Errata

TMS320F280015x Real-Time MCUs Silicon Errata Silicon Revisions A, 0

ABSTRACT

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

Table of Contents

List of Figures

List of Tables

1 Usage Notes and Advisories Matrices

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

1.1 Usage Notes Matrix

Table 1-1. Usage Notes Matrix

NUMBER	TITLE	SILICON REVISIONS AFFECTED	
		0	A
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

1.2 Advisories Matrix

Table 1-2. Advisories Matrix

MODULE	DESCRIPTION	SILICON REVISIONS AFFECTED	
		0	A
ADC	ADC: Interrupts may Stop if INTxCONT (Continue-to-Interrupt Mode) is not Set	Yes	Yes
ADC	ADC: Degraded ADC Performance With ADCCLK Fractional Divider	Yes	Yes
BOR	BOR: VDDIO Between 2.45 V and 3.0 V can Result in Multiple XRSn Pulses	Yes	Yes
DCAN	During DCAN FIFO Mode, Received Messages May be Placed Out of Order in the FIFO Buffer	Yes	Yes
MCAN	Message Order Inversion When Transmitting From Dedicated Tx Buffers Configured With Same Message ID	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: Trip Events Will Not be Filtered by the Blanking Window for the First 3 Cycles After the Start of a Blanking Window	Yes	Yes
eQEP	eQEP: Position Counter Incorrectly Reset on Direction Change During Index	Yes	Yes
Flash	Flash: Single-Bit ECC Error Interrupt is Not Generated	Yes	Yes
FPU	FPU: FPU-to-CPU Register Move Operation Preceded by Any FPU 2p Operation	Yes	Yes
LIN	LIN: Inconsistent Sync Field Error (ISFE) Flag/Interrupt Not Set When Sync Field is Erroneous	Yes	Yes
Memory	Memory: Prefetching Beyond Valid Memory	Yes	Yes
SYSTEM	SYSTEM: Multiple Successive Writes to CLKSRCCTL1 Can Cause a System Hang	Yes	Yes
Watchdog	Watchdog: WDKEY Register is not EALLOW-Protected	Yes	Yes

2 Nomenclature, Package Symbolization, and Revision Identification

2.1 Device and Development-Support Tool Nomenclature

To designate the stages in the product development cycle, TI assigns prefixes to the part numbers of all DSP devices and support tools. Each DSP commercial family member has one of three prefixes: TMX, TMP, or TMS (for example, **TMS**320F2800157). Texas Instruments recommends two of three possible prefix designators for its support tools: TMDX and TMDS. These prefixes represent evolutionary stages of product development from engineering prototypes (TMX and TMDX) through fully qualified production devices and tools (TMS and TMDS).

Device development evolutionary flow:

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

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

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

Support tool development evolutionary flow:

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

TMDS Fully-qualified development-support product.

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

"Developmental product is intended for internal evaluation purposes."

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

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

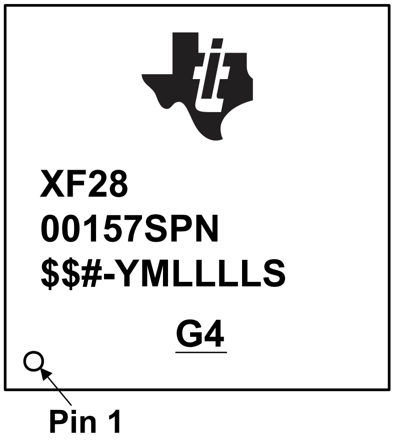
2.2 Devices Supported

This document supports the following devices:

- TMS320F2800157-Q1
- TMS320F2800157
- TMS320F2800156-Q1
- TMS320F2800155-Q1
- TMS320F2800155
- TMS320F2800154-Q1
- TMS320F2800153-Q1
- TMS320F2800152-Q1

2.3 Package Symbolization and Revision Identification

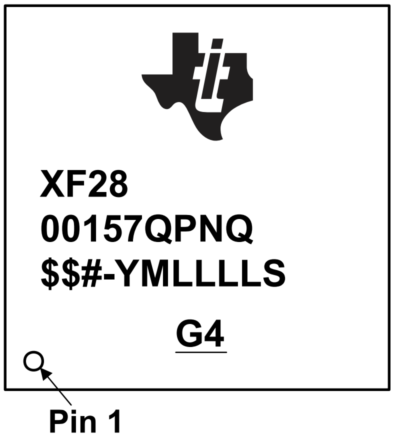
Figure 2-1, Figure 2-2, Figure 2-3, Figure 2-4, and Figure 2-5 show the package symbolization. Table 2-1 lists the silicon revision codes.



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

G4 = ECAT

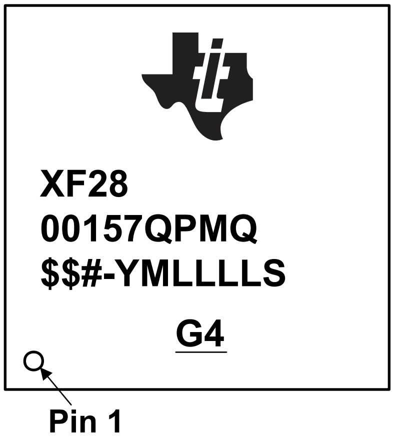
Figure 2-1. Package Symbolization for PN Package



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

G4 = ECAT

Figure 2-2. Package Symbolization for PN Package (AEC-Q100 Grade 1 Qualification)



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

G4 = ECAT

Figure 2-3. Package Symbolization for PM Package (AEC-Q100 Grade 1 Qualification)

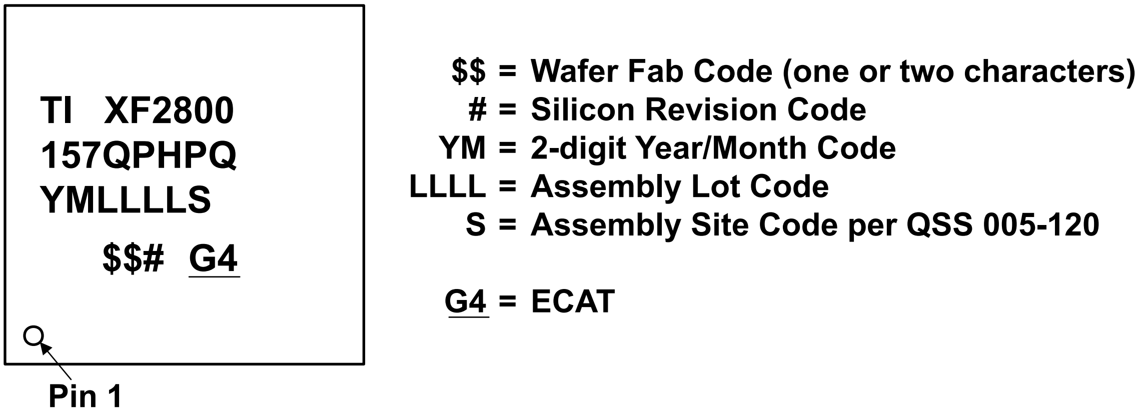

TI XF2800
157QPHPQ
YMLLLLS
$$# G4
Pin 1
$$ = Wafer Fab Code (one or two characters)
# = Silicon Revision Code
YM = 2-digit Year/Month Code
LLLL = Assembly Lot Code
S = Assembly Site Code per QSS 005-120
G4 = ECAT

Figure 2-4. Package Symbolization for PHP Package (AEC-Q100 Grade 1 Qualification)

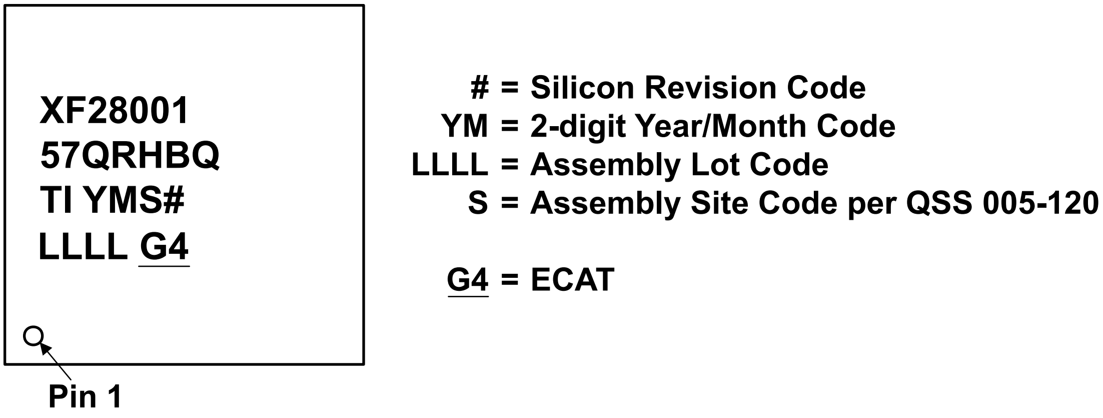

XF28001
57QRHBQ
TI YMS#
LLLL G4
Pin 1
# = Silicon Revision Code
YM = 2-digit Year/Month Code
LLLL = Assembly Lot Code
S = Assembly Site Code per QSS 005-120
G4 = ECAT

Figure 2-5. Package Symbolization for RHB Package (AEC-Q100 Grade 1 Qualification)

Table 2-1. Revision Identification

SILICON REVISION CODE	SILICON REVISION	REVID[1] Address: 0x5D00C	COMMENTS[2]
Blank	0	0x0000 0001	This silicon revision is available as TMX.
A	A	0x0000 0002	This silicon revision is available as TMX.

(1) Silicon Revision ID
(2) For orderable device numbers, see the PACKAGING INFORMATION table in the *TMS320F280015x Real-Time Microcontrollers* data sheet.

3 Silicon Revision A Usage Notes and Advisories

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

3.1 Silicon Revision A Usage Notes

This section lists all the usage notes that are applicable to silicon revision A 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: 0, A

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
```

3.1.2 Caution While Using Nested Interrupts

Revisions Affected: 0, A

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.

3.2 Silicon Revision A Advisories

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

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

Revisions Affected 0, A

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: Degraded ADC Performance With ADCCLK Fractional Divider***

Revisions Affected 0, A

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

Table 3-1. ADCCTL2 Register

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

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

Advisory — *BOR: VDDIO Between 2.45 V and 3.0 V can Result in Multiple XRSn Pulses*

Revisions Affected 0, A

Details

The BOR can generate repeating XRSn assertions and deassertions when the VDDIO supply voltage is between 2.45 V and 3.0 V. It is recommended that the XRSn pin *not* be used directly as a reset to any other devices in the system.

The F280015x BOR is effective for internally holding the device in a known reset state, even when these XRSn pulses are occurring. The device will not branch to application code or bootloaders, and all other pins will be held in their reset state until the VDDIO supply rises above 3.0 V.

Workarounds

1. Ignore the extra XRSn transitions during power up, power down, and BOR events. The extra XRSn pulses will have no effect on the F280015x device operation itself.
2. If XRSn pulses would cause undesired system behavior with other system components, then do not use XRSn to drive other devices. An external voltage supervisor can be used for these applications.
3. For applications that need to avoid these pulses during normal power up and power down:
   a. Power up: Follow the SR_{SUPPLY} requirement in the Recommended Operating Conditions table of the *TMS320F280015x Real-Time Microcontrollers* data sheet; no extra XRSn low pulses will occur.
   b. Power Down: To avoid any deassertion of XRSn during power down, design the power supply so that VDDIO passes through the range from 3.0 V to 2.45 V within 25 µs. If some voltage rise on XRSn is acceptable, then the time constant of the RC circuit implemented on XRSn can be calculated to ensure the voltage does not rise above a system-specified threshold.

Advisory	***During DCAN FIFO Mode, Received Messages May be Placed Out of Order in the FIFO Buffer***
Revisions Affected	0, A
Details	In DCAN FIFO mode, received messages with the same arbitration and mask IDs are supposed to be placed in the FIFO in the order in which they are received. The CPU then retrieves the received messages from the FIFO via the IF1/IF2 interface registers. Some messages may be placed in the FIFO out of the order in which they were received. If the order of the messages is critical to the application for processing, then this behavior will prevent the proper use of the DCAN FIFO mode.
Workaround	None

Advisory ***Message Order Inversion When Transmitting From Dedicated Tx Buffers Configured With Same Message ID***

Revisions Affected 0, A

Details Multiple Tx Buffers are configured with the same Message ID. Transmission of these Tx buffers is requested sequentially in ascending order with a delay between the individual Tx requests. Depending on the delay between the individual Tx requests, the Tx Buffers may not be transmitted in the expected ascending order of the Tx Buffer number.

Workarounds First, write the group of Tx messages with same Message ID to the Message RAM. Then, request transmission of all of these messages concurrently by a single write access to **TXBAR**.

Use the Tx FIFO instead of dedicated Tx Buffers for the transmission of several messages with the same Message ID in a specific order.

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

Revisions Affected 0, A

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 3-1 illustrates the time period which could result in an undesired ePWM output.

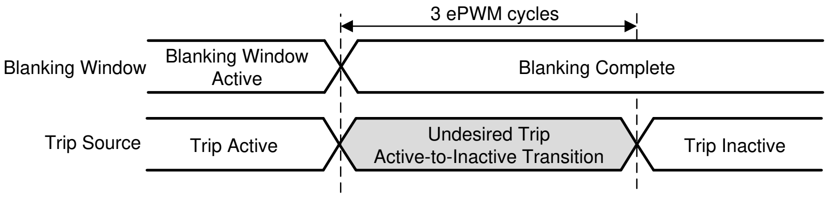


Figure 3-1. Undesired Trip Event and Blanking Window Expiration

Figure 3-2 illustrates the two potential ePWM outputs possible if the trip event ends within 1 cycle before or 3 cycles after the blanking window closes.

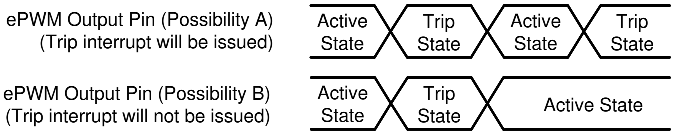

ePWM Output Pin (Possibility A)
(Trip interrupt will be issued)
Active State
Trip State
Active State
Trip State
ePWM Output Pin (Possibility B)
(Trip interrupt will not be issued)
Active State
Trip State
Active State

Figure 3-2. Resulting Undesired ePWM Outputs Possible

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

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 0, A

Details The Blanking Window will not blank trip events for the first 3 cycles after the start of a Blanking Window. DCEVTFILT may continue to reflect changes in the DCxEVTy signals. If DCEVTFILT is enabled, this may impact subsequent subsystems that are configured (for example, the Trip Zone submodule, TZ interrupts, ADC SOC, or the PWM output).

Workaround Start the Blanking Window 3 cycles before blanking is required. If a Blanking Window is needed at a period boundary, start the Blanking Window 3 cycles before the beginning of the next period. This works because Blanking Windows persist across period boundaries.

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

Revisions Affected 0, A

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	***Flash: Single-Bit ECC Error Interrupt is Not Generated***
Revisions Affected	0, A
Details	If the single-bit ECC error threshold is configured as 0, the single-bit error interrupt is not generated when there is a single-bit error.
Workaround	Set the error threshold bit field (FLASH_ECC_REGS ERR_THRESHOLD.ERR_THRESHOLD field) to a value greater than or equal to 1. Note that the default value of the threshold bit field is 0.

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

Revisions Affected 0, A

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.

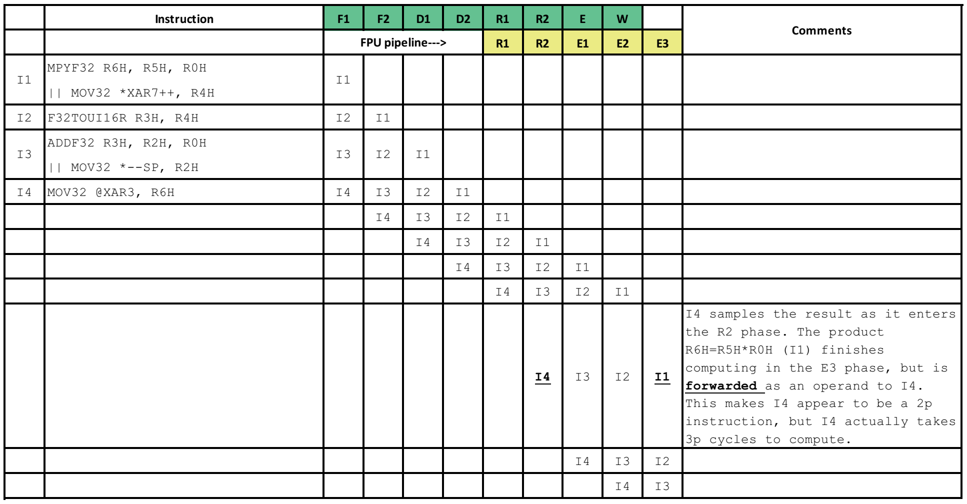
The 2p instructions impacted by this advisory are MPYF32, ADDF32, SUBF32, and MACF32. The destination of the FPU register read must be a CPU register (ACC, P, T, XAR0...XAR7). This advisory does not apply if the register read is a FPU-to-FPU register transfer.

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

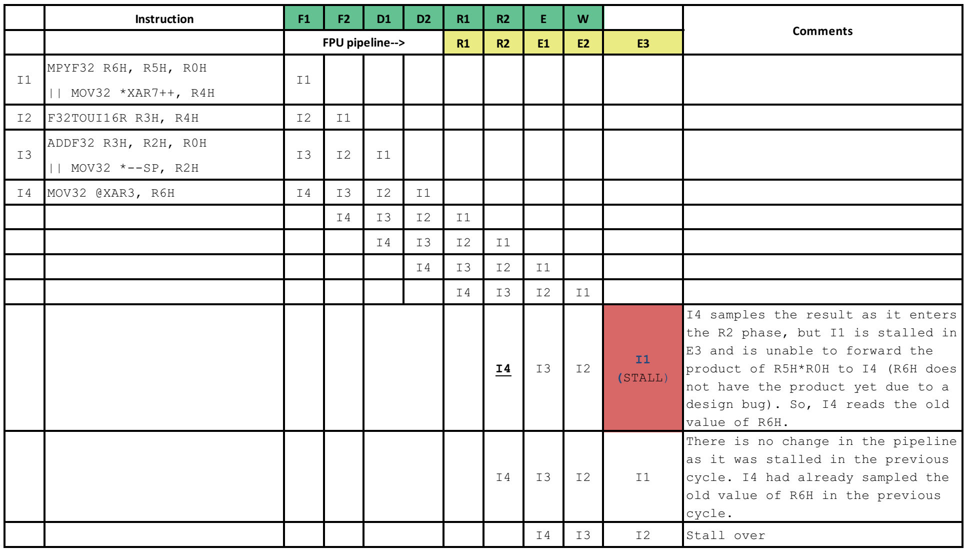


	Instruction	F1	F2	D1	D2	R1	R2	E	W		Comments
		FPU pipeline--->				R1	R2	E1	E2	E3	
I1	MPYF32 R6H, R5H, R0H || MOV32 *XAR7++, R4H	I1									
I2	F32TOUI16R R3H, R4H	I2	I1								
I3	ADDF32 R3H, R2H, R0H || MOV32 *--SP, R2H	I3	I2	I1							
I4	MOV32 @XAR3, R6H	I4	I3	I2	I1						
			I4	I3	I2	I1					
				I4	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 3-3. 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 3-4 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 3-4. Pipeline Diagram of the Issue if There is a Stall in the E3 Slot of the Instruction I1

Workaround

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

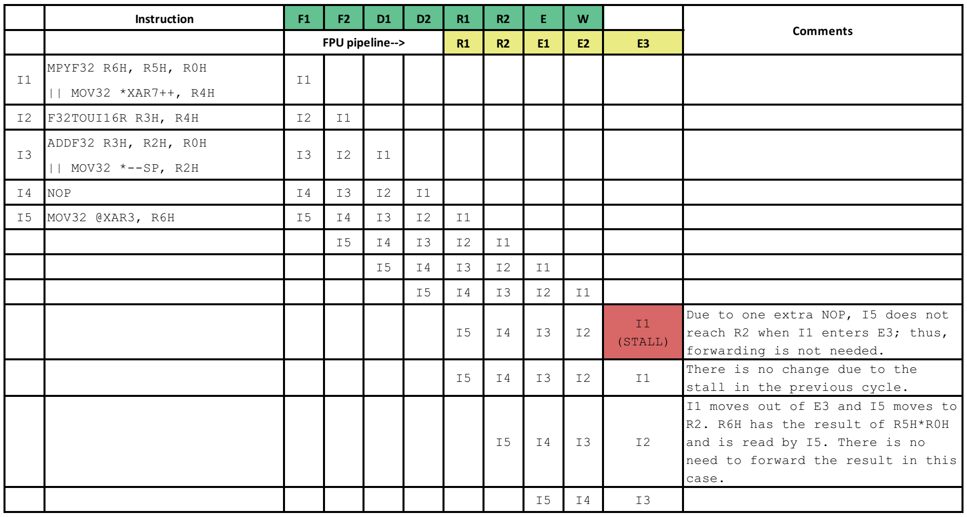
The C28x Code Generation Tools v.6.2.0 and later will both generate the correct instruction sequence and detect the error in assembly code. In previous versions, v6.0.5 (for the 6.0.x branch) and v.6.1.2 (for the 6.1.x branch), the compiler will generate the correct instruction sequence but the assembler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

Example of Workaround:

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

Figure 3-5 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 3-5. Pipeline Diagram With Workaround in Place

Advisory ***LIN: Inconsistent Sync Field Error (ISFE) Flag/Interrupt Not Set When Sync Field is Erroneous***

Revisions Affected 0, A

Details During LIN communications, if the Sync field received (on RX) is erroneous (that is, if the Sync field receives any value other than 0x55), the LIN does not set the ISFE Flag in the SCIFLR.ISFE register or trigger the ISFE interrupt. Communication gets terminated without data being received or the RX receive interrupt being set. There is no way for an application to detect an error in the Sync field. The application can detect if the Sync field is completely blank or if the Sync field is not received within the given tolerances (as explained in the *TMS320F280015x Real-Time Microcontrollers Technical Reference Manual*), but the application cannot detect any error in the value of Sync field.

Workarounds **Method 1:** Keep polling the SCIFLR.RXRDY flag and time out if it is not set within a certain amount of time.

Use the following steps as a guideline:

1. Poll for the SCIFLR.BUSY flag to set.
2. Once the BUSY flag goes high, poll for the SCIFLR.RXRDY flag. Concurrently within this loop, also have a SW timeout, which times out and exits the loop if the RXRDY flag is not set within a user-defined time interval.

Method 2: Configure the CPU timer to interrupt if the RX interrupt is not triggered. This method does not use CPU bandwidth.

Use the following steps as a guideline:

1. Configure XINT to trigger an ISR when the LINRX goes from high to low (indicating LIN is busy).
2. Inside the XINT ISR, configure the CPU timer, which starts timing the frame completion.
3. If the frame is received correctly with the correct Sync field, it should trigger the LIN RX ISR, inside which you can turn off the timer so that you do not get a false timeout.
4. If the frame is not received correctly, it does not trigger the LIN RX ISR but triggers the CPU timer ISR (timeout occurred), which indicates an error in the Sync field.

Advisory ***Memory: Prefetching Beyond Valid Memory***

Revisions Affected 0, A

Details The C28x CPU prefetches instructions beyond those currently active in its pipeline. If the prefetch occurs past the end of valid memory, then the CPU may receive an invalid opcode.

Workaround **M1 –** The prefetch queue is 8 x16 words in depth. Therefore, code should not come within 8 words of the end of valid memory. Prefetching across the boundary between two valid memory blocks is all right.

Example 1: M1 ends at address 0x7FF and is not followed by another memory block. Code in M1 should be stored no farther than address 0x7F7. Addresses 0x7F8–0x7FF should not be used for code.

Example 2: M0 ends at address 0x3FF and valid memory (M1) follows it. Code in M0 can be stored up to and including address 0x3FF. Code can also cross into M1, up to and including address 0x7F7.

Table 3-2. Memories Impacted by Advisory

MEMORY TYPE	ADDRESSES IMPACTED
M1	0x0000 07F8–0x0000 07FF

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

Revisions Affected 0, A

Details When the CLKSRCCTL1 register is written more than once without delay between writes, the system can hang and can only be recovered by an external XRSn reset or Watchdog reset. The occurrence of this condition depends on the clock ratio between SYSCLK and the clock selected by OSCCLKSRCSEL, and may not occur every time.

If this issue is encountered while using the debugger, then after hitting pause, the program counter will be at the Boot ROM reset vector.

Implementing the workaround will avoid this condition for any SYSCLK to OSCCLK ratio.

Workaround Add a software delay of 300 SYSCLK cycles using an NOP instruction after every write to the CLKSRCCTL1 register.

Example:

```
ClkCfgRegs.CLKSRCCTL1.bit.INTOSC2OFF=0;         // Turn on INTOSC2
asm("  RPT #250 || NOP");                       // Delay of 250 SYSCLK Cycles
asm("  RPT #50 || NOP");                        // Delay of 50 SYSCLK Cycles
ClkCfgRegs.CLKSRCCTL1.bit.OSCCLKSRCSEL = 0;     // Clk Src = INTOSC2
asm("  RPT #250 || NOP");                       // Delay of 250 SYSCLK Cycles
asm("  RPT #50 || NOP");                        // Delay of 50 SYSCLK Cycles
```

C2000Ware_3_00_00_00 and later revisions will have this workaround implemented.

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

4 Silicon Revision 0 Usage Notes and Advisories

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

4.1 Silicon Revision 0 Usage Notes

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

4.2 Silicon Revision 0 Advisories

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

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 TMS320F280015x devices, see the following documents:

- *TMS320F280015x Real-Time Microcontrollers* data sheet
- *TMS320F280015x Real-Time Microcontrollers Technical Reference Manual*

6 Trademarks

All trademarks are the property of their respective owners.

7 Revision History

DATE	REVISION	NOTES
January 2023	*	Initial Release

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