TMS320C6701

Digital Signal Processor

Silicon Errata

SPRZ179A
January 2001

TEXAS INSTRUMENTS

Contents

1 Introduction

This document describes the silicon updates to the functional specifications for the TMS3206701 silicon releases 0.0 and 2.0.

1.1 Quality and Reliability Conditions

TMX Definition

Texas Instruments (TI) does not warranty either (1) electrical performance to specification, or (2) product reliability for products classified as “TMX.” By definition, the product has not completed data sheet verification or reliability performance qualification according to TI Quality Systems Specifications.

The mere fact that a “TMX” device was tested over a particular temperature and voltage ranges should not, in any way, be construed as a warranty of performance.

TMP Definition

TI does not warranty product reliability for products classified as “TMP.” By definition, the product has not completed reliability performance qualification according to TI Quality Systems Specifications; however, products are tested to a published electrical and mechanical specification.

TMS Definition

Fully-qualified production device.

1.2 Revision Identification

The device revision can be determined by the lot trace code marked on the top of the package. The location for the lot trace codes for the GJC package are shown in Figure 1 and Table 1.

Figure 1. Example, Lot Trace Code for TMS3206701

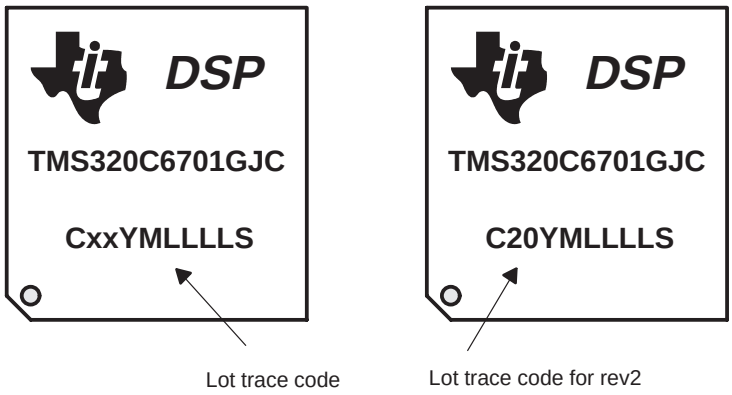

Table 1. Lot Trace Codes

Lot Trace Code	Silicon Revision	Comments
20	2.0	
1x	1.x	Functionally the same as revision 2.0
00	0.0	

2 Silicon Revision 2.0 Known Design Exceptions to Functional Specifications

Revision 1x is functionally the equivalent of revision 2.0.

Advisory 2.0.1	*EMIF: Invalid SDRAM Access to Last 1K Byte of CE3*

Revision(s) Affected: Rev 2.0, 0.0

Details: If 16M bytes of SDRAM (2 64M bit in a 1Mx16x4 organization) is used in CE3 then you can have invalid accesses to the last 1K byte of CE3 (0x03FFFC00).

This occurs when the following is true:

- After a DCAB (deactivate all pages) to all SDRAM CE spaces (forced by Refresh or MRS command)
- The first access to CE3 is to the last page of CE3 (0x03FFFC00).

A page activate will not be issued to CE3. Since the SDRAM in CE3 is in a deactivated state at that point, invalid accesses will occur. (Internal reference number C630280)

Workaround: Best Case: Avoid designing a board with a 64M bit (1Mx16x4) SDRAM mapped into CE3.

Alternative: If a 64M-bit SDRAM is located in CE3, avoid using the last 1K byte in the CE3 memory map (0x03FFFC00).

Advisory 2.0.2	*Cache During Emulation With Extremely Slow External Memory*

Revision(s) Affected: Rev 2.0, 0.0

Details: If a program requests fetch packet “A” followed immediately by fetch packet “B”, and all of the following four conditions are true:

1. A and B are separated by a multiple of 64k in memory (that is, they will occupy the same cache frame)
2. B is currently located in cache
3. You are using the emulator to single-step through the branch from A to B
4. The code is running off of an extremely slow external memory that transfers one 32-bit word every 8000+ CPU clock cycles (CPU running at 200 MHz)

Then A will be registered as a “miss” and B will be registered as a “hit”. B will not be reloaded into cache, and A will be executed twice. This condition is extremely rare because B has to be in cache memory, and must be the next fetch packet requested after A (which is not in cache memory). In addition, this problem only occurs if you single-step through the branch from A to B using the emulator, *and* if the code is located in an extremely slow external memory. (Internal reference number C630283)

Workaround: Do not single-step through the branch from A to B if the above conditions are true.

Do not use an extremely slow external memory (transfers one 32-bit word every 8000+ CPU clock cycles) if conditions 1, 2, and 3 are true.

Advisory 2.0.3

DMA: Split-Mode Transfers Corrupted if Channel 1, 2, 3 Are Stopped

Revision(s) Affected: Rev 2.0, 0.0

Details: There is a problem with stopping DMA channel 1, 2, or 3 when operating in split-mode transfers. If the DMA split-mode receive-and-transmit transfers are not in sync with one another when the channel is stopped, and then the same DMA channel is programmed for a new split-mode transfer, the new transfer will execute correctly but may not terminate completely. This problem does not exist in channel 0. (Internal reference number C621764)

Workaround: Do not stop DMA channels 1, 2, and 3 when they are operating in split-mode or manually force the number of elements received and transmitted transfers to be equal. Split-mode is most commonly used with the on-chip McBSPs. In typical McBSP applications, the transmit data is two elements ahead of the receive data; therefore, to stop the serial transfer do the following:

- Reset the McBSP to prevent additional sync events
- Set RSYNC_STAT twice for the DMA channel to force two receive transfers
- Stop the DMA channel

To ensure that the same number of elements are transferred, the source and destination addresses can be checked.

Advisory 2.0.4

EMIF: Full Speed (1x Mode) SBSRAM Reads Do Not Achieve Full Bus Utilization

Revision(s) Affected: Rev 2.0, 0.0

Details: When the SBSRAM output clock is configured as 1x, the CPU clock frequency read bursts do not achieve maximum bus utilization. The EMIF performs consecutive accesses up to 11 elements long, then stalls for up to 10 cycles. For example, if a 22-word burst is performed, the EMIF will perform an 11-cycle burst followed by a 10-cycle stall, followed by an 11-cycle burst. Writes to 1x SBSRAM do not have this problem. Also, if the SBSRAM interface is configured to operate in 12x mode, this problem does not exist.

Workaround: None

3 Silicon Revision 0.0 Known Design Exceptions to Functional Specifications

Advisory 0.0.1 *Data Memory Controller: LDDW Bug*

Revision(s) Affected: Rev 0.0

Details:

LDDW from external data memory (any CE space) fetches only the lower 32 bits instead of 64 bits. However, LDDW from internal data memory works correctly and fetches the full 64-bit data, except for any one of the following cases listed below, in which LDDW from internal data memory incorrectly fetches only the lower 32 bits instead of 64 bits:

1. Code sequence causes

Two successive execution packets with either one of the following patterns can cause the LDDW error:

PACKET	A-SIDE INSTRUCTION	B-SIDE INSTRUCTION	COMMENTS
1	LDDW	Any store instruction	Internal data memory bank conflict
2	Any load instruction	Any store instruction	Internal data memory bank conflict

OR

PACKET	A-SIDE INSTRUCTION	B-SIDE INSTRUCTION	COMMENTS
1	Any load/store	LDDW	Internal data memory bank conflict
2	Any load/store	Any LOAD	Internal data memory bank conflict

Both of the above code sequences cause the LDDW instruction to return corrupted data.

2. Step mode causes

Stepping through any code sequence that contains an LDDW instruction will cause the internal LDDW error.

3. DMA causes

If a DMA access causes an internal data memory bank conflict with another load or store instruction in the same execute packet with an LDDW instruction, the LDDW instruction will return only the lower 32 bits of data. This problem only occurs if DMA has priority, since the bug is caused by the CPU stalling. If the CPU has priority, the CPU will not stall (unless you also have cause 1 or cause 2 happening). (Internal reference number 1, 3)

WORKAROUND: Do not use LDDW to fetch data from external memory. When using the compiler, allocate all accessed data to internal data memory since there is no assurance that the compiler will not use the LDDW instruction. In addition, some of the Double Precision math library functions in rts6701.lib and rts6701e.lib are found to use the LDDW instruction. In those cases, try to use the equivalent single-precision library function. For example, use ”float logf(float x)” instead of ”double log(double x)”. When using hand-coded or linear assembly code, if it is not possible to allocate data to internal data memory, avoid using the LDDW instruction to access this data. LDB, LDH, and LDW can all be used instead.

DATA MEMORY CONTROLLER: LDDW Bug (Continued)

To use LDDW from internal memory without failure, you must ensure that the code pattern outlined in (1) above is never generated (note that the data bank conflicts are required in this pattern for a failure to occur).

You may single-step code to debug, but DO NOT single step over the execution of an LDDW instruction and all five of the cycles of latency of the LDDW instruction. Use a breakpoint after the five-cycle latency to resume single stepping of the program.

To use DMA in programs that use LDDW from internal memory, you must ensure that the execute packets that contain a LDDW instruction do *not* contain another load or store access, so that DMA accesses will not cause internal data memory bank conflicts.

Advisory 0.0.2 — *Multicycle Stalls During Internal Data Memory Bank Conflicts*

Revision(s) Affected: Rev 0.0

Details: Program flow will get corrupted data if all of the following are true:

- The program contains an execute packet with a B-side internal data load.
- This B-side internal data load is followed by an execute packet with a parallel load that generates an internal data bank conflict (address bits 1, 2, 3, and 15 are the same between the loads).
- AND a multicycle stall occurs during the execute of the parallel load packet.

The data for the first B-side load will be corrupted by the data for the second B-side load. The original B-side load data will be lost.

In this description, B-side refers to the destination register for the load, *not* the D-unit or address register. (Internal reference number 2)

Workaround(s):

1. Ensure the code does not contain any internal data bank conflicts (a brute-force method is to make sure there are no execute packets with parallel loads).
2. Make sure the code that includes parallel loads with internal data bank conflicts will not have any stalls generated (due to external data fetches, external instruction fetches, high priority DMA activity, user single-steps or breakpoints, or any other cause of a stall). In that case, only a single-cycle stall will occur due to data bank conflicts. The program will work correctly.

Advisory 0.0.3

DMA: Transfer Incomplete When Pausing a Frame-Synchronized Transfer in Midframe

Revision(s) Affected: Rev 0.0

Details: If a frame-synchronized transfer is paused in mid-frame and then restarted again, a DMA channel does not continue the transfer. Instead, the channel waits for synchronization. If the channel is manually synchronized, it will properly complete the frame, but will immediately begin the transfer of the next frame, too. This behavior occurs for both a software pause (setting START = 10b) and for an emulation halt (with EMOD = 1). (Internal reference number C601257)

Workaround: If pausing the DMA channel in software, do the following to restart:

1. Set the RSYNC bit in the Secondary Control Register.
2. Read the Transfer Count Register and then write back to Transfer Count Register. This enables the present frame to transfer but waits for the next sync event to trigger the next frame transfer.
3. Set START to 01b or 11b.

If pausing the DMA channel with an emulation halt, do the following to restart:

Workaround(s):

1. Double-click on the Transfer Count Register and hit enter (rewrite current transfer count).
2. Set the RSYNC STAT bit in the Secondary Control Register (change 0xXXXX4XXX to 0xXXXX1XXX).
3. Run.

NOTE: The order of 1 & 2 is critical for an emulator halt (EMOD = 1), but not for the software pause.

Advisory 0.0.4

DMA Multiframe Split-Mode Transfers Source Address Indexing Not Functional

Revision(s) Affected: Rev 0.0

Details: If a DMA channel is configured to do a multiframe split-mode transfer with SRC_DIR = Index (11b), the source address is always modified using the Element Index, even during the last element transfer of a frame. The transfer of the last element in a frame should index the source address using the Frame Index instead of the Element Index. DST_DIR = 11b functions properly. (Internal reference number C601256)

Workaround: For multiframe transfers, use two DMA channels instead of using the split-mode. Source Index works properly for non-split-mode transfers.

Advisory 0.0.5 — *DMA: Issues When a DMA Channel is Paused at a Block Boundary*

Revision(s) Affected: Rev 0.0

Details: The following problems exist when a DMA channel is paused at a block boundary:

- DMA does not flush internal FIFO when a channel is paused across block boundary. As a result, data from old and new blocks of that channel are in FIFO simultaneously. This prevents other channels from using the FIFO for high performance until that channel is restarted. Data is not lost when that channel is started again. (Internal reference number C601299)
- For DMA transfers with auto–initialization, if a channel is paused just as the last transfer in a block completes (just as the transfer counter reaches zero), none of the register reloads take place (count, source address, and destination address). When the same channel is restarted, the channel will not transfer anything due to the zero transfer count. This problem only occurs at block boundaries. (Internal reference number C601258)

Workaround: Do not pause across block boundary if the internal FIFO is to be used by other channels for high performance. For DMA transfers with auto–initialization, if a channel is paused with a zero transfer count, manually reload all registers before restarting the channel.

Advisory 0.0.6 — *DMA: Stopped Transfer Reprogrammed Does Not Wait for Sync*

Revision(s) Affected: Rev 0.0

Details: If any non-synchronized transfer (ex: Auto-init Transfer) is stopped, and then the same channel is programmed to do a Write Synchronized Transfer (ex: Split–mode transfer), the write transfer does not wait for the Sync event. (Internal reference number C601261)

Perform a non-synchronized dummy transfer of one element to/from the same location before starting the synchronized transfer.

Advisory 0.0.7 — *DMA Freezes if Post–increment/Decrement Across Port Boundary*

Revision(s) Affected: Rev 0.0

Details: For any DMA transfers with source/dest address post–increment/decrement, if the last element to be transferred is aligned on a port boundary, then the DMA may freeze before transferring this element. A port boundary is the address boundary between external memory and program memory, between external memory and the peripheral address space, or between program memory and the peripheral address space.

DMA Freezes if Post–increment/Decrement Across Port Boundary (Continued)

The following conditions cause DMA to freeze:

- For non–sync and frame–sync transfers: if a channel is paused after the second–to–last element is read, when the channel is then restarted with a request to the address at a port boundary the DMA will freeze.
- For split–mode transfers or read/write–sync transfers: the DMA will freeze while transferring the element aligned on the port boundary. A continuous burst transfer with post–increment/decrement source/dest address does not exhibit this problem. Internal reference number C601300.

Workaround: Do not transfer to boundary addresses if the DMA source/dest address is post–incremented/decremented.

Advisory 0.0.8 — *DMA Paused During Emulation Halt*

Revision(s) Affected: Rev 0.0

Details: When running an auto-initialized transfer, the DMA write state machine is halted during an emulation halt regardless of the value of EMOD in the DMA Channel Primary Control Register. The read state machine functions properly in this case. The problem exists only at block boundaries. If EMOD=1, this problem is irrelevant since the DMA channel is expected to pause during an emulation halt. Internal reference number C601301.

Workaround: There is no workaround for EMOD=0. Expect DMA transfers to pause when the emulator stops the processor.

Advisory 0.0.9 — *DMA: RSYNC=10000b (DSPINT) Does Not Wait for Sync*

Revision(s) Affected: Rev 0.0

Details: If RSYNC in the DMA Channel Primary Control Register is set to Host-port host to DSP interrupt (DSPINT – 10000b), the DMA channel does the read transfer without waiting for the sync event. There is not a problem if WSYNC is set to DSPINT. (Internal reference number C601302)

Workaround: Do not synchronize DMA reads to DSPINT. If a DMA read is desired during a Host–port host to DSP interrupt, set RSYNC in the Primary Control Register to one of the EXT_INT events instead (EXT_INT4 – EXT_INT7) and have the host trigger an interrupt on that pin rather then by writing to HPIC.

Advisory 0.0.10

CPU: L–unit Interprets Some Integer Instructions as Double Precision Floating Point Instructions

Revision(s) Affected: Rev 0.0

Details:

The floating point .L unit incorrectly interprets an integer instruction as a double precision floating-point instruction. As a result, the .L unit fails to execute a floating-point instruction that follows. The following set of .L unit integer instructions may result in a subsequent .L unit floating point instruction failing to execute:

INTEGER .L UNIT INSTRUCTION	INTERPRETED BY FLOATING POINT .L UNIT AS
CMPGT (all opfields)†	ADDDP / SUBDP
CMPGTU (all opfields)†	ADDDP / SUBDP
SAT	ADDDP / SUBDP

† Code can be written so that the CMPGT and CMPGTU opcodes are not obvious. CMPLT/CMPLTU and CMPGT/CMPGTU are pseudo operations of each other, in the case when the operands are incorrectly arranged. For example, for the piece of code below:

```
CMPLT    .L1x B1,A0,A0 ; src1 should not use the cross path, pseudo-op will
                       ; be substituted
CMPGT    .L1x A1,8,A1  ; only src1 can be a constant, pseudo-op will be
                       ; substituted
```

The assembler leaves the instructions above in the list file (.lst), but performs the following operations instead:

```
CMPGT    .L1x A0,B1,A0 ; only src2 uses the cross path
CMPLT    .L1x 8,A1,A1  ; only src1 could be a constant
```

When determining which int/fp instruction scenarios will result in a floating point failure, treat the integer instruction as if it were the floating point instruction specified in the table above and refer to Table 6–15 in the *TMS320C62x/C67x CPU and Instruction Set Reference Guide*. When applying the rules of the hazard table, note that it is only possible for a subsequent same-unit floating-point instruction to fail.

An example failing code sequence is:

```
LDH      .D1T1        *+A7(2),A5
||       CMPGT        .L1    A4,A0,A4
||       SUB          .L2x   B0, A5, B5
         XOR          .S1    1,A4,A4
||       INTSP        .L2    B5,B4
||       INTSP        .L1    A5,A7; failing instruction
```

The failure occurs on INTSP.L1, because the .L1 FP unit is still busy executing the false ADDDP triggered by the CMPGT.L1 executed in the previous cycle. Internal reference number 4.

Workaround: For any code sequences with an integer CMPGT/CMPGTU/SAT in the first execute packet and a floating point operation to the same .L unit in the second execute packet, ensure that if the integer instruction were treated as an ADDDP/SUBDP instruction, the second execute packet would not encounter any hazards as outlined in Table 6–15 of the *TMS320C62x/C67x CPU and Instruction Set Reference Guide*.

Advisory 0.0.11

CPU: S-Unit Interprets Some Integer Instructions as Double Precision Floating Point Instructions

Revision(s) Affected: Rev 0.0

Details: The .S unit instruction decode block incorrectly instructs the floating-point pipeline to perform a double precision floating-point operation as the result of an integer instruction. A subsequent floating-point instruction to the same .S unit, then fails to execute. The following set of .S unit integer instructions may cause a subsequent .S unit floating-point instruction to fail to execute:

SHR (opfields 110110 or 110100)

CLR (opfields 11 or 111111)

EXT (constant form or register form)

If the SHR/CLR/EXT instruction is not followed by any other non–SHR/CLR/EXT integer instruction to the same .S unit, a subsequent floating-point instruction to the same .S unit may fail to execute. This applies even if more than one execute packet exists between the SHR/CLR/EXT instruction and the floating-point instruction.

An example failing code sequence is:

```
         ADD      .L2    4, B5, B5
|| [ A2] STW      .D2T2  B6,*B3
|| [ A1] SUB      .D1A1,1, A1
||       SHR      .S1    A6, 14, A8

         NOP      1
         ADD      .L2    B9, B4, B4
||       MPYU     .M1    A8, A3, A8

         ADDSP    .L1    A4, A5, A5
||       CMPLTSP  .S1    A5, A9, A2     ; failing instruction
||       LDW      .D1T1  *++A0,A9
||       SHR      .S2    B4, 14, B8
```

The failure occurs on CMPLTSP.S1, because the .S1 FP unit is still busy executing the false floating point instruction triggered by the SHR.L1 executed previously. (Internal reference number 5)

CPU: S-Unit Interprets Some Integer Instructions as Double Precision Floating Point Instructions (Continued)

Workaround: Ensure that the above three forms of .S unit integer operations (SHR, CLR, EXT) are followed by any other .S unit integer operation BEFORE executing an .S unit floating–point operation on that particular .S unit. Note that a NOP instruction does not count as a non–SHR/CLR/EXT instruction.

Advisory 0.0.12 — *CPU: MPYSP/MPYDP Underflow Failure*

Revision(s) Affected: Rev 0.0

Details: In some cases the floating point .M unit produces an incorrect destination result for MPYSP and MPYDP instructions which underflow.

If each of the following conditions is true, an MPYSP or MPYDP instruction may deliver an incorrect destination result:

1. The expected result of an MPYSP or MPYDP instruction underflows.
2. The expected destination result is +/–SFPN.

The .M unit incorrectly produces an exponent equal to Emax instead of the expected Emin. The fraction, sign, and UNDER status bits are correct. If the instruction underflows and should produce a destination result of +/–0 instead of +/–SFPN, then the result produced is correct. Internal reference number 8.

Workaround: Do not use MPYSP or MPYDP for numbers that may generate an underflow.

Advisory 0.0.13 — CPU: DPSP Underflow Failure

Revision(s) Affected: Rev 0.0

Details:

In some cases the floating point .L unit produces an incorrect result for DPSP instructions which underflow. Internal reference number 6, 10.

CASE1: If each of the four following conditions is true, a DPSP instruction may deliver an incorrect destination result and incorrect INEX and UNDER status bit (in the Floating-Point Multiplier Configuration Register) results:

1. the expected result underflows
2. the intermediate result fraction is incremented due to rounding
3. the pre-rounded intermediate result exponent is non–zero
4. Rmode is not 01 (truncate)

An example code/data sequence that will generate this error is:

```
MVK    0x17373ff5, A8
MVKH   0x17373ff5, A8
MVK    0x2f35e46b, A9
MVKH   0x2f35e46b, A9
NOP
NOP
NOP
NOP
DPSP   .L1     A9:A8, A5          ;A5 = 39af2359 (should be 00000000)
NOP
```

CASE 2: If each of the four following conditions is true, a DPSP instruction will deliver an incorrect destination result and incorrect UNDER and OVER status bit results:

1. The expected result underflows.
2. The intermediate result fraction is incremented due to rounding.
3. Rounding causes a carry out of the incremented intermediate result fraction.
4. The intermediate result exponent (calculated as the source operand's biased exponent minus 0x380) has a value of '1111111x' (in binary).

The delivered result incorrectly reflects an overflow condition, and not an underflow condition as expected.

CPU: DPSP Underflow Failure (Continued)

Example

Before instruction: Round mode = 0 (round toward nearest even integer)
A1:A0 = 07ffffff ff800000
Failing instruction:
DPSP A1:A0, A2
Incorrect result: A2 = 0x7f800000 with OVER and INEX
(Expected A2 = 0x00000000 with UNDER and INEX)

Workaround: If conversion results may underflow, disable rounding mode by setting Rmode = 01 (truncate). Do not use DPSP if the above conditions are true.

Advisory 0.0.14 — *CPU: DPTRUNC/DPINT Overflow Failure*

Revision(s) Affected: Rev 0.0

Details: In some cases the floating point .L unit produces an incorrect result for DPINT and DPTRUNC instructions which overflow. Internal reference number 9.

If each of the three following conditions is true then a DPINT or DPTRUNC instruction may deliver an incorrect destination result and incorrect INEX and OVER status bit results:

1. The source operand has a negative sign.
2. The source operand has a biased exponent equal to 1055 (0x41f) causing the expected result to overflow.
3. The intermediate result fraction is not rounded (DPTRUNC always meets this condition).
4. Rmode=10 (round up) for DPINT instruction. This is irrelevant for DPTRUNC instruction.

Example 1

Before instruction: A1:A0 = 0xc1f232bf 7321a000
Failing instruction: DPTRUNC .L1 A1:A0, A2
Incorrect result: A2 = 0xdcd408ce (should be A2 = 0x80000000)

Example 2

Before instruction: FADCR = 0x00000400
A1:A0 = 0xc1f4775a 6d3fc000
Failing instruction: DPINT .L1 A1:A0, A2
Incorrect result: A2 = 0xb88a592d (should be A2 = 0x80000000)

Workaround: Do not use DPINT or DPTRUNC if the above four conditions are true.

Advisory 0.0.15

CPU: L-unit Floating Point Instructions Failed to Execute After ADDDP/SUBDP Re-execution

Revision(s) Affected: Rev 0.0

Details:

The floating point .L unit incorrectly interprets an integer instruction and as a result re–executes a preceding double precision floating point instruction, causing a subsequent floating point instruction to fail.

If the following sequence occurs:

1. An ADDDP or SUBDP is executed on an .L unit.
2. Two execute packets later, any of the following integer .L unit instructions is executed in the SAME .L unit: AND, OR, LMBD, NORM, CMPLT, SADD, CMPEQ, or ABS. (See “instr X” in the example below.)
3. Two execute packets later after (2), a non-ADDDP/SUBDP floating-point instruction is executed in the SAME .L unit. (See “failing instruction” in the example below.)

Then the final floating point instruction will fail to execute correctly. An example of this failure is shown below:

```
        SUBDP   .L1           A13:A12, A9:A8, A3:A2

        NOP

        AND     .L1           6, A8, A5       ; instr X
||      STW     .D1           A2, *A15++
||      MV      .S2X          A3, B3

        AND     .L1           A9, A12, A6     ; instr Y
||      STW     .D1           A4, *A15++
||      MV      .S2X          A5, B5

||      STW     .D2           B3, *B15++
        SPINT   .L1           A8, A4          ; failing instruction
```

In the above code sequence, instr X must be one of the previously defined eight integer .L unit instructions for the failure to occur, instr Y can be any .L unit instruction. Any number of X/Y instr pairs can exist between the SUBDP and SPINT. Internal reference number 7.

Workaround: Insert an additional NOP between the SUBDP and instr X.

4 Documentation Support

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

To access documentation on the web site:

1. Go to **http://www.ti.com**
2. Open the "**Products & Services**" dialog box and choose "**DSP**"
3. Scroll to the "**TMS320C6000™**" and click on "**Product List**"
4. Click on a device name and then click on the documentation type you prefer.

TMS320C6000 and TMS320 are trademarks of Texas Instruments.

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgment, including those pertaining to warranty, patent infringement, and limitation of liability.

TI warrants performance of its products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Customers are responsible for their applications using TI components.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, license, warranty or endorsement thereof.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations and notices. Representation or reproduction of this information with alteration voids all warranties provided for an associated TI product or service, is an unfair and deceptive business practice, and TI is not responsible nor liable for any such use.

Resale of TI's products or services with *statements different from or beyond the parameters* stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service, is an unfair and deceptive business practice, and TI is not responsible nor liable for any such use.

Also see: Standard Terms and Conditions of Sale for Semiconductor Products. www.ti.com/sc/docs/stdterms.htm

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265