Errata

MSP430F5505 Microcontroller

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 F	Rev E	Rev D	Rev C
ADC30				✓
ADC31				✓
ADC39	✓	✓	✓	✓
ADC42	✓	✓	✓	✓
ADC69	✓	✓	✓	✓
CPU46	✓	✓	✓	✓
CPU47	✓	✓	✓	✓
DMA4	✓	✓	✓	✓
DMA7	✓	✓	✓	✓
DMA10	✓	✓	✓	✓
FLASH37				✓
MPY1	✓	✓	✓	✓
PMAP1	✓	✓	✓	✓
PMM9	✓	✓	✓	✓
PMM10				✓
PMM11	✓	✓	✓	✓
PMM12	✓	✓	✓	✓
PMM14	✓	✓	✓	✓
PMM15	✓	✓	✓	✓
PMM17				✓
PMM18	✓	✓	✓	✓
PMM20	✓	✓	✓	✓
PORT15	✓	✓	✓	✓
PORT16	✓	✓	✓	✓
PORT19	✓	✓	✓	✓
PORT30	✓	✓	✓	✓
RTC3	✓	✓	✓	✓
RTC6	✓	✓	✓	✓
SYS12	✓	✓	✓	✓
SYS14				✓
SYS16	✓	✓	✓	✓
SYS18	✓	✓	✓	✓
TAB23				✓
TB25	✓	✓	✓	✓
UCS6				✓
UCS7	✓	✓	✓	✓
UCS9	✓	✓	✓	✓
UCS10				✓
UCS11	✓	✓	✓	✓
USB8				✓
USB9	✓	✓	✓	✓
USB10	✓	✓	✓	✓

Errata Number	Rev F	Rev E	Rev D	Rev C
USB12	✓	✓	✓	✓
USCI26	✓	✓	✓	✓
USCI30				✓
USCI31	✓	✓	✓	✓
USCI34	✓	✓	✓	✓
USCI35	✓	✓	✓	✓
USCI39	✓	✓	✓	✓
USCI40	✓	✓	✓	✓
WDG4	✓	✓	✓	✓

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 F	Rev E	Rev D	Rev C
BSL6		✓	✓	
BSL7		✓	✓	✓
JTAG20	✓	✓	✓	✓

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 F	Rev E	Rev D	Rev C
EEM11	✓	✓	✓	✓
EEM13	✓	✓	✓	✓
EEM17	✓	✓	✓	✓
EEM19	✓	✓	✓	✓
EEM21	✓	✓	✓	✓
EEM23	✓	✓	✓	✓
JTAG26	✓	✓	✓	✓
JTAG27	✓	✓	✓	✓

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 F	Rev E	Rev D	Rev C
CPU21	✓	✓	✓	✓
CPU22	✓	✓	✓	✓
CPU39	✓	✓	✓	✓
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

RGZ48 ***QFN (RGZ), 48 Pin***

○
MSP430
Fxxxx
TI NNN #
NNNNG4

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

○
M430
Fxxxx
TI NNN #
NNNNG4

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

○
MSP430™
Fxxxx
TI NNN #
NNNN G4

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

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

5.3 Memory-Mapped Hardware Revision (TLV Structure)

Die Revision	TLV Hardware Revision
Rev F	15h
Rev E	14h
Rev D	13h
Rev C	12h

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

ADC30	***ADC Module***
Category	Functional
Function	ADC hangs when a slow ADC clock source is used
Description	When using ADC10SC bit to software trigger ADC conversions, the ADC10 state machine may hang (that is, no further interrupts are generated) if the ADC clock source (ADC10CLK) is significantly slower than the system clock (MCLK). This issue can be observed when ADC10CLK < MCLK/8. If the ADC conversions are re-triggered by setting the ADC10SC bit before eight MCLK cycles have elapsed since the ADCIFG interrupt bit is set or the ADC10BUSY bit is reset, then this behavior can be seen.
Workaround	- Use MODOSC or any clock that is > MCLK/8 as ADC10CLK or - When using a clock source < = MCLK/8 as ADC10CLK, ensure the application code provides a delay of at least one ADC10CLK cycle before setting the ADC10SC bit again.

ADC31	***ADC Module***
Category	Functional
Function	Sporadic occurrence of false ADC conversion results
Description	Sporadic errors in the ADC conversion results occur when the input voltage is an integer fraction of the reference voltage Vref. The probability of these sporadic errors is extremely low but possible. The false ADC results deviate from the actual input value depending on which integer fraction of Vref the input voltage is; for example: - With Vin= 1/2 Vref, the most significant bit of the ADC conversion result might be affected, thus affecting all the bits that follow the most significant bit and, therefore, a completely wrong reading that ranges anywhere in the ADC range might be read. - With Vin = 1/4 or 3/4 Vref, the second most significant bit along with all the bits that follow this bit are affected. In this case, the false reading is confined to one-half of the ADC range (as the first most significant bit value is correct). - With Vin = 1/8, 3/8, 5/8, or 7/8 Vref, the third most significant bit and all the bits that follow this bit are affected. The false reading is confined to one-fourth of the ADC range (as the first two most significant bit values are correct) - Similarly this behavior continues to the last bit.
Workaround	For measuring dc signals: Depending on the dc signal range being measured, discard the obvious outliers in measurement results and average out the false reading for errors in the lower bits. For measuring ac signals: If possible, use a higher over-sampling rate to average out the error readings.

ADC39	***ADC Module***
Category	Functional
Function	Erroneous ADC10 results in extended sample mode
Description	If the extended sample mode is selected (ADC10SHP = 0) and the ADC10CLK is asynchronous to the SHI signal, the ADC10 may generate erroneous results.

Workaround 1) Use the pulse sample mode (ADC10SHP=1)
OR
2) Use a synchronous clock for ADC10 and the SHI signal.

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

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

BSL6 — *BSL Module*

Category: Software in ROM

Function: USB BSL does not respond properly to suspend/reset events from the USB host

Description: The USB BSL in affected revisions contains an improper configuration of the USB module. As a result, errors might occur in response to suspend/reset events from the USB host. (Since enumeration of the USB device often involves suspend and/or reset events, an enumeration might trigger the failure.) If the failure occurs, the device becomes unresponsive to the USB host.

If the failure occurs, and if application code exists in main flash, a reset (BOR/POR/PUC) can be issued to switch execution away from the BSL, to the application. Given the same USB host/setup circumstances, the problem is likely to occur again on subsequent attempts.

Applications that do not use the USB BSL are unaffected.

Workaround: 1. The BSL can be updated via JTAG with a version that does not contain this bug. Use the code published in BSL documentation starting with version 00.07.85.36.

BSL7 — *BSL Module*

Category: Software in ROM

Function: BSL does not start after waking up from LPMx.5

Description: When waking up from LPMx.5 mode, the BSL does not start as it does not clear the Lock I/O bit (LOCKLPM5 bit in PM5CTL0 register) on start-up.

Workaround: 1. Upgrade the device BSL to the latest version (see Creating a Custom Flash-Based Bootstrap Loader (BSL) Application Note - SLAA450 for more details)
OR
2. Do not use LOCKLPM5 bit (LPMx.5) if the BSL is used but cannot be upgraded.

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.

IDE/Compiler	Version Number	Notes
IAR Embedded Workbench	Not affected	

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=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	

CPU39 — *CPU Module*

Category Compiler-Fixed

Function PC is corrupted when single-stepping through an instruction that clears the GIE bit

Description Single-stepping over an instruction that clears the General Interrupt Enable bit (for example DINT or BIC #GIE,SR) when the GIE bit was previously set may corrupt the PC. For example, the DINT or BIC #GIE,SR is a 2-byte instruction. Single stepping through this instruction increments the PC by a value of 4 instead of 2 thus corrupting the next PC value.

Note: This erratum applies to debug mode only.

Workaround Insert a NOP or __no_operation() intrinsic immediately after the line of code that clears the GIE bit.

OR

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

Note that compilers implementing the fix may lead to double stack usage when RET/RETA follows the compiler-inserted NOP.

IDE/Compiler	Version Number	Notes
IAR Embedded Workbench	IAR EW430 v5.60 until v6.20	User is required to add the compiler flag option below. --hw_workaround=CPU39 For the command line version add the following information Compiler: --core=430 Assembler:-v1
IAR Embedded Workbench	IAR EW430 v6.20 or later	Workaround is automatically enabled
TI MSP430 Compiler Tools (Code Composer Studio)	v4.1.3 or later	
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

IDE/Compiler	Version Number	Notes
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

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 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.

IDE/Compiler	Version Number	Notes
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*

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.

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.

EEM11 *EEM Module*

Category Debug

Function Conditional register write trigger fails while executing rotate instructions

Description A conditional register write trigger will fail to generate the expected breakpoint if the trigger condition is a result of executing one of the following rotate instructions: RRUM,RRCM, RRAM and RLAM.

Workaround None

Note

This erratum applies to debug mode only.

EEM13 *EEM Module*

Category Debug

Function Halting the debugger does not return correct PC value when in LPM

Description When debugging, if the device is in any low power mode and the debugger is halted, the program counter update by the debugger is corrupted. The debugger is unable to halt at the correct location.

Workaround None.

Note

This erratum applies to 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.

EEM21 — *EEM Module*

Category Debug

Function LPMx.5 debug limitations

Description Debugging the device in LPMx.5 mode might wake the device up from LPMx.5 mode inadvertently, and it is possible that the device enters a lock-up condition; that is, the device cannot be accessed by the debugger any more.

Workaround Follow the debugging steps in Debugging MSP430 LPM4.5 SLAA424 .

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.

FLASH37 *FLASH Module*

Category Functional

Function Corrupted flash read when SVM low-side flag is triggered

Description If the SVM low side is enabled, a change in the VCORE voltage level (an increase in the VCORE level) may cause the currently executed read operation from flash to be incorrect and may lead to unexpected code execution or incorrect data. This can happen under any one of the following conditions:

- When the VCORE is changed in application, the SVM low side is used to indicate if the core voltage has settled by using the SVMDLYIFG flag. The failure occurs only when a flash access is concurrent to the expiration of the settling time delay.

- Unexpected changes in the VCORE voltage level

For code examples and detailed guidance on the PMM operation and software APIs for PMM configuration see the driverlib APIs from 430Ware (MSP430Ware).

Workaround - Execute the procedure to change the VCORE level from RAM.

or

- If executing from flash, follow the procedure below when increasing the VCORE level.
Note: To apply this workaround, the SVM low-side comparator must operate in normal mode (SVMLFP = 0 in SVMLCTL).

```
// Set SVM highside to new level and check if a VCore increase is possible
SVSMHCTL = SVMHE | SVSHE | (SVSMHRRL0 * level);
// Wait until SVM highside is settled
while ((PMMIFG & SVSMHDLYIFG) == 0);
// Clear flag
PMMIFG &= ~SVSMHDLYIFG;

// Set also SVS highside to new level
// Vcc is high enough for a Vcore increase
SVSMHCTL |= (SVSHRVL0 * level);
// Wait until SVM highside is settled
while ((PMMIFG & SVSMHDLYIFG) == 0);
// Clear flag
PMMIFG &= ~SVSMHDLYIFG;

//*************flow change for errata workaround ************
// Set VCore to new level
PMMCTL0_L = PMMCOREV0 * level;
```

```
// Set SVM, SVS low side to new level
SVSMLCTL = SVMLE | (SVSMLRRL0 * level)| SVSLE | (SVSLRVL0 * level);
// Wait until SVM, SVS low side is settled
while ((PMMIFG & SVSMLDLYIFG) == 0);
// Clear flag
PMMIFG &= ~SVSMLDLYIFG;

//*************flow change for errata workaround ************
```

JTAG20 ***JTAG Module***

Category Software in ROM

Function BSL does not exit to application code

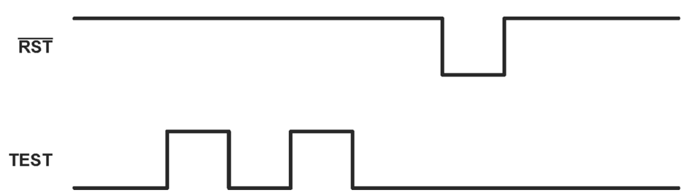
Description The methods used to exit the BSL per MSP430 Programming Via the Bootstrap Loader (SLAU319) are invalid.

Workaround To exit the BSL one of the following methods must be used.

- A Power cycle

or

- Toggle the TEST pin twice when nRST is high and after 50us pull nRST low.



Note: This toggling of TEST pins is not subject to timing constraints. The appropriate level transitions on TEST pin, followed by a RST pulse after 50us, are sufficient to trigger an exit from BSL mode.

JTAG26 ***JTAG Module***

Category Debug

Function LPMx.5 Debug Support Limitations

Description The JTAG connection to the device might fail at device-dependent low or high supply voltage levels if the LPMx.5 debug support feature is enabled. To avoid a potentially unreliable debug session or general issues with JTAG device connectivity and the resulting bad customer experience Texas Instruments has chosen to remove the LPMx.5 debug support feature from common MSP430 IDEs including TIs Code Composer Studio 6.1.0 with msp430.emu updated to version 6.1.0.7 and IARs Embedded Workbench 6.30.2, which are based on the MSP430 debug stack MSP430.DLL 3.5.0.1 http://www.ti.com/tool/MSPDS

TI plans to re-introduce this feature in limited capacity in a future release of the debug stack by providing an IDE override option for customers to selectively re-activate LPMx.5 debug support if needed. Note that the limitations and supply voltage dependencies outlined in this erratum will continue to apply.

For additional information on how the LPMx.5 debug support is handled within the MSP430 IDEs including possible workarounds on how to debug applications using LPMx.5 without toolchain support refer to Code Composer Studio User's Guide for MSP430 chapter F.4 and IAR Embedded Workbench User's Guide for MSP430 chapter 2.2.5.

Workaround

1. If LPMx.5 debug support is deemed functional and required in a given scenario:

a) Do not update the IDE to continue using a previous version of the debug stack such as MSP430.DLL v3.4.3.4.

OR

b) Roll back the debug stack by either performing a clean re-installation of a previous version of the IDE or by manually replacing the debug stack with a prior version such as MSP430.DLL v3.4.3.4 that can be obtained from http://www.ti.com/tool/MSPDS.

2. In case JTAG connectivity fails during the LPMx.5 debug mode, the device supply voltage level needs to be raised or lowered until the connection is working.

Do not enable the LPMx.5 debug support feature during production programming.

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.'

MPY1 — *MPY Module*

Category

Functional

Function

Save and Restore feature on MPY32 not functional

Description

The MPY32 module uses the Save and Restore method which involves saving the multiplier state by pushing the MPY configuration/operand values to the stack before using the multiplier inside an Interrupt Service Routine (ISR) and then restoring the state by popping the configuration/operand values back to the MPY registers at the end of the ISR. However due to the erratum the Save and Restore operation fails causing the write operation to the OP2H register right after the restore operation to be ignored as it is not preceded by a write to OP2L register resulting in an invalid multiply operation.

Workaround None. Disable interrupts when writing to OP2L and OP2H registers.

Note: When using the C-compiler, the interrupts are automatically disabled while using the MPY32

PMAP1 *PMAP Module*

Category Functional

Function Port Mapping Controller does not clear unselected inputs to mapped module.

Description The Port Mapping Controller provides the logical OR of all port mapped inputs to a module (Timer, USCI, etc). If the PSEL bit (PxSEL.y) of a port mapped input is cleared, then the logic level of that port mapped input is latched to the current logic level of the input. If the input is in a logical high state, then this high state is latched into the input of the logical OR. In this case, the input to the module is always a logical 1 regardless of the state of the selected input.

Workaround 1. Drive input to the low state before clearing the PSEL bit of that input and switching to another input source.

or

2. Use the Port Mapping Controller reconfiguration feature, PMAPRECFG, to select inputs to a module and map only one input at a time.

PMM9 *PMM Module*

Category Functional

Function False SVSxIFG events

Description The comparators of the SVS require a certain amount of time to stabilize and output a correct result once re-enabled; this time is different for the Full Performance versus the Normal mode. The time to stabilize the SVS comparators is intended to be accounted for by a built-in event-masking delay of 2 us when Full Performance mode is enabled. However, the comparators of the SVS in Full Performance mode take longer than 2 us to stabilize so the possibility exists that a false positive will be triggered on the SVSH or SVSL. This results in the SVSxIFG flags being set and depending on the configuration of SVSxPE bit a POR can also be triggered.
Additionally when the SVSxIFGs are set, all GPIOs are tri-stated i.e. floating until the SVSx comparators are settled.

The SVS IFG's are falsely set under the following conditions:

1. Wakeup from LPM2/3/4 when SVSxMD = 0 (default setting) && SVSxFP=1. The SVSx comparators are disabled automatically in LPM2/3/4 and are then re-enabled on return to active mode.

2. SVSx is turned on in full performance mode (SVSxFP=1).

3. A PUC/POR occurs after SVSx is disabled. After a PUC or POR the SVSx are enabled

automatically but the settling delay does not get triggered. Based on SVSxPE bit this may lead to POR events until the SVS comparator is fully settled.

Workaround For each of the above listed conditions the following workarounds apply:

1. If the Full Performance mode is to be enabled for either the high- or low-side SVS comparators, the respective SVSxMD bits must be set (SVSxMD = 1) such that the SVS comparators are not temporarily shut off in LPM2/3/4. Note that this is equivalent to a 2 uA (typical) adder to the low power mode current, per the device-specific datasheet, for each SVSx that remains enabled.

2. The SVSx must be turned on in normal mode (SVSxFP=0). It can be reconfigured to use full performance mode once the SVSx/SVMx delay has expired.

3. Ensure that SVSH and SVSL are always enabled.

PMM10 — *PMM Module*

Category Functional

Function SVS/SVM flags disabled after Power Up Clear reset

Description SVS/SVM interrupt flag functionality is disabled after a Power Up Clear (PUC) Reset if the SVS was disabled before the PUC reset was applied.

Workaround A write access to the intended SVSx register after PUC re-enables the SVS & SVM interrupt flags.

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

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.

SVSL	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
	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	$t_{WAKE-UP\ FAST}$
SVML	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
	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 state when moving from Active/LPM0/LPM1 into LPM2/LPM3/LPM4 modes. The affected SVSMHCTL register settings are shaded in the following table.

SVSH	SVSHE	SVSHMD	SVSHFP	AM, LPM0/1 SVSH state	Manual SVSMHACE = 0 LPM2/3/4 SVSH State	Automatic SVSMHACE = 1 LPM2/3/4 SVSH State
	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
SVMH	SVMHE	SVMHFP		AM, LPM0/1 SVMH state	Manual SVSMHACE = 0 LPM2/3/4 SVMH State	Automatic SVSMHACE = 1 LPM2/3/4 SVMH State
	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
}
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.

PMM17 *PMM Module*

Category Functional

Function Vcore exceed maximum limit of 2.0V.

Description If the device is switching between active mode and LPM2/3/4 with very high frequency, the core voltage of the device, VCORE, may rise incrementally until it is beyond 2.0 V, which is the maximum allowable limit for digital circuitry internal to the MSP430. This increase may remain undetected in an application with no functional impact but could potentially result in decreased endurance and increased wear over the lifetime of the device, because the digital circuitry is continually subjected to overvoltage.

The accumulation of Vcore affects only older lot trace codes of mentioned revisions.

Workaround The VCORE accumulation is fixed by enabling the prolongation mechanism in software. The following lines of code need to be implemented before periodic execution of LPM-to-AM-LPM. It is recommended to execute the code at program start:

ASM code:

```
mov.w #0x9602, &0110h;
bis.w #0x0800, &0112h;
```

C code:

```
*(unsigned int*)(0x0110)=0x9602;
*(unsigned int*)(0x0112)|=0x0800;
```

The automatic prolongation mechanism is disabled with a BOR and must be enabled after each boot code execution.

For detailed background information, affected LTCs and possible workaround(s) see Vcore Accumulation documentation in SLAA505.

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 as 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).

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.

PORT16 — *PORT Module*

Category Functional

Function GPIO pins are driven low during device start-up

Description During device start-up, all of the GPIO pins are expected to be in the floating input state. Due to this erratum, some of the GPIO pins are driven low for the duration of boot code execution during device start-up, if an external reset event (via the RST pin) interrupted the previous boot code execution. Boot code is always executed after a BOR, and the duration of this boot code execution is approximately 500us.

For a given device family, this erratum affects only the GPIO pins that are not available in the smallest package device family member, but that are present on its larger package variants.

Note

This erratum does not affect the smallest package device variants in a particular device family.

Workaround Ensure that no external reset is applied via the RST pin during boot code execution of the device, which occurs 1us after device start-up.

Note

System application needs to account for this erratum in to ensure there is no increased current draw by the external components or damage to the external components in the system during device start-up.

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

PORT30 — *PORT Module*

Category Functional

Function Port J pins pulled up during RST and BOR

Description Port J pins - PJ.1, PJ.2 and PJ.3 are unexpectedly pulled-up during a BOR or when a RST is active on the RSTNMI pin.
The pull-up on the impacted pins are de-activated when the device exits the reset state.
This issue does not occur if a BOR or RST takes place while the device is in LPM2 , LPM3 or LPM4.

Workaround 1) Use pins other than PJ.1, PJ.2 and PJ.3 for driving external components sensitive to voltage levels especially if BOR or RST events are expected in the application.

OR
2) If any of the three pins (PJ.1, PJ.2, and PJ.3) are used to drive external components sensitive to the pull-up, then use an external pull-down on these pins. Note that this will cause an increase in current consumption during the BOR or RST event.

RTC3	***RTC Module***
Category	Functional
Function	Unreliable write to RTC register
Description	A write access to the RTC registers (SEC, MIN, HOUR, DATE, MON, YEAR, DOW) may result in unexpected results. As a consequence the addressed register might not contain the written data, or some data can be accidentally written to other RTC registers.
Workaround	Use the RTC library routines, available as F541x/F543x code examples on the MSP430 Code Examples page (www.ti.com/msp430 > Software > Code Examples), which use carefully aligned MOV instructions. Library is listed as RTC_Workaround.zip and includes both CCE and IAR example projects that show proper usage. Using this library, full access to RTC registers is possible.

RTC6	***RTC Module***
Category	Functional
Function	the step size of the RTC frequency adjustment is twice the specified size.
Description	In BCD mode of operation, the step size of the RTC frequency adjustment is =+8ppm/-4ppm. This is twice the size specified in the User's Guide. In BCD mode, for up calibration this results in a step size per step of 8ppm (1024 cycles) instead of 4ppm (512 cycles). For down calibration this results in a step size per step of 4ppm (512 cycles) instead of 2ppm (256 cycles). In Binary mode, the step size = +4ppm/-2ppm as per the spec.
Workaround	In BCD mode of operation, half the calibration value could be written into RTCCAL register to compensate the doubled step size.

SYS12	***SYS Module***
Category	Functional
Function	Invalid ACCVIFG when DVcc in the range of 2.4 to 2.6V
Description	A Flash Access Violation Interrupt Flag (ACCVIFG) may be triggered by the Voltage Changed During Program Error bit (VPE) when DVcc is in the range of 2.4 to 2.6V. However the VPE does not signify an invalid flash operation has occurred. If the ACCVIE bit is set and a flash operation is executed in the affected voltage range, an unnecessary interrupt is requested. The bootstrap loader also cannot be used to execute write/erase flash operations in this voltage range, because it exits the flash operation and returns an error on an ACCVIFG event.
Workaround	None

SYS14	***SYS Module***
Category	Functional

Function Increased current consumption after a PUC

Description After a PUC, an increased current consumption is seen.

Note

This erratum ONLY applies for MSP430F53xx devices.

Workaround Insert the following memory initialization code at the beginning of the application firmware.

Assembly Initialization Code:

```
mov.w #0x9628, &0x0900
mov.w #0x0800, &0x0908
mov.w #0x9600, &0x0900
```

C Initialization Code:

```
unsigned int *Address = ((unsigned int*)INIT_MEMORY_ADDR);
*Address = 0x9628;
*(Address+4) = 0x0800;
*Address = 0x9600;
```

where INIT_MEMORY_ADDR is defined as:

```
#define INIT_MEMORY_ADDR 0x0900
```

SYS16 *SYS Module*

Category Functional

Function Fast Vcc ramp after device power up may cause a reset

Description At initial power-up, after Vcc crosses the brownout threshold and reaches a constant level, an abrupt ramp of Vcc at a rate dV/dT > 1V/100us can cause a brownout condition to be incorrectly detected even though Vcc does not fall below the brownout threshold. This causes the device to undergo a reset.

Workaround Use a controlled Vcc ramp to power up the device.

SYS18 *USB Module*

Category Functional

Function USB registers are unlocked and ACCVIFG is set at start-up

Description During device start-up, an incorrect line of code in the start-up code causes the USB registers to remain unlocked and causes an access violation, setting ACCVIFG bit.
In the BSL430_Low_Level_Init code, the following line of code accesses USBKEY (incorrect register address) instead of USBKEYPID, causing an access violation setting ACCVIFG bit, and leaving the USB registers unlocked.

```
mov.w #0x0000, &USBKEY ; lock USB
```

The correct line of code should read:

```
mov.w #0x0000, &USBKEYPID ; lock USB correctly
```

Note: This code does not run when using the JTAG debugger - the behavior only appears when running standalone.

Workaround

1. Load the latest version of the USB BSL from Custom BSL Download

OR

2. Load a non-USB or custom BSL

OR

3. Erase the BSL

OR

4. Clear the access violation flag at the beginning of the application code with the following C code (or its assembly equivalent):

```
USBKEYPID = 0;        // Lock USB correctly
FCTL3 = 0xA558;       // Clear violation flag
```

TAB23 — *TAB Module*

Category

Functional

Function

TAxR/TBxR read can be corrupted when TAxR/TBxR = TAxCCR0/TBxCCR0

Description

When a timer in Up mode is stopped and the counter register (TAxR/TBxR) is equal to the TAxCCR0/TBxCCR0 value, a read of the TAR/TBR register may return an unexpected result.

Workaround

1. Use 'Up/Down' mode instead of 'Up' mode

OR

2. In 'Up' mode, use the timer interrupt instead of halting the counter and reading out the value in TAxR/TBxR

OR

3. When halting the timer counter in 'Up' mode, reinitialize the timer before starting to run again.

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.

UCS6 — *UCS Module*

Category Functional

Function USCI source clock does not turn off in LPM3/4 when UART is idle

Description The USCI clock source (ACLK/SMCLK) remains enabled in LPM3 and LPM4 when the USCI is configured in UART mode and the communication is idle (UCSWRST = 0 but no TX or RX currently executing). This is contrary to the expected automatic clock activation described in the User's Guide and can lead to higher current consumption in low power modes, depending on the oscillator that feeds ACLK / SMCLK.

Workaround Use the oscillator that is already active in LPM3 (ACLK) to source the USCI and utilize the low-power baud rate generator (UCOS16 = 0). For UART baud rates where a fast SMCLK sourced by the internal DCO is required use LPM0 instead of LPM3.

UCS7 — *UCS Module*

Category Functional

Function DCO drifts when servicing short ISRs when in LPM0 or exiting active from ISRs for short periods of time

Description The FLL uses two rising edges of the reference clock to compare against the DCO frequency and decide on the required modifications to the DCOx and MODx bits. If the device is in a low power mode with FLL disabled (LPM0 with DCO not sourcing ACLK/SMCLK or LPM2, LPM3, LPM4 where SCG1 bit is set) and enters a state which enables FLL (enter ISR from LPM0/LPM2 or exit active from ISRs) for a period less than 3x reference clock cycles, then the FLL will cause the DCO to drift.
This occurs because the FLL immediately begins comparing an active DCO with its reference clock and making the respective modifications to the DCOx and MODx bits. If the FLL is not given sufficient time to capture a full reference clock cycle (2 x reference clock periods) and adjust accordingly (1 x reference clock period), then the DCO will keep drifting each time the FLL is enabled.

Workaround (1) If DCO is not sourcing ACLK or SMCLK in the application, use LPM1 instead of LPM0 to make sure FLL is disabled when interrupt service routine is serviced.
(2) When exiting active from ISRs, insert a delay of at least 3 x reference clock periods. To save on power budget, the 3 x reference clock periods could also be spent in LPM0 with TimerA or TimerB using ACLK/SMCLK sourced from DCO. This way, the FLL and DCO are still active in LPM0.

UCS9 — *UCS Module*

Category Functional

Function Digital Bypass mode prevents entry into LPM4

Description
When entering LPM4, if an external digital input applied to XT1 in HF mode or XT2 is not turned off, the PMM does not switch to low-current mode causing higher than expected power consumption.

Workaround
Before entering LPM4:
(1) Switch to a clock source other than external bypass digital input.
OR
(2) Turn off external bypass mode (UCSCTL6.XT1BYPASS = 0).

UCS10 *UCS Module*

Category
Functional

Function
Modulation causes shift in DCO frequency

Description
When the FLL is enabled, the DCO frequency can be tracked automatically by modifying the DCOx and MODx bits. The MODx bits switch between the frequency selected by the DCO bits and the next-higher frequency set by (DCO + 1). The erroneous behavior is seen when the FLL is tracking close to a DCO step boundary and the MOD counter is expected to rollover, but instead the DCO bits increment and the MOD bits decrement. This causes the DCO to shift by up to 12% and remain at an increased frequency until approximately 15 REFCLK cycles have elapsed. The frequency reverts to the expected value immediately afterward.

For example, the modulator moves from DCOx = n and MODx = 31 to DCOx = n + 1 and MODx = 30, causing a large increase in the DCO frequency.

Applications could be impacted as follows:
When using the DCO frequency for asynchronous serial communication and timer operation, the effect can be seen as corrupted data or incorrect timing events.

Workaround
(1) Turn off the FLL.

Or

(2) Implement a Software FLL, comparing the DCO frequency to a known reference such as REFO or LFXT1 using a timer capture and tuning the value of the DCO and MOD bits periodically.

Or

(3) Execute the following sequence in periodic intervals.

1. Disable peripherals sourced by the DCO such as UART and Timer.

2. Turn on the FLL.

3. Wait the worst case settling time of 32 X 32 X fFLLREFCLK to allow it to lock to the target frequency.

4. Turn off the FLL.

5. Compare the DCO frequency to a known reference such as REFO or LFXT1 using a timer capture.

- If the DCO frequency is higher than expected, repeat from step (2) until the frequency reaches to the expected range.

- Else proceed with code execution.

See the application report UCS10 Guidance SLAA489 for more detailed information regarding working with this erratum. This erratum does not affect proper operation of the CPU when MCLK = DCO/FLL and is set to the maximum clock frequency specified in the device datasheet.

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 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.

USB8 — *USB Module*

Category Functional

Function USB PLL may fail to initialize when DCO is not used

Description If the DCO is not used or does not source any clock in the system, the PLL may not initialize properly, leading to a "bus error" NMI when the CPU attempts to access USB memory. Under certain conditions, the PLL requires the DCO to be active in order to initialize properly. If any of the system clocks (MCLK, SMCLK, ACLK) are derived from the DCO during PLL initialization, this condition is satisfied and the PLL initializes in a stable manner.

Workaround - Configure one of the system clocks to use the DCO for normal operation. (In most applications, it's convenient to source MCLK from the DCO.)

or

- If it's necessary to use a clock configuration that doesn't include the DCO, then briefly activate it, then de-activate it. This should be done after setting the UPLLEN bit but before polling the USBPLLIR flags. The code below shows a simplified PLL initialization procedure. (In the MSP430 USB API Stacks, this is performed in USB_enable().)

```
USBPLLCTL |= UPLLEN; // Enable PLL

// Add the following code to enable the DCO, then immediately revert back
```

```
// to the original clock settings for the application. This is long
// enough to allow PLL initialization to proceed. The actual UCSCTL4
// settings depend on the application's clock requirements.

UCSCTL4 = SELA__REFOCLK + SELS__XT2CLK + SELM__DCOCLK; // Enable the
DCO
UCSCTL4 = SELA__REFOCLK + SELS__XT2CLK + SELM__XT2CLK; // Revert back

//Wait for the PLL to settle
do {
USBPLLIR = 0x0000; // Clear the flags
for (i =0; i < 400; i++); // Wait for flags to set if not stable yet
}while (USBPLLIR != 0);

// PLL is now stable
```

USB9 — *USB Module*

Category Functional

Function VBUS detection may fail after powerup

Description In rare cases, some USB-equipped MSP430 devices may experience a failure in the bandgap that aids in detecting the presence of 5V on the VBUS pin. Two primary effects of this are:
- The USBBGVBV bit fails to show the presence of a valid voltage on the VBUS pin.

and

- The USB LDOs fail to start.

Workaround This error state can be "reset" by clearing all the bits in the USBPWRCTL register, which disables the USB LDOs, among other actions. The bits can then be set again normally, and the device functions properly.

This has been added to the USB_Init() function in v3.11 and later of the MSP430 USB API. Therefore, this problem is automatically addressed in applications using the API.

However, if the integrated 3.3V USB LDO (the output of the VUSB pin) is used to power the devices's DVCC pin, as in many bus-powered applications, and if the rare bandgap error occurs, the CPU will fail to power up, because the USB LDO fails to operate. The problem might be resolved by cycling power to the VBUS pin; for example, if the end user responds to the failure by unplugging and re-plugging the USB cable. The bandgap failure is also known to occur more often with slow DVCC ramps >200ms; for example, when there is excessive capacitance on the DVCC pin, in excess of what the USB specification allows. However, the only sure way to prevent the problem from occuring in the first place is to avoid making DVCC power reliant on VUSB.

USB10 — *USB Module*

Category Functional

Function USB interface may begin to endlessly transmit to the USB host when a rare timing event occurs between the USB host and MSP430 software execution

Description When the host sends a SETUP packet for an IN transaction, the SETUPIFG bit always gets set by hardware, and the USB ISR is triggered. While SETUPIFG is high, the host's attempts to continue the transaction with IN packets are automatically NAKed.

When the SETUP packet has been decoded and the IN data prepared, the USB ISR clears the SETUPIFG bit. But if it happens to do so within the 2nd CRC bit of an IN packet from the host, the USB module enters an errant state and can begin to endlessly transmit to the host, irrespective of the protocol. The errant state can be cleared by resetting the module with the USB_EN bit; but there's no way for software to reliably detect the condition.

Since the 2nd CRC bit is only an 83ns window, the problem is extremely rare. However, since the timing of IN packets relative to their preceding SETUP packets can vary according to the host's timing, there's no way to ensure for certain that it will never happen.

Workaround

If the problem behavior occurs, and if the MSP430 is bus-powered, the user may naturally unplug/re-plug the devices USB connection. If this occurs, the behavior will be corrected because power to the MSP430 will be cycled. After this, its unlikely the problem will occur again soon, since the failure is usually rare.

The behavior can be prevented altogether by clearing the UBME bit immediately before clearing SETUPIFG, and setting it again immediately after:

```
USBIEPCNF_0 &= ~EPCNF_UBME; // Clear ME to gate off SETUPIFG clear event
USBOEPCNF_0 &= ~EPCNF_UBME; // Clear ME to gate off SETUPIFG clear event
USBIFG &= ~SETUPIFG; // clear the interrupt bit
USBIEPCNF_0 |= EPCNF_UBME; // Set ME to continue with normal operation
USBOEPCNF_0 |= EPCNF_UBME; // Set ME to continue with normal operation
```

This workaround is reliable and effective. However, as a side effect, it results in the creation of orphan tokens on the USB interface. Although the workaround is field-tested, and no problems have been reported with these orphan packets, it is recommended to use the workaround only if the errata behavior is problematic for the application in question.

USB12

USB Module

Category

Functional

Function

The 2nd byte of a slave-to-host transmission is sent twice.

Description

In extremely rare cases, when the USB module's PLL is disabled (by clearing the UPLLEN bit), the USB module can be placed into an undetermined state, resulting in an extra byte being sent to the host over the bus. The PLL is usually disabled by software when the USB module detects that the USB device has been suspended by the host. Suspend events can occur at any time, but are typically invoked during periods of inactivity.

Workaround

Once this error occurs, the USB module needs to be reset (by clearing the USBEN bit), and then the module can be re-initialized. For example, software can call the MSP430 USB API USB_disable() followed by USB_enable(). These actions are taken by the USB APIs when the user unplugs and replugs the USB cable, which is likely to happen when the user realizes the bus is no longer working.

If automatic detection of the error is required, then software on the host and device could implement a CRC check on the data payload (above the USB API) to detect the extra byte. If detected, software could then disable/re-enable the USB module. (The CRC inherent in the USB protocol calculates over the data packet, and thus cannot detect the erroneously added byte.)

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

USCI30

USCI Module

Category Functional

Function I2C mode master receiver / slave receiver

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

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

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

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

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

OR

b) In case the receive buffer cannot be read out in time, test the I2C clock line before the UCBxRXBUF is read out to ensure that the critical window has elapsed. This is done by checking if the clock line low status indicator bit UCSCLLOW is set for atleast three USCI bit clock cycles i.e. 3 X t(BitClock).

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

Code flow for workaround

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

USCI31 ***USCI Module***

Category Functional

Function Framing Error after USCI SW Reset (UCSWRST)

Description While receiving a byte over USCI-UART (with UCBUSY bit set), if the application resets the USCI module (software reset via UCSWRST), then a framing error is reported for the next receiving byte.

Workaround 1. If possible, do not reset USCI-UART during an ongoing receive operation; that is, when UCBUSY bit is set.

2. If the application software resets the USCI module (via the UCSWRST bit) during an ongoing receive operation, then set and reset the UCSYNC bit before releasing the software USCI reset.

Workaround code sequence:

```
bis #UCSWRST, &UCAxCTL1 ; USCI SW reset
;Workaround begins
bis #UCSYNC, &UCAxCTL0 ; set synchronous mode
bic #UCSYNC, &UCAxCTL0 ; reset synchronous mode
;Workaround ends

bic #UCSWRST, &UCAxCTL1 ; release USCI reset
```

USCI34 ***USCI Module***

Category Functional

Function I2C multi-master transmit may lose first few bytes.

Description In an I2C multi-master system (UCMM =1), under the following conditions:

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

AND

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

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

Workaround

Verify that the START condition has been sent (UCTXSTT =0) before loading TXBUF with data.

Example:

```
#pragma vector = USCIAB0TX_VECTOR
__interrupt void USCIAB0TX_ISR(void)
{
// Workaround for USCI34
if(UCB0CTL1&UCTXSTT)
{
// TXData = pointer to the transmit buffer start
// PTxData = pointer to transmit in the ISR
PTxData = TXData; // restore the transmit buffer pointer if the Start bit is set
}
//
if(IFG2&UCB0TXIFG)
{
if (PTxData < = PTxDataEnd) // Check TX byte counter
{
UCB0TXBUF = *PTxData++; // Load TX buffer
}
else
{
UCB0CTL1 |= UCTXSTP; // I2C stop condition
IFG2 &= ~UCB0TXIFG; // Clear USCI_B0 TX int flag
__bic_SR_register_on_exit(CPUOFF); // Exit LPM0
}
}
}
```

USCI35

USCI Module

Category

Functional

Function

Violation of setup and hold times for (repeated) start in I2C master mode

Description

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

Workaround

If using repeated start, ensure SCL clock frequencies is < 50kHz in I2C standard mode (100 kbps).

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;
```

WDG4 *WDG Module*

Category Functional

Function The WDT failsafe can be disabled

Description The UCS is capable of masking clock requests (ACLK, SMCLK, MCLK) from peripheral modules; see request enable (REQEN) bits in the UCS control register, UCSCTL8.

The clock request logic of the UCS is used by the WDT module to ensure a fail-safe clock source in all low-power modes. Therefore, de-asserting the request enable bit of the watchdog clock source (xCLKREQEN = 0) allows the respective clock to be disabled upon entry into a low-power mode. Without an active clock source, the WDT timer stops incrementing and a watchdog event will not occur.

Workaround None

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	Page
• TB25 was added to the errata documentation	7
• TB25 Description was updated	28
• TB25 Workaround was updated	28

IMPORTANT NOTICE AND DISCLAIMER