TMS570LS12x/11x Microcontroller

Silicon Revision C

Silicon Errata

Literature Number: SPNZ218C
January 2015–Revised May 2016

Contents

List of Figures

List of Tables

TMS570LS12x/11x *Microcontroller*

This document describes the known exceptions to the functional specifications for the device.

1 Device Nomenclature

To designate the stages in the product development cycle, TI assigns prefixes to the part numbers of all devices. Each commercial family member has one of three prefixes: TMX, TMP, or TMS (for example, **TMS**570LS3137). These prefixes represent evolutionary stages of product development from engineering prototypes (TMX) through fully qualified production devices/tools (TMS).

Device development evolutionary flow:

TMX — Experimental device that is not necessarily representative of the final device's electrical specifications.

TMP — Final silicon die that conforms to the device's electrical specifications but has not completed quality and reliability verification.

TMS — Fully-qualified production device.

TMX and TMP devices are shipped against the following disclaimer:

"Developmental product is intended for internal evaluation purposes."

TMS devices have been characterized fully, and the quality and reliability of the device have been demonstrated fully. TI's standard warranty applies.

Predictions show that prototype devices (TMX or TMP) have a greater failure rate than the standard production devices. Texas Instruments recommends that these devices not be used in any production system because their expected end-use failure rate still is undefined. Only qualified production devices are to be used.

2 Revision Identification

Figure 1 provides examples of the TMS570LSx device markings. The device revision can be determined by the symbols marked on the top of the device.

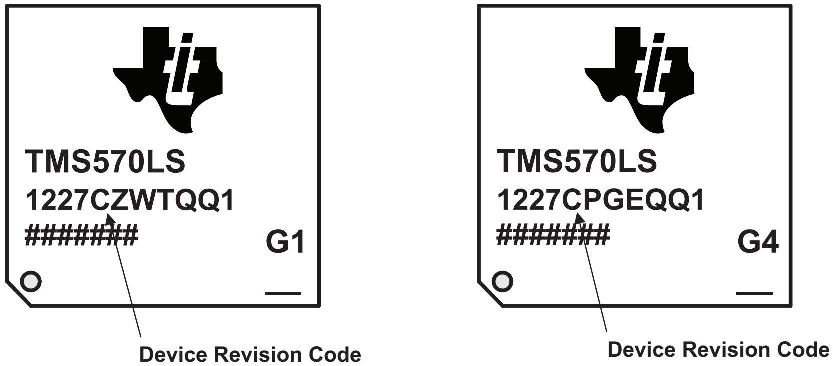

Figure 1. Device Revision Code Identification

Silicon revision is identified by a device revision code. The symbolization is of the format TMS570LS1227x, where "x" denotes the silicon revision. If x is "C" in the device part number, it represents device silicon revision C.

3 Silicon Changes from Previous Device Revision

The following errata have been fixed going from silicon revision B to silicon revision C. For a description of these errata see the Silicon Revision B errata document SPNZ199.

Table 1. Errata Which have been Fixed

Erratum	Considerations When Migrating From Revision B to Revision C Silicon
DEVICE#B064	May leave workaround in place, no change to SW required. Change can be made to re-gain EMIF performance.
DEVICE#B071	May leave workaround in place, no change to software required.
PBIST#4	May leave workaround in place, no change to software required.
SSWF021#44	Evaluate impact of additional 384 OSCIN cycles to PLL lock time.
STC#31	May leave workaround in place, no change to software required.

4 Known Design Exceptions to Functional Specifications

Table 2. Known Design Exceptions to Functional Specifications

Title **Page**

Table 2. Known Design Exceptions to Functional Specifications (continued)

ADC#1 *Injecting current into an input channel shared between the two ADCs causes a DC offset in conversion results of other channels*

Severity 3 - Medium

Expected Behavior External circuit connected to one channel must not affect the conversion result of another channel.

Issue This microcontroller (MCU) has two Analog-to-Digital Converters (ADCs). Some of the input channels are unique to ADC1 while some are shared between ADC1 and ADC2. Figure 2 shows a block diagram of an input channel shared between ADC1 and ADC2.

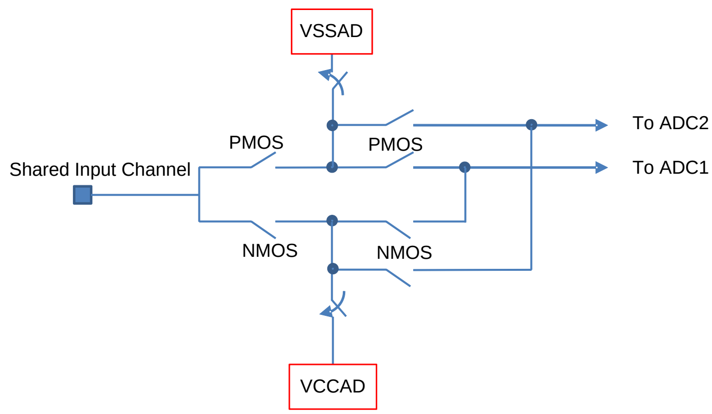

Figure 2. Shared Input Channel in "Open" State

The PMOS and NMOS switches are open indicating that this shared input channel is not currently being sampled either by ADC1 or by ADC2. Also, there are switches to VCCAD and VSSAD that are closed. If any current is injected into this analog input, any leakage through the open PMOS switch will be shunted to VSSAD. These switches to VSSAD and VCCAD are opened as soon as this shared input channel is being sampled by either ADC1 or ADC2.

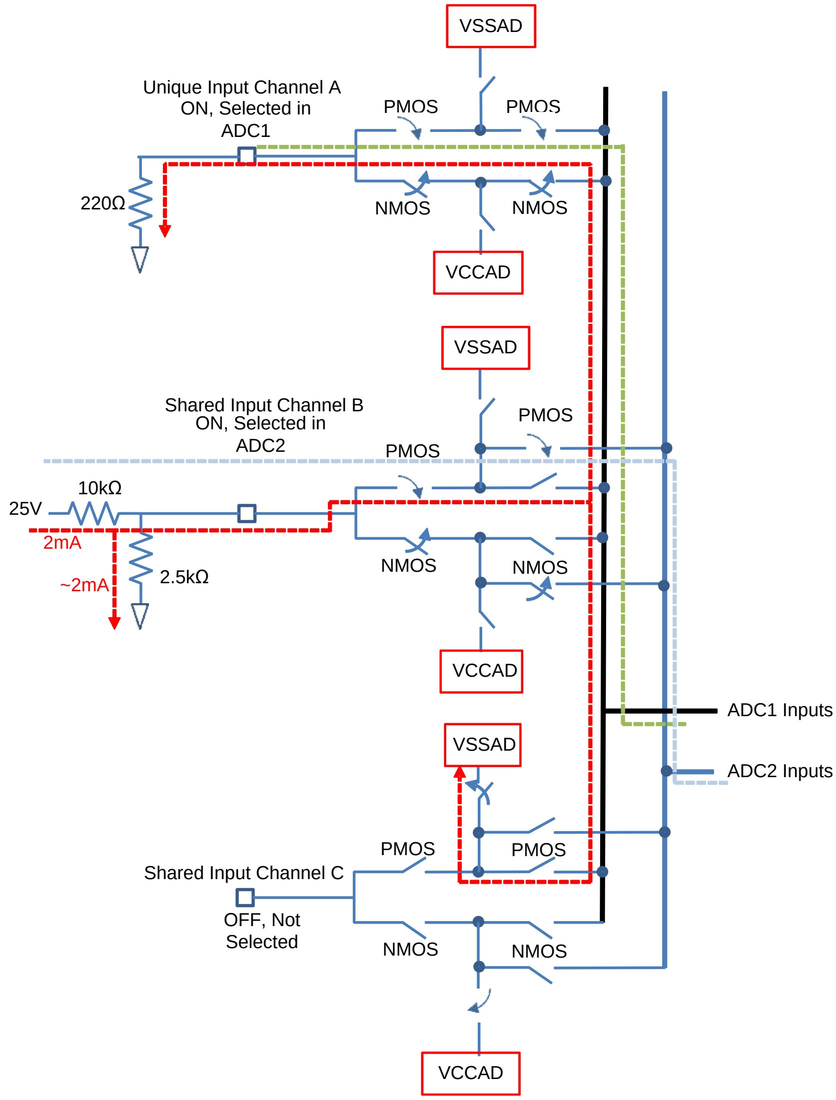

Figure 3. Example ADC1/ADC2 Channel Connection

Figure 3 shows an example where a ADC1 is sampling input channel A which is unique to ADC1, and ADC2 is sampling input channel B, which is a shared-input channel. This is shown by the dashed green and light blue current paths.

Another current path is shown in dashed dark red. This is a current injected into channel B as the input level on terminal B is greater than VCCAD - 0.3V. This is a parasitic current that passes through the "open" PMOS switch, and a part of this current flows to ground through the external 220 ohm resistor connected to input channel A. This causes an offset in the conversion result of channel A being sampled by ADC1.

Conditions

This issue occurs if:

1. Input voltage on a shared input channel being sampled by one ADC is (VCCAD - 0.3V) or higher, and
2. The second ADC samples another channel such that there is some overlap between the sampling windows of the two ADCs

Implications

An offset error is introduced in the conversion result of any channel if a current is being injected into a shared input channel.

Workaround(s)

There are two options to minimize the impact of this issue:

1. Configure the two ADC modules such that their sampling periods do not overlap, or
2. Limit the shared analog input upper limit to be lower than (VCCAD - 0.3V). The PMOS leakage is reduced exponentially if the input is lower than VCCAD - 0.3V.

AHB_ACCES_PORT#3 (ARM ID-529470) *Debugger may display unpredictable data in the memory browser window if a system reset occurs*

Severity	3-Medium
Expected Behavior	If a system reset (nRST goes low) occurs while the debugger is performing an access on the system resource using system view, a slave error should be replied to the debugger.
Issue	Instead, the response might indicate that the access completed successfully and return unpredictable data if the access was a read.
Condition	System reset is asserted LOW on a specific cycle while the debugger is completing an access on the system using the system view. An example would be the debugger like the CCS's memory browser window is refreshing its content using the system view. This is not an issue for a CPU only reset. This is not an issue during power-on reset (nPORRST) either.
Implication(s)	Data read using the debugger in system view while a system reset occurs may be corrupt, writes may be lost.
Workaround(s)	This is a workaround for users and tools vendors. Avoid performing debug reads and writes while the device might be reset.

CORTEX-R4#26 (ARM ID-577077) *Thumb STREXD Treated As NOP If Same Register Used For Both Source Operands*

Severity

3-Medium

Expected Behavior

The STREXD instruction should work in Thumb mode when Rt and Rt2 are the same register.

Issue

The ARM Architecture permits the Thumb STREXD instruction to be encoded with the same register used for both transfer registers (Rt and Rt2). Because of this issue, the Cortex-R4 processor treats such encoding as UNPREDICTABLE and executes it as a NOP.

Condition

This error occurs when the processor is in Thumb state and a STREXD instruction is executed with Rt = Rt2.

Note: this instruction is new in ARM Architecture version 7 (ARMv7). It is not present in ARMv6T2 or other earlier architecture versions.

Implication(s)

If this error occurs, the destination register, Rd, which indicates the status of the instruction, is not updated and no memory transaction takes place. If the software is attempting to perform an exclusive read-modify-write sequence, then it might either incorrectly complete without memory being written, or loop forever attempting to complete the sequence.

Workaround(s)

This issue can be avoided by using two different registers for the data to be transferred by a STREXD instruction. This may involve copying the data in the transfer register to a second, different register for use by the STREXD.

Comment: TI Code Generation tool does not generate exclusive access load or store instructions. On these Hercules devices there is no reason to use exclusive access instructions.

CORTEX-R4#27 (ARM ID-412027) *Debug Reset Does Not Reset DBGDSCR When In Standby Mode*

Severity 3-Medium

Expected Behavior The debug reset input, PRESETDBGn, resets the processor's debug registers as specified in the ARMv7R Architecture. The debug reset is commonly used to set the debug registers to a known state when a debugger is attached to the target processor.

Issue When the processor is in Standby Mode and the clock has been gated off, PRESETDBGn fails to reset the Debug Status and Control Register (DBGDSCR).

Condition

1. The DBGDSCR register has been written so that its contents differ from the reset values (most fields in this register reset to zero, though a few are UNKNOWN at reset), and
2. The processor is in Standby Mode, and the clocks have been gated off, that is STANDBYWFI is asserted, and
3. The debug reset, PRESETDBGn, is asserted and de-asserted while the processor clocks remain gated off.

Note: the debug reset is commonly used to set the debug registers to a known state when a debugger is attached to the target processor.

Implication(s) This issue affects scan based debug utility developers. The end user should not be affected by this issue if the development tool vendor has implemented the workaround.

If this issue occurs, then after the reset, the DBGDSCR register contains the values that it had before reset rather than the reset values. If the debugger relies on the reset values, then it may cause erroneous debug of the processor. For example, the DBGDSCR contains the ExtDCCmode field which controls the Data Communications Channel (DCC) access mode. If this field was previously set to Fast mode but the debugger assumes that it is in Non-blocking mode (the reset value) then debugger accesses to the DCC will cause the processor to execute instructions which were not expected.

Workaround(s) This can be avoided by a workaround in the debug control software. Whenever the debugger (or other software) generates a debug reset, follow this with a write of zero to the DBGDSCR which forces all the fields to their reset values.

CORTEX-R4#33 (ARM ID-452032) *Processor Can Deadlock When Debug Mode Enables Cleared*

Severity

3-Medium

Expected Behavior

The Cortex-R4 processor supports two different debugging modes: Halt-mode and Monitor-mode. Both modes can be disabled. Bits [15:14] in the Debug Status and Control Register (DBGDSCR) control which, if any, mode is enabled. Additionally, debug events can only occur if the invasive debug enable pin, DBGEN is asserted. Deadlocks should not occur when the debug mode is changed.

Issue

If there are active breakpoints or watchpoints at the time when the debugging modes are disabled via the DBGDSCR or DBGEN, this issue can cause the processor to deadlock (in the case of a breakpoint) or lose data (in the case of a watchpoint).

Condition

1. DBGEN is asserted and the processor is running, and
2. At least one breakpoint or watchpoint is programmed and active, and
3. Either halt-mode debugging or monitor mode debugging is enabled, and
4. Either an instruction is fetched which matches a breakpoint, or an item of data is accessed which matches a watchpoint, and
5. After the instruction or data is accessed, but before the instruction completes execution, either the DBGEN input is de-asserted or both halt-mode and monitor-mode debugging are disabled by means of a write the DBGDSCR.

Implication(s)

This issue affects scan based debug utility developers. The end user should not be affected by this issue if the development tool vendor has implemented the workaround.

Depending on which of the conditions are met, the processor will either lose data or deadlock. If the processor deadlocks because of this issue it will still respond to interrupts provided they are not masked.

Workaround(s)

This issue can be avoided by ensuring that all watchpoints and breakpoints are made inactive before either de-asserting DBGEN or changing the debug mode enables.

CORTEX-R4#46 (ARM ID-599517) *CP15 Auxiliary ID And Prefetch Instruction Accesses Are UNDEFINED*

Severity

3-Medium

Expected Behavior

The ARMv7-R architecture requires implementation of the following two features in CP15:

1. An Auxiliary ID Register (AIDR), which can be read in privileged modes, and the contents and format of which are IMPLEMENTATION DEFINED.
2. The operation to prefetch an instruction by MVA, as defined in the ARMv6 architecture, to be executed as a NOP.

Because of this issue, both of these CP15 accesses generate an UNDEFINED exception on Cortex-R4.

Issue

CP15 accesses to Auxiliary ID Register (AIDR) or an operation to prefetch an instruction by MVA will generate an UNDEFINED exception on Cortex-R4.

Condition

Either of the following instructions is executed in a privileged mode:

- MRC p15,1,<Rt>,c0,c0,7 ; Read IMPLEMENTATION DEFINED Auxiliary ID Register
- MCR p15,0,<Rt>,c7,c13,1 ; NOP, was Prefetch instruction by MVA in ARMv6

Implication(s)

This issue should only affect portable code supposed to run on different ARM architecture or code running on cached Cortex-R4. Code written for Hercules products should not be affected.

Workaround(s)

The CP15 AIDR and MVA registers are not implemented on Cortex-R4 CPU. To avoid this issue, don't read or write to them.

CORTEX-R4#55 (ARM ID-722412) *CPACR.ASEDIS and CPACR.D32DIS return incorrect value when implementation includes floating point unit.*

Severity 3-Medium

Expected Behavior Because the Cortex-R4F CPU does not include the Advance SIMD (NEON) unit or registers D16-D32, it should return a value of 11 (disabled) for CP15 CPACR [31:30] (AESDIS and D32DIS).

Issue Because of this issue, these bits read zero in implementations of Cortex- R4F which include the floating-point unit.

Condition On reads, CPACR[31:30] actually read as 00 not 11.

Implication(s) Software cannot uses the CPACR to determine whether Advanced SIMD functionality and registers D16-D32 are available.

Workaround(s) Hercules products do not include Advance SIMD (NEON) unit or registers D16-D32 so there is no need to check for these features.

CORTEX-R4#57 (ARM ID-737195) *Conditional VMRS APSR_Nzcv, FPSCR May Evaluate With Incorrect Flags*

Severity

3-Medium

Expected Behavior

A conditional VMRS APSR_nzcv, FPSCR instruction should evaluate its condition codes using the correct flags.

Issue

Under certain circumstances, a conditional VMRS APSR_nzcv, FPSCR instruction may evaluate its condition codes using the wrong flags and incorrectly execute or not execute.

Condition

The issue requires the following sequence of instructions in ARM state:

1. - VMRS<c> APSR_nzcv, FPSCR (formerly FMSTAT<c>), where the condition on the instruction is not always. This instruction immediately following:
2. A flag-setting integer multiply or multiply and accumulate instruction (e.g. MULS)
3. A single-precision floating-point multiply-accumulate (FP-MAC) instruction (e.g. VMLA), timed such that the accumulate operation is inserted into the pipeline in the cycle in which the VMRS instruction is first attempted to be issued.

To meet the above timing requirements, the VMRS instruction must be three pipeline stages behind the FPMAC. Depending on the rate in which the instructions are fetched, interlocks within this sequence and dual-issuing, this can be up to three other instructions between this pair, plus the multiply. Out-of-order completion of FP-MAC instructions must be enabled.

Implication(s)

If this issue occurs, the VMRS instruction will pass or fail its condition codes incorrectly, and this will appear in any trace produced by the ETM. This can corrupt the N, Z, C, V flag values in the CPSR which will typically affect the program flow.

Workaround(s)

Workaround: This issue can be avoided by disabling out-of-order single-precision floating-point multiply-accumulate (SPMAC) instruction completion. Set DOOFMACS, bit [16] in the Secondary Auxiliary Control Register. This will have the side-effect of reducing the performance of SP-MAC operations, though the impact will depend on how these instructions are used in your code.

CORTEX-R4#58 (ARM ID-726554) *DBGDSCR.Adadiscard Is Wrong When DBGDSCR.Dbgack Set*

Severity 3-Medium

Expected Behavior When the DBGDSCR.ADAdiscard bit is set, asynchronous data aborts are discarded, except for setting the DBGDSCR.ADAbort sticky flag. The Cortex-R4 processor ensures that all possible outstanding asynchronous data aborts have been recognized before it enters debug halt state. The flag is immediately on entry to debug halt state to indicate that the debugger does not need to take any further action to determine whether all possible outstanding asynchronous aborts have been recognized.

Issue Because of this issue, the Cortex-R4 processor also sets the DBGDSCR.ADAdiscard bit when the DBGDSCR.DBGack bit is set. This can cause the DBGDSCR.ADAbort bit to become set when the processor is not in debug halt state, and it is not cleared when the processor enters debug halt state. However, the processor does not discard the abort. It is pending or generates an exception as normal.

Condition

1. The processor is not in debug halt state
2. The DBGDSCR.DBGack bit is set
3. An asynchronous data abort (for example, SLVERR response to a store to Normal-type memory) is recognized

NOTE: it is not expected that DBGDSCR.DBGack will be set in any Cortex-R4 system

Implication(s) Hercules users will not be impacted by this issue, because Code Composer Studio takes care of this condition.

If this issue occurs, and the processor subsequently enters debug halt state, the DBGDSCR.ADAbort bit will be set, when in fact no asynchronous data abort has occurred in debug state. Before exiting debug state, the debugger will check this bit and will typically treat it as an error. If no other asynchronous data abort has occurred in debug state, this is a false error.

Workaround(s) None.

CORTEX-R4#61 (ARM ID-720270) *Latched DTR-Full Flags Not Updated Correctly On DTR Access.*

Severity 3-Medium

Expected Behavior When the debug Data Transfer Register (DTR) is in non-blocking mode, the latched DTR-full flags (RXfull_l and TXfull_l) record the state of the DTR registers as observed by the debugger and control the flow of data to and from the debugger to prevent race hazards. For example, when the target reads data from DBGDTRRXint, the associated flag RXfull is cleared to indicate that the register has been drained, but the latched value Rxfull_l remains set. Subsequent debugger writes to DBGDTRRXext are ignored because RXfull_l is set. RXfull_l is updated from RXfull when the debugger reads DBGDSCRext such that a debugger write to DBGDTRRXext will only succeed after the debugger has observed that the register is empty. The ARMv7 debug architecture requires that RXfull_l be updated when the debugger reads DBGDSCRext and when it writes DBGDTRRXext. Similarly, TXfull_l must be updated when the debugger reads DBGDSCRext and when it reads DBGDTRTXext.

Issue Because of this issue, RXfull_l and TXfull_l are only updated when the debugger reads DBGDSCRext.

Condition The DTR is in non-blocking mode, that is, DBGDSCR.ExtDCCmode is set to 0b00 and EITHER:

1. The debugger reads DBGDSCRext which shows that RXfull is zero, that is, DBGDTRRX is empty, and then
2. The debugger writes data to DBGDTRRXext, and
3. Without first reading the DBGDSCRext, and before the processor has read from DBGDTRRXint, the debugger performs another write to DBGDTRRXext.

OR

1. The debugger reads DBGDSCRext which shows that TXfull is one, that is, DBGDTRTX is full, and then
2. The debugger reads data from DBGDTRTXext, and then
3. The processor writes new data into DBGDTRTXint, and
4. Without first reading the DBGDSCRext, the debugger performs another read from DBGDTRTXext.

Implication(s) The ARMv7 debug architecture requires the debugger to read the DBGDSCRext before attempting to transfer data via the DTR when in non-blocking mode. This issue only has implications for debuggers that violate this requirement. If the issue occurs via data transfer, data loss may occur. The architecture requires that data transfer never occur.

Texas Instruments has verified that TI's Code Composer Studios IDE is not affected by this issue.

Workaround(s) None.

CORTEX-R4#66 (ARM ID-754269) *Register Corruption During a Load-Multiple Instruction at an Exception Vector*

Severity 3-Medium

Expected Behavior LDM will execute properly when used as the first instruction of an exception routine.

Issue Under certain circumstances, a load multiple instruction can cause corruption of a general purpose register.

Condition All the following conditions are required for this issue to occur:

1. A UDIV or SDIV instruction is executed with out-of-order completion of divides enabled
2. A multi-cycle instruction is partially executed before being interrupted by either an IRQ, FIQ or imprecise abort. In this case, a multi-cycle instruction can be any of the following:
   - LDM/STM that transfers 3 or more registers
   - LDM/STM that transfers 2 registers to an unaligned address without write back
   - LDM/STM that transfers 2 registers to an aligned address with write back
   - TBB/TBH
3. A load multiple instruction is executed as the first instruction of the exception handler
4. The load multiple instruction itself is interrupted either by an IRQ, FIQ, imprecise abort or external debug halt request.

This issue is very timing sensitive and requires the UDIV or SDIV to complete when the load multiple is in the Issue stage of the CPU pipeline. The register that is corrupted is not necessarily related to the load-multiple instruction and will depend on the state in the CPU store pipeline when the UDIV or SDIV completes.

Implication(s) For practical systems, it is not expected that an interruptible LDM will be executed as the first instruction of an exception handler, because the handler is usually required to save the registers of the interrupted context. Therefore, it is not expected that this issue has any implications for practical systems. If the situation of the issue occurs it will result in the corruption of the register bank state and could cause a fatal failure if the corrupted register is subsequently read before being written.

Workaround(s) To work around this issue, set bit [7] of the Auxiliary Control Register to disable out-of-order completion for divide instructions. Code performance may be reduced depending on how often divide operations are used.

CORTEX-R4#67 (ARM ID-758269) *Watchpoint On A Load Or Store Multiple May Be Missed.*

Severity 3-Medium

Expected Behavior The Cortex-R4 supports synchronous watchpoints. This implies that for load and store multiples, a watchpoint on any memory access will generate a debug event on the instruction itself.

Issue Due to this issue, certain watchpoint hits on multiples will not generate a debug event.

Condition All the following conditions are required for this issue to occur:

1. A load or store multiple instruction is executed with at least 5 registers in the register list.
2. The address range accessed corresponds to Strongly-Ordered or Device memory.
3. A watchpoint match is generated for an access that does not correspond to either the first two or the last two registers in the list.

Under these conditions the processor will lose the watchpoint. Note that for a "store multiple" instruction, the conditions are also affected by pipeline state making them timing sensitive.

Implication(s) Due to this issue, a debugger may not be able to correctly watch accesses made to Device or Strongly-ordered memory. The ARM architecture recommends that watchpoints should not be set on individual Device or Strongly-ordered addresses that can be accessed as part of a load or store multiple. Instead, it recommends the use of the address range masking functionality provided to set watchpoints on an entire region, ensuring that the watchpoint event will be seen on the first access of a load or store multiple to this region.

If this recommendation is followed, this issue will not occur.

Workaround(s) None.

DCC#24 *Single Shot Mode Count may be Incorrect*

Severity 3-Medium

Expected Behavior When the first clock source counts down to zero, the countdown value remaining for the other clock source is accurately captured.

Issue The first issue is that there is an offset in starting and stopping the two counters due to synchronization with VCLK that leads to a fixed offset. The second issue is that the value remaining in the counter that did not reach zero may be latched while the bits are in transition, giving an erroneous value.

Condition When used in single shot mode and the count value captured is not from VCLK.

Implication(s) The cycle count captured may be incorrect.

Workaround(s) Static frequency offset can be removed by making two measurements and subtracting. The sporadic offset can be removed by making multiple measurements and discarding outliers -- an odd filtering algorithm.

DEVICE#142 *CPU Abort Not Generated on Write to Unimplemented MCRC Space*

Severity Low

Expected Behavior A write to the unimplemented region (0xFE00_0200 to 0xFEFF_FFFF) of the MCRC module will generate an abort

Issue Sometimes a cpu abort does not get generated.

Conditions When single stepping through the instruction that does the illegal write,
or
when there is a breakpoint on the instruction immediately after the illegal write.

Implications The abort will not be generated when debugging.

Workaround(s) None

DEVICE#B053 — *CPU code execution could be halted on a device warm reset if the core power domain # 2 is disabled by software.*

Severity
3-Medium

Expected Behavior
The CPU code execution must start from the reset vector (address 0x00000000) upon a device warm reset and is not affected by the state of any switchable device power domain.

Issue
CPU code execution could be halted upon a warm reset if the core power domain # 2 has been disabled by software prior to the device warm reset.

Condition
The behavior is not dependent on any particular operating condition.

Implication(s)
CPU code execution is halted so that a system hang occurs. An external monitor must be present to prevent the system from entering an unsafe state when this happens.

Workaround(s)
The application must not disable the core power domain # 2 in software via the Power Management Module (PMM) registers, even if the modules inside this core power domain are not used in the application.

DEVICE#B063 *Incorrect PSCON Compare Error*

Severity

3 - Medium

Expected Behavior

No Power-State Controller (PSCON) compare errors are expected when disabling a logic power domain

Issue

A false PSCON compare error is generated when disabling a logic power domain.

Conditions

This problem might occur if either:

1. A logic power domain is disabled at reset by a factory OTP setting, or
2. Software explicitly disables a power domain via the Power Management Module (PMM).

Implications

ESM group 1 channel 38 and channel 39 errors may be incorrectly generated during reset or when software disables a power domain. These PSCON compare errors are not real and can be cleared and ignored.

Workaround(s)

This workaround must be implemented in the system initialization after reset, but before enabling either the interrupt (through register ESMIESR4) or nERROR pin action (through register ESMIEPSR4) for ESM group 1 channels 38 and 39. Switching disabled power domains back on requires a system reset so this workaround is only required during system initialization.

1. Disable power domains that are to be turned off (if no power domains are to be disabled by software, skip to step 2)
   - Write a ‘1’ to each bit of the PDCLKDISSET register corresponding to the power domain that you intend to disable.
   - Write 0xA to the appropriate bit-fields of the LOGICPDPWRCTRL0 register to power down the domains you intend to disable.
   - Poll the appropriate LOGICPDPWRSTATx register for bits [1:0] to become 00 for each domain you have disabled. The power domain is now powered down (Allows for delay time of Power Good signal)
2. Clear any PSCON compare error flags. (You can write to clear all four of the flags because if there is a true error condition in the PSCON, the compare error flag will immediately be set again.)
   - Write 0x000F0000 to LPDDCSTAT1 (0xFFFF00B0)
3. Clear ESM Group 1 flags 38 and 39
   - Write 0x000000C0 to ESMSR4 (0xFFFFF558)
4. Enable the effect of ESM group 1 channels 38 and 39, interrupt request or error pin toggle if desired.

The PSCON is now in lock step with its diagnostic partner, and any difference will now result in a true ESM error.

DEVICE#B066 — *HCLK Stops Prematurely when Executing from Flash*

Severity
3-Medium

Expected Behavior
To reduce power consumption, the CPU may request that the memory clock, HCLK, is disabled by setting bit 1 of the Clock Domain Disable Register (CDDIS.1). After the CPU makes this request, the flash bank is expected to monitor CPU activity and delay the actual disable of HCLK until the flash bank's Active Grace Period (BAGP) has expired (meaning that the CPU has stopped requesting instructions and data from the flash bank for some number of clock cycles).

Issue
The flash bank fails to delay the disable of HCLK. Therefore the CPU may freeze before it executes the WFI instruction.

Condition
The code requests to disable HCLK by setting bit 1 of the Clock Domain Disable register (CDDIS.1).

Implication(s)
If HCLK is disabled, and the CPU stops before executing the "WFI" instruction, the CPU will not resume execution on a wakeup interrupt.

Workaround(s)
A WFI instruction should immediately follow the instruction that sets bit 1 of the Clock Domain Disable Register.

DMA#27 *DMA Requests Lost During Suspend Mode*

Severity 3-Medium

Expected Behavior While the device is halted in suspend mode by the debugger the DMA is expected to complete the remaining transfers of a block if the DEBUGMODE of the GCTRL register is configured to '01'.

Issue The DMA does not complete the remaining transfers of a block but rather stops after two more frames of data are transferred. Subsequent DMA requests from a peripheral to trigger the remaining frames of a block can be lost.

Condition This only happens when:

- The device is suspended by a debugger &
- A peripheral continues to generate requests while the device is suspended &
- The DMA is setup to continue the current block transfer during suspend mode with DEBUGMODE field of the GCTRL register set to '01' &
- And the request trigger type TTYPE is set to frame trigger

Implication(s) When the DMA comes out of the suspend mode to resume the transfer, the data transfers corresponding to the third and subsequent requests will be lost.

Workaround(s) Workaround 1: Use TTYPE = Block transfer when DEBUGMODE is '01' (Finish Current Block Transfer) or

Workaround 2: Use DMA DEBUGMODE = '00' (Ignore suspend) when using TTYPE = Frame transfer to complete block transfer even after suspend/halt is asserted.

Either use TTYPE = Block transfer when DMA DEBUG MODE is '01' (Finish Current Block Transfer) or use DMA DEBUG MODE = '00' (Ignore suspend) when using TTYPE = Frame transfer to complete block transfer even after suspend/halt is asserted.

EMIF#3 *EMIF generates data abort on register read after time-out error*

Severity

3-Medium

Expected Behavior

The EMIF should not cause an abort when accessing EMIF registers.

Issue

After an EMIF time-out error when an external asynchronous memory fails to respond, a read to an EMIF register generates data abort.

Condition

1. The EMIF is used for asynchronous memory accesses in Extended Wait mode.

2. A time-out error occurs. For example, the memory does not de-assert the EMIF_nWAIT input.

3. The asynchronous memory access with time-out error is followed by an EMIF register read.

Implication(s)

Aborts will be generated on EMIF register reads until the "time-out" status is corrected by a successful EMIF region read.

Workaround(s)

If a timeout error occurs, complete a dummy read from the EMIF memory that does not return an error. This can be a synchronous read, a read from another asynchronous chip select that is not configured to be in Extended Wait mode, or to the same asynchronous chip select after disabling the Extended Wait mode on that chip select.

EMIF#4 — *Write to external asynchronous memory configured as “normal” causes extra WE pulses*

Severity	3-Medium
Expected Behavior	The number of WE pulses should match the correct number of writes required by the size of the data being written and the memory width configuration of the EMIF. For example, a 32-bit data written to a 16-bit wide memory should cause two write pulses.
Issue	One additional WE pulse is observed on the EMIF outputs. The byte enable signals (EMIF_nDQM) are not asserted for the extra write pulse. For example, the EMIF_nWE signal is asserted three times for a 32-bit write over a 16-bit interface.
Condition	1. MPU configuration for external asynchronous memory is normal. 2. Write to external asynchronous memory.
Implication(s)	An additional write could be performed if the external memory or FPGA does not use the byte-enable signals to actually perform the write. This could cause incorrect data written to external memory.
Workaround(s)	External asynchronous memory must be configured to be "device" type or "strongly-ordered" type using the CPU's MPU.

ERAY#52 (FLEXRAY#52) *Wakeup Symbol (WUS) Generates Redundant Wakeup Interrupts (SIR.WUPA/B)*

Severity 4-Low

Expected Behavior If a sequence of wakeup symbols (WUS) is received and all are separated by appropriate idle phases then a valid wakeup pattern (WUP) should be detected after *every second WUS.*

Issue The FlexRay module detects a valid wakeup pattern (WUP) after the second WUS and then after each following WUS.

Condition A sequence of wakeup symbols (WUS) is received, all separated by appropriate idle phases.

Implication(s) More SIR.WUPA/B events are seen than expected especially when an application program frequently resets the appropriate SIR.WUPA/B bits

Workaround(s) Ignore redundant SIR.WUPA/B events.

ERAY#58 (FLEXRAY#58) *Erroneous Cycle Offset During Startup after abort of startup or normal operation*

Severity 4-Low

Expected Behavior Correct cycle offset in spite of abort of startup or normal operation by a READY command.

Issue The state INITIALIZE_SCHEDULE may be one macrotick too short during an integration attempt. This leads to an early cycle start in state INTEGRATION_COLDSTART_CHECK or INTEGRATION_CONSISTENCY_CHECK.

Condition An abort of startup or normal operation by a READY command near the macrotick border. The issue is limited to applications where READY command is used to leave STARTUP, NORMAL_ACTIVE, or NORMAL_PASSIVE state

Implication(s) As a result the integrating node calculates a cycle offset of one macrotick at the end of the first even/odd cycle pair in the states INTEGRATION_COLDSTART_CHECK or INTEGRATION_CONSISTENCY_CHECK and tries to correct this offset.

If the node is able to correct the offset of one macrotick (pOffsetCorrectionOut >> gdMacrotick), the node enters NORMAL_ACTIVE with the first startup attempt.

If the node is not able to correct the offset error because pOffsetCorrectionOut is too small (pOffsetCorrectionOut <= gdMacrotick), the node enters ABORT_STARTUP and is ready to try startup again. The next (second) startup attempt is not affected by this erratum.

Workaround(s) With a configuration ofpOffsetCorrectionOut >> gdMacrotick*(1+cClockDeviationMax) the node will be able to correct the offset and therefore also be able to successfully integrate.

ERAY#59 (FLEXRAY#59) *First Wakeup Symbol (WUS) Following Received Valid Wakeup Pattern (WUP) May Be Ignored*

Severity 4-Low

Expected Behavior The FlexRay controller protocol engine should recognize all wakeup symbols (WUS).

Issue The FlexRay controller protocol engine may ignore the first wakeup symbol (WUS) following the below stated state transition, therefore it sets the wakeup status interrupt flags (SIR.WUPA/B) at the third WUS instead of the second WUS.

Condition The issue is limited to the reception of redundant wakeup patterns. When the protocol engine is in WAKEUP_LISTEN state and receives a valid wakeup pattern (WUP), it transfer into READY state and updates the wakeup status vector CCSV.WSV[2:0] as well as the status interrupt flags SIR.WST and SIR.WUPA/B.

Implication(s) Delayed setting of status interrupt flags SIR.WUPA/B for redundant wakeup patterns.

Workaround(s) None.

ERAY#60 (FLEXRAY#60) *READY Command Accepted In READY State*

Severity	4-Low
Expected Behavior	The FlexRay module should ignore a READY command while in READY state.
Issue	The FlexRay module does not ignore a READY command while in READY state.
Condition	The Protocol Operation Controller (POC) issues a READY command while in READY state.
Implication(s)	The coldstart inhibit bit CCSV.CSI is set whenever the POC enters READY state.
Workaround(s)	None.

ERAY#61 (FLEXRAY#61) *The Transmission Slot Mode Bit Is Reset Immediately When Entering HALT State*

Severity	4-Low
Expected Behavior	According to the FlexRay protocol specification, the slot mode should not be reset to SINGLE slot mode before the following state transition from HALT to DEFAULT_CONFIG state. The mode can be changed in DEFAULT_CONFIG or CONFIG state only.
Issue	Transmission slot mode bit is immediately reset to SINGLE slot mode (CCSV.SLM[1:0] = "00").
Condition	The protocol engine is in NORMAL_ACTIVE or NORMAL_PASSIVE state, and a HALT or FREEZE command is issued by the CPU
Implication(s)	The transmission slot mode is reset to SINGLE when entering HALT state.
Workaround(s)	None.

ERAY#68 (FLEXRAY#68) *Data transfer overrun for message transfers Message RAM to Output Buffer (OBF) or from Input Buffer (IBF) to Message RAM*

Severity

Medium

Expected Behavior

Data transfers should not overrun the expected receive buffer.

Issue

1) A message buffer transfer from Message RAM to OBF When the message buffer has its payload configured to maximum length (PLC = 127), the OBF word on address 00h (payload data bytes 0 to 3) is overwritten with unexpected data at the end of the transfer.

2) A message buffer transfer from IBF to Message RAM After the Data Section of the selected message buffer in the Message RAM has been written, one additional write access overwrites the following word in the Message RAM which might be the first word of the next Data Section

Conditions

The problem occurs under the following conditions:

1) A received message is transferred from the Transient Buffer RAM (TBF) to the message buffer that has its data pointer pointing to the first word of the Message RAM's Data Partition located directly after the last header word of the Header Partition of the Last Configured Buffer as defined by MRC.LCB.

2) The Host triggers a transfer from / to the Last Configured Buffer in the Message RAM with a specific time relation to the start of the TBF transfer described under 1).

Implications

1) When a message is transferred from the Last Configured Buffer in the Message RAM to the OBF and PLC = 127 it may happen, that at the end of the transfer the OBF word on address 00h (payload data bytes 0 to 3) is overwritten with unexpected data (see Figure 4).

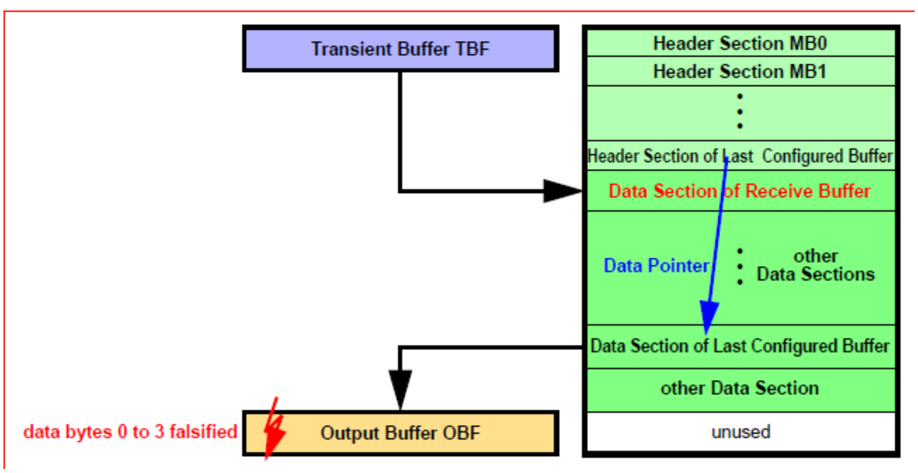

Figure 4. First Fail Mode

2) When a message is transferred from IBF to the Last Configured Buffer in the Message RAM, it may happen, that at the end of the transfer of the Data Section one additional write access overwrites the following word, which may be the first word of another message's Data Section in the Message RAM (see Figure 5).

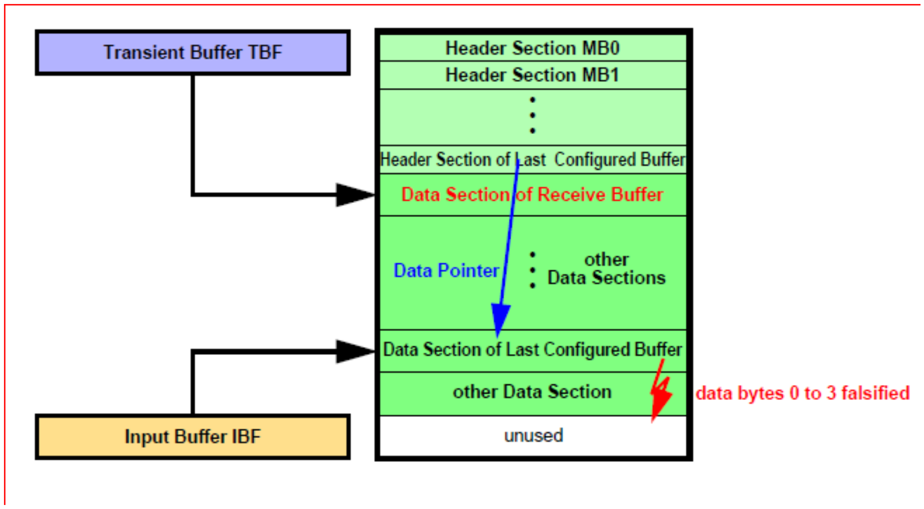

Figure 5. Second Fail Mode

Workaround(s)

1) Leave at least one unused word in the Message RAM between Header Section and Data Section.

OR

2) Ensure that the Data Section directly following the Header Partion is assigned to a transmit buffer.

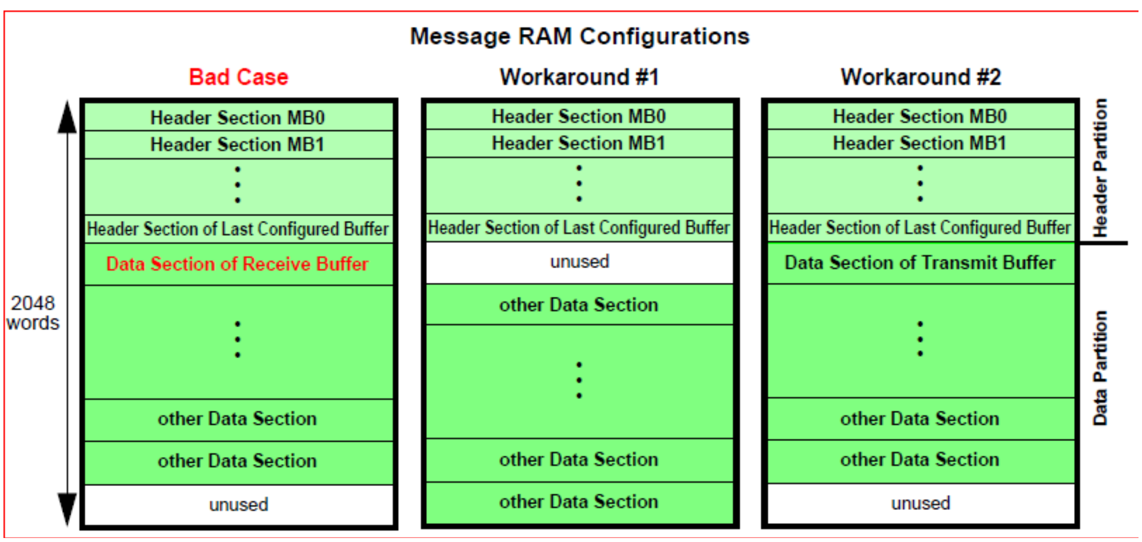

Figure 6. Workarounds

ERAY#69 (FLEXRAY#69) *Missing startup frame in cycle 0 at coldstart after FREEZE or READY command*

Severity

3-Medium

Expected Behavior

When a coldstart node re-enters startup, it listens to its attached channels and attempts to receive FlexRay frames. If no communication is received, the node commences a coldstart attempt which begins with the transmission of a collision avoidance symbol (CAS). Only the coldstart node that transmits the CAS transmits the startup frames in the first four cycles (from cycle 0 to cycle 3) after the CAS.

Issue

The FlexRay may not transmit its startup frame in the first cycle after CAS (cycle 0) when it is restarted as the leading coldstarter (was stopped by FREEZE or READY).

Condition

The issue is limited to the following condition:

1. FlexRay has been stopped by FREEZE or READY command.
2. FlexRay is configured with startup frames with 1 to 7 slots
3. A coldstart after hardware reset is not affected.

Implication(s)

Startup frame is not sent in cycle 0 after entering COLDSTART_COLLISION_RESOLUTION state from COLDSTART_LISTEN state.

Workaround(s)

Configure the FlexRay to use a static slot greater or equal 8 for the startup / sync message.

FMC#79 — *Abort on Unaligned Access at End of Bank*

Severity

4-Low

Expected Behavior

Since packed code and data can be linked to unaligned boundaries, the CPU should be able to read these locations in memory space independent of the flash bank boundaries.

Issue

The CPU will sometimes get an abort when making an unaligned access near the physical end of the bank boundary (in the range from 0xnnnnFFF1 through 0xnnnnFFFF). Examples of unaligned accesses capable of causing an abort:

Condition

This only occurs within the ATCM space. It only occurs when the flash is in single cycle mode and operating above 20MHz speed.

- a 32 bit data read such as a LDR at an address not on a 4 byte boundary

- a 16 bit data read such as a LDRH at an address not on a 2 byte boundary

- fetching a 32-bit thumb2 instruction which is not aligned on a four byte boundary

Implication(s)

An abort exception may be generated when accessing unaligned data or instructions in this range

Workaround(s)

Use an option to keep the compiler from generating unaligned data or instructions. For the TI compiler use --unaligned_access=off. Also ensure that hand generated assembly language routines do not create an unaligned access to these locations.

OR

Do not use single cycle mode (RWAIT=0) at frequencies above 20MHz.

OR

Reserve the last fifteen bytes of flash in each bank on the ATCM with either a dummy structure that is not accessed, or with a structure that will not create an unaligned access.

FTU#08 — *FlexRay Transfer Unit Not Disabled On Memory Protection Violation (MPV) Error*

Severity

3-Medium

Expected Behavior

On memory protection violation (MPV) errors the FTU should get disabled.

Issue

The FTU does not get disabled under some conditions.

Condition

If an MPV error occurs during the following transfer scenarios:

- During header transfer from system memory to FlexRay RAM, when FTU is configured to transfer header and payload
- During payload transfer from FlexRay RAM to system memory, when FTU is configured to transfer header and payload
- During a transfer from FlexRay RAM to system memory, when FTU is configured to transfer payload only

Implication(s)

The MPV error flag in the Transfer Error Interrupt Flag (TEIF) register is set, but the Transfer Unit Enabled (TUE) flag in the Global Control Set/Reset (GCS/R) register does not get cleared. As a result, the FTU does not get disabled.

Workaround(s)

This erratum can be avoided in the following ways:

- For transfers from system memory to FlexRay RAM, transfer the payload only
- Generate an MPV interrupt and clear the TUE flag in the Global Control Set/Reset (GCS/R) register in the interrupt service routine

FTU#19 *TCCOx Flag Clearing Masked*

Severity
4-Low

Expected Behavior
When TOOFF (Transfer Occurred Offset) is read, the corresponding message flag in TCCOx must be cleared.

Issue
In some conditions, the read of TOOFF register would not be up to date and would not reflect the last buffer completed.

Condition
There may be a timing condition when TCCOx flag clearing could be masked due to the state machine clearing of TTCCx (Trigger transfer to communication controller) within the same cycle as software reading TOOFF.

Implication(s)
The TCCOx flag is not being cleared.

Workaround(s)
After reading the TOOFF to determine the highest buffer completed, clear the corresponding flag in TCCOx.

GCM#59 *Oscillator can be disabled while PLL is running*

Severity 4-Low

Expected Behavior No clock source can be disabled if it is being used

Issue The oscillator can be disabled if the PLL is the only thing using it as a clock source

Condition The oscillator may be disabled if:

1. no clock domain relies upon the oscillator
2. no clock domain relies upon any PLL

Implication(s) This issue allows the oscillator to be disabled while used by the PLL. When the oscillator disables, the PLL will slip. The system behaves exactly like it would in case of a PLL slip. The response includes:

1. setting the RF SLIP flag (GBLSTAT.8)
2. switching Clock Source 1 from the PLL (if enabled). This autonomous switch prevents use of the PLL until the fault is cleared.
3. the device generates an ESM error (if enabled)
4. Cause a reset if the Reset-On-Slip Failure bit is set in PLLCTRL1.

If the software now uses the PLL as a clock source, there will be a long delay (mS) for the oscillator and the PLL to restart and provide a clock. Additionally, the SLIP flag(s) must be cleared in order for the PLL to propagate to the clock domains.

Normally this is not an issue as the software should not attempt to disable the oscillator when it is being used by the PLL. Also, once the PLL is stable and used as a clock source, the oscillator can no longer be disabled.

Workaround(s) Since the PLL is a secondary clock source dependent on the Oscillator input, the user software should not disable the Oscillator while the PLL is enabled while neither of them are sources for any of the clock domains.

MCRC#18 *CPU Abort Generated on Write to Implemented CRC Space After Write to Unimplemented CRC Space*

Severity
4-Low

Expected Behavior
A write to the legal address region (0xFE00_0000 to 0xFE00_01FF) of the CRC module should not generate an abort

Issue
An abort is generated on a write to a legal address region (0xFE000000-0xFE0001FF) of the CRC register space.

Condition
When a normal mode write to an unimplemented address region (0xFE00_0200 to 0xFE00_FFFF) of the CRC register space is followed by a write to a legal address region (0xFE00_0000 to 0xFE00_01FF) of the CRC register space.

Implication(s)
A write to an unimplemented address region of the CRC register space generates a data abort as expected. The next write to a legal address region of the CRC register space generates an unexpected second data abort.

Workaround(s)
None.

MIBSPI#110 — *Multibuffered SPI in Slave Mode In 3- or 4-Pin Communication Transmits Data Incorrectly for Slow SPICLK Frequencies and for Clock Phase = 1*

Severity

3-Medium

Expected Behavior

The SPI must be able to transmit and receive data correctly in slave mode as long as the SPICLK is slower than the maximum frequency specified in the device datasheet.

Issue

The MibSPI module, when configured in multi-buffered slave mode with 3 functional pins (CLK, SIMO, SOMI) or 4 functional pins (CLK, SIMO, SOMI, nENA), could transmit incorrect data.

Condition

This issue can occur under the following condition:

- Module is configured to be in multi-buffered mode, AND
- Module is configured to be a slave in the SPI communication, AND
- SPI communication is configured to be in 3-pin mode or 4-pin mode with nENA, AND
- Clock phase for SPICLK is 1, AND
- SPICLK frequency is VCLK frequency / 12 or slower

Implication(s)

Under the above described condition, the slave MibSPI module can transmit incorrect data.

Workaround(s)

The issue can be avoided by setting the CSHOLD bit in the control field of the TX RAM. The nCS is not used as a functional signal in this communication, hence setting the CSHOLD bit does not cause any other effect on the SPI communication.

MIBSPI#111 *Data Length Error Is Generated Repeatedly In Slave Mode when I/O Loopback is Enabled*

Severity 3-Medium

Expected Behavior After a data length (DLEN) error is generated and the interrupt is serviced the SPI should abort the ongoing transfer and stop.

Issue When a DLEN error is created in Slave mode of the SPI using nSCS pins in IO Loopback Test mode, the SPI module re-transmits the data with the DLEN error instead of aborting the ongoing transfer and stopping.

Condition This is only an issue for an IOLPBK mode Slave in Analog Loopback configuration, when the intentional error generation feature is triggered using CTRL_DLENERR(IOLPBKTSTCR.16).

Implication(s) The SPI will repeatedly transmit the data with the DLEN error when configured in the above configuration.

Workaround(s) After the DLEN_ERR interrupt is detected in IOLPBK mode, disable the transfers by clearing the SPIEN bit of SPIGCR1 register (bit 24) and then re-enable the transfers by setting SPIEN.

MIBSPI#137 *Spurious RX DMA REQ from a Slave mode MIBSPI*

Severity 4-Low

Expected Behavior The MIBSPI should not generate DMA requests when it has not received data from the SPI master

Issue A spurious DMA request is generated even when the SPI slave is not transferring data.

Condition This erratum is only valid when all below conditions are true:

- The MIBSPI is configured in standard (not multi-buffered) SPI mode as a slave.
- SPIINT0.16 (DMA_REQ_EN) bit is set to enable DMA requests.
- The nSCS (Chip Select) pin is in active state, but no transfers are active.
- The SPI is disabled by clearing SPIGCR1.24 (SPIEN) bit from '1' to '0'.

The above sequence triggers a false request pulse on the Receive DMA Request as soon as SPIEN bit is cleared from '1' to '0'.

Implication(s) The SPI generates a false DMA request to the DMA module when the data is not yet available for the DMA module to retrieve.

Workaround(s) Whenever the SPI is to be disabled by clearing SPIEN bit, clear the DMA_REQ_EN bit to '0' first and then clear the SPIEN bit.

MIBSPI#139 *Mibspi RX RAM RXEMPTY bit does not get cleared after reading*

Severity

3-Medium

Expected Behavior

The MibSPI RXEMPTY flag is auto-cleared after a CPU or DMA read.

Issue

Under a certain condition, the RXEMPTY flag is not auto-cleared after a CPU or DMA read.

Condition

The TXFULL flag of the latest buffer that the sequencer read out of transmit RAM for the currently active transfer group is 0, AND

A higher priority transfer group interrupts the current transfer group and the sequencer starts to read the first buffer of the new transfer group from the transmit RAM, AND

Simultaneously, the host (CPU/DMA) is reading out a receive RAM location that contains valid received data from the previous transfers.

Implication(s)

The fake RXEMPTY '1' suspends the next Mibspi transfer with BUFMODE 6 or 7.

With other BUFMODEs, a false "Receive data buffer overrun" will be reported for the next Mibspi transfer.

Workaround(s)

1. If at all possible, avoid transfer groups interrupting one another.

2. If dummy buffers are used in lower priority transfer group, select appropriate "BUFMODE" for them (like SKIP/DISABLED) unless there is a specific need to use the "SUSPEND" mode.

NHET#54 *PCNT incorrect when low phase is less than one loop resolution*

Severity
3-Medium

Expected Behavior
PCNT instruction can correctly capture a low going pulse width if the pulse width is greater than two high resolution clocks

Issue
PCNT instruction may capture incorrect low resolution clock (control field) and high resolution clock value

Condition
When measuring from falling edge to rising edge and the low pulse width is less than one low resolution clock width.

Implication(s)
PCNT cannot be used for capturing the pulse width of a low pulse less than one low resolution clock wide.

Workaround(s)
Connect the input pulse to be measured on two nHET channels using the high resolution share feature. Then use two WCAP instructions, one to measure the falling edge, the second to measure the rising edge. Use the CPU to calculate the time difference. In this workaround the period of the input signal must be two loop resolutions or longer.

NHET#55 *More than one PCNT instruction on the same pin results in measurement error*

Severity 3 - Medium

Expected Behavior It should be possible to use more than one Period/Pulse Count (PCNT) instruction to measure a single pin, as long as only one of the PCNT instructions is configured for high resolution (hr_lr=HIGH). For example, consider the following code fragments.

Code Fragment 1 - Should Be OK, But Fails Due to This Issue

```
PC1     PCNT { hr_lr=HIGH, type=RISE2FALL, pin=2};
PC2     PCNT { hr_lr=LOW,  type=FALL2FALL, pin=2};
```

Code Fragment 2 - Should Be OK, But Fails Due to This Issue

```
PC1     PCNT { hr_lr=LOW,  type=RISE2FALL, pin=2};
PC2     PCNT { hr_lr=HIGH, type=FALL2FALL, pin=2};
```

Code fragments 1 and 2 should work properly because only one of the two PCNT instructions are configured for hr_lr=HIGH, and there is one hi-res structure available.

Issue There are two issues.

1. A measurement error is introduced into the result of the PCNT instruction with hr_lr=HIGH. Normally this instruction would return a result to within ±½ high resolution clock periods of the actual result, due to quantization noise. However another PCNT instruction on the same pin causes an error of up to ±1 loop resolution period. Note that this error is greater than the normal loop resolution period error of ±½ loop resolution period; because the high-resolution bits also contribute to the error in this case.
2. A measurement error is introduced into the result of the PCNT instruction with hr_lr=LOW. The PCNT instruction with hr_lr=LOW should return a value with 0's in bit positions 6:0 (the high-resolution portion of the measurement result). This is the case when both PCNT instructions are set for hr_lr=LOW (Code Fragment 3) but for Code Fragments 1 and 2 the loop resolution PCNT returns a non-zero in bit positions 6:0.

Conditions This problem occurs when both conditions are true:

1. More than one PCNT selecting the same pin number is executed during the same loop resolution period.
2. One of the PCNT instructions is configured for high resolution (hr_lr=HIGH).

Please also note that the N2HET assembler defaults to high resolution for PCNT if the hr_lr field is not specified as part of the instruction. Therefore unless the instruction is coded explicitly with 'hr_lr=LOW as an option, the assembler will create N2HET machine code with hr_lr=HIGH.'

Implications The impact is greatest when workaround option 1 cannot be applied due to the number of timer pins required by the application. If Option 1 cannot be applied, then the PCNT measurements on this pin are reduced to ±½ loop resolution period.

Workaround(s) Option 1 - Use the HR Share feature and make both measurements with hr_lr=HIGH. First, set the appropriate HRSHARE bit in the HETHRSH register. In the following example this means setting HETHRSH bit 1 - "HRSHARE3/2". This bit causes the input of device pin 2 to drive the N2HET pin inputs 2 and 3. Then modify the N2HET code sequence to use pin 3 for one of the PCNT instructions:

Code Fragment 1 Modified for HR Share

```
PC1     PCNT { hr_lr=HIGH, type=RISE2FALL, pin=2};
PC2     PCNT { hr_lr=HIGH, type=FALL2FALL, pin=3};
```

This option exceeds the original measurement resolution objective because both PCNT measurements are made with high-resolution. The disadvantage of this workaround is that it requires the high-resolution structure of pin 3, leaving pin 3 only useable as a GPIO pin rather than as a timer pin.

Option 2 - Use only loop resolution mode PCNT instructions (as in Code Fragment 3). This will work properly while leaving pin 3 available for timing functions, but the resolution on both the period and duty cycle measurements are reduced to loop resolution.

Code Fragment 3 - OK

```
PC1    PCNT { hr_lr=LOW, type=RISE2FALL, pin=2};
PC2    PCNT { hr_lr=LOW, type=FALL2FALL, pin=2};
```

SSWF021#45 *PLL Fails to Start*

Severity 2-High

Expected Behavior When the PLL control registers are properly initialized and the appropriate clock source disable bit is cleared, after the prescribed number of OSCIN cycles, the PLL should be locked and the appropriate CSVSTAT bit should be set.

Issue On rare occasions the PLL does not start properly. The fail has one of three signatures:

1. CSVSTAT is set, but the ESM flag for PLL slip is set.
2. CSVSTAT is not set and the ESM flag for PLL slip is set.
3. CSVSTAT is set, the ESM flag for PLL slip is not set, but the PLL as measured by the DCC is not running.

Condition This issue applies to both PLLs (if the device has more than one PLL). This condition occurs only from a power-on. Once the PLL has locked, the PLL stays locked. Once properly locked, the PLL can be disabled and re-enabled with no issues.

Implication(s) If the PLL is used as the main clock source when it has not properly started, the CPU may stop executing instructions.

Workaround(s) While the main clock is being driven by the oscillator, the software loop checking that the PLL has locked (CSVSTAT = 1) should also check if the ESM flag for PLL slip has been set. When the CSVSTAT bit is set, the PLL frequency should be measured with the DCC before using the PLL as a clock source. If either the ESM flag for PLL slip is set, or the PLL has an incorrect frequency, the PLL should be disabled and the lock procedure should be repeated; TI recommends allowing a minimum of five attempts.

A more detailed explanation of the workaround with associated source code can be found in the application note:

Hercules PLL Advisory SSWF021#45 Workaround

STC#26 *The value programmed into the Self Test Controller (STC) Self-Test Run Timeout Counter Preload Register (STCTPR) is restored to its reset value at the end of each self test run.*

Severity 4-Low

Expected Behavior Once the Self-Test Run Timeout Counter Preload Register (STCTPR) is written, the value written into the register will be maintained until it is overwritten or a system or power on reset occurs and it will be used to preload the timeout counter for each self test run.

Issue The STCTPR is reset to the reset default value (0xFFFFFFFF) at the end of each CPU self test run and the value previously written to the STCTPR register is lost.

Condition Execution of any CPU self test with a STCTPR value other than the default value (0xFFFFFFFF).

Implication(s) Subsequent self test runs will use a maximum timeout value of 0xFFFFFFFF if not re-written to the desired value.

Workaround(s) The Timeout preload value in STCTPR register needs to be programmed to the required time out value before starting each self test if a timeout count other than 0xFFFFFFFF is desired.

STC#29 — *Inadvertent Performance Monitoring Unit (PMU) interrupt request generated if a system reset [internal or external] occurs while a CPU Self-Test is executing.*

Severity
4-Low

Expected Behavior
If an internal or external system reset is asserted the CPU should be reset cleanly with no inadvertent interrupt requests.

Issue
An unexpected PMU interrupt request may be generated.

Condition
This condition can occur when am internal or external system reset is asserted and the CPU is executing a CPU self test.

Implication(s)
The interrupt request signal from the performance monitoring unit (PMUIRQ) may inadvertently be set. This signal will generate an interrupt to the Vector Interrupt Module (VIM) and later become an interrupt to the CPU. Therefore, it is possible to see an unexpected interrupt after the CPU comes out of the system reset.

Workaround(s)
Clear VIM interrupt request 22 by writing 0x00400000 to location 0xFFFFFE20 before enabling this interrupt.

SYS#046 *Clock Source Switching Not Qualified With Clock Source Enable And Clock Source Valid*

Severity 4-Low

Expected Behavior An attempt to switch to a clock source which is not valid yet should be discarded.

Issue Switching a clock source by simply writing to the GHVSRC bits of the GHVSRC register may cause unexpected behavior. The clock will switch to a source even if the clock source was not ready.

Condition A clock domain that is programmed to take the clock source which is not yet valid as indicated by the CSVSTAT register.

Implication(s) Unexpected behavior stated above.

Workaround(s) Always check the CSDIS register to make sure the clock source is turned on and check the CSVSTAT register to make sure the clock source is valid. Then write to GHVSRC to switch the clock.

SYS#102 *Bit field EFUSE_Abort[4:0] in SYSTASR register is read-clear instead of write-clear*

Severity	3-Medium
Expected Behavior	The Technical Reference Manual states that EFUSE_Abort[4:0] of the SYSTASR register should be write-clear in privilege mode.
Issue	However, these bits are implemented as read-clear.
Condition	Always.
Implication(s)	Software implementation for error handling needs to take care of this as the subsequent reads of the register can return value of zero.
Workaround(s)	Avoid multiple read accesses of the SYSTASR register.
	None

VIM#27 — *Unexpected phantom interrupt*

Severity

2-High

Expected Behavior

When responding to an interrupt and a subsequent interrupt is received, the corresponding VIM request should be flagged as pending in the VIM status registers. When the CPU is ready to service the subsequent interrupt, the correct service routine address should be fetched by the CPU.

Issue

On rare occasions the VIM may return the phantom interrupt vector instead of the real interrupt vector.

Condition

This condition is specific to software and hardware vectored modes. This is not applicable for legacy interrupt servicing mode. This condition occurs when the ratio of GCLK to VCLK is 3:1 or greater for hardware vectored mode, or the ratio of GCLK to VCLK is 5:1 or greater for software vectored mode. A subsequent interrupt request must occur when the VIM is finishing acknowledging a previous interrupt.

Implication(s)

The subsequent interrupt request vectors to the phantom interrupt routine instead of the correct service routine.

Workaround(s)

The issue can be completely avoided if the GCLK:VCLK ratio is configured as 1:1 or 2:1.

For other VCLK ratios, the phantom interrupt handler simply needs to exit as normal, without taking any special steps. If this issue is present, the VIM will interrupt the CPU again, providing the correct vector.

5 Revision History

This silicon errata revision history highlights the technical changes made from the previous to the current revision.

Table 3. Revision History from Document Revision B to Document Revision C

Advisory Changes in Advisory List	Advisory ID
Added advisory(s)	SSWF021#45
Removed advisory(s)	None
Modified advisory(s)	None
Other	None