Errata

TMS320F28003x Real-Time MCUs Silicon Errata Silicon Revision 0

ABSTRACT

This document describes the known exceptions to the functional specifications (advisories). This document may also contain usage notes. Usage notes describe situations where the device's behavior may not match presumed or documented behavior. This may include behaviors that affect device performance or functional correctness.

Table of Contents

List of Figures

List of Tables

1 Usage Notes and Advisories Matrices

Table 1-1 lists all usage notes and the applicable silicon revisions. Table 1-2 lists all advisories, modules affected, and the applicable silicon revisions.

1.1 Usage Notes Matrix

Table 1-1. Usage Notes Matrix

NUMBER	TITLE	SILICON REVISIONS AFFECTED
		0
Section 3.1.1	PIE: Spurious Nested Interrupt After Back-to-Back PIEACK Write and Manual CPU Interrupt Mask Clear	Yes
Section 3.1.2	Caution While Using Nested Interrupts With Repeat Block	Yes
Section 3.1.4	Security: The primary layer of defense is securing the boundary of the chip, which begins with enabling JTAGLOCK and Zero-pin Boot to Flash feature	Yes
Section 3.1.3	I2C Slave Mode: Managing Status Flags in Applications with Long Interrupt Response Times	Yes

1.2 Advisories Matrix

Table 1-2. Advisories Matrix

MODULE	DESCRIPTION	SILICON REVISIONS AFFECTED
		0
ADC	ADC: Interrupts may Stop if INTxCONT (Continue-to-Interrupt Mode) is not Set	Yes
ADC	ADC: Degraded ADC Performance With ADCCLK Fractional Divider	Yes
ADC	ADC: DMA Read of Stale Result	Yes
Analog Subsystem	Analog Subsystem: Writes to Bit 1 of the ADCDACLOOPBACK Register are not Functional	Yes
BOR	BOR: VDDIO Between 2.45 V and 3.0 V can Result in Multiple XRSn Pulses	Yes
C28	C28: Fast Integer Division (FID) and DIVF64 Incorrect Results	Yes
CMPSS	CMPSS: COMPxLATCH May Not Clear Properly Under Certain Conditions	Yes
CMPSS	CMPSS: A CMPSS Glitch can Occur if Comparator Input Pin has AGPIO Functionality and ADC is Sampling the Input Pin	Yes
DCAN	DCAN: During DCAN FIFO Mode, Received Messages May be Placed Out of Order in the FIFO Buffer	Yes
MCAN	MCAN: Message Order Inversion When Transmitting From Dedicated Tx Buffers Configured With Same Message ID	Yes
ePWM	ePWM: An ePWM Glitch can Occur if a Trip Remains Active at the End of the Blanking Window	Yes
ePWM	ePWM One-Shot/CBC Trip Event DCxEVTy.force Does Not Set the Trip Condition	Yes
ePWM	ePWM: Trip Events Will Not be Filtered by the Blanking Window for the First 3 Cycles After the Start of a Blanking Window	Yes
eQEP	eQEP: Position Counter Incorrectly Reset on Direction Change During Index	Yes
FPU	FPU: FPU-to-CPU Register Move Operation Preceded by Any FPU 2p Operation	Yes
I2C	I2C: Target Transmitter Mode, Standard Mode SDA Timings Limitation	Yes
LIN	LIN: Inconsistent Sync Field Error (ISFE) Flag/Interrupt Not Set When Sync Field is Erroneous	Yes
Memory	Memory: Prefetching Beyond Valid Memory	Yes
Boot ROM	Boot ROM: Data Overrun With MCAN Bootloader on TMX Devices	Yes
Boot ROM	Secure Live Firmware Update (LFU) Boot Modes are Deprecated	Yes
Flash	Flash: Execution of Fapi_setActiveFlashBank() Without Disabling Flash Prefetch may Cause ITRAP	Yes
SYSTEM	SYSTEM: HIC Illegal Read Error Flag Does not Get Asserted in Pagesel=0 Mode	Yes
SYSTEM	SYSTEM: Multiple Successive Writes to CLKSRCCTL1 Can Cause a System Hang	Yes
Diagnostics	Avoiding Spurious Interrupts While Using HWBIST	Yes
PLL	PLL Reference Clock Lost Detection: Missing Clock Flag may be Incorrectly Activated	Yes
SDFM	SDFM: Dynamically Changing Threshold Settings (LLT, HLT), Filter Type, or COSR Settings Will Trigger Spurious Comparator Events	Yes
SDFM	SDFM: Dynamically Changing Data Filter Settings (Such as Filter Type or DOSR) Will Trigger Spurious Data Acknowledge Events	Yes
SDFM	SDFM: Two Back-to-Back Writes to SDCPARMx Register Bit Fields CEVT1SEL, CEVT2SEL, and HZEN Within Three SD-Modulator Clock Cycles can Corrupt SDFM State Machine, Resulting in Spurious Comparator Events	Yes

Table 1-2. Advisories Matrix (continued)

MODULE	DESCRIPTION	SILICON REVISIONS AFFECTED
		0
Watchdog	Watchdog: WDKEY Register is not EALLOW-Protected	Yes

2 Nomenclature, Package Symbolization, and Revision Identification

2.1 Device and Development-Support Tool Nomenclature

To designate the stages in the product development cycle, TI assigns prefixes to the part numbers of all DSP devices and support tools. Each DSP commercial family member has one of three prefixes: TMX, TMP, or TMS (for example, TMS320F280039C). Texas Instruments recommends two of three possible prefix designators for its support tools: TMDX and TMDS. These prefixes represent evolutionary stages of product development from engineering prototypes (TMX and TMDX) through fully qualified production devices and tools (TMS and TMDS).

Device development evolutionary flow:

TMX Experimental device that is not necessarily representative of the final device's electrical specifications and may not use production assembly flow.

TMP Prototype device that is not necessarily the final silicon die and may not necessarily meet final electrical specifications.

TMS Production version of the silicon die that is fully qualified.

Support tool development evolutionary flow:

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

TMDS Fully-qualified development-support product.

TMX and TMP devices and TMDX development-support tools are shipped against the following disclaimer:

"Developmental product is intended for internal evaluation purposes."

Production devices and TMDS development-support tools 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 (X or P) have a greater failure rate than the standard production devices. Texas Instruments 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.

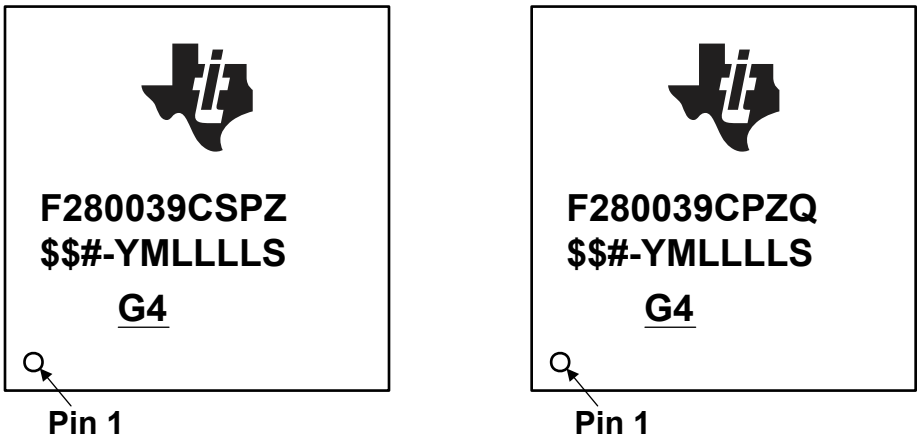
2.2 Devices Supported

This document supports the following devices:

- TMS320F280039C
- TMS320F280039C-Q1
- TMS320F280039
- TMS320F280039-Q1
- TMS320F280038C-Q1
- TMS320F280038-Q1
- TMS320F280037C
- TMS320F280037C-Q1
- TMS320F280037
- TMS320F280037-Q1
- TMS320F280036C-Q1
- TMS320F280036-Q1
- TMS320F280034
- TMS320F280034-Q1
- TMS320F280033
- TMS320F280033-Q1

2.3 Package Symbolization and Revision Identification

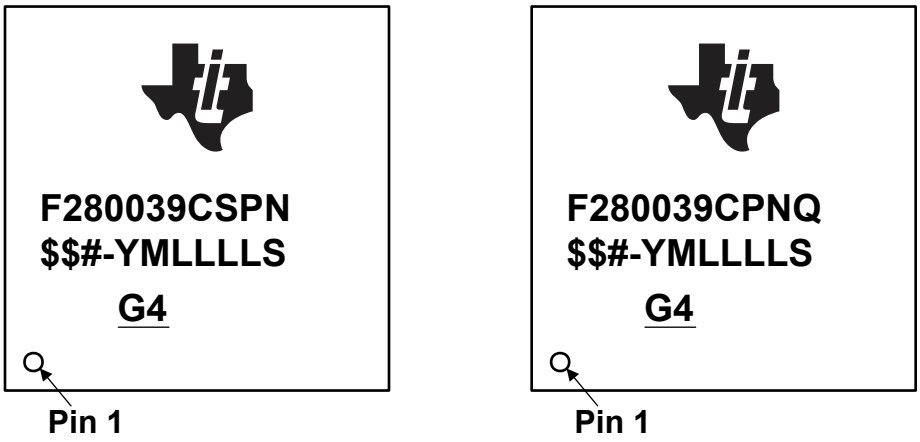
Figure 2-1, Figure 2-2, Figure 2-3, and Figure 2-4 show the package symbolization. Table 2-1 lists the silicon revision codes.



$$ = Wafer Fab Code (one or two characters)
# = Silicon Revision Code
YM = 2-digit Year/Month Code
LLLL = Assembly Lot Code
S = Assembly Site Code per QSS 005-120

G4 = Green (Low Halogen and RoHS-compliant)

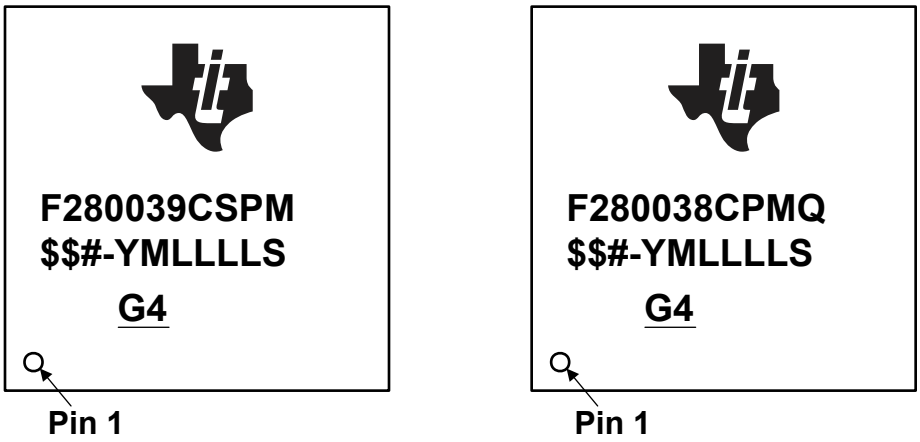
Figure 2-1. Package Symbolization for PZ Package



$$ = Wafer Fab Code (one or two characters)
# = Silicon Revision Code
YM = 2-digit Year/Month Code
LLLL = Assembly Lot Code
S = Assembly Site Code per QSS 005-120

G4 = Green (Low Halogen and RoHS-compliant)

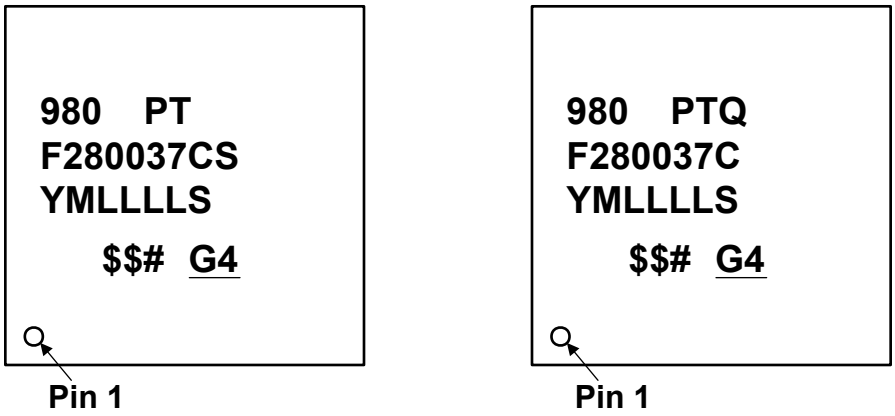
Figure 2-2. Package Symbolization for PN Package



$$ = Wafer Fab Code (one or two characters)
# = Silicon Revision Code
YM = 2-digit Year/Month Code
LLLL = Assembly Lot Code
S = Assembly Site Code per QSS 005-120

G4 = Green (Low Halogen and RoHS-compliant)

Figure 2-3. Package Symbolization for PM Package



980 = TI EIA Code
YM = 2-digit Year/Month Code
LLLL = Assembly Lot Code
S = Assembly Site Code per QSS 005-120
$$ = Wafer Fab Code (one or two characters)
= Silicon Revision Code

G4 = Green (Low Halogen and RoHS-compliant)

Figure 2-4. Package Symbolization for PT Package

Table 2-1. Revision Identification

SILICON REVISION CODE	SILICON REVISION	REVID(1) Address: 0x5D00C	COMMENTS(2)
Blank	0	0x0000 0000	This silicon revision is available as TMX and TMS.

(1) Silicon Revision ID
(2) For orderable device numbers, see the PACKAGING INFORMATION table in the *TMS320F28003x Real-Time Microcontrollers* data sheet.

3 Silicon Revision 0 Usage Notes and Advisories

3.1 Silicon Revision 0 Usage Notes

This section lists all the usage notes that are applicable to silicon revision 0.

3.1.1 PIE: Spurious Nested Interrupt After Back-to-Back PIEACK Write and Manual CPU Interrupt Mask Clear

Revision Affected: 0

Certain code sequences used for nested interrupts allow the CPU and PIE to enter an inconsistent state that can trigger an unwanted interrupt. The conditions required to enter this state are:

1. A PIEACK clear is followed immediately by a global interrupt enable (EINT or asm(" CLRC INTM")).
2. A nested interrupt clears one or more PIEIER bits for its group.

Whether the unwanted interrupt is triggered depends on the configuration and timing of the other interrupts in the system. This is expected to be a rare or nonexistent event in most applications. If it happens, the unwanted interrupt will be the first one in the nested interrupt's PIE group, and will be triggered after the nested interrupt reenables CPU interrupts (EINT or asm(" CLRC INTM")).

Workaround: Add a NOP between the PIEACK write and the CPU interrupt enable. Example code is shown below.

```
//Bad interrupt nesting code
PieCtrlRegs.PIEACK.all = 0xFFFF;      //Enable nesting in the PIE
EINT;                                 //Enable nesting in the CPU

//Good interrupt nesting code
PieCtrlRegs.PIEACK.all = 0xFFFF;      //Enable nesting in the PIE
asm(" NOP");                          //Wait for PIEACK to exit the pipeline
EINT;                                 //Enable nesting in the CPU
```

3.1.2 Caution While Using Nested Interrupts With Repeat Block

Revision Affected: 0

If the user is enabling interrupts using the EINT instruction inside an interrupt service routine (ISR) in order to use the nesting feature, then the user must disable the interrupts before exiting the ISR by using the DINT assembly instruction. Failing to do so may cause the bits in the RB register to not be restored correctly, resulting in undefined code behavior.

If the RPTB ASM instruction is not used inside the application, then there is no issue. In the case of C code source, analysis of the generated disassembly would need to be performed to verify this.

If the ISR is coded in C, then the C28x C compiler may take care of the above and no action is required. If the ISR is coded in C28x assembly language, then the above guidance must be followed.

Note

CGT v15.12.2.LTS (released in April 2016) or newer CGT packages addresses this requirement automatically. DINT only needs to be added for earlier revisions of the CGT tools.

3.1.3 I2C Slave Mode: Managing Status Flags in Applications with Long Interrupt Response Times

Revision Affected: 0

If the application delays servicing of an I2C interrupt while I2C bus activity continues to progress, future bus events can cause multiple status flags in the I2C Status Register (I2CSTR) to become active concurrently. If more than one of these interrupt events is enabled in I2CIER, future interrupts could be lost.

- This doesn’t apply when operating as an I2C master.
- This doesn’t apply if each I2CSTR flag is cleared by software before another gets set.

Workaround: Poll the I2CISRC.INTCODE field and execute the corresponding action for each interrupt source until the I2CISRC.INTCODE field has a value of 0 (no interrupt pending).

```
__interrupt void I2CA_ISR(void){
    uint16_t isrc;
    while((isrc = I2C_getInterruptSource(I2CA_BASE)) != I2C_INTSRC_NONE){
        switch(isrc) {
            case I2C_INTSRC_ARB_LOST:
                    /* ARBLINT ISR code */
                    break;
            case I2C_INTSRC_NO_ACK:
                    /* NACKINT ISR code */
                    break;
            case I2C_INTSRC_REG_ACCESS_RDY:
                    /* ARDYINT ISR code */
                    I2C_clearStatus(I2CA_BASE, I2C_STS_REG_ACCESS_RDY);
                    break;
            case I2C_INTSRC_RX_DATA_RDY:
                    /* RRDYINT ISR code, including I2C_getData() call */
                    break;
            case I2C_INTSRC_TX_DATA_RDY:
                    /* XRDYINT ISR code, including I2C_putData() call */
                    break;
            case I2C_INTSRC_STOP_CONDITION:
                    /* SCDINT ISR code */
                    break;
            case I2C_INTSRC_ADDR_TARGET:
                    /* AASINT ISR code */
                    break;
        }
        NOP;
        NOP;
    }
    Interrupt_clearACKGroup();
}
```

3.1.4 Security: The primary layer of defense is securing the boundary of the chip, which begins with enabling JTAGLOCK and Zero-pin Boot to Flash feature

Revisions Affected: 0

Device security relies on the premise that unauthorized code is not allowed to enter the device and execute under any circumstances. To that end, the device provides two features that a user concerned about security should always enable.

- **JTAGLOCK**

  When enabled in the USER OTP area of flash, the JTAGLOCK feature disables JTAG access (for example, debugger connection) to resources on the device, blocking an unauthorized party from using the JTAG interface to download any code into the device. When JTAGLOCK is enabled, the user can still allow an authorized party to unlock it by entering a password, or they can lock it permanently by programming a password value of all all-zeros.

- **Zero-pin Boot to Flash**

  The external bootloaders built into the TI ROM do not perform any authentication of the downloaded code. Enabling the Zero-pin boot option along with a flash boot mode in the USER OTP blocks all pin-based external bootloader options (for example, SCI, CAN, Parallel) from running at boot by forcing the boot process to jump immediately to internal flash after the base boot ROM execution concludes. For highest security, the Secure Flash boot mode can be chosen. This enables a pre-check of the flash code by the base boot ROM before jumping to it.

If JTAG is locked permanently and the Zero-pin Boot to Flash option is enabled, programming tools that communicate with the device through JTAG or the built-in bootloaders will not work. If the ability to perform firmware upgrades is desired, the user must pre-store code in flash to securely manage and perform the update.

3.2 Silicon Revision 0 Advisories

This section lists all the advisories that are applicable to silicon revision 0.

Advisory	***ADC: Interrupts may Stop if INTxCONT (Continue-to-Interrupt Mode) is not Set***
Revision Affected	0
Details	If ADCINTSELxNx[INTxCONT] = 0, then interrupts will stop when the ADCINTFLG is set and no additional ADC interrupts will occur. When an ADC interrupt occurs simultaneously with a software write of the ADCINTFLGCLR register, the ADCINTFLG will unexpectedly remain set, blocking future ADC interrupts.
Workaround	1. Use Continue-to-Interrupt Mode to prevent the ADCINTFLG from blocking additional ADC interrupts: 2. Ensure there is always sufficient time to service the ADC ISR and clear the ADCINTFLG before the next ADC interrupt occurs to avoid this condition. 3. Check for an overflow condition in the ISR when clearing the ADCINTFLG. Check ADCINTOVF immediately after writing to ADCINTFLGCLR; if it is set, then write ADCINTFLGCLR a second time to ensure the ADCINTFLG is cleared. The ADCINTOVF register will be set, indicating an ADC conversion interrupt was lost.

```
ADCINTSEL1N2[INT1CONT] = 1;
ADCINTSEL1N2[INT2CONT] = 1;
ADCINTSEL3N4[INT3CONT] = 1;
ADCINTSEL3N4[INT4CONT] = 1;
```

```
AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1;        //clear INT1 flag
if(1 == AdcaRegs.ADCINTOVF.bit.ADCINT1)       //ADCINT overflow
{
    AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1;    //clear INT1 again
// If the ADCINTOVF condition will be ignored by the application
// then clear the flag here by writing 1 to ADCINTOVFCLR.
// If there is a ADCINTOVF handling routine, then either insert
// that code and clear the ADCINTOVF flag here or do not clear
// the ADCINTOVF here so the external routine will detect the
// condition.
//    AdcaRegs.ADCINTOVFCLR.bit.ADCINT1 = 1;  // clear OVF
}
```

Advisory ***ADC: Degraded ADC Performance With ADCCLK Fractional Divider***

Revision Affected

0

Details

Using fractional SYSCLK-to-ADCCLK dividers (controlled by the ADCCTL2.PRESCALE field) has been shown to cause degradation in ADC performance on this device. See Table 3-1.

Table 3-1. ADCCTL2 Register

REDUCED PERFORMANCE			
BIT	**FIELD**	**VALUE**	**DESCRIPTION**
3–0	PRESCALE	0001	ADCCLK = SYSCLK/1.5
		0003	ADCCLK = SYSCLK/2.5
		...	
NORMAL PERFORMANCE			
BIT	**FIELD**	**VALUE**	**DESCRIPTION**
3–0	PRESCALE	0000	ADCCLK = SYSCLK/1.0
		0002	ADCCLK = SYSCLK/2.0
		...	

Workaround

Use even PRESCALE clock divider values. Even PRESCALE values result in integer clock dividers which do not impact the ADC performance.

Advisory ***ADC: DMA Read of Stale Result***

Revision Affected

0

Details

The ADCINT flag can be set before the ADCRESULT value is latched (see the t_{LAT} and $t_{INT(LATE)}$ columns in the ADC Timings table of the *TMS320F28003x Real-Time Microcontrollers* data sheet). The DMA can read the ADCRESULT value as soon as 3 cycles after the ADCINT trigger is set. As a result, the DMA could read a prior ADCRESULT value when the user expects the latest result if all of the following are true:

- The ADC is in late interrupt mode.
- The ADC operates in a mode where $t_{INT(LATE)}$ occurs 3 or more cycles before t_{LAT} (ADCCTL2 [PRESCALE] > 2).
- The DMA is triggered from the ADCINT signal.
- The DMA immediately reads the ADCRESULT value associated with that ADCINT signal without reading any other values first.
- The DMA was idle when it received the ADCINT trigger.

Only the DMA reads listed above could result in reads of stale data; the following non-DMA methods will always read the expected data:

- The ADCINT flag triggers a CLA task.
- The ADCINT flag triggers a CPU ISR.
- The CPU polls the ADCINT flag.

Workaround

Trigger two DMA channels from the ADCINT flag. The first channel acts as a dummy transaction. This will result in enough delay that the second channel will always read the fresh ADC result.

Advisory	***Analog Subsystem: Writes to Bit 1 of the ADCDACLOOPBACK Register are not Functional***
Revisions Affected	0
Details	The register bit ADCDACLOOPBACK.ENLB2ADCB is non-functional, and writes to the bit have no effect. Consequently, the ADC-DAC loopback functionality for ADCB cannot be activated using this register.
Workaround	• To test the functionality of the ADC as a diagnostic safety measure, use one of the two buffered DACs to output a known voltage, and sample the DAC voltage using ADCB. The output of DACA is on the same pin as channel B15, and the output of DACB is on B7. • Use the INTERNALTEST/ADC gain balancing mechanism to sample 0.9*VREFHI. For more details, see the Internal Test Mode section in the Analog-to-Digital Converter (ADC) chapter of the *TMS320F28003x Real-Time Microcontrollers Technical Reference Manual* .

Advisory	***BOR: VDDIO Between 2.45 V and 3.0 V can Result in Multiple XRSn Pulses***
Revision Affected	0
Details	The BOR can generate repeating XRSn assertions and deassertions when the VDDIO supply voltage is between 2.45 V and 3.0 V. It is recommended that the XRSn pin *not* be used directly as a reset to any other devices in the system. The F28003x BOR is effective for internally holding the device in a known reset state, even when these XRSn pulses are occurring. The device will not branch to application code or bootloaders, and all other pins will be held in their reset state until the VDDIO supply rises above 3.0 V.
Workarounds	1. Ignore the extra XRSn transitions during power up, power down, and BOR events. The extra XRSn pulses will have no effect on the F28003x device operation itself. 2. If XRSn pulses would cause undesired system behavior with other system components, then do not use XRSn to drive other devices. An external voltage supervisor can be used for these applications. 3. For applications that need to avoid these pulses during normal power up and power down: a. Power up: Follow the SR_{SUPPLY} requirement in the Recommended Operating Conditions table of the *TMS320F28003x Real-Time Microcontrollers* data sheet; no extra XRSn low pulses will occur. b. Power Down: To avoid any deassertion of XRSn during power down, design the power supply so that VDDIO passes through the range from 3.0 V to 2.45 V within 25 µs. If some voltage rise on XRSn is acceptable, then the time constant of the RC circuit implemented on XRSn can be calculated to ensure the voltage does not rise above a system-specified threshold.

Advisory ***C28: Fast Integer Division (FID) and DIVF64 Incorrect Results***

Revision Affected

0

Details

Incorrect results can occur for certain 2-cycle FPU instructions when using compiler versions before 22.6.3.LTS or 21.6.2.LTS.

When the below affected 2p FPU Instructions are followed by any of the FID or DIVF64 instructions with less than two delay slots, register corruption can occur which leads to incorrect results.

2-cycle (2p) FPU Instructions which need two delay slots before a FID or DIVF64 instruction:

- ADDF32
- SUBF32
- MPYF32
- MACF32

Fast Integer Division (FID) instructions:

- ABSI32DIV32
- ABSI32DIV32U
- ABSI64DIV32
- ABSI64DIV32U
- ABSI64DIV64
- ABSI64DIV64U
- ENEGI32DIV32
- ENEGI64DIV32
- ENEGI64DIV64
- MNEGI32DIV32
- MNEGI64DIV32
- MNEGI64DIV64
- NEGI32DIV32
- NEGI64DIV32
- NEGI64DIV64
- SUBC2UI64
- SUBC4UI32

DIVF64 instructions:

- PREDIVF64
- SUBC3F64
- POSTDIVF64

Example of issue:

The issue can be reproduced with a sequence like:

```
ADDF32    R0H, R0H, #32768    ; 2p FPU instruction
NOP                           ; Only one delay slot (insufficient)
ABSI32DIV32 R2H, R1H, R3H     ; Fast Integer Division instruction
```

In this case, the value in R0H can leak into R1H, causing incorrect results. Adding an additional NOP resolves the issue:

```
ADDF32    R0H, R0H, #32768    ; 2p FPU instruction
NOP                           ; First delay slot
NOP                           ; Second delay slot (required)
```

Advisory (continued) ***C28: Fast Integer Division (FID) and DIVF64 Incorrect Results***

```
ABSI32DIV32 R2H, R1H, R3H     ; Fast Integer Division instruction
```

Workarounds

1. Update to compiler version 22.6.3.LTS, 21.6.2.LTS or later which has fixed this issue.
2. If using assembly code, ensure two delay slots between affected instruction combinations.

Advisory	***CMPSS: COMPxLATCH May Not Clear Properly Under Certain Conditions***
Revisions Affected	0

Details

The CMPSS latched path is designed to retain a tripped state within a local latch (COMPxLATCH) until it is cleared by software (via COMPSTSCLR) or by PWMSYNC.

COMPxLATCH is set indirectly by the comparator output after the signal has been digitized and qualified by the Digital Filter. The maximum latency expected for the comparator output to reach COMPxLATCH may be expressed in CMPSS module clock cycles as:

LATENCY = 1 + (1 x FILTER_PRESCALE) + (FILTER_THRESH x FILTER_PRESCALE)

When COMPxLATCH is cleared by software or by PWMSYNC, the latch itself is cleared as desired, but the data path prior to COMPxLATCH may not reflect the comparator output value for an additional LATENCY number of module clock cycles. If the Digital Filter output resolves to a logical 1 when COMPxLATCH is cleared, the latch will be set again on the following clock cycle.

Workarounds

Allow the Digital Filter output to resolve to logical 0 before clearing COMPxLATCH.

If COMPxLATCH is cleared by software, the output state of the Digital Filter can be confirmed through the COMPSTS register prior to clearing the latch. For instances where a large LATENCY value produces intolerable delays, the filter FIFO may be flushed by reinitializing the Digital Filter (via CTRIPxFILCTL).

If COMPxLATCH is cleared by PWMSYNC, the user application should be designed such that the comparator trip condition is cleared at least LATENCY cycles before PWMSYNC is generated.

Advisory ***CMPSS: A CMPSS Glitch can Occur if Comparator Input Pin has AGPIO Functionality and ADC is Sampling the Input Pin***

Revisions Affected

0

Details

The combinations of use cases for a specific analog input pin that need special considerations are shown in Table 3-2. As shown in this table, special considerations or workarounds need to be used for the combination of CMPSS Input, ADC Sampling, and AGPIO.

Table 3-2. Combinations of Use Cases for a Specific Analog Input Pin

FUNCTION USED ON A SPECIFIC ANALOG PIN	COMPONENT USED				
CMPSS Comparator Input	Yes	-	Yes	-	Yes
ADC Sampling	Yes	Yes	-	Yes	Yes
AGPIO Analog Pin Type	Yes	Yes	Yes	-	-
AIO Analog Pin Type	-	-	-	Yes	Yes
Result	**Workaround needed**	**No special analysis or workaround needed**			

The AGPIO analog pin path contains an extra series switch of 53Ω. This creates a low-capacitance isolated node shared by the ADC and CMPSS comparator, as shown in Figure 3-1. This node can be disturbed when the ADC samples the channel (depending on the prior voltage stored on the ADC sample-and-hold capacitor), and this disturbance can cause a false CMPSS event of up to 50ns. To accommodate this potential disturbance, the workarounds below can be implemented.

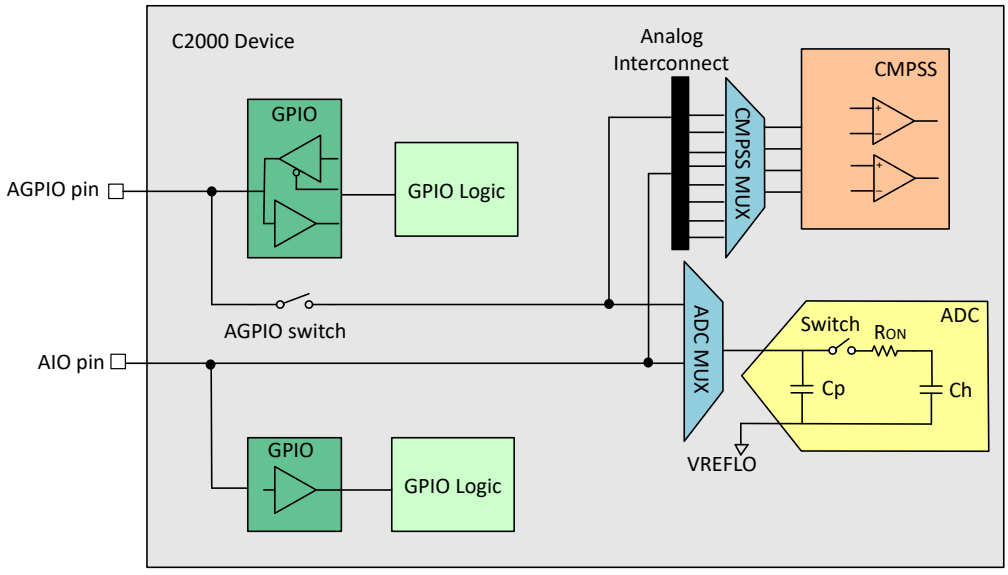


Figure 3-1. Analog Subsystem Diagram with AGPIO and AIO Analog Pin Types

Workarounds

1. Use a different pin (that is AIO pin type) for analog channels that need both ADC and CMPSS together.
2. Use the CMPSS Digital Filter with a setting of 50ns or greater, which will filter the temporary disturbance.
3. Precondition the sample-and-hold capacitor of the ADC so that the disturbance will not cause a false trip. For example, perform a dummy read of a 3.3V connection from a different channel on the ADC immediately before the impacted channel is read,

Advisory (continued) ***CMPSS: A CMPSS Glitch can Occur if Comparator Input Pin has AGPIO Functionality and ADC is Sampling the Input Pin***

so the disturbance is in the positive direction, away from the false trip. The opposite dummy read of a 0V signal would be used if the false trip is inverted in polarity.

Advisory	***DCAN: During DCAN FIFO Mode, Received Messages May be Placed Out of Order in the FIFO Buffer***
Revision Affected	0
Details	In DCAN FIFO mode, received messages with the same arbitration and mask IDs are supposed to be placed in the FIFO in the order in which they are received. The CPU then retrieves the received messages from the FIFO via the IF1/IF2 interface registers. Some messages may be placed in the FIFO out of the order in which they were received. If the order of the messages is critical to the application for processing, then this behavior will prevent the proper use of the DCAN FIFO mode.
Workaround	Use the DMA to read out the FIFO via the IF3 register. Each time a message is received into the FIFO, the data is also copied to the IF3 register, and a DMA request Is generated to the DMA module to read out the data.

Advisory	***MCAN: Message Order Inversion When Transmitting From Dedicated Tx Buffers Configured With Same Message ID***
Revision Affected	0
Details	Multiple Tx Buffers are configured with the same Message ID. Transmission of these Tx buffers is requested sequentially in ascending order with a delay between the individual Tx requests. Depending on the delay between the individual Tx requests, the Tx Buffers may not be transmitted in the expected ascending order of the Tx Buffer number.
Workarounds	First, write the group of Tx messages with same Message ID to the Message RAM. Then, request transmission of all of these messages concurrently by a single write access to **TXBAR**. Use the Tx FIFO instead of dedicated Tx Buffers for the transmission of several messages with the same Message ID in a specific order.

Advisory ***ePWM: An ePWM Glitch can Occur if a Trip Remains Active at the End of the Blanking Window***

Revision Affected

0

Details

The blanking window is typically used to mask any PWM trip events during transitions which would be false trips to the system. If an ePWM trip event remains active for less than three ePWM clocks after the end of the blanking window cycles, there can be an undesired glitch at the ePWM output.

Figure 3-2 illustrates the time period which could result in an undesired ePWM output.

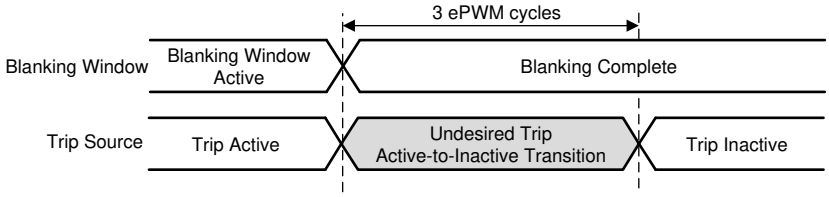


Figure 3-2. Undesired Trip Event and Blanking Window Expiration

Figure 3-3 illustrates the two potential ePWM outputs possible if the trip event ends within 1 cycle before or 3 cycles after the blanking window closes.

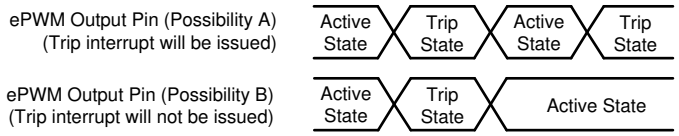


Figure 3-3. Resulting Undesired ePWM Outputs Possible

Workaround

Extend or reduce the blanking window to avoid any undesired trip action.

Advisory	***ePWM: ePWM One-Shot/CBC Trip Event DCxEVTy.force Does Not Set the Trip Condition***
Revisions Affected	0
Details	The DCxEVTy.force signal can be used as input for the one-shot trip events and cycle-by-cyle trip events. When setting DCxCTL[EVT1/2FRCSYNCSEL] = 1 (passes the DCxEVT signal through the async path) does not properly set the one-shot or cycle-by-cycle trip event condition and trip event is missed.
Workarounds	For devices with DCxCTL.EVT1/2FRCSYNCSEL and DCxCTL.EVT1/2LATSEL, there are two options. Otherwise, Sync Path is the only option to use for the one-shot or cycle-by-cycle trip event. **Async Path** (passes the DCxEVT signal through the async path) The following configurations must be made to latch the one-shot or cycle-by-cycle trip condition through the async path: To Trip the PWMs, you can invert the EPWMXBAR state using XBAR_invertEPWMSignal(). 1. DCxCTL.EVT1/2FRCSYNCSEL = 1 2. DCxCTL.EVT1/2LATSEL = 1 **Sync Path** (passes the DCxEVT signal through the sync path) 1. DCxCTL.EVT1/2FRCSYNCSEL = 0

Advisory	***ePWM: Trip Events Will Not be Filtered by the Blanking Window for the First 3 Cycles After the Start of a Blanking Window***
Revision Affected	0
Details	The Blanking Window will not blank trip events for the first 3 cycles after the start of a Blanking Window. DCEVTFILT may continue to reflect changes in the DCxEVTy signals. If DCEVTFILT is enabled, this may impact subsequent subsystems that are configured (for example, the Trip Zone submodule, TZ interrupts, ADC SOC, or the PWM output).
Workaround	Start the Blanking Window 3 cycles before blanking is required. If a Blanking Window is needed at a period boundary, start the Blanking Window 3 cycles before the beginning of the next period. This works because Blanking Windows persist across period boundaries.

Advisory	***eQEP: Position Counter Incorrectly Reset on Direction Change During Index***
Revision Affected	0
Details	While using the PCRM = 0 configuration, if the direction change occurs when the index input is active, the position counter (QPOSCNT) could be reset erroneously, resulting in an unexpected change in the counter value. This could result in a change of up to ±4 counts from the expected value of the position counter and lead to unexpected subsequent setting of the error flags. While using the PCRM = 0 configuration [that is, Position Counter Reset on Index Event (QEPCTL[PCRM] = 00)], if the index event occurs during the forward movement, then the position counter is reset to 0 on the next eQEP clock. If the index event occurs during the reverse movement, then the position counter is reset to the value in the QPOSMAX register on the next eQEP clock. The eQEP peripheral records the occurrence of the first index marker (QEPSTS[FIMF]) and direction on the first index event marker (QEPSTS[FIDF]) in QEPSTS registers. It also remembers the quadrature edge on the first index marker so that same relative quadrature transition is used for index event reset operation. If the direction change occurs while the index pulse is active, the module would still continue to look for the relative quadrature transition for performing the position counter reset. This results in an unexpected change in the position counter value. The next index event without a simultaneous direction change will reset the counter properly and work as expected.
Workarounds	Do not use the PCRM = 0 configuration if the direction change could occur while the index is active and the resultant change of the position counter value could affect the application. Other options for performing position counter reset, if appropriate for the application [such as Index Event Initialization (IEI)], do not have this issue.

Advisory ***FPU: FPU-to-CPU Register Move Operation Preceded by Any FPU 2p Operation***

Revision Affected

0

Details

This advisory applies when a multicycle (2p) FPU instruction is followed by a FPU-to-CPU register transfer. If the FPU-to-CPU read instruction source register is the same as the 2p instruction destination, then the read may be of the value of the FPU register before the 2p instruction completes. This occurs because the 2p instructions rely on data-forwarding of the result during the E3 phase of the pipeline. If a pipeline stall happens to occur in the E3 phase, the result does not get forwarded in time for the read instruction.

The 2p instructions impacted by this advisory are MPYF32, ADDF32, SUBF32, and MACF32. The destination of the FPU register read must be a CPU register (ACC, P, T, XAR0...XAR7). This advisory does not apply if the register read is a FPU-to-FPU register transfer.

In the example below, the 2p instruction, MPYF32, uses R6H as its destination. The FPU register read, MOV32, uses the same register, R6H, as its source, and a CPU register as the destination. If a stall occurs in the E3 pipeline phase, then MOV32 will read the value of R6H before the MPYF32 instruction completes.

Example of Problem:

```
   MPYF32 R6H, R5H, R0H  ; 2p FPU instruction that writes to R6H
|| MOV32 *XAR7++, R4H
   F32TOUI16R R3H, R4H   ; delay slot
   ADDF32 R2H, R2H, R0H
|| MOV32 *--SP, R2H      ; alignment cycle
   MOV32 @XAR3, R6H      ; FPU register read of R6H
```

Figure 3-4 shows the pipeline diagram of the issue when there are no stalls in the pipeline.

	Instruction	F1	F2	D1	D2	R1	R2	E	W		Comments
		FPU pipeline--->				R1	R2	E1	E2	E3	
I1	MPYF32 R6H, R5H, R0H || MOV32 *XAR7++, R4H	I1									
I2	F32TOUI16R R3H, R4H	I2	I1								
I3	ADDF32 R3H, R2H, R0H || MOV32 *--SP, R2H	I3	I2	I1							
I4	MOV32 @XAR3, R6H	I4	I3	I2	I1						
			I4	I3	I2	I1					
				I4	I3	I2	I1				
					I4	I3	I2	I1			
						I4	I3	I2	I1		
							I4	I3	I2	**I1**	I4 samples the result as it enters the R2 phase. The product R6H=R5H*R0H (I1) finishes computing in the E3 phase, but is **forwarded** as an operand to I4. This makes I4 appear to be a 2p instruction, but I4 actually takes 3p cycles to compute.
								I4	I3	I2	
									I4	I3	

Figure 3-4. Pipeline Diagram of the Issue When There are no Stalls in the Pipeline

Advisory (continued) ***FPU: FPU-to-CPU Register Move Operation Preceded by Any FPU 2p Operation***

Figure 3-5 shows the pipeline diagram of the issue if there is a stall in the E3 slot of the instruction I1.

	Instruction	F1	F2	D1	D2	R1	R2	E	W		Comments
		FPU pipeline-->				R1	R2	E1	E2	E3	
I1	MPYF32 R6H, R5H, R0H || MOV32 *XAR7++, R4H	I1									
I2	F32TOUI16R R3H, R4H	I2	I1								
I3	ADDF32 R3H, R2H, R0H || MOV32 *--SP, R2H	I3	I2	I1							
I4	MOV32 @XAR3, R6H	I4	I3	I2	I1						
			I4	I3	I2	I1					
				I4	I3	I2	I1				
					I4	I3	I2	I1			
						I4	I3	I2	I1		
							I4	I3	I2	I1 (STALL)	I4 samples the result as it enters the R2 phase, but I1 is stalled in E3 and is unable to forward the product of R5H*R0H to I4 (R6H does not have the product yet due to a design bug). So, I4 reads the old value of R6H.
							I4	I3	I2	I1	There is no change in the pipeline as it was stalled in the previous cycle. I4 had already sampled the old value of R6H in the previous cycle.
								I4	I3	I2	Stall over

Figure 3-5. Pipeline Diagram of the Issue if There is a Stall in the E3 Slot of the Instruction I1

Workaround

Treat MPYF32, ADDF32, SUBF32, and MACF32 in this scenario as 3p-cycle instructions. Three NOPs or non-conflicting instructions must be placed in the delay slot of the instruction.

The C28x Code Generation Tools v.6.2.0 and later will both generate the correct instruction sequence and detect the error in assembly code. In previous versions, v6.0.5 (for the 6.0.x branch) and v.6.1.2 (for the 6.1.x branch), the compiler will generate the correct instruction sequence but the assembler will not detect the error in assembly code.

Example of Workaround:

```
   MPYF32 R6H, R5H, R0H
|| MOV32 *XAR7++, R4H      ; 3p FPU instruction that writes to R6H
   F32TOUI16R R3H, R4H     ; delay slot
   ADDF32 R2H, R2H, R0H
|| MOV32 *--SP, R2H        ; delay slot
   NOP                     ; alignment cycle
   MOV32 @XAR3, R6H        ; FPU register read of R6H
```

Figure 3-6 shows the pipeline diagram with the workaround in place.

Advisory (continued) *FPU: FPU-to-CPU Register Move Operation Preceded by Any FPU 2p Operation*

	Instruction	F1	F2	D1	D2	R1	R2	E	W		Comments
		FPU pipeline-->				R1	R2	E1	E2	E3	
I1	MPYF32 R6H, R5H, R0H || MOV32 *XAR7++, R4H	I1									
I2	F32TOUI16R R3H, R4H	I2	I1								
I3	ADDF32 R3H, R2H, R0H || MOV32 *--SP, R2H	I3	I2	I1							
I4	NOP	I4	I3	I2	I1						
I5	MOV32 @XAR3, R6H	I5	I4	I3	I2	I1					
			I5	I4	I3	I2	I1				
				I5	I4	I3	I2	I1			
					I5	I4	I3	I2	I1		
						I5	I4	I3	I2	I1 (STALL)	Due to one extra NOP, I5 does not reach R2 when I1 enters E3; thus, forwarding is not needed.
						I5	I4	I3	I2	I1	There is no change due to the stall in the previous cycle.
							I5	I4	I3	I2	I1 moves out of E3 and I5 moves to R2. R6H has the result of R5H*R0H and is read by I5. There is no need to forward the result in this case.
								I5	I4	I3	

Figure 3-6. Pipeline Diagram With Workaround in Place

Advisory ***I2C: Target Transmitter Mode, Standard Mode SDA Timings Limitation***

Revision Affected

0

Details

The I2C peripheral present on the MCU is a Fast-mode device; it will clock-stretch the SCL (Clock) line when used with a Standard-mode host.

There is a requirement from the I2C Specification for a Fast-mode device used in a Standard-mode system to meet $t_{SU:DAT}$ (data set-up time) + $t_{r(max)}$ (rise time) before releasing the SCL line. See Footnote 4 of the "Characteristics of the SDA and SCL bus lines for Standard, Fast, and Fast-mode Plus I^2C-bus devices" table in the NXP Semiconductors *I^2C-bus specification and user manual* (UM10204).

However, the C2000 I2C clock-stretches the SCL line by a fixed amount = 6 * f_{mod} Clock (I2C Clock rate of the C2000) in the above scenario. When the C2000™ microcontroller is acting as a target transmitter with a Standard-mode host, it is possible for the clock line (SCL) to be released by the C2000 before the data (SDA) is ready, if the t_r of SDA is too long.

The "Pull-up resistor sizing" section in the NXP Semiconductors *I^2C-bus specification and user manual* (UM10204) gives more details on choosing the appropriate PU resistor (R_p), based on the rise time (t_r) and bus capacitance (C_b) shown in Equation 1.

$$R_{p(max)} = \frac{t_r}{0.8473 \times C_b} \quad (1)$$

Workaround

1. **Reducing t_r with a strong pullup**
   In order to ensure that $t_{SU:DAT}$ + $t_{r(max)}$ is met, the user can configure the pullup resistance on the SDA line such that it meets the constraints listed in the SDA Data Rise Time Requirement column of Table 3-3 based on the value of f_{mod} Clock in their system. This will ensure that the data present on the SDA line is valid when the C2000 releases the SCL signal.

   Table 3-4 gives suggested R_p resistor values for a given f_{mod} Clock (MHz) and C_b (bus capacitance). For other values of C_b, please use Equation 1 to calculate the value of R_p needed in the system.

Table 3-3. Data Rise Time Requirements for C2000 as Target Transmitter with Standard-Mode Host

f_{mod} Clock (MHz)	f_{mod} Period (ns)	SCL Clock-Stretch Delay from C2000 I2C (ns): (6*f_{mod} Clock)	Data Set-up Time (ns): $t_{SU:DAT}$ (Standard Mode)	SDA Data Rise Time Requirement (ns): t_r
7	142.9	857	250	607
8	125	750		500
9	111	666		416
10	100	600		350
11	90.9	545		295
12	83.3	500		250

Advisory (continued) ***I2C: Target Transmitter Mode, Standard Mode SDA Timings Limitation***

Table 3-4. Pullup Resistor (R_p) Values for Common Bus Capacitances (C_b)

f_{mod} Clock (MHz)	SDA Data Rise Time Requirement (ns): t_r	R_p (kΩ) for C_b = 100 pF	R_p (kΩ) for C_b = 200 pF	R_p (kΩ) for C_b = 300 pF	R_p (kΩ) for C_b = 400 pF
7	607	7.1	3.5	2.3	1.7
8	500	5.9	2.9	1.9	1.4
9	416	4.9	2.4	1.6	1.2
10	350	4.1	2.0	1.3	1.0
11	295	3.4	1.7	1.1	0.8
12	250	2.9	1.4	0.9	0.7

2. **t_r = 1000 ns**
   This workaround is not preferred due to restrictions in general I2C usage, use Workaround 1 when possible.
   If the system is such that it requires 1000 ns of rise time on the SDA line, the C2000 I2C f_{mod} Clock can be configured to 4.8 MHz so the clock-stretching (6 * f_{mod} Clock) satisfies this requirement. This results in t_r = (1/4.8 MHz) * 6 = 1000 ns. This workaround is only valid in systems where the C2000 I2C is the target on the I2C bus. Note that 4.8 MHz is outside the data sheet's required range of 7 MHz to 12 MHz for f_{mod} Clock. Using f_{mod} at 4.8 MHz, even though it is outside of the data sheet's required range, will work for the C2000 I2C in Target mode on a Standard-mode host bus. Using f_{mod} = 4.8 MHz in any other configurations except the one listed in this workaround will cause other timing parameters to be violated and is not allowed.

Advisory	***LIN: Inconsistent Sync Field Error (ISFE) Flag/Interrupt Not Set When Sync Field is Erroneous***
Revision Affected	0
Details	During LIN communications, if the Sync field received (on RX) is erroneous (that is, if the Sync field receives any value other than 0x55), the LIN does not set the ISFE Flag in the SCIFLR.ISFE register or trigger the ISFE interrupt. Communication gets terminated without data being received or the RX receive interrupt being set. There is no way for an application to detect an error in the Sync field. The application can detect if the Sync field is completely blank or if the Sync field is not received within the given tolerances (as explained in the *TMS320F28003x Real-Time Microcontrollers Technical Reference Manual*), but the application cannot detect any error in the value of Sync field.
Workarounds	**Method 1:** Keep polling the SCIFLR.RXRDY flag and time out if it is not set within a certain amount of time. Use the following steps as a guideline: 1. Poll for the SCIFLR.BUSY flag to set. 2. Once the BUSY flag goes high, poll for the SCIFLR.RXRDY flag. Concurrently within this loop, also have a SW timeout, which times out and exits the loop if the RXRDY flag is not set within a user-defined time interval. **Method 2:** Configure the CPU timer to interrupt if the RX interrupt is not triggered. This method does not use CPU bandwidth. Use the following steps as a guideline: 1. Configure XINT to trigger an ISR when the LINRX goes from high to low (indicating LIN is busy). 2. Inside the XINT ISR, configure the CPU timer, which starts timing the frame completion. 3. If the frame is received correctly with the correct Sync field, it should trigger the LIN RX ISR, inside which you can turn off the timer so that you do not get a false timeout. 4. If the frame is not received correctly, it does not trigger the LIN RX ISR but triggers the CPU timer ISR (timeout occurred), which indicates an error in the Sync field.

Advisory	***Memory: Prefetching Beyond Valid Memory***
Revision Affected	0
Details	The C28x CPU prefetches instructions beyond those currently active in its pipeline. If the prefetch occurs past the end of valid memory, then the CPU may receive an invalid opcode.
Workaround	**M1, GS3 –** The prefetch queue is 8 x16 words in depth. Therefore, code should not come within 8 words of the end of valid memory. Prefetching across the boundary between two valid memory blocks is all right. Example 1: M1 ends at address 0x7FF and is not followed by another memory block. Code in M1 should be stored no farther than address 0x7F7. Addresses 0x7F8–0x7FF should not be used for code. Example 2: M0 ends at address 0x3FF and valid memory (M1) follows it. Code in M0 can be stored up to and including address 0x3FF. Code can also cross into M1, up to and including address 0x7F7. **Flash –** The prefetch queue is 16 x16 words in depth. Therefore, code should not come within 16 words of the end of valid memory; otherwise, it generates a Flash ECC uncorrectable error.

Table 3-5. Memories Impacted by Advisory

MEMORY TYPE	ADDRESSES IMPACTED
M1	0x0000 07F8–0x0000 07FF
GS3	0x0000 FFF8–0x0000 FFFF
Flash	0x000A FFF0–0x000A FFFF

Advisory	***Boot ROM: Data Overrun With MCAN Bootloader on TMX Devices***
Revision Affected	0 *(TMX only)*
Details	In TMX devices, when the MCAN bootloader is used, communication data overrun occurs on the F28003x side, leading to bootloader failure.
Workaround	The host needs to ensure a delay of 400 µs between successive frames to ensure successful communication. This issue is fixed in TMS devices.

Advisory	***Secure Live Firmware Update (LFU) Boot Modes are Deprecated***
Revision Affected	0
Details	Secure LFU boot mode, if configured, will result in device reset if the memories have been secured.
Workaround	The latest firmware bank selection algorithm can be implemented as part of the flash custom bootloader instead of calling from ROM. This same custom bootloader can call the CMAC authentication algorithm (located in secure ROM) for flash content authentication after the latest firmware bank selection. The flash memory where this custom bootloader is located should be configured as Z1 secure memory.

Advisory	***Flash: Execution of Fapi_setActiveFlashBank() Without Disabling Flash Prefetch may Cause ITRAP***
Revisions Affected	0
Details	Flash prefetch mechanism should be disabled only by the code executing from RAM and only when there is not any active fetch access to flash. Fapi_setActiveFlashBank() disables the prefetch mechanism. Any flash execution can result in ITRAP when the prefetch-disable process is in progress; therefore, prefetch should be disabled before executing Fapi_setActiveFlashBank().
Workarounds	Disable the prefetch mechanism before calling Fapi_setActiveFlashBank(). Prefetch can be enabled after the execution of the Fapi_setActiveFlashBank(). Code that disables and enables the prefetch mechanism should be executed from RAM. When this workaround is implemented, note that Fapi_setActiveFlashBank() can be executed from flash (not from the bank on which the erase/program operations are targeted) since prefetch is already disabled. Flash API usage examples provided in C2000Ware_5_02_00_00 (and onwards) depict the implementation of this workaround. Path to the TMS320F28003x example: C2000Ware_5_02_00_00\driverlib\f28003x\examples\flash\flashapi_ex1_programming.c In this example, Flash_DisablePrefetch_SW_Workaround() is executed from RAM to disable the prefetch before calling Fapi_setActiveFlashBank(). Flash_EnablePrefetch_SW_Workaround() is executed from RAM to enable the prefetch after calling Fapi_setActiveFlashBank().

Advisory	***SYSTEM: HIC Illegal Read Error Flag Does not Get Asserted in Pagesel=0 Mode***
Revision Affected	0
Details	When a Host Read access is initiated to the same address of a pending write location (an illegal access sequence), the Illegal Read error flag does not get asserted in Pagesel=0 Mode. The error flag gets set for the same sequence in Pagesel=1 mode. The impact is low since it is an illegal sequence and SW is not expected to initiate a read to a write-pending location in the regular application flow.
Workaround	None

Advisory	***SYSTEM: Multiple Successive Writes to CLKSRCCTL1 Can Cause a System Hang***
Revision Affected	0
Details	When the CLKSRCCTL1 register is written more than once without delay between writes, the system can hang and can only be recovered by an external XRSn reset or Watchdog reset. The occurrence of this condition depends on the clock ratio between SYSCLK and the clock selected by OSCCLKSRCSEL, and may not occur every time. If this issue is encountered while using the debugger, then after hitting pause, the program counter will be at the Boot ROM reset vector. Implementing the workaround will avoid this condition for any SYSCLK to OSCCLK ratio.

Workaround	Add a software delay of 300 SYSCLK cycles using an NOP instruction after every write to the CLKSRCCTL1 register.
	Example:
	`ClkCfgRegs.CLKSRCCTL1.bit.INTOSC2OFF=0;  // Turn on INTOSC2`
	`asm("  RPT #250 || NOP");  // Delay of 250 SYSCLK Cycles`
	`asm("  RPT #50 || NOP");  // Delay of 50 SYSCLK Cycles`
	`ClkCfgRegs.CLKSRCCTL1.bit.OSCCLKSRCSEL = 0;  // Clk Src = INTOSC2`
	`asm("  RPT #250 || NOP");  // Delay of 250 SYSCLK Cycles`
	`asm("  RPT #50 || NOP");  // Delay of 50 SYSCLK Cycles`
	C2000Ware_3_00_00_00 and later revisions will have this workaround implemented.

Advisory	***Avoiding Spurious Interrupts While Using HWBIST***
Revisions Affected	0
Details	HWBIST has the capability to log interrupts that are received while the CPU is under test and re-issue them after HWBIST completes. Interrupts that are received in the clock cycle before the interrupt logging is enabled will be executed before the HWBIST runs. In the next cycle, once interrupt logging is enabled, interrupts will be logged and re-issued once the HWBIST completes. The interrupt events for CPU Timer 1 and CPU Timer 2 are valid for 2 SYSCLK cycles. If the first cycle happens a cycle before interrupt logging is enabled and the second cycle coincides with the enabling of interrupt logging, the interrupt will be executed once before the logging (clearing the CPU Timer TCR.TIF flag), but then will be logged by the interrupt logger and triggered again after HWBIST completes. Since the TCR.TIF flag was already cleared by the previous ISR, this is an unexpected spurious interrupt. Note that this scenario is only applicable to the non-PIE CPU Timer interrupts. The CPU Timer 0 interrupt is managed by the PIE and its pulse width is only one SYSCLK cycle.
Workaround	Disable CPU Timer 1 and 2 interrupts before enabling interrupt logging and restore them later. This workaround is already implemented in the C2000 Software Diagnostic Library in the stl_hwbist_s.asm file. The steps are: 1. Clear the timer interrupt enable bit TCR.TIE for CPU Timers 1 and 2. 2. Run normal HWBIST sequence: save registers, enable interrupt logging, run HWBIST, restore registers, end interrupt logging. 3. Check if the CPU Timers' TCR.TIF flags are set. If the flags are set, set the corresponding CPU IFR bit to trigger the interrupt. 4. Restore TCR.TIE.

Advisory	***PLL Reference Clock Lost Detection: Missing Clock Flag may be Incorrectly Activated***
Revisions Affected	0
Details	The register bit SYSPLLSTS.REF_LOSTS may be incorrectly set, falsely indicating a reference clock loss.
Workaround	Do not use the PLL Reference Clock Lost Detection feature. As an alternative, use the Missing Clock Detect (MCD) feature or the Dual-Clock Comparator (DCC) to detect reference clock loss. • For the MCD method, refer to C2000Ware example sysctl_ex1_missing_clock_detection under the sysctl folder. • For the DCC method, refer to C2000Ware example dcc_ex4_clock_fail_detect under the dcc folder.

Advisory	***SDFM: Dynamically Changing Threshold Settings (LLT, HLT), Filter Type, or COSR Settings Will Trigger Spurious Comparator Events***
Revision Affected	0
Details	When SDFM comparator settings—such as filter type, lower/upper threshold, or comparator OSR (COSR) settings—are dynamically changed during run time, spurious comparator events will be triggered. The spurious comparator event will trigger a corresponding CPU interrupt, CLA task, ePWM X-BAR events, and GPIO output X-BAR events if configured appropriately.
Workaround	When comparator settings need to be changed dynamically, follow the procedure below to ensure spurious comparator events do not generate a CPU interrupt, CLA task, or X-BAR events (ePWM X-BAR/GPIO output X-BAR events): 1. Disable the comparator filter. 2. Delay for at least a latency of the comparator filter + 3 SD-Cx clock cycles. 3. Change comparator filter settings such as filter type, COSR, or lower/upper threshold. 4. Delay for at least a latency of the comparator filter + 5 SD-Cx clock cycles. 5. Enable the comparator filter.

Advisory	***SDFM: Dynamically Changing Data Filter Settings (Such as Filter Type or DOSR) Will Trigger Spurious Data Acknowledge Events***
Revision Affected	0
Details	When SDFM data settings—such as filter type or DOSR settings—are dynamically changed during run time, spurious data-filter-ready events will be triggered. The spurious data-ready event will trigger a corresponding CPU interrupt, CLA task, and DMA trigger if configured appropriately.
Workaround	When SDFM data filter settings need to be changed dynamically, follow the procedure below to ensure spurious data-filter-ready events are not generated: 1. Disable the data filter. 2. Delay for at least a latency of the data filter + 3 SD-Cx clock cycles. 3. Change data filter settings such as filter type and DOSR. 4. Delay for at least a latency of the data filter + 5 SD-Cx clock cycles. 5. Enable the data filter.

Advisory	***SDFM: Two Back-to-Back Writes to SDCPARMx Register Bit Fields CEVT1SEL, CEVT2SEL, and HZEN Within Three SD-Modulator Clock Cycles can Corrupt SDFM State Machine, Resulting in Spurious Comparator Events***
Revision Affected	0
Details	Back-to-back writes to SDCPARMx register bit fields CEVT1SEL, CEVT2SEL, and HZEN within three SD-modulator clock cycles can potentially corrupt the SDFM state machine, resulting in spurious comparator events, which can potentially trigger CPU interrupts, CLA tasks, ePWM XBAR events, and GPIO output X-BAR events if configured appropriately.
Workaround	Avoid back-to-back writes within three SD-modulator clock cycles or have the SDCPARMx register bit fields configured in one register write.

Advisory	*Watchdog: WDKEY Register is not EALLOW-Protected*
Revision Affected	0
Details	The WDKEY register is not EALLOW-protected. Issuing the EALLOW and EDIS instructions to write to this register is not required. To enable software reuse on other devices where WDKEY is EALLOW-protected, using EALLOW and EDIS is recommended.
Workaround	None

4 Documentation Support

For device-specific data sheets and related documentation, visit the TI web site at: https://www.ti.com.

For more information regarding the TMS320F28003x devices, see the following documents:

- *TMS320F28003x Real-Time Microcontrollers* data sheet
- *TMS320F28003x Real-Time Microcontrollers Technical Reference Manual*

5 Trademarks

C2000™ is a trademark of Texas Instruments.
All trademarks are the property of their respective owners.

6 Revision History

Changes from May 30, 2024 to August 31, 2026 (from Revision D (May 2024) to Revision E (August 2026)) **Page**

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), 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 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, TI’s General Quality Guidelines, 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. Unless TI explicitly designates a product as custom or customer-specified, TI products are standard, catalog, general purpose devices.

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

Last updated 10/2025