

**ABSTRACT**

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

Table of Contents

1 Errata notes	1
2 Functional Advisories	1
3 Preprogrammed Software Advisories	3
4 Debug Only Advisories	3
5 Fixed by Compiler Advisories	3
6 Device Nomenclature	3
6.1 Device Symbolization and Revision Identification	4
7 Advisory Descriptions	5
8 Revision History	35

1 Errata notes

The following notes should be considered before reading the erratas list and entries:

- ADC_ERR_04 does no longer affect the device
- FLASH_ERR_04 does no longer affect the device

2 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 C	Rev D
ADC_ERR_01	✓	✓
ADC_ERR_02	✓	✓
ADC_ERR_03	✓	✓
ADC_ERR_04	✓	
ADC_ERR_05	✓	✓
ADC_ERR_06	✓	✓
ADC_ERR_11	✓	✓
COMP_ERR_01	✓	✓
COMP_ERR_02	✓	✓
COMP_ERR_03	✓	✓
COMP_ERR_05	✓	✓
CPU_ERR_01	✓	✓
CPU_ERR_02	✓	✓
CPU_ERR_03	✓	✓
CPU_ERR_04	✓	✓
CRC/CRCP_ERR_01	✓	✓
FCC_ERR_01	✓	✓
FLASH_ERR_02	✓	✓

Errata Number	Rev C	Rev D
FLASH_ERR_03	✓	✓
FLASH_ERR_04	✓	
FLASH_ERR_05	✓	✓
FLASH_ERR_06	✓	✓
FLASH_ERR_08	✓	✓
GPIO_ERR_01	✓	✓
GPIO_ERR_02	✓	✓
GPIO_ERR_03	✓	✓
GPIO_ERR_04	✓	✓
GPIO_ERR_06	✓	✓
I2C_ERR_01	✓	✓
I2C_ERR_02	✓	✓
I2C_ERR_03	✓	✓
I2C_ERR_04	✓	✓
I2C_ERR_05	✓	✓
I2C_ERR_06	✓	✓
I2C_ERR_07	✓	✓
I2C_ERR_08	✓	✓
I2C_ERR_09	✓	✓
I2C_ERR_10	✓	✓
I2C_ERR_13	✓	✓
I2C_ERR_15	✓	✓
PMCU_ERR_01	✓	✓
PMCU_ERR_02	✓	✓
PMCU_ERR_03	✓	✓
PMCU_ERR_10	✓	✓
PMCU_ERR_13	✓	✓
PWREN_ERR_01	✓	✓
RST_ERR_01	✓	✓
SPI_ERR_01	✓	✓
SPI_ERR_03	✓	✓
SPI_ERR_04	✓	✓
SPI_ERR_05	✓	✓
SPI_ERR_06	✓	✓
SPI_ERR_07	✓	✓
SPI_ERR_08	✓	✓
SPI_ERR_09	✓	✓
SPI_ERR_10	✓	✓
SYSCTL_ERR_03	✓	✓
SYSCTL_ERR_05	✓	✓
SYSCTL_ERR_06	✓	✓
SYSCTL_ERR_11	✓	✓
SYSOSC_ERR_01	✓	✓
SYSOSC_ERR_02	✓	✓
SYSOSC_ERR_04	✓	✓
SYSOSC_ERR_05	✓	✓

Errata Number	Rev C	Rev D
SYSOSC_ERR_06	✓	✓
TIMER_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_09	✓	✓
UART_ERR_10	✓	✓
UART_ERR_11	✓	✓
VREF_ERR_01	✓	✓

3 Preprogrammed Software Advisories

Advisories that affect factory-programmed software.

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

4 Debug Only Advisories

Advisories that affect only debug operation.

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

5 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.

6 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.

6.1 Device Symbolization and Revision Identification

The package diagrams below indicate the package symbolization scheme, and [Table 6-1](#) defines the device revision to version ID mapping.

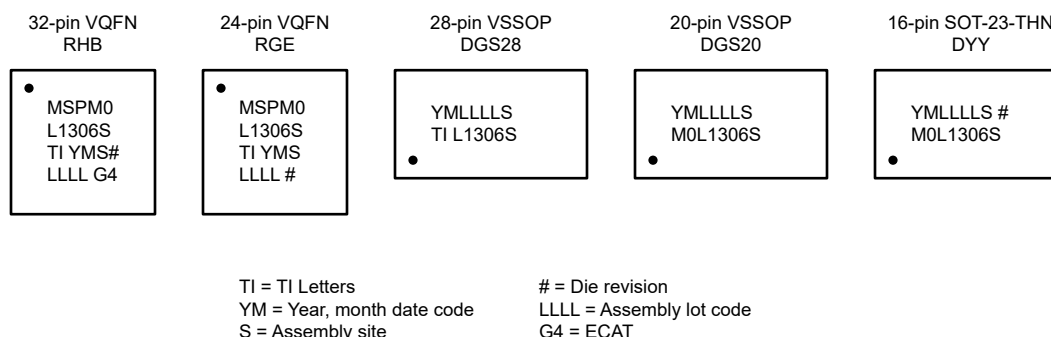


Figure 6-1. Package Symbolization

Table 6-1. Die Revisions

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

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.

7 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

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. The current numbers seen will be comparable to the SLEEP0 current characteristics in the datasheet.

Workaround

Workaround 1: Configure ADC to be in Single or Sequence mode (no Repeat), and use an interrupt after conversion to re-enable ADC for next trigger. Workaround 2: Reset ADC or change the configuration of the ADC when entering the LPM mode. Recommended not to use repeat mode when entering LPM.

ADC_ERR_03 *ADC Module*

Category

Functional

Function

ADC SNR and DNL performance is worse when using VSSA for ADC ground reference.

Description

When the ADC is using VSS as the ground reference, noise sources on the VSS ground will impact the ADC results, which results in worsened SNR and DNL performance.

Workaround

Workaround 1: Use VREF+ and VREF- pins for the ADC reference. With the VREF- pin, the ADC will have its analog ground reference lessening the influence of external noise sources. Workaround 2: Reduce noise sources on VSS; common sources are CPU or Digital switching noise

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.
ADC_ERR_05	ADC Module
Category	Functional
Function	HW Event generated before enabling IP, ADC Trigger will stay in queue
Description	When the ADC receives a HW event trigger the pending event will go into the ADC trigger queue, even if CTL0.ENC = 0x0. Once ADC is set to CTL1.TRIGSRC = 0x1 (HW Trigger) and CTL0.ENC = 1 then the queued HW trigger will start the ADC sampling and conversion process. The pending HW request can get into the queue even when the CTL1.TRIGSRC = 0x0 (SW trigger), but will only start the ADC sampling when CTL1.TRIGSRC = 0x1 && CTL0.ENC = 0x1.
Workaround	Only configure the ADC F_SUB when expecting to use the HW Trigger, otherwise a pending request can get queued. If switching between SW and HW Trigger modes, a reset on the ADC (RSTCTL) will clear any pending queues, but will require the ADC to be reconfigured.
ADC_ERR_06	ADC Module
Category	Functional
Function	ADC Output code jumps degrading DNL/INL specification
Description	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. At worst case scenario, -40C, the error rate is 1 in 12M converted samples in 12-bit mode. At 0C, error rate is 1 in 24M At 55C, error rate is 1 in 60M (VDD voltage and reference used has no impact on errata rate)

ADC_ERR_06

(continued)

ADC Module

Workaround

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.

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

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.

Workaround 3: Use ADC sample averaging to minimize the effect of the results of any single incorrect conversion.

ADC_ERR_11

ADC Module

Category

Functional

Function

When in repeat modes with MEMCTL.TRIG = 0x1 the ADC Busy remains high and ADC ENC does not get cleared

Description

When in repeat single or repeat sequence mode with the MEMCTL.TRIG = 0x1 (next conversion requires a trigger). The ADC remains in the BUSY state as the ADC is waiting for a new trigger. This causes the ENC bit to not get cleared when explicitly writing 0.

Workaround

Workaround 1: Reset the ADC peripheral, ADCx.RSTCTL = 0xB1000003, to clear the BUSY status and the ENC bit. Will need to reconfigure the

Workaround 2: Set the ADC into single mode ADCx.CTL1.CONSEQ = 0x0 or 0x1. Then set the last memory control register, MEMCTL[Y].TRIG to 0x0 (next conversion is automatic). The ADC will sample the next channel but won't repeat afterwards, this will allow the BUSY status and ENC to clear.

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.

COMP_ERR_01

(continued)

COMP Module**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.

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

COMP_ERR_05

(continued)

COMP Module

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

CPU_ERR_01

CPU Module

Category

Functional

Function

CPU cache content can get corrupted when switching between Main flash and other flash regions.

Description

Cache corruption can occur when switching between accessing Main flash memory, and other non-volatile memory regions such as NONMAIN or Factory Region.

Workaround

Use the following procedure to access areas outside main memory safely:

1. Disable the cache by setting `CPUSS.CTL.ICACHE = 0x0`.
2. Read from SHUTDOWN Memory `SYSCTL.SOCLOCK.SHUTDNSTORE0`
3. Perform needed access to NONMAIN or Factory Region memory.
4. Re-enable cache by setting `CPUSS.CTL.ICACHE = 0x1`

CPU_ERR_02

CPU Module

Category

Functional

Function

Limitation of disabling prefetch/cache for CPOSS

CPU_ERR_02

(continued)

CPU Module**Description**

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

Workaround

Disable the cache and the prefetcher (order is not mandatory), and then issue a memory access to the shutdown memory (SHUTDNSTORE) in SYSTCL (please check the SHUTDNSTORE registers in the devices TRM for more reference).

After the memory access completes, the prefetcher/cache will be disabled.

Example:

```
CPUSS->CTL = ( CPOSS_CTL_PREFETCH_DISABLE |
CPOSS_CTL_ICACHE_DISABLE); //disables instruction caching and pre-fetching
DL_SYSCTL_setShutdownStorageByte(DL_SYSCTL_SHUTDOWN_STORAGE_BYTE_X
, data) // Save a byte to SHUTDOWN memory
```

CPU_ERR_03**CPU Module****Category**

Functional

Function

Prefetcher can fetch wrong instructions when transitioning into Low power modes

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
```

CPU_ERR_04**CPU Module****Category**

Functional

Function

Cache doesn't return correct data from an address in FLASH that caused a hard fault

Description

When the device enters a hard fault from a FLASH address, after the device recovers from the hard fault and attempts to retrieve information from its cache (which stores the

CPU_ERR_04 (continued)

CPU Module

data from the FLASH address), the returned value becomes zero. Furthermore, if the same FLASH address gets accessed, a new hard fault won't be triggered.

Workaround

Disable and re-enable the cache by changing the CPUSS.CTL.ICACHE bit.
Example:
CPUSS->CTL &= ~(CPUSS_CTL_ICACHE_ENABLE); //disables instruction caching
CPUSS->CTL |= CPUSS_CTL_ICACHE_ENABLE; //enable instruction caching

CRC/ CRCP_ERR_01

CRC/CRCP Module

Category

Functional

Function

DMA cannot be triggered to access the CRC/CRCP module in suspended LPM

Description

In RUN0/1/2 and SLEEP0/1/2 modes, DMA can access the CRC/CRCP module normally. In STOP/STANDBY mode, any trigger to DMA will put the device in the suspended low power mode where, however, in this mode, DMA fails to access to CRC/CRCP module for some device.

Workaround

1. If DMA access to CRC/CRCP is needed, do not enter STOP/STANDBY mode. 2. In STOP/STANDBY mode, use another wake-up function (such as an interrupt) to bring the device out of STOP/STANDBY mode, and then trigger DMA access to CRC/CRCP or use CPU instead DMA to access CRC/CRCP.

FCC_ERR_01

FCC Module

Category

Functional

Function

FCC behaves incorrectly when BUSCLK is sourced from LFCLK

Description

When BUSCLK is sourced from LFCLK, FCC does not operate as expected. Either the FCC done signal does not assert, or an incorrect FCC value is reported.

Workaround

No workaround

FLASH_ERR_02

FLASH Module

Category

Functional

Function

Debug disable in NONMAIN can be re-enabled using the password

FLASH_ERR_02

(continued)

FLASH Module

Description

If debug is disabled in NONMAIN configuration (DEBUGACCESS = 0x5566), the device can still be accessed using the password that is programmed (default password if not explicitly programmed).

Workaround

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.

Workaround 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.

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 not in the main flash region.

Description

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.

If the return address of the SYSCTL_DEDERRADDR returns a 0x00Dxxxxx, do an OR operation with 0x41000000 to get the proper address for the Databank return address.

FLASH_ERR_04

(continued)

FLASH Module

For example, if SYSCCTL_DEDERRADDR = 0x00D0012A, the real address would be 0x00D0012A.

FLASH_ERR_05

FLASH Module

Category

Functional

Function

DEDERRADDR can have incorrect reset value

Description

The reset value of the SYSCCTL->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_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.

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

FLASH_ERR_08

(continued)

FLASH Module**Workaround**

No

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_02**GPIO Module****Category**

Functional

Function

High current draw following a negative current injection on PA2 pin

Description

A negative current injection into the PA2 pin can lead to an unexpected and sustained high current draw in the device.

This errata is not in effect if the PA2 is used in ROSC mode (100k ohm resistor is populated between ROSC and VSS).

GPIO_ERR_02

(continued)

GPIO Module

Workaround

Place a series resistor with a typical value of 100 ohm between the PA2 pin and any connected circuit, with the resistor placed near to the PA2 pin. This resistor is intended to limit fast transients seen at the PA2 pin and is sufficient to prevent this scenario from occurring.

OR

Avoid using the PA2 pin and use an alternate pin instead.

GPIO_ERR_03

GPIO Module

Category

Functional

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.

GPIO_ERR_04

GPIO Module

Category

Functional

Function

Configuring global fastwake prevents a GPIO pin from sending data to the DIN register.

Description

When configuring the fastwake-only bit of the CTL register and forcing data to a GPIO pin in run mode, the device will wake up, but the data on the GPIO pin will not appear in the DIN register. This is because the CTL register configuration prevents any data from flowing from the GPIO pin to the DIN register.

Workaround

Avoid using the GPIO fastwake-only function when expecting data on a GPIO pin entering the DIN register.

GPIO_ERR_06

GPIO Module

Category

Functional

Function

Writing to GPIO DOUT, DOUTSET and DOUTCLR registers might get missed when a DMA transfer is ongoing

GPIO_ERR_06

(continued)

GPIO Module**Description**

The GPIO DOUT, DOUTSET and DOUTCLR registers cannot be accessed by the DMA. Due to mistake in the implementation, the CPU access to the GPIO DOUT, DOUTSET and DOUTCLR will be also be blocked when a concurrent DMA transfer is in progress.

Workaround

In the application code, instead of writing to the DOUT, DOUTSET, and DOUTCLR registers, software should perform equivalent writes to the DOUTTGL register (see workaround GPIO_ERR_05 for restrictions on CPU writes to the DOUTTGL register).

In the pseudo code below, "pins" denotes the bit vector of pins in the GPIO module to be configured.

```
DL_GPIO_setPins(GPIO_Regs* gpio, uint32_t pins)
{
    gpio->DOUTTGL31_0 = ~(gpio->DOUT31_0) & pins;
}

DL_GPIO_clearPins(GPIO_Regs* gpio, uint32_t pins)
{
    gpio->DOUTTGL31_0 = gpio->DOUT31_0 & pins;
}

DL_GPIO_writePins(GPIO_Regs* gpio, uint32_t pins)
{
    gpio->DOUTTGL31_0 = ~(gpio->DOUT31_0) & pins;
    gpio->DOUTTGL31_0 = gpio->DOUT31_0 & (~pins);
}

DL_GPIO_writePinsVal(GPIO_Regs* gpio, uint32_t pinsMask, uint32_t pinsVal)
{
    uint32_t doutVal = gpio->DOUT31_0;
    doutVal &= ~pinsMask;
    doutVal |= (pinsVal & pinsMask);
    gpio->DOUTTGL31_0 = ~(gpio->DOUT31_0) & doutVal;
    gpio->DOUTTGL31_0 = gpio->DOUT31_0 & (~doutVal);
}
```

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

I2C_ERR_01 (continued)

I2C Module

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

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.

I2C_ERR_04

I2C Module

Category

Functional

Function

When SCL is low and SDA is high the Target i2c is not able to release the stretch.

I2C_ERR_04

(continued)

I2C Module

Description

- 1: SCL line grounded and released, device indefinitely pulls SCL low.
- 2: Post clock stretch, timeout, and release; if there is another clock low on the line, device indefinitely pulls SCL low.

Workaround

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.

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.

I2C_ERR_05

I2C Module

Category

Functional

Function

I2C SDA may get stuck to zero if we toggle ACTIVE bit during ongoing transaction

Description

If ACTIVE bit is toggled during an ongoing transfer, its state machine will be reset. However, the SDA and SCL output which is driven by the I2C controller will not get reset. There is a situation where SDA is 0 and I2C controller has gone into IDLE state, here the I2C controller won't be able to move forward from the IDLE state or update the SDA value. I2C 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 I2C controller will not be able to drive a STOP to clear it.

Workaround

Do not toggle the ACTIVE bit during an ongoing transaction.

I2C_ERR_06

I2C Module

Category

Functional

Function

SMBus High timeout feature fails at I2C clock less than 24KHz onwards

Description

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

I2C_ERR_06 (continued)

I2C Module

(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.

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 can cause I2C to not start.

Description

Back to back CTR register writes can 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 causes erroneous data to be read.

Description

When the RXDONE interrupt happens the FIFO can not be 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.

I2C_ERR_09

(continued)

I2C Module**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. Any I2C transfer can always be terminated with a STOP condition to provide proper BUSY status and Asynchronous clock request behavior (for low power mode reentry).

I2C_ERR_13**I2C Module****Category**

Functional

Function

Polling the I2C BUSY bit might not guarantee that the controller transfer has completed

Description

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.

Workaround

Add software delay before polling BUSY status. Software delay = $3 \times \text{CPU CLK} / \text{I2C functional clock}$ = $3 \times \text{CPU CLK} / (\text{CLKSEL} / \text{CLKDIV})$ For example, with a clock divider (CLKDIV) of 8, a clock source of 4 MHz(MFCLK), and CPU CLK of 32 MHz: Software delay = $3 \times 32 \text{ MHz} / (4 \text{ MHz} / 8) = 192 \text{ CPU cycles}$

I2C_ERR_15**I2C Module****Category**

Functional

Function

Glitch on I2C SDA causes I2C peripheral active in low power mode

Description

Glitch within 4us on I2C SDA causes I2C peripheral switch to active status in low power mode(STOP/STANDBY), and doesn't go back to low power mode.

I2C_ERR_15 (continued)

I2C Module

Workaround

1. Increase the capacitance on the I2C bus, but ensure its total is no larger than the I2C standard allows. 2. Increase the voltage of the I2C bus. 3. Move the I2C lines away from the source of the noise. 4. Periodically wakeup device to check I2C peripheral status, if it's not in a IDLE status, reset I2C and initialize it again.

PMCU_ERR_01

GPIO Module

Category

Functional

Function

Current leakage on PA2/ROSC pin

Description

When using pin PA2/ROSC in a functional configuration that is not ROSC, a large leakage current can be observed if the pin is driven high. This is due an unintended connection to ground through an impedance of approximately 17k ohm.

This errata does not affect ROSC functionality.

Workaround

IF PA2/ROSC is used for alternative functionality to ROSC, THEN one must account for the extra leakage current in energy budgets when the pin is driven high, either externally or internally.

IF the pin is pulled up externally with a resistor, THEN external voltage will be lower than expected due to the creation of a voltage divider circuit due to this erratum.

PMCU_ERR_02

GPIO Module

Category

Functional

Function

Transient voltage output during the device startup

Description

When using pin PA2/ROSC in a functional configuration that is not ROSC, unintended transient voltage pulses can be observed on the PA2/ROSC pin during startup.

This errata does not affect ROSC functionality.

Workaround

No workaround available.

PMCU_ERR_03

BOR Module

Category

Functional

Function

BOR1, BOR2, and BOR3 do not work in STANDBY mode

PMCU_ERR_03

(continued)

BOR Module**Description**

The user-selectable alternate BOR thresholds (BOR1, BOR2, BOR3) are not functional when operating in the device in STANDBY mode, thus the device does not reset properly when passing these thresholds.

Workaround

Do not use BOR1, BOR2, or BOR3 when operating in STANDBY mode. Configure the device to use the default BOR0 level before entering STANDBY mode. Upon exit from STANDBY mode, alternate BOR levels can be re-enabled.

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 $VDD \geq 1.8V$ and use VBOOST in ONALWAYS mode using `GENCLKCFG[23:22]=0x2`.

PMCU_ERR_13***PMCU Module*****Category**

Functional

Function

MCU may get stuck while waking up from STOP2 & STANDBY0

Description

If prefetch access is pending when the device transitions to STOP2 or STANDBY, when the device wakes up, the pending prefetch can prevent the device from resuming normal execution. The errata occurs if the WFI instruction is not word aligned, and the flash wait state is 2. In such a case, neither a DMA transfer nor a pending interrupt will be serviced.

Workaround

User should disable prefetch and issue a shutdown store memory read, which prevents a new prefetch from issuing and allows a pending prefetch to complete.

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).

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

Device Remains in Reset When NRST Release Edge Coincides with LFOSCGOOD Rising Edge after A Loss of LFXT or LFCLK_IN

Description

In a specific corner case, if the release edge (low-to-high transition) of the NRST signal coincides with the rising edge of LFOSCGOOD, the device fails to detect the deassertion of NRST and remains in a reset state indefinitely. Conditions that might trigger the issue: When the LFCLK source is switched from LFCLK_IN or LFXT to LFOSC, the LFOSC is re-enabled and LFOSCGOOD status asserts high within 250us (min) to 1.5ms (max) of the re-enable event. If an NRST low pulse is applied such that its release edge (low-to-high transition) aligns with the LFOSCGOOD rising edge within a 50 ns window, the NRST deassertion is not detected and the device remains in reset.

Workaround

Keep the NRST low pulse width higher than 2ms to avoid this issue

SPI_ERR_01

SPI Module

Category

Functional

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_03	SPI Module
Category	Functional
Function	When configured as peripheral, enable CSCLR will result in the received data will have a right shift in SPH=0 mode
Description	When enabled CSCLR in peripheral mode, if there has glitch on SCK line when CS is active or inactive, the received data will have 1-bit right shift in the next first frame. This issue is seen in Motorola SPI Frame Format with SPH=0, and will impact multi-peripheral mode which has the case that SCK toggle during CS inactive.
Workaround	1. Set CSCLR=0h. 2. Always drop the first frame when set CSCLR=1h in SPH=0 mode.
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.

SPI_ERR_08

SPI Module

Category

Functional

Function

SPI_DMA_DONE_RX Interrupt will impact RX interrupt when DMA enabled

Description

If SPI_DMA_DONE_RX interrupt flag is asserted, but the interrupt is not enabled, then RX interrupt will not be triggered at next time.

Workaround

Enable DMA_DONE_RX interrupt. Or, manually clear SPI_DMA_DONE_RX interrupt flag.

SPI_ERR_09

SPI TX FIFO interrupt can just be triggered once Module

Category

Functional

Function

SPI_ERR_09

(continued)

SPI TX FIFO interrupt can just be triggered once Module**Description**

SPI Tx FIFO Trigger only sets one time and after that if the Tx FIFO doesn't cross the threshold and come back from the Interrupt service routine, RIS doesn't get re-triggered even though the FIFO is below the threshold level.

Workaround

Please configure the "TX FIFO threshold level" to "TX FIFO is empty".

SPI_ERR_10***SPI Module*****Category**

Functional

Function

DMA is not sampling the latest SPI trigger count

Description

When SPI is configured to trigger a DMA transfer on a receive timeout (RTOUT) event, if the DMA channel is busy servicing another transfer at the time of the trigger request, any additional bytes received by SPI before the DMA acknowledges the request will not be included in the transfer. The DMA transfers only the number of bytes received when the trigger request was issued, leaving the subsequently received bytes in the RX FIFO unread.

Workaround

Configure DMA_TRIG_RX to use the RX trigger condition in combination with RTOUT. The RX trigger fires when the RX FIFO reaches the configured threshold level, ensuring the full expected number of bytes is present in the FIFO before DMA servicing begins. This guarantees the DMA transfer count is accurate regardless of whether the DMA channel is delayed in acknowledging the request.

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_05 *LFCLK Module***Category**

Functional

Function

LFCLK not working on exit from shutdown

SYSCTL_ERR_05

(continued)

LFCLK Module

Description

If LFCLK_IN pin is configured as a general input (or) LFCLK_IN function with pull-up, in this configuration, exiting shutdown mode will cause the LFCLK to be stuck.

Workaround

Choose either: 1.Enable pull-down instead of pull-up on this LFCLK_IN I/O 2.Avoid configuring it as an input

SYSCTL_ERR_06 CLK_OUT Module

Category

Functional

Function

Glitch can occur on External Clock Output (CLK_OUT) when user disables the CLK_OUT while an asynchronous clock was selected as CLK_OUT source

Description

If the clock source for CLK_OUT is asynchronous with the current bus clock (for example, the bus clock is SYSOSC, while the clock source for CLK_OUT is selected as LFCLK), then in this case, glitches may appear on the CLK_OUT pin when it is enabled for a period of time and then disabled

Workaround

To avoid the glitch output to external pin, follow below sequence: CLK_OUT enable case: IOMUX enable configuration should be after CLK_OUT enable configuration. CLK_OUT disable case: IOMUX disable configuration should be before CLK_OUT disable configuration.

SYSCTL_ERR_11 SYSCTL Module

Category

Functional

Function

Unexpected BOR reset cycle happens when device in RUN2/SLEEP2 mode with FCL enabled

Description

When FCL is enable and device is running in RUN2/SLEEP2 mode, there is unexpected BOR followed by NMI happens for BOR1/2/3

Workaround

When using BOR1/2/3 monitor, avoid enabling FCL and making device run in RUN2 or SLEEP2 mode.

SYSOSC_ERR_01 SYSOSC Module

Category

Functional

Function

MFCLK drift when using SYSOSC FCL together with STOP1 mode

SYSOSC_ERR_01

(continued)

SYSOSC Module**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 Degradation in FCL ON Mode

Description

When using the FCL ON mode of the internal oscillator, SYSOSC, accuracy can degrade up to +/-3%. The accuracy degradation is due to a synchronization between the 4MHz FCL sampling clock and noise in the system caused by other peripherals operating at harmonics of 4MHz.

SYSOSC_ERR_04

(continued)

SYSOSC Module

Workaround

There is no workaround to completely remove noise from the system. However, if using the FCL ON mode, the degradation can be minimized when peripherals are operating at frequencies that are not multiples of 4MHz.

SYSOSC_ERR_05 **SYSOSC Module**

Category

Functional

Function

SYSOSC May Operate Below Base Freq During Async Fast Wakeup from LPM When FCL Is Enabled

Description

When FCL=enabled, SYSOSC may operate up to 10% below its base frequency during low power modes and while there is an active asynchronous fast clock request. This occurs while the device is in low power modes because the FCL = Disabled trim is always applied instead of using FCL= enabled trim even though FCL = enabled. Once the device has woken up, exited LPM, and serviced the request, SYSOSC applies correct trim resumes normal FCL = Enabled operation at its intended target frequency.

Most use cases should have no meaningful impact and can continue to use both FCL and Async fast clock requests as needed. The main applications in which this erratum will have meaningful impact are those where UART is the source of the async fast clock request, as this temporary shift could have impact on the effective baud rate until the device fully wakes.

Workaround

1. Keep FCL = disabled
2. If FCL =Enabled needed, disable Async fast clock requests.

SYSOSC_ERR_06 **SYSOSC Module**

Category

Functional

Function

MFCLK work with 32MHz instead of 4MHz when async clock request is received in an LPM where SYSOSC was disabled in FCL mode

Description

MFCLK is incorrectly configured to 32MHz 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. Then MFCLK is incorrectly configured to 32MHz.

SYSOSC_ERR_06

(continued)

SYSOSC Module**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 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 ~100us 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**TIMER Module****Category**

Functional

TIMER_ERR_06

(continued)

TIMER Module

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

TIMG Module

Category

Functional

Function

Initial repeat counter has 1 less period than next repeats

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.

UART_ERR_02

UART Module

Category

Functional

UART_ERR_02

(continued)

UART Module**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 ULPCCLK 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 make sure 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

UART_ERR_06

(continued)

UART Module

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 it's 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

Description

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

UART_ERR_08
(continued)

UART Module

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.

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.

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 RXTSEL setting is too small. For example, if the baud rate was 1Mbps, and RXTSEL was set to

UART_ERR_11

(continued)

UART Module

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

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 data sheet for VREF startup time.

8 Revision History

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

Changes from August 30, 2025 to July 30, 2026 (from Revision D (August 2025) to Revision E (July 2026))

	Page
• ADC_ERR_02 Description was updated.....	5
• ADC_ERR_02 Workaround was updated.....	5
• ADC_ERR_05 Workaround was updated.....	6
• ADC_ERR_05 Description was updated.....	6
• ADC_ERR_06 Workaround was updated.....	6
• ADC_ERR_06 Description was updated.....	6
• ADC_ERR_11 Module was updated.....	7
• ADC_ERR_11 Function was updated.....	7
• ADC_ERR_11 Description was updated.....	7
• ADC_ERR_11 Workaround was updated.....	7
• CPU_ERR_01 Function was updated.....	9
• CPU_ERR_01 Description was updated.....	9
• CPU_ERR_01 Workaround 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 Function was updated.....	10
• CPU_ERR_03 Description was updated.....	10

• CPU_ERR_03 Workaround was updated.....	10
• CPU_ERR_04 Category was updated.....	10
• CPU_ERR_04 Module was updated.....	10
• CPU_ERR_04 Function was updated.....	10
• CPU_ERR_04 Description was updated.....	10
• CPU_ERR_04 Workaround was updated.....	10
• CRC/CRCP_ERR_01 Category was updated.....	11
• CRC/CRCP_ERR_01 Module was updated.....	11
• CRC/CRCP_ERR_01 Function was updated.....	11
• CRC/CRCP_ERR_01 Description was updated.....	11
• CRC/CRCP_ERR_01 Workaround was updated.....	11
• FCC_ERR_01 Category was updated.....	11
• FCC_ERR_01 Workaround was updated.....	11
• FCC_ERR_01 Function was updated.....	11
• FCC_ERR_01 Description was updated.....	11
• FCC_ERR_01 Module was updated.....	11
• FLASH_ERR_02 Function was updated.....	11
• FLASH_ERR_02 Description was updated.....	11
• FLASH_ERR_02 Workaround was updated.....	11
• Added FLASH_ERR_03.....	12
• FLASH_ERR_08 Module was updated.....	13
• FLASH_ERR_08 Function was updated.....	13
• FLASH_ERR_08 Description was updated.....	13
• FLASH_ERR_08 Workaround was updated.....	13
• GPIO_ERR_04 Module was updated.....	15
• GPIO_ERR_04 Category was updated.....	15
• GPIO_ERR_04 Function was updated.....	15
• GPIO_ERR_04 Workaround was updated.....	15
• GPIO_ERR_04 Description was updated.....	15
• GPIO_ERR_06 Category was updated.....	15
• GPIO_ERR_06 Module was updated.....	15
• GPIO_ERR_06 Function was updated.....	15
• GPIO_ERR_06 Description was updated.....	15
• GPIO_ERR_06 Workaround was updated.....	15
• I2C_ERR_09 Description was updated.....	19
• I2C_ERR_09 Workaround was updated.....	19
• I2C_ERR_13 Module was updated.....	20
• I2C_ERR_13 Function was updated.....	20
• I2C_ERR_13 Workaround was updated.....	20
• I2C_ERR_13 Description was updated.....	20
• I2C_ERR_15 Category was updated.....	20
• I2C_ERR_15 Module was updated.....	20
• I2C_ERR_15 Function was updated.....	20
• I2C_ERR_15 Workaround was updated.....	20
• I2C_ERR_15 Description was updated.....	20
• Added PMCU_ERR_10.....	22
• PMCU_ERR_13 Function was updated.....	22
• PMCU_ERR_13 Description was updated.....	22
• PMCU_ERR_13 Workaround was updated.....	22
• RST_ERR_01 Function was updated.....	23
• RST_ERR_01 Description was updated.....	23
• RST_ERR_01 Workaround was updated.....	23
• SPI_ERR_03 Workaround was updated.....	24
• SPI_ERR_07 Description was updated.....	25
• SPI_ERR_07 Workaround was updated.....	25

• SPI_ERR_08 Module was updated.....	25
• SPI_ERR_08 Function was updated.....	25
• SPI_ERR_08 Description was updated.....	25
• SPI_ERR_08 Workaround was updated.....	25
• SPI_ERR_09 Module was updated.....	25
• SPI_ERR_09 Description was updated.....	25
• SPI_ERR_09 Workaround was updated.....	25
• SPI_ERR_10 Category was updated.....	26
• SPI_ERR_10 Module was updated.....	26
• SPI_ERR_10 Function was updated.....	26
• SPI_ERR_10 Description was updated.....	26
• SPI_ERR_10 Workaround was updated.....	26
• Added SYSCTL_ERR_03.....	26
• SYSCTL_ERR_05 Category was updated.....	26
• SYSCTL_ERR_05 Module was updated.....	26
• SYSCTL_ERR_05 Function was updated.....	26
• SYSCTL_ERR_05 Description was updated.....	26
• SYSCTL_ERR_05 Workaround was updated.....	26
• SYSCTL_ERR_06 Category was updated.....	27
• SYSCTL_ERR_06 Module was updated.....	27
• SYSCTL_ERR_06 Function was updated.....	27
• SYSCTL_ERR_06 Description was updated.....	27
• SYSCTL_ERR_06 Workaround was updated.....	27
• SYSCTL_ERR_11 Category was updated.....	27
• SYSCTL_ERR_11 Module was updated.....	27
• SYSCTL_ERR_11 Function was updated.....	27
• SYSCTL_ERR_11 Workaround was updated.....	27
• SYSCTL_ERR_11 Description was updated.....	27
• SYSOSC_ERR_04 Category was updated.....	28
• SYSOSC_ERR_04 Module was updated.....	28
• SYSOSC_ERR_04 Workaround was updated.....	28
• SYSOSC_ERR_04 Function was updated.....	28
• SYSOSC_ERR_04 Description was updated.....	28
• SYSOSC_ERR_05 Category was updated.....	29
• SYSOSC_ERR_05 Module was updated.....	29
• SYSOSC_ERR_05 Function was updated.....	29
• SYSOSC_ERR_05 Workaround was updated.....	29
• SYSOSC_ERR_05 Description was updated.....	29
• SYSOSC_ERR_06 Category was updated.....	29
• SYSOSC_ERR_06 Module was updated.....	29
• SYSOSC_ERR_06 Function was updated.....	29
• SYSOSC_ERR_06 Description was updated.....	29
• SYSOSC_ERR_06 Workaround was updated.....	29
• TIMER_ERR_04 Description was updated.....	30
• TIMER_ERR_04 Workaround was updated.....	30
• TIMER_ERR_06 Module was updated.....	30
• Added TIMER_ERR_07.....	31
• UART_ERR_10 Module was updated.....	34
• UART_ERR_10 Function was updated.....	34
• UART_ERR_10 Description was updated.....	34
• UART_ERR_10 Workaround was updated.....	34
• UART_ERR_11 Module was updated.....	34
• UART_ERR_11 Function was updated.....	34
• UART_ERR_11 Description was updated.....	34
• UART_ERR_11 Workaround was updated.....	34

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](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. 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 © 2026, Texas Instruments Incorporated

Last updated 10/2025