Errata

MSPM0L222x, MSPM0L122x, MSPM0L222x-Q1, MSPM0L122x-Q1 Microcontrollers

ABSTRACT

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

Table of Contents

Trademarks

All trademarks are the property of their respective owners.

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 A	Rev B
ADC_ERR_05	✓	✓
ADC_ERR_06	✓	
AES_ERR_01	✓	✓
COMP_ERR_03	✓	✓
COMP_ERR_04	✓	✓
CPU_ERR_01	✓	✓
CPU_ERR_02	✓	✓
CPU_ERR_03	✓	✓
GPIO_ERR_04	✓	✓
I2C_ERR_01	✓	
I2C_ERR_03	✓	✓
I2C_ERR_04	✓	✓
I2C_ERR_05	✓	✓
I2C_ERR_06	✓	✓
I2C_ERR_07	✓	✓
I2C_ERR_08	✓	✓
I2C_ERR_09	✓	✓
I2C_ERR_10	✓	✓

Errata Number	Rev A	Rev B
I2C_ERR_13	✓	✓
KEYSTORE_ERR_01	✓	✓
LCD_ERR_01	✓	
LFSS_ERR_01	✓	
LFSS_ERR_02	✓	
LFSS_ERR_03	✓	✓
LFXT_ERR_01	✓	✓
LFXT_ERR_02	✓	✓
PMCU_ERR_08	✓	✓
PMCU_ERR_09	✓	✓
PMCU_ERR_10	✓	✓
RST_ERR_01	✓	✓
RTC_A_ERR_02	✓	✓
SPI_ERR_03	✓	✓
SPI_ERR_04	✓	✓
SPI_ERR_05	✓	✓
SPI_ERR_06	✓	✓
SPI_ERR_07	✓	✓
SRAM_ERR_01	✓	✓
SYSCTL_ERR_01	✓	✓
SYSCTL_ERR_02	✓	✓
SYSCTL_ERR_03	✓	✓
SYSCTL_ERR_04	✓	✓
SYSOSC_ERR_01	✓	✓
TAMPERIO_ERR_01	✓	✓
TIMER_ERR_01	✓	✓
TIMER_ERR_04	✓	✓
TIMER_ERR_06	✓	✓
TIMER_ERR_07	✓	✓
UART_ERR_01	✓	✓
UART_ERR_02	✓	✓
UART_ERR_03	✓	✓
UART_ERR_04	✓	✓
UART_ERR_05	✓	✓
UART_ERR_06	✓	✓
UART_ERR_07	✓	✓
UART_ERR_08	✓	✓
UART_ERR_09	✓	✓
UART_ERR_10	✓	✓
UART_ERR_11	✓	✓

2 Preprogrammed Software Advisories

Advisories that affect factory-programmed software.

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

3 Debug Only Advisories

Advisories that affect only debug operation.

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

Errata Number	Rev A	Rev B
GPIO_ERR_03	✓	✓

4 Fixed by Compiler Advisories

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

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

5 Device Nomenclature

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

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

MSP – Fully qualified production device

Support tool naming prefixes:

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

null: Fully-qualified development-support product.

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

"Developmental product is intended for internal evaluation purposes."

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

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

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

5.1 Device Symbolization and Revision Identification

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

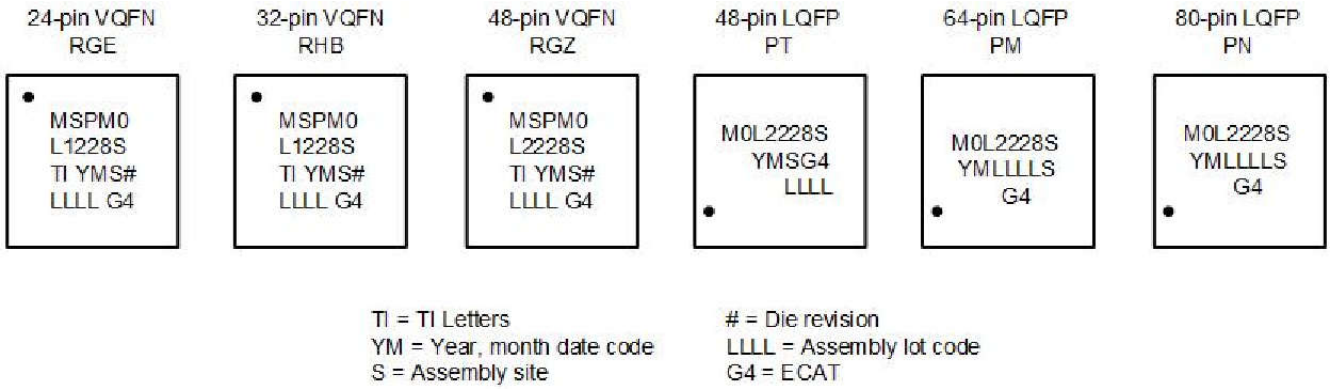

Figure 5-1. Package Symbolization

Table 5-1. Die Revisions

Revision Letter (package marking)	Version (in the device factory constants memory)
A	0x1
B	0x2

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

6 Advisory Descriptions

ADC_ERR_05 *ADC Module*

Category

Functional

Function

HW Event generated before enabling IP, ADC Trigger will stay in queue

Description

When the ADC receives a HW event trigger the pending event will go into the ADC trigger queue, even if CTL0.ENC = 0x0. Once ADC is set to CTL1.TRIGSRC = 0x1 (HW Trigger) and CTL0.ENC = 1 then the queued HW trigger will start the ADC sampling and conversion process. The pending HW request can get into the queue even when the CTL1.TRIGSRC = 0x0 (SW trigger), but will only start the ADC sampling when CTL1.TRIGSRC = 0x1 && CTL0.ENC = 0x1.

Workaround

Only configure the ADC F_SUB when expecting to use the HW Trigger, otherwise a pending request can get queued. If switching between SW and HW Trigger modes, a reset on the ADC (RSTCTL) will clear any pending queues, but will require the ADC to be reconfigured.

ADC_ERR_06 *ADC Module*

Category

Functional

Function

ADC Output code jumps degrading DNL/INL specification

Description

When a conversion error occurs, the error will be a fixed +/- 64LSB jump in the digital output code
of the ADC without a corresponding change in the ADC input voltage.

At worst case scenario, -40C, the error rate is 1 in 12M converted samples in 12-bit mode.
At 0C, error rate is 1 in 24M
At 55C, error rate is 1 in 60M
(VDD voltage and reference used has no impact on errata rate)

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.

ADC_ERR_06
(continued)

ADC Module

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

AES_ERR_01

AES Module

Category

Functional

Function

AES Saved Context Ready interrupt is not generating as expected

Description

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

Workaround

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

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_04

COMP Module

Category

Functional

Function

Connecting Temperature Sensor as Comparator input channel in standby makes Comparator output continuously toggle

Description

Temperature sensor is OFF in standby mode. Connecting the temperature sensor as a Comparator input channel in standby mode makes Comparator output continuously toggle.

Workaround

Do not use Temperature Sensor as Comparator input in standby mode.

CPU_ERR_01 *CPU Module*

Category

Functional

Function

CPU cache content can get corrupted

Description

Cache corruption can occur when switching between accessing Main flash memory, and other memory regions such as NONMAIN or Calibration data areas.

Workaround

Use the following procedure to access areas outside main memory safely:
1. Disable the cache by setting CTL.ICACHE to "0".
2. Perform needed access to memory area.
3. Re-enable cache by setting CTL.ICACHE to "1".

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 Event wake is another example of a process that will wake the device, but not flush the prefetcher.

CPU_ERR_03 (continued)

CPU Module

Workaround

Disable prefetcher before entering low power modes.

Example:
CPUSS.CTL.PREFETCH = 0x0; // disables prefetcher
SYSCTL.SOCLOCK.SHUTDNSTORE0 // Read from SHUTDOWN Memory
__WFI(); // or __WFE(); this function calls the transition into low power mode
CPUSS.CTL.PREFETCH = 0x1; // enables prefetcher

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 is not allowing PAD data to go to DIN register

Description

When configuring the fast wake only bit in the CTL register and forcing data to PAD in run mode, the data in PAD is not reflected in the DIN register. This is because the CTL register configuration prevents any data from flowing from the PAD to the DIN register.

Workaround

Avoid using the GPIO fastwake-only function when expecting data on the PAD 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

Description

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

I2C_ERR_01
(continued)

I2C Module

at the same time that the bus controller is attempting to signal the I2C STOP condition, preventing the STOP condition from completing successfully.

Workaround

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

I2C_ERR_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

KEYSTORE_ERR_01 — *KEYSTORE Module*

Category

Functional

Function

STATUS.STAT value can be 0 or 1 without key access

Description

STATUS.STAT has a reset value of 1 and turns to 0 under these conditions: 1. After reset, debugger access via the register window returns 0x00. 2. After reset, the first CPU read returns 0x01, while subsequent CPU reads return 0x00. 3) After reset, first reading any other KEYSTORE register and then reading STATUS.STAT return 0x00.

Workaround

STATUS.STAT=0x0 means "No Error" . For checking if a slot is valid or not (Whether key is present), check STATUS.VALID.

LCD_ERR_01 *LCD Module*

Category

Functional

Function

Increased time to achieve low-power mode specification when LCD module is off

Description

Increased time to achieve data sheet IDD current specification for low-power modes when LCD module is off.

Workaround

Turn on LCD module for standby mode to quickly achieve accurate IDD current draw. No workaround for shutdown mode.

LFSS_ERR_01 *LFSS Module*

Category

Functional

Function

Increased VBAT current when VDD is less than VBAT

Description

When VBAT is greater than VDD increased current is drawn through VBAT.

Workaround

No workaround. VDD must be greater than VBAT to maintain low VBAT current.

LFSS_ERR_02 *LFSS Module*

Category

Functional

Function

VBAT current increased when using LFXT and IWDT simultaneously

Description

VBAT current is significantly increased when IWDT is enabled with LFXT as clock source for VBAT.

Workaround

Do not run IWDT and use LFXT at the same time.

LFSS_ERR_03 *LFSS Module*

Category

Functional

Function

The LFSS interrupt flag is generated in RIS only when IMASK is enabled corresponding to the interrupt settings in EVENT0/1.

Description

The LFSS interrupt flag is generated in RIS only when IMASK is enabled corresponding to the interrupt settings in EVENT0/1. This is seen with all LFSS interrupts.

LFSS_ERR_03 (continued)

LFSS Module

Workaround

Generally, directly poll the RIS status for knowing the status of an interrupt is not recommended. But in case if user wants to know the interrupt status by polling (without enabling LFSS interrupt at CPU NVIC level), IMASK bit corresponding to that event can be enabled in either EVENT0 or EVENT1 MMR. This will update the RIS & MIS as expected.

LFXT_ERR_01

LFXT Module

Category

Functional

Function

LFXT clock is not getting active until writing something into LFCLKCFG mmr

Description

After setting the STARTLFXT, the CLKSTATUS is updating as expected, but LFXT clock is not seen on LFCLK_OUT until a write into the LFCLKCFG is made. And if LFXT is configured as a source to LFCLK tree, VBAT supply goes down and comes up back again, in that case the LFCLK is blank until a write into LFCLKCFG is made.

Workaround

Manually configuring LFCLKCFG register (such as XT1DRIVE field) to make LFXT valid, every time LFXT starts from blanking state.

LFXT_ERR_02

LFXT Module

Category

Functional

Function

Incorrect LFCLKMUX status for LFXT after a VDD domain power-cycle

Description

LFXT (in LFSS) is configured by user. If there is a VDD power-cycle, then LFCLKMUX status is getting incorrectly reset to the default state (LFCLKMUX status showing LFOSC) after VDD boots-up. The device does continue to use LFXT as the LFCLK source.

Workaround

1.Ignore the LFCLKMUX bit status if there is a VDD power-cycle. 2.Re-configure the LFXT if there is a VDD power-cycle.

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.

PMCU_ERR_08 (continued) — ***PMCU Module***

Workaround

No workaround.

PMCU_ERR_09 — ***PMCU Module***

Category

Functional

Function

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

Description

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

Workaround

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

PMCU_ERR_10 — ***PMCU Module***

Category

Functional

Function

VBOOST might have larger delay under certain operating conditions

Description

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

Workaround

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

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_A_ERR_02 *RTC Module*

Category

Functional

Function

Contents of LFSS-RTC TS data registers not reset on LFSS SW POR reset (LFSSRST)

Description

On LFSS SW POR reset (LFSSRST), the RTC TS(Time Stamp) data registers do not reset to 0x0.

Workaround

Use TSCLR register to manually clear the value in RTC TS data registers.

SPI_ERR_03 *SPI Module*

Category

Functional

Function

When configured as peripheral, enable CSCLR will result in the received data will have a right shift in SPH=0 mode

Description

When enabled CSCLR in peripheral mode, if there has glitch on SCK line when CS is active or inactive, the received data will have 1-bit right shift in the next first frame. This issue is seen in Motorola SPI Frame Format with SPH=0, and will impact multi-peripheral mode which has the case that SCK toggle during CS inactive.

Workaround

1. Set CSCLR=0h.
2. Always drop the first frame when set CSCLR=1h in SPH=0 mode.

SPI_ERR_04 *SPI Module*

Category

Functional

Function

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

Description

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

Workaround

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

SPI_ERR_05 *SPI Module*

Category

Functional

SPI_ERR_05
(continued)

SPI Module

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.

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_01

SRAM Module

Category

Functional

SRAM_ERR_01 (continued) ***SRAM Module***

Function

Mixing No-ECC/Parity aperture and ECC aperture accesses from CPU/DMA can lead to functional errors

Description

If CPU accesses are configured to use the No-ECC/Parity aperture and DMA to use the ECC aperture, it may lead to intermittent faults.

Workaround

Do not mix and match No-ECC/Parity/ECC aperture accesses between CPU and DMA. Use the same type of aperture for both CPU and DMA accesses. If a safety mechanism requires injecting faults, then avoid using DMA concurrently while this safety diagnostic mechanism is being exercised by the CPU software.

SYSCTL_ERR_01 ***SYSCTL Module***

Category

Functional

Function

SW-POR functionality is combined with HW-POR

Description

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

Workaround

No

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

IF 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 two cycles when SYSOSC shifts from 4MHz back to 32MHz (either upon exit from STOP1 to RUN mode or upon an asynchronous fast clock request that forces SYSOSC to 32MHz).

Workaround

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

OR

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

TAMPERIO_ERR_01	*TAMPERIO Module*
Category	Functional
Function	Tamper IOMUX state will lose latch status when VDD supply lost
Description	LFSS I/O state (when configured in IOMUX mode) will be latched & retained over SHUTDOWN mode transition of SoC. But over a VDD power-cycle the state will be lost.
Workaround	To retain a LFSS I/O state over a VDD power-cycle, configure it in Tamper mode.

TIMER_ERR_01	*TIMx Module*
Category	Functional
Function	Capture mode captures incorrect value when using hardware event to start timer
Description	When using any timer instance in capture mode, starting the timer using a zero (ZCOND) or load (LCOND) condition causes the timer to capture the zero or load value instead of the captured value in the respective TIMx.CC register. This affects periodic use cases such as period and pulse width capture.
Workaround	Use the below software flow to calculate the period or pulse width. See the timx_timer_mode_capture_duty_and_period in the MSPM0-SDK for an example of the workaround. 1.Disable ZCOND or LCOND by setting to 0h. 2.When a capture occurs, the capture value is correctly captured in TIMx.CC 3.Restart the timer by setting TIMx.CTR to the reload value (load or 0).

TIMER_ERR_04	*TIMER Module*
Category	Functional
Function	TIMER re-enable may be missed if done close to zero event
Description	When using a TIMER in one shot mode, TIMER re-enable may be missed if done close to zero event. The HW update to the timer enable bit will take a single functional clock cycle. For example, if the timer's clock source is 32.768kHz and clock divider of 3, then it will take ~100us to have the enable bit set to 0 properly.
Workaround	Wait 1 functional clock cycle before re-enabling the timer OR the timer can be disabled first before re-enabling.

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

Disable the counter with CTRCTL.EN = 0, then reenable 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.

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.

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

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_03 ***UART Module***

Category

Functional

Function

UART RX interrupt erroneously set with 3x oversampling and MFCLK or BUSCLK as clock source

Description

When using BUSCLK or MFCLK for UART and 3x oversampling, the RXINT is getting set incorrectly. When using the UART module with BUSCLK or MFCLK as the source, and 3x oversampling mode, the RX interrupt may be set erroneously. Under these conditions TX data may also be corrupted.

Workaround

Use a higher oversampling rate when using BUSCLK or MFCLK. If 3x oversampling is required, utilize LFCLK.

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

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

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

UART_ERR_07 (continued)

UART Module

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

7 Revision History

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

Changes from December 18, 2024 to November 30, 2025 (from Revision B (December 2024) to Revision C (November 2025)) **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