

**ABSTRACT**

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

**Table of Contents**

<b>1 Functional Advisories</b> .....	<b>1</b>
<b>2 Preprogrammed Software Advisories</b> .....	<b>3</b>
<b>3 Debug Only Advisories</b> .....	<b>3</b>
<b>4 Fixed by Compiler Advisories</b> .....	<b>3</b>
<b>5 Device Nomenclature</b> .....	<b>3</b>
<b>6 Advisory Descriptions</b> .....	<b>4</b>
<b>7 Revision History</b> .....	<b>29</b>

**1 Functional Advisories**

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

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

Errata Number	Rev B
<a href="#">ADC_ERR_01</a>	✓
<a href="#">ADC_ERR_02</a>	✓
<a href="#">ADC_ERR_04</a>	✓
<a href="#">ADC_ERR_05</a>	✓
<a href="#">ADC_ERR_06</a>	✓
<a href="#">BSL_ERR_01</a>	✓
<a href="#">CLK_ERR_01</a>	✓
<a href="#">COMP_ERR_01</a>	✓
<a href="#">COMP_ERR_02</a>	✓
<a href="#">COMP_ERR_03</a>	✓
<a href="#">COMP_ERR_05</a>	✓
<a href="#">CPU_ERR_01</a>	✓
<a href="#">CPU_ERR_02</a>	✓
<a href="#">CPU_ERR_03</a>	✓
<a href="#">DMA_ERR_01</a>	✓
<a href="#">FLASH_ERR_02</a>	✓
<a href="#">FLASH_ERR_03</a>	✓
<a href="#">FLASH_ERR_04</a>	✓
<a href="#">FLASH_ERR_05</a>	✓
<a href="#">FLASH_ERR_06</a>	✓
<a href="#">GPIO_ERR_01</a>	✓
<a href="#">GPIO_ERR_03</a>	✓
<a href="#">I2C_ERR_01</a>	✓

Errata Number	Rev B
<a href="#">I2C_ERR_02</a>	✓
<a href="#">I2C_ERR_03</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>	✓
<a href="#">MATHACL_ERR_01</a>	✓
<a href="#">MATHACL_ERR_02</a>	✓
<a href="#">PMCU_ERR_08</a>	✓
<a href="#">PMCU_ERR_13</a>	✓
<a href="#">PWREN_ERR_01</a>	✓
<a href="#">RST_ERR_01</a>	✓
<a href="#">RTC_ERR_01</a>	✓
<a href="#">SPI_ERR_01</a>	✓
<a href="#">SPI_ERR_02</a>	✓
<a href="#">SPI_ERR_03</a>	✓
<a href="#">SPI_ERR_04</a>	✓
<a href="#">SPI_ERR_05</a>	✓
<a href="#">SPI_ERR_06</a>	✓
<a href="#">SPI_ERR_07</a>	✓
<a href="#">SRAM_ERR_02</a>	✓
<a href="#">SYSOSC_ERR_01</a>	✓
<a href="#">SYSOSC_ERR_02</a>	✓
<a href="#">TIMER_ERR_01</a>	✓
<a href="#">TIMER_ERR_04</a>	✓
<a href="#">TIMER_ERR_06</a>	✓
<a href="#">UART_ERR_01</a>	✓
<a href="#">UART_ERR_02</a>	✓
<a href="#">UART_ERR_04</a>	✓
<a href="#">UART_ERR_05</a>	✓
<a href="#">UART_ERR_06</a>	✓
<a href="#">UART_ERR_07</a>	✓
<a href="#">UART_ERR_08</a>	✓
<a href="#">UART_ERR_09</a>	✓
<a href="#">VREF_ERR_01</a>	✓
<a href="#">VREF_ERR_02</a>	✓
<a href="#">VREF_ERR_03</a>	✓
<a href="#">WWDT_ERR_01</a>	✓
<a href="#">WWDT_ERR_02</a>	✓

## 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.

## 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.

## 6 Advisory Descriptions

### ADC\_ERR\_01 *ADC Module*

---

**Category**

Functional

**Function**

ADC is unable to trigger fast clock in STANDBY1 mode

**Description**

When the device is operating in STANDBY1 mode, the ADC module may not correctly assert an asynchronous fast clock request when it is triggered through the event system (for example, when an event publisher such as a timer generates an event to the ADC via the ADC's event subscriber port).

**Workaround**

Use STANDBY0 or higher power modes when triggering ADC conversions via the event fabric.

### ADC\_ERR\_02 *ADC Module*

---

**Category**

Functional

**Function**

ADC does not release the fast clock request in between periodical triggers

**Description**

When the ADC is set up in repeat mode and triggered periodically via the event fabric, it does not release its fast clock request in between triggers. IF so, the ADC will hold the clock request AND consume extra power.

**Workaround**

Configuring the ADC in single mode (run to completion with ADC disabling at end of channel/sequence), AND THEN using a software interrupt at the end of the ADC sequence to re-enable the ADC to wait for the next trigger.

### ADC\_ERR\_04 *ADC Module*

---

**Category**

Functional

**Function**

Increased current in low power mode when ADC is in Repeat mode

**Description**

When the ADC is configured to Repeat Sequence or Repeat Single mode with EVENT trigger in low power mode (LPM), the ADC module will consume additional current until disabled.

**Workaround**

Configure ADC to be in Single or Sequence mode (no Repeat), and use an interrupt after conversion to re-enable ADC for next trigger.

<b>ADC_ERR_05</b> <b>ADC Module</b>	
<b>Category</b>	Functional
<b>Function</b>	HW Event generated before enabling IP will stay in queue
<b>Description</b>	When ADC is configured in HW event trigger mode and trigger is generated before enabling, ADC will stay in queue. Once ADC is enabled, it will trigger conversion.
<b>Workaround</b>	After configuring ADC in HW trigger mode, enable ADC first before giving external trigger.

<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>The ADC may have errors at a rate as high as 1 in 2M conversions in 12-bit mode.</p> <p>When a conversion error occurs, it will be a +/- 64LSB random jump in the digital output of the ADC without a corresponding change in the ADC input voltage.</p> <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>
<b>Workaround</b>	<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>BSL_ERR_01</b>	<b>BSL Module</b>
<b>Category</b>	Functional
<b>Function</b>	Invoking the BSL through application software will fail under certain conditions
<b>Description</b>	<p>BSL fails to start through invocation within an application (BSL Software Invoke) due to SRAM error code. After a reset, the device will go back to the application instead of invoking BSL due to the error.</p> <p>This errata does not apply to BSL invocation through hardware invoke methods</p>
<b>Workaround</b>	<p>For applications needing a software invocation of the BSL, clear the entirety of SRAM with assembly code before resetting device. The MSPM0 bsl_software_invoke example within the MSPM0 SDK v 1.20.01.xx or higher contains an example fix within the "bsl_software_invoke" example.</p>

## **COMP\_ERR\_01**      **COMP Module**

---

### **Category**

Functional

### **Function**

Comparator output keeps toggling in STANDBY0 mode

### **Description**

When the comparator inverting input mux (IMSEL) is set to 0 (COMP0\_IN0-), and the device operating mode is set to STANDBY0, the comparator output may toggle unexpectedly regardless of the applied input voltages to the comparator.

### **Workaround**

If utilizing comparator in STANDBY0, and applying an external voltage to the negative terminal, utilize channel 1 of the comparator (IMSEL = 1).  
If utilizing comparator in STANDBY0, and applying an internal reference voltage (DAC8, VREF, etc.), set IMSEL to a value other than 0 (IMSEL 0).

## **COMP\_ERR\_02**      **COMP Module**

---

### **Category**

Functional

### **Function**

COMP output toggles when DACCODEx is used for hysteresis

### **Description**

IF using the 8-bit DAC within the COMP module as an input to the COMP,  
AND DAC output switches between values placed in DACCODEx,  
THEN the COMP output toggles as if immediately crossing the new reference point.

This can happen regardless of the setting of the COMPx.CTL2.DACCTL bit.

This is most commonly seen in applications that utilize the two DACCODEx codes for custom or asymmetrical hysteresis for the COMP module.

### **Workaround**

Utilize the established hysteresis values provided in COMPx.CTL1.HYST register bits.

## **COMP\_ERR\_03**      **COMP Module**

---

### **Category**

Functional

### **Function**

COMP hysteresis features are non-functional when using input exchange feature

### **Description**

When using hysteresis features of the COMP module, and exchanging the inputs of the COMP ( COMPx.CTL1.EXCH = 1 ), the COMP module becomes unstable.

**COMP\_ERR\_03**

(continued)

**COMP Module****Workaround**

Do not apply internal hysteresis methods when utilizing COMP module in input exchange feature.

**COMP\_ERR\_05****COMP Module****Category**

Functional

**Function**

Comparator output will set the rising and falling interrupt when comparator is enabled

**Description**

Comparator will set the rising and falling when the comparator is enabled.

**Workaround**

1. Clear the CPU interrupts by using ICLR bit.  
ICLR will not work for clearing generic events. Follow below steps to clear COMP generic events (below are DriverLib functions, you can see the bits manipulation by looking in the function contents in our MSPM0 SDK)

- Before COMP is enabled, configure COMP publisher with some dummy ID.  
`DL_COMP_setPublisherChanID(COMP_0_INST, 0); // remove the actual publisher`
- `DL_COMP_enableEvent(COMP_0_INST, (DL_COMP_EVENT_OUTPUT_EDGE)); // Enable the COMP events in IMASK`
- `DL_COMP_enable(COMP_0_INST); // Enable the COMP module, this step clearing the events in RIS.`
- `DL_COMP_disableEvent(COMP_0_INST, (DL_COMP_EVENT_OUTPUT_EDGE)); // Disable the COMP events by clearing in IMASK`
- `DL_COMP_setPublisherChanID(COMP_0_INST, COMP_0_INST_PUB_CH); // configure the actual publisher`
- `DL_COMP_enableEvent(COMP_0_INST, (DL_COMP_EVENT_OUTPUT_EDGE)); // Re-enable COMP events in IMASK`

Or

Read the interrupt after the comparator is enabled, knowing that the first interrupt happened due to comparator being enabled.

**CLK\_ERR\_01****CLK Module****Category**

Functional

**Function**

Hardfault observed when HFXT 4MHz used as MCLK with debugger connected

**Description**

When MCLK is configured to be HFXT 4MHz with the configurations of: HFXTRSEL set to 0 (recommended value for crystal of 4 to 8 MHz range), the program can jump to hard fault or NMI at random.

**Workaround**

Set HFXTRSEL value to 1 or higher (8MHz to 48MHz range) when debugging the code with HFXT 4MHz as the MCLK.

<b>CPU_ERR_01</b>	<b><i>CPU Module</i></b>
<b>Category</b>	Functional
<b>Function</b>	CPU cache content can get corrupted
<b>Description</b>	Cache corruption can occur when switching between accessing Main flash memory, and other non-volatile memory regions such as NONMAIN or Calibration data areas.
<b>Workaround</b>	Use the following procedure to access areas outside main memory safely: 1. Disable the cache by setting CTL.ICACHE to "0". 2. Perform needed access to memory area. 3. Re-enable cache by setting CTL.ICACHE to "1".
<b>CPU_ERR_02</b>	<b><i>CPU Module</i></b>
<b>Category</b>	Functional
<b>Function</b>	Limitation of disabling prefetch feature for CPUSS
<b>Description</b>	CPU prefetch disable will not take effect if there is a pending flash memory access.
<b>Workaround</b>	Disable prefetch, 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.
<b>CPU_ERR_03</b>	<b><i>CPU Module</i></b>
<b>Category</b>	Functional
<b>Function</b>	Prefetcher can cause data integrity issues when transitioning into SLEEP mode
<b>Description</b>	When transitioning into SLEEP0 the prefetcher can erroneously fetch incorrect data (all 0's). When coming out of sleep mode, 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.
<b>Workaround</b>	Disable prefetcher before entering SLEEP.
<b>DMA_ERR_01</b>	<b><i>DMA Module</i></b>
<b>Category</b>	Functional

## DMA\_ERR\_01

(continued)

### **DMA Module**

---

#### **Function**

DMA or CPU may lose operation when simultaneously accessing peripheral register across clock resource

#### **Description**

DMA and CPU are sourced from MCLK. When MCLK is running at a higher frequency than ULPCLK, DMA or CPU may lose operation when simultaneously accessing the peripheral register sourced from ULPCLK, which include all PD0 peripherals. The accessed peripheral or register does not have to be the same. Note: ADC is a PD0 peripheral, while DMA or CPU accessing SVT\_MEMRES or SVT\_FIFODATA of ADC has no restriction. Example: Assume that DMA access UART0 register, for example DMA write data to TXDATA, if CPU also access PD0 peripheral during DMA operation, for example CPU read data from TIMG0 CTR, then DMA or CPU may lose the data.

#### **Workaround**

CPU and DMA should not be accessing PD0 peripherals at the same time when MCLK is running at a higher frequency than ULPCLK.

## FLASH\_ERR\_02

### **FLASH Module**

---

#### **Category**

Functional

#### **Function**

Debug disable in NONMAIN can be re-enable using default password

#### **Description**

If debug is disabled in NONMAIN configuration (DEBUGACCESS = 0x5566), the device can still be accessed using the default password.

#### **Workaround**

1. Set the DEBUGACCESS to Debug Enabled with Password option (DEBUGACCESS = 0xCCDD) and provide a unique password in the PWDDEBUGLOCK field. For higher security, it is recommended to have device-unique passwords that are cryptographically random. This would allow debug access with the right 128-bit password but can still allow some debug commands and access to the CFG-AP and SEC-AP.
2. Disable the physical SW Debug Port entirely by disabling SWDP\_MODE. This fully prevents any debug access or requests to the device, but may affect Failure Analysis and return flows.

## 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.

**FLASH\_ERR\_03**

(continued)

***FLASH Module***

---

**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 gets reported in the SYSCTL-&gt;DEDERRADDR

**Description**

When a FLASHDED error appears the data truncates 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 can 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 is 0x41C4013C.

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

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

---

**Category**

Functional

**Function**

DEDERRADDR may have incorrect reset value

**Description**

The reset value of the SYSCTL->DEDERRADDR may 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\_06 *Flash Module*

---

### Category

Functional

### Function

CPU AND DMA are not able to access the flash at the same time

### Details

CPU AND DMA cannot concurrently access the flash; this simultaneous access results in reading incorrect data from the flash.

### Workaround

Do not access the flash via CPU AND DMA concurrently. In case of typical Flash operations of program/erase operation, read verify/ blank verify operations, as well as the DMA reads from flash; software needs to make sure that CPU does not access flash while the flash is busy. This can be provided by putting code in SRAM while a flash operation is on-going or move the flash memory into SRAM for data the DMA needs to read.

## GPIO\_ERR\_01 *GPIO Module*

---

### Category

Functional

### Function

GPIO wakeup edges can be lost in STANDBY mode

### Description

After waking up once through a single GPIO edge, subsequent GPIO wakeup edges can be missed in STANDBY/STOP/SLEEP modes.

Case 1:

STANDBY0 wakeup - IF the MCU is set into STANDBY/STOP/SLEEP mode with an IO in "wake" state AND one sets the IO back to the "non-wake" state for < 3 LFCLK cycles and THEN asserts it again, the next wakeup edge will not be detected.

Case 2:

STANDBY1 wakeup - IF a GPIO edge is used to wakeup AND the GPIO pulse is still active when the device returns to STANDBY1, THEN the device will not detect any subsequent wakeup edges.

### Workaround

Case 1:

Make sure that the GPIO is de-asserted while the device is in active mode

OR

Ensure GPIO wakeup pulse is longer than 3 LFCLK cycles

Case 2:

Set GPIO wakeup edge to both falling and rising edges

OR

Ensure GPIO wakeup pulse is not active before entering STANDBY1

## GPIO\_ERR\_03 *GPIO and DEBUGSS Module*

---

### Category

Functional

## GPIO\_ERR\_03

(continued)

### GPIO and DEBUGSS Module

#### Function

On a debugger read to GPIO EVENT0 IIDX, interrupt is cleared.

#### Description

EVENT0's IIDX of GPIO, on a debugger read is treated as a CPU read and interrupt is getting cleared.

#### Workaround

During the debug, the IIDX of event0 can be read by software reading RIS.

## I2C\_ERR\_01

### I2C Module

#### Category

Functional

#### Function

I2C module may hold the SDA line in SBMUS mode when a SMBUS quick command is issued

#### Description

When the I2C module is target mode and configured for SBMUS, IF the bus controller issues an SMBUS quick command addressed to the device (an I2C START condition followed by a 7-bit address, 1-bit R/W signal, 1-bit ACK, and an I2C STOP condition) with the R/W bit set to read, THEN the I2C module may attempt to pull the SDA line low at the same time that the bus controller is attempting to signal the I2C STOP condition, preventing the STOP condition from completing successfully.

#### Workaround

Load data into the I2C module transmit FIFO with the MSB set to 1 before the address ACK is completed to prevent the I2C module from driving the SDA line low. This will allow the bus controller to issue the STOP condition successfully and complete the SMBUS quick command.

## I2C\_ERR\_02

### I2C Module

#### Category

Functional

#### Function

I2C quick command read mode only works with specific conditions

#### Description

I2C quick command in read mode works only if TXFIFO has data with MSB set to 1 and clock stretching is disabled.

#### Workaround

Provide a dummy data in the TXFIFO with MSB set to 1 AND disable the clock stretching (CLKSTRETCH = 0).

## I2C\_ERR\_03

### I2C Module

#### Category

Functional

**I2C\_ERR\_03**

(continued)

**I2C Module**

---

**Function**

I2C peripheral mode cannot wake up device when sourced from MFCLK

**Description**

IF I2C module is configured in peripheral mode  
AND I2C is clocked from MFCLK (Middle Frequency Clock)  
AND device is placed in STOP2 or STANDBY0/1 power modes,  
THEN I2C fails to wakeup the device when receiving data.

**Workaround**

Set I2C to be clocked by BUSCLK instead of MFCLK, if needing low power wakeup upon receiving data in I2C peripheral mode.

<b>I2C_ERR_04</b>	<b><i>I2C Module</i></b>
<b>Category</b>	Functional
<b>Function</b>	When SCL low occurs and target wakeup is enabled, device may clock stretch indefinitely.
<b>Description</b>	When the device is in target mode and SCL is pulled low due to clock stretching or external grounding, if the controller releases the bus, the I2C target is unable to release the clock stretch.
<b>Workaround</b>	Disable the target wakeup enable bit (SWUEN).
<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.

## **I2C\_ERR\_06** (continued)

### ***I2C Module***

---

#### **Workaround**

Do not use SMBus High timeout feature when I2C clock is less than 24KHz onwards.

## **I2C\_ERR\_07**

### ***I2C Module***

---

#### **Category**

Functional

#### **Function**

Back to back controller control register writes will cause I2C to not start.

#### **Description**

Back-to-Back CTR register writes will cause the next CTR.START to not properly cause the start condition.

#### **Workaround**

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

## **I2C\_ERR\_08**

### ***I2C Module***

---

#### **Category**

Functional

#### **Function**

FIFO Read directly after RXDONE interrupt will cause 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 guarantee to have the latest data. I2C CLK will be 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 atypical 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 atypical I2C speeds, wait at least 1 I2C CLK cycle before reading the ADDRMATCH bit.

## **I2C\_ERR\_10**

### ***I2C Module***

---

#### **Category**

Functional

**I2C\_ERR\_10**  
(continued)

***I2C Module***


---

**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).

**MATHACL\_ERR\_0**  
**1**
***MATHACL Module***


---

**Category**

Functional

**Function**

MATHACL status error bit does not get cleared

**Description**

If there is a status error generated by the mathacl (ex. divide by 0), then the status register never gets cleared.

**Workaround**

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\_06**
***PMCU Module***


---

**Category**

Functional

**Function**

CPU AND DMA are not able to access the flash at the same time

**Description**

CPU AND DMA cannot concurrently access the flash; for instance, this simultaneous access results in reading incorrect data from the flash during the flash erase operation. The issue can be seen whenever HREADY of the pulled low to CPU due to an ongoing

## PMCU\_ERR\_06

(continued)

### PMCU Module

---

DMA access, program/ erase operation, read Verify/ blank verify operations, basically anything other than CPU.

#### Workaround

Do not access the flash via CPU AND DMA concurrently. In case of program/ erase operation, read Verify/ blank verify operations, software needs to make sure that CPU does not access flash. This can be specified by putting code in SRAM while a flash operation is on-going.

## PMCU\_ERR\_08

### PMCU Module

---

#### Category

Functional

#### Function

More wakeup time than expected when trigger is given to device while it is transitioning to LPM

#### Description

If wakeup signal is given to device when it is transitioning to low power mode, an additional wakeup time of approximately 3us will occur.

#### Workaround

No workaround.

## PMCU\_ERR\_13

### PMCU Module

---

#### Category

Functional

#### Function

MCU may get stuck while waking up from STOP2 & STANDBY0 at certain scenario

#### Description

When there is a pending prefetch access before device transitions to STOP2 & STANDBY0. A pending prefetch access such as a timer has just run to completion and DMA has received the event from the GPIO, in that scenario neither DMA transfer happens nor timer ISR execution takes place as well as CPU gets stuck. This issue arises when WFI instruction is half word aligned, wait state of device is 2 and there is a pending prefetch access before device transitions to LPM.

#### Workaround

Before going to LPM, customer can disable the prefetch and run some dummy instructions (like shutdown register read or any peripheral read) which doesn't access prefetch so that prefetch can get disabled and doesn't cause the device to hang when waking up from LPM as no prefetch access will be pending.

## PWREN\_ERR\_01

### GPIO Module

---

#### Category

Functional

#### Function

Peripheral registers are still accessible after disabling PWREN register

## PWREN\_ERR\_01

(continued)

### GPIO Module

#### Description

When disabling the power of a peripheral by setting the PWREN register to 0, the peripherals registers may appear to retain data values if read. Reading or writing to the registers when PWREN is 0 has no affect as the peripheral has no effect.

The following peripherals are affected: comparator (COMP), operational amplifier (OPA), TimerA, TimerG, general-purpose input/output (GPIO), and windowed watchdog timer (WWDT), AES and TRNG.

#### Workaround

When the PWREN register of the peripheral is set to 0, the values of the associated registers should be disregarded or considered invalid.

## 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 should 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\_01

### SPI Module

#### Category

Functional

## **SPI\_ERR\_01**

(continued)

### ***SPI Module***

---

#### **Function**

SPI Parity bit is not functional

#### **Description**

SPI hardware parity modes are not functional.

#### **Workaround**

Do not enable hardware parity via the PTEN or PREN bit in the CTL1 register of the SPI module. Parity computation and checking may be implemented with application software.

## **SPI\_ERR\_02**

### ***SPI Module***

---

#### **Category**

Functional

#### **Function**

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

#### **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\_03**

### ***SPI Module***

---

#### **Category**

Functional

#### **Function**

When configured as peripheral for a multi-peripheral application, received data will have a right shift

#### **Description**

In multi-peripheral scenario, SPI controller first communicates with peripheral0 and then communicates with peripheral1. After finishing communication with peripheral1, the controller again communicates with peripheral0. During the second communication with peripheral0, received data of peripheral0 will have a right shift in the first frame. The peripheral0 is getting first data as 0x3B when the controller sent data 0x76.

#### **Workaround**

To support multi peripheral scenario CSCLR needs to be enabled at peripheral end to reset it's RX and TX the bit counters, when there is no active communication happening with that peripheral (CS of that peripheral will be disabled).

<b>SPI_ERR_04</b>	<b><i>SPI Module</i></b>
<b>Category</b>	Functional
<b>Function</b>	IDLE/BUSY status toggle after each frame receive when SPI peripheral is in only receive mode.
<b>Description</b>	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.
<b>Workaround</b>	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.
<b>SPI_ERR_05</b>	<b><i>SPI Module</i></b>
<b>Category</b>	Functional
<b>Function</b>	SPI Peripheral Receive Timeout interrupt is setting irrespective of RXFIFO data
<b>Description</b>	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.
<b>Workaround</b>	Disable the RXTIMEOUT after the last packet is received (this can be done in the ISR) and re-enable when SPI communication starts again.
<b>SPI_ERR_06</b>	<b><i>SPI Module</i></b>
<b>Category</b>	Functional
<b>Function</b>	IDLE/BUSY status does not reflect the correct status of SPI IP when debug halt is asserted
<b>Description</b>	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
<b>Workaround</b>	Don't use IDLE/BUSY status when SPI IP is halted.
<b>SPI_ERR_07</b>	<b><i>SPI Module</i></b>
<b>Category</b>	Functional

## **SPI\_ERR\_07** (continued)

### ***SPI Module***

---

#### **Function**

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

#### **Description**

When SPH = 0 and device is configured as the SPI peripheral: if there is a write to the TXFIFO WHILE there is a read request, then an underflow event can not be generated as the read/write request is happening simultaneously.

#### **Workaround**

Customer must verify that TXFIFO on peripheral can never be empty when the controller is addressing the peripheral. Additionally, data checking strategies, like CRC, can be used to verify the packets were sent properly.

## **SRAM\_ERR\_02**

### ***SRAM Module***

---

#### **Category**

Functional

#### **Function**

SRAM Error address returns value with the most significant byte missing

#### **Description**

SRAM DEDERRADDR will return the wrong value, the most significant byte gets truncated. For example a parity error at 0x20001234 would return the parity error address as 0x00001234.

#### **Workaround**

Perform a bit wise OR operation with the SRAM origin and the SYSCCTRL->DEDERRADDR return value, when an SRAM DED or SRAM Parity fault is triggered. View the device datasheet for SRAM Memory start locations, the linker file will also have the origin of the SRAM address.

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

---

#### **Category**

Functional

#### **Function**

MFCLK drift when using SYSOSC FCL together with STOP1 mode

#### **Description**

When MFCLK is enabled AND SYSOSC is using the frequency correction loop (FCL) mode AND the STOP1 low power operating mode is used, THEN the MFCLK may drift by 2 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**

Workaround1: Use STOP0 mode instead of STOP1 mode. There is no MFCLK drift when STOP0 mode is used. Workaround2: 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.

## TIMER\_ERR\_01 *TIMx Module*

---

### Category

Functional

### Function

Capture mode captures incorrect value when using hardware event to start timer

### Description

When using any timer instance in capture mode, starting the timer using a zero (ZCOND) or load (LCOND) condition causes the timer to capture the zero or load value instead of the captured value in the respective TIMx.CC register. This affects periodic use cases such as period and pulse width capture.

### Workaround

Use the below software flow to calculate the period or pulse width. See the `timx_timer_mode_capture_duty_and_period` in the MSPM0-SDK for an example of the workaround.

1. Disable ZCOND or LCOND by setting to 0h.
2. When a capture occurs, the capture value is correctly captured in TIMx.CC
3. Restart the timer by setting TIMx.CTR to the reload value (load or 0).

## TIMER\_ERR\_04 *TIMER Module*

---

### Category

Functional

### Function

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

### Description

When using a GP TIMER in one shot mode and CLKDIV.RATIO is not 0, TIMER re-enable may be missed if done close to zero event.

## **TIMER\_ERR\_04**

(continued)

### ***TIMER Module***

---

#### **Workaround**

TIMER can be disabled first before re-enabling.

## **TIMER\_ERR\_06**

### ***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.

## **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.

## **UART\_ERR\_02**

### ***UART Module***

---

#### **Category**

Functional

#### **Function**

UART End of Transmission interrupt not set when only TXE is enabled

#### **Description**

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)

#### **Workaround**

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.

## UART\_ERR\_04 *UART Module*

---

### Category

Functional

### Function

Incorrect UART data received with the fast clock request is disabled when clock transitions from SYSOSC to LFOSC

### Description

Scenario:

1. LFCLK selected as functional clock for UART
2. Baud rate of 9600 configured with 3x oversampling
3. UART fast clock request has been disabled

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

### Workaround

Enable UART fast clock request while using UART in LPM modes.

## UART\_ERR\_05 *UART Module*

---

### Category

Functional

### Function

Limitation of debug halt feature in UART module

### Description

All Tx FIFO elements are sent out before the communication comes to a halt against the expectation of completing the existing frame and halt.

### Workaround

Please ensure data is not written into the TX FIFO after debug halt is asserted.

## UART\_ERR\_06 *UART Module*

---

### Category

Functional

### 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

## UART\_ERR\_06 (continued)

### **UART Module**

---

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

#### **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\_09

### **UART Module**

---

#### **Category**

Functional

#### **Function**

UART ADDR\_MATCH may not be set in time for the read when running at slow UART speeds.

**UART\_ERR\_09**  
(continued)

---

**UART Module**


---

**Description**

During an Address match interrupt, when the code jumps into ISR and reads the FIFO. The UART is not able to receive the data which is sent as the address on the RX line, due to the address match interrupt being generated before the STOP bit.

**Workaround**

Wait 1 UART CLK cycle before reading the data to allow the ADDR\_MATCH to be set.

**VREF\_ERR\_01**


---

**VREF Module**


---

**Category**

Functional

**Function**

VREF READY bit does not get cleared after disabling VREF

**Description**

The first time the VREF module is enabled after a SYSRST, the VREF READY bit can be used for its intended functionality. If the VREF module is disabled in application, the VREF READY bit does not get cleared. As a result of this errata, subsequent enabling of the VREF module cannot use the VREF READY bit to indicate the VREF module is stable.

**Workaround**

When re-enabling the VREF module in application, utilize a TIMER module to wait the maximum VREF startup time, before utilizing the VREF module. Please see the device datasheet for VREF startup time.

**VREF\_ERR\_02**


---

**VREF Module**


---

**Category**

Functional

**Function**

Switch VREF from 2.5V to 1.4V mode has a very low slew rate

**Description**

VREF configuration change from 2.5V to 1.4V mode has a very slow slew rate.

**Workaround**

Use the below procedure to switch VREF mode. 1.Disable VREF by using the ENABLE bit in the CTL0 register before changing configuration to 1.4V mode. 2.Configure pin PA23(VREF+) as GPIO output and drive this pin to logic low for 100us. A 1uF external capacitor is assumed on the VREF+ pin 3.Re-enable VREF in 1.4V by setting the BUGCONFIG bit in the CTL0 register to 1. With this procedure, the time required to switch from 2.5V to 1.4V mode is 200us.

**VREF\_ERR\_03**


---

**VREF Module**


---

**Category**

Functional

**Function**

Large inrush current from VDD to charge external 1uF cap.

## VREF\_ERR\_03

(continued)

### VREF Module

#### Description

During power-up there is a large inrush current of  $\approx 70\text{mA}$  from VDD to charge the external  $1\mu\text{F}$  cap on VREF. If there are frequent power cycling of the VREF then damage can occur.

#### Workaround

Pre-charge the VREF+ capacitor by using the GPIO before enabling VREF. This provides the capacitor is charged more than required. Then enable the VREF module and disable the GPIO pull-up.

## WWDT\_ERR\_01

### WWDT Module

#### Category

Functional

#### Function

Watchdog timer 1 (WWDT1) event always does a SYSRST.

#### Description

WWDT1 event always does a SYSRST irrespective of the SYSTEMCFG.WWDTLP1RSTDIS bit setting. Therefore, WWDT1 cannot trigger "only NMI" events.

#### Workaround

Use WWDT0 for NMI purposes.

## WWDT\_ERR\_02

### WWDT Module

#### Category

Functional

#### Function

Window Watchdog Timer 1 (WWDT1) does not create a reset cause

#### Description

After a reset caused by WWDT1, the SYSCCTL.RSTCAUSE register does not accurately portray that WWDT1 caused the reset.

#### Workaround

None.

## 7 Revision History

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

### Changes from November 30, 2024 to July 7, 2025 (from Revision B (December 2024) to Revision C (July 2025))

	Page
• COMP_ERR_03 Description was updated.....	7
• COMP_ERR_03 Workaround was updated.....	7
• COMP_ERR_05 Category was updated.....	8
• COMP_ERR_05 Module was updated.....	8
• COMP_ERR_05 Function was updated.....	8
• COMP_ERR_05 Description was updated.....	8

• COMP_ERR_05 Workaround was updated.....	8
• CPU_ERR_01 Workaround was updated.....	9
• CPU_ERR_01 Description was updated.....	9
• CPU_ERR_02 Category was updated.....	9
• CPU_ERR_02 Module was updated.....	9
• CPU_ERR_02 Function was updated.....	9
• CPU_ERR_02 Description was updated.....	9
• CPU_ERR_02 Workaround was updated.....	9
• CPU_ERR_03 Category was updated.....	9
• CPU_ERR_03 Module was updated.....	9
• CPU_ERR_03 Function was updated.....	9
• CPU_ERR_03 Description was updated.....	9
• CPU_ERR_03 Workaround was updated.....	9
• FLASH_ERR_02 Category was updated.....	10
• FLASH_ERR_02 Module was updated.....	10
• FLASH_ERR_02 Function was updated.....	10
• FLASH_ERR_02 Description was updated.....	10
• FLASH_ERR_02 Workaround was updated.....	10
• FLASH_ERR_03 Category was updated.....	10
• FLASH_ERR_03 Module was updated.....	10
• FLASH_ERR_03 Function was updated.....	10
• FLASH_ERR_03 Description was updated.....	10
• FLASH_ERR_03 Workaround was updated.....	10
• FLASH_ERR_05 Category was updated.....	12
• FLASH_ERR_05 Module was updated.....	12
• FLASH_ERR_05 Function was updated.....	12
• FLASH_ERR_05 Description was updated.....	12
• FLASH_ERR_05 Workaround was updated.....	12
• I2C_ERR_01 Description was updated.....	14
• I2C_ERR_01 Workaround was updated.....	14
• I2C_ERR_01 Function was updated.....	14
• I2C_ERR_07 Category was updated.....	17
• I2C_ERR_07 Module was updated.....	17
• I2C_ERR_07 Function was updated.....	17
• I2C_ERR_07 Description was updated.....	17
• I2C_ERR_07 Workaround was updated.....	17
• I2C_ERR_08 Category was updated.....	17
• I2C_ERR_08 Module was updated.....	17
• I2C_ERR_08 Function was updated.....	17
• I2C_ERR_08 Workaround was updated.....	17
• I2C_ERR_09 Category was updated.....	17
• I2C_ERR_09 Module was updated.....	17
• I2C_ERR_09 Function was updated.....	17
• I2C_ERR_09 Description was updated.....	17
• I2C_ERR_09 Workaround was updated.....	17
• I2C_ERR_10 Category was updated.....	17
• I2C_ERR_10 Module was updated.....	17
• I2C_ERR_10 Function was updated.....	17
• I2C_ERR_10 Description was updated.....	17
• I2C_ERR_10 Workaround was updated.....	17
• MATHACL_ERR_01 Category was updated.....	18
• MATHACL_ERR_01 Module was updated.....	18
• MATHACL_ERR_01 Function was updated.....	18
• MATHACL_ERR_01 Description was updated.....	18
• MATHACL_ERR_01 Workaround was updated.....	18

• MATHACL_ERR_02 Category was updated.....	18
• MATHACL_ERR_02 Module was updated.....	18
• MATHACL_ERR_02 Workaround was updated.....	18
• MATHACL_ERR_02 Function was updated.....	18
• MATHACL_ERR_02 Description was updated.....	18
• PMCU_ERR_13 Category was updated.....	19
• PMCU_ERR_13 Module was updated.....	19
• PMCU_ERR_13 Function was updated.....	19
• PMCU_ERR_13 Workaround was updated.....	19
• PMCU_ERR_13 Description was updated.....	19
• RST_ERR_01 Category was updated.....	20
• RST_ERR_01 Module was updated.....	20
• RST_ERR_01 Workaround was updated.....	20
• RST_ERR_01 Function was updated.....	20
• RST_ERR_01 Description was updated.....	20
• SPI_ERR_04 Description was updated.....	22
• SPI_ERR_04 Workaround was updated.....	22
• SPI_ERR_05 Description was updated.....	22
• SPI_ERR_05 Workaround was updated.....	22
• SPI_ERR_06 Category was updated.....	22
• SPI_ERR_06 Module was updated.....	22
• SPI_ERR_06 Function was updated.....	22
• SPI_ERR_06 Workaround was updated.....	22
• SPI_ERR_06 Description was updated.....	22
• SPI_ERR_07 Category was updated.....	22
• SPI_ERR_07 Module was updated.....	22
• SPI_ERR_07 Function was updated.....	22
• SPI_ERR_07 Description was updated.....	22
• SPI_ERR_07 Workaround was updated.....	22
• SRAM_ERR_02 Category was updated.....	23
• SRAM_ERR_02 Module was updated.....	23
• SRAM_ERR_02 Description was updated.....	23
• SRAM_ERR_02 Function was updated.....	23
• SRAM_ERR_02 Workaround was updated.....	23
• SYSOSC_ERR_02 Description was updated.....	24
• SYSOSC_ERR_02 Workaround was updated.....	24
• TIMER_ERR_04 Module was updated.....	24
• TIMER_ERR_04 Description was updated.....	24
• UART_ERR_04 Category was updated.....	26
• UART_ERR_04 Module was updated.....	26
• UART_ERR_04 Function was updated.....	26
• UART_ERR_04 Description was updated.....	26
• UART_ERR_04 Workaround was updated.....	26
• UART_ERR_05 Category was updated.....	26
• UART_ERR_05 Module was updated.....	26
• UART_ERR_05 Function was updated.....	26
• UART_ERR_05 Description was updated.....	26
• UART_ERR_05 Workaround was updated.....	26
• UART_ERR_06 Category was updated.....	26
• UART_ERR_06 Module was updated.....	26
• UART_ERR_06 Function was updated.....	26
• UART_ERR_06 Workaround was updated.....	26
• UART_ERR_06 Description was updated.....	26
• UART_ERR_07 Category was updated.....	27
• UART_ERR_07 Module was updated.....	27

• UART_ERR_07 Function was updated.....	27
• UART_ERR_07 Workaround was updated.....	27
• UART_ERR_07 Description was updated.....	27
• UART_ERR_08 Category was updated.....	27
• UART_ERR_08 Module was updated.....	27
• UART_ERR_08 Function was updated.....	27
• UART_ERR_08 Description was updated.....	27
• UART_ERR_08 Workaround was updated.....	27
• UART_ERR_09 Category was updated.....	27
• UART_ERR_09 Module was updated.....	27
• UART_ERR_09 Function was updated.....	27
• UART_ERR_09 Description was updated.....	27
• UART_ERR_09 Workaround was updated.....	27
• VREF_ERR_03 Category was updated.....	28
• VREF_ERR_03 Module was updated.....	28
• VREF_ERR_03 Function was updated.....	28
• VREF_ERR_03 Description was updated.....	28
• VREF_ERR_03 Workaround was updated.....	28

## IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), 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 will 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](#) or other applicable terms available either on [ti.com](https://www.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.

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

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265  
Copyright © 2025, Texas Instruments Incorporated