Errata

TMS320F28P65x 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
Section 3.1.3	GPIO: GPIO Data Register is Reset by CPU1 Reset Only	Yes	Yes
Section 3.1.4	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	Yes

1.2 Advisories Matrix

Table 1-2. Advisories Matrix

MODULE	DESCRIPTION	SILICON REVISIONS AFFECTED	
		0	A
ADC	ADC: ADC-C Does Not Meet 16-bit Mode Specifications	Yes	No
ADC	ADC: Interrupts may Stop if INTxCONT (Continue-to-Interrupt Mode) is not Set	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
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
GPIO	GPIO: Open-Drain Configuration may Drive a Short High Pulse	Yes	Yes
I2C	I2C: Target Transmitter Mode, Standard Mode SDA Timings Limitation	Yes	Yes
MCD	MCD: Missing Clock Detect Should be Disabled When the PLL is Enabled (PLLCLKEN = 1)	Yes	Yes
Memory	Memory: Prefetching Beyond Valid Memory	Yes	Yes
MPOST	MPOST: Execution of Memory Power-On Self-Test will not Execute on Some Early Material	Yes	Yes
SDFM	SDFM: Dynamically Changing Threshold Settings (LLT, HLT), Filter Type, or COSR Settings Will Trigger Spurious Comparator Events	Yes	Yes
SDFM	SDFM: Dynamically Changing Data Filter Settings (Such as Filter Type or DOSR) Will Trigger Spurious Data Acknowledge Events	Yes	Yes
SDFM	SDFM: Two Back-to-Back Writes to SDCPARMx Register Bit Fields CEVT1SEL, CEVT2SEL, and HZEN Within Three SD-Modulator Clock Cycles can Corrupt SDFM State Machine, Resulting in Spurious Comparator Events	Yes	Yes
USB	USB: USB DMA Event Triggers are not Supported	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**320F28P659DK-Q1). 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.

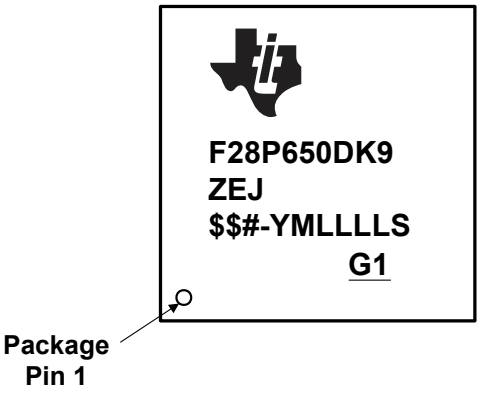
2.2 Devices Supported

This document supports the following devices:

- TMS320F28P650DK
- TMS320F28P650DH
- TMS320F28P650SK
- TMS320F28P650SH
- TMS320F28P659DK-Q1
- TMS320F28P659DH-Q1
- TMS320F28P659SH-Q1

2.3 Package Symbolization and Revision Identification

Figure 2-1, Figure 2-2, Figure 2-3, Figure 2-4, Figure 2-5, Figure 2-6, and Figure 2-7 show the package symbolization. Table 2-1 lists the silicon revision codes.

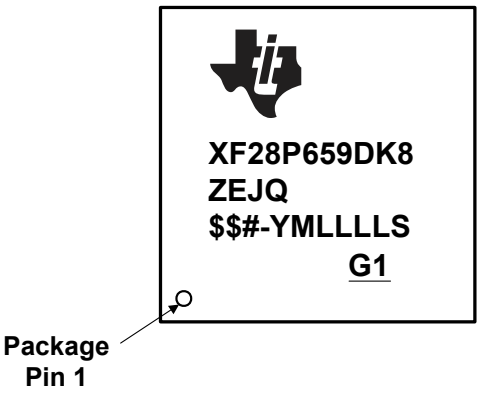


YMLLLLS = Lot Trace Code

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

G1 = ECAT

Figure 2-1. Package Symbolization for ZEJ Package – Non-Automotive

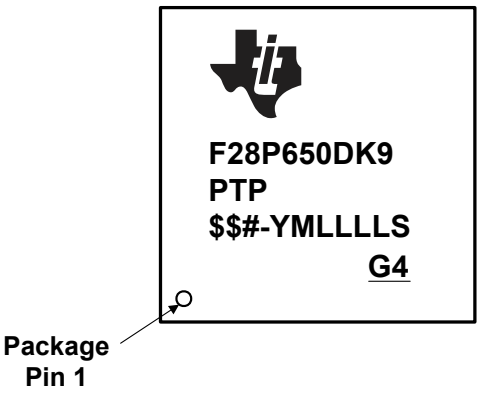


YMLLLLS = Lot Trace Code

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

G1 = ECAT

Figure 2-2. Package Symbolization for ZEJ Package – Automotive

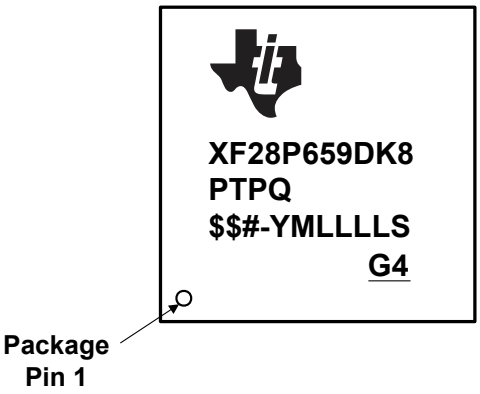


YMLLLLS = Lot Trace Code

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 = ECAT

Figure 2-3. Package Symbolization for PTP Package – Non-Automotive

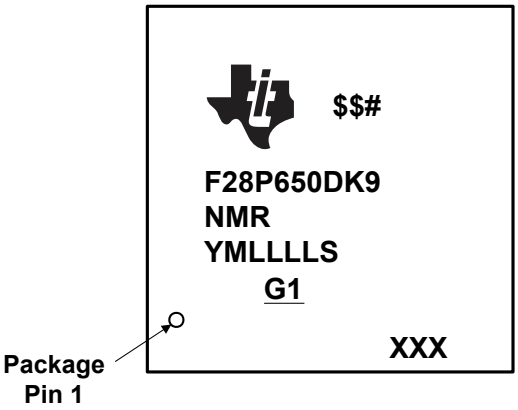


YMLLLLS = Lot Trace Code

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 = ECAT

Figure 2-4. Package Symbolization for PTP Package – Automotive



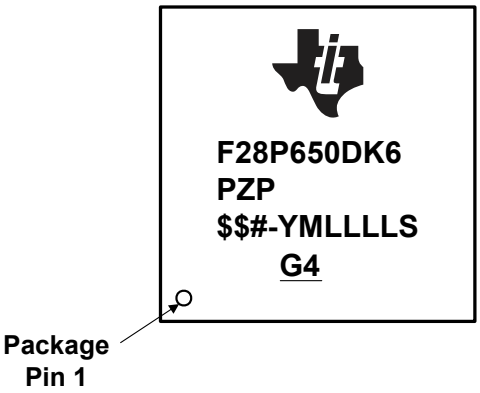
YMLLLLS = Lot Trace Code

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

G1 = ECAT

XXX = 0–119
Serialized Number to indicate unit location on strip

Figure 2-5. Package Symbolization for NMR Package – Non-Automotive



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 = ECAT

Figure 2-6. Package Symbolization for PZP Package – Non-Automotive

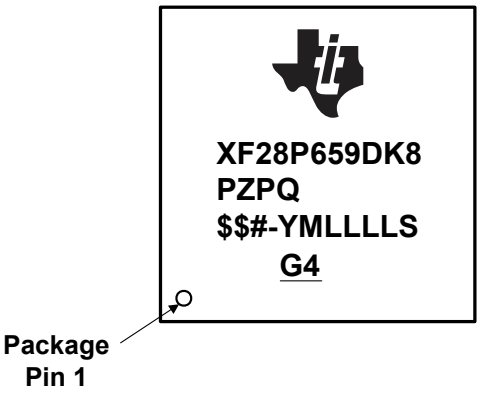


Figure 2-7. Package Symbolization for PZP Package – Automotive

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 TMS.

(1) Silicon Revision ID
(2) For orderable device numbers, see the PACKAGING INFORMATION table in the *TMS320F28P65x 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 reenables CPU interrupts (EINT or asm(" CLRC INTM")).

Workaround: Add a NOP between the PIEACK write and the CPU interrupt enable. Example code is shown below.

```
//Bad interrupt nesting code
PieCtrlRegs.PIEACK.all = 0xFFFF;      //Enable nesting in the PIE
EINT;                                 //Enable nesting in the CPU

//Good interrupt nesting code
PieCtrlRegs.PIEACK.all = 0xFFFF;      //Enable nesting in the PIE
asm(" NOP");                          //Wait for PIEACK to exit the pipeline
EINT;                                 //Enable nesting in the CPU
```

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.

3.1.3 GPIO: GPIO Data Register is Reset by CPU1 Reset Only

Revisions Affected: 0, A

GPIO data registers are reset by CPU1 reset even though a GPIO pin is assigned to CPU2. This causes the GPIO pin to continue to drive the active value even when CPU2 is reset (by a reset source that only resets CPU2).

3.1.4 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, A

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 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	***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	Use the DMA to read out the FIFO via the IF3 register. Each time a message is received into the FIFO, the data is also copied to the IF3 register, and a DMA request Is generated to the DMA module to read out the data.

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.

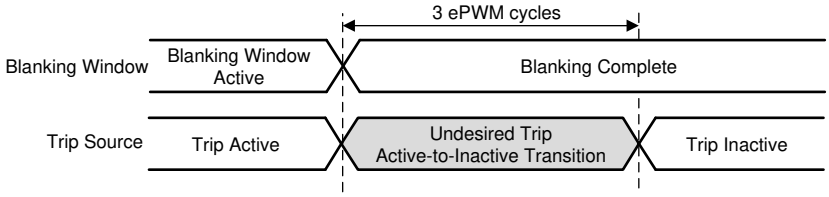


Figure 3-1. Undesired Trip Event and Blanking Window Expiration

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.

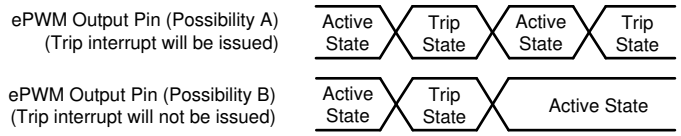


Figure 3-2. Resulting Undesired ePWM Outputs Possible

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

Revisions Affected

0, A

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

Revision Affected 0, A

Details

The I2C peripheral present on the MCU is a Fast-mode device; it will clock-stretch the SCL (Clock) line when used with a Standard-mode host.

There is a requirement from the I2C Specification for a Fast-mode device used in a Standard-mode system to meet $t_{SU:DAT}$ (data set-up time) + $t_{r(max)}$ (rise time) before releasing the SCL line. See Footnote 4 of the "Characteristics of the SDA and SCL bus lines for Standard, Fast, and Fast-mode Plus I^2C-bus devices" table in the NXP Semiconductors *I^2C-bus specification and user manual* (UM10204).

However, the C2000 I2C clock-stretches the SCL line by a fixed amount = 6 * f_{mod} Clock (I2C Clock rate of the C2000) in the above scenario. When the C2000™ microcontroller is acting as a target transmitter with a Standard-mode host, it is possible for the clock line (SCL) to be released by the C2000 before the data (SDA) is ready, if the t_r of SDA is too long.

The "Pull-up resistor sizing" section in the NXP Semiconductors *I^2C-bus specification and user manual* (UM10204) gives more details on choosing the appropriate PU resistor(R_p), based on the rise time (t_r) and bus capacitance (C_b) shown in Equation 1.

$$R_{p(max)} = \frac{t_r}{0.8473 \times C_b} \quad (1)$$

Workaround

1. **Reducing t_r with a strong pullup**
   In order to ensure that $t_{SU:DAT}$ + $t_{r(max)}$ is met, the user can configure the pullup resistance on the SDA line such that it meets the constraints listed in the SDA Data Rise Time Requirement column of Table 3-1 based on the value of f_{mod} Clock in their system. This will ensure that the data present on the SDA line is valid when the C2000 releases the SCL signal.
   Table 3-2 gives suggested R_p resistor values for a given f_{mod} Clock (MHz) and C_b (bus capacitance). For other values of C_b please use Equation 1 to calculate the value of R_p needed in the system.

Table 3-1. Data Rise Time Requirements for C2000 as Target Transmitter with Standard-Mode Host

f_{mod} Clock (MHz)	f_{mod} Period (ns)	SCL Clock-Stretch Delay from C2000 I2C (ns): (6*f_{mod} Clock)	Data Set-up Time (ns): $t_{SU:DAT}$ (Standard Mode)	SDA Data Rise Time Requirement (ns): t_r
7	142.9	857	250	607
8	125	750		500
9	111	666		416
10	100	600		350
11	90.9	545		295
12	83.3	500		250

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

Table 3-2. Pullup Resistor (R_p) Values for Common Bus Capacitances (C_b)

f_{mod} Clock (MHz)	SDA Data Rise Time Requirement (ns): t_r	R_p (kΩ) for C_b = 100 pF	R_p (kΩ) for C_b = 200 pF	R_p (kΩ) for C_b = 300 pF	R_p (kΩ) for C_b = 400 pF
7	607	7.1	3.5	2.3	1.7
8	500	5.9	2.9	1.9	1.4
9	416	4.9	2.4	1.6	1.2
10	350	4.1	2.0	1.3	1.0
11	295	3.4	1.7	1.1	0.8
12	250	2.9	1.4	0.9	0.7

2. **t_r = 1000 ns**
   This workaround is not preferred due to restrictions in general I2C usage, use Workaround 1 when possible.
   If the system is such that it requires 1000 ns of rise time on the SDA line, the C2000 I2C f_{mod} Clock can be configured to 4.8 MHz so the clock-stretching (6 * f_{mod} Clock) satisfies this requirement. This results in t_r = (1/4.8 MHz) * 6 = 1000 ns. This workaround is only valid in systems where the C2000 I2C is the target on the I2C bus. Note that 4.8 MHz is outside the data sheet's required range of 7 MHz to 12 MHz for f_{mod} Clock. Using f_{mod} at 4.8 MHz, even though it is outside of the data sheet's required range, will work for the C2000 I2C in Target mode on a Standard-mode host bus. Using f_{mod} = 4.8 MHz in any other configurations except the one listed in this workaround will cause other timing parameters to be violated and is not allowed.

Advisory	***MCD: Missing Clock Detect Should be Disabled When the PLL is Enabled (PLLCLKEN = 1)***
Revisions Affected	0, A
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, 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, GS15 – The prefetch queue is 8 x16 words in depth. Therefore, code should not come within 8 words of the end of valid memory. Prefetching across the boundary between two valid memory blocks is all right.

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

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

Flash – The prefetch queue is 16 x16 words in depth. Therefore, code should not come within 16 words of the end of valid memory; otherwise, it generates a Flash ECC uncorrectable error.

Table 3-3. Memories Impacted by Advisory

MEMORY TYPE	CORE	ADDRESSES IMPACTED
M1	CPU1, CPU2	0x0000 07F8–0x0000 07FF
GS4	CPU2	0x0001 9FF8–0x0001 9FFF
LS9	CPU1	0x0002 5FF8–0x0002 5FFF
Flash	CPU1, CPU2	0x0011 FFF8–0x0011 FFFF

Advisory ***MPOST: Execution of Memory Power-On Self-Test will not Execute on Some Early Material***

Revisions Affected 0, A

Details

MPOST (Memory Power-On Self-Test) can be used in functional-safety applications to test the device memory on power up. This feature is activated by writing to the Z1_GPREG2.MPOST and Z1_DIAG.MPOST_EN bits using the DCSM Security tool. On impacted material, MPOST will not execute even if the Z1_GPREG2.MPOST and Z1_DIAG.MPOST_EN bits are written to.

Workaround

None. MPOST will not be able to execute. Fixed material will have an OTP revision number greater than 1. The OTP revision number can be determined using Table 3-4.

Table 3-4. OTP Revision Number Location

ADDRESS	8-bit MSB	8-bit LSB
0x0007 2246	0x5A	OTP revision

Advisory	***SDFM: Dynamically Changing Threshold Settings (LLT, HLT), Filter Type, or COSR Settings Will Trigger Spurious Comparator Events***
Revisions Affected	0, A
Details	When SDFM comparator settings—such as filter type, lower/upper threshold, or comparator OSR (COSR) settings—are dynamically changed during run time, spurious comparator events will be triggered. The spurious comparator event will trigger a corresponding CPU interrupt, CLA task, ePWM X-BAR events, and GPIO output X-BAR events if configured appropriately.
Workaround	When comparator settings need to be changed dynamically, follow the procedure below to ensure spurious comparator events do not generate a CPU interrupt, CLA task, or X-BAR events (ePWM X-BAR/GPIO output X-BAR events): 1. Disable the comparator filter. 2. Delay for at least a latency of the comparator filter + 3 SD-Cx clock cycles. 3. Change comparator filter settings such as filter type, COSR, or lower/upper threshold. 4. Delay for at least a latency of the comparator filter + 5 SD-Cx clock cycles. 5. Enable the comparator filter.

Advisory	***SDFM: Dynamically Changing Data Filter Settings (Such as Filter Type or DOSR) Will Trigger Spurious Data Acknowledge Events***
Revisions Affected	0, A
Details	When SDFM data settings—such as filter type or DOSR settings—are dynamically changed during run time, spurious data-filter-ready events will be triggered. The spurious data-ready event will trigger a corresponding CPU interrupt, CLA task, and DMA trigger if configured appropriately.
Workaround	When SDFM data filter settings need to be changed dynamically, follow the procedure below to ensure spurious data-filter-ready events are not generated: 1. Disable the data filter. 2. Delay for at least a latency of the data filter + 3 SD-Cx clock cycles. 3. Change data filter settings such as filter type and DOSR. 4. Delay for at least a latency of the data filter + 5 SD-Cx clock cycles. 5. Enable the data filter.

Advisory	***SDFM: Two Back-to-Back Writes to SDCPARMx Register Bit Fields CEVT1SEL, CEVT2SEL, and HZEN Within Three SD-Modulator Clock Cycles can Corrupt SDFM State Machine, Resulting in Spurious Comparator Events***
Revisions Affected	0, A
Details	Back-to-back writes to SDCPARMx register bit fields CEVT1SEL, CEVT2SEL, and HZEN within three SD-modulator clock cycles can potentially corrupt the SDFM state machine, resulting in spurious comparator events, which can potentially trigger CPU interrupts, CLA tasks, ePWM XBAR events, and GPIO output X-BAR events if configured appropriately.
Workaround	Avoid back-to-back writes within three SD-modulator clock cycles or have the SDCPARMx register bit fields configured in one register write.

Advisory	***USB: USB DMA Event Triggers are not Supported***
Revisions Affected	0, A
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	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.

Advisory	***ADC: ADC-C Does Not Meet 16-bit Mode Specifications***
Revisions Affected	0
Details	ADC-C does *not* meet 16-bit mode specifications; but ADC-C does meet 12-bit specifications. ADC-A and ADC-B meet 12-bit and 16-bit specifications.
Workaround	If the application requires ADC-C to be used in 16-bit mode, performance will be degraded. Use one of the following workarounds: • Operate ADC-C at a lower frequency of 44 MHz to meet specifications. • Use ADC-C in 12-bit mode without having to lower the operating frequency. To satisfy synchronous conversion requirements if more than one ADC is used, see the Ensuring Synchronous Operation section of the *TMS320F28P65x Real-Time Microcontrollers Technical Reference Manual*.

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 TMS320F28P65x devices, see the following documents:

- *TMS320F28P65x Real-Time Microcontrollers* data sheet
- *TMS320F28P65x Real-Time Microcontrollers Technical Reference Manual*

6 Trademarks

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

7 Revision History

Changes from November 15, 2023 to March 19, 2024 (from Revision B (November 2023) to Revision C (March 2024)) **Page**

- Added MPOST: Execution of Memory Power-On Self-Test will not Execute on Some Early Material advisory.....22

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