

Abstract

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

Table of Contents

1 Functional Advisories	1
2 Preprogrammed Software Advisories	2
3 Debug Only Advisories	2
4 Fixed by Compiler Advisories	2
5 Device Nomenclature	2
5.1 Device Symbolization and Revision Identification	3
6 Advisory Descriptions	4
7 Trademarks	18
8 Revision History	18

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
AES_ERR_01	✓
CPU_ERR_05	✓
FLASH_ERR_02	✓
GPIO_ERR_05	✓
GSC_ERR_01	✓
IOMUX_ERR_01	✓
I2S_ERR_01	✓
KEYSTORE_ERR_01	✓
PMCU_ERR_15	✓
RST_ERR_01	✓
RST_ERR_02	✓
SYSCTL_ERR_01	✓
SYSPLL_ERR_01	✓
TIMER_ERR_04	✓
TIMER_ERR_06	✓
TIMER_ERR_07	✓
TIMER_ERR_08	✓
UNICOMMI2CC_ERR_01	✓
UNICOMMI2CC_ERR_02	✓
UNICOMMI2CC_ERR_03	✓
UNICOMMSPI_ERR_02	✓
UNICOMMSPI_ERR_03	✓

Errata Number	Rev A
UNICOMMSPI_ERR_04	✓
UNICOMMUART_ERR_01	✓
UNICOMMUART_ERR_02	✓
UNICOMMUART_ERR_03	✓
UNICOMMUART_ERR_04	✓
UNICOMMUART_ERR_05	✓
UNICOMMUART_ERR_06	✓
UNICOMMUART_ERR_07	✓
UNICOMMUART_ERR_10	✓
UNICOMMUART_ERR_12	✓
UNICOMMUART_ERR_13	✓
UNICOMMUART_ERR_14	✓
UNICOMMUART_ERR_15	✓

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, and [Table 5-1](#) defines the device revision to version ID mapping.

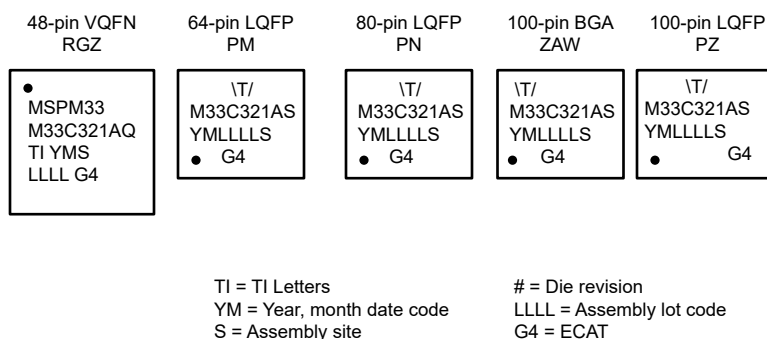


Figure 5-1. Package Symbolization

Table 5-1. Die Revisions

Revision Letter	Version (in the device factory constants memory)
A	0

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_05 ***CPU Module***

Category

Functional

Function

MTB_BASE register value gets read improperly

Description

When Reading the MTB_BASE register the MTB_SRAM base location will be right shifted by a bit. For example, if the MTB_SRAM shows 0x80806000 the actual MTB_SRAM base location is 0x40403000.

Workaround

When reading the MTB_BASE register to find the MTB_SRAM base location right shift the read value by one bit. Please use the code below to work around the issue: `uint32_t mtb_base = (*(uint32_t*)(0xE004300C) >> 1);`

FLASH_ERR_02 ***FLASH Module***

Category

Functional

Function

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

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:

FLASH_ERR_02

(continued)

FLASH Module

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.

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).

GSC_ERR_01

GSC Module

Category

Functional

Function

Certain timers have the same security attribute

Description

In the PPC_SECATTRIB_TIMER register the SEC_TIMA0_0 and SEC_TIMA0_1 registers are tied together such that the security attribute for both TIMERA0_0 and TIMERA0_1 must be the same. TIMERG4_2 and TIMERG4_1 also have the same configuration but only when the DMA is attempting to access the timers.

Workaround

Try to keep both TIMERA0_0 and TIMERA0_1 as the same security level. If that's not possible use a different pair of timer instances. For TIMERG4_2 and TIMERG4_1 keep them at the same security level especially if using a DMA to access the TIMERG4_2 or TIMERG4_1.

IOMUX_ERR_01

IOMUX Module

Category

Functional

IOMUX_ERR_01

(continued)

IOMUX Module**Function**

TDO pin drives a low signal after reset.

Description

The TDO pin out of reset is driven as a low signal. For other pins they are in a Hi-Z state out of reset.

Workaround

If you need this pin to be a Hi-Z state set both the PC and PF bits to zero. Otherwise connect the peripheral you plan to use on this IO.

I2S_ERR_01***I2S Module*****Category**

Functional

Function

The first TXIFG interrupt does not get fired after I2S/TDM module enabled

Description

Upon I2S/TDM module enable, the first TXIFG interrupt does not get fired.

Workaround

Write a 1 to the INTCTL.INTERVAL bit to generate the initial TXIFG interrupt.

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_15***PMCU Module*****Category**

Functional

Function

GENCLKEN Clock related configurations before SYSPLL is enabled are ignored

Description

When configuring the SYSPLL, configure the GENCLKEN register after setting the SYSPLLEN bit in the HSCLKEN register. If the EXTDIVCAN, MCLKEXTDIVEN and I2SPLLCLKDIVEN, I2SPLLCLKDIVCFG bits in the GENCLKEN register are programmed

PMCU_ERR_15

(continued)

PMCU Module

before the SYSPLLEN bit is set, the configuration in the GENCLKEN register is not properly configured and the EXTDIVCAN and I2SPLLCLKDIVCFG configurations are ignored.

Workaround

Configure the EXTDIVCAN, MCLKEXTDIVEN and I2SPLLCLKDIVEN, I2SPLLCLKDIVCFG bits in the GENCLKEN registers, after the SYSPLLEN bit is set.

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

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.

RST_ERR_02

(continued)

RST Module**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. Note that the required pull-up min resistance depends on the IO drive strength of the external reset controller (if applicable). A known compliant configuration is a 4.7 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).

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

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**Frequency Verification Process**

Monitor the SYSPLL frequency output using the Frequency Clock Counter (FCC) whenever the SYSPLLEN bit is set to 1. Once the correct frequency is established, it will remain stable until the SYSPLL is disabled and re-enabled (SYSPLLEN bit toggled from 0 to 1). If an incorrect frequency is detected, disable and re-enable the SYSPLL to perform another verification.

Workaround 1: FCC Count Check

Use LFCLK as the FCC trigger source to count the SYSPLL output clock frequency. Execute the FCC and verify the measured value against the configured SYSPLL frequency using LFCLK as reference.

SYSPLL_ERR_01

(continued)

SYSPLL Module

Example calculation:

- $\text{SYSPLLCLK0} = 160\text{MHz}$; $\text{LFCLK} = 32.768\text{kHz}$
- Measured FCC Count = $160,000,000/32,768 = 4,882$

FCC Count Tolerance:

The real FCC count will vary depending on the combined clock accuracies (SYSPLLCLK0 and LFCLK). Recommend to add +/- 5~10% to allowed FCC check range.

- FCC count upper limit = $4,882 \times 1.05 = 5,126$
- FCC count lower limit = $4,882 \times 0.95 = 4,638$

Timing considerations:

- Clock synchronization time: 5-6 LFCLK cycles
- FCC trigger time: 1-32 LFCLK cycles (user-configurable)

Register Configuration:

- FCC Settings: `SYSCTL.GENCLKCFG.FCCTRIGSRC = 1;`
- `SYSCTL.GENCLKCFG.FCCLVLTRIG = 0;`
- `SYSCTL.GENCLKCFG.FCCTRIGCNT = 0;`
- `SYSCTL.GENCLKCFG.FCCSELCLK = 4;`
- Start FCC: `SYSCTL.FCCCMD = 0x0E000001U`
- Check FCC Done Status: `SYSCTL.CLKSTATUS.FCCDONE`
- Read FCC Count: `SYSCTL.FCC`

Timeout Protection:

Implement a software-based timeout during FCC Done status monitoring to prevent longer wait time under the condition the unlocked SYSPLL frequency is less than FCC trigger clock frequency (LFCLK).

```
fccTimeOutCounter = 0;
while (DL_SYSCTL_isFCCDone() == 0) {
    delay_cycles(977); /* 1x LFCLK cycle = 32MHz/32.768kHz */
    fccTimeOutCounter++;
    if(fccTimeOutCounter > 65) break;
    /* Timeout set to approximately 2ms (user-customizable) */
}
```

FCC Check Restart:

If the FCC measurement falls outside the expected range, disable and re-enable the SYSPLL (set `SYSPLLEN` to 0, then 1) and repeat the FCC verification.

Workaround 2: FCC Ratio Check

Use LFCLK as the FCC trigger source to count both SYSPLL output and input clock frequency. Execute the FCC and verify the measured ratio of the FCC check value between output and input clock to the expected ratio.

Example calculation:

- $\text{SYSPLL} = 160\text{MHz}$; $\text{HFCLK} = 40\text{MHz}$; $\text{LFCLK} = 32.768\text{kHz}$
- Expect clock ratio = $160\text{MHz}/40\text{MHz} = 4.0000$
- Measured FCC count (SYSPLL) = $160,000,000/32,768 = 4,882$
- Measured FCC count (HFCLK) = $40,000,000/32,768 = 1,220$
- Measured clock ratio = $4,882/1,220 = 4.0016$

FCC Ratio Tolerance:

The FCC ratio method eliminates combined clock accuracy errors and depends only on FCC uncertainty (2 counted clock cycles) and calculation rounding error. This allows for much tighter tolerance ranges compared to FCC count check method, for example +/- 0.3%.

Timing considerations:

- Clock synchronization time: 5-6 LFCLK cycles

SYSPLL_ERR_01

(continued)

SYSPLL Module

- FCC trigger time: 1-32 LFCLK cycles (user-configurable)
- Total time per complete FCC ratio check: 2*(sync time + trigger time)

FCC Ratio Check Flow:

1. Configure FCC for SYSPLL output clock (SYSPLL0 or SYSPLL2X)
2. Start FCC and wait for FCC done (Add Timeout Protection, refer to FCC Count Check)
3. Read FCC check count back
4. Configure FCC for SYSPLL input clock (SYSOSC or HFCLK)
5. Start FCC and wait for FCC done (Add Timeout Protection, refer to FCC Count Check)
6. Read FCC check count back
7. Calculate the FCC check ratio and compared to the expected ratio range
8. If the FCC ratio falls outside the expected range, disable and re-enable the SYSPLL (set SYSPLLEN to 0, then 1) and repeat the FCC ratio verification.

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**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**Initial repeat counter has 1 less period than next repeats Module****Category**

Functional

TIMER_ERR_07

(continued)

Initial repeat counter has 1 less period than next repeats Module

Function

TIMER

Description

When using the timer repeat counter mode, the first repeat will have 1 less count than the subsequent repeats because the following repeat counters will include the transition between 0 and the load value. For example if the TIMx.RCLD = 0x3 then 3 observable zero events would appear on the first repeat counter and 4 observable zero events would appear on the following repeat counter sequences.

Workaround

Set the initial RCLD value to 1 more than the expected RCLD, then in the ISR for the Repeat Counter Zero Event (REPC), set the RCLD to the intended RCLD value. For example, if intending to have 4 repeats, set the initial RCLD value to RCLD = 0x5, then in the timer ISR for the REPC interrupt, set RCLD = 0x4. Now all timer repeats will have the same number of zero/load events.

TIMER_ERR_08

TIMx Module

Category

Functional

Function

Sequential event generation for advanced timers GEN_EVENT1 output is ignored

Description

In the advanced timer's (TIMA) the GEN_EVENT1 only works for one event and will not properly propagate through the event handler after the first event is triggered.

Workaround

Do not use GEN_EVENT1 for repeated event outputs on advanced timers. Either use GEN_EVENT0 or use a general-purpose timer (TIMG) instance instead.

UNICOMMI2CC_ERR_01

UNICOMMI2CC Module

Category

Functional

Function

I2C Controller BUSY Status Polling Issue

Description

When initiating an I2C controller transfer by setting the BUSRTRUN/FRAME_START bit, the BUSY status flag requires approximately 2-3 I2C functional clock cycles to assert. If polling the BUSY bit immediately after setting BUSRTRUN/FRAME_START, the status might be checked by the application before it's properly set. This issue is more pronounced with higher CLKDIV values (resulting in slower I2C functional clock) or under higher compiler optimization levels.

Workaround

Add a software delay before polling BUSY status. The recommended delay should be: 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 2, a clock source of 4

UNICOMMI2CC_E**RR_01 (continued) UNICOMMI2CC Module**

MHz(MFCLK), and CPU_CLK of 80 MHz: Software delay = $3 \times 2 \times (80 \text{ MHz} / 4 \text{ MHz}) = 120$ CPU cycles

UNICOMMI2CC_E**RR_02 UNICOMMI2CC Module**

Category

Functional

Function

RXDONE interrupt is triggered twice when I2C controller ACKOEN bit is enabled

Description

When I2C controller CTR.ACKONE bit is enable, software controls the sending of ACK/NACK by writing CTR.ACK bit 0 or 1. This can be done in the RXDONE interrupt before the stop condition. After writing the CTR.ACK bit, there will be valid ACK/NACK signal on I2C bus, and an additional RXDONE interrupt will be triggered in the subsequent I2C stop status.

Workaround

Disable the RXDONE interrupt before writing CTR.ACK bit, and manually enable it before next transmission.

UNICOMMI2CC_E**RR_03 UNICOMMI2CC Module**

Category

Functional

Function

I2C controller SCL frequency is lower when clock stretching function is enable

Description

- Please do never recommend to use an ETW feature as a workaround. - Please check that the ERRATA proposal does not compromise other existing ERRATAs on this device revision or previous revisions. When clock stretching is enabled, CR.CLKSTRETCH = 1, SCL frequency will be lower than the settings. For example, I2C controller bus speed is set to 400kHz, if clock stretching function is enabled, I2C controller bus clock will be lower than 400kHz, for example, 360kHz or 390kHz. Same phenomenon will be seen on other bus speed, for example, 100kHz or 1Mhz.

Workaround

Not recommended to disable the clock stretching function, because there is no large impact to I2C communication function. After disable the clock stretching function, expected I2C bus clock speed will be seen.

UNICOMMSPI_ER**R_02 UNICOMMSPI Module**

Category

Functional

UNICOMMSPI_ER

R_02 (continued)

UNICOMMSPI Module

Function

Controller Side Busy status read incorrectly

Description

SPI busy status doesn't depend on the TXFIFO's data elements, so essentially once we load the SPI TXFIFO with some data, the busy bit will not go high instantly in case the CLKSEL is a slower clock (LFCLK for instance) or CLKDIV > /1. This leads to incorrect status being read during the time between loading TXFIFO data and until the communication actually starts since the SPI source clock is slower.

Workaround

Can introduce a check for TXFIFO empty STATUS bit when writing application code.

UNICOMMSPI_ER

R_03

UNICOMMSPI Module

Category

Functional

Function

RTOUT won't be set at second time when CPU is used to clear RXFIFO

Description

After SPI communication is completed with rxtimeout enabled and then RTOUT is set, if use CPU to read out the RXFIFO data and read IIDX to clear the interrupt but don't do any reset or disable the IP, the RTOUT interrupt won't be set when you do the same transmission at the second time even though rxtimeout counter is "0" and RXFIFO is not empty.

Workaround

Use DMA to read out the RXFIFO data instead of using CPU directly .Or reset the UNICOMM SPI after reading out the RXFIFO data using CPU directly.

UNICOMMSPI_ER

R_04

UNICOMMSPI Module

Category

Functional

Function

RX stored data is incorrect when do re-transfer after last one transfer with DSS mismatch

Description

When sending data with different data packet size setting (DSS) on controller & peripheral side and after that doing RXCLR to clear the FIFO, again sending data with proper data package size but still older data is coming into RXFIFO when first SCLK pulse comes.

Workaround

Please make sure the data package size setting (DSS) on controller side and peripheral side is same.

UNICOMMUART_E
RR_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_E
RR_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_E
RR_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.

Workaround

No

UNICOMMUART_E
RR_04 *UNICOMMUART Module*

Category

Functional

UNICOMMUART_E

RR_04 (continued) *UNICOMMUART Module*

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_E

RR_05 *UNICOMMUART Module*

Category Functional

Function Issue due to UART Glitch Filter not being present

Description 1.IrDA Mode Data Integrity: 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. 2.Idleline detection: During IDLELINE detection, a signal glitch can disrupt the detection process by resetting the internal idle line counter to zero. This interference prevents the IDLELINE status flag from being properly set, causing the module to fail to recognize a valid idle condition on the communication line. However, if the idle line condition persists without additional glitches after this disruption, the detection mechanism will eventually recover and properly detect the idle state. 3.Manchester Mode Data Integrity: When signal glitches occur during the data sampling period of Manchester-encoded transmissions, the receiver captures incorrect data, leading to misinterpreted bits. These sampling errors result in corrupted data values being stored in the receive buffer, causing data integrity failures in the received message. Glitches occurring outside the sampling window typically do not affect data interpretation. 4.LIN mode sensitivity around BREAK / SYNC field: a.BREAK Field: During LIN communication, if signal glitches occur specifically during clock sampling periods of the BREAK FIELD, the detection mechanism may prematurely terminate. However, the system demonstrates resilience - if the BREAK FIELD condition continues without additional glitches after the disruption, the detection logic will recover and successfully identify the BREAK FIELD. b.SYNC Field: The system exhibits comparable vulnerability during SYNC field calibration, where glitches can interfere with both negative and positive edge (RXNE & RXPE) detection. 5.LTOUT/RTOUT Detection issue with glitches: When signal glitches occur during idle line detection conditions, they temporarily disrupt the LTOUT/RTOUT timing calculations, and the timeout counters are cleared back to 0. If the line remains free from additional glitches after the disruption, LTOUT/RTOUT conditions are detected after the configured time-period.

Workaround No workaround is available.

UNICOMMUART_E**RR_06** *UNICOMMUART Module***Category**

Functional

Function

RTOUT/LTOUT computation issues due to STOP bit handling

Description

On the receiver side, the function state machine transitions from STOP bit to IDLE at the middle of the STOP bit. This causes the receive timeout (RTOUT) & line timeout (LTOUT) counter to start counting at the middle of the STOP bit and not at its end. Leading to RTOUT/LTOUT being triggered half a baud-period early. This is especially pronounced for low-baud rates with higher UART functional clock frequency.

Workaround

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

UNICOMMUART_E**RR_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_E**RR_10** *UNICOMMUART Module***Category**

Functional

Function

LIN Registers CLKDIV Restriction

Description

When CLKDIV value is other than 0, writing into LINC0/1 will not have any effect.

Workaround

To properly configure LIN registers: 1. First set CLKDIV to '0' 2. Update the LINC0/1 register configurations with desired values 3. Restore CLKDIV to its intended operating value

UNICOMMUART_E**RR_12** *UNICOMMUART Module***Category**

Functional

UNICOMMUART_E

RR_12 (continued) *UNICOMMUART Module*

Function CTL0.TXE bit setting to 0 in middle of frame makes current transmission error

Description Start transmitting packet and then in middle of transfer, set the CTL0.TXE bit to 0, the current packet won't complete and the TX line will be pulled low, which shows the busy behavior as BUSY. The TX line will be released only when CTL0.TXE = 1, and till then BUSY remains 1.

Workaround User must poll for BUSY STATUS to go low and CTL0.TXE = 0, so as to avoid this issue.

UNICOMMUART_E

RR_13 *UNICOMMUART Module*

Category Functional

Function RTOUT interrupt coming 1 fclk cycle earlier

Description RTOUT (Receive timeout) interrupt coming 1 fclk cycle earlier from the exact time which it should come for.

Workaround The way user can take care of this is to expect the RTOUT interrupt to come one fclk cycle earlier.

UNICOMMUART_E

RR_14 *UNICOMMUART Module*

Category Functional

Function CTS interrupt gets toggled on both falling and rising edges of CTS line

Description CTS interrupt gets toggled on both RISE edge as well as FALL edge of CTS line. Ideally, it should just toggle on the FALL edge of CTS line.

Workaround Instead of RIS bit, user can monitor STAT.CTS bit's toggle to 1, since this toggling of STAT.CTS = 1 indicates that falling of CTS line has been encountered.

UNICOMMUART_E

RR_15 *UNICOMMUART Module*

Category Functional

Function LCRH.SENDIDLE access missed in between fclk and new IDLELINE frame not produced

UNICOMMUART_E

RR_15 (continued) *UNICOMMUART Module*

Description

If toggle LCRH.SENDIDLE from 0 to 1 and then back from 1 to 0, or toggle from 1 to 0 and then back from 0 to 1 in between two fclk edges, the toggle will be missed and no new IDLELINE frame is sent.

Workaround

After toggling LCRH.SENDIDLE, don't put it back immediately, please keep more than one fclk period, because LCRH.SENDIDLE is sampled on fclk.

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 June 30, 2026 to September 16, 2026 (from Revision B (June 2026) to Revision C (September 2026))

	Page
• I2S_ERR_01 Advisory was added.....	6
• RST_ERR_01 Workaround was updated.....	7
• RST_ERR_02 Advisory was added	7
• UNICOMMI2CC_ERR_02 Advisory was added.....	12
• UNICOMMI2CC_ERR_03 Advisory was added.....	12
• UNICOMMSPI_ERR_02 Advisory was added.....	12
• UNICOMMSPI_ERR_03 Advisory was added.....	13
• UNICOMMSPI_ERR_04 Advisory was added.....	13
• UNICOMMUART_ERR_12 Advisory was added.....	16
• UNICOMMUART_ERR_13 Advisory was added.....	17
• UNICOMMUART_ERR_14 Advisory was added.....	17
• UNICOMMUART_ERR_15 Advisory was added.....	17

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.

Copyright © 2026, Texas Instruments Incorporated

Last updated 10/2025