MSP430FR5994 Device Erratasheet

1 Revision History

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

Errata Number	Rev C
ADC42	✓
CPU21	✓
CPU22	✓
CPU40	✓
CPU46	✓
CS12	✓
RTC12	✓
USCI42	✓
USCI45	✓

2 Package Markings

PN80

LQFP (PN), 80 Pin

YMLLLLSG4
MSP430™
FRxxxx
REV #
○

YM	= Year and Month Date Code
LLLL	= Assembly Lot Code
S	= Assembly Site Code
#	= Die Revision
○	= Pin 1

ZVW87

NFBGA (ZVW), 87 pin

MSP430™
FRxxxx
YMLLLLS #
TI G1
○

YM	= Year and Month Date Code
LLLL	= Assembly Lot Code
S	= Assembly Site Code
#	= Die Revision
TI	= TI Letters
○	= Pin 1 Location

PM64

LQFP (PM), 64 Pin

YMLLLLSG4
MSP430™
FRxxxx
REV #
○

YM	= Year and Month Date Code
LLLL	= Assembly Lot Code
S	= Assembly Site Code
#	= Die Revision
○	= Pin 1

RGZ48

QFN (RGZ), 48 Pin

○
MSP430™
FRxxxx
TI YMS #
LLLL G4

YM	= Year and Month Date Code
S	= Assembly Site Code
#	= Die Revision
LLLL	= Assembly Lot Code
○	= Pin 1

3 Detailed Bug Description

ADC42 *ADC12_B 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

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	

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
```

CS12 — *CS Module*

Function

DCO overshoot at frequency change

Description

When changing frequencies (CSCTL1.DCOFSEL), the DCO frequency may overshoot and exceed the datasheet specification. After a time period of 10us has elapsed, the frequency overshoot settles down to the expected range as specified in the datasheet. The overshoot occur when switching to and from any DCOFSEL setting and impacts all peripherals using the DCO as a clock source. A potential impact can also be seen on FRAM accesses, since the overshoot may cause a temporary violation of FRAM access and cycle time requirements.

Workaround

When changing the DCO settings, use the following procedure:

1) Store the existing CSCTL3 divider into a temporary unsigned 16-bit variable

2) Set CSCTL3 to divide all corresponding clock sources by 4 or higher

3) Change DCO frequency

4) Wait ~10us

5) Restore the divider in CSCTL3 to the setting stored in the temporary variable.

The following code example shows how to increase DCO to 16MHz.

```
uint16_t tempCSCTL3 = 0;
CSCTL0_H = CSKEY_H;         // Unlock CS registers
/* Assuming SMCLK and MCLK are sourced from DCO */
/* Store CSCTL3 settings to recover later */
```

```
tempCSCTL3 = CSCTL3;
/* Keep overshoot transient within specification by setting clk sources
to divide by 4*/
/* Clear the DIVS & DIVM masks (~0x77)and set both fields to 4 divider */
CSCTL3 = CSCTL3 & (~(0x77)) | DIVS__4 | DIVM__4;
CSCTL1 = DCOFSEL_4 | DCORSEL;        // Set DCO to 16MHz
/* Delay by ~10us to let DCO settle. 60 cycles = 20 cycles buffer +
(10us / (1/4MHz)) */
__delay_cycles(60);
CSCTL3 =  tempCSCTL3;      // Set all dividers
CSCTL0_H = 0;                    // Lock CS registers
```

RTC12 — *RTC_C Module*

Function Real-time clock temperature compensation RTCTCOK bit not retained after LPM3.5 wake up

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

Workaround Store the RTCTCMP register content into Flash content or SRAM bank 0 for retention after wake up from LPM3.5

USCI42 — *eUSCI Module*

Function UART asserts UCTXCPTIFG after each byte in multi-byte transmission

Description UCTXCPTIFG flag is triggered at the last stop bit of every UART byte transmission, independently of an empty buffer, when transmitting multiple byte sequences via UART. The erroneous UART behavior occurs with and without DMA transfer.

Workaround None.

USCI45 — *eUSCI Module*

Function Unexpected SPI clock stretching possible

Description In rare cases, during SPI communication, the clock high phase of the first data bit may be stretched significantly. The SPI operation completes as expected with no data loss. This issue only occurs when the USCI SPI module clock (UCxCLK) is asynchronous to the system clock (MCLK).

Workaround Ensure that the USCI SPI module clock (UCxCLK) and the CPU clock (MCLK) are synchronous to each other.

4 Document Revision History

Changes from device specific erratasheet to document Revision A.

1. Errata ADC38 was removed from the errata documentation.
2. Errata CS12 was added to the errata documentation.
3. Module name for ADC42 was modified.
4. Errata USCI42 was added to the errata documentation.
5. Errata JTAG27 was added to the errata documentation.

Changes from document Revision A to Revision B.

1. LEA1 was added to the errata documentation.
2. RTC10 was added to the errata documentation.
3. ADC43 was added to the errata documentation.
4. ADC38 was added to the errata documentation.
5. USCI45 was added to the errata documentation.
6. PMM28 was added to the errata documentation.
7. PMM27 was added to the errata documentation.
8. USCI43 was added to the errata documentation.
9. CPU46 was added to the errata documentation.

Changes from document Revision B to Revision C.

1. Device name changed from "XMS" to "MSP430"
2. LEA1 was removed from the errata documentation.
3. RTC10 was removed from the errata documentation.
4. ADC43 was removed from the errata documentation.
5. JTAG27 was removed from the errata documentation.
6. ADC38 was removed from the errata documentation.
7. PMM25 was removed from the errata documentation.
8. COMP10 was removed from the errata documentation.
9. PMM28 was removed from the errata documentation.
10. PMM27 was removed from the errata documentation.
11. USCI43 was removed from the errata documentation.
12. CPU21 was added to the errata documentation.
13. CPU22 was added to the errata documentation.
14. Silicon Revision A was removed from the errata documentation.
15. Silicon Revision C was added to the errata documentation.
16. ZVW87 was added to errata documentation
17. RGZ48 was added to errata documentation
18. PM64 was added to errata documentation
19. Workaround for CPU40 was updated.

IMPORTANT NOTICE FOR TI DESIGN INFORMATION AND RESOURCES