Freescale Semiconductor
Application Note

Document Number: AN2129
Rev. 1, 07/2006

Instruction and Data Cache Locking on the e300 Processor Core

by Jim Robertson and Kalpesh Gala
NCSG Applications
Freescale Semiconductor, Inc.

This document describes the instruction and data cache locking features on the e300 processor core, an MPC603E-based core used in the MPC8349E integrated microprocessor. Table 1 shows the cache characteristics of the e300 processor core.

Table 1. e300 Processor Core Cache Organization

Instruction Cache Size	Data Cache Size	Associativity	Block Size	Way Size
32 Kbytes	32 Kbytes	8 ways	8 words	4 Kbytes

Contents

1 Cache Locking Terminology

In this document, a cache is either an instruction or data cache unless otherwise specified. Cache locking is the ability to prevent overwriting of some or all of a microprocessor instruction or data cache. Cache locking can occur for either an entire cache or for individual ways within the cache.

When an entire cache is locked, hits within the cache are supplied in the same manner as hits to an unlocked cache. Any access that misses in the cache is treated as a cache-inhibited access. Cache entries that are invalid at the time of locking remain invalid and inaccessible until the

freescale semiconductor

cache is unlocked. When a cache is unlocked, all entries (including invalid entries) are available. Entire cache locking is inefficient if the number of instructions or the size of data to be locked is small compared to the cache size.

Way locking locks only a portion of the cache by locking ways within the cache. Locking always begins with the first way (way0) and is sequential; that is, you can lock ways 0, 1, and 2 but not ways 0 and 2. At least one way must be left unlocked. The maximum number of lockable ways is seven on the e300 processor (way0–way6).

Unlike entire cache locking, invalid entries in a locked way are accessible and available for data placement. As hits to the cache fill invalid entries within a locked way, the entries become valid and locked. This behavior differs from entire cache locking in which nothing is placed into the locked cache, even if there are invalid entries in the cache. Unlocked ways of the cache behave normally.

2 Overview of Cache Locking on the e300 Processor

To lock the instruction cache, set the instruction cache enable bit HID0[ICE], bit 16. To lock the data cache, set the data cache enable bit HID0[DCE], bit 17. The following assembly code enables the instruction and data caches:

```
# Enable the instruction and data caches. This corresponds
# to setting the ICE and DCE bits in HID0 (bits 16 and 17)

mfspr    r1, HID0
ori      r1, r1, 0xc000
sync
mtspr    HID0, r1
```

Two distinct memory areas must be set up to enable cache locking:

- The first area is where the locking code resides and is executed.
- The second area contains the data to be locked.

Both areas of memory must be in locations that are translated by the memory management unit (MMU). This translation can be performed either with the page table[1] or the block address translation (BAT) registers. This document describes the use of BAT register translations.

The instruction cache locking procedure is as follows; the remainder of this document discusses each of these steps in detail:

1. Set up memory.
2. Disable interrupts.
3. Invalidate the entire cache to ensure known state and force placement to way0.
4. Load the cache with the data to be locked.
5. Lock the cache, either way or entire cache.

1. Issues arising from using page table address translation are beyond the scope of this document.

3 Set Up Memory

For the cache locking example in this document, two areas of memory are defined through the BAT registers:

- A 1 Mbyte area in the upper region of memory that contains the code for cache locking. It must be cache-inhibited for instruction cache locking.
- A 256 Mbyte block of memory that contains the data to lock (not all of the 256 Mbyte of memory is locked in the cache; this area is set up as an example).

Both memory areas use identity translation (the logical memory address equals the physical memory address). Table 2 summarizes the BAT settings for this example.

Table 2. Example BAT Settings for Cache Locking

Area	Base Address	Memory Size	WIMG Bits	BATU Setting	BATL Setting
First	0xFFF0_0000	1 Mbyte	0b0100 [1]	0xFFF0_001F	0xFFF0_0022 [1]
Second	0x0000_0000	256 Mbyte	0b0000	0x0000_1FFF	0x0000_0002

[1] 0xFFF0_0022 defines a cache-inhibited memory area for instruction cache locking and corresponds to a WIMG of 0b0100. Cache-inhibited memory is not a requirement for data cache locking. A value of 0xFFF0_0002 with a corresponding WIMG of 0b0000 marks the memory area as cacheable.

General block address translation (BAT) programming is beyond the scope of this document. For a full discussion of BAT register programming, see the *Programming Environments Manual for 32-Bit Implementations of the PowerPC™ Architecture* (MPCFPE32B), which is available at the web site listed on the back cover of this document.

The block address translation upper (BATU) and block address translation lower (BATL) settings in Table 2 are for both instruction block address translation (IBAT) and data block address translation (DBAT) registers. After the BAT registers are set up, the MMU must be enabled. The following assembly code enables both instruction and data memory address translation:

```
# Enable the instruction and data caches. This corresponds
# to setting the ICE and DCE bits in HID0 (bits 16 and 17)

mfspr    r1, HID0
ori      r1, r1, 0xc000
sync
mtspr    HID0, r1

# Enable instruction and data memory address translation. This
# corresponds to setting IR and DR in the MSR (bits 26 & 27)

mfmsr    r1
ori      r1, r1, 0x0030
mtmsr    r1
sync
```

4 Disable Interrupts

To ensure that interrupt service routines do not execute while the cache is loaded, which can pollute the cache with undesired contents, all interrupts should be disabled by clearing the appropriate bits in the machine state register (MSR) register. Table 3 lists the MSR bits that must be cleared to ensure that interrupts are disabled.

Table 3. MSR Bits for Disabling Interrupts

Bit	Name	Description
16	EE	External interrupt enable
19	ME	Machine check enable
20	FE0 [1]	Floating-point exception mode 0
23	FE1[1]	Floating-point exception mode 1

[1] The floating-point exception does not need to be disabled because the code that performs cache locking does not execute floating -point operations.

The following assembly code disables all interrupts:

```
# Clear the following bits from the MSR:
# EE (16) ME (19)
# FE0 (20) FE1 (23)

mfmsr    r1
lis      r2, 0xffff
ori      r2, r2, 0x66ff
and      r1, r1, r2
mtmsr    r1
sync
```

5 Invalidate the Caches

This section describes the invalidation of the instruction and data caches.

5.1 Instruction Cache

The entire instruction cache for the e300 microprocessor is invalidated through the instruction cache flash invalidate bit HID0[ICFI], bit 20. Setting HID0[ICFI] and then immediately clearing it causes the entire instruction cache to be invalidated. The following assembly code invalidates the entire instruction cache:

```
# Set and then clear the HIDO[ICFI] bit, bit 20

mfspr    r1, HID0
mr       r2, r1
ori      r1, r1, 0x0800
mtspr    HID0, r1
mtspr    HID0, r2
sync
```

5.2 Data Cache

If a non-empty data cache contains modified data that cannot be discarded, the data cache must be flushed before it can be invalidated. To flush the data cache, fill the data cache with known data and then flush this data with a series of **dcbf**[1] instructions. The following code sequence shows how to flush the data cache:

```
        # r6 contains a block-aligned address in memory with which to fill
        # the data cache. For this example, address 0x0 is used

        li      r6, 0x0

        # CTR = number of data blocks to load
        # Number of blocks = (32K) / (32 Bytes/block)
        # = 2^15 / 2^5 = 2^10 = 0x400

        li      r1, 0x400
        mtctr   r1

        # Save the total number of blocks in cache to r8

        mr      r8, r1

        # Load the entire cache with known data

loop:   lwz     r2, 0(r6)
        addi    r6, r6, 32        # Find the next block
        bdnz    loop              # Decrement the counter, and
                                  # branch if CTR != 0

        # Now, flush the cache with dcbf instructions

        li      r6, 0x0           # Address of first block
        mtctr   r8                # Number of blocks
loop2:  dcbf    r0, r6
        addi    r6, r6, 32        # Find the next block
        bdnz    loop2             # Decrement the counter, and
                                  # branch if CTR != 0
```

If the contents of the data cache do not need to be flushed to memory, the cache can be directly invalidated. The entire data cache is invalidated through the data cache invalidate bit HIDO[DCFI], bit 21. Setting HID0[DCFI] and then immediately clearing it causes the entire instruction cache to be invalidated. The following assembly code invalidates the entire data cache (does not flush modified entries):

```
        # Set and then clear the HID0[DCFI] bit, bit 21

        mfspr   r1, HID0
        mr      r2, r1
        ori     r1, r1, 0x0400
        mtspr   HID0, r1
        mtspr   HID0, r2
        sync
```

6 Load the Caches

This section discusses preloading and loading instructions and data into the instruction and data cache, respectively.

1. The **dcbf** instruction forces a flush and invalidation of a data cache block.

6.1 Preload Instructions into the Instruction Cache

Instructions are preloaded into the instruction cache by speculatively fetching the instructions to be loaded. These instructions are speculatively fetched for execution when it is known that they are to be canceled. Although the execution of instructions is canceled, the instructions remain valid in the instruction cache.

Because instructions are intentionally executed speculatively, care must be taken to ensure that all I/O memory is marked guarded. Otherwise, speculative loads and stores to I/O space can cause data loss. (For a full discussion of guarded memory, see *PowerPC Microprocessor Family: The Programming Environments for 32-Bit Microprocessors*). The code that prefetches must be in cache-inhibited memory as shown in the following example:

```
        # Assuming interrupts are turned off, cache is flushed,
        # the MMU is on, and we are executing in a cache-inhibited
        # location in memory

        # LR and r6 = Starting address of code to lock
        # CTR = Number of cache blocks to lock
        # r2 = nonzero numerator and denominator

        # 'loop' must begin on an 8-byte boundary to ensure that
        # the divw and beqlr+ are fetched on the same cycle.

        .orig 0xFFF04000

loop:   divw.    r2, r2, r2        # LONG divide w/ nonzero result
        beqlr+                     # Cause the prefetch to happen

        addi     r6, r6, 32        # Find next block to prefetch
        mtlr     r6                # set the next block

        bdnz-    loop              # Decrement the counter and
                                   # branch if CTR != 0
```

In this example, both the **divw.** and **beqlr+** instructions are fetched at the same time because of their placement on a (double-word) boundary. A 64-bit coherent system bus (CSB) data bus is assumed. The preloading code does not work for a 32-bit data bus. The divide instruction is chosen because it takes many cycles to execute. During the divide execution, the processor starts fetching instructions speculatively at the target destination of the branch instruction. The speculation occurs because the branch is statically predicted as taken. This speculative fetching causes the cache block to which the link register (LR) points to be loaded into the cache. Because the **divw.** instruction always produces a non-zero result, the **beqlr+** is not taken, and all speculatively fetched instructions that begin execution are canceled. However, the instructions remain valid in the cache.

If the destination instruction stream contains an unconditional branch to another memory location, the destination of the unconditional branch instruction can also be prefetched. This does not cause a problem if the destination of the unconditional branch is also inside the area of memory to be preloaded. However, if the destination of the unconditional branch is not in the area of memory to be loaded, care must be taken to ensure that the branch destination is to an area of memory that is cache inhibited. Otherwise, unintentional instructions may be locked in the cache and the desired instructions may not be in their expected way within the cache.

6.2 Load Data into the Data Cache

The data cache can be loaded in several ways. The example in this document loads the data from memory. The following assembly code loads the data cache:

```
        # Assuming interrupts are turned off, cache has been flushed,
        # MMU on, and loading from contiguous cacheable memory.
        # r6 = Starting address of code to lock
        # r20 = Temporary register for loading into
        # CTR = Number of cache blocks to lock

loop:   lwz     r20, 0(r6)        # Load data into d-cache
        addi    r6, r6, 32        # Find next block to load
        bdnz    loop              # CTR = CTR-1, branch if CTR != 0
```

7 Lock the Cache

This section describes the methods for locking the entire instruction and data caches, instruction and data cache way locking, and invalidation of a locked cache. It concludes with a cache locking register summary.

7.1 Lock the Entire Instruction Cache

Locking the entire instruction cache is controlled by the instruction cache lock bit, HID0[ILOCK],bit 18. Setting HID0[ILOCK] locks the entire instruction cache, and clearing HID0[ILOCK]allows the instruction cache to operate normally. Setting of HID0[ILOCK] should be preceded by an **isync** instruction to prevent the instruction cache from being locked during an instruction access. The following assembly code locks the entire instruction cache.

```
        # Set the ILOCK bit in HID0 (bit 18)

        mfspr   r1, HID0
        ori     r1, r1, 0x2000
        isync
        mtspr   HID0, r1
```

7.2 Instruction Cache Way Locking

Way locking is controlled by HID2[IWLCK], (bits 16–18). Table 4 shows the HID2[IWLCK](0–2) settings for the e300 processor code. Setting HID0[ILOCK] locks all ways.

Table 4. e300 Processor Core IWLCK0–IWLCK2 Encoding

IWLCK0–IWLCK2	Ways Locked
0b000	No ways locked
0b001	Way 0 locked
0b010	Ways 0 through way 1 locked
0b011	Ways 0 through way 2 locked
0b100	Ways 0 through way 3 locked

Table 4. e300 Processor Core IWLCK0–IWLCK2 Encoding (continued)

0b101	Ways 0 through way 4 locked
0b110	Ways 0 through way 5 locked
0b0111	Ways 0 through way 6 locked

The following assembly code locks way0 through way3 of the e300 processor instruction cache:

```
# Lock way0 of the e300 processor instruction cache
# This corresponds to setting IWLCK0-IWLCK2 to 0b0100 (bits 16-18)

mfspr    r1, HID2
lis      r2, 0xffff
ori      r2, r2, 0x1fff
and      r1, r1, r2
ori      r1, r1, 0x8000
isync
mtspr    HID2, r1
```

7.3 Lock the Entire Data Cache

Locking the entire data cache is controlled by the data cache lock bit HID0[DLOCK], bit 19. Setting the HID0[DLOCK] bit locks the entire data cache. To unlock the data, clear the HID0[DLOCK] bit. Setting the DLOCK bit should be preceded by a sync instruction to prevent the data cache from being locked during a data access. The following assembly code locks the entire data cache:

```
# Set the DLOCK bit in HID0 (bit 19)

mfspr    r1, HID0
ori      r1, r1, 0x1000
sync
mtspr    HID0, r1
```

7.4 Data Cache Way Locking

Way locking is controlled by the HID2[DWLCK], (bits 24–26). Table 5 shows the DWLCK0–DWLCK2 settings for the e300 processor.

Table 5. e300 Processor Core DWLCK0–DWLCK2 Encoding

DWLCK0–DWLCK2	Ways locked
0b000	No ways locked
0b001	Way 0 locked
0b010	Ways 0 through way 1 locked
0b011	Ways 0 through way 2 locked
0b100	Ways 0 through way 3 locked
0b101	Ways 0 through way 4 locked

Table 5. e300 Processor Core DWLCK0–DWLCK2 Encoding (continued)

0b110	Ways 0 through way 5 locked
0b0111	Ways 0 through way 6 locked

The following assembly code locks way0 through way3 of the e300 processor data cache:

```
# Lock way0 of the e300 processor data cache
# This corresponds to setting DWLCK0–DWLCK2 to 0b0100 (bits 24-26)

mfspr    r1, HID2
lis      r2, 0xffff
ori      r2, r2, 0xff1f
and      r1, r1, r2
ori      r1, r1, 0x0080
sync
mtspr    HID2, r1
```

7.5 Invalidate a Locked Cache

There are two methods to invalidate a locked cache:

- Invalidate the entire cache by setting and then immediately clearing either the instruction cache flash invalidate bit HID0[ICFI], bit 20, or data cache flash invalidate bit HIDO[DCFI], bit 21. Even when a cache is locked, toggling either the ICFI or DCFI bit invalidates all of the instruction or data cache, respectively.
- Use either the instruction cache block invalidate **icbi** or data cache block invalidate **dcbi** instruction to invalidate individual cache blocks. The **dcbi** instruction invalidates blocks locked (either entire or way locked) within the data cache. Similarly, the **icbi** instruction invalidates blocks in an entirely locked instruction cache for both microprocessors. On the e300 processor, the **icbi** instruction invalidates way locked blocks within the instruction cache.

7.6 Cache Locking Register Summary

Table 6 through Table 8 outline the registers and bits to perform cache locking on the e300 processor.

Table 6. HID0 Bits to Perform Cache Locking

Name	Bit	Description
ICE	16	Instruction cache enable. Must be set for instruction cache locking.
DCE	17	Data cache enable. Must be set for data cache locking.
ILOCK	18	Instruction cache LOCK. Set this bit to lock the entire instruction cache.
DLOCK	19	Data cache LOCK. Set this bit to lock the entire data cache.

Table 6. HID0 Bits to Perform Cache Locking (continued)

Name	Bit	Description
ICFI	20	Instruction cache flash invalidate. Setting and then clearing this bit invalidates the entire instruction cache.
DCFI	21	Data cache flash invalidate. Setting and then clearing this bit invalidates the entire data cache.

Table 7. HID2 Bits to Perform Cache Locking

Name	Bit	Description
IWLCK	16–18	Instruction cache way lock. Use these bits to lock individual ways in the instruction cache.
DWLCK	24–26	Data cache way lock. Use these bits to lock individual ways in the data cache.

Table 8. MSR Bits to Perform Cache Locking

Name	Bits	Description
EE	16	External interrupt enable. This bit must be cleared during instruction and data cache loading.
ME	19	Machine check enable. This bit must be cleared during instruction and data cache loading.
IR	26	Instruction address translation. This bit must be set to enable instruction address translation by the MMU.
DR	27	Data address translation. This bit must be set to enable data address translation by the MMU.

8 Revision History

Table 9 provides a revision history for this application note.

Table 9. Document Revision History

Rev. Number	Date	Substantive Change(s)
0	4/1999	Initial release.
1	7/2006	Updated the document to discuss the use of the icbt instruction, which makes locking the cache much more straightforward. Also, this document now references the e300 core instead of the G2 core.

THIS PAGE INTENTIONALLY LEFT BLANK

How to Reach Us:

Home Page:
www.freescale.com

email:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
1-800-521-6274
480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku
Tokyo 153-0064, Japan
0120 191014
+81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate,
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor
Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447
303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor
@hibbertgroup.com

Document Number: AN2129
Rev. 1
07/2006

freescale™
semiconductor