Errata

TMS320F2837xS Real-Time MCUs Silicon Errata Silicon Revisions C, B

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	
		B	C
Section 3.1.1	PIE: Spurious Nested Interrupt After Back-to-Back PIEACK Write and Manual CPU Interrupt Mask Clear	Yes	Yes
Section 3.1.2	Caution While Using Nested Interrupts	Yes	Yes
Section 3.1.3	SYS/BIOS: Version Implemented in Device ROM is not Maintained	Yes	Yes
Section 3.1.4	SDFM: Use Caution While Using SDFM Under Noisy Conditions	Yes	Yes
Section 3.1.5	McBSP: XRDY Bit can Hold the Not-Ready Status (0) if New Data is Written to the DX1 Register Without Verifying if the XRDY Bit is in its Ready State (1)	Yes	Yes

1.2 Advisories Matrix

Table 1-2. Advisories Matrix

MODULE	DESCRIPTION	SILICON REVISIONS AFFECTED	
		B	C
	Analog Bandgap References	Yes	Yes
	Analog Trim of Some TMX Devices	Yes	
ADC	ADC: ADC Post-Processing Block Limit Compare	Yes	Yes
ADC	ADC: Interrupts may Stop if INTxCONT (Continue-to-Interrupt Mode) is not Set	Yes	Yes
ADC	ADC: ADC Offset Trim in Different Modes	Yes	Yes
ADC	ADC: DMA Read of Stale Result	Yes	Yes
ADC	ADC: Random Conversion Errors	Yes	
ADC	ADC: ADC PPB Event Trigger (ADCxEVT) to ePWM Digital Compare Submodule	Yes	
ADC	ADC: 12-Bit Switch Resistance	Yes	
ADC	ADC: 12-Bit Input Capacitance When Switching Channel Groups	Yes	
	$\overline{\text{XRS}}$ may Toggle During Power Up	Yes	
CLB	CLB: Back-to-Back PUSH or PULL Instructions With More Than One Active High-Level Controller (HLC) Channel is Not Supported	Yes	Yes
USB	USB: USB DMA Event Triggers are not Supported	Yes	Yes
VREG	VREG: VREG Will be Enabled During Power Up Irrespective of VREGENZ	Yes	
Flash	Flash: A Single-Bit ECC Error May Cause Endless Calls to Single-Bit-Error ISR	Yes	Yes
Flash	Flash: Minimum Programming Word Size	Yes	Yes
ePIE	ePIE: Spurious VCU Interrupt (ePIE 12.6) Can Occur When First Enabled	Yes	
eQEP	eQEP: Position Counter Incorrectly Reset on Direction Change During Index	Yes	Yes
eQEP	eQEP: eQEP Inputs in GPIO Asynchronous Mode	Yes	Yes
HWBIST	HWBIST: Avoiding Spurious Interrupts While Using HWBIST	Yes	Yes
PLL	PLL: May Not Lock on the First Lock Attempt	Yes	Yes
PLL	PLL: Power Down and Bypass May Take up to 120 SYSCLK Cycles to be Effective	Yes	Yes
SDFM	SDFM: Data Filter Output Does Not Saturate at Maximum Value With Sinc3 and OSR = 256	Yes	Yes
SDFM	SDFM: Spurious Data Acknowledge Event When Data Filter is Configured and Enabled for the First Time	Yes	Yes
SDFM	SDFM: Spurious Data Acknowledge Event When Data Filter is Synchronized Using PWM FILRES Signal	Yes	Yes

Table 1-2. Advisories Matrix (continued)

MODULE	DESCRIPTION	SILICON REVISIONS AFFECTED	
		B	C
SDFM	SDFM: Comparator Filter Module may Generate Spurious Over-Value and Under-Value Conditions	Yes	Yes
SDFM	SDFM: Dynamically Changing Threshold Settings (LLT, HLT), Filter Type, or COSR Settings Will Trigger Spurious Comparator Events	Yes	Yes
SDFM	SDFM: Dynamically Changing Data Filter Settings (Such as Filter Type or DOSR) Will Trigger Spurious Data Acknowledge Events	Yes	Yes
SDFM	SDFM: Manchester Mode (Mode 2) Does Not Produce Correct Filter Results Under Several Conditions	Yes	Yes
FPU	FPU: FPU-to-CPU Register Move Operation Preceded by Any FPU 2p Operation	Yes	Yes
FPU	FPU: LUF, LVF Flags are Invalid for the EINVF32 and EISQRTF32 Instructions	Yes	Yes
Memory	Memory: Prefetching Beyond Valid Memory	Yes	Yes
INTOSC	INTOSC: V_{DDOSC} Powered Without V_{DD} Can Cause INTOSC Frequency Drift	Yes	Yes
Low-Power Modes	Low-Power Modes: Power Down Flash or Maintain Minimum Device Activity	Yes	Yes
I2C	I2C: SDA and SCL Open-Drain Output Buffer Issue	Yes	Yes
I2C	I2C: Target Transmitter Mode, Standard Mode SDA Timings Limitation	Yes	Yes
ePWM	ePWM: An ePWM Glitch can Occur if a Trip Remains Active at the End of the Blanking Window	Yes	Yes
ePWM	ePWM: ePWM Dead-Band Delay Value Cannot be Set to 0 When Using Shadow Load Mode for RED/FED	Yes	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	Yes
SYSTEM	SYSTEM: Multiple Successive Writes to CLKSRCCTL1 Can Cause a System Hang	Yes	Yes
CMPSS	CMPSS: COMPxLATCH May Not Clear Properly Under Certain Conditions	Yes	Yes
CMPSS	CMPSS: Ramp Generator May Not Start Under Certain Conditions	Yes	Yes
GPIO	GPIO: Open-Drain Configuration May Drive a Short High Pulse	Yes	Yes
	During DCAN FIFO Mode, Received Messages May be Placed Out of Order in the FIFO Buffer	Yes	Yes
Boot ROM	Boot ROM: Calling SCI Bootloader from Application	Yes	Yes
Boot ROM	Boot ROM: Device Will Hang During Boot if X1 Clock Source is not Present	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 [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).

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.

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).

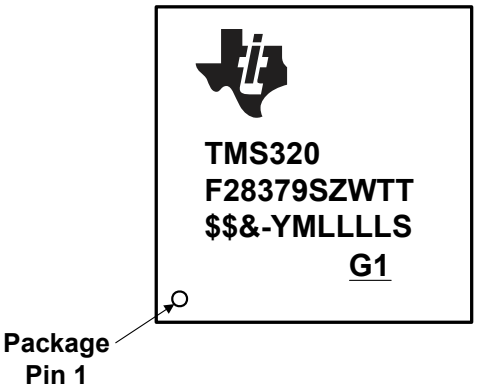
2.2 Devices Supported

This document supports the following devices:

- TMS320F28379S
- TMS320F28378S
- TMS320F28377S
- TMS320F28377S-Q1
- TMS320F28376S
- TMS320F28375S
- TMS320F28375S-Q1
- TMS320F28374S

2.3 Package Symbolization and Revision Identification

Figure 2-1 and Figure 2-2 provide examples of the 2837xS device markings and define each of the markings. The device revision can be determined by the symbols marked on the top of the package as shown in Figure 2-1 and Figure 2-2. Some prototype devices may have markings different from those illustrated. Figure 2-3 shows an example of the device nomenclature.

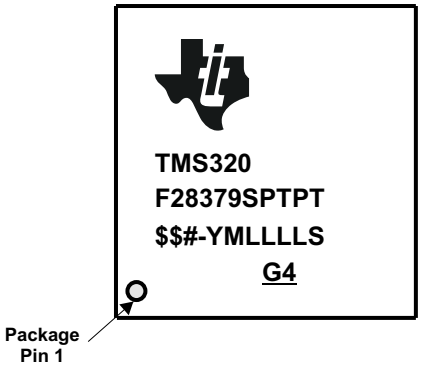


YMLLLLS = Lot Trace Code

YM = 2-digit Year/Month Code
LLLL = Assembly Lot Code
S = Assembly Site Code
$$ = Wafer Fab Code (one or two characters as applicable)
& = PG Revision

G1 = Green (Low Halogen and RoHS-compliant)

Figure 2-1. Example of Package Symbolization – ZWT



YMLLLLS = Lot Trace Code

YM = 2-Digit Year/Month Code
LLLL = Assembly Lot
S = Assembly Site Code
$$ = Wafer Fab Code as applicable
 # = Silicon Revision Code

G4 = Green (Low Halogen and RoHS-compliant)

Figure 2-2. Example of Package Symbolization – PTP

Table 2-1. Determining Silicon Revision From Lot Trace Code

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

(1) Silicon Revision ID

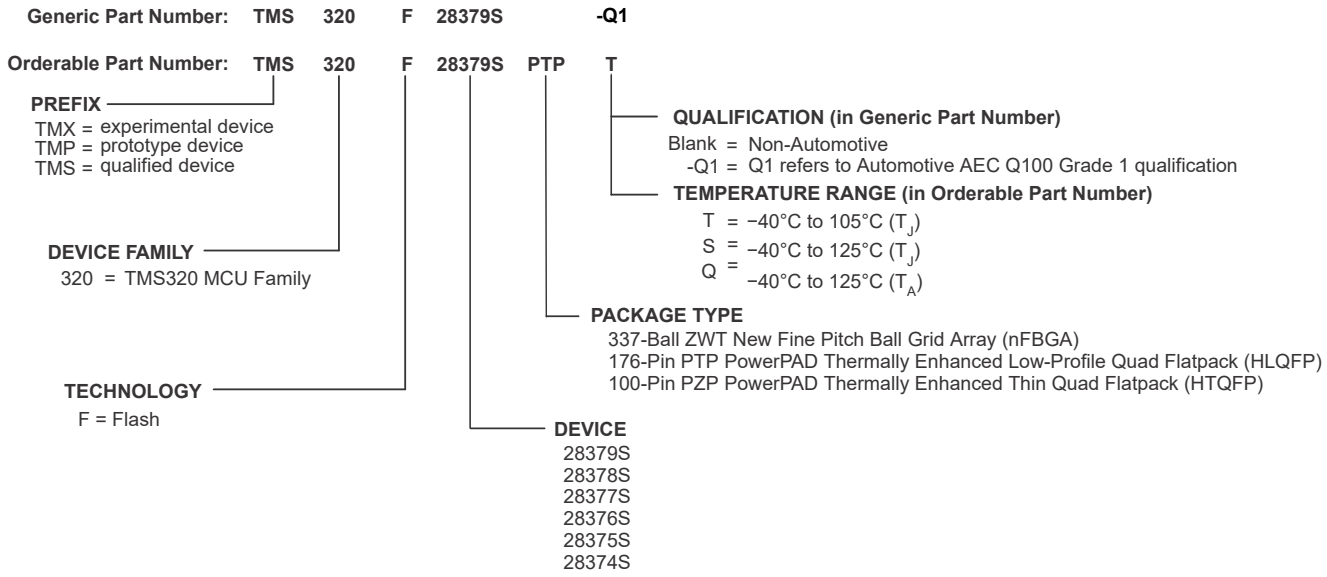


Figure 2-3. Example of Device Nomenclature

3 Silicon Revision C Usage Notes and Advisories

This section lists the usage notes and advisories for this silicon revision.

3.1 Silicon Revision C Usage Notes

This section lists all the usage notes that are applicable to silicon revision C [and earlier silicon revisions].

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

Revisions 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")).

Workarounds: 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

Revisions 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.

3.1.3 SYS/BIOS: Version Implemented in Device ROM is not Maintained

Revisions Affected: B, C

The SYS/BIOS version 6.37 in ROM cannot be updated to address bug fixes or enhancements. For applications using this ROM version of SYS/BIOS, the user should review the release notes on the SYS/BIOS Product Releases page to assess any application impact to known bugs.

To use the latest maintained SYS/BIOS version, download the SYS/BIOS library and include in the application flash image instead of using the ROM version.

3.1.4 SDFM: Use Caution While Using SDFM Under Noisy Conditions

Revisions Affected: B, C

The SDFM clock input (SDx_Cy) directly clocks the SDFM module when there is no GPIO input synchronization. Any glitches or ringing noise on the SDx_Cy input beyond V_{IH} or V_{IL} can corrupt the SDFM module, leading to unpredictable results.

Workarounds:

SDFM GPIO Asynchronous Mode:

Special attention should be taken during board design to ensure a clean and noise-free signal that meets the SDFM timing requirements. Precautions such as series termination for ringing due to any impedance mismatch of the clock driver, and spacing of traces from other high-frequency signals are recommended.

SDFM GPIO Qualification (3-sample) Mode:

It is highly recommended that the SDFM GPIO qualification mode be used in noisy conditions. This mode provides additional protection by filtering both SDx_Cy and SDx_Dy inputs from system noise. Refer to the "SDFM Timing Requirements When Using GPIO Input Qualification (3-Sample Window)" table in the *TMS320F2837xS Real-Time Microcontrollers* data sheet when using this option.

When a noise event occurs while using the GPIO Qualification mode, there may still be data disturbance, but it will be proportional to the duration of the noise event and typically filtered by the oversampling of the SDFM module. Below is a relative listing of each SDFM mode's sensitivity to data variation in the presence of severe system noise.

- Mode 0: This mode is the best performing mode. It is the recommended mode under noisy conditions.
- Mode 1 and Mode 3: The error in these modes can be twice as large as the error in Mode 0 since each noise glitch can introduce error for multiple data bits.
- Mode 2: This option is unavailable when using GPIO qualification. This mode is not recommended for either GPIO ASYNC or GPIO qualification.

3.1.5 McBSP: XRDY Bit can Hold the Not-Ready Status (0) if New Data is Written to the DX1 Register Without Verifying if the XRDY Bit is in its Ready State (1)

Revisions Affected: B, C

If the XRDY bit is used to properly gate writes to the DX2/DX1 registers, this condition will not occur.

Per the operation of the McBSP, a write to the DX1 data transmit register will automatically clear the XRDY bit, indicating a not-ready status. Once this data is transferred to the internal transmit shift register (XSR1), the McBSP HW will set the XRDY bit, indicating a ready status, and new data can be written to DX2/DX1 data transmit registers.

If the set and clear of XRDY occur on the same CPU clock cycle, the XRDY bit will remain cleared and the new data in the DX2/DX1 will not be transmitted.

In this state of XRDY = 0, the McBSP will be in the not-ready status indefinitely.

Any subsequent writes to DX2/DX1 will behave normally and the XRDY bit will function as normal.

Workarounds:

When transmitting multiple words of data using the McBSP module, poll the XRDY bit in the SPCR2 register before writing new data to the DX2/DX1 registers to prevent overwriting. For those modules that do not have access to the XRDY bit (such as the DMA controller), the XINT interrupt inside the McBSP module can be configured to reflect XRDY (through the XINTM bits in SPCR2 register), and this can also be used to gate writes to the DX2/DX1 registers. This will also ensure that the XRDY bit is not set and cleared on the same CPU cycle, causing the above “not-ready indefinitely” condition.

If the system allows multiple bus controllers, such as the C28x CPU and the DMA controller, to write to the DX2/DX1 registers, then the ready state of the XRDY bit should be validated before passing control of the McBSP to a different bus controller. This will ensure that the state of XRDY is accurate and the simultaneous set/clear action does not occur.

3.2 Silicon Revision C Advisories

This section lists all the advisories that are applicable to silicon revision C [and earlier silicon revisions].

Advisory ***Analog Bandgap References***

Revisions Affected

B, C

Details

The Analog Subsystem includes internal bandgap reference circuits that are shared by the embedded analog modules. Table 3-1 shows the bandgap usage by module.

Table 3-1. Bandgap Usage by Module

BANDGAP	ADC	BUFFERED DAC	CMPSS
BGA	ADCA	DACA DACB	CMPSS1 CMPSS2
BGB	ADCB	DACC	CMPSS3 CMPSS4
BGC	ADCC		CMPSS5 CMPSS6
BGD	ADCD		CMPSS7 CMPSS8

Each bandgap reference—BGA, BGB, BGC, or BGD—will power up when one or more of the dependent modules are enabled. An active bandgap reference will power down if all dependent modules are disabled.

For example, bandgap B (BGB) is powered down unless one or more of the following register bits are set:

- AdcbRegs.ADCCTL1.bit.ADCPWDNZ
- DaccRegs.DACOUTEN.bit.DACOUTEN
- Cmpss3Regs.COMPCTL.bit.COMPDACE
- Cmpss4Regs.COMPCTL.bit.COMPDACE

The CMPSS and GPDAC power-up time specification in the *TMS320F2837xS Real-Time Microcontrollers* data sheet previously did not account for the bandgap power-up time. This 10-µs value has been increased to 500 µs to account for the bandgap power-up time.

Workarounds

If your application was utilizing a power-up time of 10 µs for the CMPSS and GPDAC, you do not need to increase it to 500 µs if the respective ADC on that bandgap was turned on before the CMPSS and GPDAC, and the ADC power-up time of 500 µs was adhered to.

For simplicity, it is recommended that 500 µs be used as the power-up time for both CMPSS and GPDAC.

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

Revisions 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.

Workarounds

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: Interrupts may Stop if INTxCONT (Continue-to-Interrupt Mode) is not Set***

Revisions Affected

B, C

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.

Workarounds

1. Use Continue-to-Interrupt Mode to prevent the ADCINTFLG from blocking additional ADC interrupts:

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

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.

```
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: ADC Offset Trim in Different Modes***
Revisions 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.
Workarounds	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, offset and linearity trims are not loaded, and must be populated by calling AdcSetMode() from the user application.

Advisory	***ADC: DMA Read of Stale Result***
Revisions Affected	B, C
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 tables of the *TMS320F2837xS 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 for 12-bit mode). • 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.
Workarounds	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	***CLB: Back-to-Back PUSH or PULL Instructions With More Than One Active High-Level Controller (HLC) Channel is Not Supported***
Revisions Affected	B, C
Details	This issue only affects HLC operations if using back-to-back PUSH or PULL instructions with more than one active HLC channel.
Workarounds	Re-order the HLC instructions so that every PUSH instruction is followed by a non-PUSH instruction. The same applies for the PULL instruction. If only one HLC channel is active, this issue does not occur.

Advisory	***USB: USB DMA Event Triggers are not Supported***
Revisions 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.
Workarounds	None

Advisory	***Flash: A Single-Bit ECC Error May Cause Endless Calls to Single-Bit-Error ISR***
Revisions Affected	B, C
Details	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.
Workarounds	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.

Advisory	***Flash: Minimum Programming Word Size***
Revisions Affected	B, C
Details	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. 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. 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: 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. 2. The DCSM Zx-LINKPOINTER3 value in the DCSM OTP may be programmed 1 bit at a time as required by the DCSM operation.
Workarounds	All applications should follow the restrictions outlined in this advisory. Contact TI for devices already in production which violate this advisory.

Advisory	***eQEP: Position Counter Incorrectly Reset on Direction Change During Index***
Revisions Affected	B, C
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	***eQEP: eQEP Inputs in GPIO Asynchronous Mode***
Revisions Affected	B, C
Details	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. 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. The asynchronous mode should not be used for eQEP module input pins.
Workarounds	Configure GPIO inputs configured as eQEP pins for non-asynchronous mode (any GPxQSELn register option except “11b = Asynchronous”).

Advisory	***HWBIST: Avoiding Spurious Interrupts While Using HWBIST***
Revisions Affected	B, C

Details

HWBIST has the capability to log interrupts received while the CPU is under test and reissue them after HWBIST completes. Interrupts received in the clock cycle before the interrupt logging is enabled are executed before the HWBIST runs. In the next cycle, when interrupt logging is enabled, interrupts are logged and reissued when 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 is executed once as expected before the logging (clearing the CPU Timer TCR.TIF flag), but then is logged by the interrupt logger and triggered again after HWBIST completes. Because the TCR.TIF flag was already cleared by the previous ISR, this is an unexpected spurious interrupt.

This 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.

Workarounds

Disable CPU Timer 1 and 2 interrupts before enabling interrupt logging and restore them later. 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: May Not Lock on the First Lock Attempt***
Revisions 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.

Workarounds

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.

Advisory (continued) ***PLL: May Not Lock on the First Lock Attempt***

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.

Advisory ***PLL: Power Down and Bypass May Take up to 120 SYSCLK Cycles to be Effective***

Revisions Affected

B, C

Details

When the PLL is powered down (that is, SYPLLCTL1.PLLEN = 0) or bypassed (that is, SYSPLLCTL1.PLLCLKEN = 0), there is a necessary period of clock synchronization before the PLL bypass completes. During this time, if PLLSYSCLKDIV (or other clock divider) is set to a smaller value, the resulting system clock could be unexpectedly more than the rated device frequency.

Implementing the workaround below will allow the PLL bypass operation to complete before any other code is executed, ensuring expected device frequencies and proper system operation.

Workarounds

Add a software delay of 120 SYSCLK cycles using a NOP instruction while performing either a PLL power down or a PLL bypass operation.

Example:

```
SYSPLLCTL1.PLLCLKEN = 0;       // Bypassing the PLL
asm("  RPT #120 || NOP");      // Delay of 120 SYSCLK Cycles
SYSPLLCTL1.PLLEN = 0;          // Powering down the PLL
asm("  RPT #120 || NOP");      // Delay of 120 SYSCLK Cycles
```

The latest released C2000Ware, which has this workaround implemented, can be used as reference.

Advisory	***SDFM: Data Filter Output Does Not Saturate at Maximum Value With Sinc3 and OSR = 256***
Revisions Affected	B, C
Details	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).
Workarounds	Maintain the differential input of the SDFM in the specified linear input range as specified in the modulator data sheet.

Advisory	***SDFM: Spurious Data Acknowledge Event When Data Filter is Configured and Enabled for the First Time***
Revisions Affected	B, C
Details	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.
Workarounds	Do the following: 1. Configure and enable the SDFM data filter. 2. Delay for at least latency of data filter + 5 SD-Cx clock cycles. 3. Enable SDFM data acknowledge interrupts/DMA events.

Advisory	***SDFM: Spurious Data Acknowledge Event When Data Filter is Synchronized Using PWM FILRES Signal***
Revisions Affected	B, C
Details	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.
Workarounds	Do the following: 1. Choose any PWMx to work in the same time base as the PWM that generates the FILRES pulse. 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. 3. SDFM_CPUISR or SDFM_CLATask: a. Collect the required number of samples, N, after the FILRES pulse. b. 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.

Advisory	***SDFM: Comparator Filter Module may Generate Spurious Over-Value and Under-Value Conditions***
Revisions Affected	B, C
Details	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 *TMS320F2837xS Real-Time Microcontrollers Technical Reference Manual*.
Workarounds	These erroneous interrupts can be eliminated by implementing the following workaround: • Comparator OSR (COSR) value should be greater than or equal to 5. • 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.

Advisory ***SDFM: Dynamically Changing Threshold Settings (LLT, HLT), Filter Type, or COSR Settings Will Trigger Spurious Comparator Events***

Revisions Affected

B, C

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.

Workarounds

When comparator settings need to be changed dynamically, follow the procedure below to ensure spurious comparator events do not generate a CPU interrupt or CLA task:

1. Disable the SDFM comparator interrupt.
2. Change comparator settings such as lower/upper threshold, filter type, or COSR.
3. COSR value should be greater than or equal to 5.
4. Delay for at least a latency of comparator filter + 5 SD-Cx clock cycles.
5. Enable the SDFM comparator interrupt.

When comparator settings need to be changed dynamically, follow the procedure below to ensure spurious comparator events do not trigger X-BAR events (ePWM X-BAR and GPIO output X-BAR events):

1. Disable the SDFM X-BAR trip events in the corresponding X-BAR registers (ePWM X-BAR or GPIO X-BAR event).
2. Change comparator settings such as lower/upper threshold, filter type, or COSR.
3. COSR value should be greater than or equal to 5.
4. Delay for at least a latency of comparator filter + 5 SD-Cx clock cycles.
5. Enable the SDFM X-BAR trip events in the corresponding X-BAR registers (ePWM X-BAR or GPIO X-BAR event).

Advisory ***SDFM: Dynamically Changing Data Filter Settings (Such as Filter Type or DOSR) Will Trigger Spurious Data Acknowledge Events***

Revisions Affected

B, C

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.

Workarounds

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 SDFM data filter.
2. Change SDFM data filter settings such as filter type or DOSR.
3. Delay for at least a latency of data filter + 5 SD-Cx clock cycles.
4. Enable the SDFM data filter.

Advisory ***SDFM: Manchester Mode (Mode 2) Does Not Produce Correct Filter Results Under Several Conditions***

Revisions Affected

B, C

Details

The Manchester decoding algorithm samples the Manchester bitstream with SYSCLK in a calibration window of 1024 SDx_Dy signal transitions. The derived clock from the Manchester bitstream is used to sample for data in the subsequent calibration window cycle.

There are several scenarios that can cause large errors in the filter results:

- Any single noise event on SDx_Dy can corrupt the decoded Manchester clock and cause subsequent data to be sampled at an incorrect frequency.
- If the Manchester bitstream clock rate is a near exact integer multiple of SYSCLK, then an occasional Manchester bit can be skipped when the phases of the Manchester stream and internal SYSCLK drift past each other in phase before the next 1024 transition calibration window becomes effective. Deviations in duty cycle from 50% of the Manchester clock also need to be accounted for to ensure the longer Manchester pulses are not an integer multiple of SYSCLK. This situation can be unavoidable if the clock sources for either the SD modulator or this device have a wide variation since a wide range of keep out frequencies become problematic
- If the Manchester edge delay variation between rising and falling (duty cycle of the bitstream) is greater than one SYSCLK, then the SDFM clock decode algorithm can incorrectly identify the clock period as shorter than it is.

Workarounds

The workarounds available are:

- Avoid using Manchester mode and consider using Mode 0, which provides the best filter performance under noisy conditions. This is the recommended workaround.
- Avoid any noise on the Manchester bitstream and avoid integer multiples of SYSCLK for the selected Manchester clock source. A precision clock source for the modulator and this device must be used.
- Ensure rising and falling edge delays (high and low pulses) are within one SYSCLK of each other in length.
- Design an application-level algorithm that is robust against occasional incorrect SDFM results.

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

Revisions Affected

B, C

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 3-1 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 3-1. 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 3-2 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 3-2. Pipeline Diagram of the Issue if There is a Stall in the E3 Slot of the Instruction I1

Workarounds

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 3-3 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 3-3. Pipeline Diagram With Workaround in Place

Advisory — *FPU: LUF, LVF Flags are Invalid for the EINVF32 and EISQRTF32 Instructions*

Revisions 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.

Workarounds

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.

Advisory (continued) ***FPU: LUF, LVF Flags are Invalid for the EINVF32 and EISQRTF32 Instructions***

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
}
```

Advisory (continued) ***FPU: LUF, LVF Flags are Invalid for the EINVF32 and EISQRTF32 Instructions***

Note

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 Workarounds 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***
Revisions 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.

Workarounds

M1, GS11, GS15 – 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 3-2. Memories Impacted by Advisory

MEMORY TYPE	ADDRESSES IMPACTED	F28379S 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 (Bank 0)	0x000B FFF0–0x000B FFFF	No	Yes
Flash (Bank 1)	0x000F FFF0–0x000F FFFF	Yes	N/A

Advisory	***INTOSC: V_{DDOSC} Powered Without V_{DD} Can Cause INTOSC Frequency Drift***
Revisions Affected	B, C
Details	The "G" revision of the *TMS320F2837xS Real-Time Microcontrollers* data sheet (SPRS881G) has updated power sequencing requirements. Revision "F" and earlier revisions of the data sheet 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 sheet. This would impact applications using INTOSC2 as the clock source for the PLL, with the system operating at a lower frequency than expected.
Workarounds	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.

Advisory ***Low-Power Modes: Power Down Flash or Maintain Minimum Device Activity***

Revisions Affected

B, C

Details

The device has an intentional current path from V_{DD3VFL} (flash supply) to V_{DD}. Since the HALT, STANDBY, IDLE, or other low-activity device conditions can have low current demand on V_{DD}, this V_{DD3VFL} current can cause V_{DD} to rise above the recommended operating voltage.

There will be zero current load to the external system V_{DD} 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.

Workarounds

Workaround 1: 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.

```
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***
Revisions 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.

Workarounds

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 3-4.

Advisory (continued) ***I2C: SDA and SCL Open-Drain Output Buffer Issue***

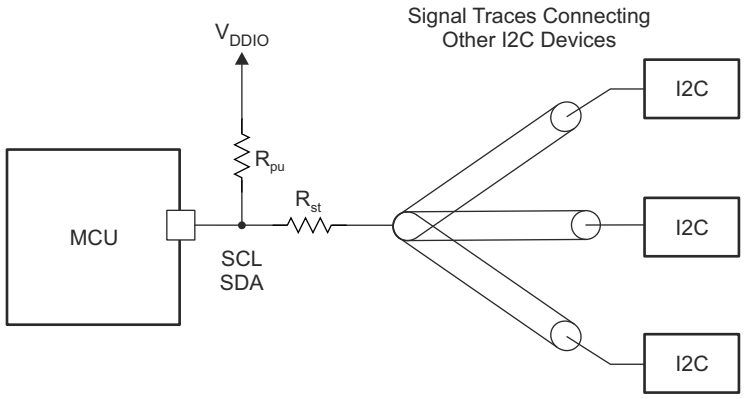


Figure 3-4. Placement of Series Termination Resistor and Pullup Resistor

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

Revision Affected

B, C

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

Revisions 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 3-5 illustrates the time period which could result in an undesired ePWM output.

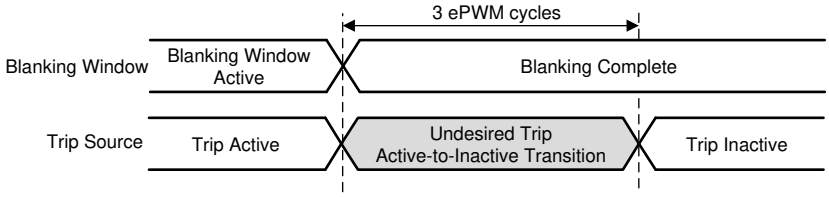


Figure 3-5. Undesired Trip Event and Blanking Window Expiration

Figure 3-6 illustrates the two potential ePWM outputs possible if the trip event ends within 1 cycle before or 3 cycles after the blanking window closes.

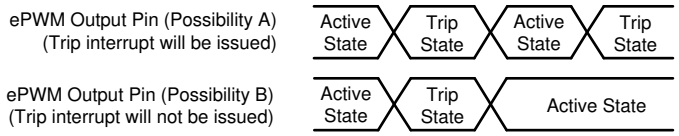


Figure 3-6. Resulting Undesired ePWM Outputs Possible

Workarounds

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***

Revisions 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).

Workarounds

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.

Advisory ***ePWM: Trip Events Will Not be Filtered by the Blanking Window for the First 3 Cycles After the Start of a Blanking Window***

Revisions Affected

B, C

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).

Workarounds

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 ***SYSTEM: Multiple Successive Writes to CLKSRCCTL1 Can Cause a System Hang***

Revisions Affected

B, C

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 $\overline{\text{XRS}}$ 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.

Workarounds

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	***CMPSS: COMPxLATCH May Not Clear Properly Under Certain Conditions***
Revisions Affected	B, C
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: Ramp Generator May Not Start Under Certain Conditions***
Revisions Affected	B, C
Details	The Ramp Generator is designed to produce a falling-ramp DAC reference that is synchronized with a PWMSYNC signal. Upon receiving a PMWSYNC 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. 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.
Workarounds	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.

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

Revisions 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.

Workarounds

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;                                  // disable pullup
     GpioCtrlRegs.GPxPUD.bit.GPIOx = 1;       // 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;
}
```

Advisory	***During DCAN FIFO Mode, Received Messages May be Placed Out of Order in the FIFO Buffer***
Revisions Affected	B, C
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.
Workarounds	None

Advisory	***Boot ROM: Calling SCI Bootloader from Application***
Revisions Affected	B, C
Details	The ROM SCI bootloader uses autobaud lock to lock the baud rate. The SCI baud rate is split between two registers, SCILBAUD and SCIHBAUD. The ROM SCI bootloader expects SCIHBAUD to contain its default reset value of zero. If calling the ROM SCI bootloader from an application that modified the contents of SCIHBAUD to be non-zero, then the SCI will not autobaud lock and the SCI bootloader will not execute.
Workarounds	Clear SCIHBAUD to zero before calling the ROM SCI bootloader.

4 Silicon Revision B Usage Notes and Advisories

This section lists the usage notes and advisories for this silicon revision.

4.1 Silicon Revision B Usage Notes

Silicon revision-applicable usage notes have been found on a later silicon revision. For more details, see Silicon Revision C Usage Notes.

4.2 Silicon Revision B Advisories

Silicon revision-applicable advisories have been found on a later silicon revision. For more details, see Silicon Revision C Advisories.

Advisory ***Analog Trim of Some TMX Devices***

Revisions 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
ADC reference	AnalogSubsysRegs.ANAREFTRIMA AnalogSubsysRegs.ANAREFTRIMB AnalogSubsysRegs.ANAREFTRIMC AnalogSubsysRegs.ANAREFTRIMD	Degraded performance of the ADC for all specifications.
ADC linearity	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.
ADC offset	AdcaRegs.ADCOFFTRIM AdcbRegs.ADCOFFTRIM AdccRegs.ADCOFFTRIM AdcdRegs.ADCOFFTRIM	Degraded performance of the ADC offset error specification.
Internal oscillator	AnalogSubsysRegs.INTOSC1TRIM AnalogSubsysRegs.INTOSC2TRIM	Degraded frequency accuracy and temperature drift of the internal oscillators.
Buffered DAC offset	DacaRegs.DACTRIM DacbRegs.DACTRIM DaccRegs.DACTRIM	Degraded offset error specification of the buffered DAC. No workaround available.

Workarounds

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). This check is implemented in the Device_init() function, which will then call the Device_configureTMXAnalogTrim() function if needed. The user can place any additional self-calibration or static calibration code in the Device_configureTMXAnalogTrim() function.

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.

Advisory (continued) ***Analog Trim of Some TMX Devices***

Missing **ADC offset trim** can be generated by following the instructions in the "ADC Zero Offset Calibration" section of the *TMS320F2837xS Real-Time 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: Random Conversion Errors***

Revisions 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.

Workarounds

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).

Advisory ***ADC: ADC PPB Event Trigger (ADCxEVT) to ePWM Digital Compare Submodule***

Revisions Affected

B

Details

The ADCxEVT trigger to the ePWM digital compare submodule may not be detected by the ePWM.

Workarounds

The ADCxEVT can generate an ADCx_EVT interrupt to the PIE. The ISR can be used to perform the desired task in software.

Advisory ***ADC: 12-Bit Switch Resistance***

Revisions Affected

B

Details

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 kΩ should be used to select the S+H duration for these revisions.

Workarounds

For the revisions affected, the S+H duration should be chosen to account for the additional switch resistance.

Advisory ***ADC: 12-Bit Input Capacitance When Switching Channel Groups***

Revisions Affected

B

Details

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 Figure 4-1.

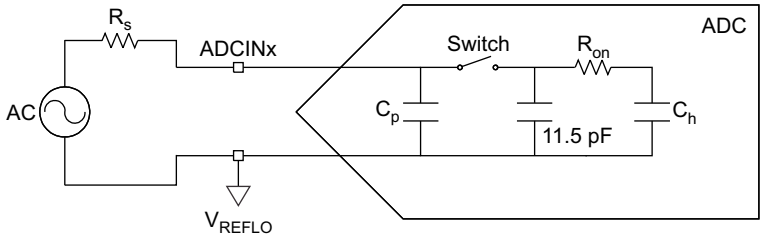


Figure 4-1. Single-Ended Input Model

Workarounds

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***
Revisions 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.
Workarounds	Disregard $\overline{XRS}$ activity on the board prior to supplies reaching recommended operating conditions.

Advisory	***VREG: VREG Will be Enabled During Power Up Irrespective of VREGENZ***
Revisions 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.
Workarounds	None

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

Workarounds	Before enabling VCU interrupt 12.6, execute the following instructions to avoid the spurious interrupt.
	`// Clear VCU divide by zero status`
	`asm(" VCLRDIVE");`
	`// 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.

Advisory	***Boot ROM: Device Will Hang During Boot if X1 Clock Source is not Present***
Revisions Affected	B
Details	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.
Workarounds	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: https://www.ti.com.

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

- *TMS320F2837xS Real-Time Microcontrollers* data sheet
- *TMS320F2837xS Real-Time Microcontrollers Technical Reference Manual*

6 Trademarks

TMS320™ is a trademark of Texas Instruments.
C2000™ is a trademark of Texas Instruments.
All trademarks are the property of their respective owners.

7 Revision History

Changes from March 2, 2023 to May 15, 2024 (from Revision J (March 2023) to Revision K (May 2024)) **Page**

- *Example of Package Symbolization – ZWT* figure: Updated definition of G1.....5
- Updated Workarounds in ADC: ADC Offset Trim in Different Modes advisory.....11
- Added HWBIST: Avoiding Spurious Interrupts While Using HWBIST advisory.....15
- Added I2C: Target Transmitter Mode, Standard Mode SDA Timings Limitation advisory.....33

IMPORTANT NOTICE AND DISCLAIMER

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

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

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

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

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

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