Errata

MSPM0C1103, MSPM0C1104, MSPS003F3, MSPS003F4 Microcontrollers

ABSTRACT

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

Table of Contents

1 Functional Advisories

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

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

Errata Number	Rev B
ADC_ERR_03	✓
ADC_ERR_05	✓
ADC_ERR_06	✓
ADC_ERR_09	✓
CPU_ERR_01	✓
CPU_ERR_02	✓
I2C_ERR_03	✓
I2C_ERR_05	✓
PMCU_ERR_06	✓
PMCU_ERR_13	✓
SPI_ERR_03	✓
SPI_ERR_04	✓
SPI_ERR_05	✓
SPI_ERR_06	✓
SPI_ERR_07	✓
SYSOSC_ERR_02	✓
TIMER_ERR_01	✓
TIMER_ERR_04	✓
TIMER_ERR_06	✓
UART_ERR_01	✓
UART_ERR_02	✓
UART_ERR_04	✓

Errata Number	Rev B
UART_ERR_05	✓
UART_ERR_06	✓
UART_ERR_07	✓
UART_ERR_08	✓

2 Preprogrammed Software Advisories

Advisories that affect factory-programmed software.

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

3 Debug Only Advisories

Advisories that affect only debug operation.

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

4 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 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.

4.1 Device Symbolization and Revision Identification

The package diagrams below indicate the package symbolization scheme, and Table 4-1 defines the device revision to version ID mapping.

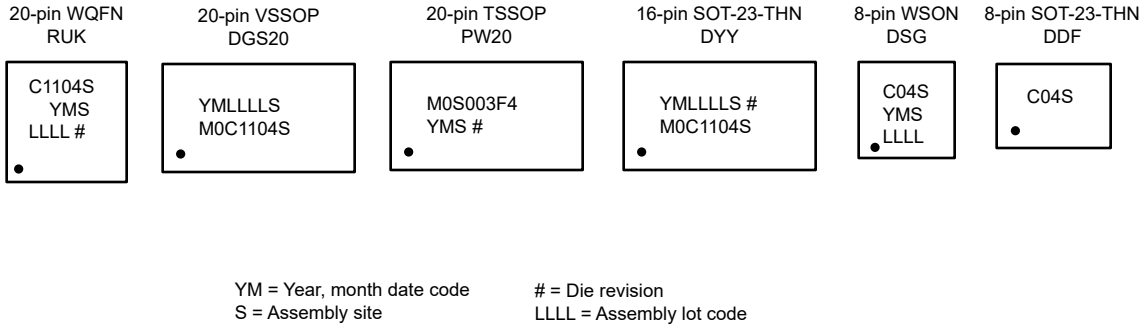

Figure 4-1. Package Symbolization

Table 4-1. Die Revisions

Revision Letter (package marking)	Version (in the device factory constants memory)
B	2

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.

5 Advisory Descriptions

5.1 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.

ADC_ERR_03	***ADC Module***
Category	Functional
Function	ADC SNR and DNL performance is worse when using VSSA for ADC ground reference.
Description	When the ADC is using VSS as the ground reference, noise sources on the VSS ground will impact the ADC results, which results in worsened SNR and DNL performance.
Workaround	Workaround 1: Use VREF+ and VREF- pins for the ADC reference. With the VREF- pin, the ADC will have its analog ground reference lessening the influence of external noise sources. Workaround 2: Reduce noise sources on VSS; common sources are CPU or Digital switching noise

ADC_ERR_05	***ADC Module***
Category	Functional
Function	HW Event generated before enabling IP, ADC Trigger will stay in queue
Description	When ADC is configured in HW event trigger mode and the trigger is generated before enabling the ADC, the ADC trigger will stay in queue. Once ADC is enabled, it will trigger sampling and 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	The ADC may have errors at a rate as high as 1 in 25M conversions in 12-bit mode. When a conversion error occurs, it will be a maximum +/- 64LSB random jump in the digital output of the ADC without a corresponding change in the ADC input voltage. The magnitude of this jump is larger near major transitions in the bit values of the ADC result (more bits transitioning from 1->0, or 0->1), and largest around midscale (2048 or 0x800). Depending on the application needs the best workaround may vary, but the following

ADC_ERR_06
(continued)

ADC Module

workarounds in software are proposed. Selection of the best workaround is left to the judgment of the system designer.

Workaround

Workaround 1: Upon ADC result outside of application threshold (via ADC Window Comparator or software thresholding), trigger or wait for another ADC result before making critical system decisions

Workaround 2: During post-processing, discard ADC values which are sufficiently far from the median or expected value. The expected value should be based on the average of real samples taken in the system, and the threshold for rejection should be based on the magnitude of the measured system noise.

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

ADC_ERR_09

ADC Module

Category

Functional

Function

ADC offset error needs to be calibrated in application code

Description

The calibration data of ADC offset error is not applied correctly AND needs to be implemented in application code.

Workaround

The calibration data is stored at address 0x41C40040 in factory region. Two DriverLib APIs DL_ADC12_getADCOffsetCalibration AND DL_FactoryRegion_getADCOffset have been implemented in the SDK to facilitate this.

```
__STATIC_INLINE int16_t DL_ADC12_getADCOffsetCalibration(float userRef)
{
float adcBuff = DL_FactoryRegion_getADCOffset() * (3.3 / userRef);
return (int16_t)(round(adcBuff));
}
__STATIC_INLINE float DL_FactoryRegion_getADCOffset(void)
{
return ((float) (*(int16_t *) (0x41C40040)));
}
```

The calibration data can be saved into a variable AND subsequently applied to the ADC conversion result.

Below is the example code demonstrating how to apply the calibration data, which has been integrated into the ADC examples provided in the SDK:

```
volatile uint16_t gAdcResult;
```

ADC_ERR_09
(continued)

ADC Module

```
volatile int16_t gADCOffset;
gADCOffset =
DL_ADC12_getADCOffsetCalibration(ADC12_0_ADCMEM_0_REF_VOLTAGE_V);
gAdcResult = DL_ADC12_getMemResult(ADC12_0_INST, DL_ADC12_MEM_IDX_0);
int16_t adcRaw = (int16_t) gAdcResult + gADCOffset;
if (adcRaw < 0)
{
adcRaw = 0;
}
if (adcRaw > 4095)
{
adcRaw = 4095;
}
gAdcResult = (uint16_t) adcRaw;
```

The ADC offset calibration data can be applied for the example use cases below.

ADC offset error needs to be calibrated in application code

- Utilizing ADC without DMA - the offset needs to be added to ADC result from the register MEMRES.
- Utilizing ADC with DMA - the offset needs to be added to ADC result stored in the memory address.

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

CPU_ERR_02 (continued)

CPU Module

Function

Limitation of disabling prefetch feature for CPUSS

Description

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

Workaround

Disable prefetch. Then issue a memory access to the shutdown memory (SHUTDNSTORE) in SYSCTL.

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_05

I2C Module

Category

Functional

Function

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

Description

If ACTIVE bit is toggled during an ongoing transfer, its state machine will be reset. However, the SDA and SCL output which is driven by the controller will not get reset. There is a situation where SDA is 0 and 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. Target's USBUSY 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.

PMCU_ERR_06

PMCU Module

Category

Functional

PMCU_ERR_06
(continued)

PMCU Module

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 ensured by putting code in SRAM while a flash operation is on-going.

PMCU_ERR_13

PMCU Module

Category

Functional

Function

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

Description

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

Workaround

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

SPI_ERR_03

SPI Module

Category

Functional

Function

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

Description

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

SPI_ERR_03 (continued)

SPI Module

Workaround

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

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 receiving 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 in peripheral simultaneous transmit and receive mode.

SPI_ERR_05

SPI Module

Category

Functional

Function

SPI Peripheral Receive Timeout interrupt is setting irrespective of RXFIFO data

Description

When using SPI timeout interrupt, the RXTIMEOUT counter started decrementing from the point that peripheral is stopped receiving SPI clock and setting the RXTIMEOUT interrupt irrespective of data exists in RXFIFO or not, which does not match the description in the TRM: SPI peripheral receive timeout(RTOUT) interrupt is "asserted when the receive FIFO is not empty, and no further data is received in the specified time at CTL1.RXTIMEOUT.

Workaround

Repeat load RXTIMEROUT counter value while receive FIFO is empty, and start timeout counting only when receive FIFO gets any data.

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

SPI_ERR_06 (continued) — ***SPI Module***

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

TXFIFO for SPI peripheral uses a bus synchronization scheme. While the control signal from BUSCLK domain to SPICLK domain is getting transferred, if read and write pointer points to the same head of FIFO, there is a possibility of data coherence issues. This issue only happens in corner case scenarios, and cause no underflow event being generated.

Workaround

Customer must ensure that TXFIFO on peripheral can never be empty when Controller is addressing the peripheral.

SYSOSC_ERR_02 — ***SYSOSC Module***

Category

Functional

Function

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

Description

MFCLK will not start to toggle in below scenario:
1. FCL mode is enabled and then MFCLK is enabled
2. Enter a low power mode where SYSOSC is disabled (SLEEP2/STOP2/STANDBY0/ STANDBY1).
3. Now async requst is received from some peripherals which use MFCLK as functional clock.
This is happening because on receiving async request, SYSOSC gets enabled and ulpclk becomes 32MHz. But the SYSOSCTRIM FSM does not move from DISABLE state. Due to this, the FCL bases MFCLK is gated off and it does not toggle at all.

Workaround

Avoid using this scenario condition.

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 *TIMG Module*

Category

Functional

Function

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

Description

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

Workaround

TIMER can be disabled first before re-enabling.

TIMER_ERR_06 *TIMG Module*

Category

Functional

Function

Writing 0 to CLKEN bit does not disable counter

Description

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

Workaround

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

UART_ERR_01 *UART Module*

Category

Functional

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

Function

UART start condition not detected when transitioning to STANDBY1 Mode

Description

After servicing an asynchronous fast clock request that was initiated by a UART transmission while the device was in STANDBY1 mode, the device will return to STANDBY1 mode. If another UART transmission begins during the transition back to STANDBY1 mode, the data is not correctly detected and received by the device.

Workaround

Use STANDBY0 mode or higher low power mode when expecting repeated UART start conditions.

UART_ERR_02 — ***UART Module***

Category

Functional

Function

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

Description

UART End Of Transmission (EOT) interrupt does not trigger when the device is set for transmit only (CTL0.TXE = 1, CTL0.RXE = 0). EOT successfully triggers when device is set for transmit and receive (CTL0.TXE = 1, CTL0.RXE = 1)

Workaround

Set both CTL0.TXE and CTL0.RXE bits when utilizing the UART end of transmission interrupt. Note that you do not need to assign a pin as UART receive.

UART_ERR_04 — ***UART Module***

Category

Functional

Function

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

Description

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

Workaround

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

UART_ERR_05 — ***UART Module***

Category

Functional

UART_ERR_05
(continued)

UART Module

Function

Limitation of debug halt feature in UART module

Description

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

Workaround

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

UART_ERR_06

UART Module

Category

Functional

Function

Unexpected behavior RTOUT/Busy/Async in UART 9-bit mode

Description

UART receive timeout (RTOUT) is not working correctly in multi node scenario, where one UART will act as controller and other UART nodes as peripherals , each peripheral is configured with different address in 9-bit UART mode.
First UART controller communicated with UART peripheral1, by sending peripheral1's address as a first byte and then data, peripheral1 has seen the address match and received the data. Once controller is done with peripheral1, peripheral1 is not setting the RTOUT after the configured timeout period, if controller immediately starts the communication with another UART peripheral (peripheral2) which is configured with different address on the bus. The peripheral1 RTOUT counter is resetting while communication ongoing with peripheral2 and peripheral1 setting 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 at even when the line is IDLE and FIFO has some elements. This means that RTOUT interrupts will not work in IDLE LINE MODE.
In case of an Address Mismatch, RTOUT counter is reloaded when it sees toggles on the Rx line.
In case of a multi-responder scenario this could lead to an indefinite delay in getting

UART_ERR_07 (continued) — *UART Module*

an RTOUT event when communication is happening between the commander and some other responder.

Workaround

Do not enable RTOUT feature when UART module is used either in IDLELINE mode/ multi-node UART application.

UART_ERR_08 — *UART Module*

Category

Functional

Function

STAT BUSY does not represent the correct status of UART module

Description

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

Workaround

Poll TXFIFO status and the CTL0.ENABLE register bit to identify BUSY status.

6 Revision History

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

Changes from Revision A (May 2024) to Revision B (March 2025)	Page
• Updated Device Revision, Updated ADC_ERR_06, Added ADC_ERR_03, ADC_ERR_05, ADC_ERR_09, CPU_ERR_01, CPU_ERR_02, I2C_ERR_05, PMCU_ERR_06, PMCU_ERR_07, PMCU_ERR_13, SPI_ERR_03, SPI_ERR_04, SPI_ERR_05, SPI_ERR_06, SPI_ERR_07, SYSOSC_ERR_02, TIMER_ERR_01, TIMER_ERR_04, TIMER_ERR_06, UART_ERR_01, UART_ERR_02, UART_ERR_04, UART_ERR_05, UART_ERR_06, UART_ERR_07, UART_ERR_08,	1

Changes from Revision * (October 2023) to Revision A (May 2024)	Page
• Removed ADC_ERR_04, ADC_ERR_05, PMCU_ERR_04, PMCU_ERR_05, PMCU_ERR_06, and UART_ERR_01; Added ADC_ERR_06, I2C_ERR_03, and PMCU_ERR_07	1

IMPORTANT NOTICE AND DISCLAIMER

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

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

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

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

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

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265