MSP430FR5738 Device Erratasheet

1 Revision History

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

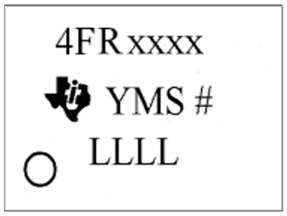
The revision of the device can be identified by the revision letter on the Package Markings or by the HW_ID located inside the TLV structure of the device

Errata Number	Rev J	Rev H
ADC38	✓	✓
ADC39	✓	✓
ADC42	✓	✓
COMP10	✓	✓
COMP11	✓	✓
CPU21	✓	✓
CPU22	✓	✓
CPU40	✓	✓
CPU46	✓	✓
CS12	✓	✓
DMA7	✓	✓
DMA9	✓	✓
DMA10	✓	✓
EEM19	✓	✓
EEM23	✓	✓
EEM25	✓	✓
EEM30	✓	✓
EEM31	✓	✓
GC3	✓	✓
JTAG27	✓	✓
MPY1	✓	✓
PORT16	✓	✓
PORT19	✓	✓
PORT26	✓	✓
RTC14	✓	✓
TA23	✓	✓
USCI36	✓	✓
USCI37	✓	✓
USCI41	✓	✓
USCI42	✓	✓
USCI44	✓	✓
WDG6	✓	✓
XOSC13	✓	✓

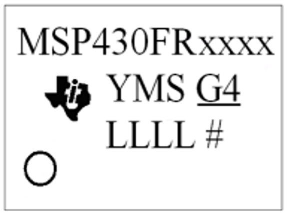
2 Package Markings

PW28

TSSOP (PW), 28 Pin



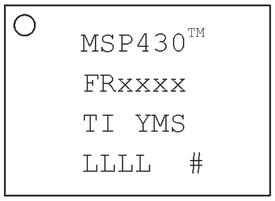
YM	= Year and Month Date Code
LLLL	= Assembly Lot Code
S	= Assembly Site Code
#	= DIE Revision
o	= PIN 1



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

RGE24

QFN (RGE), 24 Pin



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

3 Memory-Mapped Hardware Revision (TLV Structure)

Die Revision	TLV Hardware Revision
Rev J	26h
Rev H	24h

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

ADC38 *ADC10 Module*

Function External ADC trigger without toggling ENC bit might prevent further ADC conversions.

Description The ADC may stop sampling and converting until the module is reset if an external (timer) trigger occurs without toggling the ADC12CTL0.ADC12ENC bit at:

- The end of sequence in the sequence-of-channel mode.

- The end of conversion in single-channel mode.

Workaround Ensure ADC12CTL0.ADC12ENC bit is always toggled before providing any new External Trigger to ADC.

ADC39 *ADC10 Module*

Function Erroneous ADC10 results in extended sample mode

Description If the extended sample mode is selected (ADC10SHP = 0) and the ADC10CLK is asynchronous to the SHI signal, the ADC10 may generate erroneous results.

Workaround 1) Use the pulse sample mode (ADC10SHP=1)

OR

2) Use a synchronous clock for ADC10 and the SHI signal.

ADC42 *ADC10 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

COMP10 *COMP_D 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.

COMP11 *COMP_D Module*

Function Polling comparator interrupts may result in a missed interrupt

Description Polling the comparator output interrupt and the output inverted interrupt flags (CEIFG.CExINT and CEIIFG.CExINT) may result in a missed interrupt if the flags are modified while being read.

Workaround Using an interrupt service routine to service CEIFG and CEIIFG interrupts significantly reduces the probability of the issue occurring.

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

IDE/Compiler	Version Number	Notes
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.
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.

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
```

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.

DMA9 *DMA Module*

Function DMA stops transferring bytes unexpectedly

Description When the DMA is configured to transfer bytes from the eUSCI_A or eUSCI_B transmit or receive buffers, the transmit or receive triggers (TXIFG and RXIFG) may not be seen by the DMA module and the transfer of the bytes is missed. Once the first byte in a transfer sequence is missed, all the following bytes are missed as well. All eUSCI_A modes (UART, SPI, and IrDA) and all eUSCI_B modes (SPI and I2C) are affected.

Workaround 1) Use Interrupt Service Routines to transfer data to and from the eUSCI_A or eUSCI_B.

OR

2) When using DMA channel 0 for transferring data to and from the eUSCI_A or eUSCI_B, use DMA channel 2 (lower priority than DMA channel 0) to read the same register of the eUSCI_A or eUSCI_B that DMA channel 0 is working with. Use the same USCI IFG (e.g. UCA0RXIFG) as trigger source for these both DMA channels.

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.

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.

EEM25 — *EEM Module*

Function Unexpected wakeup from LPMx.5

Description When the device is in LPMx.5, debugging in SBW mode can cause an unexpected wakeup from the low power mode.

Workaround Use 4-wire JTAG when debugging.

NOTE: This issue only affects debug mode.

EEM30 — *EEM Module*

Function Missed breakpoint if FRAM power supply is disabled

Description The FRAM power supply can be disabled (GCCTL0.FRPWR = 0) prior to LPM entry to save power. Upon wakeup, if a breakpoint is set on an the first instruction that accesses FRAM, the breakpoint may be missed.

Workaround None. This issue affects debug mode only.

EEM31 — *EEM Module*

Function Breakpoint trigger may be lost when MPU is enabled

Description A data value written to FRAM can be used as a trigger condition for breakpoints during a debug session. This trigger can be lost if the FRAM access is made to an address that has been write-protected by the MPU.

Workaround None. This issue affects debug mode only.

GC3 — *GC Module*

Function Error flags set incorrectly after reset or wake-up from LPM

Description The error flags GCCTL1.UBDIFG and GCCTL1.CBDIFG are incorrectly set after the first power-up or wake up from LPM2/LPM3/LPM4/LPM3.5/LPM4.5. If the corresponding interrupt enable bits are set, then an erroneous interrupt can be generated.

Workaround Disable the UBDIEN and CBDIEN interrupts (GCCTL0.UBDIEN=0 and GCCTL0.CBDIEN=0) prior to entering LPM2/LPM3/LPM4/LPM4.5 . After LPM exit, re-initialize the interrupt flags only after the first FRAM access has been completed.

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 *MPY32 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

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.

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

PORT26 *PORT Module*

Function Incorrect values for P1.3 / P1.4 input pins during power-up

Description If P1.3/P1.4 is pulled up externally to DVCC during power-up the logical HIGH value might not be read correct by the device (ZERO is read instead of ONE).

Workaround 1) Switch the P1.3/P1.4 Port to logical ZERO after power cycle by:

a) Switch critical GPIO to output-low (with series resistance to limit current) or

b) Remove external pull up connection to pull GPIO via internal pull-down

OR

2) Use different GPIOs (not P1.3 & P1.4)

OR

3) Change the polarity of the logical check in SW (enable internal pull-up resistor for the GPIO and pull the external pin to DVSS)

RTC14 *RTC_B Module*

Function Polling RTC interrupts may result in a lost interrupt flag

Description Polling any RTC interrupt flag mapped to the RTCIV register may result in a missed interrupt if the flags are modified while being read.

Workaround Using an interrupt service routine to service flags mapped to the RTCIV register significantly reduces the probability of the issue occurring.

TA23 *TIMER_A Module*

Function Polling timer interrupts may result in a lost interrupt flag

Description Polling any Timer interrupt flag may result in a missed interrupt if the flags are modified while being read.

Workaround Using an interrupt service routine to service timer interrupt flags significantly reduces the probability of the issue occurring.

USCI36 *eUSCI Module*

Function UCLKI not usable in I2C master mode

Description When EUSCIB is configured as I2C Master with the external UCLKI as clock source, the UCLKI signal is not available and cannot be used to source I2C clock.

Workaround Use LFXTCLK via ACLK or HFXTCLK via SMCLK as clock source (BRCLK) for I2C in master mode with external clock source.

USCI37 *eUSCI Module*

Function Reading RXBUF during an active I2C communication might result in unintended bus stalls.

Description The falling edge of SCL bus line is used to set an internal RXBUF-written flag register, which is used to detect a potential RXBUF overflow. If this flag is cleared with a read access from the RXBUF register during a falling edge of SCL, the clear condition might be missed. This could result in an I2C bus stall at the next received byte.

Workaround (1) Execute two consecutive reads of RXBUF, if $t_{SCL} > 4 \times t_{MCLK}$.

or

(2) Provoke an I2C bus stall before reading RXBUF. A bus stall can be verified by checking if the clock line low status indicator bit UCSCLLOW is set for at least three USCI bit clock cycles i.e. $3 \times t_{BitClock}$.

USCI41 *eUSCI Module*

Function UCBUSY bit of eUSCIA module stuck to 1 when device is in SPI mode.

Description When eUSCIA is configured in SPI mode, and the last transfer bit changes from 0 to 1, the UCBUSY bit gets stuck to 1. This happens in all four combinations of Clock Phase and Clock Polarity options (UCAxCTLW0.UCCKPH & UCAxCTLW0.UCCKPL bits). There is no data loss or corruption. Because the USCBUSY bit is stuck to 1, the clock request stays enabled and adds additional current consumption in low power mode operation.

Workaround Check on transmit or receive interrupt flag UCTXIFG/UCRXIFG instead of UCBUSY to know if the UCAxTXBUF buffer is empty or ready for the next complete character.

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.

USCI44 *eUSCI Module*

Function Differing clock sources may cause UART communication failure

Description When using the USCI_A UART module with differing clock sources for the system clock (MCLK) and the UART source clock (BRCLK), a read or write of the UCAxIFG, UCAxCTLW0, UCAxSTATW, UCAxRXBUF, UCAxTXBUF, UCAxABCTL & UCAxIV registers, while the UCATXIFG or UCARXIFG flag is being set by a UART event could unintentionally clear the UCATXIFG or UCARXIFG. This may result in the UART communication being stalled.

Workaround Workaround 1: Use synchronous clocks to source BRCLK and MCLK. Note that the clock frequencies need not be identical and dividers may be used as long as they are using the same clock source.

Workaround 2: Avoid polling UCAxTXIFG and UCAxRXIFG. Using the standard interrupt service routine to service the interrupt flag significantly reduces the probability of this errata occurring. Avoid register access to UCAxCTLW0, UCAxSTATW, UCAxRXBUF, UCAxTXBUF, UCAxABCTL, UCAxIFG, & UCAxIV while transmit or receive operation is ongoing and UCAxRXIFG or UCAxTXIFG is expected to be set.

WDG6 *WDT_A Module*

Function	Clock Fail-Safe feature in LPMx.5
Description	The watchdog clock fail-safe feature does not prevent the device from going into LPMx.5. As a result, the device enters LPMx.5 state independently of running the watchdog. Note that the watchdog is off in LPMx.5.
Workaround	None.

XOSC13 *XOSC Module*

Function	XT1 in bypass mode does not work with RTC enabled
Description	When the RTC module is enabled and XT1 is configured in bypass mode to be sourced by an external digital signal, the XT1OFFG flag is set indefinitely, resulting in ACLK defaulting its clock source to VLO instead of XT1.
Workaround	Do not use RTC module with XT1 in bypass mode with external digital clock source.

5 Document Revision History

Changes from family erratasheet to device specific erratasheet.

1. Revision F was removed
2. Revision G was removed

Changes from device specific erratasheet to document Revision A.

1. Errata PORT19 was added to the errata documentation.
2. Module name for MPY1 was modified.
3. Errata DMA9 was added to the errata documentation.

Changes from document Revision A to Revision B.

1. Errata XOSC13 was added to the errata documentation.

Changes from document Revision B to Revision C.

1. Errata DMA10 was added to the errata documentation.

Changes from document Revision C to Revision D.

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

Changes from document Revision D to Revision E.

1. DMA10 Description was updated.
2. Errata WDG6 was added to the errata documentation.
3. MPY1 Description was updated.
4. Errata EEM23 was added to the errata documentation.

Changes from document Revision E to Revision F.

1. DMA9 Description was updated.
2. DMA9 Workaround was updated.
3. Device TLV Hardware Revision information added to erratasheet.

Changes from document Revision F to Revision G.

1. Errata USCI37 was added to the errata documentation.

Changes from document Revision G to Revision H.

1. EEM19 Workaround was updated.
2. PORT16 Workaround was updated.
3. EEM23 Workaround was updated.
4. EEM23 Description was updated.
5. Errata ADC39 was added to the errata documentation.
6. Errata USCI36 was added to the errata documentation.
7. PORT16 Description was updated.
8. EEM19 Description was updated.

Changes from document Revision H to Revision I.

1. DMA10 Workaround was updated.
2. Errata EEM25 was added to the errata documentation.
3. DMA10 Description was updated.
4. DMA10 Function was updated.

Changes from document Revision I to Revision J.

1. CPU40 Workaround was updated.
2. EEM19 Workaround was updated.
3. Package Markings section was updated.

4. EEM23 Workaround was updated.
5. EEM23 Description was updated.
6. Errata ADC42 was added to the errata documentation.
7. EEM23 Function was updated.
8. EEM19 Description was updated.

Changes from document Revision J to Revision K.

1. Errata DMA7 was added to the errata documentation.
2. DMA9 Description was updated.
3. Errata PORT26 was added to the errata documentation.
4. DMA9 Workaround was updated.

Changes from document Revision K to Revision L.

1. Silicon Revision J was added to the errata documentation.
2. DMA7 Workaround was updated.
3. EEM23 Description was updated.
4. DMA7 Description was updated.

Changes from document Revision L to Revision M.

1. Errata ADC38 was added to the errata documentation.
2. DMA9 Workaround was updated.

Changes from document Revision M to Revision N.

1. Errata USCI41 was added to the errata documentation.

Changes from document Revision N to Revision O.

1. USCI37 Workaround was updated.

Changes from document Revision O to Revision P.

1. EEM19 Workaround was updated.

Changes from document Revision P to Revision Q.

1. Errata CS12 was added to the errata documentation.
2. Errata GC3 was added to the errata documentation.
3. Errata USCI42 was added to the errata documentation.
4. Errata EEM30 was added to the errata documentation.
5. Errata EEM31 was added to the errata documentation.
6. Errata JTAG27 was added to the errata documentation.
7. Errata COMP10 was added to the errata documentation.

Changes from document Revision Q to Revision R.

1. Errata CPU46 was added to the errata documentation.

Changes from document Revision R to Revision S.

1. TA23 was added to the errata documentation.
2. RTC14 was added to the errata documentation.
3. CPU21 was added to the errata documentation.
4. CPU22 was added to the errata documentation.
5. USCI44 was added to the errata documentation.
6. COMP11 was added to the errata documentation.
7. Workaround for CPU40 was updated.
8. Workaround for CPU46 was updated.
9. Description for USCI41 was updated.

Changes from document Revision S to Revision T.

1. TLV hardware revision ID for Rev J was updated.
2. Workaround for CPU46 was updated.