Errata

MSPM0H3215, MSPMH03216 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 A
CPU_ERR_02	✓
CPU_ERR_03	✓
FLASH_ERR_02	✓
FLASH_ERR_04	✓
FLASH_ERR_05	✓
FLASH_ERR_08	✓
GPIO_ERR_04	✓
GPIO_ERR_08	✓
I2C_ERR_04	✓
I2C_ERR_05	✓
I2C_ERR_06	✓
I2C_ERR_07	✓
I2C_ERR_08	✓
I2C_ERR_09	✓
I2C_ERR_10	✓
I2C_ERR_13	✓
LFXT_ERR_03	✓
LFXT_ERR_04	✓
PMCU_ERR_13	✓
RST_ERR_01	✓
SPI_ERR_04	✓
SPI_ERR_05	✓
SPI_ERR_06	✓

Errata Number	Rev A
SPI_ERR_07	✓
SWD_ERR_01	✓
SYSCTL_ERR_01	✓
SYSCTL_ERR_02	✓
SYSCTL_ERR_03	✓
SYSOSC_ERR_02	✓
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.

null: Fully-qualified development-support product.

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

"Developmental product is intended for internal evaluation purposes."

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

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

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

5.1 Device Symbolization and Revision Identification

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

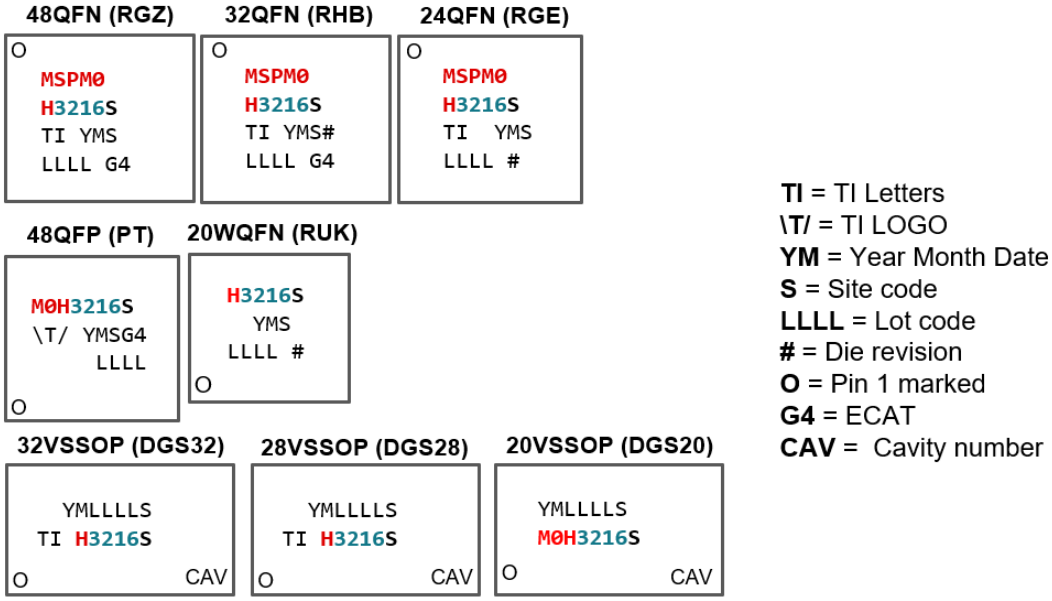

Figure 5-1. Package Symbolization

Table 5-1. Die Revisions

Revision Letter	Version (in the device factory constants memory)
A	1

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

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.

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

FLASH_ERR_02 *FLASH Module*

Category

Functional

Function

Debug disable in NONMAIN can be re-enabled using the password

FLASH_ERR_02 (continued)

FLASH Module

Description

If debug is disabled in NONMAIN configuration (DEBUGACCESS = 0x5566), the device can still be accessed using the password that is programmed (default password if not explicitly programmed).

Workaround

Workaround 1:
Set the DEBUGACCESS to Debug Enabled with Password option (DEBUGACCESS = 0xCCDD) and provide a unique password in the PWDDEBUGLOCK field. For higher security, it is recommended to have device-unique passwords that are cryptographically random. This would allow debug access with the right 128-bit password but can still allow some debug commands and access to the CFG-AP and SEC-AP.

Workaround 2:
Disable the physical SW Debug Port entirely by disabling SWDP_MODE. This fully prevents any debug access or requests to the device, but may affect Failure Analysis and return flows.

FLASH_ERR_04

FLASH Module

Category

Functional

Function

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

Description

Workaround

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

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

FLASH_ERR_05

FLASH Module

Category

Functional

Function

DEDERRADDR can have incorrect reset value

Description

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

FLASH_ERR_05 (continued) — ***FLASH Module***

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

Function

Hard fault isn't generated for typical invalid memory region

Description

Hard fault isn't generated while trying to access illegal memory address space as shown below: 1. 0x010053FF - 0x20000000 2. 0x40BFFFFF - 0x41C00000 3. 0x41C007FF - 0x41C40000

Workaround

No

GPIO_ERR_04 — ***GPIO Module***

Category

Functional

Function

Configuring global fastwake prevents a GPIO pin from sending data to the DIN register.

Description

When configuring the fastwake-only bit of the CTL register and forcing data to a GPIO pin in run mode, the device will wake up, but the data on the GPIO pin will not appear in the DIN register. This is because the CTL register configuration prevents any data from flowing from the GPIO pin to the DIN register.

Workaround

Avoid using the GPIO fastwake-only function when expecting data on a GPIO pin entering the DIN register.

GPIO_ERR_08 — ***GPIO Module***

Category

Functional

Function

GPIOs can trigger a fast wake(regardless of whether they were set) when the device is in low-power mode

Description

When the device enters low-power mode, the GPIOs can still register a fast wake, regardless of the GPIO FASTWAKE register configuration or the pin's configuration. The two cases where this error applies are the following:
Case 1: When a pin-specific fast wake is enabled via the GPIO FASTWAKE register (e.g.,

GPIO_ERR_08 (continued)

GPIO Module

PA2), any toggle will generate a fast wake regardless of the configured function's state. Case 2: If using any communication peripheral on any pin, glitches on the line filtered by the peripheral may trigger a fast wake.

Workaround

For case 1: Don't enable the GPIO FASTWAKE bit in that specific pin (e.g., PA2). Ex: DL_GPIO_disableFastWakePins(GPIOA, DL_GPIO_PIN_2); // Disables Specific GPIO FASTWAKE for PA2
For case 2: Don't enable global fast wake for any pins. Ex: DL_GPIO_disableGlobalFastWake(GPIO_X); // Disables Global fast wake
For a general workaround, turn off the Asynchronous fast clock requests by setting the BLOCKASYNCALL in the SYSOSCCFG register in SYSCTL. Ex: SYSCTL->SOCLOCK.SYSOSCCFG |= SYSCTL_SYSOSCCFG_BLOCKASYNCALL_MASK;

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 will indefinitely pull SCL low.

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

Workaround

If the I2C target application does not require data reception in low power mode using Async fast clock request, it is recommended to disable SWUEN by default, including during reset or power cycle. In this case, bug description 1 and 2 will 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 I2C controller will not get reset. There is a situation where SDA is 0 and I2C controller has gone into IDLE state, here

I2C_ERR_05 (continued) — *I2C Module*

the I2C controller won't be able to move forward from the IDLE state or update the SDA value. I2C 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 I2C 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 I2C START bit to SCL low is 60us, which is >50us. It will trigger the timeout event and let I2C Controller go into IDLE without completing the transaction at the start of transfer. Below is detailed explanation.

For SCL is configured as 20KHz, SCL low and high period is 30us and 20us respectively. First, I2C START bit write at the same time high timeout counter starts decrementing. Then, it takes one SCL low period (30us) from the START 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 can not be 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
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. Any I2C transfer can always be terminated with a STOP condition to provide 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

I2C_ERR_13
(continued)

I2C Module

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

LFXT_ERR_03

LFXT Module

Category

Functional

Function

PA26/27 can't be controlled through IOMUX when PA3/4 are selected as LFXT pins

Description

In the DGS28, DGS32, RGE, RHB, RGZ and PT packages, LFXIN/LFXOUT is just supported on PA3/4, when LFXT is enabled, PA26/27 can't be controlled through IOMUX for I/O operation.

Workaround

If you need to use LFXT function, please avoid using PA26/27 as I/O operation.

LFXT_ERR_04

LFXT Module

Category

Functional

Function

Power consumption is abnormal when LFXT is enabled

Description

In DGS28, DGS32, RGE, RHB, RGZ and PT packages, which support LFXT function on PA3/4, if LFXT is enabled and PA26 is left floating, there can be high leakage 5mA from VDD pin.

Workaround

In DGS28, DGS32, RGE, RHB, RGZ and PT packages, which support LFXT function on PA3/4, force PA26 connecting with GND or VDD externally on the board.

PMCU_ERR_13

PMCU Module

Category

Functional

Function

MCU may get stuck while waking up from STOP2 & STANDBY0

PMCU_ERR_13
(continued)

PMCU Module

Description

If prefetch access is pending when the device transitions to STOP2 or STANDBY, when the device wakes up, the pending prefetch can prevent the device from resuming normal execution. The errata occurs if the WFI instruction is not word aligned, and the flash wait state is 2. In such a case, neither a DMA transfer nor a pending interrupt will be serviced.

Workaround

User should disable prefetch and issue a shutdown store memory read, which prevents a new prefetch from issuing and allows a pending prefetch to complete.

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 there can be 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.

SPI_ERR_04

SPI Module

Category

Functional

Function

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

Description

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

Workaround

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

SPI_ERR_05

SPI Module

Category

Functional

Function

SPI Peripheral Receive Timeout interrupt is setting irrespective of RXFIFO data

SPI_ERR_05
(continued)

SPI Module

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.

SWD_ERR_01

SWD Module

Category

Functional

Function

Device drawing more current at SWDCLK pin

SWD_ERR_01
(continued)

SWD Module

Description

For devices which doesn't support internal pull down in IO structure, the SWCLK pin remains in a floating state and draw more current.

Workaround

In initialization SW code, override the SWCLK IOMUX configuration to set the PIPU bit in PINCM register to 1, to enable the pull-up functionality on the SWCLK pin, or switch this pin to GPIO / other function through PF bit in PINCM register.
OR
Add external pull-down resistor near SWCLK pin on the board if customer also wants to fix the floating node current during boot code execution or when the NRST pin is applied.

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

SYSCTL_ERR_03 (continued) ***SYSCTL Module***

contradicts the Technical Reference Manual (TRM), which specifies its initial reset value as zero.

Workaround

No workaround

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. Async request is received from some peripherals which use MFCLK as functional clock. On receiving async request, SYSOSC gets enabled and ulpclk becomes 32MHz. But MFCLK is gated off and it does not toggle at all as the device is still set to the LPM.

Workaround

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

TIMER_ERR_04 ***TIMER Module***

Category

Functional

Function

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

Description

When using a TIMER in one shot mode, TIMER re-enable may be missed if done close to zero event. The HW update to the timer enable bit will take a single functional clock cycle. For example, if the timer's clock source is 32.768kHz and clock divider of 3, then it will take ~100us to have the enable bit set to 0 properly.

Workaround

Wait 1 functional clock cycle before re-enabling the timer OR the timer can be disabled first before re-enabling.

Disable the counter with CTRCTL.EN = 0, then re-enable with CTRCTL.EN = 1

TIMER_ERR_06 ***TIMA and TIMG Module***

Category

Functional

TIMER_ERR_06 (continued)

TIMA and TIMG Module

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 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

UART_ERR_02 (continued) — ***UART Module***

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

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

UART_ERR_06 (continued)

UART Module

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.

UART_ERR_08 (continued) — *UART Module*

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 Trademarks

All trademarks are the property of their respective owners.

8 Revision History

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

Changes from July 1, 2025 to January 31, 2026 (from Revision * (July 2025) to Revision A (January 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