# SPC564Bxx, SPC56ECxx Errata sheet

## SPC564Bxx, SPC56ECxx devices errata JTAG_ID = 0x0AE4A041

## Introduction

This errata sheet describes all the functional and electrical problems known in the revision of SPC564Bxx, SPC56ECxx cut 3.0 and describes the functional and electrical deviations to device documentation (RM0070 Rev 6 and SPC564Bxx, SPC56ECxx datasheet Rev 9 (see *Appendix A: Reference documents*).

Device identification:

- JTAG_ID = 0x0AE4A041
- MCU ID Register 1
  - MAJOR_MASK: 02
  - MINOR_MASK: 00
- Package device marking mask identifier: BA

This errata sheet applies to SPC564Bxx, SPC56ECxx devices in accordance with *Table 1*.

**Table 1. Device summary**

| Package | Part number | | |
|---|---|---|---|
| | 1.5 MByte | 2 MByte | 3 MByte |
| LQFP176 | SPC564B64L7<br>SPC56EC64L7 | SPC564B70L7<br>SPC56EC70L7 | SPC564B74L7<br>SPC56EC74L7 |
| LQFP208 | SPC564B64L8<br>SPC56EC64L8 | SPC564B70L8<br>SPC56EC70L8 | SPC564B74L8<br>SPC56EC74L8 |
| BGA256 | SPC56EC64B3 | SPC56EC70B3 | SPC56EC74B3 |

# Contents

# 1 Functional problems

## 1.1 ERR003476: MC_RGM: Clearing a flag at RGM_DES or RGM_FES register may be prevented by a reset

**Description:**

Clearing a flag at RGM_DES and RGM_FES registers requires two clock cycles because of a synchronization mechanism. As a consequence if a reset occurs while clearing is ongoing the reset may interrupt the clearing mechanism leaving the flag set. Note that this failed clearing has no impact on further flag clearing requests.

**Workaround:**

No workaround for all reset sources except SW reset. Note that in case the application requests a SW reset immediately after clearing a flag in RGM_xES the same issue may occur. To avoid this effect the application must ensure that flag clearing has completed by reading the RGM_xES register before the SW reset is requested.

## 1.2 ERR003512: ECSM: ECSM_PFEDR displays incorrect endianness

**Description:**

The ECSM_PFEDR register reports ECC data using incorrect endiannes. For example, a flash location that contains the data 0xAABBCCDD would be reported as 0xDDCCBBAA at ECSM_PFEDR. This 32-bit register contains the data associated with the faulting access of the last, properly-enabled flash ECC event. The register contains the data value taken directly from the data bus.

**Workaround:**

Software must correct endianness.

## 1.3 ERR003556: DMA_MUX: Low Power Entry may not be completed when peripherals run on divided clock with DMA enabled mode

**Description:**

System may not enter into Low Power Mode (HALT/STOP/STANDBY) when all the below conditions are true simultaneously:

1. A Peripheral with DMA capability is programmed to work on divided clock.
2. Above peripheral is programmed to be stopped in Low Power Mode and active in RUN Mode.
3. Above Peripheral is active with DMA transfer while Software requests change to Low Power mode.

**Workaround:**

Software should ensure that all the DMA enabled peripherals have completed their transfer before requesting Low Power mode Entry.

## 1.4 ERR003570: MC_ME: Possibility of Machine Check on Low-Power Mode Exit

**Description:**

When executing from the flash and entering a Low-Power Mode (LPM) where the flash is in low-power or power-down mode, 2-4 clock cycles exist at the beginning of the RUNx to LPM transition during which a wakeup or interrupt will generate a checkstop due to the flash not being available on RUNx mode re-entry. This will cause either a checkstop reset or machine check interrupt.

**Workaround:**

If the application must avoid the reset, two workarounds are suggested:

1. Configure the application to handle the machine check interrupt in RAM dealing with the problem only if it occurs
2. Configure the MCU to avoid the machine check interrupt, executing the transition into low power modes in RAM

There is no absolute requirement to work around the possibility of a checkstop reset if the application can accept the reset, and associated delays, and continue. In this event, the WKPU.WISR will not indicate the channel that triggered the wakeup though the F_CHKSTOP flag will indicate that the reset has occurred. The F_CHKSTOP flag could still be caused by other error conditions so the startup strategy from this condition should be considered alongside any pre-existing strategy for recovering from an F_CHKSTOP condition.

## 1.5 ERR003697: e200z: Exceptions generated on speculative prefetch

**Description:**

The e200z4 core can prefetch up to 2 cache lines (64 bytes total) beyond the current instruction execution point. If a bus error occurs when reading any of these prefetch locations, the machine check exception will be taken. For example, executing code within the last 64 bytes of a memory region such as internal SRAM or FLASH, may cause a bus error when the core prefetches past the end of memory. An ECC exception can occur if the core prefetches locations that are valid, but not yet initialized for ECC.

**Workaround:**

Do not place code to be executed within the last 64 bytes of a memory region. When executing code from internal ECC SRAM, initialize memory beyond the end of the code until the next 32-byte aligned address and then an additional 64 bytes to ensure that prefetches cannot land in uninitialized SRAM.

## 1.6 ERR004146: When an ADC conversion is injected, the aborted channel is not restored under certain conditions

**Description:**

When triggered conversions interrupt the ADC, it is possible that the aborted conversion does not get restored to the ADC and is not converted during the chain. Vulnerable configurations are:

- Injected chain over a normal chain
- CTU trigger over a normal chain
- CTU trigger over an injected chain

When any of these triggers arrive whilst the ADC is in the conversion stage of the sample and conversion, the sample is discarded and is not restored. This means that the channel data register will not show the channel as being valid and the CEOCFRx field will not indicate a pending conversion. The sample that was aborted is lost.

When the trigger arrives during the final channel in a normal or injected chain, the same failure mode can cause two ECH/JECH interrupts to be raised.

If the trigger arrives during the sampling phase of the last channel in the chain, an ECH is triggered immediately, the trigger is processed and the channel is restored and after sampling/conversion, a second ECH interrupt occurs.

In scan mode, the second ECH does not occur if the trigger arrives during the conversion phase. In one-shot mode, the trigger arriving during the conversion phase of the last channel restarts the whole conversion chain and the next ECH occurs at completion of that chain.

**Workaround:**

It is suggested that the application check for valid data using the CDR status bits or the CEOCFRx registers to ensure all expected channels have converted. This can be tested by running a bitwise AND and an XOR with either the JCMRx or NCMRx registers and the CEOCFRx registers during the ECH of JECH handler. Any non-zero value for (xCMRx & (xCMRx CEOCFRx)) indicates that a channel has been missed and conversion should be requested again.

Spurious ECH/JECH interrupts can be detected by checking the NSTART/JSTART flags in the ADC Module Status Registers – if the flag remains set during an ECH/JECH interrupt then another interrupt will follow after the restored channel or chain has been sampled and converted.

The spurious ECH/JECH workaround above applies to single-shot conversions. In single-shot mode, NSTART changes from 1 to 0. Therefore, the user can rely on checking the NSTART bit to confirm if a spurious ECH has occurred. However, for scan mode, the NSTART bit will remain set during normal operation, so it cannot be relied upon to check for the spurious ECH/JECH issue. Consequently, if CTU is being used in trigger mode, the conversions must be single-shot and not scan mode.

## 1.7 ERR004168: ADC: "Abort switch" aborts the ongoing injected channel as well as the upcoming normal channel

**Description:**

If an Injected chain (jch1,jch2,jch3) is injected over a Normal chain (nch1,nch2,nch3,nch4) the Abort switch does not behave as expected.

Expected behavior:

- Correct Case (without SW Abort on jch3): Nch1-> Nch2(aborted)->Jch1 -> Jch2 -> Jch3 ->Nch2(restored) -> Nch3->Nch4
- Correct Case (with SW Abort on jch3): Nch1-> Nch2(aborted)->Jch1 -> Jch2 ->Jch3(aborted) ->Nch2(restored) -> Nch3->Nch4

Observed unexpected behavior:

- Fault1 (without SW abort on jch3): Nch1-> Nch2(aborted) -> Jch1 -> Jch2 -> Jch3 -> Nch3 -> Nch4 (Nch2 not restored)
- Fault2 (with SW abort on jch3): Nch1-> Nch2 (aborted)->Jch1 -> Jch2 -> Jch3(aborted) -> Nch4 (Nch2 not restored & Nch3 conversion skipped)

**Workaround:**

It is possible to detect the unexpected behavior by using the CEOCFRx register. The CEOCFRx fields will not be set for a not restored or skipped channel, which indicates this issue has occurred. The CEOCFRx fields need to be checked before the next Normal chain execution (in scan mode). The CEOCFRx fields should be read by every ECH interrupt at the end of every chain execution.

## 1.8 ERR004186: ADC: triggering an ABORT or ABORTCHAIN before the conversion starts

**Description:**

When ABORTCHAIN is programmed and an injected chain conversion is programmed afterwards, the injected chain is aborted, but neither JECH is set, nor ABORTCHAIN is reset. In case a CTU conversion is demanded, the CTU conversion is aborted by the ADC digital logic but the CTU awaits ADC analogue acknowledgment that never arrives, stopping all CTU operation until next reset.

When ABORT is programmed and normal/injected chain conversion comes afterwards, the ABORT bit is reset and chain conversion runs without a channel abort.

If ABORT, or ABORTCHAIN, feature is programmed after the start of the chain conversion, it works properly.

**Workaround:**

Do not program ABORT/ABORTCHAIN before starting the execution of the chain conversion.

If the CTU is being used in trigger mode, there will always be a small vulnerable window preventing reliably reading the ADC status and using ADC.MCR[ABORT] or ADC.MCR[ABORTCHAIN]. If these functions are required, disable all CTU triggers to that ADC first and do not start the CTU if the ABORT or ABORTCHAIN flags have been set whilst the ADC was IDLE. If the CTU cannot be disabled, an optimized ABORT should be

implemented in software that de-configures further channel conversions using the ADC. NMCRx registers before waiting for ADC.MSR[NSTART] == 0. At this point, the ADC should be IDLE and the NMCRx registers can be reinstated for next use.

## 1.9 ERR004340: LINFlexD: Buffer overrun can not be detected in UART Rx FIFO mode

**Description:**

When the LINFlexD is configured in UART Receive (Rx) FIFO mode, the Buffer Overrun Flag (BOF) bit of the UART Mode Status Register (UARTSR) register is cleared in the subsequent clock cycle after being asserted.

User software can not poll the BOF to detect an overflow.

The LINFlexD Error Combined Interrupt can still be triggered by the buffer overrun. This interrupt is enabled by setting the Buffer Overrun Error Interrupt Enable (BOIE) bit in the LIN Interrupt enable register (LINIER). However, the BOF bit will be cleared when the interrupt routine is entered, preventing the user from identifying the source of error.

**Workaround:**

Buffer overrun errors in UART FIFO mode can be detected by enabling only the Buffer Overrun Interrupt Enable (BOIE) in the LIN interrupt enable register (LINIER).

## 1.10 ERR005569: ADC: The channel sequence order will be corrupted when a new normal conversion chain is started prior to completion of a pending normal conversion chain

**Description:**

If One shot mode is configured in the Main Configuration Register (MCR[MODE] = 0) the chained channels are automatically enabled in the Normal Conversion Mask Register 0 (NMCR0). If the programmer initiates a new chain normal conversion, by setting MCR[NSTART] = 0x1, before the previous chain conversion finishes, the new chained normal conversion will not follow the requested sequence of converted channels.

For example, if a chained normal conversion sequence includes three channels in following sequence: channel0, channel1 and channel2, the conversion sequence is started by MCR[NSTART] = 0x1. The software re-starts the next conversion sequence when MCR[NSTART] is set to 0x1 just before the current conversion sequence finishes.

The conversion sequence should be: channel0, channel1, channel2, channel0, channel1, channel2.

However, the conversion sequence observed will be: channel0, channel1, channel2, channel1, channel1, channel2. Channel0 is replaced by channel1 in the second chain conversion and channel1 is converted twice.

**Workaround:**

Ensure a new conversion sequence is not started when a current conversion is ongoing. This can be ensured by issuing the new conversion setting MCR[NSTART] only when MSR[NSTART] = 0.

*Note: MSR[NSTART] indicates the present status of conversion. MSR[NSTART] = 1 means that a conversion is ongoing and MSR[NSTART] = 0 means that the previous conversion is finished.*

## 1.11 ERR006026: DSPI: Incorrect SPI Frame Generated in Combined Serial Interface Configuration

**Description:**

In the Combined Serial Interface (CSI) configuration of the Deserial Serial Peripheral Interface (DSPI) where data frames are periodically being sent (Deserial Serial Interface, DSI), a Serial Peripheral Interface (SPI) frame may be transmitted with incorrect framing. The incorrect frame may occur in this configuration if the user application writes SPI data to the DSPI Push TX FIFO Register (DSPI_PUSHR) during the last two peripheral clock cycles of the Delay-after-Transfer (DT) phase. In this case, the SPI frame is corrupted.

**Workaround:**

1. Workaround 1: Perform SPI FIFO writes after halting the DSPI.
   To prevent writing to the FIFO during the last two clock cycles of DT, perform the following steps every time a SPI frame is required to be transmitted:
   Step 1: Halt the DSPI by setting the HALT control bit in the Module Configuration Register (DSPI_MCR[HALT]).
   Step 2: Poll the Status Register's Transmit and Receive Status bit (DSPI_SR[TXRXS]) to ensure the DSPI has entered the HALT state and completed any in-progress transmission.
   Alternatively, if continuous polling is undesirable in the application, wait for a fixed time interval such as 35 baud clocks to ensure completion of any in-progress transmission and then check once for DSPI_SR[TXRXS].
   Step 3: Perform the write to DSPI_PUSHR for the SPI frame.
   Step 4: Clear bit DSPI_MCR[HALT] to bring the DSPI out of the HALT state and return to normal operation.
2. Workaround 2: Do not use the CSI configuration. Use the DSPI in either DSI-only mode or SPI-only mode.
3. Workaround 3: Use the DSPI's Transfer Complete Flag (TCF) interrupt to reduce worst-case wait time of Workaround 1.
   Step 1: When a SPI frame is required to be sent, halt the DSPI as in Step 1 of Workaround 1 above.
   Step 2: Enable the TCF interrupt by setting the DSPI DMA/Interrupt Request Select and Enable Register's Transmission Complete Request Enable bit (DSPI_RSER[TCF_RE])
   Step 3: In the TCF interrupt service routine, clear the interrupt status (DSPI_SR[TCF]) and the interrupt request enable (DSPI_RSER[TCF_RE]). Confirm that DSPI is halted by checking DSPI_SR[TXRXS] and then write data to DSPI_PUSHR for the SPI frame. Finally, clear bit DSPI_MCR[HALT] to bring the DSPI out of the HALT state and return to normal operation.

## 1.12 ERR006082: LINFlexD: LINS bits in LIN Status Register(LINSR) are not usable in UART mode

**Description:**

When the LINFlexD module is used in the Universal Asynchronous Receiver/Transmitter (UART) mode, the LIN state bits (LINS3:0]) in LIN Status Register (LINSR) always indicate the value zero. Therefore, these bits cannot be used to monitor the UART state.

**Workaround:**

LINS bits should be used only in LIN mode.

## 1.13 ERR006481: NZ4C3/NZ7C3: Erroneous Resource Full Message under certain conditions

**Description:**

The e200zx core Nexus interface may transmit an erroneous Resource Full Message (RFM)

following a Program Trace history message with a full history buffer. The History information for both of the messages are the same and the RFM should have not been transmitted. This occurs when the instruction following the indirect branch instruction (which triggers the Program Trace History message) would have incremented the History field. The instructions must be executed in back to back cycles for this problem to occur. This is the only case that cases this condition.

**Workaround:**

There are three possible workarounds for this issue:

1. Tools can check to see if the Program Trace History message and proceeding Resource Full Message (RFM) have the same history information. If the history is the same, then the RFM is extraneous and can be ignored.
2. Code can be rewritten to avoid the History Resource Full Messages at certain parts of the code. Insert 2 NOP instructions between problematic code. Or inserting an "isync" or a indirect branch some where in the code sequence to breakup/change the flow.
3. If possible, use Traditional Program Trace mode can be used to avoid the issue completely. However, depending on other conditions (Nexus port width, Nexus Port speed, and other enabled trace types), overflows of the port could occur.

## 1.14 ERR006620: FLASH: ECC error reporting is disabled for Address Pipelining Control (APC) field greater than Read Wait-State Control (RWSC) field

**Description:**

The reference manual states the following at the Platform flash memory controller Access pipelining functional description.

“The platform flash memory controller does not support access pipelining since this capability is not supported by the flash memory array. As a result, the APC (Address Pipelining Control) field should typically be the same value as the RWSC (Read Wait-State Control) field for best performance, that is, BKn_APC = BKn_RWSC. It cannot be less than the RWSC”.

The reference manual advises that the user must not configure APC to be less than RWSC and typically APC should equal RWSC. However the documentation does not prohibit the configuration of APC greater than RWSC and for this configuration ECC error reporting will be disabled. Flash ECC error reporting will only be enabled for APC = RWSC.

For the case when flash ECC is disabled and data is read from a corrupt location the data will be transferred via the system bus however a bus error will not be asserted and neither a core exception nor an ECSM interrupt will be triggered. For the case of a single-bit ECC error the data will be corrected but for a double-bit error the data will be corrupt.

*Note: Both CFlash & DFlash are affected by this issue.*

*For single-bit and double-bit Flash errors neither a core exception nor an ECSM interrupt will be triggered unless APC=RWSC.*

*The Flash Array Integrity Check feature is not affected by this issue and will successfully detect an ECC error for all configurations of APC >= RWSC.*

*For the APC > RWSC configuration other than flash ECC error reporting there will be no other unpredictable behavior from the flash.*

*The write wait-state control setting at PFCRx[BKn_WWSC] has no affect on the flash. It is recommend to set WWSC = RWSC = APC.*

**Workaround:**

PFCRx[BKy_APC] must equal PFCRx PFCRx[BKy_RWSC]. See datasheet for correct setting of RWSC.

## 1.15 ERR006685: CFlash: Page buffer and prefetch not available for address range 0x0020_0000 to 0x002F_FFFF (upper 1MB)

**Description:**

For the CFlash address range 0x0020_0000 to 0x002F_FFFF (upper 1 MB) the CFlash page buffer and prefetch functions are not available. This affects all master accesses (e200z4d instruction port, e200z4d data port, e200z0h instruction port, e200z0h data port, eDMA, FEC, FlexRay and CSE) and affects both CFlash slave ports P0 (e200z4d instruction port) and P1 (all other masters). As no read to the affected CFlash region can be buffered all accesses will be delayed by the flash array access time as defined in the datasheet.

**Workaround:**

It is recommended to place e200z4d cacheable code in the affected region 0x0020_0000 to

0x002F_FFFF (upper 1MB), as the cache shall negate the affect of the errata and performance impact should be minimal. The e200z0h core does not support cache and therefore code for the e200z0h core should be placed in the unaffected region 0x0000_0000 to 0x001F_FFFF (lower 2MB).

It has been observed that by utilizing this workaround that there has been no degradation of e200z4d core performance. However, the user should be aware that it is difficult to quantify the affect that this errata will have on MCU performance, as it will depend on the flow of the software in the affected CFlash region.

## 1.16 ERR006967: eDMA: Possible misbehavior of a preempted channel when using continuous link mode

**Description:**

When using Direct Memory Access (DMA) continuous link mode Control Register Continuous Link Mode (DMA_CR[CLM]) = 1) with a high priority channel linking to itself, if the high priority channel preempts a lower priority channel on the cycle before its last read/write sequence, the counters for the preempted channel (the lower priority channel) are corrupted. When the preempted channel is restored, it continues to transfer data past its "done" point (that is the byte transfer counter wraps past zero and it transfers more data than indicated by the byte transfer count (NBYTES)) instead of performing a single read/write sequence and retiring.

The preempting channel (the higher priority channel) will execute as expected.

**Workaround:**

Disable continuous link mode (DMA_CR[CLM]=0) if a high priority channel is using minor loop channel linking to itself and preemption is enabled. The second activation of the preempting channel will experience the normal startup latency (one read/write sequence + startup) instead of the shortened latency (startup only) provided by continuous link mode.

## 1.17 ERR007120: NZxC3: DQTAG implemented as variable length field in DQM message

**Description:**

The e200zx core implements the Data Tag (DQTAG) field of the Nexus Data Acquisition Message (DQM) as a variable length packet instead of an 8-bit fixed length packet. This may result in an extra clock ("beat") in the DQM trace message depending on the Nexus port width selected for the device.

**Workaround:**

Tools should decode the DQTAG field as a variable length packet instead of a fixed length packet.

## 1.18 ERR007322: FlexCAN: Bus Off Interrupt bit is erroneously asserted when soft reset is performed while FlexCAN is in Bus Off state

**Description:**

Under normal operation, when FlexCAN enters in Bus Off state, a Bus Off Interrupt is issued to the CPU if the Bus Off Mask bit (CTRL[BOFF_MSK]) in the Control Register is set. In consequence, the CPU services the interrupt and clears the ESR[BOFF_INT] flag in the Error and Status Register to turn off the Bus Off Interrupt.

In continuation, if the CPU performs a soft reset after servicing the bus off interrupt request, by either requesting a global soft reset or by asserting the MCR[SOFT_RST] bit in the Module Configuration Register, once MCR[SOFT_RST] bit transitions from 1 to 0 to acknowledge the soft reset completion, the ESR[BOFF_INT] flag (and therefore the Bus Off Interrupt) is re-asserted.

The defect under consideration is the erroneous value of Bus Off flag after soft reset under the scenario described in the previous paragraph.

The Fault Confinement State (ESR[FLT_CONF] bit field in the Error and Status Register) changes from 0b11 to 0b00 by the soft reset, but gets back to 0b11 again for a short period, resuming after certain time to the expected Error Active state (0b00). However, this late correct state does not reflect the correct ESR[BOFF_INT] flag which stays in a wrong value and in consequence may trigger a new interrupt service.

**Workaround:**

To prevent the occurrence of the erroneous Bus Off flag (and eventual Bus Off Interrupt) the following soft reset procedure must be used:

1. Clear CTRL[BOFF_MSK] bit in the Control Register (optional step in case the Bus Off Interrupt is enabled).
2. Set MCR[SOFT_RST] bit in the Module Configuration Register.
3. Poll MCR[SOFT_RST] bit in the Module Configuration Register until this bit is cleared.
4. Wait for 4 peripheral clocks.
5. Poll ESR[FLTCONF] bit in the Error and Status Register until this field is equal to 0b00.
6. Write "1" to clear the ESR[BOFF_INT] bit in the Error and Status Register.
7. Set CTRL[BOFF_MSK] bit in the Control Register (optional step in case the Bus Off Interrupt is enabled).

## 1.19 ERR007394: MC_ME: Incorrect mode may be entered on low-power mode exit

**Description:**

For the case when the Mode Entry (MC_ME) module is transitioning from a run mode (RUN0/1/2/3) to a low power mode (HALT/STOP/STANDBY*) if a wake-up or interrupt is detected one clock cycle after the second write to the Mode Control (ME_MCTL) register, the MC_ME will exit to the mode previous to the run mode that initiated the low power mode transition.

Example correct operation DRUN->RUN1-> RUN3->STOP->RUN3

Example failing operation DRUN->RUN1-> RUN3->STOP->RUN1

*Note: STANDBY mode is not available on all SPC56xx microcontrollers.*

**Workaround:**

To ensure the application software returns to the run mode (RUN0/1/2/3) prior to the low power mode (HALT/STOP/STANDBY*) it is required that the RUNx mode prior to the low power mode is entered twice.

The following example code shows RUN3 mode entry prior to a low power mode transition.

```
ME.MCTL.R = 0x70005AF0; /* Enter RUN3 Mode & Key */
ME.MCTL.R = 0x7000A50F; /* Enter RUN3 Mode & Inverted Key */
while (ME.GS.B.S_MTRANS) {} /* Wait for RUN3 mode transition to complete */
ME.MCTL.R = 0x70005AF0; /* Enter RUN3 Mode & Key */
ME.MCTL.R = 0x7000A50F; /* Enter RUN3 Mode & Inverted Key */
while (ME.GS.B.S_MTRANS) {} /* Wait for RUN3 mode transition to complete */
/* Now that run mode has been entered twice can enter low power mode */
/* (HALT/STOP/STANDBY*) when desired. */
```

## 1.20 ERR007589: LINFlexD: Erroneous timeout error when switching from UART to LIN mode

**Description:**

When the LINFlexD module is enabled in Universal Asynchronous Receiver/Transmitter (UART) mode and the value of the MODE bit of the LIN Timeout Control Status register (LINTCSR) is 0 (default value after reset), any activity on the transmit or receive pins will cause an unwanted change in the value of the 8-bit field Output Compare Value 2 (OC2) of the LIN Output Compare register (LINOCR).

As a consequence, if the module is reconfigured from UART to Local Interconnect Network (LIN) mode, an incorrect timeout exception is generated when a LIN communication starts.

**Workaround:**

Before enabling UART communication, set to 1 the MODE bit of the LIN Timeout Control Status register (LINTCSR) (selecting the output compare mode). This is preventing the LINOCR.OC2 field from being updated during UART communications.

Then, after reconfiguring the LINFlexD to LIN mode, reset the LINRCSR.MODE bit (selecting the LIN mode) before starting LIN communications.

## 1.21 ERR007688: RTC: An API interrupt may be triggered prematurely after programming the API timeout value

**Description:**

When the API is enabled (RTCC[APIEN]), the API interrupt flag is enabled (RTCC[APIIE]) and the API timeout value (RTCC[APIVAL]) is programmed the next API interrupt may be triggered before the programmed API timeout value. Successive API Interrupts will be triggered at the correct time interval.

**Workaround:**

The user must not use the first API interrupt for critical timing tasks.

## 1.22 ERR007732: FXOSC is active during standby mode if FIRC_CTL[FIRCDIV] bit is configured to be greater than 1

**Description:**

Irrespective of the ME_STANBY_MC[FXOSCON] bit configuration, FXOSC is active (powered ON) during standby mode if the FIRC_CTL[FIRCDIV] value is greater than 1.

**Workaround:**

Do not configure FIRC_CTL[FIRDIV] value to be greater than one while switching from DRUN or RUN to Standby Mode.

## 1.23 ERR007938: ADC: Possibility of missing CTU conversions

**Description:**

The CTU prioritizes and schedules trigger sources so that the ADC will receive only one CTU trigger at a time. However, whilst a Normal or Injected ADC conversion is ongoing as the ADC moves state from IDLE-to-SAMPLE, SAMPLE-to-WAIT, WAIT-to-SAMPLE and WAIT-to- IDLE there are 2 clock cycles at the state transition that a CTU trigger may be missed by the ADC.

**Workaround:**

To ensure all CTU triggers are received at the ADC Normal and Injected modes must be disabled.

## 1.24 ERR007953: ME: All peripherals that will be disabled in the target mode must have their interrupt flags cleared prior to target mode entry

**Description:**

Before entering the target mode, software must ensure that all interrupt flags are cleared for those peripheral that are programmed to be disabled in the target mode. A pending interrupt from these peripherals at target mode entry will block the mode transition or possibly lead to unspecified behavior.

**Workaround:**

For those peripherals that are to be disabled in the target mode the user has 2 options:

1. Mask those peripheral interrupts and clear the peripheral interrupt flags prior to the target mode request.
2. Through the target mode request ensure that all those peripheral interrupts can be serviced by the core.

## 1.25 ERR008081: Debug – The e200z0 core instruction pointer may be corrupted when attaching the e200z0 to a debugger when the e200z0 system clock divider is configured in 2:1 mode

**Description:**

For the case when both e200z4 and e200z0 cores are active and the e200z4:e200z0 clocks are configured in 2:1 mode (by setting CGM_Z0_DCR[DIV] = 1), the e200z0 core instruction pointer may be corrupted when a debugger is attached to the e200z0 core.

Standalone operation of the device (no debugger attached) is not impacted.

*Note: Typically the NPC_PCR is written by the debug tool automatically when it connects to the MCU. The method of manually configuring the NPC_PCR will depend on the debugger vendor.*

**Workaround:**

Recommended configuration for JTAG (Nexus1 non-trace) z0 debug:

- NPC_PCR[MCKO_EN] = 0, NPC_PCR[EVTEN] = 0, and MCKO_DIV = /1, /4 or /8 (do not set /2)

Recommended configuration for Nexus3+ (trace) z0 debug:

- Configuration of NPC_PCR and CGM_Z0_DCR[DIV] (for 2:1 mode) should be done before z0 is released from reset by the z4
- NPC_PCR[MCKO_EN] = 1 and NPC_PCR [MCKO_DIV] = /4 or /8 (do not set /2)
- The NPC_PCR must be written by doing an “attach" with the z4 debugger

*Note: NPC_PCR[MCKO_EN] = 1 & NPC_PCR[MCKO_DIV] = /2 is a valid configuration for Nexus3+ e200z4 debug when the e200z0 is not enabled or e200z4 and e200z0 are operating in 1:1 frequency mode.*

## 1.26 ERR008227: CGM & ME: The peripheral set clock must be active during a peripheral clock enable or disable request

**Description:**

An individual peripheral clock can be enabled or disabled for a target mode via the Mode Entry Peripheral Control register (ME_PCTL) and the Mode Entry RUN/Low Power Peripheral Configuration register (ME_RUN_PC & ME_LP_PC). For this process to complete the user must ensure that the peripheral set clock relative to the specific peripheral is enabled for the duration of the current-mode-to-target-mode transition. The peripheral set clock is configured at the Clock Generation Module System Clock Divider Configuration Register (CGM_SC_DC).

A caveat for FlexCAN is for the case when the FXOSC is selected for the CAN Engine Clock

Source (FLEXCAN_CTRL[CLK_SRC]). In this instance to enable or disable the FlexCAN

peripheral clock the user must ensure FXOSC is enabled through the target mode transition i.e. FXOSC must be enabled for the target mode.

**Workaround:**

To enable a peripheral clock:

1. Enable the peripheral set clock at CGM_SC_DC.
2. Enable the peripheral clock for the target mode at ME_PCTL & ME_RUN_PC/ ME_LP_PC.
3. Note steps 1 & 2 are interchangeable.
4. Transition to the target mode to enable the peripheral clock.

To disable a peripheral clock:

1. Disable the peripheral clock for the target mode at ME_PCTL & ME_RUN_PC/ ME_LP_PC.
2. Transition to the target mode to disable the peripheral clock.
3. Optionally disable peripheral set clock at CGM_SC_DC. Note to check other peripherals in this peripheral set are not required.

# Appendix A Reference documents

- *SPC564Bxx, SPC56ECxx 32-bit MCU family built on the embedded Power Architecture®* (RM0070, DocID18196)
- *32-bit MCU family built on the Power Architecture® for automotive body electronics applications* (Datasheet, DocID17478).

# Revision history

Table 2. Document revision history

| Date | Revision | Changes |
|---|---|---|
| 19-Oct-2015 | 1 | Initial release. |
| 19-Apr-2016 | 2 | On cover page, changed device datasheet version from Rev 8 to Rev 9. |