# SPC56xP54x, SPC56xP60x Errata sheet

## SPC56xP54x, SPC56xP60x devices errata JTAG_ID = 0x1AE2_4041

## Introduction

This errata sheet describes all the functional and electrical problems known in the revision 1.1 of the SPC56xP54x and SPC56xP60x devices, identified with the JTAG_ID = 0x1AE2_4041.

All the topics covered in this document refer to *RM0083* Rev. 5 and *SPC56xP54x, SPC56xP60x, datasheet* Rev. 6 (see *A.1: Reference document*).

Device identification:

- JTAG_ID = 0x1AE2_4041
- MIDR1 register:
  - MAJOR_MASK[3:0]: 4'b0000
  - MINOR_MASK[3:0]: 4'b0001
- Package device marking mask identifier: AB
- Die mask ID: FP60X1

This errata sheet applies to SPC56xP54x and SPC56xP60x devices in accordance with *Table 1*.

**Table 1. Device summary**

| Package | Part number | |
|---|---|---|
| | 768 KB Flash | 1 MB Flash |
| LQFP144 | SPC560P54L5<br>SPC56AP54L5 | SPC560P60L5<br>SPC56AP60L5 |
| LQFP100 | SPC560P54L3<br>SPC56AP54L3 | SPC560P60L3<br>SPC56AP60L3 |

# Contents

# List of tables

# 1 Functional problems

## 1.1 ERR001388: FlexRay: Incomplete transmission of message frame in key slot

**Description:**

The FlexRay module will transmit an incomplete message in the key slot under the following circumstances:

1. The transmit message buffer n assigned to the key slot is located in the message buffer segment 2, that is, FR_MBSSUTR[MB_LAST_SEG1] < n.
2. The data size of the message buffer segment 1 is smaller than the static payload length, that is, FR_MBDSR[MBSEG1DS] < PCR19[payload_length_static].

In this case, the FlexRay module will transmit only FR_MBDSR[MBSEG1DS] payload words from message buffer n. The remaining words are padded with 0's.

**Workaround:**

The transmit message buffer assigned to key slot must be located in message buffer segment 1.

## 1.2 ERR002302: FlexRay: Message buffer can not be disabled and not locked after CHI command FREEZE

**Description:**

If a complete message was transmitted from a transmit message buffer or received into a message buffer and the Controller Host Interface (CHI) command FREEZE is issued by the application before the end of the current slot, then this message buffer can not be disabled and locked until the module has entered the protocol state normal active.

Consequently, this message buffer can not be disabled and locked by the application in the protocol config state, which prevents the application from clearing the commit bit CMT and the module from clearing the status bits. The configuration bits in the message buffer configuration, control, status registers (MBCCSRn) and the message buffer configuration registers MBCCFRn, MBFIDRn, and MBIDXRn are not affected.

At most one message buffer per channel is affected.

**Workaround:**

There are two types of workaround.

1. The application should not send the CHI command FREEZE and use the CHI command HALT instead.
2. Before sending the CHI command FREEZE the application should repeatedly try to disable all message buffers until all message buffers are disabled. This maximum duration of this task is three static or three dynamic slots.

## 1.3 ERR002360: FlexCAN: Global masks misalignment

**Description:**

Convention: MSB = 0.

During CAN messages reception by FlexCAN, the RXGMASK (Rx Global Mask) is used as acceptance mask for most of the Rx message buffers (MB). When the FIFO Enable bit in the FlexCAN module configuration register (CANx_MCR[FEN], bit 2) is set, the RXGMASK also applies to most of the elements of the ID filter table. However there is a misalignment between the position of the ID field in the Rx MB and in RXIDA, RXIDB and RXIDC fields of the ID Tables. In fact RXIDA filter in the ID Tables is shifted one bit to the left from Rx MBs ID position as shown below:

- Rx MB ID = bits 3–31 of ID word corresponding to message ID bits 0–28
- RXIDA = bits 2–30 of ID Table corresponding to message ID bits 0–28

Note that the mask bits one-to-one correspondence occurs with the filters bits, not with the incoming message ID bits. This leads the RXGMASK to affect Rx MB and Rx FIFO filtering in different ways.

For example, if the user intends to mask out the bit 24 of the ID filter of message buffers then the RXGMASK is configured as 0xffff_ffef. As result, bit 24 of the ID field of the incoming message is ignored during filtering process for message buffers. This very same configuration of RXGMASK would lead bit 24 of RXIDA to be “don't care” and thus bit 25 of the ID field of the incoming message would be ignored during filtering process for Rx FIFO.

Similarly, both RXIDB and RXIDC filters have multiple misalignments with regards to position of ID field in Rx MBs, which can lead to erroneous masking during filtering process for either Rx FIFO or MBs.

RX14MASK (Rx 14 Mask) and RX15MASK (Rx 15 Mask) have the same structure as the RXGMASK. This includes the misalignment problem between the position of the ID field in the Rx MBs and in RXIDA, RXIDB and RXIDC fields of the ID Tables.

**Workaround:**

Therefore it is recommended that one of the following actions be taken to avoid problems:

- Do not enable the RxFIFO. If CANx_MCR[FEN] = 0 then the Rx FIFO is disabled and thus the masks RXGMASK, RX14MASK and RX15MASK do not affect it.
- Enable Rx individual mask registers. If the backwards compatibility configuration bit in the FlexCAN module configuration register (CANx_MCR[BCC], bit 15) is set then the Rx individual mask registers (RXIMR[0:63]) are enabled and thus the masks RXGMASK, RX14MASK and RX15MASK are not used.
- Do not use masks RXGMASK, RX14MASK and RX15MASK (that is, let them in reset value which is 0xffff_ffff) when CANx_MCR[FEN] =1 and CANx_MCR[BCC] = 0. In this case, filtering processes for both Rx MBs and Rx FIFO are not affected by those masks.
- Do not configure any MB as Rx (that is, let all MBs as either Tx or inactive) when CANx_MCR[FEN] = 1 and CANx_MCR[BCC] = 0. In this case, the masks RXGMASK, RX14MASK and RX15MASK can be used to affect ID tables without affecting filtering process for Rx MBs.

## 1.4 ERR002421: FlexRay: Message buffer can not be disabled in POC state INTEGRATION_LISTEN

**Description:**

If the communication controller is started as a non-coldstart node and configured and enabled message buffers in the POS config state for slot 1, then the message buffer can not be disabled in the INTEGRATION_LISTEN state, which is entered when no communication can be established.

**Workaround:**

A Software work-around is available, which is as follows: Run a freeze command just before running the message buffer disable for slot 1. This should enable the message buffer disable during the Listen States.

## 1.5 ERR002656: FlexCAN: Abort request blocks the CODE field

**Description:**

An Abort request to a transmit message buffer (TxMB) can block any write operation into its CODE field. Therefore, the TxMB cannot be aborted or deactivated until it completes a valid transmission (by winning the CAN bus arbitration and transmitting the contents of the TxMB).

**Workaround:**

Instead of aborting the transmission, use deactivation instead.

*Note:* *Note that there is a chance the deactivated TxMB can be transmitted without setting IFLAG and updating the CODE field if it is deactivated.*

## 1.6 ERR003022: SWT: Watchdog is disabled during BAM execution

**Description:**

The watchdog is disabled at the start of BAM execution. In the case of an unexpected issue during BAM execution the CPU may be stalled and it will be necessary to generate an external reset to recover.

**Workaround:**

No workaround.

## 1.7 ERR003165: BAM: Code download via FlexCAN not functioning in a CAN network

**Description:**

When the serial download via FlexCAN is selected setting the FAB (force alternate boot) pin, and ABS (alternate boot selector) pins (ABS0 = 1 and ABS1 = 0) and the micro is part of a CAN network, the serial download protocol may unexpectedly stop in case of CAN traffic. After the code has been downloaded, the BAM tries to disable the FlexCAN module writing the MCR (module configuration register) without waiting for the acknowledge bit LPM_ACK (low power mode acknowledge) to be set. The FlexCAN cannot enter the low power mode until all current transmissions or receptions have finished, further writings into any FlexCAN register may cause the low power mode not to be entered and, as consequence, the BAM to stop.

**Workaround:**

Since the higher the traffic, the higher the chance for the BAM to try to disable the FlexCAN module during a CAN frame reception, make sure that no other CAN frame is sent until the code download protocol has been completed.

## 1.8 ERR003204: LINFlex: LDIV lower than 1.5 value are not valid when in UART mode

**Description:**

Maximum baud rate is Fsys / 24 = Fsys/(16 × LDIV) with LDIV = LINIBRR + LINFBRR / 16.

Configuration with LINIBRR = 1 and LINFBRR < 8 are invalid when UART mode is selected.

**Workaround:**

No workaround.

## 1.9 ERR003262: Register protection on full CMU_CSR

**Description:**

The register protection on CMU_CSR of CMU0 works only on the full 32 bit, while it should protect only the bits 24–31. As a consequence, when register protection is active on CMU_CSR the frequency meter cannot be used anymore.

**Workaround:**

In order to perform a frequency meter operation, the register protection of the relevant CMU must be disabled first; this workaround would work only when soft lock is active.

## 1.10 ERR003263: Serial boot and censorship: Flash read access

**Description:**

In a secured device, starting with a serial boot, it is possible to read the content of the four Flash locations where the RCHW is stored. For example if the RCHW is stored at address 0x00000000, the reads at address 0x00000000, 0x00000004, 0x00000008 and 0x0000000C will return a correct value. Any other Flash address is not readable.

**Workaround:**

No workaround.

## 1.11 ERR003264: MCM: MRSR does not report power on reset event

**Description:**

The flag MRSR[POR] stays low after power on reset event on the device.

**Workaround:**

Do not use MRSR[POR] to determine power on reset cause. Use RGM_DES instead.

## 1.12 ERR003269: MC_ME: Peripheral clocks may get incorrectly disabled or enabled after entering debug mode

**Description:**

If ME_RUN_PCx, ME_LP_PCx, ME_PCTLx registers are changed to enable or disable a peripheral, and the device enters debug mode before a subsequent mode transition, the peripheral clock gets enabled or disabled according to the new configuration programmed. Also ME_PSx registers will report incorrect status as the peripheral clock status is not expected to change on debug mode entry.

**Workaround:**

After modifying any of the ME_RUN_PCx, ME_LP_PCx, ME_PCTLx registers, request a mode change and wait for the mode change to be completed before entering debug mode in order to have consistent behaviour on peripheral clock control process and clock status reporting in the ME_PSx registers.

## 1.13 ERR003407: FlexCAN: CAN transmitter stall in case of no remote frame in response to Tx packet with RTR = 1

**Description:**

FlexCAN does not transmit an expected message when the same node detects an incoming Remote Request message asking for any remote answer.

The issue happens when two specific conditions occur:

1. The Message Buffer (MB) configured for remote answer (with code “a”) is the last MB. The last MB is specified by Maximum MB field in the Module Configuration Register (MCR[MAXMB]).
2. The incoming Remote Request message does not match its ID against the last MB ID.

While an incoming Remote Request message is being received, the FlexCAN also scans the transmit (Tx) MBs to select the one with the higher priority for the next bus arbitration. It is expected that by the Intermission field it ends up with a selected candidate (winner). The coincidence of conditions (1) and (2) above creates an internal corner case that cancels the Tx winner and therefore no message will be selected for transmission in the next frame. This gives the appearance that the FlexCAN transmitter is stalled or “stops transmitting”.

The problem can be detectable only if the message traffic ceases and the CAN bus enters into Idle state after the described sequence of events.

There is NO ISSUE if any of the conditions below holds:

1. The incoming message matches the remote answer MB with code “a”.
2. The MB configured as remote answer with code “a” is not the last one.
3. Any MB (despite of being Tx or Rx) is reconfigured (by writing its CS field) just after the Intermission field.
4. A new incoming message sent by any external node starts just after the Intermission field.

**Workaround:**

Do not configure the last MB as a Remote Answer (with code “a”).

## 1.14 ERR003584: 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 machine check due to the Flash not being available on RUNx mode re-entry. This will cause either a checkstop reset or machine check interrupt.

**Workaround:**

This issue can be handled in one of the following ways. Workaround #1 configures the application to handle the machine check interrupt in RAM dealing with the problem if it occurs. Workaround #2 configures the MCU to avoid the machine check interrupt.

Workaround #1: The application can be configured to handle the machine check interrupt in RAM; when this occurs, the mode entry module can be used to bring up the Flash normally and resume execution. Before stop mode entry, ensure the following:

1. Enable the machine check interrupt at the core MSR[ME] - this prevents a machine check reset occurring
2. Copy IVOR vector table to RAM
3. Point IVPR to vector table in RAM
4. Implement machine check interrupt handler in RAM to power-cycle Flash to synchronise status of Flash between Mode Entry and Flash module

The interrupt handler should perform the following steps:

1. Test machine check at LPM exit due to wakeup/interrupt event
2. ME_RUNx_MC[CFLAON] = 0b01 (power-down)
3. Re-enter mode RUNx (x = 0,1,2,3) to power down Flash
4. Wait for transition to RUNx mode to complete (ME_GS[S_MTRANS] = 1)
5. ME_RUNx_MC[CFLAON] = 0b11 (normal)
6. Re-enter mode RUNx (x = previous x) to power up Flash
7. Wait for transition to RUNx mode to complete (ME_GS[S_MTRANS] = 1)
8. On completion, code execution will return to Flash (via se_rfci)

Workaround #2: The application can be configured to avoid the machine check interrupt; low-power mode can be entered from a RAM function and mode entry configured to have Flash off on return to the current RUNx mode. Flash can then be re-enabled by mode entry within the RAM function before returning to execution from Flash.

1. Prior to LPM mode entry request branch to code execution in RAM while Flash is still in normal mode
2. Set ME_RUNx_MC[CFLAON] = 0b01 (power-down) or 0b10 (low-power) for STOP0/HALT0
3. Set ME_STOP0/HALT0_MC[CFLAON] = ME_RUNx_MC[CFLAON]
4. Enter STOP0/HALT0 mode
5. At wakeup or interrupt from STOP0/HALT0, MCU enters RUNx mode executing from RAM with Flash in low-power or power-down as per the ME_RUNx_MC configuration from step 2.
6. After the STOP0/HALT0 request, set ME_RUNx_MC[CFLAON] = 0b11 (normal)
7. Enter RUNx mode
8. Wait for transition to RUNx mode to complete (ME_GS[S_MTRANS] = 0)
9. Return to code execution in Flash

## 1.15 ERR003609: CRC: Limitation of hardware comparison for CRC result via CRC_OUTP_CHK

**Description:**

The comparison via CRC_OUTP_CHK register of the calculated CRC stored in CRC_OUTP register without CPU load doesn't work for Context 2 and Context 3. Moreover if the comparison via CRC_OUTP_CHK register is used for Context 1, then Context 2 and Context 3 cannot be used for CRC calculation If user needs to use more than one context, the comparison has to be done via SW with negligible CPU load.

**Workaround:**

Using the Context 1, do not use the Context 2 and Context 3 or to do the comparison via SW. Using the Context 2 and/or Context 3, always to do the comparison via SW.

## 1.16 ERR003610: FlexCAN: Wrong data transmission exiting from STOP mode in case EXTAL frequency is greater than IRC

**Description:**

The FlexCAN module has got a programmable clock source that can be either the system clock (SYS_CLK) or oscillator clock (XOSC_CLK) selected by CLK_SRC bit in the FlexCAN_CTRL register. In case FlexCAN module has selected the oscillator clock as clock source and XOSC_CLK is bigger than IRC frequency @16 MHz and the system clock is PLL_CLK if the device enters STOP mode and the FlexCAN module is in transmission then when device exits from STOP mode the FlexCAN module can transmit wrong data. This behavior happens because during STOP mode exit, SYS_CLK will be IRC @16 MHz till PLL gets locked and if a frame transmission happens during this time itself then there will be a CAN Spec violation. The FlexCAN module clock source should not be faster than SYS_CLK.

**Workaround:**

Just before entering/requesting the STOP mode, set the “FRZ” and “HALT” bit of CAN_MCR register to '1' to request for freeze mode. During the STOP mode exit, check for the mode transition completion. As mode transition will be over, only when all clock sources are on and the PLL is selected as system clock, unfreeze the CAN by resetting the “FRZ” or “HALT” bit.

## 1.17 ERR003702: Nexus pins may drive an unknown value immediately after power up but before the 1st clock edge

**Description:**

The Nexus output pins (message data outputs 0:12 [MDO] and Message start/end outputs 0:1 [MSEO]) may drive an unknown value (high or low) immediately after power up but before the 1st clock edge propagates through the device (instead of being weakly pulled low). This may cause high currents if the pins are tied directly to a supply/ground or any low resistance driver (when used as a general purpose input [GPI] in the application).

**Workaround:**

1. Do not tie the Nexus output pins directly to ground or a power supply.
2. If these pins are used as GPI, limit the current to the ability of the regulator supply to guarantee correct start up of the power supply. Each pin may draw up to few hundred mA current.

If not used, the pins may be left unconnected.

## 1.18 ERR005113: ADC: triggering an ABORT or ABORTCHAIN before the conversion starts

**Description:**

When ABORTCHAIN is programmed (ADC_MCR[ABORTCHAIN] = 1) and an injected chain conversion is programmed afterwards, the injected chain is aborted, but neither ADC_ISR[JECH] is set, nor ADC_MCR [ABORTCHAIN] is reset.

When ABORT is programmed (ADC_MCR[ABORT] = 1) 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 ADC_MCR[ABORT]/ ADC_MCR[ABORTCHAIN] before starting the execution of the chain conversion.

## 1.19 ERR005203: ADC: "Abort command" 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 ADC_CDATA[x] register. The ADC_CDATAx[VALID] fields will not be set for a not restored or skipped channel, which indicates this issue has occurred. The ADC_CDATAx[VALID] fields need to be checked before the next Normal chain execution (provided ADC_MCR[OWREN] bit is set in scan mode).

The ADC_CDATAx[VALID] fields should be read by every ECH interrupt at the end of every chain execution.

## 1.20 ERR007804: LINFlex: Consecutive headers received by LIN Slave triggers error interrupt

**Description:**

As per the Local Interconnect Network (LIN) specification, the processing of one frame should be aborted by the detection of a new header sequence.

But in LINFlex IP, if the LIN Slave receives a new header instead of data response corresponding to a previous header received, it triggers a framing error during the new header's reception. Also the LIN Slave remains waiting for the data response corresponding to the first header received.

**Workaround:**

The following workaround should be applied:

1. Set the LTOM bit in the LIN Time-Out Control Status Register (LINTCSR[LTOM]) to '0'.
2. Set Idle on Timeout in the LINTCSR[IOT] register to '1'.
3. Configure master to wait until the occurrence of the Output Compare flag in LIN Error Status Register (LINESR[OCF]) before sending the next header. This flag causes the LIN Slave to go to an IDLE state before the next header arrives, which will be accepted without any framing error.

## 1.21 ERR010115: FCCU: Possible Fake Fault for FCCU CF[6] and CF[7]

**Description:**

Despite no Double ECC Error present, a fake Fault may happen for following FCCU critical fault:

- CF[6] Code Flash ECC Multi bit error
  The issue do not happen in case of # of wait state for Code Flash more or equal to
  3WS @ 50 < freq <= 64 MHz
  2WS @ 25 < freq <= 50 MHz
  1WS @ freq <= 25 MHz
- CF[7] Data Flash ECC Multi bit error
  The issue do not happen in case of # of wait state for Data Flash more or equal to 8 (RWSC >=8)

Flash MCR correctly does not signal the ECC error and the interrupt IVOR 2 is not invoked.

**Workaround:**

Verify that the setting of CF[6] and CF[7] of CSF0 in FCCU is confirmed reading the value of EER of MCR Flash register.

# Appendix A Further information

## A.1 Reference document

- *32-bit MCU family built on the Power Architecture® embedded category for automotive chassis and safety electronics applications* (RM0083, Doc ID 018714)
- *32-bit Power Architecture® based MCU with 1088 KB Flash memory and 80 KB RAM for automotive chassis and safety applications* (SPC560P54x, SPC560P60x, SPC56AP54x, SPC56AP60x datasheet, Doc ID 18340)

## A.2 Acronyms

**Table 2. Acronyms**

| Acronym | Name |
|---|---|
| RWCS | Read/write access control/status register |
| RWD | Read/write access data register |
| CHI | Controller host interface |
| MB | Message buffer |
| TxMB | Transmit message buffer |
| FAB | Force alternate boot |
| MCR | Module configuration register |
| LPM | Low-power mode |
| SYS_CLK | System clock |
| GPI | General purpose input |
| LIN | Local Interconnect Network |

# Revision history

Table 3. Document revision history

| Date | Revision | Changes |
|---|---|---|
| 02-Apr-2012 | 1 | Initial release. |
| 17-Sep-2013 | 2 | Updated Disclaimer. |
| 02-Mar-2016 | 3 | Added the following errata:<br>– ERR005113<br>– ERR005203<br>– ERR007804<br>– ERR010115<br><br>Removed the following errata:<br>– ERR003324<br>Removed Appendix A: Defects across silicon version<br>Added *Table 2: Acronyms*. |
| 26-Jul-2016 | 4 | Updated *Table 1: Device summary*<br><br>Removed the following errata:<br>– ERR000817<br>– ERR002423<br>– ERR003611 |