Errata **MSP430FR6005 Microcontroller**

TEXAS INSTRUMENTS

ABSTRACT

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

Table of Contents

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



1 Functional Advisories

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

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

Errata Number	Rev B	Rev A
ADC42	1	\checkmark
ADC64	1	
ADC65	1	✓
ADC68	\checkmark	
ADC69	1	1
ADC70	۰ ۱	
ADC71	↓	1
CPU46	1	✓ ✓
CPU47	1	1
CS12	1	1
PMM31	1	1
PMM32	✓ ✓	<
RTC12	↓	
TB25	1	✓
USCI41	√ √	
USCI42	1	1
USCI45	1	
USCI47	イ イ	1
USCI50	1	1
USCI52	1	\checkmark

2 Preprogrammed Software Advisories

Advisories that affect factory-programmed software.

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

Errata Number	Rev B	Rev A
ADC67	1	1

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 B	Rev A
JTAG27	1	

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.

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



Errata Number	Rev B	Rev A
CPU21		✓
CPU22		\checkmark
CPU40		\checkmark

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.

#

 \bigcirc

N

5.2 Package Markings

PZ100

LQFP (PZ) 100 Pin



Die revisionPin 1 locationLot trace code

5.3 Memory-Mapped Hardware Revision (TLV Structure)

Die Revision	TLV Hardware Revision
Rev B	20h
Rev A	10h

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

ADC42	ADC Module	
Category	Functional	
Function	ADC stops converting when successive ADC is triggered before the previous conversion ends	
Description	Subsequent ADC conversions are halted if a new ADC conversion is triggered while ADC is busy. ADC conversions are triggered manually or by a timer. The affected ADC modes are:	
	- sequence-of-channels	
	- repeat-single-channel	
	- repeat-sequence-of-channels (ADC12CTL1.ADC12CONSEQx)	
	In addition, the timer overflow flag cannot be used to detect an overflow (ADC12IFGR2.ADC12TOVIFG).	
Workaround	1. For manual trigger mode (ADC12CTL0.ADC12SC), ensure each ADC conversion is completed by first checking ADC12CTL1.ADC12BUSY bit before starting a new conversion.	
	2. For timer trigger mode (ADC12CTL1.ADC12SHP), ensure the timer period is greater than the ADC sample and conversion time.	
	To recover the conversion halt:	
	1. Disable ADC module (ADC12CTL0.ADC12ENC = 0 and ADC12CTL0.ADC12ON = 0)	
	2. Re-enable ADC module (ADC12CTL0.ADC12ON = 1 and ADC12CTL0.ADC12ENC = 1)	
	3. Re-enable conversion	
ADC64	ADC Module	
Category	Functional	
Function	Incorrect conversion result in extended sample mode in some conditions	
Description	The ADC12 conversion result can be incorrect if the extended sample mode is selected (ADC12SHP = 0), ADC12VRSEL is set to 0, 2, 4, 6, 12, 14 (VR+ and VR- unbuffered), and the ADC sample time is less than 6 ADC clock cycles.	
Workaround	1) Use Pulse sample mode (ADC12SHP=1) if sample time less than 6 ADC clock cycles is needed;	
	OR	
	2) In extended sample mode (ADC12SHP = 0) increase the sample time to at least 6 ADC clock cycles;	
	OR	

Advisory	Descri	ptions



ADC64 (continued)) ADC Module		
	3) Use reference mode corresponding to ADC12VRSEL =1,3,5,7,9,13,15		
ADC65	ADC Module		
Category	Functional		
Function	ADC12_B clock stays on between conversions in sequence-of-channels or repeated sequence-of-channels mode		
Description	When using the ADC in sequence-of-channels or repeat-sequence-of-channels mode (ADC12CONSEQx = 01 or 11), the ADC12_B always requests the ADC clock even between conversions. In this scenario, although the device may still enter LPM0, LPM1, LPM2 or LPM3, the selected ADC12_B clock source will always remain on, resulting in increased current consumption between ADC conversions.		
Workaround	To avoid the additional current consumption impact, different options will be needed depending on use case:		
	1. Configure ADC to Repeated-Single-Channel mode (ADC12CONSEQx = 10). Use the DMA or software to change the selected ADC12INCHx between conversions. With this option, the timing between conversions of different channels remains the same as normal ADC12 usage.		
	OR		
	2. Configure ADC to Sequence-of-Channels mode (ADC12CONSEQx = 01) with sequence of channels in Multiple Sample and Convert mode (ADC12CTL0.ADC12MSC = 1), then toggle the ADC12ENC bit by DMA or software after completing of each conversion sequence. With this option, the conversions of each channel in the sequence will happen immediately after the previous channel instead of waiting for the next trigger. This needs to be considered if timing between the sampling of different channels in the sequence matters for the application.		
ADC67	ADC Module		
Category	Software in ROM		
Function	Invalid ADC12 temperature sensor calibration data		
Description	The ADC12 reference temperature sensor calibration data stored in the TLV data structure (0x1A1A - 0x1A25) can be incorrect depending on the production lot trace code. As a result the temperature measurement when using these data can be wrong.		
	Devices with lot trace code > 88XXXXX are not affected by this issue.		
Workaround	Record the calibration data by taking ADC measurements of the temperature sensor at 30C and 85C for the required reference voltage. The calibration data in the TLV section (0x1A1A - 0x1A25) can't be overwritten but the new calibration data can be stored in user FRAM or info memory for further temperature calculations.		
ADC68	ADC Module		
Category	Functional		
Function	ADC may stop converting when the extended sample mode is selected		

ADC68 (continued)	ADC Module		
Description The ADC may stop converting, if the extended sample mode is selected (ADC12CTL1.ADC12SHP = 0) and the interrupt flag will not be set. The only way to recover the ADC conversion is to Disable ADC12_B module (ADC12CTL0.ADC12ENC = 0 and ADC12CTL0./ = 0)			
Workaround	Use the pulse sample mode (ADC12CTL1.ADC12SHP = 1).		
ADC69	ADC Module		
Category	Functional		
Function	ADC stops operating if ADC clock source is changed from SMCLK to another source while SMCLKOFF = 1.		
Description	When SMCLK is used as the clock source for the ADC (ADC12CTL1.ADC12SSELx = 11) and CSCTL4.SMCLKOFF = 1, the ADC will stop operating if the ADC clock source is changed by user software (e.g. in the ISR) from SMCLK to a different clock source. This issue appears only for the ADC12CTL1.ADC12DIVx settings /3/5/7. The hang state can be recovered by PUC/POR/BOR/Power cycle.		
Workaround	1. Set CSCTL4.SMCLKOFF = 0 before switch ADC clock source.		
	OR		
	2. Only use ADC12CTL1.ADC12DIVx as /1, /2, /4, /6, /8		
ADC70	ADC Module		
Category	Functional		
Function	DMA gets stuck when switching between ADC data transfer trigger types		
Description	If the ADC performs a data transfer by the CPU ,e.g. via interrupt or flag polling,		
AND the ADC is then configured for an edge-triggered DMA transfer THEN			
	the DMA cannot be triggered and no data transfer will occur.		
Workaround	 Do not switch between the ADC triggered CPU data transfer and ADC triggered DMA data transfer to avoid this condition. Or, Apply a POR reset to clear the trigger DMA trigger logic inside the ADC. Or, Perform a dummy DMA transfer via level trigger option (DMAxCTL.DMALEVEL=1) to clear ADC trigger logic. In this case, it is recommended to throw out current ADC conversion and ignore the data of the DMA dummy transfer. 		
ADC71	ADC Module		
Category	Functional		
Function	ADC12 stops converting when ENC and SC bits are set simultaneously and ADC12 was previously set to trigger from a source that was not the SC bit.		



ADC71 (continued)	ADC Module			
Description	When using the ADC12 after being setup and triggered by a trigger source that is not the ADC12SC bit (ADC Start Conversion), if ADC12ENC (ADC Enable) and ADC12SC bits are set simultaneously, then ADC12BUSY flag stays high and ADC12SC does not get cleared. As a result, the ADC conversion never finishes.			
Workaround	Set ADC12ENC and ADC12SC bits with separate instructions if the trigger source is changed from a source that is not ADC12SC.			
CPU21	CPU Module			
Category	Compiler-Fixed			
Function	Using POPM instruction on St	atus register may result in dev	ice hang up	
Description	When an active interrupt service request is pending and the POPM instruction is used to set the Status Register (SR) and initiate entry into a low power mode , the device may hang up.			
Workaround	None. It is recommended not	to use POPM instruction on the	e Status Register.	
	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)	v4.0.x or later	User is required to add the compiler or assembler flag option below silicon_errata=CPU21	
	MSP430 GNU Compiler (MSP430- GCC)	MSP430-GCC 4.9 build 167 or later		
CPU22	CPU Module			
Category	Compiler-Fixed			
Function	Indirect addressing mode with the Program Counter as the source register may produce unexpected results			
Description	When using the indirect addressing mode in an instruction with the Program Counter (PC) as the source operand, the instruction that follows immediately does not get executed. For example in the code below, the ADD instruction does not get executed.			
	mov @PC, R7 add #1h, R4			
Workaround	Refer to the table below for co	ompiler-specific fix implementa	tion information.	

IDE/Compiler	Version Number	Notes
IAR Embedded Workbench	Not affected	

8 MSP430FR6005 Microcontroller

CPU22 (continued) CPU Module

IDE/Compiler	Version Number	Notes
TI MSP430 Compiler Tools (Code Composer Studio)	v4.0.x or later	User is required to add the compiler or assembler flag option below silicon_errata=CPU22
MSP430 GNU Compiler (MSP430- GCC)	MSP430-GCC 4.9 build 167 or later	

CPU40 CPU Module

Category Compiler-Fixed

Function PC is corrupted when executing jump/conditional jump instruction that is followed by instruction with PC as destination register or a data section

Description If the value at the memory location immediately following a jump/conditional jump instruction is 0X40h or 0X50h (where X = don't care), which could either be an instruction opcode (for instructions like RRCM, RRAM, RLAM, RRUM) with PC as destination register or a data section (const data in flash memory or data variable in RAM), then the PC value is auto-incremented by 2 after the jump instruction is executed; therefore, branching to a wrong address location in code and leading to wrong program execution.

For example, a conditional jump instruction followed by data section (0140h).

@0x8012 Loop DEC.W R6 @0x8014 DEC.W R7 @0x8016 JNZ Loop @0x8018 Value1 DW 0140h

Workaround In assembly, insert a NOP between the jump/conditional jump instruction and program code with instruction that contains PC as destination register or the data section.

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

IDE/Compiler	Version Number	Notes
IAR Embedded Workbench	IAR EW430 v5.51 or later	For the command line version add the following information Compiler: hw_workaround=CPU40 Assembler:-v1
TI MSP430 Compiler Tools (Code Composer Studio)	v4.0.x or later	User is required to add the compiler or assembler flag option below silicon_errata=CPU40
MSP430 GNU Compiler (MSP430- GCC)	Not affected	

CPU46

CPU Module

Category Functional

Function

POPM peforms unexpected memory access and can cause VMAIFG to be set



CPU46 CPU Module Description When the POPM assembly instruction is executed, the last Stack Pointer increment is followed by an unintended read access to the memory. If this read access is performed on vacant memory, the VMAIFG will be set and can trigger the corresponding interrupt (SFRIE1.VMAIE) if it is enabled. This issue occurs if the POPM assembly instruction is performed up to the top of the STACK. Workaround If the user is utilizing C, they will not be impacted by this issue. All TI/IAR/GCC pre-built libraries are not impacted by this bug. To ensure that POPM is never executed up to the memory border of the STACK when using assembly it is recommended to either 1. Initialize the SP to a. TOP of STACK - 4 bytes if POPM.A is used b. TOP of STACK - 2 bytes if POPM.W is used OR

2. Use the POPM instruction for all but the last restore operation. For the last restore operation use the POP assembly instruction instead.

For instance, instead of using:

POPM.W #5,R13

Use:

```
POPM.W #4,R12
POP.W R13
```

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

IDE/Compiler	Version Number	Notes
IAR Embedded Workbench	Not affected	C code is not impacted by this bug. User using POPM instruction in assembler is required to implement the above workaround manually.
TI MSP430 Compiler Tools (Code Composer Studio)	Not affected	C code is not impacted by this bug. User using POPM instruction in assembler is required to implement the above workaround manually.
MSP430 GNU Compiler (MSP430- GCC)	Not affected	C code is not impacted by this bug. User using POPM instruction in assembler is required to implement the above workaround manually.

CPU47

CPU Module

Functional

Category

CPU47 (continued)	CPU Module	
Function	An unexpected Vacant Memory Access Flag (VMAIFG) can be triggered	
Description	An unexpected Vacant Memory Access Flag (VMAIFG) can be triggered, if a PC- modifying instruction (e.g ret, push, call, pop, jmp, br) is fetched from the last addresses (last 4 or 8 byte) of a memory (e.g FLASH, RAM, FRAM) that is not contiguous to a higher, valid section on the memory map. In debug mode using breakpoints the last 8 bytes are affected. In free running mode the last 4 bytes are affected.	
Workaround	Edit the linker command file to make the last 4 or 8 bytes of affected memory sections unavailable, to avoid PC-modifying instructions on these locations. Remaining instructions or data can still be stored on these locations.	
CS12	CS Module	
Category	Functional	
Function	DCO overshoot at frequency change	
Description	When changing frequencies (CSCTL1.DCOFSEL), the DCO frequency may overshoot and exceed the datasheet specification. After a time period of 10us has elapsed, the frequency overshoot settles down to the expected range as specified in the datasheet. The overshoot occur when switching to and from any DCOFSEL setting and impacts all peripherals using the DCO as a clock source. A potential impact can also be seen on FRAM accesses, since the overshoot may cause a temporary violation of FRAM access and cycle time requirements.	
Workaround	When changing the DCO settings, use the following procedure:	
	1) Store the existing CSCTL3 divider into a temporary unsigned 16-bit variable	
	2) Set CSCTL3 to divide all corresponding clock sources by 4 or higher	
	3) Change DCO frequency	
	4) Wait ~10us	
	5) Restore the divider in CSCTL3 to the setting stored in the temporary variable.	
	The following code example shows how to increase DCO to 16MHz.	
	<pre>uint16_t tempCSCTL3 = 0; CSCTL0_H = CSKEY_H; // Unlock CS registers /* Assuming SMCLK and MCLK are sourced from DCO */</pre>	

```
/* Store CSCTL3 settings to recover later */
tempCSCTL3 = CSCTL3;
/* Keep overshoot transient within specification by setting clk sources to
divide by 4*/
/* Clear the DIVS & DIVM masks (~0x77) and set both fields to 4 divider */
CSCTL3 = CSCTL3 & (~(0x77)) | DIVS_4 | DIVM_4;
CSCTL1 = DCOFSEL_4 | DCORSEL; // Set DCO to 16MHz
/* Delay by ~10us to let DCO settle. 60 cycles = 20 cycles buffer + (10us /
(1/4MHz)) */
delay_cycles(60);
CSCTL3 = tempCSCTL3; // Set all dividers
CSCTL0_H = 0; // Lock CS registers
```



JTAG27	JTAG Module
Category	Debug
Function	Unintentional code execution after programming via JTAG/SBW
Description	The device can unintentionally start executing code from uninitialized RAM addresses 0x0006 or 0x0008 after being programming via the JTAG or SBW interface. This can result in unpredictable behavior depending on the contents of the address location.
Workaround	1. If using programming tools purchased from TI (MSP-FET, LaunchPad), update to CCS version 6.1.3 later or IAR version 6.30 or later to resolve the issue.
	2. If using the MSP-GANG Production Programmer, use v1.2.3.0 or later.
	3. For custom programming solutions refer to the specification on MSP430 Programming Via the JTAG Interface User's Guide (SLAU320) revision V or newer and use MSPDebugStack v3.7.0.12 or later.
	For MSPDebugStack (MSP430.DLL) in CCS or IAR, download the latest version of the development environment or the latest version of the MSPDebugStack
	NOTE: This only affects debug mode.'
PMM31	PMM Module
Category	Functional
Function	Device may enter lockup state during transition from AM to LPM2/3/4
Description	The device might enter lockup state if the MODOSC is requested (e.g. triggered by ADC) or removed (e.g. end of ADC conversion) during a power mode transition from AM to LPM2/3/4 (e.g. during ISR exits or Status Register modifications). The same behavior can appear when SMCLK is requested during a power mode transition from AM to LPM3/4. The device will remain in a lockup state unable to respond to interrupts or continue application execution until a power cycle or external reset brings it back to reset state.
	Modules which can trigger MODCLK clock requests/removals are ADC and eUSCI in I2C mode using the clock low timeout feature (e.g. SMBus, PMBus). Modules which can trigger SMCLK clock requests are ADC, eUSCI in I2C Master mode, eUSCI in SPI Master mode and eUSCI in UART mode.
	If clock requests are started by the CPU/DMA (e.g. eUSCI during SPI master transmission), they can't occur at the same time as the power mode transition and thus should not be affected. The device should only be affected when the clock request is asynchronous to the power mode transition.
Workaround	1. Avoid using the aforementioned combinations of clock requests and power mode transitions:
	Use LPM0/1 instead of LPM2/3/4 when expecting asynchronous MODCLK requests and removals.
	OR
	Use LPM0/1/2 instead of LPM3/4 when expecting asynchronous SMCLK requests.

_ - - - - - - -- -PMM31 (c

PMM31 (continued) PMM Module		
	OR	
	Use LPMx.5 instead of LPM2/3/4.	
	OR	
	Use a clock different than MODCLK/SMCLK when applicable (e.g. ACLK).	
	Prevent the power mode transition from happening when an asynchronous clock request/removal is expected:	
	Wake-up device before a UART byte is received.	
	AND	
	Wake-up device before an asynchronous ADC trigger and stay in Active Mode until conversion is completed.	
	AND	
	Keep device in AM/LPM0/LPM1 during ADC measurement.	
PMM32	PMM Module	
Category	Functional	
Function	Device may enter lockup state or execute unintentional code during transition from AM to LPM2/3/4	
Description	The device might enter lockup state or start executing unintentional code resulting in unpredictable behavior depending on the contents of the address location- if any of the two conditions below occurs:	
	Condition1:	
	The following three events happen at the same time:	
	1) The device transitions from AM to LPM2/3/4 (e.g. during ISR exits or Status Register modifications),	
	AND	
	2) An interrupt is requested (e.g. GPIO interrupt),	
	AND	
	3) MODCLK is requested (e.g. triggered by ADC) or removed (e.g. end of ADC conversion).	
	Modules which can trigger MODCLK clock requests/removals are ADC and eUSCI.	
	If clock events are started by the CPU (e.g. eUSCI during SPI master transmission), they can not occur at the same time as the power mode transition and thus should not be affected. The device should only be affected when the clock event is asynchronous to the power mode transition	

SLAZ734B - FEBRUARY 2020 - REVISED AUGUST 2021 Submit Document Feedback

power mode transition.



PMM32 (continued) PMM Module

The device can recover from this lockup condition by a PUC/BOR/Power cycle (e.g. enable Watchdog to trigger PUC).
Condition2:
The following events happen at the same time:
1) The device transitions from AM to LPM2/3/4 (e.g. during ISR exits or Status Register modifications),
AND
2) An interrupt is requested (e.g. GPIO interrupt),
AND
3) Neither MODCLK nor SMCLK are running (e.g. requested by a peripheral),

AND

4) SMCLK is configured with a different frequency than MCLK.

The device can recover from this lockup condition by a BOR/Power cycle.

Workaround 1. Use LPM0/1/x.5 instead of LPM2/3/4.

OR

2. Place the FRAM in INACTIVE mode before any entry to LPM2/3/4 by clearing the FRPWR bit and FRLPMPWR bit (if exist) in the GCCTL0 register. This must be performed from RAM as shown below:

// define a function in RAM
#pragma CODE_SECTION(enterLpModeFromRAM,".TI.ramfunc")
void enterLpModeFromRAM(unsigned short lowPowerMode);

//call this function before any entry to LPM2/3/4
void enterLpModeFromRAM(unsigned short lowPowerMode)

FRCTL0 = FRCTLPW; GCCTL0 &= ~(FRPWR+FRLPMPWR); //clear FRPWR and FRLPMPWR FRCTL0_H = 0; //re-lock FRCTL __bis_SR_register(lowPowerMode); }

RTC12 RTC Module

Category	Functional
Function	Real-time clock temperature compensation RTCTCOK bit not retained after LPM3.5 wake up
Description	The RTC real-time clock temperature compensation write OK bit (RTCTCMP.RTCTCOK) is reset on wake up from LPM3.5 mode and does not get retained.

RTC12 (continued)	RTC Module	
Workaround	Store the RTCTCMP register content into FRAM for retention after wake up from LPM3.5	
TB25	TB Module	
Category	Functional	
Function	In up mode, TBxCCRn value is immediately transferred to TBxCLn when TBxCCTLn.CLLD bits are set or 0x01 or 0x10	
Description	IF Timer B is configured for Up mode, AND the compare latch load event (TBxCCTLn.CLLD bits) setting is configured to update TBxCCRn when TBxR reaches 0, THEN TBxCCRn will update immediately instead of the described condition. This is contrary to the user guide description of TBxCCTLn.CLLD = 0x01 or 0x10 modes. If user needs to update TBxCCRn value when TBxR counts to 0 in Timer B up mode: 1. Set TBxCCTLn. CLLD = 0x00 2. Enable the Timer B interrupt (TBIE) in TBxCTL 3. Update TBxCCRn value within interrupt routine. Timer B Interrupt would need to be serviced in a timely manner to mitigate disruption or unintended timer output if an output mode is used.	
USCI41	USCI Module	
Category	Functional	
Function	UCBUSY bit of eUSCIA module might not work reliable when device is in SPI mode.	
Description	When eUSCIA is configured in SPI mode, the UCBUSY bit might get stuck to 1 or start toggling after transmission is completed. This happens in all four combinations of Clock Phase and Clock Polarity options (UCAxCTLW0.UCCKPH & UCAxCTLW0.UCCKPL bits) as well as in Master and Slave mode. There is no data loss or corruption. However the UCBUSY cannot be used in its intended function to check if transmission is completed. Because the UCBUSY bit is stuck to 1 or toggles, the clock request stays enabled and this adds additional current consumption in low power mode operation.	
Workaround	For correct functional implementation check on transmit or receive interrupt flag UCTXIFG/UCRXIFG instead of UCBUSY to know if the UCAxTXBUF buffer is empty or ready for the next complete character. To reduce the additional current it is recommended to either reset the SPI module (UCAxCTLW0.UCSWRST) in the UCBxCTLW0 or send a dummy byte 0x00 after the intended SPI transmission is completed.	
USCI42	USCI Module	
Category	Functional	
Function	UART asserts UCTXCPTIFG after each byte in multi-byte transmission	



USCI42 (continued)	USCI Module		
Description	UCTXCPTIFG flag is triggered at the last stop bit of every UART byte transmission, independently of an empty buffer, when transmitting multiple byte sequences via UART. The erroneous UART behavior occurs with and without DMA transfer.		
Workaround	None.		
USCI45	USCI Module		
Category	Functional		
Function	Unexpected SPI clock stretching possible when UCxCLK is asynchronous to MCLK		
Description	In rare cases, during SPI communication, the clock high phase of the first data bit may be stretched significantly. The SPI operation completes as expected with no data loss. This issue only occurs when the USCI SPI module clock (UCxCLK) is asynchronous to the system clock (MCLK).		
Workaround	Ensure that the USCI SPI module clock (UCxCLK) and the CPU clock (MCLK) are synchronous to each other.		
USCI47	USCI Module		
Category	Functional		
Function	eUSCI SPI slave with clock phase UCCKPH = 1		
Description The eUSCI SPI operates incorrectly under the following conditions:			
	1. The eUSCI_A or eUSCI_B module is configured as a SPI slave with clock phase mode UCCKPH = 1		
	AND		
	2. The SPI clock pin is not at the appropriate idle level (low for UCCKPL = 0, high for UCCKPL = 1) when the UCSWRST bit in the UCxxCTLW0 register is cleared.		
	If both of the above conditions are satisfied, then the following will occur: eUSCI_A: the SPI will not be able to receive a byte (UCAxRXBUF will not be filled and UCRXIFG will not be set) and SPI slave output data will be wrong (first bit will be missed and data will be shifted). eUSCI_B: the SPI receives data correctly but the SPI slave output data will be wrong (first byte will be duplicated or replaced by second byte).		
Workaround	Use clock phase mode UCCKPH = 0 for MSP SPI slave if allowed by the application.		
	OR		
	The SPI master must set the clock pin at the appropriate idle level (low for UCCKPL = 0, high for UCCKPL = 1) before SPI slave is reset (UCSWRST bit is cleared).		
	OR		
	For eUSCI_A: to detect communication failure condition where UCRXIFG is not set, check both UCRXIFG and UCTXIFG. If UCTXIFG is set twice but UCRXIFG is not set, reset the MSP SPI slave by setting and then clearing the UCSWRST bit, and inform the SPI master to resend the data.		

USCI50	USCI Module
Category	Functional
Function	Data may not be transmitted correctly from the eUSCI when operating in SPI 4-pin master mode with UCSTEM = 0
Description	When the eUSCI is used in SPI 4-pin master mode with UCSTEM = 0 (STE pin used as an input to prevent conflicts with other SPI masters), data that is moved into UCxTXBUF while the UCxSTE input is in the inactive state may not be transmitted correctly. If the eUSCI is used with UCSTEM = 1 (STE pin used to output an enable signal), data is transmitted correctly.
Workaround	When using the STE pin in conflict prevention mode (UCSTEM = 0), only move data into UCxTXBUF when UCxSTE is in the active state. If an active transfer is aborted by UCxSTE transitioning to the master-inactive state, the data must be rewritten into UCxTXBUF to be transferred when UCxSTE transitions back to the master-active state.
USCI52	USCI Module
Category	Functional
Function	Interrupt Flag polling can cause unrpedictable behavior if eUSCI is running with different clock then CPU
Description	If the interrupt flags e.g. UCAIFG of the eUSCI are polled via the CPU during a while loop or during the following assembly instruction the CPU behavior can become unpredictable when the corresponding interrupt flag is set asynchronously during the execution of the bit instruction. The corresponding CPU flags used by the conditional jump can become unstable causing unpredictable CPU execution. CHECK_FLAG
	BIT.W #8,&UCAxIFG JEQ CHECK_FLAG
Workaround	Buffer the interrupt flag which should be polled, to a CPU internal register e.g. R15 and use this register for the decision (bit + jeq). The bit of interest remains stable during the CPU decision because the value is settled in the CPU internal register. In C, declaring the buffer variable as volatile avoids compiler optimizations that may result in the assembly code above.
	Workaround in C: volatile unsigned short flag; do { flag = UCA0IG; }while ((flag & UCRXIFG) == 0x00); Workaround in asm: PUSH R15 loop: MOV.W &UCAxIFG, R15 BIT.W #8,R15
	JEQ loop POP R15



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

Changes from July 14, 2021 to August 27, 2021	
ADC70 was added to the errata documentation	
ADC71 was added to the errata documentation	5
TB25 was added to the errata documentation	5
ADC70 Description was updated	
ADC70 Workaround was updated	
ADC71 Description was updated	
ADC71 Workaround was updated	
TB25 Description was updated	
TB25 Workaround was updated	

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES "AS IS" AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you 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 or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI's provision of these resources does not expand or otherwise alter TI's applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265 Copyright © 2021, Texas Instruments Incorporated