

Errata

TMS320C2834x Delfino MCUs Silicon Revision 0



1 Introduction

This document describes the silicon updates to the functional specifications for the TMS320C2834x microcontrollers (MCUs).

The updates are applicable to:

- 179-ball MicroStar BGA™, ZHH and ZAY Suffix
- 256-ball Plastic BGA, ZFE 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 [TMS320] DSP devices and support tools. Each TMS320™ DSP commercial family member has one of three prefixes: TMX, TMP, or TMS (for example, **TMS320C28345**). 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).

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.

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

3 Device Markings

Figure 3-1 provides examples of the 2834x 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 3-1. Some prototype devices may have markings different from those illustrated. Figure 3-2 shows the device nomenclature.

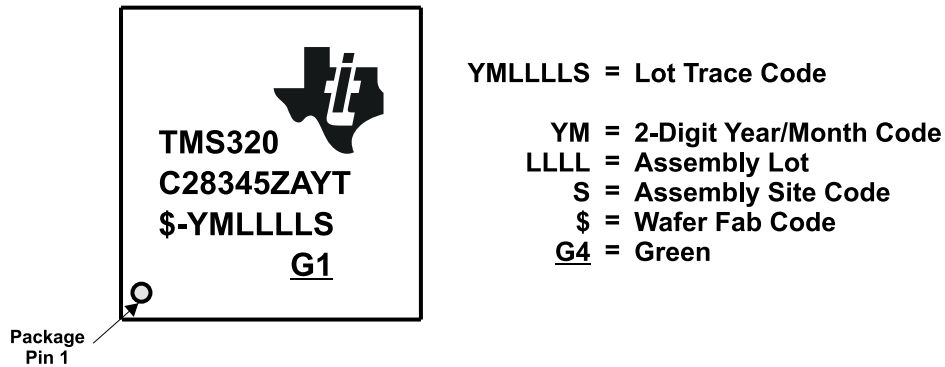


Figure 3-1. Examples of Device Markings

Table 3-1. Determining Silicon Revision From Lot Trace Code

LETTER FOLLOWING WAFER FAB CODE	SILICON REVISION	REVISION ID Address: 0x0883	COMMENTS
Blank (no letter follows the Wafer Fab Code)	Indicates Revision 0	0x0000	This silicon revision is available as TMS.

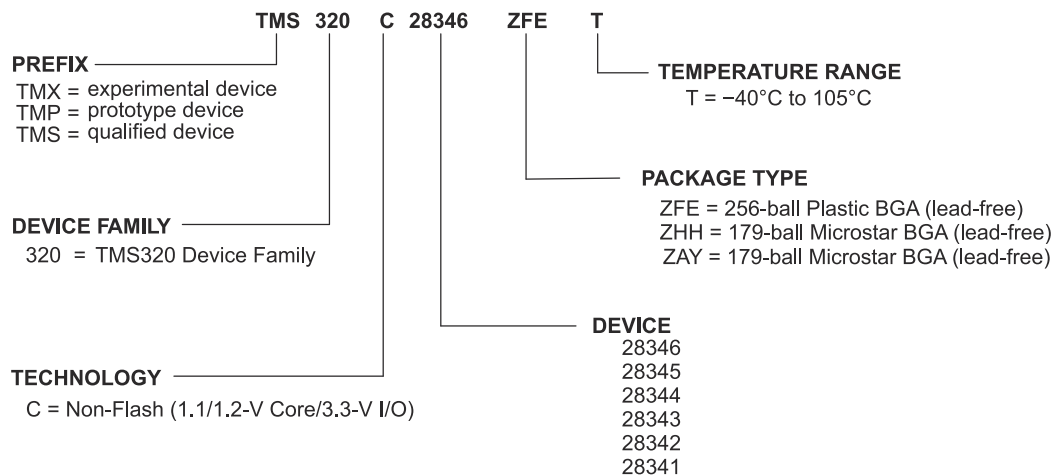


Figure 3-2. Device Nomenclature

4 Usage Notes and Known Design Exceptions to Functional Specifications

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.

4.1.1 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 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 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.

4.2 Known Design Exceptions to Functional Specifications

Table 4-1 shows which silicon revision(s) are affected by each advisory

Table 4-1. List of Advisories

TITLE	SILICON REVISION(S) AFFECTED
	0
Boot ROM - Incorrect MUX Configuration for Jump to XINTF x32	Yes
Memory: Prefetching Beyond Valid Memory	Yes
SCI: Incorrect Operation of SCI in Address Bit Mode	Yes
eCAN: Abort Acknowledge Bit Not Set	Yes
eCAN: Unexpected Cessation of Transmit Operation	Yes
GPIO: GPIO Qualification	Yes
PWM: Internal Pullups Briefly Enabled During Power Up	Yes
eQEP: Missed First Index Event	Yes
eQEP: Position Counter Incorrectly Reset on Direction Change During Index	Yes
eQEP: eQEP Inputs in GPIO Asynchronous Mode	Yes
FPU: CPU-to-FPU Register Move Operation Followed By F32TOUI32, FRACF32, or UI16TOF32 Operations	Yes
FPU: FPU-to-CPU Register Move Operation Preceded by Any FPU 2p Operation	Yes

Advisory ***Boot ROM - Incorrect MUX Configuration for Jump to XINTF x32***

Revision(s) Affected 0**Details** The boot ROM incorrectly configures GPBMUX2 for peripheral operation instead of XD[31:16]. This issue affects the jump to XINTF x32 boot mode.**Workaround** None. Use "Jump to XINTF x16 boot mode" instead as x32 mode is not supported in this device family.**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. This restriction applies to all memory regions and all memory types (flash, OTP, SARAM, XINTF) on the device. Prefetching across the boundary between two valid memory blocks is all right.

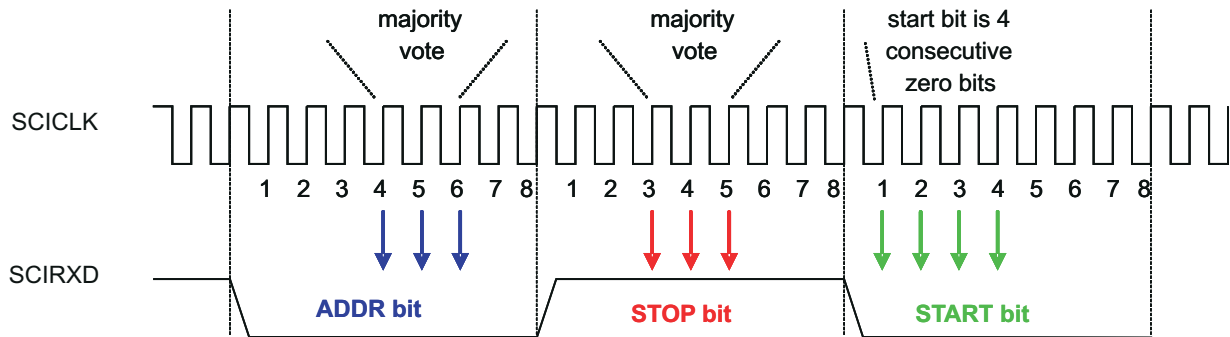
Example 1: M1 ends at address 0x7FF and is not followed by another memory block. Code in M1 should be stored no farther than address 0x7F7. Addresses 0x7F8–0x7FF should not be used for code.

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 **SCI: Incorrect Operation of SCI in Address Bit Mode**
Revision(s) Affected 0

Details SCI does not look for STOP bit after the ADDR bit. Instead, SCI starts looking for the start bit beginning on sub-sample 6 of the ADDR bit. Slow rise-time from ADDR to STOP bit can cause the false START bit to occur since the 4th sub-sample for the start bit may be sensed low.

Expected Operation:



Erroneous Operation:

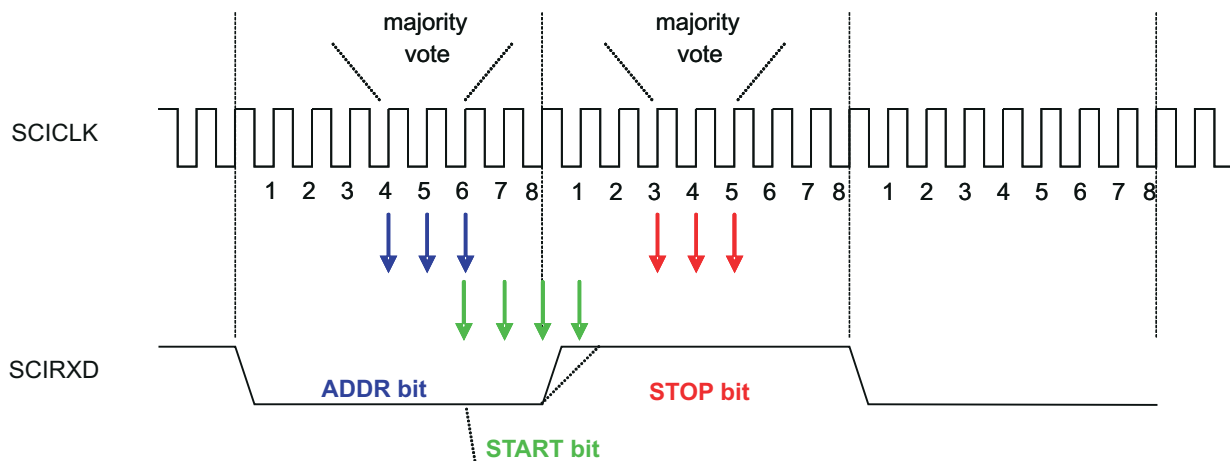


Figure 4-1. Difference Between Expected and Erroneous Operation of START Bit

Workaround(s) Program the baud rate of the SCI to be slightly slower than the actual. This will cause the 4th sub-sample of the false START bit to be delayed in time, and therefore occur more towards the middle of the STOP bit (away from the signal transition region). The amount of baud slowing needed depends on the rise-time of the signal in the system. Alternatively, IDLE mode of the SCI module may be used, if applicable.

Advisory **eCAN: Abort Acknowledge Bit Not Set**

Revision(s) Affected 0

Details

After setting a Transmission Request Reset (TRR) register bit to abort a message, there are some rare instances where the TRRn and TRSn bits will clear without setting the Abort Acknowledge (AAn) bit. The transmission itself is correctly aborted, but no interrupt is asserted and there is no indication of a pending operation.

In order for this rare condition to occur, all of the following conditions must happen:

1. The previous message was not successful, either because of lost arbitration or because no node on the bus was able to acknowledge it or because an error frame resulted from the transmission. The previous message need not be from the same mailbox in which a transmit abort is currently being attempted.
2. The TRRn bit of the mailbox should be set in a CPU cycle immediately following the cycle in which the TRSn bit was set. The TRSn bit remaining set due to incompleteness of transmission satisfies this condition as well; that is, the TRSn bit could have been set in the past, but the transmission remains incomplete.
3. The TRRn bit must be set in the exact SYSCLKOUT cycle where the CAN module is in idle state for one cycle. The CAN module is said to be in idle state when it is not in the process of receiving/transmitting data.

If these conditions occur, then the TRRn and TRSn bits for the mailbox will clear t_{clr} SYSCLKOUT cycles after the TRR bit is set where:

$$t_{clr} = [(\text{mailbox_number}) * 2] + 3 \text{ SYSCLKOUT cycles}$$

The TAn and AAn bits will not be set if this condition occurs. Normally, either the TA or AA bit sets after the TRR bit goes to zero.

Workaround(s)

When this problem occurs, the TRRn and TRSn bits will clear within t_{clr} SYSCLKOUT cycles. To check for this condition, first disable the interrupts. Check the TRRn bit t_{clr} SYSCLKOUT cycles after setting the TRRn bit to make sure it is still set. A set TRRn bit indicates that the problem did not occur.

If the TRRn bit is cleared, it could be because of the normal end of a message and the corresponding TAn or AAn bit is set. Check both the TAn and AAn bits. If either one of the bits is set, then the problem did not occur. If they are both zero, then the problem did occur. Handle the condition like the interrupt service routine would except that the AAn bit does not need clearing now.

If the TAn or AAn bit is set, then the normal interrupt routine will happen when the interrupt is re-enabled.

Advisory **eCAN: Unexpected Cessation of Transmit Operation**

Revision(s) Affected 0

Details

In rare instances, the cessation of message transmission from the eCAN module has been observed (while the receive operation continues normally). This anomalous state may occur without any error frames on the bus.

Workaround(s)

The Time-out feature (MOTO) of the eCAN module may be employed to detect this condition. When this occurs, set and clear the CCR bit (using the CCE bit for verification) to remove the anomalous condition.

Advisory ***GPIO: GPIO Qualification***

Revision(s) Affected 0

Details If a GPIO pin is configured for "n" SYSCLKOUT cycle qualification period (where $1 \leq n \leq 510$) with "m" qualification samples ($m = 3$ or 6), it is possible that an input pulse of $[n * m - (n - 1)]$ width may get qualified (instead of $n * m$). This depends upon the alignment of the asynchronous GPIO input signal with respect to the phase of the internal prescaled clock, and hence, is not deterministic. The probability of this kind of wrong qualification occurring is "1/n".

Worst-case example:

If $n = 510$, $m = 6$, a GPIO input width of $(n * m) = 3060$ SYSCLKOUT cycles is required to pass qualification. However, because of the issue described in this advisory, the minimum GPIO input width which may get qualified is $[n * m - (n - 1)] = 3060 - 509 = 2551$ SYSCLKOUT cycles.

Workaround(s) None. Ensure a sufficient margin is in the design for input qualification.

Advisory ***PWM: Internal Pullups Briefly Enabled During Power Up***

Revision(s) Affected 0

Details During power up, the pullups on the PWM pins are forced on briefly. This causes a low current glitch on the pin. ***This is not an issue during a warm reset.***

Workaround(s) This can be overcome by using a 1–2 k Ω resistor to ground the pin.

Advisory ***eQEP: Missed First Index Event***

Revision(s) Affected 0

Details

If the first index event edge at the QEPI input occurs at any time from one system clock cycle before the corresponding QEPA/QEPB edge to two system clock cycles after the corresponding QEPA/QEP edge, then the eQEP module may miss this index event. This can result in the following behavior:

- QPOSCNT will not be reset on the first index event if QEPCTL[PCRM] = 00b or 10b (position counter reset on an index event or position counter reset on the first index event).
- The first index event marker flag (QEPSTS[FIMF]) will not be set.

Workaround(s)

Reliable operation is achieved by delaying the index signal such that the QEPI event edge occurs at least two system clock cycles after the corresponding QEPA/QEPB signal edge. For cases where the encoder may impart a negative delay (t_d) to the QEPI signal with respect to the corresponding QEPA/QEPB signal (that is, QEPI edge occurs before the corresponding QEPA/QEPB edge), the QEPI signal should be delayed by an amount greater than " $t_d + 2 \cdot \text{SYSCLKOUT}$ ".

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

Revision(s) Affected 0

Details

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

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

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

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

Revision(s) Affected 0**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
FPU: CPU-to-FPU Register Move Operation Followed By F32TOUI32, FRACF32, or UI16TOF32 Operations
Revision(s) Affected 0

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

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 4-2 shows the pipeline diagram of the issue when there are no stalls in the pipeline.

	Instruction	F1	F2	D1	D2	R1	R2	E	W		Comments
		FPU pipeline-->					R1	R2	E1	E2	
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 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 4-2. Pipeline Diagram of the Issue When There are no Stalls in the Pipeline

Figure 4-3 shows the pipeline diagram of the issue if there is a stall in the E3 slot of the instruction I1.

	Instruction	F1	F2	D1	D2	R1	R2	E	W	Comments	
		FPU pipeline-->					R1	R2	E1		E2
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 4-3. 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 4-4 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	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	I1	There is no change due to the stall in the previous cycle.
						I5	I4	I3	I2	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	I3	

Figure 4-4. Pipeline Diagram With Workaround in Place

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 2834x devices, see the [TMS320C2834x Delfino Microcontrollers Data Manual](#).

6 Trademarks

MicroStar BGA™ and TMS320™ are trademarks of Texas Instruments.
All other trademarks are the property of their respective owners.

Revision History

Changes from August 7, 2018 to September 30, 2020 (from Revision I (August 2018) to Revision J (September 2020))

Page

-
- [Figure 3-1](#) (Examples of Device Markings): Updated Figure.....2
 - [Figure 3-2](#) (Examples of Device Nomenclature): Updated Figure.....2
-

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
Copyright © 2020, Texas Instruments Incorporated