Errata

MSPM0G511x, MSPM0G5187, MSPM0G511x-Q1 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 (Prototype X-Marked Products)
AES_ERR_01	✓
CPU_ERR_02	✓
CPU_ERR_03	✓
FLASH_ERR_03	✓
FLASH_ERR_04	✓
FLASH_ERR_05	✓
FLASH_ERR_08	✓
GPIO_ERR_05	✓
GPIO_ERR_06	✓
KEYSTORE_ERR_01	✓
PMCU_ERR_13	✓
RST_ERR_01	✓
SYSCTL_ERR_01	✓
SYSCTL_ERR_02	✓
SYSCTL_ERR_03	✓
SYSCTL_ERR_04	✓
SYSOSC_ERR_01	✓
SYSOSC_ERR_02	✓
SYSPLL_ERR_01	✓

Errata Number	Rev B (Prototype X-Marked Products)
TIMER_ERR_04	✓
TIMER_ERR_06	✓
TIMER_ERR_07	✓
UNICOMMI2CC_ERR_01	✓
UNICOMMI2CT_ERR_01	✓
UNICOMMI2CT_ERR_02	✓
UNICOMMI2CT_ERR_03	✓
UNICOMMSPI_ERR_01	✓
UNICOMMUART_ERR_01	✓
UNICOMMUART_ERR_02	✓
UNICOMMUART_ERR_03	✓
UNICOMMUART_ERR_04	✓
UNICOMMUART_ERR_05	✓
UNICOMMUART_ERR_06	✓
UNICOMMUART_ERR_07	✓
UNICOMMUART_ERR_08	✓
UNICOMMUART_ERR_09	✓
UNICOMMUART_ERR_10	✓
UNICOMMUART_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 pre-prod version. After device release, RTM version will be added here. Table 5-1 defines the device revision to version ID mapping.

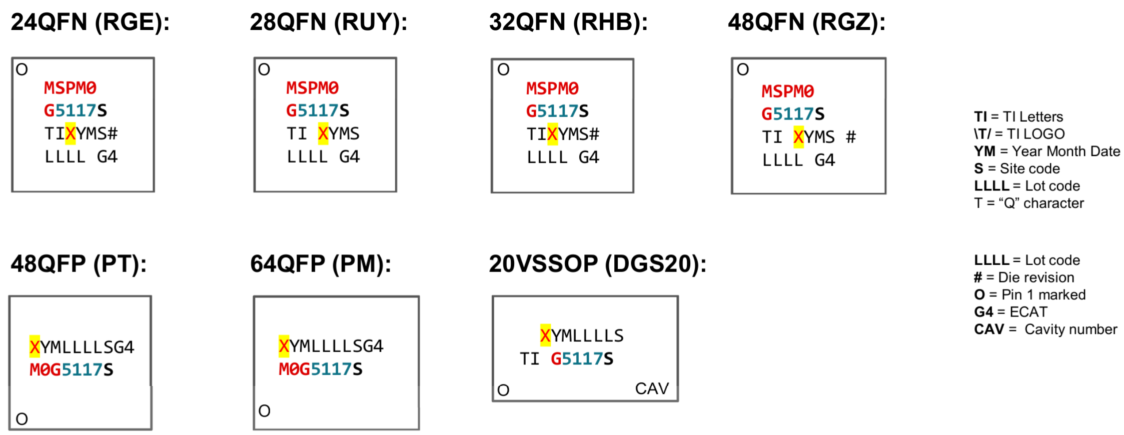

Figure 5-1. Package Symbolization

Table 5-1. Die Revisions

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

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.

CPU_ERR_02 — *CPU Module*

Category

Functional

Function

Limitation of disabling prefetch for CPUSS

Description

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

Workaround

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

After the memory access completes the prefetcher will be disabled.
Example:

```
CPUSS.CTL.PREFETCH = 0x0; //disables prefetcher
SYSCTL.SOCLOCK.SHUTDNSTORE0; // memory access to shutdown memory
```

CPU_ERR_03 — *CPU Module*

Category

Functional

Function

Prefetcher can fetch wrong instructions when transitioning into Low power modes

Description

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

CPU_ERR_03 (continued) ***CPU Module***

Workaround

Disable prefetcher before entering low power modes.

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

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

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 in the NONMAIN or Factory region

Description

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

Workaround

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

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

FLASH_ERR_05 ***FLASH Module***

Category

Functional

FLASH_ERR_05 (continued) — *FLASH Module*

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

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_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 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 DOUTCLK 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);
}
```

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_13 — *PMCU Module*

Category

Functional

Function

MCU may get stuck while waking up from STOP2 & STANDBY0

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

Workaround

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

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

SYSCTL_ERR_02
(continued)

SYSCTL Module

Function

SYSSTATUS.FLASHSEC is non-zero after a BOOTRST

Description

After BOOTRST/ bootcode completion SYSSTATUS.FLASHSEC is non-zero. This the customer will see after bootcode completion.

Workaround

No

SYSCTL_ERR_03 *SYSCTL Module*

Category

Functional

Function

DEDERRADDR persists after a SYSRESET or a write to the SYSSTATUSCLR

Details

DEDERRADDR persists after either a SYSRESET or a write to the SYSSTATUSCLR register. Its value is overwritten only when a new FLASHDED error occurs. This behavior contradicts the Technical Reference Manual (TRM), which specifies its initial reset value as zero.

Workaround

No workaround

SYSCTL_ERR_04 *SYSCTL Module*

Category

Functional

Function

SYSSTATUS.FLASHSEC is not cleared after a SYSRESET

Description

SYSSTATUS.FLASHSEC is not cleared after a SYSRESET and is only cleared by writing to the SYSSTATUSCLR register.

Workaround

No

SYSOSC_ERR_01 *SYSOSC Module*

Category

Functional

Function

MFCLK drift when using SYSOSC FCL together with STOP1 mode

Description

If MFCLK is enabled AND SYSOSC is using the frequency correction loop (FCL) mode AND STOP1 low power operating mode is used, then the MFCLK may drift by two cycles when SYSOSC shifts from 4MHz back to 32MHz (either upon exit from STOP1 to RUN mode or upon an asynchronous fast clock request that forces SYSOSC to 32MHz).

Workaround

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

OR

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

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.

SYSPLL_ERR_01 *SYSPLL Module*

Category

Functional

Function

SYSPLL Frequency may not lock to correct frequency when enabled.

Description

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

Workaround

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

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

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

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

Workaround 2: Output SYSOSC/2 from the CLK_OUT pin and route the signal into FCC_IN. Use the SYSPLLCLK0 as the FCC CLK and the FCC_IN for the trigger

SYSPLL_ERR_01 (continued)

SYSPLL Module

source. Run the FCC for 16 Clock cycles, and check the value for the configured SYSPLL frequency with reference to the SYSOSC; for example, with SYSPLL = 80MHz and SYSOSC/2 = 16MHz, the resultant FCC count should be 80,000,000/16,000,000 * 16 = ~80. The count will vary depending on the combined clock accuracies, so it is recommended to add a +-5% to allowed range. Estimated time for FCC is 1us.

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

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

TIMER_ERR_04 — *TIMER Module*

Category

Functional

Function

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

Description

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

Workaround

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

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

TIMER_ERR_06 — *TIMG Module*

Category

Functional

Function

Writing 0 to CLKEN bit does not disable counter

Description

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

Workaround

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

TIMER_ERR_07 — *TIMG Module*

Category

Functional

TIMER_ERR_07 (continued)

TIMG Module

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.

UNICOMMI2CC_ERR_01

UNICOMMI2CC Module

Category

Functional

Function

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

Description

After setting the BUSRTRUN/FRAME_START bit to initiate an I2C controller transfer, it takes approximately 2 I2C functional clock cycles for the BUSY status to be asserted. If polling for the BUSY bit is used immediately after setting BUSRTRUN/FRAME_START 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 I2C functional clock = 3 x clock_divider x (CPU_CLK / selected clock source frequency) For example, with a clock_divider of 8, a clock source of 4 MHz(MFCLK), and CPU_CLK of 32 MHz: Software delay = 3 x 8 x (32 MHz / 4 MHz)= 192 CPU cycles

UNICOMMI2CT_ERR_01

UNICOMMI2CT Module

Category

Functional

Function

I2C target will keep in IDLE when CTR is written separately

Description

I2C target will keep in IDLE in below scenario: 1. I2C uses MFCLK as the clock source. 2. CLKDIV=0. 3. Set CTR.START, CTR.STOP or CTR.BLEN bits in first register write. 4. Set CTR.FRM_START in the second register write.

UNICOMMI2CT_ERR_01 (continued) ***UNICOMMI2CT Module***

Workaround

When I2C uses MFCLK as the clock source and CLKDIV=0, suggest to set CTR.START, CTR.STOP, CTR.BLEN or CTR.FRM_START bits in one register write.

UNICOMMI2CT_ERR_02 *UNICOMMI2CT Module*

Category

Functional

Function

CPU reads RXFIFO failure when I2C clock much lower than CPU clock

Description

In UNICOMMI2CT RX mode, the RXDONE interrupt flag asserts immediately after the last bit of the received frame. However, the RXFIFO buffer content updates exhibit a 2-clock-cycle latency relative to RXDONE assertion. If the CPU initiates an RXFIFO read operation more than 2 UNICOMMI2CT clock cycles after RXDONE flag triggering, it may access stale FIFO data, leading to a data integrity violation. This will be a problem especially when UNICOMMI2CT clock is much lower than CPU clock.

Workaround

Add a fix delay before reading the RXFIFO.

UNICOMMI2CT_ERR_03 *UNICOMMI2CT Module*

Category

Functional

Function

TSTART flag gets set before the SR.ADDRMATCH in 7 bit mode

Description

On the UNICOMMI2CT bus, the RIS.TSTART flag is set three bus clock cycles before the SR.ADDRMATCH status updates. However, if the interval between the assertion of TSTART and the CPU's read operation of ADDRMATCH exceeds two UNICOMMI2CT clock cycles, the CPU may read a stale ADDRMATCH value, potentially causing an error.

Workaround

UNICOMMSPI_ERR_01 *UNICOMMSPI Module*

Category

Functional

Function

SPI underflow event may not generate if read/write to TXFIFO happen at the same time

Description

In UNICOMMSPI-TXFIFO, underflow event will not get generated during a critical window when the bellow conditions happen together: 1. CPU/DMA write data into TXFIFO 2. SPI peripheral read TXFIFO to send data Not suggest to use underflow event function. User

UNICOMMSPI_ERR_01 (continued)

UNICOMMSPI Module

must ensure that the TXFIFO of UNICOMMSPI can never be empty when CPU/DMA access the TXFIFO.

Workaround

No

UNICOMMUART_ERR_01

UNICOMMUART Module

Category

Functional

Function

Data integrity issues in case glitches are introduced in IrDA mode

Description

UNICOMM-UART supports IrDA but has limitations in noisy environments. Due to the lack of a glitch filter, data integrity issues can occur in IrDA mode when glitches appear on the data line. Since IrDA relies on edge detection logic, these glitches are misinterpreted as valid pulses, leading to unexpected data in the FIFO and issues with LTOUT computation.

Workaround

No

UNICOMMUART_ERR_02

UNICOMMUART Module

Category

Functional

Function

IDLE period detection issue with glitch in IDLELINE mode

Description

A glitch during the idle period can corrupt the receiver's idle detection, causing the Rx side to drop the subsequent address byte. Crucially, this failure mode is timing-dependent: only glitches inserted at a specific time relative to the idle period trigger the issue. Not every glitch within the idle period causes corruption.

Workaround

No

UNICOMMUART_ERR_03

UNICOMMUART Module

Category

Functional

Function

Data Integrity issues with glitch in Manchester mode

Description

Glitches in the Manchester-encoded data stream corrupt the signal's precise edge timing, leading to erroneous received data and data integrity issues.

UNICOMMUART_ERR_03 (continued) — *UNICOMMUART Module*

Workaround

No

UNICOMMUART_ERR_04 — *UNICOMMUART Module*

Category

Functional

Function

Data will be missed with glitches during break field and/or sync field in LIN Mode

Description

Break Field Scenario (High-Pulse Glitch): During the Break Field period, a negative edge will reset the counter to zero, cause break field detection failure and result in data loss. Sync Field Scenario: A glitch during Sync Field will trigger false LINC0/LINC1 interrupts, reset the counter on Rx-line negative edges, cause sync field validation failure and result in data loss.

Workaround

No

UNICOMMUART_ERR_05 — *UNICOMMUART Module*

Category

Functional

Function

Glitches lead to LTOUT/RTOUT period computation issues when the line is idle

Description

In the normal UART mode, a glitch during idle line conditions disrupt LTOUT timing by shifting the LTOUT event 1-2 baud clocks.

Workaround

No

UNICOMMUART_ERR_06 — *UNICOMMUART Module*

Category

Functional

Function

Inconsistent STOP bit length causing RTOUT/LTOUT period computation issues

Description

On receiver side the function state machine is designed in such a way that the state goes from stop bit to idle at the mid of the stop bit, this causes RTOUT counter to start counting before it should actually have started. This leads to incorrect RTOUT period and RTOUT interrupt comes earlier than expected.

Workaround

Add compensation with a half stop bit period to the RTOUT counter

UNICOMMUART_ERR_07 — *UNICOMMUART Module*

Category

Functional

Function

RTS line does not go HIGH if UART is disabled in RS-232 mode

Description

When UART is disabled, the RTS line fails to return to its idle state (HIGH), remaining stuck at LOW.

Workaround

Use software to enable the internal pull-up resistor and set the RTS line IO to Hiz mode.

UNICOMMUART_ERR_08 — *UNICOMMUART Module*

Category

Functional

Function

LTOUT Interrupt will keep on coming after every time-out expiry

Description

After one LTOUT interrupt comes, the counter restarts and keeps on counting and gives another LTOUT interrupt even without any Rx frame in between. This would lead to continuous LTOUT interrupts at every time out expiry interval. The behavior is not similar to RTOUT as RTOUT interrupt comes only once until a new Rx transaction takes place. This limitation is arising as the same counter is being used for both RTOUT and LTOUT events.

Workaround

UNICOMMUART_ERR_09 — *UNICOMMUART Module*

Category

Functional

Function

ISO-7816 Smartcard Mode can't support 9600 baud rate with lower than 57MHz UARTCLK

Description

To achieve a 9600 baud rate in ISO-7816 Smartcard Mode, and considering the limitations below, a UARTCLK frequency exceeding 57 MHz is required. 1. The ISO-7816 standard dictates that 1 bit requires 372 clock cycles. 2. In MSPM0 ISO-7816 mode, the oversampling rate (OVS) is fixed in the UART peripheral at 16x. Here is the minimum UARTCLK calculation: Required UARTCLK = 9600 * 372 * 16 = 57.139 MHz

Workaround

No

UNICOMMUART_ERR_10	*UNICOMMUART Module*
Category	Functional
Function	LIN Registers are accessible only if CLKDIV is set to /1
Description	In the UNICOMMUART (UART+LIN) variant, LIN register configurations remain valid only when CLKDIV is set to 1. Any CLKDIV value other than 1 causes these settings to be discarded.
Workaround	No

UNICOMMUART_ERR_11	*UNICOMMUART Module*
Category	Functional
Function	STAT.BUSY bit stays high when UNICOMMUART is disabled and data is available in txfifo
Description	STAT.BUSY bit stays high when UNICOMMUART power is disabled and data is available in txfifo.
Workaround	Reset UNICOMMUART to initialize all the UNICOMMUART registers.

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.

DATE	REVISION	NOTES
November 2025	*	Initial Release