Errata

MSP430FR4133 Microcontroller

ABSTRACT

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

Table of Contents

1 Functional Advisories

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

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

Errata Number	Rev B
ADC39	✓
ADC50	✓
ADC63	✓
CPU46	✓
CS11	✓
CS13	✓
GC1	✓
GC4	✓
GC5	✓
PMM32	✓
PORT28	✓
RTC15	✓
SYS23	✓
USCI41	✓
USCI42	✓
USCI45	✓
USCI47	✓
USCI50	✓

2 Preprogrammed Software Advisories

Advisories that affect factory-programmed software.

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

The device does not have any errata for this category.

3 Debug Only Advisories

Advisories that affect only debug operation.

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

Errata Number	Rev B
EEM23	✓
EEM28	✓
EEM30	✓

4 Fixed by Compiler Advisories

Advisories that are resolved by compiler workaround. Refer to each advisory for the IDE and compiler versions with a workaround.

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

Errata Number	Rev B
CPU21	✓
CPU22	✓

Errata Number	Rev B
CPU40	✓

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

TI MSP430 Compiler Tools (Code Composer Studio IDE)

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

MSP430 GNU Compiler (MSP430-GCC)

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

IAR Embedded Workbench

- IAR workarounds for msp430 hardware issues

5 Nomenclature, Package Symbolization, and Revision Identification

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

5.1 Device Nomenclature

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

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

MSP – Fully qualified production device

Support tool naming prefixes:

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

null: Fully-qualified development-support product.

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

"Developmental product is intended for internal evaluation purposes."

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

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

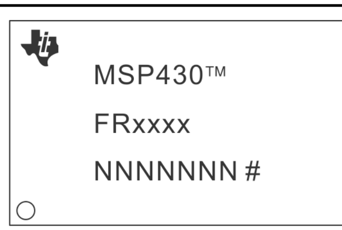
TI device nomenclature also includes a suffix with the device family name. This suffix indicates the temperature range, package type, and distribution format.

5.2 Package Markings

DGG48 *(DGG), 48 Pin*

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

DGG56 *(DGG), 56 Pin*



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

PM64 *LQFP (PM), 64 Pin*

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

5.3 Memory-Mapped Hardware Revision (TLV Structure)

Die Revision	TLV Hardware Revision
Rev B	20h

Further guidance on how to locate the TLV structure and read out the HW_ID can be found in the device User's Guide.

6 Advisory Descriptions

ADC39 — *ADC Module*

Category Functional

Function Erroneous ADC results in extended sample mode

Description If the extended sample mode is selected (ADCSHP = 0) and the ADCCLK is asynchronous to the SHI signal, the ADC may generate erroneous results.

Workaround 1) Use the pulse sample mode (ADCSHP=1)
OR
2) Use a synchronous clock for ADC and the SHI signal. For example, if SMCLK is used to source Timer to trigger SHI, ADCCLK should also be sourced by SMCLK.

ADC50 — *ADC Module*

Category Functional

Function Erroneous ADC conversion result for internal temperature sensor in LPM3 mode

Description When ACLK is used as ADC clock source and device is in LPM3 mode while sampling the on-chip temperature sensor, the ADC may generate erroneous conversion results.

Workaround 1) Use SMCLK or MODCLK as the ADC clock source. A 100us sampling time is required if triggering ADC conversion from LPM3.

OR

2) Use LPM0 or Active Mode.

ADC63 — *ADC Module*

Category Functional

Function ADCHI/ADCLO may be reset unexpectedly when ADCCTL2 high byte is written byte-wise

Description ADCHI/ADCLO may be reset unexpectedly when ADCCTL2 high byte is written byte-wise.

Workaround Write to ADCCTL2 high byte in word-wise method.

CPU21 — *CPU Module*

Category Compiler-Fixed

Function Using POPM instruction on Status register may result in device hang up

Description When an active interrupt service request is pending and the POPM instruction is used to set the Status Register (SR) and initiate entry into a low power mode , the device may hang up.

Workaround None. It is recommended not to use POPM instruction on the Status Register.

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

IDE/Compiler	Version Number	Notes
IAR Embedded Workbench	Not affected	
TI MSP430 Compiler Tools (Code Composer Studio)	v4.0.x or later	User is required to add the compiler or assembler flag option below. --silicon_errata=CPU21
MSP430 GNU Compiler (MSP430-GCC)	MSP430-GCC 4.9 build 167 or later	

CPU22

CPU Module

Category

Compiler-Fixed

Function

Indirect addressing mode with the Program Counter as the source register may produce unexpected results

Description

When using the indirect addressing mode in an instruction with the Program Counter (PC) as the source operand, the instruction that follows immediately does not get executed. For example in the code below, the ADD instruction does not get executed.

```
mov @PC, R7
add #1h, R4
```

Workaround

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

IDE/Compiler	Version Number	Notes
IAR Embedded Workbench	Not affected	
TI MSP430 Compiler Tools (Code Composer Studio)	v4.0.x or later	User is required to add the compiler or assembler flag option below. --silicon_errata=CPU22
MSP430 GNU Compiler (MSP430-GCC)	MSP430-GCC 4.9 build 167 or later	

CPU40

CPU Module

Category

Compiler-Fixed

Function

PC is corrupted when executing jump/conditional jump instruction that is followed by instruction with PC as destination register or a data section

Description

If the value at the memory location immediately following a jump/conditional jump instruction is 0X40h or 0X50h (where X = don't care), which could either be an instruction opcode (for instructions like RRCM, RRAM, RLAM, RRUM) with PC as destination register or a data section (const data in flash memory or data variable in RAM), then the PC value is auto-incremented by 2 after the jump instruction is executed; therefore, branching to a wrong address location in code and leading to wrong program execution.

For example, a conditional jump instruction followed by data section (0140h).

@0x8012 Loop DEC.W R6
@0x8014 DEC.W R7

@0x8016 JNZ Loop
@0x8018 Value1 DW 0140h

Workaround

In assembly, insert a NOP between the jump/conditional jump instruction and program code with instruction that contains PC as destination register or the data section.

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

IDE/Compiler	Version Number	Notes
IAR Embedded Workbench	IAR EW430 v5.51 or later	For the command line version add the following information Compiler: --hw_workaround=CPU40 Assembler:-v1
TI MSP430 Compiler Tools (Code Composer Studio)	v4.0.x or later	User is required to add the compiler or assembler flag option below. --silicon_errata=CPU40
MSP430 GNU Compiler (MSP430-GCC)	Not affected	

CPU46

CPU Module

Category

Functional

Function

POPM peforms unexpected memory access and can cause VMAIFG to be set

Description

When the POPM assembly instruction is executed, the last Stack Pointer increment is followed by an unintended read access to the memory. If this read access is performed on vacant memory, the VMAIFG will be set and can trigger the corresponding interrupt (SFRIE1.VMAIE) if it is enabled. This issue occurs if the POPM assembly instruction is performed up to the top of the STACK.

Workaround

If the user is utilizing C, they will not be impacted by this issue. All TI/IAR/GCC pre-built libraries are not impacted by this bug. To ensure that POPM is never executed up to the memory border of the STACK when using assembly it is recommended to either

1. Initialize the SP to
a. TOP of STACK - 4 bytes if POPM.A is used
b. TOP of STACK - 2 bytes if POPM.W is used

OR

2. Use the POPM instruction for all but the last restore operation. For the the last restore operation use the POP assembly instruction instead.

For instance, instead of using:

```
POPM.W #5,R13
```

Use:

```
POPM.W #4,R12
POP.W R13
```

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

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

CS11 *CS Module*

Category Functional

Function FLL cannot lock at low or high temperature

Description The FLL may fail to lock under the following conditions:

(1) at low temperature -40C if DCOTAP (CSCTL0.DCO) value is too close to 0x1FF when FLL is locked at room temperature.

OR

(2) The FLL may fail to lock at high temperature 85C if DCOTAP (CSCTL0.DCO) value is too close to 0x0 when FLL is locked at room temperature.

Workaround 1. Calibration once at room temperature

(a) Check the DCOTAP value when FLL is locked at room temperature 22C-28C.

(b) If DCOTAP is between 0x190 and 0x70, go to step (d).

(c) If DCOTAP is greater than 0x190, set register DCOFTRIMEN=1 and increase register DCOFTRIM by 1; If DCOTAP is less than 0x70, set register DCOFTRIMEN=1 and decrease register DCOFTRIM by 1. Then repeat from step (a).

(d) Save calibrated DCOFTRIM value into FRAM.

(e) Set calibration flag and save it into FRAM.

AND

2. When user code boot up again, and if the calibration flag in FRAM is set, configure register DCOFTRIMEN=1 and load calibrated DCOFTRIM value from FRAM into register DCOFTRIM before enabling FLL to lock.

CS13 — *CS Module*

Category Functional

Function Device may enter lockup state during transition from AM to LPM3/4 if DCO frequency is above 2 MHz.

Description The device might enter lockup state if DCO frequency is above 2 MHz and two events happen at the same time:

1) The device transitions from AM to LPM3/4 (e.g. during ISR exits or Status Register modifications)

2) An interrupt is requested (e.g. GPIO interrupt).

This condition can be recovered by BOR/Power cycle.

Workaround 1. Use DCOCLK at 2MHz or lower.

OR

2. Use LPM0/x.5 instead of LPM3/4.

OR

3. Use external high-frequency crystal if it is available on the device.

OR

4. Set DCOCLK to 2MHz or lower before entering LPM3/4, then restore DCOCLK after wake-up. Note using peripherals using clocks derived from DCOCLK might be affected during this interval.

EEM23 — *EEM Module*

Category Debug

Function EEM triggers incorrectly when modules using wait states are enabled

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

Workaround None.

Note

This erratum affects debug mode only.

EEM28 — *EEM Module*

Category Debug

Function Clock outputs observed on port module during LPMx in debug mode

Description When the device is in LPMx mode, if a debug halt is requested and if the port pin is configured as MCLK, SMCLK, or ACLK output, these clocks are observed on the port pin. Depending on the LPM mode (see Device User's Guide), peripherals that are clocked from MCLK, SMCLK, or ACLK are still halted during debug halt state.

For example, if the device is in debug halt in LPM3 mode and a port pin is configured as SMCLK output, SMCLK can be observed on the pin. But the peripherals sourced from SMCLK are still halted as expected.

Workaround None

EEM30 — *EEM Module*

Category Debug

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.

GC1 — *GC Module*

Category Functional

Function Uncorrectable memory bit error flag (GCCTL1.UBDIFG) does not trigger NMI

Description The GCCTL1.UBDIFG flag is an interrupt flag that gets set if an uncorrectable bit error has been detected in non-volatile memory. Even the GCCTL1.UBDIFG flag is set to 1 (GCCTL0.UBDRSTEN = 0 and GCCTL0.UBDIE = 1), it does not trigger a NMI request. In this case, the application is not notified via a NMI request that an uncorrectable bit error occurred in non-volatile memory (SYSSNIV = 0).

Workaround Set GCCTL0.UBDRSTEN = 1 and GCCTL0.UBDIE = 0 to trigger a PUC and check GCCTL1.UBDIFG = 1 after each PUC for manual interrupt flag handling.
Please consider GC4 errata for side effects.

GC4 — *GC Module*

Category Functional

Function Unexpected PUC is triggered

Description During execution from FRAM a non-existent uncorrectable bit error can be detected and trigger a PUC if the uncorrectable bit error detection flag is set (GCCTL0.UBDRSTEN = 1). This behavior appears only if:

(1) MCLK is sourced from DCO frequency of 16 MHz

OR

(2) MCLK is sourced by external high frequency clock above 12 MHz at pin HFXIN

OR

(3) MCLK is sourced by High-Frequency crystals (HFXT) above 12 MHz.

This PUC will not be recognized by the SYSRSTIV register (SYSRSTIV = 0x00).
A PUC RESET will be executed with not defined reset source.
Also the corresponding bit error detection flag is not set (GCCTL1.UBDIFG = 0).

Workaround

1. Check the reset source for SYSRSTIV = 0 and ignore the reset.

OR

2. Set GCCTL0.UBDRSTEN = 0 to prevent unexpected PUC. NMI event will not be triggered, even if GCCTL0.UBDIE = 1 -> consider GC1 Errata for more details.

OR

3. Set the MCLK to maximum 12MHz to leverage the uncorrectable bit error PUC feature.

GC5 — *GC Module*

Category

Functional

Function

Nonexistent FRAM failures can be detected after wake-up from LPM 1/2/3/4

Description

The FRAM bit error detection may indicate bit errors, even the memory has no failure, after wakeup from LPM1/2/3/4.
Based on the setting inside the FRAM controller registers (GCCTL0), following behaviors can appear.

1. Unexpected PUC for an uncorrectable FRAM error can be triggered and causing the corresponding value in the SYSRSTIV register.
This happens only if GCCTL0.UBDRSTEN =1.

2. Unexpected NMI for an uncorrectable FRAM error can be triggered and causing the corresponding value in the SYSSNIV register.
This happens only if the GCCTL0.UBDIE = 1.

3. Unexpected NMI for a correctable FRAM error can be triggered and causing the corresponding value in the SYSSNIV register.
This happens only if the GCCTL0.CBDIE =1.

Workaround

1. Disable PUC (GCCTL0.UBDRSTEN=0), UBDIE and CBDIE interrupts (GCCTL0.UBDIE=0 and GCCTL0.CBDIE=0) prior to entering LPM 1/2/3/4.

2. After LPM wake up, clear GCCTL1.UBDIFG and GCCTL1.CBDIFG, and then reinitialize the GCCTL0 register after the first valid FRAM access has been completed.
For the valid FRAM access the user has to consider possible cache hits which depends on implementation.

PMM32 — *PMM Module*

Category

Functional

Function

Device may enter lockup state or execute unintentional code during transition from AM to LPM3/4

Description

The device might enter lockup state or start executing unintentional code resulting in unpredictable behavior depending on the contents of the address location- if any of the two conditions below occurs:

Condition1:

The following three events happen at the same time:

1) The device transitions from AM to LPM3/4 (e.g. during ISR exits or Status Register modifications),

AND

2) An interrupt is requested (e.g. GPIO interrupt),

AND

3) MODCLK is requested (e.g. triggered by ADC) or removed (e.g. end of ADC conversion).

Modules which can trigger MODCLK clock requests/removals are ADC, eUSCI and CapTIvate (if exist).

If clock events are started by the CPU (e.g. eUSCI during SPI master transmission), they can not occur at the same time as the power mode transition and thus should not be affected. The device should only be affected when the clock event is asynchronous to the power mode transition.

The device can recover from this lockup condition by a PUC/BOR/Power cycle (e.g. enable Watchdog to trigger PUC).

Condition2:

The following events happen at the same time:

1) The device transitions from AM to LPM3/4 (e.g. during ISR exits or Status Register modifications),

AND

2) An interrupt is requested (e.g. GPIO interrupt),

AND

3) Neither MODCLK nor SMCLK are running (e.g. requested by a peripheral),

AND

4) SMCLK is configured with a different frequency than MCLK.

The device can recover from this lockup condition by a BOR/Power cycle.

Workaround

1. Use LPM0/1/x.5 instead of LPM3/4.

OR

2. Place the FRAM in INACTIVE mode before any entry to LPM3/4 by clearing the FRPWR bit and FRLPMPWR bit (if exist) in the GCCTL0 register. This must be performed from RAM as shown below:

```
// define a function in RAM
#pragma CODE_SECTION(enterLpModeFromRAM,".TI.ramfunc")
void enterLpModeFromRAM(unsigned short lowPowerMode);

//call this function before any entry to LPM3/4
void enterLpModeFromRAM(unsigned short lowPowerMode)
{
FRCTL0 = FRCTLPW;
GCCTL0 &= ~(FRPWR+FRLPMPWR); //clear FRPWR and FRLPMPWR
FRCTL0_H = 0; //re-lock FRCTL
__bis_SR_register(lowPowerMode);
}
```

PORT28 — *PORT Module*

Category Functional

Function Pull-down resistor of TEST/SBWTCK pin

Description The device's internal pull-down resistor on the TEST/SBWTCK pin gets disabled if the SYS control bit SFRRPCR.SYSRSTRE is cleared. This can lead to increased current consumption and unintentionally-enabled JTAG access to the device.

Workaround 1) Do not clear the SFRRPCR.SYSRSTRE bit, use the SFRRPCR.SYSRSTRUP bit to define direction of the internal resistor on RST/NMI/SBWTDIO pin instead.

OR

2) Ensure a zero voltage level of TEST/SBWTCK pin by connecting the pin to an external component (e.g. external pull-down resistor) on the PCB.

RTC15 — *RTC Module*

Category Functional

Function RTC Counter stops operating if RTC Counter clock source is changed from XT1CLK to another source while XT1CLK is stopped

Description If XT1CLK is used as the clock source for the RTC Counter and XT1CLK stops (e.g. oscillator fault), if the RTC Counter clock source is changed by user software (e.g. in the clock fault handling ISR) from XT1CLK to a different clock source while XT1CLK is stopped the RTC Counter hangs. In this hang state, the RTC Counter stops operating and cannot be restarted without a device reset via the hardware RST pin, a power-cycle of the device, or recovery of XT1CLK oscillation.

Workaround To change the RTC Counter clock source due to an oscillator fault, in the ISR for handling the OFIFG fault, use this software sequence:

1) Change the RTC Counter clock source away from XT1CLK normally

2) Reconfigure the XIN pin as a GPIO output, then toggle the GPIO twice with at least 2 rising or falling edges.

At this point the RTC Counter will be able to resume operation.

SYS23 ***SYS Module***

Category Functional

Function RAM not preserved after BOR

Description RAM content at addresses 0x20FE and 0x20FF is not preserved on BOR following one of the following conditions:
- RST/NMI pin reset (RST/NMI BOR)
- Software BOR (PMMSWBOR)
- Access security area (Security violation BOR)
- SVSH low condition event

Workaround None.

USCI41 ***USCI Module***

Category Functional

Function UCBUSY bit of eUSCIA module might not work reliable when device is in SPI mode.

Description When eUSCIA is configured in SPI mode, the UCBUSY bit might get stuck to 1 or start toggling after transmission is completed. This happens in all four combinations of Clock Phase and Clock Polarity options (UCAxCTLW0.UCCKPH & UCAxCTLW0.UCCKPL bits) as well as in Master and Slave mode. There is no data loss or corruption. However the UCBUSY cannot be used in its intended function to check if transmission is completed. Because the UCBUSY bit is stuck to 1 or toggles, the clock request stays enabled and this adds additional current consumption in low power mode operation.

Workaround For correct functional implementation 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.
To reduce the additional current it is recommended to either reset the SPI module (UCAxCTLW0.UCSWRST) in the UCBxCTLW0 or send a dummy byte 0x00 after the intended SPI transmission is completed.

USCI42 ***USCI Module***

Category Functional

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 ***USCI Module***

Category Functional

Function Unexpected SPI clock stretching possible when UCxCLK is asynchronous to MCLK

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.

USCI47 *USCI Module*

Category Functional

Function eUSCI SPI slave with clock phase UCCKPH = 1

Description The eUSCI SPI operates incorrectly under the following conditions:

1. The eUSCI_A or eUSCI_B module is configured as a SPI slave with clock phase mode UCCKPH = 1

AND

2. The SPI clock pin is not at the appropriate idle level (low for UCCKPL = 0, high for UCCKPL = 1) when the UCSWRST bit in the UCxxCTLW0 register is cleared.

If both of the above conditions are satisfied, then the following will occur:
eUSCI_A: the SPI will not be able to receive a byte (UCAxRXBUF will not be filled and UCRXIFG will not be set) and SPI slave output data will be wrong (first bit will be missed and data will be shifted).
eUSCI_B: the SPI receives data correctly but the SPI slave output data will be wrong (first byte will be duplicated or replaced by second byte).

Workaround Use clock phase mode UCCKPH = 0 for MSP SPI slave if allowed by the application.

OR

The SPI master must set the clock pin at the appropriate idle level (low for UCCKPL = 0, high for UCCKPL = 1) before SPI slave is reset (UCSWRST bit is cleared).

OR

For eUSCI_A: to detect communication failure condition where UCRXIFG is not set, check both UCRXIFG and UCTXIFG. If UCTXIFG is set twice but UCRXIFG is not set, reset the MSP SPI slave by setting and then clearing the UCSWRST bit, and inform the SPI master to resend the data.

USCI50 *USCI Module*

Category Functional

Function Data may not be transmitted correctly from the eUSCI when operating in SPI 4-pin master mode with UCSTEM = 0

Description When the eUSCI is used in SPI 4-pin master mode with UCSTEM = 0 (STE pin used as an input to prevent conflicts with other SPI masters), data that is moved into UCxTXBUF while the UCxSTE input is in the inactive state may not be transmitted correctly. If the eUSCI is used with UCSTEM = 1 (STE pin used to output an enable signal), data is transmitted correctly.

Workaround When using the STE pin in conflict prevention mode (UCSTEM = 0), only move data into UCxTXBUF when UCxSTE is in the active state. If an active transfer is aborted by UCxSTE transitioning to the master-inactive state, the data must be rewritten into UCxTXBUF to be transferred when UCxSTE transitions back to the master-active state.

7 Revision History

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

Changes from October 9, 2019 to May 11, 2021	Page
• Changed the document format and structure; updated the numbering format for tables, figures, and cross references throughout the document	6

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES "AS IS" AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI's products are provided subject to TI's Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI's provision of these resources does not expand or otherwise alter TI's applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

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