoint. The CPU writes the address of a descriptor to the QMGR register for a for a DMA endpoint. The QMGR fires the USB DMA to read the descriptor at the address pointed to in the QMGR register and sets up the DMA endpoints for future transfers. When a USB host connects and performs a transfer, the QMGR copies the address of the descriptor to a completed queue. Upon receiving the USB interrupt, the USB driver should read the 32-bit descriptor address in the QMGR completion queue to determine which DMA endpoint has completed transferring data. Even though the CPU can write a 32-bit value into the QMGR register, it can only read the lower 16-bits of this register. Thus, the descriptor can only be allocated in the CPU memory map to the same lower 16-bit address and all descriptors must be placed in one contiguous block of 64K words in SARAM.

For example:

- USB descriptor A is located at 0x008000
- USB descriptor B is located at 0x018000

The descriptor A and B will be considered the same descriptor.

**Workaround(s)** The USB descriptors should be placed in ONE CONTIGUOUS BLOCK of 64K words ($2^{16}$) in memory.

## Advisory 2.0.8 *DMA: DMA Transfer Length Must be a Multiple of 4 x 2(Burst Mode)*

**Revision(s) Affected**

2.0

**Details**

If the transfer length register has a value that is zero or *not* a multiple of 4 x 2(Burst Mode) when the DMA transfer begins, it will cause an unexpected operation of DMA.

While the DMA transfers 32-bit words from a source address to a destination address, the value in the DMA transfer length register is the length in bytes. For example, if the total DMA transfer length is 4 32-bit words and the burst size = 1, 4 x 4 = 16 should be written to the DMA transfer length register. The burst size should also be considered. Burst size is the minimum data transfer size; therefore, the total DMA Transfer Length should be 4 x 2(Burst Mode). For more details, see Table 7

**Table 7. DMA Transfer Lengths**

| BURST MODE | BURST SIZE (32-BIT WORD) | DMA TRANSFER LENGTH (BYTES) |
|---|---|---|
| 0 | 1 | Multiple of 4 (minimum 4) |
| 1 | 2 | Multiple of 8 (minimum 8) |
| 2 | 4 | Multiple of 16 (minimum 16) |
| 3 | 8 | Multiple of 32 (minimum 32) |
| 4 | 16 | Multiple of 64 (minimum 64) |

**Workaround(s)**

Write only a multiple of 4 x 2(Burst Mode) to the DMA transfer register before the DMA starts.

## Advisory 2.0.9 *USB Boot Does Not Work if the On-Chip USB_LDO Supplies Power to the USB Core*

**Revision(s) Affected** 2.0

**Details** The on-chip USB_LDO is intended to power the USB_ $V_{DD1P3}$ and USB_$V_{DDA1P3}$ pins. At reset, the USB_LDO is turned off and can be turned on via the USBLDOEN bit (bit 0) in the LDOCNTL register (0x7004). However, the C5505/14 on-chip Bootloader does not enable the USB_LDO before attempting USB boot, so USB boot will not work if the USB_LDO supplies power to the USB_ $V_{DD1P3}$ or USB_$V_{DDA1P3}$ pins.

**Workaround(s)** For the applications that require USB boot, the USB Core (USB_$V_{DD1P3}$, USB_$V_{DDA1P3}$) must be powered externally.

## Advisory 2.0.10 *USB CPPI Receive Starvation Interrupt*

**Revision(s) Affected** 2.0

**Details** When an endpoint is enabled for receive transfer(s) that will be serviced via CPPI DMA and data has been received prior to allocating the DMA resource, the DMA will generate a starvation interrupt to notify the application a lack of resource (starvation) in anticipation that the application will furnish the required resource. Once the starvation occurs, the CPPI DMA continues generating interrupts periodically whenever the host tries to send data. It does not stop until the application furnishes a resource.

The USB starvation interrupt is always enabled and cannot be masked off at the USB controller level. Since the DMA continues to generate the starvation interrupt periodically and there exists no capability to mask the starvation interrupt at the USB controller level, the CPU is forced either to fully service the DMA interrupt as it is received or disable all USB interrupts at the CPU level. Disabling the entire USB interrupt might not be the desired option since the CPU needs to be aware of other USB interrupts that are more critical.

**Workaround(s)** Dedicated data receiving buffers are recommended. The data buffers should be allocated and available to the CPPI DMA during USB initialization. The required data buffer size is highly dependent on USB host applications. For Windows XP USB Mass Storage device driver, it is recommended to allocate at least 64 KB (128 descriptors; 128*512 = 64 KB) of data buffer space.

## Advisory 2.0.13 *USB Controller in TI's C55xx Device Responds Abnormally to Certain Control Out Transfers*

**Revision(s) Affected** 2.0

**Details**

USB is in device mode and connected to a standard PC (host) and is operating in full speed mode. The problem has been observed when using Control-OUT transfers with odd byte payload. USB engine occasionally provides abnormal response in the status phase of Control-OUT transfers with data payload. The abnormal response seen is, the status phase should contain 0-length IN packet, but occasionally 1-byte IN packets are sent by the device.

**The Issue:**

Whenever host sends an odd byte control OUT transfer to device, the out packet gets written into FIFO and then, FIFO contents will be read by the device CPU (through bridge responsible for bus and protocol conversion, etc.). Since this is an odd byte transfer, the last byte needs to be read by switching to byte mode access, but in the C55xx device topology the read byte enable signal which is supposed to generate the byte enables for performing odd byte read is tied off (10'h2) in design. This will force all reads and writes to be half words (2 bytes). The writes are still okay, as their byte enables come from the byte enable signal directly through software configurations, but for reads considering byte enable signal is tied-off this will always result in 2 bytes read, even if there is only 1 byte of data that needs to be read. This results in non-clearing of FIFO pointers.

Since FIFO pointers were not cleared, and gets cleared only when the DATAEND and RXPKTRDY bit is set, there exists a race condition when DATAEND & RXPKYRDY is set, FIFO pointers are getting cleared and at the same time IN packet is received. Due to this race condition, where IN packet is received while FIFO pointers getting cleared, the data in the FIFO is sent out during the status phase.

The above mentioned issue will not exist with even bytes OUT transfers. For even byte OUT packets, as the need doesn't exist to perform a byte mode switch to accomplish byte read, and device needs to perform half-word read, this in turn will clear the FIFO pointers. Hence, the device will respond with the correct status information.

**Workaround(s)**

With the following workaround solution the issue will recede.

1. Once the control OUT packet is received, perform read of all the odd bytes from the FIFO, considering read is always half word (16 bits) the FIFO pointers will not get cleared to zero, but instead will move to 'h7F (negative one). This can be observed in the count register (COUNT_INDEX0) register.
2. After the first read sequence, instead of setting ServiceRxPktRdy and DataEnd bit, set FIFO_Access bit in TestMode Register (Addr = 840Eh).
3. Setting the FIFO Access bit will readjust the FIFO pointers. That is, USB host side FIFO pointer will set to previous odd bytes transfer plus one and CPU side FIFO pointer will be set to zero. To illustrate further, say if the odd bytes transfer is 63 bytes then the host side FIFO pointer will be set to 64 whereas CPU side FIFO pointer will reset to zero.
4. Configuring the FIFO access bit to '1' will generate an interrupt and RxPktRdy will be set.
5. The generated interrupt (from FIFO Access bit) needs to be disabled and the corresponding interrupt flag needs to be cleared.
6. As next step software needs to again perform dummy read from the FIFO, this will clear the FIFO pointers.
7. After the dummy reads, set ServiceRxPktRdy and DataEnd bit.
8. Device will respond with the correct status packet.

The below Pseudo code provides workaround implementation details.

```
// Read endpoint-0 Buffer Function
void USB_readEP0Buf( .. )
{
// Declarations
    .
    .
    .

    /* select EP0 registers */
    usbRegisters->INDEX_TESTMODE &= ~(CSL_USB_INDEX_TESTMODE_EPSEL_MASK);

    /* get Receive packet size */
    packetSize = ((usbRegisters->COUNT0_INDX) &
                  CSL_USB_COUNT0_INDX_EP0RXCOUNT_MASK);

    // perform 1st read sequence
    for(count = 0; count < (packetSize)/2; count++)
    {
        *pBuf = usbRegisters->FIFO0R1;
        pBuf++;
    }

    // this will perform last half word read in first read sequence.

    if (packetSize & 0x1)
    {
        *pBuf = usbRegisters->FIFO0R1 & 0xFF;
    }
    // Check if the packet size is odd bytes or even bytes.
    // for odd bytes the second read operations are performed.
    // If it's even bytes then do not perform 2nd read sequence.

    if(packetSize & 0x1) //if True then its odd bytes transfer
    {
        // following operations are atomic
        value = IRQ_globalDisable();

        // Set FIFO access bit in Test mode register

        usbRegisters->INDEX_TESTMODE |=
                CSL_USB_INDEX_TESTMODE_FIFO_ACCESS_MASK;

        // perform the second sequence of read operations.

        for(count = 0; count < (packetSize)/2; count++)
        {
            *TempBuf = usbRegisters->FIFO0R1;
            TempBuf++;
        }

        if (packetSize & 0x1)
        {
            *TempBuf = usbRegisters->FIFOR1;
        }

        // Clear USB interrupt flags

        // enable the hardware interrupt
        IRQ_globalRestore(value);
        }
}
```

The above procedure will result in the device sending the correct status information that is zero length packet to the USB host.

# Revision History

**Changes from C Revision (May 2014) to D Revision** **Page**

- Added Advisory 2.0.13, *USB Controller in TI's C55xx Device Responds Abnormally to Certain Control Out Transfers* .. 18

## IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as “components”) are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

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

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license 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 significant portions 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. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

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

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

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

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