Errata
MSP430F4618 Microcontroller

TEXAS INSTRUMENTS

ABSTRACT

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

Table of Contents

1 Functional Advisories

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

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

Errata Number	Rev G	Rev F	Rev E
ADC18	✓	✓	✓
ADC25	✓	✓	✓
DMA3	✓	✓	✓
DMA4	✓	✓	✓
FLL3	✓	✓	✓
FLL6	✓	✓	✓
LCDA5	✓	✓	✓
LCDA7	✓	✓	✓
RTC1	✓	✓	✓
TA12	✓	✓	✓
TA16	✓	✓	✓
TA18	✓	✓	✓
TA21	✓	✓	✓
TAB22	✓	✓	✓
TB2	✓	✓	✓
TB16	✓	✓	✓
TB18	✓	✓	✓
TB24	✓	✓	✓
USCI16			✓
USCI19	✓	✓	✓
USCI20	✓	✓	✓
USCI21	✓	✓	✓
USCI22	✓	✓	✓
USCI23	✓	✓	✓
USCI24	✓	✓	✓
USCI25	✓	✓	✓
USCI26	✓	✓	✓
USCI27	✓	✓	✓
USCI30	✓	✓	✓
USCI34	✓	✓	✓
USCI35	✓	✓	✓
USCI40	✓	✓	✓
WDG2	✓	✓	✓
XOSC5	✓	✓	✓
XOSC8	✓	✓	✓
XOSC9	✓	✓	✓

2 Preprogrammed Software Advisories

Advisories that affect factory-programmed software.

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

The device does not have any errata for this category.

3 Debug Only Advisories

Advisories that affect only debug operation.

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

The device does not have any errata for this category.

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.

Errata Number	Rev G	Rev F	Rev E
CPU8	✓	✓	✓
CPU16	✓	✓	✓
CPU19	✓	✓	✓

Refer to the following MSP430 compiler documentation for more details about the CPU bugs workarounds.

TI MSP430 Compiler Tools (Code Composer Studio IDE)

- MSP430 Optimizing C/C++ Compiler: Check the --silicon_errata option
- MSP430 Assembly Language Tools

MSP430 GNU Compiler (MSP430-GCC)

- MSP430 GCC Options: Check -msilicon-errata= and -msilicon-errata-warn= options
- MSP430 GCC User's Guide

IAR Embedded Workbench

- IAR workarounds for msp430 hardware issues

5 Nomenclature, Package Symbolization, and Revision Identification

The revision of the device can be identified by the revision letter on the Package Markings or by the HW_ID located inside the TLV structure of the device.

5.1 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.

5.2 Package Markings

ZQW113   *BGA (ZQW), 113 Pin*


```
          ᵀⁱ
      M430Fxxxx
      NNNNNNN #
         G1
○
```
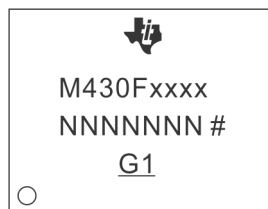
\# = Die revision
○ = Pin 1 location
N = Lot trace code

PZ100   *LQFP (PZ) 100 Pin*

```
┌─────────────────┐
│  ti  NNNNNNN    │     #   = Die revision
│                 │     ○   = Pin 1 location
│  M430Fxxxx      │     N   = Lot trace code
│                 │
│  REV #          │
│                 │
│ ○               │
└─────────────────┘

┌─────────────────┐
│  ti  NNNNNNNG4  │     #   = Die revision
│                 │     ○   = Pin 1 location
│  M430Fxxxx      │     N   = Lot trace code
│                 │
│  Rev #          │
│                 │
│ ○               │
└─────────────────┘

┌─────────────────┐
│  ti  NNNNNNNG4  │     #   = Die revision
│  MSP430™        │     ○   = Pin 1 location
│                 │     N   = Lot trace code
│  Fxxxx          │
│  Rev #          │
│ ○               │
└─────────────────┘
```

NOTE: Package marking with "TM" applies only to devices released after 2011.

5.3 Memory-Mapped Hardware Revision (TLV Structure)

This device does not support reading the hardware revision from memory.

Further guidance on how to locate the TLV structure and read out the HW_ID can be found in the device User's Guide.

6 Advisory Descriptions

ADC18	***ADC Module***
Category	Functional
Function	Incorrect conversion result in extended sample mode
Description	The ADC12 conversion result can be incorrect if the extended sample mode is selected (SHP = 0), the conversion clock is not the internal ADC12 oscillator (ADC12SSEL > 0), and one of the following two conditions is true: - The extended sample input signal SHI is asynchronous to the clock source used for ADC12CLK and the undivided ADC12 input clock frequency exceeds 3.15 MHz. or - The extended sample input signal SHI is synchronous to the clock source used for ADC12CLK and the undivided ADC12 input clock frequency exceeds 6.3 MHz.
Workaround	- Use the pulse sample mode (SHP = 1). or - Use the ADC12 internal oscillator as the ADC12 clock source. or - Limit the undivided ADC12 input clock frequency to 3.15 MHz. or - Use the same clock source (such as ACLK or SMCLK) to derive both SHI and ADC12CLK, to achieve synchronous operation, and also limit the undivided ADC12 input clock frequency to 6.3 MHz.

ADC25	***ADC Module***
Category	Functional
Function	Write to ADC12CTL0 triggers ADC12 when CONSEQ = 00
Description	If ADC conversions are triggered by the Timer_B module and the ADC12 is in single-channel single-conversion mode (CONSEQ = 00), ADC sampling is enabled by write access to any bit(s) in the ADC12CTL0 register. This is contrary to the expected behavior that only the ADC12 enable conversion bit (ADC12ENC) triggers a new ADC12 sample.
Workaround	When operating the ADC12 in CONSEQ=00 and a Timer_B output is selected as the sample and hold source, temporarily clear the ADC12ENC bit before writing to other bits in the ADC12CTL0 register. The following capture trigger can then be re-enabled by setting ADC12ENC = 1.

CPU8	***CPU Module***
Category	Compiler-Fixed
Function	Using odd values in the SP register
Description	If the stack pointer (SP) is written with an odd value then the first time that the SP is used, the LSB of the SP is forced to zero.
Workaround	Do not use odd values with the SP. Refer to the table below for compiler-specific fix implementation information.

IDE/Compiler	Version Number	Notes
IAR Embedded Workbench	Not affected	
TI MSP430 Compiler Tools (Code Composer Studio)	Not affected	
MSP430 GNU Compiler (MSP430-GCC)	MSP430-GCC 4.9 build 389 or later	User is required to add the compiler flag option below. -msilicon-errata=cpu8 -msilicon-errata-warn=cpu8 generates a warning in addition
MSP430 GNU Compiler (MSP430-GCC)	MSP430-GCC 5.x build 14 or later	User is required to add the compiler flag option below. -msilicon-errata=cpu8 -msilicon-errata-warn=cpu8 generates a warning in addition

CPU16

CPU Module

Category Compiler-Fixed

Function Indexed addressing with instructions calla, mova and bra.

Description With indexed addressing mode and instructions calla, mova, and bra, it is not possible to reach memory above 64k if the register content is < 64k.

Example: Assume R5 = FFFEh. The instruction calla 0004h(R5) will result in a 20-bit call of address 0002h instead of 10002h.

Workaround - Use different addressing mode to reach memory above 64k.
- First use adda [index],[Rx] to calculate address in upper memory and then do a calla [Rx]

Refer to the table below for compiler-specific fix implementation information.

IDE/Compiler	Version Number	Notes
IAR Embedded Workbench	IAR EW430 v6.30.1 or later	
TI MSP430 Compiler Tools (Code Composer Studio)	Fix not available	
MSP430 GNU Compiler (MSP430-GCC)	Fix not available	

CPU19

CPU Module

Category Compiler-Fixed

Function CPUOFF modification may result in unintentional register read

Description If an instruction that modifies the CPUOFF bit in the Status Register is followed by an instruction with an indirect addressed operand (e.g. MOV @R8, R9, RET, POP, POPM), an unintentional register read operation can occur during the wakeup of the CPU. If the unintentional read occurs to a read sensitive register (e.g. UCB0RXBUF, TAIV), which changes its value or the value of other registers (IFG's), the bug leads to lost interrupts or wrong register read values.

Workaround   Insert a NOP instruction after each CPUOFF instruction.

OR

Refer to the table below for compiler-specific fix implementation information.
Note that compilers implementing the fix may lead to double stack usage when RET/
RETA follows the compiler-inserted NOP.

IDE/Compiler	Version Number	Notes
IAR Embedded Workbench	IAR EW430 v6.20.1 until v6.40	User is required to add the compiler or assembler flag option below. --hw_workaround=nop_after_lpm
IAR Embedded Workbench	IAR EW430 v6.40 or later	Workaround is automatically enabled
TI MSP430 Compiler Tools (Code Composer Studio)	15.12.0.LTS	User is required to add the compiler or assembler flag option below. --silicon_errata=CPU19
MSP430 GNU Compiler (MSP430-GCC)	MSP430-GCC 4.9 build 389 or later	User is required to add the compiler or assembler flag option below. -msilicon-errata=cpu19 -msilicon-errata-warn=cpu19 generates a warning in addition
MSP430 GNU Compiler (MSP430-GCC)	MSP430-GCC 5.x build 14 or later	User is required to add the compiler or assembler flag option below. -msilicon-errata=cpu19 -msilicon-errata-warn=cpu19 generates a warning in addition

DMA3

DMA Module

Category   Functional

Function   Read-modify-write instructions may corrupt DMA address registers

Description   When a 16-bit wide read-modify-write instruction (such as add.w and sub.w) is directly used on a DMA address register (DMAxSA or DMAxDA), the register contents will get corrupted.

Workaround   1. Do not use 16-bit wide read-modify-write instructions on DMA address registers. Instead, in case address calculations are necessary, do the calculations first, and then assign the result to the DMA address registers.
OR
2. Use 20-bit wide read-modify-write instructions (such as addx.a, subx.a) on the DMA address registers if needed.

DMA4

DMA Module

Category   Functional

Function   Corrupted write access to 20-bit DMA registers

Description   When a 20-bit wide write to a DMA address register (DMAxSA or DMAxDA) is interrupted by a DMA transfer, the register contents may be unpredictable.

Workaround	1. Design the application to guarantee that no DMA access interrupts 20-bit wide accesses to the DMA address registers. OR 2. When accessing the DMA address registers, enable the Read Modify Write disable bit (DMARMWDIS = 1) or temporarily disable all active DMA channels (DMAEN = 0). OR 3. Use word access for accessing the DMA address registers. Note that this limits the values that can be written to the address registers to 16-bit values (lower 64K of Flash).

FLL3 — *FLL Module*

Category	Functional
Function	FLLDx = 11 for /8 may generate an unstable MCLK frequency
Description	When setting the FLL to higher frequencies using FLLDx = 11 (/8) the output frequency of the FLL may have a larger frequency variation (e.g. averaged over 2sec) as well as a lower average output frequency than expected when compared to the other FLLDx bit settings.
Workaround	None

FLL6 — *FLL Module*

Category	Functional
Function	LFXT1DIG bit is read incorrectly
Description	The LFXT1DIG bit always reads as '0' even when the bit is programmed to '1'. This affects only the readout of the bit and not the clock bypass implementation which functions as expected.
Workaround	None

LCDA5 — *LCDA Module*

Category	Functional
Function	Wrong cycle time for first cycle of COMx/Sx signals
Description	The time of the first cycle of COMx/Sx signals after enabling the LCD_A module is only half of the selected value. All following cycles are correct
Workaround	Not required, because it does not influence the LCD function.

LCDA7 — *LCDA Module*

Category	Functional
Function	Higher current consumption when using shared LCD ports as fast toggling outputs

Description	If a shared LCD pin (segment or com line) is used as digital fast toggling output (f>10kHz) and the VLCD is >0V (BG enabled) the device current consumption increases with higher toggling frequencies.
Workaround	1. Do not use shared LCD pins as fast toggling outputs if an LCD is used. 2. Reduce the toggle frequency of the shared pin to <10kHz.

RTC1 — *RTC Module*

Category	Functional
Function	Incorrect RTCDAY count in BCD mode
Description	When using the RTC in BCD mode, RTCDAY will count from 0x29 to 0x31 instead of counting to 0x30 in the month of December (RTCMON=0x12). Furthermore, due to a malfunction in the leap year detection logic, RTCMON/RTCDAY may incorrectly count from 0x02/0x28 to 0x03/0x01 instead of 0x02/0x29, or it may incorrectly count from 0x02/0x28 to 0x02/0x29 instead of 0x03/0x01.
Workaround	Do not operate the RTC module in BCD mode. Use the RTC in hexadecimal format mode (RTCBCD = 0) instead. Convert RTC registers to BCD on demand using software. NOTE: The CPU instruction DADD.B/.W can be used to efficiently implement a hex to BCD conversion. An Assembly language example of such an optimized 8-bit conversion is shown below: mov.b #8,R14 // Loop counter, process 8 bits clr.b R12 // Result will get assembled in R12 loop rlc.b R13 // Get MSB from input variable in R13 dadd.b R12,R12 dec.b R14 jnz loop

TA12 — *TA Module*

Category	Functional
Function	Interrupt is lost (slow ACLK)
Description	Timer_A counter is running with slow clock (external TACLK or ACLK)compared to MCLK. The compare mode is selected for the capture/compare channel and the CCRx register is incremented by one with the occurring compare interrupt (if TAR = CCRx). Due to the fast MCLK the CCRx register increment (CCRx = CCRx+1) happens before the Timer_A counter has incremented again. Therefore the next compare interrupt should happen at once with the next Timer_A counter increment (if TAR = CCRx + 1). This interrupt gets lost.
Workaround	Switch capture/compare mode to capture mode before the CCRx register increment. Switch back to compare mode afterwards.

TA16 — *TA Module*

Category	Functional
Function	First increment of TAR erroneous when IDx > 00
Description	The first increment of TAR after any timer clear event (POR/TACLR) happens immediately following the first positive edge of the selected clock source (INCLK, SMCLK, ACLK or

TACLK). This is independent of the clock input divider settings (ID0, ID1). All following TAR increments are performed correctly with the selected IDx settings.

Workaround   None

TA18	*TA Module*

Category   Functional

Function   MOV to TACTL may clear TAR

Description   When TACTL is modified with a MOV instruction, the contents of TAR may be cleared, even when TACLR is not set.

Workaround   Use BIS or BIC instructions to modify TACTL.

Note

A DMA transfer must not occur while these BIS and BIC instructions execute. This can be prevented by disabling the DMA prior to these instructions, or by using the DMAONFETCH bit to align DMA transfers to instruction fetch boundaries.

TA21	*TA Module*

Category   Functional

Function   TAIFG Flag is erroneously set after Timer A restarts in Up Mode

Description   In Up Mode, the TAIFG flag should only be set when the timer counts from TACCR0 to zero. However, if the Timer A is stopped at TAR = TACCR0, then cleared (TAR=0) by setting the TACLR bit, and finally restarted in Up Mode, the next rising edge of the TACLK will erroneously set the TAIFG flag.
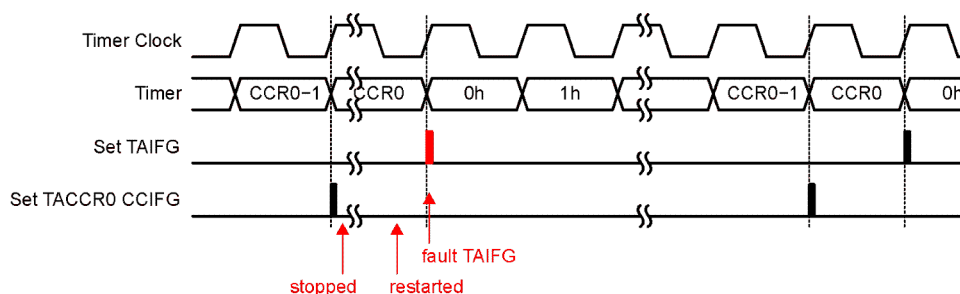
Workaround   None.

TAB22	*TAB Module*

Category   Functional

Function   Timer_A/Timer_B register modification after Watchdog Timer PUC

Description   Unwanted modification of the Timer_A/Timer_B registers TACTL/TBCTL and TAIV/TBIV can occur when a PUC is generated by the Watchdog Timer(WDT) in Watchdog mode and any Timer_A/Timer_B counter register TACCRx/TBCCRx is incremented/decremented (Timer_A/Timer_B does not need to be running).

Workaround	Initialize TACTL/TBCTL register after the reset occurs using a MOV instruction (BIS/BIC may not fully initialize the register). TAIV/TBIV is automatically cleared following this initialization.

Example code:

MOV.W #VAL, &TACTL
or
MOV.W #VAL, &TBCTL

Where, VAL=0, if Timer is not used in application otherwise, user defined per desired function.

TB2	***TB Module***
Category	Functional
Function	Interrupt is lost (slow ACLK)
Description	Timer_B counter is running with slow clock (external TBCLK or ACLK) compared to MCLK. The compare mode is selected for the capture/compare channel and the CCRx register is incremented by 1 with the occurring compare interrupt (if TBR = CCRx). Due to the fast MCLK, the CCRx register increment (CCRx = CCRx + 1) happens before the Timer_B counter has incremented again. Therefore, the next compare interrupt should happen at once with the next Timer_B counter increment (if TBR = CCRx + 1). This interrupt is lost.
Workaround	Switch capture/compare mode to capture mode before the CCRx register increment. Switch back to compare mode afterward.

TB16	***TB Module***
Category	Functional
Function	First increment of TBR erroneous when IDx > 00
Description	The first increment of TBR after any timer clear event (POR/TBCLR) happens immediately following the first positive edge of the selected clock source (INCLK, SMCLK, ACLK, or TBCLK). This is independent of the clock input divider settings (ID0, ID1). All following TBR increments are performed correctly with the selected IDx settings.
Workaround	None

TB18	***TB Module***
Category	Functional
Function	MOV to TBCTL may clear TBR
Description	When TBCTL is modified with a MOV instruction, the contents of TBR may be cleared, even when TBCLR is not set.
Workaround	Use BIS or BIC instructions to modify TBCTL.

Note

A DMA transfer must not occur while these BIS and BIC instructions execute.
This can be prevented by disabling the DMA prior to these instructions or
by using the DMAONFETCH bit to align DMA transfers to instruction fetch
boundaries.

TB24 — *TB Module*

Category	Functional
Function	TBIFG Flag is erroneously set after Timer B restarts in Up Mode
Description	In Up Mode, the TBIFG flag should only be set when the timer resets from TBCCR0 to zero. However, if the Timer B is stopped at TBR = TBCCR0, then cleared (TBR=0) by setting the TBCLR bit, and finally restarted in Up Mode, the next rising edge of the TBCLK will erroneously set the TBIFG flag.
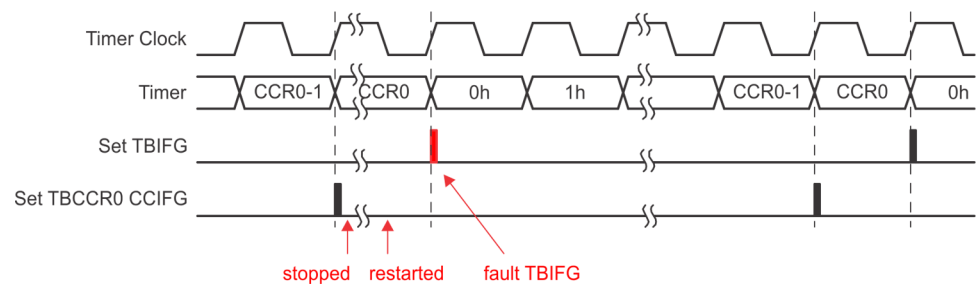
Workaround	None.

USCI16 — *USCI Module*

Category	Functional
Function	UART/IrDA Mode Lost Characters
Description	When configured for UART/IrDA mode, the USCI baud rate generator may halt operation under the following conditions: 1 - IrDA mode: repeated invalid start bits on the receive line or 2 - UART/IrDA modes: positive pulse on the receive line during break character reception inside the stop bit time slot (the second stop bit time slot in case of UCSPB=1) with a pulse width that passes the deglitch filter but is shorter than half a bit time. After halting, additional characters will be ignored. Transmit functionality is not affected.
Workaround	Check the UCBUSY flag status periodically in software. If the flag is set and no character has been received in the expected time, reset the USCI module in software. To reset the USCI module, toggle UCSWRST and re-enable the USCI interrupts.

USCI19 — *USCI Module*

Category	Functional
Function	LPM4 may affect USCI operation

Description	When SMCLK is used as the USCI clock source, and SMCLK gets deactivated due to a LPM4 entry, ongoing SPI master, I2C master, and UART transmit transaction will be interrupted. Also, while in LPM4, UART receive operation is non-functional.
Workaround	Do not enter LPM4 while SPI master, I2C master, or UART transmit operations are active. Wait for the operation to be completed prior entering LPM4, or enter a different low-power mode instead. Also, do not use LPM4 in case of UART receive operation. Instead, use a different low-power mode.

USCI20	***USCI Module***
Category	Functional
Function	I2C Mode Multi-master transmitter issue
Description	When configured for I2C master-transmitter mode, and used in a multi-master environment, the USCI module can cause unpredictable bus behavior if all of the following four conditions are true: 1 - Two masters are generating SCL And 2 - The slave is stretching the SCL low phase of an ACK period while outputting NACK on SDA And 3 - The slave drives ACK on SDA after the USCI has already released SCL, and then the SCL bus line gets released And 4 - The transmit buffer has not been loaded before the other master continues communication by driving SCL low The USCI will remain in the SCL high phase until the transmit buffer is written. After the transmit buffer has been written, the USCI will interfere with the current bus activity and may cause unpredictable bus behavior.
Workaround	1 - Ensure that slave doesn't stretch the SCL low phase of an ACK period Or 2 - Ensure that the transmit buffer is loaded in time Or 3 - Do not use the multi-master transmitter mode

USCI21	***USCI Module***
Category	Functional
Function	UART IrDA receive filter
Description	The IrDA receive filter can be used to filter pulses with length UCAIRRXFL configured in UCAxIRRCTL register. If UCIRRXFE is set the IrDA receive decoder may filter out pulses longer than the configured filter length depending on frequency of BRCLK. This is resulting in framing errors or corrupted data on the receiver side.
Workaround	Depending on the used baud rate and the configured filter length a maximum frequency for BRCLK needs to be set to avoid this issue: For baud rates equal and higher than 115.000 the maximum allowed BRCLK frequency is equal to the max specified system frequency.

$$\text{Max BRCLK} = \frac{\text{Filter Length} + 64}{2} \times \frac{\text{Baud Rate} \times 16}{3 \times 10^6}$$

Baud Rate	Filter Length UCIRRXFL (dec)	Max BRCLK (MHz)
9600	64	3.28
	32	2.46
	16	2.05
	8	1.84
	4	1.74
	2	1.69
	1	1.66
	0	1.64
19200	64	6.55
	32	4.92
	16	4.1
	8	3.69
	4	3.48
	2	3.38
	1	3.33
	0	3.28
38400	64	13.11
	32	9.83
	16	8.19
	8	7.37
	4	6.96
	2	6.76
	1	6.66
	0	6.55
56000	64	19.11
	32	14.34
	16	11.95
	8	10.75
	4	10.15
	2	9.86
	1	9.71
	0	9.56

USCI22	***USCI Module***
Category	Functional
Function	I2C Master Receiver with 10-bit slave addressing
Description	Unexpected behavior of the USCI_B can occur when configured in I2C master receive mode with 10-bit slave addressing under the following conditions: 1) The USCI sends first byte of slave address, the slave sends an ACK and when second address byte is sent, the slave sends a NACK. 2) Master sends a repeat start condition (If UCTXSTT=1). 3) The first address byte following the repeated start is acknowledged. However, the second address byte is not sent, instead the Master incorrectly starts to receive data and sets UCBxRXIFG=1.
Workaround	Do not use repeated start condition instead set the stop condition UCTXSTP=1 in the NACK ISR prior to the following start condition (USTXSTT=1).

USCI23	*USCI Module*
Category	Functional
Function	UART transmit mode with automatic baud rate detection
Description	Erroneous behavior of the USCI_A can occur when configured in UART transmit mode with automatic baud rate detection. During transmission if a "Transmit break" is initiated (UCTXBRK=1), the USCI_A will not deliver a stop bit of logic high, instead, it will send a logic low during the subsequent synch period.
Workaround	1) Follow User's Guide instructions for transmitting a break/synch field following UCSWRST=1. Or, 2) Set UCTXBRK=1 before an active transmission, i.e. check for bit UCBUSY=0 and then set UCTXBRK=1.

USCI24	*USCI Module*
Category	Functional
Function	Incorrect baud rate information during UART automatic baud rate detection mode
Description	Erroneous behavior of the USCI_A can occur when configured in UART mode with automatic baud rate detection. After automatic baud rate measurement is complete, the UART updates UCAxBR0 and UCAxBR1. Under Oversampling mode (UCOS16=1), for baud rates that should result in UCAxBRx=0x0002, the UART incorrectly reports it as UCAxBRx=0x5555.
Workaround	When break/synch is detected following the automatic baud rate detection, the flag UCBRK flag is set to 1. Check if UCAxBRx=0x5555 and correct it to 0x0002.

USCI25	*USCI Module*
Category	Functional
Function	TXIFG is not reset when NACK is received in I2C mode
Description	When the USCI_B module is configured as an I2C master transmitter the TXIFG is not reset after a NACK is received if the master is configured to send a restart (UCTXSTT=1 & UCTXSTP=0).
Workaround	Reset TXIFG in software within the NACKIFG interrupt service routine

USCI26	*USCI Module*
Category	Functional
Function	Tbuf parameter violation in I2C multi-master mode
Description	In multi-master I2C systems the timing parameter Tbuf (bus free time between a stop condition and the following start) is not guaranteed to match the I2C specification of 4.7us in standard mode and 1.3us in fast mode. If the UCTXSTT bit is set during a running I2C transaction, the USCI module waits and issues the start condition on bus release causing the violation to occur. Note: It is recommended to check if UCBBUSY bit is cleared before setting UCTXSTT=1.
Workaround	None

USCI27	*USCI Module*

Category Functional

Function Timing of USCI I2C interrupts may cause device reset due to automatic clear of an IFG.

Description When certain USCI I2C interrupt flags (IFG) are set and an automatic flag-clearing event on the I2C bus occurs, the program counter may become corrupted. This will only happen when the IFG is cleared within a critical time window (~6 CPU clock cycles) after a USCI interrupt request occurs and before the interrupt servicing is initiated. The affected interrupts are UCBxTXIFG, UCSTPIFG, UCSTTIFG and UCNACKIFG.

The automatic flag-clearing scenarios are described in the following situations:
(1) A pending UCBxTXIFG interrupt request is cleared on the falling SCL clock edge following a NACK.
(2) A pending UCSTPIFG, UCSTTIFG, or UCNACKIFG interrupt request is cleared by a following Start condition.

Workaround (1) Polling the affected flags instead of enabling the interrupts.
or
(2) Ensuring the above mentioned flag-clearing events occur after a time delay of 6 CPU clock cycles has elapsed since the interrupt request occurred and was accepted.

USCI30	*USCI Module*

Category Functional

Function I2C mode master receiver / slave receiver

Description When the USCI I2C module is configured as a receiver (master or slave), it performs a double-buffered receive operation. In a transaction of two bytes, once the first byte is moved from the receive shift register to the receive buffer the byte is acknowledged and the state machine allows the reception of the next byte.

If the receive buffer has not been cleared of its contents by reading the UCBxRXBUF register while the 7th bit of the following data byte is being received, an error condition may occur on the I2C bus. Depending on the USCI configuration the following may occur:

1) If the USCI is configured as an I2C master receiver, an unintentional repeated start condition can be triggered or the master switches into an idle state (I2C communication aborted). The reception of the current data byte is not successful in this case.
2) If the USCI is configured as I2C slave receiver, the slave can switch to an idle state stalling I2C communication. The reception of the current data byte is not successful in this case. The USCI I2C state machine will notify the master of the aborted reception with a NACK.

Note that the error condition described above occurs only within a limited window of the 7th bit of the current byte being received. If the receive buffer is read outside of this window (before or after), then the error condition will not occur.

Workaround a) The error condition can be avoided altogether by servicing the UCBxRXIFG in a timely manner. This can be done by (a) servicing the interrupt and ensuring UCBxRXBUF is read promptly or (b) Using the DMA to automatically read bytes from receive buffer upon UCBxRXIFG being set.

OR

b) In case the receive buffer cannot be read out in time, test the I2C clock line before the

UCBxRXBUF is read out to ensure that the critical window has elapsed. This is done by checking if the clock line low status indicator bit UCSCLLOW is set for atleast three USCI bit clock cycles i.e. 3 X t(BitClock).

Note that the last byte of the transaction must be read directly from UCBxRXBUF. For all other bytes follow the workaround:

Code flow for workaround

(1) Enter RX ISR for reading receiving bytes
(2) Check if UCSCLLOW.UCBxSTAT == 1
(3) If no, repeat step 2 until set
(4) If yes, repeat step 2 for a time period > 3 x t (BitClock) where t (BitClock) = 1/ f (BitClock)
(5) If window of 3 x t(BitClock) cycles has elapsed, it is safe to read UCBxRXBUF

USCI34	*USCI Module*
Category	Functional
Function	I2C multi-master transmit may lose first few bytes.
Description	In an I2C multi-master system (UCMM =1), under the following conditions:

(1)the master is configured as a transmitter (UCTR =1)

AND

(2)the start bit is set (UCTXSTT =1);

if the I2C bus is unavailable, then the USCI module enters an idle state where it waits and checks for bus release. While in the idle state it is possible that the USCI master updates its TXIFG based on clock line activity due to other master/slave communication on the bus. The data byte(s) loaded in TXBUF while in idle state are lost and transmit pointers initialized by the user in the transmit ISR are updated incorrectly. |
| **Workaround** | Verify that the START condition has been sent (UCTXSTT =0) before loading TXBUF with data.

Example:
```
#pragma vector = USCIAB0TX_VECTOR
__interrupt void USCIAB0TX_ISR(void)
{
// Workaround for USCI34
if(UCB0CTL1&UCTXSTT)
{
// TXData = pointer to the transmit buffer start
// PTxData = pointer to transmit in the ISR
PTxData = TXData; // restore the transmit buffer pointer if the Start bit is set
}
//
if(IFG2&UCB0TXIFG)
{
if (PTxData < = PTxDataEnd) // Check TX byte counter
{
UCB0TXBUF = *PTxData++; // Load TX buffer
}
``` |

```
else
{
UCB0CTL1 |= UCTXSTP; // I2C stop condition
IFG2 &= ~UCB0TXIFG; // Clear USCI_B0 TX int flag
__bic_SR_register_on_exit(CPUOFF); // Exit LPM0
}
}
}
```

USCI35

USCI Module

Category	Functional
Function	Violation of setup and hold times for (repeated) start in I2C master mode
Description	In I2C master mode, the setup and hold times for a (repeated) START, $t_{SU,STA}$ and $t_{HD,STA}$ respectively, can be violated if SCL clock frequency is greater than 50kHz in standard mode (100kbps). As a result, a slave can receive incorrect data or the I2C bus can be stalled due to clock stretching by the slave.
Workaround	If using repeated start, ensure SCL clock frequencies is < 50kHz in I2C standard mode (100 kbps).

USCI40

USCI Module

Category	Functional
Function	SPI Slave Transmit with clock phase select = 1
Description	In SPI slave mode with clock phase select set to 1 (UCAxCTLW0.UCCKPH=1), after the first TX byte, all following bytes are shifted by one bit with shift direction dependent on UCMSB. This is due to the internal shift register getting pre-loaded asynchronously when writing to the USCIA TXBUF register. TX data in the internal buffer is shifted by one bit after the RX data is received.
Workaround	Reinitialize TXBUF before using SPI and after each transmission. If transmit data needs to be repeated with the next transmission, then write back previously read value:

```
UCAxTXBUF = UCAxTXBUF;
```

WDG2

WDG Module

Category	Functional
Function	Incorrectly accessing a flash control register
Description	If a key violation is caused by incorrectly accessing a flash control register, the watchdog interrupt flag is set in addition to the expected PUC.
Workaround	None

XOSC5

XOSC Module

Category	Functional

Function	LF crystal failures may not be properly detected by the oscillator fault circuitry
Description	The oscillator fault error detection of the LFXT1 oscillator in low frequency mode (XTS = 0) may not work reliably causing a failing crystal to go undetected by the CPU, i.e. OFIFG will not be set.
Workaround	None

XOSC8 — *XOSC Module*

Category	Functional
Function	ACLK failure when crystal ESR is below 40 kOhm.
Description	When ACLK is sourced by a low frequency crystal with an ESR below 40 kOhm, the duty cycle of ACLK may fall below the specification; the OFIFG may become set or in some instances, ACLK may stop completely.
Workaround	Please refer to "XOSC8 Guidance" found at SLAA423 for information regarding working with this erratum.

XOSC9 — *XOSC Module*

Category	Functional
Function	XT1 Oscillator may not function as expected in HF mode
Description	XT1 oscillator does not work correctly in high frequency mode at supply voltages below 2.0V with crystal frequency > 4MHz.
Workaround	None. When XT1 oscillator is used in HF mode with crystal frequency > 4MHz ensure a supply voltage > 2.2V.

7 Revision History

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

Changes from May 12, 2021 to May 17, 2021	Page
• Changed the document format and structure; updated the numbering format for tables, figures, and cross references throughout the document..6	

IMPORTANT NOTICE AND DISCLAIMER