Errata

MSPM0L111x Microcontrollers

ABSTRACT

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

Table of Contents

Trademarks

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, B
ADC_ERR_05	✓
AES_ERR_01	✓
CLK_ERR_02	✓
CPU_ERR_02	✓
CPU_ERR_03	✓
CPU_ERR_04	✓
DMA_ERR_02	✓
FCC_ERR_01	✓
FLASH_ERR_03	✓
FLASH_ERR_04	✓
FLASH_ERR_05	✓
FLASH_ERR_08	✓
FLASH_ERR_09	✓
GPIO_ERR_05	✓
GPIO_ERR_06	✓
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, B
I2C_ERR_13	✓
I2C_ERR_15	✓
KEYSTORE_ERR_01	✓
PMCU_ERR_10	✓
PMCU_ERR_12	✓
PMCU_ERR_14	✓
RST_ERR_01	✓
RST_ERR_02	✓
SPI_ERR_02	✓
SPI_ERR_04	✓
SPI_ERR_05	✓
SPI_ERR_06	✓
SPI_ERR_07	✓
SPI_ERR_10	✓
SYSCTL_ERR_01	✓
SYSCTL_ERR_02	✓
SYSCTL_ERR_03	✓
SYSCTL_ERR_04	✓
SYSCTL_ERR_05	✓
SYSCTL_ERR_06	✓
SYSCTL_ERR_10	✓
SYSOSC_ERR_01	✓
SYSOSC_ERR_02	✓
SYSOSC_ERR_04	✓
SYSOSC_ERR_05	✓
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_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.

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. Please note that X marked development samples should be operated in -40°C to 85°C temperature range.

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.

The device package diagrams below indicate the package symbolization scheme.

The package diagrams below indicate the package symbolization scheme, and x defines the device revision to version ID mapping.

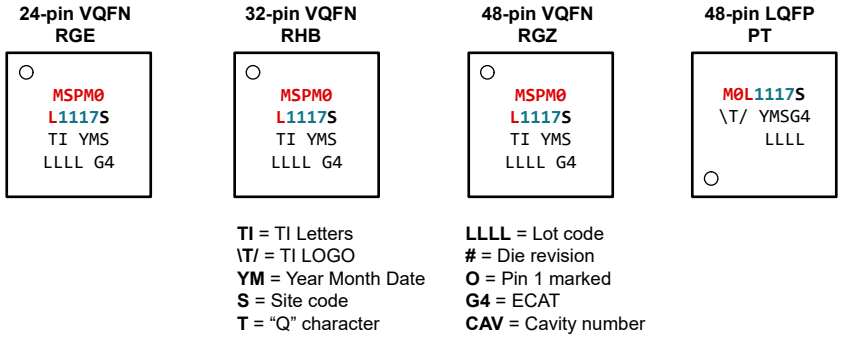

Figure 5-1. Package Symbolization

Die Revisions

Revision Letter (package marking)	Version (in the device constants memory)
A	0x0
B	0x0

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 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.

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.

CLK_ERR_02 *CLK Module*

Category

Functional

Function

When HFCLK_IN is the source of HFCLK and HSCLK, MFCLK is incorrectly sourced from BUSCLK

Description

When HSCLK is sourced exclusively from HFCLK, and HFCLK_IN is configured as the source of HFCLK, MFCLK is incorrectly set to BUSCLK instead of 4MHz.

Workaround

No workaround

CPU_ERR_02 *CPU Module*

Category

Functional

Function

Limitation of disabling prefetch/cache for CPUSS

CPU_ERR_02 (continued) — ***CPU Module***

Description

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

Workaround

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

After the memory access completes, the prefetcher/cache will be disabled.
Example:
CPUSS->CTL = (CPUSS_CTL_PREFETCH_DISABLE |
CPUSS_CTL_ICACHE_DISABLE); //disables instruction caching and pre-fetching
DL_SYSCTL_setShutdownStorageByte(DL_SYSCTL_SHUTDOWN_STORAGE_BYTE_X
, data) // Save a byte to SHUTDOWN memory

CPU_ERR_03 — ***CPU Module***

Category

Functional

Function

Prefetcher can fetch wrong instructions when transitioning into Low power modes

Description

When transitioning into low power modes and there is a pending prefetch, the prefetcher can erroneously fetch incorrect data (all 0's). When the device wakes up, if the prefetcher and cache do not get overwritten by ISR code, then the main code execution from flash can get corrupted. For example, if the ISR is in the SRAM, then the incorrect data that was prefetched from Flash does not get overwritten. When the ISR returns the corrupted data in the prefetcher can be fetched by the CPU resulting in incorrect instructions. A HW Event wake is another example of a process that will wake the device, but not flush the prefetcher.

Workaround

Disable prefetcher before entering low power modes.

Example:
CPUSS->CTL &= 0x6; // disables prefetcher, maintains other settings
SYSCTL.SOCLOCK.SHUTDNSTORE0 // Read from SHUTDOWN Memory
__WFI(); // or __WFE(); this function calls the transition into low power mode
CPUSS->CTL |= 0x1; // enables prefetcher

CPU_ERR_04 — ***CPU Module***

Category

Functional

Function

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

Description

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

CPU_ERR_04 (continued) — ***CPU Module***

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

Workaround

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

DMA_ERR_02 — ***DMA Module***

Category

Functional

Function

DMA mixed byte transfers of 64-bit to 128-bit and 128-bit to 64-bit do not work

Description

When performing a DMA transfer with a source width of 128-bits and a destination width of 64-bits, or a source width of 64-bits and a destination width of 128-bits, the transfer will not complete correctly.

Workaround

No workaround exists. Use matched-width transfers only (64-bit to 64-bit or 128-bit to 128-bit).

FCC_ERR_01 — ***FCC Module***

Category

Functional

Function

FCC behaves incorrectly when BUSCLK is sourced from LFCLK

Description

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

Workaround

No workaround

FLASH_ERR_03 — ***FLASH Module***

Category

Functional

Function

Flash access with 2 wait states followed by invalid bootcode access will cause next flash access to also throw a violation

FLASH_ERR_03 (continued) ***FLASH Module***

Description

Doing a Flash access followed by a access to BOOTCODE when you have 2 wait states will cause the next flash access to also cause a violation.

Workaround

Do not attempt to access boot-code region post-boot phase. Otherwise, there will need to be 4 clock cycles in between the bootcode violation and next correct flash access.

FLASH_ERR_04 ***FLASH Module***

Category

Functional

Function

Wrong Address will get reported in the SYSCTL_DEDERRADDR if the error is not in the main flash region.

Description

Workaround

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

If the return address of the SYSCTL_DEDERRADDR returns a 0x00Dxxxxx, do an OR operation with 0x41000000 to get the proper address for the Databank return address. For example, if SYSCTL_DEDERRADDR = 0x00D0012A, the real address would be 0x00D0012A.

FLASH_ERR_05 ***FLASH Module***

Category

Functional

Function

DEDERRADDR can have incorrect reset value

Description

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

Workaround

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

FLASH_ERR_08 ***FLASH Module***

Category

Functional

FLASH_ERR_08 (continued) ***FLASH Module***

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

FLASH_ERR_09 ***FLASH Module***

Category

Functional

Function

Static write protection on BANK1 is not functional with bank1 erase under specific flash swap policy

Description

If the device flash memory is not the maximum device(e.g. MSPM0L111x family have two different devices as MSPM0L1117 and MSPM0L1116, just MSPM0L1116 has this issue), will get the issue that bank1 be erased unexpectedly under the conditions below: Failed case 1: Static write protection on BANK1 enabled, flash swap policy disabled, USEUPPER bit of SECCFG_Reg.FLBANKSWP register is 0 or 1, the BANK1 is erased unexpectedly with bank erase command. Failed case 2: Static write protection on BANK1 enabled, flash swap policy enabled, USEUPPER bit of SECCFG_Reg.FLBANKSWP register is 0, the BANK1 is erased unexpectedly with bank erase command.

Workaround

1. Use sector erase instead of bank erase for Bank1. Or 2. Use maximum flash device in the same device family.

GPIO_ERR_05 ***GPIO Module***

Category

Functional

Function

Writing to GPIO DOUTTGL registers might get missed when a DMA transfer is ongoing

Description

The GPIO DMAMASK register information is mistakenly applied to a CPU write to the DOUTTGL register when a concurrent DMA transfer is in progress.

Workaround

In the application code, ensure that the GPIO DMAMASK bit is set to 1 for the corresponding bit in the DOUTTGL register, before a CPU write access to the DOUTTGL register is issued. If no DMA transfer to any of the GPIO registers is required, the GPIO DMAMASK can be configured as 0xFFFFFFFF during the IO initialization step. This will solve the conflict of this errata. If the application also requires DMA write transfers to the GPIO registers, it is recommended that the application not use both DMA and CPU to write to the DOUTTGL register of the same GPIO module in the device. If the device has

GPIO_ERR_05
(continued)

GPIO Module

multiple GPIO modules, the DMA and the CPU can simultaneously write to the DOUTTGL register of different GPIO modules (while still requiring that the GPIO DMAMASK be configured for the GPIO module the CPU is writing to).

GPIO_ERR_06

GPIO Module

Category

Functional

Function

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

Description

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

Workaround

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

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

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

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

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

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

I2C_ERR_04 *I2C Module*

Category

Functional

Function

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

Description

1: SCL line grounded and released, device indefinitely pulls SCL low.

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

Workaround

If the I2C target application does not require data reception in low power mode using Async fast clock request, disabling SWUEN by default is recommended, including during reset or power cycle. In this case, bug description 1 and 2 does not occur.

If the I2C target application requires data reception in low power mode using Async fast clock request, enable SWUEN just before entering low power and clear SWUEN after low power exit. Even in this scenario, bug description 1 and 2 can occur when the I2C target is in low power, it will indefinitely stretch the SCL line if there is a continuous clock stretching or timeout caused by another device on the bus. To recover from this situation, enable the low timeout interrupt on the I2C target device, reset and re-initialize the I2C module within the low timeout ISR.

I2C_ERR_05 *I2C Module*

Category

Functional

Function

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

Description

If ACTIVE bit is toggled during an ongoing transfer, its state machine will be reset. However, the SDA and SCL output which is driven by the master will not get reset. There is a situation where SDA is 0 and master has gone into IDLE state, here the master won't be able to move forward from the IDLE state or update the SDA value. Slave'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 master 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 I2C Master goes into IDLE without

I2C_ERR_06 (continued) — *I2C Module*

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.

Workaround

Do not use SMBus High timeout feature when I2C clock 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.

I2C_ERR_09
(continued)

I2C Module

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

Function

I2C Busy status is enabled preventing low power entry

Description

When in I2C Target mode, the I2C Busy Status stays high after a transaction if there is no STOP bit.

Workaround

Program the I2C controller to send the STOP bit and don't send a NACK for the last byte. Terminate any I2C transfer with a STOP condition to maintain proper BUSY status and asynchronous clock request behavior (for low power mode reentry).

I2C_ERR_13

I2C Module

Category

Functional

Function

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

Description

After setting the CCTR.BURSTRUN bit to initiate an I2C controller transfer, it takes approximately 3 I2C functional clock cycles for the BUSY status to be asserted. If polling for the BUSY bit is used immediately after setting CCTR.BURSTRUN to wait for transfer completion, the BUSY status might be checked before it is set. This problem is more likely to occur with high CLKDIV values (resulting in a slower I2C functional clock) or under higher compiler optimization levels.

Workaround

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

I2C_ERR_15

I2C Module

Category

Functional

Function

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

I2C_ERR_15
(continued)

I2C Module

Description

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

Workaround

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

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.

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.

PMCU_ERR_12

PMCU Module

Category

Functional

Function

BOR cold boot spec violation

Description

The BOR cold boot spec could be violated, the max spec at is 1.65V.

PMCU_ERR_12
(continued)

PMCU Module

Workaround

No Workaround. While cold booting the device operate the device above 1.65V.

PMCU_ERR_14

PMCU Module

Category

Functional

Function

GPIO triggered DMA transfer is got pending until wakeup in low power mode with FCL enabled

Description

When FCL is enabled and in low power mode(STOP2/STANDBY0/STANDBY1), DMA is triggered by event channel from GPIO, this DMA transmission won't start until device is woken up by event or interrupt, and additional power consumption will be seen before waking up.

Workaround

Use GPIO interrupt to start DMA transfer by software, instead of using DMA triggered by GPIO event. And after exit interrupt, re-enter low power mode again.

RST_ERR_01

RST Module

Category

Functional

Function

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

Description

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

Workaround

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

RST_ERR_02

RST Module

Category

Functional

Function

Device May Fail to Power Up When NRST deassertion coincides with initial LFOSCGOOD assertion

RST_ERR_02
(continued)

RST Module

Description

At power-up event, the internal low-frequency oscillator (LFOSC) is enabled once the internal core voltage (VCORE) reaches a stable level. The LFOSCGOOD signal asserts high within 250us (min) to 1.5ms (max) of VCORE reaching stability. The VBOR+ threshold of VDD can be used as a reference point for when VCORE becomes stable. If the NRST deassertion (low-to-high transition) coincides with the LFOSCGOOD rising edge within a 200ns window, the NRST deassertion is not detected and the device remains in reset, failing to exit the boot phase of the startup sequence. Since the NRST deassertion timing is governed by both the VDD ramp rate and the external resistor and capacitor (RC) network on the NRST circuit, this condition can be triggered in practice - a known susceptible configuration includes a 47 k resistor and 10 nF capacitor on the NRST circuit.

Workaround

Workaround 1: Use a small enough pull-up resistor in the RC filter to ensure the NRST signal rises to 90% of VDD at lesat 50us before the LFOSCGOOD min assertion time (250us), ensuring the NRST deassertion edge is well clear of the LFOSCGOOD assertion window. A known compliant configuration is a 1 k resistor with a 10 nF capacitor on the NRST circuit. Workaround2: Employ an external reset controller that holds NRST at a logic low level for a minimum of 2.0 ms from the start of the VDD ramp, ensuring the NRST deassertion occurs after the LFOSCGOOD maximum assertion time (1.5ms).

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 may not properly propagate the first few clock cycles and data bits of the first byte sent out.

Workaround

To ensure 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_04

SPI Module

Category

Functional

Function

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

SPI_ERR_04
(continued)

SPI Module

Description

In case of SPI peripheral in only 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(slave) TXFIFO and TXFIFO is empty.

Workaround

Do not use SPI peripheral only receive mode. Set SPI in peripheral(slave) 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

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

SPI_ERR_07
(continued)

SPI Module

Description

When SPI.CTL0.SPH = 0 and the device is configured as the SPI peripheral.

If there is a write to the TXFIFO WHILE there is a read request from the SPI controller, then an underflow event may not be generated as the read/write request is happening simultaneously.

Workaround

Ensure the TXFIFO is not empty when the SPI Controller is addressing the device, this can be done by preloading data to avoid a write and read to the same TXFIFO address. Alternatively, data checking strategies, like CRC, can be used to verify the packets were sent properly, then the data can be resent if the CRC doesn't match.

SPI_ERR_10

SPI Module

Category

Functional

Function

DMA is not sampling the latest SPI trigger count

Description

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

Workaround

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

SYSCTL_ERR_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

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

SYSCTL_ERR_05 *LFCLK Module*

Category

Functional

Function

LFCLK not working on exit from shutdown

Description

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

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

Workaround

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

SYSCTL_ERR_06 ***CLK_OUT Module***

Category

Functional

Function

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

Description

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

Workaround

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

SYSCTL_ERR_10 ***SYSCTL Module***

Category

Functional

Function

High wakeup times needed while LPM entry with ULPCLK 4MHz clock

Description

The wakeup time would be additional ~10us time than the typical value when entry with ULPCLK 4MHz like STOP0.

Workaround

Avoid entering low-power mode directly when ULPCLK is 4MHz. MCU can wakeup to RUN0 mode firstly then enter into LPM.

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).

SYSOSC_ERR_01
(continued)

SYSOSC Module

Workaround

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

OR

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

SYSOSC_ERR_02 *SYSOSC Module*

Category

Functional

Function

MFCLK fails to start up in certain conditions.

Description

MFCLK will not start to toggle in the 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/ STANBY1).

3. Now asynchronous clock request is received from some peripheral which uses MFCLK as its functional clock.

Workaround

Avoid the above scenario - check that your system does not use FCL and MFCLK along with the listed low power modes while asynchronous fast clock requests are enabled.

SYSOSC_ERR_04 *SYSOSC Module*

Category

Functional

Function

SYSOSC Accuracy Degradation in FCL ON Mode

Description

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

Workaround

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

SYSOSC_ERR_05 *SYSOSC Module*

Category

Functional

Function

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

Description

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

SYSOSC_ERR_05 (continued) ***SYSOSC Module***

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

Workaround

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

TIMER_ERR_04 ***TIMER 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 ***TIMER Module***

Category

Functional

Function

Writing 0 to CLKEN bit does not disable counter

Description

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

Workaround

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

TIMER_ERR_07 ***TIMG Module***

Category

Functional

Function

Initial repeat counter has 1 less period than next repeats

Description

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

TIMER_ERR_07 (continued) — *TIMG Module*

zero events would appear on the first repeat counter and 4 observable zero events would appear on the following repeat counter sequences.

Workaround

Set the initial RCLD value to 1 more than the expected RCLD, then in the ISR for the Repeat Counter Zero Event (REPC), set the RCLD to the intended RCLD value.

For example, if intending to have 4 repeats, set the initial RCLD value to RCLD = 0x5, then in the timer ISR for the REPC interrupt, set RCLD = 0x4. Now all timer repeats will have the same number of zero/load events.

UART_ERR_01 — *UART Module*

Category

Functional

Function

UART start condition not detected when transitioning to STANDBY1 Mode

Description

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

Workaround

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

UART_ERR_02 — *UART Module*

Category

Functional

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

UART_ERR_04 (continued)

UART Module

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

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_10 *UART Module*

Category

Functional

Function

BUSY bit setting is delayed for UART IrDA mode

Description

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

Workaround

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

UART_ERR_11	*UART Module*
Category	Functional
Function	UART Receive timeout starts counting earlier than expected during the STOP bit transaction
Description	During the STOP bit transaction the Receive timeout will start counting in the middle of the STOP bit transaction, which can cause an unintended RTOUT interrupt if the 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 November 30, 2025 to July 31, 2026 (from Revision A (November 2025) to Revision B (July 2026)) **Page**

IMPORTANT NOTICE AND DISCLAIMER

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

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

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

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

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

Last updated 10/2025