

## Errata

# MSPM0G351x, MSPM0G151x, MSPM0G351x-Q1, MSPM0G3529-Q1 Microcontrollers



### ABSTRACT

This document describes the known exceptions to the functional specifications (advisories).

### Table of Contents

<b>1 Functional Advisories</b> .....	1
<b>2 Preprogrammed Software Advisories</b> .....	2
<b>3 Debug Only Advisories</b> .....	2
<b>4 Fixed by Compiler Advisories</b> .....	3
<b>5 Device Nomenclature</b> .....	3
5.1 Device Symbolization and Revision Identification.....	3
<b>6 Advisory Descriptions</b> .....	5
<b>7 Trademarks</b> .....	24
<b>8 Revision History</b> .....	25

### 1 Functional Advisories

Advisories that affect the device operation, function, or parametrics.

✓ The check mark indicates that the issue is present in the specified revision.

Errata Number	Rev A (Prototype X-Marked Products)	Rev C
<a href="#">ADC_ERR_06</a>	✓	✓
<a href="#">ADC_ERR_10</a>	✓	✓
<a href="#">AES_ERR_01</a>	✓	✓
<a href="#">CPU_ERR_02</a>	✓	✓
<a href="#">CPU_ERR_03</a>	✓	✓
<a href="#">FLASH_ERR_01</a>	✓	✓
<a href="#">FLASH_ERR_03</a>	✓	✓
<a href="#">FLASH_ERR_04</a>	✓	✓
<a href="#">FLASH_ERR_05</a>	✓	✓
<a href="#">FLASH_ERR_08</a>	✓	✓
<a href="#">GPIO_ERR_04</a>	✓	✓
<a href="#">I2C_ERR_04</a>	✓	✓
<a href="#">I2C_ERR_05</a>	✓	✓
<a href="#">I2C_ERR_06</a>	✓	✓
<a href="#">I2C_ERR_07</a>	✓	✓
<a href="#">I2C_ERR_08</a>	✓	✓
<a href="#">I2C_ERR_09</a>	✓	✓
<a href="#">I2C_ERR_10</a>	✓	✓

Errata Number	Rev A (Prototype X-Marked Products)	Rev C
I2C_ERR_13	✓	✓
KEYSTORE_ERR_01	✓	✓
MATHACL_ERR_01	✓	✓
MATHACL_ERR_02	✓	✓
PMCU_ERR_09	✓	✓
PMCU_ERR_10	✓	
PMCU_ERR_11	✓	✓
RST_ERR_01	✓	✓
RTC_ERR_01	✓	✓
SPI_ERR_02	✓	✓
SPI_ERR_04	✓	✓
SPI_ERR_05	✓	✓
SPI_ERR_06	✓	✓
SPI_ERR_07	✓	✓
SRAM_ERR_03	✓	
SYSCTL_ERR_01	✓	✓
SYSCTL_ERR_02	✓	✓
SYSCTL_ERR_03	✓	✓
SYSCTL_ERR_04	✓	✓
SYSOSC_ERR_01	✓	✓
SYSOSC_ERR_02	✓	✓
SYSOSC_ERR_04	✓	✓
SYSPLL_ERR_01	✓	✓
TIMER_ERR_04	✓	✓
TIMER_ERR_06	✓	✓
TIMER_ERR_07	✓	✓
UART_ERR_01	✓	✓
UART_ERR_02	✓	✓
UART_ERR_04	✓	✓
UART_ERR_05	✓	✓
UART_ERR_06	✓	✓
UART_ERR_07	✓	✓
UART_ERR_08	✓	✓
UART_ERR_10	✓	✓
UART_ERR_11	✓	✓

## 2 Preprogrammed Software Advisories

Advisories that affect factory-programmed software.

✓ The check mark indicates that the issue is present in the specified revision.

## 3 Debug Only Advisories

Advisories that affect only debug operation.

✓ The check mark indicates that the issue is present in the specified revision.

Errata Number	Rev A	Rev B
GPIO_ERR_03	✓	✓

## 4 Fixed by Compiler Advisories

Advisories that are resolved by compiler workaround. Refer to each advisory for the IDE and compiler versions with a workaround.

✓ The check mark indicates that the issue is present in the specified revision.

## 5 Device Nomenclature

To designate the stages in the product development cycle, TI assigns prefixes to the part numbers of all MSP MCU devices. Each MSP MCU commercial family member has one of two prefixes: MSP or XMS. These prefixes represent evolutionary stages of product development from engineering prototypes (XMS) through fully qualified production devices (MSP).

**XMS** – Experimental device that is not necessarily representative of the final device's electrical specifications

**MSP** – Fully qualified production device

Support tool naming prefixes:

**X**: Development-support product that has not yet completed Texas Instruments internal qualification testing.

**null**: Fully-qualified development-support product.

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

"Developmental product is intended for internal evaluation purposes."

MSP 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 (XMS) have a greater failure rate than the standard production devices. TI 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.

TI device nomenclature also includes a suffix with the device family name. This suffix indicates the temperature range, package type, and distribution format.

### 5.1 Device Symbolization and Revision Identification

The package diagrams below indicate the package symbolization scheme, and [Table 5-1](#) defines the device revision to version ID mapping.

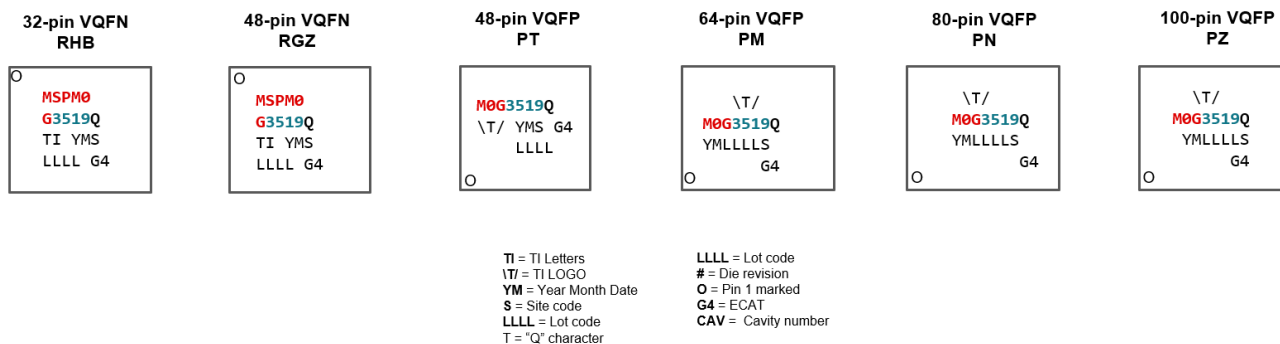


Figure 5-1. Package Symbolization

**Table 5-1. Die Revisions**

Revision Letter	Version (in the device factory constants memory)
A	1
C	2

The revision letter indicates the product hardware revision. Advisories in this document are marked as applicable or not applicable for a given device based on the revision letter. This letter maps to an integer stored in the memory of the device, which can be used to look up the revision using application software or a connected debug probe.

## 6 Advisory Descriptions

<b>ADC_ERR_06</b>	<b>ADC Module</b>
<b>Category</b>	Functional
<b>Function</b>	ADC Output code jumps degrading DNL/INL specification
<b>Description</b>	<p>When a conversion error occurs, the error will be a fixed +/- 64LSB jump in the digital output code of the ADC without a corresponding change in the ADC input voltage.</p> <p>At worst case scenario, -40C, the error rate is 1 in 24M converted samples in 12-bit mode. (VDD voltage and reference used has no impact on errata rate)</p>
<b>Workaround</b>	<p>Depending on the application needs the best workaround may vary, but the following workarounds in software are proposed. Selection of the best workaround is left to the judgment of the system designer.</p> <p>Workaround 1: Upon ADC result outside of application threshold (via ADC Window Comparator or software thresholding), trigger or wait for another ADC result before making critical system decisions</p> <p>Workaround 2: During post-processing, discard ADC values which are sufficiently far from the median or expected value. The expected value should be based on the average of real samples taken in the system, and the threshold for rejection should be based on the magnitude of the measured system noise.</p> <p>Workaround 3: Use ADC sample averaging to minimize the effect of the results of any single incorrect conversion.</p>
<b>ADC_ERR_10</b>	<b>ADC Module</b>
<b>Category</b>	Functional
<b>Function</b>	ADCMEMRES swap is seen when PA15/PA18/PA22/PA21 is toggling
<b>Description</b>	<p>Set up condition:</p> <ol style="list-style-type: none"> <li>1. ADC is in repeat sequence mode</li> <li>2. ADC can use any sequence of channels to read data</li> <li>3. PA15/PA18/PA22/PA21 is toggling by either an external device or the MCU itself (such as PWM).</li> </ol> <p>Observation:</p> <p>When the software is starting the ADC conversion, sometimes MEMRES data will swap. The data which needs to be in MEMRES0 is coming into MEMRES1, MEMRES1 data is coming into MEMRES2..so on. In repeat mode, when this errata happens, the last MEMRES will appear in MEMRES0.</p>

**ADC\_ERR\_10**  
(continued)

---

**ADC Module**


---

The toggling signal on PA15/PA18/PA22/PA21 can affect the conversion clock of the ADC. Which causes the ADC to store the previous result before the correct channel data comes.

**Workaround**

Avoid fast switching signals (12kHz or faster) on PA15/PA18/PA22/PA21.

**AES\_ERR\_01**


---

**AES Module**


---

**Category**

Functional

**Function**

AES Saved Context Ready interrupt is not generating as expected

**Description**

Saved Context Ready interrupt is not getting generated. The interrupt is generated if an access (read or write) is made to any AES register.

**Workaround**

Use polling based mechanism to check the status bit for Saved Context Ready in CTRL register instead of interrupt.

**CPU\_ERR\_02**


---

**CPU Module**


---

**Category**

Functional

**Function**

Limitation of disabling prefetch for CPUSS

**Description**

CPU prefetch disable will not take effect if there is a pending flash memory access.

**Workaround**

Disable the prefetcher, then issue a memory access to the shutdown memory (SHUTDNSTORE) in SYSCTL, this can be done with `SYSCTL.SOCLOCK.SHUTDNSTORE0;`

After the memory access completes the prefetcher will be disabled.

Example:

```
CPUSS.CTL.PREFETCH = 0x0; //disables prefetcher
```

```
SYSCTL.SOCLOCK.SHUTDNSTORE0; // memory access to shutdown memory
```

**CPU\_ERR\_03**


---

**CPU Module**


---

**Category**

Functional

**Function**

Prefetcher can fetch wrong instructions when transitioning into Low power modes

**CPU\_ERR\_03**

(continued)

**CPU Module**


---

**Description**

When transitioning into low power modes and there is a pending prefetch, the prefetcher can erroneously fetch incorrect data (all 0's). When the device wakes up, if the prefetcher and cache do not get overwritten by ISR code, then the main code execution from flash can get corrupted. For example, if the ISR is in the SRAM, then the incorrect data that was prefetched from Flash does not get overwritten. When the ISR returns the corrupted data in the prefetcher can be fetched by the CPU resulting in incorrect instructions. A HW Event wake is another example of a process that will wake the device, but not flush the prefetcher.

**Workaround**

Disable prefetcher before entering low power modes.

Example:

```

CPUSS->CTL &= 0x6; // disables prefetcher, maintains other settings
SYSCTL.SOCLOCK.SHUTDNSTORE0 // Read from SHUTDOWN Memory
__WFI(); // or __WFE(); this function calls the transition into low power mode
CPUSS->CTL |= 0x1; // enables prefetcher
  
```

**FLASH\_ERR\_01**
**FLASH Module**


---

**Category**

Functional

**Function**

Access to FACTORY region will lead to hard fault with flash wait state equal to 2.

**Description**

Access to FACTORY region when the flash wait state is set to 2 will trigger a hard fault. When MCLK is set to beyond 32MHz, the flash wait states needs to be 2.

**Workaround**

Set MCLK at a lower frequency(with flash wait state as 0 or 1) to access FACTORY region. Access the FACTORY region while the flash wait states is less than 2 (requires MCLK to be 32MHz or less). Cache the data in SRAM, MAIN flash, or DATA flash, if the application needs to access these values during run time. A typical value would be the Temperature Sensor's calibration value.

**FLASH\_ERR\_03**
**FLASH Module**


---

**Category**

Functional

**Function**

Flash access with 2 wait states followed by invalid bootcode access will cause next flash access to also throw a violation

**Description**

Doing a Flash access followed by a access to BOOTCODE when you have 2 wait states will cause the next flash access to also cause a violation.

**Workaround**

Do not attempt to access boot-code region post-boot phase. Otherwise, there will need to be 4 clock cycles in between the bootcode violation and next correct flash access.

**FLASH\_ERR\_04** *FLASH Module*


---

**Category**

Functional

**Function**

Wrong Address will get reported in the SYSCTL\_DEDERRADDR if the error is in the NONMAIN or Factory region

**Description**

When a FLASHDED error appears the data will truncate the most significant byte. In the memory limits of the device, the most significant byte does not have an impact to the return address for MAIN flash. For NONMAIN flash or Factory region the MSB should be listed as 0x41xx.xxxx

**Workaround**

If the return address of the SYSCTL\_DEDERRADDR returns a 0x00Cxxxxx, do an OR operation with 0x41000000 to get the proper address for the NONMAIN or Factory region return address. For example, if SYSCTL\_DEDERRADDR = 0x00C4013C, the real address would be 0x41C4013C.

For MAIN Flash DED, the SYSCTL\_DEDERRADDR can be used as is.

**FLASH\_ERR\_05** *FLASH Module*


---

**Category**

Functional

**Function**

DEDERRADDR can have incorrect reset value

**Description**

The reset value of the SYSCTL->DEDERRADDR can return a 0x00C4013C instead of the correct 0x00000000. The location of the error is in the Factory Trim region and is not indicative of a failure, it can be properly ignored. The reset value tends to change once NONMAIN has been programmed on the device.

**Workaround**

Accept 0x00C4013C as another reset value, so the default value from boot can be 0x00000000 or 0x00C4013C. The return value is outside of the range of the MAIN flash on the device so there is no potential of this return coming from an actual FLASH DED status.

**FLASH\_ERR\_08** *FLASH Module*


---

**Category**

Functional

**Function**

Hard fault isn't generated for typical invalid memory region

**Description**

Hard fault isn't generated while trying to access illegal memory address space as shown below: 1. 0x010053FF - 0x20000000 2. 0x40BFFFFFF - 0x41C00000 3. 0x41C007FF - 0x41C40000

**Workaround**

No



<b>GPIO_ERR_03</b>	<b>GPIO Module</b>
<b>Category</b>	Functional
<b>Function</b>	On a debugger read to GPIO EVENT0 IIDX, interrupt is cleared.
<b>Description</b>	EVENT0's IIDX of GPIO, on a debugger read is treated as a CPU read and interrupt is getting cleared.
<b>Workaround</b>	During the debug, the IIDX of event0 can be read by software reading RIS.
<b>GPIO_ERR_04</b>	<b>GPIO Module</b>
<b>Category</b>	Functional
<b>Function</b>	Configuring global fastwake is not allowing PAD data to go to DIN register
<b>Description</b>	When configuring the fast wake only bit in the CTL register and forcing data to PAD in run mode, the data in PAD is not reflected in the DIN register. This is because the CTL register configuration prevents any data from flowing from the PAD to the DIN register.
<b>Workaround</b>	Avoid using the GPIO fastwake-only function when expecting data on the PAD entering the DIN register.
<b>I2C_ERR_04</b>	<b>I2C Module</b>
<b>Category</b>	Functional
<b>Function</b>	When SCL is low and SDA is high the Target i2c is not able to release the stretch.
<b>Description</b>	<p>1: SCL line grounded and released, device indefinitely pulls SCL low.</p> <p>2: Post clock stretch, timeout, and release; if there is another clock low on the line, device indefinitely pulls SCL low.</p>
<b>Workaround</b>	<p>If the I2C target application does not require data reception in low power mode using Async fast clock request, disabling SWUEN by default is recommended, including during reset or power cycle. In this case, bug description 1 and 2 does not occur.</p> <p>If the I2C target application requires data reception in low power mode using Async fast clock request, enable SWUEN just before entering low power and clear SWUEN after low power exit. Even in this scenario, bug description 1 and 2 can occur when the I2C target is in low power, it will indefinitely stretch the SCL line if there is a continuous clock stretching or timeout caused by another device on the bus. To recover from this situation, enable the low timeout interrupt on the I2C target device, reset and re-initialize the I2C module within the low timeout ISR.</p>

<b>I2C_ERR_05</b>	<b><i>I2C Module</i></b>
<b>Category</b>	Functional
<b>Function</b>	I2C SDA can get stuck to zero if we toggle ACTIVE bit during ongoing transaction
<b>Description</b>	If ACTIVE bit is toggled during an ongoing transfer, the state machine will be reset. However, the SDA and SCL output which is driven by the controller will not get reset. There is a situation where SDA is 0 and the controller has gone into IDLE state, here the controller won't be able to move forward from the IDLE state or update the SDA value. The target's BUSBUSY is set (toggling of the ACTIVE bit is leading to a start being detected on the line) and the BUSBUSY won't be cleared as the controller will not be able to drive a STOP to clear it.
<b>Workaround</b>	Do not toggle the ACTIVE bit during an ongoing transaction.
<b>I2C_ERR_06</b>	<b><i>I2C Module</i></b>
<b>Category</b>	Functional
<b>Function</b>	SMBus High timeout feature fails at I2C clock less than 24KHz onwards
<b>Description</b>	SMBus High timeout feature is failing at I2C clock rate less than 24KHz onwards (20KHz, 10KHz). From SMBUS Spec, the upper limit on SCL high time during active transaction is 50us. Total time taken from writing of START MMR bit to SCL low is 60us, which is >50us. It will trigger the timeout event and let the I2C controller goes into IDLE without completing the transaction at the start of transfer itself. Below is detailed explanation. For SCL is configured as 20KHz, SCL low and high period is 30us and 20us respectively. First, START MMR bit write at the same time high timeout counter starts decrementing. Then, it takes one SCL low period (30us) from START MMR bit write to SDA goes low (start condition). Next, it takes another SCL low period (30us) from SDA goes low (start condition) to SCL goes low (data transfer starts) which should stop the high timeout counter at this point. As a total, it takes 60us from counter start to end. However, due to the upper limit(50us) of the high timeout counter, the timeout event will still be triggered although the I2C transaction is working fine without issue.
<b>Workaround</b>	Do not use SMBus High timeout feature when I2C clock is less than 24KHz onwards.
<b>I2C_ERR_07</b>	<b><i>I2C Module</i></b>
<b>Category</b>	Functional
<b>Function</b>	Back to back controller control register writes will cause I2C to not start.
<b>Description</b>	Back-to-Back CTR register writes will cause the next CTR.START to not properly cause the start condition.

**I2C\_ERR\_07**

(continued)

***I2C Module***

---

**Workaround**

Write all the CTR bits including CTR.START in a single write or wait one clock cycle between the CTR writes and CTR.START write.

**I2C\_ERR\_08**

***I2C Module***

---

**Category**

Functional

**Function**

FIFO Read directly after RXDONE interrupt causes erroneous data to be read

**Description**

When the RXDONE interrupt happens the FIFO is not always updated for the latest data.

**Workaround**

Wait 2 I2C CLK cycles for the FIFO to make sure to have the latest data. I2C CLK is based on the CLKSEL register in the I2C registers.

**I2C\_ERR\_09**

***I2C Module***

---

**Category**

Functional

**Function**

Start address match status might not be updated in time for a read through the ISR if running I2C at slow speeds.

**Description**

If running at I2C speeds less than 100kHz then the ADDRMATCH bit (address match in the TSR register) might not be set in time for the read through an interrupt.

**Workaround**

If running at below 100kHz on I2C, wait at least 1 I2C CLK cycle before reading the ADDRMATCH bit.

**I2C\_ERR\_10**

***I2C Module***

---

**Category**

Functional

**Function**

I2C Busy status is enabled preventing low power entry

**Description**

When in I2C Target mode, the I2C Busy Status stays high after a transaction if there is no STOP bit.

**Workaround**

Program the I2C controller to send the STOP bit and don't send a NACK for the last byte. Terminate any I2C transfer with a STOP condition to maintain proper BUSY status and asynchronous clock request behavior (for low power mode reentry).

<b>I2C_ERR_13</b>	<b><i>I2C Module</i></b>
<b>Category</b>	Functional
<b>Function</b>	Polling the I2C BUSY bit might not guarantee that the controller transfer has completed
<b>Description</b>	After setting the CCTR.BURSTRUN bit to initiate an I2C controller transfer, it takes approximately 3 I2C functional clock cycles for the BUSY status to be asserted. If polling for the BUSY bit is used immediately after setting CCTR.BURSTRUN to wait for transfer completion, the BUSY status might be checked before it is set. This problem is more likely to occur with high CLKDIV values (resulting in a slower I2C functional clock) or under higher compiler optimization levels.
<b>Workaround</b>	Add software delay before polling BUSY status. Software delay = 3 x CPU CLK / I2C functional clock = 3 x CPU CLK / (CLKSEL / CLKDIV) For example, with a clock divider (CLKDIV) of 8, a clock source of 4 MHz(MFCLK), and CPU CLK of 32 MHz: Software delay = 3 x 32 MHz / (4 MHz/ 8) = 192 CPU cycles
<b>KEYSTORE_ERR_01</b>	<b><i>KEYSTORE Module</i></b>
<b>Category</b>	Functional
<b>Function</b>	STATUS.STAT value can be 0 or 1 without key access
<b>Description</b>	STATUS.STAT has a reset value of 1 and turns to 0 under these conditions: 1. After reset, debugger access via the register window returns 0x00. 2. After reset, the first CPU read returns 0x01, while subsequent CPU reads return 0x00. 3) After reset, first reading any other KEYSTORE register and then reading STATUS.STAT return 0x00.
<b>Workaround</b>	STATUS.STAT=0x0 means "No Error" . For checking if a slot is valid or not (Whether key is present), check STATUS.VALID.
<b>MATHACL_ERR_01</b>	<b><i>MATHACL Module</i></b>
<b>Category</b>	Functional
<b>Function</b>	MATHACL status error bit does not get cleared
<b>Description</b>	If there is a status error generated by the mathacl (ex. divide by 0), then the status register never gets cleared.
<b>Workaround</b>	Reset the peripheral to clear the status bit.

## **MATHACL\_ERR\_0**

2

### ***MATHACL Module***

---

**Category**

Functional

**Function**

MATHACL COS(-180) gives 1 instead of -1, SIN(-90) will give 1 instead of -1

**Description**

MATHACL will return a 1 instead of a -1 when performing COS(-180) or SIN(-90)

**Workaround**

No workaround, make the result negative in software.

## **PMCU\_ERR\_09**

### ***PMCU Module***

---

**Category**

Functional

**Function**

RSTCAUSE is updating incorrectly as 0xC, after a POR (NRST>1s) reset

**Description**

When using NRST to trigger POR (NRST>1s) reset, RSTCAUSE is updating incorrectly as 0xC, which is expected as 0x2.

**Workaround**

When need check reset cause, use one of the available SHUTDOWN memory bytes (SHUTDNSTOREx) to store a non-zero data after get RSTCAUSE status. When RSTCAUSE returns 0xC, then a POR occurs if the SHUTDNSTOREx data cleared, or a BOR occurs if the SHUTDNSTOREx data maintained.

## **PMCU\_ERR\_10**

### ***PMCU Module***

---

**Category**

Functional

**Function**

VBOOST might have larger delay under certain operating conditions

**Description**

VBOOST for analog MUX has large delay at VDD<1.8V, which delays settling time of other modules like HFXT, COMP, SYSOSC(FCL-external R),OPA and GPAMP.

**Workaround**

Keep VDD>=1.8V and use VBOOST in ONALWAYS mode using GENCLKCFG[23:22]=0x2.

## **PMCU\_ERR\_11**

### ***PMCU Module***

---

**Category**

Functional

**Function**

NRST<1sec pulse giving wrong rstcause in shutdown mode

**PMCU\_ERR\_11**

(continued)

***PMCU Module***

---

**Description**

The rstcause value is wrong under the following condition. Though the expected rstcause is 0x05.

- (i) Device is configured for shutdown mode
- (ii) WFI() is called
- (iii) Give NRST<1sec pulse to bring device out from shutdown mode

**Workaround**

No workaround.

**RST\_ERR\_01*****RST Module***

---

**Category**

Functional

**Function**

NRST release doesn't get detected when LFCLK\_IN is LFCLK source and LFCLK\_IN gets disabled

**Description**

When LFCLK = LFCLK\_IN and we disable the LFCLK\_IN, then comes a corner scenario where NRST pulse edge detection is missed and the device doesn't come out of reset. This issue is seen if the NRST pulse width is below 608us. NRST pulse above 608us, the reset can appear normally.

**Workaround**

Keep the NRST pulse width higher than 608us to avoid this issue.

**RTC\_ERR\_01*****RTC Module***

---

**Category**

Functional

**Function**

Some RTC Interrupts are not available in STANDBY1

**Description**

When in STANDBY1, the RTCRDY and RTC\_PRESCALER1 interrupts cannot wakeup the device.

**Workaround**

When waking up the device from STANDBY1 with the RTC, use other available interrupts such as RTC\_ALARM and RTC\_PRESCALER0.

**SPI\_ERR\_02*****SPI Module***

---

**Category**

Functional

**Function**

Missing SPI Clock and data bytes after wake-up from low power mode (LPM)

## SPI\_ERR\_02

(continued)

### SPI Module

---

#### Description

After device wake-up from a low power state, the SPI module can not properly propagate the first few clock cycles and data bits of the first byte sent out.

#### Workaround

To maintain SPI data integrity after a wakeup, use the following sequence when entering and exiting LPMs:

1. Disable SPI module
2. Wait for Interrupt(WFI)- enter LPM
3. Wake up from LPM (any source).
4. Enable the SPI module.

## SPI\_ERR\_04

### SPI Module

---

#### Category

Functional

#### Function

IDLE/BUSY status toggle after each frame receive when SPI peripheral is in only receive mode.

#### Description

In case of SPI peripheral in only receive mode, the IDLE interrupt and BUSY status are toggling after each frame receive while SPI is receiving data continuously(SPI\_PHASE=1). Here there is no data loaded into peripheral TXFIFO and TXFIFO is empty.

#### Workaround

Do not use SPI peripheral only receive mode. Set SPI peripheral in transmit and receive mode. You do not need to set any data in the TX FIFO for SPI.

## SPI\_ERR\_05

### SPI Module

---

#### Category

Functional

#### Function

SPI Peripheral Receive Timeout interrupt is setting irrespective of RXFIFO data

#### Description

When using the SPI timeout interrupt the RXTIMEOUT can continue decrementing even after the final SPI CLK is received, which can cause a false RXTIMEOUT.

#### Workaround

Disable the RXTIMEOUT after the last packet is received (this can be done in the ISR) and re-enable when SPI communication starts again.

## SPI\_ERR\_06

### SPI Module

---

#### Category

Functional

**SPI\_ERR\_06**

(continued)

---

**SPI Module**


---

**Function**

IDLE/BUSY status does not reflect the correct status of SPI IP when debug halt is asserted

**Description**

IDLE/BUSY is independent of halt, it is only gating the RXFIFO/TXFIFO writing/reading strobes. So, if controller is sending data, although it's not latched in FIFO but the BUSY is getting set. The POCI line transmits the previously transmitted data on the line during halt

**Workaround**

Don't use IDLE/BUSY status when SPI IP is halted.

**SPI\_ERR\_07**


---

**SPI Module**


---

**Category**

Functional

**Function**

SPI underflow event may not generate if read/write to TXFIFO happen at the same time for SPI peripheral

**Description**

When SPI.CTL0.SPH = 0 and the device is configured as the SPI peripheral.

If there is a write to the TXFIFO WHILE there is a read request from the SPI controller, then an underflow event may not be generated as the read/write request is happening simultaneously.

**Workaround**

Ensure the TXFIFO is not empty when the SPI Controller is addressing the device, this can be done by preloading data to avoid a write and read to the same TXFIFO address. Alternatively, data checking strategies, like CRC, can be used to verify the packets were sent properly, then the data can be resent if the CRC doesn't match.

**SRAM\_ERR\_03**


---

**SRAM Module**


---

**Category**

Functional

**Function**

SRAM Parity and ECC function is not supported on Rev A devices

**Description**

SRAM Parity and ECC function is not supported on Rev A devices. Please do not use SRAM Parity and ECC function on Rev A devices.

**Workaround**

None.

**SYSCTL\_ERR\_01** **SYSCTL Module**


---

**Category**

Functional



## **SYSTL\_ERR\_01**

(continued)

### ***SYSTL Module***

---

#### **Function**

SW-POR functionality is combined with HW-POR

#### **Description**

When a user writes to the LFSSRST register with the correct key to generate a software-triggered POR, the RSTCAUSE register will display 0x2 (indicating an NRST-triggered POR) instead of the expected 0x3 (Software-Triggered POR). This occurs because the SW-POR functionality is combined with the HW-POR path.

#### **Workaround**

No

## **SYSTL\_ERR\_02**

### ***SYSTL Module***

---

#### **Category**

Functional

#### **Function**

SYSSTATUS.FLASHSEC is non-zero after a BOOTRST

#### **Description**

After BOOTRST/ bootcode completion SYSSTATUS.FLASHSEC is non-zero. This the customer will see after bootcode completion.

#### **Workaround**

No

---

**SYSCTL\_ERR\_03** *SYSCTL Module*


---

**Category**

Functional

**Function***DEDERRADDR persists after a SYSRESET or a write to the SYSSTATUSCLR***Details**

DEDERRADDR persists after either a SYSRESET or a write to the SYSSTATUSCLR register. Its value is overwritten only when a new FLASHDED error occurs. This behavior contradicts the Technical Reference Manual (TRM), which specifies its initial reset value as zero.

**Workaround**

No workaround

---

**SYSCTL\_ERR\_04** *SYSCTL Module*


---

**Category**

Functional

**Function**

SYSSTATUS.FLASHSEC is not cleared after a SYSRESET

**Description**

SYSSTATUS.FLASHSEC is not cleared after a SYSRESET and is only cleared by writing to the SYSSTATUSCLR register.

**Workaround**

No

---

**SYSOSC\_ERR\_01** *SYSOSC Module*


---

**Category**

Functional

**Function**

MFCLK drift when using SYSOSC FCL together with STOP1 mode

**Description**

If MFCLK is enabled AND SYSOSC is using the frequency correction loop (FCL) mode AND STOP1 low power operating mode is used, then the MFCLK may drift by two cycles when SYSOSC shifts from 4MHz back to 32MHz (either upon exit from STOP1 to RUN mode or upon an asynchronous fast clock request that forces SYSOSC to 32MHz).

**Workaround**

Use STOP0 mode instead of STOP1 mode, there is no MFCLK drift when STOP0 mode is used.

OR

Do not use SYSOSC in the FCL mode (leave FCL disabled) when using STOP1.

## **SYSOSC\_ERR\_02** *SYSOSC Module*

---

**Category**

Functional

**Function**

MFCLK does not work when Async clock request is received in an LPM where SYSOSC was disabled in FCL mode

**Description**

MFCLK will not start to toggle in below scenario:  
 1. FCL mode is enabled and then MFCLK is enabled  
 2. Enter a low power mode where SYSOSC is disabled (SLEEP2/STOP2/STANDBY0/STANDBY1).  
 3. Async request is received from some peripherals which use MFCLK as functional clock. On receiving async request, SYSOSC gets enabled and ulpclk becomes 32MHz. But MFCLK is gated off and it does not toggle at all as the device is still set to the LPM.

**Workaround**

If SYSOSC is using the FCL mode - Do not enable the MFCLK for a peripheral when you're entering a LPM mode which would typically turn off the SYSOSC.

## **SYSOSC\_ERR\_04** *SYSOSC Module*

---

**Category**

Functional

**Function**

SYSOSC accuracy degrades in FCL ON mode when SYSPLL is used

**Description**

When using FCLON for internal oscillator, SYSOSC, accuracy can degrade up to +/-3% when using SYSPLL. The accuracy degradation is due to a synchronization between the 4MHz SYSOSC sampling clock and noise in the system.

**Workaround**

If using the SYSPLL FCL ON mode, use a non-4MHz multiple for the SYSPLL frequency, for example: 78MHz, 79MHz, 81MHz

Do not put the SYSPLL at 16, 32, 48, 40, 64, 80MHz etc.

For 78MHz:

Set SYSPLLCFG1.PDIV = 0x3 and SYSPLLCFG1.QDIV to 38

## **SYSPLL\_ERR\_01** *SYSPLL Module*

---

**Category**

Functional

**Function**

SYSPLL Frequency may not lock to correct frequency when enabled.

**Description**

When setting the SYSPLLEN bit to 1 in SYSCTL HSCLKEN register, the SYSPLL will run the phase locked loop search. The search can potentially fail where the frequency will not be set to the correct value, instead the resultant frequency will be drastically different than the configured frequency.

**SYSPLL\_ERR\_01**

(continued)

**SYSPLL Module****Workaround**

Check the frequency output of the SYSPLL using the Frequency Clock Counter (FCC) anytime the SYSPLLEN bit is set to 1. Once the frequency is correct it will maintain the correct value until disabled and reenabled (SYSPLLEN set to 0 then 1), once reenabled the PLL will re-run the search and the SYSPLL output will need to be rechecked.

Workaround 1: Set FCC with SYSPLLCLK0 as the CLK input and LFCLK as the Trigger source. Run the FCC and check the value for the configured SYSPLL frequency with reference to the LFCLK; for example, with SYSPLL = 80MHz and LFCLK = 32kHz, the resultant FCC count should be  $80,000,000/32,768 = \sim 2441$ . The count will vary depending on the combined clock accuracies, so it is recommended to add a +5% to allowed range. Estimated time for FCC is 30us.

FCC Settings: SYSCTL.GENCLKCFG.FCCTRIGCNT = 0,  
SYSCTL.GENCLKCFG.FCCTRIGSRC = 1, SYSCTL.GENCLKCFG.FCCSELCLK = 4;

If the FCC value is incorrect, disable and reenable the SYSPLL by setting SYSPLLEN to 0 then 1. Rerun the FCC check.

Workaround 2: Output SYSOSC/2 from the CLK\_OUT pin and route the signal into FCC\_IN. Use the SYSPLLCLK0 as the FCC CLK and the FCC\_IN for the trigger source. Run the FCC for 16 Clock cycles, and check the value for the configured SYSPLL frequency with reference to the SYSOSC; for example, with SYSPLL = 80MHz and SYSOSC/2 = 16MHz, the resultant FCC count should be  $80,000,000/16,000,000 * 16 = \sim 80$ . The count will vary depending on the combined clock accuracies, so it is recommended to add a +5% to allowed range. Estimated time for FCC is 1us.

FCC Settings: SYSCTL.GENCLKCFG.FCCTRIGCNT = 0x0F,  
SYSCTL.GENCLKCFG.FCCTRIGSRC = 0, SYSCTL.GENCLKCFG.FCCSELCLK = 4;

If the FCC value is incorrect, disable and reenable the SYSPLL by setting SYSPLLEN to 0 then 1. Rerun the FCC check.

**TIMER\_ERR\_04****TIMER Module****Category**

Functional

**Function**

TIMER re-enable may be missed if done close to zero event

**Description**

When using a TIMER in one shot mode, TIMER re-enable may be missed if done close to zero event. The HW update to the timer enable bit will take a single functional clock cycle. For example, if the timer's clock source is 32.768kHz and clock divider of 3, then it will take  $\sim 100\mu\text{s}$  to have the enable bit set to 0 properly.

**Workaround**

Wait 1 functional clock cycle before re-enabling the timer OR the timer can be disabled first before re-enabling.

Disable the counter with CTRCTL.EN = 0, then re-enable with CTRCTL.EN = 1

**TIMER\_ERR\_06**     ***TIMA and TIMG Module***

---

**Category**

Functional

**Function**

Writing 0 to CLKEN bit does not disable counter

**Description**

Writing 0 to the Counter Clock Control Register(CCLKCTL) Clock Enable bit(CLKEN) does not stop the timer.

**Workaround**

Stop the timer by writing 0 to the Counter Control(CTRCTL) Enable(EN) bit.

**TIMER\_ERR\_07**     ***Initial repeat counter has 1 less period than next repeats Module***

---

**Category**

Functional

**Function**

TIMER

**Description**

When using the timer repeat counter mode, the first repeat will have 1 less count than the subsequent repeats because the following repeat counters will include the transition between 0 and the load value. For example if the TIMx.RCLD = 0x3 then 3 observable zero events would appear on the first repeat counter and 4 observable zero events would appear on the following repeat counter sequences.

**Workaround**

Set the initial RCLD value to 1 more than the expected RCLD, then in the ISR for the Repeat Counter Zero Event (REPC), set the RCLD to the intended RCLD value. For example, if intending to have 4 repeats, set the initial RCLD value to RCLD = 0x5, then in the timer ISR for the REPC interrupt, set RCLD = 0x4. Now all timer repeats will have the same number of zero/load events.

**UART\_ERR\_01**     ***UART Module***

---

**Category**

Functional

**Function**

UART start condition not detected when transitioning to STANDBY1 Mode

**Description**

After servicing an asynchronous fast clock request that was initiated by a UART transmission while the device was in STANDBY1 mode, the device will return to STANDBY1 mode. If another UART transmission begins during the transition back to STANDBY1 mode, the data is not correctly detected and received by the device.

**Workaround**

Use STANDBY0 mode or higher low power mode when expecting repeated UART start conditions.

<b>UART_ERR_02</b>	<b>UART Module</b>
<b>Category</b>	Functional
<b>Function</b>	UART End of Transmission interrupt not set when only TXE is enabled
<b>Description</b>	UART End Of Transmission (EOT) interrupt does not trigger when the device is set for transmit only (CTL0.TXE = 1, CTL0.RXE = 0). EOT successfully triggers when device is set for transmit and receive (CTL0.TXE = 1, CTL0.RXE = 1)
<b>Workaround</b>	Set both CTL0.TXE and CTL0.RXE bits when utilizing the UART end of transmission interrupt. Note that you do not need to assign a pin as UART receive.
<b>UART_ERR_04</b>	<b>UART Module</b>
<b>Category</b>	Functional
<b>Function</b>	Incorrect UART data received with the fast clock request is disabled when clock transitions from SYSOSC to LFOSC
<b>Description</b>	<p>Scenario:</p> <ol style="list-style-type: none"> <li>1. LFCLK selected as functional clock for UART</li> <li>2. Baud rate of 9600 configured with 3x oversampling</li> <li>3. UART fast clock request has been disabled</li> </ol> <p>If the ULPClk changes from SYSOSC to LFOSC in the middle of a UART RX transfer, it is observed that one bit is read incorrectly</p>
<b>Workaround</b>	Enable UART fast clock request while using UART in LPM modes.
<b>UART_ERR_05</b>	<b>UART Module</b>
<b>Category</b>	Functional
<b>Function</b>	Limitation of debug halt feature in UART module
<b>Description</b>	All Tx FIFO elements are sent out before the communication comes to a halt against the expectation of completing the existing frame and halt.
<b>Workaround</b>	Please make sure data is not written into the TX FIFO after debug halt is asserted.
<b>UART_ERR_06</b>	<b>UART Module</b>
<b>Category</b>	Functional

## UART\_ERR\_06

(continued)

### *UART Module*

---

#### Function

Unexpected behavior RTOUT/Busy/Async in UART 9-bit mode

#### Description

UART receive timeout (RTOUT) is not working correctly in multi node scenario, where one UART will act as controller and other UART nodes as peripherals, each peripheral is configured with different address in 9-bit UART mode.

First UART controller communicated with UART peripheral1, by sending peripheral1's address as a first byte and then data, peripheral1 has seen the address match and received the data. Once controller is done with peripheral1, peripheral1 is not setting the RTOUT after the configured timeout period, if controller immediately starts the communication with another UART peripheral (peripheral2) which is configured with different address on the bus. The peripheral1 RTOUT counter is resetting while communication ongoing with peripheral2 and peripheral1 setting its RTOUT only after UART controller is completed the communication with peripheral2.

Similar behavior observed with BUSY and Async request. Busy and Async request is setting even if address does not match while controller communicating with other peripheral on the bus.

#### Workaround

Do not use RTOUT/ BUSY /Async clock request behavior in multi node UART communication where single controller is tied to multiple peripherals.

## UART\_ERR\_07

### *UART Module*

---

#### Category

Functional

#### Function

RTOUT counter not counting as per expectation in IDLE LINE MODE

#### Description

In IDLE LINE MODE in UART, RTOUT counter gets stuck, even when the line is IDLE and FIFO has some elements. This means that RTOUT interrupts will not work in IDLE LINE MODE.

In case of an address mismatch, RTOUT counter is reloaded when it sees toggles on the Rx line.

In case of a multi-responder scenario this could lead to an indefinite delay in getting an RTOUT event when communication is happening between the commander and some other responder.

#### Workaround

Do not enable RTOUT feature when UART module is used either in IDLELINE mode/ multi-node UART application.

## UART\_ERR\_08

### *UART Module*

---

#### Category

Functional

#### Function

STAT BUSY does not represent the correct status of UART module

**UART\_ERR\_08**

(continued)

**UART Module**


---

**Description**

STAT BUSY is staying high even if UART module is disabled and there is data available in TXFIFO.

**Workaround**

Poll TXFIFO status and the CTL0.ENABLE register bit to identify BUSY status.

**UART\_ERR\_10**
**UART Module**


---

**Category**

Functional

**Function**

BUSY bit setting is delayed for UART IrDA mode

**Description**

In IrDA mode, the UART.STAT.BUSY bit is set on the second edge of the IrDA start pulse; which means a whole bit transmission would complete before the BUSY status is properly set. During this time if the software polls the BUSY bit, an incorrect indication of UART not being busy would be observed even when the IrDA start pulse is ongoing. BUSY status will be influenced by the baud rate of the UART, the slower the UART transmission the longer time before BUSY is properly set.

**Workaround**

Delay for the length of a bit transmission before checking the BUSY status. Alternatively, checking for UART.STAT.BUSY == 0x0, then UART.STAT.BUSY == 0x1, is another workaround to make a dynamic delay independent of baud rate or other ISRs.

**UART\_ERR\_11**
**UART Module**


---

**Category**

Functional

**Function**

UART Receive timeout starts counting earlier than expected during the STOP bit transaction

**Description**

During the STOP bit transaction the Receive timeout will start counting in the middle of the STOP bit transaction, which can cause an unintended RTOUT interrupt if the RXTOSEL setting is too small. For example, if the baud rate was 1Mbps, and RXTOSEL was set to 1, the expected RTOUT should happen 1us after the STOP bit transaction, instead the RTOUT interrupt is getting set at 0.5 us.

**Workaround**

The UART.IFLS.RXTOSEL register selects the bit time before the Receive Time out (RTOUT) interrupt will fire. The RXTOSEL value needs to be greater than 1 in order to prevent an early interrupt. The receive timeout time can be calculated as: Receive timeout = (RXTOSEL - 0.5) / Baud Rate

## 7 Trademarks

All trademarks are the property of their respective owners.



## 8 Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

### Changes from November 30, 2025 to December 31, 2025 (from Revision C (November 2025) to Revision D (December 2025))

	Page
• ADC_ERR_10 Function was updated.....	5
• ADC_ERR_10 Description was updated.....	5
• ADC_ERR_10 Workaround was updated.....	5
• CPU_ERR_03 Workaround was updated.....	6
• SYSOSC_ERR_04 Function was updated.....	19
• SYSOSC_ERR_04 Description was updated.....	19
• SYSOSC_ERR_04 Workaround was updated.....	19

## IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES "AS IS" AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you fully indemnify TI and its representatives against any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI's products are provided subject to [TI's Terms of Sale](#), [TI's General Quality Guidelines](#), or other applicable terms available either on [ti.com](#) or provided in conjunction with such TI products. TI's provision of these resources does not expand or otherwise alter TI's applicable warranties or warranty disclaimers for TI products. Unless TI explicitly designates a product as custom or customer-specified, TI products are standard, catalog, general purpose devices.

TI objects to and rejects any additional or different terms you may propose.

Copyright © 2025, Texas Instruments Incorporated

Last updated 10/2025