

Errata  
**MSP430G2403 Microcontroller**



**ABSTRACT**

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

---

**Table of Contents**

<b>1 Functional Advisories</b> .....	2
<b>2 Preprogrammed Software Advisories</b> .....	2
<b>3 Debug Only Advisories</b> .....	2
<b>4 Fixed by Compiler Advisories</b> .....	2
<b>5 Nomenclature, Package Symbolization, and Revision Identification</b> .....	4
5.1 Device Nomenclature.....	4
5.2 Package Markings.....	4
5.3 Memory-Mapped Hardware Revision (TLV Structure).....	5
<b>6 Advisory Descriptions</b> .....	6
<b>7 Revision History</b> .....	15

## 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 C	Rev B	Rev A
<a href="#">BCL12</a>	✓	✓	✓
<a href="#">SYS15</a>	✓	✓	✓
<a href="#">TA12</a>	✓	✓	✓
<a href="#">TA16</a>	✓	✓	✓
<a href="#">TA21</a>	✓	✓	✓
<a href="#">TAB22</a>	✓	✓	✓
<a href="#">USCI20</a>	✓	✓	✓
<a href="#">USCI22</a>	✓	✓	✓
<a href="#">USCI23</a>	✓	✓	✓
<a href="#">USCI24</a>	✓	✓	✓
<a href="#">USCI25</a>	✓	✓	✓
<a href="#">USCI26</a>	✓	✓	✓
<a href="#">USCI29</a>	✓	✓	✓
<a href="#">USCI30</a>	✓	✓	✓
<a href="#">USCI34</a>	✓	✓	✓
<a href="#">USCI35</a>	✓	✓	✓
<a href="#">USCI40</a>	✓	✓	✓
<a href="#">XOSC5</a>	✓	✓	✓

## 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.

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

## 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 C	Rev B	Rev A
<a href="#">CPU4</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 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.

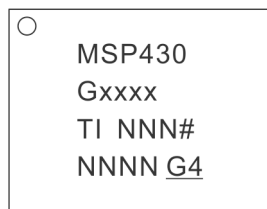
Predictions show that prototype devices (XMS) have a greater failure rate than the standard production devices. TI recommends that these devices not be used in any production system because their expected end-use failure rate still is undefined. Only qualified production devices are to be used.

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

**RHB32**

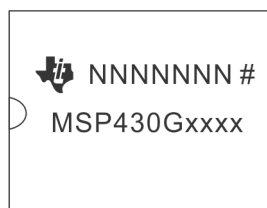
***QFN (RHB), 32 Pin***



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

**N20**

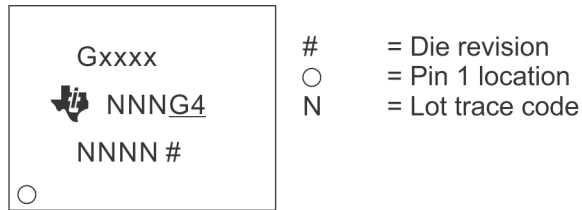
***PDIP (N), 20 Pin***



# = Die revision  
N = Lot trace code

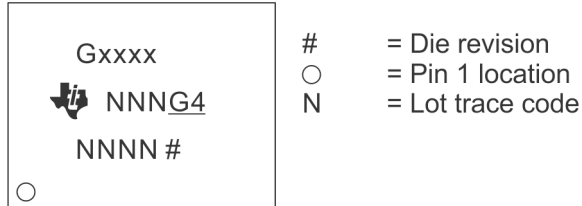
**PW20**

***TSSOP (PW), 20 Pin***



**PW28**

**TSSOP (PW), 28 Pin**



### 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

### BCL12 *BCL Module*

---

**Category** Functional

**Function** Switching RSELx or modifying DCOCTL can cause DCO dead time or a complete DCO stop

**Description** After switching RSELx bits (located in register BCSTL1) from a value of >13 to a value of <12 OR from a value of <12 to a value of >13, the resulting clock delivered by the DCO can stop before the new clock frequency is applied. This dead time is approximately 20 us. In some instances, the DCO may completely stop, requiring a power cycle.

Furthermore, if all of the RSELx bits in the BCSTL1 register are set, modifying the DCOCTL register to change the DCOx or the MODx bits could also result in DCO dead time or DCO hang up.

**Workaround** - When switching RSEL from >13 to <12, use an intermediate frequency step. The intermediate RSEL value should be 13.

Current RSEL	Target RSEL	Recommended Transition Sequence
15	14	Switch directly to target RSEL
14 or 15	13	Switch directly to target RSEL
14 or 15	0 to 12	Switch to 13 first, and then to target RSEL (two step sequence)
0 to 13	0 to 12	Switch directly to target RSEL

AND

- When switching RSEL from <12 to >13 it's recommended to set RSEL to its default value first (RSEL = 7) before switching to the desired target frequency.

AND

- In case RSEL is at 15 (highest setting) it's recommended to set RSEL to its default value first (RSEL = 7) before accessing DCOCTL to modify the DCOx and MODx bits. After the DCOCTL register modification the RSEL bits can be manipulated in an additional step.

In the majority of cases switching directly to intermediate RSEL steps as described above will prevent the occurrence of BCL12. However, a more reliable method can be implemented by changing the RSEL bits step by step in order to guarantee safe function without any dead time of the DCO.

Note that the 3-step clock startup sequence consisting of clearing DCOCTL, loading the BCSTL1 target value, and finally loading the DCOCTL target value as suggested in the in the "TLV Structure" chapter of the [MSP430x2xx Family User's Guide](#) is not affected by BCL12 if (and only if) it is executed after a device reset (PUC) prior to any other modifications being made to BCSTL1 since in this case RSEL still is at its default value of 7. However any further changes to the DCOx and MODx bits will require the consideration of the workaround outlined above.

### CPU4 *CPU Module*

---

**Category** Compiler-Fixed

**Function** PUSH #4, PUSH #8

**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	

**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.

**SYS15** ***SYS Module***

---

**Category** Functional

**Function** LPM3 and LPM4 currents exceed specified limits

**Description** LPM3 and LPM4 currents may exceed specified limits if the SMCLK source is switched from DCO to VLO or LFXT1 just before the instruction to enter LPM3 or LPM4 mode.

**Workaround** After clock switching, a delay of at least four new clock cycles (VLO or LFXT1) must be implemented to complete the clock synchronization before going into LPM3 or LPM4.

**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.

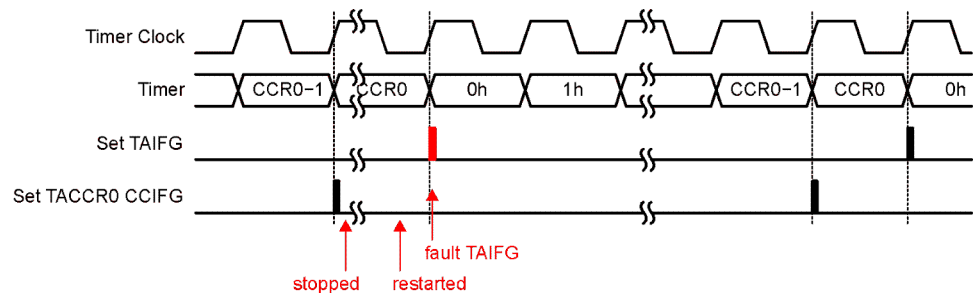
**Workaround** None

## 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.

## 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.

<b>Workaround</b>	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).
<b>USCI23</b>	<b><i>USCI Module</i></b>
<hr/>	
<b>Category</b>	Functional
<b>Function</b>	UART transmit mode with automatic baud rate detection
<b>Description</b>	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.
<b>Workaround</b>	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.
<b>USCI24</b>	<b><i>USCI Module</i></b>
<hr/>	
<b>Category</b>	Functional
<b>Function</b>	Incorrect baud rate information during UART automatic baud rate detection mode
<b>Description</b>	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.
<b>Workaround</b>	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>
<hr/>	
<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>
<hr/>	
<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.

**Workaround** None

**USCI29** *USCI Module*

**Category** Functional

**Function** Timing of USCI I2C interrupts may result in call to a reserved ISR location

**Description** When certain USCI I2C interrupt flags (IFG) are set and an automatic flag-clearing event on the I2C bus occurs, the device makes a call to the TRAPINT interrupt vector. This will only happen if 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) Poll the affected flags instead of enabling the interrupts.

(2) Define an ISR for the interrupt vector TRAPINT. If the failure condition occurs; a call to the TRAPINT ISR is made. After the interrupt is serviced, the device returns to the application code and continues execution.

Include the following ISR definition in the application code.

```
#pragma vector= TRAPINT_VECTOR
__interrupt void TRAPINT_ISR(void)
{
    __no_operation();
}
```

For IDE versions earlier than IAR V4.22 and CCS V4.2 in addition to the above code; include the ISR definition in the device header file.

In IAR Embedded Workbench include the following line in the device header file msp430xxx.h.

```
/*
 * Interrupt Vectors (offset from 0xFFE0)
 */
#define TRAPINT_VECTOR (0 * 2u) /*INCLUDE THIS LINE IN .h FILE*/
#define PORT1_VECTOR (2 * 2u) /* 0xFFE4 Port 1 */
```

In Code Composer Essentials/Studio include the following line in the device header file msp430xxx.h.

```
/*
 * Interrupt Vectors (offset from 0xFFE0)
 */
```

```
#define TRAPINT_VECTOR (0 * 1u) /*INCLUDE THIS LINE IN .h FILE*/
#define PORT1_VECTOR (2 * 1u) /* 0xFFE4 Port 1 */
```

## 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.  $3 \times t(\text{BitClock})$ .

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  $> 3 \times t(\text{BitClock})$  where  $t(\text{BitClock}) = 1/f(\text{BitClock})$
- (5) If window of  $3 \times t(\text{BitClock})$  cycles has elapsed, it is safe to read UCBxRXBUF

## USCI34

### *USCI Module*

---

<b>Category</b>	Functional
<b>Function</b>	I2C multi-master transmit may lose first few bytes.
<b>Description</b>	<p>In an I2C multi-master system (UCMM =1), under the following conditions:</p> <p>(1)the master is configured as a transmitter (UCTR =1)</p> <p>AND</p> <p>(2)the start bit is set (UCTXSTT =1);</p> <p>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.</p>
<b>Workaround</b>	<p>Verify that the START condition has been sent (UCTXSTT =0) before loading TXBUF with data.</p>

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
}
}
}
```

<b>USCI35</b>	<b><i>USCI Module</i></b>
<b>Category</b>	Functional
<b>Function</b>	Violation of setup and hold times for (repeated) start in I2C master mode
<b>Description</b>	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

## 7 Revision History

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

<b>Changes from May 12, 2021 to May 17, 2021</b>	<b>Page</b>
<ul style="list-style-type: none"><li>Changed the document format and structure; updated the numbering format for tables, figures, and cross references throughout the document.....</li></ul>	<b>6</b>

## 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 (<https://www.ti.com/legal/termsofsale.html>) or other applicable terms available either on [ti.com](https://www.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 © 2021, Texas Instruments Incorporated