Errata

MSPM0G3x0x, MSPM0G1x0x, MSPM0G3x0x-Q1 Microcontrollers

ABSTRACT

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

Table of Contents

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.

Table 1-1. Functional Advisories

Errata Number	Rev B	Rev C	Rev D
ADC_ERR_01	✓	✓	✓
ADC_ERR_02	✓	✓	✓
ADC_ERR_05	✓	✓	✓
ADC_ERR_06	✓	✓	✓
BSL_ERR_01	✓	✓	✓
CLK_ERR_01	✓	✓	✓
COMP_ERR_02	✓	✓	✓
COMP_ERR_03	✓	✓	✓
COMP_ERR_06	✓	✓	✓
CPU_ERR_01	✓	✓	✓
CPU_ERR_02	✓	✓	✓
CPU_ERR_03	✓	✓	✓
CPU_ERR_04	✓	✓	✓
CRC/CRCP_ERR_01	✓	✓	✓
DAC_ERR_01	✓	✓	✓
DMA_ERR_01	✓	✓	✓
FCC_ERR_01	✓	✓	✓
FLASH_ERR_02	✓	✓	✓
FLASH_ERR_04	✓	✓	✓
FLASH_ERR_05	✓	✓	✓

Table 1-1. Functional Advisories (continued)

Errata Number	Rev B	Rev C	Rev D
FLASH_ERR_06	✓	✓	✓
FLASH_ERR_08	✓	✓	✓
GPIO_ERR_01	✓	✓	✓
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	✓	✓	✓
IOMUX_ERR_02	✓	✓	✓
MATHACL_ERR_01	✓	✓	✓
MATHACL_ERR_02	✓	✓	✓
PMCU_ERR_08	✓	✓	✓
PMCU_ERR_10	✓	✓	✓
PWREN_ERR_01	✓	✓	✓
RST_ERR_01	✓	✓	✓
RST_ERR_02	✓	✓	✓
RTC_ERR_01	✓	✓	✓
SPI_ERR_01	✓	✓	✓
SPI_ERR_02	✓	✓	✓
SPI_ERR_03	✓	✓	✓
SPI_ERR_04	✓	✓	✓
SPI_ERR_05	✓	✓	✓
SPI_ERR_06	✓	✓	✓
SPI_ERR_07	✓	✓	✓
SPI_ERR_10	✓	✓	✓
SRAM_ERR_02	✓	✓	✓
SYSCTL_ERR_02	✓	✓	✓
SYSCTL_ERR_03	✓	✓	✓
SYSCTL_ERR_04	✓	✓	✓
SYSCTL_ERR_05	✓	✓	✓
SYSCTL_ERR_06	✓	✓	✓
SYSOSC_ERR_01	✓	✓	✓
SYSOSC_ERR_02	✓	✓	✓

Table 1-1. Functional Advisories (continued)

Errata Number	Rev B	Rev C	Rev D
SYSOSC_ERR_04	✓	✓	✓
SYSOSC_ERR_05	✓	✓	✓
SYSOSC_ERR_06	✓	✓	✓
SYSPLL_ERR_01	✓	✓	
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	✓	✓	✓
VREF_ERR_02	✓	✓	✓
VREF_ERR_04	✓	✓	✓
WWDT_ERR_01	✓	✓	✓
WWDT_ERR_02	✓	✓	✓

2 Preprogrammed Software Advisories

Advisories that affect factory-programmed software.

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

3 Debug Only Advisories

Advisories that affect only debug operation.

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

Errata Number	Rev B	Rev C
GPIO_ERR_03	✓	✓

4 Fixed by Compiler Advisories

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

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

5 Device Nomenclature

To designate the stages in the product development cycle, TI assigns prefixes to the part numbers of all MSP MCU devices. Each MSP MCU commercial family member has one of two prefixes: MSP or XMS. These

prefixes represent evolutionary stages of product development from engineering prototypes (XMS) through fully qualified production devices (MSP).

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

MSP – Fully qualified production device

Support tool naming prefixes:

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

null: Fully-qualified development-support product.

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

"Developmental product is intended for internal evaluation purposes."

MSP devices have been characterized fully, and the quality and reliability of the device have been demonstrated fully. TI's standard warranty applies.

Predictions show that prototype devices (XMS) have a greater failure rate than the standard production devices. TI recommends that these devices not be used in any production system because their expected end-use failure rate still is undefined. Only qualified production devices are to be used.

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

5.1 Device Symbolization and Revision Identification

The package diagrams below indicate the package symbolization scheme, which is the pre-prod version. After device release, RTM version will be added here. Table 5-1 defines the device revision to version ID mapping.

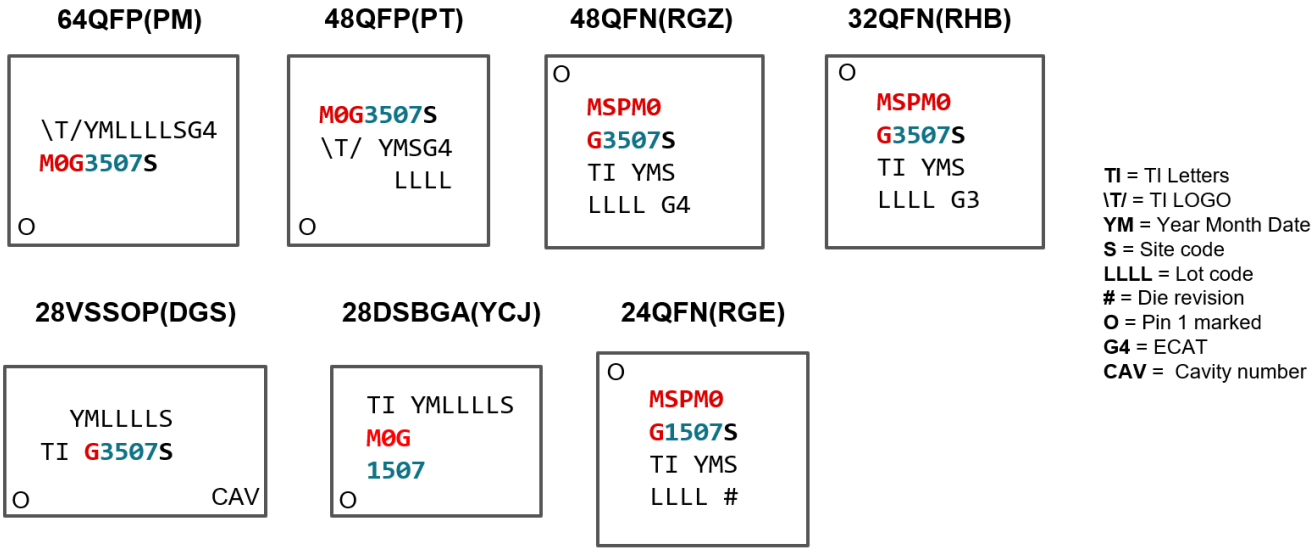

Figure 5-1. Package Symbolization

Table 5-1. Die Revisions

Revision Letter	Version (in the device factory constants memory)
B	1
C	2
D	3

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 (DEVICEID.VERSION field), which can be used to look up the revision using application software or a connected debug probe.

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 ADCs 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_05 *ADC Module*

Category

Functional

Function

HW Event generated before enabling IP will stay in queue

Description

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.

Workaround

After configuring ADC in HW trigger mode, enable ADC first before giving external trigger.

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)

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.

BSL_ERR_01 *BSL Module*

Category

Functional

Function

Invoking the BSL through application software will fail under certain conditions

Description

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.

This errata does not apply to BSL invocation through hardware invoke methods

Workaround

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.

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.

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_06 *COMP Module*

Category

Functional

Function

COMP output will not become ready or generate interrupts if SYSOSC is disabled before output is ready

Description

Enabling the COMP or changing any of its input configurations (IPSEL/IMSEL) will trigger the OUTRDYIFG bit to be set once the COMPs output is ready. OUTRDYIFG gates all COMP interrupts from firing until the output is ready.

The COMP OUTRDYIFG bit requires SYSOSC to be at least 4MHz to become set.
If SYSOSC is disabled, OUTRDYIFG will never set and will forever block all COMP interrupts from firing.

Workaround

For COMP to work as expected, SYSOSC must be enabled until the OUTRDYIFG bit is set. Anytime you enable the COMP or configure IPSEL/IMSEL you must wait for OUTRDYIFG bit to be set before entering a LPM that disables SYSOSC or changing the clock to LFCLK.
1. Enable Comparator
2. Configure the IPSEL/IMSEL, as required
3. Wait for OUTRDYIFG interrupt
4. Enter Low Power Mode (LPM) or change clock to LFCLK, if required

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 CPUSS

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 = (CPUSS_CTL_PREFETCH_DISABLE |
CPUSS_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.

DAC_ERR_01

DAC Module

Category

Functional

Function

DMADONE interrupt not generated when DMA and DAC operate at different frequencies

Description

When the DMA and DAC are at different frequencies then the DAC does not properly capture the DMADONE interrupt status, which will cause DAC.RIS, DAC.MIS, DAC.IIDX registers to not be updated.

The DMA uses MCLK as the clock source and the DAC uses the ULPCLK as the clock source. When ULPCLK is not equal to MCLK, you will run into this erratum.

Workaround

Workaround 1:
Do not use the DMA and DAC if MCLK is not equal to ULPCLK.

Workaround 2:

DAC_ERR_01 (continued) — ***DAC Module***

Poll the DMASZ.SIZE register to check if all the DMA transfers have completed. If DMASZ.SIZE is 0, then the DMA is done.

DMA_ERR_01 — ***DMA Module***

Category

Functional

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.

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

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).

FLASH_ERR_02 (continued)

FLASH Module

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

Workaround

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

FLASH_ERR_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. 0x40BFFFFF - 0x41C00000 3. 0x41C007FF - 0x41C40000

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.

GPIO_ERR_01 (continued)

GPIO Module

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 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

CPU write access to GPIO DOUT, DOUTSET, and DOUTCLR might get missed when a concurrent DMA access is happening in the system

GPIO_ERR_06
(continued)

GPIO Module

Description

The GPIO DMAMASK controls which register bits in the DOUT MMR the DMA can modify. Due to a mistake in implementation, the DMAMASK is also applied to CPU accesses to the GPIO DOUT, DOUTSET, and DOUTCLR when a concurrent DMA access is occurring in the system. As a result, any register bit assigned to the DMA by setting DMAMASK = 1 might not be set by the CPU.

Workaround

1. Keep the GPIOs that need to be accessed by the CPU and DMA separate. Use the DMAMASK to define which GPIO is assigned to DMA and which is defined for the CPU. Dont access the DMA-assigned bits by the CPU.
2. If concurrent CPU and DMA access to the same GPIO is required, the DMAMASK must be cleared before the CPU accesses the GPIO. Pseudocode Ex:
I. DL_GPIO_disableDMAAccess(GPIO_REG, SELECTED_PINS); // Disable DMA access on a group of pins from a specific GPIO Register
II. /* Implement GPIO DOUT, DOUTSET and/or DOUTCLR registers modification */
III. DL_GPIO_enableDMAAccess(GPIO_REG, SELECTED_PINS); //Enable DMA access on a group of pins from a specific GPIO Register

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.

I2C_ERR_02
(continued)

I2C Module

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 low occurs and target wakeup is enabled, device may clock stretch indefinitely.
Description	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.
Workaround	Disable the target wakeup enable bit (SWUEN).

I2C_ERR_05	*I2C Module*
Category	Functional
Function	I2C SDA can get stuck to zero if we toggle ACTIVE bit during ongoing transaction
Description	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.
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 (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 one clock cycle between the CTR writes and CTR.START write.

I2C_ERR_08

I2C Module

Category

Functional

Function

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

Description

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

Workaround

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

I2C_ERR_09

I2C Module

Category

Functional

Function

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

Description

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

Workaround

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

I2C_ERR_10

I2C Module

Category

Functional

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).

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 x CPU CLK / I2C functional clock = 3 x CPU CLK / (CLKSEL / CLKDIV) For example, with a clock divider (CLKDIV) of 8, a clock source of 4 MHz(MFCLK), and CPU CLK of 32 MHz: Software delay = 3 x 32 MHz / (4 MHz/ 8)= 192 CPU cycles

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.

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.

IOMUX_ERR_02 *IOMUX Module*

Category

Functional

Function

IOMUX register reads may return incorrect values when CPU clock exceeds 40MHz.

Description

The IOMUX read path for the MSPM0G3x0x and MSPM0G1x0x series is rated for a maximum frequency of 40MHz. When the CPU clock exceeds this threshold, the timing margin for the read path is insufficient, leading to potential data corruption during register read access.

Workaround

Workaround 1: Avoid software reads of IOMUX registers. Limit IOMUX operations to write-only configuration changes.

Workaround 2: Ensure the CPU clock (MCLK) frequency is less or equal to 40MHz when performing IOMUX register reads. To temporarily reduce the MCLK frequency, follow the sequence:
1. Disable peripherals which are sourced from MCLK.
2. Switch MCLK from SYSPLL to SYSOSC.
3. Perform IOMUX read/write operations.
4. Switch MCLK back from SYSOSC to SYSPLL.
5. Re-enable the peripherals.

Workaround 3: Perform two consecutive, uninterrupted reads of the target IOMUX register. Discard the first result and utilize the second result as the valid value.
Example (Workaround 3):
__disable_irq(); /* IRQs are disabled to prevent an interrupt from executing between the two reads */
uint32_t iomuxBaseAddr = (uint32_t) &IOMUX->SECCFG.PINCM[usedPinIndex]; /* get IOMUX register address */
(*((volatile uint32_t *)(iomuxBaseAddr))); /* dummy read discard */
correctIomuxValue = (*((volatile uint32_t *)(iomuxBaseAddr))); /* valid result */
__enable_irq();
Note: If an NMI (Non-Maskable Interrupt) occurs between the two reads and the NMI ISR performs a PD0 peripheral read, the second IOMUX read may still return incorrect data. System-level handling is required for NMI-critical applications in this case.

MATHACL_ERR_01 *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_02	*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 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_10	*PMCU Module*
Category	Functional

PMCU_ERR_10
(continued)

PMCU Module

Function

VBOOST might have larger delay under certain operating conditions

Description

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

Workaround

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

PWREN_ERR_01

GPIO Module

Category

Functional

Function

Peripheral registers are still accessible after disabling PWREN register

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

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

RST_ERR_02	*RST Module*
Category	Functional
Function	Device May Fail to Power Up When NRST deassertion coincides with initial LFOSCGOOD assertion
Description	At power-up event, the internal low-frequency oscillator (LFOSC) is enabled following the first deassertion of NRST (low-to-high transition). The LFOSCGOOD signal transitions high within a window of 250us (min) to 1.5ms (max) after this NRST deassertion. If a subsequent NRST low pulse occurs and its deassertion coincides with the LFOSCGOOD rising edge within a 200ns window, the device may fail to recognize the NRST deassertion event. As a result, the device remains in reset and fail to exit the boot phase of the startup sequence. This scenario may occur if an external circuit is controlling the NRST logic.
Workaround	Ensure that the external reset controller holds the NRST line high (deasserted) for a minimum duration of 1.5 ms following the initial NRST deassertion.

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
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, enable CSCLR will result in the SPI data having 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, there will be right shift in the next first received data when TXFIFO = 2,3,4 and in all following received data when TXFIFO = 0,1.

Workaround

When using SPH = 0 mode, and enable CSCLR, please select one of the following workarounds:
1. Add 2 SPI communication clock cycle delay before reading data from RXFIFO. And, both controller and peripheral discard the first received data.
2. Add 2 SPI communication clock cycle delay before reading data from RXFIFO. And, add CRC function for SPI frame communication to monitor the whole frame.

Noted: Only adding delay can make number of right-shifted data from infinite to finite, but the first TX and RX data may be incorrect.

SPI_ERR_04 *SPI Module*

Category

Functional

Function

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

SPI_ERR_04
(continued)

SPI Module

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

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.

SPI_ERR_07
(continued)

SPI Module

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_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.

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 SYSCTRL->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.

SYSCTL_ERR_02 *SYSCTL Module*

Category

Functional

Function

SYSSTATUS.FLASHSEC is non-zero after a BOOTRST

Description

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

Workaround

No

SYSCTL_ERR_03 *SYSCTL Module*

Category

Functional

Function

DEDERRADDR persists after a SYSRESET or a write to the SYSSTATUSCLR

Details

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

Workaround

No workaround

SYSCTL_ERR_04 *SYSCTL Module*

Category

Functional

Function

SYSSTATUS.FLASHSEC is not cleared after a SYSRESET

Description

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

Workaround

Upon returning from SYSRESET, manually clear the FLASHSEC status with SYSSTATUSCLR = 0xCE000001.

SYSCTL_ERR_05 *LFCLK Module*

Category

Functional

Function

LFCLK not working on exit from shutdown

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.

SYSCTL_ERR_05 (continued) ***LFCLK Module***

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.

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

SYSOSC_ERR_02 (continued) ***SYSOSC Module***

Description

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

Workaround

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

SYSOSC_ERR_04 ***SYSOSC Module***

Category

Functional

Function

SYSOSC accuracy degrades in FCL ON mode

Description

When using 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.

Workaround

If using the SYSPLL FCL ON mode, use a non-4MHz multiple for the SYSPLL frequency, for example: 78MHz or 79MHz
Do not put the SYSPLL at 16, 32, 48, 40, 64, 80MHz etc.
For 78MHz:
Set SYSPLLCFG1.PDIV = 0x3 and SYSPLLCFG1.QDIV to 38

SYSPLL operating frequencies that are not multiples of 4MHz will still align with the FCL sampling clock at least once per microsecond, and more often when the two clocks share a common factor.

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

SYSOSC_ERR_05 *SYSOSC Module*
(continued)

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.

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.

SYSPLL_ERR_01 *SYSPLL Module*

Category

Functional

Function

SYSPLL Frequency may not lock to correct frequency when enabled.

Description

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

Workaround

Frequency Verification Process
Monitor the SYSPLL frequency output using the Frequency Clock Counter (FCC)

SYSPLL_ERR_01
(continued) ***SYSPLL Module***

whenever the SYSPLLEN bit is set to 1. Once the correct frequency is established, it will remain stable until the SYSPLL is disabled and re-enabled (SYSPLLEN bit toggled from 0 to 1). If an incorrect frequency is detected, disable and re-enable the SYSPLL to perform another verification.

Workaround 1: FCC Count Check
Use LFCLK as the FCC trigger source to count the SYSPLL output clock frequency. Execute the FCC and verify the measured value against the configured SYSPLL frequency using LFCLK as reference.
Example calculation:
- SYSPLLCLK0 = 80MHz ; LFCLK = 32.768kHz
- Measured FCC Count = 80,000,000/32,768 = 2,441
FCC Count Tolerance:
The real FCC count will vary depending on the combined clock accuracies (SYSPLLCLK0 and LFCLK). Recommend to add +/- 5~10% to allowed FCC check range.
- FCC count upper limit = 2,441*1.05 = 2,563
- FCC count lower limit = 2,441*0.95 = 2,318
Timing considerations:
- Clock synchronization time: 5-6 LFCLK cycles
- FCC trigger time: 1-32 LFCLK cycles (user-configurable)
Register Configuration:
- FCC Settings: SYSCTL.GENCLKCFG.FCCTRIGSRC = 1;
- SYSCTL.GENCLKCFG.FCCLVLTRIG = 0;
- SYSCTL.GENCLKCFG.FCCTRIGCNT = 0;
- SYSCTL.GENCLKCFG.FCCSELCLK = 4;
- Start FCC: SYSCTL.FCCCMD = 0x0E000001U
- Check FCC Done Status: SYSCTL.CLKSTATUS.FCCDONE
- Read FCC Count: SYSCTL.FCC
Timeout Protection:
Implement a software-based timeout during FCC Done status monitoring to prevent longer wait time under the condition the unlocked SYSPLL frequency is less than FCC trigger clock frequency (LFCLK).

```
fccTimeOutCounter = 0;
while (DL_SYSCTL_isFCCDone() == 0) {
delay_cycles(977); /* 1x LFCLK cycle = 32MHz/32.768kHz */
fccTimeOutCounter++;
if(fccTimeOutCounter > 65) break;
/* Timeout set to approximately 2ms (user-customizable) */
}
```

FCC Check Restart:
If the FCC measurement falls outside the expected range, disable and re-enable the SYSPLL (set SYSPLLEN to 0, then 1) and repeat the FCC verification.

Workaround 2: FCC Ratio Check
Use LFCLK as the FCC trigger source to count both SYSPLL output and input clock frequency. Execute the FCC and verify the measured ratio of the FCC check value between output and input clock to the expected ratio.
Example calculation:
- SYSPLL = 80MHz ; HFCLK = 40MHz ; LFCLK = 32.768kHz
- Expect clock ratio = 80MHz/40MHz = 2.0000
- Measured FCC count (SYSPLL) = 80,000,000/32,768 = 2,441

SYSPLL_ERR_01 (continued)

SYSPLL Module

- Measured FCC count (HFCLK) = 40,000,000/32,768 =1,220
- Measured clock ratio = 2,441/1,220 = 2.0008

FCC Ratio Tolerance:

The FCC ratio method eliminates combined clock accuracy errors and depends only on FCC uncertainty (2 counted clock cycles) and calculation rounding error. This allows for much tighter tolerance ranges compared to FCC count check method, for example +/- 0.3%.

Timing considerations:

- Clock synchronization time: 5-6 LFCLK cycles
- FCC trigger time: 1-32 LFCLK cycles (user-configurable)
- Total time per complete FCC ratio check: 2*(sync time + trigger time)

FCC Ratio Check Flow:

1. Configure FCC for SYSPLL output clock (SYSPLL0 or SYSPLL2X)
2. Start FCC and wait for FCC done (Add Timeout Protection, refer to FCC Count Check)
3. Read FCC check count back
4. Configure FCC for SYSPLL input clock (SYSOSC or HFCLK)
5. Start FCC and wait for FCC done (Add Timeout Protection, refer to FCC Count Check)
6. Read FCC check count back
7. Calculate the FCC check ratio and compared to the expected ratio range
8. If the FCC ratio falls outside the expected range, disable and re-enable the SYSPLL (set SYSPLLEN to 0, then 1) and repeat the FCC ratio verification.

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

TIMER_ERR_04 (continued) — ***TIMER Module***

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

Function

Writing 0 to CLKEN bit does not disable counter

Description

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

Workaround

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

TIMER_ERR_07 — ***Initial repeat counter has 1 less period than next repeats Module***

Category

Functional

Function

TIMER

Description

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

Workaround

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

UART_ERR_01 — ***UART Module***

Category

Functional

UART_ERR_01 (continued) — ***UART Module***

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

UART_ERR_05
(continued)

UART Module

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

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

UART_ERR_07 (continued) — *UART Module*

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.

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.

UART_ERR_10 (continued)

UART Module

Workaround

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

UART_ERR_11

UART Module

Category

Functional

Function

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

Description

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

Workaround

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

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

VREF_ERR_02 (continued)

VREF Module

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_04

VREF Module

Category

Functional

Function

VREF can't be used by ADC when Comparator uses VREF in Sample and Hold mode

Description

Configuration : Both the ADC and COMP are configured to use internal VREF as their reference, with the COMP operating in sampled mode.
Issue : The VREF module enters sample-and-hold mode due to the COMP's sample-and-hold request, preventing ADC access to VREF.

Workaround

Before enabling the ADC sampling, make sure the COMP is not utilizing VREF in sample-and-hold mode.
Example:
VREF.CTL0.SHMODE = 0;

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

WWDT_ERR_02 (continued)	***WWDT Module***
Function	Window Watchdog Timer 1 (WWDT1) does not create a reset cause
Description	After a reset caused by WWD1, the SYSCTL.RSTCAUSE register does not accurately portray that WWDT1 caused the reset.
Workaround	None.

Trademarks

All trademarks are the property of their respective owners.

7 Revision History

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

Changes from June 30, 2026 to August 31, 2026 (from Revision G (June 2026) to Revision H (August 2026)) **Page**

IMPORTANT NOTICE AND DISCLAIMER

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

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

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

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

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

Last updated 10/2025