

# **TMS320F2837xS Delfino™ MCUs Silicon Revisions C, B**

---

---

---

## **1 Introduction**

This document describes the silicon updates to the functional specifications for the TMS320F2837xS microcontrollers (MCUs).

The updates are applicable to the following:

- 337-ball New Fine Pitch Ball Grid Array, ZWT Suffix
- 176-pin PowerPAD™ Thermally Enhanced Low-Profile Quad Flatpack, PTP Suffix
- 100-Pin PowerPAD Thermally Enhanced Thin Quad Flatpack, PZP Suffix

## **2 Device and Development Support Tool Nomenclature**

To designate the stages in the product development cycle, TI assigns prefixes to the part numbers of all [TMS320] DSP devices and support tools. Each TMS320™ DSP commercial family member has one of three prefixes: TMX, TMP, or TMS (for example, **TMS**320F28379S). 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 (with TMX for devices and TMDX for tools) through fully qualified production devices and tools (with TMS for devices and TMDS for tools).

<b>TMX</b>	Experimental device that is not necessarily representative of the final device's electrical specifications
<b>TMP</b>	Final silicon die that conforms to the device's electrical specifications but has not completed quality and reliability verification
<b>TMS</b>	Fully qualified production device

Support tool development evolutionary flow:

<b>TMDX</b>	Development-support product that has not yet completed Texas Instruments internal qualification testing
<b>TMDS</b>	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."

TMS 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 (TMX or TMP) 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.

TI device nomenclature also includes a suffix with the device family name. This suffix indicates the package type (for example, PTP) and temperature range (for example, T).

### 3 Device Markings

Figure 1 provides an example of the 2837xS device markings and defines each of the markings. The device revision can be determined by the symbols marked on the top of the package as shown in Figure 1. Some prototype devices may have markings different from those illustrated. Figure 2 shows an example of the device nomenclature.

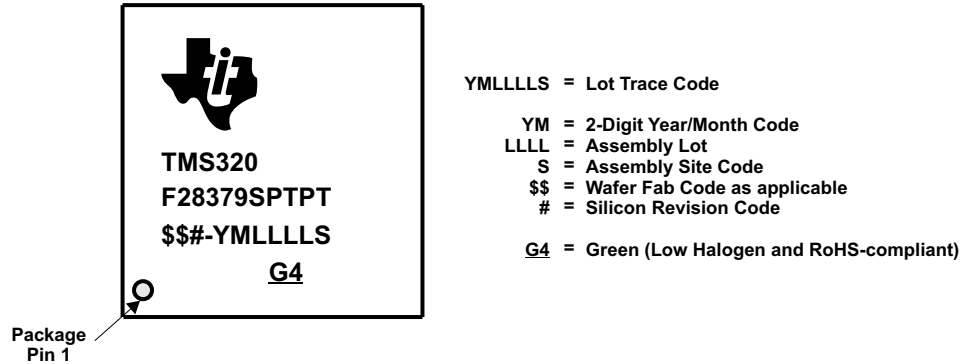


Figure 1. Example of Device Markings

Table 1. Determining Silicon Revision From Lot Trace Code

SILICON REVISION CODE	SILICON REVISION	REVID <sup>(1)</sup> Address: 0x5D00C	COMMENTS
B	B	0x0002	This silicon revision is available as TMX.
C	C	0x0003	This silicon revision is available as TMS.

<sup>(1)</sup> Silicon Revision ID

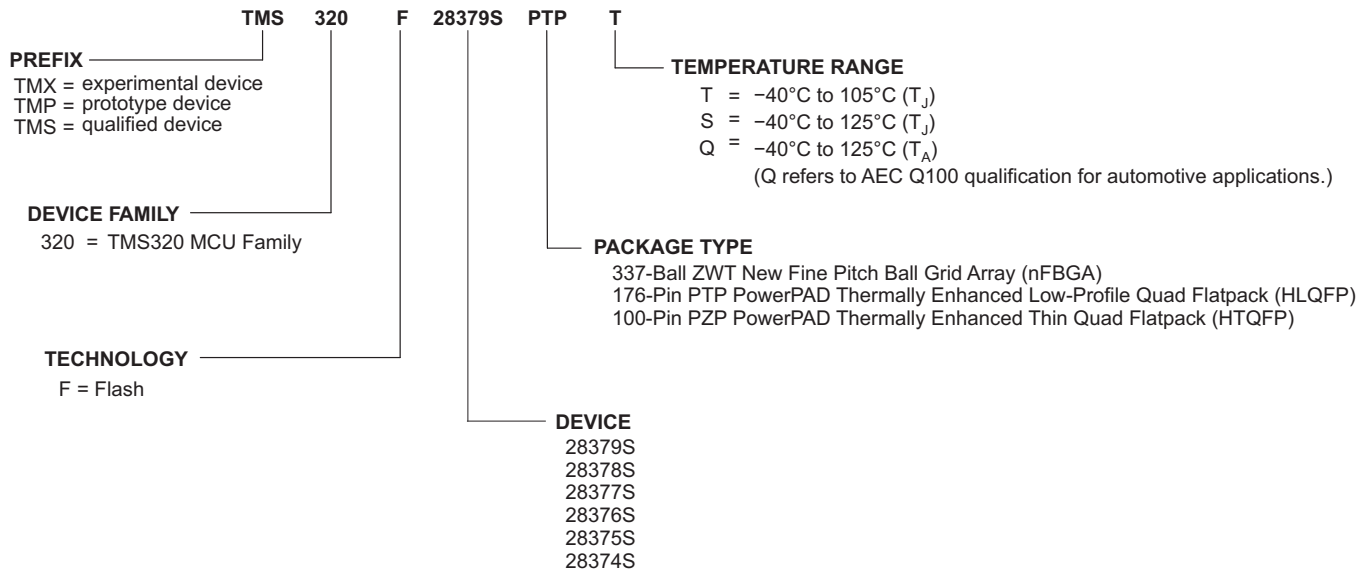


Figure 2. Example of Device Nomenclature

## 4 Usage Notes and Known Design Exceptions to Functional Specifications

### 4.1 Usage Notes

Usage notes highlight and describe particular situations where the device's behavior may not match presumed or documented behavior. This may include behaviors that affect device performance or functional correctness. These usage notes will be incorporated into future documentation updates for the device (such as the device-specific data sheet), and the behaviors they describe will not be altered in future silicon revisions.

Table 2 shows which silicon revision(s) are affected by each usage note.

**Table 2. List of Usage Notes**

TITLE	SILICON REVISION(S) AFFECTED	
	B	C
PIE: Spurious Nested Interrupt After Back-to-Back PIEACK Write and Manual CPU Interrupt Mask Clear	Yes	Yes
Caution While Using Nested Interrupts	Yes	Yes

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

**Revision(s) Affected:** B, C

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
```

#### 4.1.2 Caution While Using Nested Interrupts

**Revision(s) Affected:** B, C

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. Failing to do so may cause undefined behavior of CPU execution.

## 4.2 Known Design Exceptions to Functional Specifications

Table 3 shows which silicon revision(s) are affected by each advisory.

**Table 3. List of Advisories**

TITLE	SILICON REVISION(S) AFFECTED	
	B	C
Analog Trim of Some TMX Devices	Yes	
ADC: ADC Post-Processing Block Limit Compare	Yes	Yes
ADC: ADC Offset Trim in Different Modes	Yes	Yes
ADC: Random Conversion Errors	Yes	
ADC: ADC PPB Event Trigger (ADCxEVT) to ePWM Digital Compare Submodule	Yes	
ADC: 12-Bit Switch Resistance	Yes	
ADC: 12-Bit Input Capacitance When Switching Channel Groups	Yes	
XRS may Toggle During Power Up	Yes	
USB: USB DMA Event Triggers are not Supported	Yes	Yes
VREG: VREG Will be Enabled During Power Up Irrespective of VREGENZ	Yes	
Flash: A Single-Bit ECC Error May Cause Endless Calls to Single-Bit-Error ISR	Yes	Yes
Flash: Minimum Programming Word Size	Yes	Yes
ePIE: Spurious VCU Interrupt (ePIE 12.6) Can Occur When First Enabled	Yes	
eQEP: Position Counter Incorrectly Reset on Direction Change During Index	Yes	Yes
eQEP: eQEP Inputs in GPIO Asynchronous Mode	Yes	Yes
PLL: May Not Lock On the First Lock Attempt	Yes	Yes
SDFM: Data Filter Output Does Not Saturate at Maximum Value With Sinc3 and OSR = 256	Yes	Yes
SDFM: Spurious Data Acknowledge Event When Data Filter is Configured and Enabled for the First Time	Yes	Yes
SDFM: Spurious Data Acknowledge Event When Data Filter is Synchronized Using PWM FILRES Signal	Yes	Yes
SDFM: Comparator Filter Module may Generate Spurious Over-Value and Under-Value Conditions	Yes	Yes
SDFM: Dynamically Changing Threshold Settings (LLT, HLT), Filter Type, or COSR Settings Will Trigger Spurious Comparator Events	Yes	Yes
SDFM: Dynamically Changing Data Filter Settings (Such as Filter Type or DOSR) Will Trigger Spurious Data Acknowledge Events	Yes	Yes
FPU: FPU-to-CPU Register Move Operation Preceded by Any FPU 2p Operation	Yes	Yes
FPU: LUF, LVF Flags are Invalid for the EINVF32 and EISQRTF32 Instructions	Yes	Yes
Memory: Prefetching Beyond Valid Memory	Yes	Yes
INTOSC: V <sub>DDOSC</sub> Powered Without V <sub>DD</sub> Can Cause INTOSC Frequency Drift	Yes	Yes
Low-Power Modes: Power Down Flash or Maintain Minimum Device Activity	Yes	Yes
I2C: SDA and SCL Open-Drain Output Buffer Issue	Yes	Yes
ePWM: An ePWM Glitch can Occur if a Trip Remains Active at the End of the Blanking Window	Yes	Yes
ePWM: ePWM Dead-Band Delay Value Cannot be Set to 0 When Using Shadow Load Mode for RED/FED	Yes	Yes
CMPSS: COMPxLATCH May Not Clear Properly Under Certain Conditions	Yes	Yes
CMPSS: Ramp Generator May Not Start Under Certain Conditions	Yes	Yes
GPIO: Open-Drain Configuration May Drive a Short High Pulse	Yes	Yes
Boot ROM: Device Will Hang During Boot if X1 Clock Source is not Present	Yes	

**Table 4. Table of Contents for Advisories**

Title	Page
<b>Advisory</b> —Analog Trim of Some TMX Devices .....	6
<b>Advisory</b> —ADC: ADC Post-Processing Block Limit Compare .....	7
<b>Advisory</b> —ADC: ADC Offset Trim in Different Modes.....	7
<b>Advisory</b> —ADC: Random Conversion Errors .....	7
<b>Advisory</b> —ADC: ADC PPB Event Trigger (ADCxEVT) to ePWM Digital Compare Submodule.....	8
<b>Advisory</b> —ADC: 12-Bit Switch Resistance .....	8
<b>Advisory</b> —ADC: 12-Bit Input Capacitance When Switching Channel Groups .....	8
<b>Advisory</b> —XRS may Toggle During Power Up.....	9
<b>Advisory</b> —USB: USB DMA Event Triggers are not Supported .....	9
<b>Advisory</b> —VREG: VREG Will be Enabled During Power Up Irrespective of VREGENZ .....	9
<b>Advisory</b> —Flash: A Single-Bit ECC Error May Cause Endless Calls to Single-Bit-Error ISR .....	10
<b>Advisory</b> —Flash: Minimum Programming Word Size.....	10
<b>Advisory</b> —ePIE: Spurious VCU Interrupt (ePIE 12.6) Can Occur When First Enabled.....	11
<b>Advisory</b> —eQEP: Position Counter Incorrectly Reset on Direction Change During Index .....	12
<b>Advisory</b> —eQEP: eQEP Inputs in GPIO Asynchronous Mode .....	12
<b>Advisory</b> —PLL: May Not Lock on the First Lock Attempt .....	13
<b>Advisory</b> —SDFM: Data Filter Output Does Not Saturate at Maximum Value With Sinc3 and OSR = 256.....	15
<b>Advisory</b> —SDFM: Spurious Data Acknowledge Event When Data Filter is Configured and Enabled for the First Time	15
<b>Advisory</b> —SDFM: Spurious Data Acknowledge Event When Data Filter is Synchronized Using PWM FILRES Signal	15
<b>Advisory</b> —SDFM: Comparator Filter Module may Generate Spurious Over-Value and Under-Value Conditions .....	16
<b>Advisory</b> —SDFM: Dynamically Changing Threshold Settings (LLT, HLT), Filter Type, or COSR Settings Will Trigger Spurious Comparator Events.....	16
<b>Advisory</b> —SDFM: Dynamically Changing Data Filter Settings (Such as Filter Type or DOSR) Will Trigger Spurious Data Acknowledge Events.....	16
<b>Advisory</b> —FPU: FPU-to-CPU Register Move Operation Preceded by Any FPU 2p Operation .....	17
<b>Advisory</b> —FPU: LUF, LVF Flags are Invalid for the EINVF32 and EISQRTF32 Instructions .....	20
<b>Advisory</b> —Memory: Prefetching Beyond Valid Memory .....	23
<b>Advisory</b> —INTOSC: $V_{DDOSC}$ Powered Without $V_{DD}$ Can Cause INTOSC Frequency Drift .....	23
<b>Advisory</b> —Low-Power Modes: Power Down Flash or Maintain Minimum Device Activity .....	24
<b>Advisory</b> —I2C: SDA and SCL Open-Drain Output Buffer Issue .....	25
<b>Advisory</b> — ePWM: An ePWM Glitch can Occur if a Trip Remains Active at the End of the Blanking Window.....	27
<b>Advisory</b> —ePWM: ePWM Dead-Band Delay Value Cannot be Set to 0 When Using Shadow Load Mode for RED/FED.....	27
<b>Advisory</b> —CMPSS: COMPxLATCH May Not Clear Properly Under Certain Conditions .....	28
<b>Advisory</b> —CMPSS: Ramp Generator May Not Start Under Certain Conditions .....	28
<b>Advisory</b> —GPIO: Open-Drain Configuration May Drive a Short High Pulse .....	29
<b>Advisory</b> —Boot ROM: Device Will Hang During Boot if X1 Clock Source is not Present .....	30

**Advisory**                      **Analog Trim of Some TMX Devices**
**Revision(s) Affected**      B

**Details**                      Some TMX samples may not have analog trims programmed. This could degrade the performance of the ADC, buffered DAC, and internal oscillators. A value of all zeros in these trim registers due to lack of trim will have the following impact.

TRIM	REGISTER	IMPACT OF UNTRIMMED REGISTER
<b>ADC reference</b>	AnalogSubsysRegs.ANAREFTRIMA AnalogSubsysRegs.ANAREFTRIMB AnalogSubsysRegs.ANAREFTRIMC AnalogSubsysRegs.ANAREFTRIMD	Degraded performance of the ADC for all specifications.
<b>ADC linearity</b>	AdcaRegs.ADCINLTRIM1-6 AdcbRegs.ADCINLTRIM1-6 AdccRegs.ADCINLTRIM1-6 AdcdRegs.ADCINLTRIM1-6	Degraded INL and DNL specifications of the ADC in 16-bit mode. No workaround available.
<b>ADC offset</b>	AdcaRegs.ADCOFFTRIM AdcbRegs.ADCOFFTRIM AdccRegs.ADCOFFTRIM AdcdRegs.ADCOFFTRIM	Degraded performance of the ADC offset error specification.
<b>Internal oscillator</b>	AnalogSubsysRegs.INTOSC1TRIM AnalogSubsysRegs.INTOSC2TRIM	Degraded frequency accuracy and temperature drift of the internal oscillators.
<b>Buffered DAC offset</b>	DacaRegs.DACTRIM DacbRegs.DACTRIM DaccRegs.DACTRIM	Degraded offset error specification of the buffered DAC. No workaround available.

**Workaround(s)**              The following workarounds can be used for improved performance, though it still may not meet data sheet specifications.

To determine if a device is TMX in software, check the status of the PARTIDL[QUAL]. If this field is 0, the device is TMX. PARTIDL[QUAL] can be read via the function call SysCtl\_getDeviceParametric(SYSCTL\_DEVICE\_QUAL).

If the **ADC reference trim** registers contain all zeros, write the static reference trim value of 0x7BDD to the reference trim register for all ADCs.

Missing **ADC offset trim** can be generated by following the instructions in the “ADC Zero Offset Calibration” section of the [TMS320F2837xS Delfino Microcontrollers Technical Reference Manual](#).

If the **internal oscillator trim** contains all zeros, the user can adjust the lowest 10 bits of the oscillator trim register between 1 (minimum) and 1023 (maximum) while observing the system clock on the XCLOCKOUT pin.

**Advisory** *ADC: ADC Post-Processing Block Limit Compare*


---

**Revision(s) Affected** B, C

**Details** When using a non-zero offset reference in the ADC post-processing block (PPB), the resultant ADCPPBxRESULT can be signed. TRIPHI or TRIPLO limit compares do not function correctly with this result if it is signed.

**Workaround(s)** When using the TRIPHI or TRIPLO limit compares, leave the offset reference as zero. The offset reference (and zero compare) can be used as long as the limit compares are disabled.

If the limit compares, the offset reference, and the zero-crossing compare are to be used at the same time, then two PPBs can be used. Both PPBs should be configured to use the same SOC. One PPB can implement the TRIPHI and/or TRIPLO limit compares while the other can implement offset reference subtraction and zero-crossing detection.

**Advisory** *ADC: ADC Offset Trim in Different Modes*


---

**Revision(s) Affected** B, C

**Details** A different offset trim is required when switching between 12-bit and 16-bit resolution and when switching between single-ended and differential signaling mode.

**Workaround(s)** Whenever setting the resolution or signal mode of the ADC, use the "AdcSetMode" function in C2000Ware. This will ensure the correct trims are loaded into the offset trim register. Note that on start-up, trims will be loaded for 12-bit, single-ended operation.

**Advisory** *ADC: Random Conversion Errors*


---

**Revision(s) Affected** B

**Details** The ADC may have errors at a rate as high as 1 in  $10^{6.5}$  ADC conversions in 12-bit mode and as high as 1 in  $10^{8.75}$  conversions in 16-bit mode. When a conversion error occurs, it will be a significant random jump in the digital output of the ADC without a corresponding change in the ADC input voltage, otherwise known as a "sparkle code". The magnitude of this jump will typically be in the range of 20 LSBs to 200 LSBs; however, larger or smaller jumps may occur.

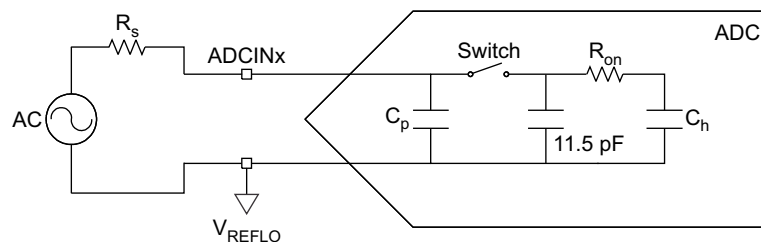
**Workaround(s)** For the revisions affected, the error rate will be lower than 1 error in  $10^{14.5}$  ADC conversions for both 12-bit mode and 16-bit mode when all of the following configurations are used:

- The S+H duration is at least 320 ns
- ADCCLK is 40 MHz or less
- ADCCLK prescale is a whole number: /1.0, /2.0, /3.0, /4.0, /5.0, /6.0, /7.0, or /8.0
- The value of 0x7000 is written to memory locations 0x0000 743F, 0x0000 74BF, 0x0000 753F, and 0x0000 75BF (writing this value is only valid when the ADCCLK prescale is a whole number).

<b>Advisory</b>	<b><i>ADC: ADC PPB Event Trigger (ADCxEVT) to ePWM Digital Compare Submodule</i></b>
<b>Revision(s) Affected</b>	B
<b>Details</b>	The ADCxEVT trigger to the ePWM digital compare submodule may not be detected by the ePWM.
<b>Workaround(s)</b>	The ADCxEVT can generate an ADCx_EVT interrupt to the PIE. The ISR can be used to perform the desired task in software.

<b>Advisory</b>	<b><i>ADC: 12-Bit Switch Resistance</i></b>
<b>Revision(s) Affected</b>	B
<b>Details</b>	The ADC input model should be used to select the sample-and-hold (S+H) duration for each ADC input. For the revisions affected, the 12-bit input model under-estimates the value of the sampling switch resistance ( $R_{on}$ ). A $R_{on}$ value of 2 k $\Omega$ should be used to select the S+H duration for these revisions.
<b>Workaround(s)</b>	For the revisions affected, the S+H duration should be chosen to account for the additional switch resistance.

<b>Advisory</b>	<b><i>ADC: 12-Bit Input Capacitance When Switching Channel Groups</i></b>
<b>Revision(s) Affected</b>	B
<b>Details</b>	The ADC input model should be used to select the sample-and-hold (S+H) duration for each ADC input. For the revisions affected, if the currently converting channel is an even-numbered channel and the previously converted channel was an odd-numbered channel (or vice versa), then the 12-bit input model will not accurately predict ADC input performance. Under these conditions, an additional capacitance should be added to the model. This capacitance has a value of 11.5 pF and should be placed between the S+H switch and $R_{on}$ as shown in <a href="#">Figure 3</a> .



**Figure 3. Single-Ended Input Model**

<b>Workaround(s)</b>	For the revisions affected, when subsequent conversions switch between channel groups, the S+H duration should be chosen to account for the additional capacitance.
----------------------	---



**Advisory**  ***$\overline{XRS}$  may Toggle During Power Up***


---

**Revision(s) Affected** B

**Details** During device power up, the  $\overline{XRS}$  pin may toggle high prematurely. After the  $V_{DDIO}$  and  $V_{DD}$  supplies reach the recommended operation conditions, the  $\overline{XRS}$  pin behavior will be per the pin description. This is only an issue with the external state of the  $\overline{XRS}$  pin. Internally, the device will be held in reset by the POR logic until the supplies are within an acceptable range and  $\overline{XRS}$  is high.

**Workaround(s)** Disregard  $\overline{XRS}$  activity on the board prior to supplies reaching recommended operating conditions.

**Advisory** ***USB: USB DMA Event Triggers are not Supported***


---

**Revision(s) Affected** B, C

**Details** The USB module generates inadvertent extra DMA requests, causing the FIFO to overflow (on IN endpoints) or underflow (on OUT endpoints). This causes invalid IN DATA packets (larger than the maximum packet size) and duplicate receive data.

**Workaround(s)** None

**Advisory** ***VREG: VREG Will be Enabled During Power Up Irrespective of VREGENZ***


---

**Revision(s) Affected** B

**Details** During power up of the 3.3-V  $V_{DDIO}$ , the internal Voltage Regulator (VREG) will be active until the 1.2-V  $V_{DD}$  supply reaches approximately 0.7 V. After this time, the VREGENZ pin tied to  $V_{DDIO}$  will disable the internal VREG. This will not impact device operation.

**Workaround(s)** None

---

<b>Advisory</b>	<b><i>Flash: A Single-Bit ECC Error May Cause Endless Calls to Single-Bit-Error ISR</i></b>
<b>Revision(s) Affected</b>	B, C
<b>Details</b>	<p>When a single-bit ECC error is detected, the CPU executes the single-bit-error interrupt service routine (ISR). When the ISR returns, the same instruction that caused the first error is fetched again. If the ECC error threshold (ERR_THRESHOLD.THRESHOLD) is 0, then the same error is detected and another ISR is executed. This continues in an endless loop. This sequence of events only occurs if the error is caused by a program fetch operation, not a data read.</p>
<b>Workaround(s)</b>	<p>Set the error threshold bit-field (ERR_THRESHOLD.THRESHOLD) to a value greater than or equal to 1. Note that the default value of the threshold bit-field is 0.</p>
<b>Advisory</b>	<b><i>Flash: Minimum Programming Word Size</i></b>
<b>Revision(s) Affected</b>	B, C
<b>Details</b>	<p>The Main Array flash programming must be aligned to 64-bit address boundaries and each 64-bit word may only be programmed once per write/erase cycle.</p> <p>Applications using Fapi_issueProgrammingCommand() in Fapi_AutoEccGeneration or Fapi_DataAndEcc modes are implicitly performing 64-bit programming since ECC is programmed for each 64 bits. Applications using Fapi_DataOnly mode with fewer than 64 bits may be impacted by this advisory.</p> <p>The DCSM OTP programming must be aligned to 128-bit address boundaries and each 128-bit word may only be programmed once. The exceptions are:</p> <ol style="list-style-type: none"><li>1. The DCSM Zx-LINKPOINTER1 and Zx-LINKPOINTER2 values in the DCSM OTP should be programmed together, and may be programmed 1 bit at a time as required by the DCSM operation.</li><li>2. The DCSM Zx-LINKPOINTER3 value in the DCSM OTP may be programmed 1 bit at a time as required by the DCSM operation.</li></ol>
<b>Workaround(s)</b>	<p>All applications should follow the restrictions outlined in this advisory. Contact TI for devices already in production which violate this advisory.</p>

**Advisory** ***ePIE: Spurious VCU Interrupt (ePIE 12.6) Can Occur When First Enabled***

---

**Revision(s) Affected** B**Details** The VCU-II can power up in a state which incorrectly sets the VCU VSTATUS[DIVE] error bit and, subsequently PIEIFR12[INTx6], when the CPU is released from reset. When the VCU interrupt enable PIEIER12[INTx6] is enabled for the first time by the application, a spurious interrupt can occur due to the erroneous pending interrupt.**Workaround(s)** Before enabling VCU interrupt 12.6, execute the following instructions to avoid the spurious interrupt.

```
// Clear VCU divide by zero status  
asm(" VCLR DIVE");  
// Clear PIE interrupt for VCU  
PieCtrlRegs.PIEIFR12.bit.INTx6 = 0;
```

Beginning with revision C silicon, the Boot ROM will perform the above workaround before branching to the application.

<b>Advisory</b>	<b><i>eQEP: Position Counter Incorrectly Reset on Direction Change During Index</i></b>
<b>Revision(s) Affected</b>	B, C
<b>Details</b>	<p>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 <math>\pm 4</math> counts from the expected value of the position counter and lead to unexpected subsequent setting of the error flags.</p> <p>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.</p> <p>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.</p> <p>The next index event without a simultaneous direction change will reset the counter properly and work as expected.</p>
<b>Workaround(s)</b>	<p>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.</p> <p>Other options for performing position counter reset, if appropriate for the application [such as Index Event Initialization (IEI)], do not have this issue.</p>
<b>Advisory</b>	<b><i>eQEP: eQEP Inputs in GPIO Asynchronous Mode</i></b>
<b>Revision(s) Affected</b>	B, C
<b>Details</b>	<p>If any of the eQEP input pins are configured for GPIO asynchronous input mode via the GPxQSELn registers, the eQEP module may not operate properly because the eQEP peripheral assumes the presence of external synchronization to SYSCLKOUT on inputs to the module. For example, QPOSCNT may not reset or latch properly, and pulses on the input pins may be missed.</p> <p>For proper operation of the eQEP module, input GPIO pins should be configured via the GPxQSELn registers for synchronous input mode (with or without qualification), which is the default state of the GPxQSEL registers at reset. All existing eQEP peripheral examples supplied by TI also configure the GPIO inputs for synchronous input mode.</p> <p>The asynchronous mode should not be used for eQEP module input pins.</p>
<b>Workaround(s)</b>	Configure GPIO inputs configured as eQEP pins for non-asynchronous mode (any GPxQSELn register option except "11b = Asynchronous").

**Advisory**
***PLL: May Not Lock on the First Lock Attempt***
**Revision(s) Affected**

B, C

**Details**

The PLL may not start properly at device power up or wake up from Hibernate. The PLLSTS[LOCKS] bit is set, but the PLL does not produce a clock.

Once the PLL has properly started, the PLL can be disabled and reenabled with no issues and will stay locked. However, the PLL lock problem could reoccur on a subsequent power-up or Hibernate cycle.

If the SYSPLL has not properly started and is selected as the CPU clock source, the CPU will stop executing instructions. The occurrence rate of this transient issue is low and after an initial occurrence, this issue may not be subsequently observed in the system again. Implementation of the workaround reduces the rate of occurrence.

This advisory applies to both PLLs, with a different workaround for each.

**Workaround(s)**
***SYSPLL Workaround:***

Repeated lock attempts will reduce the likelihood of seeing the condition on the final attempt. TI recommends a minimum of five lock sequences in succession when the PLL is configured the first time after a power up. A lock sequence means disabling the PLL, starting the PLL locking, and waiting for the LOCKS bit to set. After the final sequence, the clock source is switched to use the PLL output as normal.

The Watchdog timer can be used to detect that the condition has occurred because it is not clocked by the PLL output. The Watchdog should be enabled before selecting the PLL as the clock source and configured to reset the device. If the PLL is not producing a clock, the Watchdog will reset the device and the user initialization software will therefore repeat the PLL initialization.

Many applications do not have a different initialization sequence for a Watchdog-initiated reset; for these applications, no further action is required. For applications that do use a different device initialization sequence when a Watchdog reset is detected, a flag can be used to identify the Watchdog reset as a PLL cause. The SYSDBGCTL[BIT\_0] bit (which is bit 0 at 0x0005D12C) can be set active during the PLL lock sequence and used to distinguish a Watchdog PLL retry attempt versus a different Watchdog reset source.

The SYSPLLSTS[SLIPS] should also be checked immediately after setting the PLL as the SYSCLK source with SYSPLLCTL1[PLLCLKEN]. If SLIPS indicates a PLL slip, then the PLL should be disabled and locked again until there are no slips detected.

See the C2000Ware InitSysPll() function for an example implementation of this workaround, as well as the DriverLib function SysCtl\_setClock().

The workaround can also be applied at the System level by a supervisor resetting the device if it is not responding.

**AUXPLL Workaround:**

CPU Timer 2 can be used to detect that the AUXPLL is active before it is used as a clock source for USB. If the AUXPLL is not active, repeat the lock attempt until successful.

The AUXPLLSTS[SLIPS] should also be checked immediately after setting the PLL as the AUXPLLCLK source with AUXPLLCTL1[PLLCLKEN]. If SLIPS indicates a PLL slip, then the PLL should be disabled and locked again until there are no slips detected.

See the C2000Ware InitAuxPll() function for an example implementation of this workaround, as well as the DriverLib function SysCtl\_setAuxClock().

---

**NOTE:** The USB Boot Mode does not implement the previous workarounds. Applications using USB Boot will need to implement any retry attempts at the system level.

---

<b>Advisory</b>	<b><i>SDFM: Data Filter Output Does Not Saturate at Maximum Value With Sinc3 and OSR = 256</i></b>
<b>Revision(s) Affected</b>	B, C
<b>Details</b>	If the differential input of the Sigma-Delta Filter Module (SDFM) is greater than or equal to +FSR (full-scale differential voltage input range), then the output of the SDFM clips with a stream of ones. When this stream of ones is fed to a data filter that is configured as a sinc3 filter with an OSR = 256, the output of the filter does not saturate at the maximum value (16777215 in 32-bit mode or 32767 in 16-bit mode); but, instead roll over to the minimum value (–16777216 in 32-bit mode or –32768 in 16-bit mode).
<b>Workaround(s)</b>	Maintain the differential input of the SDFM in the specified linear input range as specified in the modulator data sheet.
<b>Advisory</b>	<b><i>SDFM: Spurious Data Acknowledge Event When Data Filter is Configured and Enabled for the First Time</i></b>
<b>Revision(s) Affected</b>	B, C
<b>Details</b>	When the SDFM data filter is configured and enabled for the first time, it is possible to get one spurious data acknowledge event (AFx) before the data filter settles to give correct digital data. Subsequent data acknowledge events (AFx)/DMA events occur correctly as per data filter configuration.
<b>Workaround(s)</b>	Do the following: <ol style="list-style-type: none"> <li>1. Configure and enable the SDFM data filter.</li> <li>2. Delay for at least latency of data filter + 5 SD-Cx clock cycles.</li> <li>3. Enable SDFM data acknowledge interrupts/DMA events.</li> </ol>
<b>Advisory</b>	<b><i>SDFM: Spurious Data Acknowledge Event When Data Filter is Synchronized Using PWM FILRES Signal</i></b>
<b>Revision(s) Affected</b>	B, C
<b>Details</b>	When the SDFM data filters are synchronized using the PWM FILRES signal, it is possible to get a spurious data acknowledge event (AFx) before the data filter settles to give correct digital data. Subsequent data acknowledge events (AFx) occur correctly as per data filter configuration before the next PWM FILRES signal.
<b>Workaround(s)</b>	Do the following: <ol style="list-style-type: none"> <li>1. Choose any PWMx to work in the same time base as the PWM that generates the FILRES pulse.</li> <li>2. PWMx should also interrupt the CPU/CLA at least 1.2 μs after the PWM FILRES pulse gets applied in order to clear the SDIFLG register that may be set because of the spurious data acknowledge event.</li> <li>3. SDFM_CPUISR or SDFM_CLATask: <ol style="list-style-type: none"> <li>i. Collect the required number of samples, N, after the FILRES pulse.</li> <li>ii. If the number of samples is less than or equal to N, clear the SDIFLG register; otherwise, do not clear the SDIFLG register to prevent further SDFM interrupts.</li> </ol> </li> </ol>

<b>Advisory</b>	<b><i>SDFM: Comparator Filter Module may Generate Spurious Over-Value and Under-Value Conditions</i></b>
<b>Revision(s) Affected</b>	B, C
<b>Details</b>	When interrupts are enabled in the SDFM comparator module, it may trigger spurious over-value (SDIFLG.IEHx, x = 1 to 4) or under-value (SDIFLG.IELx, x = 1 to 4) conditions. These are depicted as IELx and IEHx interrupt outputs in the “Block Diagram of One Filter Module” figure in the <a href="#">TMS320F2837xS Delfino Microcontrollers Technical Reference Manual</a> .
<b>Workaround(s)</b>	These erroneous interrupts can be eliminated by implementing the following workaround: <ul style="list-style-type: none"> <li>• Comparator OSR (COSR) value should be greater than or equal to 5.</li> <li>• After changing COSR, wait for at least latency of comparator filter and 5 SD-Cx cycles before enabling comparator interrupts SDCPARMx.IEH and SDCPARMx.IEL.</li> </ul>
<b>Advisory</b>	<b><i>SDFM: Dynamically Changing Threshold Settings (LLT, HLT), Filter Type, or COSR Settings Will Trigger Spurious Comparator Events</i></b>
<b>Revision(s) Affected</b>	B, C
<b>Details</b>	When SDFM comparator settings—such as filter type, lower/upper threshold, or COSR settings—are changed while low-level/high-level events are enabled, a spurious comparator lower threshold (or) higher threshold event will be triggered.
<b>Workaround(s)</b>	When comparator settings need to be changed dynamically, follow the procedure below: <ol style="list-style-type: none"> <li>1. Disable the SDFM comparator interrupts.</li> <li>2. Change comparator settings such as lower/upper threshold, filter type, or COSR.</li> <li>3. Comparator OSR (COSR) value should be greater than or equal to 5.</li> <li>4. Delay for at least a latency of comparator filter + 5 SD-Cx clock cycles.</li> <li>5. Enable the SDFM comparator interrupts.</li> </ol>
<b>Advisory</b>	<b><i>SDFM: Dynamically Changing Data Filter Settings (Such as Filter Type or DOSR) Will Trigger Spurious Data Acknowledge Events</i></b>
<b>Revision(s) Affected</b>	B, C
<b>Details</b>	When SDFM data filter settings—such as filter type or DOSR settings—are changed while the data filter and its data acknowledge events are enabled, spurious data acknowledge events will be triggered.
<b>Workaround(s)</b>	When data filter settings need to be changed dynamically, follow the procedure below: <ol style="list-style-type: none"> <li>1. Disable the SDFM data filter.</li> <li>2. Change data filter settings such as filter type or DOSR.</li> <li>3. Delay for at least a latency of data filter + 5 SD-Cx clock cycles.</li> <li>4. Enable the SDFM data filter.</li> </ol>



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

**Revision(s) Affected** B, C

**Details**

This advisory applies when a multi-cycle (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 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	I3	I2		
								I4	I3		
									I4	I3	

**Figure 4. 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	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	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 6. Pipeline Diagram With Workaround in Place**

**Advisory** *FPU: LUF, LVF Flags are Invalid for the EINVF32 and EISQRTF32 Instructions*
**Revision(s) Affected** B, C

**Details** This advisory applies to the EINVF32 and EISQRTF32 instructions. The expected results for these instructions are correct; however, the underflow (LUF) and overflow (LVF) flags are not. These flags are invalid and should not be used.

The LUF and LVF flags are not accessible using C code, so the overall impact of this advisory is expected to be small. If the user chooses to use these flags (for example, when coding a time-critical algorithm) in assembly as part of a mixed C/ASM project, the user will need to disable interrupts around the assembly code using the flags, and also preserve the flags through any use of EINVF32 or EISQRTF32 instructions.

**Workaround(s)** There is no workaround for using these flags in C code, and they should be considered invalid for the reasons presented under **NOTES ON COMPILER AND TOOLS USAGE**.

The workaround shown here provides a way to preserve the LVF, LUF flags across the use of EISQRTF32 and EINVF32 in assembly-only code.

Do not rely on the LUF and LVF flags to catch underflow/overflow conditions resulting from the EINVF32 and EISQRTF32 instructions. Instead, check the operands for the following conditions (in code) before using each instruction:

EINVF32	Divide by 0
EISQRTF32	Divide by 0, Divide by a negative input

Disregard the contents of the LUF and LVF flags by saving the flags to the stack before calling the instruction, and subsequently restoring the values of the flags once the instruction completes.

```

MOV32          *SP++,STF      ; Save off current status flags
EISQRTF32/EINVF32      ; Execute operation
NOP            ; Wait for operations to complete
MOV32          STF, *--SP    ; Restore previous status flags

```

If the PIE interrupts are tied to the LUF and LVF flags, disable the interrupts (at the PIE) before using either the EINVF32 or EISQRTF32 instruction. Check to see if the LUF and LVF flags are set; if they are, a variable can be set to indicate that a false LUF/LVF condition is detected. Clear the flags in the STF (FPU status flag) before re-enabling the interrupts.

Once the interrupts are reenabled at the PIE, the interrupt may occur (if the LUF/LVF interrupt lines were asserted by either of the two instructions) and execution branches to the Interrupt Service Routine (ISR). Check the flag to determine if a false condition has occurred; if it has, disregard the interrupt.

Do not clear the PIE IFR bits (that latch the LUF and LVF flags) directly because an interrupt event on the same PIE group (PIE group 12) may inadvertently be missed.

Here is an example:

```

_flag_LVFLUF_set    .usect ".ebss",2,1,1
...
MOV32    *SP++,STF                ; Save off current status flags
; Load the PieCtrlRegs page to the DP
MOVW     DP, #_PieCtrlRegs.PIEIER12.all
; Zero out PIEIER12.7/8, i.e. disable LUF/LVF interrupts
AND      @_PieCtrlRegs.PIEIER12.all, #0xFF3F
EISQRTF32/EINVF32                ; Execute operation
MOVL     XAR3, #_flag_LVFLUF_set  ; Wait for operation to complete
MOV32    *+XAR3[0], STF           ; save STF to _flag_LVFLUF_set
AND      *+XAR3[0], #0x3          ; mask everything but LUF/LVF
; Clear Latched overflow, underflow flag
SETFLG   LUF=0, LVF=0
; Re-enable PIEIER12.7/8, i.e. re-enable the LUF/LVF interrupts
OR       @_PieCtrlRegs.PIEIER12.all, #0x00C0
MOV32    STF,*--SP                ; Restore previous status flags

```

In the ISR,

```

__interrupt void fpu32_luf_lvf_isr (void)
{
// Check the flag for whether the LUF, LVF flags set by
// either EISRTF32 or EINVF32
if((flag_LVFLUF_set & 0x3U) != 0U)
{
//Reset flag
flag_LVFLUF_set = 0U;
// Do Nothing
}
else
{
//If flag_LVFLUF_set was not set then this interrupt
// is the legitimate result of an overflow/underflow
// from an FPU operation (not EISQRTF32/EINVF32)
...
// Handle Overflow/Underflow condition
...
...
...
}
// Ack the interrupt and exit
}

```

**NOTES: NOTES ON COMPILER AND TOOLS USAGE**

The compiler does not use LVF/LUF as condition codes for conditional instructions and neither does the Run Time Support (RTS) Library test LVF/LUF in any way.

The compiler may generate code that modifies LVF/LUF, meaning the value of the STF register (that contain these flags) is undefined at function boundaries. Thus, although the sqrt routine in the library may cause LVF/LUF to be set, there is no assurance in the CGT that the user can read these bits after sqrt returns.

Although the compiler does provide the `__eisqrtf` and `__einvf32` intrinsics, it does not provide an intrinsic to read the LVF/LUF bits or the STF register. Thus, the user has no way to access these bits from C code.

The use of inline assembly code to read the STF register is unreliable and is discouraged. The workaround presented in the Workaround(s) section is applicable to assembly code that uses the EISQRTF32 and EINVF32 instructions and does not call any compiler-generated code. For C code, the user must consider these flags to be unreliable, and therefore, neither poll these flags in code nor trigger interrupts off of them.

---

**Advisory** **Memory: Prefetching Beyond Valid Memory**
**Revision(s) Affected** B, C

**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** 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.

**Table 5. Memories Impacted by Advisory**

MEMORY TYPE	ADDRESSES IMPACTED	F28378S F28377S F28375S	F28376S F28374S
M1	0x0000 07F8–0x0000 07FF	Yes	Yes
GS11	0x0001 7FF8–0x0001 7FFF	No	Yes
GS15	0x0001 BFF8–0x0001 BFFF	Yes	N/A
Flash	0x000B FFF8–0x000B FFFF	No	Yes

**Advisory** **INTOSC:  $V_{DDOSC}$  Powered Without  $V_{DD}$  Can Cause INTOSC Frequency Drift**
**Revision(s) Affected** B, C

**Details** The "G" revision of the [TMS320F2837xS Delfino™ Microcontrollers Data Manual](#) (SPRS881G) has updated power sequencing requirements. Revision "F" and earlier revisions of the data manual did not require  $V_{DDOSC}$  and  $V_{DD}$  to be powered on and powered off at the same time.

If  $V_{DDOSC}$  is powered on while  $V_{DD}$  is not powered, there will be an accumulating and persistent downward frequency drift for INTOSC1 and INTOSC2. The rate of drift accumulated will be greater when  $V_{DDOSC}$  is powered without  $V_{DD}$  at high temperatures.

As a result of this drift, the INTOSC1 and INTOSC2 internal oscillator frequencies could fall below the minimum values specified in the data manual. This would impact applications using INTOSC2 as the clock source for the PLL, with the system operating at a lower frequency than expected.

**Workaround(s)**

1. Keep  $V_{DDOSC}$  and  $V_{DD}$  powered together.
2. Use the external X1 and X2 crystal oscillators as the PLL clock source. The crystal oscillator does not have any drift related to  $V_{DDOSC}$  and  $V_{DD}$  supply sequencing.

<b>Advisory</b>	<b><i>Low-Power Modes: Power Down Flash or Maintain Minimum Device Activity</i></b>
<b>Revision(s) Affected</b>	B, C
<b>Details</b>	<p>The device has an intentional current path from <math>V_{DD3VFL}</math> (flash supply) to <math>V_{DD}</math>. Since the HALT, STANDBY, IDLE, or other low-activity device conditions can have low current demand on <math>V_{DD}</math>, this <math>V_{DD3VFL}</math> current can cause <math>V_{DD}</math> to rise above the recommended operating voltage.</p> <p>There will be zero current load to the external system <math>V_{DD}</math> regulator while in this condition. This is not an issue for most regulators; however, some system voltage regulators require a minimum load for proper operation.</p>
<b>Workaround(s)</b>	<p><b>Workaround 1:</b> Power down the flash before entering HALT, STANDBY, IDLE, or other low-activity device conditions. This will disable the internal current path. This workaround must be executed from RAM.</p>

```

EALLOW;
// seize the pump semaphore
while (IpcRegs.PUMPREQUEST.bit.SEM != 0x2)
{
    IpcRegs.PUMPREQUEST.all = IPC_PUMP_KEY | 0x2;
}

// power down bank
Flash0CtrlRegs.FBFALLBACK.bit.BNKPWR0 = 0;
asm(" RPT #8 || NOP");
// power down pump
Flash0CtrlRegs.FPAC1.bit.PMPPWR = 0;
asm(" RPT #8 || NOP");

// seize the pump semaphore
IpcRegs.PUMPREQUEST.all = IPC_PUMP_KEY | 0x1;

Flash1CtrlRegs.FBFALLBACK.bit.BNKPWR0 = 0;
asm(" RPT #8 || NOP");
Flash1CtrlRegs.FPAC1.bit.PMPPWR = 0;
asm(" RPT #8 || NOP");

// release pump semaphore
IpcRegs.PUMPREQUEST.all = IPC_PUMP_KEY | 0x0;
EDIS;
// enter low power mode
asm(" IDLE");

```

**Workaround 2:** Keep SYSCLK at a minimum of 100 MHz during STANDBY or IDLE. This activity will be sufficient to consume the internal current.

**Workaround 3:** An external 82-Ω resistor can be added to the board between  $V_{DD}$  and  $V_{SS}$ .



**Advisory**
***I2C: SDA and SCL Open-Drain Output Buffer Issue***
**Revision(s) Affected**

B, C

**Details**

The SDA and SCL outputs are implemented with push-pull 3-state output buffers rather than open-drain output buffers as required by I2C. While it is possible for the push-pull 3-state output buffers to behave as open-drain outputs, an internal timing skew issue causes the outputs to drive a logic-high for a duration of 0–5 ns before the outputs are disabled. The unexpected high-level pulse will only occur when the SCL or SDA outputs transition from a driven low state to a high-impedance state and there is sufficient internal timing skew on the respective I2C output.

This short high-level pulse injects energy in the I2C signals traces, which causes the I2C signals to sustain a period of ringing as a result of multiple transmission line reflections. This ringing should not cause an issue on the SDA signal because it only occurs at times when SDA is expected to be changing logic levels and the ringing will have time to damp before data is latched by the receiving device. The ringing may have enough amplitude to cross the SCL input buffer switching threshold several times during the first few nanoseconds of this ringing period, which may cause clock glitches. This ringing should not cause a problem if the amplitude is damped within the first 50 ns because I2C devices are required to filter their SCL inputs to remove clock glitches. Therefore, it is important to design the PCB signal traces to limit the duration of the ringing to less than 50 ns. One possible solution is to insert series termination resistors near the SCL and SDA terminals to attenuate transmission line reflections.

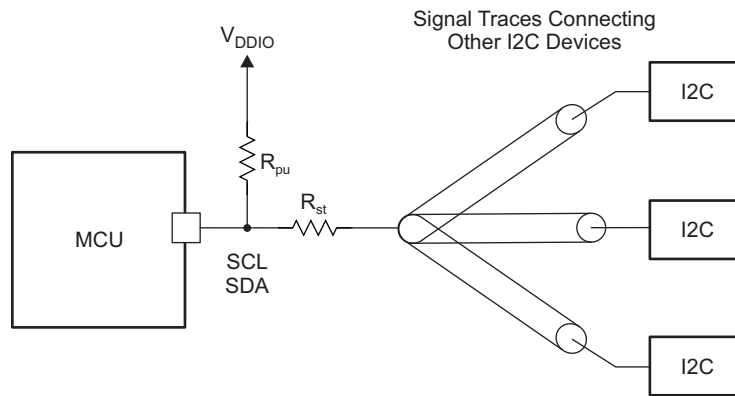
This issue may also cause the SDA output to be in contention with the slave SDA output for the duration of the unexpected high-level pulse when the slave begins its ACK cycle. This occurs because the slave may already be driving SDA low before the unexpected high-level pulse occurs. The glitch that occurs on SDA as a result of this short period of contention does not cause any I2C protocol issue but the peak current applies unwanted stress to both I2C devices and potentially increases power supply noise. Therefore, a series termination resistor located near the respective SDA terminal is required to limit the current during the short period of contention.

A similar contention problem can occur on SCL when connected to I2C slave devices that support clock stretching. This occurs because the slave is driving SCL low before the unexpected high-level pulse occurs. The glitch that occurs on SCL as a result of this short period of contention does not cause any I2C protocol issue because I2C devices are required to apply a glitch filter to their SCL inputs. However, the peak current applies unwanted stress to both I2C devices and potentially increases power supply noise. Therefore, a series termination resistor located near the respective SCL terminal is required to limit the current during the short period of contention.

If another master is connected, the unexpected high-level pulses on the SCL and SDA outputs can cause contention during clock synchronization and arbitration. The series termination resistors described above will also limit the contention current in this use case without creating any I2C protocol issue.

**Workaround(s)**

Insert series termination resistors on the SCL and SDA signals and locate them near the SCL and SDA terminals. The SCL and SDA pullup resistors should also be located near the SCL and SDA terminals. The placement of the series termination resistor and pullup resistor should be connected as shown in [Figure 7](#).



**Figure 7. Placement of Series Termination Resistor and Pullup Resistor**

Table 6 provides series termination and pullup resistor value recommendations. The I2C signal level and respective  $V_{DDIO}$  power supply voltage is shown in the first column. Two resistor value combination options are provided for each voltage. One option supports a maximum high-level input current of 200  $\mu\text{A}$  to all attached I2C devices, while the other option supports a maximum high-level input current of 100  $\mu\text{A}$  to all attached I2C devices.

**Table 6. Recommended Values for Series Termination Resistor and Pullup Resistor**

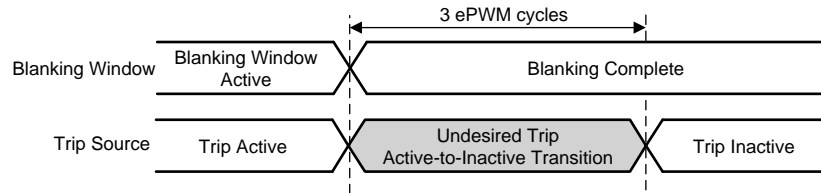
I2C SIGNAL LEVEL AND RESPECTIVE $V_{DDIO}$ POWER SUPPLY (V)	SERIES TERMINATION RESISTOR ( $\Omega$ )	PULLUP RESISTOR ( $\Omega$ )	NOTES
3.3	60	3300	Maximum high-level input current up to 200 $\mu\text{A}$
3.3	75	6600	Maximum high-level input current up to 100 $\mu\text{A}$

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

**Revision(s) Affected** B, C

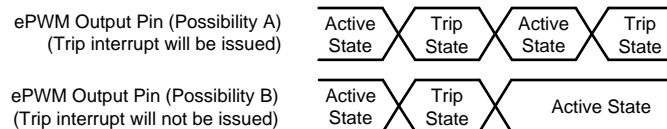
**Details** The blanking window is typically used to mask any PWM trip events during transitions which would be false trips to the system. If an ePWM trip event remains active for less than three ePWM clocks after the end of the blanking window cycles, there can be an undesired glitch at the ePWM output.

Figure 8 illustrates the time period which could result in an undesired ePWM output.



**Figure 8. Undesired Trip Event and Blanking Window Expiration**

Figure 9 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 9. Resulting Undesired ePWM Outputs Possible**

**Workaround(s)** Extend or reduce the blanking window to avoid any undesired trip action.

**Advisory** ***ePWM: ePWM Dead-Band Delay Value Cannot be Set to 0 When Using Shadow Load Mode for RED/FED***

**Revision(s) Affected** B, C

**Details** ePWM dead-band delay value cannot be set to 0 when using Shadow Load Mode for rising-edge delay (RED) and falling-edge delay (FED).

- Workaround(s)**
1. Use Immediate Load Mode if DBRED/DBFED = 0.
  2. Do not use DBRED/DBFED = 0 if in Shadow Load Mode.
- This is for both RED and FED.

<b>Advisory</b>	<b><i>CMPSS: COMPxLATCH May Not Clear Properly Under Certain Conditions</i></b>
<b>Revision(s) Affected</b>	B, C
<b>Details</b>	<p>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.</p> <p>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:</p> $\text{LATENCY} = 1 + (1 \times \text{FILTER\_PRESCALE}) + (\text{FILTER\_THRESH} \times \text{FILTER\_PRESCALE})$ <p>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.</p>
<b>Workaround(s)</b>	<p>Allow the Digital Filter output to resolve to logical 0 before clearing COMPxLATCH.</p> <p>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).</p> <p>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.</p>
<b>Advisory</b>	<b><i>CMPSS: Ramp Generator May Not Start Under Certain Conditions</i></b>
<b>Revision(s) Affected</b>	B, C
<b>Details</b>	<p>The Ramp Generator is designed to produce a falling-ramp DAC reference that is synchronized with a PWMSYNC signal. Upon receiving a PWMSYNC signal, the Ramp Generator will start to decrement its DAC value. When COMPSTS[COMPHSTS] is asserted by a trip event, the Ramp Generator will stop decrementing its DAC value.</p> <p>If COMPSTS[COMPHSTS] is asserted simultaneously with a PWMSYNC signal, the desired behavior is for the PWMSYNC signal to take priority such that the Ramp Generator starts to decrement in the new EPWM cycle. Instead of the desired behavior, the COMPSTS[COMPHSTS] trip condition will take priority over PWMSYNC such that the Ramp Generator stops decrementing for a full EPWM cycle until the next PWMSYNC signal is detected.</p>
<b>Workaround(s)</b>	<p>Avoid COMPSTS[COMPHSTS] trip conditions when PWMSYNC is generated. For example, peak current mode control applications can limit the PWM duty cycle to a maximum value that will avoid simultaneous COMPSTS[COMPHSTS] and PWMSYNC assertions.</p>

---

**Advisory** ***GPIO: Open-Drain Configuration May Drive a Short High Pulse***


---

**Revision(s) Affected** B, C

**Details**

Each GPIO can be configured to an open-drain mode using the GPxODR register. However, an internal device timing issue may cause the GPIO to drive a logic-high for up to 0–10 ns during the transition into or out of the high-impedance state.

This undesired high-level may cause the GPIO to be in contention with another open-drain driver on the line if the other driver is simultaneously driving low. The contention is undesirable because it applies stress to both devices and results in a brief intermediate voltage level on the signal. This intermediate voltage level may be incorrectly interpreted as a high level if there is not sufficient logic-filtering present in the receiver logic to filter this brief pulse.

**Workaround(s)**

If contention is a concern, do not use the open-drain functionality of the GPIOs; instead, emulate open-drain mode in software. Open-drain emulation can be achieved by setting the GPIO data (GPxDAT) to a static 0 and toggling the GPIO direction bit (GPxDIR) to enable and disable the drive low. For an example implementation, see the code below.

```
void main(void)
{ ...

    // GPIO configuration
    EALLOW;
    GpioCtrlRegs.GPxPUD.bit.GPIOx = 1; // disable pullup
    GpioCtrlRegs.GPxODR.bit.GPIOx = 0; // disable open-drain mode
    GpioCtrlRegs.GPxODR.bit.GPIOx = 0; // set GPIO to drive static 0 before
    // enabling output

    GpioDataRegs.GPxCLEAR.bit.GPIOx = 1;
    EDIS;
    ...

    // application code
    ...

    // To drive 0, set GPIO direction as output
    GpioCtrlRegs.GPxDIR.bit.GPIOx = 1;

    // To tri-state the GPIO(logic 1),set GPIO as input
    GpioCtrlRegs.GPxDIR.bit.GPIOx = 0;
}

```

---

<b>Advisory</b>	<b><i>Boot ROM: Device Will Hang During Boot if X1 Clock Source is not Present</i></b>
<b>Revision(s) Affected</b>	B
<b>Details</b>	The device boot code will attempt to configure the device using X1 as the clock source. When X1 is not present, the device boot will hang. This advisory applies to any system which is designed to use the INTOSC as the primary clock source with no clock on X1 during boot. This issue only affects some silicon revision B devices and it will be fixed in all future silicon revisions.
<b>Workaround(s)</b>	Apply external clock source to X1 on silicon revision B devices, even if using INTOSC as the application clock source.

## 5 Documentation Support

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

For more information regarding the TMS320F2837xS Delfino devices, see the following documents:

- [TMS320F2837xS Delfino™ Microcontrollers Data Manual](#)
- [TMS320F2837xS Delfino Microcontrollers Technical Reference Manual](#)

**Trademarks**

Delfino, PowerPAD, TMS320 are trademarks of Texas Instruments.  
All other trademarks are the property of their respective owners.



## Revision History

Changes from January 4, 2018 to November 16, 2018 (from F Revision (January 2018) to G Revision)	Page
• <b>Global:</b> Added TMS320F28378S. ....	1
• <a href="#">Section 4.2</a> (Known Design Exceptions to Functional Specifications): Replaced controlSUITE with C2000Ware in <a href="#">ADC: ADC Offset Trim in Different Modes</a> advisory. ....	7
• <a href="#">Section 4.2</a> : Updated second exception in Details section of <a href="#">Flash: Minimum Programming Word Size</a> advisory. ....	10
• <a href="#">Section 4.2</a> : Added <a href="#">INTOSC: VDDOSC Powered Without VDD Can Cause INTOSC Frequency Drift</a> advisory. ....	23
• <a href="#">Section 4.2</a> : Updated <a href="#">Low-Power Modes: Power Down Flash or Maintain Minimum Device Activity</a> advisory. ....	24
• <a href="#">Section 4.2</a> : Added <a href="#">I2C: SDA and SCL Open-Drain Output Buffer Issue</a> advisory. ....	25
• <a href="#">Section 4.2</a> : Added <a href="#">ePWM: An ePWM Glitch can Occur if a Trip Remains Active at the End of the Blanking Window</a> advisory. ....	27
• <a href="#">Section 4.2</a> : Added <a href="#">ePWM: ePWM Dead-Band Delay Value Cannot be Set to 0 When Using Shadow Load Mode for RED/FED</a> advisory. ....	27
• <a href="#">Section 4.2</a> : Added <a href="#">GPIO: Open-Drain Configuration May Drive a Short High Pulse</a> advisory. ....	29

## 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, 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 ([www.ti.com/legal/termsofsale.html](http://www.ti.com/legal/termsofsale.html)) or other applicable terms available either on [ti.com](http://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.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265  
Copyright © 2018, Texas Instruments Incorporated