

# MSP430F478 Device Erratasheet

---



---



---

## 1 Functional Errata Revision History

Errata impacting device's operation, function or parametrics.

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

Errata Number	Rev B
FLASH19	✓
FLASH24	✓
FLASH27	✓
FLL3	✓
FLL6	✓
LCDA5	✓
LCDA7	✓
SDA6	✓
TA12	✓
TA16	✓
TA21	✓
TAB22	✓
TB2	✓
TB16	✓
TB24	✓
USCI20	✓
USCI22	✓
USCI23	✓
USCI24	✓
USCI25	✓
USCI26	✓
USCI28	✓
USCI30	✓
USCI34	✓
USCI35	✓
USCI40	✓
XOSC5	✓
XOSC8	✓
XOSC9	✓

## 2 Preprogrammed Software Errata Revision History

Errata impacting pre-programmed software into the silicon by Texas Instruments.

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

The device doesn't have Software in ROM errata.

### 3 Debug only Errata Revision History

Errata only impacting debug operation.

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

Errata Number	Rev B
<a href="#">EEM20</a>	✓
<a href="#">JTAG23</a>	✓

### 4 Fixed by Compiler Errata Revision History

Errata completely resolved by compiler workaround. Refer to specific erratum for IDE and compiler versions with workaround.

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

Errata Number	Rev B
<a href="#">CPU4</a>	✓
<a href="#">CPU19</a>	✓

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](#)

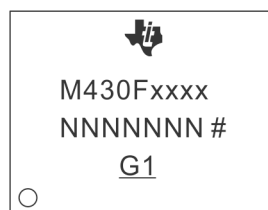
#### IAR Embedded Workbench

- [IAR workarounds for msp430 hardware issues](#)

### 5 Package Markings

#### ZQW113

#### **BGA (ZQW), 113 Pin**



# = Die revision  
 ○ = Pin 1 location  
 N = Lot trace code

#### PN80

#### **LQFP (PN), 80 Pin**

 NNNNNNN  
M430Fxxxx  
REV #  
○

# = Die revision  
○ = Pin 1 location  
N = Lot trace code

 NNNNNNNG4  
M430Fxxxx  
REV #  
○

# = Die revision  
○ = Pin 1 location  
N = Lot trace code

## 6 Detailed Bug Description

### CPU4 *CPU Module*

**Category** Compiler-Fixed

**Function** PUSH #4, PUSH #8CPU4 - Bug

**Description** The single operand instruction PUSH cannot use the internal constants (CG) 4 and 8. The other internal constants (0, 1, 2, -1) can be used. The number of clock cycles is different:

PUSH #CG uses address mode 00, requiring 3 cycles, 1 word instruction

PUSH #4/#8 uses address mode 11, requiring 5 cycles, 2 word instruction

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

IDE/Compiler	Version Number	Notes
IAR Embedded Workbench	IAR EW430 v2.x until v6.20	User is required to add the compiler flag option below. --hw_workaround=CPU4
IAR Embedded Workbench	IAR EW430 v6.20 or later	Workaround is automatically enabled
TI MSP430 Compiler Tools (Code Composer Studio)	v1.1 or later	
MSP430 GNU Compiler (MSP430-GCC)	MSP430-GCC 4.9 build 167 or later	

### 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

IDE/Compiler	Version Number	Notes
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

**EEM20**
***EEM Module***


---

**Category**

Debug

**Function**

Debugger might clear interrupt flags

**Description**

During debugging read-sensitive interrupt flags might be cleared as soon as the debugger stops. This is valid in both single-stepping and free run modes.

**Workaround**

None.

**FLASH19**
***FLASH Module***


---

**Category**

Functional

**Function**

EEI feature does not work for code execution from RAM

**Description**

When the program is executed from RAM, the flash controller EEI feature does not work. The erase cycle is suspended and the interrupt is serviced, but there is a problem while resuming with the erase cycle.

Addresses applied to flash are different than the actual values while resuming erase cycle after ISR execution.

**Workaround**

None

**FLASH24**
***FLASH Module***


---

**Category**

Functional

**Function**

Write or erase emergency exit can cause failures

**Description**

When a flash write or erase is abruptly terminated, the following flash accesses by the CPU may be unreliable resulting in erroneous code execution. The abrupt termination can be the result of one the following events:

1) The flash controller clock is configured to be sourced by an external crystal. An oscillator fault occurs thus stopping this clock abruptly.

or

2) The Emergency Exit bit (EMEX in FCTL3) when set forces a write or an erase operation to be terminated before normal completion.

or

3) The Enable Emergency Interrupt Exit bit (EEIEX in FCTL1) when set with GIE=1 can lead to an interrupt causing an emergency exit during a Flash operation.

**Workaround**

1) Use the internal DCO as the flash controller clock provided from MCLK or SMCLK.

or

2) After setting EMEX = 1, wait for a sufficient amount of time before Flash is accessed again.

or

3) No Workaround. Do not use EEIEX bit.

**FLASH27**
***FLASH Module***


---

**Category**

Functional

**Function**

EEL feature can disrupt segment erase

**Description**

When a flash segment erase operation is active with EEL feature selected (EEL=1 in FLCTL1) and GIE=0, the following can occur:

An interrupt event causes the flash erase to be stopped, and the flash controller expects an RETI to resume the erase. Because GIE=0, interrupts are not serviced and RETI will never happen.

**Workaround**

1) Do not set bit EEL=1 when GIE = 0.

or,

2) Force an RETI instruction during the erase operation during the check for BUSY=1 (FCTL3).

Sample code:

```
MOV R5, 0(R5) ; Dummy write, erase segment
```

```
LOOP: BIT #BUSY, &FCTL3 ; test busy bit
```

```
JMP SUB_RETI ; Force RETI instruction
```

```
JNZ LOOP ; loop while BUSY=1
```

```
SUB_RETI: PUSH SR
```

```
RETI
```

**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

<b>FLL6</b>	<b><i>FLL+ Module</i></b>
<b>Category</b>	Functional
<b>Function</b>	LFXT1DIG bit is read incorrectly
<b>Description</b>	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.
<b>Workaround</b>	None
<b>JTAG23</b>	<b><i>JTAG Module</i></b>
<b>Category</b>	Debug
<b>Function</b>	PSA checksum calculation does not work in marginal read mode.
<b>Description</b>	If the PSA checksum is calculated via JTAG interface in marginal read mode the MRG0 and MRG1 bits in the FCTL4 register are reset.
<b>Workaround</b>	None.
<b>LCDA5</b>	<b><i>LCD_A Module</i></b>
<b>Category</b>	Functional
<b>Function</b>	Wrong cycle time for first cycle of COMx/Sx signals
<b>Description</b>	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
<b>Workaround</b>	Not required, because it does not influence the LCD function.
<b>LCDA7</b>	<b><i>LCD_A Module</i></b>
<b>Category</b>	Functional
<b>Function</b>	Higher current consumption when using shared LCD ports as fast toggling outputs
<b>Description</b>	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.
<b>Workaround</b>	<ol style="list-style-type: none"> <li>1. Do not use shared LCD pins as fast toggling outputs if an LCD is used.</li> <li>2. Reduce the toggle frequency of the shared pin to &lt;10kHz.</li> </ol>
<b>SDA6</b>	<b><i>SD16_A Module</i></b>
<b>Category</b>	Functional
<b>Function</b>	SD16CCTL0 write leads to unexpected results from SD16_A.
<b>Description</b>	Writing to the reserved bit (Bit-15) of the SD16CCTL0 register may lead to unexpected results from the SD16_A.

**Workaround** None

---

**TA12** *TIMER\_A 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** *TIMER\_A 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

---

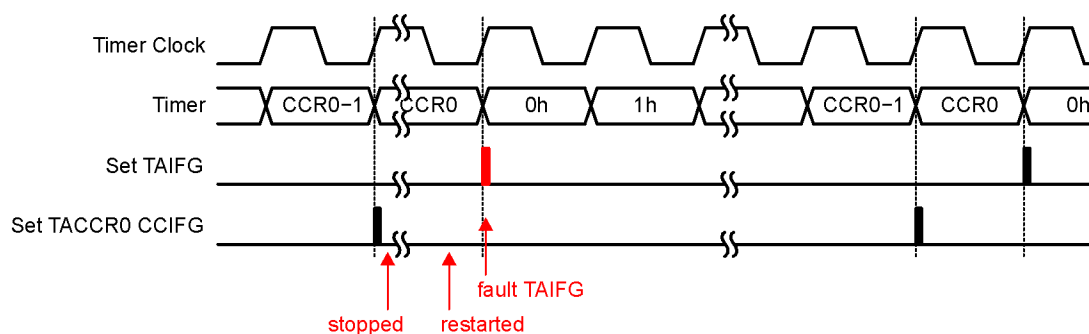
**TA21** *TIMER\_A Module*

---

**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.



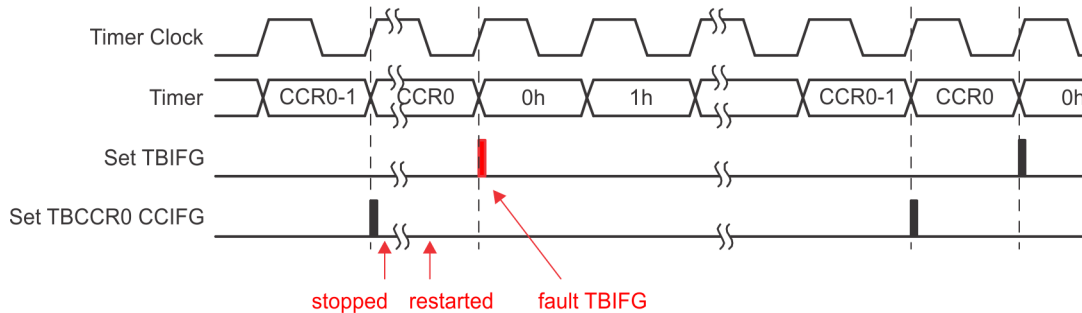
**Workaround** None.



<b>TAB22</b>	<b><i>TIMER_A/TIMER_B Module</i></b>
<b>Category</b>	Functional
<b>Function</b>	Timer_A/Timer_B register modification after Watchdog Timer PUC
<b>Description</b>	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).
<b>Workaround</b>	<p>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.</p> <p>Example code:</p> <pre>MOV.W #VAL, &amp;TACTL</pre> <p>or</p> <pre>MOV.W #VAL, &amp;TBCTL</pre> <p>Where, VAL=0, if Timer is not used in application otherwise, user defined per desired function.</p>
<b>TB2</b>	<b><i>TIMER_B Module</i></b>
<b>Category</b>	Functional
<b>Function</b>	Interrupt is lost (slow ACLK)
<b>Description</b>	<p>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).</p> <p>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.</p>
<b>Workaround</b>	Switch capture/compare mode to capture mode before the CCRx register increment. Switch back to compare mode afterward.
<b>TB16</b>	<b><i>TIMER_B Module</i></b>
<b>Category</b>	Functional
<b>Function</b>	First increment of TBR erroneous when IDx > 00
<b>Description</b>	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.
<b>Workaround</b>	None

**TB24** *TIMER\_B 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.

**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

## 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.

<b>USCI25</b>	<b><i>USCI Module</i></b>
<b>Category</b>	Functional
<b>Function</b>	TXIFG is not reset when NACK is received in I2C mode
<b>Description</b>	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).
<b>Workaround</b>	Reset TXIFG in software within the NACKIFG interrupt service routine
<b>USCI26</b>	<b><i>USCI Module</i></b>
<b>Category</b>	Functional
<b>Function</b>	Tbuf parameter violation in I2C multi-master mode
<b>Description</b>	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.
<b>Workaround</b>	None
<b>USCI28</b>	<b><i>USCI Module</i></b>
<b>Category</b>	Functional
<b>Function</b>	Timing of USCI I2C interrupts may cause device reset due to automatic clear of an IFG.
<b>Description</b>	When certain USCI I2C interrupt flags (IFG) are set and an automatic flag-clearing event on the I2C bus occurs, it results in an errant ISR call to the reset vector. 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.
<b>Workaround</b>	(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.  or  (3) At program start, check any applicable enabled IE bits such as UCBxTXIE, UCBxRXIE, UCSTTIE, UCSTPIE or UCNACKIE for a reset (A PUC will clear all of the IE bits of interest). If no PUC occurred then the device ran into the above mentioned errant

condition and the program counter will need to be restored using an RETI instruction.

; ----- Workaround (3) example for TXIFG -----

Note: For assembly code use code snippet shown below and insert prior to user code

main

bit.b #UCBxTXIE ,&IE2 ; if TXIE is set, errant call occurred

jz start\_normal ; if not start main program

reti ; else return from interrupt call

start\_normal

... ; Application code continues

Note: For C code the workaround will need to be executed prior to the CSTARTUP routine. The steps for modifying the CSTARTUP routine are IDE dependent.

Examples for Code Composer and IAR Embedded Workbench are shown below.

IAR Embedded Workbench:

1) The file cstartup.s43 is found at: ...\\IAR Systems\\<Current Embedded Workbench Version>\\430\\src\\lib\\430

2) Create a local copy of this file and link it to the project. Do not rename the file.

3) In the copy insert the following code prior to stack pointer initialization as shown:

```
#define IE2 (0x0001)
BIT.B #0x08,&IE2 ; if TXIE is set, errant call occurred
JZ Start_Normal ; if not start main program
RETI ; else return from interrupt call
// Initialize SP to point to the top of the stack.
Start_Normal
MOV #SFE(CSTACK), SP
// Ensure that main is called.
```

Code Composer:

1) The file boot.c is found at ...\\Texas Instruments\\<Current Code Composer Version>\\tools\\compiler\\MSP430\\lib\\rtssrc.zip

2) Extract the file from rtssrc.zip and create a local copy. Link the copy to the project. Do not rename this file.

3) In the copy insert the following code prior to stack pointer initialization as shown:

```
__asm("\t BIT.B\t #0x08,&0x0001"); // if TXIE is set, errant call occurred
__asm("\t JZ\t Start_Normal"); // if not start main program
__asm("\t RETI"); // else return from interrupt call
__asm("Start_Normal");
```

```
/*----- */
```

```
/* Initialize stack pointer. Stack grows toward lower memory. */
```

```
/*-----*/
```

Insert the code here:

```

/*****/
/* C_INT00() - C ENVIRONMENT ENTRY POINT */
/*****/

#pragma CLINK(_c_int00)
extern void __interrupt _c_int00()
{
// <-- INSERT USCI28 WORKAROUND HERE
STACK_INIT();

```

**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 UCSCLOW 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 UCSCLOW.UCBxSTAT == 1
- (3) If no, repeat step 2 until set

(4) If yes, repeat step 2 for a time period  $> 3 \times t$  (BitClock) where  $t$  (BitClock) =  $1/f$  (BitClock)

(5) If window of  $3 \times t$  (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;
```

**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 Document Revision History

Changes from family erratasheet to device specific erratasheet.

1. Errata LCDA7 was added

Changes from device specific erratasheet to document Revision A.

1. Errata EEM20 was added to the errata documentation.

Changes from document Revision A to Revision B.

1. Errata TA21 was added to the errata documentation.

Changes from document Revision B to Revision C.

1. Errata TB24 was added to the errata documentation.

Changes from document Revision C to Revision D.

1. Errata USCI35 was added to the errata documentation.

Changes from document Revision D to Revision E.

1. Errata JTAG23 was added to the errata documentation.

Changes from document Revision E to Revision F.

1. Silicon Revision C was added to the errata documentation.

Changes from document Revision F to Revision G.

Changes from document Revision G to Revision H.

1. Package Markings section was updated.

Changes from document Revision H to Revision I.

1. Errata USCI40 was added to the errata documentation.

Changes from document Revision I to Revision J.

1. TA21 Description was updated.

Changes from document Revision J to Revision K.

1. USCI28 Workaround was updated.

Changes from document Revision K to Revision L.

1. Workaround for CPU19 was updated.
2. Function for CPU4 was updated.
3. Workaround for CPU4 was updated.

Changes from document Revision L to Revision M.

1. Erratasheet format update.
2. Added errata category field to "Detailed bug description" section

Changes from document Revision M to Revision N.

1. USCI34 was added to the errata documentation.

Changes from document Revision N to Revision O.

1. Description for TB24 was updated.

## 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, 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 will 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 ([www.ti.com/legal/termsofsale.html](http://www.ti.com/legal/termsofsale.html)) or other applicable terms available either on [ti.com](http://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.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265  
Copyright © 2019, Texas Instruments Incorporated