TMS320F2838x MCUs Silicon Errata Silicon Revision 0

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.

Topic **Page**

1 Usage Notes and Advisories Matrices

Table 1 lists all usage notes and the applicable silicon revision(s). Table 2 lists all advisories, modules affected, and the applicable silicon revision(s).

Table 1. Usage Notes Matrix

DESCRIPTION	SILICON REVISIONS AFFECTED
	0
Section 3.1.1, PIE: Spurious Nested Interrupt After Back-to-Back PIEACK Write and Manual CPU Interrupt Mask Clear	Yes
Section 3.1.2, Caution While Using Nested Interrupts	Yes

Table 2. Advisories Matrix

MODULE	DESCRIPTION	SILICON REVISIONS AFFECTED
		0
ADC	ADC: External SOC Trigger (ADCEXTSOC) Cannot be Used to Trigger Multiple ADCs	Yes
ADC	ADC: Interrupts may Stop if INTxCONT (Continue-to-Interrupt Mode) is not Set	Yes
ESC	ESC: EtherCAT Slave Controller Distributed Clocking (DC) Mode is not Supported	Yes
ESC	ESC: EtherCAT Slave Controller may not Work with a Non-Gigabit PHY	Yes
ESC	ESC: The CPU1 DMA Access is Only Available to the Lower 4KB of the ESC RAM Memory Map	Yes
ePWM	ePWM: An ePWM Glitch can Occur if a Trip Remains Active at the End of the Blanking Window	Yes
FPU	FPU: FPU-to-CPU Register Move Operation Preceded by Any FPU 2p Operation	Yes
FSI	FSI: RX FIFO Spurious Overrun	Yes
GPIO	GPIO: Open-Drain Configuration may Drive a Short High Pulse	Yes
INTOSC	INTOSC: VDDOSC Powered Without VDD can Cause INTOSC Frequency Drift	Yes
MCD	MCD: Missing Clock Detect Should be Disabled When the PLL is Enabled (PLLCLKEN = 1)	Yes
Memory	Memory: Prefetching Beyond Valid Memory	Yes
MPOST	MPOST: Memory Power-on-Self-Test will not Work at Full Speed	Yes
Reset	Reset: Elevated VDD Current During Reset	Yes
SDFM	SDFM: Dynamically Changing Comparator Settings will Trigger Spurious Comparator Events	Yes
SDFM	SDFM: Dynamically Changing Data Filter Settings will Trigger Spurious Data Acknowledge Events	Yes
USB	USB: USB DMA Event Triggers are not Supported	Yes
Watchdog	Watchdog: WDKEY Register is not EALLOW-Protected	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, TMS320F28388D). 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/TMDX) through fully qualified production devices/tools (TMS/TMDS).

Device development evolutionary flow:

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

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

TMS — Fully-qualified production device.

Support tool development evolutionary flow:

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

TMDS — Fully qualified development-support product.

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

"Developmental product is intended for internal evaluation purposes."

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

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

2.2 Devices Supported

This document supports the following devices:

- TMS320F28388D
- TMS320F28386D
- TMS320F28384D
- TMS320F28388S
- TMS320F28386S
- TMS320F28384S

2.3 Package Symbolization and Revision Identification

Figure 1 shows the package symbolization and Table 3 lists the silicon revision codes.

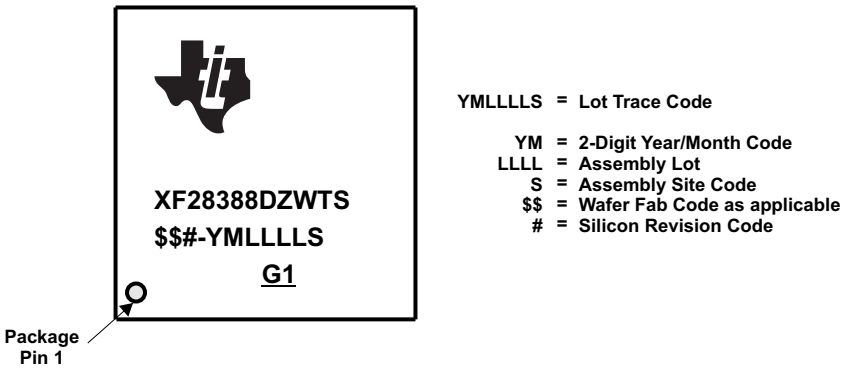

Figure 1. Package Symbolization

Table 3. Revision Identification

SILICON REVISION CODE	SILICON REVISION	REVID(1) Address: 0x5D00C	COMMENTS(2)
Blank	0	0x0000 0000	This silicon revision is available as TMX.

(1) Silicon Revision ID

(2) For orderable device numbers, see the PACKAGING INFORMATION table in the *TMS320F2838x Microcontrollers With Connectivity Manager Data Manual*.

3 Silicon Revision 0 Usage Notes and Advisories

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

3.1 Silicon Revision 0 Usage Notes

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

Revision(s) Affected: 0

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

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

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

Revision(s) Affected: 0

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

3.2 Silicon Revision 0 Advisories

Table 4. Silicon Revision 0 Advisory List

Advisory

ADC: External SOC Trigger (ADCEXTSOC) Cannot be Used to Trigger Multiple ADCs

Revision(s) Affected

0

Details

When multiple ADCs are to be used in parallel (for simultaneous sampling), the ADCs should use the same trigger source to ensure all ADCs start synchronously for best ADC performance.

The external SOC trigger source (ADCEXTSOC) from the Input-XBAR may not trigger all ADCs simultaneously due to internal timing skew from the XBAR to each ADC.

Workaround(s)

1. Use an alternate trigger source when multiple simultaneous ADC conversions are required.
2. Only use ADCEXTSOC to trigger a single ADC.
3. Account for the reduced ADC performance when using ADCEXTSOC to trigger multiple ADCs for ADC modes that specify reduced performance for asynchronous operation.

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

Revision(s) Affected

0

Details

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

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

Workaround(s)

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 — *ESC: EtherCAT Slave Controller Distributed Clocking (DC) Mode is not Supported*

Revision(s) Affected 0

Details The EtherCAT Slave Controller Distributed Clocking registers responsible for offset and drift compensation are only writable by the host CPU and not by the EtherCAT master. Because of this, Distributed Clocking Mode is not supported on the affected revision.

Workaround(s) None

Advisory — *ESC: EtherCAT Slave Controller may not Work with a Non-Gigabit PHY*

Revision(s) Affected 0

Details The EtherCAT Slave Controller configures the PHY registers assuming it is gigabit-capable. This may result in undesirable PHY operation if a non-gigabit PHY is used.

Workaround(s) Use a gigabit PHY.

Advisory — *ESC: The CPU1 DMA Access is Only Available to the Lower 4KB of the ESC RAM Memory Map*

Revision(s) Affected 0

Details The CPU1 DMA access is only available to the lower 4KB of the ESC RAM memory map. This memory region is 0x52000 to 0x527FF.

Workaround(s) None

Advisory

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

Revision(s) Affected

0

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

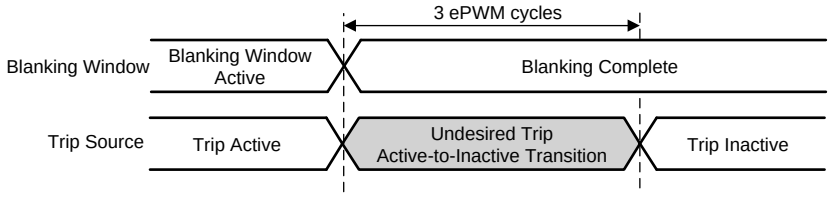

Figure 2. Undesired Trip Event and Blanking Window Expiration

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

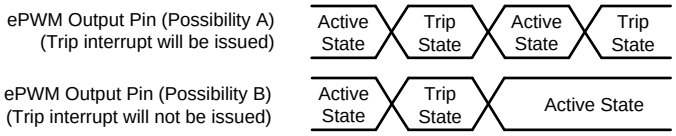

Figure 3. Resulting Undesired ePWM Outputs Possible

Workaround(s)

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

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

Revision(s) Affected 0

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

Figure 5 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 5. 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 6 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 6. Pipeline Diagram With Workaround in Place

Advisory — *FSI: RX FIFO Spurious Overrun*

Revision(s) Affected

0

Details

A buffer overrun is asserted when the last location of the FIFO is written.

Workaround(s)

Two possible workarounds are available.

1. Set up the communication between the transmitting and receiving modules in such a way that the maximum number of data words received, before the first data word is read, is 15 (not 16). Under this condition, buffer overrun behavior is reliable.
2. If the application must fill all 16 data words in the receive buffer before the first data word is read (NWORD packet with 16 words), then the following sequence can be used:
   - Ignore RX buffer overrun RX_EVT_STS.BUF_OVERRUN.
   - On RX_EVT_STS.FRAME_DONE, read RX_BUF_PTR_STS.CURR_WORD_CNT and check that it is 16.
   - Use DMA or software to move the data out of the RX buffer.
   - Read RX_BUF_PTR_STS.CURR_WORD_CNT and check that it is 0.
   - Clear the RX_EVT_STS.FRAME_DONE Flag by writing a 1 to RX_EVT_CLR.FRAME_DONE.

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

Revision(s) Affected

0

Details

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

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

Workaround(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 (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 — *INTOSC: VDDOSC Powered Without VDD can Cause INTOSC Frequency Drift*

Revision(s) Affected

0

Details

If VDDOSC is powered on while VDD is not powered, there will be an accumulating and persistent downward frequency drift for INTOSC1 and INTOSC2. The rate of drift accumulated will be greater when VDDOSC is powered without VDD at high temperatures.

As a result of this drift, the INTOSC1 and INTOSC2 internal oscillator frequencies could fall below the minimum values specified in the *TMS320F2838x Microcontrollers With Connectivity Manager Data Manual*. This would impact applications using INTOSC2 as the clock source for the PLL, with the system operating at a lower frequency than expected.

Workaround(s)

1. Keep VDDOSC and VDD powered together.
2. Use the external X1 and X2 crystal oscillators as the PLL clock source. The crystal oscillator does not have any drift related to VDDOSC and VDD supply sequencing.

Advisory

MCD: Missing Clock Detect Should be Disabled When the PLL is Enabled (PLLCLKEN = 1)

Revision(s) Affected

0

Details

The PLL has a limp mode feature to provide a slow PLLRAWCLK output even if its input OSCCLK is absent. Independently, the Missing Clock Detect (MCD) circuit will forcibly switch the system clock source to INTOSC1 when a missing OSCCLK input is detected. The MCD mux to switch between these system clock sources is not ensured to be glitch-free when both clock sources (PLLRAWCLK and INTOSC1) are still active. In rare cases, this may lead to unpredictable device behavior during a missing clock failure event.

Workaround(s)

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*

Revision(s) Affected

0

Details

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

Workaround

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

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

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

Table 5. Memories Impacted by Advisory

MEMORY TYPE	ADDRESSES IMPACTED
M1	0x0000 07F8–0x0000 07FF
GS15	0x0001 BFF8–0x0001 BFFF
Flash	0x000B FFF8–0x000B FFFF

Advisory	***MPOST: Memory Power-on-Self-Test will not Work at Full Speed***
Revision(s) Affected	0
Details	Memory Power-on-Self-Test (MPOST) may fail with any clock option except PLL bypass.
Workaround(s)	If using MPOST, use only PLL bypass option (GPREG2[7:4] = 0×9).

Advisory — *Reset: Elevated VDD Current During Reset*

Revision(s) Affected

0

Details

There is an elevated current of approximately 70 mA for each C28x CPU while held in reset. When the XRSn pin is low, both C28x cores will be in reset, resulting in 140 mA of additional current. After XRSn is released, there will be approximately 70 mA of current from the C28x CPU2 until it is released by from reset by CPU1.

Workaround(s)

Do not hold XRSn low or keep CPU2 in reset for extended durations when current consumption is a concern.

Advisory — *SDFM: Dynamically Changing Comparator Settings will Trigger Spurious Comparator Events*

Revision(s) Affected 0

Details When SDFM comparator settings—such as filter type or COSR—are changed while low-level/high-level events are enabled, a spurious comparator lower-threshold or higher-threshold event will be triggered.

Workaround(s) When comparator settings—such as filter type or COSR—need to be changed dynamically, follow the procedure below:

1. Disable the SDFM comparator.
2. Change comparator settings (such as filter type or COSR).
3. Delay for at least a latency of comparator filter + 5 SD-Cx clock cycles.
4. Enable the SDFM comparator.

Advisory — *SDFM: Dynamically Changing Data Filter Settings will Trigger Spurious Data Acknowledge Events*

Revision(s) Affected 0

Details When SDFM data filter settings—such as filter type or DOSR—are changed while the data filter and its data acknowledge events are enabled, spurious data acknowledge events will be triggered.

Workaround(s) When data filter settings—such as filter type or DOSR—need to be changed dynamically, follow the procedure below:

1. Disable the SDFM data filter.
2. Change data filter settings (such as filter type or DOSR).
3. Delay for at least a latency of data filter + 5 SD-Cx clock cycles.
4. Enable the SDFM data filter.

Advisory — *USB: USB DMA Event Triggers are not Supported*

Revision(s) Affected 0

Details The USB module generates inadvertent extra DMA requests, causing the FIFO to overflow (on IN endpoints) or underflow (on OUT endpoints). This causes invalid IN DATA packets (larger than the maximum packet size) and duplicate receive data.

Workaround(s) None

Advisory — *Watchdog: WDKEY Register is not EALLOW-Protected*

Revision(s) Affected 0

Details The WDKEY register is not EALLOW-protected. Issuing the EALLOW and EDIS instructions to write to this register is not required. To enable software reuse on other devices where WDKEY is EALLOW-protected, using EALLOW and EDIS is recommended.

Workaround(s) None

4 Documentation Support

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

For more information regarding the TMS320F2838x devices, see the following documents:

- *TMS320F2838x Microcontrollers With Connectivity Manager Data Manual*
- *TMS320F2838x Microcontrollers Technical Reference Manual*

Trademarks

All trademarks are the property of their respective owners.

Revision History

DATE	REVISION	NOTES
May 2019	*	Initial Release

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