TMS320F2806x Piccolo™ MCUs Silicon Revisions B, A, 0

1 Introduction

This document describes the silicon updates to the functional specifications for the TMS320F2806x microcontrollers (MCUs).

The updates are applicable to:

- 80-pin PowerPAD™ Thermally Enhanced Thin Quad Flatpack, PFP Suffix
- 80-pin Low-Profile Quad Flatpack, PN Suffix
- 100-pin PowerPAD Low-Profile Quad Flatpack, PZP Suffix
- 100-pin Low-Profile Quad Flatpack, PZ 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**320F28069). 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).

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

TMP Final silicon die that conforms to the device's electrical specifications but has not completed quality and reliability verification

TMS Fully qualified production device

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."

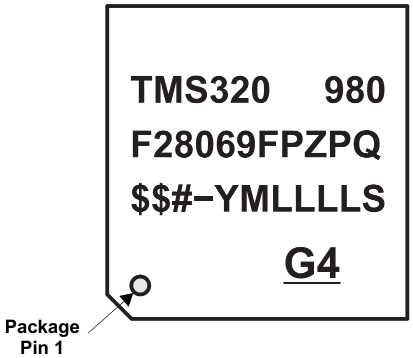
TMS devices and TMDS development-support tools have been characterized fully, and the quality and reliability of the device have been demonstrated fully. TI's standard warranty applies.

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, PZP) and temperature range (for example, S).

3 Device Markings

Figure 1 provides an example of the 2806x 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.



YMLLLLS = Lot Trace Code

YM = 2-Digit Year/Month Code
LLLL = Assembly Lot
S = Assembly Site Code
980 = TI E.I.A. Code
$$ = Wafer Fab Code (one or two characters) as applicable
= Silicon Revision Code

G4 = Green (Low Halogen and RoHS-compliant)

Figure 1. Example of Device Markings

Table 1. Determining Silicon Revision From Lot Trace Code

SILICON REVISION CODE	SILICON REVISION	REVISION ID Address: 0x0883	COMMENTS
Blank (no second letter in prefix)	Indicates Revision 0	0x0000	This silicon revision is available as TMX.
A	Indicates Revision A	0x0001	This silicon revision is available as TMS.
B	Indicates Revision B	0x0002	This silicon revision is available as TMS.

TMS 320 F 28069 PZP S

PREFIX
TMX = experimental device
TMP = prototype device
TMS = qualified device

DEVICE FAMILY
320 = TMS320 MCU Family

TECHNOLOGY
F = Flash

TEMPERATURE RANGE
T = −40°C to 105°C
S = −40°C to 125°C
Q = −40°C to 125°C
(Q refers to AEC Q100 qualification for automotive applications.)

PACKAGE TYPE
80-Pin PN Low-Profile Quad Flatpack (LQFP)
80-Pin PFP PowerPAD Thermally Enhanced Thin Quad Flatpack (HTQFP)
100-Pin PZ Low-Profile Quad Flatpack (LQFP)
100-Pin PZP PowerPAD Thermally Enhanced Thin Quad Flatpack (HTQFP)

DEVICE

28069	28069U	28069M	28069F
28068	28068U	28068M	28068F
28067	28067U		
28066	28066U		
28065	28065U		
28064	28064U		
28063	28063U		
28062	28062U		28062F

A For more information on peripheral, temperature, and package availability for a specific device, see the *TMS320F2806x Piccolo™ Microcontrollers Data Manual*.

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		
	0	**A**	**B**
PIE: Spurious Nested Interrupt After Back-to-Back PIEACK Write and Manual CPU Interrupt Mask Clear	Yes	Yes	Yes
CAN Bootloader: Internal Oscillator Tolerance is Not Sufficient for CAN Operation at High Temperatures	Yes	Yes	Yes
FPU32 and VCU Back-to-Back Memory Accesses	Yes	Yes	Yes
Caution While Using Nested Interrupts	Yes	Yes	Yes
Flash: MAX "Program Time" and "Erase Time" in Revision G of the *TMS320F2806x Piccolo™ Microcontrollers Data Manual* are only Applicable for Devices Manufactured After January 2018	Yes	Yes	Yes

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

Revision(s) Affected: 0, A, B

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 re-enables 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 CAN Bootloader: Internal Oscillator Tolerance is Not Sufficient for CAN Operation at High Temperatures

Revision(s) Affected: 0, A, B

The CAN bootloader in the device's boot ROM uses the internal oscillator as the source for the CAN bit clock. At high temperatures, the frequency of the internal oscillator can deviate enough to prevent messages from being received.

Workaround: Recalibrate the internal oscillator before invoking the CAN bootloader. This can be done in application code. For more flexibility, a wrapper function may be programmed into the device's OTP memory. See *Using the Piccolo™ CAN Bootloader at High Temperature* for details on how to implement this workaround.

4.1.3 FPU32 and VCU Back-to-Back Memory Accesses

Revision(s) Affected: 0, A, B

This usage note applies when a VCU memory access and an FPU memory access occur back-to-back. There are three cases:

Case 1. Back-to-back memory reads: one read performed by a VCU instruction (VMOV32) and one read performed by an FPU32 instruction (MOV32).

If an R1 pipeline phase stall occurs during the first read, then the second read will latch the wrong data. If the first instruction is not stalled during the R1 pipeline phase, then the second read will occur properly.

The order of the instructions—FPU followed by VCU or VCU followed by FPU—does not matter. The address of the memory location accessed by either read does not matter.

Case 1 Workaround: Insert one instruction between the two back-to-back read instructions. Any instruction, except a VCU or FPU memory read, can be used.

Case 1, Example 1:

```
VMOV32  VR1,mem32       ; VCU memory read
NOP                     ; Not a FPU/ VCU memory read
MOV32   R0H,mem32       ; FPU memory read
```

Case 1, Example 2:

```
VMOV32  VR1,mem32       ; VCU memory read
VMOV32  mem32, VR2      ; VCU memory write
MOV32   R0H,mem32       ; FPU memory read
```

Case 2. Back-to-back memory writes: one write performed by a VCU instruction (VMOV32) and one write performed by an FPU instruction (MOV32).

If a pipeline stall occurs during the first write, then the second write can corrupt the data. If the first instruction is not stalled in the write phase, then no corruption will occur.

The order of the instructions—FPU followed by VCU or VCU followed by FPU—does not matter. The address of the memory location accessed by either write does not matter.

Case 2 Workaround: Insert two instructions between the back-to-back VCU and FPU writes. Any instructions, except VCU or FPU memory writes, can be used.

Case 2, Example 1:

```
VMOV32  mem32,VR0        ; VCU memory write
NOP                      ; Not a FPU/VCU memory write
NOP                      ; Not a FPU/VCU memory write
MOV32   mem32,R3H        ; FPU memory write
```

Case 2, Example 2:

```
VMOV32  mem32,VR0        ; VCU memory write
VMOV32  VR1,  mem32      ; VCU memory read
NOP
MOV32   mem32,R3H        ; FPU memory write
```

Case 3. Back-to-back memory writes followed by a read or a memory read followed by a write. In this case, there is no interaction between the two instructions. No action is required.

Workaround: See Case 1 Workaround and Case 2 Workaround.

4.1.4 Caution While Using Nested Interrupts

Revision(s) Affected: 0, A, B

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.

4.1.5 Flash: MAX "Program Time" and "Erase Time" in Revision G of the *TMS320F2806x Piccolo™ Microcontrollers Data Manual* are only Applicable for Devices Manufactured After January 2018

Revision(s) Affected: 0, A, B

The MAX parameters added for the flash "Program Time" and "Erase time" in revision G of the *TMS320F2806x Piccolo™ Microcontrollers Data Manual* are only applicable for devices manufactured after January 2018.

4.2 Known Design Exceptions to Functional Specifications

Table 3. Table of Contents for Advisories

Table 4 shows which silicon revision(s) are affected by each advisory.

Table 4. List of Advisories

TITLE	SILICON REVISION(S) AFFECTED		
	0	A	B
FPU: CPU-to-FPU Register Move Operation Followed By F32TOUI32, FRACF32, or UI16TOF32 Operations	Yes	Yes	Yes
FPU: FPU-to-CPU Register Move Operation Preceded by Any FPU 2p Operation	Yes	Yes	Yes
FPU: LUF, LVF Flags are Invalid for the EINVF32 and EISQRTF32 Instructions	Yes	Yes	Yes
ADC: Initial Conversion	Yes	Yes	Yes
ADC: Temperature Sensor Minimum Sample Window Requirement	Yes	Yes	Yes
ADC: ADC Result Conversion When Sampling Ends on 14th Cycle of Previous Conversion, ACQPS = 6 or 7	Yes	Yes	Yes
ADC: Offset Self-Recalibration Requirement	Yes	Yes	Yes
ADC: ADC Revision Register (ADCREV) Limitation	Yes	Yes	Yes
ADC: ADC can Become Non-Responsive When ADCNONOVERLAP or RESET is Written During a Conversion	Yes	Yes	Yes
Memory: Prefetching Beyond Valid Memory	Yes	Yes	Yes
GPIO: GPIO Qualification	Yes	Yes	Yes
eCAN: Abort Acknowledge Bit Not Set	Yes	Yes	Yes
eCAN: Unexpected Cessation of Transmit Operation	Yes	Yes	Yes
eQEP: Missed First Index Event	Yes	Yes	Yes
eQEP: eQEP Inputs in GPIO Asynchronous Mode	Yes	Yes	Yes
eQEP: Incorrect Operation of EQEP2B Function on GPIO25 Pin (This advisory is applicable for the 100-pin packages only.)	Yes	Yes	Yes
eQEP: Position Counter Incorrectly Reset on Direction Change During Index	Yes	Yes	Yes
Watchdog: Incorrect Operation of CPU Watchdog When WDCLK Source is OSCCLKSRC2	Yes	Yes	Yes
Oscillator: CPU Clock Switching to INTOSC2 May Result in Missing Clock Condition After Reset	Yes	Yes	Yes
DMA: ePWM Interrupt Trigger Source Selection via PERINTSEL is Incorrect	Yes	Yes	Yes
CLA: Memory and Clock Configuration (MMEMCFG) Register Bits 8, 9, and 10 are Write-Only	Yes	Yes	Yes
ePWM: SWFSYNC Does Not Properly Propagate to Subsequent ePWM Modules or Output on EPWMSYNCO Pin	Yes	Yes	Yes
ePWM: An ePWM Glitch can Occur if a Trip Remains Active at the End of the Blanking Window	Yes	Yes	Yes
VCU: First CRC Calculation May Not be Correct	Yes	Yes	Yes
VCU: Overflow Flags Not Set Properly	Yes		
USB: USB DMA Event Triggers Cause Too Many DMA Transfers	Yes	Yes	Yes
USB: Host Mode — Cannot Communicate With Low-Speed Device Through a Hub	Yes	Yes	Yes
USB: End-of-Packet Symbol Not Generated	Yes		
Boot ROM: Boot ROM GetMode( ) Boot Option Selection	Yes		

NOTE: Revision B silicon was released with an updated read-only-memory (ROM) section to support the InstaSPIN-FOC™-enabled versions of F2806x (F28062F, 68F, 69F) and the InstaSPIN-MOTION™-enabled versions of F2806x (F28068M, 69M). Besides the updated ROM, Revision B silicon is functionally equivalent to Revision A silicon. All standard (non-InstaSPIN-enabled) F2806x devices ship as Revision A.

Advisory — *FPU: CPU-to-FPU Register Move Operation Followed By F32TOUI32, FRACF32, or UI16TOF32 Operations*

Revision(s) Affected 0, A, B

Details This advisory applies when the write phase of a CPU-to-FPU register write coincides with the execution phase of the F32TOUI32, FRACF32, or UI16TOF32 instructions. If the F32TOUI32 instruction execution and CPU-to-FPU register write operation occur in the same cycle, the target register (of the CPU-to-FPU register write operation) gets overwritten with the output of the F32TOUI32 instruction instead of the data present on the C28x data write bus. This scenario also applies to the following instructions:

- F32TOUI32 RaH, RbH
- FRACF32 RaH , RbH
- UI16TOF32 RaH , mem16
- UI16TOF32 RaH , RbH

Workaround(s) A CPU-to-FPU register write must be followed by a gap of five NOPs or non-conflicting instructions before F32TOUI32, FRACF32, or UI16TOF32 can be used.

The C28x code generation tools v6.0.5 (for the 6.0.x branch), v6.1.2 (for the 6.1.x branch), and later check for this scenario.

Example of Problem:

```
   SUBF32 R5H, R3H, R1H
|| MOV32 *--XAR4, R4H
   EISQRTF32 R4H, R2H
   UI16TOF32 R2H, R3H
   MOV32 R0H, @XAR0   ; Write to R0H register
   NOP                ;
   NOP                ;
   F32TOUI32 R1H, R1H ; R1H gets written to R0H
   I16TOF32 R6H, R3H
```

Example of Workaround:

```
   SUBF32 R5H, R3H, R1H
|| MOV32 *--XAR4, R4H
   EISQRTF32 R4H, R2H
   UI16TOF32 R2H, R3H
   MOV32 R0H, @XAR0     ; Write to R0H register
   NOP
   NOP
   NOP
   NOP
   NOP
   F32TOUI32 R1H, R1H
   I16TOF32 R6H, R3H
```

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

Revision(s) Affected 0, A, B

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

<table>
<tr><th></th><th>Instruction</th><th>F1</th><th>F2</th><th>D1</th><th>D2</th><th>R1</th><th>R2</th><th>E</th><th>W</th><th></th><th rowspan="2">Comments</th></tr>
<tr><th></th><th></th><th colspan="4">FPU pipeline---></th><th>R1</th><th>R2</th><th>E1</th><th>E2</th><th>E3</th></tr>
<tr><td>I1</td><td>MPYF32 R6H, R5H, R0H
|| MOV32 *XAR7++, R4H</td><td>I1</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>I2</td><td>F32TOUI16R R3H, R4H</td><td>I2</td><td>I1</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>I3</td><td>ADDF32 R3H, R2H, R0H
|| MOV32 *--SP, R2H</td><td>I3</td><td>I2</td><td>I1</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>I4</td><td>MOV32 @XAR3, R6H</td><td>I4</td><td>I3</td><td>I2</td><td>I1</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td>I4</td><td>I3</td><td>I2</td><td>I1</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td>I4</td><td>I3</td><td>I2</td><td>I1</td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td>I4</td><td>I3</td><td>I2</td><td>I1</td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td>I4</td><td>I3</td><td>I2</td><td>I1</td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td><u>I4</u></td><td>I3</td><td>I2</td><td><u>I1</u></td><td>I4 samples the result as it enters the R2 phase. The product R6H=R5H*R0H (I1) finishes computing in the E3 phase, but is <u>forwarded</u> as an operand to I4. This makes I4 appear to be a 2p instruction, but I4 actually takes 3p cycles to compute.</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>I4</td><td>I3</td><td>I2</td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>I4</td><td>I3</td><td></td></tr>
</table>

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

Figure 4 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 4. Pipeline Diagram of the Issue if There is a Stall in the E3 Slot of the Instruction I1

Workaround(s)

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 5 shows the pipeline diagram with the workaround in place.

	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 5. Pipeline Diagram With Workaround in Place

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

Revision(s) Affected 0, A, B

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 — *ADC: Initial Conversion*

Revision(s) Affected 0, A, B

Details When the ADC conversions are initiated by any source of trigger in either sequential or simultaneous sampling mode, the first sample may not be the correct conversion result.

Workaround(s) For sequential mode, discard the first sample at the beginning of every series of conversions. For instance, if the application calls for a given series of conversions, SOC0→SOC1→SOC2, to initiate periodically, then set up the series instead as SOC0→SOC1→SOC2→SOC3 and only use the last three conversions, ADCRESULT1, ADCRESULT2, ADCRESULT3, thereby discarding ADCRESULT0.

For simultaneous sample mode, discard the first sample of both the A and B channels at the beginning of every series of conversions.

User application should validate if this workaround is acceptable in their application.

The magnitude of error is significantly reduced by writing a 1 to the ADCNONOVERLAP bit in the ADCCTRL2 register, which only allows the sampling of ADC channels when the ADC is finished with any pending conversion. Typically, the difference between the first sample and subsequent samples, with ADCNONOVERLAP enabled, will be less than or equal to four LSBs.

Advisory — *ADC: Temperature Sensor Minimum Sample Window Requirement*

Revision(s) Affected 0, A, B

Details If the minimum sample window is used (6 ADC clocks at 45 MHz, 155.56 ns), the result of a temperature sensor conversion can have a large error, making it unreliable for the system.

Workaround(s)

1. If double-sampling of the temperature sensor is used to avoid the corrupted first sample issue, the temperature sensor result is valid. Double-sampling is equivalent to giving the sample-and-hold circuit adequate time to charge.
2. In all other conditions, the sample-and-hold window used to sample the temperature sensor should not be less than 550 ns.

Advisory — *ADC: ADC Result Conversion When Sampling Ends on 14th Cycle of Previous Conversion, ACQPS = 6 or 7*

Revision(s) Affected 0, A, B

Details The on-chip ADC takes 13 ADC clock cycles to complete a conversion after the sampling phase has ended. The result is then presented to the CPU on the 14th cycle post-sampling and latched on the 15th cycle into the ADC result registers. If the next conversion's sampling phase terminates on this 14th cycle, the results latched by the CPU into the result register are not assured to be valid across all operating conditions.

Workaround(s) Some workarounds are as follows:

- Due to the nature of the sampling and conversion phases of the ADC, there are only two values of ACQPS (which controls the sampling window) that would result in the above condition occurring—ACQPS = 6 or 7. One solution is to avoid using these values in ACQPS.
- When the ADCNONOVERLAP feature (bit 1 in ADCTRL2 register) is used, the above condition will never be met; so the user is free to use any value of ACQPS desired.
- Depending on the frequency of ADC sampling used in the system, the user can determine if their system will hit the above condition if the system requires the use of ACQPS = 6 or 7. For instance, if the converter is continuously converting with ACQPS = 6, the above condition will never be met because the end of the sampling phase will always fall on the 13th cycle of the current conversion in progress.

Advisory — *ADC: Offset Self-Recalibration Requirement*

Revision(s) Affected 0, A, B

Details The factory offset calibration from Device_cal() may not ensure that the ADC offset remains within specifications under all operating conditions in the customer's system.

Workaround(s)

- To ensure that the offset remains within the data sheet's “single recalibration” specifications, perform the AdcOffsetSelfCal() function after Device_cal() has completed and the ADC has been configured.
- To ensure that the offset remains within the data sheet's “periodic recalibration” specifications, perform the AdcOffsetSelfCal() function periodically with respect to temperature drift.

For more details on AdcOffsetSelfCal(), refer to the “ADC Zero Offset Calibration” section in the Analog-to-Digital Converter and Comparator chapter of the *TMS320x2806x Technical Reference Manual*.

Advisory — *ADC: ADC Revision Register (ADCREV) Limitation*

Revision(s) Affected 0, A, B

Details The ADC Revision Register, which is implemented to allow differentiation between ADC revisions and ADC types, will always read "0" for both fields.

Workaround(s) On devices with CLASSID (at address 0x882) of 0x009F, 0x008F, 0x007F, or 0x006F, the “TYPE” field in the ADCREV register should be assumed to be 3 and the “REV” field” should be inferred from the table below.

REVID (0x883)	ADCREV.REV Field
0	2
1	2

Advisory — *ADC: ADC can Become Non-Responsive When ADCNONOVERLAP or RESET is Written During a Conversion*

Revision(s) Affected 0, A, B

Details The ADC can get into a non-responsive state when the ADCCTL2[ADCNONOVERLAP] is modified while a conversion is in progress. When in this condition, no further conversion from the ADC will be possible without a device reset.

There are two different ways to cause this condition:

- Writing to ADCCTL2[ADCNONOVERLAP] while a conversion is in progress.
- Writing to ADCCTL1[RESET] while a conversion is in progress.

Workaround(s) Follow this sequence to modify ADCCTL2[ADCNONOVERLAP] or write ADCCTL1[RESET]:

1. Set all SOC trigger sources ADCSOCxCTL[TRIGSEL] to 0.
2. Set all ADCINTSOCSEL1/2 to 0.
3. Ensure there is not another SOC pending (This can be accomplished by polling SOC Flags).
4. Wait for all conversions to complete.
   a. ADCCTL2[CLKDIV2EN] = 0, ADCCTL2[CLKDIV4EN] = x → (ACQPS + 13) * 1 SYSCLKs
   b. ADCCTL2[CLKDIV2EN] = 1, ADCCTL2[CLKDIV4EN] = 0 → (ACQPS + 13) * 2 SYSCLKs
   c. ADCCTL2[CLKDIV2EN] = 1, ADCCTL2[CLKDIV4EN] = 1 → (ACQPS + 13) * 4 SYSCLKs
5. Modify ADCCTL2[ADCNONOVERLAP] or write ADCCTL1[RESET].

An example code follows.

```
EALLOW;
// Set all SOC trigger sources to software
AdcRegs.ADCSOC0CTL.bit. TRIGSEL  = 0;
AdcRegs.ADCSOC1CTL.bit. TRIGSEL  = 0;
AdcRegs.ADCSOC2CTL.bit. TRIGSEL  = 0;
AdcRegs.ADCSOC3CTL.bit. TRIGSEL  = 0;
AdcRegs.ADCSOC4CTL.bit. TRIGSEL  = 0;
AdcRegs.ADCSOC5CTL.bit. TRIGSEL  = 0;
AdcRegs.ADCSOC6CTL.bit. TRIGSEL  = 0;
AdcRegs.ADCSOC7CTL.bit. TRIGSEL  = 0;
AdcRegs.ADCSOC8CTL.bit. TRIGSEL  = 0;
AdcRegs.ADCSOC9CTL.bit. TRIGSEL  = 0;
AdcRegs.ADCSOC10CTL.bit. TRIGSEL  = 0;
AdcRegs.ADCSOC11CTL.bit. TRIGSEL  = 0;
AdcRegs.ADCSOC12CTL.bit. TRIGSEL  = 0;
AdcRegs.ADCSOC13CTL.bit. TRIGSEL  = 0;
AdcRegs.ADCSOC14CTL.bit. TRIGSEL  = 0;
AdcRegs.ADCSOC15CTL.bit. TRIGSEL  = 0;

// Set all ADCINTSOCSEL1/2 to 0.
    AdcRegs.ADCINTSOCSEL1.bit.SOC0  = 0;
    AdcRegs.ADCINTSOCSEL1.bit.SOC1  = 0;
    AdcRegs.ADCINTSOCSEL1.bit.SOC2  = 0;
    AdcRegs.ADCINTSOCSEL1.bit.SOC3  = 0;
    AdcRegs.ADCINTSOCSEL1.bit.SOC4  = 0;
    AdcRegs.ADCINTSOCSEL1.bit.SOC5  = 0;
    AdcRegs.ADCINTSOCSEL1.bit.SOC6  = 0;
    AdcRegs.ADCINTSOCSEL1.bit.SOC7  = 0;
    AdcRegs.ADCINTSOCSEL2.bit.SOC8  = 0;
    AdcRegs.ADCINTSOCSEL2.bit.SOC9  = 0;
    AdcRegs.ADCINTSOCSEL2.bit.SOC10 = 0;
    AdcRegs.ADCINTSOCSEL2.bit.SOC11 = 0;
    AdcRegs.ADCINTSOCSEL2.bit.SOC12 = 0;
    AdcRegs.ADCINTSOCSEL2.bit.SOC13 = 0;
    AdcRegs.ADCINTSOCSEL2.bit.SOC14 = 0;
    AdcRegs.ADCINTSOCSEL2.bit.SOC15 = 0;

// Ensure there is not another SOC pending
while (AdcRegs.ADCSOCFLG1.all != 0x0);

// Wait for conversions to complete
// Delay time based on ACQPS = 6 , ADCCTL2[CLKDIV2EN] = 1, ADCCTL2[CLKDIV4EN] = 0
// 7 + 13 ADC Clocks = 20 ADCCLKS -> 40 SYSCLKS
asm(" RPT#40||NOP");
// ADCCTL2[ADCNONOVERLAP] = <new value>;
// ADCCTL1[RESET] = 1;

EDIS;
```

Advisory — *Memory: Prefetching Beyond Valid Memory*

Revision(s) Affected 0, A, B

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. This restriction applies to all memory regions and all memory types (flash, OTP, SARAM) on the device. 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.

Advisory — *GPIO: GPIO Qualification*

Revision(s) Affected 0, A, B

Details If a GPIO pin is configured for "n" SYSCLKOUT cycle qualification period (where 1 ≤ n ≤ 510) with "m" qualification samples (m = 3 or 6), it is possible that an input pulse of [n * m – (n – 1)] width may get qualified (instead of n * m). The occurrence of this incorrect behavior depends upon the alignment of the asynchronous GPIO input signal with respect to the phase of the internal prescaled clock, and hence, is not deterministic. The probability of this kind of wrong qualification occurring is "1/n".

Worst-case example:

If n = 510, m = 6, a GPIO input width of (n * m) = 3060 SYSCLKOUT cycles is required to pass qualification. However, because of the issue described in this advisory, the minimum GPIO input width which may get qualified is [n * m – (n – 1)] = 3060 – 509 = 2551 SYSCLKOUT cycles.

Workaround(s) None. Ensure a sufficient margin is in the design for input qualification.

Advisory — *eCAN: Abort Acknowledge Bit Not Set*

Revision(s) Affected 0, A, B

Details

After setting a Transmission Request Reset (TRR) register bit to abort a message, there are some rare instances where the TRRn and TRSn bits will clear without setting the Abort Acknowledge (AAn) bit. The transmission itself is correctly aborted, but no interrupt is asserted and there is no indication of a pending operation.

In order for this rare condition to occur, all of the following conditions must happen:

1. The previous message was not successful, either because of lost arbitration or because no node on the bus was able to acknowledge it or because an error frame resulted from the transmission. The previous message need not be from the same mailbox in which a transmit abort is currently being attempted.
2. The TRRn bit of the mailbox should be set in a CPU cycle immediately following the cycle in which the TRSn bit was set. The TRSn bit remaining set due to incompletion of transmission satisfies this condition as well; that is, the TRSn bit could have been set in the past, but the transmission remains incomplete.
3. The TRRn bit must be set in the exact SYSCLKOUT cycle where the CAN module is in idle state for one cycle. The CAN module is said to be in idle state when it is not in the process of receiving or transmitting data.

If these conditions occur, then the TRRn and TRSn bits for the mailbox will clear t_{clr} SYSCLKOUT cycles after the TRR bit is set where:

$$t_{clr} = [(\text{mailbox\_number}) * 2] + 3 \text{ SYSCLKOUT cycles}$$

The TAn and AAn bits will not be set if this condition occurs. Normally, either the TA or AA bit sets after the TRR bit goes to zero.

Workaround(s)

When this problem occurs, the TRRn and TRSn bits will clear within t_{clr} SYSCLKOUT cycles. To check for this condition, first disable the interrupts. Check the TRRn bit t_{clr} SYSCLKOUT cycles after setting the TRRn bit to make sure it is still set. A set TRRn bit indicates that the problem did not occur.

If the TRRn bit is cleared, it could be because of the normal end of a message and the corresponding TAn or AAn bit is set. Check both the TAn and AAn bits. If either one of the bits is set, then the problem did not occur. If they are both zero, then the problem did occur. Handle the condition like the interrupt service routine would except that the AAn bit does not need clearing now.

If the TAn or AAn bit is set, then the normal interrupt routine will happen when the interrupt is re-enabled.

Advisory — *eCAN: Unexpected Cessation of Transmit Operation*

Revision(s) Affected 0, A, B

Details

In rare instances, the cessation of message transmission from the eCAN module has been observed (while the receive operation continues normally). This anomalous state may occur without any error frames on the bus.

Workaround(s)

The Time-out feature (MOTO) of the eCAN module may be employed to detect this condition. When this occurs, set and clear the CCR bit (using the CCE bit for verification) to remove the anomalous condition.

Advisory — *eQEP: Missed First Index Event*

Revision(s) Affected 0, A, B

Details If the first index event edge at the QEPI input occurs at any time from one system clock cycle before the corresponding QEPA/QEPB edge to two system clock cycles after the corresponding QEPA/QEP edge, then the eQEP module may miss this index event. This condition can result in the following behavior:

- QPOSCNT will not be reset on the first index event if QEPCTL[PCRM] = 00b or 10b (position counter reset on an index event or position counter reset on the first index event).
- The first index event marker flag (QEPSTS[FIMF]) will not be set.

Workaround(s) Reliable operation is achieved by delaying the index signal such that the QEPI event edge occurs at least two system clock cycles after the corresponding QEPA/QEPB signal edge. For cases where the encoder may impart a negative delay (t_d) to the QEPI signal with respect to the corresponding QEPA/QEPB signal (that is, QEPI edge occurs before the corresponding QEPA/QEPB edge), the QEPI signal should be delayed by an amount greater than "t_d + 2*SYSCLKOUT".

Advisory — *eQEP: eQEP Inputs in GPIO Asynchronous Mode*

Revision(s) Affected 0, A, B

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.

Workaround(s) Configure GPIO inputs configured as eQEP pins for non-asynchronous mode (any GPxQSELn register option except “11b = Asynchronous”).

Advisory — *eQEP: Incorrect Operation of EQEP2B Function on GPIO25 Pin (This advisory is applicable for the 100-pin packages only.)*

Revision(s) Affected 0, A, B

Details When the GPIO25 pin is configured for EQEP2B function, activity on the MFSXA pin will be reflected on this pin, regardless of which GPIO pin is configured for MFSXA operation. This issue surfaces only when the GPIO25 pin is configured for EQEP2B operation. This issue does not surface when the GPIO25 pin is configured for GPIO, ECAP2, or SPISOMIB operation.

Workaround(s) Use GPIO55 for EQEP2B operation.

Advisory — *eQEP: Position Counter Incorrectly Reset on Direction Change During Index*

Revision(s) Affected 0, A, B

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.

Workaround(s)

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	***Watchdog: Incorrect Operation of CPU Watchdog When WDCLK Source is OSCCLKSRC2***
Revision(s) Affected	0, A, B
Details	When OSCCLKSRC2 is used as the clock source for CPU watchdog, the watchdog may fail to generate a device reset intermittently.
Workaround(s)	WDCLK should be sourced only from OSCCLKSRC1 (INTOSC1). The CPU may be sourced from OSCCLKSRC2 or OSCCLKSRC1 (INTOSC1).

Advisory — *Oscillator: CPU Clock Switching to INTOSC2 May Result in Missing Clock Condition After Reset*

Revision(s) Affected 0, A, B

Details After at least two system resets (not including power-on reset), when the application code attempts to switch the CPU clock source to internal oscillator 2, a missing clock condition will occur, and the clock switching will fail under the following conditions:

- X1 and X2 are unused (X1 is always tied low when unused).
- GPIO38 (muxed with TCK and XCLKIN) is used as JTAG TCK pin only.
- JTAG emulator is disconnected.

The missing clock condition will recover only after a power-on reset when the failure condition occurs.

Workaround(s) Before switching the CPU clock source to INTOSC2 via the OSCCLKSRCSEL and OSCCLKSRC2SEL bits in the CLKCTL register, the user must toggle the XCLKINOFF and XTALOSCOFF bits in the CLKCTL register as illustrated in the below sequence:

```
CLKCTL |= 0x6000;      // XCLKINOFF = 1, XTALOSCOFF = 1
CLKCTL &=~0x6000;      // XCLKINOFF = 0, XTALOSCOFF = 0
CLKCTL |= 0x6000;      // XCLKINOFF = 1, XTALOSCOFF = 1
CLKCTL &=~0x6000;      // XCLKINOFF = 0, XTALOSCOFF = 0
CLKCTL |= 0x6000;      // XCLKINOFF = 1, XTALOSCOFF = 1
```

Once the above procedure is executed, then the OSC2 selection switches can be configured.

If the JTAG emulator is connected, and GPIO38 (TCK) is toggling, then the above procedure is unnecessary, but will do no harm.

If no clock is applied to GPIO38, TI also recommends that a strong pullup resistor on GPIO38 be added to V_{DDIO}.

Advisory — *DMA: ePWM Interrupt Trigger Source Selection via PERINTSEL is Incorrect*

Revision(s) Affected 0, A, B

Details The MODE.CHx[PERINTSEL] field bit values of 18–29 should select ePWM1SOCA–ePWM6SOCB as DMA trigger sources. Instead, PERINTSEL values of 18–29 select ePWM2SOCA–ePWM7SOCB as DMA trigger sources as shown below in Table 5. ePWM1SOCA and ePWM1SOCB are not implemented as PERINTSEL trigger sources.

Workaround(s) None

Table 5. PERINTSEL Field of Mode Register (MODE)

BIT	FIELD	VALUE	DESCRIPTION		
4-0	PERINTSEL		Peripheral Interrupt Source Select Bits: These bits select which interrupt triggers a DMA burst for the given channel. Only one interrupt source can be selected. A DMA burst can also be forced via the PERINTFRC bit.		
		VALUE	INTERRUPT	SYNC	PERIPHERAL
		0	None	None	No peripheral connection
		1	ADCINT1	None	ADC
		2	ADCINT2	None	
		3	XINT1	None	External Interrupts
		4	XINT2	None	
		5	XINT3	None	
		6	Reserved	None	No peripheral connection
		7	USB0EP1RX	None	USB-0 End Points
		8	USB0EP1TX	None	
		9	USB0EP2RX	None	
		10	USB0EP2TX	None	
		11	TINT0	None	CPU Timers
		12	TINT1	None	
		13	TINT2	None	
		14	MXEVTA	MXSYNCA	McBSP-A
		15	MREVTA	MRSYNCA	
		16	Reserved	None	No peripheral connection
		17	Reserved	None	
		18	ePWM2SOCA	None	ePWM2
		19	ePWM2SOCB	None	
		20	ePWM3SOCA	None	ePWM3
		21	ePWM3SOCB	None	
		22	ePWM4SOCA	None	ePWM4
		23	ePWM4SOCB	None	
		24	ePWM5SOCA	None	ePWM5
		25	ePWM5SOCB	None	
		26	ePWM6SOCA	None	ePWM6
		27	ePWM6SOCB	None	
		28	ePWM7SOCA	None	ePWM7
		29	ePWM7SOCB	None	
		30	USB0EP3RX	None	USB-0 End Points
		31	USB0EP3TX	None	

Advisory — *CLA: Memory and Clock Configuration (MMEMCFG) Register Bits 8, 9, and 10 are Write-Only*

Revision(s) Affected 0, A, B

Details CPU reads of bits 8, 9, and 10 of the MMEMCFG register in the CLA module will always return a zero. Writes to these bits will work as expected.

Workaround(s) None. To modify the bits of this register, a single write to the entire register with the complete configuration should be performed. Read-Modify-Write should not be used as any Read-Modify-Write operation to the register will read a zero for bits 8, 9, and 10 and can write back a zero to those bits and thus modifying these bits unintentionally. An example is shown below:

```
#define CLA_PROG_ENABLE      0x0001
#define CLARAM0_ENABLE       0x0010
#define CLARAM1_ENABLE       0x0020
#define CLARAM2_ENABLE       0x0040
#define CLA_RAM0CPUE         0x0100
#define CLA_RAM1CPUE         0x0200
#define CLA_RAM1CPUE         0x0400

Cla1Regs.MMEMCFG.all  = CLA_PROG_ENABLE1|
CLARAM0_ENABLE|CLARAM1_ENABLE|CLARAM2_ENABLE|CLA_RAM1CPUE;
```

Advisory — *ePWM: SWFSYNC Does Not Properly Propagate to Subsequent ePWM Modules or Output on EPWMSYNCO Pin*

Revision(s) Affected 0, A, B

Details When generating a software synchronization pulse using the TBCTL[SWFSYNC] bit, the sync signal does not propagate down to subsequent ePWM modules' EPWMSYNCI inputs. In the case of ePWM1, the software sync also does not generate an output on the EPWMSYNCO pin. Propagation of the synchronization signal between ePWM modules and to EPWMSYNCO operates as expected when generating a synchronization pulse via an external signal on the EPWMSYNCI pin.

Workaround(s) If the application needs to generate a software sync:

1. The application code should configure a single GPIO mux setting for EPWMSYNCI operation.
2. The EPWMSYNCI pin should be tied to ground or consistently driven low.
3. After the above, the application can use the TBCTL[SWFSYNC] bit as intended to generate a software synchronization pulse that passes to subsequent ePWM modules and onto the EPWMSYNCO pin.

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

Revision(s) Affected 0, A, B

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

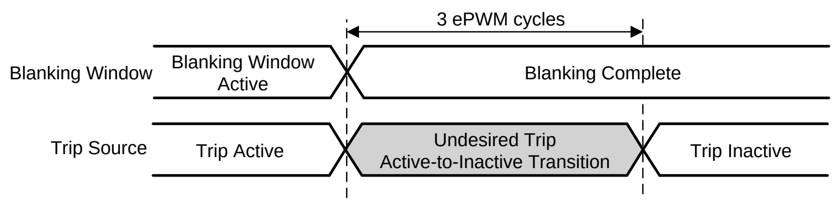


Figure 6. Undesired Trip Event and Blanking Window Expiration

Figure 7 illustrates the two potential ePWM outputs possible if the trip event ends within 1 cycle before or 3 cycles after the blanking window closes.

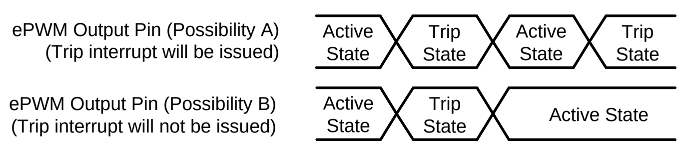


Figure 7. Resulting Undesired ePWM Outputs Possible

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

Advisory — *VCU: First CRC Calculation May Not be Correct*

Revision(s) Affected 0, A, B

Details Due to the internal power-up state of the VCU module, it is possible that the first CRC result will be incorrect. This condition applies to the first result from each of the eight CRC instructions. This ***rare*** condition can only occur after a power-on reset, but will not necessarily occur on every power on. A warm reset will not cause this condition to reappear.

Workaround(s) The application can reset the internal VCU CRC logic by performing a CRC calculation of a single byte in the initialization routine. This routine only needs to perform one CRC calculation and can use any of the CRC instructions. At the end of this routine, clear the VCU CRC result register to discard the result.

An example is shown below.

```
_VCUcrc_reset
      MOVZ          XAR7, #0
      VCRC8L_1      *XAR7
      VCRCCLR
      LRETR
```

Advisory — *VCU: Overflow Flags Not Set Properly*

Revision(s) Affected 0

Details The instructions listed in Table 6 do not set the VSTATUS OVFR and OVFI flags for the expected conditions. For instructions not listed in Table 6, the OVFR and OVFI flags are set as described in the Viterbi, Complex Math and CRC Unit (VCU) chapter of the *TMS320x2806x Technical Reference Manual*.

Table 6. Instructions Affected

INSTRUCTIONS	DESCRIPTION	COMMENTS

`VCADD VR5, VR4, VR3, VR2`
 	
`VCADD VR5, VR4, VR3, VR2`
`|| VMOV32 VRa, mem32`
 	
`VCADD VR7, VR6, VR5, VR4`
	32-bit complex addition	**Expected behavior:** OVFI and OVFR should be set if the final result overflows 32 bits.
		Actual behavior: If the shift-right operation (before the addition) overflows 16 bits, then OVFI or OVR is set.
		If the imaginary-part addition overflows 16 bits, OVFI is set.[(1)]

`VCDADD16 VR5, VR4, VR3, VR2`
 	
`VCDADD16 VR5, VR4, VR3, VR2`
`|| VMOV32 VRa, mem32`
	16 + 32 = 16-bit complex addition	**Expected behavior:** OVFI and OVFR should be set if the final 16-bit result overflows.
		Actual behavior: OVFR and OVFI are only set if the intermediate 32-bit calculation overflows.
		If only the final 16-bit result overflows, then OVFR and OVFI are not set.

`VCDSUB16 VR6, VR4, VR3, VR2`
 	
`VCDSUB16 VR6, VR4, VR3, VR2`
`|| VMOV32 VRa, mem32`
	16 + 32 = 16-bit complex subtraction	

(1) If the real-part addition overflows 16 bits, OVFR is not set. This is the expected behavior.

Workaround Algorithms using these instructions should not rely on the state of the OVFR and OVFI flags to determine if overflow has occurred. Algorithms should use techniques, such as scaling, to avoid overflow. This erratum does not affect the behavior of saturation when performed by these instructions. If saturation is enabled, results that overflow will still be properly saturated.

This issue has been fixed on the Revision A silicon.

Advisory — *USB: USB DMA Event Triggers Cause Too Many DMA Transfers*

Revision(s) Affected 0, A, B

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) Use software DMA triggering instead of USB peripheral requests to start DMA transfers. To start a DMA transfer in software, set the CONTROL.CHx[PERINTFRC] bit.

Advisory — *USB: Host Mode — Cannot Communicate With Low-Speed Device Through a Hub*

Revision(s) Affected 0, A, B

Details When the USB controller is operating as a Host and a low-speed packet is sent to a device through a hub, the subsequent Start-of-Frame is corrupted. After a period of time, this corruption causes the USB controller to lose synchronization with the hub, which results in data corruption.

Workaround(s) None

Advisory — *USB: End-of-Packet Symbol Not Generated*

Revision(s) Affected 0

Details In all USB modes, the USB peripheral is not capable of generating the Single-Ended Zero symbol that signifies the end of a packet. This condition prevents any connected device from properly receiving USB data from the Piccolo device and renders USB inoperable.

Workaround(s) None. This issue has been fixed in the revision A silicon.

Advisory — *Boot ROM: Boot ROM GetMode( ) Boot Option Selection*

Revision(s) Affected 0

Details DevEmuRegs in the Boot ROM is linked to an incorrect memory address, which causes the Boot ROM to read the state of the $\overline{\text{TRST}}$ pin incorrectly. This condition affects the ability of the device to boot into stand-alone/Emulation boot modes properly.

Workaround(s) A workaround function is implemented in the OTP area (reserved for TI) which bypasses this section of the Boot ROM, executes code that correctly reads the state of the $\overline{\text{TRST}}$ pin, and branches back to the Boot ROM to continue booting.

The implemented workaround modifies the operation of the Get-Mode boot option as listed in Table 7.

Table 7. Get-Mode Boot Option Selection

OTP_KEY	OTP_BMODE	EXPECTED BOOT MODE	BOOT MODE SELECTED
!= 0x005A	x	Get Mode: Flash	Get Mode: Flash
0x005A	0x0001	Get Mode: SCI	Get Mode: SCI
	0x0004	Get Mode: SPI	Get Mode: SPI
	0x0005	Get Mode: I2C	Get Mode: I2C
	0x0006	Get Mode: OTP	Get Mode: OTP
	0x0007	Get Mode: CAN	Get Mode: CAN
	0x000B	Get Mode: Flash	Get Mode: Flash
	Other	Stand-alone boot: Get Mode: Flash Emulation boot: Wait Boot Mode	For both stand-alone and Emulation booting: • If bit 7 of Part ID of device == 0, Get Mode – Flash • If bit 7 of Part ID of device == 1, Wait Boot Mode

NOTE: The implemented workaround needs memory locations 0x0002–0x0200 in M0 RAM to be reserved for Boot-ROM usage. Applications can reuse this memory after Boot-ROM execution is completed.

This issue has been fixed on the Revision A silicon.

5 Documentation Support

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

For further information regarding the Piccolo devices, see the following documents:

- *TMS320F2806x Piccolo™ Microcontrollers Data Manual*
- *TMS320x2806x Technical Reference Manual*

Trademarks

Piccolo, PowerPAD, TMS320, InstaSPIN-FOC, InstaSPIN-MOTION are trademarks of Texas Instruments.
All other trademarks are the property of their respective owners.

Revision History

Changes from August 15, 2018 to October 1, 2018 (from M Revision (August 2018) to N Revision)	Page
• Section 4.2 (Known Design Exceptions to Functional Specifications): Added ePWM: An ePWM Glitch can Occur if a Trip Remains Active at the End of the Blanking Window advisory.	29

Changes from April 4, 2018 to August 14, 2018 (from L Revision (April 2018) to M Revision)	Page
• Section 4.1 (Usage Notes): Added Flash: MAX "Program Time" and "Erase Time" in Revision G of the TMS320F2806x Piccolo Microcontrollers Data Manual are only Applicable for Devices Manufactured After January 2018 Usage Note.	6

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