Errata

F28E12x Real-Time MCUs Silicon Errata
Silicon Revision 0

ABSTRACT

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

Table of Contents

List of Figures

List of Tables

1 Usage Notes and Advisories Matrices

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

1.1 Usage Notes Matrix

Table 1-1. Usage Notes Matrix

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

1.2 Advisories Matrix

Table 1-2. Advisories Matrix

MODULE	DESCRIPTION	SILICON REVISIONS AFFECTED
		0
ADC	ADC: Interrupts may Stop if INTxCONT (Continue-to-Interrupt Mode) is not Set	Yes
ADC	ADC: Degraded ADC Performance With ADCCLK Fractional Divider	Yes
BOR	BOR: VDDIO Between 2.45 V and 3.0 V can Result in Multiple XRSn Pulses	Yes
CMPSS	CMPSS: COMPxLATCH May Not Clear Properly Under Certain Conditions	Yes
CMPSS	CMPSS: A CMPSS Glitch can Occur if Comparator Input Pin has AGPIO Functionality and ADC is Sampling the Input Pin	Yes
eQEP	eQEP: Position Counter Incorrectly Reset on Direction Change During Index	Yes
Flash	Flash: Single-Bit ECC Error Interrupt is Not Generated	Yes
FPU	FPU: FPU-to-CPU Register Move Operation Preceded by Any FPU 2p Operation	Yes
GPIO	GPIO: Open-Drain Configuration may Drive a Short High Pulse	Yes
MCD	MCD: Missing Clock Detect Should be Disabled When the PLL is Enabled (PLLCLKEN = 1)	Yes
Memory	Memory: Prefetching Beyond Valid Memory	Yes
SYSTEM	SYSTEM: Multiple Successive Writes to CLKSRCCTL1 Can Cause a System Hang	Yes
Watchdog	Watchdog: WDKEY Register is not EALLOW-Protected	Yes
Watchdog	Watchdog: Writes to the WDHALTI Bit Affects PLL Lock Status	Yes

2 Nomenclature, Package Symbolization, and Revision Identification

2.1 Device and Development-Support Tool Nomenclature

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 (TMDX) through fully qualified production tools (TMDS).

Device development evolutionary flow:

X	Experimental device that is not necessarily representative of the final device's electrical specifications and may not use production assembly flow.
P	Prototype device that is not necessarily the final silicon die and may not necessarily meet final electrical specifications.
null	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.

X and P 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.

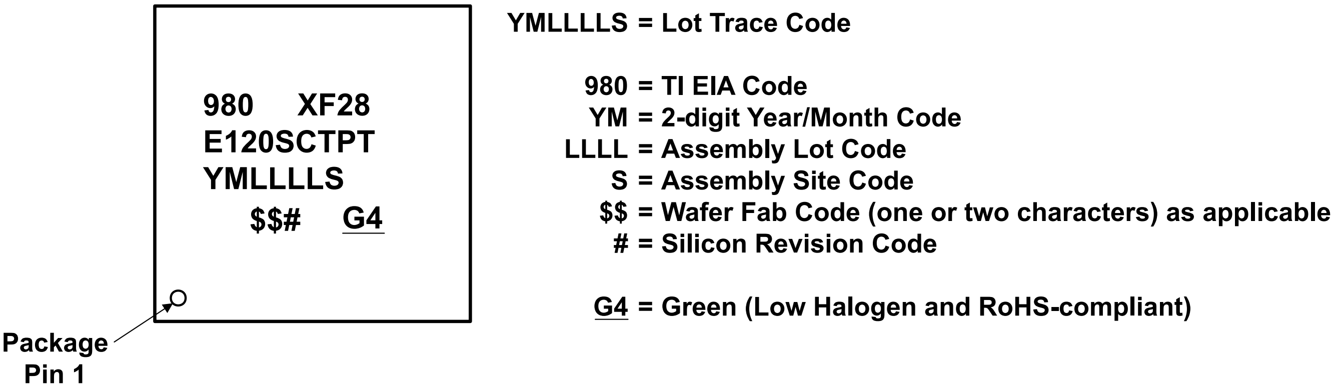
2.2 Devices Supported

This document supports the following devices:

- F28E120SC
- F28E120SB

2.3 Package Symbolization and Revision Identification

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

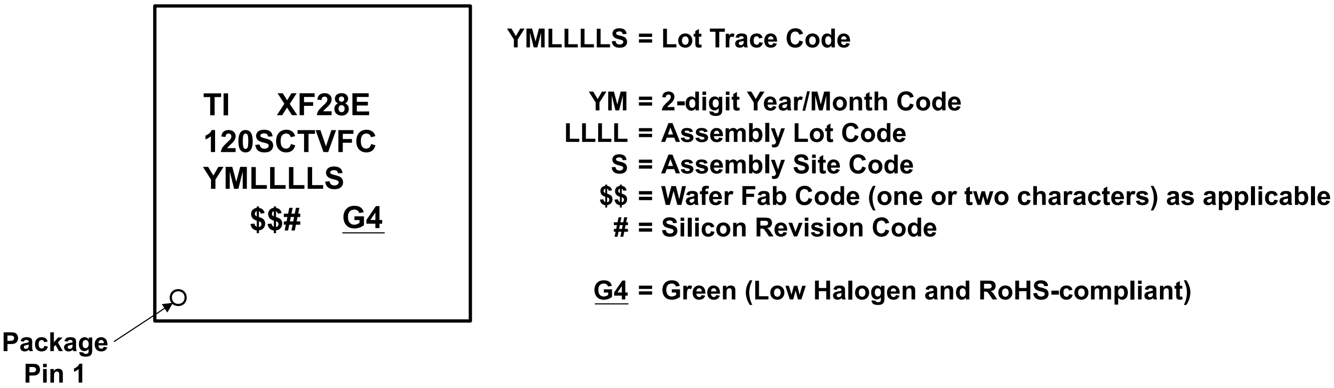


YMLLLLS = Lot Trace Code

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

G4 = Green (Low Halogen and RoHS-compliant)

Figure 2-1. Package Symbolization for PT Package

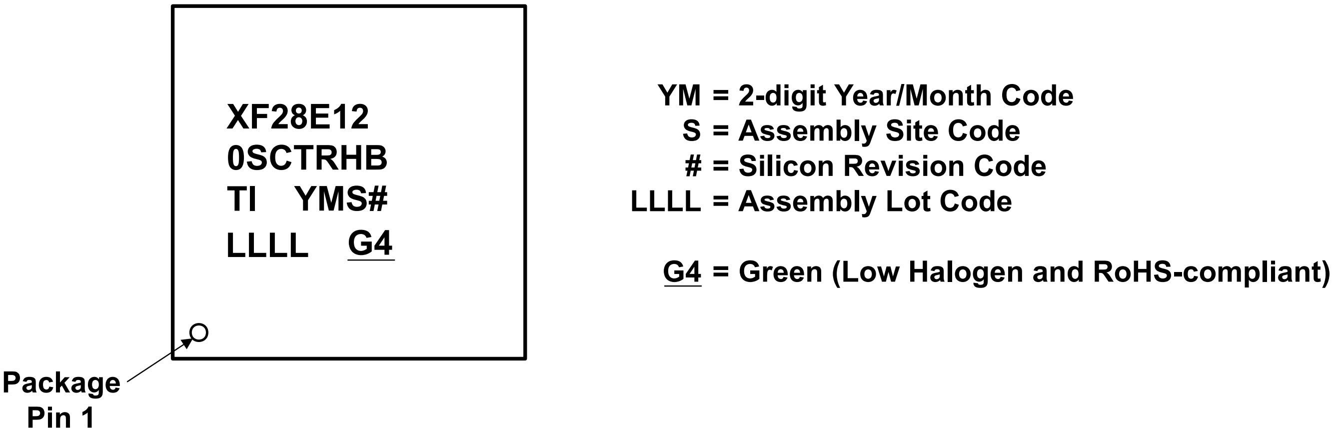


YMLLLLS = Lot Trace Code

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

G4 = Green (Low Halogen and RoHS-compliant)

Figure 2-2. Package Symbolization for VFC Package



YM = 2-digit Year/Month Code
S = Assembly Site Code
# = Silicon Revision Code
LLLL = Assembly Lot Code

G4 = Green (Low Halogen and RoHS-compliant)

Figure 2-3. Package Symbolization for RHB Package

Table 2-1. Revision Identification

SILICON REVISION CODE	SILICON REVISION	REVID(1) Address: 0x5D006	COMMENTS(2)
Blank	0	0x0000 0001	This silicon revision is available as TMX.

(1) Silicon Revision ID
(2) For orderable device numbers, see the PACKAGING INFORMATION table in the *F28E12x Real-Time Microcontrollers* data sheet.

3 Silicon Revision 0 Usage Notes and Advisories

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

3.1 Silicon Revision 0 Usage Notes

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

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

Revisions Affected: 0

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

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

Whether the unwanted interrupt is triggered depends on the configuration and timing of the other interrupts in the system. This is expected to be a rare or nonexistent event in most applications. If it happens, the unwanted interrupt will be the first one in the nested interrupt's PIE group, and will be triggered after the nested interrupt 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 With Repeat Block

Revisions Affected: 0

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

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

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

Note

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

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

Revisions Affected: 0

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

- **JTAGLOCK**

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

- **Zero-pin Boot to Flash**

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

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

3.2 Silicon Revision 0 Advisories

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

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

Revisions Affected

0

Details

If ADCINTSELxNx[INTxCONT] = 0, then interrupts will stop when the ADCINTFLG is set and no additional ADC interrupts will occur.

When an ADC interrupt occurs simultaneously with a software write of the ADCINTFLGCLR register, the ADCINTFLG will unexpectedly remain set, blocking future ADC interrupts.

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

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

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 F28E12x 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 F28E12x 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 *F28E12x 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 — *CMPSS: COMPxLATCH May Not Clear Properly Under Certain Conditions*

Revisions Affected

0

Details

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

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

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

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

Workarounds

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

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

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

Advisory

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

Revisions Affected

0

Details

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

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

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

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

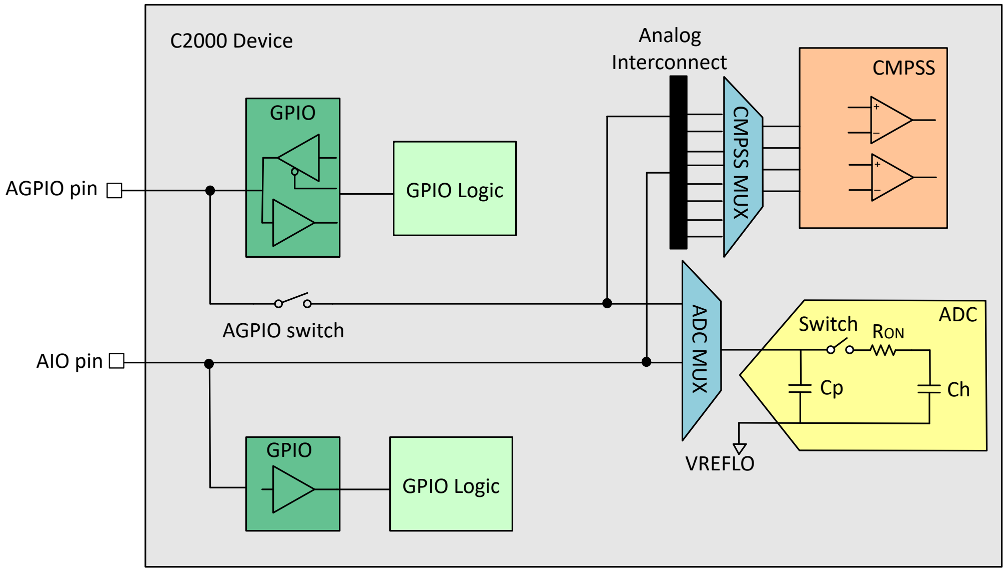


Figure 3-1. Analog Subsystem Diagram with AGPIO and AIO Analog Pin Types

Workarounds

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

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

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

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

Revisions Affected

0

Details

While using the PCRM = 0 configuration, if the direction change occurs when the index input is active, the position counter (QPOSCNT) could be reset erroneously, resulting in an unexpected change in the counter value. This could result in a change of up to ±4 counts from the expected value of the position counter and lead to unexpected subsequent setting of the error flags.

While using the PCRM = 0 configuration [that is, Position Counter Reset on Index Event (QEPCTL[PCRM] = 00)], if the index event occurs during the forward movement, then the position counter is reset to 0 on the next eQEP clock. If the index event occurs during the reverse movement, then the position counter is reset to the value in the QPOSMAX register on the next eQEP clock. The eQEP peripheral records the occurrence of the first index marker (QEPSTS[FIMF]) and direction on the first index event marker (QEPSTS[FIDF]) in QEPSTS registers. It also remembers the quadrature edge on the first index marker so that same relative quadrature transition is used for index event reset operation.

If the direction change occurs while the index pulse is active, the module would still continue to look for the relative quadrature transition for performing the position counter reset. This results in an unexpected change in the position counter value.

The next index event without a simultaneous direction change will reset the counter properly and work as expected.

Workarounds

Do not use the PCRM = 0 configuration if the direction change could occur while the index is active and the resultant change of the position counter value could affect the application.

Other options for performing position counter reset, if appropriate for the application [such as Index Event Initialization (IEI)], do not have this issue.

Advisory	***Flash: Single-Bit ECC Error Interrupt is Not Generated***
Revisions Affected	0
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

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.

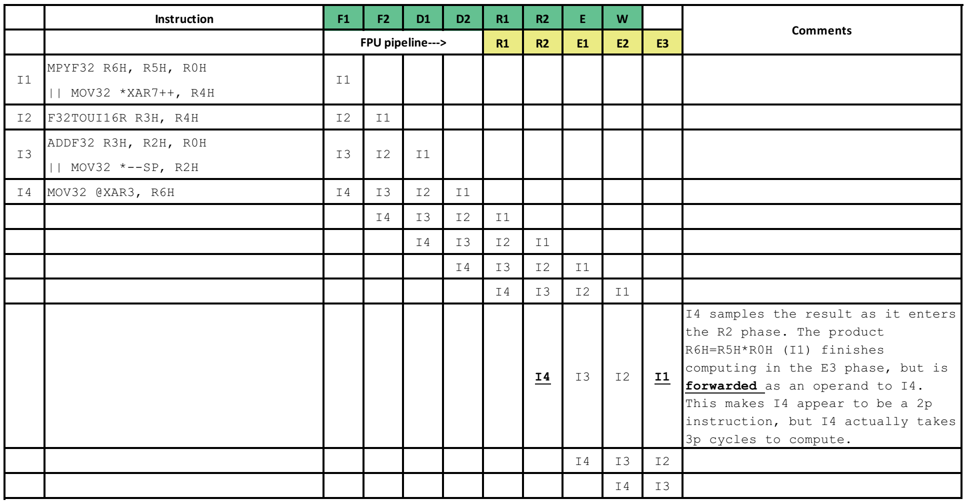
The 2p instructions impacted by this advisory are MPYF32, ADDF32, SUBF32, and MACF32. The destination of the FPU register read must be a CPU register (ACC, P, T, XAR0...XAR7). This advisory does not apply if the register read is a FPU-to-FPU register transfer.

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 3-2 shows the pipeline diagram of the issue when there are no stalls in the pipeline.

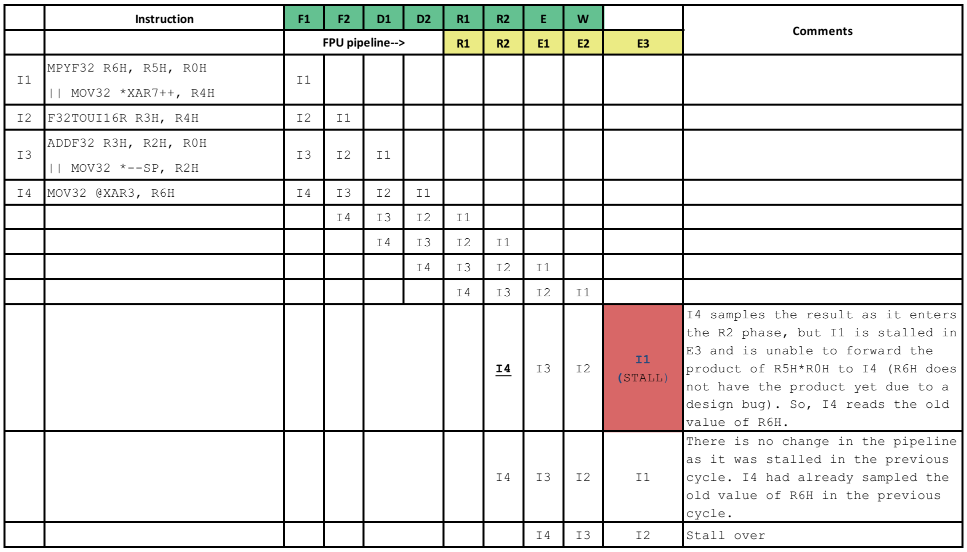


	Instruction	F1	F2	D1	D2	R1	R2	E	W		Comments
		FPU pipeline--->				R1	R2	E1	E2	E3	
I1	MPYF32 R6H, R5H, R0H || MOV32 *XAR7++, R4H	I1									
I2	F32TOUI16R R3H, R4H	I2	I1								
I3	ADDF32 R3H, R2H, R0H || MOV32 *--SP, R2H	I3	I2	I1							
I4	MOV32 @XAR3, R6H	I4	I3	I2	I1						
			I4	I3	I2	I1					
				I4	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 3-2. 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 3-3 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 3-3. 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.

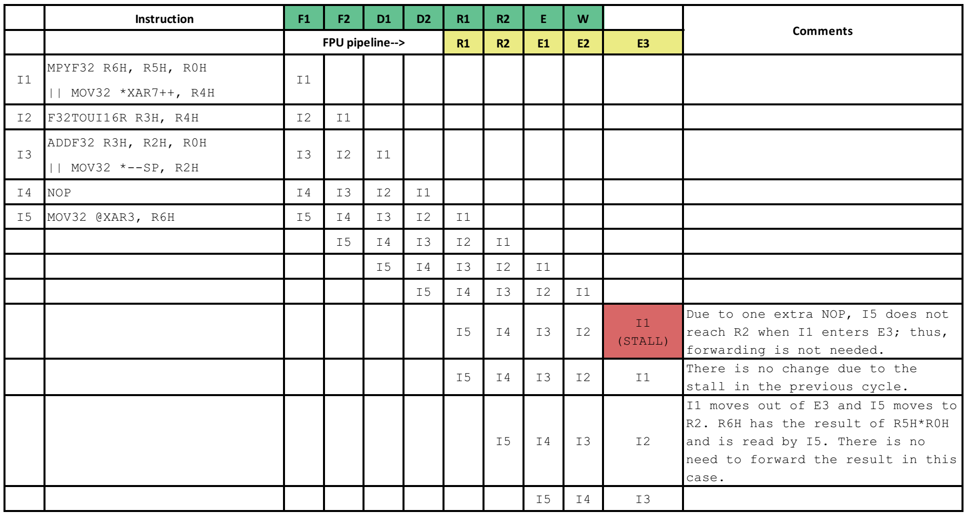
The C28x Code Generation Tools v.6.2.0 and later will both generate the correct instruction sequence and detect the error in assembly code. In previous versions, v6.0.5 (for the 6.0.x branch) and v.6.1.2 (for the 6.1.x branch), the compiler will generate the correct instruction sequence but the assembler 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 3-4 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 3-4. Pipeline Diagram With Workaround in Place

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

Revisions Affected

0

Details

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

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

Workaround

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	***MCD: Missing Clock Detect Should be Disabled When the PLL is Enabled (PLLCLKEN = 1)***
Revisions Affected	0
Details	The PLL has a limp mode feature to provide a slow PLLRAWCLK output even if its input OSCCLK is absent. Independently, the Missing Clock Detect (MCD) circuit will forcibly switch the system clock source to INTOSC1 when a missing OSCCLK input is detected. The MCD mux to switch between these system clock sources is not ensured to be glitch-free when both clock sources (PLLRAWCLK and INTOSC1) are still active. In rare cases, this may lead to unpredictable device behavior during a missing clock failure event.
Workarounds	When the PLL is used by the system (PLLCLKEN = 1), disable the MCD by writing MCDCR.MCLKOFF = 1. The Dual Clock Comparator (DCC) circuit can be configured to quickly detect if the SYSCLK frequency drops outside the desired frequency to its limp mode due to a missing clock event. When the system is operating in PLL bypass mode (PLLCLKEN = 0), the MCD circuit can still be used to detect missing clock events and switch the clock source to INTOSC1.

Advisory	***Memory: Prefetching Beyond Valid Memory***
Revisions Affected	0

Details

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

Workaround

M1 – 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 3-3. 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

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

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

Advisory *Watchdog: Writes to the WDHALTI Bit Affects PLL Lock Status*

Revisions Affected

0

Details

Once the system PLL is locked (SYSPLLSTS.LOCKS = 1), a write to CLKSRCCTL1.WDHALTI results in the PLL unlocking (SYSPLLSTS.LOCKS = 0).

Workaround

Do WDHALTI configurations before locking the PLL.

4 Documentation Support

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

For more information regarding the F28E12x devices, see the following documents:

- *F28E12x Real-Time Microcontrollers* data sheet
- *F28E12x Real-Time Microcontrollers Technical Reference Manual*

5 Trademarks

All trademarks are the property of their respective owners.

6 Revision History

DATE	REVISION	NOTES
July 2025	*	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
Copyright © 2025, Texas Instruments Incorporated