OMAP5910

Dual-Core Processor

Silicon Errata

SPRZ016F
August 2002 − Revised February 2006

REVISION HISTORY

This revision history highlights the technical changes made to the SPRZ016E errata to make it an SPRZ016F revision.

Scope: Added new Advisory

PAGE(s) NO.	ADDITIONS/CHANGES/DELETIONS
56	Added Advisory MMC_12

Contents

1 Introduction

This document describes the silicon updates to the functional specifications for the OMAP5910, silicon Revision J. Issues related to DSP operation are documented in the *TMS320C55x DSP CPU Programmer's Reference Supplement* (literature number SPRU652).

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 OMAP™ processors and support tools. Each commercial OMAP platform member has one of three prefixes: X, P, or null (no prefix). 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 (TMDX) through fully qualified production devices/tools (TMDS).

Device development evolutionary flow:

X Experimental device that is not necessarily representative of the final device's electrical specifications and may not use production assembly flow. (TMX definition)

P Prototype device that is not necessarily the final silicon die and may not necessarily meet final electrical specifications. (TMP definition)

null Production version of the silicon die that is fully qualified. (TMS definition)

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."

Production 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 (X or P), 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 Device Markings

Figure 1 provides an example of the OMAP5910 device markings and defines each of the markings. The device revision can be determined by the symbols marked on the top of the package as shown in Figure 1. Some prototype devices may have markings different from those illustrated.

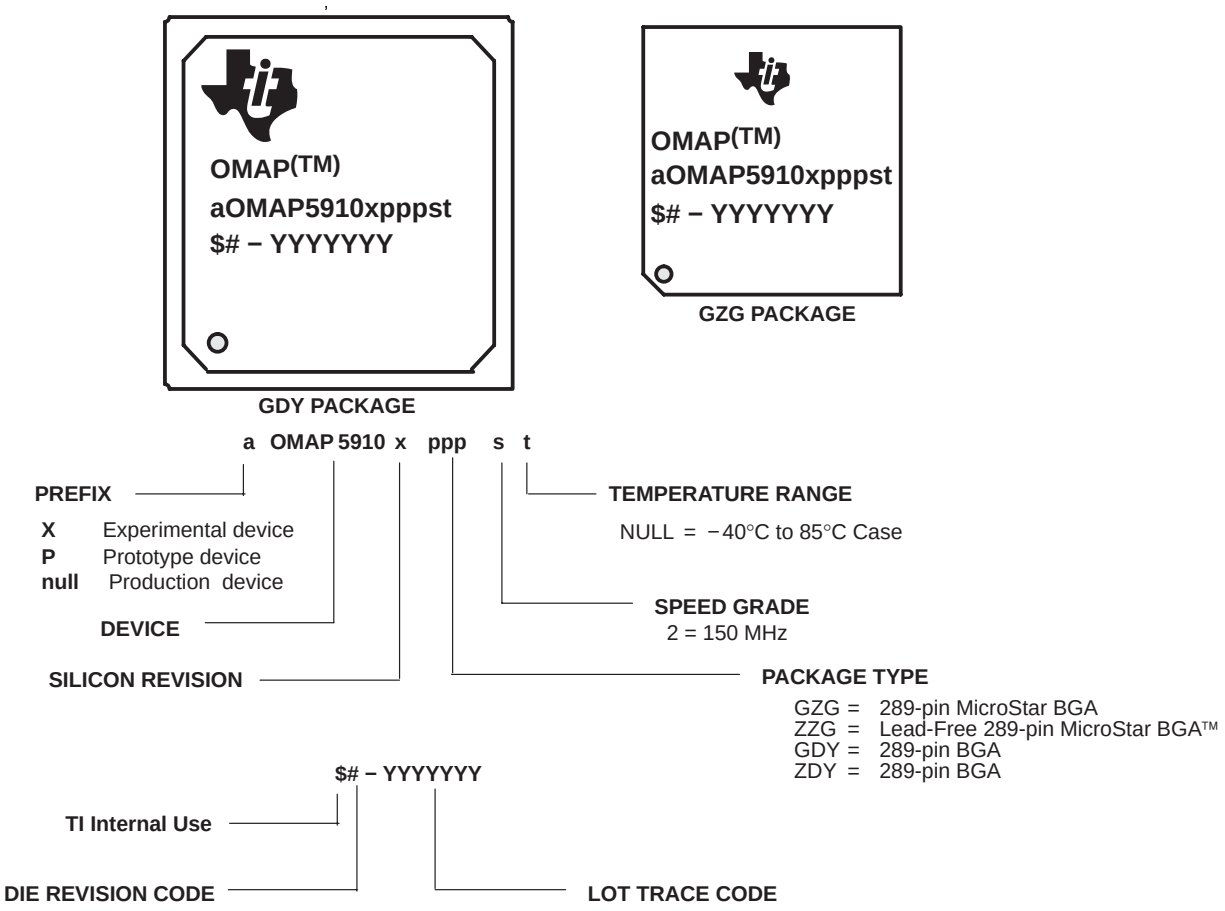


Figure 1. Example Markings for OMAP5910 Packages, Revision J

MicroStar BGA is a trademark of Texas Instruments.

2 Important Notices and Information About OMAP5910

2.1 Useful Information Regarding TMS320C55x™ Assembler Diagnostic Messages

The TMS320C55x™ (C55x™) DSP assembler will generate three types of diagnostic messages when it detects a potential or probable Silicon Exception.

2.1.1 ERROR Diagnostics

The assembler generates ERROR diagnostics in cases where it can fully determine that the code will cause a silicon exception to occur on hardware.

2.1.2 WARNING Diagnostics

The assembler generates WARNING diagnostics in cases where it can fully determine that the code will cause a silicon exception to occur on hardware, but which, under certain circumstances, may not be an issue for the user.

2.1.3 REMARK Diagnostics

The assembler generates REMARK diagnostics in conditions where it can fully determine that the code may cause a silicon exception to occur on hardware, but the exception itself also depends on non-visible trigger conditions that the assembler has no knowledge of, such as whether interrupts are enabled.

Since the assembler cannot determine the state of these trigger conditions, it cannot know that the exception will affect this code. Therefore, it generates a REMARK to instruct the user to examine the code and evaluate whether this is a potential silicon exception situation. (Please see the following sections for how to suppress remarks in situations where you have determined that the other trigger conditions do not exist.)

Intended Treatment of REMARK Diagnostics

The intent of generating REMARK diagnostics is to inform the user that the code could potentially cause a silicon exception and that it should be reviewed by the user side by side with the trigger conditions and a determination be made whether the code is a potential silicon exception situation.

If the code is determined to be a potential silicon exception situation, users should modify their code to prevent that exception from occurring.

If users determine that their code will not cause a silicon exception based on the trigger conditions, then the REMARK that the assembler generates can be suppressed. There are two methods of doing so; please see the “Suppressing REMARK Diagnostics” section.

Suppressing REMARK Diagnostics

Once the user determines that a silicon exception REMARK diagnostic is not appropriate for the code as written, the REMARK diagnostic can be suppressed in one of the following ways.

- REMARK directives
- REMARK command-line options

TMS320C55x and C55x are trademarks of Texas Instruments.

REMARK Directives:

The .noremark.remark directives can be used to suppress the generation of a REMARK diagnostic for particular regions of code. The .noremark directive turns off the generation of a particular REMARK diagnostic. The .remark directive re-enables the generation of a particular REMARK diagnostic.

A '.noremark ##' (where ## is the remark id) directive is placed at the beginning of the region, and a '.remark ##' directive is placed at the end of the region.

> **NOTE:** The .noremark.remark directive combination should always be placed around the entire region of code that participates in the potential silicon exception. Otherwise, spurious diagnostics may still be generated.

Additionally, the user has the option of disabling a silicon exception diagnostic for the entire file by placing just the .noremark directive at the top of the assembly file. However, this may be dangerous if, during inevitable code maintenance, the code is modified by someone not familiar with all the exception conditions. Please take great care when using the directives in this manner.

REMARK Command-Line Options:

The compiler shell (cl55) supports a command line option to suppress a particular REMARK diagnostic. The shell option −ar# (where # is the assembler's silicon exception id as described above) will suppress the named REMARK for the entire scope of all assembly files compiled with that command. Using the option −ar without a number will suppress all REMARK diagnostics.

Again, this may be dangerous if, during inevitable code maintenance, the code is modified by someone not familiar with all the silicon exception conditions. Please take great care when using the command-line REMARK options. Using the .noremark/.remark directives covering the shortest possible range of source lines is much safer.

3 Usage Notes

Usage Notes highlight and describe particular situations where the device's behavior may not match the 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.

Usage_Note_1*:* MPU Public/Private TIPB Strobe0 and 1 Frequency Limitation

The frequency of the MPU Public and Private TIPB strobe0 and 1 are derived from the traffic controller clock or CLKM3 block. Both Strobe0 and Strobe1 frequencies must be less than or equal to 40 MHz.

The TIPB_CNTL register bit fields [3:0] and [7:4] control the divide factor for strobe0/ strobe1 frequencies. The divide factor for both is shown below.

BIT FIELD TIPB_CNTL [3:0] and [7:4]	MPU TIPB STROBE1 and STROBE2 FREQUENCY
0	TC Clk / 1
1	TC Clk / 2
2	TC Clk / 4
3	TC Clk / 6
…	…
15	TC Clk / 30

Example:

If the TC clock is running at 70MHz, then the strobe divide factor should be set to 1. That is 70MHz / 2 = 35MHz which is less than 40MHz and meets the requirement.

Usage_Note_2: DSP TIPB Strobe1 and 2 Frequency Limitation

The strobe frequency for DSP peripherals is derived from the DSP clock or CLKM2 block and is configured by the control mode register (CMR). In the CMR, bits 5−3 control strobe1 frequency and bits 8−6 control strobe2 frequency. The frequency of these strobes should be less than or equal to 32 MHz. The divide factor for the strobes is shown below.

BIT FIELD CMR [5:3] and [8:6]	DSP TIPB STROBE1 and STROBE2 FREQUENCY
0	DSP Clk / 2
1	DSP Clk / 3
2	DSP Clk / 4
3	DSP Clk / 5
4	DSP Clk / 6
5	DSP Clk / 7
6	DSP Clk / 8
7	DSP Clk / 9

Example:

If the DSP clock frequency is 120MHz, then the bit fields should be set to a wait state of 2. That is 120MHz / 4 = 30MHz which meets the requirement. The bit fields should be programmed with a 010 binary.

When the DSP is in SAM (Shared Access Mode), the DSP clock is used as the input to the wait state generator that creates the DSP TIPB strobe1 and strobe2 signals. However, while the DSP is in HOM (Host Only Mode) the Strobe 1 and 2 are derived by the MPUI Strobe.

Usage_Note_3: Configuring the MPUI Strobe

When the MPU accesses the DSP memory or its peripherals it communicates via the MPUI port. The MPUI strobe clock is derived from the TC or CLKM3 block. If the DSP is in HOM mode, then the MPUI strobe is used for DSP peripheral access. The frequency of this strobe should be less than or equal to 24MHz The strobe frequency for the MPUI is controlled by the MPUI : CTRL_REG bits 7−4. The divide factor for the strobes is shown below.

BIT FIELD CNTL_REG [7:4]	MPUI STROBE FREQUENCY
0	TC Clk / 1
1	TC Clk /2
2	TC Clk / 4
3	TC Clk / 6
…	…
15	TC Clk / 30

Example:

If the TC clock is configured to 60MHz then the access factor should be set to 2.
That is 60MHz / 4 = 15MHz which is less than the required 24MHz. The bit field is to be programmed with a 010.

Usage_Note_4: ARMPER_CK Maximum Frequency

The ARM peripheral clock frequency (ARMPER_CK) is determined by dividing CK_GEN1 by the value associated with the PERDIV[1:0] field of the ARM_CKCTL register (0xFFFECE00). The permissible divisors are 1, 2, 4, or 8. For OMAP5910, the ARMPER_CK frequency is limited to a maximum of 50MHz. Therefore, depending on the output frequency selected for CK_GEN1, an appropriate divisor must be selected for PERDIV.

BIT FIELD PERDIV	ARMPER_CK FREQUENCY
0	DPLL1 Clk / 1
1	DPLL1 Clk /2
2	DPLL1 Clk / 4
3	DPLL1 Clk / 8

Example:

If the frequency of CK_GEN1 is 120MHz, then it is necessary to divide this clock by 4 to produce a 30MHz ARMPER_CK. In this case, PERDIV[1:0] would be programmed to 10b.

Usage_Note_5 DSP MMU Clock frequency limitations

The DSPMMU clock must adhere to all the following rules:

1. DSPMMU clock frequency must ≥ traffic controller clock
2. DSPMMU clock frequency must be 1/2x or 1x the DSP clock
3. DSPMMU frequency should not be greater than max frequency of the traffic controller

Usage_Note_6 13 MHz ULPD Clock

A new feature has been added to the TMS OMAP5910 device allowing a 13MHz−clock reference to be used instead of a 12MHz−clock reference. The feature allows the APLL in the ULPD module to generate the required 48MHz−clock signal from either the 12MHz or 13MHz reference. When using the on−chip oscillator (normal mode), either a 12MHz crystal or a 13MHz crystal can be connected to the OSC1_IN and OSC1_OUT. In external master mode, the on−chip oscillator is disabled and a 12MHz or 13MHz reference clock must be provided by an external oscillator connected to the OSC1_IN pin. In both of these cases, the APLL may be configured to generate the 48MHz clock from either a 12MHz or a 13MHz−clock. If a 13MHz−clock is used, then the APLL should be used to generate the 48MHz clock as opposed to using the ULPD DPLL. The ULPD DPLL can only generate a 48MHz clock when the reference is 12MHz.

Usage_Note_7: PLL Usage Recommendation

To reduce jitter, minimize power supply noise on V_{DD4} and V_{DDA} by the use of filters or bypass capacitors. A potential solution is to use the following circuit:

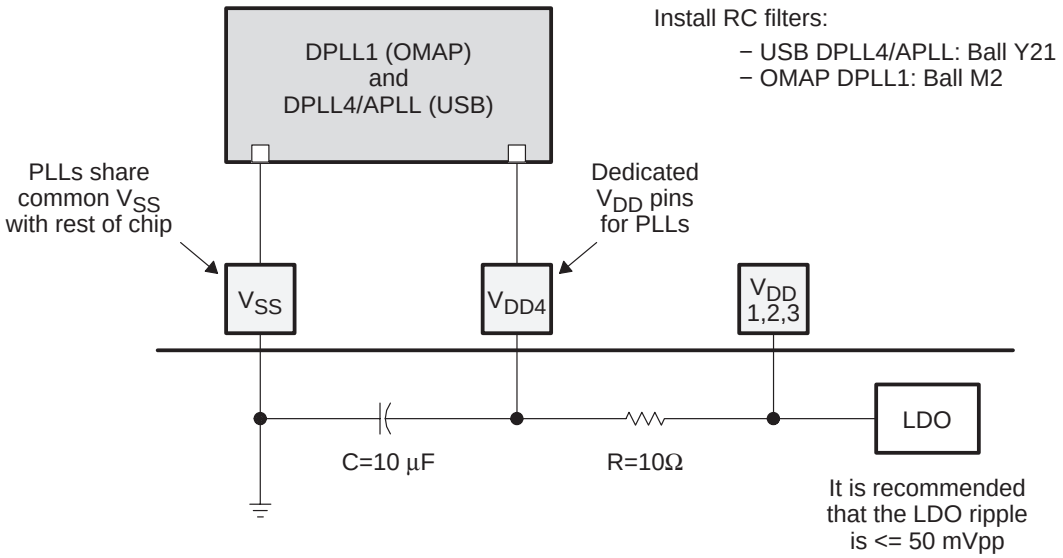


Figure 2. PLL Diagram

Usage_Note_8: Clock Control and Configuration Guidelines.

OMAP CLK GLOBAL RULES: The following rules need to be followed while choosing any dividers for OMAP. For all clock modes:

1. TC cannot run faster than ARM.
2. TC cannot run faster than DSPMMU
3. DSP speed has to be 1x or 2x of DSPMMU
4. All dividers should be chosen not to exceed maximum physical speed limits for each domain.

GLOBAL RULES apply to all modes of operations. In addition to global rules, each clock mode and local rules outlined in the following. For each clock mode, both the global rules and its own local rules have to be followed.

Local rules for each clock mode are as follows:

Fully Synchronous Mode (ARM_SYSST register, CLOCK_SELECT[2:0] = 000b)

The following rules apply to fully synchronous mode:

- In fully synchronous mode ARM, TC, and DSPMMU must run at the same frequency.
- In fully synchronous mode, DSP Speed can be 1x or 2x of DSPMMU.
- Valid dividers for fully synchronous mode:

Fully Synchronous Mode Dividers			
ARM DIV	**TC DIV**	**MMU DIV**	**DSP DIV**
00	00	00	00
01	01	01	00
01	01	01	01
10	10	10	01
10	10	10	10
11	11	11	10
11	11	11	11

Synchronous Scalable Mode (ARM_SYSST register, CLOCK_SELECT[2:0] = 010b)

The following rules apply to synchronous scalable mode:

- In synchronous scalable mode, the ARM and DSPMMU must be at least a 2x multiple of TC.
- The ARM and DSPMMU can be the same or different multiples of each other (i.e., DSPMMU can be 2x and ARM can be 4x of TC).
- The DSPMMU cannot run faster than the maximum physical speed of the TC.
- Valid dividers for synchronous scalable mode:

Usage Note 8: PLL Usage Recommendation (Continued)

Table 1. Synchronous Scalable Mode Dividers

ARM DIV	TC DIV	MMU DIV	DSP DIV
00	01	00	00
00	10	00	00
00	10	01	00
00	10	01	01
01	10	01	00
01	10	01	01
00	11	00	00
00	11	01	00
00	11	01	01
00	11	10	01
00	11	10	10
01	11	01	00
01	11	01	01
01	11	10	01
01	11	10	10
10	11	10	01
10	11	10	10

Mixed Mode #3 (ARM_SYSST register, CLOCK_SELECT[2:0] = 100b)

The following rules apply to mixed mode #3:

- The ARM and TC must be at the same frequency
- The DSPMMU must be at least a 2x multiple of the TC.
- The DSPMMU cannot run faster than the maximum physical speed of the TC.
- Valid dividers for mixed mode #3:

Table 2. Mixed Mode #3 Dividers

ARM DIV	TC DIV	MMU DIV	DSP DIV
01	01	00	00
10	10	00	00
10	10	01	00
10	10	01	01
11	11	00	00
11	11	01	00
11	11	01	01
11	11	10	01
11	11	10	10

Mixed Mode #4 (ARM_SYSST register, CLOCK_SELECT[2:0] = 111b)

The following rules apply to mixed mode #3:

- DSPMMU and TC must be at the same frequency
- ARM must be at least a 2x multiple of the TC.
- Valid dividers for mixed mode #4:

Table 3. Mixed Mode #4 Dividers

ARM DIV	TC DIV	MMU DIV	DSP DIV
01	01	01	00
00	01	01	01
00	10	10	01
00	10	10	10
01	10	10	01
01	10	10	10
01	11	11	10
01	11	11	11
10	11	11	10
10	11	11	11
00	11	11	10
00	11	11	11

Usage_Note_9: Programming Guidelines for LCD Clock and Pixel Clock Frequencies

General Rule:

The LCD clock divider and the Pixel clock divider are not designed to be modified while the LCD controller is on. There is no hardware synchronization handshaking between LCD output to panel (vysnc) and when to change the pixel clock or LCD functional clock frequencies. If you attempt to change these frequencies without turning off the LCD controller first, then be aware the pixel clock frequency will be changed somewhere in the middle of the current video frame.

However, depending upon the tolerance of the specific panel connected to the controller, here are some guidelines you should follow.

To change LCD Pixel Clock frequency:

Option 1:

- Change PCD value at any time, preferably using the DMS's led channel "End Of Frame" interrupt as a trigger (Not guaranteed to update before the next frame starts).

Option 2:

- Disable the LCD ControllerWait for the frame done interrupt (be sure frame done is not masked) of the LCD controller.
- Program the new PCD value
- Program the new TC and LCD frequencies (if needed)
- Re-enable the LCD controller

NOTE: Some panels will flicker when the LCD controller is disabled.

Usage_Note_10: USB_W2FC Suspend Functionality in HMC_MODE13 and HMC_MODE15

When in HMC_MODE13 or HMC_MODE15, the TLL receives its suspend and pullup enable signals from the USB_W2FC. In its default operating mode, the USB_W2FC will not transition from suspend to enabled until it senses that it has successfully enabled the pullup. The TLL prioritizes its suspend input over its pullup enable input, so the TLL will not signal the presence of the pullup, and the USB_W2FC cannot sense that it has enabled the pullup. The USB_W2FC will not exit Suspend mode. Do not use transceiverless link logic with the USB_W2FC and an external USB Host Controller. Use a transceiver-based solution instead.

Usage_Note_11: Remote Wake Non-functional Through TLL in HMC_MODEs 9, 10, 11, 12, 14, 21, 23, 24, and 25

When in HMC_MODE 9, 10, 11, 12, 14, 21, 23, 24, or 25, the USB host will not receive remote wake from an external USB function controller connected to a transceiverless link logic. The external USB function controller cannot wake the USB link from suspend using remote wake. This does not affect other connectivity that does not use the transceiverless link logic.

1. Systems that do not use USB Remote wake through the transceiverless link logic may ignore this usage note.

2. Use a transceiver-based connection (may require use of a different HMC_MODE or a different set of OMAP5910 pins).
3. Use an OMAP5910 GPIO pin and software monitoring to indicate when software should take the appropriate USB host controller port out of USB suspend.
4. Avoid putting the USB host port that uses the transceiverless link logic into USB suspend.

Usage_Note_12: DMA Configuration Recommendations

Table 4 shows how the DMACK_REQ, IDLIF_ARM and DMA_autogating bits are to be configured for correct functionality and optimal power saving.

Table 4. NIL Configuration for DMACK_REQ, IDLIF_ARM, and DMA_autogating Bits

#	DMACK_REQ	IDLIF_ARM	Autogating_on	Comments
1	1	X	1	DO NOT USE
2	0	1	1	Use this only when: the LCD is DISABLED and it is appropriate for the DMA to go to IDLE mode when the ARM goes into IDLE mode. Characteristics: When the ARM goes into IDLE mode, the DMA goes into IDLE mode if there is no activity on ports. The DMA will then stay in IDLE mode until the ARM comes out of IDLE. Even a hardware request cannot wake up the DMA from IDLE mode. The hardware request is processed after the ARM comes out of IDLE mode. **Note:** This mode must be used for OMAP to enter into CHIP IDLE. This is the recommended mode.
3	0	0	1	Use this only when: It is required for the DMA to process requests and transfer data even when the ARM is in IDLE mode. OR the LCD is ENABLED. **Note:** For OMAP to go into CHIP IDLE, the LCD must be DISABLED first, then the IDLIF_ARM bit must be set to 1 (case #2).

NOTES:

1. DMACK_REQ −> CLKM IDLECT2(8)
2. IDLIF_ARM −> CLKMIDLECT1(6)
3. DMA_autogating_ON −> DMA GCR(3)
4. LCD ENABLE/DISABLE −> LCD Control Register (LCD_EN)

4 DSP Subsystem Advisories

4.1 DSP System Advisories

Advisory DSP_SYS_1	*Use Caution When Reading Following a Configuration Change on The DSP*

Revision(s) Affected: All Revisions

Details: Within the C55x CPU, writes occur later in the pipeline than reads do. This allows reads from a later instruction to sometimes occur prior to the write of an earlier instruction.

This can be a problem when the write is to a configuration register in a peripheral that affects the read. If the user intends to reads memory that can be affected by the new configuration it is recommended that the user read from the configuration register being programmed prior to memory reads.

Reads from the TIPB interface are pipeline protected, so writes and reads of peripheral registers are not affected. Also, sequential writes to peripherals work correctly.

Workaround: After configuring a peripheral the software should ensure that the writes have completed prior to using that configuration. This can be done with a read from the last register written or waiting 3 cycles. This configuration constraint is common to pipeline architectures.

This exception will not be fixed in future silicon revisions.

4.2 DSP DMA Advisories

Advisory DSP_DMA_1	*DSP EMIF/DMA Port Hangs During EMIF Bus Error*

Revision(s) Affected: All Revisions

Details: If the EMIF times out on an access, the DSP will get a time-out bus-error interrupt. The time-out condition may also cause a DMA interrupt. During this DMA interrupt, the DMA will not time out and go into an unknown state.

Workaround: Whenever an EMIF bus error interrupt occurs, the software needs to RESET the DMA and reschedule the transfer.

This exception will not be fixed in future silicon revisions.

Advisory DSP_DMA_2 *DSP DMA IDLE Prevents Transfer Completion*

Revision(s) Affected: All Revisions

Details: When DSP peripherals are placed in IDLE, there is an internal hardware handshaking mechanism between the DSP and its peripherals that ensure that the IDLE can occur. If a DMA transfer is occurring, this handshaking is supposed to prevent IDLEing until after the transfer is complete. The DSP DMA, however, can go into IDLE during the middle of a transfer resulting in the transfer not completing.

Workaround: In order to enforce that all DMA transfers are complete before attempting to IDLE the DMA, the DMA status first needs to be checked. Afterwards, the DMA channels need to be disabled from which the IDLE instruction can then be safely executed.

This exception will not be fixed in future silicon revisions.

Advisory DSP_DMA_3 *Potential Deadlock in Burst Accesses*

Revision(s) Affected: All Revisions

Details: If a transfer is configured with burst enabled and any of the accessed addresses (notably start address) are not 4x32-bits aligned (i.e., byte address is not multiple of 16) then the DMA may deadlock and the transfer may never terminate.

Workaround: Configure start address, block size, frame size, element size and indexes such that all DMA burst accesses are made on 4x32-bits aligned addresses.

This same functional limitation is present on the System DMA controller, but the programming restrictions stated in this workaround are documented in the *OMAP5910 Dual-Core Processor System DMA Controller Reference Guide* (literature number SPRU674).

This exception will not be fixed in future silicon revisions.

Advisory DSP_DMA_4 *DSP DMA IDLE and Transfer Completion*

Revision(s) Affected: All Revisions

Details: When system peripherals are placed in IDLE, there is hardware handshaking to ensure that an IDLE can occur without any system consequences. The DSP DMA, however, can go into IDLE in the middle of a transfer. This may prevent the transfer from completing.

Workaround: In order to put the DSP DMA into IDLE after the completion of a transfer, the DSP DMA status needs to be checked and the channels need to be disabled. Then, the IDLE instruction can safely be executed.

This exception will not be fixed in future silicon revisions.

Advisory DSP_DMA_5 *DSP DMA Potential Deadlock in Burst Accesses*

Revision(s) Affected: All Revisions

Details: If a DSP DMA transfer is configured with burst enabled and some of the accessed addresses (notably the start address) are not 4x32-bits aligned (i.e., byte address is not multiple of 16), then the DSP DMA may deadlock and the transfer may never terminate.

Workaround: Two workarounds are available:

1. To use the DSP DMA for burst access on both source and destination, the start address, block size, frame size, element size and indexes must be set in such a way that all DMA accesses are made on 4x32−bit aligned addresses.
2. If the DSP DMA is configured for burst access on only one side (source or destination). Burst accesses do not need to be 4x32 bit aligned. On the side that does not use bursting, packing should be used for improved performance.

This exception will not be fixed in future silicon revisions.

4.3 DSP ICACHE Advisories

Advisory DSP_ICACHE_1	*DSP Instruction Cache Two-Way Set-Associative Line Valid Bit Set Incorrectly*

Revision(s) Affected: Fixed in Revision J. Applicable to all other revisions.

Details: Under an unlikely combination of events, there is a possibility for an instruction cache line valid bit to be set incorrectly to valid while the instruction packet (data contained in the line) is not actually correct. Thus, if there is ever a hit to this line, the instruction cache will return an invalid instruction packet and the DSP could try to decode this invalid instruction packet. The following configuration and events are necessary for this issue to occur:

1. DSP instruction cache is configured in 2-way, set-associative mode (note that neither direct mapped or RAMSET modes are affected by this issue).
2. DSP, DSP_MMU, and TC clocks are the same frequency (synchronous with each other).
3. DSP program is in either external SDRAM or internal RAM.
4. The DSP instruction cache has been globally flushed during the invocation of a new program. (The DSP's program is changed and the cache is flushed.)
5. And there is, particular to this condition, a combination of out-of-order DSP accesses to a cache line (speculative branch) and specific timing-related events.

Workaround: Due to the timing nature of the problem, the condition should not occur if the fill rate of the instruction cache is always 1/4th or lower the DSP frequency. It is possible to respect this constraint by observing the following rules:

1. The DSP program is located in either DSP internal SARAM/DARAM or external EMIFF (SDRAM) or EMIFS (FLASH, SRAM) memory space and not the internal 1.5M-bit RAM space.
2. The Traffic Controller and DSP MMU clock rate are one-half or lower the DSP clock rate. That is, the DSP should be running at twice or higher the frequency of the Traffic Controller and DSP MMU. Because the external interfaces are 16 bits wide and the instruction cache fills at 32 bit rate, the 1/4th fill rate constraint will be respected.

Advisory DSP_ICACHE_2 — *Two-Way Misses to the Same Line Can Become Corrupted*

Revision(s) Affected: All Revisions

Details:

A failure may occur when a speculative branch occurs inside the DSP which coincides with a miss to the same line inside the I-cache.

Within the C55x DSP core, the Instruction Buffer Queue (IBQ) always tries to stay several instruction fetches ahead of the decoder. Any time a conditional branch occurs within the DSP, the false condition would be to continue executing the next instruction, which is already in the IBQ. Therefore, in anticipation that the conditional test might be true, the IBQ sometimes fetches the instruction from the true condition as well. This allows for the necessary code to exist within the IBQ for quick access, regardless of whether the condition proves to be true or false. This operation of fetching the instructions for both true and false conditions is referred to as speculative fetching.

Since instructions are fed into the IBQ 4 bytes per fetch, but are extracted from the IBQ at a variable rate of 1 to 6 bytes per instruction, the IBQ will stop fetching when it is sufficiently far ahead of the instruction decoder.

- For relative branches, a speculative fetch happens for offsets of 0x20 to 0x27.
- For absolute branches, speculative fetches always occur.
- For conditional goto instructions, the IBQ (instruction buffer queue) will fetch the target address prior to the condition being evaluated.

This failure exists if the speculative fetches and normal execution fetches miss within the same I-cache line as will be shown below.

Each I-cache line consists of 16 bytes, which are addressed in a sequence of 4 bytes, addressable on 4-byte boundaries.

Consider the following program sequence...

```
if ( cond false ) goto A3...
...
A1 ...
A2 ...
...
A3 ...
```

Typically, when A1 is returned from the I-cache, A2 is requested next. When the conditional goto is decoded, the DSP will request A3 for the speculative fetch.

The IBQ stores requests prior to decode, so many instructions can be fetched from the I-cache prior to decode. While the IBQ is filling up and addresses are sequentially fetched, the address A1 is eventually requested.

Two-Way Misses to the Same Line Can Become Corrupted (Continued)

If the IBQ is sufficiently far ahead of the decoder, the DSP will stop fetching prior to requesting A2. If the conditional branch is decoded at this point, the speculative fetch will occur next, retrieving A3. The resulting order of fetches would then be A1−A3−A2.

If the test of the conditional branch results in a false condition and following occurs:

−The resulting order of A1−A3−A2 matches one of the conditions listed below.

−The instructions all exist within the same I-cache line.

−The instruction request from the cache results in a cache miss.

Then the program may become corrupted.

A1	A3	A2	
0	0	8	
0	0	C	
0	8	0	
0	8	4	
0	C	0	
0	C	4	← This is the SW example used below.
0	C	8	
4	0	C	
4	4	C	
4	C	0	
4	C	4	
4	C	8	

Addresses coinciding with this sequence will not necessarily cause a problem since the corner condition depends on the latency of the target memory and the time when the DSP issues the program request. In order to guarantee that this condition is not present, the following sequences need to be avoided.The hardware corner case is very rare (17 out of 100 million cycles of targeted random simulation) since it is based on protocol interactions between the DSP and target memory, IBQ state, and the I-cache line needs to be a miss. Of all of the theoretical values for A1, A2 and A3 within the same I-cache line, only those above have the potential to exhibit the error. However, since this corner case is difficult to isolate in software, speculative fetches need to be isolated entirely to avoid this condition.

If this condition occurs, an invalid 32 bit instruction packet is returned by the I-cache. Specifically, the data packet for A2 is returned incorrectly to CPU with the same content as the data packet for A3. The result may cause unexpected CPU operations or even halt the execution of CPU. So far this condition has manifested itself as:

1) Corruption of PC.
2) Cause extra looping.
3) Cause less looping.

Two-Way Misses to the Same Line Can Become Corrupted (Continued)

This problem can be masked if a breakpoint is set before A1 and then stepping though A2 and A3. The fetches will return correct data this way. Therefore, one way to confirm this condition is to set a breakpoint before A1, change A2's content to be same as A3 with a debugger write to external memory and, then, step through those portions of the code. The result should be the same as the condition found by continued execution of the normal code. See example 1.

Example 1)

Absolute Conditional Branch is the failing test case shown here:

```
MAIN_2
.align 16
0x400000 nop
0x400001 nop
0x400002 nop
0x400003 goto next0
0x400005 nop
0x400006 nop
0x400007 nop
0x400008 next0 nop
0x400009 nop
0x40000a nop
0x40000b nop
0x40000c nop
0x40000d nop
0x40000e repeat (#79)
0x400010 nop
0x400011 nop
0x400012 if (ac1 != #0) goto 0x40002c ; goto A3
0x400017 nop
0x400018 nop_16 || nop_16
0x40001c nop_16 || nop_16
0x400020 nop_16 || nop_16 ;A1
0x400024 AC0 = #3 ;A2
0x400026 nop_16
0x400028 nop_16 || nop_16
0x40002c AR0 = AR0 + #1 ;A3
0x40002e nop
0x40002f nop
0x400030 nop_16 || nop_16
0x400034 nop_16 || nop_16
0x400038 nop_16 || nop_16
0x40003c nop_16 || nop_16
```

Workaround: Note that because of the corner case nature of this condition and the low likelihood of encountering this problem, the I-cache can be used successfully. However, if a problem is suspected or found then the following actions can be taken:

1. Disable I-cache
2. Freeze I-cache 32 bytes prior to a conditional branch
3. Freeze I-cache 32 bytes prior to a relative branch >= 0x20

TI is implementing a fix on revisions C and above.

Advisory DSP_ICACHE_3	*DSP ICACHE is Servicing a Missed Program Bus Request During a RAMSET Preload*

Revision(s) Affected: All Revisions

Details:

The two ramset banks inside the DSP ICACHE are preloaded when the corresponding ramset tag registers' content are updated through I/O access. The ramset preload uses the same line fill mechanism as cache miss line−fill, therefore there is a conflict of resource when ramset refill happens while I−cache miss line−fill is ongoing. Although there is logic to arbitrate this conflict, a functional bug can corrupt the content of Tag ram under the following corner condition:

- I-cache is turned on with ramset(s) enabled.
- The instruction(s) for updating the ramset tag register(s) are embedded with other program code in external memory.
- The external memory space where the instruction(s) of updating the ramset tag(s) are not present in ICACHE, i.e. will generate a cache miss.

Example

The following diagram describes assembly code that is likely to encounter this corner case:

```
Address      Instruction
Ext Mem      DSP code
Ext Mem      DSP code
……
......
Ext Mem      DSP code
Ext Mem      DSP code
Ext Mem      *port(#0x1400) = #0xc 2f ; Configure icache
Ext Mem      bit(ST3,#14) = #1 ; Turn on icache
Ext Mem      *port(#0x1406) = #0x0800 ; Update ramset tag
;for bank1
Ext Mem      *port(#0x1408) = #0x0801 ; Update ramset tag
;for bank2
Ext Mem      DSP code
Ext Mem      DSP coI......
......
Ext Mem      DSP code
Ext Mem      DSP code
```

While the ramset tag is being updated by the I/O bus, the CPU IBQ is prefetching instructions not present in the ICACHE. There are some latencies in completing I/O access to ramset tag registers, the exact time of when ramset tag update happens is not controllable.

Workaround(s):

1. I-cache must be turned on with ramset(s) enabled.
2. Branch from external memory program code to the ramset tag update code.
3. Continue polling the ramset tag valid bit to make sure the ramset preload has finished.
4. Branch back to external memory for normal program execution after the preload has finished.

DSP ICACHE is Servicing a Missed Program Bus Request During a RAMSET Preload (Continued)

Example:

```
Address      Instruction

Ext Mem      DSP code
Ext Mem      DSP code
…….
.......
Ext Mem      DSP code
Ext Mem      DSP code
Ext Mem      *port(#0x1400) #0xce2f          ; Configure icache
Ext Mem      bit(ST3,#14) = #1               ; Turn on icache
EXt Mem      goto Preload_ramsets
Preload_     ramsets:
Int Mem      *port(#0x1406) = #0x0800        ; Update ramset tag
                                             ; for bank1
Scan0:
Int Mem      TC1 = bit(*port(0x1405), #15)   ; Read the Tag Valid
                                             ; bit of ramset bank 0
Int Mem      nop_16
Int Mem      if (!TC1) goto Scan0            ; Continue polling if the
                                             ; Tag Valid bit is not 1.
Int Mem      *port(#0x1408) = #0x0801        ; Update ramset tag
                                             ; for bank1
Scan1:
Int Mem      TC1 = bit(*port(0x1407), #15)   ; Read the Tag Valid
                                             ; bit of ramset bank 1
Int Mem      nop_16
Int Mem      if (!TC1) goto Scan1            ; Continue polling if the
                                             ; Tag Valid bit is not 1.
Int Mem      goto Back_from_ramset_preload
Back_from_ramset_preload:
Ext Mem      DSP code
Ext Mem      I Ie
.......
.......
Ext Mem      DSP code
Ext Mem      DSP code
```

This exception will not be fixed in future silicon revisions.

4.4 DSP Emulation Advisories

Advisory DSP_EMU_1	*Hardware Breakpoint Set on the Instruction Immediately Following a Conditional Instruction Fails to Halt CPU*

Revision(s) Affected: All Revisions

Details: A hardware breakpoint set on the instruction immediately following a conditional instruction (conditional branch, conditional return, conditional call) will not halt the CPU when the condition check fails. Application code continues to execute without error, but the user will not see execution halt at the expected location. The source of the hardware breakpoint does not matter. This applies only to hardware breakpoints set on the first instruction after the conditional. Breakpoints set using the Address Control Unit, Data Control Unit, or External Breakpoint Watch Point all fail. Hardware breakpoints work in all other cases. An assembly code example is shown below.

Example 1)

```
 if (cond) call P24
       INST1        << Putting hardware breakpoint here is the problem.
       INST2
       ...

       ...
if (cond) call P24
       INST1
       INST2        << you can set hardware breakpoint here.
       ...
```

When debugging assembly code in ROM, users must avoid setting breakpoints (ROM breakpoints must be hardware breakpoints) on the instruction following a conditional. There may be some assembly constructs like jump tables that have several conditional calls sequentially in code. Users must be aware that hardware breakpoints should not be used on these type of constructs. An example of several sequential conditional calls is shown below.

Example 2)

```
...
if (cond1) call Func1
if (cond2) call Func2  << Putting HWBP here is a problem.
if (cond3) call Func3  << Putting HWBP here is a problem.
if (cond4) call Func4  << Putting HWBP here is a problem.
...
```

When using Code Composer Studio™ Integrated Development Environment (IDE) to debug “C” code in ROM, the debugger automatically sets hardware breakpoints when the user wants to “step over” and “step out” of function calls. The user cannot prevent the breakpoint from being set on the instruction immediately after a conditional. This results in the user seeing the target continue to run when stepping in “C” code. This does not corrupt the target state.

Code Composer Studio is a trademark of Texas Instruments.

Hardware Breakpoint Set on the Instruction Immediately Following a Conditional Instruction Fails to Halt CPU (Continued)

Workaround: Avoid putting a breakpoint at the instruction immediately following the conditional branch/call/return instruction.

If a breakpoint is absolutely needed, then add a NOP between the conditional branch/call/return and the first instruction immediately following it. See the example below.

Example:

```
if (cond) call P24
NOP
INST1                << you can set HWBP here.
INST2
 ...
```

This exception will not be fixed in future silicon revisions.

Advisory DSP_EMU_2	*DSP IDLE Interrupt Not Serviced When Emulator is Connected*

Revision(s) Affected: All Revisions

Details: A DSP IDLE interrupt is not serviced when an emulator is connected as shown in the following sequence:

1. Emulator connected
2. RT mode is set
3. Run test code
   ASM codes sets INTM
   ASM code clears all pending interrupts in IFR
   ASM code configures timers, but does not enable
   ASM code sets ICR to 0x3F
4. Emulator halts CPU
   PC indicates halted at IDLE instruction
   DBGSTAT = 0x805F (IDLE, DSUSP, and FXWOK)
   ISR = 0x3F
   No interrupts pending in IFR
5. Provide single interrupt.
6. Enable RT int by emulation write to DBIER0
   Halted background code immediately comes out of idle
   ISR = 0x3F (interrupt not taken)

Workaround: There is no workaround. The emulator result is different from the functional mode (which will service the ISR). This only affects the debugger mode. Therefore, do not use IDLE when the emulator is connected.

4.5 MPU Emulation Advisories

Advisory MPU_EMU_1	*The W2FC Data Register Should Not Be Read From The Emulator*

Revision(s) Affected: All Revisions

Details:

Emulation read access (when the ARM TIPB suspend is active) for the USB_W2FCDATA register only works if the end point (the last offset address) is selected. If endpoint is not selected, there is no response from the USB W2FC and the ARM processor will hang.

Emulation read access (when the ARM TIPB suspend is active) for the USB_W2FCDATA_DMA register works only when the DMA request is active. If DMA is not active, there is no response from the USB W2FC and the system DMA controller hangs.

The register affected are:

DATA = 0xFFFB:4008
DATA_DMA = 0xFFFB:4048

No other registers in the USB W2FC are affected.

Workaround:

Do not read these register using emulation. Code Composer Studio's memory map may be configured so that memory window will not cause accesses to these registers. See the options below:

Option 1:

Use Code Composer Studio "Options" pull down menu's "Memory Map" window to define the two registers as "protected". This sequence will need to be performed every time Code Composer Studio is launched. Perform the following steps:

1. Select "Memory Map" on the "Options" menu.
2. Set the Starting Address field to "0xFFFB4008".
3. Set the Length field to "4".
4. Set the Attribute field to "None" (No Memory/Protected).
5. Set the Access Size field to "Auto".
6. Make sure the Memory Mapping box is checked.
7. Click the "Add" button
8. Set the Starting Address field to "0xFFFB4048".
9. Set the Length field to "4".
10. Set the Attribute field to "None" (No Memory/Protected).
11. Set the Access Size field to "Auto".
12. Make sure the Memory Mapping box is checked.

The W2FC Data Register Should Not Be Read From The Emulator (Continued)

13. Click the ”Add” button
14. The Memory Map List window should now include the following lines:
    0xFFFB4008−0xFFFB400B: NONE
    0xFFFB4048-0xFFFB4004B: NONE
15. Click the OK button. See the Code Composer Studio on-line Help for further details on Memory Mapping.

Option 2:

Modify the Code Composer Studio GEL file that is automatically loaded at Code Composer Studio boot time so that it pre-initializes the memory map in order for these register to be protected. Add the following lines to the GEL file:

- GEL_MapAddStr(0xFFFB4000,0,0x00000008,“W|A|S2”,0
  GEL_MapAddStr(0xFFFB4008,0,0x00000004,“NONE”,0
  GEL_MapAddStr(0xFFFB400C,0,0x0000003C,“W|A|S2”,0
  GEL_MapAddStr(0xFFFB4048,0,0x00000004,“NONE”,0
  GEL_MapAddStr(0xFFFB404C,0,0x000007B4,“W|A|S2”,0
- You may need to modify other GEL_MapAddStr or GEL_MapAdd operations in the GEL file that already defines this area or overlap this area. See the Code Composer Studio on-line Help for further details on Memory Mapping. GEL files, and the GEL_MapAddStr and GEL_MapAdd operations.

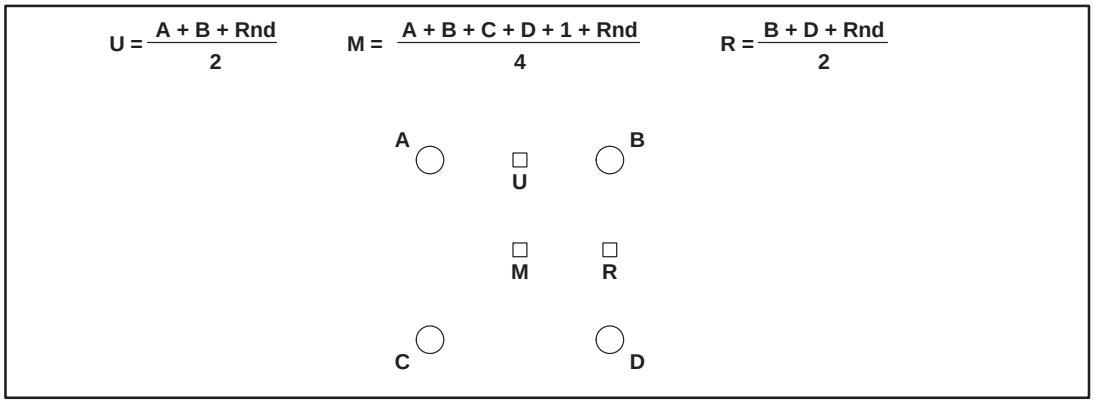
This exception will not be fixed in future silicon revisions.

4.6 DSP Hardware Accelerator Advisories

Advisory DSP_HWA_1	*Pixel Interpolation Hardware Accelerator*

Revision(s) Affected: All Revisions

Details: The pixel interpolation computation is wrong by one pixel unit in “Decoder” mode, when “Rounding Mode” is set to zero and when half-pixel interpolation is performed in the middle of four full-resolution pixels (i.e., in the middle of two rows of pixels).
In “Decoder” mode, the Pixel Interpolator data path is configured to deliver two interpolated pixels per cycle. Each section of the data path implements the computations shown below, normalized according to video decoding standards:



U and R results are interpolated from lines and columns, respectively. M is computed from two following full-resolution pixels lines. Two U, R, or M results are computed during each cycle by the data path, according to the instruction being executed by the accelerator for the M results set (the results are denoted as M0 and M1). When “Rounding Mode” (Rnd in the above picture) is set to 1, the faulty data path section is implemented, M0 = (A+B+C+D+2)/4, which is the correct result; but when “Rounding Mode” is set to 0, the data path section implements M0=(A+B+C+D)/4. This is not correct and generates a bit exactness issue. The M1 result is always correct, regardless of the “Rounding Mode” state.

Workaround: Do not use the PI data path in the “Decoder” software when computing M type points and when software-rounding is needed. Instead, use a routine consisting of regular C55x instruction combinations. This routine must be called from the point where the one using PI HWA instructions is and should take the same parameters and data organization. Hence, the new routine should perform following steps:

- Unpack the pixels in the CPU
- Diagonal interpolation

Pixel Interpolation Hardware Accelerator (Continued)

The routine below describes the M points computation (diagonal interpolation) for a full block:

Presetting:

DR1 is set with the block size (8 or 16 pixels)

AR2 is pointing to the first element in the block

Code example:

Begin:

```
DR2 = DR1 + #2                    ; DR2 = blk_size+2
AR4 = AR2                         ; AR4 -> block[0][0]
AR5 = AR2
DR3 = DR1 - #1                    ; DR3 = block_size - 1
BRC0 = DR3                        ; BRC0 = block_size - 1
AR5 = AR5 + DR2                   ; AR5 -> block[1][0]
DR0 = #2                          ; DR0 = 2
AC3 = DR0 - @rounding_control     ; AC3 = 2-rounding_control
|| DR1 = DR1 >> #1                ; DR1 = block_size/2
@rnd_temp = AC3                   ; rnd_temp = 2-rounding_control
|| DR1 = DR1 - #1                 ; DR1 = block_size/2 - 1
BRC1 = DR1                        ; BRC1 = block_size/2 - 1
XAR3 = XDP
AR3 = AR3 + #rnd_temp             ; AR3 -> rnd_temp

|| localrepeat {                  ; repeat blk_size times

AC0 = (*AR4+ << #16) + (*AR5+ << #16)   ; AC0 = b[i][j]+b[i+1][j]
AC0 = AC0 + (*AR3 << #16)         ; AC0 = AC0 + rnd

|| localrepeat {
AC1 = (*AR4- << #16) + (*AR5+ << #16)   ; AC0 = b[i][j+1]+b[i+1][j+1]
AC0 = AC0 + AC1                   ; AC0 = sum(b[i][j]) + rnd

*(AR4+DR0) = HI(AC0 << #(-2))     ; b[i][j] = (sum(b[i][j])+rnd)/4
|| AC1 = AC1 + (*AR3 << #16)             ; AC1 = AC1 + rnd

AC0 = (*AR4- << #16) + (*AR5+ << #16)   ; AC0 = b[i][j+2]+b[i+1][j+2]
AC1 = AC1 + AC0                         ; AC1 = sum(b[i][j]) + rnd
*(AR4+DR0) = HI(AC1 << #(-2))     ; b[i][j+1] = (sum(b[i][j])+rnd)/4
|| AC0 = AC0 + (*AR3 << #16)             ; AC0 = AC0 + rnd
}

mar (*AR4+) || mar (*AR5+)

}
```

A fix is being considered for this exception in future silicon revisions.

4.7 DSP EMIF Advisories

Advisory DSP_EMIF_1	*DSP IDLE Wakeup With Program Code in External Memory and EMIF Domain in IDLE Hangs*

Revision(s) Affected: All Revisions

Details: The DSP EMIF will not exit IDLE unless it is manually released by setting the corresponding bit in the ICR, then executing the IDLE instruction.

Workaround(s):

1. If the EMIF is IDLEd along with other domains (or by itself), then when an interrupt occurs waking up the other domains, and the CPU continues code execution, the EMIF must be manually removed from IDLE. This is accomplished by writing a 0 to the ICR bits corresponding to all of the domains to be awaken (including the EMIF) and then executing the IDLE instruction.

   Furthermore, ensure that the following are executing from internal memory:

   - The vector table and the wakeup ISR.
     - If the code is awakened by an unmaskable interrupt, it must get the vector address from the vector table, and vector off to the wakeup ISR.
   - IDLE routine
     - In the event that the CPU is awakened by a masked interrupt, the code will continue execution from the point in code that it was IDLEd.

2. Do not IDLE the EMIF

5 MPU Subsystem Advisories

5.1 MPU System Advisories

Advisory MPU_SYS_1	*Access Factor Must Be >1 for API Access to DSP Peripherals*

Revision(s) Affected: All Revisions

Details: A false read may occur for API accesses to DSP peripherals if the API access factor is set to 1 (reset value). The resultant affect is a decrease in performance to the DSP peripherals through the API. The API access factor is controlled by bits [7:4] of the API control register at address 0xFFFE:C900. All DSP peripherals are affected during API accesses. The address range affected is 0xE100:0000 to 0xF101:FFFF.

Workaround: Software needs to write a value of at least 0010 to bits [7:4] of the API Control register.

This exception will not be fixed in future silicon revisions.

5.2 MPU Data-Cache Advisories

Advisory MPU_DCACHE_1	*Data Cache Transparent Mode Restriction During Copy-Back Operation*

Revision(s) Affected: All Revisions

Details: The data cache copy-back mode can only be used when the Transparent bit is disabled (set to 0). When this bit is set to 1, an interleaving of writes and reads will occur during a cache line replacement. For example, write one word of the cache line, read the word that will replace it; write the next word of the cache line, read the word that will replace it; etc. Setting the transparent bit to 0 will cause the writing of the dirty cache line to occur before the new cache line has been read into the cache.

This bit is located in the TI925 Configuration Register in the CP15 coprocessor address space.

Workaround: Do not enable Transparent Mode in the ARM925T Configuration register (it is disabled by default).

This exception will not be fixed in future silicon revisions.

Advisory MPU_DCACHE_2	*Data Cache Entry Operations With VA Depends On CleanCache Mode*

Revision(s) Affected: All Revisions

Details: The ARM925T data cache is 2-way set-associative. When any of the following cache operations are performed:

Data Cache Flush Entry (MCR p15, 0, Rd, c7, c6, 1)
Data Cache Clean and Flush Entry (MCR p15, 0, Rd, c7, c14, 1)
Data Cache Clean (MCR p15, 0, Rd, c7, c10, 2)

only two of two of cache sets are operational.

Workaround: This problem can be completely avoided by ensuring that bit 2 of the CP15 register (TI925T Configuration) is set to 0 before performing the above operations. The TI925T Configuration register is read using the CP15 operation:

MRC p15, 0, Rd, c15, c1, 0

When zero, bit 2 (the C bit) causes the above operations to simultaneously act on both data cache sets. This ensures that the cache flush or clean operation will be successful. The default state of this bit is zero.

This exception will not be fixed in future silicon revisions.

5.3 MPU Instruction Advisories

Advisory MPU_INST_1	*Unpredictable Results After MCR Instructions*

Revision(s) Affected: All Revisions. Applies to all ARM925T devices.

Details: The ARM pipeline causes two previous instructions to already be in the pipeline while the current instruction is being executed. Thus, when the configuration of ARM925T is changed using an MCR the two following instructions should be NOP's to ensure predictable results.

Workaround: Care must be taken when executing instructions that affect the configuration of ARM925T. Add two NOP's after MCR instructions to ensure predictable results.

This exception will not be fixed in future silicon revisions.

5.4 MPU DMA Advisories

Advisory MPU_DMA_3	*DMA Hardware Synchronized Channel is Not Disabled for TIPB Access When DMA's Root Clock is Cut Off*

Revision(s) Affected: All Revisions

Details: If the System DMA's root clock is turned off (by setting ARM_IDLECT2<DMACK_REQ>=0), and the channel's DMA_CCR register is written to disable the channel, the channel will NOT get disabled until the System DMA clock has been restarted and has been running for a minimum of 2 clock cycles. Therefore, when the next DMA HW request occurs, it is possible for the previous transfer to take place instead of a new transfer.

Workaround: Software must keep the DMA root clock running at all times by setting ARM_IDLECT2<DMACK_REQ>=0 and always setting ARM_IDLECT2(DMACK_REQ) bit to 0.

Since the DMA itself has internal automatic clock gating that turns off clocks by itself, whenever there are no active DMA requests (assuming internal auto_clock_gating is ON), there is no benefit in software manipulating the DMACK_REQ bit. Automatic clock gating is controlled by bit 3 in the DMA Global Control Register (DMA_GCR<autogating_on>) which by default is set to enable the DMA clock autogating.

There is no impact to power or performance with this workaround.

This exception will not be fixed in future silicon revisions.

5.5 System DMA Advisories

Advisory SYS_DMA_1	*DMA Clocks Turned Off During Transfers Allows Corruption*

Revision(s) Affected: All Revisions

Details: When bit 8 of the ARM_IDLECT2 register (DMACK_REQ) is set to 1, the root clock to the system DMA is turned off if there is no valid DMA request. However, it takes about 25 cycles to turn off the DMA clocks. During these 25 clock cycles, if a new DMA request is activated, then the clocks to DMA may be turned off in the middle of the new transfer, which could lead to a corruption of data.

Workaround: Always set ARM_IDLECT2(DMACK_REQ) bit to 0. Since the DMA itself has internal auto clock gating which will turn off clocks by itself when there is no active DMA request (assuming AUTOGATING_ON=1 in the DMA_GCR). There is no benefit to software setting the DMACK_REQ bit to “1”. There is no impact to power or performance with this workaround.

This exception will not be corrected in future silicon revisions.

Advisory SYS_DMA_2	*Pending System DMA Request Causes Erroneous Transfer*

Revision(s) Affected: All Revisions

Details: If either the DMA channel is disabled during a transfer or a peripheral makes a DMA request after the DMA has completed a transfer, there will be a pending DMA request. This pending DMA request will cause the DMA to initiate a transfer as soon as the channel is enabled again and could create a problem if the associated peripheral is not ready.

Workaround: This issue can be worked around as follows:

1. Enable the DMA channel
2. Flush out any pending DMA request on this channel
3. Configure the peripheral

This exception will not be corrected in future silicon revisions.

Advisory SYS_DMA_6	*System DMA Potential Deadlock in Burst Accesses*

Revision(s) Affected: All Revisions

Details: If a non LCD System DMA transfer is configured with burst enabled on both source and destination and some of the accessed addresses (notably start address) are not 4x32−bit aligned (i.e. byte address is not multiple of 16) then the System DMA may deadlock and the transfer may never terminate.

Workaround: Two workarounds are available:

1. To use the System DMA for burst access on both source and destination, the start address, block size, frame size, element size and indexes must be set in such a way that all DMA accesses are made on 4x32−bit aligned addresses.
2. If the System DMA is configured for burst access on only one side (source or destination). Burst accesses do not need to be 4x32 bit aligned. On the side that does not use bursting, packing should be used for improved performance.

This exception will not be fixed in future silicon revisions.

6 Traffic Controller Subsystem Advisories

6.1 Traffic Controller (TC) Advisories

Advisory TC_1	*Traffic Controller ELRU Arbitration Does Not Work Properly*

Revision(s) Affected: All Revisions

Details: The ELRU (Enhanced Least Recently Used or consecutive access Least Recently Used) arbitration in the traffic controller (EMIFF, EMIFS, IMIF) does not work properly.

NOTE: All transactions complete correctly with no loss of data, but the order of transactions is incorrect.

When there is a priority change, the consecutive access counter, which keeps track of the number of consecutive accesses performed by a host, does not get cleared. As a result, when a host performs less than the maximum number of consecutive accesses allowed, and the priority changes because the host de-asserted its request, the consecutive access counter, which keeps track of number of consecutive accesses allowed, is not cleared. As a result, the next time the same host gets priority. the counter is not cleared so it will get less than the full number of consecutive accesses it normally should.

NOTE: Only the DMA is capable of issuing multiple consecutive accesses. The ARM can at most give two consecutive accesses. The LB and DSP, always have a dead cycle in−between two consecutive accesses. In this case, the system impact on performance is small.

Workaround: Use LRU/Round Robin arbitration. This is controlled by the OMAP5910 configuration register LRU_SEL. At reset this register is zero and LRU arbitration is selected. By choosing LRU and setting the consecutive access registers of all hosts to zero (reset condition), the priority becomes LRU/Round Robin.

This exception will not be fixed in future silicon revisions.

6.2 EMIF Slow (EMIFS) Advisories

Advisory EMIFS_1 — *Burst Writes in EMIFS Causes Latency of Two TC Clock Cycles Extra From the Second Data Write in the Data Path*

Revision(s) Affected: All Revisions

Details: EMIFS handles burst writes by splitting the transactions into 4 (if the memory connected is 32-bit width) or 8 (if the memory connected is 16-bit width). During split transfers, EMIFS has to follow single write specifications. But from data write onwards, the data comes 2 TC clock cycles later than the expected. This problem exists irrespective of the initiator (ARM, DSP, DMA, or LB).

Workaround: Increase the WELEN bit field of the EMIFS Chip Select Configuration register to compensate for the extra 2 TC clock cycle latency in the data path. The following table shows the requirement for increment of WELEN in case of burst write operation.

FCLKDIV	WELEN (new)	Flash clock
“00”	X + 2	TC_clock/1
“01”	X + 1	TC_clock/2
“10”	X + 1	TC_clock/4
“11”	X + 1	TC_clock/6

This exception will not be fixed in future silicon revisions.

Advisory EMIFS_2 — *WELEN = 0 and FDIV = 1 With 16-Bit Memory*

Revision(s) Affected: All Revisions

Details: When performing asynchronous operations to 16-bit memories with the following bits set: WELEN = 0 and FCLKDIV = 00 (configured in the EMIFS chip select configuration register), EMIFS generates an extra ready for each access to the host. This can cause the host to interpret the extra ready as the ready for the next access, which can put the system in an unknown state.

Workaround: Use one of the following two workarounds:

- Program WELEN in the EMIFS configuration register to greater than 0
- Set FCLKDIV in the EMIFS configuration register to greater than “00”

This exception will not be fixed in future silicon revisions.

Advisory EMIFS_3	*Minimum EMIFS Wait States for Proper Operation*

Revision(s) Affected: All Revisions

Details: Data pipelining in EMIFS is broken during burst writes with the following configuration:

(WRWST < 3 and FCLKDIV = 0).

Workaround: The EMIFS must be setup with a minimum of 3 write wait states (WRWST > 2) for proper operation if FCLKDIV = 0. For most devices used with EMIFS, this is not a performance constraint since the memories available are slower than this speed.

This exception will not be fixed in future silicon revisions.

Advisory EMIFS_4	*FLASH.RDY Does Not Operate Correctly*

Revision(s) Affected: All Revisions

Details: FLASH.RDY is improperly synchronized to the internal FLASH clock. Metastability issues can occur.

Workaround: Do not use FLASH.RDY and tie the pin high with an external pullup. Increase the wait state setting for external memory spaces connected to slow external devices through the wait state bits in the EMIF Slow Chip-Select Configuration Registers (EMIFS_CSx_CONFIG, x=0−3).

This exception will not be fixed in future silicon revisions.

Advisory EMIFS_5	*Extra Flash Clock is Generated With FDIV4 and FDIV6 in Synchronous Mode*

Revision(s) Affected: All Revisions

Details: Whenever EMIFS is configured to FDIV= 4 or 6 in synchronous RDMODE, an extra clock is generated during an access. This extra clock cycle is generated for the transaction between write to write operations and the first read operation after any write operation. Since normally flash memories ignore flash clock during write operations, this isn't a problem. However, an extra flash clock cycle may corrupt memory contents during reads.

Figure 3 shows the problem in detail. It can be observed from the waveform that the first read immediately after the write operation has an extra clock pulse of width 1 TC clock cycle using FDVI = 4. For FDIV = 6, the pulse width is 2 TC clock cycles.

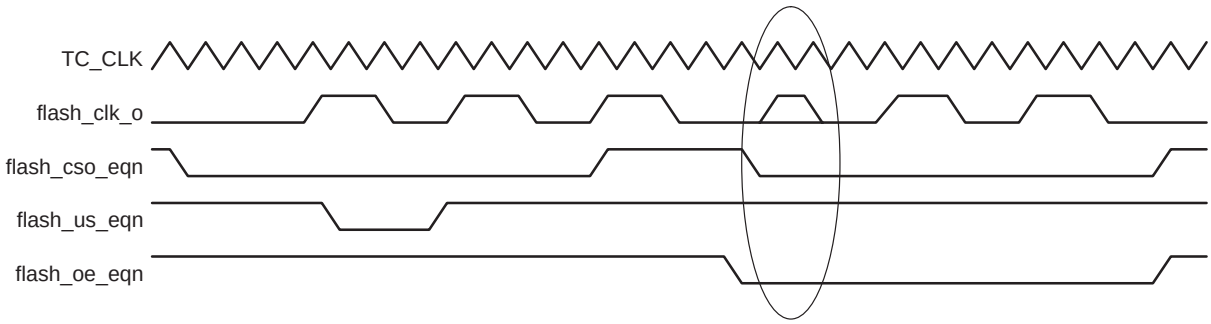


Figure 3. Flash Clock Glitch With FDIV = 4

Figure 4 shows the flash clock with the correct configuration.

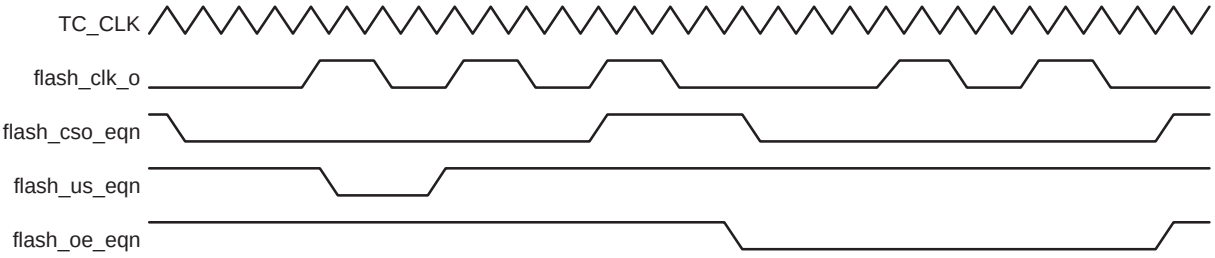


Figure 4. EMIFS Write and Read With Correct Flash Clock SDRAM

Workaround: In order to avoid data corruption, software needs to insert a dummy read/write to a asynchronous chip select.

This exception will not be fixed in future silicon revisions.

6.3 EMIF Fast (EMIFF) Advisories

Advisory EMIFF_1	*EMIFF Configuration Preventing Deep Sleep Entry*

Revision(s) Affected: All Revisions

Details: Due to the implementation of the SDRAM power down feature in the EMIFF, it is possible to have a sequence of events where the SDRAM clock will not properly be disabled making it impossible to enter deep sleep.

The following table details every combination of relevant events which lead to deep sleep entry. The sequence of events is from left to right in the table, i.e., the register bit specified in column one is step1 followed by the register bit in step2, etc. Also, all register bits are assumed to be low before the sequence is started. The Deep Sleep? column indicates whether the device will successfully enter deep sleep mode for the corresponding sequence (OK) or whether it will fail to enter deep sleep (NOT OK).

STEP1	STEP2	STEP3	STEP4	DEEP SLEEP?
EMIFS_PDE	EMIFF_CLK	EMIFF_PWD	SLFR	OK
EMIFS_PDE	EMIFF_CLK	SLFR	EMIFF_PWD	OK
EMIFS_PDE	EMIFF_PWD	EMIFF_CLK	SLFR	OK
EMIFS_PDE	EMIFF_PWD	SLFR	EMIFF_CLK	OK
EMIFS_PDE	SLFR	EMIFF_PWD	EMIFF_CLK	***NOT OK***
EMIFS_PDE	SLFR	EMIFF_CLK	EMIFF_PWD	***NOT OK***
EMIFF_CLK	EMIFS_PDE	EMIFF_PWD	SLFR	OK
EMIFF_CLK	EMIFS_PDE	SLFR	EMIFF_PWD	OK
EMIFF_CLK	EMIFF_PWD	EMIFS_PDE	SLFR	OK
EMIFF_CLK	EMIFF_PWD	SLFR	EMIFS_PDE	OK
EMIFF_CLK	SLFR	EMIFF_PWD	EMIFS_PDE	OK
EMIFF_CLK	SLFR	EMIFS_PDE	EMIFF_PWD	OK
EMIFF_PWD	EMIFS_PDE	EMIFF_CLK	SLFR	OK
EMIFF_PWD	EMIFS_PDE	SLFR	EMIFF_CLK	OK
EMIFF_PWD	EMIFF_CLK	EMIFS_PDE	SLFR	OK
EMIFF_PWD	EMIFF_CLK	SLFR	EMIFS_PDE	OK
EMIFF_PWD	SLFR	EMIFF_CLK	EMIFS_PDE	OK
EMIFF_PWD	SLFR	EMIFS_PDE	EMIFF_CLK	***NOT OK***
SLFR	EMIFS_PDE	EMIFF_CLK	EMIFF_PWD	***NOT OK***
SLFR	EMIFS_PDE	EMIFF_PWD	EMIFF_CLK	***NOT OK***
SLFR	EMIFF_CLK	EMIFS_PDE	EMIFF_PWD	OK
SLFR	EMIFF_CLK	EMIFF_PWD	EMIFS_PDE	OK
SLFR	EMIFF_PWD	EMIFF_CLK	EMIFS_PDE	OK
SLFR	EMIFF_PWD	EMIFS_PDE	EMIFF_CLK	***NOT OK***

EMIFF Configuration Preventing Deep Sleep Entry (Continued)

Important notes:

- The above table was generated with
  - RFRSH_STBY of the EMIFF_SDRAM_CONFIG_2 (bit 0) register = 1, and,
  - PWD_EN of EMIFS_CONFIG_REG (bit 2) register = 1
- EMIFS_PDE = Global Power Down Enable, EMIFS_CONFIG_REG(3)
- EMIFF_CLK = EMIFF sdram clock control, EMIFF_SDRAM_CONFIG(27).
- EMIFF_PWD = EMIFF Power Down Enable, EMIFF_SDRAM_CONFIG(26).
- SLFR = EMIFF Self Refresh Control, EMIFF_SDRAM_CONFIG(0).

Workaround: In order to ensure that deep sleep state is entered correctly and that the clocks are turned off, it is necessary to make sure that the Global Power Down Enable [EMIFS_CONFIG_REG(3)] is set as the last event of the TC Idle entry procedure and it has been toggled from 0 −> 1.

The recommended sequence is:

1. Set EMIFS_PDE = 0
2. Set EMIFF_CLK, SLFR, EMIFF_PWD to 1 (in any order, or all at once)
3. Make sure that IMIF_PWD bit is set to 1.
4. Set EMIFS_PDE = 1

This exception will not be fixed in future silicon revisions.

Advisory EMIFF_2	*EMIFF MRS Request Could Be Lost*

Revision(s) Affected: All Revisions

Details: When a MRS request (sent to configure CAS latency) occurs at the same time as an auto-refresh request, the EMIFF may lose the MRS request. This behavior depends on the timing
between the MRS and auto-refresh request. If the MRS request is lost, then the CAS latency configured internally into traffic controller versus the CAS Latency configured into the SDRAM may be different. This can potentially corrupt the data in the SDRAM.

Workaround: If the application needs to re-configure SDRAM CAS latency after enabling auto-refresh, then software needs to disable the auto-refresh before configuring MRS and then re-enable it after the CAS latency configuration. This procedure is outlined below:

If auto-refresh is enabled (self-refresh is disabled):

1. Disable the auto-refresh by writing 00 in the ARE bit of the EMIFF configuration register (0xFFFECC20).
2. Write the new MRS value to issue an MRS request to the SDRAM.
3. Re-enable the auto-refresh.

If self-refresh is enabled:

4. Self-refresh must be disabled in addition to the above sequence. This is done writing 0 to the SLFR bit of the EMIFF configuration register (0xFFFECC20).
5. DO NOT WRITE a new value into the EMIFF_MRS or make any transfer with the SDRAM within 100 TC clock cycles of step two.
6. Re-enable auto-refresh and self-refresh. If multiple writes to MRS register are needed (EMRS command is the only example that would need such a transaction) then see the Errata named: EMRS command.

This exception will not be fixed in future silicon revisions.

Advisory EMIFF_3

EMIFF MRS Can Cause Read Failure

Revision(s) Affected: All Revisions

Details: A single register is used for MRS and EMRS initialization on the EMIFF SDRAM interface. To initialize these commands, a combination of writes to the EMIFF_MRS write and CONF_MOD_EMRS_CTRL bits is needed as shown below:

COMMAND TO SDRAM	CONF_MOD_EMRS_CTRL	EMIFF_MRS = EMIFF_EMRS = 0xFFFECC24
EMRS	1	write
MRS	0	write

The last value written in the EMIFF_MRS_REGISTER is taken in account for read access with the SDRAM. If the SDRAM MRS value and EMIFF MRS value are different, a read data failure occurs.

Workaround: To generate an EMRS command, follow the procedure below:

1. Configure the CONF_MOD_EMRS_CTRL bit to 1.
2. Write to the EMIFF_EMRS register with the EMRS command.
3. Configure the CONF_MOD_EMRS_CTRL bit to 0.
4. Wait at least 100 TC clock cycles..
5. Write to the EMIFF_MRS register with the MRS value desired.
6. Do not do any transfer with SDRAM or any new EMIFF_MRS write before 100 TC clock cycles.
7. If memory (EMIFF) is in self-refresh, self-refresh must be disabled in addition to the above sequence (Step one). This can be done writing 0 in the SLFR bit field of EMIFF configuration register (FFFECC20).

7 OMAP5910 Peripheral Advisories

7.1 LCD Advisories

Advisory LCD_1	*Missing Palette Loading Interrupt*

Revision(s) Affected: All Revisions

Details: When the LCD peripheral is in palette and data loading mode and the palette loading interrupt occurs, the LCD peripheral sends an interrupt but the status register bit does not get set. This results in the MPU seeing the interrupt, but not being able to identify what caused it. The status register bit is used for checking the status in palette loading mode only . If the user is in palette loading mode, then the operation is correct; the interrupt is sent and the status bits are properly set.

Workaround: Do not use this interrupt when operating in palette and data loading mode. If operating in this mode then this interrupt should be masked.

This exception will not be fixed in future silicon revisions.

7.2 UART Advisories

Advisory UART_1	*Software Flow Control Mode of UART1/2/3*

Revision(s) Affected: All Revisions

Details: When software flow control mode is enabled, the UART (MODEM or IrDA) will compare incoming data with XOFF1 (and/or XOFF2), programmed characters, or XON1/2:

- If a correct XOFF is received, the transmission is halted.
- If a correct XON is received the transmission is restarted.
- If XOFF1 and XOFF2 are used they must be received sequentially in order to halt the transmission.

If parity, framing, or break errors occur when XOFF is received, the transmission should not be stopped and an error should be reported.

There are two issues:

- XOFF with framing error stops the transmission.
- With XOFF1 + word with Parity or break error + XOFF2, the transmission is stopped. It is not correct because XOFF2 does not follow immediately XOFF1. (Note: the sequence XOFF1 + word without error + XOFF2 does not stop the transmission)

Workaround: Use one of the following two workarounds:

- use hardware flow control.
- use software to perform software flow control without hardware assist.

This exception will not be fixed in future silicon revisions.

Advisory UART_2	*UART Clock Request Prevents Deep Sleep*

Revision(s) Affected: Revision C

Details: OMAP5910 was intended to have a feature that if communication is sent on the UART2.RX signal while the device is in deep sleep mode, then the device will automatically wake up from deep sleep. This function does not work properly and consequently, if UART2.RX activity occurs while the device is awake, this will incorrectly create a pending wakeup request from UART2 preventing the device from entering deep sleep mode even when all other conditions required for deep sleep are met.

Workaround: To initiate deep sleep mode properly, it is recommended that UART2 be used for data transmission only and that a different UART (UART1 or UART3) be used to receive serial data. Note that this issue in no way impacts other deep sleep operation and is limited to the constraint outlined in this notice.

Advisory UART_3	*OSC_12M_SEL and EBLR Registers are not Readable*

Revision(s) Affected: All Revisions

Details: Several UART configuration registers are write-only and cannot be read by the MPU or the DSP. These registers are: the OSC_12M_SEL register of UART1, 2, and 3 and EBLR register of UART3.

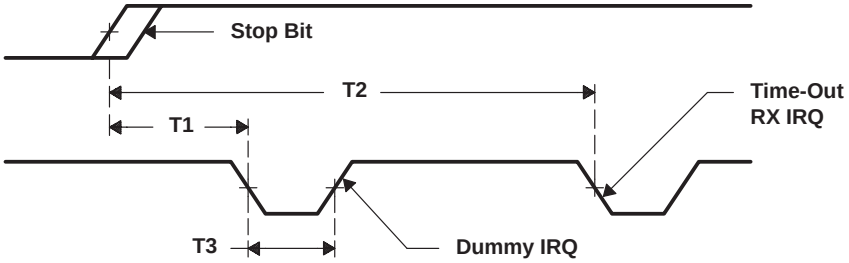
Workaround: Writes to these registers operate correctly but software will not be able to confirm written value with a read. Write errors to the OSC_12M_SEL registers will exhibit themselves as baud-rate inaccuracies.

This exception will not be fixed in future silicon revisions.

Advisory UART_4	*UART1, 2, 3 RX Time-Out IRQ With No Status*

Revision(s) Affected: All Revisions

Details: The UART1, 2, 3 peripherals generate an interrupt without corresponding status when a receive time-out occurs. This time-out is generated when received data is present in the FIFO and is not read (MPU or DMA) during a defined time. See Figure 5.



	BAUD RATE 115200 bps	BAUD RATE 19200 bps
T1	6.8 µs	37.2 µs
T2	380 µs	2.280 ms

	UART1 CLOCK 12 MHz	UART1 CLOCK 48 MHz
T3	83 ns	20.8 ns

Figure 5. UART Receive Time-Out

Workaround: In the interrupt service routine, check bit 0 of the IIR register which signals that no interrupt has been generated by the UART peripheral. Consequently, no associated action is necessary. However, a second receive time-out interrupt occurs which does show the correct status of time-out RX IRQ and thus can be serviced.

This exception will not be fixed in future silicon revisions.

Advisory UART_5	*Incorrect Behavior of UART3 TX_EMPTY_CTL_IT Bit in SIR Mode*

Revision(s) Affected: All Revisions

Details:

The SCR(3) bit TX_EMPTY_CTL_IT is non-functional and cannot be used to generate an interrupt when the TX FIFO and TX shift registers are empty. The status bit LSR(7) THR_EMPTY signals that the TX FIFO is empty. LSR(7) does not indicate that the TX shift register is empty.

When using UART3 in IRDA mode, there is no easy way by hardware to know when the last character has been transmitted after the FIFO is empty. There will still be some bits to transmit (the last character, the EOF, and 2 bytes of CRC16). This can create a problem in the following situations:

1. Before going into IDLE mode, the user has to know when all the characters have been transmitted.
2. Enabling the receiver: When there is an echo signal on the receiver due to the transmitter.
3. Programming a new transmit frame: the transmitted frame is unexpected.

Workaround:

SW can set a timer to track the minimum amount of time needed to send one character.

Twait_min= character_length * Baud clock period
character_length: 5, 6, 7, or 8 bits

Then, when the TX FIFO is empty and gives an interrupt, the SW needs to wait to ensure that the contents of the TX shift register have been emptied before initializing a new operation. The minimum delay is equal to the duration of a 56-bit transmission. The following is an example using 8-bit data and 2 stop bits (worst case):

Twait_min = 56 / baud rate
At 9600 bps, a delay of 5.833 ms is necessary.
At 115200 bps, a delay of 486 µs is necessary.

This exception will not be fixed in future silicon revisions.

Advisory UART_6	*UART3 RX_LAST_BYTE Not Cleared Upon Read in the SIR_LSR Register*

Revision(s) Affected: All Revisions

Details: Bit 5 RX_LAST_BYTE of the SIR_LSR register signals that the last byte of the frame has been received. Contrary to what it was specified in the OMAP5910 TRM documentation, it is not automatically cleared when the corresponding SIR_LSR register is read.

Workaround: In order to reset the “RX_LAST_BYTE” bit of the SIR_LSR register, the following steps have to be taken:

1. First read the last received byte.
2. Then read the SIR_LSR Register.

This exception will not be fixed in future silicon revisions.

Advisory UART_7	*UART1/2/3 Sleep Mode*

Revision(s) Affected: All Revisions

Details: When UARTx is set in the sleep mode, and if a new data is received, this data gets shifted and consequently is corrupted.

When sleep mode is set, the following occurs:

- Bit 4 of the IER register for UART1/2
- Bit 4 of the UART_IER register for UART3 in UART mode
- bit 3 of the MDR1 register for UART3 in SIR mode

UARTx enters sleep mode (that is, the peripheral clock and the baud clock are switched off) after some amount of time, which corresponds to the end of the current activity. A counter (idle_cnt), which has a handshake mechanism with the baud clock, is used to wait for this amount of time. The limit value of this counter depends on the UART configuration:

- In UART mode: limit value = [4 * (word_length) + 12] * 16. Where the word_length = 1 start bit + number of bits in the word + number of stop bit(s) + parity bit (if applicable).
- In SIR mode: limit_value = 52

If new data arrives on the RX line in the same cycle or one cycle before the counter reaches this limit value, the peripheral and baud clock are switched off. Consequently, the start bit is missed and the received data will be shifted.

Note: This problem could only occur if the sleep mode is set.

Workaround: There is no workaround to use the sleep mode.

Advisory UART_8	*FIFO LAST BYTE Interrupt in the UART_IRDA Peripheral is Not Reliable*

Revision(s) Affected: All Revisions

Details: The generation of a RX FIFO last byte interrupt can get postponed until the next subsequent access to the RX FIFO. This can result in an interrupt being generated when reading the next character which is not the true last byte (2nd CRC).

The expected behavior is that the source for this interrupt is the last byte of the frame at the top of the RX FIFO. To reset this interrupt, read the IIR register after reading the RHR register to remove the last byte from the top of the RX FIFO.

Workaround: Two workarounds are available:

1. Do not use the RX_FIFO_LAST_BYTE interrupt, instead use the EOF_IT interrupt and check the frame length with the STS FIFO register (SFREGL and SFREGH).
2. While reading an ISR to determine the type of interrupt, anytime the LSR[0] indicates there is data in the Rx FIFO, the data is processed. But before the data is processed, software needs to check the RX_FIFO_LAST_BYTE status flag. If this is set, then a 500uS wait is exercised to make sure that the newly detected ”last byte” is actually in the FIFO. Then software reads the FIFO until empty.

This exception will not be fixed in future silicon revisions.

7.3 MMC/SD Advisories

Advisory MMC_1 — *MMC/SD Does Not Support Stream Mode Reads*

Revision(s) Affected: All Revisions

Details: The MMC/SD controller does not support stream mode read operations with MMC devices (SD cards do not support stream mode). The command associated with stream mode reads is:

```
CMD11 – READ_DAT_UNTIL_STOP
```

Workaround: Block oriented read commands (class 2) must be used for read operations.

This exception will not be fixed in future silicon revisions.

Advisory MMC_2 — *Stop Transmissions Command Cannot be Sent During Multi-Block Transfer of the MMC/SD Module*

Revision(s) Affected: All Revisions

Details: If the Stop Transmissions command (CMD12) is sent during a multiple block data transfer (read/write), the occurrence of interrupt request Block_RS at the end of the data transfer can put the MMC/SD controller in an unknown state.

Thus, there is a limitation that during multiple block data transfers the stop command must not be sent before all of the blocks programmed in MMC_NBLK register have been sent/received.

Workaround: Software needs to send the CMD12-Stop_Transmission only after the interrupt for the Block_RS has been received at the end of the multiple block data transfer (read/write).

This exception will not be fixed in future silicon revisions.

Advisory MMC_10 — *SPI Mode on the MMC/SD Peripheral is Not Supported*

Revision(s) Affected: All Revisions

Details: The SPI mode on the MMC/SD peripheral is not supported on this device.

Workaround: This exception will not be fixed in future silicon revisions.

Advisory MMC_11	*MMC Exit Busy State With 16 Mbytes Card*

Revision(s) Affected: All Revisions

Details: With 16-MB MMC cards there is no reception of an exit busy state interrupt after a write block (command24) followed by a send status (command13). A command24 followed by command13 also causes a block received/sent and card busy interrupt to be generated.

Workaround: 16-MB MMC/SD cards should not be used. This exception does not occur on larger capacity MMC/SD cards (e.g., 32, 64, 128 MB).

This exception will not be fixed in future silicon revisions.

Advisory MMC_12	MMC/SD Peripheral in SD 4 Wire Mode Gives the Same Performance as SD 1 Wire Mode

Revision(s) Affected: All Revisions

Details: The MMC/SD peripheral in SD 4 wire mode gives the same performance as SD1 wire mode.

Workaround: There is no workaround.

This exception will not be fixed in future silicon revisions.

7.4 MICROWIRE™ Advisories

Advisory UWIRE_1	*Pulldown on the UWIRE.SDI Pin Needs to be Disabled by Software*

Revision(s) Affected: All Revisions

Details: On the UWIRE.SDI pin, there is a pulldown active by default. The inactive level of the signal is high, so the pulldown needs to be disabled in software in OMAP5910 native mode otherwise power is wasted. The pulldown is disabled in compatibility mode.

Workaround: Write 0x0000. to the COMP_MODE_CTRL_0 register (base address: FFFE:1000 + offset 0x0C) to put the part in OMAP5910 native mode.

Before putting the device into OMAP5910 native mode, set CONF_PDEN_WIRE_SDI_R (bit 18) to 1 of the PULL_DWN_CTRL_1 register (MPU byte address FFFE:1044).

This exception will not be fixed in future silicon revisions.

MICROWIRE is a registered trademark of National Semiconductor Corporation.

Advisory UWIRE_2 *MICROWIRE Interface RX Data Failures Possible*

Revision(s) Affected: Revision C

Details: There are RX data failures which occur when specific configurations are used:

Configuration #1:

```
CSx_EDGE_RD=0
CSx_EDGE_WR=1
Delay of 1.5 SCLK cycles between last bit transmitted and first receive bit
captured.
```

Configuration #2:

```
CSx_EDGE_RD=1
CSx_EDGE_WR=0
Delay of 2.5 SCLK cycles between last bit transmitted and first receive bit
captured.
```

In Configuration #1, an extra bit of receive data is captured in the RX register. In Configuration #2, the first bit of RX data will not be captured.

Workaround: Configuration #1: Bit 0 can be ignored by right shifting the RX register value by 1 after reading the data from the RX register. This workaround cannot be used in 16-bit mode since 1 bit of the data is lost due to the extra captured bit. In 16-bit mode, a different configuration must be used.

Configuration #2: No workaround exists. A different configuration must be used.

7.5 I2C Advisories

Advisory I2C_1	*I2C Prescalar Value of 0 Not Supported in Slave Mode*

Revision(s) Affected: All Revisions

Details: The I2C peripheral does not detect a start condition on the I2C Bus™ when the prescalar value is programmed to a divide by 1 value. When the I2C_PSC register stays at its default value of 0 (divide by 1 of the system 12-MHz clock), the Bus Busy (bit 12) bit of the I2C_STAT is not set. Additionally in this case, the I2C peripheral does not recognize the address sent by the master.

Register addresses affected are:

I2C_PSC register = byte address FFFB:380C

BB bit (bit 12) of I2C_STAT register = byte address FFFB:3802

Workaround: If operating in Slave Mode set PSC greater than or equal to 1. If operating in Master Mode above 100 kbps then set PSC to 0.

As the System Clock is a 12-MHz clock (83.3 ns), the internal sampling clock will have a period of 166.6 ns when PSC is set to 1. The I2C performance and functionality is still in conformance to the I2C specification, however, a pulse width less than 166.6 ns (which includes 83.3 ns pulses) will be filtered.

This exception will not be fixed in future silicon revisions.

I2C Bus is a trademark of Philips Electronics N.V. Corporation.

Advisory I2C_2	*Transfer Using Polling of I2C XRDY Bit Fails*

Revision(s) Affected: All Revisions

Details:

The resynchronization between the MPU and the I2C by polling the XRDY bit (bit 4 of the I2C_STAT register) for a data transmission does not work. During transmission, there is a possibility that XRDY is high (I2C_DATA is ready to receive a new data word of two bytes) while there is only one byte empty in the transmitter FIFO. Thus, when the MPU writes two bytes, then one of the bytes in the FIFO is overwritten and lost.

It should be noted, however, that this condition does not affect the transmission when using interrupt mode. A transmit data interrupt (INTCODE = 101b) only occurs when there are two bytes available for the MPU to write into the transmit FIFO. Thus, there is no possibility for the MPU to overwrite data in the transmit FIFO on reception of a transmit data interrupt.

Workaround:

There are two possibilities for workaround:

1. Use interrupt mode transmit data interrupt (INTCODE = 101b).
2. If polling mode is mandatory, then data transmission using polling can be accomplished by the use of the XUDF bit (bit 10 of the I2C_STAT register).

The XUDF bit signals that the transmitter has experienced underflow:

- In master transmit mode, underflow occurs when the transmit shift register (ICXSR) is empty, the transmit FIFO is empty, and there are still bytes to transmit (DCOUNT not equal to 0).
- In the slave transmit mode, underflow occurs when the transmit shift register (ICXSR) is empty, the transmit FIFO is empty, and there are still bytes to transmit (read request from external I2C master).

This exception will not be fixed in future silicon revisions.

Advisory I2C_3 — *Maximum SCL Frequency Configuration in I2C Master Mode With PSC = 1*

Revision(s) Affected: All Revisions

Details: When the I2C is configured in master mode and the PSC is set to 1 (I2C functional clock internally divided by 2), the SCL clock frequency is limited to a minimum setting of 3 (SCLL and SCLH registers setting to 3). This configuration corresponds to a maximum SCL frequency while PSC is set to 1.

Note: The slave mode and the configuration of PSC='0' in master mode are not affected by this issue.

Workaround: In master mode, set the PSC to 0.

However, as PSC = 0 does not work in slave mode, the switch from slave to master or master to slave will require a PSC change. The PSC change requires the following sequence:

- Disable I2C
- Change PSC
- Enable I2C

Note: When the I2C is disabled, it is not possible to observe the status of the I2C system bus.

This exception will not be fixed in future silicon revisions.

Advisory I2C_4 — *Test Mode for Driving SCL Clock Does Not Work on the I2C Peripheral*

Revision(s) Affected: All revisions

Details: In the SCL counter test mode (I2C_SYSTEST[13:12] = 10), the SCL pin should drive a clock according to the parameters in the PSC, SCLL, and SCLH register fields. However, this test mode does not work.

Workaround: Use SDA/SCL I/O mode (I2C_SYSTEST[13:12] = 11). In the SDA/SCL I/O mode, the SCL I/O and SDA I/O are controlled via the I2C_SYSTEST[3:0] register bits.

7.6 USB Function Advisories

Advisory USBF_1	*Read of USB Function Data Register Has a Side-Effect and Should Not be Read From Emulator*

Revision(s) Affected: All Revisions

Details:

Emulation/debug read access (ARM TIPB Suspend active) for USB Function DATA register works only if the end point (the last offset address) is selected. If endpoint is not selected, there is no response from USB Function, and the ARM processor hangs.

Emulation/debug read access (ARM TIPB Suspend active) for USB Function DATA_DMA register works only when the DMA Request is active. If DMA request is not active, there is no response from USB Function, and the system DMA Controller hangs.

The registers affected are:

DATA = byte address FFFB:4008

DATA_DMA = byte address FFFB:4048

No other registers in the USB Function are affected.

Workaround:

Do not read this register from emulator. The Code Composer Studio memory map may be configured so that memory windows will not cause accesses to these registers. Either of these two methods can be used.

Option 1 − use Code Composer Studio “Options” pull down menu’s “Memory Map” window to define the two registers as “protected”. This sequence will need to be performed every time you open Code Composer Studio. Perform the following steps:

```
Select ”Memory Map on the ”Options” menu
Set the Starting Address field to 0xFFFB4008.
Set the Length field to 4.
Set the Attribute field to None – No Memory/Protected.
Set the Access Size field to Auto.
Make sure the Memory Mapping box is checked.
Click the Add Button
Set the Starting Address field to 0xFFFB4048.
Set the Length field to 4.
Set the Attribute field to None – No Memory/Protected.
Set the Access Size field to Auto.
Make sure the Memory Mapping box is checked.
Click the Add Button.
The Memory Map List window should now include the following lines:
0xFFFB4008–0xFFFB400B: NONE
0xFFFB4048–0xFFFB404B: NONE
Click the OK Button. See the Code Composer Studio on-line Help for further
details on Memory Mapping.
```

Read of USB Function Data Register Has a Side-Effect and Should Not be Read From Emulator (Continued)

Option 2 − Modify the Code Composer Studio GEL file that is automatically loaded at Code Composer Studio boot time so that it preinitializes the memory map so that these registers are protected. Add the following lines to the GEL file:

```
GEL_MapAddStr(0xFFFB4000, 0, 0x00000008, "R|W|AS2", 0);
GEL_MapAddStr(0xFFFB4008, 0, 0x00000004, "NONE", 0);
GEL_MapAddStr(0xFFFB400C, 0, 0x0000003C, "R|W|AS2", 0);
GEL_MapAddStr(0xFFFB4048, 0, 0x00000004, "NONE", 0);
GEL_MapAddStr(0xFFFB404C, 0, 0x000007B4, "R|W|AS2", 0);
```

You may need to modify other `GEL_MapAddStr` or `GEL_MapAdd` operations in the GEL file that already define this area or overlap this area.

See the Code Composer Studio on-line Help for further details on Memory Mapping, GEL files, and the `GEL_MapAddStr` and `GEL_MapAdd` operations.

This exception will not be fixed in future silicon revisions.

Advisory USBF_2 — *USB Function Suspend Functionality in HMC_MODE 13 and HMC_MODE 15*

Revision(s) Affected: All Revisions

Details: When in HMC_MODE 13 or HMC_MODE 15, the TLL receives its Suspend and Pullup Enable signals from the USB Function. In its default operating mode, the USB Function will not transition from Suspend to enabled until it senses that it has successfully enabled the Pullup. The TLL prioritizes its Suspend input over its Pullup Enable input, so the TLL will not signal the presence of the pullup, and the USB Function cannot sense that it has enabled the pullup. The USB Function will not exit Suspend mode.

Workaround: Do not use the Transceiverless Link Logic with the USB Function and an external USB Host Controller. Use a transceiver-based solution instead.

This exception will not be fixed in future silicon revisions.

Advisory USBF_3 — *USB Function Double-Buffering Not Supported*

Revision(s) Affected: All Revisions

Details: USB Function Double Buffering does not function properly.

Workaround: Use Single-buffer mode instead of double buffering mode.

This exception will not be fixed in future silicon revisions.

7.7 USB Host Advisories

Advisory USBH_1	*Remote Wake Non-Functional Through TLL in HMC_Mode Settings 9, 10, 11, 12, 14, 21, 23, 24, and 25*

Revision(s) Affected: All Revisions

Details: When in HMC_MODE 9, 10, 11, 12, 14, 21, 23, 24, or 25, the USB Host will not receive Remote Wake from the external USB Function Controller connected to the pins associated with the Transceiverless Link Logic. The external USB Function Controller cannot wake the USB link from USB Suspend using Remote Wake.

This does not affect other connectivity which does not use the Transceiverless Link Logic.

Workaround(s):

1. Systems which do not use USB Remote Wake through the Transceiverless Link Logic may ignore this errata.
2. Use a transceiver-based connection (may require use of a different HMC_MODE or a different set of OMAP5910 pins).
3. Use an OMAP5910 GPIO pin and software monitoring to indicate when software should take the appropriate USB Host Controller port out of USB Suspend state.
4. Avoid putting the USB Host Port which uses the Transceiverless Link Logic into USB Suspend.

This exception will not be fixed in future silicon revisions.

Advisory USBH_2	*Fast Multiple Accesses on EP_NUM Register in USB_W2F Can Cause Failure Following USB Transactions*

Revision(s) Affected: All Revisions

Details: The EP_Sel field in the EP_NUM (address 0xFFFB4004) is used to configure the USB device while running. The EP_NUM field must not be reconfigured within six PVCI clock cycles after being written to, else a failure in an USB transaction can occur. This is the worst-case delay because, up to three hosts can access the register at the same time. They are:

- CPU
- USB itself
- USB DMA State Machine

Each host access takes two PVCI clock cycles, so six PVCI clock periods are needed before any changes can be made. If one of the hosts is not present then four PVCI clock cycles is enough. For example if USB is not working in dma mode, then four PVCI clock cycles will suffice.

NOTE: PVCI clock is the USB peripheral clock. It is the arm_per clock provided by clock manager. It is equal to TC clock divided by arm_per divider in arm clock control.

Workaround: Do not make any other CPU write accesses on EP_NUM Register in the six PVCI clock pulses following modification of the EP_Sel field. The PVCI clock is the arm_per clock.

This exception will not be fixed in future silicon revisions.

7.8 32K Timer Advisories

Advisory TIMER_1	*32K Timer Erroneous Interrupt*

Revision(s) Affected: All Revisions

Details: When using the 32K timer with TVR = 1 or 0 (tick value register), an interrupt will be created as soon as the interrupt is unmasked even though the timer32K is not started. Also, the interrupt will stay asserted (low) until the timer is reloaded. This is not an issue as the interrupt should be set up to be edge-sensitive.

Workaround: Do not use the 32K timer with TVR = 1 or 0.

This exception will not be fixed in future silicon revisions.

Advisory TIMER_2	*Timer32K Reload TRB Bit Does Not Work Correctly*

Revision(s) Affected: All Revisions

Details:

The Timer32K Reload bit (TRB) is used to load the TCR register with the value contained in the TVR register. Once the TCR is loaded, the TRB bit should be automatically reset to '0'. The user should be able to use the TRB bit (by polling) to know when to load a new value in the TVR.

However (see Figure 6), the TRB bit is reset to zero by the falling edge of the 32-kHz clock only after a maximum of 1.5 x 32-kHz clock period (or minimum 0.5 x 32-kHz clock period) after the TRB bit is set. The TCR register is loaded after a maximum of two 32-kHz clock periods (or minimum of 1 x 32-kHz clock periods) on the rising edge of the 32-kHz clock. After the TRB is reset, there is one 32-kHz clock period where a new TRB setting will not be taken into account by the module. If the user programs the TVR and updates the TRB bit within this 32-kHz clock period, the new TVR will not be loaded correctly into the TCR.

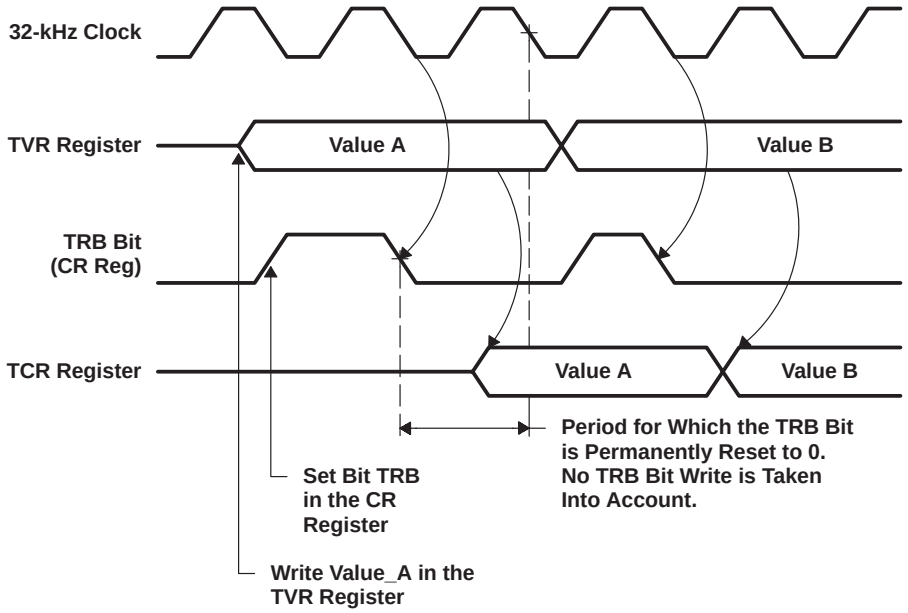


Figure 6. TRB Timing Diagram

Workaround:

To load a new value into the TCR, poll the TRB bit. When the TRB is equal to '0' (signaling that the previous value has been loaded), the user has to wait for an additional one 32-kHz clock period, before loading a new value in the TVR and setting the TRB bit to '1'.

This exception will not be fixed in future silicon revisions.

7.9 MPUIO Advisories

Advisory MPUIO_1	*Latch of MPUIO INPUT_LATCH Register is Disabled During TIPB Read Access*

Revision(s) Affected: All Revisions

Details: A continuous polling on the MPUIO INPUT_LATCH register does not work correctly.

The MPUIO_INPUT_LATCH register has been implemented with a LATCH, this LATCH is:

- Enabled when there is no TIPB read access to the MPUIO INPUT_LATCH register. The register is directly updated by the MPUIO inputs.
- Disabled when there is a TIPB read access to the MPUIO INPUT_LATCH register. The register holds its last value until LATCH is enabled.

Furthermore, in the MPUIO module, the TIPB STROBE is not used to validate a TIPB read access. Consequently, as long as an MPUIO INPUT_LATCH register address is present on the TIPB bus, the corresponding MPUIO INPUT_LATCH register LATCH is selected and therefore disabled (does not take anymore into account of the MPUIO inputs).

Workaround: In order to avoid this during the MPUIO INPUT_LATCH register polling, a dummy TIPB access (the only condition is to do this access to another TIPB address) has to be done just after the MPUIO INPUT_LATCH register read.

This exception will not be fixed in future silicon revisions.

Advisory MPUIO_2	*MPUIO GPIO_INT is no Longer Generated*

Revision(s) Affected: All Revisions

Details: The reset of the MPUIO GPIO Interrupt is done asynchronously when a GPIO_INT register read occurs. The release of this reset is done synchronously with the 32-kHz clock to avoid any reset recovery time violation. A read of the GPIO_INT register while Clock 32k is low or transitioning can result in a deadlock of the state machine that causes all further interrupts on the subject MPUIO pin to be ignored.

Workaround: After receiving the MPUIO GPIO_INT, the MPU should observe the 32-kHz system clock and then read the GPIO_INT register only just after a 0-to-1 transition on the 32-kHz system clock.

There are different ways to detect a 0-to-1 transition on the 32-kHz system clock:

- Polling the timer 32k counter until there is an increment. See the waveforms in Figure 7, which describe the procedure.

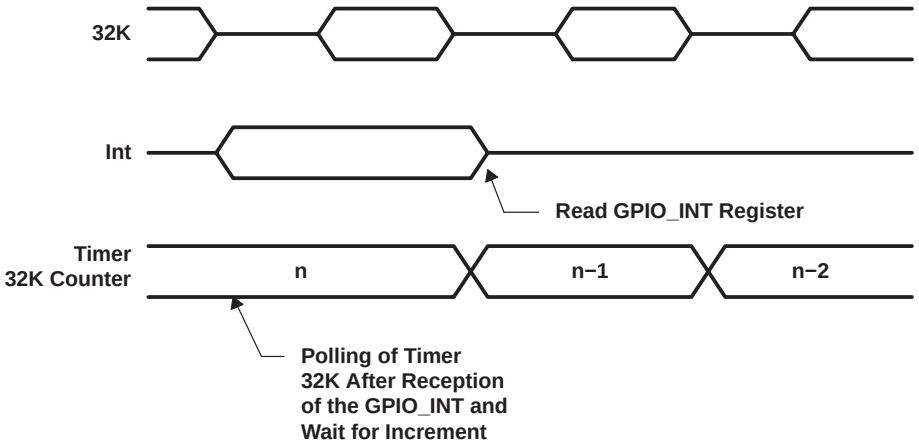


Figure 7. MPUIO Waveforms

- Loop-back the CLK32K_OUT output on one available MPUIO, GPIO, or KB.R input, and polling the corresponding register.

Note that the maximum delay inserted by this workaround, before the interrupt clears, is 31 µs (one 32K clock period).

This exception will not be fixed in future silicon revisions.

Advisory MPUIO_3	*ARM_BOOT Multiplexing When MPU_RESET is Activated*

Revision(s) Affected: All Revisions

Details: The ARM_BOOT pin (In/Out) is multiplexed with the USB_SUSPEND output signal. consequently if the user selects this output signal on the ARM_BOOT pin and then activates the MPU_RESET button, the value which is sampled as the boot mode is used to select witch CS is used to boot (0 $ boot from CS0, 1$ boot from CS3). In this case, it depends directly on the USB_SUSPEND signal and not on the external driven potential. consequently an application could boot from the wrong CS space.

Note that if PWRON_RESET is activated, the problem doesn't exist since the OMAP5910 functional multiplexing is reset by the PWRON_RESET (and not by the MPU_RESET).

Workaround: Connect CS0/CS3 signals to an external AND gate with output to the Flash's CS pin. Thus, the device will always boot out the flash correctly − even if USB1_SUSP is inadvertently muxed to AA17.

This exception will not be fixed in future silicon revisions.

7.10 PWT Advisories

Advisory PWT_1	*PWT Signal Can Stop in its High State*

Revision(s) Affected: All Revisions

Details: The PWT pin (UART3.TX) will sometimes stop in the high state after it is disabled. The PWT is disabled by:

- First, switching off the tone (VCR[15:0]=0x0)
- Then shutting off the PWT clock (GCR[15:0]=0x0)

In a typical application, the PWT pin is connected to the gate of a transistor, which drives a PIEZO speaker. If the PWT pin stops in its high state after being disabled, it can cause (significant) unwanted current across the speaker and consequently increase system power consumption. The high state on the PWT pin persists when OMAP5910 enters Deep Sleep Mode.

Workaround: After disabling the PWT, modify the pin multiplexing of this pin to select configuration mode 0, the UART3.TX pin. When UART3 is disabled, a low level is permanently forced on this pin. The pin-MUXing is controlled by FUNC_MUX_CTRL_6 bits 2:0 (CONF_TX3_R).

This exception will not be fixed in future silicon revisions.

7.11 Camera Interface Advisories

Advisory CMR_1	*Shutdown of Camera Interface*

Revision(s) Affected: All Revisions

Details: The camera interface logic can generate CAM.EXCLK frequencies based on either the 12-MHz system clock or by a 48-MHz clock provided by the DPLL in the ULPD peripheral. The 12-MHz clock is enabled by setting bit 5 (MCLK_EN) of the camera interface CTRLCLOCK register. The 48-MHz clock is enabled either by setting bit 6 (DPLL_EN) or by selecting a clock mode where bit 2 is ‘1’ (this is the MSB of the FOSCMODE field).

To enter Deep Sleep mode, it is necessary to eliminate the camera module requests for both the 12-MHz and 48-MHz clocks. To eliminate the 48-MHz clock request, the 12-MHz clock needs to be running (i.e., these two clocks cannot be stopped simultaneously).

Workaround: To completely shut down the camera module, two separate writes to the CTRLCLOCK register are required.

1. The first write must clear DPLL_EN (bit 6 = ‘0’) and select an FOSCMODE where bit 2 is ‘0’. During this same write, other register bits may also be changed, but MCLK_EN must still be ‘1’. This keeps the 12-MHz clock running.
2. The second write must clear MCLK_EN (bit 5 = ‘0’).

This workaround is not necessary if neither bit 6 nor bit 2 were enabled. There are no other restrictions on these consecutive writes (i.e., they can be consecutive operations).

This exception will not be fixed in future silicon revisions.

7.12 HDQ/1-Wire™ Advisories

Advisory HDQ_1	*1-Wire Interface is Susceptible to Locking-Up if Noise is Seen on the DQ Line During IDLE Time*

Revision(s) Affected: All Revisions

Details: This errata applies only to when the HDQ/1-wire peripheral is operating in the 1-wire mode. The HDQ/1-wire peripheral is susceptible to locking-up, if noise is seen on the DQ line during idle time. Adding additional 1-wire devices to the DQ line while under operation can often provide enough noise to cause the lock-up condition. This condition is characterized by the absence of any further interrupts being sent to the CPU.

Workaround: If this lock-up occurs it can be easily fixed by disabling the peripheral's clock (set bit 5 of the control and status register to 0), and then re-enabling the clock (set same bit 5 to 1). To avoid this during operation, always disable clocks before an idle period and then enable them before transmitting. The master never needs to watch an idle bus since it initiates all activity. Therefore, disabling clocks during idle time is not a problem. For example:

Sequence of Bytes:

Transmit...(Disable 1-wire clock)....IDLE...(Enable 1-wire clock)... Transmit...

NOTE: No register access must be performed to the module registers after software puts the module in power-down mode (by setting bit 5 of the control and status register to 0) other than a write to the power-down bit to take it out of power-down mode.

This exception will not be fixed in future silicon revisions.

1-Wire is a trademark of Dallas Semiconductor Corporation.

8 OMAP5910 Device/System-Level Advisories

8.1 System Advisories

Advisory SYS_1	*Timeout Abort on a Posted-Write Access in the TIPB Bridge*

Revision(s) Affected: All Revisions

Details: An issue has been detected when **posted-write** is enabled in the TIPB Bridge, and a **posted-write** transaction causes a timeout abort. In this case the TIPB Bridge wrongly generates a data strobe to the ARM, which could cause the ARM to hang.

Workaround: Disable the **posted-write** in the both the Public and Private TIPB Bridges. By default (coming out of RESET), the **posted-write** is disabled. The posted-write feature can be disabled by setting *ARM_TIPB_CNTL[1:0] = "00".*

Advisory SYS_2	*Write Followed by Immediate Read Not Supported on Specific Addresses (TIPB Switch and PWT Module)*

Revision(s) Affected: All Revisions

Details: A write followed by an immediate read does not work on the ARM address space defined below. If a read occurs immediately after the write to the same address, then the read will return incorrect data (not the data that was just written).

The affected address spaces are:

TIPB Switch module: Address **FFFB:C800 to End Address FFFB:CFFF**

PWT module: Address space **FFFB:6000 to End Address FFFB:67FF**

Workaround: When an immediate read is required after a write to any register in the above address space, the workaround is to make two consecutive writes to the same address prior to the read. By this procedure, it ensures that the first write must complete and the read data will be correct.

This exception will not be fixed in future silicon revisions.

Advisory SYS_3	*Impact on $I_{DDC(0)}$ Current if DSP Held in Reset Without Proper Initialization*

Revision(s) Affected: All Revisions

Details: Due to the synchronous implementation of resets within the DSP subsystem, if the DSP is simply held in reset, it will not be in its lowest power state. The DSP must be properly initialized to enter its lowest power state.

Workaround: The OMAP5910 DSP subsystem can go properly into a low current consumption state in any one of three different ways:

- Method 1: use the program in DSP PDROM to put the DSP into idle
- Method 2: toggle the DSP reset (DSP_EN of ARM_RSCT1): 0 then 1 then 0 again.
- Method 3: download a program into the DSP that puts the DSP into idle

All of these must be done with DSP clocks enabled.

Method 1: Use the program code in DSP PDROM to put the DSP into idle.

Enable DSP clock
Enable MPUI clock
Release DSP Interface Reset (Bit DSP_RST of ARM_RSTCT1)
Set up the MPUI
Set up the MPUI boot to the value 0x2 (API_DSPBootConfig(API_DSP_BOOT_IDLE))
Set API_SIZE_REGISTER register to 0 (Make SARAM inaccessible by MPU to allow the DSP to go into idle)
Release DSP reset (Bit DSP_EN of ARM_RSCT1)
The DSP boots from the PDROM and executes code that put itself in IDLE
Set all the IDLE bits of ARM_IDLECT1

Method 2: Toggle the DSP reset.

Cut DSP clock
Set all the IDLE bits of ARM_IDLECT1

Method 3: Download a program into the DSP that puts the DSP into idle.

Turn on DSP clock
Turn on API clock
Release DSP Interface Reset (Bit DSP_RST of ARM_RSTCT1)
Set up the MPUI
Set up the MPUI boot to the value 0x5
(API_DSPBootConfig(API_DSP_BOOT_INTERNAL))
Write DSP code into DSP SARAM0
Set API_SIZE_REGISTER register to 0 (Make SARAM inaccessible by MPU to allow the DSP to go into IDLE)
Assert DSP reset (Bit DSP_EN of ARM_RSCT1)
Release DSP reset (Bit DSP_EN of ARM_RSCT1)
The DSP boots from SARAM0 and executes code that puts itself in idle
Set all the IDLE bits of ARM_IDLECT1

This exception will not be fixed in future silicon revisions.

Advisory SYS_5	*ARM BOOT Multiplexing When mpu_n_reset is Activated*

Revision(s) Affected: All Revisions

Details: The ARM_BOOT pin is multiplexed with the USB_SUSPEND output signal. Consequently, if the user selects this output signal on the ARM_BOOT pin and then asserts the MPU_RESET signal, the value which is sampled as the boot mode is used to select which CS is used to boot from (0 to boot from CS0, 1 to boot from CS3). This results in the in boot selection by being chosen by the USB_SUSPEND signal and not the ARM_BOOT pin. Consequently the user application could boot from the wrong CS space.
Note that if the PWRON_RESET is activated, the problem doesn't exist since the OMAP5910 functional multiplexing is reset by the PWRON_RESET and not by the MPU_RESET.

Workaround: Connect CS0/CS3 signals to an external AND gate with the output connected to the Flash's CS pin. This will force the device to always boot out of flash correctly.

This exception will not be fixed in future revisions.

Advisory SYS_6	*OMAP Cannot Go Into Chip_IDLE When the Internal Auto_Clock_Gating of the System DMA is Turned OFF*

Revision(s) Affected: All Revisions

Details: When the internal auto_clock_gating of the system DMA is turned OFF, OMAP cannot go into Chip IDLE because DMA clocks get turned OFF.

Workaround: The internal auto_clock_gating of system DMA has to be ON, and this can be done by setting DMA_GCR(Auto_gating_on) bit (bit 3) to HIGH. This is the lowest power consumption mode of operation and is therefore the desirable state in any case.

This exception will not be fixed in future silicon revisions.

9 Documentation Support

For device-specific data sheets and related documentation, visit the TI web site at: **http://www.ti.com**

For further information regarding the OMAP5910, please refer to:

- *OMAP5910 Dual-Core Processor Data Manual* (literature number SPRS197)
- *OMAP5910 Dual-Core Processor Functional and Peripheral Overview Reference Guide* (literature number SPRU602)
- *OMAP5910 Dual-Core Processor System DMA Controller Reference Guide* (literature number SPRU674)

For additional information, see the latest versions of:

- *TMS320C55x™ DSP Functional Overview* (literature number SPRU312)
- *TMS320C55x DSP CPU Reference Guide* (literature number SPRU371)
- *TMS320C55x DSP Mnemonic Instruction Set Reference Guide* (literature number SPRU374)
- *TMS320C55x DSP Algebraic Instruction Set Reference Guide* (literature number SPRU375)
- *TMS320C55x DSP CPU Programmer's Reference Supplement* (literature number SPRU652)

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

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

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

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

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
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
		Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

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