Errata
MSP430FG6425 Microcontroller

TEXAS INSTRUMENTS

ABSTRACT

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

Table of Contents

1 Functional Advisories

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

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

Errata Number	Rev A
COMP10	✓
CPU37	✓
CPU47	✓
CTSD1	✓
DAC5	✓
DMA4	✓
DMA7	✓
DMA10	✓
LCDB5	✓
LCDB6	✓
PMM11	✓
PMM12	✓
PMM14	✓
PMM15	✓
PMM18	✓
PMM20	✓
PMM26	✓
PORT15	✓
PORT19	✓
RTC16	✓
TB25	✓
UCS11	✓
USCI26	✓
USCI34	✓
USCI35	✓
USCI39	✓
USCI40	✓

2 Preprogrammed Software Advisories

Advisories that affect factory-programmed software.

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

Errata Number	Rev A
BSL14	✓

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 A
EEM16	✓

Errata Number	Rev A
EEM17	✓
EEM19	✓
EEM23	✓

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 A
CPU21	✓
CPU22	✓
CPU40	✓

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

TI MSP430 Compiler Tools (Code Composer Studio IDE)

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

MSP430 GNU Compiler (MSP430-GCC)

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

IAR Embedded Workbench

- IAR workarounds for msp430 hardware issues

5 Nomenclature, Package Symbolization, and Revision Identification

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

5.1 Device Nomenclature

To designate the stages in the product development cycle, TI assigns prefixes to the part numbers of all MSP MCU devices. Each MSP MCU commercial family member has one of two prefixes: MSP or XMS. These prefixes represent evolutionary stages of product development from engineering prototypes (XMS) through fully qualified production devices (MSP).

XMS – Experimental device that is not necessarily representative of the final device's electrical specifications

MSP – Fully qualified production device

Support tool naming prefixes:

X: Development-support product that has not yet completed Texas Instruments internal qualification testing.

null: Fully-qualified development-support product.

XMS devices and X development-support tools are shipped against the following disclaimer:

"Developmental product is intended for internal evaluation purposes."

MSP devices have been characterized fully, and the quality and reliability of the device have been demonstrated fully. TI's standard warranty applies.

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

TI device nomenclature also includes a suffix with the device family name. This suffix indicates the temperature range, package type, and distribution format.
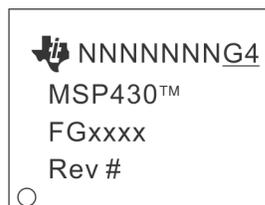
5.2 Package Markings

ZQW113 **BGA (ZQW), 113 Pin**

```
MSP430™
FGxxxx
NNNNNNN #
TI G1
○
```

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

PZ100 **LQFP (PZ) 100 Pin**

```
TI NNNNNNNG4
MSP430™
FGxxxx
Rev #
○
```

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

ZCA113   *NFBGA (ZCA), 113 Pin*

```
MSP430™

xxxx

NNNNNNN #

TI  G1
○
```

xxxx = Part number
N   = Lot trace code
= Die revision
○   = Pin 1 location

5.3 Memory-Mapped Hardware Revision (TLV Structure)

Die Revision	TLV Hardware Revision
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

BSL14	*BSL Module*
Category	Software in ROM
Function	BSL request to unlock the JTAG
Description	The feature in the BSL to keep the JTAG unlocked by setting the bit BSL_REQ_JTAG_OPEN in the return value has been disabled in this device.
Workaround	None

COMP10	*COMP Module*
Category	Functional
Function	Comparator port output toggles when entering or leaving LPM3/LPM4
Description	The comparator port pin output (CECTL1.CEOUT) erroneously toggles when device enters or leaves LPM3/LPM4 modes under the following conditions: 1) Comparator is disabled (CECTL1.CEON = 0) AND 2) Output polarity is enabled (CECTL1.CEOUTPOL = 1) AND 3) The port pin is configured to have CEOUT functionality. For example, if the CEOUT pin is high when the device is in Active Mode, CEOUT pin becomes low when the device enters LPM3/LPM4 modes.
Workaround	When the comparator is disabled, ensure at least one of the following: 1) Output inversion is disabled (CECTL.CEOUTPOL = 0) OR 2) Change pin configuration from CEOUT to GPIO with output low.

CPU21	*CPU Module*
Category	Compiler-Fixed
Function	Using POPM instruction on Status register may result in device 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 Status Register. Refer to the table below for compiler-specific fix implementation information.

CPU21 (continued) *CPU Module*

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 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=CPU22
MSP430 GNU Compiler (MSP430-GCC)	MSP430-GCC 4.9 build 167 or later	

CPU37 *CPU Module*

Category Functional

Function Wrong program trace display in the debugger while using conditional jump instructions

Description The state storage window displays an incorrect sequence of instructions when:

1. Conditional jump instructions are used to form a software loop

AND

2. A false condition on the jump breaks out of the loop

In such cases the trace buffer incorrectly displays the first instruction of the loop as the instruction that is executed immediately after exiting the loop.

Example:

CPU37 (continued)　　*CPU Module*

Actual Code:
mov #4,R4
LABEL mov #1,R5
dec R4
jnz LABEL
mov #2,R6
nop

State Storage Window Displays:
LABEL mov #1,R5
dec R4
jnz LABEL
mov #1,R5
nop

Workaround

None

Note: This erratum affects the trace buffer display only. It does not affect code execution in debugger or free run mode

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

CPU40 (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=CPU40
MSP430 GNU Compiler (MSP430-GCC)	Not affected	

CPU47   *CPU Module*

Category   Functional

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.

CTSD1   *CTSD Module*

Category   Functional

Function   CTSD16OFFG bit erroneously set while CTSD16 module is inactive

Description   The CTSD16CTL.CTSD16OFFG bit is erroneously set when the CTSD16 module is disabled and not actively converting (CTSD16CCTLx.CTSD16SC = 0). This CTSD16CTL.CTSD16OFFG bit can only be cleared once the CTSD16 module is enabled and actively converting (CTSD16CCTLx.CTSD16SC = 1).

This errata effectively nullifies the ability to trigger NMI interrupts in response to oscillator faults, unless CTSD16 is kept enabled.

Workaround   1) If CTSD16 is enabled, and the fault condition is ensured not to be present, then CTSD16OFFG and OFIFG can function normally. The only way to keep CTSD16 enabled indefinitely is by setting CTSD16SC.

2) While CTSD16 is not enabled, the OFIFG bit cannot be used. The other bits sourcing into it besides CTSD16OFFG (that is, XT1LFOFFG, XT1HFOFFG, XT2OFFG, and DCOFFG) can be polled or checked by software individually; but the ability to trigger the NMI upon OFIFG becoming set is no longer possible.

DAC5   *DAC Module*

Category   Functional

Function   Switching events on alternative DAC output pins can change output level of active DAC output pin.

DAC5 (continued)	*DAC Module*

Description

When the DAC output has multiple pin output options, switching events on unused alternate pin output options, such as GPIO output voltage level transitions and pulses or input signals that pass GPIO high or low thresholds, dynamically affect the output voltage level of the DAC output in use. This effect only lasts for a short period after the switching event.

Workaround

If the dynamic voltage level effect on the chosen DAC output cannot be tolerated, see the workarounds below:
A. Avoid high to low or low to high transitions on the alternate DAC output pin when selected DAC output pin is active. Keep alternate pin pulled high or low when DAC is in use.
B. Do not use pin featuring alternative DAC output, and keep in an output low state.

DMA4	*DMA Module*

Category

Functional

Function

Corrupted write access to 20-bit DMA registers

Description

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

Workaround

1. Design the application to guarantee that no DMA access interrupts 20-bit wide accesses to the DMA address registers.

OR

2. When accessing the DMA address registers, enable the Read Modify Write disable bit (DMARMWDIS = 1) or temporarily disable all active DMA channels (DMAEN = 0).

OR

3. Use word access for accessing the DMA address registers. Note that this limits the values that can be written to the address registers to 16-bit values (lower 64K of Flash).

DMA7	*DMA Module*

Category

Functional

Function

DMA request may cause the loss of interrupts

Description

If a DMA request starts executing during the time when a module register containing an interrupt flags is accessed with a read-modify-write instruction, a newly arriving interrupt from the same module can get lost. An interrupt flag set prior to DMA execution would not be affected and remain set.

Workaround

1. Use a read of Interrupt Vector registers to clear interrupt flags and do not use read-modify-write instruction.

OR

2. Disable all DMA channels during read-modify-write instruction of specific module registers containing interrupts flags while these interrupts are activated.

DMA10	***DMA Module***
Category	Functional
Function	DMA access may cause invalid module operation
Description	The peripheral modules MPY, CRC, USB, RF1A and FRAM controller in manual mode can stall the CPU by issuing wait states while in operation. If a DMA access to the module occurs while that module is issuing a wait state, the module may exhibit undefined behavior.
Workaround	Ensure that DMA accesses to the affected modules occur only when the modules are not in operation. For example with the MPY module, ensure that the MPY operation is completed before triggering a DMA access to the MPY module.

EEM16	***EEM Module***
Category	Debug
Function	The state storage display does not work reliably when used on instructions with CPU Wait cycles.
Description	When executing instructions that require wait states; the state storage window updates incorrectly. For example a flash erase instruction causes the CPU to be held until the erase is completed i.e. the flash puts the CPU in a wait state. During this time if the state storage window is enabled it may incorrectly display any previously executed instruction multiple times.
Workaround	Do not enable the state storage display when executing instructions that require wait states. Instead set a breakpoint after the instruction is completed to view the state storage display.

Note

This erratum affects debug mode only.

EEM17	***EEM Module***
Category	Debug
Function	Wrong Breakpoint halt after executing Flash Erase/Write instructions
Description	Hardware breakpoints or Conditional Address triggered breakpoints on instructions that follow Flash Erase/Write instructions, stops the debugger at the actual Flash Erase/Write instruction even though the flash erase/write operation has already been executed. The hardware/conditional address triggered breakpoints that are placed on either the next two single opcode instructions OR the next double opcode instruction that follows the Flash Erase/Write instruction are affected by this erratum.
Workaround	None. Use other conditional/advanced triggered breakpoints to halt the debugger right after Flash erase/write instructions.

Note

This erratum affects debug mode only.

EEM19	***EEM Module***

Category Debug

Function DMA may corrupt data in debug mode

Description When the DMA is enabled and the device is in debug mode, the data written by the DMA may be corrupted when a breakpoint is hit or when the debug session is halted.

Workaround This erratum has been addressed in MSPDebugStack version 3.5.0.1. It is also available in released IDE EW430 IAR version 6.30.3 and CCS version 6.1.1 or newer.
If using an earlier version of either IDE or MSPDebugStack, do not halt or use breakpoints during a DMA transfer.

Note
This erratum applies to debug mode only.

EEM23	***EEM Module***

Category Debug

Function EEM triggers incorrectly when modules using wait states are enabled

Description When modules using wait states (USB, MPY, CRC and FRAM controller in manual mode) are enabled, the EEM may trigger incorrectly. This can lead to an incorrect profile counter value or cause issues with the EEMs data watch point, state storage, and breakpoint functionality.

Workaround None.

Note
This erratum affects debug mode only.

LCDB5	***LCDB Module***

Category Functional

Function Static DC charge can built up on dedicated COMx pins.

Description If the device is set into LPMx.5, its dedicated COMx pins (not shared with GPIO function) are floating. External leakage paths to these pins can result in dedicated COMx pins being charged. This can lead to static DC voltages being applied to the external LCD display. This might cause long term over-stress to the LCD display and/or cause certain LCD segments to flare up when device wakes up from LPMx.5 mode.

Workaround Connect a high-resistance resistor between the dedicated COM pins and Vss to permanently discharge the affected pins.

LCDB6	***LCDB Module***

Category Functional

Function LCD outputs may be corrupted by modifying register fields VLCDx and/or LCDCPEN of LCDCVCTL register while LCDON (LCDCCTL0) is set

LCDB6 (continued)   *LCDB Module*

Description	Writing to VLCDx and/or LCDCPEN register bits in LCDCVCTL register while LCDC is enabled (LCDON = '1' in LCDCCTL0 register) may corrupt the LCD output due to incorrect start-up of LCD-controller and internal voltage generation.
Workaround	Do not modify VLCDx and/or LCDCPEN bits in LCDCVCTL register while LCDON = '1'

PMM11   *PMM Module*

Category	Functional
Function	MCLK comes up fast on exit from LPM3 and LPM4
Description	The DCO exceeds the programmed frequency of operation on exit from LPM3 and LPM4 for up to 6 us. This behavior is masked from affecting code execution by default: SVSL and SVML run in normal-performance mode and mask CPU execution for 150 us on wakeup from LPM3 and LPM4. However ,when the low-side SVS and the SVM are disabled or are operating in full-performance mode (SVMLE= 0 and SVSLE= 0, or SVMLFP= 1 and SVSLFP= 1) AND MCLK is sourced from the internal DCO running over 4 MHz, 7 MHz,11 MHz,or 14 MHz at core voltage levels 0, 1, 2, and 3, respectively, the mask lasts only 2 us. MCLK is, therefore, susceptible to run out of spec for 4 us.
Workaround	Set the MCLK divide bits in the Unified Clock System Control 5 Register (UCSCTL5) to divide MCLK by two prior to entering LPM3 or LPM4 (set DIVMx= 001). This prevents MCLK from running out of spec when the CPU wakes from the low-power mode. Following the wakeup fromthe low-power mode, wait 32, 48, 80, or 100 cycles for core voltage levels 0, 1, 2, and 3, respectively, before resetting DIVM xto zero and running MCLK at full speed [for example, __delay_cycles(100)]

PMM12   *PMM Module*

Category	Functional
Function	SMCLK comesup fast on exit from LPM3 and LPM4
Description	The DCO exceeds the programmed frequency of operationon exit from LPM3 and LPM4 for up to 6 us. When SMCLK is sourced by the DCO, it is not masked on exit from LPM3 or LPM4. Therefore, SMCLK exceeds the programmed frequency of operation on exit from LPM3 and LPM4 for up to 6 us. The increased frequency has the potential to change the expected timing behavior of peripherals that select SMCLK as the clock source.
Workaround	- Use XT2 as the SMCLK oscillator source instead of the DCO or - Do not disable the clock request bit for SMCLKREQEN in the Unified Clock System Control 8 Register (UCSCTL8). This means that all modules that depend on SMCLK to operate successfully should be halted or disabled before entering LPM3 or LPM4. If the increased frequency prevents the proper function of an affected module, wait 32, 48, 80 or 100 cycles for core voltage levels 0, 1, 2, or 3, respectively, before re-enabling the module. (for example, __delay_cycles(100)

PMM14   *PMM Module*

Category	Functional

PMM14 (continued)	**PMM Module**

Function

Increasing the core level when SVS/SVM low side is configured in full-performance mode causes device reset

Description

When the SVS/SVM low side is configured in full performance mode (SVSMLCTL.SVSLFP = 1), the setting time delay for the SVS comparators is ~2us. When increasing the core level in full-performance mode; the core voltage does not settle to the new level before the settling time delay of the SVS/SVM comparator expires. This results in a device reset.

Workaround

When increasing the core level; enable the SVS/SVM low side in normal mode (SVSMLCTL.SVSLFP=0). This provides a settling time delay of approximately 150us allowing the core sufficient time to increase to the expected voltage before the delay expires.

PMM15	**PMM Module**

Category

Functional

Function

Device may not wake up from LPM2, LPM3, or LPM4

Description

Device may not wake up from LPM2, LPM3 or LMP4 if an interrupt occurs within 1 us after the entry to the specified LPMx; entry can be caused either by user code or automatically (for example, after a previous ISR is completed). Device can be recovered with an external reset or a power cycle. Additionally, a PUC can also be used to reset the failing condition and bring the device back to normal operation (for example, a PUC caused by the WDT).

This effect is seen when:
- A write to the SVSMHCTL and SVSMLCTL registers is immediately followed by an LPM2, LPM3, LPM4 entry without waiting the requisite settling time ((PMMIFG.SVSMLDLYIFG = 0 and PMMIFG.SVSMHDLYIFG = 0)).

or

The following two conditions are met:

- The SVSL module is configured for a fast wake-up or when the SVSL/SVML module is turned off. The affected SVSMLCTL register settings are shaded in the following table.

	SVSLE	SVSLMD	SVSLFP	AM, LPM0/1 SVSL state	Manual SVSMLACE = 0 LPM2/3/4 SVSL State	Automatic SVSMLACE = 1 LPM2/3/4 SVSL State	Wakeup Time LPM2/3/4
SVSL	0	x	x	OFF	OFF	OFF	$t_{WAKE-UP\ FAST}$
	1	0	0	Normal	OFF	OFF	$t_{WAKE-UP\ SLOW}$
	1	0	1	Full Performance	OFF	OFF	$t_{WAKE-UP\ FAST}$
	1	1	0	Normal	Normal	OFF	$t_{WAKE-UP\ SLOW}$
	1	1	1	Full Performance	Full Performance	Normal	$t_{WAKE-UP\ FAST}$

	SVMLE	SVMLFP		AM, LPM0/1 SVML state	Manual SVSMLACE = 0 LPM2/3/4 SVML State	Automatic SVSMLACE = 1 LPM2/3/4 SVML State	Wakeup Time LPM2/3/4
SVML	0	x		OFF	OFF	OFF	$t_{WAKE-UP\ FAST}$
	1	0		Normal	Normal	OFF	$t_{WAKE-UP\ SLOW}$
	1	1		Full Performance	Full Performance	Normal	$t_{WAKE-UP\ FAST}$

and

-The SVSH/SVMH module is configured to transition from Normal mode to an OFF

PMM15 (continued) *PMM Module*

state when moving from Active/LPM0/LPM1 into LPM2/LPM3/LPM4 modes. The affected SVSMHCTL register settings are shaded in the following table.

	SVSHE	SVSHMD	SVSHFP	AM, LPM0/1 SVSH state	Manual SVSMHACE = 0	Automatic SVSMHACE = 1
					LPM2/3/4 SVSH State	LPM2/3/4 SVSH State
SVSH	0	x	x	OFF	OFF	OFF
	1	0	0	Normal	OFF	OFF
	1	0	1	Full Performance	OFF	OFF
	1	1	0	Normal	Normal	OFF
	1	1	1	Full Performance	Full Performance	Normal

	SVMHE	SVMHFP	AM, LPM0/1 SVMH state	Manual SVSMHACE = 0	Automatic SVSMHACE = 1
				LPM2/3/4 SVMH State	LPM2/3/4 SVMH State
SVMH	0	x	OFF	OFF	OFF
	1	0	Normal	Normal	OFF
	1	1	Full Performance	Full Performance	Normal

Workaround

Any write to the SVSMxCTL register must be followed by a settling delay (PMMIFG.SVSMLDLYIFG = 0 and PMMIFG.SVSMHDLYIFG = 0) before entering LPM2, LPM3, LPM4.

and

1. Ensure the SVSx, SVMx are configured to prevent the issue from occurring by the following:

- Configure the SVSL module for slow wake up (SVSLFP = 0). Note that this will increase the wakeup time from LPM2/3/4 to twakeupslow (~150 us).

or

- Do not configure the SVSH/SVMH such that the modules transition from Normal mode to an OFF state on LPM entry and ensure SVSH/SVMH is in manual mode. Instead force the modules to remain ON even in LPMx. Note that this will cause increased power consumption when in LPMx.

Refer to the MSP430 Driver Library(MSPDRIVERLIB) for proper PMM configuration functions.
Use the following function, PMM15Check (void), to determine whether or not the existing PMM configuration is affected by the erratum. The return value of the function is 1 if the configuration is affected, and 0 if the configuration is not affected.

```
unsigned char PMM15Check (void)
{
// First check if SVSL/SVML is configured for fast wake-up
if ( (!(SVSMLCTL & SVSLE)) || ((SVSMLCTL & SVSLE) && (SVSMLCTL & SVSLFP)) ||
(!(SVSMLCTL & SVMLE)) || ((SVSMLCTL & SVMLE) && (SVSMLCTL & SVMLFP)) )
{ // Next Check SVSH/SVMH settings to see if settings are affected by PMM15
if ((SVSMHCTL & SVSHE) && (!(SVSMHCTL & SVSHFP)))
{
if ( (!(SVSMHCTL & SVSHMD)) || ((SVSMHCTL & SVSHMD) &&
(SVSMHCTL & SVSMHACE)) )
return 1; // SVSH affected configurations
```

PMM15 (continued)   *PMM Module*

```
}
if ((SVSMHCTL & SVMHE) && (!(SVSMHCTL & SVMHFP)) && (SVSMHCTL &
SVSMHACE))
return 1; // SVMH affected configurations
}
return 0; // SVS/M settings not affected by PMM15
}
}
```

2. If fast servicing of interrupts is required, add a 150us delay either in the interrupt service routine or before entry into LPM3/LPM4.

PMM18   *PMM Module*

Category	Functional
Function	PMM supply overvoltage protection falsely triggers POR
Description	The PMM Supply Voltage Monitor (SVM) high side can be configured as overvoltage protection (OVP) using the SVMHOVPE bit of SVSMHCTL register. In this mode a POR should typically be triggered when DVCC reaches ~3.75V. If the OVP feature of SVM high side is enabled going into LPM234, the SVM might trigger at DVCC voltages below 3.6V (~3.5V) within a few ns after wake-up. This can falsely cause an OVP-triggered POR. The OVP level is temperature sensitive during fail scenario and decreases with higher temperature (85 degC ~3.2V).
Workaround	Use automatic control mode for high-side SVS & SVM (SVSMHCTL.SVSMHACE=1). The SVM high side is inactive in LPM2, LPM3, and LPM4.

PMM20   *PMM Module*

Category	Functional
Function	Unexpected SVSL/SVML event during wakeup from LPM2/3/4 in fast wakeup mode
Description	If PMM low side is configured to operate in fast wakeup mode, during wakeup from LPM2/3/4 the internal VCORE voltage can experience voltage drop below the corresponding SVSL and SVML threshold (recommendation according to User's Guide) leading to an unexpected SVSL/SVML event. Depending on PMM configuration, this event triggers a POR or an interrupt.

> **Note**
> As soon the SVSL or the SVML is enabled in Normal performance mode the device is in slow wakeup mode and this erratum does not apply. In addition, this erratum has sporadic characteristic due to an internal asynchronous circuit. The drop of Vcore does not have an impact on specified device performance.

Workaround	If SVSL or SVML is required for application (to observe external disruptive events at Vcore pin) the slow wakeup mode has to be used to avoid unexpected SVSL/SVML events. This is achieved if the SVSL or the SVML is configured in "Normal" performance mode (not disabled and not in "Full" Performance Mode).

PMM26	*PMM Module*
Category	Functional
Function	Device lock-up if RST pin pulled low during write to SVSMHCTL or SVSMLCTL
Description	Device results in lock-up condition under one of the two scenarios below: 1) If RST pin is pulled low during write access to SVSMHCTL, with the RST/NMI pin is configured to reset function and is pulled low (reset event) the device will stop code execution and is continuously held in reset state. RST pin is no longer functional. The only way to come out of the lock-up situation is a power cycle. OR 2) If RST pin is pulled low during write access to SVSMLCTL and only if the code that checks for SVSMLDLYIFG==1 is implemented without a timeout. The device will be stuck in the polling loop polling since SVSMLDLYIFG will never be cleared.
Workaround	Follow the sequence below to prevent the lock-up for both use cases: 1) Disable RST pin reset function and switch to NMI before access SVSMHCTL or SVSMLCTL. then 2) Activate NMI interrupt and handle reset events in this time by SW (optional if reset functionality required during access SVSMHCTL or SVSMLCTL) then 3) Enable RST pin reset function after access to SVSMHCTL or SVSMLCTL To prevent lock-up caused by use case #2 a timeout for the SVSMLDLYIFG flag check should be implemented to 300us.

PORT15	*PORT Module*
Category	Functional
Function	In-system debugging causes the PMALOCKED bit to be always set
Description	The port mapping controller registers cannot be modified when single-stepping or halting at break points between a valid password write to the PMAPWD register and the expected lock of the port mapping (PMAP) registers. This causes the PMAPLOCKED bit to remain set and not clear as expected. Note: This erratum only applies to in-system debugging and is not applicable when operating in free-running mode.
Workaround	Do not single step through or place break points in the port mapping configuration section of code.

PORT19	*PORT Module*
Category	Functional
Function	Port interrupt may be missed on entry to LPMx.5
Description	If a port interrupt occurs within a small timing window (~1MCLK cycle) of the device entry into LPM3.5 or LPM4.5, it is possible that the interrupt is lost. Hence this interrupt will not trigger a wakeup from LPMx.5.
Workaround	None

RTC16	***RTC Module***
Category	Functional
Function	RTC_B module can seem stuck or function abnormally (jumping RTC)
Description	If VBAT and DVCC (VPRIM) power up slowly and cross around the VBAK switching threshold, internal functions may not reset properly. This can lead to a stuck RTC_B module or to unexpected functionality e.g. RTC_B is running faster which causes the observed time value to jump or skip forward.
Workaround	Prevent DVCC (VPRIM) and VBAT from crossing each other below 2V during power up. It does not matter which signal comes up first.

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.
Workaround	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.

UCS11	***UCS Module***
Category	Functional
Function	Modifying UCSCTL4 clock control register triggers an additional erroneous clock request
Description	Changing the SELM/SELS/SELA bits in the UCSCTL4 register will correctly configure the respective clock to use the intended clock source but might also erroneously set XT1/XT2 fault flag if the crystals are not present at XT1/XT2 or not configured in the application firmware. If the NMI interrupt for the OFIFG is enabled, an unintentional NMI interrupt will be triggered and needs to be handled.

> **Note**
> The XT1/XT2 fault flag can be set regardless of which SELM/SELS/SELA bit combinations are being changed.

Workaround	Clear all the fault flags in UCSCTL7 register once after changing any of the SELM/SELS/SELA bits in the UCSCTL4 register. If OFIFG-NMI is enabled during clock switching, disable OFIFG-NMI interrupt during

UCS11 (continued) **UCS Module**

changing the SELM/SELS/SELA bits in the UCSCTL4 register to prevent unintended NMI. Alternatively it can be handled accordingly (clear falsely set fault flags) in the Interrupt Service Routine to ensure proper OFIFG clearing.

USCI26 **USCI Module**

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

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)
{
```

USCI34 (continued)   *USCI Module*

```
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).

USCI39   *USCI Module*

Category	Functional
Function	USCI I2C IFGs UCSTTIFG, UCSTPIFG, UCNACKIFG
Description	Unpredictable code execution can occur if one of the hardware-clear-able IFGs UCSTTIFG, UCSTPIFG or UCNACKIFG is set while the global interrupt enable is set by software (GIE=1). This erratum is triggered if ALL of the following events occur in following order: 1. Pending Interrupt: One of the UCxIFG=1 AND UCxIE=1 while GIE=0 2. The GIE is set by software (e.g. EINT) 3. The pending interrupt is cleared by hardware (external I2C event) in a time window of 1 MCLK clock cycle after the "EINT" instruction is executed.
Workaround	Disable the UCSTTIE, UCSTPIE and UCNACKIE before the GIE is set. After GIE is set, the local interrupt enable flags can be set again. Assembly example: bic #UCNACKIE+UCSTPIE+UCSTTIE, UCBxIE ; disable all self-clearing interrupts NOP EINT bis #UCNACKIE+UCSTPIE+UCSTTIE, UCBxIE ; enable all self-clearing interrupts

---

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;
```

7 Revision History

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

IMPORTANT NOTICE AND DISCLAIMER