TMS320C6474
Digital Signal Processor
Silicon Revisions 1.3, 1.2

Silicon Errata

Literature Number: SPRZ283
October 2008

Contents

1 Introduction 5

1.1 Device and Development Support Tool Nomenclature 5

1.2 Package Symbolization and Revision Identification 6

2 Silicon Revision 1.3 Known Design Exceptions to Functional Specifications 7

3 Silicon Revision 1.2 Known Design Exceptions to Functional Specifications 22

List of Figures

1 Lot Trace Code Examples for TMS320C6474 (ZUN Package) 6
2 IDMA, SDMA, and MDMA Paths 9
3 IDMA, SDMA, and MDMA Paths 24
4 Correct Device Input Clocks, Clock Selects, and Scaled Supply Timings 36
5 Prog Set Options Register 39

List of Tables

1 Lot Trace Codes 6
2 Silicon Revision Variables 6
3 Silicon Revision 1.3 Advisory List 7
4 Silicon Revision 1.2 Advisory List 22
5 TC Registers Summary 39

TMS320C6474 DSP Silicon Revisions 1.3, 1.2

1 Introduction

This document describes the silicon updates to the functional specifications for the TMS320C6474 digital signal processor; see the device-specific data manual, *TMS320C6474 Multicore Digital Signal Processor* (literature number SPRS552).

1.1 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 (e.g., TMS320C6474ZUN). 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.

1.2 Package Symbolization and Revision Identification

The device revision can be determined by the lot trace code marked on the top of the package. The location of the lot trace code for the ZUN package is shown in Figure 1. Figure 1 also shows an example of C6474 package symbolization.

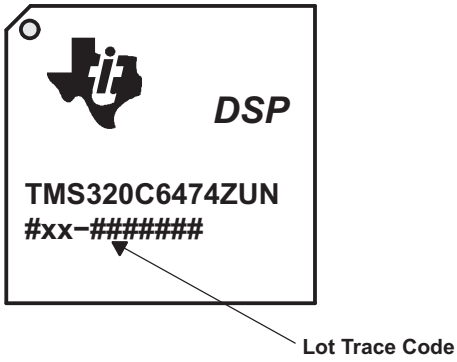


Figure 1. Lot Trace Code Examples for TMS320C6474 (ZUN Package)

Silicon revision correlates to the lot trace code marked on the package. This code is of the format #xx-#######. If xx is "12", then the silicon is revision 1.2. Table 1 lists the silicon revisions associated with each lot trace code for the C6474 device.

Each silicon revision uses a specific revision of the CPU and the C64x+ megamodule. The CPU revision ID identifies the silicon revision of the CPU. Table 2 lists the CPU and C64x+ megamodule revision associated with each silicon revision. The CPU revision can be read from the REVISION_ID field of the CPU control status register (CSR). The C64x+ megamodule revision can be read from the REVISION field of the megamodule revision ID register (MM_REVID) located at address 0181 2000h.

The VARIANT field of the JTAG ID register (located at 0288 0814h) changes between silicon revisions. Table 2 lists the contents of the JTAG ID register for each revision of the device. More details on the JTAG ID register can be found in the device-specific data manual, *TMS320C6474 Multicore Digital Signal Processor* (literature number SPRS552).

Table 1. Lot Trace Codes

LOT TRACE CODE (xx)	SILICON REVISION	COMMENTS
13	1.3	Silicon revision 1.3
12	1.2	Silicon revision 1.2

Table 2. Silicon Revision Variables

SILICON REVISION	CPU REVISION	C64X+ MEGAMODULE REVISION	JTAG ID REGISTER VALUE
1.3	1.0 (REVISION_ID = 0h)	Rev. 0 (MM_REVID[REVISION] = 0h)	0x2009 202Fh (VARIANT = 0010b)
1.2	1.0 (REVISION_ID = 0h)	Rev. 0 (MM_REVID[REVISION] = 0h)	0x1009 202Fh (VARIANT = 0001b)

2 Silicon Revision 1.3 Known Design Exceptions to Functional Specifications

Table 3. Silicon Revision 1.3 Advisory List

Title	Page
Advisory 1.3.1 DSP SDMA/IDMA: Unexpected Stalling of SDMA/IDMA Access to L2 SRAM	8
Advisory 1.3.2 Potential Data Corruption on SCR Bridge	15
Advisory 1.3.3 Potential Insertion or Deletion of 2 Bits in SerDes Data Stream	16
Advisory 1.3.4 MAC EOI Register Write Causes Potential CPU Lockup	18
Advisory 1.3.5 Potential SerDes Clocking Issue	19
Advisory 1.3.6 I2C: Slave Boot Aborts	20
Advisory 1.3.7 EMAC Boot Issue	21

Advisory 1.3.1 *DSP SDMA/IDMA: Unexpected Stalling of SDMA/IDMA Access to L2 SRAM*

Revision(s) Affected: 1.3, 1.2

Details:

Note: Only when DSP level 2 (L2) memory is configured as non-cache (RAM), unexpected stalling may occur on DSP SDMA/IDMA accesses. If DSP L2 memory is used only as cache ***or*** if L2 RAM is ***not*** accessed by IDMA or via the SDMA interface during run-time, then this exception does not apply.

The C64x+ megamodule has a Master Direct Memory Access (MDMA) bus interface and a Slave Direct Memory Access (SDMA) bus interface. The MDMA interface provides DSP access to resources outside the C64x+ megamodule (i.e., DDR2 memory). The MDMA interface is used for CPU/cache accesses to memory beyond the level 2 (L2) memory level. These accesses include cache line allocates, write-backs, and non-cacheable loads and stores to/from system memories. The SDMA interface allows other master peripherals in the system to access level 1 data (L1D), level 1 program (L1P), and L2 RAM DSP memories. The masters allowed accesses to these memories are DMA controllers, EMAC, and SRIO. The DSP Internal Direct Memory Access (IDMA) is a C64x+ megamodule DMA engine used to move data between internal DSP memories (L1, L2) and/or the DSP peripheral configuration bus. The IDMA engine shares resources with the SDMA interface.

The C64x+ megamodule has an L1D cache and an L2 caches, both of which implement write-back data caches. The C64x+ megamodule holds updated values for external memory as long as possible. It writes these updated values, called *victims*, to external memory when it needs to make room for new data, when requested to do so by the application, or when a load is performed from a non-cacheable memory for which there is a set match in the cache (i.e., the non-cacheable line would replace a dirty line if cached). The L1D sends its victims to L2. The caching architecture has pipelining, meaning multiple requests could be pending between L1, L2, and MDMA. For more Details: on the C64x+ megamodule and its MDMA and SDMA ports, see the *TMS320C64x+ Megamodule Reference Guide* (literature number SPRU871).

Ideally, the MDMA (the blue lines in Figure 2) and SDMA/IDMA paths (the orange lines in Figure 2) operate independently with minimal interference. Normally, MDMA accesses may stall for extended periods of time (clock cycles) due to expected system level delays (e.g., bandwidth limitations, DDR2 memory refreshes). However, when using L2 as RAM, SDMA and/or IDMA accesses to L2/L1 may experience unexpected stalling in addition to the normal stalls seen by the MDMA interface. For latency-sensitive traffic, the SDMA stall can result in missing real-time deadlines.

Note: SDMA/IDMA accesses to L1P/D will not experience an unexpected stall if there are no SDMA/IDMA accesses to L2. Unexpected SDMA/IDMA stalls to L1 happen only when they are pipelined behind L2 accesses.

Figure 2 is a simplified view for illustrative purposes only. The IDMA/SDMA path (orange lines) can also go to L1D/L1P memories and IDMA can go to the DSP CFG peripherals. MDMA transactions (blue lines) can also originate from L1P or L1D through the L2 controller or directly from the DSP.

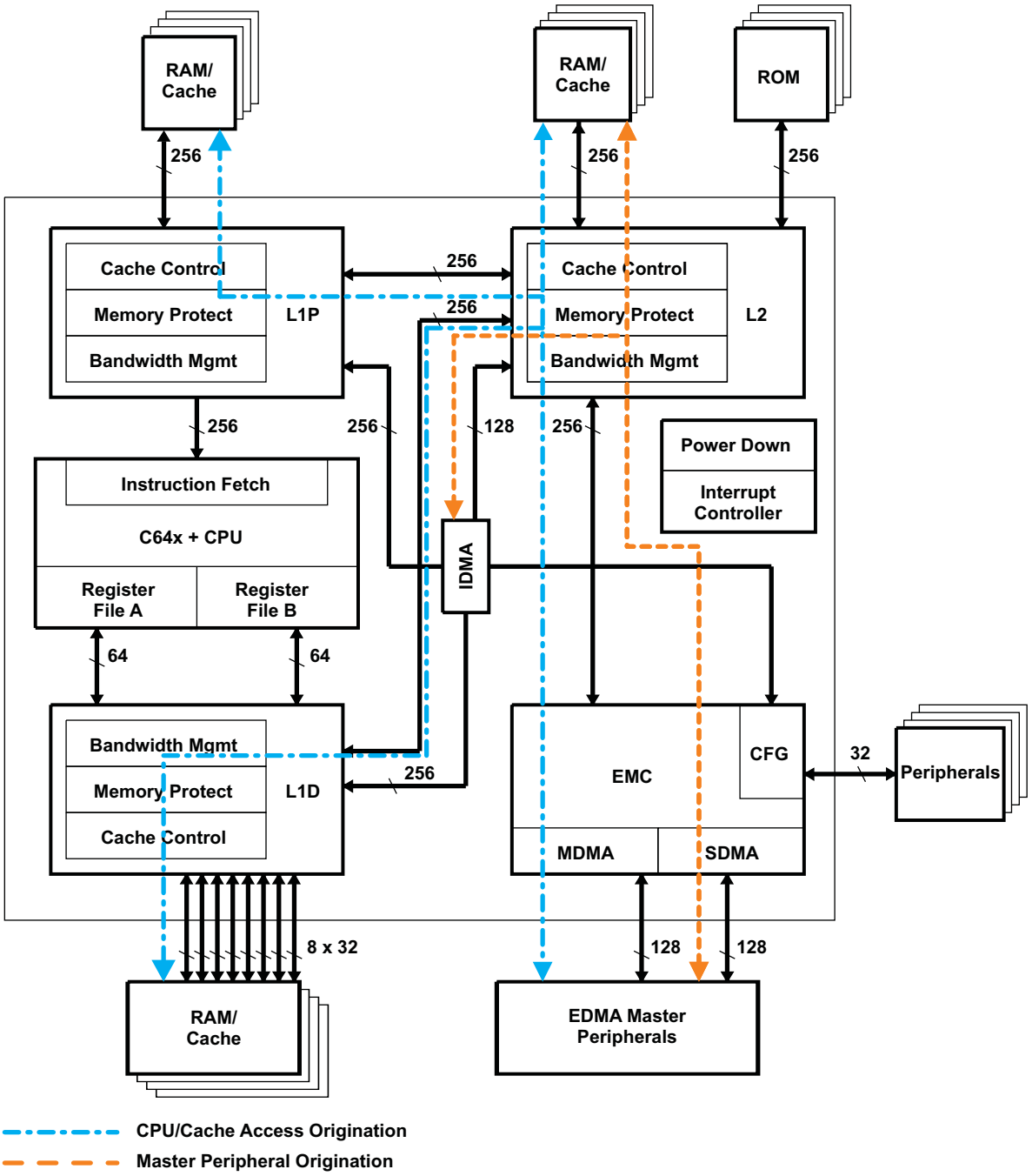


Figure 2. IDMA, SDMA, and MDMA Paths

SDMA/IDMA stalls may occur during the following scenarios. Each of these scenarios describes expected normal DSP functionality, but the SDMA/IDMA access potentially exhibits additional unexpected stalling.

1. Bursts of writes to non-cacheable MDMA space (i.e., DDR2). The DSP buffers up to 4 non-cacheable writes. When this buffer fills, SDMA/IDMA is blocked until the buffer is no longer full. Therefore, bursts of non-cacheable writes longer than three writes can stall SDMA/IDMA traffic.
2. Various combinations of L1 and L2 cache activity:
   a. L1D read miss generating victim traffic to L2 (cache or SRAM) or external memory. The SDMA/MDMA may be stalled while servicing the read miss and the victim. If the read miss also misses L2 cache, the SDMA/IDMA may be stalled until data is fetched from external memory to service the read miss. If the read access is to non-cacheable memory there will still potentially be an L1D victim generated even though the read data will not replace the line in the L1D cache.
   b. L1D read request missing L2 (going external) while another L1D request is pending. The SDMA/IDMA may be stalled until the external memory access is complete.
   c. L2 victim traffic to external memory during any pending L1D request. The SDMA/IDMA may be stalled until external memory access and the pending L1D request are complete.

The duration of the SDMA/IDMA stalls depends on the quantity/characteristics of the L1/L2 cache and the MDMA traffic in the system. In cases 2a, 2b, and 2c, stalling may or may not occur depending on the state of the cache request pipelines and the traffic target locations. These stalling mechanisms may also interact in various ways, causing longer stalls. Therefore, it is difficult to predict if stalling will occur and for how long.

SDMA/IDMA stalling and any system impact is most likely in systems with excessive context switching, L1/L2 cache miss/victim traffic, and heavily loaded EMIF.

Use the following steps to determine if SDMA/IDMA stalling is the cause of real-time deadline misses for existing applications. Situations where real-time deadlines may be missed include loss of McBSP samples and low peripheral throughput.

1. Determine if the transfer missing the real-time deadline is accessing L2 or L1D memory. If not, then SDMA/IDMA stalling is not the source of the real-time deadline miss.
2. Identify all SDMA transfers to/from L2 memory (e.g., EDMA transfer to/from L2 from/to a McBSP or from/to AIF, TCP, or VCP). If there are no SDMA transfers going to L2, then SDMA/IDMA stalling is not the source of the problem.
3. Redirect all SDMA transfers to L2 memory to other memories using one of the following methods:
   - Temporarily transfer all the L2 SDMA transfers to L1D SRAM.
   - If not all L2 SDMA transfers can be moved to L1D memory, temporarily direct some of the transfers to DDR memory and keep the rest in L1D memory. There should be ***no*** L2 SDMA transfers.
   - If neither of the above approaches are possible, move the transfer with the real-time deadline to the EMAC CPPI RAM. If the EMAC CPPI RAM is not big enough, a two-step mechanism can be used to page a small working buffer defined in the EMAC CPPI RAM into a bigger buffer in L2 SRAM. The EDMA module can be setup to automate this double buffering scheme without CPU intervention for moving data from the EMAC CPPI RAM. Some throughput degradation is expected when the buffers are moved to the EMAC CPPI RAM.

     Note: Note that EMAC CPPI RAM memory is word-addressable only and, therefore, must be accessed using an EDMA index of 4 bytes.

If real-time deadlines are still missed after implementing any of the options in Step 3, then SDMA/IDMA stalling is likely ***not*** the cause of the problem. If real-time deadline misses are solved using any of the options in Step 3, then SDMA/IDMA stalling ***is*** likely the source of the problem.

An extreme manifestation of the IDMA/SDMA stall bug is the C64x+ MDMA-SDMA deadlock that requires a device reset or power-on reset in order for the system to recover. The following summarizes the deadlock conditions:

- Master(s) on a single main MSCR port write to the GEM's SDMA followed by a write to slaveX
- The GEM issues victim traffic or a non-cacheable write to slaveX
- Any one of the following:
  - A write data path pipelined in main MSCR between master(s) and the GEM's SDMA
  - A bridge exists between master(s) and the main MSCR
  - Master(s) are able to issue a command to slaveX concurrent with the write to the GEM's SDMA.

A load (either cacheable or non-cacheable) from another core's L1D or L2 memory can additionally create a deadlock condition. When the load is issued the read command is propagated to the SDMA port of the other core through a bridge that is shared with the EDMA TC1, EMAC, RapidIO (both data and CPPI), and other GEM MDMA. When the load is issued, if a victim is generated in L1D cache, then the SDMA may stall until the load completes. If other masters are issuing commands through the shared bridge, then the bridge may fill due to the stalled SDMA before the read command can propagate through the bridge and complete. In summary, a deadlock can occur if the following is true:

- GEMx issues a read to GEMy or GEMz L1D or L2 SRAM
- Any of the following are issuing commands to GEMx L2: TC1, EMAC (data or CPPI), RapidIO (data or CPPI), GEMy, or GEMz.

Workarounds:

Method 1

Issues such as dropped McBSP samples can be worked around by moving latency-sensitive buffers outside the C64x+ megamodule. For example, rather than placing buffers for the McBSP into L1/L2, those buffers can instead be placed in other memory, such as the EMAC CPPI RAM.

Note: Note that EMAC CPPI RAM memory is word-addressable only and, therefore, must be accessed using an EDMA index of 4 bytes.

Method 2

To reduce the SDMA/IDMA stalling system impact, perform any of the following:

1. Improve system tolerance on DMA side (SDMA/IDMA/MDMA):
   - Understand and minimize latency-critical SDMA/IDMA accesses to L2 or L1P/D.
   - Directly reduce critical real-time deadlines, if possible, at peripheral/IO level (e.g., increase word size and/or reduce bit rates on serial ports).
   - To reduce DSP MDMA latency:
     – Increase the priority of the DSP access to DDR2 such that MDMA latency of MDMA accesses causing stalls is minimized.
       Note: Other masters may have real-time deadlines that dictate higher priority than the DSP.
     – Lower the PRIO_RAISE field setting in the DDR2 memory controller's burst priority register. Values ranging between 0x10 and 0x20 should give decent performance and minimize latency; lower values may cause excessive SDRAM row thrashing.
2. Minimize offending scenarios on DSP/caching side:
   - If the DSP performing non-cacheable writes is causing the issue, insert protected non-cacheable reads (as shown in the last list item below) every few writes to allow the write buffer to empty.
   - Use explicit cache commands to trigger cache writebacks during appropriate times (L1D Writeback All, L2 Writeback All). ***Do not*** use these commands when real-time deadlines must be met.
   - Restructure program data and data flow to minimize the offending cache activity.
     – Define the read-only data as *const*. The const C keyword tells the compiler not to write to the array. By default, such arrays are allocated to the .const section as opposed to BSS. With a suitable linker command file, the developer can link the .const section off chip, while linking .bss on chip. Because programs initialize .bss at run time, this reduces the program's initialization time and total memory image.
     – Explicitly allocate lookup tables and writeable buffers to their own sections. The #pragma DATA_SECTION (label, *section*) directive tells the compiler to place a particular variable in the specified COFF section. The developer can explicitly control the layout of the program with this directive and an appropriate linker command file.
     – Avoid directly accessing data in slow memories (e.g., flash); copy at initialization time to faster memories.
   - Modify troublesome code.
     – Rewrite using DMAs to minimize data cache writebacks. If the code accesses a large quantity of data externally, consider using DMAs to bring in the data, using double buffering and related techniques. This will minimize cache write-back traffic and the likelihood of SDMA/IDMA stalling.
     – Re-block the loops. In some cases, restructuring loops can increase reuse in the cache and reduce the total traffic to external memory.
     – Throttle the loops. If restructuring the code is impractical, then it is reasonable to slow it down. This reduces the likelihood that consecutive SDMA/IDMA blocks stack up in the cache request pipelines, resulting in a long stall.

- Protect non-cacheable reads from generating an SDMA stall by freezing the L1D cache during the non-cacheable read access(es). The following example code contains a function that protects non-cacheable reads, avoids blocking during the reads, and, therefore, avoids the deadlock state.

```
;; ======================================================================== ;;
;;  Long Distance Load Word                                                 ;;
;;                                                                          ;;
;;      int long_dist_load_word(volatile int *addr)                         ;;
;;                                                                          ;;
;;  This function reads a single word from a remote location with the L1D   ;;
;;  cache frozen.  This prevents L1D from sending victims in response to    ;;
;;  these reads, thus preventing the L1D victim lock from engaging for the  ;;
;;  corresponding L1D set.                                                  ;;
;;                                                                          ;;
;;  The code below does the following:                                      ;;
;;                                                                          ;;
;;      1.  Disable interrupts                                              ;;
;;      2.  Freeze L1D                                                      ;;
;;      3.  Load the requested word                                         ;;
;;      4.  Unfreeze L1D                                                    ;;
;;      5.  Restore interrupts                                              ;;
;;                                                                          ;;
;;  Interrupts are disabled while the cache is frozen to prevent affecting  ;;
;;  the performance of interrupt handlers.  Disabling interrupts during     ;;
;;  the long distance load does not greatly impact interrupt latency,       ;;
;;  because the CPU already cannot service interrupts when it's stalled by  ;;
;;  the cache.  This function adds a small amount of overhead (~20 cycles)  ;;
;;  to that operation.                                                      ;;
;;                                                                          ;;
;; ======================================================================== ;;

        .asg    0x01840044,     L1DCC           ; L1D Cache Control
        .global _long_dist_load_word
        .text
        .asmfunc
; int long_dist_load_word(volatile int *addr)
_long_dist_load_word:
        MVKL    L1DCC,          B4
        MVKH    L1DCC,          B4
||      DINT                                    ; Disable interrupts
||      MVK     1,              B5
        STW     B5,             *B4             ; \_ Freeze cache
        LDW     *B4,            B5              ; /
        NOP     4
        SHR     B5,     16,     B5              ; POPER -> OPER
||      LDW     *A4,            A4              ; read value remotely
        NOP     4
        STW     B5,             *B4             ; \_ Restore cache
        RET     B3
||      LDW     *B4,            B5              ; /
        NOP     4
        RINT                                    ; Restore interrupts
        .endasmfunc

;; ======================================================================== ;;
;;  End of file: ldld.asm                                                   ;;
;; ======================================================================== ;;
```

In the C6474 multicore device, when one GEM is accessing another GEM's L1 or L2 memory it is an MDMA access, so the potential SDMA/IDMA stall can occur. The stall can be avoided by using the EDMA to transfer data from one GEM's memory to another.

Method 3

Entirely eliminate the exception by removing all SDMA/IDMA accesses to L2 SRAM. For example, EMAC descriptors and EMAC payload cannot reside in L2. Master peripherals like the EDMA/QDMA, IDMA, and SRIO cannot access L2. There are no issues with the CPU itself accessing code/data in L2. This issue only pertains to SDMA/IDMA accesses to L2.

Deadlock Avoidance

To avoid the manifestation of a C64x+ deadlock, several Workarounds: are suggested depending on the VBUSM master in question:

VBUSM MASTER	WORKAROUND
GEM	GEMs should not write to the memory of any other GEM. This will cause complications across any master peripheral that is transferring data to multiple L2s. GEMs must not directly read from the memory of any other GEMs without providing the L1D cache disable workaround mentioned in Method 2 to ensure that the load will not stall itself indefinitely and hang the system.
EDMA3TCx	Inbound and outbound traffic should be programmed on different TC ports (i.e., two different EDMA queues, since a given queue maps to a given TC). Note that in-/out-bound direction is defined as the write direction, meaning that a DDR2-to-DDR2 transfer is outbound and L2-to-L2 is inbound. Any TC used to write to DDR should not be used to write to a GEM even when the TC writing to the DDR is also reading from DDR.
EMAC	EMAC should write to the GEM's memory or the DDR, but not both. This includes buffers and buffer descriptors. EMAC CPPI descriptors should be placed wholly in the local wrapper memory, any combination of wrapper and L2 memory (must match other master transactions), or any combination of wrapper and DDR2 SDRAM (must match other master transactions). Buffer descriptors should not be placed in any combination of L2 and DDR2 SDRAM.
SRIO	SRIO should transfer payload data to only GEM memories or to DDR2 SDRAM, but not both. This includes any direct I/O writes as well as any inbound RX messaging transfer.
SRIO CPPI	SRIO CPPI descriptors should be placed wholly in the local wrapper memory, any combination of wrapper and L2 memory, or any combination of wrapper and DDR2 SDRAM. Buffer descriptors should not be placed in any combination of L2 and DDR2 SDRAM.

Advisory 1.3.2 *Potential Data Corruption on SCR Bridge*

Revision(s) Affected: 1.3, 1.2

Details: This issue manifests itself when two masters write to a bridge endpoint and the commands arrive on the same clock cycle. The VBUS protocol is violated and the data is corrupted. The consequence is that one of the writes goes through with corrupt data, the other completes normally.

On some occasions the bridge may not recover without a reset. There is no software indication of this nor a means to reset only the bridge. Therefore, the situation must be avoided. The affected bridges are: TCP, VCP, AIF write port, and DMA bridge to configuration bus (where key endpoints beyond this bus could include Semaphore configuration port, EMAC configuration ports, PaRAM configuration port, or SRIO configuration ports).

Workarounds: Corrective action is taken by avoiding the issue as described in the following:

- TCP: Access to R/W ports is controlled by the Semaphore module; no issue.
- VCP: Access to R/W ports is controlled by the Semaphore module; no issue.
- AIF: DMA access is from a single transfer controller (TC); no issue. If more than one TC is used, the issue will be exposed.
- DMA bridge to configuration bus: Dedicate a single TC for use of the DMA to write through this bridge to all the endpoints beyond or use the configuration bus directly and do not use the DMA to program the following:
  - Semaphore configuration port: Configuration registers
  - SRIO configuration ports: Configuration registers
  - EMAC configuration ports: Configuration registers (caution on using EMAC CPPI buffer to work around the DSP SDMA/IDMA unexpected stalling, see Advisory 1.3.1)
  - PaRAM configuration port: Do not auto-program channels from one TC to another

Advisory 1.3.3 *Potential Insertion or Deletion of 2 Bits in SerDes Data Stream*

Revision(s) Affected: 1.3, 1.2

Details:

For arbitrary phase mode, a FIFO function is integrated into the SerDes TX serializer. This FIFO has three states (minus1, center, plus1) and is supposed to be reset to the center state at startup. From this position, the SerDes is then tolerant to variations of phase between the input clock (TXBCLKIN) and the SerDes internal clock, caused by temperature and voltage variations. However, as a result of a logic bug, the possibility exists that under some circumstances, the FIFO may not start in the center state. When this happens, there is a risk that the FIFO may subsequently overflow or underflow.

Whether the FIFO fails to initialize to the center state depends on the timing relationships between several signals, including the SerDes internal clock. Even if the FIFO fails to initialize to the center state, the FIFO will only underflow or overflow if the phase relationship between the TXBCLKIN input and the internal SerDes clock vary (due to temperature or voltage changes) in such a way as to cause their edges to cross in one particular direction. Overflow results in two bits being added to the data stream. Underflow results in two bits being deleted. If overflow or underflow occurs at all, it only happens once per TX lane because after it has occurred the FIFO is configured exactly as if it had initialized to the center state at startup.

The precise silicon process of the device will also be a factor in whether the overflow or underflow occurs. Some devices may exhibit this behavior at some particular PVT combinations, others may never exhibit it. It is not possible to predict whether, or under what conditions, a device is susceptible. If overflow or underflow occur, it could be at any time ranging from immediately after startup to weeks, months, or years later.

Workarounds:

The issue can be worked around by software control of two ports on the SerDes. At initialization, cycling of bits resets the circuit and resolves the issue.

- AIF has a software workaround as follows:

  The software workaround limits restart to per macro, not per lane. There is one set of software control bits for the B8 and another for the B4. For details, see the device-specific data manual, *TMS320C6474 Multicore Digital Signal Processor* (literature number SPRS552). There are new recommendations for the initialization sequence that is shown in the following code example:

```
//Enable the Tx Link
CSL_FINST(hAifLink[0]->regs->LCFG[1].LINK_CFG, AIF_LINK_CFG_TX_LINK_EN, ENABLED);
//Set the Link Rate
if (aCommoncfg[0].linkRate == CSL_AIF_LINK_RATE_1x){
      CSL_FINST(hAifLink[0]->regs->LCFG[1].LINK_CFG, AIF_LINK_CFG_LINK_RATE, 1X);
}
else if (aCommoncfg[0].linkRate == CSL_AIF_LINK_RATE_2x)
{
      CSL_FINST(hAifLink[0]->regs->LCFG[1].LINK_CFG, AIF_LINK_CFG_LINK_RATE, 2X);
}
else if (aCommoncfg[0].linkRate == CSL_AIF_LINK_RATE_4x)
{
      CSL_FINST(hAifLink[0]->regs->LCFG[1].LINK_CFG, AIF_LINK_CFG_LINK_RATE, 4X);
}
//Toggle the ENFTP bit
CSL_FINS( hAifLink[0]->regs->AI_SERDES0_TST_CFG, AIF_AI_SERDES0_TST_CFG_INVPATT,
1);
CSL_FINS(hAifLink[0]->regs->AI_SERDES0_TST_CFG, AIF_AI_SERDES0_TST_CFG_INVPATT,
0);
```

  CSL version 3.0.6.2 for the C6474 device has a new hardware control command (CSL_AIF_CMD_ENABLE_DISABLE_TX_LINK_SI1_1) that has the fix for this advisory.

- EMAC has a software workaround and an auto-recovery for this advisory as follows:

  There is a new recommendation for initialization sequence as shown in the following code example. This example code should be used with CSL version 03.00.06.01.

```
SgmiiCfg.masterEn   = 0x1;
SgmiiCfg.loopbackEn = 0x1;
SgmiiCfg.auxConfig  = 0x0000000b;

if (0 == SGMII_config(&SgmiiCfg))
        printf("SGMII config successful........\n");
else
        printf("SGMII config NOT successful........\n");

LocalTicks = 0;
while(LocalTicks !=3); // wait for 2us

SgmiiCfg.txConfig   = 0x00000e21; // enable transmitter
SgmiiCfg.rxConfig   = 0x00081021;

if (0 == SGMII_config(&SgmiiCfg))
        printf("SGMII config successful........\n");
else
        printf("SGMII config NOT successful........\n");

SgmiiCfg.txConfig   = 0x00001e21; // toggle the ENFTP bit

if (0 == SGMII_config(&SgmiiCfg))
        printf("SGMII config successful........\n");
else
        printf("SGMII config NOT successful........\n");

SgmiiCfg.masterEn   = 0x1;
SgmiiCfg.loopbackEn = 0x1;
SgmiiCfg.txConfig   = 0x00000e21; // toggle the ENFTP bit

if (0 == SGMII_config(&SgmiiCfg))
        printf("SGMII config successful........\n");
else
        printf("SGMII config NOT successful........\n");

// wait for the Auto-negotiation Complete
SGMII_REGS->CONTROL |= 0x1; // Loopback mode is selected

// set full dupex and Gig bits
SGMII_REGS->MR_ADV_ABILITY = 0x9801;
```

- SRIO has an auto-recovery as follows:

  Auto-recovery resets the link and re-exposes the issue. TI is working to understand the likelihood of repeated recovery and whether there could be performance impacts due to repeated recovery.

The software workaround is enabled with a partial fix for AIF and EMAC. A complete fix will be available in silicon revision 2.x.

Advisory 1.3.4 *MAC EOI Register Write Causes Potential CPU Lockup*

Revision(s) Affected: 1.3, 1.2; Fixed in CSL version 03.00.06.01

Details: A bug has been found affecting multiple cores trying to write the EOI register via the MAC interface. It causes a lockup of one of the three MAC interfaces that is attempting to write the EOI register.

When multiple cores try to access the MAC interface one of the three cores that requested the EOI write gets locked up. This situation occurs when the MAC interface receives the EOI register write requests from multiple cores like "X" write followed by "Y" write at same clock, the EOI register updates only one write at a time, X or Y, and ignores the other write. The EOI write request that was ignored by the MAC locks up the CPU that requested the write.

Workarounds: Semaphores can be used to fix the EOI issue. There are two new APIs added to write the EOI register, one for receive and the other for transmit. The application can make use of those APIs with the semaphore module, to protect the EOI write when all the 3 cores try to access EOI register at same time.

This workaround is required only when more than one core requests the EOI write. Code examples for receive and transmit writes are shown below.

- Before and after `rxEoiWrite`, the semaphore APIs are called:

```
/* Check Whether Handle opened successfully and then read module status*/
if(hSemHandle!= NULL){
      /* Check whether semaphore resource is Free or not */
      do{
      /* Get the semaphore*/
      CSL_semGetHwStatus(hSemHandle,CSL_SEM_QUERY_DIRECT,&response);
      }while(response.semFree != CSL_SEM_FREE);

/* write the EOI register */
      EMAC_rxEoiWrite(coreNum);

/* Release the semaphore*/
CSL_semHwControl(hSemHandle, CSL_SEM_CMD_FREE_DIRECT,NULL);
```

- Before and after `txEoiWrite` the semaphore APIs are called:

```
/* Check Whether Handle opened successfully and then read module status*/
if(hSemHandle!= NULL){
/* Check whether semaphore resource is Free or not*/
      do {
      /* Get the semaphore*/
CSL_semGetHwStatus(hSemHandle,CSL_SEM_QUERY_DIRECT,&response);
      } while (response.semFree != CSL_SEM_FREE);

/* write the EOI register */
      EMAC_txEoiWrite(coreNum);

/* Release the semaphore*/
CSL_semHwControl(hSemHandle, CSL_SEM_CMD_FREE_DIRECT,NULL);
```

This issue is fixed in the CSL version 03.00.06.01

Advisory 1.3.5 *Potential SerDes Clocking Issue*

Revision(s) Affected: 1.3, 1.2

Details: A bug has been found in the SerDes interfaces that causes a SerDes clocking problem in normal functional operation. This problem will not occur when external pull-down is applied on the TCK pin (JTAG controller clock). SerDes are used in the Ethernet interface (EMAC), Serial RapidIO interface (SRIO) and the Antenna Interface (AIF).

The TCK pin (JTAG controller clock) is internally assigned to an internal signal that is used by the SerDes macro. For the SerDes macro to get proper clocking in the normal functional operation, it needs the internal signal to be held low. However, there is an internal pull-up on the TCK, creating problems for SerDes operation. This problem exists on all SerDes interfaces.

Workaround: The TCK pin should be externally pulled down with an 1-kΩ resistor.

Advisory 1.3.6 *I2C: Slave Boot Aborts*

Revision(s) Affected: 1.3, 1.2

Details: I2C Slave Boot is intended to speed the boot process for a system with more than two devices by allowing a single master read of the I2C EEPROM followed by a broadcast by that master to all remaining devices on the I2C bus. However, during the I2C slave boot process an internal exception is encountered, causing the boot sequence to abort on the slave device(s). Consequently, I2C slave boot does not complete.

Workaround: Use I2C master boot for all devices in the system. Other boot modes with SRIO or EMAC may also be utilized, if available on system.

Advisory 1.3.7 *EMAC Boot Issue*

Revision(s) Affected: 1.3, 1.2

Details: The EMAC ready announcement frame is not transmitted when the C6474 device is booted in master and slave modes.

When the DSP is booted in EMAC master/slave boot modes (boot modes 4, 5), the DSP transmits an Ethernet Ready Announcement (ERA) frame in the form of a BOOTP request. The BOOTP request is intended to inform the host server that the DSP is ready to receive boot packets. The ERA frame packet is described in more detail in the *TMS320C6474 Bootloader User's Guide* (literature number SPRUG24).

Texas Instruments will fix the Ethernet Ready Announcement frame transmission in the next silicon revision for C6474 devices.

Workaround 1: Have the host that is responsible for sending the boot packets broadcast a small boot table with the program that is shown in the example below. This will cause any C6474 device to restart the EMAC boot procedure (without configuring the MAC peripheral again) and re-transmit the ERA.

Re-send ERA packet code:

```
BOOT_REENTRY_ADDR .equ 03c000110h
BOOT_EMAC_OPT     .equ 01088480Ah

   MVKL   BOOT_EMAC_OPT, A1
   MVKH   BOOT_EMAC_OPT, A1

   MVKL    0x00000026, A4  ;overwrite option field in EMAC bootparam
   MVKH    0x00000026, A4
   STH     A4, *A1
   NOP     4

   MVKL    BOOT_REENTRY_ADDR,  B3
   MVKH    BOOT_REENTRY_ADDR,  B3
   BNOP    B3, 5
```

Workaround 2: The host server would need to rely on prior knowledge of the DSP MAC address to transmit boot packets to the correct DSP. The DSP will be ready to receive EMAC boot packets within 2 ms following deassertion of reset.

In the scenario where the boot server reads the MAC address of the DSP from the ERA packet, the procedure would need to be changed. After some customer TBD delay where the ERA is not received, the host sends the broadcast packet with the payload described in Workaround 1.

3 Silicon Revision 1.2 Known Design Exceptions to Functional Specifications

Table 4. Silicon Revision 1.2 Advisory List

Title	Page
Advisory 1.2.1 DSP SDMA/IDMA: Unexpected Stalling of SDMA/IDMA Access to L2 SRAM	23
Advisory 1.2.2 Potential Data Corruption on SCR Bridge	30
Advisory 1.2.3 Potential Insertion or Deletion of 2 Bits in SerDes Data Stream	31
Advisory 1.2.4 MAC EOI Register Write Causes Potential CPU Lockup	33
Advisory 1.2.5 Potential SerDes Clocking Issue	34
Advisory 1.2.6 I2C: Slave Boot Aborts	35
Advisory 1.2.7 Potential Random E-fuse Blow	36
Advisory 1.2.8 EMAC Boot Issue	37
Advisory 1.2.9 EDMA3CC COMPACTV Issue	38

Advisory 1.2.1 *DSP SDMA/IDMA: Unexpected Stalling of SDMA/IDMA Access to L2 SRAM*

Revision(s) Affected: 1.3, 1.2

Details:

Note: Only when DSP level 2 (L2) memory is configured as non-cache (RAM), unexpected stalling may occur on DSP SDMA/IDMA accesses. If DSP L2 memory is used only as cache ***or*** if L2 RAM is ***not*** accessed by IDMA or via the SDMA interface during run-time, then this exception does not apply.

The C64x+ megamodule has a Master Direct Memory Access (MDMA) bus interface and a Slave Direct Memory Access (SDMA) bus interface. The MDMA interface provides DSP access to resources outside the C64x+ megamodule (i.e., DDR2 memory). The MDMA interface is used for CPU/cache accesses to memory beyond the level 2 (L2) memory level. These accesses include cache line allocates, write-backs, and non-cacheable loads and stores to/from system memories. The SDMA interface allows other master peripherals in the system to access level 1 data (L1D), level 1 program (L1P), and L2 RAM DSP memories. The masters allowed accesses to these memories are DMA controllers, EMAC, and SRIO. The DSP Internal Direct Memory Access (IDMA) is a C64x+ megamodule DMA engine used to move data between internal DSP memories (L1, L2) and/or the DSP peripheral configuration bus. The IDMA engine shares resources with the SDMA interface.

The C64x+ megamodule has an L1D cache and an L2 caches, both of which implement write-back data caches. The C64x+ megamodule holds updated values for external memory as long as possible. It writes these updated values, called *victims*, to external memory when it needs to make room for new data, when requested to do so by the application, or when a load is performed from a non-cacheable memory for which there is a set match in the cache (i.e., the non-cacheable line would replace a dirty line if cached). The L1D sends its victims to L2. The caching architecture has pipelining, meaning multiple requests could be pending between L1, L2, and MDMA. For more Details: on the C64x+ megamodule and its MDMA and SDMA ports, see the *TMS320C64x+ Megamodule Reference Guide* (literature number SPRU871).

Ideally, the MDMA (the blue lines in Figure 3) and SDMA/IDMA paths (the orange lines in Figure 3) operate independently with minimal interference. Normally, MDMA accesses may stall for extended periods of time (clock cycles) due to expected system level delays (e.g., bandwidth limitations, DDR2 memory refreshes). However, when using L2 as RAM, SDMA and/or IDMA accesses to L2/L1 may experience unexpected stalling in addition to the normal stalls seen by the MDMA interface. For latency-sensitive traffic, the SDMA stall can result in missing real-time deadlines.

Note: SDMA/IDMA accesses to L1P/D will not experience an unexpected stall if there are no SDMA/IDMA accesses to L2. Unexpected SDMA/IDMA stalls to L1 happen only when they are pipelined behind L2 accesses.

Figure 3 is a simplified view for illustrative purposes only. The IDMA/SDMA path (orange lines) can also go to L1D/L1P memories and IDMA can go to the DSP CFG peripherals. MDMA transactions (blue lines) can also originate from L1P or L1D through the L2 controller or directly from the DSP.

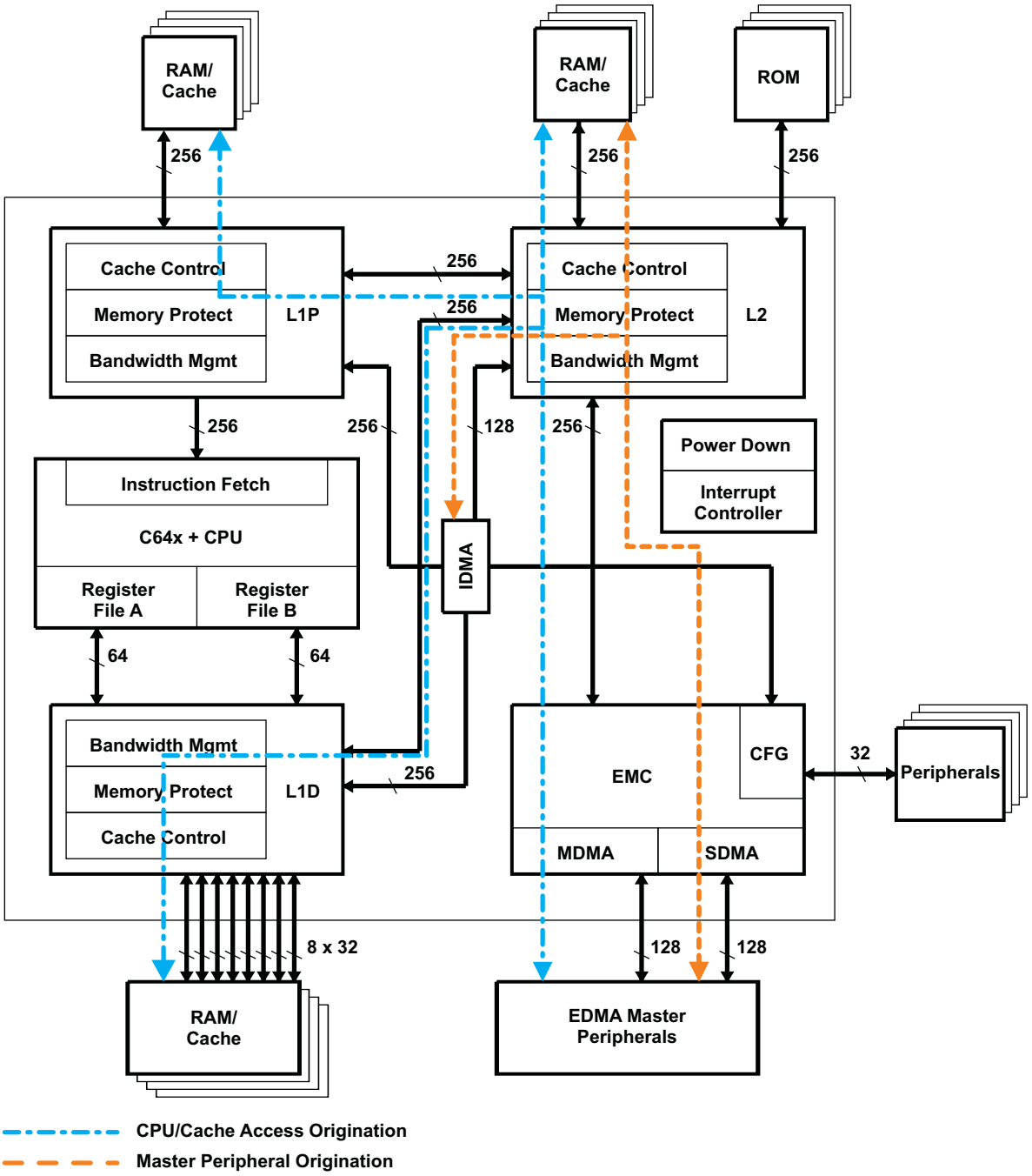


Figure 3. IDMA, SDMA, and MDMA Paths

SDMA/IDMA stalls may occur during the following scenarios. Each of these scenarios describes expected normal DSP functionality, but the SDMA/IDMA access potentially exhibits additional unexpected stalling.

1. Bursts of writes to non-cacheable MDMA space (i.e., DDR2). The DSP buffers up to 4 non-cacheable writes. When this buffer fills, SDMA/IDMA is blocked until the buffer is no longer full. Therefore, bursts of non-cacheable writes longer than three writes can stall SDMA/IDMA traffic.
2. Various combinations of L1 and L2 cache activity:
   a. L1D read miss generating victim traffic to L2 (cache or SRAM) or external memory. The SDMA/MDMA may be stalled while servicing the read miss and the victim. If the read miss also misses L2 cache, the SDMA/IDMA may be stalled until data is fetched from external memory to service the read miss. If the read access is to non-cacheable memory there will still potentially be an L1D victim generated even though the read data will not replace the line in the L1D cache.
   b. L1D read request missing L2 (going external) while another L1D request is pending. The SDMA/IDMA may be stalled until the external memory access is complete.
   c. L2 victim traffic to external memory during any pending L1D request. The SDMA/IDMA may be stalled until external memory access and the pending L1D request are complete.

The duration of the SDMA/IDMA stalls depends on the quantity/characteristics of the L1/L2 cache and the MDMA traffic in the system. In cases 2a, 2b, and 2c, stalling may or may not occur depending on the state of the cache request pipelines and the traffic target locations. These stalling mechanisms may also interact in various ways, causing longer stalls. Therefore, it is difficult to predict if stalling will occur and for how long.

SDMA/IDMA stalling and any system impact is most likely in systems with excessive context switching, L1/L2 cache miss/victim traffic, and heavily loaded EMIF.

Use the following steps to determine if SDMA/IDMA stalling is the cause of real-time deadline misses for existing applications. Situations where real-time deadlines may be missed include loss of McBSP samples and low peripheral throughput.

1. Determine if the transfer missing the real-time deadline is accessing L2 or L1D memory. If not, then SDMA/IDMA stalling is not the source of the real-time deadline miss.
2. Identify all SDMA transfers to/from L2 memory (e.g., EDMA transfer to/from L2 from/to a McBSP or from/to AIF, TCP, or VCP). If there are no SDMA transfers going to L2, then SDMA/IDMA stalling is not the source of the problem.
3. Redirect all SDMA transfers to L2 memory to other memories using one of the following methods:
   - Temporarily transfer all the L2 SDMA transfers to L1D SRAM.
   - If not all L2 SDMA transfers can be moved to L1D memory, temporarily direct some of the transfers to DDR memory and keep the rest in L1D memory. There should be ***no*** L2 SDMA transfers.
   - If neither of the above approaches are possible, move the transfer with the real-time deadline to the EMAC CPPI RAM. If the EMAC CPPI RAM is not big enough, a two-step mechanism can be used to page a small working buffer defined in the EMAC CPPI RAM into a bigger buffer in L2 SRAM. The EDMA module can be setup to automate this double buffering scheme without CPU intervention for moving data from the EMAC CPPI RAM. Some throughput degradation is expected when the buffers are moved to the EMAC CPPI RAM.

     Note: Note that EMAC CPPI RAM memory is word-addressable only and, therefore, must be accessed using an EDMA index of 4 bytes.

If real-time deadlines are still missed after implementing any of the options in Step 3, then SDMA/IDMA stalling is likely ***not*** the cause of the problem. If real-time deadline misses are solved using any of the options in Step 3, then SDMA/IDMA stalling ***is*** likely the source of the problem.

An extreme manifestation of the IDMA/SDMA stall bug is the C64x+ MDMA-SDMA deadlock that requires a device reset or power-on reset in order for the system to recover. The following summarizes the deadlock conditions:

- Master(s) on a single main MSCR port write to the GEM's SDMA followed by a write to slaveX
- The GEM issues victim traffic or a non-cacheable write to slaveX
- Any one of the following:
  - A write data path pipelined in main MSCR between master(s) and the GEM's SDMA
  - A bridge exists between master(s) and the main MSCR
  - Master(s) are able to issue a command to slaveX concurrent with the write to the GEM's SDMA.

A load (either cacheable or non-cacheable) from another core's L1D or L2 memory can additionally create a deadlock condition. When the load is issued the read command is propagated to the SDMA port of the other core through a bridge that is shared with the EDMA TC1, EMAC, RapidIO (both data and CPPI), and other GEM MDMA. When the load is issued, if a victim is generated in L1D cache, then the SDMA may stall until the load completes. If other masters are issuing commands through the shared bridge, then the bridge may fill due to the stalled SDMA before the read command can propagate through the bridge and complete. In summary, a deadlock can occur if the following is true:

- GEMx issues a read to GEMy or GEMz L1D or L2 SRAM
- Any of the following are issuing commands to GEMx L2: TC1, EMAC (data or CPPI), RapidIO (data or CPPI), GEMy, or GEMz.

Workarounds:

Method 1

Issues such as dropped McBSP samples can be worked around by moving latency-sensitive buffers outside the C64x+ megamodule. For example, rather than placing buffers for the McBSP into L1/L2, those buffers can instead be placed in other memory, such as the EMAC CPPI RAM.

Note: Note that EMAC CPPI RAM memory is word-addressable only and, therefore, must be accessed using an EDMA index of 4 bytes.

Method 2

To reduce the SDMA/IDMA stalling system impact, perform any of the following:

1. Improve system tolerance on DMA side (SDMA/IDMA/MDMA):
   - Understand and minimize latency-critical SDMA/IDMA accesses to L2 or L1P/D.
   - Directly reduce critical real-time deadlines, if possible, at peripheral/IO level (e.g., increase word size and/or reduce bit rates on serial ports).
   - To reduce DSP MDMA latency:
     – Increase the priority of the DSP access to DDR2 such that MDMA latency of MDMA accesses causing stalls is minimized.
       Note: Other masters may have real-time deadlines that dictate higher priority than the DSP.
     – Lower the PRIO_RAISE field setting in the DDR2 memory controller's burst priority register. Values ranging between 0x10 and 0x20 should give decent performance and minimize latency; lower values may cause excessive SDRAM row thrashing.
2. Minimize offending scenarios on DSP/caching side:
   - If the DSP performing non-cacheable writes is causing the issue, insert protected non-cacheable reads (as shown in the last list item below) every few writes to allow the write buffer to empty.
   - Use explicit cache commands to trigger cache writebacks during appropriate times (L1D Writeback All, L2 Writeback All). ***Do not*** use these commands when real-time deadlines must be met.
   - Restructure program data and data flow to minimize the offending cache activity.
     – Define the read-only data as *const*. The const C keyword tells the compiler not to write to the array. By default, such arrays are allocated to the .const section as opposed to BSS. With a suitable linker command file, the developer can link the .const section off chip, while linking .bss on chip. Because programs initialize .bss at run time, this reduces the program's initialization time and total memory image.
     – Explicitly allocate lookup tables and writeable buffers to their own sections. The #pragma DATA_SECTION (label, *section*) directive tells the compiler to place a particular variable in the specified COFF section. The developer can explicitly control the layout of the program with this directive and an appropriate linker command file.
     – Avoid directly accessing data in slow memories (e.g., flash); copy at initialization time to faster memories.
   - Modify troublesome code.
     – Rewrite using DMAs to minimize data cache writebacks. If the code accesses a large quantity of data externally, consider using DMAs to bring in the data, using double buffering and related techniques. This will minimize cache write-back traffic and the likelihood of SDMA/IDMA stalling.
     – Re-block the loops. In some cases, restructuring loops can increase reuse in the cache and reduce the total traffic to external memory.
     – Throttle the loops. If restructuring the code is impractical, then it is reasonable to slow it down. This reduces the likelihood that consecutive SDMA/IDMA blocks stack up in the cache request pipelines, resulting in a long stall.

- Protect non-cacheable reads from generating an SDMA stall by freezing the L1D cache during the non-cacheable read access(es). The following example code contains a function that protects non-cacheable reads, avoids blocking during the reads, and, therefore, avoids the deadlock state.

```
;; ======================================================================== ;;
;;  Long Distance Load Word                                                 ;;
;;                                                                          ;;
;;      int long_dist_load_word(volatile int *addr)                         ;;
;;                                                                          ;;
;;  This function reads a single word from a remote location with the L1D   ;;
;;  cache frozen.  This prevents L1D from sending victims in response to    ;;
;;  these reads, thus preventing the L1D victim lock from engaging for the  ;;
;;  corresponding L1D set.                                                  ;;
;;                                                                          ;;
;;  The code below does the following:                                      ;;
;;                                                                          ;;
;;      1.  Disable interrupts                                              ;;
;;      2.  Freeze L1D                                                      ;;
;;      3.  Load the requested word                                         ;;
;;      4.  Unfreeze L1D                                                    ;;
;;      5.  Restore interrupts                                              ;;
;;                                                                          ;;
;;  Interrupts are disabled while the cache is frozen to prevent affecting  ;;
;;  the performance of interrupt handlers.  Disabling interrupts during     ;;
;;  the long distance load does not greatly impact interrupt latency,       ;;
;;  because the CPU already cannot service interrupts when it's stalled by  ;;
;;  the cache.  This function adds a small amount of overhead (~20 cycles)  ;;
;;  to that operation.                                                      ;;
;;                                                                          ;;
;; ======================================================================== ;;

        .asg    0x01840044,     L1DCC           ; L1D Cache Control
        .global _long_dist_load_word
        .text
        .asmfunc
; int long_dist_load_word(volatile int *addr)
_long_dist_load_word:
        MVKL    L1DCC,          B4
        MVKH    L1DCC,          B4
||      DINT                                    ; Disable interrupts
||      MVK     1,              B5
        STW     B5,             *B4             ; \_ Freeze cache
        LDW     *B4,            B5              ; /
        NOP     4
        SHR     B5,     16,     B5              ; POPER -> OPER
||      LDW     *A4,            A4              ; read value remotely
        NOP     4
        STW     B5,             *B4             ; \_ Restore cache
        RET     B3
||      LDW     *B4,            B5              ; /
        NOP     4
        RINT                                    ; Restore interrupts
        .endasmfunc

;; ======================================================================== ;;
;;  End of file: ldld.asm                                                   ;;
;; ======================================================================== ;;
```

In the C6474 multicore device, when one GEM is accessing another GEM's L1 or L2 memory it is an MDMA access, so the potential SDMA/IDMA stall can occur. The stall can be avoided by using the EDMA to transfer data from one GEM's memory to another.

Method 3

Entirely eliminate the exception by removing all SDMA/IDMA accesses to L2 SRAM. For example, EMAC descriptors and EMAC payload cannot reside in L2. Master peripherals like the EDMA/QDMA, IDMA, and SRIO cannot access L2. There are no issues with the CPU itself accessing code/data in L2. This issue only pertains to SDMA/IDMA accesses to L2.

Deadlock Avoidance

To avoid the manifestation of a C64x+ deadlock, several Workarounds: are suggested depending on the VBUSM master in question:

VBUSM MASTER	WORKAROUND
GEM	GEMs should not write to the memory of any other GEM. This will cause complications across any master peripheral that is transferring data to multiple L2s. GEMs must not directly read from the memory of any other GEMs without providing the L1D cache disable workaround mentioned in Method 2 to ensure that the load will not stall itself indefinitely and hang the system.
EDMA3TCx	Inbound and outbound traffic should be programmed on different TC ports (i.e., two different EDMA queues, since a given queue maps to a given TC). Note that in-/out-bound direction is defined as the write direction, meaning that a DDR2-to-DDR2 transfer is outbound and L2-to-L2 is inbound. Any TC used to write to DDR should not be used to write to a GEM even when the TC writing to the DDR is also reading from DDR.
EMAC	EMAC should write to the GEM's memory or the DDR, but not both. This includes buffers and buffer descriptors. EMAC CPPI descriptors should be placed wholly in the local wrapper memory, any combination of wrapper and L2 memory (must match other master transactions), or any combination of wrapper and DDR2 SDRAM (must match other master transactions). Buffer descriptors should not be placed in any combination of L2 and DDR2 SDRAM.
SRIO	SRIO should transfer payload data to only GEM memories or to DDR2 SDRAM, but not both. This includes any direct I/O writes as well as any inbound RX messaging transfer.
SRIO CPPI	SRIO CPPI descriptors should be placed wholly in the local wrapper memory, any combination of wrapper and L2 memory, or any combination of wrapper and DDR2 SDRAM. Buffer descriptors should not be placed in any combination of L2 and DDR2 SDRAM.

Advisory 1.2.2 *Potential Data Corruption on SCR Bridge*

Revision(s) Affected: 1.3, 1.2

Details: This issue manifests itself when two masters write to a bridge endpoint and the commands arrive on the same clock cycle. The VBUS protocol is violated and the data is corrupted. The consequence is that one of the writes goes through with corrupt data, the other completes normally.

On some occasions the bridge may not recover without a reset. There is no software indication of this nor a means to reset only the bridge. Therefore, the situation must be avoided. The affected bridges are: TCP, VCP, AIF write port, and DMA bridge to configuration bus (where key endpoints beyond this bus could include Semaphore configuration port, EMAC configuration ports, PaRAM configuration port, or SRIO configuration ports).

Workarounds: Corrective action is taken by avoiding the issue as described in the following:

- TCP: Access to R/W ports is controlled by the Semaphore module; no issue.
- VCP: Access to R/W ports is controlled by the Semaphore module; no issue.
- AIF: DMA access is from a single transfer controller (TC); no issue. If more than one TC is used, the issue will be exposed.
- DMA bridge to configuration bus: Dedicate a single TC for use of the DMA to write through this bridge to all the endpoints beyond or use the configuration bus directly and do not use the DMA to program the following:
  - Semaphore configuration port: Configuration registers
  - SRIO configuration ports: Configuration registers
  - EMAC configuration ports: Configuration registers (caution on using EMAC CPPI buffer to work around the DSP SDMA/IDMA unexpected stalling, see Advisory 1.2.1)
  - PaRAM configuration port: Do not auto-program channels from one TC to another

Advisory 1.2.3 *Potential Insertion or Deletion of 2 Bits in SerDes Data Stream*

Revision(s) Affected: 1.3, 1.2

Details:

For arbitrary phase mode, a FIFO function is integrated into the SerDes TX serializer. This FIFO has three states (minus1, center, plus1) and is supposed to be reset to the center state at startup. From this position, the SerDes is then tolerant to variations of phase between the input clock (TXBCLKIN) and the SerDes internal clock, caused by temperature and voltage variations. However, as a result of a logic bug, the possibility exists that under some circumstances, the FIFO may not start in the center state. When this happens, there is a risk that the FIFO may subsequently overflow or underflow.

Whether the FIFO fails to initialize to the center state depends on the timing relationships between several signals, including the SerDes internal clock. Even if the FIFO fails to initialize to the center state, the FIFO will only underflow or overflow if the phase relationship between the TXBCLKIN input and the internal SerDes clock vary (due to temperature or voltage changes) in such a way as to cause their edges to cross in one particular direction. Overflow results in two bits being added to the data stream. Underflow results in two bits being deleted. If overflow or underflow occurs at all, it only happens once per TX lane because after it has occurred the FIFO is configured exactly as if it had initialized to the center state at startup.

The precise silicon process of the device will also be a factor in whether the overflow or underflow occurs. Some devices may exhibit this behavior at some particular PVT combinations, others may never exhibit it. It is not possible to predict whether, or under what conditions, a device is susceptible. If overflow or underflow occur, it could be at any time ranging from immediately after startup to weeks, months, or years later.

Workarounds:

The issue can be worked around by software control of two ports on the SerDes. At initialization, cycling of bits resets the circuit and resolves the issue.

- AIF has a software workaround as follows:

  The software workaround limits restart to per macro, not per lane. There is one set of software control bits for the B8 and another for the B4. For details, see the device-specific data manual, *TMS320C6474 Multicore Digital Signal Processor* (literature number SPRS552). There are new recommendations for the initialization sequence that is shown in the following code example:

```
//Enable the Tx Link
CSL_FINST(hAifLink[0]->regs->LCFG[1].LINK_CFG, AIF_LINK_CFG_TX_LINK_EN, ENABLED);
//Set the Link Rate
if (aCommoncfg[0].linkRate == CSL_AIF_LINK_RATE_1x){
      CSL_FINST(hAifLink[0]->regs->LCFG[1].LINK_CFG, AIF_LINK_CFG_LINK_RATE, 1X);
}
else if (aCommoncfg[0].linkRate == CSL_AIF_LINK_RATE_2x)
{
      CSL_FINST(hAifLink[0]->regs->LCFG[1].LINK_CFG, AIF_LINK_CFG_LINK_RATE, 2X);
}
else if (aCommoncfg[0].linkRate == CSL_AIF_LINK_RATE_4x)
{
      CSL_FINST(hAifLink[0]->regs->LCFG[1].LINK_CFG, AIF_LINK_CFG_LINK_RATE, 4X);
}
//Toggle the ENFTP bit
CSL_FINS( hAifLink[0]->regs->AI_SERDES0_TST_CFG, AIF_AI_SERDES0_TST_CFG_INVPATT,
1);
CSL_FINS(hAifLink[0]->regs->AI_SERDES0_TST_CFG, AIF_AI_SERDES0_TST_CFG_INVPATT,
0);
```

  CSL version 3.0.6.2 for the C6474 device has a new hardware control command (CSL_AIF_CMD_ENABLE_DISABLE_TX_LINK_SI1_1) that has the fix for this advisory.

- EMAC has a software workaround and an auto-recovery for this advisory as follows:

  There is a new recommendation for initialization sequence as shown in the following code example. This example code should be used with CSL version 03.00.06.01.

```
SgmiiCfg.masterEn   = 0x1;
SgmiiCfg.loopbackEn = 0x1;
SgmiiCfg.auxConfig  = 0x0000000b;

if (0 == SGMII_config(&SgmiiCfg))
        printf("SGMII config successful........\n");
else
        printf("SGMII config NOT successful........\n");

LocalTicks = 0;
while(LocalTicks !=3); // wait for 2us

SgmiiCfg.txConfig   = 0x00000e21; // enable transmitter
SgmiiCfg.rxConfig   = 0x00081021;

if (0 == SGMII_config(&SgmiiCfg))
        printf("SGMII config successful........\n");
else
        printf("SGMII config NOT successful........\n");

SgmiiCfg.txConfig   = 0x00001e21; // toggle the ENFTP bit

if (0 == SGMII_config(&SgmiiCfg))
        printf("SGMII config successful........\n");
else
        printf("SGMII config NOT successful........\n");

SgmiiCfg.masterEn   = 0x1;
SgmiiCfg.loopbackEn = 0x1;
SgmiiCfg.txConfig   = 0x00000e21; // toggle the ENFTP bit

if (0 == SGMII_config(&SgmiiCfg))
        printf("SGMII config successful........\n");
else
        printf("SGMII config NOT successful........\n");

// wait for the Auto-negotiation Complete
SGMII_REGS->CONTROL |= 0x1; // Loopback mode is selected

// set full dupex and Gig bits
SGMII_REGS->MR_ADV_ABILITY = 0x9801;
```

- SRIO has an auto-recovery as follows:

  Auto-recovery resets the link and re-exposes the issue. TI is working to understand the likelihood of repeated recovery and whether there could be performance impacts due to repeated recovery.

The software workaround is enabled with a partial fix for AIF and EMAC. A complete fix will be available in silicon revision 2.x.

Advisory 1.2.4 *MAC EOI Register Write Causes Potential CPU Lockup*

Revision(s) Affected: 1.3, 1.2; Fixed in CSL version 03.00.06.01

Details: A bug has been found affecting multiple cores trying to write the EOI register via the MAC interface. It causes a lockup of one of the three MAC interfaces that is attempting to write the EOI register.

When multiple cores try to access the MAC interface one of the three cores that requested the EOI write gets locked up. This situation occurs when the MAC interface receives the EOI register write requests from multiple cores like "X" write followed by "Y" write at same clock, the EOI register updates only one write at a time, X or Y, and ignores the other write. The EOI write request that was ignored by the MAC locks up the CPU that requested the write.

Workarounds: Semaphores can be used to fix the EOI issue. There are two new APIs added to write the EOI register, one for receive and the other for transmit. The application can make use of those APIs with the semaphore module, to protect the EOI write when all the 3 cores try to access EOI register at same time.

This workaround is required only when more than one core requests the EOI write. Code examples for receive and transmit writes are shown below.

- Before and after `rxEoiWrite`, the semaphore APIs are called:

```
/* Check Whether Handle opened successfully and then read module status*/
if(hSemHandle!= NULL){
      /* Check whether semaphore resource is Free or not */
      do{
      /* Get the semaphore*/
      CSL_semGetHwStatus(hSemHandle,CSL_SEM_QUERY_DIRECT,&response);
      }while(response.semFree != CSL_SEM_FREE);

/* write the EOI register */
      EMAC_rxEoiWrite(coreNum);

/* Release the semaphore*/
CSL_semHwControl(hSemHandle, CSL_SEM_CMD_FREE_DIRECT,NULL);
```

- Before and after `txEoiWrite` the semaphore APIs are called:

```
/* Check Whether Handle opened successfully and then read module status*/
if(hSemHandle!= NULL){
/* Check whether semaphore resource is Free or not*/
      do {
      /* Get the semaphore*/
CSL_semGetHwStatus(hSemHandle,CSL_SEM_QUERY_DIRECT,&response);
      } while (response.semFree != CSL_SEM_FREE);

/* write the EOI register */
      EMAC_txEoiWrite(coreNum);

/* Release the semaphore*/
CSL_semHwControl(hSemHandle, CSL_SEM_CMD_FREE_DIRECT,NULL);
```

This issue is fixed in the CSL version 03.00.06.01

Advisory 1.2.5 *Potential SerDes Clocking Issue*

Revision(s) Affected: 1.3, 1.2

Details: A bug has been found in the SerDes interfaces that causes a SerDes clocking problem in normal functional operation. This problem will not occur when external pull-down is applied on the TCK pin (JTAG controller clock). SerDes are used in the Ethernet interface (EMAC), Serial RapidIO interface (SRIO) and the Antenna Interface (AIF).

The TCK pin (JTAG controller clock) is internally assigned to an internal signal that is used by the SerDes macro. For the SerDes macro to get proper clocking in the normal functional operation, it needs the internal signal to be held low. However, there is an internal pull-up on the TCK, creating problems for SerDes operation. This problem exists on all SerDes interfaces.

Workaround: The TCK pin should be externally pulled down with an 1-kΩ resistor.

Advisory 1.2.6 *I2C: Slave Boot Aborts*

Revision(s) Affected: 1.3, 1.2

Details: I2C Slave Boot is intended to speed the boot process for a system with more than two devices by allowing a single master read of the I2C EEPROM followed by a broadcast by that master to all remaining devices on the I2C bus. However, during the I2C slave boot process an internal exception is encountered, causing the boot sequence to abort on the slave device(s). Consequently, I2C slave boot does not complete.

Workaround: Use I2C master boot for all devices in the system. Other boot modes with SRIO or EMAC may also be utilized, if available on system.

Advisory 1.2.7 *Potential Random E-fuse Blow*

Revision(s) Affected: 1.2

Details: In the final stages of screening the C6474 device for qualification, a subtle issue has been uncovered involving e-fuses being inadvertently blown during power up if an improper power and clock sequence is applied to the device. The e-fuse controller on the C6474 device may unintentionally blow e-fuses during power up when an invalid power sequence is used:

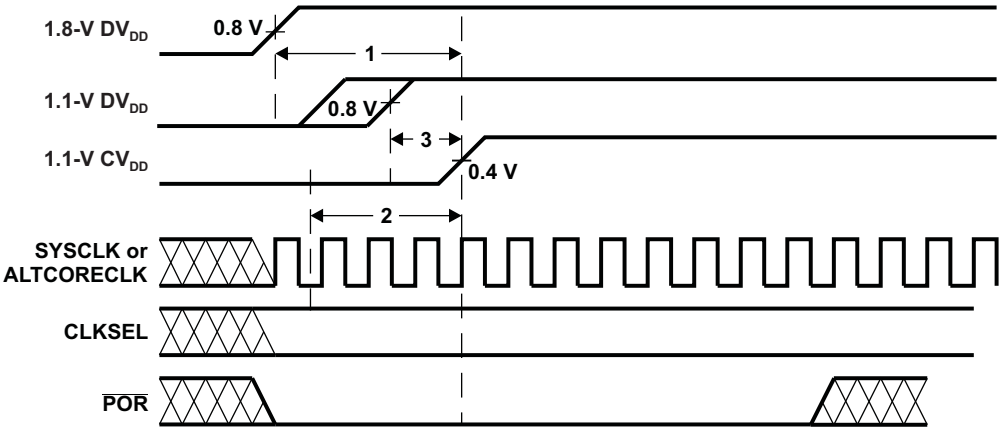
- The e-fuse controller has a defect that gates the output of an accidental programming prevention circuit with a clocked register.
- If proper sequencing of supplies and clocks is not maintained, then the program enable on the e-fuse ROM will be active until a valid reset (SYS_INITZ) is propagated to the register.
- The result is susceptibility to inadvertent blowing of e-fuses.

If the 1.1-V CV_{DD} scaled supply ramps before the 1.8-V and 1.1-V fixed supplies, the logic in the CV_{DD} domain powers up in random state. In this random state, there is small probability that conditions are met for inadvertent blowing of e-fuses.

The possible impact is that e-fuses that are not supposed to be blown are blown.

- Which e-fuses could inadvertently be blown is random.
- The type of e-fuse randomly blown will determine the end impact to the system, ranging from no impact to severe impact.
- Each power up event is a new opportunity for exposure to the issue in which e-fuses could be unintentionally blown.
- The probability of this is low, but not low enough.

Workaround: Guarantee that the 1.8-V DV_{DD} device input clocks and clock selects are active before a 1.1-V CV_{DD} scaled supply ramps (see Figure 4).



A 1.8-V DV_{DD} valid to 1.1-V CV_{DD} valid, ≥0.5 ms (min).

B Stable clock to 1.1-V CV_{DD} start, ≥100 µs (min).

C 1.1-V DV_{DD} to 1.1-V CV_{DD} start, ≥0.5 µs (min).

Figure 4. Correct Device Input Clocks, Clock Selects, and Scaled Supply Timings

TI's TMS test procedure follows the above recommendation to ensure that no devices are shipped with unintentionally blown e-fuses.

Note: E-fuses are used in multiple areas within the device. They are used for memory repair, device ID, EMAC ID, etc.

Advisory 1.2.8 *EMAC Boot Issue*

Revision(s) Affected: 1.3, 1.2

Details: The EMAC ready announcement frame is not transmitted when the C6474 device is booted in master and slave modes.

When the DSP is booted in EMAC master/slave boot modes (boot modes 4, 5), the DSP transmits an Ethernet Ready Announcement (ERA) frame in the form of a BOOTP request. The BOOTP request is intended to inform the host server that the DSP is ready to receive boot packets. The ERA frame packet is described in more detail in the *TMS320C6474 Bootloader User's Guide* (literature number SPRUG24).

Texas Instruments will fix the Ethernet Ready Announcement frame transmission in the next silicon revision for C6474 devices.

Workaround 1: Have the host that is responsible for sending the boot packets broadcast a small boot table with the program that is shown in the example below. This will cause any C6474 device to restart the EMAC boot procedure (without configuring the MAC peripheral again) and re-transmit the ERA.

Re-send ERA packet code:

```
BOOT_REENTRY_ADDR .equ 03c000110h
BOOT_EMAC_OPT     .equ 01088480Ah

   MVKL   BOOT_EMAC_OPT, A1
   MVKH   BOOT_EMAC_OPT, A1

   MVKL    0x00000026, A4  ;overwrite option field in EMAC bootparam
   MVKH    0x00000026, A4
   STH     A4, *A1
   NOP     4

   MVKL    BOOT_REENTRY_ADDR,  B3
   MVKH    BOOT_REENTRY_ADDR,  B3
   BNOP    B3, 5
```

Workaround 2: The host server would need to rely on prior knowledge of the DSP MAC address to transmit boot packets to the correct DSP. The DSP will be ready to receive EMAC boot packets within 2 ms following deassertion of reset.

In the scenario where the boot server reads the MAC address of the DSP from the ERA packet, the procedure would need to be changed. After some customer TBD delay where the ERA is not received, the host sends the broadcast packet with the payload described in Workaround 1.

Advisory 1.2.9 *EDMA3CC COMPACTV Issue*

Revision(s) Affected: 1.2

Details: A bug has been found inside the EDMA3 channel controller (EDMA3CC). The logic for decrementing the completion request active (COMPACTV) counter is incorrect for devices having six or more EDMA3 transfer controllers (EDMA3TCs). Therefore, the C6474 devices are affected by this bug.

The COMPACTV field inside the channel controller status register (CCSTAT) indicates the count for the number of outstanding transfer requests requiring completion status that have been submitted to the transfer controllers. The channel controller increments this count every time a transfer request (TR) is submitted and is programmed to report completion (the TCINTEN or TCCHEN or the ITCINTEN or ITCCHEN bits in OPT in the parameter entry associated with the TR are set). The counter decrements for every valid transfer completion code (TCC) received back from the transfer controllers. The bug occurs because the channel controller decrements the counter by an insufficient value when multiple responses are received concurrently from multiple (two or more) transfer controllers. Thus, the counter may gradually increase over time until it saturates at 0x3F. If at any time the count reaches a value of 0x3F, the channel controller does not service new TRs until the count is less than 0x3F (which will happen when a transfer completion code is received from a transfer controller for an in-flight request). Once the state is reached where the counter is close to the saturation value of 0x3F, the performance of the EDMA decreases dramatically. This decreased performance happens because the channel controller will artificially limit its number of TRs in flight to the COMPACTV saturation value thereby preventing full usage of the available TCs. When the count reaches 0x3F, the TCCERR bit is set in the channel controller error register (CCERR) causing an error interrupt when enabled.

Workaround: The workaround is achieved by having the DSP directly program one of the transfer controllers (bypassing the channel controller) with a transfer request that requires completion. This request avoids the COMPACTV increment (because TC is programmed directly) and forces a COMPACTV decrement when the TC responds to the CC with the completion signaling.

A specific transfer controller and a specific TCC value should be dedicated in the system for this workaround. TC2 or TC5 are suggested. TC2 is suggested because TC0 and TC1 can replace its connectivity. TC0 can be used for TCP/VCP transfers. TC5 is suggested because TC4 can replace its connectivity. TC4/TC3 can be used for AIF transfers.

The DSP should poll the COMPACTV field often enough such that the counter is not allowed to exceed 0x30. The actual COMPACTV polling interval may need to be set through experimentation on the specific end system, since the rate of increment of the counter is system and load specific.

Upon polling, if the value of the COMPACTV field is greater than a certain threshold (0x20 is suggested), then the DSP should program the TC with a COMPACTV decrement transfer. Upon completion of that transfer (as signaled in the CC IPR register) the COMPACTV field should be re-checked, and another COMPACTV decrement transfer submitted until the value of the counter is less than the threshold.

Note: Care must be taken such that the software does not over-decrement the counter since at the time of polling multiple requests may be in flight in the system and may result in additional decrements compared to the current observed value. If too many decrements occur, the counter may roll under from 0x0 to 0x3F and accidentally result in saturation of the counter. This is why a value of 0x20 is suggested as the threshold value (sufficiently large with respect to the number of actual requests that may be outstanding).

This workaround requires that a specific TC instance is dedicated to the COMPACTV decrement transfer. The reason is that, depending on the nature of the traffic on a given queue/TC, it may be difficult to control the timing of the normal CC TR submission to that TC versus the DSP programming of that TC. There is no hardware protection to prevent corruption of the TC registers in the case that both CC and DSP software attempt to program the TC simultaneously.

For the base addresses of the TCs, see the device-specific data manual, *TMS320C6474 Multicore Digital Signal Processor* (literature number SPRS552). A brief summary of the TC registers to be configured are provided in Table 5.

Table 5. TC Registers Summary

ADDRESS	REGISTER DESCRIPTION	SUGGESTED VALUE
TCx Base + 0x0200	Prog Set Options	See the Prog Set Options Register description below
TCx Base + 0x0204	Prog Set Src Address	See Prog Set Src/Dst Address Register description below
TCx Base + 0x0208	Prog Set Count	0x00010004 (ACNT = 4 and BCNT =1)
TCx Base + 0x020C	Prog Set Dst Address	See Prog Set Src/Dst Address Register description below
TCx Base + 0x0210	Prog Set B-Dim Idx	0x0 (don't care since BCNT=1). Writing to the PBIDX register triggers the transfer. Thus, this register should be written.

Note: The five registers listed in Table 5 should be written in the sequence shown (i.e., top to bottom). The last write, to the Prog Set B-Dim Idx register, triggers the transfer.

Prog Set Options Register

The Prog Set Option register is shown in Figure 5. The TCINTEN bit should be set to 0x1. The TCC code should be set to some known value that is not used by other requests in the system. The other fields should be set to 0x0. Upon completion of the transfer, the TCC value will be set in the corresponding bit in the IPR/IPRH registers. The software should poll for this bit in the IPR/IPRH registers and then clear it with the ICR/ICRH registers before programming the next COMPACTV decrement transfer.

Figure 5. Prog Set Options Register

31	23	22	21	20	19	18	17	16
Reserved		TCCH_EN	Rsvd	TCINT_EN	Reserved		TCC	
R-0		R/W-0	R-0	R/W-0	R-0		R/W-0	

15	12	11	10	8	7	6	4	3	2	1	0
TCC		Rsvd	FWID		Rsvd	PRI		Reserved		DAM	SAM
R/W-0		R-0	R/W-0		R-0	R/W-0		R-0		R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Prog Set Src/Dst Address Register

Although the user can specify any address for src/dst, one of the following settings is suggested:

1. Set the src/dst address as 0x31000000. This is a reserved location and transfer to this address takes less latency. However, the bus error (BUSERR) bit in the TCx error status register(ERRSTAT) will be set (the TCx error details register (ERRDET) will also be set). This TCx error should be ignored. This error is localized to the dedicated TC for this transfer and will not affect the system. Also, by default, the BUSERR will not cause the EDMA3TC error interrupt. This interrupt gets generated only when the TCx error enable (ERREN) register is set.
2. The other option is to set the src/dst address to the EDMA3TCx or the EDMA3CC peripheral ID (PID) register location. This transfer has more latency when compared to option 1, but will not cause TCx BUSERR condition.

Example code for programming the TC for this workaround (this example uses TC5):

```
#include <csl_edma3.h>
#include <soc.h>

#define EDMA3TC_POPT_REG (*(volatile Uint32*)(CSL_EDMA3TC_5_REGS + 0x200))
#define EDMA3TC_PSRC_REG (*(volatile Uint32*)(CSL_EDMA3TC_5_REGS + 0x204))
#define EDMA3TC_PCNT_REG (*(volatile Uint32*)(CSL_EDMA3TC_5_REGS + 0x208))
#define EDMA3TC_PDST_REG (*(volatile Uint32*)(CSL_EDMA3TC_5_REGS + 0x20C))
#define EDMA3TC_PBIDX_REG (*(volatile Uint32*)(CSL_EDMA3TC_5_REGS + 0x210))
#define COMPACTV_XFER_ADDRESS (0x31000000)
#define COMPACTV_XFER_COMPLETION_CODE (63) /* dedicate one TCC value for this */

void triggerCompactvDecTransfer ()
{
       EDMA3TC_POPT_REG = CSL_EDMA3_OPT_MAKE(FALSE, FALSE, FALSE, TRUE,\
              COMPACTV_XFER_COMPLETION_CODE, FALSE,\
              CSL_EDMA3_FIFOWIDTH_NONE, FALSE, FALSE,\
              CSL_EDMA3_ADDRMODE_INCR, CSL_EDMA3_ADDRMODE_INCR);

       EDMA3TC_PSRC_REG = COMPACTV_XFER_ADDRESS;
       EDMA3TC_PCNT_REG = CSL_EDMA3_CNT_MAKE(4, 1);
       EDMA3TC_PDST_REG = COMPACTV_XFER_ADDRESS;
       EDMA3TC_PBIDX_REG = CSL_EDMA3_BIDX_MAKE(0, 0);
}
```

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

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

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI 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. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI 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 and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Clocks and Timers	www.ti.com/clocks	Digital Control	www.ti.com/digitalcontrol
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
RFID	www.ti-rfid.com	Telephony	www.ti.com/telephony
RF/IF and ZigBee® Solutions	www.ti.com/lprf	Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2008, Texas Instruments Incorporated