OMAP5910

Dual-Core Processor

Silicon Errata

SPRZ016
August 2002

Texas Instruments

Contents

1 Introduction

This document describes the silicon updates to the functional specifications for the OMAP5910, silicon Revision C.

1.1 Quality and Reliability Conditions

TMX Definition

Texas Instruments (TI) does not warranty either (1) electrical performance to specification, or (2) product reliability for products classified as TMX. By definition, the product has not completed data sheet verification or reliability performance qualification according to TI Quality Systems Specifications.

The mere fact that a TMX device was tested over a particular temperature and voltage ranges should not, in any way, be construed as a warranty of performance.

TMP Definition

TI does not warranty product reliability for products classified as TMP. By definition, the product has not completed reliability performance qualification according to TI Quality Systems Specifications; however, products are tested to a published electrical and mechanical specification.

TMS Definition

Fully-qualified production device

1.2 Revision Identification

To designate the stages in the product development cycle, TI assigns prefixes to the part numbers of all OMAP devices 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 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

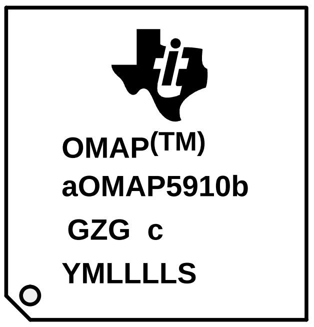
X and P devices and TMDX development-support tools are shipped against the following disclaimer:

“Developmental products intended for 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.

The device revision can be determined by the symbols marked on the top of the GZG package as shown in Figure 1. Some prototype devices may have markings different from those shown in Figure 1 with the device name in the following format: aOMAP5910bGZGc where a = product level, b = the revision number, and c = an internal TI designator (has no meaning to customer).

Figure 1. Example Markings for OMAP5910 GZG Package, Revision C



NOTE: Qualified devices are marked with no prefix at the beginning of the device name, while nonqualified devices are marked with the letter X at the beginning of the device name.

2 Important Notices and Information About OMAP5910

2.1 Useful Information Regarding C55x™ 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

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 DSP Subsystem Advisories

The “CPU_x” portion of the exception numbers correspond to the TMS320C5510 DSP CPU exception numberings because the C55x assembler generates remarks, warnings, and error messages that correspond to the 5510 exception numbers.

3.1 DSP Processor Core Advisories

Advisory DSP_CPU_73	*Certain Instructions Not Pipeline-Protected From Resets*

Revision(s) Affected: Revision C

Details: In the following cases, instructions may not execute properly due to insufficient pipeline protection from reset conditions:

Case 1

The following instruction(s) is not executed properly when closely preceded by a hardware or software reset:

```
DP = #K16    ;OR
DAGEN operation affected by any status bit     ;OR
if (cond) execute (AD Unit)
```

These instructions (which depend on ST0_55, ST1_55 and ST2_55) will not execute correctly if they are located in the first four instructions following the reset (including the delay slot in the reset vector).

Case 2

IFR0/1 or ST1 MMR read instructions may return invalid read data when followed by a software reset.

Case 3

The BRAF bit is not cleared correctly by a software reset which follows the bit (ST1, #BRAF) = #1 instruction.

Assembler Notification: None

Workaround: Use the appropriate workaround, based on the Case. This exception will not be fixed in future silicon revisions.

Case 1

Do not put the following instruction(s) in the delay slot (last four bytes after the interrupt vector). Also do not use the following instruction(s) as the first, second, or third instructions at beginning of program space:

```
DP = #K16    ;OR
DAGEN-operation affected by any status bit     ;OR
if (cond) execute (AD Unit)
```

Case 2

Ensure at least 3 cycles between IFR0/1 or ST1 MMR read and a software reset.

Case 3

Ensure at least 5 cycles between bit(ST1, #BRAF) = #1 and a software reset

Advisory DSP_CPU_76

DELAY Smem Does Not Work With Circular Addressing

Revision(s) Affected: Revision C

Details: When using circular addressing mode with the ‘DELAY Smem’ instruction in the following case:

smem = (end address of a circular buffer)

the incorrect destination address is used for the delay instruction. The destination address used is (end of circular buffer)+1, which is outside of the circular buffer. The correct functionality would be for the destination address to wrap around to the beginning address of the circular buffer.

Assembler Notification: Assembler (version 2.3 and later) will detect the use of delay (Smem) and generate a REMARK.

Workaround: Do not use circular addressing mode with the ‘DELAY’ instruction. This exception will not be fixed in future silicon revisions.

Advisory DSP_CPU_82

‘if (cond true) goto’ at the End of Local Repeat Fails

Revision(s) Affected: Revision C

Details: Within any local repeat block if a conditional branch instruction is placed at the second to last position, and the branch target is at the last position of the loop, the program flow is corrupted. This is the case regardless of whether the local repeat is the outer loop or a nested inner loop.

Mnemonic example

```
localrepeat{
.
.
.
if (cond true) goto TARGET

TARGET
nop
}
.
```

Assembler Notification: Problem is always avoided. The complier does not generate the problem sequence.

Workaround: Do not use this instruction sequence. Incidentally, this sequence would be very impractical in actual application code. This exception will not be fixed in future silicon revisions.

Advisory DSP_CPU_83

BRAF Updated Incorrectly in Certain Cases of Conditional Execution

Revision(s) Affected: Revision C

Details: When C54CM=1 and one of the following cases occurs, the BRAF bit is modified regardless of the condition.

- if(cond=false)Execute(D_unit) || bit(ST1, @BRAF) = #0/1
- while(cond=false && (RPTC < k8)) bit(ST1, @BRAF) = #0/1

Assembler Notification: Assembler (version 2.3 and greater) will attempt to detect the cases above and generate a WARNING.

Workaround: Use one of the following workarounds. This exception will not be fixed in future silicon revisions.

1. Use the AD-unit instead of the D-unit in the conditional execution instruction
   OR
   Do not use parallelism (use conditional execute of next instruction as opposed to conditional execute of parallel instruction).
2. Do not use a bit instruction that modifies BRAF within the WHILE instruction

Advisory DSP_CPU_84 — *SPI/SSP Access Followed by a Conditional Execute is not Protected Against Interrupts*

Revision(s) Affected: Revision C

Details: Any of the following instructions are not protected against interrupts when followed by a AD-unit conditional execute instruction for which the condition is false. (This exception only applies to conditional execution of the next instruction and not a conditional execute of a parallel instruction):

?? SP = SP + k8 (revision 1.x only)
?? MMR-read access to SP/SSP
?? dst = XSP/XSSP
?? dbl(Lmem) = XSP/XSSP
?? push_both(XSP/XSSP)
?? XSP/XSSP = pop()
?? MMR-write access to SP/SSP

Algebraic example

```
...{
nop
SP = SP - #1
.if (TC1) execute (SD_Unit)  ;where TC1=0, condition is false.
<interrupt occurs>
AR6 –= #1
...
```

Assembler Notification: Assembler (version 2.3 and greater) will attempt to identify a code sequence that may cause the Exception, and will generate a REMARK.

Assembler Notification: This exception is avoided in compiler version 2.04 when the v5510.2 switch is used.

Workaround: Use one of the following workarounds. This exception will not be fixed in future silicon revisions.

1. When SP/SSP is read in the read phase, insert two (2) NOPs between the SP/SSP instruction and the conditional execute instruction.
2. When SP/SSP is read or written in the execute phase, insert three (3) NOPs between the SP/SSP instruction and the conditional execute instruction.
3. When SP/SSP is written in the write phase, insert four (4) NOPs between the SP/SSP instruction and the conditional execute instruction.

Advisory DSP_CPU_85

Local Repeat with C54CM = 1 may be Corrupted on its Last Iteration

Revision(s) Affected: Revision C

Details: Under the following conditions during a local repeat loop:

- C54CM = 1
- The program fetch is occurring to restart the last iteration of the local repeat loop
- The program fetch is occurring to stalled

The local repeat body may be overwritten even though the last iteration has not been completed.

Assembler Notification: Assembler (version 2.3 and greater) will generate a WARNING when a .C54CM_ON directive is seen and a local repeat is encountered.

Workaround: Do not use local repeat loops with C54CM = 1. This exception will not be fixed in future silicon revisions.

Advisory DSP_CPU_86

Corruption of CSR or BCRx Register Read When Executed in Parallel With Write

Revision(s) Affected: Revision C

Details: Under the following conditions:

- CSR, BRC0, or BRC1 register is read in the EXE phase in parallel with a write to the same register
- The instruction is stalled due to a previous write access

The register read may be corrupted, returning the new value from the register write instruction. The possible parallel instruction pairs which may cause this condition are as follows:

```
Smem  =  CSR    ||  CSR = TAx     ;Smem should be updated by old register value, but
Smem  =  CSR    ||  CSR = Smem    ;updated to TAx value instead

Smem  =  BRC0   ||  BRC0 = TAx
Smem  =  BRC0   ||  BRC0 = Smem
TAx   =  BRC0   ||  BRC0 = TAx
TAx   =  BRC0   ||  BRC0 = Smem

Smem  =  BRC1   ||  BRC1 = TAx
Smem  =  BRC1   ||  BRC1 = Smem
TAx   =  BRC1   ||  BRC1 = TAx
TAx   =  BRC1   ||  BRC1 = Smem
```

Assembler Notification: Assembler (version 2.3 and greater) will detect the above parallel pairs and generate a WARNING.

Assembler Notification: This exception is always avoided by the complier since the compiler currently does ot generate any instruction that reads CSR or BRCx in the execution stage of the pipline.

Workaround: Do not execute these instructions in parallel. This exception will not be fixed in future silicon revisions.

Advisory DSP_CPU_87 — *Context Restore Just Before Return Instruction Sometimes Fail*

Revision(s) Affected: Revision C

Details: A context restore just before the return instruction sometimes fails. There are two cases in which this condition may occur:

Case 1: When the C54CM bit in ST1_55 is updated via MMR write just before the return instruction, a failure may occur. In the following sequence:

```
*(ST1_55) = <value>
return
```

the new value of the C54CM bit is not used by the return instruction. This may eventually lead to a BRAF recovery error. When C54CM=1, BRAF is not recovered by return. When C54CM=0, BRAF is recovered.

This failure occurs under the following conditions:

- C54CM bit is modified by ST1_55 context restore, AND
- the return condition is either ‘return’ with slow-return configuration, OR, ‘if() return’ with fast or slow return configuration.

Case 2: Altering the BRAF bit just before ‘return_int’ instruction. In the following sequence:

```
C54CM = #1
...
any BRAF update
return_int
```

In the fast-return configuration, BRAF is recovered immediately after return_int is decoded (along with return address). Due to lack of pipeline protection, the BRAF contents recovered by ‘return_int’ is overwritten by the instruction preceding ‘return_int’..

This failure occurs under the following conditions:

- C54CM = 1, AND
- the return condition is either ‘return’ with fast-return configuration.

Assembler Notification: Assembler (version 2.3 and greater) will generate a REMARK when it detects the above instruction sequences.

Workaround: Use one of the following workarounds. This exception will not be fixed in future silicon revisions.

Case 1: Insert at least one NOP between the MMR access and the return instruction.

Case 2: Do not recover the BRAF context with an instruction that accesses BRAF. Instead, let return recover the BRAF content.

Advisory DSP_CPU_88 — *Incorrect Context Store of BRAF During Interrupt Servicing*

Revision(s) Affected: Revision C

Details: When an interrupt is serviced while a blockrepeat loop is active, the context pushed onto the stack incorrectly stores the BRAF bits as 0. Upon returning from the interrupt service routine, the CPU acts as if no loop is active. The program execution will continue sequentially past the end of the active loop. in other words, the blockrepeat loop is not re-activated upon return from an interrupt.

This condition occurs when the second instruction of a parallel instruction pair is a call (only call L16 is legal for such an instruction pair). The condition can occur when these parallel instructions are placed before the loop as well as within the loop.

Algebraic example

```
...
<instruction 1> || call L16
...
blockrepeat{
...
<interrupt decoded>
...                      ; upon return from interrupt, loop becomes inactive.
}
```

OR

```
...
blockrepeat{
...
<instruction 1> || call L16
...
<interrupt decoded>
...                      ; upon return from interrupt, loop becomes inactive.
}
```

Assembler Notification: Assembler (version 2.3 and greater) will detect any instruction with a parallel call L16 and generate a REMARK.

Assembler Notification: This exception is avoided in compiler version 2.04 when the v5510.2 switch is used.

Workaround: Since interrupts are asynchronous, the only workaround is NOT to utilize the following parallel instruction pair.

```
<instruction 1> || call L16
```

This exception will not be fixed in future silicon revisions.

Advisory DSP_CPU_89

Internal Overflow Not Detected When Using the Left Shift Command

Revision(s) Affected: Revision C

Details: In native 55x mode (C54CM=0) when performing left shifts using 32-bit computational mode (M40=0) with the sign extension mode bit set to UNSIGNED (SXMD=0), any overflow in ACx should result in a saturate 40-bit value of 0x 00 7FFFF FFFF. However, if ACx[39..32] = 0xFF and a left shift occurs with the shift value ≥ 0x8, then ACx gets zeroed.

Two instructions could cause this failure to occur as they both perform a store of a saturate of a shifted value all in the same instruction.

```
Smem = HI(saturate(uns(rnd(ACx << DRx))))
Smem = HI(saturate(uns(rnd(ACx << SHIFTW))))
```

If guard bits of the accumulator are)xFF, bit 31 is 0, and the result of the sifft is such that bits 39:31 are zero, the value is not recognized as an overflow, and thus is not saturated.

Example

```
SXMD = #0
M40 = #0

AC2 = FF 0000 0000h
DR1 = 0x0008h
TARGET*AR4 = HI(saturate(AC2 << DR1))

; *AR4 = 0x0000

; Expected value should be *AR4 = 0x7FFF.
```

Assembler Notification: Pending

Workaround: The user should avoid these compressed insturction forms. Software may either:

1. Perform the shift is one instruction followed by the saturate-and-store command.
2. Perform the shift-and-saturate in one instruction followed by the store.

This exception will not be fixed in future silicon revisions.

Advisory DSP_CPU_91

C16, XF, and HM Bits Not Reinitialized by Software Reset

Revision(s) Affected: Revision C

Details: According to the specification, the software reset only affects (IFR0/1, STO_55, ST1_55, and ST2_55. In this case, the reset value should be the same as those forced by a hardware reset (C16=0, HM=0, XF=1). Instead, the software reset does not affect the C16, XF, and HM bits and they retain their previous values.

Assembler Notification: None

Workaround: After a RESET, hardware of software initializes these bits as follows:

```
C16 = 0
HM = 0
XF = 1
```

This will correct the problem for a software RESET and will have no effect on a hardware RESET since these bits would already be set to the specified state. This exception will not be fixed in future silicon revisions.

Advisory DSP_CPU_94

Interrupted Conditional Execution After Long Memory-Mapped Register Write is Executed Unconditionally in the D Unit / AD Unit

Revision(s) Affected: Revision C

Details: When a long memory-mapped register (MMR)† write instruction is executed just before or during a conditional statement in the D unit / AD unit and:

- an interrupt is asserted between the conditional execute and the next instruction to be executed
- no single MMR write follows before or during the return from interrupt

then, the instruction to be executed based on the conditional gets executed regardless of the conditional's value as shown in the following examples.

Example 1

```
       long MMR Write
       .                                               ;No single MMR write
       If (Conditional) Execute (AD Unit / D Unit)    ;No single MMR write
       <Hardware Interrupt Asserted>
       Instruction to be executed based on the Conditional gets executed
       regardless of the Conditional
       .
       .
ISR    .                                               ;No single MMR write
       .                                               ;No single MMR write
       .Return-int                                     ;No single MMR write
```

† Long memory mapped register (MMR): Any of the following instructions that point to 0x0 – 0x5F with "Lmem". Such as:

dbl(Lmem) = pop()
dbl(Lmem) = ACx, copr()
dbl(Lmem) = LCRPV
dbl(Lmem) = src
dbl(Lmem) = ACx
dbl(Lmem) = saturate(uns(ACx))
Lmem = pair(DAx)
HI(Lmem) = HI(ACx) >> #1, LO(Lmem) = LO(ACx) >> #1
Lmem = pair(HI(ACx))
Lmem = pair(LO(ACx))
Lmem = dbl(coeff)

Interrupted Conditional Execution After Long Memory-Mapped Register Write is Executed Unconditionally in the D Unit / AD Unit (Continued)

Example 2

```
       If (Conditional) Execute (AD Unit / D Unit)    ;No single MMR write
       <Hardware Interrupt Asserted>
       Instruction to be executed based on the Conditional gets executed
       regardless of the Conditional
       .
       .
ISR    .
       .
       long MMR write
       .                                              ;No single MMR write
       .Return-int                                    ;No single MMR write
```

Example 3

```
       If (Conditional) Execute (AD Unit / D Unit)    ;No single MMR write
       <Hardware Interrupt Asserted>
       Instruction to be executed based on the Conditional gets executed
       regardless of the Conditional
       .
       .
ISR    .
       .
       long MMR write ∥ Return_int
```

Assembler Notification: None

Workaround: Put a dummy single memory write (i.e., @#0x1F = AR0 ∥ mmap() : 0x1F is a reserved space.) in front of all “Return_int” and ensure that no long memory writes are in parallel with a “Return_int.”

Example

```
       long MMR Write
       .                                              ;No single MMR write
       If (Conditional) Execute (AD Unit / D Unit)    ;No single MMR write
       <Hardware Interrupt Asserted>
       Instruction to be executed
       .
       .
ISR    .                                              ;No single MMR write
       .                                              ;No single MMR write
       Single MMR write
       .Return-int                                    ;No single MMR write
```

This exception will not be fixed in future silicon revisions.

Advisory DSP_CPU_95

BRCx Decement May Not Work When GotoP24 Is Put at End of Blockrepeat With C54CM = 0

Revision(s) Affected: Revision C

Details: When a branch, such as gotoP24, is performed at the end of blockrepeat with C54CM =0, then the corresponding BRCx may not get decremented. This bug occurs in both outer and inner blockrepeats. See the following example.

Example

```
             RC = x
             lockrepeat{

             goto tgt  ; assembled to gotoP24

             .       .
      tgt:   BRC == x or (x–1) ?
```

NOTE: If the destination of the goto is within a 16-bit range i.e. gotoL16 is assigned, this problem does not occur.

Assembler Notification: Pending

Workaround: Do not put a goto instruction at the end of a blockrepeat. This exception will not be fixed in future silicon revisions.

Advisory DSP_CPU_96

BRCx Decement May Not Work When GotoP24 Is Put at End of Blockrepeat

Revision(s) Affected: Revision C

Details: When a branch, such as gotoP24, occurs within a blockrepeat with C54CM =0 and its target is within the same loop, the loop ends immediately. If a nested loop starts after the branch, it is handled as a non–nested one with the 1st–level (RSA0/REA0 utilized, BRC0 decremented). See the following examples.

Example 1

```
                blockrepeat{
                  .
                  goto tgt
                  .
        tgt:      .
                  .
                }              ; Exit from the loop regardless BRCx value.
```

Example 2

```
                blockrepeat{
                  .
                  goto tgt
                  .
        tgt:      blockrepeat { ; Regarded as outer loop, use of RSA0/REA0/BRC0
                        .
                       }
                  .
                }
```

NOTE: If the destination of the goto is within a 16-bit range, i.e. gotoL16 is used, this bug does not occur. This implies that the size of the blockrepeat must be greater than 0x8000.

Assembler Notification: Pending

Workaround: Do not put a goto instruction, in which the target is within the same loop, in a blockrepeat which is greater than 0x8000 in size. This exception will not be fixed in future silicon revisions.

Advisory DSP_CPU_97	*LCRPC – Lmem ‖ Lmem = LCRPC May Not Work*

Revision(s) Affected: Revision C

Details: LCRPC = Lmem **||** Lmem = LCRPC can be used to swap the data between Lmem and LCRPC as shown below:

New LCRPC <– Old Lmem

Old LCRPC –> New Lmem

However, when this store operation is stalled during a parallel execution, the content of the old LCRPC is lost as shown below:

New LCRPC <– Old Lmem

New LCRPC –> New Lmem

Example

Before execution : LCRPC is 0x00123456, Lmem is 0xffffffff

After execution : LCRPC is 0xffffffff, Lmem is 0xffffffff (Should be 0x00123456)

Assembler Notification: Pending

Workaround: Do not use this parallel execution. This exception will not be fixed in future silicon revisions.

Advisory DSP_CPU_98

BANZ at the End of Inner Loop in Native Mode May Corrupt Program Flow

Revision(s) Affected: Revision C

Details: When all of the following conditions are met:

– C54CM=0 (Native mode),

– Two blockrepeats (not localrepeat) are nested,

– the instruction at very end of inner loop is BANZ with a false condition,

– the size of inner loop is less than 32 byte.

– the distance between the end of the two loops is greater than 0 and less than 24byte.

The program flow may be corrupted. The instruction immediately after the inner loop, although outside of the inner loop, gets executed during first iteration of the inner loop. See the example below.

Example

```
”INST–A” is executed at the first iteration of the inner loop.
bit(ST1,@C54CM) = #0
blockrepeat{
  .
  blockrepeat{
    .                                    Less than 32 bytes
    .
    BANZ with false condition
  }
  INST–A
  .                                      Greater 0 and less than 24 bytes
  }
```

Assembler Notification: Pending

Workaround: Put a NOP immediately after the BANZ within the inner loop. This exception will not be fixed in future silicon revisions.

Advisory DSP_CPU_99

Return_int (Under a Fast – Return Configuration) May Cause Improper Operation of Single Repeats and Conditional Executions

Revision(s) Affected: Revision C

Details: Under a fast return configuration, when an interrupt is asserted during any of the following:

- Single repeat
- Just before a conditional execute instruction
- Etc.

And if the corresponding return_int is stalled at an ADDRESS or ACCESS1 phase, then the following may occur:

- The single repeat is executed more than expected and if it is located at the end of blockrepeat / localrepeat, the BRCx may be not get decremented.
- The instruction to be executed conditionally gets executed UNconditionally.

See the following examples.

Example 1

```
             AR0 = #0
             repeat(#15)
             AR0 = AR0 + #1             ; An interrupt is asserted here.
             AR0 = AR0 ^ #16            ; AR0 is expected to be 0 but not.
             if(AR0 != #0 ) goto ERROR
             .
      ISR:   .
             AR1 = AR1 – #1
             mar(*AR1+) || return_int   ; Stalled at ADDRESS phase.
```

Example 2

```
             .
             << An interrupt is asserted here >>
             if(cond=false)Execute(AD_Unit/D_Unit)
             Instruction to be executed conditionally always gets executed.
             .
      ISR:   .
             AR1 = AR1 – #1
             mar(*AR1+) || return_int   ; Stalled at ADDRESS phase.
```

Assembler Notification: Pending

Workaround: If the ”hold” feature, which can cause the CPU to stall, is not used, place 6 NOPs immediately before the return_int to avoid it from stalling.

Example

```
             nop
             nop
             nop
             nop
             nop
             nop
             return_int    ; No stalling during an ADDRESS or ACCESS1 phase.
```

Or do not use the fast return configuration This exception will not be fixed in future silicon revisions.

Advisory DSP_CPU_100

Interrupted Single Repeat Is Not Resumed After RETI

Revision(s) Affected: Revision C

Details: When an interrupt is asserted during any of the following single repeat instructions:

- while (cond && (RPTC < k8)) repeat
- repeat (k16)
- repeat (CSR)
- repeat (CSR) , CSR += DAx
- repeat (CSR) , CSR += k4
- repeat (CSR) , CSR –= k4
- repeat (k8)

The single repeat doesn't resume after returning from the interrupt under all of the following conditions:

- the restore of the repeat counter(RPTC) by MMR write in ISR is close(*) to the "return_int".
- the RPTC is 0 before the restore.

(*) if the instruction between restore RPTC and return_int is less than

- six for the fast return configuration.
- two for the slow return configuration.

Assembler Notification: Pending

Workaround: Insert 6 nops between the restore RPTC and return_int for the fast return configuration. Insert 2 nops between restore RPTC and return_int for the slow return configuration.

Example

```
ISR:   .                    ; Fast return configuration.
       .
       @RPTC_L=pop()||mmap()
       nop
       nop
       nop
       nop
       nop
       nop
       return_int.
```

This exception will not be fixed in future silicon revisions.

Advisory DSP_CPU_102

Page Register Update and CPU Bypass Corrupts Following Memory Read

Revision(s) Affected: Revision C

Details: In the following sequence,

```
INST0 : any
INST1 : any
INST2 : any
INST3 : Write to a memory
INST4 : Read from the same memory with CPU STALL
INST5 : Read from any memory (Doesn’t have to be same address with
      INST3,4)
```

INST5 may get wrong data from memory if the corresponding page register for the data read address generation has been updated as follows.

```
– by MMR write             at INST0,1 or 2 position
– by EXE phase instruction at INST  1 or 2 position
```

The following table shows all Page registers with MMR address. Instructions to update in EXE phase and the events (data read) to be used.

Page Register (MMR address)	EXE phase Inst.	Used by (Candidate of INST5)

DPH (2Bh)	XDP = xsrc
	XDP = dbl(Lmem)
	XDP = popboth()
		– Direct addressing (CPL=0)

SPH,SSPH (4Eh)	XSP = xsrc
	XSP = dbl(Lmem)
	XSP = popboth()
	XSSP = xsrc
	XSSP = dbl(Lmem)
	XSSP = popboth()
		– Direct addressing (CPL=1)
		– All kind of return INST.
		– All kind of pop INST.

CDPH (4Fh)	XCDP = xsrc
	XCDP = dbl(Lmem)
	XCDP = popboth()
		– Indirect addressing with CDP pointer

ARx_H (None)	XARx = xsrc
	XARx = dbl(Lmem)
	XARx = popboth()
		– Indirect addressing with with ARx pointer

Page Register Update and CPU Bypass Corrupts Following Memory Read (Continued)

Example:

In the following example, the DR1 gets corrupted value.

```
XAR1 = XAR3                   ; The page of AR1 is updated in EXE.
nop
*AR6 = #0xABCD || DR3 = AR7   ; Write to a memory
DR0 = *AR6     || AC0 = DR3   ; Read from the same memory with CPU stall
DR1 = *AR1     || DR2 = *AR2  ; Reading data from *AR1 using XAR1
```

Assembler Notification: Pending

Workaround: Use one of the following workarounds:

Case 1: Have at least 3 instructions between the page register update by the MMR write and the next write instruction as follows. As the assembler cannot detect ”Page register update by MMR write”, this condition must be confirmed by users.

```
Page register update by MMR write in WRITE phase
INST
INST
INST
Write to a memory
Read from the memory
Read from memory/stack using the updated page register
```

Case 2: Have at least 2 instructions between the page register update by EXE phase instruction and the next write instruction as follows. If there’s less than 2 instructions, it is planned that a future revision of the assembler will reject it.

```
Page register update by EXE phase Instructions
INST
INST
Write to a memory
Read from the memory
Read from memory/stack using the updated page register
```

Case 3: Use ”dst = mar(Smem)” which is to update the Page Register at ADDRESS phase like follows.

```
dst = mar(Smem) ; dst can be XARn, XCDP, XDP, XSP or XSSP.
Write to a memory
Read from the memory
Read from memory/stack using the updated page register
```

This exception will not be fixed in future silicon revisions.

3.2 DSP DMA Advisories

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

Revision(s) Affected: Revision C

Details: If the EMIF times out on an access, the DSP will get a timeout bus–error interrupt. The time–out condition may also cause a DMA interrupt. In the case where a DMA interrupt occurs, the DMA won't timeout, but hangs instead.

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 on future silicon revisions.

Advisory DSP_DMA_2 — *DSP DMA IDLE Prevents Transfer Completion*

Revision(s) Affected: Revision C

Details: When system modules are placed in IDLE, there is hardware handshaking to ensure IDLE can occur without any system consequences.

The DSP DMA, however, goes into IDLE even though a transfer is occurring. This may prevent an expected from transfer 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. The DMA channels then need to be disabled, and 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: Revision C

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 exception will not be fixed in future silicon revisions.

This same functional limitation is present on the System DMA controller, but the programmation restrictions stated in this workaround are documented in the *OMAP5910 Technical Reference Manual* (literature number SPRU602).

4 MPU Subsystem Advisories

4.1 MPU Data-Cache Advisories

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

Revision(s) Affected: Revision C

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 – write one word of the cache line, read the word that will replace it; write the next word of the cache line one word, 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 to the cache.

Workaround: Do not enable Transparent Mode in the ARM925T Configuration register (it is disabled by default). This exception will not be fixed on future silicon revisions.

4.2 System DMA Advisories

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

Revision(s) Affected: Revision C

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 DMA 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 register, which is the only supported setting of this bit) there is no benefit to software setting the DMACK_REQ bit to "0". There is no impact to power or performance with this workaround.

This exception will not be corrected on future silicon revisions.

5 Traffic Controller Subsystem Advisories

5.1 EMIF Slow (EMIFS) Advisories

Advisory EMIFS_1	*FLASH.RDY Should Not Be Used With Intel Burst Flash WAIT Signal*

Revision(s) Affected: Revision C

Details: Intel’s most recent burst flash devices assert the WAIT signal during the addressing phase of a transaction and de–assert it when the first data word is available. However, the timing of the initial data word is independent of the de–assertion of WAIT (i.e. it is a fixed access time from address to data). Essentially, this makes the WAIT signal “extra” information. This behavior includes the W18, K18 and K3 families of burst flash.

During syncronous burst read mode, the OMAP5910 flash interface does not expect a transition to occur on the WAIT signal during the addressing phase of a transaction. This WAIT assertion can cause the Memory Interface to incorrectly process the flash transaction.

This will only occur if the FLASH.RDY signal of the flash interface is connected to the flash WAIT signal.

Workaround: Do NOT connect the WAIT signal from the Intel burst flash to the flash interface WAIT#FRDY input. Instead, tie the WAIT#FRDY signal through a pull–up to DVdd. There are NO performance implications to this workaround.

This workaround is possible because the OMAP5910’s Traffic Controller will only permit a burst to an aligned address, preventing any possibility to cross an Intel burst flash page boundary. For instances where an unaligned access is required, the Traffic Controller translates these into single accesses until a burstable boundary is reached. Therefore, the WAIT from Intel burst flash can be completely ignored by the OMAP5910 processor.

This exception will not be fixed on future silicon revisions.

Advisory EMIFS_2

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

Revision(s) Affected: Revision C

Details: EMIFS supports burst writes. 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 16bit width). During split transfers, EMIFS has to follow the specifications single write specifications. But from the 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 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(old)	WELEN(new)	Flash clock
“00”	X	X + 2	TC_clock/1
“01”	X	X + 1	TC_clock/2
“10”	X	X + 1	TC_clock/4
“11”	X	X + 1	TC_clock/6

This exception will not be fixed in future silicon revisions.

Advisory EMIFS_3

WELEN = 0 and FDIV = 1 WIth 16-Bit Memory

Revision(s) Affected: Revision C

Details: When performing an asynchronous operation to 16–bit memories, connected to EMIFS interface, with WELEN=0 and FCLKDIV=00 (configured in 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 cause the system to hang.

Workaround: Use one of the following 2 workarounds:

1. Program WELEN (of EMIFS configuration registers) greater than 0
2. Use FCLKDIV (of EMIFS configuration registers) greater than “00”

This exception will not be fixed in future silicon revisions.

Advisory EMIFS_4

EMIFS Wait States

Revision(s) Affected:	Revision C
Details:	Data pipeline in EMIFS is broken during burst write with certain 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 on future silicon revisions.

5.2 EMIF Fast (EMIFF) Advisories

Advisory EMIFF_1

EMIFF Configuration Preventing Deep Sleep Entry

Revision(s) Affected: Revision C

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 (bit0) 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 the deep sleep state is entered correctly and 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 to be 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

As there is a simple software workaround, this exception will not be fixed in future silicon revisions.

6 OMAP5910 Peripheral Advisories

6.1 LCD Advisories

Advisory LCD_1	*Missing the Palette Loading Interrupt*

Revision(s) Affected: Revision C

Details: When LCD is in Palette and Data loading mode and the palette loading interrupt occurs, the LCD sends an interrupt, but the status register bit does not get set. ARM will see the interrupt, but will not be able to identify what caused it because LCD status register doesn't capture it. The real intention of this bit, however, is for palette loading mode only. If the user is in palette loading mode only, 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.

6.2 UART Advisories

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

Revision(s) Affected: Revision C

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 re–started.
- If XOFF1 and XOFF2 are used they must be received sequentially in order to halt the transmission.

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

There are two issues:

- XOFF with framing error (only this one) 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:

1. use hardware flow control.
2. use software to perform software flow control without hardware assist.

This exception will not be fixed on 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 allow deep sleep mode is to be used, 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.

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

Advisory UART_3

OSC_12M_SEL and EBLR Registers Are Not Readable

Revision(s) Affected:	Revision C
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.

6.3 MMC/SD Advisories

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

Revision(s) Affected: Revision C

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 on future silicon revisions.

6.4 Microwire Advisories

Advisory UWIRE_1

Pulldown On the UWIRE.SDI Pin Needs to be Disabled by Software

Revision(s) Affected: Revision C

Details: On the UWIRE.SDI pin (ball J14 of the 289–pin BGA device when configured for UWIRE) pin, there is a pull–down active by default. The inactive level of the signal is high, so the pull down needs to be disabled in software in OMAP5910 native mode otherwise power is wasted. The pull–down is disabled in compatibility mode.

Workaround: Write 0x0000EAEFh 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.

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: In this configuration, bit 0 can be ignored by right shifting the RX register value by 1 after reading the data from the RX register. This work–around cannot be used in the 16–bit mode since 1 bit of data is lost due to the extra captured bit. In 16–bit mode, a different configuration must be used.

Configuration #2: No work–around exists. A different configuration must be used.

6.5 I^{2}C Advisories

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

Revision(s) Affected: Revision C

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

This exception will not be fixed in future silicon revisions.

6.6 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: Revision C

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 pre–initializes 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 1 USBF_2

USB Function Supend Functionality in HMC_MODE 13 and HMC_MODE 15

Revision(s) Affected: Revision C

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: Revision C

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.

6.7 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: Revision C

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 on future silicon revisions.

7 OMAP5910 Device/System Level Advisories

7.1 System Advisories

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

Revision(s) Affected: Revision C

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".*

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

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

Revision(s) Affected: Revision C

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 is guaranteed that the first write must complete and the read data will be correct. This exception will not be fixed on future silicon revisions.

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

Revision(s) Affected: Revision C

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.

8 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 Fixed-Point Digital Signal Processor* data manual*,* literature number SPRS076
- *TMS320C55x™ DSP Functional Overview,* literature number SPRU312

For additional information, see the latest versions of:

- *TMS320C55x DSP CPU Reference Guide* (literature number SPRU371)
- *TMS320C55x DSP Mnemonic Instruction Set Reference Guide* (SPRU374)
- *TMS320C55x DSP Algebraic Instruction Set Reference Guide* (SPRU375)