

**TMS320C6711, TMS320C6711B,  
TMS320C6711C, TMS320C6711D**  
**Digital Signal Processors**  
**Silicon Errata**

**C6711 Silicon Revisions 1.0, 1.2, and 1.3**

**C6711B Silicon Revisions 2.0 and 2.1**

**C6711C Silicon Revision 1.1**

**C6711D Silicon Revision 2.0**

*SPRZ1730*  
May 2000  
Revised April 2005



Copyright © 2005, Texas Instruments Incorporated

## REVISION HISTORY

This errata revision history highlights the technical changes made to the SPRZ173N device-specific errata to make it an SPRZ173O revision.

**Scope:** Applicable updates to the C67x device family, specifically relating to the C6711/11B and C6711C/11D devices, have been incorporated.

PAGE(S) NO.	ADDITIONS/CHANGES/DELETIONS
14	Usage Notes for C6711D Silicon Revision 2.0: Added "JTAG, Clock/PLL Oscillator, McBSP0/1, and TIMER1: MCBSP1DIS Control Bit (DEVCFG.0) Affects IPU/IPDs on Specific Peripheral Pins"

## Contents

<b>1</b>	<b>Introduction</b>	<b>5</b>
1.1	Quality and Reliability Conditions	5
	TMX Definition	5
	TMP Definition	5
	TMS Definition	5
1.2	Revision Identification [C6711/C6711B/C6711C/C6711D]	6
<b>2</b>	<b>C6711D Silicon Revision 2.0 Known Design Exceptions to Functional Specifications and Usage Notes</b>	<b>7</b>
2.1	Usage Notes for C6711D Silicon Revision 2.0	7
	EMIF: L2 Cache Operations Block Other EDMA Operations to EMIF (C671x/C621x Devices: All Silicon Revisions)	7
	HPI: Illegal Memory Access Can Result in Unexpected HPI Behavior	12
	BUSREQ Asserted During HPI Boot	12
	Cache Configuration (CCFG) Register — P-Bit Function [New Enhancement]	13
	EMIF: Control Signals Not Inactive Before Asserting HOLDA	13
	EMIF: Data Corruption can Occur in SDRAM When the HOLD Feature is Used	13
	RESET Pin Has No Internal Pullup Resistor	14
	Boundary Scan: IDCODE is Only Loaded Onto Instruction Register When TRST Becomes Inactive	14
	JTAG, Clock/PLL Oscillator, McBSP0/1, and TIMER1: MCBSP1DIS Control Bit (DEVCFG.0) Affects IPUs/IPDs on Specific Peripheral Pins	14
2.2	C6711D Silicon Revision 2.0 Known Design Exceptions to Functional Specifications	15
<b>3</b>	<b>C6711C Silicon Revision 1.1 Known Design Exceptions to Functional Specifications and Usage Notes</b>	<b>16</b>
3.1	Usage Notes for C6711C Silicon Revision 1.1	16
	EMIF/HPI: EMIF Output and HPI Signals Can Drive During Reset	
3.2	C6711C Silicon Revision 1.1 Known Design Exceptions to Functional Specifications	16
Advisory 1.1.1	HPI: HPID Read/Write Accesses Must Be Terminated With a Fixed-Mode Access	16
Advisory 1.1.3	RESET Pin Has Internal Pullup Resistor	17
Advisory 1.1.4	PLL: Incorrect PLL Controller Peripheral Identification (PID) Register Value	17
Advisory 1.1.10	EDMA: EDMA Blocked from Accessing L2 During Long String of Stores to the Same Bank in L2 RAM	18
<b>4</b>	<b>C6711B Silicon Revision 2.1 Known Design Exceptions to Functional Specifications and Usage Notes</b>	<b>24</b>
4.1	Usage Notes for C6711 Silicon Revision 2.1	24
	HPI: Illegal Memory Access Can Result in Unexpected HPI Behavior	24
4.2	C6711B Silicon Revision 2.1 Known Design Exceptions to Functional Specifications	24
Advisory 2.1.2	EMIF: Async Read Setup Uses Write Setup Value	24
Advisory 2.1.4	EMIF: One Cycle Asynchronous Write Setup	24
Advisory 2.1.5	JTAG: Boundary Scan Shift-DR Register is Latched on the Falling Edge of TCK	25
Advisory 2.1.6	EDMA/L2 Controller: Potential EDMA Lockout From L2 SRAM	25
	Example GEL File	29

---

<b>5</b>	<b>C6711B Silicon Revision 2.0 Known Design Exceptions to Functional Specifications</b>	<b>31</b>
	Advisory 2.0.1 L1D Cache: Data Corruption if L1D Powered up to Wrong State	31
	Assembly Code (bug.asm)	33
	Linker File (lnk.cmd)	33
<b>6</b>	<b>C6711 Silicon Revision 1.3 Known Design Exceptions to Functional Specifications</b>	<b>34</b>
	Advisory 1.3.2 HPI: Read Data Corrupted in Fixed-Address Mode and FETCH Read Requests	34
	Advisory 1.3.4 JTAG: Boundary Scan Does Not Function	35
	Advisory 1.3.7 HPI: Write Request During HPI Time-Out Causes HPI Lock-Up	35
	Advisory 1.3.13 HPI: HRDY Behavior	36
<b>7</b>	<b>C6711 Silicon Revision 1.2 Known Design Exceptions to Functional Specifications</b>	<b>36</b>
<b>8</b>	<b>C6711 Silicon Revision 1.0 Known Design Exceptions to Functional Specifications</b>	<b>36</b>

## 1 Introduction

This document describes the known exceptions to the functional specifications for the TMS320C6711, TMS320C6711B, TMS320C6711C, and TMS320C6711D digital signal processors. [See the *TMS320C6711, TMS320C6711B, TMS320C6711C, TMS320C6711D Floating-Point Digital Signal Processors* data sheet (literature number SPRS088).] These exceptions are applicable to the TMS320C6711 and TMS320C6711B devices (256-pin Ball Grid Array, GFN suffix) and to the TMS320C6711C and TMS320C6711D devices (272-pin Ball Grid Array, GDP suffix).

For additional information, see the latest version of *TMS320C6000 DSP Peripherals Overview Reference Guide* (literature number SPRU190).

The advisory numbers in this document are not sequential. Some advisories have been *moved* to the next revision and others have been removed and documented in the user's guide. When advisories are moved or deleted, the remaining advisory numbers remain the same and are not resequenced.

### 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 range and voltage range should not, in any way, be construed as a warranty of performance.

#### TMP Definition

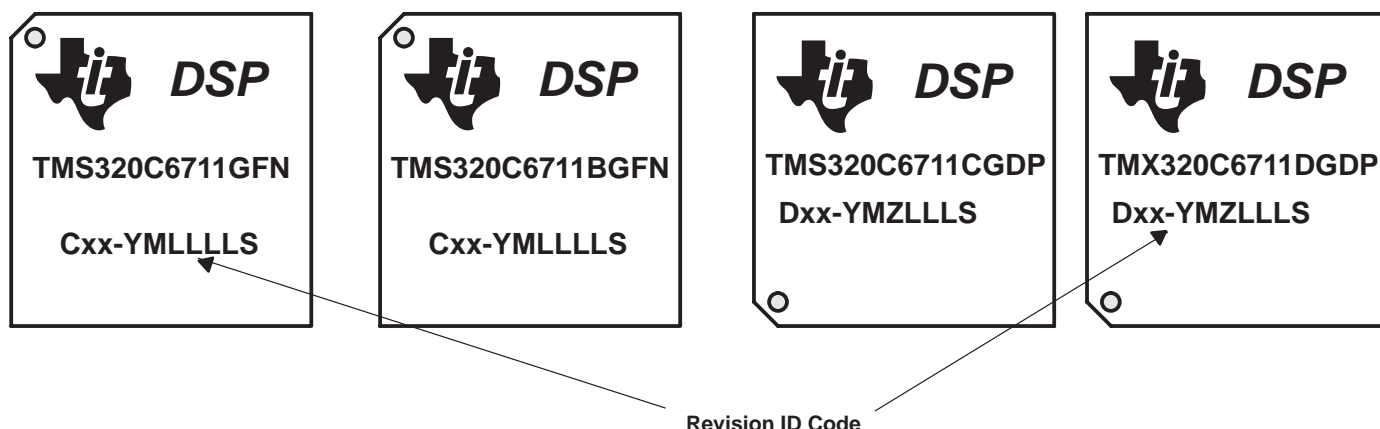
TI does not warranty product reliability for products classified as "TMP." By definition, the product has not completed reliability performance qualification according to TI Quality Systems Specifications; however, products are tested to a published electrical and mechanical specification.

#### TMS Definition

Fully-qualified production device

1.2 Revision Identification [C6711/C6711B/C6711C/C6711D]

The device revision can be determined by the lot trace code on the top of the package. The lot trace codes location for the GFN and GDP packages are shown in Figure 1 and the revision ID codes are listed in Table 1 and Table 2. The revision ID described here is not to be confused with the CPU revision ID that is in the Control Status Register.



NOTE: Qualified devices are marked with the letters “TMS” at the beginning of the device name, while nonqualified devices are marked with the letters “TMX” or “TMP” at the beginning of the device name.

Figure 1. Example, Lot Trace Code for C6711/C6711B (GFN) and C6711C/C6711D (GDP)

Silicon revision is identified by a code on the chip. The code is of the format Cxx-YMLLLLLS [for C6711/C6711B (GFN)] and of the format Dxx-YMZLLLLS [for C6711C/C6711D (GDP)]. If xx is 10, then the silicon is revision 1.0. If xx is 11, then the silicon is revision 1.1. If xx is 20, then the silicon is revision 2.0 and so on.

Table 1. Revision ID Codes [C6711C/C6711D]

Revision ID Code	Silicon Revision	Comments
11	1.1	TMX320C6711C, TMS320C6711C
20	2.0	TMS320C6711D

Table 2. Revision ID Codes [C6711/C6711B]

Revision ID Code	Silicon Revision	Comments
10	1.0	TMX320C6711
12	1.2	TMX320C6711
13	1.3	TMS320C6711 Silicon Revision 1.3 is functionally the same as revision 1.2. It is optimized from revision 1.2 for yield improvement.
20	2.0	TMX320C6711B
21	2.1	TMS320C6711B

## 2 C6711D Silicon Revision 2.0 Known Design Exceptions to Functional Specifications and Usage Notes

### 2.1 Usage Notes for C6711D Silicon Revision 2.0

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

#### **EMIF: L2 Cache Operations Block Other EDMA Operations to EMIF (C671x/C621x Devices: All Silicon Revisions)**

When using the L2 cache on the C671x/C621x devices, for a given EMIF-to-CPU frequency ratio, an L2 writeback or L2 writeback-invalidate operation may block other EDMA requests from accessing the EMIF until the operation completes. If the other EDMA requests to the EMIF have hard real-time deadlines, these deadlines may be missed if the deadline is shorter than the time required to complete the L2 writeback operation. The McBSP peripheral is most sensitive to this issue, as the buffering local to the McBSP peripheral can only hold data for at most one sample at a time before data loss occurs.

On the C671x/C621x devices, all cache requests to EMIF address ranges are serviced on the highest priority level of the EDMA (priority 0). All programmed EDMA or QDMA transfers (e.g., EDMA transfers to service the McBSP or paging data from EMIF to/from L2) and peripheral-initiated transfers (such as HPI) are limited to using priority 1 or priority 2 queues of the EDMA; therefore an L2 writeback or L2 writeback-invalidate operation may block the lower priority request.

Program-initiated cache coherency operations (such as L2 writeback and L2 writeback-invalidate operations) are submitted to the EDMA as a long string of cache operations. For block-based writeback commands, the maximum length of the cache writeback operation is under user control via the programmed address range. The length of the range writeback directly impacts the amount of time that the cache traffic may block other accesses to EMIF. The total potential block-out time equals the amount of time for the cache transfer and is calculated as follows:

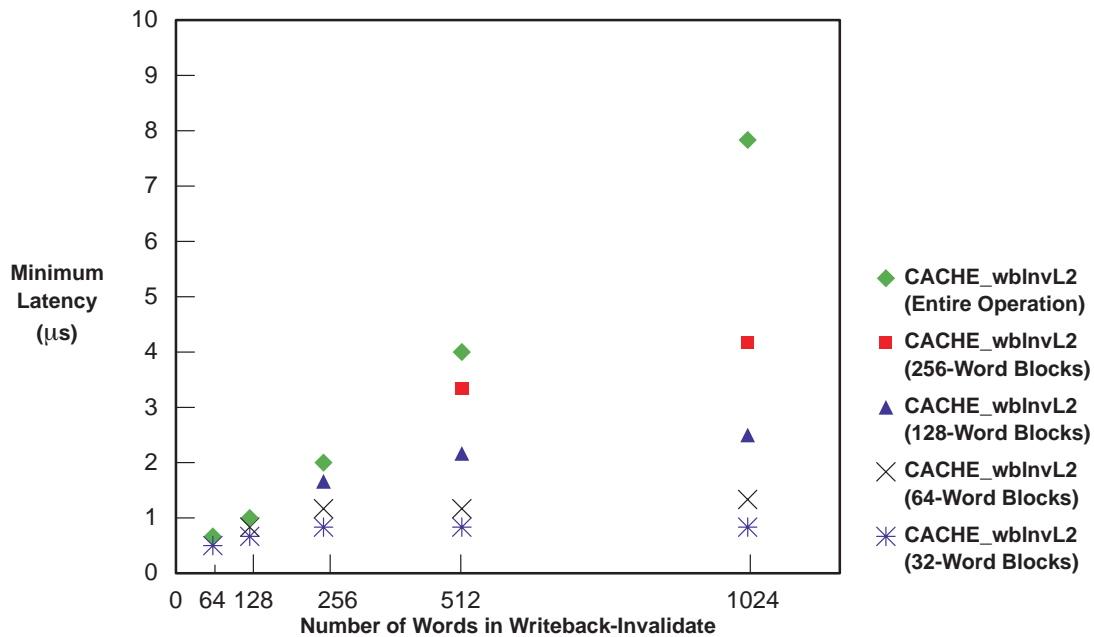
Cache transfer size \* EMIF clock cycle time = Total potential block-out time

For example, if the user performs an L2 writeback operation to external memory for 2048 words with a 100-MHz EMIF, the external EMIF bus may be blocked for: 2048 words \* 10 ns  $\cong$  20  $\mu$ s.

Global cache operations (such as L2 writeback-all or L2 writeback-invalidate-all) are also submitted to the EDMA as a long string of cache operations. However, the length of the global cache operation is not controllable by the user and can be as long as the depth of the L2 cache size (up to 64 Kbytes). If the user performs an L2 writeback-all operation to external memory using a 100-MHz EMIF, and L2 is set to the maximum cache size, the external EMIF bus may be blocked for 16 384 words \* 10 ns  $\cong$  160  $\mu$ s. Since this can block the EMIF for long periods of time, the user should avoid using global cache operations at the same time as real-time data transfers. In general, this is not a limiting factor since global cache operations are primarily performed during system initialization, task switches, or other non real-time code segments.

As the sample rate is system-dependent, the user must calculate the time between serial samples to determine the best approach to avoid data loss. The user may break large cache operations into smaller blocks, and transmit each of these blocks using the `CACHE_wbInvL2()` and `CACHE_wbL2()` CSL functions. By breaking the large cache operations into smaller blocks, other peripherals are then allowed to access the EDMA.

If the EMIF frequency is more than half of the CPU frequency, the device is able to service the L2 writeback requests faster than the requests can be issued, leaving some EMIF bandwidth available to service other EDMA requests, so the block-out problem is less noticeable. Therefore, breaking down cache operations into smaller blocks is more critical when the EMIF frequency is less than half of the CPU frequency. Figure 2 shows the minimum required latency between McBSP transfers to EMIF at 200-MHz CPU and 50-MHz EMIF when breaking down the cache operations. These McBSP transfers were performed with concurrent cache operations to EMIF, creating a block-out scenario. With the 1024-word cache writeback-invalidate operation broken into 32-word blocks, the McBSP is able to perform almost 10 times faster. The performance improvement is similar when breaking down the writeback-only operation.

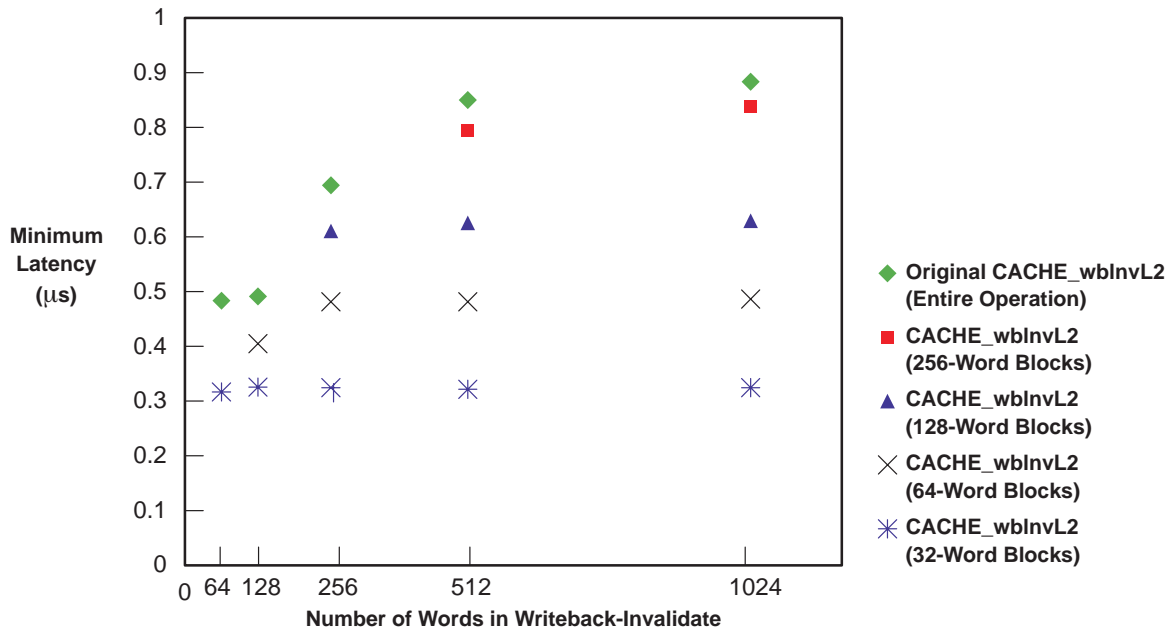


**Figure 2. Minimum Required Latency Between McBSP Events for a Successful Transfer with Concurrent L2 Writeback-Invalidates at 200-MHz CPU and 50-MHz EMIF, Using Entire Operations and Block Breakdown**

For example, if the CPU is running at 200 MHz with a 50-MHz EMIF and you have a McBSP hard real-time deadline of 5 µs, Figure 2 shows that a 1024-word L2 writeback-invalidate may cause data loss since back-to-back McBSP events can only be serviced at ~8 µs. By breaking down the L2 writeback-invalidate into 256-word blocks, you can then meet the 5-µs McBSP deadline. In other words, when performing a 1024-word L2 writeback-invalidate operation with the CPU and EMIF conditions cited above, the McBSP events can be serviced in ~8 µs for the entire operation (one whole block), in ~4 µs when breaking it into 256-word blocks, in ~2.5 µs when breaking it into 128-word blocks, etc.

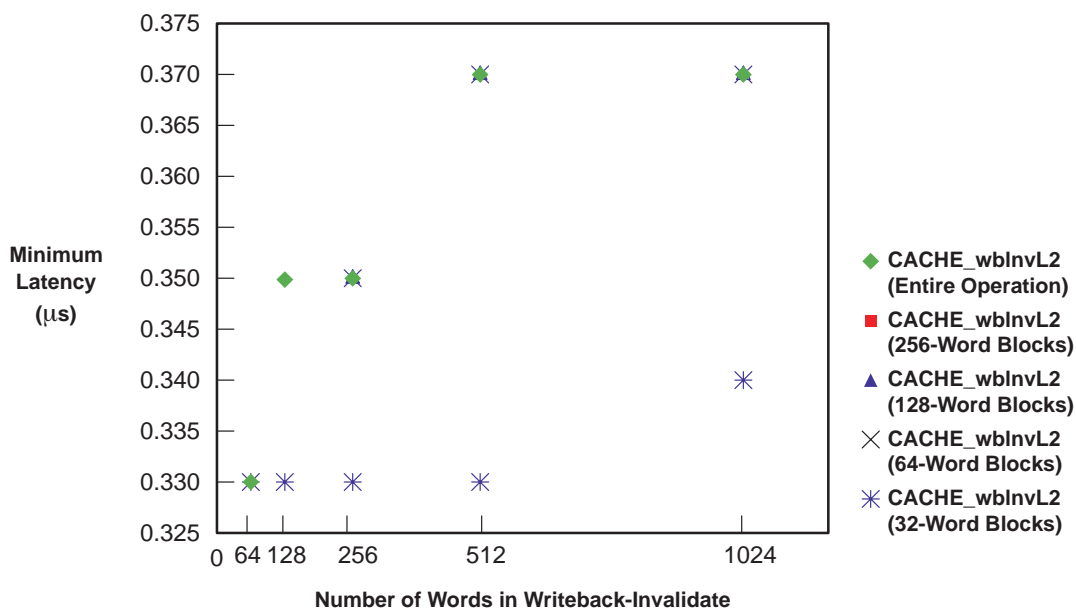
When the CPU is set to 225 MHz and the EMIF is set to 100 MHz, breaking down the cache operations will still improve the block-out problem. Figure 3 shows the improvement in the McBSP's performance with this frequency ratio.





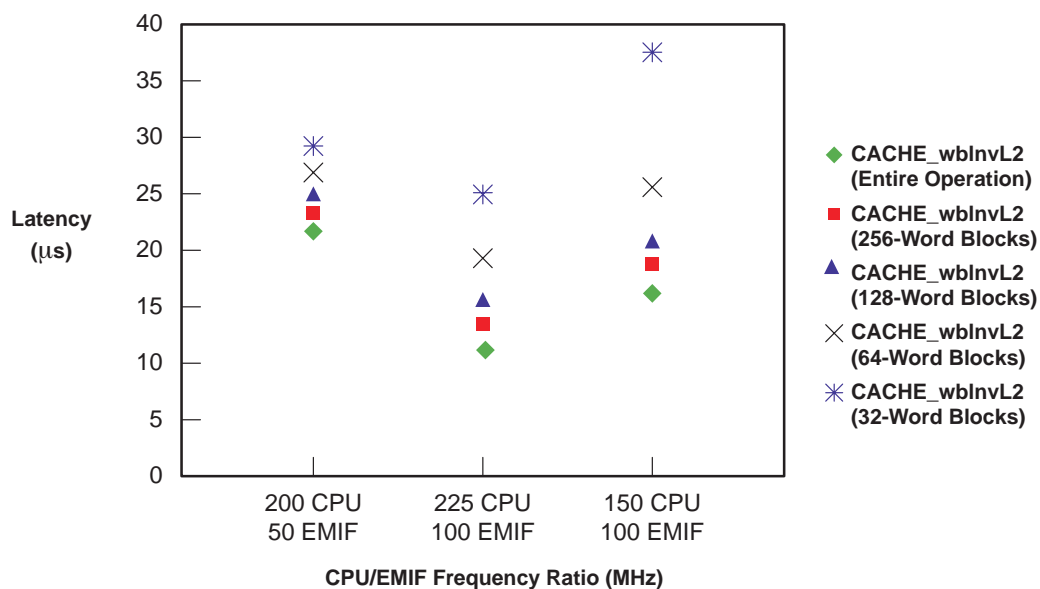
**Figure 3. Minimum Required Latency Between McBSP Events for a Successful Transfer With Concurrent Writeback-Invalidates at 225-MHz CPU and 100-MHz EMIF, Using Entire Operations and Block Breakdown**

When the CPU is set to 150 MHz and the EMIF is set to 100 MHz, there is virtually no benefit from breaking down the coherency cache operations. Figure 4 shows the McBSP’s performance with this frequency ratio.



**Figure 4. Minimum Required Latency Between McBSP Events for a Successful Transfer With Concurrent Writeback-Invalidates at 150-MHz CPU and 100-MHz EMIF, Using Entire Operations and Block Breakdown**

Breaking down the cache operations into smaller blocks takes longer to complete than performing the entire cache function as one large block. Figure 5 shows how much extra overhead is incurred by breaking down an L2 writeback-invalidate operation to transfer 1024 words with different sized blocks and at various frequency ratios. Notice that for the 200-MHz CPU and the 50-MHz EMIF frequency ratio, where the new functions are most critical for peripherals such as the McBSP, the least overhead is incurred.



**Figure 5. 1024-Word L2 Writeback-Invalidate Performance at Various Frequency Ratios With Old and New CACHE-wbInvL2()**

To avoid cache operations blocking other time-sensitive EDMA accesses, observe the following guidelines:

1. Avoid placing real-time data in EMIF address range. Instead, real-time data should be placed in the L2 address range.
2. If data must be placed in the EMIF address range:
  - Avoid global cache operations in favor of block-based cache operations.
  - Block-based cache operations should be submitted in small blocks, such that the total amount of time that the EMIF is blocked is less than the amount of time between serial samples.

### HPI: Illegal Memory Access Can Result in Unexpected HPI Behavior

On all silicon revisions of C6711/C6711B/C6711C/C6711D, the DSP has a reserved memory range that is mapped to the internal FIFO of the HPI for EDMA engine usage. This reserved memory range is located at 0x60000000 – 0x7FFFFFFF in the memory map. If CPU code (or a host access) happens to read/write from/to this memory range, the internal HPI state machine can be corrupted, causing one or more of the following occurrences:

- Host reads/writes through the HPI fail. HPI reads return incorrect data, and/or HPI writes result in incorrect data being written.
- Host reads/writes through the HPI take an unexpectedly long time. The  $\overline{\text{HRDY}}$  signal stays high (not ready) for an extended period of time.
- HPI locks up.  $\overline{\text{HRDY}}$  stays high indefinitely.

The most common cause of this illegal access is uninitialized or stray pointers. To verify that the DSP program does not perform this illegal memory access, the user can use the Advanced Event Triggering tools featured in Code Composer Studio™ Integrated Development Environment (IDE) version 2.1 or later, with the latest emulation driver. Below are the step-by-step instructions on how to trap a CPU access to the memory range 0x60000000 – 0x7FFFFFFF:

1. Start Code Composer Studio™ IDE with the proper setup and GEL file.
2. Load the program.
3. Under the *Tools* menu, select Advanced Event Triggering > Event Analysis
4. Right-click on the bottom left panel that appears, select “Set Hardware Watchpoint”.
5. Name the watchpoint; choose to watch for “Data Memory Reads” or “Data Memory Writes”; select the inclusive range start address (0x60000000) and end address (0x7FFFFFFF); select the data size from 32-, 16-, or 8-bit to watch for word, halfword, or byte reads/writes, respectively. Then, click Apply.
6. The watchpoint now is enabled, indicated by the blue “E” icon. Now, run the program.
7. When a read/write to the specified memory range is detected, the CPU halts, and the blue “E” icon changes to a red “T” icon.

#### Notes:

The CPU halts a few cycles **after** the specified memory access is detected. Without a CPU stall, the number of cycles is around 4 cycles. This means that when the CPU halts, the PC points to a few instructions after the one that caused the trap to trigger.

The hardware watchpoint restricts the trap to be set up for either read or write accesses, but not both. Therefore, the user may need to repeat this procedure several times for each read and write trap.

The above step-by-step method only catches illegal accesses made by the CPU, and does **not** catch illegal accesses made by the EDMA (through the McBSP or the HPI).

### BUSREQ Asserted During HPI Boot

On C6711D silicon revision 2.0 and C6711C silicon revision 1.1, the EMIF Bus Request (BUSREQ) signal goes active during an HPI boot when the internal SDRAM refresh counter reaches zero, even though the EMIF does not default to SDRAM use. The BUSREQ remains active until the first EMIF access after the HPI boot completes.

The BUSREQ signal is typically only used by systems that share external memory between several DSPs. When constructing such a system the false assertion of BUSREQ during reset should be taken into account when designing the external arbitration logic.

Code Composer Studio is a trademark of Texas Instruments.

### Cache Configuration (CCFG) Register — P-Bit Function [New Enhancement]

On C6711D silicon revision 2.0, the “P” bit in the cache configuration (CCFG) register ensures that requests to L2 memory by the transfer crossbar (TC) [such as the important class of TC accesses originating from the EDMA controller] will have priority over the L1D requests to L2 memory. As noted in “*Workaround*” of Advisory 1.1.10, *EDMA: EDMA Blocked from Accessing L2 During Long String of Stores to the Same Bank in L2 RAM*, without the “P” bit set it is possible to construct a loop that creates a long string of stores to the same bank in L2 RAM which could block an EDMA transfer and cause a missed deadline on time critical peripherals such as the McBSP.

**Step 2** in the “*Workaround*” of Advisory 1.1.10, “*EDMA: EDMA Blocked from Accessing L2 During Long String of Stores to the Same Bank in L2 RAM*”, explains how to use the compiler switch to identify and fix these types of loops.

With the “P” bit set on silicon revision 2.0 and later, the problem described in Advisory 1.1.10 does *not* occur because in general priority is given to the TC over the L1D when the “P” bit is set. However, there still is one case where the TC request can be *stalled* for a very short duration while stores from L1D to the L2 RAM are taking place.

This “stalled” case occurs when the L1D is storing data to the same block of data in the L2 cache as the TC is attempting to access. The blocks of data in L2 consists of **contiguous** groups of 32 words aligned at 32-word boundaries. When both the L1D and TC are operating on the same block of data in L2, the L1D has priority even if the “P” bit is set.

Since this priority inversion is bounded by the time it takes for the L1D to complete writing 32 words to L2 memory (256 cycles assuming bytes are written or approximately 1.3  $\mu$ s for a 200-MHz DSP clock rate).

It is recommended, as good practice, to avoid this delay by design. This can be accomplished simply by locating the current buffer on which the DSP is operating in a different 32-word aligned, 32-word block of data of L2 RAM than the current buffer on which the EDMA is transferring (through TC).

### EMIF: Control Signals Not Inactive Before Asserting $\overline{\text{HOLDA}}$

On all silicon revisions of C6711/C6711B/C6711C/C6711D, when the  $\overline{\text{HOLDA}}$  signal is asserted in an EMIF clock cycle, the EMIF deasserts its control signals. This deassertion may cause the control signals to float in the asserted state as well as cause undesired memory accesses.

To prevent the possible actions caused by the EMIF deasserting its control signals, connect a weak pullup resistor (1 k $\Omega$ ) to each  $\overline{\text{CE}}$  pin where the  $\overline{\text{HOLDA}}$  signal is used.

### EMIF: Data Corruption can Occur in SDRAM When the HOLD Feature is Used

On all silicon revisions of C6711/C6711B/C6711C/C6711D, data can be corrupted in the SDRAM found on the EMIF when the HOLD features is used. When the SDRAM refresh counter, found in the EMIF, expires around the same time a  $\overline{\text{HOLD}}$  request is asserted, the DSP starts a refresh of tee SDRAM. Before the  $t_{\text{RFC}}$  specification is met, the DSP generates a DCAB command and asserts  $\overline{\text{HOLDA}}$ , thus violating the  $t_{\text{RFC}}$  specification for SDRAM.

Since both the DSP and the other processor can act as a master, external arbitration logic is needed. Three options for providing external arbitration logic exist:

- Program the arbitration logic to take care of SDRAM refresh. Disable refresh on DSP. Since the DSP is no longer responsible for refresh of SDRAM, the arbitration logic ensures  $t_{\text{RFC}}$  specification is not violated.

- Use one of the DSP internal timers to provide an output signal to the arbitration logic that indicates refresh is pending. The arbitration logic would then be responsible for de-asserting  $\overline{\text{HOLD}}$  and starting its own timer to estimate when the refresh operation has completed. Once the timer within the arbitration logic expires, the arbitration logic should assert  $\overline{\text{HOLD}}$  if needed.
- Use two of the DSP internal timers to output two signals that indicate the start and end of a refresh operation to the arbitration logic. The arbitration logic would then be responsible for de-asserting  $\overline{\text{HOLD}}$  between the start and end of a refresh operation.

### $\overline{\text{RESET}}$ Pin Has No Internal Pullup Resistor

On the C6711D silicon revision 2.0, the  $\overline{\text{RESET}}$  pin *does not* have an internal pullup resistor.

When designing a new PCB using C6711D silicon revision 2.0, either incorporate a voltage supervisor that drives when in the inactive state ( $\overline{\text{RESET}} = 1$ ) or include an external pullup resistor on the  $\overline{\text{RESET}}$  pin.

### Boundary Scan: IDCODE is Only Loaded Onto Instruction Register When $\overline{\text{TRST}}$ Becomes Inactive

The IDCODE instruction is loaded into the instruction register only when the Test Logic Reset state is entered by transitioning  $\overline{\text{TRST}}$  from low to high, but not when the Test Logic Reset state is entered by holding TMS high and clocking TCK five times.

### JTAG, Clock/PLL Oscillator, McBSP0/1, and TIMER1: MCBSP1DIS Control Bit (DEVCFG.0) Affects IPU/IPDs on Specific Peripheral Pins

On the C6711D silicon revision 2.0, if a 1 is written to the MCBSP1DIS control bit (bit 0) of the DEVCFG register, the pins listed in Table 3 lose their internal pullup or pulldown resistor and external resistors must be added.

If the pins that are listed in Table 3 are not continuously driven after  $\text{DEVCFG.0} = 1$ , an external 10-k $\Omega$  pullup or pulldown resistor is required to maintain a valid logic level.

**Table 3. IPU/IPD Condition of JTAG, Clock/PLL Oscillator, McBSP0, McBSP1, and Timer1 if  $\text{DEVCFG.0} = 1$**

PIN NAME	IPU/IPD DESCRIPTION
<b>JTAG Pins</b>	
If $\text{DEVCFG.0} = 1$ :	
$\overline{\text{TRST}}$	<p><b>CAUTION:</b></p> <p>If <math>\text{DEVCFG.0} = 1</math>, <math>\overline{\text{TRST}}</math> <b>must</b> be externally pulled down to ensure the device scan chain logic <b>does not</b> become active. Even if <math>\text{DEVCFG.0} = 0</math> and <math>\overline{\text{TRST}}</math> is routed out and not driven, it is recommended that this pin be externally pulled down.</p> <p>IPD Removed</p>
TMS	IPU Removed
TDO	IPU Removed
TDI	IPU Removed
TCK	IPU Removed

**Table 3. IPU/IPD Condition of JTAG, Clock/PLL Oscillator, McBSP0, McBSP1, and Timer1 if DEVCFG.0 = 1 (Continued)**

PIN NAME	IPU/IPD DESCRIPTION
<b>Clock/PLL Oscillator Pins</b>	
If DEVCFG.0 = 1:	
CLKIN	IPD Removed
CLKOUT3	IPU Removed
<b>McBSP0 Pins</b>	
If DEVCFG.0 = 1:	
CLKR0	IPD Removed
CLKX0	IPD Removed
FSR0	IPD Removed
FSX0	IPD Removed
<b>McBSP1 Pins</b>	
If DEVCFG.0 = 1:	
CLKX1	IPD Removed
FSX1	IPD Removed
<b>Timer1 Pins</b>	
If DEVCFG.0 = 1:	
TINP1	IPD Removed

## 2.2 C6711D Silicon Revision 2.0 Known Design Exceptions to Functional Specifications

There are no known advisories for the TMS320C6711D silicon revision 2.0.

### 3 C6711C Silicon Revision 1.1 Known Design Exceptions to Functional Specifications and Usage Notes

#### 3.1 Usage Notes for C6711C Silicon Revision 1.1

All usage notes for silicon revision 1.1 still apply and have been moved up to the *Usage Notes for C6711D Silicon Revision 2.0* section of this document except the “EPI/HPI: EMIF Output and HPI Signals Can Drive During Reset” usage note, which is applicable to silicon revision 1.1 only.

##### EMIF/HPI: EMIF Output and HPI Signals Can Drive During Reset

On silicon revision 1.1 of C6711C, the EMIF *output* pins and HPI pins can drive the bus for a small amount of time while the external  $\overline{\text{RESET}}$  signal is asserted. In normal operation, the EMIF output and HPI pins should be 3-stated when the external  $\overline{\text{RESET}}$  signal is low until the internal reset signal is deasserted. However, the actual C6711C silicon revision 1.1 behavior is as follows:

- The EMIF output pins become properly 3-stated on the first SYSCLK3 clock edge after the external reset deassertion. Note that SYSCLK3 oscillates at CLKIN/2 by default after the external reset signal is deasserted.
- The HPI pins become properly 3-stated three cycles of CLKIN/8 frequency after the external  $\overline{\text{RESET}}$  signal is asserted (CLKIN is the input clock to the DSP).

However, if these EMIF output and HPI signals are 3-stated at the time that the external reset goes low, these signals stay 3-stated and not driven.

This behavior has been corrected in silicon revision 2.0.

#### 3.2 C6711C Silicon Revision 1.1 Known Design Exceptions to Functional Specifications

##### Advisory 1.1.1

*HPI: HPID Read/Write Accesses Must Be Terminated With a Fixed-Mode Access*

**Revision(s) Affected:** 2.0 and 2.1 [for C6711B] and 1.1 [for C6711C]

**Details:** The auto-increment HPI read utilizes an internal buffer and prefetch mechanism to increase throughput. The prefetch mechanism may conflict with the internal buffer flush that is caused by HPIA or HPIC write. This conflict may cause the next auto-increment read access to return stale data in the first few words.

This conflict may also cause the HPI to lock up ( $\overline{\text{HRDY}}$  staying high indefinitely)

**Workaround:** Terminate every auto-increment HPID read with a fixed-mode HPID read, and terminate every HPID write with a fixed-mode HPID write. For example, to read 14 words in auto-increment mode, do *not* do:

```
HPIA write
HPID++ read (1st word, autoincrement)
HPID++ read (2nd word, autoincrement)
...
HPID++ read (14th word, autoincrement)
HPIA write (set up HPIA for next access)
HPID++ read
...
```



*HPI: HPID Read/Write Accesses Must Be Terminated With a Fixed-Mode Access (Continued)*

But, do this instead:

```
HPIA write
HPID++ read (1st word, autoincrement)
HPID++ read (2nd word, autoincrement)
...
HPID++ read (13th word, autoincrement)
HPID read (14th word, FIXED-MODE)
HPIA write (set up HPIA for next access)
HPID++ read
...
```

This brings the HPI to a clean state after an auto-increment access. The next HPIA / HPIC write will not conflict with in-flight data from previous HPID++ read/write.

Note that different host mechanisms (software/hardware) are needed to migrate between systems with revisions 2.0/2.1 and revisions 1.0, 1.2, 1.3 silicons [C6711B and C6711, respectively] and 1.1 silicon [C6711C], due to advisories 1.1.1, 1.3.2, and 1.3.7.

### Advisory 1.1.3

*RESET Pin Has Internal Pullup Resistor*

**Revision(s) Affected:** 1.1 [for C6711C]

**Details:** The RESET pin has an internal pullup resistor with a typical value of 18 kΩ. This pullup resistor is problematic because it is stronger than the weak pulldown (10 μA near 1 V) typical of most voltage supervisors.

**Workaround:** Any new PCB designs should incorporate either a voltage supervisor with an active drive on RESET or include an external pullup resistor on the RESET pin. The internal pullup resistor has been removed in silicon revision 2.0.

### Advisory 1.1.4

*PLL: Incorrect PLL Controller Peripheral Identification (PID) Register Value*

**Revision(s) Affected:** 1.1 [for C6711C]

**Details:** The PLL Controller Peripheral Identification (PID) value is incorrect. The PLL PID register currently reads CLASS = 0x01, TYPE = 0x10. It should read CLASS = 0x08, TYPE = 0x01.

**Workaround:** None. This will be corrected in a future silicon revision.

**Advisory 1.1.10**

*EDMA: EDMA Blocked from Accessing L2 During Long String of Stores to the Same Bank in L2 RAM*

**Revision(s) Affected:** 1.0, 1.2, 1.3, 2.0, and 2.1 [for C6711/C6711B] and 1.1 [for C6711C]

**Details:** If the CPU is storing data to a bank of L2 memory on the same cycle that the EDMA is trying to access the same bank, the CPU will always be given priority. (For example, the EDMA will be blocked from accessing that bank until the CPU access is complete.) Access will be programmable in silicon revision 2.0. (Note that the EDMA and CPU can access different banks of L2 on the same cycle.) If the CPU stores to the same bank on every cycle for a long period of time, an EDMA access to that bank can be blocked long enough to miss a hard deadline.

L2 memory is organized as 4 banks with each bank 64 bits wide (see Figure 6).

Bank 0		Bank 1		Bank 2		Bank 3	
0x00000000	0x00000004	0x00000008	0x0000000C	0x00000010	0x00000014	0x00000018	0x0000001C
0x00000020	0x00000024	0x00000028	0x0000002C	0x00000030	0x00000034	0x00000038	0x0000003C
...		...		...		...	
0x0000FFE0	0x0000FFE4	0x0000FFE8	0x0000FFEC	0x0000FFF0	0x0000FFF4	0x0000FFF8	0x0000FFFC

NOTE: Each address in Figure 6 is an address for a 32-bit word.

### Figure 6. L2 Memory Organization

A conflict occurs when the CPU is trying to access a bank of L2 on the same cycle as the EDMA is trying to access the same L2 bank. For example, if the CPU were trying to store a 32-bit word to location 0x0000 0000, and on the same cycle the EDMA is trying to transfer a 32-bit word at location 0x0000 0024 to the McBSP to be transmitted, then a conflict would occur since both are trying to access Bank0. In C6711C silicon revision 1.1, the CPU will always get priority over the EDMA, so the EDMA has to wait for the store to 0x0000 0000 to complete before accessing location 0x0000 0024.

Waiting for a single store to complete would only delay the EDMA access by several cycles, which is not normally a problem. However, a problem can occur when the CPU continually stores data to the same bank of L2 for a long period of time. The problem occurs when the EDMA has a hard deadline to meet, e.g., it must transfer a word from L2 to the McBSP every 5  $\mu$ s. If the duration of the sequence of continuous stores is longer than 5 $\mu$ s, then the EDMA will be blocked from accessing that bank of L2 long enough to miss the deadline and a transmit underrun error will occur in the McBSP.

It should be noted that a series of CPU stores that causes a real-time system problem (an EDMA transfer to miss a deadline) is most likely to occur in looped code. For example, if a particular code segment caused the EDMA to be blocked for four cycles, a system problem caused by the delayed EDMA transfer would likely not occur. If that same code segment were repeated in a loop of 1000 iterations, then the EDMA transfer would be blocked for a total of (4 \* 1000 =) 4000 cycles. In this latter case, the EDMA transfer is more likely to miss a hard deadline causing a system problem.

*EDMA: EDMA Blocked from Accessing L2 During Long String of Stores to the Same Bank in L2 RAM (Continued)*

There are four criteria that must be met in order for a loop to continually block a bank of L2. All of the conditions must be met for the problem to occur.

1. The total duration of time the EDMA is blocked (or the total duration of a loop, in the case of looped code) is close to or longer than the hard deadline.
2. In a given sequence of code, the total number of stores must be greater than or equal to the number of cycles on which no store occurs. In the case of looped code, in one iteration of a loop, the total number of stores must be greater than or equal to the number of cycles on which no store occurs, or, in other words, the length of one iteration in cycles is less than or equal to twice the number of stores.

Figure 7 outlines this scenario. Note that the STW instruction represents a store of any word size (32-, 16-, or 8-bit).

EDMA: EDMA Blocked from Accessing L2 During Long String of Stores to the Same Bank in L2 RAM (Continued)

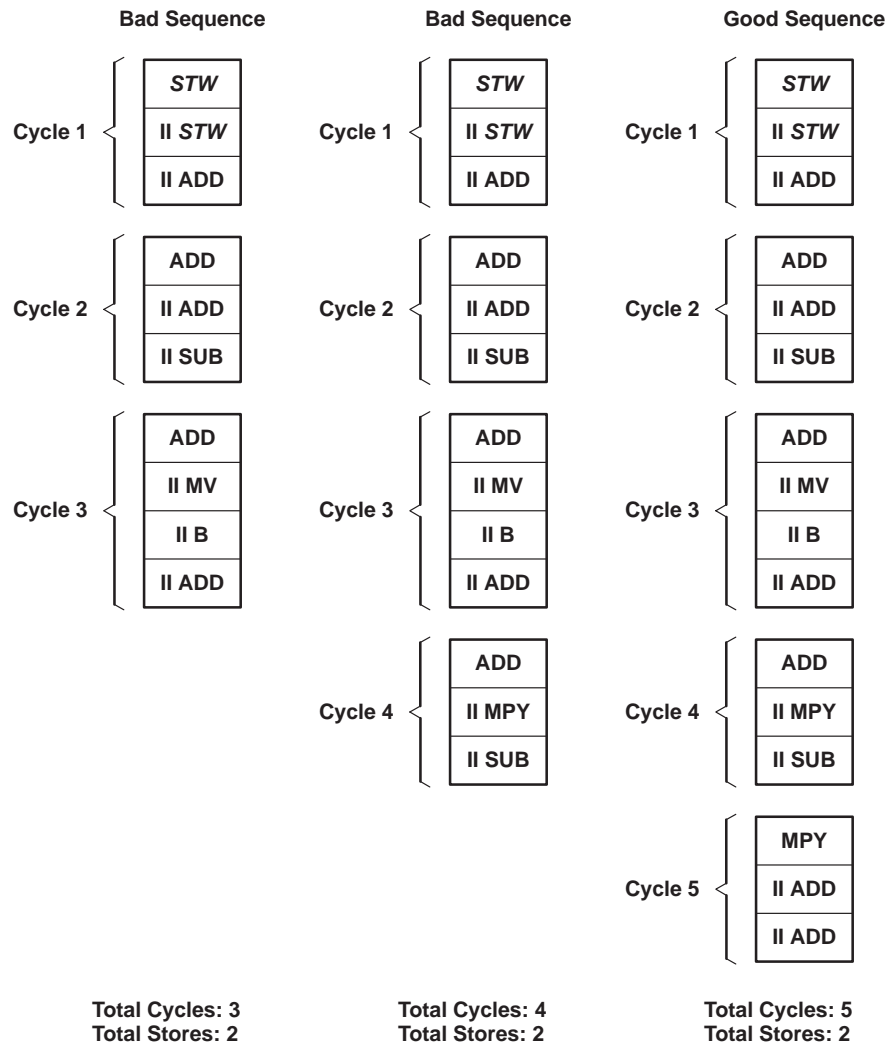


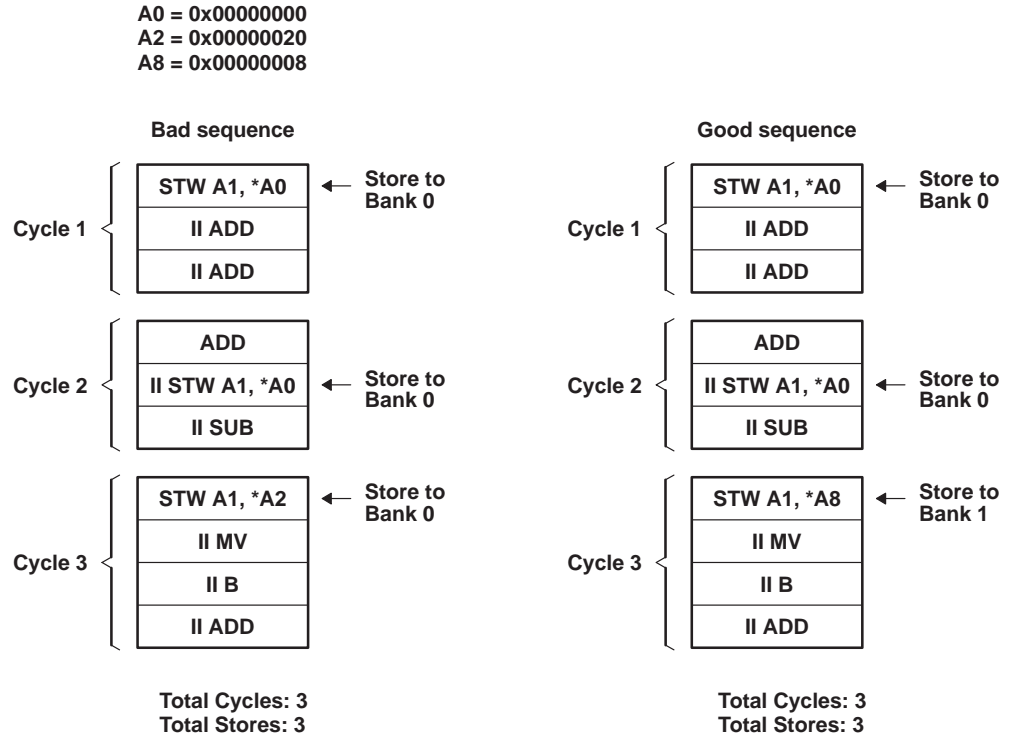
Figure 7. Pseudo Code Example With Parallel Stores (Criteria 2)

In most looped code, more than one instruction would likely be executed on each cycle, i.e., instructions would be executed in parallel. In this case, as long as a store is one of the instructions being executed in parallel on a particular cycle, that cycle counts as a cycle on which a store occurs. Instructions with parallel bars (||) at the beginning of the line of assembly execute in parallel with instructions on the preceding line.

In the C6000 core, up to 2 stores can occur per cycle. In this case, each store must be counted individually. That is, even though both stores occur on the same cycle, they still must be counted as two stores. All other rules apply (see Figure 7).

EDMA: EDMA Blocked from Accessing L2 During Long String of Stores to the Same Bank in L2 RAM (Continued)

- All stores in the loop must be to the same bank of L2. If there are any stores to another bank, this will free the first bank long enough for the EDMA access to get in (see Figure 8).



**Figure 8. Pseudo Code Example Stores to Specified Banks (Criteria 3)**

In the case of two stores occurring on the same cycle, in parallel, the same rules apply. If both stores are to the same bank (and there are no other stores in the sequence to a different bank), then a problem may occur. If each of the parallel stores is to a different bank, then the problem cannot occur.

- There are no loads that miss in L1 (therefore access L2). It does not matter which bank the load is accessing (see Figure 9).

A load from L2 provides enough of a gap to allow the EDMA to access L2. The good sequence would not cause a problem even if the load were executed in parallel with one of the stores, as long as the load occurred somewhere in the sequence. Notice that although the good sequence satisfies criteria 2 and 3, it would not cause a problem.

EDMA: EDMA Blocked from Accessing L2 During Long String of Stores to the Same Bank in L2 RAM (Continued)

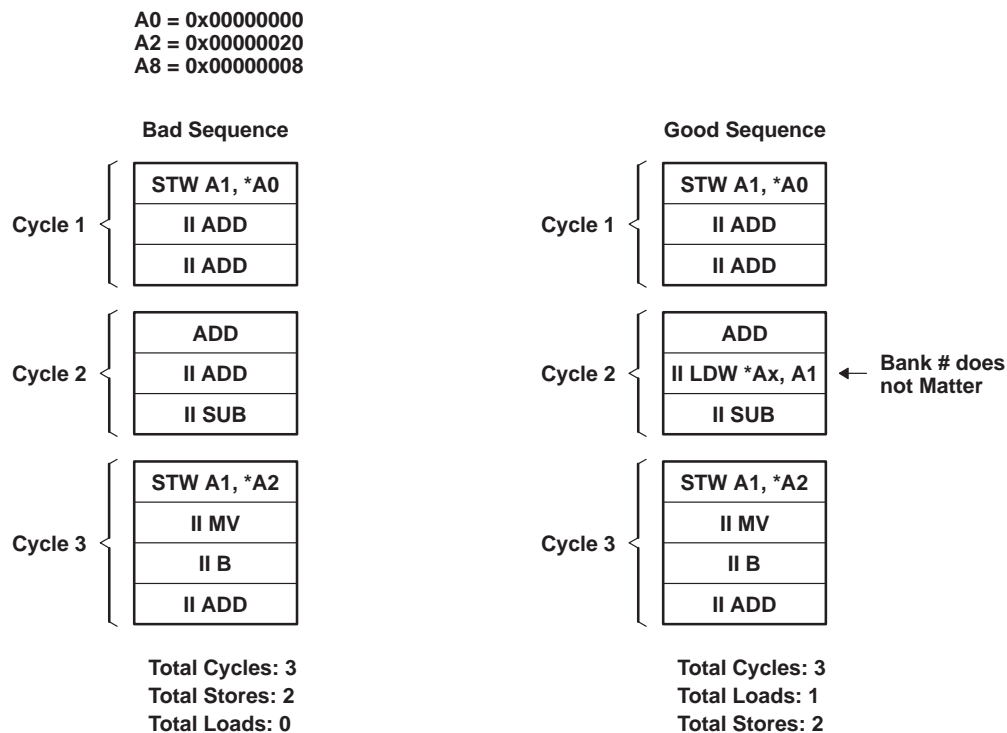


Figure 9. Pseudo Code Example Stores and Loads (Criteria 4)

**Workaround:**

Note: This will be fixed in silicon revision 2.0. The fix in silicon revision 2.0 will require setting a hardware bit to avoid the problem discussed in this erratum. The workaround discussed in this erratum is not necessary if using silicon revision 2.0.)

**Step 1**

Determine the hard deadlines for the system of interest.

**Step 2**

Use the compiler switch `-edma_warnN` to find potential problem loops. **The compiler switch only checks for criteria 1 and 2.**

For more detailed information on the compiler switch, see the *Using the -edma\_warnN Compiler Switch to Detect a CPU L2 EDMA Lockout Application Report* (literature number SPRA916).

The `-edma_warnN` option was not available until the release of Code Composer Studio IDE 2.20.23.

EDMA: EDMA Blocked from Accessing L2 During Long String of Stores to the Same Bank in L2 RAM (Continued)

### Step 3

After Step 2, a list of potential problem loops now exists. The programmer must now examine each of these potential problem loops to see if they meet the two additional criteria for being a problem loop (remember that the compiler switch in step 2 only checked for criteria 1 and 2). This is done by closely examining the source code and assembly output of that source code.

For more detailed information on how to interpret the source code and the assembly output of that source code, see the *Using the `-edma_warnN` Compiler Switch to Detect a CPU L2 EDMA Lockout* Application Report (literature number SPRA916).

### Step 4

The final step is to fix the remaining problem loops. There are a number of fixes that can be implemented, and which fix to implement is highly system dependent. Each fix attempts to break one of the criteria of a problem loop. Only one fix is needed for each loop.

- A. Reduce the duration of the loop by breaking into smaller loops. If a particular problem loop has a length of 4 cycles (determined by looking at the software pipeline kernel in the assembly file), and the loop runs 200 times, then the total loop duration is ~800 cycles. If the deadline is 586 cycles, then loop could cause a problem (assuming the loop meets all the other criteria). The workaround is to break the loop into four smaller loops of 50 iterations each. Then any one loop will only run for 200 cycles allowing the EDMA transfer to complete between the smaller loops.
- B. Try to break criteria 2. The quickest way to do this is to turn off optimization for the file containing the problem loop (this is done in the File Specific Options for the file). Instead of using the compiler switch `-o3`, use the compiler switch `-o1` for the particular problem file. This will cause the problem loop to not be software pipelined and less likely to meet criteria 2. The downside to this fix is that all the loops in the particular file will be un-optimized, not just the problem loop.
- C. Try to break criteria 3. One way of doing this is to rearrange the data structures so that a loop does not have to stride an array by a factor of 8. The other way of implementing this is to possibly combine two different loops so that the array stride becomes a different factor other than 8.
- D. Try to break criteria 4. One way of doing this is to place a dummy load in the loop.

## 4 C6711B Silicon Revision 2.1 Known Design Exceptions to Functional Specifications and Usage Notes

Some silicon revision 2.1 advisories are shared with silicon revision 1.1. See the advisories for silicon revision 1.1 [C6711C].

### 4.1 Usage Notes for C6711 Silicon Revision 2.1

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

#### HPI: Illegal Memory Access Can Result in Unexpected HPI Behavior

This usage note is applicable to silicon revision 2.0 [for C6711D], to silicon revision 1.1 [for C6711C], and to silicon revisions 2.1 and earlier [for C6711/C6711B]. To avoid extensive duplication, see the **HPI: Illegal Memory Access Can Result in Unexpected HPI Behavior** usage note under *Usage Notes for C6711D Silicon Revision 2.0*.

### 4.2 C6711B Silicon Revision 2.1 Known Design Exceptions to Functional Specifications

#### Advisory 2.1.2

*EMIF: Async Read Setup Uses Write Setup Value*

**Revision(s) Affected:** 1.0, 1.2, 1.3, 2.0, and 2.1 [C6711/C6711B]

**Details:** When the EMIF is performing read from async memory and accessing the SDRAM continuously, the EMIF sometimes incorrectly uses the async Write Setup (CECTLx[31:28]) as its Read Setup (CECTLx[19:16]) value.

This advisory only applies when EMIF reads from async memory and performs accesses to SDRAM. Other modes are not affected.

**Workaround:** Use the *same* read setup and write setup value in the appropriate fields of CE Space Control Register (CECTLx).

#### Advisory 2.1.4

*EMIF: One Cycle Asynchronous Write Setup*

**Revision(s) Affected:** 1.0, 1.2, 1.3, 2.0, and 2.1 [C6711/C6711B]

**Details:** The EMIF may give only one EMIF clock cycle of asynchronous write setup, instead of the value programmed in the EMIF CE Space Control Register. This condition *only* occurs when an async write is issued while there is an ongoing async read within the same or different CE space. This may result in async write data corruption if the setup/strobe time of the async memory is not met.



*EMIF: One Cycle Asynchronous Write Setup (Continued)*

**Workaround:** For robust operation, it must be assumed that in the above condition, the EMIF gives only one EMIF clock cycle of write setup to async memory, independent of the write setup value programmed in the CE Space Register. Most asynchronous memory latches write data at the rising edge of  $\overline{WE}$ , which sometimes can workaround the problem. Therefore, care must be taken such that write setup (one cycle) + write strobe (programmable) meets the write setup/strobe requirement of the async memory. Slowing down the EMIF clock may be needed to ensure that the timings meet.

**Advisory 2.1.5***JTAG: Boundary Scan Shift-DR Register is Latched on the Falling Edge of TCK*

**Revision(s) Affected:** 2.0 and 2.1 [C6711B]

**Details:** The Boundary Scan Shift-DR Register is latched on the falling edge of TCK, instead of the rising edge of TCK. This causes boundary scan hardware/software to see an extra cell when the chip is in boundary scan mode.

**Workaround:** The BSDL files for this part have been modified to reflect this advisory.

**Advisory 2.1.6***EDMA/L2 Controller: Potential EDMA Lockout From L2 SRAM*

**Revision(s) Affected:** 1.0, 1.2, 1.3, 2.0, and 2.1 [C6711/C6711B]

**Details:** Under certain conditions, an EDMA transfer can be denied the use of L2 for a longer period of time than expected. This can cause the EDMA to miss data transfers that have real-time service requirements (such as certain serial port transfers). This condition may manifest itself in the following phenomena:

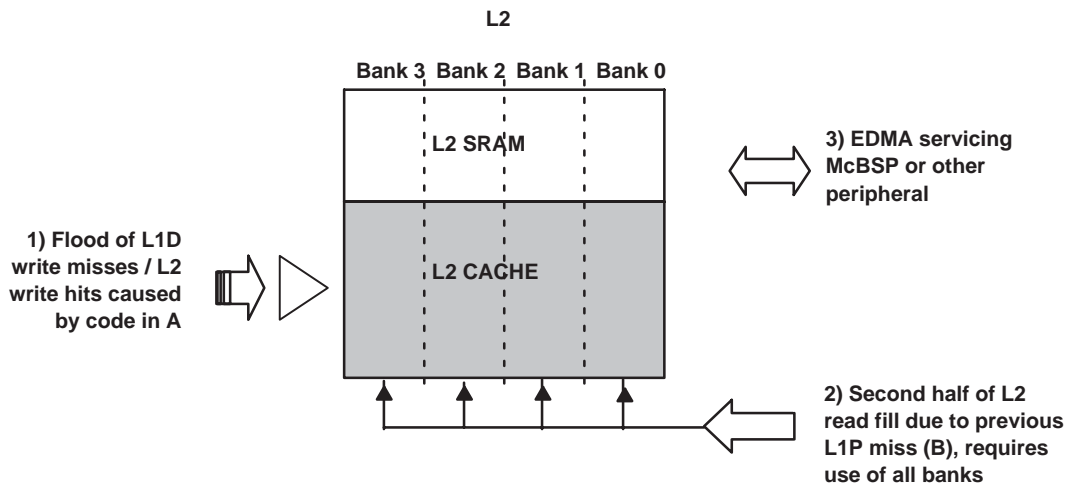
- McBSP transmits repeated data (transmit underrun)
- McBSP receives data late or misses data
- Audio channels in TDM mode “rotate”
- EDMA transfers triggered by timers or external interrupts do not happen or happen late
- EDMA transfers take longer than expected
- EDMA events are missed

EDMA/L2 Controller: Potential EDMA Lockout From L2 SRAM (Continued)

This problem is very sensitive to code/execution alignment. Any minor edit to code can make the problem seem to disappear. The scenario is *only* applicable when *all* of the following five conditions are true:

1. L2 is allocated as *both* cache and SRAM.
2. EDMA is transferring into or out of L2 SRAM
3. There is an L1P miss that also misses in L2, and must be fetched from external memory
4. This “missed” instruction is immediately followed by a section of code that produces a flood of stores that miss L1D, but hit in L2 cache or L2 SRAM
5. The “flood of stores” happens at a rate of at least 1 every other CPU cycle

In this scenario, the next CPU fetch packet does not reside in either L1P or L2 cache; therefore, it causes an L1P and L2 miss. An L2 cache line (128 bytes, 4 fetch packets) must be fetched from external memory. The fetch from external memory is split into two requests (two halves) of 64 bytes, or two fetch packets each. In this document, the two halves are labeled A and B, respectively, as shown in Figure 10. The first half (A) to arrive in L2 is the half in which the missed fetch packet resides. This first half is immediately forwarded to L1P so that CPU can resume execution before the second half (B) is fetched from external memory. The second half of the L2 line (B) is to be fetched and written to L2 later.



Note: The #) Indicates the order of Occurrence/Priority

Figure 10. Problem Scenario

At this time, the CPU executes the newly returned code in A (in the first 2 fetch packets returned). The cache architecture increases write throughput by not performing write-allocate in case of L1D miss. All CPU stores that miss in L1D are sent directly to L2. This code in A contains a series (flooding) of stores that miss L1D, but hit in L2. These stores are generated at a rate of at least 1 every other cycle.

*EDMA/L2 Controller: Potential EDMA Lockout From L2 SRAM (Continued)*

The lockout condition occurs only when the second half (B) of the L2 cache line fill starts. When the B is ready, it attempts to write to L2. It will be unable to do so, however, because the line fill operation accesses all four banks of L2 whereas one bank is always busy servicing the higher priority L1D misses from the code in A. Thus, the line fill (B) cannot complete until the flood of writes from A has stopped. The line fill stays at the head of queue in the EDMA controller, so any other EDMA transfers that access L2 will also be blocked until the flood of stores has stopped *and* the L2 line fill (B) has completed.

The amount of time that an EDMA transfer is locked out of L2 is determined by the code running on the DSP (A), and ultimately by the amount of time that L1D write misses are happening in succession. When all of the conditions are met, the second half of the L2 line fill and the EDMA will be locked out of L2 as long as the flood of stores is sustained.

Certain library functions that satisfy all of the above problem conditions may fall into this EDMA lockout problem. As an example, the memcpy library routine involves a series of CPU writes at a rate  $\geq 1$  write per every 2 cycles. THE EDMA lockout problem may exist if all other conditions are satisfied. The workarounds stated below may also be used in this case.

**Exceptions:**

The lockout problem does not exist if L2 is configured to be all cache or all SRAM:

- If L2 is all cache, then EDMA transfers servicing other peripherals do not access L2. They access external memory directly, and lockout does not occur. Since the DSP does not keep external coherency, L1/L2 caches do not know of external memory being modified. If this section in external memory is cached, frequent cache cleans may be needed so that the caches are coherent with external memory.
- If L2 is all SRAM, then there will not be second half of L2 fill (B). When there is L1P miss, L2 controller will only fetch L1P line size in a single request. The cache line is immediately passed to L1P by the L2 controller.

The flooding of write misses by itself cannot lock out the EDMA from L2 for an extended period of time. The EDMA and the write misses can access L2 concurrently, provided that the EDMA and the L1D write misses access different L2 banks.

NOTE: The lockout does not occur for L1D read misses in A that hit or miss in the L2. If an L1D read miss hits in L2, there is no L2 line fill to trigger the problem. Lockout also cannot happen if an L1D read miss also misses L2. For coherency reason, in this case L2 prevents the flood of stores from happening by stalling the CPU until the entire L2 line fill has completed.

**Workaround:**

If the problem exists, perform the following steps:

1. Use the simulator tool described below to detect and pinpoint locations of potential problems.
2. At the code locations where the simulator points, perform the software workarounds described below.
3. Apply the compiler tools workaround described below. Use this step *only* if no other workaround is feasible.

*EDMA/L2 Controller: Potential EDMA Lockout From L2 SRAM (Continued)***Software Workaround***Code Modification:*

- Avoid the flood of L1D write misses. Pre-read all data locations prior to writing them. This will allocate them in L1D, preventing the L1D write misses from occurring. Recall that L1D read misses do not cause problems.
- Make sure that any algorithm or code segment that naturally produces a long string of writes inserts gaps in the write sequence such that writes occur at a frequency slower than 1 write miss every other cycle. The term "long" is based on the system requirements. Gaps in the L1D write misses means that the second half of the L2 read fill is given a chance to access L2.

*Code Location:*

- Run the algorithm or code segment that contains A from internal SRAM. There will not be L2 code read fill (A or B) coming into L2. The simulator can be used to detect such code segment. See simulator section below for details.
- Ensure the problematic segment of algorithm or code is in cache. This can be accomplished by issuing any CPU read/load instruction from the beginning of the problematic code segment in external memory, that will cause both L1D and L2 miss. Assuming the code segment is not yet in L1/L2 cache, Performing CPU load instruction generates an L1D request, and CPU stalls until the code missed and data is returned from L2. After the load instruction, code segment will reside in the L2 cache, and an L2 read fill due to L1P miss will not occur.

**Tools***C6x1x Simulator*

The simulator can be used to detect potential EDMA lock-out problem. This capability is featured in the simulator as part of Code Composer Studio version 2.1 package.

This version of simulator includes the detection logic to monitor the amount of CPU stores that miss in L1D but hit in L2 cache or SRAM, which could possibly lock out the EDMA from accessing L2 because of the conditions described above.

The user can input a window size limit ( $W$ ) in which the stores are monitored. During simulation run, if  $W/2$  or more of such stores occur within  $W$  cycles, simulator will halt and a warning message will be printed. A log file is automatically generated and stored as "L2\_write\_hits.log" in the CCS "Start in Directory" (default to MYPROJECTS). The user can run past the warning to find other potential problems, but the log file will have only the last case detected.

*EDMA/L2 Controller: Potential EDMA Lockout From L2 SRAM (Continued)*

There are two ways to activate this capability:

- The user can manually set the window size ( $W$ ) values in the address location 0x60000000. For example, the following code can be used in the command window to set the window size of L2 store hits to 15:  

```
?*(int *) 0x60000000 = 15;
```
- The GEL file `init6x1x.gel` file (see Example GEL File) can be used to automatically add the EDMA lockout detection capability in the simulator. Upon loading this GEL file, a new menu item will appear under GEL → EDMA Lockout Detection Warning → Window\_Size\_In\_Cycles. Upon activating the menu item, a dialog window will appear in which the user can input the window size  $W$  (in cycles) to monitor the L1D–L2 writes.

By default, this memory location (0x60000000) has zero value, which means the monitor is disabled. Writing non-zero values to this location enables the monitor, and writing a zero disables it.

*C6000 Compiler*

The C compiler / codegen tools can be used to prevent this EDMA starvation problem from happening by using the `–edma_bug` compiler option. Starting from version Code Composer Studio™ IDE 2.1 (compiler revision 4.2) of the tools, this compiler option is featured:

- `–edma_bugN`, where  $N$  is an optional parameter which specifies the number of CPU cycles between the EDMA accesses being starved. So if EDMA service is needed every 10000 cycles, then the user can do `–edma_bug10000`.

- NOTES:
1. Compiling code with this option affects code size and performance. Only use this option if problem exists and no workarounds are feasible.
  2. `–mv6211` option needs to be used with the `–edma_bugN` option.
  3. This new option ensures that any software pipelined loop that takes up more than  $N$  cycles are modified such that there is less than 1 store every 2 cycles.
  4. Any software pipelined loops that are known to be shorter than  $N$  cycles are unaffected.
  5. This compiler option only affects software-pipelined code. Any store instructions outside of a software-pipelined loop are not affected by this option.
  6. Hand-coded assembly or already compiled object code will *not* be automatically corrected by this option. The simulator can be used on such code to detect potential problems.

**Example GEL File**

```
/*
 * init6x1x.gel
 *
 * This GEL file (init6x1x.gel) is loaded on the command line of Code Composer. It provides example code
 * on how to reset the C6x DSP and initialize the External Memory Interface. You may have to edit
 * settings in emif_init() to your own specifications as the example is applicable to the C6211/6711 DSK.
 *
 * The StartUp() function is called every time you start Code Composer. You can customize this function to
 * initialize wait states in the EMIF or to perform other initialization.
 */
StartUp()
{
```

```

/* uncomment the following line to initialize the
   EMIF registers on the C6x when Code Composer starts up */
emif_init();
dma_lockout_init();
}
/*
 * Menuitem creates a selection available beneath the GEL menu selection in Code Composer Studio.
 */
menuitem "Resets";
hotmenu ClearBreakPts_Reset_EMIFset()
{
    GEL_BreakPtReset();
    GEL_Reset();
    emif_init();
}
/*****/
emif_init()
{
/*-----*/
/* EMIF REGISTER VALUES FROM SPRU269B */
/*-----*/
#define EMIF_GCTL          0x01800000
#define EMIF_CE1          0x01800004
#define EMIF_CE0          0x01800008
#define EMIF_CE2          0x01800010
#define EMIF_CE3          0x01800014
#define EMIF_SDRAMCTL     0x01800018
#define EMIF_SDRAMTIMING  0x0180001C
#define EMIF_SDRAMEXT     0x01800020
*(int *)EMIF_GCTL = 0x00003040; /* EMIF global control register */
*(int *)EMIF_CE1 = 0x40f40323; /* CE1 - 32-bit asynch access after boot*/
*(int *)EMIF_CE0 = 0xFFFFFFFF30; /* CE0 - SDRAM */
*(int *)EMIF_CE2 = 0x40f40323; /* CE2 - 32-bit asynch on daughterboard */
*(int *)EMIF_CE3 = 0x40f40323; /* CE3 - 32-bit asynch on daughterboard */
*(int *)EMIF_SDRAMCTL = 0x07117000; /* SDRAM control register (100 MHz)*/
*(int *)EMIF_SDRAMTIMING = 0x0000061A; /* SDRAM Timing register */
}
/*
 * Menuitem creates a selection available beneath the GEL menu selection in Code Composer Studio.
 */
menuitem "EDMA Lockout Detection Warning";
dialog L2_Store_Hits( win_l2 "Window size in cycles" )
{
#define L2_STORE_HITS_WINDOW_REG_ADDR    0x60000000
*(int *)L2_STORE_HITS_WINDOW_REG_ADDR = win_l2;
}
dma_lockout_init()
{
#define L2_STORE_HITS_WINDOW_REG_ADDR    0x60000000
*(int *)L2_STORE_HITS_WINDOW_REG_ADDR = 0x0;
}

```

## 5 C6711B Silicon Revision 2.0 Known Design Exceptions to Functional Specifications

### Advisory 2.0.1

#### L1D Cache: Data Corruption if L1D Powered up to Wrong State

**Revision(s) Affected:** 1.0 [C6711] and 2.0 [C6711B]

**Details:** On TMS320C6711 devices, certain test logic associated with the L1D controller may power up initialized to an incorrect state. In this case, any of the following conditions will cause data corruption:

- An L1D load hit occurs during a stall
- An L1D load miss occurs during a stall
- An L1D store miss occurs during a stall

The data at the address pointed to by the current load or store operation will be corrupted with the last value present on the associated store data bus. Although infrequent, stalls can have a variety of causes, such as: program-cache miss, data-cache miss, and data-cache store buffer stall. Thus, there is no viable software workaround.

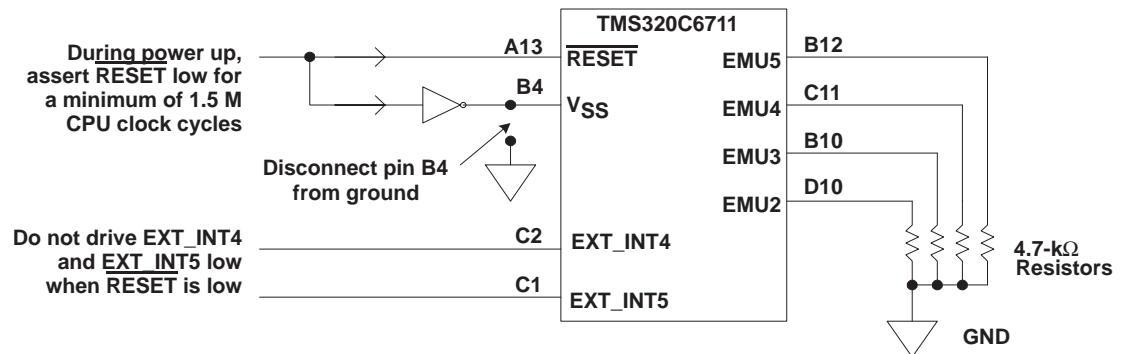
Data corruption is consistent once the L1D is powered up to the incorrect state. A power cycle may correct the problem if the L1D is initialized to a correct state. For devices and environments sensitive to this problem, the L1D is typically powered up to the invalid state approximately 60 percent – 80 percent of the power cycles. The device may be sensitive to how long it was powered down or powered on, as well as to other factors such as temperature.

**Workaround:** Perform one of the following two workarounds:

#### 1. Board Modification

Note that all of the following *must* be removed on corrected silicon (once available). Perform all of the following hardware modifications to ensure that L1D is powered up to a correct state (see Figure 11):

- Disconnect pin B4 (currently a  $V_{SS}$  ground pin) from ground. Tie pin B4 to the inverse of  $\overline{\text{RESET}}$  (pin A13). Pin B4 is actually a test pin that controls the internal test logic. It was documented as a  $V_{SS}$  ground pin to prevent users from accidentally enabling the device test modes. To ensure that the L1D controller is in the correct state, pin B4 must be asserted high to force the device into a test mode during device reset ( $\overline{\text{RESET}}$  asserted low). Upon release from reset ( $\overline{\text{RESET}}$  deasserted high), pin B4 needs to be deasserted low to return to the normal functional mode.
- Pull down EMU[5:2] (pins B12, C11, B10, and D10) to ground with 4.7-k $\Omega$  resistors.
- During power up,  $\overline{\text{RESET}}$  (pin A13) must be asserted for a minimum of 1.5 M CPU cycles. For example, when running parts at 150 MHz (6.67-ns CPU cycles), assert  $\overline{\text{RESET}}$  low for a minimum of 10 ms.
- While  $\overline{\text{RESET}}$  is active low, inputs EXT\_INT4 and EXT\_INT5 should be high to allow the test mode to run to completion. Both pins include internal pullups, so undriven inputs will meet this requirement.

*L1D Cache: Data Corruption if L1D Powered up to Wrong State (Continued)***Figure 11. Board Modification****2. Test Code**

The following test code (bug.asm) allows continued development and checks for incorrect power-up state. Keep cycling power and running bug.asm until it passes, indicating a valid power up.

Test code bug.asm creates an L1D load-miss condition during a stall. If the L1D controller is powered up to an incorrect state, the load-miss operation corrupts the data in address 0x4010 (pointed to by the load-miss operation). This test code checks for data corruption at that address.

Create an .out file with the following assembly code (bug.asm) and linker command file (lnk.cmd). Run to breakpoint "exit". If A0 = 0xFFFFDEAD, the device is powered up to an incorrect state. Recycle power and re-run the test code. If A0 = 0x01234, the device is powered up to a correct state and will remain in this correct state until the next power cycle.



*L1D Cache: Data Corruption if L1D Powered up to Wrong State (Continued)***Assembly Code (bug.asm)**

```

.global _main
.sect "vectors"
RESET B      _main
      NOP    5
      .text

_main:
init:  MVKL  0x00004010, B5      ; B5 = 0x00004010
      STW   B5, *B5            ; store data 0x4010 in address 0x4010
      MVKL  0x00003000, A3      ; A3 = 0x00003000
      MVKL  0xffffdead, B7      ; B7 = 0xffffdead
      MVKL  0x00004000, A8      ; A8 = 0x00004000

bug:   STW   B7, *A3            ; Store Miss to cache line B
      STW   A8, *A8            ; Store Miss to cache line A (Stall occurs
                               ; here due to
                               ; STW/LDW to same cache line)
      || LDW  *B5, B9           ; Load Miss to cache line A. This load
                               ; causes
                               ; corruption to address 0x4010 (B5)
                               ; but loads good data
      LDW   *B5, B10           ; Load Hit to cache line A
      NOP   4

check: CMPEQ B10, B5, B1       ; compare loaded value in B10 to expected
                               ; value (0x4010)
      [B1] MVKL  0x01234, A0     ; passed (B10 = 0x4010). Set A0 = 0x1234.
      [!B1] MVKL 0xffffdead, A0 ; failed (B10 does not equal 0x4010).
                               ; Set A0 = 0xffffdead

exit:  IDLE

```

**Linker File (lnk.cmd)**

```

MEMORY
{
    vecs:  o = 00000000h l = 00000200h
    IRAM:  o = 00000200h l = 00001000h
}
SECTIONS
{
    .text > IRAM
    vectors > vecs
}

```

## 6 C6711 Silicon Revision 1.3 Known Design Exceptions to Functional Specifications

### Advisory 1.3.2

*HPI: Read Data Corrupted in Fixed-Address Mode and FETCH Read Requests*

**Revision(s) Affected:** 1.0, 1.2, and 1.3 [C6711]

**Details:** During an HPI read access in fixed address mode (HCNTL[1:0] = 11b), the internal HPI data pointers become corrupted, and the wrong data will be returned. Any fixed-mode reads from that point onward will lag behind by 1 read. Example 1 shows a failing sequence.

#### Example 1. Failing sequence

```

HPIA  write 0x00000000
HPID++ write 0x00112233 (to address 0x0)
HPID++ write 0x44556677 (to address 0x4)
HPID++ write 0x8899AABB (to address 0x8)
HPID++ write 0xCCDDEEFF (to address 0xC)
HPIA  write 0x00000000 (reset address to 0)
HPID  read returns 0x00112233 (correct, but internal HPI data pointers become corrupted)
HPIA  write 0x00000004
HPID  read returns 0x00112233 (incorrect – expected 0x44556677, lags by 1 read)
HPIA  write 0x00000008
HPID  read returns 0x44556677 (incorrect – expected 0x8899AABB, lags by 1 read)
HPIA  write 0x0000000C
HPID  read returns 0x8899AABB (incorrect – expected 0xCCDDEEFF, lags by 1 read)

```

This problem also exists when the FETCH bit in the HPIC register is used to read from HPID. Autoincrement mode reads (HCNTL[1:0] = 10b), HPIC reads (HCNTL[1:0] = 00b), and HPIA reads (HCNTL[1:0] = 01b) function properly. In addition, host writes in all modes function properly.

Internal reference number DSPvd01120

**Workaround:** For HPI reads, avoid using the fixed mode (HCNTL[1:0] = 11b) or the FETCH bit in the HPIC. Use only the autoincrement mode (HCNTL[1:0] = 10b).

Note that different host mechanisms (software/hardware) are needed to migrate between systems with revisions 2.0/2.1 and revisions 1.0, 1.2, 1.3 silicons [C6711B and C6711, respectively] and 1.1 silicon [C6711C], due to advisories 1.1.1, 1.3.2, and 1.3.7.

**Advisory 1.3.4***JTAG: Boundary Scan Does Not Function***Revision(s) Affected:** 1.0, 1.2, and 1.3 [C6711]**Details:** Boundary-scan mode does not function.

Internal reference numbers DSPvd00725, DSPvd01207, DSPvd02060, DSPvd02061, DSPvd02062, DSPvd02063, DSPvd02064, and DSPvd02280.

**Workaround:** Do not use the boundary scan function.**Advisory 1.3.7***HPI: Write Request During HPI Time-Out Causes HPI Lock-Up***Revision(s) Affected:** 1.0, 1.2, and 1.3 [C6711]

**Details:** An internal HPI circuit times out after 128 CPU cycles since the last HPID access (i.e., 128 cycles every 2 halfword access). When HPI times out, the read/write request cycle is assumed completed and the read and write FIFOs of the HPI are flushed. The  $\overline{\text{HRDY}}$  signal goes high during the time-out/flushing process.

If an incoming write request to HPI occurs at the same time as the time-out, then this boundary condition may lead to incorrect write being issued to the HPI, and may cause the HPI to be stuck in a NOT READY state, indicated by the  $\overline{\text{HRDY}}$  signal staying high indefinitely.

**Workaround:** Perform any one of the following workarounds:

- Do writes quickly enough so that the HPI never times out. Keep them separated under 100 CPU cycles, if possible.
- If the above method is not possible, writes to HPID should be separated over 150 CPU cycles to avoid the 128-cycle mark plus the time it takes to time out and flush. After time-out, wait for  $\overline{\text{HRDY}}$  to become READY (low) again before performing additional transfers.
- For a write access around the 128-cycle time-out, perform an HPIA write to flush the FIFO before any subsequent data writes. The HPIA write must be done only when  $\overline{\text{HRDY}}$  indicates READY.

Note that different host mechanisms (software/hardware) are needed to migrate between systems with revisions 2.0/2.1 and revisions 1.0, 1.2, 1.3 silicons [C6711B and C6711, respectively] and revision 1.1 silicon [C6711C], due to advisories 1.1.1, 1.3.2, and 1.3.7.

**Advisory 1.3.13**HPI:  $\overline{\text{HRDY}}$  Behavior**Revision(s) Affected:** 1.0, 1.2, and 1.3 [C6711]**Details:** The  $\overline{\text{HRDY}}$  signal goes inactive (not ready) after each word access in autoincrement mode, even if the internal buffer is ready. In autoincrement mode,  $\overline{\text{HRDY}}$  should be inactive (not ready) after a word access only when the internal buffer is:

- Empty in HPI read operation
- Full in HPI write operation

**Workaround:** Be sure to observe the  $\overline{\text{HRDY}}$  signal when accessing the HPI. The  $\overline{\text{HRDY}}$  signal must be active (ready) when performing HPI reads/writes.**7 C6711 Silicon Revision 1.2 Known Design Exceptions to Functional Specifications**

Revision 1.2 is functionally the same as Revision 1.3. See Revision 1.3 Silicon Advisories.

**8 C6711 Silicon Revision 1.0 Known Design Exceptions to Functional Specifications**

Revision 1.0 advisories are shared with other revisions. See advisories of other revisions.

## 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 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.

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.

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

<b>Products</b>		<b>Applications</b>	
Amplifiers	<a href="http://amplifier.ti.com">amplifier.ti.com</a>	Audio	<a href="http://www.ti.com/audio">www.ti.com/audio</a>
Data Converters	<a href="http://dataconverter.ti.com">dataconverter.ti.com</a>	Automotive	<a href="http://www.ti.com/automotive">www.ti.com/automotive</a>
DSP	<a href="http://dsp.ti.com">dsp.ti.com</a>	Broadband	<a href="http://www.ti.com/broadband">www.ti.com/broadband</a>
Interface	<a href="http://interface.ti.com">interface.ti.com</a>	Digital Control	<a href="http://www.ti.com/digitalcontrol">www.ti.com/digitalcontrol</a>
Logic	<a href="http://logic.ti.com">logic.ti.com</a>	Military	<a href="http://www.ti.com/military">www.ti.com/military</a>
Power Mgmt	<a href="http://power.ti.com">power.ti.com</a>	Optical Networking	<a href="http://www.ti.com/opticalnetwork">www.ti.com/opticalnetwork</a>
Microcontrollers	<a href="http://microcontroller.ti.com">microcontroller.ti.com</a>	Security	<a href="http://www.ti.com/security">www.ti.com/security</a>
		Telephony	<a href="http://www.ti.com/telephony">www.ti.com/telephony</a>
		Video & Imaging	<a href="http://www.ti.com/video">www.ti.com/video</a>
		Wireless	<a href="http://www.ti.com/wireless">www.ti.com/wireless</a>

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