Errata

MSPM0G3x0x, MSPM0G1x0x, MSPM0G3x0x-Q1 Microcontrollers

TEXAS INSTRUMENTS

ABSTRACT

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

Table of Contents

1 Functional Advisories 1
2 Preprogrammed Software Advisories 3
3 Debug Only Advisories 3
4 Fixed by Compiler Advisories 3
5 Device Nomenclature 3
6 Advisory Descriptions 4
7 Revision History 31

1 Functional Advisories

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

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

Errata Number	Rev B	Rev C
ADC_ERR_01	✓	✓
ADC_ERR_02	✓	✓
ADC_ERR_05	✓	✓
ADC_ERR_06	✓	✓
BSL_ERR_01	✓	✓
CLK_ERR_01	✓	✓
COMP_ERR_02	✓	✓
COMP_ERR_03	✓	✓
CPU_ERR_01	✓	✓
CPU_ERR_02	✓	✓
CPU_ERR_03	✓	✓
DMA_ERR_01	✓	✓
FLASH_ERR_02	✓	✓
FLASH_ERR_04	✓	✓
FLASH_ERR_05	✓	✓
FLASH_ERR_06	✓	✓
FLASH_ERR_08	✓	✓
GPIO_ERR_01	✓	✓
GPIO_ERR_04	✓	✓
I2C_ERR_01	✓	✓
I2C_ERR_02	✓	✓

Errata Number	Rev B	Rev C
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	✓	✓
MATHACL_ERR_01	✓	✓
MATHACL_ERR_02	✓	✓
PMCU_ERR_08	✓	✓
PMCU_ERR_10	✓	✓
PWREN_ERR_01	✓	✓
RST_ERR_01	✓	✓
RTC_ERR_01	✓	✓
SPI_ERR_01	✓	✓
SPI_ERR_02	✓	✓
SPI_ERR_03	✓	✓
SPI_ERR_04	✓	✓
SPI_ERR_05	✓	✓
SPI_ERR_06	✓	✓
SPI_ERR_07	✓	✓
SRAM_ERR_02	✓	✓
SYSCTL_ERR_02	✓	✓
SYSCTL_ERR_03	✓	✓
SYSCTL_ERR_04	✓	✓
SYSOSC_ERR_01	✓	✓
SYSOSC_ERR_02	✓	✓
SYSOSC_ERR_04	✓	✓
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	✓	✓

Errata Number	Rev B	Rev C
UART_ERR_10	✓	✓
UART_ERR_11	✓	✓
VREF_ERR_01	✓	✓
VREF_ERR_02	✓	✓
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.

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.

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.

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.

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 for CPUSS

Description

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

Workaround

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

After the memory access completes the prefetcher will be disabled.
Example:
CPUSS.CTL.PREFETCH = 0x0; //disables prefetcher
SYSCTL.SOCLOCK.SHUTDNSTORE0; // memory access to shutdown memory

CPU_ERR_03 *CPU Module*

Category

Functional

Function

Prefetcher can fetch wrong instructions when transitioning into Low power modes

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

CPU_ERR_03 (continued) — *CPU Module*

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

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.

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).

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

FLASH_ERR_02 (continued) — *FLASH Module*

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 in the NONMAIN or Factory region

Description

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

Workaround

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

For MAIN Flash DED, the SYSCTL_DEDERRADDR can be used as is.

FLASH_ERR_05 — *FLASH Module*

Category

Functional

Function

DEDERRADDR can have incorrect reset value

Description

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

Workaround

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

FLASH_ERR_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 and DEBUGSS 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.

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

I2C_ERR_01 (continued)

I2C Module

Description

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

Workaround

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

I2C_ERR_02

I2C Module

Category

Functional

Function

I2C quick command read mode only works with specific conditions

Description

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

Workaround

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

I2C_ERR_03

I2C Module

Category

Functional

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

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

MATHACL_ERR_02 (continued) — *MATHACL Module*

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

Function

VBOOST might have larger delay under certain operating conditions

PMCU_ERR_10 (continued)

PMCU Module

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

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

Description

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

Workaround

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

RTC_ERR_01

RTC Module

Category

Functional

Function

Some RTC Interrupts are not available in STANDBY1

RTC_ERR_01 (continued) — *RTC Module*

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

SPI_ERR_03 (continued) — *SPI Module*

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.

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

No

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

SYSOSC_ERR_02 (continued) *SYSOSC Module*

Function

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

Description

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

Workaround

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

SYSOSC_ERR_04 *SYSOSC Module*

Category

Functional

Function

SYSOSC accuracy degrades in FCL ON mode when SYSPLL is used

Description

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

Workaround

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

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

For 78MHz:
Set SYSPLLCFG1.PDIV = 0x3 and SYSPLLCFG1.QDIV to 38

SYSPLL_ERR_01 *SYSPLL Module*

Category

Functional

Function

SYSPLL Frequency may not lock to correct frequency when enabled.

Description

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

SYSPLL_ERR_01 (continued) — *SYSPLL Module*

Workaround

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

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

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

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

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

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

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

TIMER_ERR_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.

TIMER_ERR_01 (continued)

TIMx Module

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

TIMG Module

Category

Functional

Function

Writing 0 to CLKEN bit does not disable counter

Description

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

Workaround

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

TIMER_ERR_07

Initial repeat counter has 1 less period than next repeats Module

Category

Functional

Function

TIMER

Description

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

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

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

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

UART_ERR_04 (continued) ***UART Module***

2. Baud rate of 9600 configured with 3x oversampling
3. UART fast clock request has been disabled
If the ULPCLK changes from SYSOSC to LFOSC in the middle of a UART RX transfer, it is observed that one bit is read incorrectly

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

UART_ERR_05 ***UART Module***

Category Functional

Function Limitation of debug halt feature in UART module

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

Workaround Please 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 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

UART_ERR_10 (continued) — *UART Module*

Function

BUSY bit setting is delayed for UART IrDA mode

Description

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

Workaround

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

UART_ERR_11 — *UART Module*

Category

Functional

Function

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

Description

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

Workaround

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

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.

VREF_ERR_01 (continued) — *VREF Module*

Workaround

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

VREF_ERR_02 — *VREF Module*

Category

Functional

Function

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

Description

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

Workaround

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

WWDT_ERR_01 — *WWDT Module*

Category

Functional

Function

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

Description

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

Workaround

Use WWDT0 for NMI purposes.

WWDT_ERR_02 — *WWDT Module*

Category

Functional

Function

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

Description

After a reset caused by 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 November 30, 2025 to December 31, 2025 (from Revision D (November 2025) to Revision E (December 2025)) **Page**

- CPU_ERR_03 Workaround was updated....7
- SYSOSC_ERR_04 Function was updated....23
- SYSOSC_ERR_04 Description was updated....23
- SYSOSC_ERR_04 Workaround was updated....23