TMS320C5517 Fixed-Point Digital Signal Processor
Silicon Revision 1.0, 2.0, and 2.1

Silicon Errata

Literature Number: SPRZ383B
April 2014–Revised September 2017

Contents

List of Figures

List of Tables

TMS320C5517 Silicon Revision 1.0, 2.0, and 2.1

1 Introduction

This document describes the known exceptions to the functional specifications for the TMS320C5517 device. For more detailed information on these devices, see the device-specific data manual:

TMS320C5517 Fixed-Point Digital Signal Processor data manual (SPRS787)

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.,TMS320C5517CZCHA12). 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 of the TMS320C5517 device marking. The device revision can be determined by the symbols marked on the top of the package.

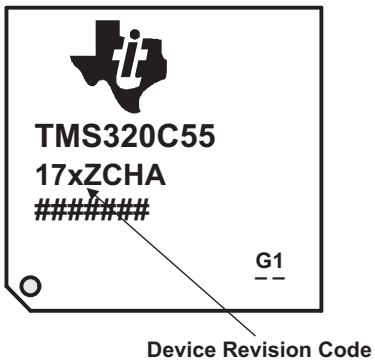

Figure 1. Example and Device Revision Codes

Silicon revision is identified by prefix with a device revision code marked on the package. The code on the package is of the format C5517x, where "x" denotes the silicon revision. If x is "A", it represents Silicon Revision 2.1 TMS devices. Table 1 lists the information associated with each silicon revision.

Table 1. C5517 Device Revision Codes

Device Part Number Device Revision Code (x)	Silicon Revision	Part Numbers/Comments
A	2.0	This silicon revision is available as TMX *only*. **TMX**320C5517AZCH
C	2.1	This silicon revision is available as TMX *only*. **TMX**320C5517CZCH
A	2.1	**TMS**320C5517AZCH

2 Silicon Revision 1.0, 2.0, and 2.1 Usage Notes and Known Design Exceptions to Functional Specifications

This section describes the usage notes and advisories that apply to silicon revision 1.0, 2.0, and 2.1 of the TMS320C5517 device.

2.1 Usage Notes for Silicon Revision 1.0, 2.0, and 2.1

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 revisions 1.0, 2.0 and 2.1, the C5517 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 Two 1149.1 JTAG Tap Controllers for JTAG Pins ($\overline{\text{TRST}}$, TCK, TMS, TDI, TDO)

The silicon revisions 1.0, 2.0, and 2.1 of the C5517 device 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: The emulation tap will normally be the TAP controller that is selected while $\overline{\text{TRST}}$ is driven low-to-high. This is because of the internal and recommended external pullup on the EMU0 pin, in addition to the emulation pods (e.g., XDS560) not driving the EMU0 pin. 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 C5517 boundary scan tap is connected to the JTAG pins.

2.1.3 Bootloader Disables Peripheral Clocks

After hardware reset on silicon revision 1.0, 2.0, and 2.1 for the C5517 device, the DSP boots via the bootloader code in ROM. 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 C5517 device, the bootloader disables both the MPORT and HWAFFT. To enable the MPORT and HWAFFT, 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 " ".
```

2.1.4 Bootloader Requires Booting NOR flash device supports "Reset" Command

The bootloader writes "Reset" (0xF0) command before trying to read the boot signature. Device that does not support "Reset" command produces an invalid boot signature to bootloader.

2.2 Silicon Revision 1.0, 2.0, and 2.1 Known Design Exceptions to Functional Specifications

Table 2. Silicon Revision 1.0, 2.0, and 2.1 Advisory List

Advisory 2.0.1 *Boot: eMMC Card of 4G or More Inaccessible after Power Up if Inserted in Card Slot 0 or 1 with No Boot Image at Power Up*

Revision(s) Affected 1.0, 2.0, 2.1

Details If an eMMC card of 4G or more is inserted in card slot 0 or 1 at power up and there is no boot image on it, the card is inaccessible after the bootloading process complete. The eMMC card can return to normal operatonal state after power cycle (remove and insert the slot).

Workaround(s) Do not set the boot mode to the SD/MMC slots (0 or 1) when there is no valid image in the eMMC card.

Advisory 2.0.2 *Boot: eMMC Card of 2G or Less Cannot Boot from Boot Partition Area*

Revision(s) Affected 1.0, 2.0

Details Bootloader cannot boot from the boot partition area of 2G or less eMMC card. However, it can boot from the first data partition formatted in FAT16 or FAT32.

Workaround(s) This is fixed in revision 2.1. Thus all fully-qualified TMS devices are not affected. Use higher than a 2G capacity eMMC card to boot from boot partition area.

Advisory 2.0.3 *DMA: DMA Transfer Length Must be a Multiple of 4 x 2(Burst Mode)*

Revision(s) Affected 1.0, 2.0, 2.1

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 3

Table 3. 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 2.0.4 *HWAFFT: Data and Scratch Buffers Must Reside at Data Locations Less Than Word Address 0x10000*

Revision(s) Affected 1.0, 2.0, 2.1

Details 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 have a software bug: pointers to data and scratch are copied as 16-bit addresses instead of 23-bit addresses. When a buffer resides in word address 0x10000 or greater, 23 bits are required to address these memory locations. The hwafft_Npts routine incorrectly copies just the 16 least significant bits, leading to incorrect FFT results and potential corruption of data incorrectly addressed by the HWAFFT.

Workaround(s)

(i) Execute hwafft_Npts from RAM or from ROM with data and scratch buffers located entirely in word addresses less than 0x10000.

(ii) For reference, hwafft_rom.asm is included in SPRABB6.zip and contain the HWAFFT routines exactly as they exist in the ROM of the affected revisions. SPRABB6.zip

(iii) Execute hwafft_Npts from RAM or from ROM while passing duplicate pointers to the scratch and data buffers. 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 XAR2 and XAR3 such that when the 16-bit address is copied from XAR0 and XAR1, 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
);
```

Advisory 2.0.5 *I2S: I2S Internal Data Delay*

Revision(s) Affected 1.0, 2.0, 2.1

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 2.0.6 *McBSP: CLKX Pin Outputs a Half of CPU Clock Once It Is Programmed to An Output Via CLKXM Bit In PCRL Register*

Revision(s) Affected 1.0, 2.0, 2.1

Details The CLKX pin is defaulted as an input pin at reset. It should stay quiescent till it is programmed to an output via CLKXM bit in PCRL Register and the Sample rate Generator Register (SRGR) is programmed. Note that the CLKR pin has same functionlity as CLKX and is function correctly.

Workaround(s) Programming the Sample Rate Generator Register first before setting the CLKX pin to output.

Advisory 2.0.7 *RTC: RTC Interrupt Does Not Work if the RTC Isolation Control is Enabled*

Revision(s) Affected 1.0, 2.0, 2.1

Details

The RTC Isolation Control bit in the RTC System Control Register is defaulted to enable isolating the CVDD_RTC power domain from the VDDC power domain when VDDC powers up. It is required to disable the isolation and write the unlock key to RTC Gate-Keeper Registers in order to write to any RTC peripheral registers.

The RTC isolation must be disabled for RTC interrupt to work properly. The external WAKEUP input to wake up the device is not affected by this Isolation Control.

For this process, see the *System Control* chapter in:

TMS320C5517 Digital Signal Processor Technical Reference Manual (SPRUH16)

Workaround(s)

Keep the RTC_ISO bit as high after clearing the write enable key in the RTC Gate-Keeper Registers, as shown in the following example:

```
// This example will leave the RTC power domain un-
isolated after zeroing out the write keys to lock

      asm("    @#IFR1_L = #0xffff || mmap() "); // clear int flags
      asm("    @#IER1_L = #0x0004 || mmap() "); // set RTC int
// Enable RTC Registers as writeable
      asm("    *port(#0x1C27) = #0x0001 "); // un-isolate CVDD_RTC power domain
      asm("    *port(#0x196C) = #0xF1E0 "); // RTC writeable unlock key
      asm("    *port(#0x196D) = #0x95A4 ");
// Enable RTC interrupt
      asm("    *port(#0x1900) = #0x0001 ");
      asm("    *port(#0x1920) = #0x003F "); // clear flags
// Bit 5 = 1 is EXTINTEN, which controls the external Wakeup input to generate a
DSP INT
// See RTC Interrupt and Wakeup Logic in the device-
specific technical reference manual
      asm("    *port(#0x1924) = #0x0020 "); // enable external interrupt
// Leaves RTC un-isolated and works with locking
      asm("    *port(#0x196C) = #0x0000 "); // clear out unlock key to lock
      asm("    *port(#0x196D) = #0x0000 ");
      // asm("    *port(#0x1C27) = #0x0000 "); // leaves CVDD_RTC power domain un-
isolate to workaround
```

Advisory 2.0.8 *SPI: SPI Can Enter a Reset State When Writing to the SPICDR Register*

Revision(s) Affected 1.0, 2.0, 2.1

Details 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.

Workaround(s) Before writing to the SPICDR register, read the SPICCR register and mask out the RST bit with the rest of the bits preserved and write back to SPICCR. Then write to the SPICDR register.

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;
```

Advisory 2.0.9 *SPI: SPI Clock State Can Be Inadvertently Altered When SPICDR Register Is Written*

Revision(s) Affected 1.0, 2.0, 2.1

Details When software writes to the SPICDR register at 0x3000, the software also evaluates and applies the content of the SPICCR register at 0x3001. Thus, the CLKEN bit (bit 15) of SPICCR will be set when writing to the SPICDR register.

Workaround(s) Before writing to the SPICDR register, read the SPICCR register and write the appropriate value of CLKEN with the remaining bits preserved back to SPICCR. Then write to the SPICDR register.

Advisory 2.0.10 *Timer: Timer Produces False Interrupt When the Timer Interrupt Flag Corresponding to an NMI Timer Interrupt is Cleared*

Revision(s) Affected 1.0, 2.0, 2.1

Details A false timer interrupt is triggered when the timer interrupt flag corresponding to a non-maskable interrupt (NMI) timer interrupt is cleared. This NMI timer interrupt should have pre-empted a regular timer interrupt ISR and before the timer ISR has had a chance to clear the timer interrupt flag.

For example, if two timers are set up as Timer0 for TINT (timer interrupt) to CPU and Timer1 for NMI interrupt to CPU. The CPU receives TINT first and starts servicing TINT ISR. But before ISR can clear the Timer0 flag in the Timer Interrupt Aggregation (TIA) register, the CPU receives an NMI interrupt from Timer1. The NMI interrupt pre-empts TINT ISR and the CPU starts servicing the corresponding NMI ISR.

After the NMI ISR clears the Timer1 flag in the TIA register, another TINT interrupt is generated to the CPU due to the uncleared Timer0 interrupt flag. When NMI ISR returns to TINT ISR and TINT ISR clears the Timer0 flag in the TIA register and returns to main function. The CPU immediately starts servicing the second (false) TINT interrupt again, but the Timer0 interrupt flag is not set in the TIA register.

Workaround(s) In the TINT ISR, exit ISR if no flags are set in the TIA register.

```
// This example is to check Timer aggregate register in Timer ISR
#define TIMER_INTR_AGG_FLG
*(volatile ioport Uint16*)(0x1C14)
interrupt void timer_Isr()
{
    Uint16 timer_intr_flag = 0;

    //Read timer interrupt aggregation flag register
    timer_intr_flag = TIMER_INTR_AGG_FLG & 0x0007;

    //Proceed with timer ISR processing if any of the flags are set
    //Otherwise exit ISR
    if (timer_intr_flag)
    {
      ...
    }
}
```

Advisory 2.0.11 *USB: Endianess Incompatibility*

Revision(s) Affected 1.0, 2.0, 2.1

Details

The C5517 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 C5517 peripherals (e.g., EMIF, USB, and MMC/SD) have sensitivity to the data endianess. EMIF: Endianess is hardcoded to big 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.

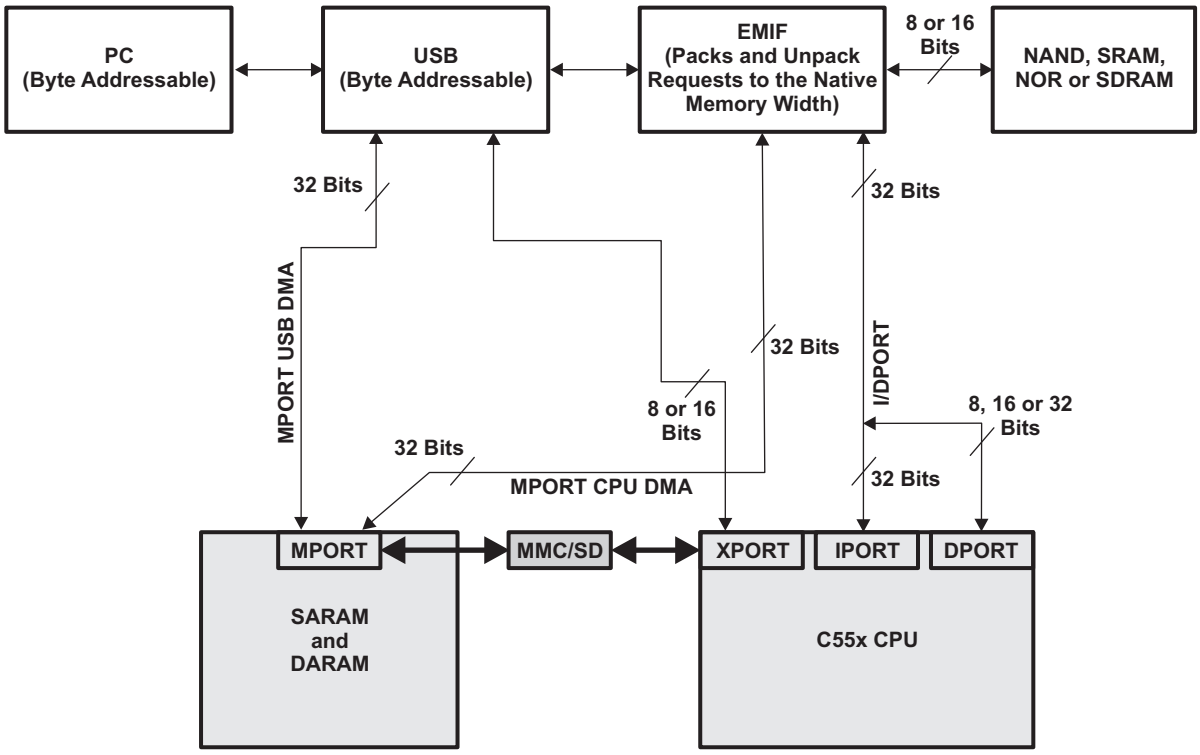

Figure 2. CPU, USB, EMIF, and MMC/SD Data Paths

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).

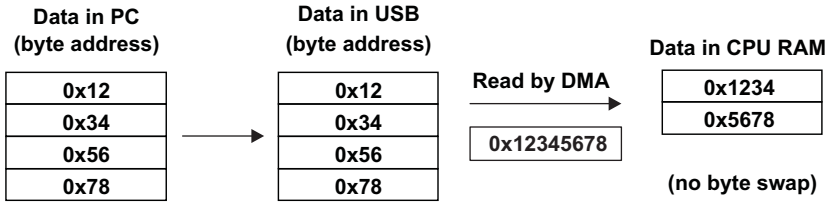

Figure 3. USB DMA Read

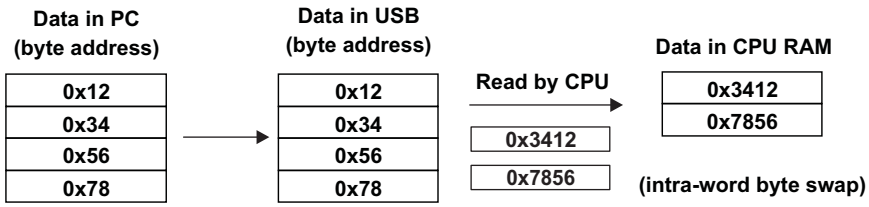

Figure 4. CPU Read From USB

Workaround(s)

To correct this issue, software is required to fix the intra-word byte swap on the data prior to processing on the C5517 CPU.

Advisory 2.0.12 *USB: USB Queue Manager Reads Only 16-bit Address of USB Descriptors*

Revision(s) Affected 1.0, 2.0, 2.1

Details

The C5517 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 2.0.13 *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.

Issues:

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.

Advisory 2.0.14 *I2S Master Mode Descope due to Frame Clock Timing Violation*

Revision(s) Affected 1.0, 2.0, 2.1

Details

I2S peripherals operating in Master Mode are not capable of meeting the Frame Clock Timing Requirements. In Master Mode, the I2S peripheral sources the I2S Clock (I2S_CLK) and Frame Clock (I2S_FS). Due to potential timing violations without a workaround, support for Master Mode is descoped.

The C5517 I2S peripherals have a unique Frame Clock Timing Requirement that requires Frame Clock to be latched on both edges of the I2S Clock. This imposes an additional constraint on the timing of Frame Clock as illustrated in Figure 5. The generated Frame Clock must meet the specified setup and hold requirements with respect to the sampling edge of the generated bit clock (refer to device datasheet). To satisfy the hold time in Slave Mode, an RC delay (or equivalent) must be added to the Frame Clock trace.

However, in Master Mode, no logic can be added to delay the Frame Clock relative to the I2S Clock. Since the switching characteristics of the Frame Clock can violate the Frame Clock hold time requirement, support for Master Mode has been descoped from the I2S peripheral. Master Mode is affected in all modes of operation including all combinations of clock polarity, data delay, word lengths, data formats (I2S/Left-justified and DSP), stereo/mono, and so forth.

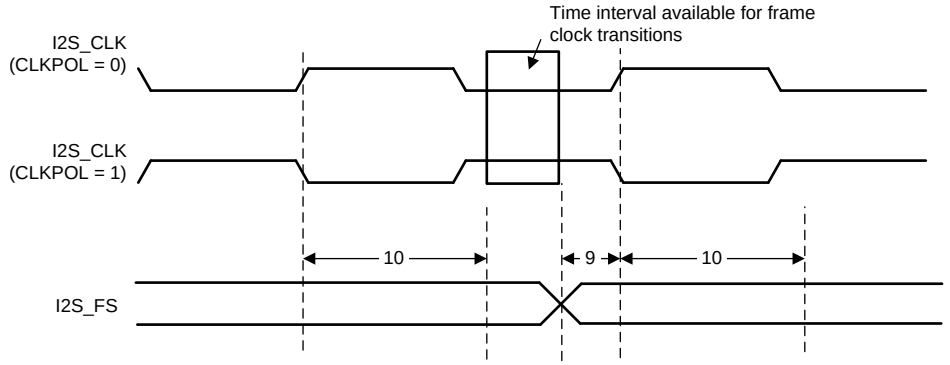

Figure 5. I2S Frame Clock Timing Constant

NO.	PARAMETER	DESCRIPTION
9	$t_{su(FSV-CLKH)}$	Minimum setup time, I2S_FS valid before I2S_CLK high (CLKPOL = 0)
	$t_{su(FSV-CLKL)}$	Minimum setup time, I2S_FS valid before I2S_CLK low (CLKPOL = 1)
10	$t_{h(CLKH-FSV)}$	Minimum hold time, I2S_CLK high to I2S_FS (CLKPOL = 0)
	$t_{h(CLKL-FSV)}$	Minimum hold time, I2S_CLK low to I2S_FS (CLKPOL = 1)

Workaround(s)

No explicit work-around exists to allow the use of I2S in Master Mode due to the potential for hold time violations. For using I2S, the following is recommended:

1. Use I2S in Slave Mode, provided the Frame Clock is delayed with an RC-delay circuit (or equivalent) to satisfy the Frame Clock Timing Requirement in Slave Mode.
2. Use McBSP; it does not have any unique Frame Clock Timing requirement in master or slave mode. The C5517 device has three I2S ports, but only one McBSP port. However, McBSP can operate in I2S mode or Time-Division-Multiplexed Mode for multiple channels.

Revision History

Changes from A Revision (July 2015) to B Revision **Page**

- Added new advisory: *Advisory 2.0.14: I2S Master Mode Descope due to Frame Clock Timing Violation* 24

IMPORTANT NOTICE FOR TI DESIGN INFORMATION AND RESOURCES

Texas Instruments Incorporated ('TI") technical, application or other design advice, services or information, including, but not limited to, reference designs and materials relating to evaluation modules, (collectively, "TI Resources") are intended to assist designers who are developing applications that incorporate TI products; by downloading, accessing or using any particular TI Resource in any way, you (individually or, if you are acting on behalf of a company, your company) agree to use it solely for this purpose and subject to the terms of this Notice.

TI's provision of TI Resources does not expand or otherwise alter TI's applicable published warranties or warranty disclaimers for TI products, and no additional obligations or liabilities arise from TI providing such TI Resources. TI reserves the right to make corrections, enhancements, improvements and other changes to its TI Resources.

You understand and agree that you remain responsible for using your independent analysis, evaluation and judgment in designing your applications and that you have full and exclusive responsibility to assure the safety of your applications and compliance of your applications (and of all TI products used in or for your applications) with all applicable regulations, laws and other applicable requirements. You represent that, with respect to your applications, you have all the necessary expertise to create and implement safeguards that (1) anticipate dangerous consequences of failures, (2) monitor failures and their consequences, and (3) lessen the likelihood of failures that might cause harm and take appropriate actions. You agree that prior to using or distributing any applications that include TI products, you will thoroughly test such applications and the functionality of such TI products as used in such applications. TI has not conducted any testing other than that specifically described in the published documentation for a particular TI Resource.

You are authorized to use, copy and modify any individual TI Resource only in connection with the development of applications that include the TI product(s) identified in such TI Resource. NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE TO ANY OTHER TI INTELLECTUAL PROPERTY RIGHT, AND NO LICENSE TO ANY TECHNOLOGY OR INTELLECTUAL PROPERTY RIGHT OF TI OR ANY THIRD PARTY IS GRANTED HEREIN, including but not limited to any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information regarding or referencing third-party products or services does not constitute a license to use such products or services, or a warranty or endorsement thereof. Use of TI Resources 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.

TI RESOURCES ARE PROVIDED "AS IS" AND WITH ALL FAULTS. TI DISCLAIMS ALL OTHER WARRANTIES OR REPRESENTATIONS, EXPRESS OR IMPLIED, REGARDING TI RESOURCES OR USE THEREOF, INCLUDING BUT NOT LIMITED TO ACCURACY OR COMPLETENESS, TITLE, ANY EPIDEMIC FAILURE WARRANTY AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

TI SHALL NOT BE LIABLE FOR AND SHALL NOT DEFEND OR INDEMNIFY YOU AGAINST ANY CLAIM, INCLUDING BUT NOT LIMITED TO ANY INFRINGEMENT CLAIM THAT RELATES TO OR IS BASED ON ANY COMBINATION OF PRODUCTS EVEN IF DESCRIBED IN TI RESOURCES OR OTHERWISE. IN NO EVENT SHALL TI BE LIABLE FOR ANY ACTUAL, DIRECT, SPECIAL, COLLATERAL, INDIRECT, PUNITIVE, INCIDENTAL, CONSEQUENTIAL OR EXEMPLARY DAMAGES IN CONNECTION WITH OR ARISING OUT OF TI RESOURCES OR USE THEREOF, AND REGARDLESS OF WHETHER TI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You agree to fully indemnify TI and its representatives against any damages, costs, losses, and/or liabilities arising out of your non-compliance with the terms and provisions of this Notice.

This Notice applies to TI Resources. Additional terms apply to the use and purchase of certain types of materials, TI products and services. These include; without limitation, TI's standard terms for semiconductor products http://www.ti.com/sc/docs/stdterms.htm), evaluation modules, and samples (http://www.ti.com/sc/docs/sampterms.htm).

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