CC430F5137 Device Erratasheet

1 Revision History

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

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

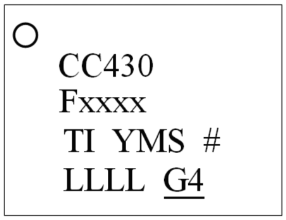
Errata Number	Rev E
ADC24	✓
ADC25	✓
ADC29	✓
ADC42	✓
ADC27--CC430	✓
AES1	✓
BSL7	✓
COMP4	✓
COMP10	✓
CPU18	✓
CPU20	✓
CPU21	✓
CPU22	✓
CPU23	✓
CPU24	✓
CPU25	✓
CPU26	✓
CPU27	✓
CPU28	✓
CPU29	✓
CPU30	✓
CPU31	✓
CPU32	✓
CPU33	✓
CPU34	✓
CPU35	✓
CPU39	✓
CPU40	✓
CPU46	✓
DMA4	✓
DMA7	✓
DMA8	✓
DMA10	✓
EEM8	✓

Errata Number	Rev E
EEM9	✓
EEM11	✓
EEM13	✓
EEM14	✓
EEM16	✓
EEM17	✓
EEM19	✓
EEM23	✓
FLASH29	✓
FLASH31	✓
FLASH37	✓
JTAG20	✓
JTAG26	✓
JTAG27	✓
MPY1	✓
PMAP1	✓
PMM8	✓
PMM9	✓
PMM10	✓
PMM11	✓
PMM12	✓
PMM14	✓
PMM15	✓
PMM17	✓
PMM18	✓
PMM20	✓
PORT15	✓
PORT16	✓
PORT17	✓
PORT19	✓
PORT21	✓
RF1A1	✓
RF1A2	✓
RF1A3	✓
RF1A5	✓
RF1A6	✓
RF1A8	✓
RTC3	✓
RTC6	✓
SYS16	✓
TAB23	✓
UCS6	✓
UCS7	✓
UCS9	✓
UCS10	✓
UCS11	✓
USCI26	✓
USCI30	✓

Errata Number	Rev E
USCI31	✓
USCI34	✓
USCI35	✓
USCI39	✓
USCI40	✓
WDG4	✓

2 Package Markings

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



TI = TI
YM = Year and Month Date Code
LLLL = Assembly Lot Code
S = Assembly Site Code
# = DIE Revision
o = PIN 1

3 Memory-Mapped Hardware Revision (TLV Structure)

Die Revision	TLV Hardware Revision
Rev E	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.

4 Detailed Bug Description

ADC24 *ADC12_A Module*

Function Unexpected ADC12 current draw when ADC12ENC = 1

Description When set, the ADC12ENC bit issues a clock request to the selected source clock, even before the conversion trigger. This causes some extra current consumption, depending on the selected clock.

Workaround None.

ADC25 *ADC12_A Module*

Function Write to ADC12CTL0 triggers ADC12 when CONSEQ = 00

Description If ADC conversions are triggered by the Timer_B module and the ADC12 is in single-channel single-conversion mode (CONSEQ = 00), ADC sampling is enabled by write access to any bit(s) in the ADC12CTL0 register. This is contrary to the expected behavior that only the ADC12 enable conversion bit (ADC12ENC) triggers a new ADC12 sample.

Workaround When operating the ADC12 in CONSEQ=00 and a Timer_B output is selected as the sample and hold source, temporarily clear the ADC12ENC bit before writing to other bits in the ADC12CTL0 register. The following capture trigger can then be re-enabled by setting ADC12ENC = 1.

ADC29 *ADC12_A Module*

Function Incorrect temperature sensor calibration data

Description In some devices, the internal temperature sensor calibration data for 30 degC are invalid for all VRef conditions. Devices with correct calibration data show a difference of at least 30 LSBs between the different VRef conditions. When using incorrect calibration data with the internal temperature sensor ADC samples, the calculated results can be unreliable. Calibration data for 85 degC are not affected.

This erratum affects devices with Lot Trace Code dated prior to 01/2011.

Workaround Recalibrate the temperature sensor for 30 degC at the application level.

ADC42 *ADC12_A Module*

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

ADC27--CC430 *ADC12_A Module*

Function

Integral and differential non-linearity exceed specifications

Description

The ADC12_A integral and differential non-linearity may exceed the limits specified in the data sheet under the following conditions:

- If the internal voltage reference generator is used

and

- If the reference voltage is not buffered off-chip

and

- If fADC12CLK > 2.7 MHz

or

If the internal voltage reference is selected for 1.5-V output mode.

The non-linearity can be up to tens of LSBs. This is due to the internal reference buffer providing insufficient drive for the switched capacitor array of the ADC12_A.

Workaround

- Turn on the output of the internal voltage reference to increase the drive strength of the reference to the ADC_12 core:

- If REFMSTR bit in REFCTL0 is 0 (allowing Shared REF to be controlled by ADC_A reference control bits)

Set ADC12REFON bit in ADC12CTL0 = 1

and

Set ADC12REFOUT bit in ADC12CTL2 = 1

- If REFMSTR bit in REFCTL0 is 1

Set REFON and REFOUT bits in REFCTL0 = 1

or

- Ensure fADC12CLK < 2.7 MHz and select the internal voltage reference in 2.5-V output mode.

Depending on the frequency of the source of fADC12CLK (ACLK, MCLK, SMCLK, or MODOSC), select the divider bits accordingly.

- If fADC12CLK = MODOSC (ADC12OSC)

ADC12CTL1 |= ADC12DIV_2; // Divide clock by 2

- If fADC12CLK = ACLK/SMCLK/MCLK > 2.7 MHz

Use ADC12DIVx and/or ADC12PDIVx bits to reduce the selected clock frequency to between 0.45 MHz and 2.7 MHz. And set both REFVSELx bits in REFCTL0 to REFVSEL_3 (select 2.5-V output).

AES1 — *AES Module*

Function Ongoing AES operation cannot be aborted by writing to AESAXIN

Description Writing to AESAXIN register when AESASTAT.AESBUSY bit is set does abort the ongoing AES operation or set the AESACTL0.AESERRFG bit.

Workaround Always let AES operation run to completion (i.e. do not abort). Ignore the encryption/decryption output if AESAXIN is written when AESASTAT.AESBUSY is set.

BSL7 — *BSL Module*

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.

COMP4 — *COMP_B Module*

Function CBEX and CBOUTPOL bits do not invert comparator I/O

Description Setting the exchange bit, CBEX, does not interchange the comparator inputs. Similarly setting the output polarity bit, CBOUTPOL, does not invert the output of the comparator.

Workaround To obtain an inverted output from the comparator, invert the input signals to the comparator using the channel input selector bits, CBIPSEL_x and CBIMSEL_x. Make sure to use a MOV instruction so that the inputs are inverted simultaneously.

COMP10 — *COMP_B Module*

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.

CPU18 *CPUXv2 Module*

Function

LPM instruction can corrupt PC/SR registers

Description

The PC and SR registers have the potential to be corrupted when:

- An instruction using register, absolute, indexed, indirect, indirect auto-increment, or symbolic mode is used to set the LPM bits AND (e.g. BIS &xyh, SR)

- This instruction is followed by a CALL or CALLA instruction

Upon servicing an interrupt service routine, the program counter (PC) is pushed twice onto the stack instead of the correct operation where the PC, then the SR registers are pushed onto the stack. This corrupts the SR and possibly the PC on RETI from the ISR.

Workaround

Insert a NOP or __no_operation() intrinsic function between the instruction to enter low power mode and the CALL or CALLA instruction.

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

IDE/Compiler	Version Number	Notes
IAR Embedded Workbench	IAR EW430 v6.20.1 until v6.40	User is required to add the compiler or assembler flag option below. --hw_workaround=nop_after_lpm
IAR Embedded Workbench	IAR EW430 v6.40 or later	Workaround is automatically enabled
TI MSP430 Compiler Tools (Code Composer Studio)	v4.0 or later	User is required to add the compiler or assembler flag option below. --silicon_errata=CPU18
MSP430 GNU Compiler (MSP430-GCC)	Not affected	

CPU20 *CPUXv2 Module*

Function

An unexpected Vacant Memory Access Flag (VMAIFG) can be triggered due to the CPU autoincrement of the MAB+2 outside the range of a valid memory block.

Description

The VMAIFG can be triggered under the following conditions:

1. If an interrupt is requested, fetched by the CPU, but lost before excution of the interrupt service routine.

OR

2. If a PC-modifying instruction (e.g. - ret, push, call, pop, jmp, br) is fetched from the last address of a section of memory (e.g.- FLASH, RAM) that is not continguous to a higher, valid section on the memory map.

Workaround

For case 1 - None.

For case 2 - If code is affected, edit the linker command file to make the last four bytes of affected memory sections unavailable.

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

IDE/Compiler	Version Number
IAR Embedded Workbench	IAR EW430 v6.40 or later
TI MSP430 Compiler Tools (Code Composer Studio)	Not affected
MSP430 GNU Compiler (MSP430-GCC)	Not affected

CPU21 *CPUXv2 Module*

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	
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 *CPUXv2 Module*

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	

CPU23 *CPUXv2 Module*

Function Rotate instruction does not function as expected

Description When repeated rotate instructions (rrcm, rram, rrum and rlam) are applied on the Program Counter(PC), unexpected instruction execution may occur.

Workaround

Insert a NOP instruction between sequential rotate instructions performed on the PC register.

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

IDE/Compiler	Version Number	Notes
IAR Embedded Workbench	Not affected	
TI MSP430 Compiler Tools (Code Composer Studio)	v4.0.x or later	User is required to add the compiler or assembler flag option below. --silicon_errata=CPU23
MSP430 GNU Compiler (MSP430-GCC)	MSP430-GCC 4.9 build 167 or later	

CPU24 — *CPUXv2 Module*

Function

Program counter corruption following entry into low power mode

Description

The program counter is corrupted when an interrupt event occurs in the time between (and including) one cycle before and one cycle after the CPUOFF bit is set in the status register. This failure occurs when the BIS instruction is followed by a CALL or CALLA instruction using the following addressing modes:

BIS &, SR

CALLA indir, indir autoinc, reg

BIS INDEX, SR

CALLA indir, indir autoinc, reg

BIS reg, SR

CALLA reg, indir, indir autoinc

NOTE: Due to the instruction emulation, the EINT instruction, as well as the __enable_interrupts() and possibly the __bis_SR_register() intrinsic functions are affected.

Workaround

Insert a NOP instruction or __no_operation() intrinsic function call between the BIS and CALL or CALLA instructions.

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

IDE/Compiler	Version Number	Notes
IAR Embedded Workbench	IAR EW430 v6.20 until v6.40	User is required to add the compiler or assembler flag option below. --hw_workaround=nop_after_lpm
IAR Embedded Workbench	IAR EW430 v6.40 or later	Workaround is automatically enabled
TI MSP430 Compiler Tools (Code Composer Studio)	v4.1.3 or later	
MSP430 GNU Compiler (MSP430-GCC)	MSP430-GCC 4.9 build 167	

CPU25 — *CPUXv2 Module*

Function

DMA transfer does not execute during low power mode

Description

If the following instruction sequence is used ([] denotes an addressing mode) to enter a low-power mode, and the DMARMWDIS bit is set, then DMA transfers are blocked for the duration of the low-power mode.

BIS [register|index|absolute|symbolic],SR

CALLA [register]

Workaround

1. Insert a NOP instruction or __no_operation() intrinsic function call between the BIS and CALLA instructions

OR

2. Temporarily clear the DMARMWDIS bit when entering low power mode

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

IDE/Compiler	Version Number	Notes
IAR Embedded Workbench	IAR EW430 v6.20 until v6.40	User is required to add the compiler or assembler flag option below. --hw_workaround=nop_after_lpm
IAR Embedded Workbench	IAR EW430 v6.40 or later	Workaround is automatically enabled
TI MSP430 Compiler Tools (Code Composer Studio)	v4.1.3 or later	
MSP430 GNU Compiler (MSP430-GCC)	MSP430-GCC 4.9 build 167	

CPU26 *CPUXv2 Module*

Function

CALL SP instruction does not behave as expected

Description

The intention of the CALL SP instruction is to execute code from the stack, instead it skips the first piece of data (instruction) on the stack. The second piece of data at SP+2 is used as the first executable instruction.

Workaround

Write the op code for a NOP as the first instruction on the stack. Begin the intended subroutine at address SP + 2.

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

IDE/Compiler	Version Number
IAR Embedded Workbench	Not affected
TI MSP430 Compiler Tools (Code Composer Studio)	v4.1.3 or later
MSP430 GNU Compiler (MSP430-GCC)	Not affected

CPU27 *CPUXv2 Module*

Function

Program Counter (PC) is corrupted during the context save of a nested interrupt

Description

When a low power mode is entered within an interrupt service routine that has enabled nested interrupts (by setting the GIE bit), and the instruction that sets the low power mode is directly followed by a RETI instruction, an incorrect value of PC + 2 is pushed to the stack during the context save. Hence, the RETI instruction is not executed on return from the nested interrupt and the PC becomes corrupted.

Workaround

Insert a NOP or __no_operation() intrinsic function between the instruction that sets the lower power mode and the RETI instruction.

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

IDE/Compiler	Version Number	Notes
IAR Embedded Workbench	IAR EW430 v6.20 until v6.40	User is required to add the compiler or assembler flag option below. --hw_workaround=nop_after_lpm
IAR Embedded Workbench	IAR EW430 v6.40 or later	Workaround is automatically enabled
TI MSP430 Compiler Tools (Code Composer Studio)	v4.1.3 or later	
MSP430 GNU Compiler (MSP430-GCC)	MSP430-GCC 4.9 build 167	

CPU28 *CPUXv2 Module*

Function

PC is corrupted when using certain extended addressing mode combinations

Description

An extended memory instruction that modifies the program counter executes incorrectly when preceded by an extended memory write-back instruction under the following conditions:

First instruction:

2-operand instruction, extended mode using (register,index), (register,absolute), OR (register,symbolic) addressing modes

Second instruction:

2-operand instruction, extended mode using the (indirect,PC), (indirect auto-increment,PC), OR (indexed [with ind 0], PC) addressing modes

Example:

BISX.A R6,&AABCD

ANDX.A @R4+,PC

Workaround

1. Insert a NOP or a __no_operation() intrinsic function between the two instructions

Or

2. Do not use an extended memory instruction to modify the PC

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

IDE/Compiler	Version Number
IAR Embedded Workbench	Not affected
TI MSP430 Compiler Tools (Code Composer Studio)	v4.1.3 or later
MSP430 GNU Compiler (MSP430-GCC)	Not affected

CPU29 *CPUXv2 Module*

Function

Using a certain instruction sequence to enter low power mode(s) affects the instruction width of the first instruction in an NMI ISR

Description

If there is a pending NMI request when the CPU enters a low power mode (LPMx) using an instruction of Indexed source addressing mode, and that instruction is followed by a 20-bit wide instruction of Register source and destination addressing modes, the first instruction of the ISR is executed as a 20-bit wide instruction.

Example:

main:

...

MOV.W [indexed],SR ; Enter LPMx

MOVX.A [register],[register] ; 20-bit wide instruction

...

ISR_start:

MOV.B [indexed],[register] ; ERROR - Executed as a 20-bit instruction!

Note: [] indicates addressing mode

Workaround

1. Insert a NOP or a __no_operation() intrinsic function following the instruction that enters the LPMx using indexed addressing mode

OR

2. Use a NOP or a __no_operation() intrinsic function as first instruction in the ISR

OR

3. Do not use the indexed mode to enter LPMx

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

IDE/Compiler	Version Number	Notes
IAR Embedded Workbench	IAR EW430 v6.20 until v6.40	User is required to add the compiler or assembler flag option below. --hw_workaround=nop_after_lpm
IAR Embedded Workbench	IAR EW430 v6.40 or later	Workaround is automatically enabled
TI MSP430 Compiler Tools (Code Composer Studio)	v4.1.3 or later	
MSP430 GNU Compiler (MSP430-GCC)	MSP430-GCC 4.9 build 167	

CPU30 *CPUXv2 Module*

Function

ADDA, SUBA, CMPA [immediate],PC behave as if immediate value were offset by -2

Description

The extended address instructions ADDA, SUBA, CMPA in immediate addressing mode are represented by 4-bytes of opcode (see the MSP430F5xx Family User's Guide MSP430F5xx Family User's Guide for more details). In cases where the program counter (PC) is used as the destination register only 2 bytes of the current instruction's 4-byte opcode are accounted for in the PC value. The resulting operation executes as if the immediate value were offset by a value of -2.

Ideal: ADDA #Immediate-4, PC

...is equivalent to...

Actual: ADDA #Immediate-2, PC

** NOTE: The MOV instruction is not affected **

Workaround

1) Modify immediate value in software to account for the offset of 2.

OR

2) Use extended 20-bit instructions (addx.a, subx.a, cmpx.a).

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

IDE/Compiler	Version Number	Notes
IAR Embedded Workbench	IAR EW430 v5.30 or later	IDE-based usage enables the workaround automatically. When using the command line, user is required to add the option below: Linker: -D?CPU30_OFFSET=2
TI MSP430 Compiler Tools (Code Composer Studio)	v4.0 or later	
MSP430 GNU Compiler (MSP430-GCC)	Not affected	

CPU31 *CPUXv2 Module*

Function SP corruption

Description When the instruction PUSHX.A is executed using the indirect auto-increment mode with the stack pointer (SP) as the source register [PUSHX.A @SP+] the SP is consequently corrupted. Instead of decrementing the value of the SP by four, the value of the SP is replaced with the data pointed to by the SP previous to the PUSHX.A instruction execution.

Workaround None. Note that compilers will not generate a PUSHX.A instruction that involves the SP.

Refer to the table below for compiler-specific information.

IDE/Compiler	Version Number
IAR Embedded Workbench	Not affected
TI MSP430 Compiler Tools (Code Composer Studio)	Not affected
MSP430 GNU Compiler (MSP430-GCC)	Not affected

CPU32 *CPUXv2 Module*

Function CALLA PC executes incorrectly

Description When the instruction CALLA PC is executed, the program counter (PC) that is pushed onto the stack during the context save is incorrectly offset by a value of -2.

Workaround None. Note that compilers will not generate a CALLA PC instruction.

Refer to the table below for compiler-specific information.

IDE/Compiler	Version Number
IAR Embedded Workbench	Not affected
TI MSP430 Compiler Tools (Code Composer Studio)	Not affected
MSP430 GNU Compiler (MSP430-GCC)	Not affected

CPU33 *CPUXv2 Module*

Function CALLA [indexed] may corrupt the program counter

Description When the Stack Pointer (SP) is used as the destination register in the CALLA index(Rdst) instruction and is preceded by a PUSH or PUSHX instruction in any of the following addressing modes: Absolute, Symbolic, Indexed, Indirect register or Indirect auto increment, the "index" of the CALLA instruction is not sign extended to 20-bits and is always treated as a positive value. This causes the Program Counter to be set to a wrong address location when the index of the CALLA instruction represents a negative offset.

NOTE:

1. This erratum only applies when the instruction sequence is: PUSH or PUSHX followed by CALLA index(SP)

2. This erratum does not apply if the PUSH or PUSHX instruction is used in the Register or Immediate addressing mode

3. This erratum only applies when SP is used as the destination register in the CALLA index(Rdst) instruction

Workaround

Place a "NOP" instruction in between the PUSH or PUSHX and the CALLA index(SP) instructions.

NOTE: This bug has no compiler impact as the compiler will not generate a CALLA instruction that uses indexed addressing mode with the SP.

Refer to the table below for compiler-specific information.

IDE/Compiler	Version Number
IAR Embedded Workbench	Not affected
TI MSP430 Compiler Tools (Code Composer Studio)	Not affected
MSP430 GNU Compiler (MSP430-GCC)	Not affected

CPU34 *CPUXv2 Module*

Function

CPU may be halted if a conditional jump is followed by a rotate PC instruction

Description

If a conditional jump instruction (JZ, JNZ, JC, JNC, JN, JGE, JL) is followed by an Address Rotate instruction on the PC (RRCM, RRAM, RLAM, RRUM) and the jump is not performed, the CPU is halted.

Workaround

Insert a NOP between the conditional jump and the rotate PC instructions.

Refer to the table below for compiler-specific information.

IDE/Compiler	Version Number
IAR Embedded Workbench	Not affected
TI MSP430 Compiler Tools (Code Composer Studio)	Not affected
MSP430 GNU Compiler (MSP430-GCC)	Not affected

CPU35 *CPUXv2 Module*

Function

Instruction BIT.B @Rx,PC uses the wrong PC value

Description

The BIT(.B/.W) instruction in indirect register addressing mode uses the wrong PC value. This instruction is represented by 2 bytes of opcode. If the Program Counter (PC) is used as the destination register, the 2 opcode bytes of the current BIT instruction are not accounted for. The resulting operation executes the instruction using the wrong PC value and this affects the results in the Status Register (SR).

Workaround

None. Note that compilers will not generate a BIT instruction that uses the PC as an operand.

Refer to the table below for compiler-specific information.

IDE/Compiler	Version Number
IAR Embedded Workbench	Not affected
TI MSP430 Compiler Tools (Code Composer Studio)	Not affected
MSP430 GNU Compiler (MSP430-GCC)	Not affected

CPU39 *CPUXv2 Module*

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 *CPUXv2 Module*

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	
MSP430 GNU Compiler (MSP430-GCC)	Not affected	

CPU46 — *CPUXv2 Module*

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.

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

DMA4 — *DMA Module*

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*

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.

DMA8 — *DMA Module*

Function DMA can corrupt values on write-access to program stack

Description If the DMA controller makes a write access to the stack while executing one of the following instructions, the data that is written may be corrupted.

CALLA [REG | IDX | SYM | ABS | IND | INA | IMM]

PUSHX.A [IDX | SYM | ABS | IND | IMM | INA]

PUSHX.A [REG]

PUSHM.A [REG]

POPM.A [REG]

Note: [...] denotes an addressing mode

Workaround Do not declare function-scope variables. Declare all variables that are intended to be modified by the DMA as global- or file-scope such that they are allocated in the data section of RAM and not on the program stack.

DMA10 *DMA Module*

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.

EEM8 *EEM Module*

Function Debugger stops responding when using the DMA

Description In repeated transfer mode, the DMA automatically reloads the size counter (DMAxSZ) once a transfer is complete and immediately continues to execute the next transfer unless the DMA Enable bit (DMAEN) has been previously cleared. In burst-block transfer mode, DMA block transfers are interleaved with CPU activity 80/20% - of ten CPU cycles, eight are allocated to a block transfer and two are allocated for the CPU.

Because the JTAG system must wait for the CPU bus to be clear to halt the device, it can only do so when two conditions are met:

- Three clock cycles after any DMA transfer, the DMA is no longer requesting the bus.

and

- The CPU is not requesting the bus.

Therefore, if the DMA is configured to operate in the repeat burst-block transfer mode, and a breakpoint is set between the line of code that triggers the DMA transfers and the line that clears the DMAEN bit, the DMA always requests the bus and the JTAG system never gains control of the device.

Workaround When operating the DMA in repeat burst-block transfer mode, set breakpoint(s) only when the DMA transfers are not active (before the start or after the end of the DMA transfers).

EEM9 *EEM Module*

Function Combined triggers on the PUSH instruction may be missed

Description When the PUSH instruction is used in any addressing mode except register or immediate modes, a combined trigger may be missed when its conditions are defined by a PUSH instruction fetch and a successful match of the value being pushed onto stack.

Workaround None

EEM11 — *EEM Module*

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*

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.

EEM14 — *EEM Module*

Function Single-step or breakpoint on module registers with WAIT capability may not work

Description In debug mode, the CPU clock is driven independently from the wait inputs of device modules (i.e., MULT, USB, RF1A, CRC). As a result, an EEM halt on an access to the module data registers (breakpoint or single-step) may show incorrect results due to incomplete execution.

Workaround Do not single-step through a data register access that holds the CPU to provide a valid result. Place breakpoints after the affected register is accessed and sufficient clock cycles have been provided.

NOTE: This erratum applies to debug mode only.

EEM16 — *EEM Module*

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*

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*

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*

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.

FLASH29 — *FLASH Module*

Function
Read disturb due to emergency exit from write/erase Flash operation

Description
When a Flash write or erase is abruptly terminated, any further reliable reads from Flash are not guaranteed. The abrupt termination can occur as a result of the Emergency Exit bit (EMEX in FCTL3) being set. This forces a write or an erase operation to be terminated before normal completion.

Workaround
After setting EMEX = 1, wait for at least 100 us after a bank or mass erase and at least 6 us after a segment erase before Flash is accessed again.

FLASH31 — *FLASH Module*

Function
Interrupts not disabled during FLASH erase operation

Description
When a flash erase operation is in progress, interrupts are not automatically disabled. The CPU will always try to service the interrupt request, whether or not the flash is busy.

Workaround
Disable interrupts using the GIE bit before erasing flash in another bank of memory. Note that all interrupts during this period of time will remain pending until GIE = 1.

FLASH37 — *FLASH Module*

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*

Function BSL does not exit to application code

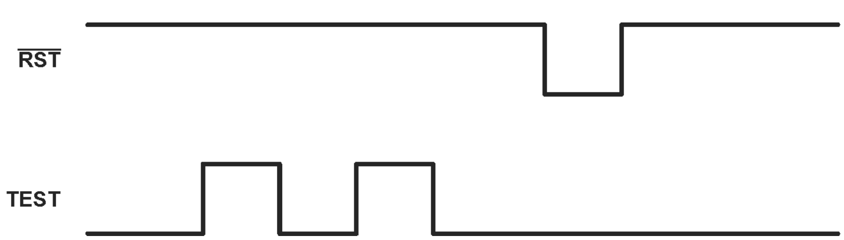
Description The methods used to exit the BSL per MSP430 Programming Via the Bootstrap Loader (SLAU319) are invalid.

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*

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*

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*

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*

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.

PMM8 *PMM Module*

Function Supply current in LPM4.5 is unpredictable

Description Due to an unpredictable value of the supply current in LPM4.5, the mode should not be used.

Workaround None.

PMM9 *PMM Module*

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*

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*

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 from the low-power mode, wait 32, 48, 80, or 100 cycles for core voltage levels 0, 1, 2, and 3, respectively, before resetting DIVMx to zero and running MCLK at full speed [for example, __delay_cycles(100)].

PMM12 *PMM Module*

Function SMCLK 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. 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*

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*

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	$t_{WAKE-UP FAST}$
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.

	SVSHE	SVSHMD	SVSHFP	AM, LPM0/1 SVSH state	Manual SVSMHACE = 0 LPM2/3/4 SVSH State	Automatic SVSMHACE = 1 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
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*

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***

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***

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).

PORT15 ***PORT Module***

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*

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.

PORT17 *PORT Module*

Function Certain pins when subject to negative high current pulses may cause latch-up in adjacent pins.

Description Pins subject to negative high current pulses may cause latch-up in adjacent pins. The latch-up condition exists only if the adjacent pin configurations also referred to as 'affected-pin' configuration are one of the following:

(1) GPIO input driven high by an external source

(2) GPIO output driven high with Full Drive strength OR Reduced Drive strength settings

(3) Peripheral configuration where the peripheral drives pin high or causes pin to be driven high externally

The following affected-pin configurations will not sustain latch-up:

(1) GPIO input driven low

(2) GPIO output driven low

(3) Peripheral configuration where the peripheral drives pin low or causes pin to be driven low externally

(4) Peripheral configuration as LCD pin

Note that for affected-pin configurations with LCD functionality, the window of latch-up when the pin is driven being high still exists but is of extremely short duration and hence there is a low probability of latch-up occurrence.

Workaround All affected pins must be driven low when not in use. If the affected pins are not driven low, then connecting a series resistor of 330 ohms to limit the latch-up current is recommended.

For more details on trigger currents, affected pin configurations and workarounds refer to the document PORT17 Guidance SLAA563

PORT19 *PORT Module*

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

PORT21 *PORT Module*

Function Setting PxSEL bit for XTAL pins

Description Setting the PxSEL bit of XIN pin does not disable the digital function of the XOUT pin (in non-bypass mode). The primary port function will still be active on the XOUT pin.

Workaround Set the PxSEL bit of XOUT pin explicitly to disable the port function of the XOUT pin.

RF1A1 *RF1A Module*

Function The PLL lock detector output is not 100% reliable

Description The PLL lock detector output is not 100% reliable and might toggle even if the PLL is in lock. The PLL is in lock if the lock detector output has a positive transition or is constantly logic high. The PLL is not in lock if the lock detector output is constantly logic low. It is not recommended to check for PLL lock by reading PKTSTATUS[0] with GDOx_CFG=0x0A or PKTSTATUS[2] register with GDOx_CFG=0x0A (x = 0 or 2).

Workaround PLL lock can be checked reliably by these methods:

- Program register IOCFGx.GDOx_CFG=0x0A and use the lock detector output available on the GDOx pin as an interrupt for the MCU. A positive transition on the GDOx pin means that the PLL is in lock. It is important to disable for interrupt when waking the chip from SLEEP state as the wake-up might cause the GDOx pin to toggle when it is programmed to output the lock detector.

or

- Read register FSCAL1. The PLL is in lock if the register content is different from 0x3F.

With both of the above workarounds the CC1101 PLL calibration should be carried out with the correct settings for TEST0.VCO_SEL_CAL_EN and FSCAL2.VCO_CORE_H_EN. These settings are depending on the operating frequency, and is calculated automatically by SmartRF Studio.

Note that the TEST0 register content is not retained in SLEEP state, and thus it is necessary to write to this register as described

above when returning from the SLEEP state.

RF1A2 *RF1A Module*

Function RXFIFO overflow flag does not work as intended

Description In addition to having a 64-byte long RX FIFO, the CC430 has a one byte long pre-fetch buffer between the FIFO and the RF1A module. It also has buffers for status registers and CRC bytes. If more than 65 bytes have been received (the FIFO and the pre-fetch buffer are full) without reading the RX FIFO, the radio will enter RXFIFO_OVERFLOW state. There are, however, some cases where the radio will be stuck in RX state instead of entering RXFIFO_OVERFLOW state. Below is a table showing the register settings that will cause this problem. APPEND_STATUS is found in the PKTCTRL1 register, and CRC_EN is found in the PKTCTRL0 register.

Setting IOCFGx=0x06 should mean that the GDO signal is deasserted when the RXFIFO overflows. In the cases where the radio is stuck in RX state, the GDOx pin will not be deasserted.

When the radio is stuck in this RX state it draws current as if it was in the RX state, but it will not be able to receive any more data. The only way to get out of this state is to issue an SIDLE strobe and then flush the FIFO (SFRX).

Workaround In applications where the packets are short enough to fit in the RX FIFO and one wants to wait for the whole packet to be received before starting to read the RX FIFO, for variable packet length mode (PKTCTRL0.LENGTH_CONFIG=1) the PKTLEN register should be set to 61 to make sure the whole packet including status bytes are 64 bytes or less (length byte (61) + 61 payload bytes + 2 status bytes = 64 bytes) or PKTLEN = 62 if fixed packet length mode is used (PKTCTRL0.LENGTH_CONFIG=0). In application where the packets do not fit in the RX FIFO, one must start reading the RX FIFO before it reaches its limit (64 bytes).

RF1A3 *RF1A Module*

Function Extra Byte Transmitted in TX

Description If a transmission is aborted (exits TX mode) during the transmission of the first half of any byte, there will be a repetition of the first byte in the next transmission. This issue is caused by a state machine controlling the mod_rd_data signal in the modulator. This signal asserts at the start of transmission of each full byte, then deasserts after half the byte has been transmitted. If the transmission is aborted after a byte has started but before half the byte is transmitted this signal remains asserted and the first byte in the next transmission is repeated.

Workaround As long as the packet handling features of the CC430 are used, this is not a problem since the chip always exits TX mode after the transmission of the last bit in the last byte of the packet. If, however, one disables the packet handling features (MDMCFG2.SYNC_MODE=0) and wants to exit TX mode manually by strobing IDLE, one should make sure that the IDLE strobe is being issued after clocking out 12 dummy bits (8 dummy bits are necessary due to the TX latency, but since this would mean that transmission is aborted within the first half of a byte, 4 extra bits are added).

RF1A5 *RF1A Module*

Function FIFO Radio Core Interrupt may be triggered independent of the RFINx condition being met

Description The radio core interrupt flags (RFIFGx) may be set and could generate a radio core interrupt although the corresponding radio input (RFINx) signal condition has not been met.

This is true for the FIFO Mapped Control Signals RFIFG3, RFIFG4, RFIFG5, RFIFG6, RFIFG7, RFIFG8, RFIFG9 (negative edge), and RFIFG10 (negative edge).

Workaround When handling the radio core interrupts RFIFG3 - RFIFG10, proceed with the ISR only after verifying that the RFINx signal respective to the RFIFGx flag is active.

RF1A6 — *RF1A Module*

Function LVERR flag set when radio in SLEEP or IDLE and VCORE = 0, 1

Description The low-voltage error flag (LVERR) is set when the radio is in the SLEEP or IDLE state and VCORE = 0 or 1, which is contrary to the behavior specified in the CC430 User's Guide.

Workaround None.

RF1A8 — *RF1A Module*

Function RF1AIN10 bit does not reset after the first byte of the RX FIFO is read

Description The intended behavior of RF1AIN10 bit is that it is set after the last byte is received [into RX FIFO] and reset after the first byte is read from the RX FIFO. However, the RF1AIN10 bit does not reset after the first byte of the RX FIFO is read.

Workaround Use RF1AIN9 for RX handling instead. To verify the RX packet CRC, enable the RF1A option to append the CRC_OK bit to the end of the RX packet. The CRC_OK bit can be checked after reading out the RX FIFO buffer.

RTC3 — *RTC Module*

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*

Function the step size of the RTC frequency adjustment is twice the specified size.

Description The step size of the RTC frequency adjustment is =+8ppm/-4ppm. This is twice the size specified in the User's Guide.

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).

Workaround Half the calibration value written into RTCCAL register to compensate the doubled step size.

SYS16 — *SYS Module*

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.

TAB23 *TIMER_A/TIMER_B Module*

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.

UCS6 *UCS Module*

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*

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*

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*

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*

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.

USCI26 — *USCI Module*

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*

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*

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*

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.

USCI35 *USCI Module*

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*

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*

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 — *WDT Module*

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

5 Document Revision History

Changes from family erratasheet to device specific erratasheet.

1. Errata JTAG21 was removed
2. Errata RTC4 was removed
3. Errata PORT16 was added
4. Errata UCS11 was added
5. Errata USCI31 was added
6. RGZ48 package markings have been updated

Changes from device specific erratasheet to document Revision A.

1. Errata PORT19 was added to the errata documentation.
2. Errata PMM18 was added to the errata documentation.
3. Errata RTC6 was added to the errata documentation.
4. Errata PORT17 was added to the errata documentation.

Changes from document Revision A to Revision B.

1. Errata DMA10 was added to the errata documentation.
2. Errata BSL7 was added to the errata documentation.
3. Errata RTC3 was added to the errata documentation.

Changes from document Revision B to Revision C.

1. DMA10 Description was updated.
2. DMA10 Function was updated.

Changes from document Revision C to Revision D.

1. DMA10 Description was updated.
2. MPY1 Description was updated.
3. Errata EEM23 was added to the errata documentation.
4. Errata CPU43 was added to the errata documentation.

Changes from document Revision D to Revision E.

1. SYS16 Description was updated.
2. CPU43 Description was updated.
3. Errata USCI34 was added to the errata documentation.
4. Errata PORT21 was added to the errata documentation.
5. Device TLV Hardware Revision information added to erratasheet.

Changes from document Revision E to Revision F.

1. Errata PMM20 was added to the errata documentation.
2. Errata USCI35 was added to the errata documentation.

Changes from document Revision F to Revision G.

1. BSL7 Workaround was updated.
2. BSL7 Function was updated.

Changes from document Revision G to Revision H.

1. Errata EEM19 was added to the errata documentation.
2. EEM13 Workaround was updated.
3. EEM23 Workaround was updated.
4. EEM17 Description was updated.
5. PORT17 Workaround was updated.
6. EEM23 Description was updated.

7. EEM17 Workaround was updated.
8. PORT16 Workaround was updated.
9. CPU43 Description was updated.
10. EEM11 Workaround was updated.
11. EEM14 Workaround was updated.
12. EEM16 Description was updated.
13. EEM23 Function was updated.
14. PORT16 Description was updated.
15. EEM16 Workaround was updated.

Changes from document Revision H to Revision I.

1. DMA10 Workaround was updated.
2. DMA10 Description was updated.
3. DMA10 Function was updated.

Changes from document Revision I to Revision J.

1. CPU40 Workaround was updated.
2. EEM19 Workaround was updated.
3. Errata USCI39 was added to the errata documentation.
4. Package Markings section was updated.
5. EEM23 Workaround was updated.
6. EEM23 Description was updated.
7. Errata ADC42 was added to the errata documentation.
8. EEM23 Function was updated.
9. EEM19 Description was updated.

Changes from document Revision J to Revision K.

1. ADC29 Description was updated.

Changes from document Revision K to Revision L.

1. Errata USCI40 was added to the errata documentation.
2. Errata CPU43 was removed from the errata documentation.
3. DMA7 Workaround was updated.
4. DMA7 Description was updated.
5. PMM18 Workaround was updated.

Changes from document Revision L to Revision M.

1. DMA7 Workaround was updated.
2. EEM23 Description was updated.
3. DMA7 Description was updated.

Changes from document Revision M to Revision N.

1. USCI39 Description was updated.
2. Errata AES1 was added to the errata documentation.

Changes from document Revision N to Revision O.

1. Errata JTAG26 was added to the errata documentation.

Changes from document Revision O to Revision P.

1. EEM19 Workaround was updated.

Changes from document Revision P to Revision Q.

1. RTC6 Description was updated.

2. UCS11 Workaround was updated.
3. UCS11 Description was updated.
4. UCS11 Function was updated.

Changes from document Revision Q to Revision R.

1. Errata SYS12 was added to the errata documentation.
2. Errata JTAG27 was added to the errata documentation.
3. Errata COMP10 was added to the errata documentation.

Changes from document Revision R to Revision S.

1. JTAG20 Workaround was updated.
2. USCI39 Workaround was updated.
3. Errata SYS12 was removed from the errata documentation.
4. Errata CPU46 was added to the errata documentation.

Changes from document Revision S to Revision T.

1. CPU21 was added to the errata documentation.
2. CPU23 was added to the errata documentation.
3. CPU22 was added to the errata documentation.
4. Workaround for CPU20 was updated.
5. Workaround for CPU27 was updated.
6. Function for CPU26 was updated.
7. Description for CPU26 was updated.
8. Workaround for CPU26 was updated.
9. Workaround for CPU29 was updated.
10. Workaround for CPU28 was updated.
11. Workaround for CPU25 was updated.
12. Workaround for CPU24 was updated.
13. Workaround for PMM15 was updated.
14. Workaround for CPU30 was updated.
15. Workaround for CPU32 was updated.
16. Workaround for CPU31 was updated.
17. Workaround for CPU39 was updated.
18. Workaround for CPU34 was updated.
19. Workaround for CPU33 was updated.
20. Workaround for CPU35 was updated.
21. Workaround for CPU40 was updated.
22. Workaround for CPU46 was updated.
23. Workaround for CPU18 was updated.

Changes from document Revision T to Revision U.

1. Workaround for CPU46 was updated.

Changes from document Revision U to Revision V.

1. Workaround for PMM15 was updated.

IMPORTANT NOTICE FOR TI DESIGN INFORMATION AND RESOURCES

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265