TMS570LS12x/11x Microcontroller

Silicon Revision B

Silicon Errata

Literature Number: SPNZ199A
February 2013–Revised August 2013

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.

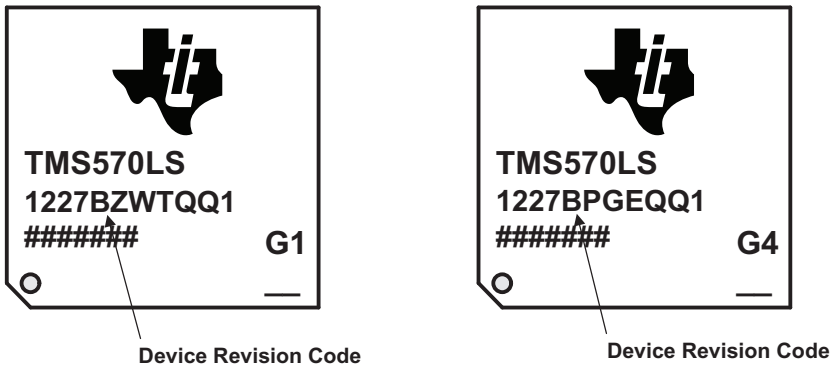

Figure 1. Device Revision Code Identification

Silicon revision is identified by a device revision code. The code is of the format TMS570LS1203x, where "x" denotes the silicon revision. If x is "A" in the device part number, it represents silicon version A. Table 1 lists the information associated with each silicon revision.

Table 1. Device Revision Codes

DEVICE PART NUMBER DEVICE REVISION CODE	SILICON REVISION	PART NUMBERS/COMMENTS
TMS570LS12x/11x	B	TMS570LS12x/11x

3 Revision History

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

Table 2. Revision History from Initial Errata Document Revison to Revision A

Advisory Changes in Advisory List	Advisory ID
Added advisory(s)	DEVICE#B063, DEVICE#B064
Removed advisory(s)	None
Modified advisory(s)	None
Other	None

4 Known Design Exceptions to Functional Specifications

The following table lists the known exceptions to the functional specifications for the device.

Table 3. Known Design Exceptions to Functional Specifications

DEVICE#B063 *Unexpected PSCON Compare Error*

Severity 3 - Medium

Expected Behavior No compare errors when disabling a logic power domain

Issue The device may generate a PSCON compare error when disabling a logic power domain.

Conditions This problem might occur if:

1. A logic power domain is disabled at reset by a factory OTP setting
2. Software explicitly disables a power domain.

Implications ESM group 1 channel 38 and channel 39 errors may be generated.

Workaround(s) Before enabling the effect of ESM group 1 channels 38 or 39

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 to the PDCLK_DISx register to disable all clocks to the power domain
   - Write 0xA to the LOGICPDONx register to power down the domain
   - Poll for LOGICPDPWRSTATx to become 00. The power domain is now powered down (Allows for delay time of PowerGood signal)
2. Clear any PSCON compare error flags
   - 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

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

DEVICE#B064 *Incorrect Write to External Memory on EMIF*

Severity	3-Medium
Expected Behavior	CPU and DMA writes to the external memory on the EMIF work as expected
Issue	Sometimes data is not written or written to the wrong location.
Condition	The first destination address of a multi-word write (STM, STRD, POP or VPOP) is not aligned to a 64-bit boundary and one of the following three conditons: 1. Back to back multi-word writes to external memory by the CPU when that memory is configured as "device" memory or 2. A multi-word write to external memory by the CPU and a DMA write at the same time or 3. A STR write to external memory by the CPU after back to back multi-word writes to the external memory
Implication(s)	In the first two cases, some data is not written. In the third case, data is written to the wrong address
Workaround(s)	Configure the external memory as "device" mode and align all STM writes to a 64 bit boundary. Use the STR instruction if the start address is not aligned to a 64 bit boundary.

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

Severity	Low
Expected Behavior	A write to the illegal address region of the MCRC module will generate an abort
Issue	A CPU Abort does not get generated per the following condition:
Conditions	When a normal mode write to an illegal address region of MCRC register space is followed by a debug mode access.
Implications	When debugging, either a breakpoint on the instruction after the illegal write, or single stepping through the illegal write will not generate an abort.
Workaround(s)	None

DEVICE#179 *CPU may Hang on STM Write to Memory on EMIF*

Severity	3-Medium
Expected Behavior	CPU completes STM (store multiple) instruction
Issue	The CPU may hang on a STM instruction
Condition	A STM instruction writes to an address in the EMIF followed by a write to a peripheral address mapped as strongly ordered OR A STM instruction writes to an address in the EMIF followed by another STM write to a peripheral which is mapped as "Strongly Ordered".
Implication(s)	The CPU hangs waiting for the memory write to complete.
Workaround(s)	Do not map either the EMIF address space or any peripheral as "Strongly Ordered" when writing to the EMIF using the STM instruction. (burst mode)

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.

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 erratum, the Cortex-R4 processor treats such an encoding as UNPREDICTABLE and executes it as a NOP.

Condition This occurs when the processor is in Thumb state and a STREXD instruction is executed which has Rt = Rt2. This instruction was new in ARM Architecture version 7 (ARMv7). It is not present in ARMv6T2 or other earlier architecture versions.

Implication(s) If this 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 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.

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

- 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)
- The processor is in Standby Mode, and the clocks have been gated off, that is STANDBYWFI is asserted
- The debug reset, PRESETDBGn, is asserted and deasserted while the processor clocks remain gated off

Implication(s) If this 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, or both modes can be disabled. Bits [15:14] in the Debug Status and Control Register (DBGDSCR) control which, if any, mode is enabled. Debug events can be generated by the breakpoint unit, matching instructions executed. Deadlocks should not occur when the debug mode is changed.

Issue If there are active breakpoints at the time when the debugging mode is changed, this can cause the processor to deadlock.

Condition

- At least one breakpoint is programmed and active
- Either halt-mode debugging or monitor mode debugging is enabled and the debug enable input, DBGEN is HIGH
- An instruction which matches a breakpoint is fetched and executed
- After the instruction is fetched, but before it has completed execution, both halt-mode and monitor-mode debugging are disabled by means of a write to DBGDSCR

Implication(s) Changing the debugging mode while breakpoints or watchpoints are active is UNPREDICTABLE. Therefore the above conditions cannot be met in normal (recommended) operation of the processor. However, if the conditions do occur, then the processor will deadlock, which is outside the bounds of permitted UNPREDICTABLE behavior.

Workaround(s) None.

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.

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)

If software attempts to read the AIDR as part of its process of identifying the processor on which it is running, it will encounter an unexpected UNDEFINED exception. If software attempts to execute an instruction prefetch using the ARMv6 CP15 prefetch operation, it will encounter an unexpected UNDEFINED exception.

Workaround(s)

In the first case, because the contents of the AIDR are IMPLEMENTATION DEFINED, it must always be read in conjunction with the Main ID Register (MIDR). A simple way of avoiding this would be to read and analyze the MIDR first and, if this indicates that the processor is Cortex-R4 skip the read of the AIDR. In the second case, any instruction to prefetch an instruction by MVA should be removed or replaced by a true NOP instruction.

CORTEX-R4#55 (ARM ID-722412) *CPACR.ASEDIS And CPACR.D32DIS Incorrect When Configured With Floating Point*

Severity 3-Medium

Expected Behavior In implementations of the VFPv3-D16 architecture that do not also implement the Advanced SIMD functionality, the ASEDIS and D32DIS bits, (bits [31] and [30] respectively of the Coprocessor Access Control Register (CPACR)) are read-as-one/writes ignored (RAO/WI). This indicates that Advanced SIMD functionality and registers D16-D31 are disabled.

Issue Because of this erratum, these bits read zero in implementations of Cortex- R4F which include the floating-point unit. When the floating point unit is not included, all bits of the CPACR correctly read-as-zero (RAZ).

Condition In an implementation of Cortex-R4F with the floating-point unit included, the CPACR is read.

Implication(s) If software uses the CPACR to determine if Advanced SIMD functionality and registers D16-D31 are available, it will incorrectly determine that they are. Any attempt to use these features will lead to an unexpected UNDEFINED exception.

Workaround(s) Because the Cortex-R4F processor never includes Advanced SIMD functionality or registers D16-D31, this can be correctly determined by reading the part number from one of the ID registers rather than examining ASEDIS and D32DIS in the CPACR.

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 erratum requires the following sequence of instructions in ARMstate: - VMRS<c> APSR_nzcv, FPSCR (formerly FMSTAT<c>), where the condition on the instruction is not always.

- A flag-setting integer multiply or multiply and accumulate instruction (e.g. MULS)
- 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 erratum 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)

This erratum 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 recognised 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 erratum, 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

- The processor is not in debug halt state
- The DBGDSCR.DBGack bit is set
- 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) If this erratum 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 erratum, 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:

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

OR

- The debugger reads DBGDSCRext which shows that TXfull is one, that is, DBGDTRTX is full.
- The debugger reads data from DBGDTRTXext,
- The processor writes new data into DBGDTRTXint,
- 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 erratum only has implications for debuggers that violate this requirement. If the erratum 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 erratum to occur:

- A UDIV or SDIV instruction is executed with out-of-order completion of divides enabled
- 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
- A load multiple instruction is executed as the first instruction of the exception handler
- The load multiple instruction itself is interrupted either by an IRQ, FIQ, imprecise abort or external debug halt request. This erratum 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 erratum has any implications for practical systems. If the situation of the erratum 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 erratum, 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 erratum, certain watchpoint hits on multiples will not generate a debug event.

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

- A load or store multiple instruction is executed with at least 5 registers in the register list
- The address range accessed corresponds to Strongly-Ordered or Device memory
- 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 erratum, 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 erratum will not occur.

Workaround(s) None.

AHB_ACCES_PORT#3 (ARM ID-529470) *DAP AHB-AP Access Port Might Fail To Return Slave Error On AHB Reset.*

Severity 3-Medium

Expected Behavior If a system reset (nRST goes low) occurs while the AHB-AP is performing an access on the AHB bus, the AHB-AP should return a slave error back to the JTAG-DP.

Issue Instead, the AHB-AP 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 AHB-AP is completing an access on the AHB bus. This is not an issue for a CPU only reset. This is not an issue during power-on reset (nPORRST) as the DAP will also be reset.

Implication(s) Data read using the DAP 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.

DMA#27 *DMA Requests Lost During Suspend Mode*

Severity 3-Medium

Expected Behavior All DMA requests will be held while the device is in suspend mode.

Issue DMA requests from a peripheral 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 DMA DEBUG MODE 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) 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 Reports Incorrect Time-out Error on Register Read*

Severity 3-Medium

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

Issue After an abort because an external asynchronous memory failed to respond, a read to an EMIF register will also generate an 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, perform a dummy read from the EMIF region 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.

ERAY#52 (FLEXRAY#52) *WUS Generates Redundant SIR.WUPA/B Events*

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	Instead, the FlexRay module detects a valid wakeup pattern 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 inspite 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 erratum 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 WUS Following Received Valid WUP May Be Ignored*

Severity	4-Low
Expected Behavior	The FlexRay controller protocol engine should recognize all wakeup symbols (WUS).
Issue	However, the protocol engine may ignore the first wakeup symbol (WUS) following the below stated state transition therefore signalling the next SIR.WUPA/B at the third WUS instead of the second WUS.
Condition	The erratum is limited to the reception of redundant wakeup patterns. When the protocol engine is in state WAKEUP_LISTEN and receives a valid wakeup pattern (WUP), it transfers into state READY 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	However, the FlexRay module does not ignore a READY command while in READY state.
Condition	The erratum is limited to the READY state.
Implication(s)	Flag CCSV.CSI is set. Cold starting needs to be enabled by the Protocol Operation Controller (POC) command ALLOW_COLDSTART (SUCC1.CMD = "1001").
Workaround(s)	None.

ERAY#61 (FLEXRAY#61) *Slot Status vPOC!SlotMode 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.
Issue	However, when the protocol engine is in NORMAL_ACTIVE or NORMAL_PASSIVE state, a HALT or FREEZE command issued by the CPU resets vPOC!SlotMode immediately to SINGLE slot mode (CCSV.SLM[1:0] = "00").
Condition	The erratum is limited to the HALT state.
Implication(s)	The slot status vPOC!SlotMode is reset to SINGLE when entering HALT state.
Workaround(s)	None.

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

Severity 4-Low

Expected Behavior The Cortex R4 can make unaligned accesses to any memory within the ATCM space (program flash space).

Issue The CortexR4 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). The unaligned access can be from a data read such as a LDR or an LDRH instruction, or from fetching a 32-bit thumb2 instruction which is not aligned on a four byte boundary.

Condition This only occurs in the program flash on the ATCM. It only occurs when the flash is in single cycle mode and operating above 20MHz speed. This is more likely to occur with high Vcc-core and at high temperature.

Implication(s) The CPU could take either an instruction abort or a data abort.

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#8 *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 (TIEF) register is set, but the 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 errata 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 According to the documentation, 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.q

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.

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

Severity	4-Low
Expected Behavior	A write to the legal address region of the MCRC module should not generate an abort
Issue	An Unexpected CPU Data Abort is generated in the following condition:
Condition	When a normal mode write to an illegal address region of MCRC register space is followed by a write to a legal address region of the MCRC register space.
Implication(s)	A write to an illegal address region of the MCRC register space generates a data abort as expected. The next write to a legal address region of the MCRC register space generates an unexpected second data abort.
Workaround(s)	None.

MIBSPI#110 *Multibuffer Slave In 3 or 4 Pin Mode 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 cause incorrect data to be transmitted.

Condition This issue can occur under the following condition:

- Module is configured to be in mult-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 a 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 unexpected DMA requests when it did not receive data from the master

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

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

- The MIBSPI is configured in compatible mode (SPI) 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.

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.

STC#26 *STCTPR is Reset at the End of Each Self Test*

Severity	4-Low
Expected Behavior	Once the STC Timeout-counter Preload Register (STCTPR) is written to, this value will be used to preload the timeout-counter for each self test run.
Issue	The STCTPR is reset at the end of each CPU self test run.
Condition	None
Implication(s)	Subsequent self test runs will use a maximum timeout value of 0xFFFFFFFF
Workaround(s)	The Timeout preload value in STCTPR register needs to be programmed to the required time out value before starting a self test every time if a timeout count of other than 0xFFFFFFFF is desired.

SYS#46 *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	The clock source is enabled (CSDIS register) but the clock source is not yet valid (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*

Severity	3-Medium
Expected Behavior	The documentation 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.
Workaround(s)	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) This issue can be avoided if the clock ratio GCLK:VCLK is 1:1 or 2:1. Otherwise the VIM status register can be checked within the phantom interrupt routine to determine the source of the subsequent interrupt.

IMPORTANT NOTICE

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

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

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

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

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

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

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

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

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

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

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

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

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