F28M36x Concerto™ MCUs Silicon Errata Silicon Revisions F, E, B, A, 0

1 Introduction

This document describes the silicon updates to the functional specifications for the F28M36x microcontrollers (MCUs).

The updates are applicable to:

- 289-ball New Fine Pitch Ball Grid Array, ZWT Suffix

2 Device and Development Support Tool Nomenclature

To designate the stages in the product development cycle, TI assigns prefixes to the part numbers of all Concerto MCU devices and support tools. Each Concerto MCU commercial family member has one of three prefixes: x, p, or no prefix (for example, **x**F28M36P63C2ZWTT). Texas Instruments recommends two of three possible prefix designators for its support tools: TMDX and TMDS. These prefixes represent evolutionary stages of product development from engineering prototypes (with prefix **x**/TMDX) through fully qualified production devices/tools (no prefix/TMDS).

xF28M36...	Experimental device that is not necessarily representative of the final device's electrical specifications
pF28M36...	Final silicon die that conforms to the device's electrical specifications but has not completed quality and reliability verification
F28M36...	Fully qualified production device

Support tool development evolutionary flow:

TMDX	Development-support product that has not yet completed Texas Instruments internal qualification testing
TMDS	Fully qualified development-support product

Devices with prefix **x** or **p** and TMDX development-support tools are shipped against the following disclaimer:
"Developmental product is intended for internal evaluation purposes."

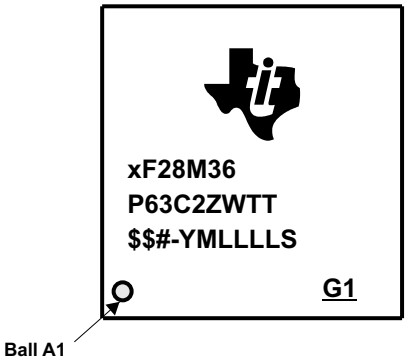
Production devices and TMDS development-support tools have been characterized fully, and the quality and reliability of the device have been demonstrated fully. TI's standard warranty applies.

Predictions show that prototype devices with prefix of **x** or **p** have a greater failure rate than the standard production devices. Texas Instruments recommends that these devices not be used in any production system because their expected end-use failure rate still is undefined. Only qualified production devices are to be used.

TI device nomenclature also includes a suffix with the device family name. This suffix indicates the package type (for example, ZWT) and temperature range (for example, T).

3 Device Markings

Figure 1 provides an example of the Concerto device markings and defines each of the markings. The device revision can be determined by the symbols marked on the top of the package as shown in Figure 1. Some prototype devices may have markings different from those illustrated. Figure 2 shows an example of the device nomenclature.



YMLLLLS = Lot Trace Code

YM = 2-Digit Year/Month Code
LLLL = Assembly Lot
S = Assembly Site Code
$$ = Wafer Fab Code as applicable
= Package Marking Code

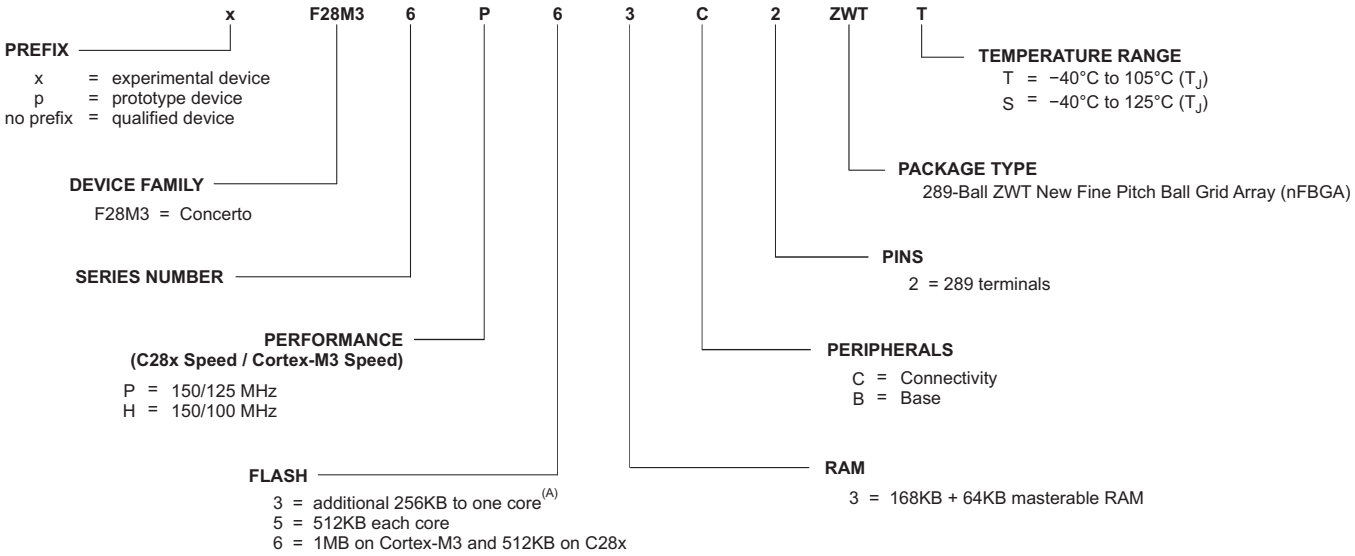
G1 = Green (Low Halogen and RoHS-compliant)

Figure 1. Example of Device Markings

Table 1. Determining Silicon Revision From Lot Trace Code

PACKAGE MARKING CODE	SILICON REVISION	REVISION ID ADDRESS	REVISION ID[1]	COMMENTS
Blank	0	ARM® Cortex®-M3: 0x400F E000 C28x: 0x0883	0x0000	Available as an experimental device.
A	A	ARM Cortex-M3: 0x400F E000 C28x: 0x0883	0x0001	Available as an experimental device and as a qualified production device.
B	B	ARM Cortex-M3: 0x400F E000 C28x: 0x0883	0x0001	Available as an experimental device and as a qualified production device.
E	E	ARM Cortex-M3: 0x400F E000 C28x: 0x0883	0x0005	Available as an experimental device and as a qualified production device.
F	F	ARM Cortex-M3: 0x400F E000 C28x: 0x0883	0x0005	Available as an experimental device and as a qualified production device.

(1) For Cortex-M3, the REVID field (bits 15:0) is embedded in the DID0 register.



A The additional 256KB is added to the Cortex-M3 core (Connectivity Devices) or to the C28x core (Base Devices).

Figure 2. Example of Device Nomenclature

4 Usage Notes and Known Design Exceptions to Functional Specifications

NOTE: For errata relating to the Cortex-M3 r2p0 core, see the *ARM Core Cortex-M3 / Cortex-M3 with ETM (AT420/AT425) Errata Notice* at the ARM Ltd. website.

4.1 Usage Notes

Usage notes highlight and describe particular situations where the device's behavior may not match presumed or documented behavior. This may include behaviors that affect device performance or functional correctness. These usage notes will be incorporated into future documentation updates for the device (such as the device-specific data sheet), and the behaviors they describe will not be altered in future silicon revisions.

Table 2 shows which silicon revision(s) are affected by each usage note.

Table 2. List of Usage Notes

TITLE	SILICON REVISION(S) AFFECTED				
	0	A	B	E	F
PIE: Spurious Nested Interrupt After Back-to-Back PIEACK Write and Manual CPU Interrupt Mask Clear	Yes	Yes	Yes	Yes	Yes
FPU32 and VCU Back-to-Back Memory Accesses	Yes	Yes	Yes	Yes	Yes
Caution While Using Nested Interrupts	Yes	Yes	Yes	Yes	Yes
PBIST: PBIST Memory Test Feature is Deprecated	Yes	Yes	Yes	Yes	Yes
HWBIST: Cortex-M3 HWBIST Feature is Deprecated	Yes	Yes	Yes	Yes	Yes
HWBIST: C28x HWBIST Feature Support is Restricted to TI-Supplied Software	Yes	Yes	Yes	Yes	Yes
Flash Tools: Device Revision Requires a Flash Tools Update	Yes	Yes	Yes	Yes	Yes
EPI: New Feature Addition to EPI Module		Yes	Yes	Yes	Yes
EPI: ALE Signal Polarity		Yes	Yes	Yes	Yes
EPI: $\overline{CS0}$/$\overline{CS1}$ Swap		Yes	Yes	Yes	Yes

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

Revision(s) Affected: 0, A, B, E, F

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

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

Whether the unwanted interrupt is triggered depends on the configuration and timing of the other interrupts in the system. This is expected to be a rare or nonexistent event in most applications. If it happens, the unwanted interrupt will be the first one in the nested interrupt's PIE group, and will be triggered after the nested interrupt re-enables CPU interrupts (EINT or asm(" CLRC INTM")).

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

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

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

4.1.2 FPU32 and VCU Back-to-Back Memory Accesses

Revision(s) Affected: 0, A, B, E, F

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

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

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

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

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

Case 1, Example 1:

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

Case 1, Example 2:

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

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

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

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

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

Case 2, Example 1:

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

Case 2, Example 2:

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

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

Workaround: See Case 1 Workaround and Case 2 Workaround.

4.1.3 Caution While Using Nested Interrupts

Revision(s) Affected: 0, A, B, E, F

If the user is enabling interrupts using the EINT instruction inside an interrupt service routine (ISR) in order to use the nesting feature, then the user must disable the interrupts before exiting the ISR. Failing to do so may cause undefined behavior of CPU execution.

4.1.4 PBIST: PBIST Memory Test Feature is Deprecated

Revision(s) Affected: 0, A, B, E, F

The Programmable Built-in Self-Test (PBIST) memory test feature is no longer supported.

Workaround: Use application memory test software to test the device memory from the CPU.

4.1.5 HWBIST: Cortex-M3 HWBIST Feature is Deprecated

Revision(s) Affected: 0, A, B, E, F

The Cortex-M3 Hardware Built-in Self-Test (HWBIST) feature is no longer supported.

Workaround: Use application test software to test the Cortex-M3 CPU.

4.1.6 HWBIST: C28x HWBIST Feature Support is Restricted to TI-Supplied Software

Revision(s) Affected: 0, A, B, E, F

Unrestricted general configuration and use of C28x HWBIST is not supported.

Workaround: The C28x HWBIST feature should be used only within the context of the C2000™ functional safety collateral. Contact your local TI support for details.

4.1.7 Flash Tools: Device Revision Requires a Flash Tools Update

Revision(s) Affected: 0, A, B, E, F

Flash Tools use the device revision to identify a device for programming. When a new device revision is released, the Flash programming tools (Code Composer Studio™ Integrated Development Environment and UniFlash) must be kept up-to-date with the latest device revision.

4.1.8 EPI: New Feature Addition to EPI Module

Revision(s) Affected: A, B, E, F

In the EPI module, many new features have been added on silicon revisions A and onwards. New configuration registers have been added to enable new features. However, in some cases, new configuration bits (which were “Reserved” on the revision 0 silicon) have been added to existing registers, without breaking the compatibility with the revision 0 silicon. Users should refer to the "External Peripheral Interface (EPI)" chapter in the *Concerto F28M36x Technical Reference Manual* to make sure their existing EPI code (which works on the revision 0 silicon) is not changing the default value of the new configuration bits. Otherwise, the old code may not work on the new silicon.

4.1.9 EPI: ALE Signal Polarity

Revision(s) Affected: A, B, E, F

On the revision 0 silicon, the polarity of the ALE (address latch enable) signal was active HIGH and it was not configurable. On new silicon revisions, a configuration bit (ALEHIGH) has been added in existing host bus configuration registers so that the user can configure the polarity of the ALE signal as per system requirement. Reset value of this bit is set to “1” to have the default polarity of ALE as active HIGH so that it is compatible with the revision 0 silicon (‘0’ will make it active LOW). Since this configuration field was reserved in the revision 0 silicon, if the application writes ‘0’ to this field (while configuring other bit fields in this register), there would be no issue for the revision 0 silicon, but the same code will not work on the revision A silicon. This is because ‘0’ means active LOW polarity for ALE on revision A silicon. This bit needs to be set to ‘1’ to make it work on the revision A silicon.

4.1.10 EPI: $\overline{CS0}$/$\overline{CS1}$ Swap

Revision(s) Affected: A, B, E, F

On revision A silicon onwards, if the following conditions are true:

- both EPADR and ERADR are not 0x0
- the ECADR field is 0x0
- the EPI is configured for dual-chip selects

then,

- $\overline{CS0}$ is asserted for either address range defined by ERADR
- $\overline{CS1}$ is asserted for either address range defined by EPADR

This has been changed from revision 0 silicon, where, in the same configuration,

- $\overline{CS0}$ is asserted for either address range defined by EPADR
- $\overline{CS1}$ is asserted for either address range defined by ERADR.

Table 3. $\overline{CS0}$/$\overline{CS1}$ Swap

SILICON REVISION	CHIP SELECT MODE	ERADR	EPADR	ECADR	$\overline{CS0}$	$\overline{CS1}$
0	Dual-chip select	0x1 or 0x2	0x1 or 0x2	0x0	EPADR defined address range (0xA000.0000 or 0xC000.0000)	ERADR defined address range (0x6000.0000 or 0x8000.0000)
A and onwards	Dual-chip select	0x1 or 0x2	0x1 or 0x2	0x0	ERADR defined address range (0x6000.0000 or 0x8000.0000)	EPADR defined address range (0xA000.0000 or 0xC000.0000)

4.2 Known Design Exceptions to Functional Specifications

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

Table 4. List of Advisories

TITLE	SILICON REVISION(S) AFFECTED				
	0	A	B	E	F
Analog Subsystem: Analog Subsystem Function *InitAnalogSystemClock()* is Incomplete	Yes	Yes	Yes	Yes	Yes
Analog Subsystem: Potential Race Condition after Executing Analog Subsystem Functions *AnalogClockEnable()* or *AnalogClockDisable()*	Yes	Yes	Yes	Yes	Yes
ADC: Initial Conversion	Yes	Yes	Yes	Yes	Yes
ADC: ADC Result Conversion When Sampling Ends on 14th Cycle of Previous Conversion, ACQPS = 6 or 7	Yes	Yes	Yes	Yes	Yes
ADC: Offset Self-Recalibration Requirement	Yes	Yes	Yes	Yes	Yes
ADC: ADC can Become Non-Responsive When ADCNONOVERLAP or RESET is Written During a Conversion	Yes	Yes	Yes	Yes	Yes
VREG: VREG 'Warn Lo/High' Feature Does Not Work as Intended	Yes	Yes	Yes	Yes	Yes
VREG: VREG Will be Enabled During Power Up Irrespective of $\overline{\text{VREG12EN}}$	Yes	Yes	Yes	Yes	Yes
eQEP: eQEP Inputs in GPIO Asynchronous Mode	Yes	Yes	Yes	Yes	Yes
eQEP: Position Counter Incorrectly Reset on Direction Change During Index	Yes	Yes	Yes	Yes	Yes
PLL: May Not Lock on the First Lock Attempt	Yes	Yes	Yes	Yes	Yes
SSI: SSI Microwire Frame Format is Not Supported on This Device	Yes	Yes	Yes	Yes	Yes
UART: RTRIS Bit in the UARTRIS Register is Only Set When the Interrupt is Enabled	Yes	Yes	Yes	Yes	Yes
PBIST: ROM Cannot be Accessed During PBIST Execution	Yes	Yes	Yes	Yes	Yes
Control Subsystem I2C: FIFO Interrupt Trigger Levels Capped at 7	Yes	Yes	Yes	Yes	Yes
ePWM: ePWM7 is Clocked by CPUCLK and Will Stop During IDLE	Yes	Yes	Yes	Yes	Yes
ePWM: ePWM Dead-Band Delay Value Cannot be Set to 0 When Using Shadow Load Mode for RED/FED	Yes	Yes	Yes	Yes	Yes
ePWM: An ePWM Glitch can Occur if a Trip Remains Active at the End of the Blanking Window	Yes	Yes	Yes	Yes	Yes
Missing Clock Detect: Slow Crystal Start-up Can Cause a Missing Clock Detect Without an Interrupt	Yes	Yes	Yes	Yes	Yes
Missing Clock Detect: Watchdog and NMI Watchdog Resets Will be Persistent While X1 Clock is Absent	Yes	Yes	Yes	Yes	Yes
FPU: CPU-to-FPU Register Move Operation Followed By F32TOUI32, FRACF32, or UI16TOF32 Operations	Yes	Yes	Yes	Yes	Yes
FPU: FPU-to-CPU Register Move Operation Preceded by Any FPU 2p Operation	Yes	Yes	Yes	Yes	Yes
Memory: Prefetching Beyond Valid Memory	Yes	Yes	Yes	Yes	Yes
Master Subsystem I2C: SDA and SCL Open-Drain Output Buffer Issue	Yes	Yes	Yes	Yes	Yes
HWBIST: Maximum Frequency	Yes	Yes	Yes	Yes	Yes
HWBIST: Unexpected Behavior When C28x HWBIST is Run With EPI Enabled	Yes	Yes	Yes	Yes	Yes
HWBIST: C28x HWBIST Restriction on Real-Time Window of EPI Module		Yes	Yes	Yes	Yes
HWBIST: C28x HWBIST Restriction on EPI Module		Yes	Yes	Yes	Yes
HWBIST: C28x HWBIST Should Not be Used	Yes	Yes	Yes		
GPIO: GPIO38 and GPIO46 Signal Latch-up to V_{SS} Due to Fast Transient Sensitivity at High Temperature	Yes	Yes	Yes	Yes	Yes
Master Subsystem GPIO: Open-Drain Configuration May Drive a Short High Pulse	Yes	Yes	Yes	Yes	Yes

Table 4. List of Advisories (continued)

TITLE	SILICON REVISION(S) AFFECTED				
	0	A	B	E	F
GPIO: Pins GPIO38 and GPIO46 Cannot be Used for Functions Other Than USB	Yes				
GPIO: Cortex-M3/C28x Reads GPIO State as '0' When the GPIO is Mapped to the Other Core	Yes				
C28x Flash: A Single-Bit ECC Error May Cause Endless Calls to Single-Bit-Error Interrupt Service Routine	Yes	Yes	Yes	Yes	Yes
C28x Flash: The SBF and BF Instructions Will Not Execute From Flash	Yes				
Flash Pump Power Down: Software Sequence Must be Followed to Power Down the Flash Pump				Yes	Yes
Crystal: Maximum Equivalent Series Resistance (ESR) Values are Reduced	Yes	Yes	Yes		
Reset: $\overline{\text{XRS}}$ May Toggle During Power Up	Yes	Yes	Yes		
Cortex-M3 Flash: C28x Reset While C28x Holding Pump Ownership Can Cause Erroneous Cortex-M3 Flash Reads	Yes	Yes	Yes		
USB: Host Mode — Cannot Communicate With Low-Speed Device Through a Hub	Yes				
EPI: C28x Access to the EPI Bus on the Device	Yes				

Table 5. Table of Contents for Advisories

Title **Page**

Table 5. Table of Contents for Advisories (continued)

Advisory

Analog Subsystem: Analog Subsystem Function** InitAnalogSystemClock() **is Incomplete

Revision(s) Affected

0, A, B, E, F

Details

The factory function *InitAnalogSystemClock()* is provided for the purpose of configuring the clock ratio used by the Analog Subsystem and returning its status. A successful, properly configured device will return the success code 0xA005.

For affected devices, *InitAnalogSystemClock()* is missing one step, which may misconfigure the Analog Subsystem clocking scheme and return a code not equal to 0xA005.

A misconfigured Analog Subsystem clocking scheme may lead to random ACIB faults. Under such error conditions, all of the following symptoms are exhibited:

1. Reads from the Analog Subsystem return unexpected values. In the vast majority of cases, the reads return the value of 0x0000.
2. Writes to the Analog Subsystem appear to have no effect.
3. The ACIBERR bit in the NMIFLG registers are set.
4. The CIBSTATUS registers show no error:
   - CIBSTATUS[15:8] increments freely
   - CIBSTATUS[7:0] = 0x01
5. ACIB activity can be recovered by an ACIB reset initiated by the Master Subsystem.

Workaround(s)

Manually complete the clock initialization sequence with an additional step in the user software immediately prior to calling the *InitAnalogSystemClock()* function from the Control Subsystem. This extra step is required for every instance where *InitAnalogSystemClock()* is called by user software.

The additional step will assign the value of 7 to the C28x memory space at address 0x4E58. The following code is an example of how the workaround can be implemented for a device where the desired clock ratio between the Control Subsystem and Analog Subsystem is 4-to-1:

```
*(unsigned int*)0x4E58 = 7;
(**InitAnalogSystemClock)(ACLKDIV4);
```

Once the workaround is in place, *InitAnalogSystemClock()* should be expected to return the success code of 0xA005 and no further actions will be necessary for proper ACIB behavior.

Advisory

Analog Subsystem: Potential Race Condition after Executing Analog Subsystem Functions *AnalogClockEnable()* ***or*** *AnalogClockDisable()*

Revision(s) Affected

0, A, B, E, F

Details

The factory functions *AnalogClockEnable()* and *AnalogClockDisable()* are provided for the purpose of configuring the peripheral clocks of the analog modules. When an analog clock configuration attempt succeeds, the function returns the value of 0.

Due to configuration latencies between the Control Subsystem and Analog Subsystem, the desired clock configuration will not be effective immediately after the function returns. A race condition may be encountered if user software attempts to use a module before the clock configuration is complete.

Further, the factory function *InitAnalogSystemClock()* expects that the analog clock configurations are static during execution. If *AnalogClockEnable()* or *AnalogClockDisable()* is executed immediately before *InitAnalogSystemClock()*, the initialization sequence may be corrupted because the analog clock configuration is not static during execution. For such a scenario, the return value of *InitAnalogSystemClock()* will not be 0xA005 as expected. If the initialization sequence is corrupted, the device may require a reset to recover from the error condition.

The *AnalogClockEnable()* and *AnalogClockDisable()* functions support back-to-back executions so a race condition between calls is not possible when these analog clock configuration functions are executed immediately after each other.

Workaround(s)

After executing one or more calls to *AnalogClockEnable()* or *AnalogClockDisable()*, flush the analog clock configuration sequence by executing the factory function *ReadAnalogClockStatus()*.

The read operation of *ReadAnalogClockStatus()* is automatically queued behind pending analog clock configuration updates so its return from execution can be used to definitively verify that the desired clock configuration sequence has completed.

For example, the following code will enable the ADC module clocks with *AnalogClockEnable()* calls, and it will flush the clock configuration operations with *ReadAnalogClockStatus()* before resetting the ADC modules:

```
while((**AnalogClockEnable)(AnalogConfig1,ADC1_ENABLE));
while((**AnalogClockEnable)(AnalogConfig2,ADC2_ENABLE));

(**ReadAnalogClockStatus)(AnalogConfig2);

Adc1Regs.ADCCTL1.bit.RESET = 1;
Adc2Regs.ADCCTL1.bit.RESET = 1;
```

Advisory — *ADC: Initial Conversion*

Revision(s) Affected

0, A, B, E, F

Details

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

Workaround(s)

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

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

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

This issue is fixed completely by writing a 1 to the ADCNONOVERLAP bit in the ADCTRL2 register, which only allows the sampling of ADC channels when the ADC is finished with any pending conversion.

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

Revision(s) Affected

0, A, B, E, F

Details

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

Workaround(s)

Some workarounds are as follows:

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

Advisory — *ADC: Offset Self-Recalibration Requirement*

Revision(s) Affected 0, A, B, E, F

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

Workaround(s)

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

For more details on AdcxOffsetSelfCal(), refer to the “ADC Zero Offset Calibration” section in the Analog Subsystem chapter of the *Concerto F28M36x Technical Reference Manual*.

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

Revision(s) Affected 0, A, B, E, F

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

There are two different ways to cause this condition:

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

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

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

An example code follows.

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

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

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

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

EDIS;
```

Advisory — *VREG: VREG 'Warn Lo/High' Feature Does Not Work as Intended*

Revision(s) Affected 0, A, B, E, F

Details The VREG "Warn Lo/High" feature should not be used or enabled in the device. Do not set the VREGWARNE bit in the MNMICFG register as it could negatively affect the VREG output voltage.

Workaround(s) None

Advisory — *VREG: VREG Will be Enabled During Power Up Irrespective of $\overline{\text{VREG12EN}}$*

Revision(s) Affected 0, A, B, E, F

Details This advisory only applies to systems that tie the $\overline{\text{VREG12EN}}$ pin high and use an external regulator for the 1.2-V V_{DD} supply.

During power up of the 3.3-V V_{DDIO}, the internal Voltage Regulator (VREG) will be active until the 1.2-V V_{DD} supply reaches approximately 0.7 V. After this time, the state of the $\overline{\text{VREG12EN}}$ pin will either keep the internal VREG active (if low) or disable the internal VREG (if high).

Workaround(s) None

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

Revision(s) Affected 0, A, B, E, F

Details If any of the eQEP input pins are configured for GPIO asynchronous input mode via the GPxQSELn registers, the eQEP module may not operate properly. For example, QPOSCNT may not reset or latch properly, and pulses on the input pins may be missed. This is because the eQEP peripheral assumes the presence of external synchronization to SYSCLKOUT on inputs to the module.

For proper operation of the eQEP module, input GPIO pins should be configured via the GPxQSELn registers for synchronous input mode (with or without qualification). This is the default state of the GPxQSEL registers at reset. All existing eQEP peripheral examples supplied by TI also configure the GPIO inputs for synchronous input mode.

The asynchronous mode should not be used for eQEP module input pins.

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

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

Revision(s) Affected 0, A, B, E, F

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.

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

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

Advisory — *PLL: May Not Lock on the First Lock Attempt*

Revision(s) Affected 0, A, B, E, F

Details The PLL may not start properly at device power up. The PLLSTS[PLLLOCKS] lock counter bit is set, but the PLL does not produce a clock. Once the PLL has properly started, the PLL can be disabled and reenabled with no issues and will stay locked. However, the PLL lock problem could reoccur on a subsequent power-up cycle. If the PLLSYSCLK has not properly started and is selected as the CPU clock source, the CPU will stop executing instructions. The occurrence rate of this transient issue is low and after an initial occurrence, this issue may not be subsequently observed in the system again. Implementation of the workaround reduces the rate of occurrence.

This advisory applies to both PLLs.

Workaround(s)

System PLL Workaround

Repeated lock attempts will reduce the likelihood of seeing the condition on the final attempt. TI recommends a minimum of five lock sequences in succession when the PLL is configured the first time after a power up. A lock sequence means disabling the PLL, starting the PLL locking, and waiting for the PLLLOCKS lock counter bit to set. After the final sequence, the Watchdog is enabled and the clock source is switched to use the PLL output as normal.

The Watchdog timer is used to detect if the condition has occurred because it is not clocked by the PLL output. The Watchdog should be enabled before selecting the PLL as the clock source and configured to reset the device. If the PLL is not producing a clock, the Watchdog will reset the device and the user initialization software will therefore repeat the PLL initialization.

Many applications do not have a different initialization sequence for a Watchdog-initiated reset; for these applications, no further action is required. For applications that do use a different device initialization sequence when a Watchdog reset is detected, a flag can be set in RAM prior to initializing the PLL. This flag can be used to identify a PLL lock failure as the Watchdog reset cause and allow the retry to occur.

See the controlSUITE™ SysCtlClockPllConfig() function for an example implementation of this workaround.

The workaround can also be applied at the System level by a supervisor resetting the device if it is not responding.

USB PLL Workaround

The workaround for the USB PLL is similar to that for the System PLL except the Watchdog cannot be used to detect a PLL lock failure on the final attempt. The application must detect a USB clocking failure at the system level and issue a reset to the device for another USB PLL lock attempt.

See the controlSUITE SysCtlUSBPLLConfigSet() function for an example implementation of this workaround.

Advisory — *SSI: SSI Microwire Frame Format is Not Supported on This Device*

Revision(s) Affected 0, A, B, E, F

Details The Synchronous Serial Interface (SSI) module does not support the Microwire frame format. SSI and SPI frame formats are not affected.

Workaround(s) None

Advisory — *UART: RTRIS Bit in the UARTRIS Register is Only Set When the Interrupt is Enabled*

Revision(s) Affected 0, A, B, E, F

Details The RTRIS (UART Receive Time-Out Raw Interrupt Status) bit in the UART Raw Interrupt Status (UARTRIS) register should be set when a receive time-out occurs, regardless of the state of the RTIM enable bit in the UART Interrupt Mask (UARTIM) register. However, currently the RTIM bit must be set in order for the RTRIS bit to be set when a receive time-out occurs.

Workaround(s) For applications that require polled operation, the RTIM bit can be set while the UART interrupt is disabled in the NVIC using the IntDisable(n) function in the StellarisWare® Peripheral Driver Library, where n is 21, 22, or 49, depending on whether UART0, UART1, or UART2 is used. With this configuration, software can poll the RTRIS bit, but the interrupt is not reported to the NVIC.

Advisory — *PBIST: ROM Cannot be Accessed During PBIST Execution*

Revision(s) Affected 0, A, B, E, F

Details During execution of a Cortex-M3 or C28x PBIST, all Cortex-M3 or C28x accesses to ROM on the device will fail. This does not impact a PBIST of the ROM itself, which will work as expected.

Workaround(s) Do not access ROM during a PBIST execution.

Advisory — *Control Subsystem I2C: FIFO Interrupt Trigger Levels Capped at 7*

Revision(s) Affected 0, A, B, E, F

Details The TXFFIL (Transmit FIFO Interrupt Level) bits in the I2C Transmit FIFO (I2CFFTX) register and the RXFFIL (Receive FIFO Interrupt Level) bits in the I2C Receive FIFO (I2CFFRX) register are capped to a maximum value of 7. Writes of values greater than 7 to these fields will be truncated to the lower 3 bits.

Workaround(s) Applications can poll the TXFFST (Transmit FIFO Status) and RXFFST (Receive FIFO Status) bits in the I2CFFTX and I2CFFRX registers, respectively, for an accurate count of the number of items in the FIFOs. Applications wishing to use I2C FIFO interrupts must set the interrupt levels to 7 or less.

Advisory — *ePWM: ePWM7 is Clocked by CPUCLK and Will Stop During IDLE*

Revision(s) Affected

0, A, B, E, F

Details

The ePWM7 is clocked by C28x CPUCLK. When the CPUCLK stops, ePWM7 will also stop. The C28x (and ePWM7) will stop during:

- C28x low-power IDLE mode
- C28x debugger halt

Other ePWM modules are clocked by SYSCLK, which does not stop during IDLE or debugger halt.

Workaround(s)

None. Use other ePWM modules if the IDLE mode is used or ePWM must remain active during debugger HALT.

Advisory — *ePWM: ePWM Dead-Band Delay Value Cannot be Set to 0 When Using Shadow Load Mode for RED/FED*

Revision(s) Affected

0, A, B, E, F

Details

ePWM dead-band delay value cannot be set to 0 when using Shadow Load Mode for rising-edge delay (RED) and falling-edge delay (FED).

Workaround(s)

1. Use Immediate Load Mode if DBRED/DBFED = 0.
2. Do not use DBRED/DBFED = 0 if in Shadow Load Mode.

This is for both RED and FED.

Advisory

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

Revision(s) Affected

0, A, B, E, F

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

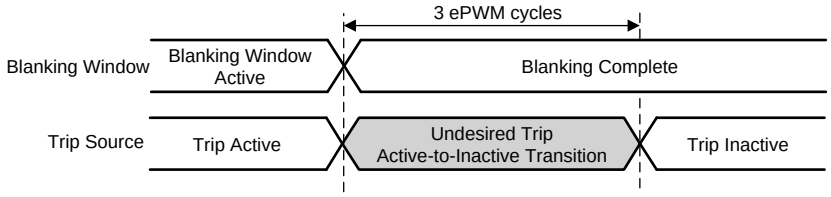


Figure 3. Undesired Trip Event and Blanking Window Expiration

Figure 4 illustrates the two potential ePWM outputs possible if the trip event ends within 1 cycle before or 3 cycles after the blanking window closes.

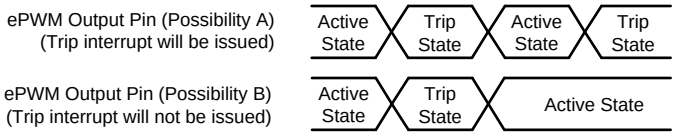


Figure 4. Resulting Undesired ePWM Outputs Possible

Workaround(s)

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

Advisory — *Missing Clock Detect: Slow Crystal Start-up Can Cause a Missing Clock Detect Without an Interrupt*

Revision(s) Affected 0, A, B, E, F

Details If the X1 crystal oscillator is slow to start, a Missing Clock Detect (MCD) event can occur. This will switch the clock source to the internal 10-MHz INTOSC.

The Boot ROM NMI ISR to service the CLOCKFAIL clears the MNMIFLG[CLOCKFAIL] bit to prevent an NMIWD reset. On start-up, the application code will not have an opportunity to execute a custom NMIWD ISR since the interrupt has already been serviced. The MCLKSTS[MCLKFLG] will be set correctly and must be polled by software to detect the MCD condition on start-up.

Workaround(s) Check the MCLKSTS[MCLKFLG] bit in the initialization code. The application should respond according to the system requirements of a missing clock source situation.

Most applications can tolerate a slow crystal start-up. For these applications, a Watchdog reset can be forced. A Watchdog reset will be held for 512 clocks on the X1 clock input and the device will be held in reset until the crystal is active.

Advisory — *Missing Clock Detect: Watchdog and NMI Watchdog Resets Will be Persistent While X1 Clock is Absent*

Revision(s) Affected 0, A, B, E, F

Details The Watchdog and NMI Watchdog resets will always be asserted for 512 cycles of the external X1 clock source. If the X1 clock source is permanently absent, this will result in a persistent assertion of $\overline{\text{XRS}}$ low for a Missing Clock Detect (MCD) event.

NOTE: The term "Watchdog" in this advisory applies to all Cortex-M3 watchdog timers: MNMIWD, WDT0, and WDT1.

Workaround(s) The application code should contain an NMIWD ISR, which properly shuts down the system when an MCD event occurs. **NOTE:** MCD switches the clock source to the 10-MHz INTOSC and code will be executed using this clock.

The NMIWD CLOCKFAIL ISR should:

- Service the Watchdog to prevent a device watchdog reset before system shutdown code is executed.
- Perform system shutdown.
- Optionally use the Watchdog to force a reset if the desired state is for the device to be held in reset due to an MCD.

NOTE: If the application response to an MCD must avoid putting the device in reset, then there is a window of vulnerability between the MCD and a Watchdog expiration. If a watchdog timer configured to cause a reset expires before the NMIWD ISR services the watchdog, a persistent reset cannot be avoided.

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

Revision(s) Affected 0, A, B, E, F

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

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

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

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

Example of Problem:

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

Example of Workaround:

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

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

Revision(s) Affected 0, A, B, E, F

Details

This advisory applies when a multi-cycle (2p) FPU instruction is followed by a FPU-to-CPU register transfer. If the FPU-to-CPU read instruction source register is the same as the 2p instruction destination, then the read may be of the value of the FPU register before the 2p instruction completes. This occurs because the 2p instructions rely on data-forwarding of the result during the E3 phase of the pipeline. If a pipeline stall happens to occur in the E3 phase, the result does not get forwarded in time for the read instruction.

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

In the example below, the 2p instruction, MPYF32, uses R6H as its destination. The FPU register read, MOV32, uses the same register, R6H, as its source, and a CPU register as the destination. If a stall occurs in the E3 pipeline phase, then MOV32 will read the value of R6H before the MPYF32 instruction completes.

Example of Problem:

```
   MPYF32 R6H, R5H, R0H  ; 2p FPU instruction that writes to R6H
|| MOV32 *XAR7++, R4H
   F32TOUI16R R3H, R4H   ; delay slot
   ADDF32 R2H, R2H, R0H
|| MOV32 *--SP, R2H      ; alignment cycle
   MOV32 @XAR3, R6H      ; FPU register read of R6H
```

Figure 5 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 5. Pipeline Diagram of the Issue When There are no Stalls in the Pipeline

Figure 6 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 6. Pipeline Diagram of the Issue if There is a Stall in the E3 Slot of the Instruction I1

Workaround(s)

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

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

Example of Workaround:

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

Figure 7 shows the pipeline diagram with the workaround in place.

	Instruction	F1	F2	D1	D2	R1	R2	E	W		Comments
		FPU pipeline-->				R1	R2	E1	E2	E3	
I1	MPYF32 R6H, R5H, R0H || MOV32 *XAR7++, R4H	I1									
I2	F32TOUI16R R3H, R4H	I2	I1								
I3	ADDF32 R3H, R2H, R0H || MOV32 *--SP, R2H	I3	I2	I1							
I4	NOP	I4	I3	I2	I1						
I5	MOV32 @XAR3, R6H	I5	I4	I3	I2	I1					
			I5	I4	I3	I2	I1				
				I5	I4	I3	I2	I1			
					I5	I4	I3	I2	I1		
						I5	I4	I3	I2	I1 (STALL)	Due to one extra NOP, I5 does not reach R2 when I1 enters E3; thus, forwarding is not needed.
						I5	I4	I3	I2	I1	There is no change due to the stall in the previous cycle.
							I5	I4	I3	I2	I1 moves out of E3 and I5 moves to R2. R6H has the result of R5H*R0H and is read by I5. There is no need to forward the result in this case.
								I5	I4	I3	

Figure 7. Pipeline Diagram With Workaround in Place

Advisory — *Memory: Prefetching Beyond Valid Memory*

Revision(s) Affected

0, A, B, E, F

Details

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

Workaround

The prefetch queue is 8 x16 words in depth. Therefore, code should not come within 8 words of the end of valid memory. This restriction applies to all memory regions and all memory types (flash, OTP, SARAM) on the device. Prefetching across the boundary between two valid memory blocks is all right.

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

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

Advisory — *Master Subsystem I2C: SDA and SCL Open-Drain Output Buffer Issue*

Revision(s) Affected

0, A, B, E, F

Details

The SDA and SCL outputs are implemented with push-pull 3-state output buffers rather than open-drain output buffers as required by I2C. While it is possible for the push-pull 3-state output buffers to behave as open-drain outputs, an internal timing skew issue causes the outputs to drive a logic-high for a duration of 0–5 ns before the outputs are disabled. The unexpected high-level pulse will only occur when the SCL or SDA outputs transition from a driven low state to a high-impedance state and there is sufficient internal timing skew on the respective I2C output.

This short high-level pulse injects energy in the I2C signals traces, which causes the I2C signals to sustain a period of ringing as a result of multiple transmission line reflections. This ringing should not cause an issue on the SDA signal because it only occurs at times when SDA is expected to be changing logic levels and the ringing will have time to damp before data is latched by the receiving device. The ringing may have enough amplitude to cross the SCL input buffer switching threshold several times during the first few nanoseconds of this ringing period, which may cause clock glitches. This ringing should not cause a problem if the amplitude is damped within the first 50 ns because I2C devices are required to filter their SCL inputs to remove clock glitches. Therefore, it is important to design the PCB signal traces to limit the duration of the ringing to less than 50 ns. One possible solution is to insert series termination resistors near the SCL and SDA terminals to attenuate transmission line reflections.

This issue may also cause the SDA output to be in contention with the slave SDA output for the duration of the unexpected high-level pulse when the slave begins its ACK cycle. This occurs because the slave may already be driving SDA low before the unexpected high-level pulse occurs. The glitch that occurs on SDA as a result of this short period of contention does not cause any I2C protocol issue but the peak current applies unwanted stress to both I2C devices and potentially increases power supply noise. Therefore, a series termination resistor located near the respective SDA terminal is required to limit the current during the short period of contention.

A similar contention problem can occur on SCL when connected to I2C slave devices that support clock stretching. This occurs because the slave is driving SCL low before the unexpected high-level pulse occurs. The glitch that occurs on SCL as a result of this short period of contention does not cause any I2C protocol issue because I2C devices are required to apply a glitch filter to their SCL inputs. However, the peak current applies unwanted stress to both I2C devices and potentially increases power supply noise. Therefore, a series termination resistor located near the respective SCL terminal is required to limit the current during the short period of contention.

If another master is connected, the unexpected high-level pulses on the SCL and SDA outputs can cause contention during clock synchronization and arbitration. The series termination resistors described above will also limit the contention current in this use case without creating any I2C protocol issue.

Workaround(s)

Insert series termination resistors on the SCL and SDA signals and locate them near the SCL and SDA terminals. The SCL and SDA pullup resistors should also be located near the SCL and SDA terminals. The placement of the series termination resistor and pullup resistor should be connected as shown in Figure 8.

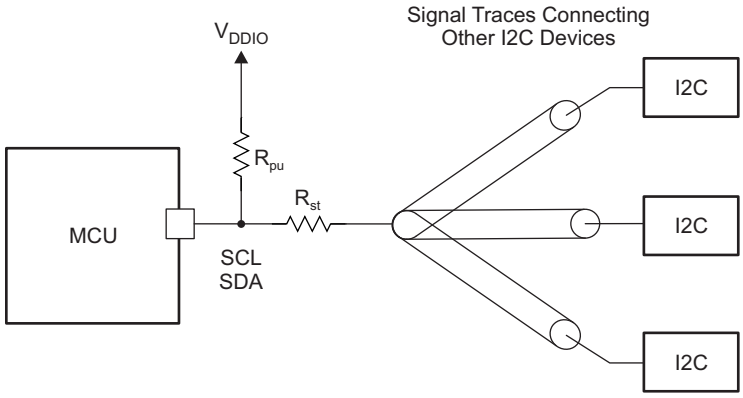


Figure 8. Placement of Series Termination Resistor and Pullup Resistor

Table 6 provides series termination and pullup resistor value recommendations. The I2C signal level and respective V_{DDIO} power supply voltage is shown in the first column. Two resistor value combination options are provided for each voltage. One option supports a maximum high-level input current of 200 uA to all attached I2C devices, while the other option supports a maximum high-level input current of 100 uA to all attached I2C devices.

Table 6. Recommended Values for Series Termination Resistor and Pullup Resistor

I2C SIGNAL LEVEL AND RESPECTIVE V_{DDIO} POWER SUPPLY (V)	SERIES TERMINATION RESISTOR (Ω)	PULLUP RESISTOR (Ω)	NOTES
3.3	60	3300	Maximum high-level input current up to 200 µA
3.3	75	6600	Maximum high-level input current up to 100 µA

Advisory — *HWBIST: Maximum Frequency*

Revision(s) Affected 0, A, B, E, F

Details The Cortex-M3 HWBIST and C28x HWBIST can only be run when both M3SSCLK and C28SYSCLK are at or below the maximum frequency given in Table 7.

Workaround(s) Set the device clock frequencies per Table 7 if using HWBIST. The maximum frequency depends on if the C28x-to-Cortex-M3 clock ratio is 2:1 or 1:1, and if the HWBIST is executed on only one CPU or both CPUs.

For part numbers with lower maximum frequencies than shown in Table 7, the lower frequency that is in the *F28M36x Concerto™ Microcontrollers Data Manual* still applies.

Table 7. Maximum Frequency (MHz) of System Clocks

CLOCK	1:1 C28SYSCLK-to-M3SSCLK Ratio			2:1 C28SYSCLK-to-M3SSCLK Ratio		
	Only C28x HWBIST	Only Cortex-M3 HWBIST	C28x and Cortex-M3 HWBIST	Only C28x HWBIST	Only Cortex-M3 HWBIST	C28x and Cortex-M3 HWBIST
C28SYSCLK	120	90	90	130	110	110
M3SSCLK	120	90	90	65	55	55

Advisory — *HWBIST: Unexpected Behavior When C28x HWBIST is Run With EPI Enabled*

Revision(s) Affected 0, A, B, E, F

Details Execution of C28x HWBIST with EPI enabled could result in unexpected device behavior. This advisory does not apply if the EPI is disabled.

1. The C28x may stall (C28x code execution will not resume) after HWBIST is executed when the C28x has write access to the EPI. The only recovery from the stalled condition is a reset of the C28x subsystem.
2. Spurious writes may occur on the EPI pins.

Workaround(s) Either of two workarounds listed below must be implemented before initiating a C28x HWBIST test interval.

1. The Cortex-M3 should disable C28x write access to the EPI by setting the MEMPROT register (0x400F_B938) to 0x55.

   If the EPI interrupt (INT12.6) has been enabled in PIE in the C28x application, it must be disabled before starting HWBIST. After HWBIST completion, the user should clear the MEMPROTERR flag in the CEPISTATUS register and the status bit for INT6 in the PIEIFR12 register, and then reenable the PIE interrupt for EPI.
2. The Cortex-M3 should disable the EPI. The EPI peripheral clock can be disabled by using the Driverlib function *SysCtlPeripheralDisable(SYSCTL_PERIPH_EPI0)* found in controlSUITE™.

The EPI can be restored to its original configuration after the C28x completes the HWBIST test interval.

Advisory	***HWBIST: C28x HWBIST Restriction on Real-Time Window of EPI Module***
Revision(s) Affected	A, B, E, F
Details	The Real-Time Window (RTW) module may get reset when a C28x HWBIST is in progress.
Workaround(s)	The RTW feature of the EPI must be disabled before executing a HWBIST. After executing the HWBIST, the RTW needs to be reconfigured.

Advisory	***HWBIST: C28x HWBIST Restriction on EPI Module***
Revision(s) Affected	A, B, E, F
Details	The C28x and C28x DMA AHB bridge may get reset during HWBIST execution.
Workaround(s)	All pending EPI accesses should be completed from the C28x and C28x DMA, and the CEPISTATUS register should be cleared before entering HWBIST. There is no restriction on the Cortex-M3 use of the EPI during C28x HWBIST execution.

Advisory	***HWBIST: C28x HWBIST Should Not be Used***
Revision(s) Affected	0, A, B
Details	Using the HWBIST feature on the C28x CPU can cause unpredictable behavior in the C28x subsystem.
Workaround(s)	None. The C28x HWBIST should not be used on the revisions affected.

Advisory

GPIO: GPIO38 and GPIO46 Signal Latch-up to V_{SS} Due to Fast Transient Sensitivity at High Temperature

Revision(s) Affected

0, A, B, E, F

Details

There is a potential signal latch-up to V_{SS} condition identified on pins GPIO38 and GPIO46. In this condition, an on-chip path to V_{SS} is turned on, which can bring down the logic level of these pins below V_{IL} and V_{OL}. The condition can occur when the pin is in input or output mode and with any of the alternate functions muxed on to this pin.

The condition is more likely to occur at high temperatures and has not been observed below 85°C under normal operating use cases. The triggering event is dependent on board design and the speed of signals switching on these pins, with fast-switching transients more likely to induce the condition. The condition has only been observed when the signal at the device pin has a rise time or fall time faster than 2 ns (measured 10% to 90% of V_{DDIO}).

The condition will resolve upon toggle of the IO at a lower temperature.

Workaround(s)

Try one of these two options:

- **Option 1:**

  Avoid the use of these pins in the revisions affected.

- **Option 2:**

  This condition is not seen on all products. Many PCB designs have enough capacitance and slow enough edge rates that the condition does not occur. If the application can be tested and functions correctly with the temperature margin above the end-use temperature, then no action may be required. If the issue is seen or additional margin is desired, then the following can be applied.

  If the pin is configured as an output without any other pullup or driver connected:

  – Place a capacitor of 56 pF or greater between each of these pins and ground, placed as closely as possible to the device. This will slow down the fast transient and avoid triggering the condition. Larger capacitors will be more effective at filtering the transient but must be balanced against the PCB level timing requirements of these pins.

  If the pin is configured as an input:

  – Connect a resistor in series with any other components on the board such that the total resistance of the driver plus the resistor is 1 kΩ or greater. This will serve two purposes:

    1. Eliminate the fast voltage transient seen at the pin.
    2. Limit the DC current if the shunt to V_{SS} is activated.

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

Revision(s) Affected 0, A, B, E, F

Details Each GPIO can be configured to an open-drain mode using the GPIOODR register. (Note that the open-drain feature is only supported on Master Subsystem GPIOs.) 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(s) 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 (GPIODATA) to a static 0 and toggling the GPIO direction bit (GPIODIR) to enable and disable the drive low.

Advisory — *GPIO: Pins GPIO38 and GPIO46 Cannot be Used for Functions Other Than USB*

Revision(s) Affected 0

Details Pins GPIO38 and GPIO46 can only be used as USB0VBUS and USB0ID. GPIO and other functions are not available.

Workaround(s) None. This is fixed in Revision A silicon.

Advisory — *GPIO: Cortex-M3/C28x Reads GPIO State as '0' When the GPIO is Mapped to the Other Core*

Revision(s) Affected 0

Details The Cortex-M3 core reads the GPIO state as '0' when the GPIO is mapped to the C28x core. Conversely, the C28x core reads the GPIO state as '0' when the GPIO is mapped to the Cortex-M3 core.

Workaround(s) None. This is fixed in Revision A silicon.

Advisory — *C28x Flash: A Single-Bit ECC Error May Cause Endless Calls to Single-Bit-Error Interrupt Service Routine*

Revision(s) Affected 0, A, B, E, F

Details When a single-bit ECC error is detected, the CPU executes the single-bit-error interrupt service routine (ISR). When the ISR returns, the same instruction that caused the first error is fetched again. If the ECC error threshold (ERR_THRESHOLD.THRESHOLD) is 0, then the same error is detected and another ISR is executed. This continues in an endless loop. This sequence of events only occurs if the error is caused by a program fetch operation, not a data read.

Workaround(s) Set the error threshold bit-field (ERR_THRESHOLD.THRESHOLD) to a value greater than or equal to 1. Note that the default value of the threshold bit-field is 0.

Advisory — *C28x Flash: The SBF and BF Instructions Will Not Execute From Flash*

Revision(s) Affected 0

Details When the flash prefetch is enabled, the SBF and BF instructions will not be correctly fetched from the Flash.

Workaround(s) Compile the application code with the –me option to avoid the SBF and BF instructions.

This is fixed in Revision A silicon.

Advisory — *Flash Pump Power Down: Software Sequence Must be Followed to Power Down the Flash Pump*

Revision(s) Affected

E, F

Details

On silicon revision E, there is a logic change to prevent one CPU from powering down the Flash Pump while the other CPU is using it.

As a consequence of this change, both the Cortex-M3 and the C28x CPU must each power down the Flash Pump without any Flash accesses in between. The Flash Pump will not enter low-power mode if this sequence is not followed.

Workaround(s)

The F28M3xx_examples_Dual\Sleep_Low_Power_Mode_Wakeup example in controlSUITE can be used as a reference.

Along with the automatic power down, the Flash bank can be forced into its low-power mode. To force the Flash to power down completely, execute the following procedure from RAM (and not from Flash):

1. When the system is ready to power down the Flash completely, synchronize the Cortex-M3 and C28x CPUs. The Cortex-M3 will enter its Flash power-down phase (steps 2 and 3) while the C28x will be waiting for it to complete.
2. Change the Cortex-M3 Flash Bank Fall Back power mode to Sleep: M3_FBFALLBACK.BNKPWR = 0.
3. Change the Cortex-M3 Flash Charge Pump Fall Back power mode to Sleep: M3_FPAC1.PMPPWR = 0.

   The Cortex-M3 should notify the C28x that it has completed the above sequence. It should wait until the C28x completes steps 4, 5, and 6.
4. Acquire the Pump Semaphore with the C28x.
5. Change the C28x Flash Bank Fall Back power mode to Sleep: C28_FBFALLBACK.BNKPWR = 0.
6. Change the C28x Flash Charge Pump Fall Back power mode to Sleep: C28_FPAC1.PMPPWR = 0.
7. Release the Pump Semaphore from the C28x.

The C28x should notify the Cortex-M3 that it has completed the power-down sequence so that both subsystems may continue. Also note that the PUMPRDY bit in the M3-FMC register space can be used by the application software to identify the current power mode status of the pump. The PUMPRDY bit (and also PMPPWR bit) in the C28x-FMC cannot be used for that purpose.

Advisory — *Crystal: Maximum Equivalent Series Resistance (ESR) Values are Reduced*

Revision(s) Affected 0, A, B

Details The maximum ESR values are reduced. For the revisions affected, the data in Table 8 supersedes the data given in the "Crystal Equivalent Series Resistance (ESR) Requirements" table in the *F28M36x Concerto™ Microcontrollers Data Manual*. The differences between the two tables are highlighted in Table 8.

Table 8. Crystal Equivalent Series Resistance (ESR) Requirements(1)

CRYSTAL FREQUENCY (MHz)	MAXIMUM ESR (Ω) (CL1/2 = 12 pF)	MAXIMUM ESR (Ω) (CL1/2 = 24 pF)
2	175	375
4	100	195
6	75	145
8	65	105
10	55	70
12	50	45
14	50	35
16	45	25
18	40	20
20	30	15

(1) Crystal shunt capacitance (C0) should be less than or equal to 7 pF.

Workaround(s) None

Advisory — *Reset: $\overline{XRS}$ May Toggle During Power Up*

Revision(s) Affected 0, A, B

Details During device power up, the $\overline{XRS}$ pin may toggle high prematurely. After the V_{DDIO} and V_{DD} supplies reach the recommended operating conditions, the $\overline{XRS}$ pin behavior will be as described in the *F28M36x Concerto™ Microcontrollers Data Manual*. This is only an issue with the external state of the $\overline{XRS}$ pin. Internally, the device will be held in reset until the supplies are within an acceptable range and $\overline{XRS}$ is high.

Workaround(s) Disregard $\overline{XRS}$ activity on the board before the supplies reach the recommended operating conditions.

Advisory — *Cortex-M3 Flash: C28x Reset While C28x Holding Pump Ownership Can Cause Erroneous Cortex-M3 Flash Reads*

Revision(s) Affected

0, A, B

Details

If the C28x Subsystem is reset while it owns the flash pump semaphore, then the flash pump itself will also reset. Since the flash pump is also used by the Cortex-M3 Subsystem, any instruction fetch or data read from flash by the Cortex-M3 will return invalid data. This will result in a hard fault, incorrect program execution, Cortex-M3 core hang condition, or an unspecified error in the application.

This erratum does not apply if the C28x Subsystem never writes to the CPUMPREQUEST register to take ownership of the flash pump semaphore.

Workaround(s)

The Cortex-M3 must not access flash while the C28x holds the flash pump semaphore ownership. The following steps describe how this can be achieved:

1. At application start-up, the C28x reads the CPUMPREQUEST semaphore register. If it is the owner, the C28x relinquishes the flash pump semaphore.
2. When the C28x wants to own the flash pump semaphore, it must notify the Cortex-M3 and wait for an acknowledgement.
3. The Cortex-M3 application branches to RAM and notifies the C28x that it has done so. Any data being accessed by the Cortex-M3 must also reside in RAM at this time.
4. The C28x takes ownership of the semaphore.
5. The Cortex-M3 will refrain from accessing the flash until the C28x releases ownership of the flash pump semaphore.

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

Revision(s) Affected 0

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

Workaround(s) None. This is fixed in Revision A silicon.

Advisory — *EPI: C28x Access to the EPI Bus on the Device*

Revision(s) Affected 0

Details On Revision 0 silicon, the Control Subsystem (C28x core) cannot access the EPI. Only the Master Subsystem (M3 core) can. Starting with Revision A silicon, the C28x has read and write data access to the EPI peripheral. Note that C28x use of the EPI for program execution is not supported in any silicon revision.

Workaround(s) This is fixed in Revision A silicon.

5 Documentation Support

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

For further information regarding the F28M36x Concerto devices, see the following documents:

- *F28M36x Concerto™ Microcontrollers Data Manual*
- *Concerto F28M36x Technical Reference Manual*

For errata relating to the Cortex-M3 r2p0 core, see the *ARM Core Cortex-M3 / Cortex-M3 with ETM (AT420/AT425) Errata Notice* at the ARM Ltd. website.

Trademarks

Revision History

Changes from December 15, 2017 to October 1, 2018 (from J Revision (December 2017) to K Revision) **Page**

- Section 4.2 (Known Design Exceptions to Functional Specifications): Added ePWM: ePWM Dead-Band Delay Value Cannot be Set to 0 When Using Shadow Load Mode for RED/FED advisory. 20
- Section 4.2: Added ePWM: An ePWM Glitch can Occur if a Trip Remains Active at the End of the Blanking Window advisory. 21
- Section 4.2: Added Master Subsystem I2C: SDA and SCL Open-Drain Output Buffer Issue advisory. 28
- Section 4.2: Added Master Subsystem GPIO: Open-Drain Configuration May Drive a Short High Pulse advisory. 33

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), 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, 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 (www.ti.com/legal/termsofsale.html) 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.

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